Rapport du projet SciMS

Camille Leplumey
Matthieu Loisel
Geoffrey Spaur

Introduction

Contexte

Étudiant en informatique dans le master GIL (Génie Logiciel et Informatique) et SSI (Sécurité des Système Informatique), nous avons été amené au cours de notre cursus à développer des applications web. Étant familiariser avec les langages web dès notre première année, nous avons modestement réalisé des projets permettant de gérer des colloques ou des QCMs. Nous avons poursuivi en troisième année avec un projet concernant la recherche d'hôtel et la réservation de chambres. A ce stade, nous avions connaissance du modèle MVC, cependant nous n'avions pas eu l'idée de l'appliquer sur une application web.

Objectif

Réalisant par ailleurs des projets personnels, nous avons pour habitude de partir de la page blanche sans architecture définie. Ayant pris connaissance de l'application du modèle MVC dans nos stages en entreprise, nous avons pris la décision pour ce projet, de construire une base solide et réutilisable pour nos futures projets universitaires ou personnels. Nous avons donc opté pour un modèle MVC en utilisant un framework. Jusqu'à ce jour nous n'avions jamais utilisé de framework PHP, mais cela nous paraissez être un plus afin de construire une architecture MVC. Nous avons donc chercher un framework avec une architecture MVC. Au fil de plusieurs discutions, le framework Silex est apparue comme la meilleur solution.
Donc outre la réalisation du projet, nous avons essayer ici de l'utiliser afin de construire une solide base pour nos futures projets.

Plan

Dans la suite de ce rapport, nous détaillerons dans un premier temps les technologie utilisées puis dans un second temps l'architecture mise en place et enfin la réalisation du projet.

Technologies

Silex

Avant de se lancer dans le développement du projet, nous avons dans un premier temps réfléchis sur les technologies que nous souhaitions utiliser. Nous avions durant nos stages découvert l'utilisation de framework coté serveur avec C# et Java. Nous avions donc eu l'envie de continuer la découverte des framework sur PHP. Nous avons donc cherché un framework simple pour débuter. C'est-à-dire un framework léger, simple à mettre en place et qui offre une prise de contrôle rapide.

Silex est un petit framework MVC PHP créer en 2010 issu de Symfony(2005). Nous avons utilisé Silex pour résoudre différent problèmes. Dans un premier temps, il permet de faire le lien avec d'autre Composant du framework de Symfony : Twig, pour les templates HTML et Doctrine, pour la gestion de la base de données. Mais nous utilisons Silex surtout pour une fonctionnalité importante : le routing.

Le routing nous était jusqu'alors inconnu. Il permet de faire le lien entre l'URL entrée dans le navigateur et la fonction d'un contrôleur à exécuter. Ainsi nous obtenons un fichier (/Config/Routes.php) remplie de routes. Au lancement de l'application, Silex regarde si l'URL correspond à l'une des routes que nous avons enregistrés, si c'est le cas alors le contrôleur associé est exécuté, sinon une erreur est retournée. Pour mettre en place le routing, il faut dans un premier temps rediriger toutes les requêtes vers un seul fichier. Nous sommes sur un serveur Apache2, nous allons donc créé un fichier .htaccess contenant cette instruction :

```
RewriteRule ^ Web/start.php
```

Une fois que vous avez redirigé l'intégralité des requêtes, vous pouvez enregistrer vos routes comme ceci :

```
$app->get('/SignIn', "\Web\Controllers\User::signIn");
```

Ici nous voyons que si nous entrons dans notre navigateur : http://scims.geoffreyspaur.fr/SignIn, Silex exécutera la méthode signIn dans le contrôleur User.

Nous utilisons par ailleurs une autre fonctionnalité liée au routing, la classe UrlGeneratorServiceProvider afin de pouvoir accéder à nos routes sans les marquer en dur dans le code. Pour cela nous devons apporter une modification à la déclaration de notre route :

```
$app->get('/SignIn', "\Web\Controllers\User::signIn")->bind("home");
```

Maintenant pour la création d'un lien dans notre vue, nous n'aurons plus qu'à appeler la fonction url avec en paramètre le bind de notre route :

```
<a id="linkhome" href="{{ url("home") }}">Accueil</a>
```

