Novius OS Documentation

Version Chiba 1.0.1

Novius

22 November 2013

Table des matières

1 Sommaire **3**
1.1 Installer Novius OS 3
1.2 Comprendre Novius OS 19
1.3 Étendre une application 38
1.4 Créer une nouvelle application 48
1.5 Notes de versions 61
1.6 Contribuer à Novius OS 74

Bienvenue sur la documentation de Novius OS. Elle est hébergée et générée par Read The Docs.

Toutes les contributions sont les bienvenues : signalement ou correction d'erreurs, propositions d'améliorations ou traductions.

Les sources sont sur Git Hub, nous attendons vos `Pull Request` avec impatience. Contactez-nous si vous avez besoin d'aide avec votre contribution.

- Documentation d'API (en anglais uniquement)
- English version

CHAPITRE 1

Sommaire

1.1 Installer Novius OS

1.1.1 Installation

Sommaire

- Prérequis généraux
- Installation rapide
  - Pré-requis
  - Installation
- Installation via Zip
- Installation avancée
  - Configuration d'un Virtual Host
  - Installation avancée avec Git

Prérequis généraux

- Disposer d'un serveur **LAMP** avec PHP 5.3+.

```
sudo apt-get install apache2 php5 mysql-server libapache2-mod-php5 php5-mysql
```

- Avoir le **mod_rewrite** d'**Apache** activé.

```
sudo a2enmod rewrite
```

Les commandes sont données à titre d'exemple si vous installez Novius OS sur votre machine locale ou un serveur sur lequel vous avez les droits d'administration.
Elles sont valables pour installation sur Ubuntu, adaptez-les en fonction de votre distribution.

Note : Théoriquement Novius OS peut fonctionner avec un serveur autre qu'**Apache**.

Installation rapide

Pré-requis

- Avoir un accès ligne de commande sur le serveur et disposer des droits d'administration **sudo**.
- Avoir **Git** installé.

Installation

Ouvrez un terminal et saisissez :

```
cd /var/www
sudo wget http://raw.github.com/novius-os/ci/master/chiba/tools/install.sh && sh install.sh
```

À la question *« Enter the directory name where you want to install Novius OS (default novius-os) »*, indiquez le nom du répertoire dans lequel vous voulez installer votre instance de Novius OS. Laissez vide pour l'installer dans un répertoire `novius-os`.

Une fois l'installation terminée :
- Ouvrez votre navigateur à l'URL `http://votredomaine/novius-os/` (remplacez `novius-os` par le nom du répertoire que vous avez saisi).
- Poursuivez l'installation avec *l'assistant de paramétrage*.

Note :
- Pour une installation en local, l'URL sera probablement `http://localhost/novius-os/`.
- Si le `DOCUMENT_ROOT` de votre serveur n'est pas `/var/www/`, modifiez la première ligne en conséquence.

Installation via Zip

Cette procédure est à privilégier si vous souhaitez installer Novius OS sur un hébergement mutualisé :
- Téléchargez novius-os.chiba.1.0.1.zip.
- Dézippez le fichier.
- Uploadez (ou déplacer) le répertoire `novius-os` dans le `DOCUMENT_ROOT` de votre serveur (par exemple via FTP).
- Ouvrez votre navigateur à l'URL `http://votredomaine/novius-os/` (remplacez `novius-os` par le nom du répertoire où vous avez dézippé Novius OS).
- Poursuivez l'installation avec *l'assistant de paramétrage*.

Installation avancée

Configuration d'un Virtual Host

Les commandes suivantes sont données à titre d'exemple si vous voulez installer Novius OS sur Ubuntu, adaptez les en fonction de votre distribution.

```
sudo nano /etc/apache2/sites-available/novius-os
```

Remplacez **nano** par n'importe quel autre éditeur de texte.
Remplacez `novius-os` par le nom que vous voulez donner à votre `Virtual Host`.

Copiez la configuration suivante dans le fichier que vous venez d'ouvrir et sauvegardez.
Adaptez la ligne `ServerName` avec votre nom de domaine dans le cas d'une installation en production.
De même, remplacez `/var/www/novius-os` par le répertoire dans lequel vous avez installé Novius OS.

```
<VirtualHost *:80>
        DocumentRoot /var/www/novius-os/public
        ServerName   novius-os
        <Directory /var/www/novius-os/public>
                AllowOverride All
                Options FollowSymLinks
        </Directory>
</VirtualHost>
```

La configuration par défaut contient un répertoire `public`. C'est vers ce lui que doit pointer `DocumentRoot`.

Activez votre nouveau `VirtualHost` :

```
sudo a2ensite novius-os
```

Relancez ensuite **Apache** pour appliquer la nouvelle configuration.

```
sudo service apache2 reload
```

Configurer le fichier hosts, dans le cas d'installation sur votre machine Si vous installez Novius OS sur votre machine locale, vous devez ajouter une ligne au fichier `/etc/hosts`, avec la valeur du `ServerName` (`novius-os` dans l'exemple ci-desssus) .

```
sudo nano /etc/hosts
```

Ajouter la ligne suivante :

```
127.0.0.1   novius-os
```

Installation avancée avec Git

Il faut cloner le dépôt disponible sur GitHub :

```
git clone --recursive git://github.com/novius-os/novius-os.git
```

Cette commande télécharge le dépôt principal, avec plusieurs submodules :
- novius-os : le cœur de Novius OS, qui contient lui-même des submodules, comme fuel-core ou fuel-orm.
- Différents submodules dans `local/applications` : les applications blog, actualités, commentaires, formulaires, diaporamas...

La branche par défaut du dépôt pointe vers la dernière version stable.
Les nouvelles versions seront disponibles dans des nouvelles branches.

Pour le moment, tous les dépôts dépendants de novius-os/novius-os partagent le même numéro de version. C'est-à-dire qu'une application disponible sur notre compte Github existe dans les mêmes versions que le cœur de Novius OS. Donc si vous utilisez `novius-os/core` en version chiba, alors vous devriez aussi utiliser `novius-os/app` dans le même numéro de version chiba.

Pour changer la version que vous voulez utiliser après un clone, n'oubliez pas de mettre à jour les submodules ! Exemple qui utilise la dernière nightly de la branche `dev` :

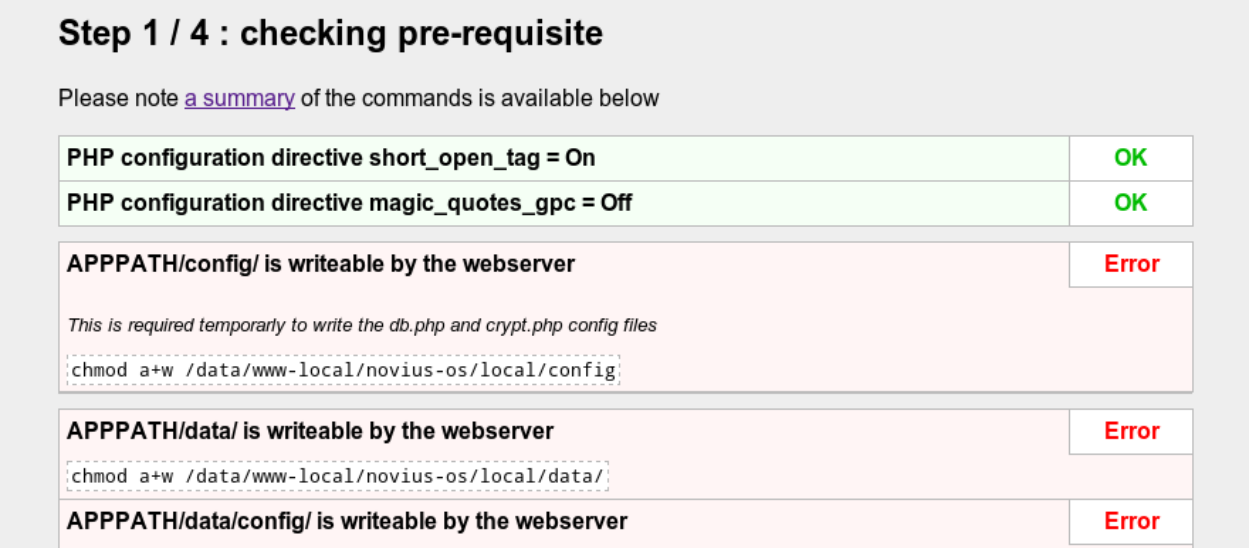
```
cd /var/www/novius-os/
git checkout dev
git submodule update --recursive
```

1.1.2 Assistant de paramétrage

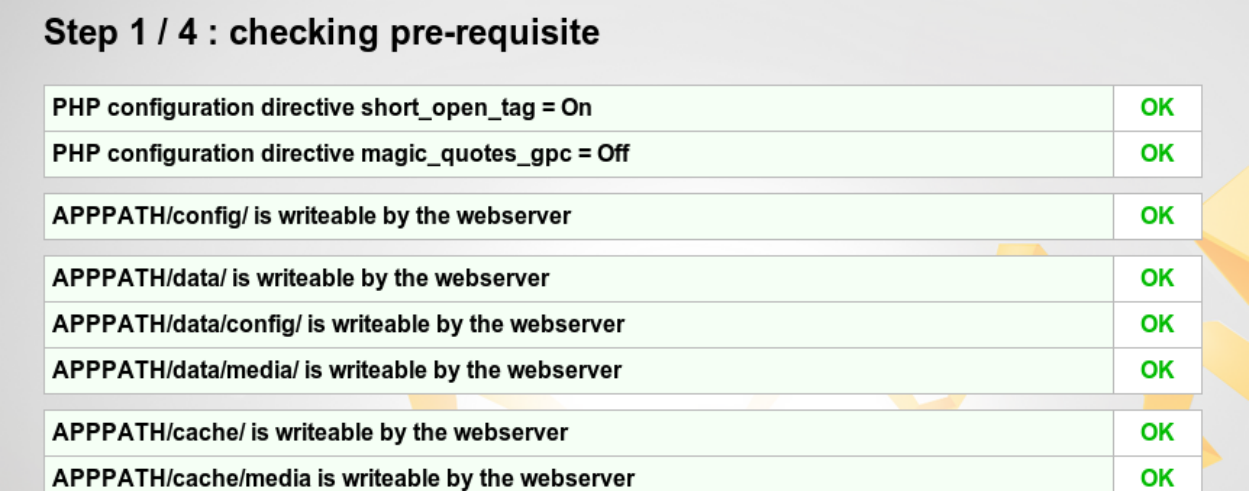
Vous avez suivi la *première étape de l'installation*. Les fichiers de Novius OS sont donc installés, le serveur paramétré et vous accédez à http ://votredomaine/novius-os/. Le plus dur est passé, la partie simple commence :-).

Étape 1 : Vérifier les prérequis

Cette étape devrait être une simple formalité si vous avez installé Novius OS avec la procédure d'installation rapide. Dans les autres cas, si vous voyez beaucoup de rouge, ne vous inquiétez pas ! Le site a juste besoin de droits en écriture dans certains répertoires. Cette étape vous donne des explications et les commandes à exécuter pour corriger tous les points.



Si vous ne voulez pas vous embêter, copiez / collez le résumé des commandes disponibles en bas de la page dans un terminal : c'est fini !



Étape 2 : Configurer la base de données MySQL

Vous avez besoin d'une base MySQL avec un utilisateur ayant les droits nécessaires. Dans le cas d'un hébergement mutualisé, ces paramètres ont dû vous être fournis par votre hébergeur. Dans les autres cas, voici un exemple pour une base en `localhost` :

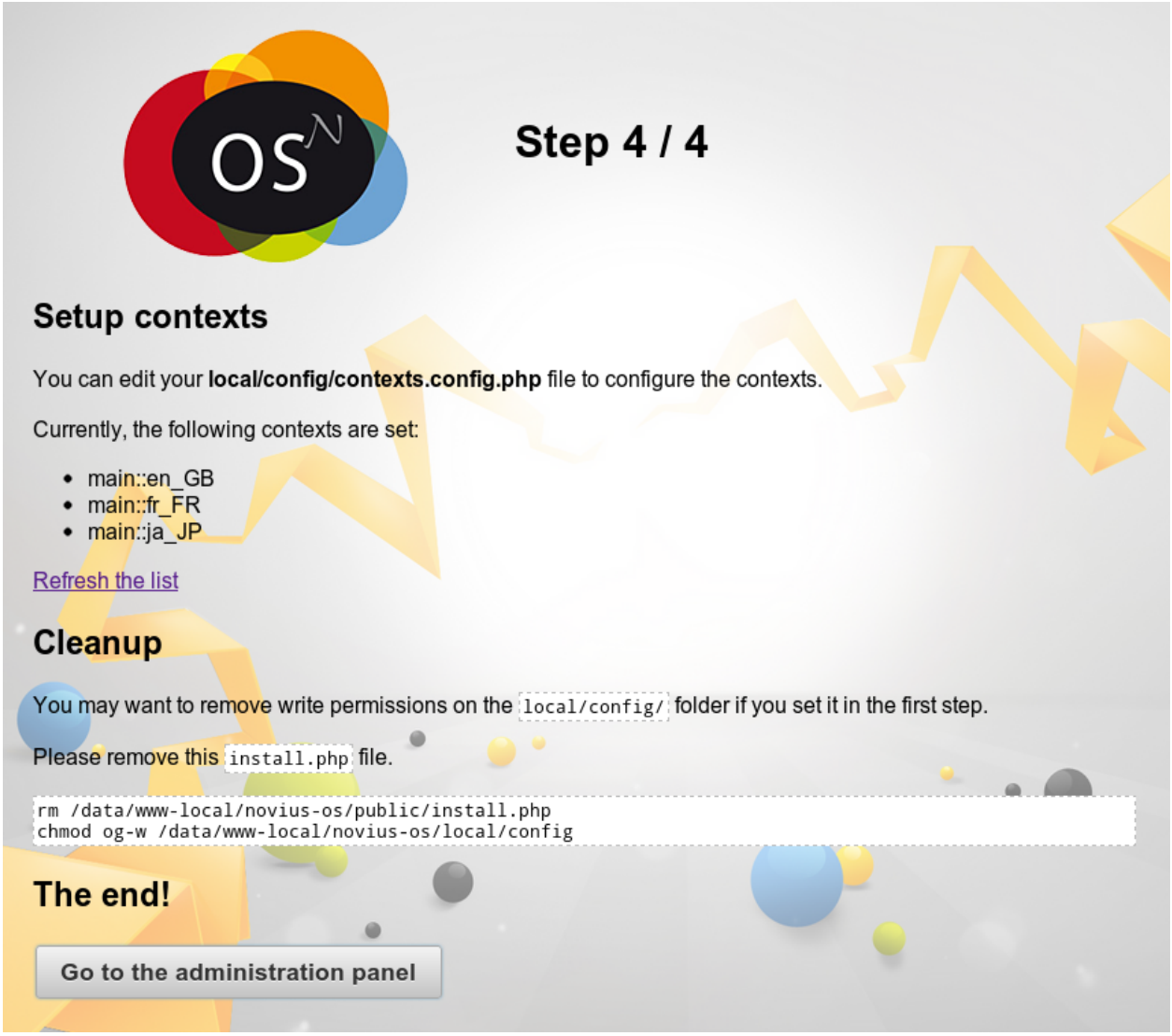
```
CREATE DATABASE `nom_de_votre_base` DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
GRANT ALL PRIVILEGES ON `nom_de_votre_base`.* TO 'nom_de_votre_utilisateur'@localhost IDENTIFIED BY '
FLUSH PRIVILEGES;
```

Remplissez ces quatre champs en fonction de la configuration de votre base :

Ceci va créer deux fichiers `local/config/db.php` et `local/config/crypt.php` et, surtout, les tables nécessaires dans votre base de données.

Étape 3 : Créer le premier compte administrateur

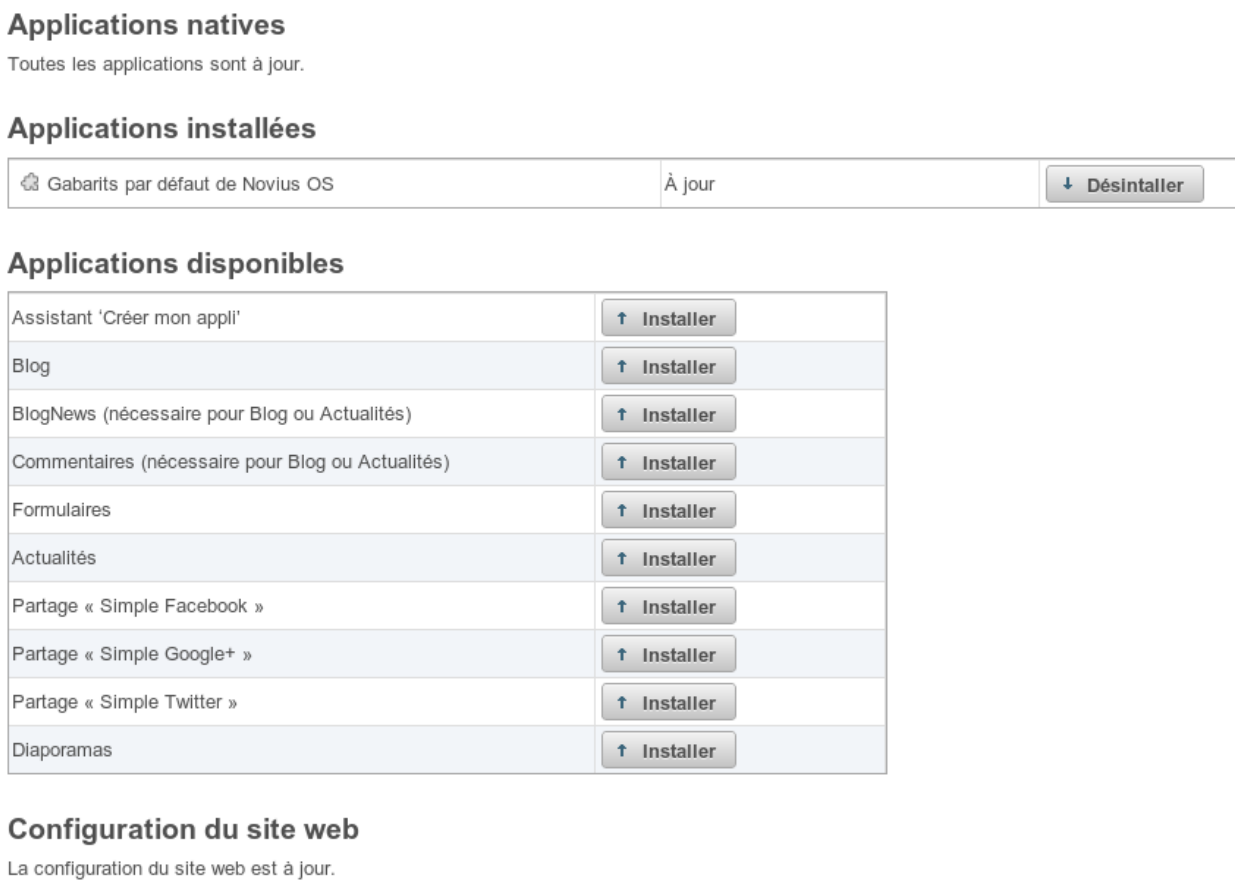
Étape 4 : Terminer l'installation



> **Warning :** Nous vous conseillons vivement de suivre les recommandation de cette page.
> Concernant le paramétrage des contextes, reportez-vous aux *principes* et *à la documentation d'API*.

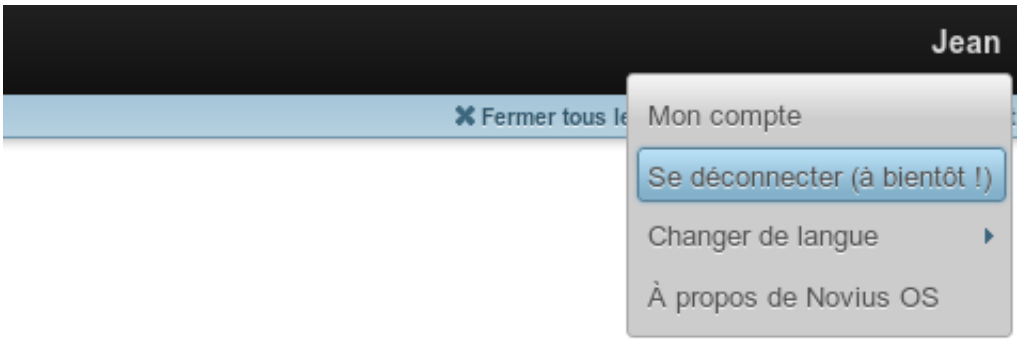
Applications

Vous arrivez sur le gestionnaire d'applications. C'est ici que vous installez les applications dont vous avez besoin.



Se déconnecter / connecter

Pour vous déconnecter, cliquez sur votre prénom en haut à droite. Un menu apparait alors :



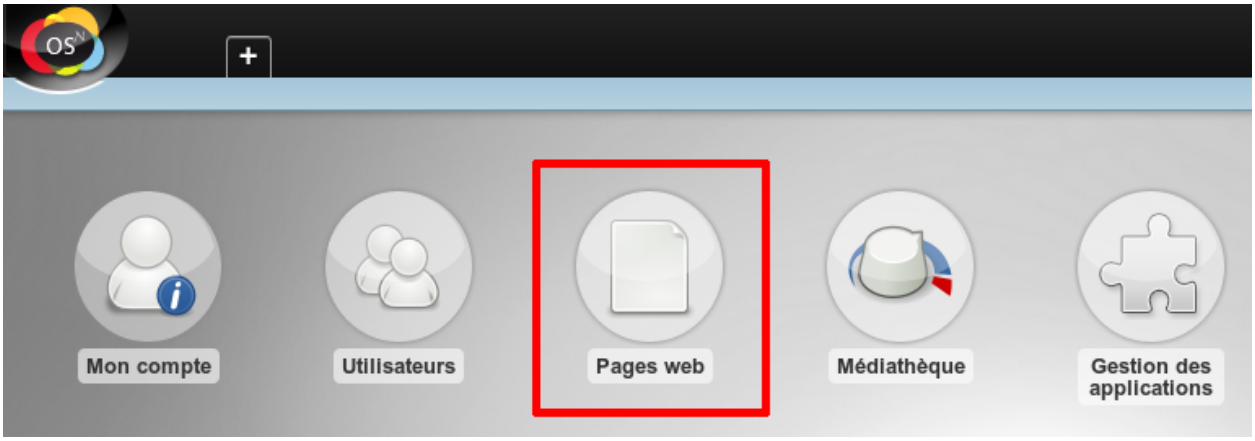
Vous être alors redirigé sur le formulaire de connexion.

1.1.3 La suite

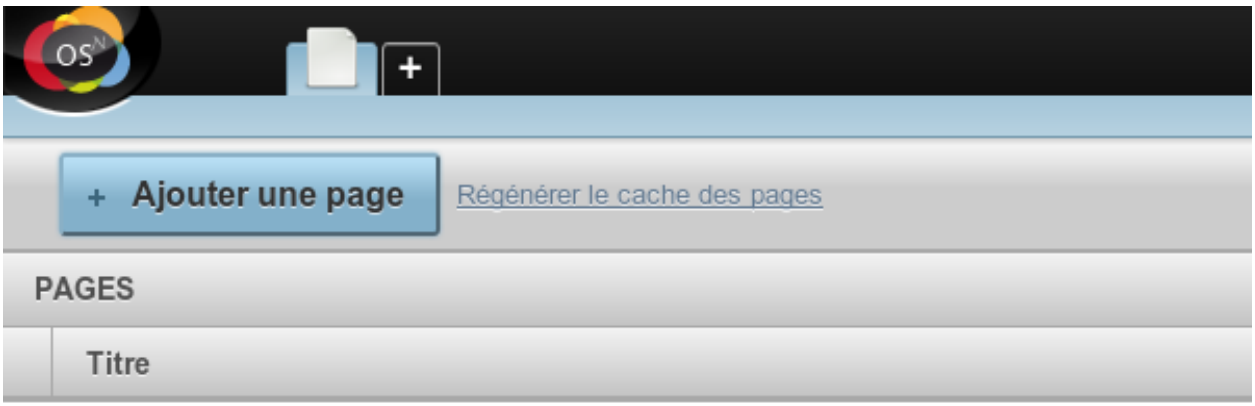
Première page

Ouvrir l'application Pages web

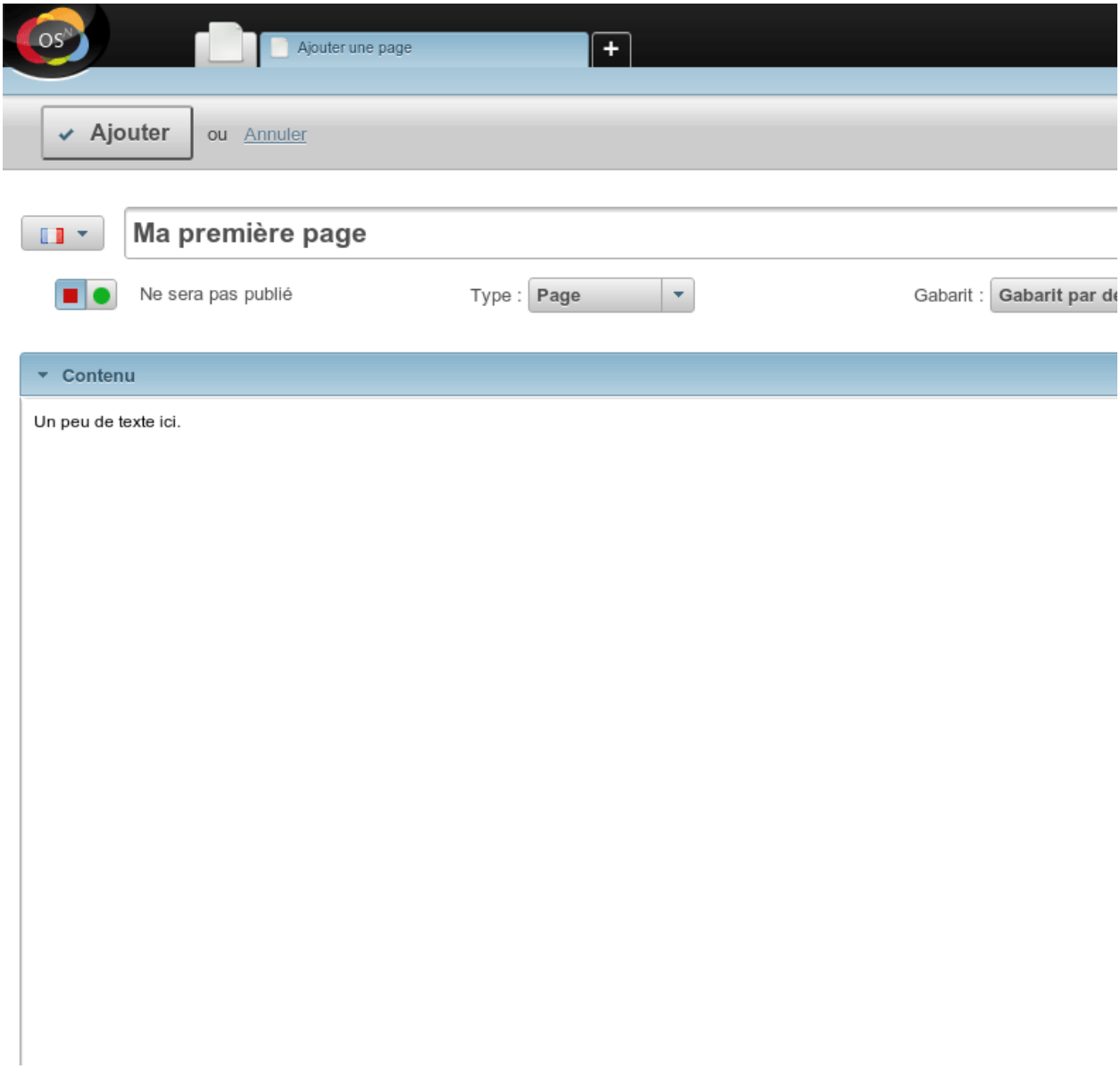
Si vous voulez créer un site web, vous devez utiliser l'application **Pages web** :



Ajouter une page (votre première)

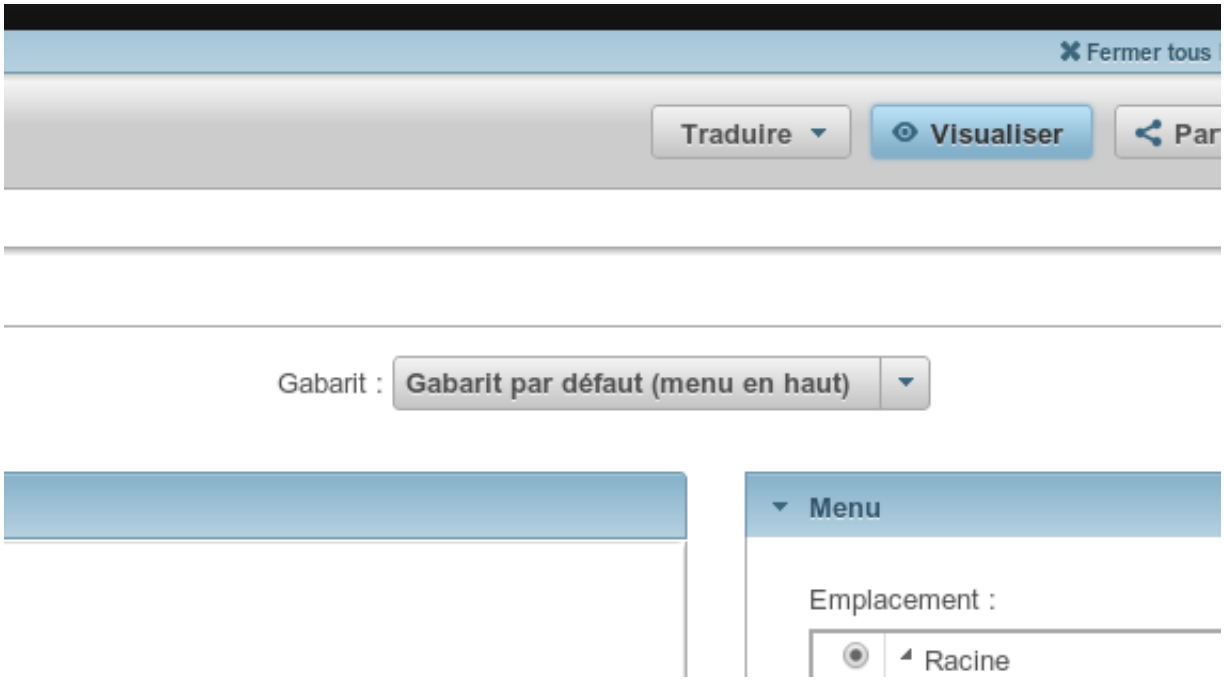


Écrire du contenu et cliquer « Ajouter »



Aperçu de votre travail

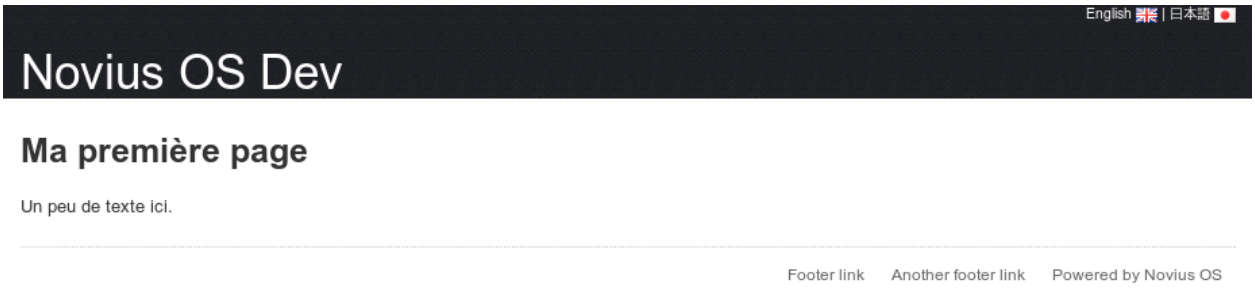
L'action **Visualiser** vous permet d'avoir un aperçu de la page avant qu'elle ne soit publiée.



Publier votre page

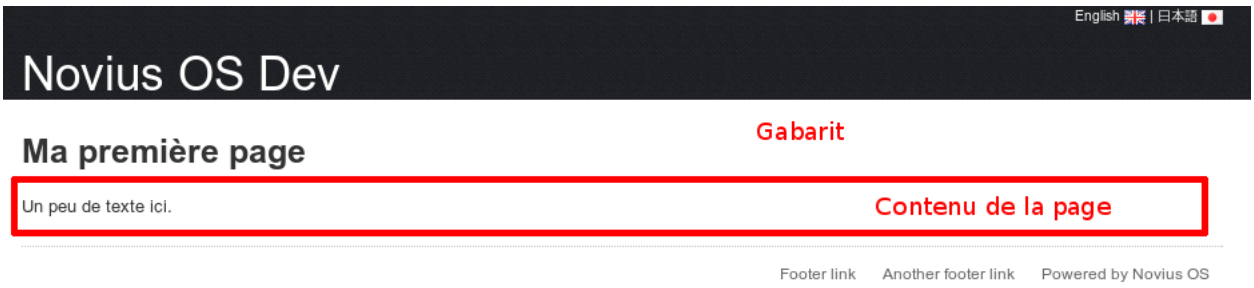
Une fois que vous êtes satisfait du contenu, choisissez « Sera publié » et enregistrez.

Admirez le travail remarquable que vous venez de faire :

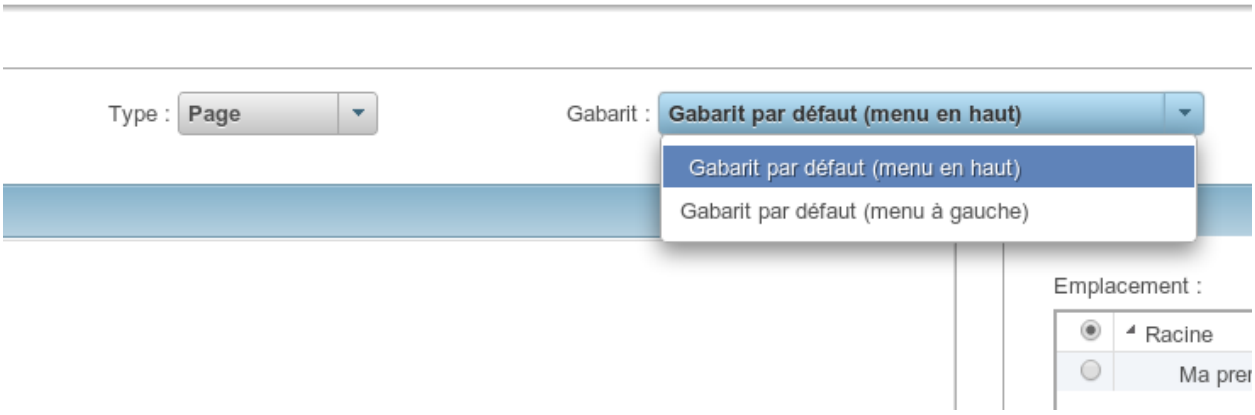


Gabarits

Les pages ont besoin de gabarits qui englobent le contenu ajouté en back-office et définissent le style.



Au moment d'ajouter ou modifier une page, on précise son gabarit.



Il est aussi possible d'ajouter de nouveaux gabarits. Les gabarits sont embarqués avec les applications. Ajouter un gabarit passe donc par l'ajout d'une application (via le *gestionnaire d'application*).

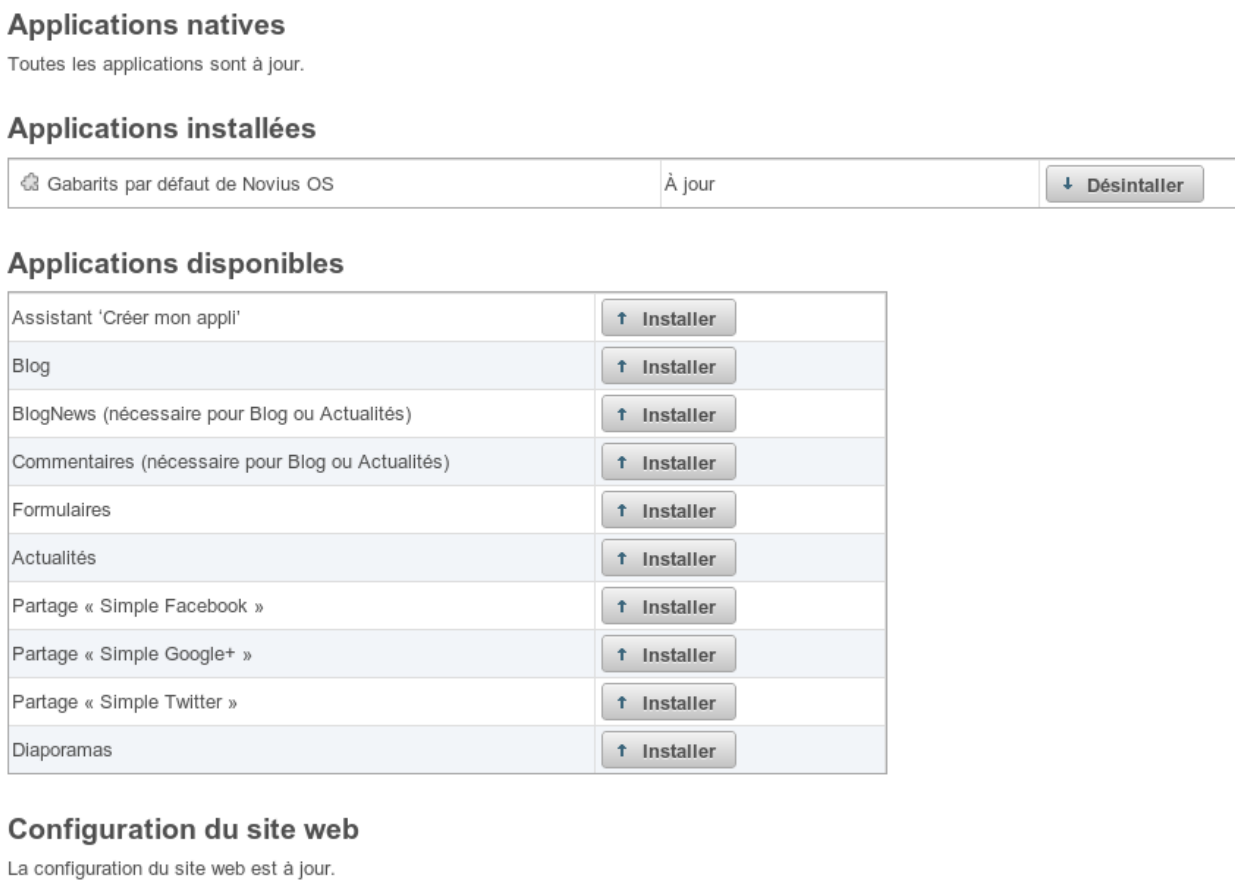
Applications

Les applications permettent de rajouter des fonctionnalités à Novius OS.

Le gestionnaire d'applications

Permet d'installer / désinstaller des applications.

Après des changements du `metadata` d'une application ou de votre site web (instance de Novius OS), une mise à jour est nécessaire dans le gestionnaire d'applications. C'est également le cas quand une application native est modifiée.



1.1.4 Mise à jour

Mise à jour des fichiers

Git

Si vous avez installé Novius OS avec **Git**, placez-vous dans le répertoire de votre Novius OS :

```
git fetch origin
git checkout master/chiba
git merge origin/master/chiba
git submodule update --recursive --init
```

Zip

Si vous avez téléchargé Zip, la procédure est plus complexe.
- Mettons que votre Novius OS est installé dans `monsite/`.
- Faites une sauvegarde de votre répertoire (copiez le répertoire ou zippez le).
- Téléchargez le nouveau zip de Novius OS et dézippez-le. Vous avez un répertoire `novius-os/`.
- **Dans `monsite/`, détruisez les répertoires suivant :**
    - `monsite/novius-os/`
    - `monsite/local/migrations/`

  - Tous les répertoires `monsite/local/applications/noviusos_*`
- **Copiez les répertoires et fichiers suivant de `novius-os/` vers `monsite/` :**
  - `novius-os/`
  - `local/migrations/`
  - Tous les répertoires `local/applications/noviusos_*`
  - `public/install.php`
  - `public/.htaccess`

Vous pouvez alors continuer votre mise à jour.

Lancer la migration

Avant de lancer la procédure de migration automatique, sauvegarder votre base de données.

Si vous avez accès à **SSH** sur le serveur, placez-vous dans le répertoire de votre Novius OS :

```
sudo php oil refine migrate
```

Si vous n'avez pas accès à **SSH**, vous pouvez faire la migration via votre navigateur :

- Au préalable vous devez renommer le fichier `public/migrate.php.sample` en `public/migrate.php`.
- Appelez ensuite ce fichier via son URL, par exemple `http://www.monsite.com/migrate.php`.

Apps Manager

Dans le back-office de votre Novius OS, ouvrez l'application `Gestion des applications` et mettez à jour les applications le nécessitant.

Mettre à jour vos développement

Si vous avez des développements personnels, suivez la procédure le *Guide de migration de la version 0.2 à la version Chiba 1*.

1.1.5 Optimisations après installation

Configuration et activation de XSendFile

Pour comprendre ce qu'est XSendFile et à quoi il sert, c'est par ici : *Médiathèque*.

Apache

Dans **Apache**, il existe un module **mod_xsendfile** qui fournit la fonctionnalité. Ensuite, il faut modifier votre fichier .htaccess pour l'activer.

```
# Post-installation optimisation
<IfModule xsendfile_module>
    XSendFile On

    # Replace "novius-os-install-dir" by the real Novius OS installed directory
    XSendFilePath /novius-os-install-dir/local/data
```

```
</IfModule>
```

Novius OS détecte tout seul la présence du module et active automatiquement l'envoi de fichiers avec XSendFile.

nginx

Dans **nginx**, XSendFile est activé par défaut, mais l'entête à utiliser est `X-Accel-Redirect`. Dans ce cas, il faut éditer votre fichier de configuration `config.php` pour y renseigner cet entête :

```
<?php
return array(
    // ...
    'novius-os' => array(
        // ...
        'use_xsendfile' => 'X-Accel-Redirect',
    ),
);
```

1.2 Comprendre Novius OS

1.2.1 Fondamentaux du logiciel

Une architecture MVC

Novius OS répond aux standards de découpage Modèle-Vue-Contrôleur, qui définissent des logiques de travail :
– dans la conception des applications ;
– dans l'organisation d'un projet sous Novius OS.

Utilisation de frameworks

L'utilisation de frameworks oriente fortement la conception et l'implémentation des applications. Il convient donc de connaitre le rôle de chacun. Pour autant, cette documentation concernant Novius OS avant tout, veuillez vous référer à de la documentation ou tutoriaux externes pour plus de précisions sur ces frameworks.

FuelPHP

Consulter les tutoriaux FuelPHP par Novius

Le framework PHP utilisé pour Novius OS est FuelPHP.