Une autre fonctionnalité que nous utilisons : l'autoload. Bien qu'invisible cette fonctionnalité nous permet de ne pas utiliser les fonctions d'inclusion de fichier (include, require,...). En effet cette fonction inclura automatiquement tous les fichiers PHP de votre projet. Ainsi vous pouvez utiliser la fonction use package\Class sans l'utilisation préalable de require. Pour mettre l'autoload en place, il faut installer Silex avec les bonnes options :

```
"autoload" : {
    "psr-4" : {
     "" : "./"
    }
```

```
    }
```
Ces lignes sont à mettre dans le fichier composer.json. Ici nous indiquons à Silex d'inclure tous les fichiers à partir de la racine du site. Il ne vous reste plus qu'à faire votre unique require afin d'inclure tous vos fichiers :
```
require_once '../vendor/autoload.php';
```
Nous pouvons dès à présent utiliser use \Folder\Class; pour pouvoir utiliser nos classes n'importe où dans le projet.

Twig

En complément à Silex, nous utilisons Twig afin de factoriser nos vues HTML. Twig est un gestionnaire de template. L'idée général est de présenter à Twig, un fichier HTML et un modèle PHP, et Twig s'occupera de fusionner les deux en plaçant les données du modèle dans la page HTML. Coté PHP, on utilise pour modèle un tableau associatif. Ce tableau peut contenir des objets dont les méthodes peuvent être utilisées du coté de la vue. L'appelle à Twig depuis le contrôleur se fait ainsi :
```
$app['twig']->render("Article/articles.html.twig", $article);
```
Maintenant nous pouvons passer à la vue afin de voir comment intégrer nos données dans le template. La syntaxe de Twig en plutôt simple :

"{{ }}" est utiliser pour afficher un contenu et

"{% %}" pour les structures logiques (if, block, for, ...)

Si nous supposons que notre $article est un objet possédant un champs Id, nous pourrions voir ceci :
```
<article article-id="{{ Id }}" >{{ Content | raw  }}</article>
```
Ici Twig affiche une chaîne de caractère, avant d'être affiché Twig s'occupera pour nous, d'échapper cette chaîne. Si vous voulez annuler cette effet, il faut utiliser l'option raw.
Avec un modèle plus complexe, nous pourrions aussi voir ceci :
```
{% if Model.article.CategoryId == category.Id %}
    <option selected="true" value="{{ category.Id }}">{{ category.Name }}</option>
{% endif %}
```

Twig permet aussi une autre fonctionnalité, celle d'étendre d'autre templates. Si l'on souhaite que toutes les pages de notre site possèdent le même header, il nous suffira d'utiliser la fonction extends de Twig. Admettons un fichier _Layout.html.twig :
```
<!doctype html>
<html>
    <head>
        <meta charset="utf-8" />
        ...
    </head>

    <body>
        {% block content %}
        {% endblock %}
    </body>
</html>
```
Nous n'avons plus qu'à étendre ce template pour nos autre pages en insérant notre contenue dans le

block content :

```
{% extends "_Layout.html.twig" %}
{% block content %}
   <div>...</div>
{% endblock %}
```

Doctrine

Doctrine est un module supplémentaire ajouté à Silex permettant une communication simple et homogène avec notre base de données. Nous utiliserons par le suite une base de donnée MySQL. Nous pouvons observer que les 3 fonctionnalités majeures de modification des données portent la même syntaxe :

```
$Db->update($tableName, $data, array('Id' => $object->get("Id")));
$Db->delete($tableName, array('Id' => $object->get("Id")));
$Db->insert($tableName, $data);
```

Toutes ces méthodes sont appelés sur l'objet `$Db` qui est typé par la classe : `Doctrine\DBAL\Connection`. Le premier argument est identique pour chaque méthodes : le nom de la table où les modifications seront apportées. L'argument `$data` est un tableau associatif représentant la table de notre base de données. Et le dernier argument (pour les deux première méthodes) est un tableau associatif permettant de remplir la condition `WHERE` de notre requête SQL. Chaque clefs correspondra à une colonne.