Les éléments de FuelPHP les plus utilisés sont ceux qui permettent de valider les données, l'ORM et le mapping des différents fichiers. Au delà de ces éléments, des outils inclus dans le framework simplifient grandement l'implémentation des applications (comme la classe Arr par exemple).

ORM de FuelPHP

ORM pour object-relational mapping. En français mapping objet-relationnel.

L'ORM permet une gestion de la base de données par des objets PHP, des classes, et en gérant notamment les relations entre les tables.

Des exemples parlent plus qu'un long discours :

```
$new_monkey = Model_Monkey::forge();
$new_monkey->monk_name = 'Julian';
$new_monkey->save();

$monkeys = Model_Monkey::find('all');
foreach ($monkeys as $monkey) {
        //...
}

$monkey = Model_Monkey::find(4);
$monkey->delete();
```

Novius OS est basé sur l'ORM de FuelPHP. Veuillez vous référer à sa documentation.

Néanmoins, Novius OS ajoute une sur-couche notable à l'ORM : les `Behaviours`.

En français, `Behaviour` veut dire comportement. Les `Behaviours` permettent d'étendre des `Model` en y ajoutant des comportements standardisés.

Ils sont similaires aux Observers de FuelPHP mais plus puissants :

- Comme les `Observers`, ils sont configurables par des options.
- Comme les `Observers`, ils peuvent intercepter des événements pour agir sur le `Model` (par exemple l'événement `before_save` se déclenchant avant la sauvegarde).
- En plus, ils fournissent aussi des méthodes, d'instance ou statiques, sur le `Model`.
- Ils peuvent également fournir de nouveaux événements.

jQuery UI / Wijmo

Bien que les actions logiques soient effectuées en PHP côté serveur, Novius OS est en majorité écrit en Javascript. Cela s'explique par la grande importance donnée à l'interface utilisateur et à l'ergonomie (cf. *Principes ergonomiques*).

Pour proposer des interfaces et interactions riches, Novius OS utilise plusieurs librairies JS :

jQuery
: Ce framework facilite l'écriture du code JS pour l'édition du contenu HTML. Il n'est pas directement orienté UI.

  Documentation

jQuery UI
: Ce complément de jQuery permet d'ajouter des éléments d'interface. Une majorité de l'UI de Novius OS est issue de cette librairie.

  Documentation

Wijmo
: Cette librairie est basée sur jQuery UI et fournit des éléments d'interface complémentaires, appelés widgets.

  Documentation et Exemples

Il y a une hiérarchie entre ces librairies, Wijmo est la plus impactante sur l'ergonomie de Novius OS.

1.2.2 Organisation des répertoires

De Novius OS

Repertoires à la racine

- `~/novius-os/` : Le `core` de Novius OS
- `~/local/` : Votre site
- `~/public/` : Le `DOCUMENT_ROOT` du site
- `~/logs/` : Un répertoire de logs

Le répertoire `public`

Un fichier peut être :
- Exécutable ou non exécutable
- Fourni par le developpeur ou le logiciel, ou généré par le logiciel

Cela donne 4 usages possibles. Chacun d'eux a un répertoire dans `~/public/` :
- `~/public/static/` : Équivalent des `assets`. Des fichiers non exécutables fournis par le développeur ou Novius OS.
- `~/public/data/` : Fichiers non exécutables générés par Novius OS.
- `~/public/htdocs/` : Fichiers exécutables fournis par par le développeur ou Novius OS.
- `~/public/cache/` : Fichiers exécutables générés par Novius OS

Note : Ici, Novius OS fait référence au core, ou toute application de votre site web.

Il y a un 5ème répertoire `~/public/media/` utilisé par la *Médiathèque*.

Là où Novius OS peut écrire, le développeur ne le peut pas et vice et versa.

`~/public/static/` et `~/public/htdocs/` ont la même structure de sous-répertoire :
- `~/novius-os/` : Pour les fichiers venant du logiciel
- `~/apps/<application_name>/` : Pour les fichiers venant des developpeurs d'*applications*.

Ces sous-répertoires sont des liens symboliques, créés à l'installation du logiciel ou à l'activation des *applications*.

Ces liens symboliques pointant respectivement vers le `htdocs` et le `static` du repertoire du logiciel ou de l'application.

Voir ci-dessous l'*organisation des répertoires d'une application*.

Le répertoire du `core`

- `~/novius-os/framework/` : Le framework de Novius OS
- `~/novius-os/fuel-core/` : Le framework FuelPHP
- `~/novius-os/packages/` : Les packages FuelPHP

Le répertoire `local`

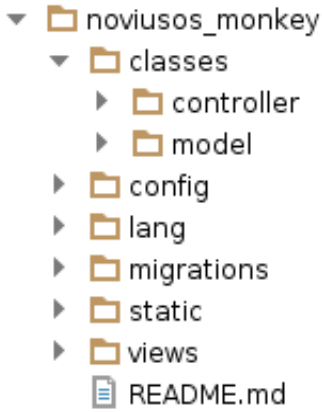
- `~/local/applications/` : Les *applications* Novius OS.
- `~/local/cache/` : Contient des médias redimensionnés.
- `~/local/classes/` : Classes PHP de vos développements.
- `~/local/config/` : Vos fichiers de configuration de Novius OS

- `~/local/data/` : Fichiers générés par Novius OS
- `~/local/metadata/` : Des fichiers de metadata de votre site, générés par Novius OS.
- `~/local/migrations/` : Des classes de migration.
- `~/local/views/` : Vos fichiers PHP de `Views` de vos développements.

Note : Les répertoires `classes` et `views` ne devrait pas contenir beaucoup de fichiers, la plupart de vos développements devrait être des *applications*..

D'une application

Tout Novius OS reprend les principes de segmentation issus de l'architecture MVC. Ils s'appliquent aussi bien au core qu'aux applications.



On distingue 6 dossiers principaux :

`classes` Ce dossier regroupe la partie logique, c'est-à-dire les classes PHP qui définissent et manipulent les données. Il s'agit a minima des contrôleurs et modèles de l'application. On y retrouve également des outils utilisés par les vues ou directement par les contrôleurs.

`config`
Ce dossier rassemble l'ensemble des informations permettant de représenter vos modèles. Les contrôleurs effectuent les opérations logiques sur vos données, mais auront besoin d'informations complémentaires à transmettre aux vues pour leur représentation. Ces informations sont ainsi séparées des contrôleurs, n'ayant pas de valeur logique, et des vues, car celles-ci reçoivent les données en paramètres et ne les recherchent jamais.

Le fichie de configuration du contrôleur `controller/admin/monkey.ctrl.php` se situe à `config/controller/admin/monkey.ctrl.php` Une classe et son fichier de configuration partagent une convention de nommage symétrique.

`lang` Ce dossier contient les fichiers de traduction, organisés en sous-dossiers par langue.

`migrations` Ce dossier contient les fichiers de migration.

`static` Ce dossier contient l'ensemble des scripts (JS et CSS) et ressources publiques (comme les images) chargées en front office.

`views` Ce dossier contient les fichiers responsables de l'affichage et de la représentation des données.

1.2.3 Différences avec FuelPHP

Chemin des constantes

Consultez la *documentation d'API des constantes*.

Autoloader

Deux `namespaces` sont ajoutés par Novius OS :

novius-os pointant vers *NOSPATH*.

local pointant vers *APPPATH*.

Bootstrap et points d'entrées

Dans Novius OS, le front-office et le back-office sont deux espaces bien séparés.

Au lieu d'un seul point d'entrée `index.php` de FuelPHP, Novius OS a deux points d'entrée :
- `~/novius-os/htdocs/admin.php` : Point d'entrée du back-office. Traite toutes les URL commençant par `/admin/`.
- `~/novius-os/htdocs/front.php` : Point d'entrée du front-office. Traite toutes les URL finissant par `.html` ou la racine de votre site.

Point d'entrée du back-office

Novius OS a une `route` configurée pour son back-office, dont la règle est la suivante :

```
<?php
'routes' => array(
        '^admin/(:segment)/(:any)' => '$1/admin/$2',
),
```

Concrètement une URL `admin/noviusos_page/page/insert_update/113` va être transformée en `noviusos_page/admin/page/insert_update/113`. Ce qui correspond à exécuter la méthode `action_insert_update` du `controller` `Controller_Admin_Page` de l'application `noviusos_page`.

Voir aussi :

Documentation de FuelPHP sur le routing.

1.2.4 Principes ergonomiques

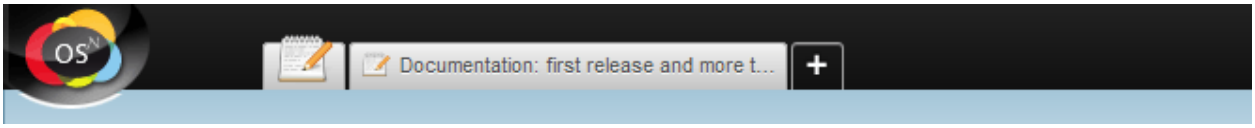
L'interface de Novius OS est construite autour de grands principes ergonomiques. Deux d'entre eux sont à connaitre pour le développement d'applications : la navigation par onglets et l'App Desk.

Navigation par onglets

Voir le screencast consacré à la navigation par onglets

Les onglets structurent le travail de l'utilisateur du back-office. Le but est de le faire gagner en productivité, en limitant les tâches répétitives et les chargement de pages.

On distingue deux types d'onglets :
- **Onglet d'application** : Les onglets d'application n'ont pas de titre, ils sont uniquement représentés par l'icône de l'application, en grand. Dans cet onglet, on trouve l'App Desk de l'application (voir plus bas).
- **Onglet d'item** : depuis l'onglet d'application, on accède à l'édition ou la visualisation d'un item, dans un nouvel onglet. Les onglets d'item portent le titre de l'item et l'icône de l'application, en petit.



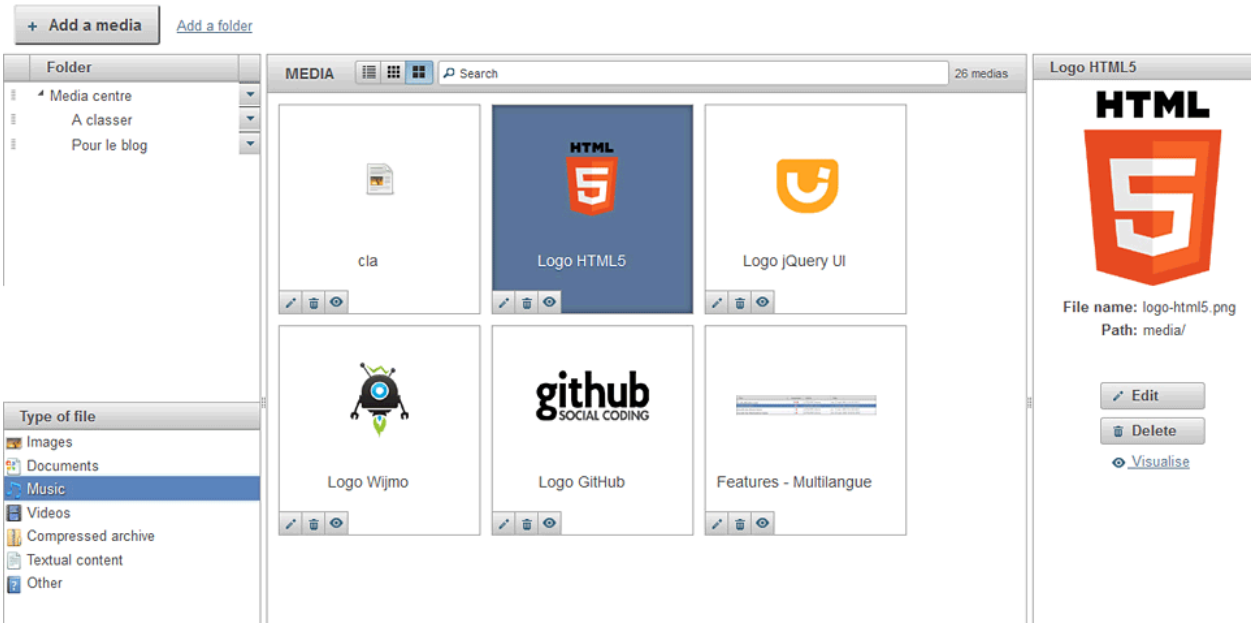
Les avantages de la navigation par onglets sont multiples. On retiendra :
- plusieurs items d'une même application peuvent être modifiés en parallèle ;
- les aller-retours entre items ou applications sont extrêmement rapides ;
- l'utilisateur reprend son travail là où elle / il l'avait laissé, l'ouverture des onglets étant conservée d'une session à une autre.

Les **pop-ups** doivent être limitées au cas modal, c'est-à-dire quand une action doit impérativement être accomplie (ou annulée) avant que le travail ne puisse se poursuivre (ex : confirmation d'une suppression, ajout d'un lien ou image à un contenu WYSIWYG).

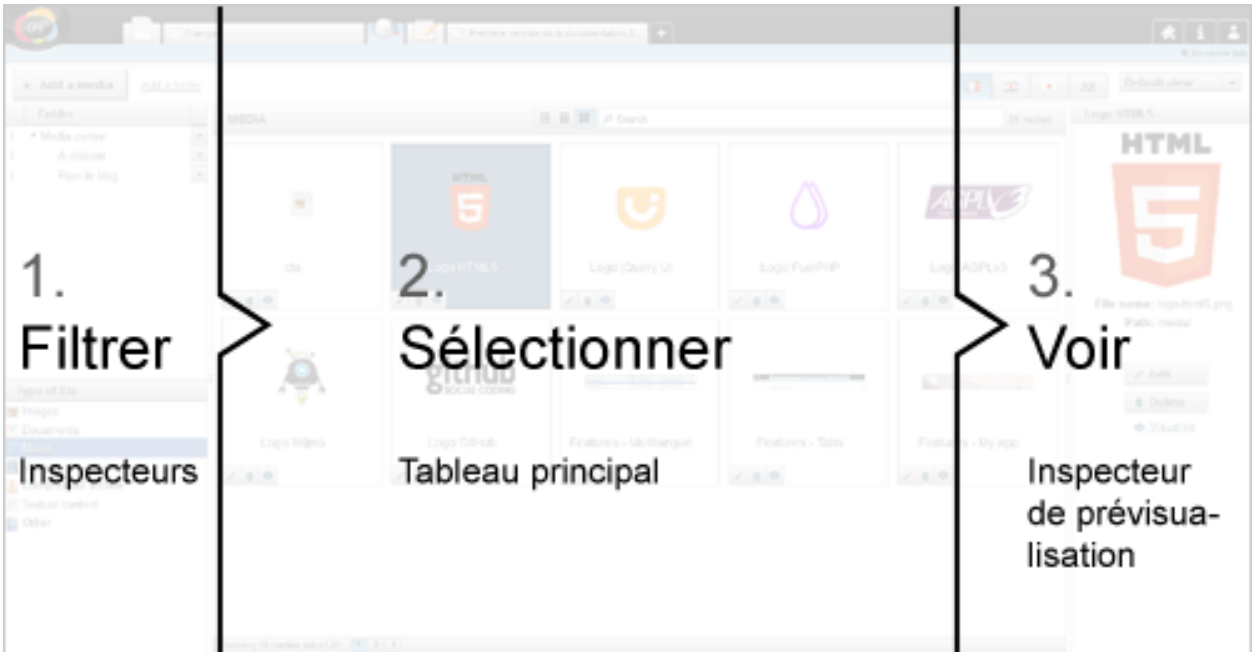
L'App Desk

Voir le screencast consacré à l'App Desk

L'App Desk est l'accueil d'une application, il permet l'accès aux différents items. Il est constitué des éléments suivants :
- **Tableau principal** : il liste les items d'une application, une ou plusieurs vues sont proposées (vignettes, tableau, arborescence, etc.). Son contenu est filtré par les inspecteurs et / ou une recherche full-text. Il ne peut y avoir qu'un seul tableau principal par App Desk.
- **Inspecteurs** : les inspecteurs regroupent les éléments meta d'une application (ex : auteurs pour un blog, dossiers pour la médiathèque). Les inspecteurs permettent de filtrer le contenu du tableau principal (ex : voir uniquement les billets d'un auteur précis). Certains inspecteurs permettent aussi de gérer des données (ex : supprimer un dossier).
  - Inspecteur de prévisualisation : l'inspecteur de prévisualisation est un cas particulier. Contrairement aux autres inspecteurs, il n'agit pas sur le tableau principal, c'est le tableau principal qui agit sur lui : quand un item est sélectionné, ses détails sont affichés dans l'inspecteur de prévisualisation (image, propriétés, récapitulatif des actions possibles pour l'item).
- **Actions** : dans la vaste majorité des cas, chaque App Desk doit proposer une et une seule action principale, généralement l'ajout d'un nouvel item. Des actions secondaires peuvent aussi être proposées, sous forme de liens : ajout d'un élément meta (ex : un dossier) ou action fréquente (ex : export).



L'App Desk offre de nombreuses possibilités de mise en page aux développeurs, comme à l'utilisateur final. Néanmoins, nous recommandons de proposer comme mise en page par défaut une Three-Pane Interface :



Alternativement, l'inspecteur de prévisualisation peut être placé sous le tableau principal.

1.2.5 Fondamentaux des applications

Une application se définit par ses modèles, mais aussi par les contrôleurs et vues associés. Ils dépendent de la nature de l'application. Néanmoins, certains éléments / principes sont génériques et réutilisables dans tous les cas.

Définition

Pour pouvoir ajouter une application au gestionnaire d'applications, il faut créer un fichier `metadata.config.php` pour votre application. Ce fichier doit contenir le namespace de l'application, qui doit être de la forme `Provider\NomApplication`. Il faut y ajouter le nom de l'application, une version et le provider (caractérisé au minimum par un nom).

Il est également possible de définir d'autres éléments dans ce fichier metadata :

Launchers Icônes de l'onglet d'accueil permet de lancer une application. Ils sont définis par un nom et une URL

Data catchers Composant d'une application permettant d'exploiter les données partagées par d'autres (dites sharable data)

Enhancers Grâce aux enhancers, une application vient enrichir le contenu édité dans un WYSIWYG.

Templates Modèles de pages pour le front-office.

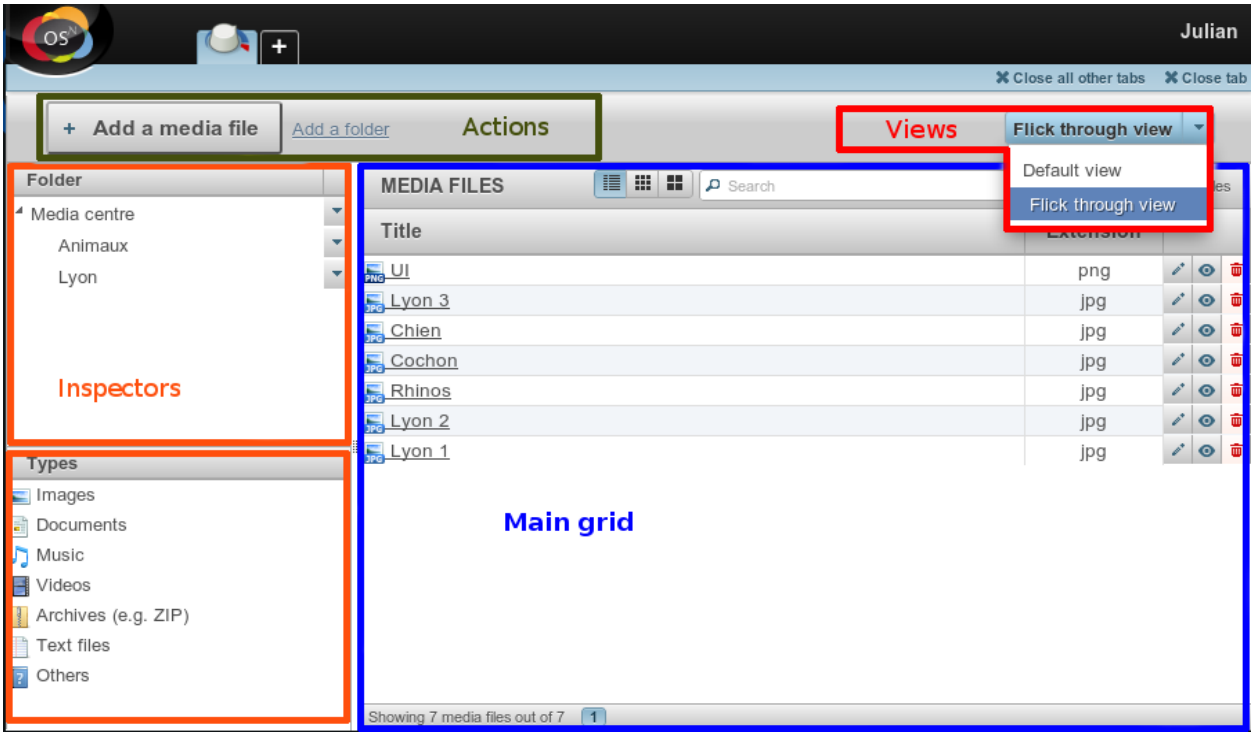
Voir aussi :

Infographie 'Comprendre les applications'

L'App Desk

Avant tout, *consulter les principes ergonomiques* pour comprendre l'App Desk.

La configuration de l'App Desk



L'App Desk est caractérisé par plusieurs éléments configurables :
- le type d'affichage des données ;
- les données à afficher ;
- les actions principales et secondaires.

Le *tableau principal* peut proposer plusieurs *vues* (à ne pas confondre avec le V de MVC) : liste, arborescence ou vignettes.

Ces vues sont définies via des fichiers de configuration, qui précisent les données à afficher ainsi que les *actions* principales et secondaires.

Les *inspecteurs* sont également définis via des fichiers de configuration. Ils indiquent sur quel attribut ou relation les données du tableau principal seront triées, ainsi que les actions associées aux éléments de l'inspecteur. À noter que les inspecteurs sont basés :
- Soit sur le même modèle que celui du tableau principal, l'inspecteur fait alors référence à des attributs (ex : date de création pour des billets de blog) ;
- Soit sur un autre modèle (ex : auteur pour des billets de blog).

Contrôleurs, formulaires et modèles

Depuis l'App Desk, il est possible d'appeler des contrôleurs qui réalisent des opérations sur les données concernées.

Certaines opérations s'effectuent directement (ex : la suppression, seule une confirmation est demandée). Elles sont, dans ce cas, attribuées au contrôleur de l'App Desk.

D'autres opérations appelent une vue et sont alors attribuées au contrôleur du modèle. Généralement, la vue appelée est un formulaire (ajout / édition). Ce dernier est construit grâce au fichier de configuration du modèle, qui peut être rempli grâce à une instance du modèle. Le contrôleur est de nouveau appelé lors de l'envoi du formulaire pour enregistrer les données.

Observers et behaviours

Les observers sont issus du framework FuelPHP.

Ce sont des procédures liées directement à un modèle. Elles sont appelées lorsque qu'un évènement identifié est déclenché. Ces procédures sont utilisées pour formatter, modifier ou valider des propriétés du modèle (ex : reformattage des données avant l'ajout en base de données).

Les behaviours, implémentées pour Novius OS, reprennent et étendent ce principe. Là où les observers effectuent une action sur une propriété du modèle, les behaviours définissent un ensemble de méthodes qui établissent un comportement particulier sur le modèle (ex : translatable, publishable). Ces méthodes sont également déclenchées via des évènements.

Ces outils ont pour intérêt de mutualiser des méthodes pour plusieurs modèles distincts.

1.2.6 Partage (Sharing)

Voir aussi :

Partage dans Novius OS

Content nuggets

Un content nuggets est un ensemble de données à partager.

Structure des données

Les données d'un content nugget ont une structure standardisées :
- Titre
- URL
- Texte
- Image

Pour pouvoir partager un item, il suffit de lui assigner le *Behaviour Sharable*, qui va définir comment extraire ces données standardisées.

Data catchers

Les data catchers sont des composants qui exploitent les content nuggets (eux-mêmes générés par les modèles).

Ils sont définis par les applications dans leur fichier `metadata.config.php`, exactement comme les gabarits, les enhancers et les launchers.

Inclus au logiciel

Déclenchés par l'utilisateur
- Twitter
- Facebook
- Google+
- Blog

Le data catcher `Blog` peut être utilisé pour créer des billets de blog à partir d'autres items, comme des singes (notre application bac-à-sable) ou des livres (celle-ci n'existe pas, c'est juste un exemple).

Exemple : Twitter

Voici comment est définit le Data catcher pour Twitter :

```
<?php

return array(
'data_catchers' => array(
    'noviusos_simpletwitter' => array(
        'title' => 'Twitter',
        'description'  => '',
        'iconUrl' => 'static/apps/noviusos_simpletwitter/img/twitter.png',
        'action' => array(
            'action' => 'window.open',
            'url' => 'https://twitter.com/intent/tweet?text={{urlencode:'.\Nos\DataCatcher::TYPE_TITI
        ),
        'onDemand' => true,
        'specified_models' => false,
        'required_data' => array(
            \Nos\DataCatcher::TYPE_TITLE,
        ),
        'optional_data' => array(
            \Nos\DataCatcher::TYPE_URL,
        ),
    ),
),
);
```

Le data catcher **Twitter** nécessite au content nuggets d'avoir un titre. L'URL est optionnelle (mais sera utilisée si elle est fournie).

1.2.7 Multi-Contextes

Principes du multi-Contextes

Novius OS peut nativement gérer plusieurs sites, chacun de ces sites pouvant avoir plusieurs versions linguistiques. Un contexte est un binome site / langue.

Exemple

Votre instance de Novius OS peut gérer votre site vitrine qui se décline en 3 langues (français, anglais et espagnol), votre site pour mobile disponible qu'en français, et un site événementiel en anglais.

Site / Langue	Français	Anglais	Espagnol
Vitrine	X	X	X
Mobile	X		
Événementiel		X	

Votre instance Novius OS gérera alors **5** contextes :
- Vitrine / Français
- Vitrine / Anglais
- Vitrine / Espagnol
- Mobile / Français
- Événementiel / Anglais

Configuration

Pour configurer les différents contextes de votre instance Novius OS, veuillez vous référer à la *documentation d'API*.

Cas particuliers

Qui peut le plus peut le moins. Votre Novius OS peut gérer :
- un seul site dans plusieurs langues
- plusieurs sites dans une seule langue
- un seul site dans une seule langue

L'interface du back-office tient compte de ses différents cas. Le terme _contexte_ s'effacera alors pour ne parler plus que de sites ou de langues.
Il disparaitra même complètement en cas d'un seul site et de une seule langue.

Ajouter des contextes

Vous pouvez ajouter à n'importe quel moment de nouveaux contextes, sites ou langues à votre configuration. Modifiez simplement votre fichier `contexts.config.php`, les nouveaux contextes sont aussitôt pris en compte.

Voir aussi :

Documentation d'API du multi-contextes.

`Contextable` / `Twinnable`

Novius OS est nativement multi-contextes. Mais chaque application décide de la façon dont elle utilise les contextes. Trois cas de figures.

Application n'utilisant pas les contextes

Le cas le plus simple. Une application n'implémente pas la notion de contexte. C'est le cas par défaut, elle n'a rien à faire.

Son contenu sera alors le même quelque soit le contexte et elle pourra être utilisée (via ses *enhancers*) dans n'importe quel contexte.

Application `Contextable`

L'application utilise les contextes. Chaque item est associé à un contexte et ne peut être utilisé que dans son contexte.

Techniquement les tables de l'application auront un champ `context` de type `varchar(25)` contenant le code du contexte et les *Models* de l'application implémenteront le behaviour *Contextable*.

Application `Twinnable`

L'application utilise les contextes et crée des ponts entre-eux. Chaque item est associé à un contexte et peut-être lié à d'autres items de contextes différents.

Techniquement les tables de l'application auront 3 champs :

context de type `varchar(25)` et contenant le code du context de l'item.

common_id_property de type `int` et contenant un ID commun aux items liés entre-eux.

is_main_property de type `boolean`, chaque groupe d'items liés entre-eux a un seul item principal.

Les *Models* de l'application implémenteront le behaviour *Twinnable*.

Exemple Struture de la table d'exemple :
- item_id (clé primaire)
- item_context
- item_common_id_property
- item_is_main_property
- item_title

Créons un premier item :

item_id	item_context	item_common_id_property	item_is_main_property	item_title
1	main : :fr_FR	1	1	Premier item

La colonne `item_common_id_property` prend le même ID que la clé primaire de l'item.
L'item est déclaré item principal, donc la colonne `item_is_main_property` prend la valeur `1`.

Ajoutons un autre item, dans un autre contexte, et lié au premier item :

item_id	item_context	item_common_id_property	item_is_main_property	item_title
1	main : :fr_FR	1	1	Premier item
2	main : :en_GB	1	0	First item

La colonne `item_common_id_property` prend le `item_common_id_property` de l'item auxquel il est lié.
La colonne `item_is_main_property` prend la valeur `0`, ce n'est pas l'item principal.

Observons la table après plusieurs ajouts :

item_id	item_context	item_common_id_property	item_is_main_property	item_title
1	main : :fr_FR	1	1	Premier item
2	main : :en_GB	1	0	First item
3	main : :en_GB	3	1	Second item
4	main : :fr_FR	3	0	Second item (fr)
5	event : :fr_FR	5	1	Item du site event
6	main : :es_ES	1	0	First item (es)

Les items `1`, `2` et `6` sont liés entre-eux et l'item principal est `1` / `main::fr_FR`.
Les items `3` et `4` sont liés entre-eux et l'item principal est `3` / `main::en_GB`.

Supprimons l'item `1` :

item_id	item_context	item_common_id_property	item_is_main_property	item_title
2	main : :en_GB	*1*	**1**	First item
3	main : :en_GB	3	1	Second item
4	main : :fr_FR	3	0	Second item
5	event : :fr_FR	5	1	Item du site vent
6	main : :es_ES	*1*	0	First item

L'item `2` a récupéré le rôle principal, mais l'`item_common_id_property` du `2` et du `6` n'a **pas** changé.

1.2.8 Médiathèque

Principe

La médiathèque est un point central qui regroupe la majorité des fichiers utilisés par les applications. Elle contient des images, des documents, des vidéos ou tout autre type de fichier.
- Les fichiers sont stockés dans le répertoire **privé** `/novius-os/local/data/media/`
- On y accède par l'URL `http://your.website.com/media/folder/ressource.ext`

Fonctionnement

Lors du premier accès au média, le gestionnaire 404 est appelé et un lien symbolic est alors créé dans `public/media` et les requêtes suivantes n'auront plus besoin de gestionnaire 404.

La raison de ce fonctionnement est pour l'ajout futur de médias **privé**. Pour ces derniers, sera retourné :
- un code d'erreur HTTP 401 (autorisation nécessaire) ;
- soit le fichier sera envoyé sur la sortie standard, mais sans création de lien symbolique (le droit d'accès est vérifié lors de chaque requête).

Optimisation

Lorsque PHP envoie le fichier, le processus est bloqué jusqu'à ce que la totalité du fichier soit transféré. Il est néanmoins possible de libérer le processus instantanément en déléguant l'envoi du fichier au serveur web sous-jacent (Apache ou nginx).

Le mécanisme utilisé s'appelle XSendfile et consiste à envoyer un header spécial depuis le script PHP. Le nom de ce header varie d'un serveur à l'autre :
- `X-Sendfile` est utilisé par **Apache** et d'autres ;
- `X-Accel-Redirect` est utilisé par **nginx**.

Voir aussi :

Optimisations après installation

Fichiers joints (hors médiathèque)

Vous n'avez pas forcément envie de stocker tous vos fichiers dans la médiathèque. Par exemple, si vous avez une application « Offres d'emploi » qui reçoit des candidatures, vous ne souhaitez pas que les CV des candidats soient visibles dans la médiathèque.

Pour traiter ce cas, il existe un mécanisme de fichiers indépendants *Attachment*. À la manière des pièces jointes dans un e-mail, il est ainsi possible de joindre un fichier CV à une candidature.

1.2.9 Permissions et droits d'accès

Rôles (ou "profils")

Les permissions s'appliquent systématiquement sur un rôle. Ensuite, chaque utilisateur se voit attribuer un ou plusieurs rôles, dont il hérite les droits d'accès.

Un rôle par utilisateur

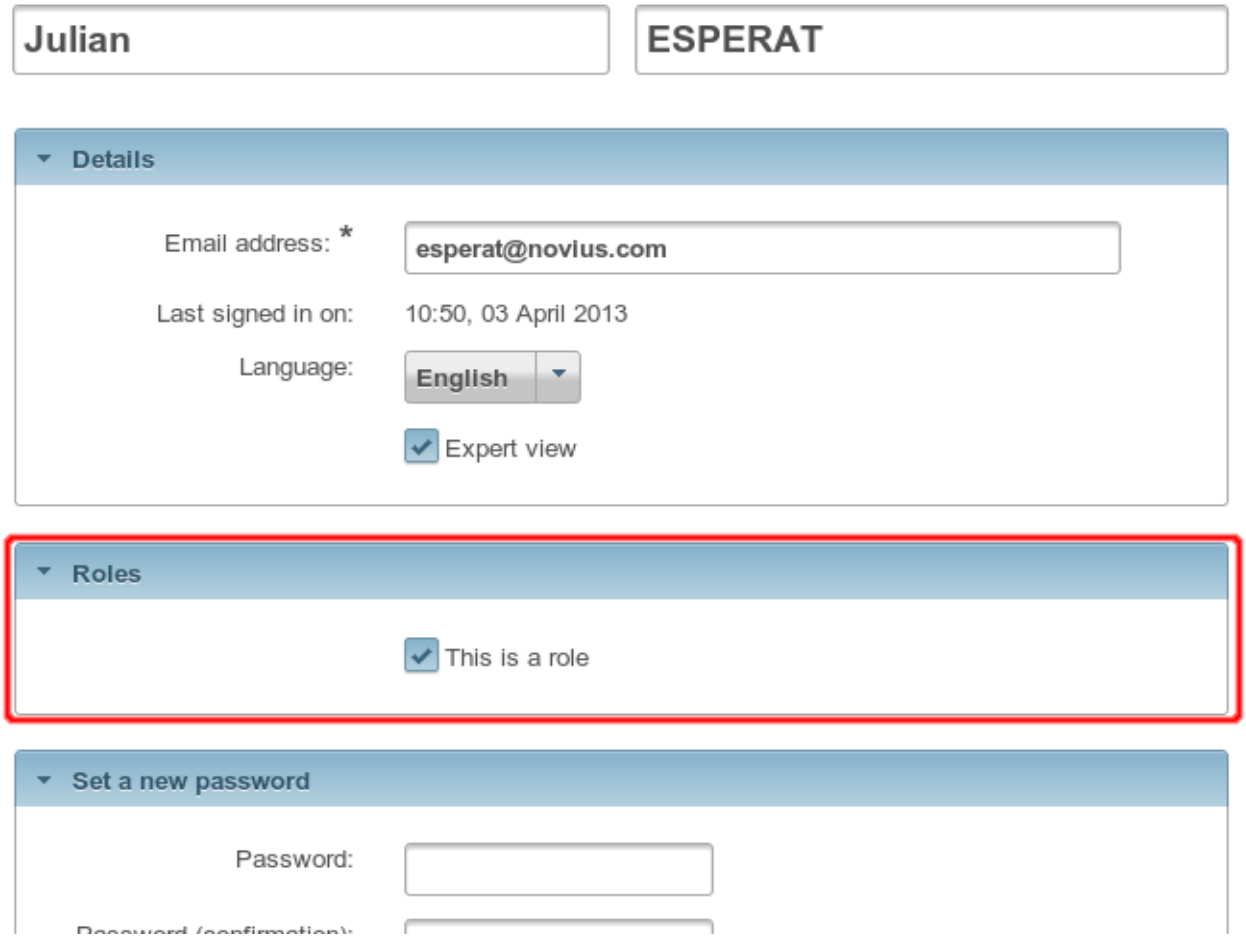
Pour des raisons de simplicité et de compréhension, dans une installation par défaut :
- ces rôles sont cachés ;
- chaque utilisateur se voit automatiquement attribuer un seul rôle ;
- il n'est pas possible de partager un rôle entre plusieurs utilisateurs.

La notion de rôle s'efface complètement et les droits d'accès se configurent via un onglet dédié sur la fiche d'un utilisateur :

Plusieurs rôles par utilisateur

Il est cependant possible d'activer la gestion des rôles multiples dans le fichier de configuration principal : `novius-os.users.enable_roles = true`.

Une fois activés, il devient donc possible de partager un rôle entre plusieurs utilisateurs. Les permissions peuvent alors se configurer plus finement :
- sur la fiche d'un utilisateur, l'onglet "Droits d'accès" disparait au profit d'un bloc "Rôles" ;
- l'AppDesk de l'application Utilisateurs s'enrichit :
  - d'un nouvel inspecteur "Rôles" ;
  - une nouvelle action "Ajouter un rôle".
- les droits d'accès se configurent désormais sur la fiche d'un rôle.



Structure d'un droit d'accès

Il existe deux types de droits d'accès :
– les simples : oui ou non ;
– les multiples : applicables sur une liste de catégories.
Le droit **simple** a du sens en lui-même, par exemple "Est-ce que je peux ajouter une page ?" ou encore "Est-ce que je peux supprimer une page verrouillée ?".

Les droits **multiples** n'ont pas de sens seuls, ils s'expriment uniquement en fonction d'une catégorie. Par exemple, "Est-ce que je peux écrire dans ce dossier ?" a besoin d'une liste de dossiers sur lesquels s'appliquer, ou bien "Est-ce que j'ai accès à cette application ?" a besoin de la liste des applications pour s'exprimer.

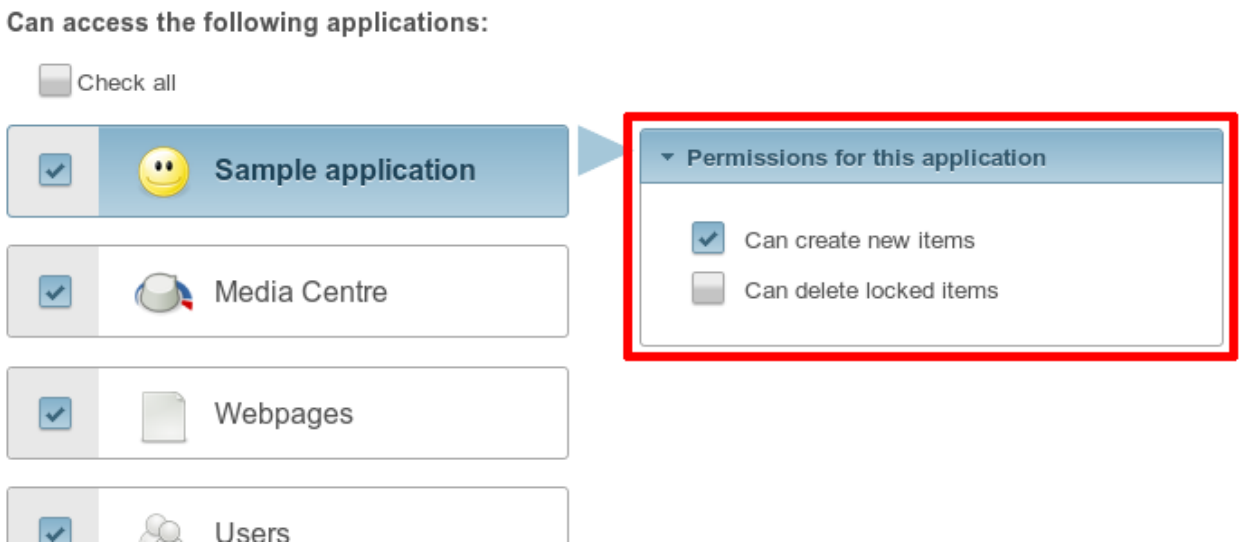
Un droit **simple** est composé d'une seule colonne `perm_name`, tandis qu'un droit **multiple** (= droit par catégorie) est composé de deux colonnes : `perm_name` (comme pour le droit simple) et `perm_category_key`.

Utilisation dans les applications

Fichier `permissions.config.php`

Grâce à ce fichier, chaque application peut définir la liste des permissions qu'elle souhaite configurer.

L'affichage se fait dans une colonne dédiée à droite du nom de l'application lors de l'édition des permissions :



Voir aussi :

API associée au fichier de configuration des permissions

API pour vérifier une permission

```
<?php
// Simple : 1 seul argument, le nom de la permission
Permission::check('noviusos_app::delete_locked');

// Multiple (réglable par catégories) : 2 arguments, le nom de la permission + la clé de la catégorie
Permission::check('noviusos_app::create_in_folder', $folder_id);
```

Voir aussi :

Documentation d'API pour la classe *Permission*.

> **Warning :** Le nom de la permission est un élément important. La partie avant les `::` doit représenter un nom d'application valide. Pour que la permission soit validée, il faut également que l'utilisateur ait accès à cette application.

CRUD

Il est possible de cacher des champs en fonction des permissions. Pour cela, il faut utiliser la clé *show_when* qui définit une fonction de callback retournant si le champ doit être visible ou non.

```
<?php
return array(
    'fields' => array(
        'my_field' => array(
            'label' => 'My field',
            'form' => array(
                'type' => 'text',
            ),
            'show_when' => function() {
                // The field will only be visible when the user has the requested permission
```

```
                return Permission::check('my_app::my_permission');
            },
        ),
    ),
);
```

Actions

Il est possible de désactiver des actions en fonction des permissions grâce à la clé *disabled*.

```
<?php
return array(
    'data_mapping' => array(/*...*/),
    'actions' => array(
        'delete' => array(
            'label' => __('Delete'),
            'primary' => false,
            'icon' => 'home',
            'action' => array(/*...*/),
            'targets' => array(
                'grid' => true,
            ),
            'disabled' => array(
                function($item) {
                    return !Permission::check('my_app::can_delete_item') ? __('You don\'t have the p
                }
            ),
        ),
    ),
);
```

1.2.10 Front-Office et cache

Novius OS utilise le **mod_rewrite** d'**Apache** (ou l'équivalent sur un autre serveur) pour afficher les pages en front-office.

Toutes les URLs finissant par `.html`, la home et les répertoires sont redirigées vers le fichier `NOSROOT/public/htdocs/novius-os/front.php`.

Ce fichier charge Novius OS et délègue le traitement de l'URL au *Controller_Front*.

Le *Controller_Front* analyse l'URL et détermine le chemin du fichier de cache correspondant. À partir de là, plusieurs possibilités.

Exécution du cache

Le fichier de cache correspondant à l'URL existe.

Le cache des pages est enregistré dans le répertoire `NOSROOT/local/cache/page/`. Le premier niveau d'arborescence correspond aux domaines des URLS. Le cache recrée ensuite l'arborescence de l'URL.

Le fichier de cache en lui-même est un fichier PHP. Le début du fichier contient toujours une vérification de la validité du cache par rapport à sa durée de vie. Il contient également un mécanisme pour recréer les propriétés du *Controller_Front* qui étaient disponibles au moment de sa génération (page, URLs, status, headers, custom data).