Pour sélectionner des données, il n'existe pas de méthodes `select`. Nous devons donc écrire nous même la requête SQL. Pour effectuer nos requêtes nous utilisons l'API `fetch` du framework Doctrine. Nous allons principalement utiliser deux méthodes : `fetchAll` et `fetchAssoc`.
La méthode `fetchAll` permet de retourner l'intégralité des résultats d'une requête, ces résultats seront transmit dans un tableau associatif. Vous retrouverez le nom de la colonne dans les clefs du tableau :

```
$Db->fetchAll("SELECT * FROM Users ORDER BY Id DESC");
```

Contrairement à `fetchAll`, `fetchAssoc` ne retourne que le premier résultat de la requête. Pour effectuer des requêtes variables, vous pouvez définir des paramètres dans la requête SQL et les transmettre sous forme d'un tableau associatif aux précédentes méthodes :

```
$sql = "SELECT * FROM Users WHERE Id=:id";
$line = $Db->fetchAssoc($sql, array("id" => $Id));
```

Toutes les méthodes précédemment citées sont protégées au injection SQL. Chaque valeurs des tableaux sont échappées avant d'être insérées dans la requête.

CSS3

Durant ce projet, nous avons eu le temps de nous attarder sur le CSS et plus particulièrement d'utiliser les nouvelles fonctionnalités de CSS3. En effet CSS3 propose quelques nouveautés très utiles pour mieux structurer son code CSS. Vous pourrez donc observer l'utilisation de variables. Bien que sa syntaxe soit lourde, elle permet de retirer toutes les constantes et ainsi les modifier à loisir. Pour pouvoir déclarer des variables accessible n'importe où, il faut déclarer ces variables dans le pseudo éléments root :

```
:root {
    --header-height: 10vh;
 }
```

Malheureusement il est nécessaire de commencer le nom de sa variable par "--". Ici la variable --header-height est accessible partout en utilisant cette syntaxe :

```
.header {
    height: var(--header-height);
}
```

Pour compléter cette fonctionnalité, il est possible d'effectuer des calcules entre plusieurs valeurs. Vous pouvez effectuer les opérations d'addition, de soustraction, de multiplication et de division en utilisant la fonction calc. Vous pouvez aussi effectuer ces opérations en utilisant vos variables précédemment définis.

```
.content {
    height: calc(100vh - var(--header-height));
}
```

Nous avons aussi beaucoup utilisés la propriété transition de CSS3. Cette propriété permet en théorie d'appliqué une animation au changement d'une autre propriété CSS. En pratique toutes les propriétés CSS ne sont pas animées comme le overflow(scrollBar) par exemple.

```
article {
    color: red;
}
article:hover {
    color: black;
    transition: color 0.5s ease-in-out;
}
```

Ici nous pouvons voir que les articles sont écris en rouge mais ils seront écris en noir au survol de la sourie. La propriété transition, nous permet d'ajouter un fondu enchaîné de 0,5 seconde sur la propriété couleur. Vous pourrez donc observé que la couleur deviendra de plus en sombre avant de prendre la couleur noir.

Il existe encore d'autre fonctionnalités que nous n'avons pas ou peu utilisées dans le projet comme les propriétés rotation ou translation. Il faut cependant faire attention car ces propriétés ne sont pas supportées par tous les navigateurs.

HTML5

Pour ce projet, nous avons jugé préférable d'utiliser HTML5 afin de disposer de certaines balises. En effet notre projet portant sur la réalisation d'article scientifique, il nous était évident d'utiliser les nouvelles balise article et section. Ces balises n'apportent aucun effet visuel, nous les utilisons afin de porter une sémantique plus forte.

```
<article>
  <section> ... </section>
  <section> ... </section>
</article>
```

Pour un meilleur confort d'utilisations nous utilisons aussi l'attribut required sur les input de nos formulaires. Ainsi l'utilisateur est susceptible de ne pas perdre les données du formulaire à la validation de ce dernier. L'attribut required empêche l'envoie du formulaire si le champs est vide. Nous utilisons aussi le type email pour l'input du courriel. Cette option permet aussi d'empêcher l'envoie du formulaire si le courriel entré semble incohérent.

```
<input type="email" name="email" required>
```

JavaScript

Pour ce projet, la totalité du JS est centralisé dans la partie de l'éditeur de texte du CMS. Pour s'y retrouver plus facilement, nous allons découper ceci en plusieurs parties :

Ajax :

Afin d'assurer une plus grande aisance d'utilisation à l'utilisateur, nous envoyons les éditions de texte par Ajax. De ce fait, l'utilisateur peux facilement sauvegarder son travail sans pour autant avoir besoin de sortir de son instance d'utilisation. Pour se faire nous utilisons la fonction

`createXhrObject() ;`

vue en cours, puis

`sendArticle() ;`

une fonction qui envoie de manière effective la requête au serveur.