Ce fichier de cache est exécuté et, s'il est toujours valide, ce qu'il affiche est renvoyé à l'internaute, avec le `status` et les `headers` enregistrés.

Génération du cache

Si le fichier de cache correspondant à l'URL n'existe pas ou n'est plus valide.

Le *Controller_Front* va déterminer la page appelée en fonction de l'URL. La page connue, il récupère le template qui lui est lié. Puis le ou les WYSIWYG de la page venant s'insérer dans le template.

Si les WYSIWYG contiennent des enhancers, ces enhancers sont exécutés et leur résultat enregistré.

Puis le template est exécuté comme une View en prenant comme paramètre le tableau des WYSIWYG (`$wysiwyg`), le titre de la page (`$title`), la page (`$page`) et le *Controller_Front* (`$main_controller`).

Le contenu généré est enregistré dans le fichier de cache.

Puis le fichier de cache est exécuté comme expliqué ci-dessus, et renvoie le contenu à l'internaute, avec le `status` et les `headers` potentiellement spécifiés lors du processus de génération du cache (notamment par les enhancers).

Interactions possibles

Au cours du processus, le *Controller_Front* envoie différents événements à des moments clés. En interceptant ces événements vous pouvez influencer ou modifier le traitement.

Voir aussi :

Événements front-office

Vous pouvez également influencer ou modifier le traitement par les méthodes du *Controller_Front* en récupérant son instance, soit par *NosNos : :main_controller()*, soit via `$this->main_controller` dans un enhancer.

Modifier le contenu généré

Dans certains cas, vous pouvez vouloir générer un contenu de sortie sans tenir compte du template, par exemple pour renvoyer un flux RSS depuis un enhancer. Pour cela, utiliser la méthode `sendContent()` du *Controller_Front*.

Voici un exemple de code d'un enhancer :

```
<?php

$this->main_controller->setHeader('Content-Type', 'application/xml');
$this->main_controller->setCacheDuration(60 * 30); // La durée de cache est fixée à 30 minutes
return $this->main_controller->sendContent($rss); // La variable $rss contient le flux RSS
```

Le fichier de cache ne contiendra alors que le contenu du flux RSS et la réponse HTTP enverra un header pour spécifier le `content-type`.

Exécution hors cache

Dans certain cas, le système de cache peut-être trop efficace. Par exemple, si une partie du template ou d'un enhancer doit être différent en fonction de si l'utilisateur est identifié ou non. Dans ce cas il est utile d'enregistrer dans le cache le code PHP a exécuter et non pas son résultat.

Pour cela, il suffit d'utiliser les méthodes `callHmvcUncached()` et `viewForgeUncached` de la classe *FrontCache*.

```php
<?php

\Nos\FrontCache::callHmvcUncached(
    'uri/controller',
    array(
        'id' => \Front_Utilisateur::get_current_user_id() // C'est un exemple, la classe \Front_Util
    )
);

// ou

\Nos\FrontCache::viewForgeUncached(
    'uri/view', // Le chemin d'une vue
    array(
        'id' => \Front_Utilisateur::get_current_user_id()
    ),
    false
);
```

Suffix Handler

Si vous voulez que le chemin de cache tienne compte d'autre chose que de la simple URL. Par exemple, que le cache tienne compte d'un paramètre GET (par défaut, le même cache est utilisé que l'URL ai ou n'ai pas de paramètre GET).

Pour cela, il suffit d'utiliser la méthode `addCacheSuffixHandler()` du *Controller_Front*.

```php
<?php

\Nos\Nos::main_controller->addCacheSuffixHandler(array(
    array(
        'type' => 'GET',
        'keys' => array('my_param'),
    ),
));

// ou

\Nos\Nos::main_controller->addCacheSuffixHandler(array(
    array(
        'type' => 'callable',
        'callable' => array('MyClasse', 'myMethod'),
        'args' => array(
            'argument pour exemple'
        ),
    ),
));
```

Dans le 1er cas, le système de cache gérera un fichier différent pour une même URL ayant un paramètre GET `my_param` avec une valeur différente.

Dans le second cas, le système de cache appellera la méthode `MyClasse::myMethod('argument pour exemple')`, charge à la méthode de renvoyer un suffixe au fichier de cache si besoin.

1.3 Étendre une application

1.3.1 Mécanismes d'extensions

Créer un fichier dans `local`

On peut modifier n'importe quel fichier de vue ou de configuration via le dossier `local`.

Ceci est possible grâce au comportement des fichiers chargés « en cascade » existant dans FuelPHP et adapté dans Novius OS. C'est très simple à faire, car il suffit de copier un fichier existant et de le modifier à notre guise !

Dans le cas d'une application, il faut copier le fichier dans `local/config/apps/{application}/` ou `local/views/apps/{application}/`.

Pour étendre un fichier du moteur de Novius OS, on utilisera `local/config/apps/novius-os/` et `local/views/apps/novius-os/`.

La syntaxe « générique » est donc `local/{section}/{application}/` avec :
- `{section}` ' est égal à `config` ou `views` ;
- `{application}` ' correspond à `apps` + un nom d'application ou `novius-os` pour le moteur.

Configuration

L'application `noviusos_page` possède un fichier de configuration `controller/admin/appdesk.config.php` (le fichier se situe donc dans `noviusos_page::config/controller/admin/appdesk.config.php`).

Si on le copie dans `local/config/apps/noviusos_page/controller/admin/appdesk.config.php` alors ce dernier sera fusionné automatiquement avec celui de l'application qui le demande.

Vues

Lorsqu'on créé le fichier `local/views/apps/noviusos_help/admin/help.view.php` ce dernier est utilisé en **remplacement** de `noviusos_help::admin/help.view.php` !

Pour étendre un fichier du moteur, on utilisera le nom d'application `novius-os`. Par exemple, on créera le fichier `local/views/novius-os/admin/login.view.php`.

Utiliser les évènements pour modifier une configuration

N'importe quel fichier de configuration peut être modifié grâce à l'évènement *config|<path>*.

Remplacer une vue par une autre

Il est possible de faire appel à la méthode `View::redirect()` pour remplacer un fichier de vue par un autre.

```
<?php

// Remplace la vue 'admin/help' de l'application 'noviusos_help' par la vue 'help' du dossier 'local'
View::redirect('noviusos_help::admin/help', 'local::help');
```

Créer une application dédiée d'extension

Pour étendre une application, on créé une autre application qui va modifier la première.

L'application 2 définit qu'elle étend `mon_application` via son fichier `metadata.config.php` :

```
<?php

return array(
    'name' => 'Application 2',
    // On définit que c'est une application d'extension
    'extends' => 'mon_application',
);
```

Une fois `application_2` installée, elle sera chargée en même temps que `mon_application`.

Lorsqu'une application étend une autre, certains comportements deviennent automatiques.

Exemple :

`application_2` étend `mon_application`.

Les fichiers de configurations des `Controller` et des `Model` de `mon_application` peuvent être automatiquement étendus par `application_2` en les créant au même endroit.

Exemple, `mon_application` définit le fichier de configuration suivant pour `Controller_Test` : `applications/mon_application/config/controller/test.config.php`

Si dans `application_2`, le fichier correspondant `applications/application_2/config/controller/test.config` existe, alors il sera fusionné.

C'est-à-dire que dans `Mon\Application\Controller_Test`, la variable `$config` contiendra la fusion 2 fichiers (celui de l'application étendue `mon_application`, et aussi celui de `application_2` qui étend la première).

1.3.2 Le bootstrap

Le fichier bootstrap permet d'exécuter du code php lors du chargement d'un site / d'une application. Il est possible d'en créer à deux emplacements :
- `local/bootstrap.php` : s'éxecutera lors du chargement du site.
- `local/applications/APPLICATION/bootstrap.php` : s'éxecutera lors du chargement de l'application `APPLICATION`.

L'intérêt d'un tel fichier est qu'il permet de facilement étendre une application. C'est dans ces fichiers que l'on pourra utiliser les *événements* et les *redirections de vue*.

1.3.3 Ajouter un champ

Nous allons partir d'un exemple pour l'explication.

Ajoutons un champ `Source` pour un billet de blog, qui permettra de renseigner une URL externe ayant produit le contenu original.

Dans la BDD

```
ALTER TABLE `nos_blog_post` ADD `post_source` VARCHAR(255);
```

Dans le Model

Deux possibilités :
- Déclarer la nouvelle colonne dans les `properties` du model.
- Activer le système de cache des `properties` des models.

Déclarer la colonne

Pour déclarer votre nouvelle colonne nous allons écouter l'événement qui charge le fichier de configuration du model.

```
<?php

Event::register_function('config|noviusos_blog::model/post', function(&$config) {
    $config['properties']['post_source'] = array(
        'default' => null,
        'data_type' => 'varchar',
        'null' => false,
    );
});
```

Voir aussi :

Définition des properties dans la documentation de FuelPHP

Activer le cache des `properties`

- Créez le fichier `local/config/config.php` à partir du fichier `local/config/config.php.sample` si ce n'est pas encore fait.
- Décommentez la ligne (ou créez la) ayant la clé `cache_model_properties` et lui assigner `true` :

```
<?php

return array(
    //...

    'novius-os' => array(
        //...
        'cache_model_properties' => true,

        //...
    ),
);
```

Une fois activé, le système va mettre toutes les `properties` des models en cache dans le répertoire `local/cache/fuelphp/model_properties/`. Quand une colonne est ajoutée et non déclarée, au premier appel à `get()` ou à `set()` pour cette colonne, les `properties` seront mise à jour avec une requête en base pour lister les colonnes du model.

> **Warning :** Ce mécanisme n'est possible qu'avec les drivers MySQL et MySQLi.

Voir aussi :

La documentation sur la configuration de Novius OS.

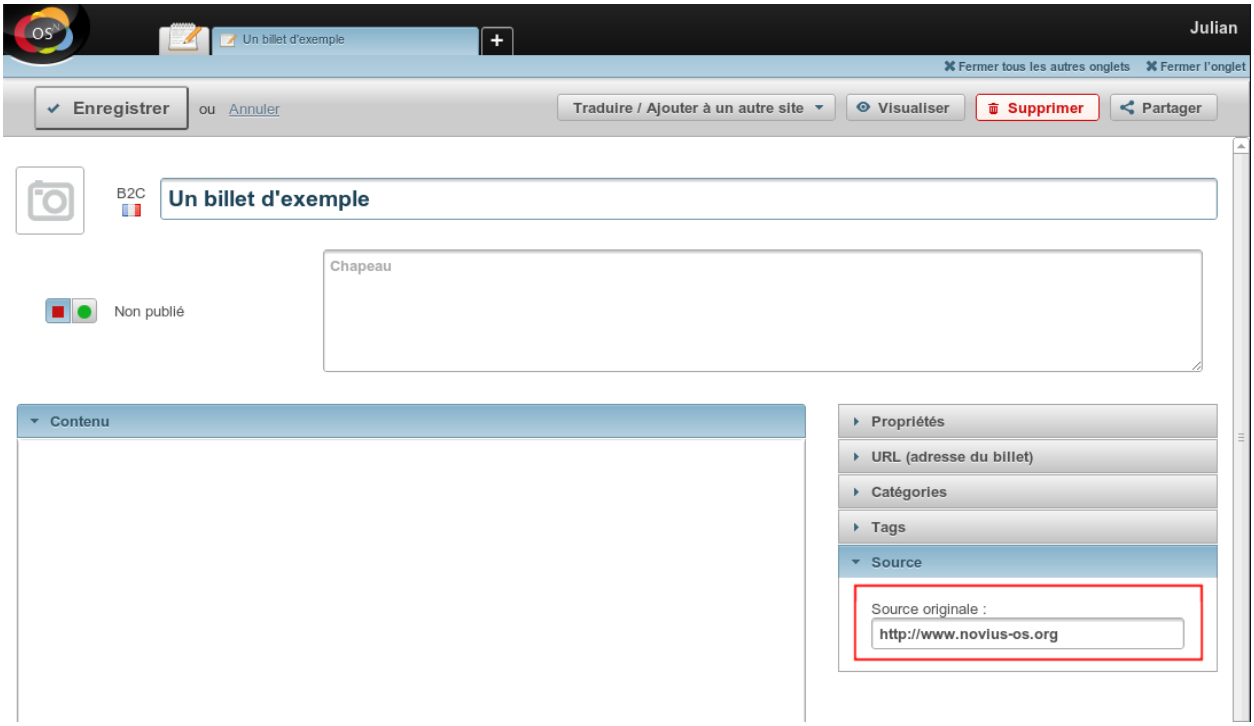
Dans le formulaire

Le formulaire d'ajout / édition d'un billet de blog est définit dans sa configuration CRUD. Pour l'étendre, nous allons utiliser un évènement !

Dans le fichier `local/bootstrap.php` (créez-le si nécessaire) :

```
<?php

Event::register_function('config|noviusos_blog::controller/admin/post', function(&$config) {

    // Ajout du champ 'post_source' de type 'text'
    $config['fields']['post_source'] = array(
        'label' => 'Source originale :',
        'form' => array(
            'type' => 'text',
            'placeholder' => 'http://',
        ),
    );

    // Affichage du champ dans le formulaire
    // Nous créons une entrée intitulée 'Source' dans le menu de droite
    $config['layout']['menu']['Source'] = array('post_source');
});
```

Le formulaire possède désormais un champ éditable supplémentaire, comme vous pouvez le voir ci-dessous :



Dans la visualisation

Pour la vue, nous créer le fichier `local/views/apps/noviusos_blognews/front/post/content.view.php`

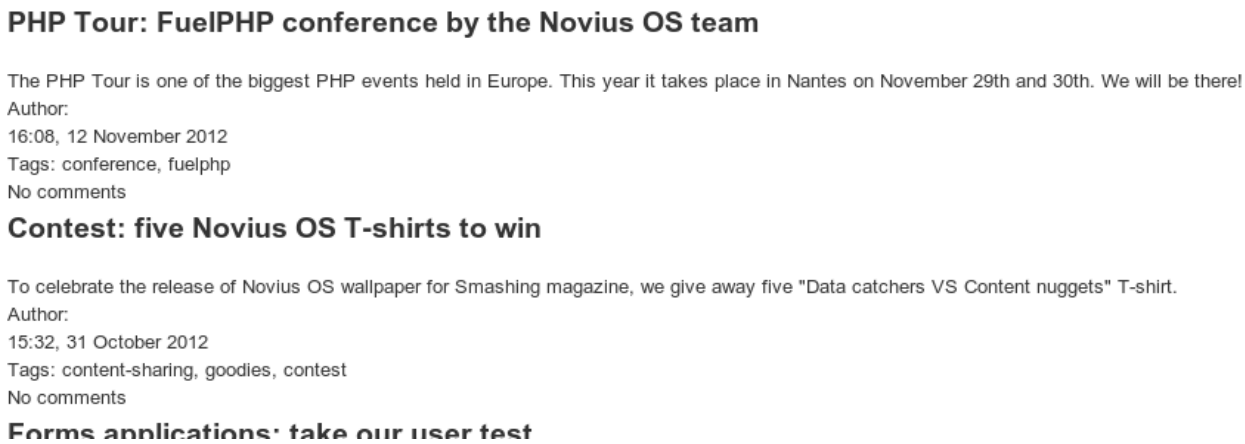
```
<?php

// On inclut le fichier d'origine (qui affiche le contenu)
include APPPATH.'/applications/noviusos_blognews/views/front/post/content.view.php';

// On rajoute la source à la fin
if (!empty($item->post_source)) {
    ?>
    <p class="blognews_source">
        <?= __('Source:') ?>
        <a href="<?= htmlspecialchars($item->post_source) ?>">
            <?= htmlspecialchars($item->post_source) ?>
        </a>
    </p>
    <?php
}
```

1.3.4 Modifier l'affichage sur le site

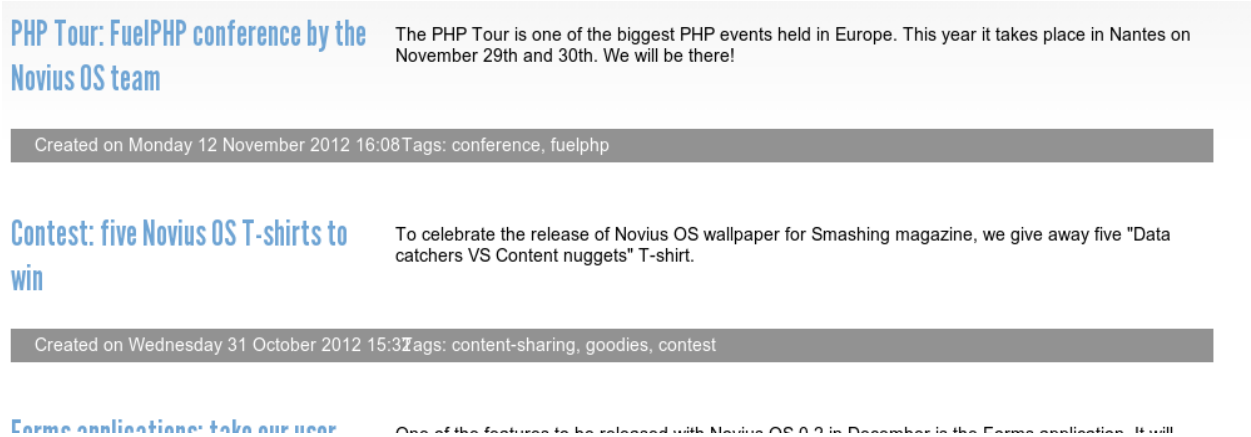
Nous allons partir d'un exemple pour l'explication.

Sur le site Novius OS nous avons personnalisé l'affichage des billets de blog. Voyons-voir comment cela fonctionne.

Apparence par défaut de l'application blog :



Apparence sur le site Novius OS.org (notre objectif) :



Modification de la vue

1ère technique : étendre la vue

Grâce à la cascade du système de fichiers, on peut copier le fichier original `noviusos_blognews::views/front/post/item.view.php` dans notre dossier `local` : `local::views/apps/noviusos_blognews/front/post/item.view.php`

```
<div class="blognews_post blognews_post_item">
    <div class="blognews_primary_information">
        <?= \View::forge('noviusos_blognews::front/post/title', array('item' => $item)) ?>
        <?= \View::forge('noviusos_blognews::front/post/summary', array('item' => $item)) ?>
    </div>
    <div class="blognews_secondary_information">
        <?= \View::forge('noviusos_blognews::front/post/publication_date', array('item' => $item)) ?
        <?= \View::forge('noviusos_blognews::front/post/tags', array('item' => $item)) ?>
    </div>
</div>
```

Nous avons supprimé l'affichage de la vignette, de l'auteur, des catégories et du nombre de commentaires.

2e technique : étendre la configuration

L'application blog permet dans sa configuration de désactiver l'affichage de certains éléments. En ce qui nous concerne, c'est possible pour tous ceux qu'on souhaite ne pas afficher, sauf pour la vignette.

Lorsqu'on utilise le fichier de configuration de l'application blog, cette dernière modifie l'affichage à la fois dans la liste des billets, mais également sur la fiche, ce qui ne nous convient pas dans notre cas (cette technique est donc montrée ici à titre d'exemple).

Grâce à la cascade du système de fichiers, on peut copier le fichier original `noviusos_blognews::config/config.php` dans notre dossier `local` : `local::config/apps/noviusos_blognews/config.php` :

```
<?php

// On laisse uniquement les clés que l'on souhaite modifier
return array(
    'categories' => array(
        'show' => false,
    ),
    'authors' => array(
        'show' => false,
    ),
    'comments' => array(
        'show' => false,
    ),
);
```

Ajout du CSS

1ère technique : étendre la vue

Nous créer le fichier `local::views/apps/noviusos_blognews/front/post/list.view.php`

```php
<?php

// On ajoute notre fichier CSS spécifique
\Nos\Nos::main_controller::addCss('static/css/blog_custom.css');

// On inclut le fichier d'origine (qui affiche la liste des billets)
include APPPATH.'applications/noviusos_blognews/views/front/post/list.view.php';
```

Notre vue modifiée inclut d'abord un fichier CSS (à créer dans `public/static/css/blog_custom.css`), puis appelle la vue d'origine.

2e technique : agir directement sur le gabarit

Il est également possible d'inclure le fichier CSS via l'évènement `front.start`, mais dans ce cas, il le sera sur toutes les pages de votre site, et pas seulement sur la page blog.

Dans le fichier `local/bootstrap.php` (créez-le si nécessaire) :

```php
<?php

// Événement qui se déclenche lorsqu'on charge une page du site
Event::register('front.start', function() {
    \Nos\Nos::main_controller::addCss('css/blog_custom.css');
});
```

Dans le cas du site Novius OS, nous avons créés nos gabarits spécialement pour le site, ils incluent directement le CSS nécessaire à la personnalisation de l'affichage du blog.

1.3.5 Ajouter une action en back-office

Les actions sont définies dans le fichier `config/common/{model}.config.php`.

Le meilleur moyen est de s'inspirer des actions par défaut qui existent dans Novius OS.

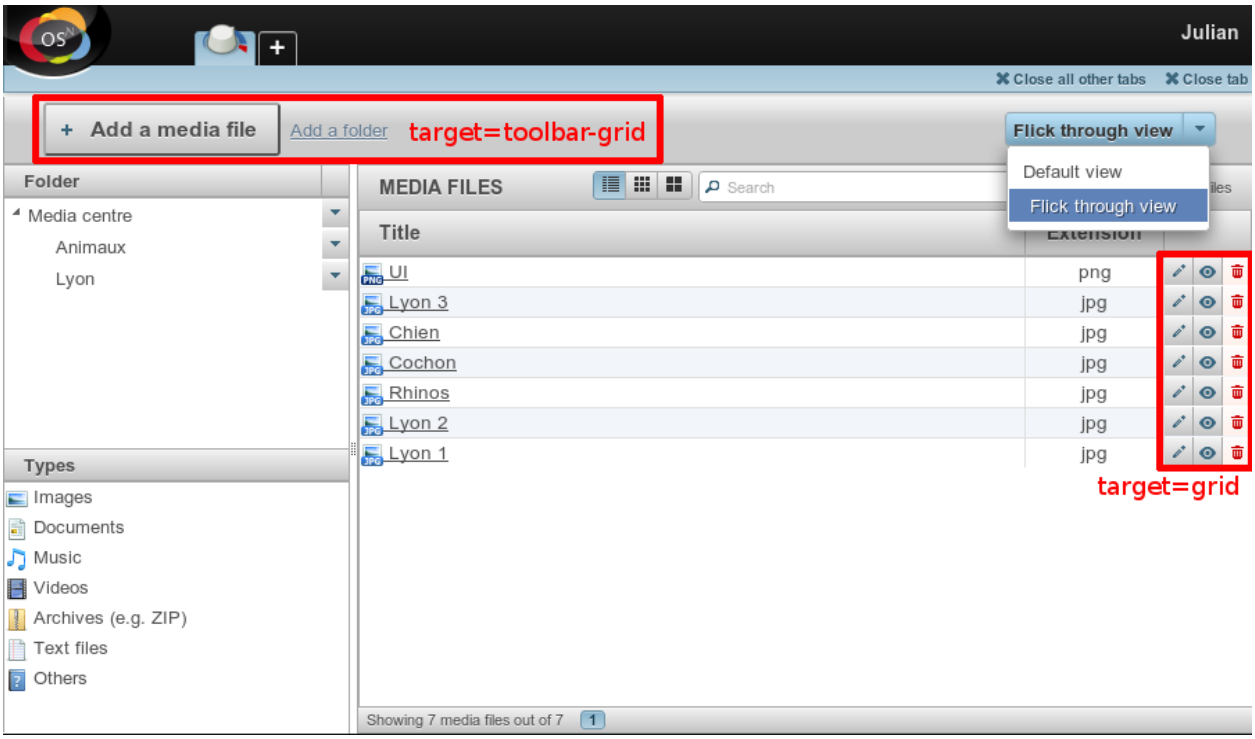
Placeholders

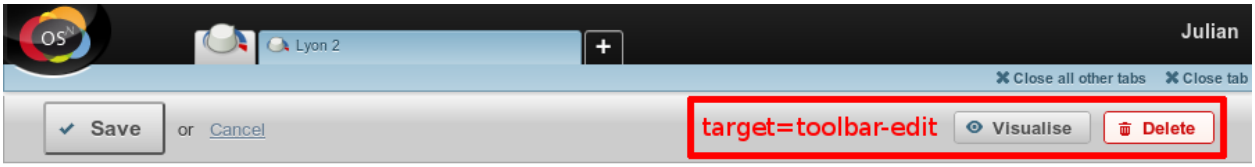
La configuration des actions contient des `{{placeholders}}`.
- Sont remplacés en **PHP** :
  - `{{model_label}}` : le nom du modèle
  - `{{controller_base_url}}` : l'URL du contrôleur associé au modèle
- **Sont remplacé par l'App Desk (en JavaScript) :**
    - `{{context}}` : contexte actuellement affiché (ou premier contexte lorsque plusieurs sont affichés)

Tous les autes placeholders sont remplacés par les données de **l'item** : `{{_id}}` et `{{_title}}` dans ce cas, mais égalements n'importe quel champ définit dans le `data_mapping`.

Cibles des actions

Il y a 3 cibles possibles pour les actions :
- **toolbar-grid** : barre d'outils de l'appdesk
- **grid** : ligne d'item dans la grille principale de l'appdesk
- **toolbar-edit** : barre d'outils sur le formulaire d'ajout / édition





Lyon 2

Change the file: Choisissez un fichier Aucun fichier choisi

SEO, Media URL: lyon-2 .jpg Use title

Select a folder where to put your media file:

Media centre
Animaux
Lyon

Add / Edit / Delete

L'action **add** :

- ouvre un nouvel onglet ;
- fait appel à la méthode `action_insert_update()` du contrôleur `Nos\Media\Controller_Admin_Media` ;
- avec un paramètre `$_GET['context']` qui permet de pré-sélectionner le contexte actif ;
- s'affiche uniquement dans la barre d'outils de l'App Desk.

L'action **edit** :
- ouvre l'onglet d'édition (la méthode n'étant pas précisée pour `nosTabs`, la valeur par défaut `open` sera utilisée : elle donnera le focus à l'onglet existant s'il est déjà ouvert, ou en créera un nouveau le cas échéant) ;
- fait appel à la méthode `action_insert_update( $id )` du contrôleur `Nos\Media\Controller_Admin_Media` ;
- avec un paramètre `id` passé à la méthode ;
- s'affiche uniquement dans la grille principale.

L'action **delete** :
- fait appel à la méthode `action_delete( $id )` du contrôleur `Nos\Media\Controller_Admin_Media` ;
- avec un paramètre `id` passé à la méthode ;
- s'affiche à la fois dans la grille principale et sur le formulaire d'édition, mais uniquement pour les items existants (pas lors de l'ajout).

```php
<?php

return array(

    // Acion par défaut ADD
    'add' => array(
        'label' => __('Add {{model_label}}'),
        'primary' => true,
        // Le clic ouvrira un nouvel onglet
        'action' => array(
            'action' => 'nosTabs',
            'method' => 'add',
            'tab' => array(
                'url' => '{{controller_base_url}}insert_update?context={{context}}',
            ),
        ),
        // L'action sera affichée uniquement dans la barre d'outils de l'App Desk
        'targets' => array(
            'toolbar-grid' => true,
        ),
    ),

    // Action par défaut EDIT
    'edit' => array(
        'label' => __('Edit'),
        'primary' => true,
        'icon' => 'pencil',
        // Le clic ouvrira (ou rendra le focus) l'item
        'action' => array(
            'action' => 'nosTabs',
            'tab' => array(
                'url' => "{{controller_base_url}}insert_update/{{_id}}",
                'label' => '{{_title}}',
            ),
        ),
        // L'action s'affiche uniquement dans la grille principale
        'targets' => array(
            'grid' => true,
        ),
    ),

    // Action par défaut DELETE
    'delete' => array(
        'label' => __('Delete'),
        'primary' => true,
```

```
        'icon' => 'trash',
        'red' => true,
        // Le clic ouvrira une popup de confirmation
        'action' => array(
            'action' => 'confirmationDialog',
            'dialog' => array(
                'contentUrl' => '{{controller_base_url}}delete/{{_id}}',
                'title' => strtr($config['i18n']['deleting item title'], array(
                    '{{title}}' => '{{_title}}',
                )),
            ),
        ),
        // L'action s'affiche à la fois dans la grille principale et sur le formulaire d'édition...
        'targets' => array(
            'grid' => true,
            'toolbar-edit' => true,
        ),
        // ...mais pas pour les nouveaux items !
        'visible' => function($params) {
            return !isset($params['item']) || !$params['item']->is_new();
        },
    ),
);
```

1.3.6 Modifier un comportement en front-office

Novius OS fournit un mécanisme d'évènements destiné à interagir avec le moteur.

Il existe 2 types d'évènements :
- Ceux qui servent à modifier une / des valeur(s) ;
- Ceux qui notifient qu'une action s'est produite.

```
<?php

// Exemple d'évènement qui écoute une action
Event::register('event_name', function($value)
{
    // L'action 'event_name' s'est produite
});

<?php

// Exemple d'évènement qui modifie une valeur
Event::register_function('event_name', function(&$value)
{
    // $value peut être modifiée
});
```

Pour une liste des évènements, il faut se référer la documenation API associée.

Voir aussi :

Events

Faire une redirection 301 en fonction de l'URL

Il est possible de créer une page en back-office avec comme type *Lien externe* pour faire une redirection 301.

Il est également possible de les configurer via un fichier `./htaccess`.

Enfin, on peut aussi le faire directement via le code, comme présenté ci-dessous.

```
<?php

Event::register_function('front.start', function($params)
{
    // Utilisation de la classe Str de FuelPHP
    if (Str::starts_with($params['url'], 'an-old-url'))
    {
        // Note : 10 == strlen('an-old-url')
        $new_url = 'my-new-url'.substr($params['url'], 10);

        // Utilisation de la classe Response de FuelPHP
        Response::redirect($new_url, 'location', 301);
    }
});
```

Envoyer un mail remerciement dans un formulaire de contact

```
<?php

Event::register_function('noviusos_form::after_submission', function(&$answer, $enhancer_args)
{
    foreach ($answer->fields as $field)
    {
        if ($field->anfi_field_type == 'email' && !empty($field->anfi_value)
        {
            $email = Email::forge();
        $email->from('mon@email.me', 'Mon Nom');
        $email->to($field->anfi_value);
        $email->subject('Votre demande de contact');

        // Email au format texte (use html_body() instead if you want to send HTML email)
        $email->body('Merci pour votre demande de contact, elle a bien été reçue. Nous allons y répon

        try
        {
            $email->send();
        }
        catch(\Exception $e)
        {
            // Could not send the email
        }
        }
    }
});
```

1.4 Créer une nouvelle application

1.4.1 Assistant "Créer mon appli"

L'assistant "Créer mon appli" permet de générer facilement et rapidement les bases d'une nouvelle application : Modèles, champs et groupe de champs, App Desk, launchers, URL enhancers, etc.

En industrialisant la création d'une application, l'assistant vous permet d'être plus productif et de vous concentrer sur l'essentiel.

Warning : La dernière étape de l'assistant "Créer mon appli" est la création de la base données et des fichiers de votre nouvelle application. Ces fichiers seront dans un nouveau répertoire, au nom de votre application, dans `local/application/`. Novius OS (le `user Apache` si Novius OS tourne sur un serveur **Apache**) doit donc avoir les droits d'écriture dans ce répertoire.

1.4.2 Créer un enhancer

1. Définition dans le fichier `metadata`

Les metadata d'un enhancer sont décrites dans la *documentation d'API*.

2. [Back-office] Créer un contrôleur pour l'enhancer

Pour gérer la popup de configuration et la prévisualisation de l'enhancer, nous avons besoin d'un contrôleur.

Créez donc le fichier `mon_appli::classes/controller/admin/enhancer.ctrl.php` en étendant le `Controller_Admin_Enhancer` adéquat.

```
<?php

namespace Mon\Appli;

class Controller_Admin_Enhancer extends \Nos\Controller_Admin_Enhancer
{

}
```

Comme tous les contrôleurs de Novius OS, ajoutons lui un fichier de configuration `mon_appli::config/controller/admin/enhancer.config.php`

```
<?php

return array(
    // Vide pour l'instant, nous allons le remplir plus bas
);
```

Popup de configuration

L'affichage ou non d'une popup de configuration est conditionnée par la présence ou non de la clé `dialog` dans le fichier `metadata.config.php`. Ajoutons cette information :

```
<?php

return array(
    'enhancers' => array(
        'mon_appli' => array(
            'dialog' => array(
                'contentUrl' => 'admin/mon_appli/enhancer/popup',
                'ajax' => true,
            ),
```

```
        ),
    ),
);
```

Ici, la popup fera appel à la méthode `action_popup()` de la classe `Mon\Appli\Controller_Admin_Enhancer`. Ce dernier est prévu pour fonctionner en Ajax.

Comme pour toute modification du fichier `metadata.config.php`, il faut aller appliquer les changements depuis le gestionnaire d'applications.

Désormais, lorsqu'on ajoute notre enhancer dans un WYSIWYG, une popup s'affiche, mais il n'y a aucune option à configurer !

Le contrôleur standard que nous avons étendu prévoit d'être configuré pour ajouter les options de l'enhancer. Rendez-vous dans le fichier de configuration `mon_appli::config/controller/admin/enhancer.config.php` :

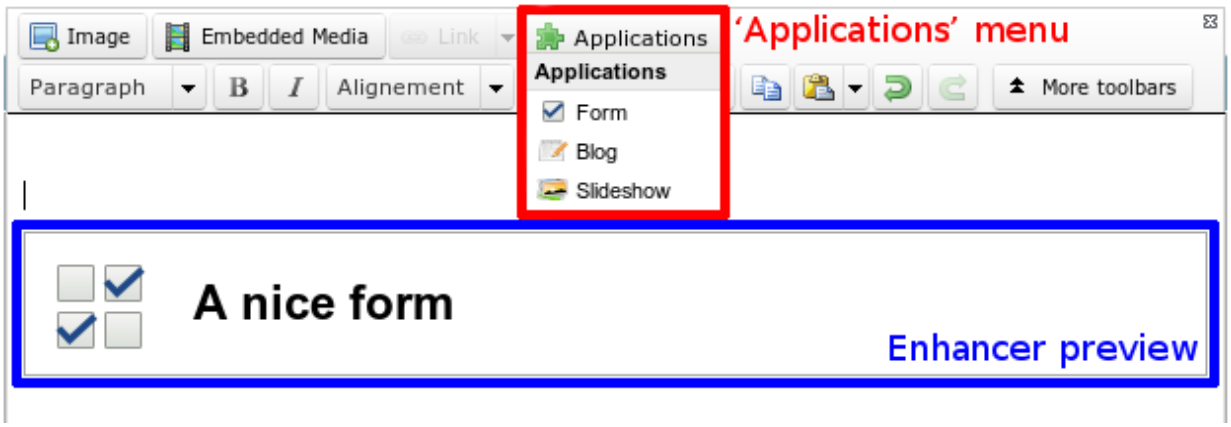
```php
<?php

return array(
    // Configuration de la popup
    'popup' => array(
        'layout' => array(
            'view' => 'mon_appli::enhancer/popup',
        ),
    ),
);
```

Ce fichier fait référence à la vue `mon_appli::enhancer/popup` qui n'existe pas, et qu'il faut donc créer. Cette dernière contiendra des champs de formulaire avec nos options configurables :

```php
<h3>Options</h3>
<p>
    <label for="item_per_page">
    <input type="text" name="item_per_page" id="item_per_page" value="<?= \Arr::get($enhancer_args, '
</p>
```

L'ancienne configuration de l'enhancer est disponible dans la variable `$enhancer_args` (utile pour pré-remplir le formulaire en cas de modification des options de configuration et de réouverture de la popup).

Modifier la prévisualisation



La prévisualisation ajoutée dans le WYSIWYG est chargée en faisant appel à la valeur `previewUrl` configurée dans le fichier `metadata.config.php`.

Généralement, vous ferez appel au même contrôleur que celui de la popup, mais en appelant la méthode `action_preview()` au lieu de `action_popup()`.

La vue fournie par défaut utilise une icône, un titre (les valeurs par défaut reprennent l'icône 64*64 de l'application, ainsi que le titre de l'enhancer) et un `layout` (fichiers de vues additionnels appelés).

`mon_appli::config/controller/admin/enhancer.config.php` :

```
<?php

return array(
    // Configuration de la popup
    'popup' => array(
        // Ce qu'on avait configuré plus tôt
    ),
    // Configuration de la prévisualisation
    'preview' => array(
        // (facultatif) vue à utiliser pour le rendu (valeur par défaut en exemple)
        //'view' => 'nos::admin/enhancer/preview',
        // (facultatif) fichiers de vues additionnels (inclus par la view au-dessus)
        //'layout' => array(),
        'params' => array(
            // (optionnel) reprend le titre de l'enhancer par défaut
            'title' => "Mon super enhancer",
            // 'icon' (optionnel) reprend celui de l'application en taille 64*64 par défaut
        ),
    ),
);
```

À noter qu'il est possible de spécifier une fonction de callback à la fois pour le titre ou pour l'icône. Elle reçoit alors un paramètre : la configuration de l'enhancer `$enhancer_args`.

Par exemple, pour l'enhancer « Formulaire », le titre du formulaire sélectionné s'affiche.

3. [Front-office] Afficher votre contenu sur le site

Une fois la page enregistrée et publiée, l'enhancer va s'exprimer sur le site.

Le contenu sera généré par le contrôleur configuré dans une des clés `enhancer` ou `urlEnhancer` du fichier `metadata.config.php` (selon si on voulait un enhancer simple ou un URL enhancer). N'oubliez pas de prendre en compte les changements dans le gestionnaires d'applications si vous faites des modification sur ce fichier.

Par exemple, l'application « Formulaires » a pour configuration `noviusos_form/front/main` ce qui fera appel à la méthode `action_main()` du `Controller_Front` de l'application `noviusos_form` (qui correspond en fait au contrôleur `Nos\Form\Controller_Front`).

Cette action prend en paramètre le tableau de configuration qui a été défini dans la popup de configuration `$enhancer_args`.

Créons le contrôleur `mon_appli::controller/front.ctrl.php`

```
<?php

namespace Mon\Appli;

class Controller_Front extends \Nos\Controller_Front_Application
{
```

```
    public function action_main($enhancer_args = array())
    {
        // Pour tester
        return print_r($enhancer_args, true);
    }
}
```

4. URL enhancers

Dans le cas d'un URL enhancer, ce dernier sera capable de gérer des URL.

Lorsqu'on parle du billet de blog `toto` ou de la catégorie `ski`, on fait en réalité référence au billet de blog dont le nom virtuel est `toto` et à la catégorie dont le nom virtuel est `ski`.

Prenons un exemple : si votre URL enhancer a été ajouté sur la page `mon/blog.html`, alors il sera en mesure de gérer des URL qui commencent par `mon/blog/**.html`, comme :
- `mon/blog.html` (liste de tous les billets) ;
- `mon/blog/toto.html` (billet de blog `toto`) ;
- `mon/blog/page/2.html` (2e page de la liste des billets) ;
- ou encore `mon/blog/category/ski.html` (liste des billets de la catégorie `ski`).

Comme précédemment, le contenu est généré par l'action `main`, mais il est possible de récupérer l'URL étendue avec `$this->main_controller->getEnhancerUrl();`.

Ensuite, le contrôleur peut générer du contenu différent en fonction de l'URL demandée. Voici un exemple (simplifié) tiré de l'application « Blog » :

```
<?php

namespace Nos\Blog;

class Controller_Front extends \Nos\Controller_Front_Application
{
    public function action_main($enhancer_args = array())
    {
        // URL complète de la page == 'mon/blog/category/ski.html'
        // => $enhancer_url == 'category/ski' (sans .html)
        $enhancer_url = $this->main_controller->getEnhancerUrl();
        $segments = explode('/', $enhancer_url);

        if (empty($enhancer_url))
        {
            // URL 'mon/blog.html' (URL de la page sur laquelle a été ajouté l'enhancer)
            // Affichage de la liste des billets (page 1)
        }
        else if (count($segments) == 1)
        {
            // URL 'mon/blog/toto.html'
            // Affichage du billet de blog 'toto'
        }
        else if (count($segments) == 2)
        {
            if ($segments[0] == 'page')
            {
                // URL 'mon/blog/page/2.html'
                $page = $segments[1];
                // Affichage de la page 2 de la liste des billets
            }
```

```
            else if ($segments[0] == 'category')
            {
                // URL 'mon/blog/category/ski.html'
                $category = $segments[1];
                // Affichage de la liste des billets de la catégorie 'ski'
            }
        }

        // L'URL demandé n'est pas gérée par cet enhancer (erreur 404 pour cet enhancer)
        throw new \Nos\NotFoundException();
    }
}
```

Lorsque l'enhancer gère des URL pour certains modèles (ORM), c'est lui qui connait la procédure de génération des ces dernières. Pour ce faire, il faut alors implémenter une méthode statique `get_url_model()` qui va s'en occuper :

```
<?php

namespace Nos\Blog;

class Controller_Front extends \Nos\Controller_Front_Application
{
    public static function get_url_model($item, $params = array())
    {
        $model = get_class($item);
        $page = isset($params['page']) ? $params['page'] : 1;

        switch ($model)
        {
            // URL pour un billet de blog particulier
            case 'Nos\Blog\Model_Post' :
                return urlencode($item->virtual_name()).'.html';
                break;

            // URL pour la liste des billets d'une catégorie (avec optionnellement une page)
            case 'Nos\Blog\Model_Category' :
                return 'category/'.urlencode($item->virtual_name()).($page > 1 ? '/'.$page : '').'.h
                break;
        }

        return false;
    }
}
```

Cette fonction est liée à la `Behaviour_Urlenhancer` et aux méthodes `url()` et `urls()` des modèles. Pour voir comment les configurer, il faut se référer à la *documentation d'API associée*.

Exemple :

```
<?php

// Sélection de la catégorie ayant pour ID 1
$category = Nos\Blog\Model_Category::find(1);

$url = $category->url(array('page' => 2));
```

`$url` aura pour valeur `mon/blog/category/ski/2.html`. Si on décompose :
- `mon/blog` : URL de la page sur laquelle est présent l'enhancer ;
- `ski` : URL virtuelle de la catégorie ayant pour ID 1 ;

– `2` : numéro de page demandée dans les paramètes.

1.4.3 Créer un gabarit

1. Définition dans le fichier `metadata`

Les metadata d'un template sont décrites dans la *documentation d'API*.

2. Création du fichier de vue

L'emplacement du fichier dépend de la clé `file` configurée dans le fichier `metadata.config.php`.

À l'intérieur du gabarit, vous avez accès à plusieurs variables intéressantes :

$wysiwyg Un tableau contenant en clé le nom du WYSIWYG configuré dans le fichier `metadata.config.php` et en valeur le contenu saisi dans le back-office.

$page L'instance du `Nos\model_Page` courant.