Le mécanisme que nous utilisos ici s'appelle Ajax. Le principe est de pouvoir envoyer et ensuite recevoir des données du serveur sans pour autant avoir à recharger la page. Il est possible ainsi de naviguer de manière asynchrone et ainsi augmenter le nombre de fonctionalités présentes.

Dans le cas de 'sauvegarder' : nous envoyons au serveur la mise à jour de l'article pour que le serveur prenne en compte toutes les modifications effectués par l'utilisateur. La réponse du serveur est ici inintéressente et nous nous en préoccupons pas (mise à part pour le cas d'erreur dans l'envoi de données au serveur).

ExecCommand :

Une grosse partie des éditions effectuées par notre éditeur de texte passent par la fonction ExecCommand, une fonction récente qui permet une édition facile du style d'éléments éditables sur lequel on l'applique. La fonction

`commande() ;`

vérifie la possibilité d'utilisation de execCommand. Nous ne l'avons pas faite nous même.

Un exemple de l'utilisation d'execCommand

`commande('bold') ;`

qui passe l'élément sélectionné en gras.

Nous ne savons pas la méthode exacte que execCommand utilise pour arriver à ce résultat, mais,

pour un des cas (link) nous avons décidés de la faire à la main. Nous pouvons imaginer que execCommand utilise une méthode pus ou mois similaire.

Architecture

Introduction

Pour ce projet, nous avons utilisé pour la première fois le modèle MVC dans un projet web universitaire. Nous allons dans cette parte détailler l'intégration de ce modèle dans notre application web.

Mise en pratique du modèle MVC

Modèle

Tous les fichiers concernant le modèle se trouvent dans le dossier /Domain/Model. Toutes les classes présentent dans ce dossier représente la structure de notre base de données. Pour chaque classes vous trouverez une table associée portant ce même nom au pluriel. Dans cette table vous trouverez les colonnes représentant les attributs de notre classe. Toutes les classes du modèle hérite d'une même classe EntityWithId, cette classe possède un attribut Id qui sera généré aléatoirement à chaque création.Cette fonctionnalité permet d'assurer une clef primaire dans nos différentes tables.

```
$this->Id = uniqid();
```

Vous pouvez remarquer que la classe EntityWithId hérite de la classe Entity. Cette classe permet de réduire le nombre de fonction dans les classes qui en hérite. L'idée général est de ne plus à avoir à écrire des getters et setters vides. Donc vous trouverez 2 fonctions dans la classe Entity :

```
public function get($property)
public function set($property, $value)
```

Pour obtenir la valeur d'un attribut ou en définir sa valeur, vous devrez utiliser ces précédentes fonctions. Si dans votre classe aucun getter (setter) n'est définie pour une propriété, la fonction get (set) appliquera le mode par défaut : retourner (modifier) la valeur. Si vous voulez que la valeur retourner (modifier) soit formaté ou altéré, vous pouvez définir une méthode get{Attribut} (set{Attribut}) pour que la méthode get (set) l'exécute.

```
public function get($property) {
  $rc_this = new ReflectionObject($this);
  $method = "get".$property;
  if ($rc_this->hasMethod($method)) {
     return $this->$method();
  }
  if ($rc_this->hasProperty($property)) {
     return $this->$property;
  }
  return null;
}
```

Pour chaque classes dans le modèle vous pouvez créer un fichier DAO dans le dossier /Domain/DAO. Vous disposerez alors d'un outil de communication avec la base de données pour mettre à jour le modèle. Nous avons pris pour convention de nommé nos fichiers DAO sous cette forme "{ModelName}DAO". Avec cette convention, nous avons pu écrire une classe ultra générique. En effet vous pouvez remarquer que toutes nos classes DAO hérite d'une même classe : « DAO ». Cette classe propose 5 fonctionnalités basique pour communiquer avec notre base de

données :

```
public function select($Id)
public function selectAll()
public function insert($object)
public function update($object)
public function delete($object)
```