$main_controller L'instance du *contrôleur s'occupant du front-office*.

```
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <link rel="shortcut icon" href="static/favicon.ico" />
    <link rel="stylesheet" type="text/css" href="static/apps/noviusos_templates_basic/css/base.css" r
</head>

<body>

    <header>Cette zone d'entête sera affichée sur toutes les pages configurées avec ce gabarit.</head

    <div id="menu">
        Mon joli menu
    </div>

    <div id="content">
        <?= $wysiwyg['content']; ?>
    </div>

</body>
</html>
```

1.4.4 Rajouter des champs

Voir aussi :

/app_extend/add_field

La majorité des champs qui sont ajoutés ont besoin d'une colonne dans la table MySQL correspondant à votre modèle.

Les champs sont ensuite ajoutés au formulaire du CRUD en passant par la clé `fields` du fichier de configuration.

La syntaxe se base sur une fonctionnalité existante de FuelPHP, qui définit comment une colonne s'affiche.

Voir aussi :

Documentation de FuelPHP sur les propriétés d'un modèle

En plus des champs de formulaires standards, Novius OS possède des *renderers*, qui sont un peu plus poussés. Ils permettent par exemple de sélectionner un média, une page, une date...

Exemple de configuration :

```
<?php
return array(
        'name' => array(
                'label' => 'Texte affiché à côté du champ',
                'form' => array(
                        'type' => 'text',
                        'value' => 'Valeur par défaut',
                ),
                'validation' => array(),
);
```

Champs standards

Le texte en gras est la valeur de la propriété `type`.
- <input type="**text**">
- <input type="**password**">
- **<textarea>**
- **<select>**
- <input type="**radio**">
- <input type="**checkbox**">
- <input type="**submit**">
- <input type="**button**">
- <input type="**file**">

```
<?php
return array(
        'gender' => array(
                'label' => 'Genre',
                'form' => array(
                        'type' => 'select',
                        'options' => array(
                                'm' => 'Masculin',
                                'f' => 'Féminin',
                        )
                ),
                'validation' => array('required'),
        ),
);
```

<button type="submit">

- `type = submit` génère `<input type="submit">`
- `type = button` génère `<input type="button">`

La propriété `tag` peut être utilisé pour forcer un tab HTML précis, pour gérer le cas bouton de type `submit`.

FuelPHP utilisera automatiquement la `value` comme texte du bouton.

```
<?php
return array(
        'save' => array(
```

```
                'form' => array(
                        'type' => 'submit',
                        'tag' => 'button',
                        'value' => 'Save',
                ),
        ),
);
```

Renderers (champs améliorés)

La liste des `renderers` est disponible dans *la documentation d'API*.

1.4.5 Ajouter des vignettes dans l'App Desk

C'est en réalité très simple. Il faut définit deux clés spéciales dans votre `data_mapping` :
– `thumbnail` : chemin vers la vignette de l'item ;
– `thumbnailAlternate` : chemin vers une image de remplacement lorsqu'il n'y a pas de vignette.
Pour ce faire, rendez-vous dans le fichier `config/common/item.config.php` :

```
<?php

return array(
    'data_mapping' => array(
        'thumbnail' => array(
            'value' => function ($item) {
                foreach ($item->medias as $media) {
                    return $media->get_public_path_resized(64, 64);
                }
                return false;
            },
        ),
        'thumbnailAlternate' => array(
            'value' => function ($item) {
                return 'static/apps/mon_appli/icons/64.png';
            }
        ),
    ),
);
```

Il faut ensuite activer la vue vignette dans la configuration de l'App Desk `mon_appli::config/controller/admin/appdesk.config.php` :

```
<?php

return array(
    'model' => '',
    'query' => array(),
    'inspectors' => array(),
    'i18n' => array(),
    'thumbnails' => true,
);
```

Facultativement, il est possible de mettre la vue vignette par défaut (au lieu de la vue liste) :

```php
<?php

return array(
    'model' => '',
    'query' => array(),
    'inspectors' => array(),
    'i18n' => array(),
    'thumbnails' => true,
    'appdesk' => array(
        'appdesk' => array(
            'defaultView' => 'thumbnails',
        ),
    ),
);
```

1.4.6 Ajouter un fichier associé

Principes

Novius OS fournit la classe *Attachment* pour gérer les fichiers associés.

Les fichiers associés sont stockés dans le répertoire `local/data/files/`.

Généralement les fichiers sont associés à un *Model* mais il est possible de les gérer comme bon vous semble. Pour définir un *Attachment* il suffit de fournir une configuration.

```php
<?php

$attachment = \Nos\Attachment::forge('my_id', array(
        'dir' => 'apps'.DS.'myapps',
));
```

Dans l'exemple ci-dessus, notre fichier associé sera enregistré dans le répertoire `local/data/files/apps/myapps/my_id/`.

Pour enregistrer un fichier, il suffira de faire :

```
$attachment->set($_FILES['file']['tmp_name'], $_FILES['file']['name']);
$attachment->save();
```

Dans cet exemple, nous enregistrons un fichier uploadé comme fichier associé. Le chemin du fichier sera alors `local/data/files/apps/myapps/my_id/nom_original.ext` où `nom_original.ext` est le nom original du fichier uploadé, récupéré via `$_FILES['file']['name']`.

Attaché à un model

Dans le cas d'un fichier attaché à un *Model*, c'est encore plus simple. Dans la définition de votre classe, il suffit de déclarer :

```
class Model_Example extends \Nos\Orm\Model
{

        protected static $_attachment = array(
                'avatar' => array(),
                'cv' => array(),
        );
```

De cette façon, chaque item de `Model_Example` aura 2 fichiers associés : `avatar` et `cv`.

```
$item = Model_Example::find('first');
$item->avatar->set($_FILES['file']['tmp_name'], $_FILES['file']['name']);
$item->avatar->save();
```

Utilisation avancée

Pour plus de détails, consultez la documentation d'API d'*Attachment*.

Extensions

A la création de votre *Attachment*, vous pouvez spécifier une liste d'extensions de fichier autorisées en ajoutant la clé `extensions` au tableau de configuration et en lui donnant un tableau d'extensions acceptées en valeur.

Si votre fichier doit être une image, une clé `image` à `true` suffira.

Alias pour l'URL

Par défaut, votre fichier attaché sera disponible à l'URL du type :

`http://www.mondomaine.com/data/files/`*`dir/id/file_name.extension`*

Si `dir` est égal, comme souvent, à `apps/mon-apps/mon-type-de-fichier/`, celà peut faire une URL assez longue.

Définissez une clé `alias` dans le tableau de configuration de votre *Attachment*. La valeur de `alias`, remplacera celle de `dir` dans l'URL.

Fichier attaché sécurisé

Si votre fichier attaché ne doit pas être accessible à n'importe qui, vous pouvez le sécuriser. Il suffit de définir, toujours dans le tableau de configuration, une clé `check` de type fonction de callback. A chaque fois que le fichier sera demandé, via son URL, le système exécutera cette fonction, en lui passant l'objet *Attachment* en paramètre, pour vérifier si la personne connectée a le droit d'y accéder.

Exemple :

```
class Verification
{
        public static function check($attachment)
        {
                return isset($_SESSION['user_connected']) && $_SESSION['user_connected'];
        }
}

$attachment = \Nos\Attachment::forge('my_id', array(
        'dir' => 'apps'.DS.'myapps',
        'check' => array('Verification', 'check'),
));
```

De cette façon, si l'internaute est connecté, donc dans notre cas la variable de session `user_connected` est à `true`, il recevra le fichier. S'il ne l'est pas, il recevra une erreur 404.

1.4.7 Traduire l'application

Chaque application possède un répertoire `lang` dans lequel sont placés les fichiers de traductions, qu'on appellera **dictionnaires**.

Ce répertoire contient autant de sous-répertoires que de langues dans lesquelles on veut traduire notre application, et qui contiennent eux-même les dictionnaires.

Ces répertoires de langues portent des noms de locales, comme par exemple `fr` (Français), ou `en` (Anglais).

Les dictionnaires sont des fichiers PHP qui retournent un tableau PHP, un peu comme les fichiers de configuration.

Voir aussi :

API de la classe I18n

Fichier `metadata.config.php`

Les `metadata` étant un peu spéciales, elles nécessitent un fichier de traduction à part. La première étape est d'indiquer dans quel dictionnaire se trouvent les traductions du fichier `metadata.config.php`.

```
<?php

return array(
    'name' => 'My app',
    'namespace' => 'My\App',
    'i18n_file' => 'my_app::metadata',
    // ... autres clés
);
```

Comme toute modification sur les `metadata`, ne pas oublier d'appliquer les changements dans le gestionnaire d'applications.

Ensuite, il faut créer le dictionnaire `my_app::lang/fr/metadata.lang.php` :

```
<?php

return array(
    'My app' => 'Mon appli',
);
```

Novius OS sait automatiquement quelles clés peuvent / ont besoin d'être traduites dans le fichier `metadata` et ira chercher les traductions correspondantes.

Autres fichiers

Partout ailleurs, le mécanisme à utiliser pour rendre les fichiers sources traduisibles est le suivant :

1. Utiliser la fonction `__()` pour récupérer la traduction ;
2. Pour chaque fichier, configurer dans quel dictionnaire sont situées les traductions.

C'est assez simple pour les vues et la configuration :

```
<?php

// Configure la fonction __() pour le reste du fichier
Nos\I18n::current_dictionary('my_app::common');

__('Translate this'); // La traduction sera récupérée depuis my_app::lang/<lang>/common.lang.php
```

C'est plus pointu pour les contrôleurs, car la langue dépend de l'utilisateur et n'est connue qu'après la phase d'authentification, qui a lieu dans le `before()`.

C'est pourquoi un point d'entrée `prepare_i18n()` a été créé :

```
<?php

namespace Nos\Form;

class Controller_Admin_Form extends \Nos\Controller_Admin_Crud
{
    public function prepare_i18n()
    {
        // Configure la langue des fichiers de traductions en fonction de l'utilisateur connecté
        parent::prepare_i18n();
        // Configure la fonction __() pour le reste du contrôleur
        \Nos\I18n::current_dictionary('noviusos_form::common');
    }

    // Autres méthodes qui font usage de __()
}
```

Il est possible de spécifier plusieurs dictionnaires pour un fichier en utilisant un tableau. Les traductions seront alors récupérés dans le premier fichier qui contient la traduction.

```
<?php

Nos\I18n::current_dictionary(array('my_app::dictionary', 'my_app::common'));

// La traduction sera récupérée depuis my_app::lang/<lang>/dictionary.lang.php si elle existe
// Ou dans my_app::lang/<lang>/common.lang.php sinon
__('Translate this');
```

1.4.8 Fichiers de migrations

Chaque application ainsi que le dossier local peut avoir un dossier *migrations*. Ce dossier contient des fichiers de migration qui sont un moyen pratique de mettre à jour la base de données ou les fichiers locaux. Le système de migration FuelPHP est utilisé. Néanmoins il y a deux choses que vous devriez savoir pour Novius OS :

- Les fichiers de migration des applications doivent être sous le namespace *{{NAMESPACE_DE_LAPPLICATION}}\Migrations*
- Quand c'est possible, les mises à jour des fichiers migrations doivent se trouver dans un fichier sql séparé (pour faciliter les mises à jour manuelles). Comme la plupart du temps il n'y a que des requêtes sql à exécuter, la classe *Nos\Migration* a été implémentée dans l'optique de faciliter ces migrations. Vous pouvez y jeter un coup d'œil dans *la documentation d'API*.

Lorsqu'une application est installée ou mise à jour, les fichiers de migration qui s'y trouvent sont exécutés (s'ils ne l'ont pas déjà été) via l'interface de l'application manager.

1.5 Notes de versions

1.5.1 Notes de la version 0.2

Nouveautés

- Une instance de Novius Os peut-être multi-contexte : gérer de un à plusieurs sites, chacun dans une ou plusieurs langues
- Intégration de l'application **Slideshow**
- Intégration de l'application **Form**
- Intégration de l'application **Assistant 'Créer mon appli'**
- Intégration du `sharer` **Simple Google+** sur le même modèle que Facebook et Twitter
- Le back-office est disponible en français et en anglais
- Les onglets du back-office sont fermables avec un click sur le bouton du milieu de la souris

  Nouveau bouton *Close all other tabs*

Développeur

- Conséquences du passage de multi-languages à multi-contextes
  - Toutes les colonnes `lang`, `lang_common_id`, `lang_is_main` de la base de données ont été renommées avec `context`
  - Les nouvelles colonnes `context` ont été agrandies, de 5 à 25 caractères
  - Le `behaviour Translatable` a été renommé en `Twinnable`
  - La configuration se fait dans un fichier dédié (plus dans `config.php`). Deux nouvelles clé `contexts` et `sites` en plus de `locales`
  - Dans le `CRUD`, la notion de `context` est remplacée celle d'`environment` pour éviter les confusions (`context_relation` -> `environment_relation`, `item_context` -> `item_environment`)
  - Toutes les variables liées ont, elles aussi, été renommées
- Mise à jour des librairies tierces :
  - **jQuery**, de la 1.7.2 à **1.8.2**
  - **jQuery UI**, de la 1.8.22 à la **1.8.24**
  - **Wijmo**, de la 2.1.4 à la **2.2.2**
  - **tinyMCE**, de la 3.5.6 à la **3.5.7**
  - **FuelPHP** et ses packages (dont `email`), de la 1.2 à la **1.4**
- Modification de l'API des pages :
  - Nouvelle classe outil `Tools_Url`
  - `Model_Page->get_link()` -> `Model_Page->link()`
  - `Model_Page->get_href()` -> `Model_Page->url()`
  - `Model_Page::get_url()` -> `Tools_Url::page()`
  - Suppression de `Model_Page::get_url_absolute()`
  - Toutes les méthodes retournent des URLs absolues
- Fusion et amélioration de la configuration des `app-desk`, des `inspector` et des `CRUD` :
  - Fichier de configuration commun pour les données propres aux models
  - Possibilité de formatter une colonne d'une `grid` via la configuration PHP (et plus seulement en Javascript)
- Dans le `Controller_CRUD`, la méthode `from_item` devient `init_item` et n'est appellée que si l'item est nouveau
- Nouvelle class `Attachment` pour gérer des fichiers attachés à un item que l'on ne pas mettre dans la médiathèque
- Disparition de la notion de `widget` au profit de `renderer`. Renommage de classes et de chemin de vues
- Toutes les vues et fichiers de configuration sont surchargeables dans le répertoire `config` du site
- Création d'un `controller` de popup d'`enhancer` pour le WYSIWYG avec prévisualisation par défaut
- La clé de configuration `upload.disabled_extensions` est déplacée dans `novius-os.upload.disabled_extensions`

- Les variables $page et $main_controller sont disponible dans le template
- Le `renderer Time Picker` peut être utiliser en dehors d'un `Fieldset`
- L'événement PHP `front.start` prend le paramètre `cache_path` en plus

1.5.2 Guide de migration de la version 0.1 à la version 0.2

Changements non rétro-compatibles obligatoires

Conséquences du passage de multi-languages à multi-contextes

- La configuration des contextes se fait dans un fichier dédié (elle n'est plus dans `config.php`). Deux nouvelles clé `contexts` et `sites` font leur apparition, en plus de `locales`
- Toutes les colonnes `lang`, `lang_common_id`, `lang_is_main` de la base de données ont été renommées avec `context`
- Les nouvelles colonnes `context` ont été agrandies, de 5 à 25 caractères
- Le `behaviour Translatable` a été renommé en `Twinnable`
- Dans le `CRUD`, la notion de `context` est remplacée celle d'`environment` pour éviter les confusions (`context_relation` -> `environment_relation`, `item_context` -> `item_environment`)
- Toutes les variables liées ont, elles aussi, été renommées

Modification de l'API des pages

- `Model_Page->get_link()` -> `Model_Page->link()`
- `Model_Page->get_href()` -> `Model_Page->url()`
- `Model_Page::get_url()` -> `Tools_Url::page()`
- Suppression de `Model_Page::get_url_absolute()`
- Toutes les méthodes retournent des URL absolues

Modification de l'API des évènements

- `front.start` prend désormais en paramètre un tableau contenant la clé `url`

Procédure à suivre

- Créer le fichier `contexts.config.php` (se baser sur le sample).
- Modifier `config.php` pour rajouter la clé `novius-os` (se baser sur le sample).
- Rechercher `_lang` et remplacer par `_context` (c-à-d `page_lang => page_context`)
- Rechercher `Nos\Model_` et remplacer par `Nos\{{app}}\Model_` (c-à-d `Nos\Model_Page => Nos\Page\Model_Page`, `Nos\Model_Media => Nos\Media\Model_Media`)
- Rechercher `Translatable` et remplacer par `Contextable`
- Rechercher `Model_Page->get_link()` et remplacer par `Model_Page->link()`
- Rechercher `Model_Page->get_href()` et remplacer par `Model_Page->url()`
- Rechercher `Model_Page::get_url()` et remplacer par `Tools_Url::page()`
- Rechercher `Nos\Widget_Page_Selector` et remplacer par `Nos\Page\Renderer_Selector`
- Rechercher `Nos\Widget_Media` et remplacer par `Nos\Renderer_Media`
- Rechercher `Nos\Widget_` et remplacer par `Nos\{{Renderer_Version}}`

Changement dans les applications

Metadata

- Launchers : la clé `url` a été remplacée par `action` (syntaxe standard *nosAction*)
- Launchers : la clé `icon64` a été remplacée par `icon`

```php
<?php

return array(
    'launcher_name' => array(
        'url' => '{{your_url_here}}',
    ),
);

// Replace with

return array(
    'launcher_name' => array(
        'action' => array(
            'action' => 'nosTabs',
            'tab' => array(
                'url' => '{{your_url_here}}',
            ),
        ),
    ),
);
```

- L'application fournit désormais une liste d'icônes :

```php
<?php

return array(
    'icons' => array(
        64 => 'static/apps/noviusos_page/img/64/page.png',
        32 => 'static/apps/noviusos_page/img/32/page.png',
        16 => 'static/apps/noviusos_page/img/16/page.png',
    ),
);
```

- Un launcher sans `icon` utilisera l'icône 64 de son applicaton
- Un onglet (`tab`) sans `iconUrl` utilisera l'icône 32 de son applicaton
- Un enhancer sans `iconUrl` utilisera l'icône 16 de son applicaton

Configuration des CRUD

- `widget` a été renommé en `renderer`

```php
<?php

return array(
    'field_name' => array(
        'widget' => 'Nos\Widget_Media',
        'widget_options' => array(),
    ),
);
```

```
// À remplacer par :
return array(
    'field_name' => array(
        'renderer' => 'Nos\Renderer_Media',
        'renderer_options' => array(),
    ),
);
```

Migration "complète" 0.2

Cette partie se base sur l'existence d'une hypothétique application `lib_agenda`.

Appdesk

Les modèles possèdent un nouveau fichier de configuration `common` qui contient : * un `data_mapping` * une liste d'`actions`

Dans `appdesk.config.php` :
- Supprimer les clés `selectedView` et `views` (si vous n'avez qu'une seule vue sans fichier de conf JS).
- Repérez le modèle principal de votre appdesk (clé `query.model`).
- **Créez le ficher common associé `config/common/{model_name}.config.php`**
  - `{{model_name}}` correspond au nom du model en minuscule, sans le préfixe `Model_`, par exemple `Model_Page` devient `page`
  - `Model_Page` correspond donc au fichier `config/common/page.config.php`

Note : Attention à bien avoir `'hideContexts' => true,` dans la configuration de votre appdesk si vos items ne sont pas `Contextable`.

Data mapping Le `data_mapping` correspond à la fusion du `dataset` et de `appdesk.grid.columns`.

```
<?php

// Ancien code de appdesk.config.php
return array(
    'query' => array(
        'model' => 'Lib\Agenda\Model_Event',
        'order_by' => array('evt_date_begin' => 'DESC'),
        'limit' => 20,
    ),
    // ...
    'dataset' => array(
        'id'            => 'evt_id',
        'title'         => 'evt_title',
        'periode'       => array(
            'search_column' => 'evt_date_begin',
            'dataType'      => 'datetime',
            'value'         => function ($object) {
                // ...
            },
        },
    ),
    // ...
    'appdesk' => array(
```

```
        // ...
        'grid' => array(
            'urlJson' => 'admin/lib_agenda/appdesk/json',
            'columns' => array(
                'id' => array(
                    'headerText' => __('Id'),
                    'dataKey' => 'id'
                ),
                'title' => array(
                    'headerText' => __('Nom'),
                    'dataKey' => 'title'
                ),
                'periode' => array(
                    'headerText' => __('Dates'),
                    'dataKey' => 'periode'
                ),
                'published' => array(
                    'headerText' => __('Status'),
                    'dataKey' => 'publication_status'
                ),
                'actions' => array(
                    'actions' => array('update', 'delete'),
                ),
            ),
        ),
        // ...
    ),
);

<?php

// Nouveau code de appdesk.config.php
return array(
    'query' => array(
        'order_by' => array('evt_date_begin' => 'DESC'),
        'limit' => 20,
    ),
    // Indique quel est le model, et donc quel fichier 'common' charger
    'model' => 'Lib\Agenda\Model_Event',
    // ...
    // DEPLACER / FUSIONNER la clé 'dataset' dans common
    // ...
    'appdesk' => array(
        // ...
        // DEPLACER / FUSIONNER la clé 'grid' dans common
        // ...
    ),
);

<?php

// Code du nouveau fichier event.config.php
return array(
    // Fusion de 'appdesk.dataset' et de 'appdesk.grid.columns'
    'data_mapping' => array(
        'id' => array(
            'title' => __('Id'),
            'column' => 'evt_id'
```

```
        ),
        'title' => array(
            'title' => __('Nom'),
            'column' => 'evt_title'
        ),
        'periode' => array(
            'title' => __('Dates'),
            'search_column' => 'evt_date_begin',
            'value'         => function ($object) {
                // ...
            }
        ),
        'published' => array(
            'title' => __('Status'),
            'method' => 'publication_status'
        ),
    ),
);
```

Quelques remarques : * `headerText` peut s'écrire `title` (plus facile / simple à retenir, utilisé dans les applis natives) * `datakey` peut s'écrire `column` * `value` est toujours possible pour une fonction de callback * `method` est une nouvelle option qui exécute une méthode au lieu de récupérer une `column`

Actions Les actions sur le modèle principal (celui de la grid de l'appdesk) doivent également être déplacées dans le fichier `common`.

```
<?php

// Ancien code de appdesk.config.php
return array(
    // ...
    'appdesk' => array(
        // ...
        // DEPLACER la clé 'actions' dans 'config/common/{{model_name}}.config.php'
        'actions' => array(
            'edit' => array(
                // ...
            ),
            'delete' => array(
                // ...
            ),
        ),
        // ...
    ),
    // ...
);

<?php

// Nouveau code de appdesk.config.php
return array(
    // ...
    'appdesk' => array(
        // ...
        // La clé 'actions' n'est plus ici
        // ...
    ),
```

```
    // ...
);

<?php

// Nouveau code de 'config/common/event.config.php'
return array(
    'data_mapping' => array(
        // ...
    ),

    // Tableau de configuration déplacé depuis 'appdesk.actions'
    'actions' => array(
        'Lib\Agenda\Model_Event.edit' => array(
            // ...
        ),
        'Lib\Agenda\Model_Event.delete' => array(
            // ...
        ),
    ),
);
```

À partir du moment où le fichier `common` est utilisé, les actions génériques suivantes apparaissent : * `add` * `edit` (et non pas `update` !) * `visualise` (si approprié, c-à-d si le modèle possède le Behaviour Urlrenhancer) * `delete` * `share` (si approprié)

Dans le cas de l'agenda et de `Model_Event` ce dernier possédait une action `update` qui apparait désormais en double... (parce qu'on avait mis le mauvais nom `update` au lieu de `edit`).