Rappelons que chaque classe possède un attribut Id, nous avons alors créer une fonction qui permet de sélectionner un objet avec son Id. Une fonction permettant de retourner tous les objets d'une table. Et d'autre fonctions permettant de mettre un objet à jour dans la base de données.
Le retour des requêtes SQL forme un tableau associatif, travaillant avec des objets, nous avons ajouter une fonction permettant de convertir le tableau associatif en objet :

```
protected function objectConverter($sqlResponse)
```

Les classes DAO sont des singletons, nous les instancions donc dès le lancement de l'application dans le fichier /Config/App.php.

```
$app['dao.article'] = function ($app) {
    return new \Domain\DAO\ArticleDAO($app['db']);
};
```

Vue

Tous les fichiers concernant la vue se trouve dans le dossier /Web/Views/. Chaque vue peut obtenir potentiellement un tableau associatif contenant les informations dynamiques à placer dans la page, ce tableau est transmit par le contrôleur. Nous avons aussi factoriser nos vus dans des templates que nous nommons "_Layout[...]". Ainsi nous ne définissons qu'une seul fois nos headers. Pour chaque méthodes de contrôleur qui nécessite une vue, vous trouverez un fichier portant le nom de cette méthode dans le dossier portant le nom du contrôleur.

Dans quelques rares cas, la vue peut appeler un contrôleur. On retrouve ce cas de figure pour l'affichage des catégories par exemple :

```
{% block topBar %}
    {{ render(controller("\\Web\\Controllers\\Category::categories")) }}
{% endblock %}
```

Contrôleur

Comme dans un modèle MVC classique, le contrôleur sert à faire le lien entre le modèle et la vue. Tous les fichiers concernant le contrôleur se trouvent dans le dossier /Web/Controllers. Ayant adopté certaines conventions définie dans le chapitre Vue, nous pouvons créer quelques fonction pouvant nous facilité la vie. Toutes ces fonctions sont dans la classe Controller que tous les autres contrôleur hériteront.

Nous disposons donc de 2 fonctions pour nous aider : une fonction de redirection et une fonction pour appeler notre vue associée. La fonction de redirection prend en paramètre le bind d'une route et redirige la requête vers le contrôleur associé :

```
protected function redirectBind($app, $str) {
    return $app->redirect($app["url_generator"]->generate($str));
}
```

La fonction pour appeler notre vue est certes très pratique mais nous avons conscience de sa complexité spatial. En effet, il suffit de lui donner ou non le modèle de la vue, et la fonction

renverra ce modèle sur la bonne vue associée. Pour savoir quel vue nous devons appeler, nous utilisons la fonction getClassName pour le dossier et debug_backtrace pour le fichier :

```
protected function render(Application $app, $object = null) {
    $method = debug_backtrace()[1]['function'];
    $className = $this->getClassName($this);
    $data  = ObjectConverter::anyToArray(array("Model" => $object));
    return $app['twig']->render($className."/".$method.".html.twig", $data);
}
```

La vue prend pour modèle un tableau associatif, il faut donc convertir nos objets ou nos tableaux d'objets en tableau associatif. Il est à noter que cette opération n'est pas obligatoire. Nous convertissons nos objets à l'aide d'une méthode utilitaire :

```
public static function anyToArray($objects)
```

La plupart du temps les objets que nous donnons à la vue sont des entités, mais ce n'est pas toujours le cas. Prenons l'exemple du menu, nous avons différents lien sur des images, nous avons créé un modèle représentant cette structure : /Web/Models/MenuItemModel.php. Nous n'avons plus qu'à instancier ces objets avant de les transmettre à la vue.

```
$menu = array(
    new MenuItemModel(
        $this->urlBind($app, "subscription"), "Mes abonnements", "star.png"),
    new MenuItemModel(
        $this->urlBind($app, "articlesByUser"), "Mes articles", "folder.png"),
    new MenuItemModel(
        $this->urlBind($app, "profile"), "Profile", "user.png"));
return $this->render($app, $menu);
```

Compléments

En complément à cette architecture, nous avons un dossier /Utilities/. Dans ce dossier nous plaçons toutes les méthodes static dites outils. Ces méthodes nous permettent de convertir des objets ou d'effectuer divers tests.

Pour finir nous avons le dossier /Web/Content/ où nous rangeons les feuilles CSS qui sont découpées en fonction des pages du site ; les feuilles JavaScript et enfin, des dossier contenant les polices d'écriture et les images utilisées.

Projet

Introduction

Ce projet a pour but de répertorier en ensemble d'articles scientifiques. Un utilisateur doit pourvoir chercher, visualiser et ajouter un article. Différentes fonctionnalités doivent être pris en charge :

- L'authentification d'un utilisateur
- Filtrer les articles à l'aide de bouton ou d'un champs de recherche
- Créer une interface claire d'édition d'article.
- Créer une interface d'administration et de statistiques

Nous allons donc dans cette partie nous arrêter sur chacun de ses points afin de les détailler et d'exposer les difficultés que nous avons pu rencontrer.
[…]

Authentification

Pour accéder aux fonctions de connexion et d'inscription, vous pouvez cliqué sur l'icône en forme de profile dans le header de la page. Vous aurez alors deux formulaires, l'un pour vous identifier, l'autre pour vous connecter.

L'inscription nécessite 3 informations : votre pseudo, votre email et votre mot de passe. Ces 3 champs sont obligatoires. Suite à la validation du formulaire, vous serez automatiquement connecté au site et redirigé sur la page d'accueil pour continuer votre navigation. Le mot de passe est chiffré dans notre base à l'aide de la fonction `sha1` En cliquant sur l'icône de profile, vous pourrez modifier vos informations ou vous déconnecter.

Le formulaire de connexion vous permettra d'accéder à vos informations personnelles et ainsi qu'à diverses fonctionnalités. Pour remplir le formulaire vous devrez vous munir de votre email ainsi que de votre mot de passe. A la validation du formulaire et si ces information sont vérifiées, vous serez authentifié et redirigé sur la page d'accueil. Pour éviter la faille "session fixation" votre identifiant de session sera régénéré à chaque connexions.

```
session_start();
session_regenerate_id();
```

Silex gère les mécanismes de session. Il nous donne accès à un tableau associatif à l'aide des fonctions `get` et `set` sur l'objet `session`.

```
$app["session"]->set("userId", $user->get("Id"));
```

Toutes les informations liées à un utilisateur est en permanence stocké dans notre base de données. Nous n'avons plus qu'à enregistrer l'identifiant de notre utilisateur dans notre session pour avoir accès à toutes ces données. Par conséquent nous stockons uniquement l'identifiant de l'utilisateur dans sa session.

Filtre

Pour permettre une navigation plus efficace et agréable aux utilisateurs, nous avons mis à leur disposition une série de filtres simples qui permettent de préciser la sélection de leur choix.

Affichage des catégories :

Une des premières choses que l'on remarque lorsqu'on voit la page principal du site est l'apparence multicolore. Chaque catégorie d'article possède un code couleur pour une identification facile et rapide des articles. Ainsi chaque article possède une banderole de couleur correspondant à sa

catégorie. Lorsqu'on clique sur une des catégories, le filtre reconnaît que l'utilisateur ne souhaite voir que les articles de cette catégorie. De plus les catégories sont classés en ordre de pertinence : plus une catégorie possède un grand nombre d'articles, plus il sera vers la gauche de la liste des catégories. S'il y a un nombre suffisant de catégories pour que les catégories sortent de l'encadrement de la page, l'utilisateur peux cliquer sur '...' qui lui offrera un panneau contenant la totalité des catégories. Finalement, un peu partout dans le site, lorsqu'une liste de catégorie est offerte à l'utilisateur, cette liste sera triée en ordre de popularité

Barre de recherches :
Une des fonctionnalités omniprésente dans le site est la barre de recherche. Celle-ci permet d'effectuer une recherche afin de trier les articles en fonction d'un mot-clé. Ce mot clé peux être un titre, un auteur, un tag, un utilisateur ou une catégorie. Le résultat des recherches seront les articles qui répondent à un ou plusieurs de ces critères.

Mécanisme privée/publique :
Un dernier point intéressant à noter est le mécanisme de privée/publique. Si un utilisateur décide que son article n'est pas encore prêt pour la lecture, il peux décider qu'il sera la seule personne qui pourra y accéder de par le référencement du site: l'article ne sera référencé que dans l'onglet 'mes articles' de l'éditeur. Néanmoins, il peux donner l'URL de son article à d'autres utilisateurs afin qu'uniquement ceux-ci puisse le lire. Par défaut, un article est privé à sa création.