Du coup, **dans le cas de l'agenda**, il faut : * Renommer `update` en `edit` * Etant donné que les actions `edit` et `delete` font le traiement par défaut, **supprimer** les clés... * Il est possible de garder uniquement les clés à redéfinir (pour les textes français dans ce cas...)

Notes : * Dans la version de NOS utilisée, il faut préfixer les actions par le nom du modèle, ce n'est plus nécessaire dans la version finale * `{{controller_base_url}}` est utilisable dans les URL d'actions. Dans le cas d'agenda, il sera remplacé par `lib_agenda/admin/agenda/` * Une nouvelle clé `targets` permet de définir où les actions doivent apparaître (cf. commentaires).

```
<?php

// Exemple de placeholder {{controller_base_url}} + 'targets'
array(
    'Lib\Agenda\Model_Event.edit' => array(
        'action' =>
            'action' => 'nosTabs',
            'tab' => array(
                'url' => "{{controller_base_url}}insert_update/{{id}}",
                'label' => __('Modifier'),
            ),
        ),
        'label' => __('Modifier'),
        'primary' => true,
        'icon' => 'pencil',
        // Nouvelle clé pour définir où cette action apparait
        'targets' => array(
            'grid' => true, // Dans la grid (dans la dernière colonne 'actions')
            'toolbar-grid' => true, // Sur l'appdesk, dans la toolbar (anciennement configuré via 'ap
            'toolbar-edit' => true, // Sur le formulaire d'édition (en haut à droite)
        ),
```

```
    )
);
```

Par défaut, les targets sont configurées comme suit pour les actions : * `grid` : edit + visualise + delete * `toolbar-grid` : add * `toolbar-edit` : visualise + share + delete

---

Note : Pour l'instant, `appdesk.appdesk.buttons` est toujours défini, il prend donc la main sur la configuration par défaut. Sachant que nous avons à la fois 'Ajouter un évènement' et 'Ajouter une catégorie', on ne peut pas (encore) le supprimer tout de suite.

I18N et traductions Les textes sont configurables via la clé `i18n`.

Se référer à la documentation, ou (en attendant) au fichier `framework/config/common_i18n.config.php` pour la liste des clés possibles.

```
<?php

return array(
    'i18n' => array(
        // Crud
        'notification item added' => __('And voilà! The page has been added.'),
        'notification item deleted' => __('The page has been deleted.'),

        // General errors
        'notification item does not exist anymore' => __('This page doesn't exist any more. It has be
        'notification item not found' => __('We cannot find this page.'),

        // Blank slate
        'translate error parent not available in context' => __('We're afraid this page cannot be ado
        'translate error parent not available in language' => __('We're afraid this page cannot be ad

        // Deletion popup
        'deleting item title' => __('Deleting the page `{{title}}"),

        # Delete action's labels
        'deleting button 1 item' => __('Yes, delete this page'),
        'deleting button N items' => __('Yes, delete these {{count}} pages'),

        '1 item' => __('1 page'),
        'N items' => __('{{count}} pages'),

        # Keep only if the model has the behaviour Contextable
        'deleting with N contexts' => __('This page exists in <strong>{{context_count}} contexts</str
        'deleting with N languages' => __('This page exists in <strong>{{language_count}} languages<

        # Keep only if the model has the behaviours Contextable + Tree
        'deleting with N contexts and N children' => __('This page exists in <strong>{{context_count
        'deleting with N contexts and 1 child' => __('This page exists in <strong>{{context_count}} c
        'deleting with N languages and N children' => __('This page exists in <strong>{{language_cou
        'deleting with N languages and 1 child' => __('This page exists in <strong>{{language_count}

        # Keep only if the model has the behaviour Tree
        'deleting with 1 child' => __('This page has <strong>1 sub-page</strong>.'),
        'deleting with N children' => __('This page has <strong>{{children_count}} sub-pages</strong>
    ),
);
```

Inspecteurs En 0.1, les inspecteurs sont configurés à 3 endroits : * La clé `appdesk.appdesk.inspectors` * La clé `inputs` * Le fichier de configuration `inspector/{model}.config.php`

En 0.2, les `inputs` doivent désormais être déplacés dans leur fichier `inspector/{model}.config.php` correspondant. Chaque clé de `appdesk.appdesk.inspectors` sera déplacée sur une clé `appdesk` du fichier `inspector/{{model}}.config.php` correspondant. La clé `appdesk.appdesk.inspectors` est remplacée par une clé `inspectors` qui contient le nom des fichiers `inspector/{{model}}.config.php`.

Category

```
<?php

// Ancien code dans appdesk.config.php
return array(
    // ...
    'inputs' => array(
        // Cet input correspond au filtre pour l'inspecteur catégorie
        // On déplace la clé (evt_cat_id) dans 'input.key' et la fonction de callback dans 'input.qu
        'evt_cat_id' => function($value, $query) {
            // ...
        },
    ),
    // ...
);
```

```
<?php

// Nouveau code dans config/controller/admin/inspector/category.config.php
return array(
    // ...
    'input' => array(
        'key' => 'evt_cat_id',
        'query' => function($value, $query) {
            // ...
        },
    ),
    // ...
);
```

Date

```
<?php

// Ancien code dans appdesk.config.php
return array(
    // ...
    'appdesk' => array(
        'appdesk' => array(
            // ...
            'inspectors' => array(
                // Il faut déplacer ce tableu dans le fichier de configuration de l'inspecteur, sous
                'startdate' => array(
                    'label' => __('Date de début'),
                    'url' => 'admin/lib_agenda/inspector/date/list',
                    'inputName' => 'startdate',
                    'vertical' => true
                ),
                // ...
```

```
            ),
        ),
        // ...
    ),
);
```

Ici l'inspecteur date n'a pas encore de fichier de configuration, on va en créer un et modifier le fichier de configuration de l'appdesk.

```
<?php

// Nouveau code dans appdesk.config.php
return array(
    // ...
    'inspectors' => array(
         'date',
         // ...
    ),
    // ...
);
```

```
<?php

// Nouveau fichier config/controller/admin/inspector/date.config.php
return array(
    'input' => array(
        'key' => 'evt_date_begin',
        // Pas besoin de 'query', l'inspecteur date en génère un automatiquement en fonction de la ke
    ),

    // Reprise de 'appdesk.appdesk.inspectors.startdate'
    'appdesk' => array(
        'label' => __('Date de début'),
    ),
);
```

L'idée est d'encapsuler le tableau `appdesk.appdesk.inspectors.{{inspector_name}}` dans une clé `appdesk` du fichier de config de l'inspecteur.

published

```
<?php

// Ancien code dans appdesk.config.php
return array(
    // ...
    'inputs' => array(
        // ...
        'evt_published' => function($value, $query) {
            // ...
        },
    ),
    // ...
    'appdesk' => array(
        'appdesk' => array(
            // ...
            'inspectors' => array(
                'published' => array(
```

```
                    'vertical' => true,
                    'url' => 'admin/lib_agenda/inspector/published/list',
                    'inputName' => 'evt_published',
                    'grid' => array(
                        'columns' => array(
                            'title' => array(
                                'visible' => false,
                                'dataKey' => 'title',
                            ),
                            'icon_title' => array(
                                'headerText' => __('Status'),
                                'dataKey' => 'icon_title',
                            ),
                            'id' => array(
                                'visible' => false,
                                'dataKey' => 'id',
                            ),
                        ),
                    ),
                ),
                // ...
            ),
            // ...
        ),
        // ...
    ),
);
```

L'inspecteur `published` a déjà un fichier de configuration, complétons le en créant une nouvelle clé `appdesk` :

```
<?php

// Nouveau fichier config/controller/admin/inspector/date.config.php
return array(
    'data' => array(
        // ...
    ),

    // Ici on reprend 'appdesk.appdesk.inspectors.published'
    'input' => array(
        'key' => 'evt_published',
        'query' => function($value, $query) {
            // ...
        },
    ),

    // Ici on reprend 'input.evt_published'
    'appdesk' => array(
        'vertical' => true,
        'inputName' => 'evt_published',
        'url' => 'admin/lib_agenda/inspector/published/list',
        'grid' => array(
            'columns' => array(
                'title' => array(
                    'visible' => false,
                    'dataKey' => 'title',
                ),
                'icon_title' => array(
```

```
                    'headerText' => __('Status'),
                    'dataKey' => 'icon_title',
                ),
                'id' => array(
                    'visible' => false,
                    'dataKey' => 'id',
                ),
            ),
        ),
    ),
);
```

```
<?php

// Nouveau code dans appdesk.config.php
return array(
    // ...
    'inspectors' => array(
         'published',
         // ...
    ),
    // ...
);
```

1.5.3 Notes de la version Chiba 1

Nouveautés

– Le comportement publiable (`Behaviour_Publishable`) supporte maintenant la publication avec dates de début et de fin.

Application Formulaire

– Ajout d'une barre de progression sur les formulaire multi-page

Blog / News

– Affichage des posts par auteur

Développeur

– Amélioration du front-office
  – Activation du `Profiling` par défaut dans l'environement `DEVELOPMENT`.
  – La configuration `novius-os.cache` est à `true` par défaut.
  – Les configurations `novius-os.cache_duration_page` et `novius-os.cache_duration_function` sont à 600 secondes par défaut, sauf en `PRODUCTION` à 3600 secondes.
  – Nouveaux événements `front.pageFound` et `front.response`.
  – Nouvelles méthodes sur le `Controller_Front` : getContext, disableCaching, setCacheDuration, setStatus, setHeader, getCustomData, setCustomData, sendContent, addCacheSuffixHandler.
  – Le status et les headers sont sauvés dans le cache.

  - Mécanisme de `Suffix Handler` permettant d'ajouter des suffixes au cache en fonction de paramètres GET ou de ce que vous voulez (par des `callables`).
  - Mécanisme pour exécuter du code tout en utilisant le cache.
- Properties des Models
  - Les properties des Models sont désormais définies dans les applications natives
  - Implémentation d'un mécanisme de cache des properties, via le cache FuelPHP. Ce cache s'auto regénère si une property, existante en base mais inconnue du Model, est appellée via le `get()` ou le `set()`.
- Les migrations sont maintenant dispatchées par application
- Nouvelle clé `requires` dans le `metadata` des applications permettant de définir qu'une application en nécessite une autre.
- Possibilité d'utiliser `href="##..."` dans les enhancers ou les templates ; les occurences seront remplacées par `href="#..."` sans être préfixées par le `base_url`.
- CRUD : Quand la clé `disabled` retourne une chaine, elle est affichée en tooltip. Les clés `disabled` et `visible` peuvent maintenant être contenir un scalaire simple, une fonction de callback ou un tableau de callbacks.
- Les miniatures d'images sont désormais sécurisées : il n'est plus possible d'en générer un grand nombre pour surcharger le serveur
- Permissions
  - Possiblité de définir des permissions par application via un fichier de configuration
  - Nouvelle API pour vérifier les permissions sur un utilisateur ou un rôle
  - Nouvelle configuration `novius-os.users.enable_roles` pour activer les rôles multiples sur les utilisateurs

Deprecated

- Ne plus utiliser la classe `Nos\Renderer_Media` mais `Nos\Media\Renderer_Media`.
- Dans la définition des `launchers`, ne plus utiliser la clé ``url` mais la clé `action`.
- Dans la configuration des renderer, ne plus utiliser la clé `widget` mais la clé `renderer` et modifier le nom de la classe appellée.
- Dans la configuration des renderer, ne plus utiliser la clé `widget_options` mais la clé `renderer_options`.
- Ne plus utiliser la classe `\Config::extendable_load()` mais `\Config::loadConfiguration()`.
- Dans la configuration du behaviour `Orm_Behaviour_Publishable`, ne plus utiliser la clé `publication_bool_property` mais la clé `publication_state_property`.

1.5.4 Guide de migration de la version 0.2 à la version Chiba 1

Peu de modification a réaliser pour passer à la version suivante. La nouvelle API est compatible avec l'ancienne. À partir de cette version, Novius OS gère les dépréciations. En voici une liste :
- Le renderer `Nos\Renderer_Media` est à renommer en `Nos\Media\Renderer_Media`.
- Dans la configuration des launchers, la clé `url` est dépréciée au profit de `action`.
- Les clés `widgets` et `widget_options` sont à remplacer par `renderer` et `renderer_options`.
- La fonction `\Config::extendable_load()` est dépréciée et doit être remplacée par `\Config::loadConfiguration`.
- Dans la configuration du behaviour `Orm_Behaviour_Publishable`, la clé `publication_bool_property` est dépréciée. Utilisez à la place la clé `publication_state_property`.

Les pages sont maintenant mises en cache par défaut (d'une durée de 10 minutes) ; celà peut entrainer des comportements inatendus dans certaines applications avec affichage en front (hors celles du coeur et celles installées par défaut).

Quelques fichiers migrations doivent être exécutées. Si vous utilisez `oil`, merci d'utiliser la commande `php oil refine migrate -m` car l'organisation des migrations a été modifiée.

1.6 Contribuer à Novius OS

1.6.1 Normes de codage

Ces normes pour le formatage du code doivent être suivies par toute personne contribuant à Novius OS.

Vous pouvez utiliser le ruleset.xml for PHP_CodeSniffer fait pour Novius OS.

Case

Tous les mots clés sont en minuscule (`class`, `interface`, `extends`, `implements`, `abstract`, `final`, `var`, `const`, `function`, `public`, `private`, `protected`, `static`, `if`, `else`, `elseif`, `foreach`, `for`, `do`, `switch`, `while`, `try`, `catch`, `true`, `false` et `null`).

Toutes les constantes sont majuscules (constante globale ou de classe).

Signature des structures de contrôle

```
try {
    ...
} catch (...) {
    ...
}

do {
    ...
} while (...);

while (...) {
    ...
}

// Un seul espace entre chaque condition des boucles 'for'.
for ($i = 0; $i < 10; $i++) {
    ...
}

// Un seul espace entre chaque condition des boucles 'foreach'.
foreach ($array as $key => $item) {
    ...
}

if (...) {
    ...
} else if (...) {
    ...
} else {
    ...
}

// Un seul espace après un cast.
$variable = (array) $variable;
```

Déclaration de fonction

```
/*
L'accolade d'ouverture d'une fonction est sur la ligne suivant la déclaration de la fonction.
Pas d'espace avant les virgules.
Un seul espace après les virgules.
Un seul espace entre le type et l'argument.
Un seul espace avant le signe '=' de la valeur par défaut.
Un seul espace après le signe '=' de la valeur par défaut.
Les paramètres avec une valeur par défaut doivent être à la fin de la signature de la fonction.
*/
function myfunction(array $first, $second, $third = array())
{
        ...
}
```

Appel de fonction

```
/*
Pas d'espace avant les virgules.
Un seul espace après les virgules.
*/
$value = myfunction($first, $second, $third);
```

Classe

```
/*
L'accolade d'ouverture d'une classe est sur la ligne suivant la déclaration de la classe.
Il doit y avoir une ligne vide après la déclaration du namespace.
Toutes les propriétés de la classe doivent avoir une portée (variables, functions and methods).
A single space after scope keywords (private, public, protected, static).
*/
namespace Name\Space;

class MyClass
{
        private $variable1 = null;

        protected static $variable1 = true;

        const MY_CONSTANT = false;

        public static function myfunction()
        {
                ...
        }
}
```

Fichier

Les caractères de fin de ligne doivent être un n. Les fichiers ne doivent pas se finir par un tag de fermeture PHP.

Seulement une instruction par ligne.

Le code PHP doit utilisé les tags longs < ?php ?> ou les tags courts d'affichage < ?= ?> ; aucune autre forme de tag ne doit être utilisée.

Les accolades de fermeture de même portée doivent être alignées. Les structures de contrôles sont correctement indentées (4 espaces, pas de tabulation).

Pas d'espace en début et fin de fichier.

1.6.2 Charte rédactionnelle (Français)

Introduction

Cette charte est destinée à toute personne écrivant des textes d'interface pour Novius OS : développeurs et designers d'applications, contributeurs au cœur, traducteurs. Elle établit des règles communes à l'ensemble des applications et langues afin d'**offrir aux utilisateurs des textes cohérents et agréables**.

Cette charte est basée sur les design personas d'Aaron Walter. Le guide Voice and Tone de MailChimp, créé par l'équipe de Walter, a également été une source d'inspiration.

À propos de la traduction

La traduction n'est pas du mot-à-mot. Il s'agit de rester fidèle au design original tout en l'adaptant à un nouveau public, à une autre culture. Une traduction littérale des textes de Novius OS n'aurait pas de sens. La traduction doit donner la sensation que le texte traduit est le texte original. Le premier document à traduire sont ces règles afin qu'une **charte localisée** soit disponible.

Personnalité, ton à adopter

Novius OS est **conçu pour des professionnels**. Ils ne l'utilisent pas pour le plaisir, ils ont un travail à faire. Novius OS doit montrer à ses utilisateurs qu'il partage le même objectif : réaliser le travail demandé le plus efficacement possible. Novius OS adopte ainsi un ton professionnel et qui va à l'essentiel. Novius OS vouvoie l'utilisateur.

Ceci étant dit, **utiliser Novius OS n'a pas à être ennuyeux** pour autant. Le discours habituel et insipide des logiciels (ex Veuillez entrer une valeur correcte) est à éviter. Nous ne voulons pas que Novius OS soit perçu comme un logiciel de plus.

Novius OS peut faire sourire ses utilisateurs (tout particulièrement s'ils viennent de réaliser une tâche), mais pas rire. Le logiciel n'est pas leur copain. Il s'agit plutôt d'**un collègue de confiance et bien informé** avec qui ils ont plaisir à travailler. Ou plutôt une équipe de collègues, car Novius OS dit « Nous » et pas « Je ».

Enfin, même si une partie du public de Novius OS sont des développeurs, les textes ne doivent pas comprendre de jargon. De même, un ton trop familier (ex : Désolé mec, ça a planté !) n'est pas bienvenu.

Caractéristiques de la personnalité

- **Professionnel** mais pas ennuyeux.
- **Efficace, direct** mais pas autoritaire.
- **Cohérent** mais pas répétitif.
- **Sympathique** mais pas familier.
- **Bien informé** mais pas donneur de leçon.

Exemples de textes

Actions :

Se connecter Allez, au travail

Se déconnecter Se déconnecter (à bientôt !)

Ajouter un nouvel item Ajouter une page [1]

Sauvegarder un item Enregistrer [1]

Confirmation :

Nouvel item ajouté Parfait ! Le billet de blog a été ajouté. [2]

Item mis à jour C'est bon, les modifications ont été enregistrées. [2]

Erreurs :

Erreur de l'utilisateur Vous devez ajouter un titre pour que le produit puisse être sauvegardé. Désolé. [3]

Erreur du système Il y a eu un problème. Merci de ré-essayer et de contacter votre développeur ou Novius OS si le problème persiste. Veuillez accepter nos excuses pour la gêne occasionnée. [4]

Prévention des erreurs (message) Des réponses à ce formulaire ont déjà été reçues. Le modifier pourrait supprimer des données collectées. [5]

Prévention des erreurs (bouton de confirmation) Ne vous inquiétez pas, je sais ce que je fais. [5]

1. Pas besoin d'en dire plus. Être efficace, c'est offrir des actions principales clairement libellées.
2. Un peu de diversité ne fait pas de mal tant que le style reste cohérent. Ex : « Ça y est est », « Et voilà ! » ou « C'est tout bon ! » pour commencer un message de confirmation.
3. Ne prenez pas pour acquis que c'est la faute de l'utilisateur. Peut-être que ce n'était pas évident que le champ était obligatoire.
4. Pas de « Oups ! ». C'est sérieux, l'utilisateur a probablement perdu du temps ou des données.
5. Quand vous avez besoin de l'attention de l'utilisateur, adressez-vous directement à elle / lui et faites des textes d'interface une conversation.