Interface d'édition

Voici la partie principal, la plus importante du site : l'interface d'édition. Ici l'utilisateur peut rédiger son article. Pour lui venir en aide, nous lui fournissons un ensemble d'outils tels que :

Champ du formulaire :
L'éditeur de texte possède un formulaire permettant l'édition des attributs de l'article soit : la manipulation des tags de l'article, la mise à jour du titre, des auteurs et des catégories. On peut aussi décider de la visibilité de l'article (publique ou privé). Finalement, on peut sauvegarder l'état courant de l'édition de l'article en appuyant sur le bouton 'sauvegarder'.

Commandes :
Ici une liste des commandes disponibles à l'utilisateur :

Bold : met en gras la sélection

Italic : met en italique la sélection

Underline : souligne la sélection

JustifyLeft : Aligne à gauche la sélection

JustifyCenter : Aligne au centre la sélection

JustifyRight : Aligne à droite la sélection

JustifyFull : Justifie la sélection

LowerFont : Baisse la taille de police de caractère

IncreaseFont : Augmente la taille de police de caractère

Undo : annule la dernière action effectué

Redo : annule les dernières actions de type ‘undo’

Section : ajoute une section (voir gestion des sections)

Summary : ajoute un sommaire (voir gestion du sommaire)

Latex : ajoute une formule latex

Link : transforme la sélection en lien hypertexte (voir ajout d’un lien)

Mécanisme des tags :
L’utilisateur a l’option de rajouter une liste de tags à son article en les entrant au travers du champ de formulaire prévu à cet effet. Ceci aura deux effets : premièrement d’envoyer au serveur en Ajax le nouveau tag rajouté à placer dans le champ ‘tags’ de ‘Article’. Ce champ est une chaîne de caractères avec tous les tags entrecoupés de ‘;’. Ensuite le JS rajoute à la vue le nouveau tag afin que l’utilisateur puisse admirer ses différents tags étant donné que, à l’origine, ces tags sont générés en faisant appel à la base de donnée (et donc si on ne faisait qu’envoyer le tag au serveur, la vue ne s’en verrait modifié que si l’utilisateur rafraichissait la page).

S’il le souhaite, l’utilisateur peux effacer un tag en cliquant sur le ‘[X]’ qui lui est associé. Ceci effacera le tag ainsi que tous les tags qui lui sont identiques en texte. Le principe de mise-à-jour employé pour la suppression est identique à l’ajout.

Ajout d’un lien :
Étant une commande basique qu’on aurait pu effectuer en utilisant execCommand, nous avons voulu essayer d’en faire au moins une nous même.
Pour faire le lien nous créeons un élément ‘a’ dans lequel nous plaçons la sélection (si la sélection est non-vide) en tant que texte et en tant qu’attribut href (que nous formatons en ajoutant un ‘https://’ devant). Ceci fait, nous l’insérons en utilisant
`insertHTML(node,range = null)`
(une fonction que nous utiliserons à chaque fois dans les prochaines sections) qui insère le node passé en paramètre au début de la sélection de l’utilisateur. Ensuite nous détruisons le node contenant la sélection de l’utilisateur afin que notre nouveau node prenne sa place.
Nous supposons que ‘execCommand’ à un fonctionnement plus ou moins similaire.

Gestion des Sections :
Afin de permettre à l’utilisateur de pouvoir découper son article en plusieurs sous-parties, nous lui permettons de créer des sections.
Une section est composé d’un titre (dans des balises <H1>), de texte (dans des balises <p>) et d’autres section.
Pour gérer les sections nous nous sommes assurés que les titres des sections étaient précédés d’un numéro signifiant leur position dans la structure de sections. Exemple :

1. Première section
Mon texte est ici
1.1 Première sous-section de ma première section
1.2 Deuximème sous-section de ma première section
2. Deuxième section.
…

Ces numéros sont générés dynamiquement lors de la création de nouvelles sections.
Pour ce faire nous avons joués avec la structure arborescente du DOM en insérant à l'endroit de la sélection l'élément voulu.

Chaque section possède son propre Id généré automatiquement et correspondant à son numéro sauf que chacun de ses '.' sont remplacés par des 's' :
Id de 1.2.1 : 1s2s1

Il est intéressant de noter, cependant, que si l'utilisateur décide d'intervenir lui-même sur les nodes à travers les outils de son navigateur web ou s'il importe du contenu d'une autre source. Pour que la gestion des sections soit efficace, il est nécessaire que la structure imposé par nos algos soient respectés. Lors d'ajouts de nouvelles sections certaines méthodes vérifient l'intégrité de la structure arborescente et essaient de corriger un certain nombre d'erreurs possible soient, par exemple :
`setHierarchy(node, parent)`
qui assure la bonne hiérarchie de la nouvelle section ajoutée.

De plus, afin de faciliter la compréhension de l'utilisateur, une indentation et une ligne de démarcation rouge, présents uniquement dans la phase d'éditions, ont étés ajoutés.

Gestion du sommaire :
Pour qu'un article soit digne de ce nom, il est nécessaire d'avoir un sommaire au début dudit article. Ce sommaire répertoriera toutes les sections des articles avec leur numérotation respectives.
Cet élément est une div non-éditable ayant des liens 'a' comme node enfants. Ces nodes enfants font référence aux sections respectives qu'elles décrivent et, quand sélectionnés renvoient l'utilisateur vers la section sélectionné.
Son fonctionnement est comme tel:
On construit d'abord la div qui contiendra le summary. Une fois ceci fait,
`constructSummary(nodeList, summary)`
se charge de créer tous les liens à partir de la nodeList passé en paramètre correspondant à la liste des sections (identifié par leur classname). Ils utilisent l'Id généré automatiquement des sections comme cible.
Si l'utilisateur a déjà un sommaire existant, le sommaire est mis-à-jour en supprimant tous ses éléments et en les regénérant.

Interface d'administration et des statistiques

Fonctionnalité administrative :
Pour ce rendre sur la page d'administration à partir de la page home du site web il faut s'authentifier avec un compte ayant les droits super-utilisateur . Pour le moment le seul compte super-utilisateur que nous ayons a pour adresse mail a@a.c et pour mot de passe «a». Après connexion il vous suffit de rentrer l'adresse : http://scims.geoffreyspaur.fr/Administration/. Si vous n'êtes pas connecté en tant que super-utilisateur, vous serez redirigé vers la page permettant la connexion.

Sur cette page d'administration, vous avez accès à trois tables. Une pour la gestion des utilisateurs, où vous retrouvez leur pseudo, leur adresse mail et un bouton de suppression de l'utilisateur. La seconde table permet la gestion des articles en affichant leur titre, l'adresse mail du rédacteur, leur statut public ou privé, et encore une fois un bouton de suppression. La dernière table permet la gestion des catégories avec leur nom, leur couleur, et cette fois un bouton d'ajout dans la première ligne du tableau, et deux boutons de suppression et d'édition par ligne de catégorie.

L'appuie sur l'un des boutons de suppression provoque la suppression de la ligne contenant le bouton sans demande de confirmation. Concernant le bouton d'ajout contenu dans la troisième table, il redirige sur un formulaire demandant le nom de la catégorie et une couleur associée, nous permettant sa création. On récupère alors ce qui a été rentré dans ce formulaire grâce au contrôleur pour en faire l'ajout dans notre base de données. Pour finir, les boutons d'éditions permettent de modifier les catégories déjà existantes, en utilisant le même formulaire que celui utilisé pour l'ajout.

Information statistiques :
Pour accéder aux informations statistiques du site web, il faut s'authentifier avec un compte ayant un rang super-utilisateur puis il vous faut rentrer l'adresse suivante : http://scims.geoffreyspaur.fr/Statistique/.

Vous pouvez ainsi accéder à certaines informations sur le site, allant du nombre d'utilisateur jusqu'à une liste des articles. Toutes ces informations ont été récupéré dans nos tables grâce aux requêtes sql présente dans les fichiers DAO.

Pour récapitulé les différentes informations mis à votre disposition, vous pouvez connaître le nombre total d'utilisateur inscrit, le nombre d'utilisateur inscrit n'ayant écrit aucun article, le nombre d'article présent sur le site, le nombre d'article privé sur le site, et le nombre d'article publique présent sur le site. Pour finir la liste de tous les articles présent sur le site qu'il soit publique ou privé. De plus si vous cliquez sur le titre d'un article vous serez redirigé vers l'article en question.

Fonctionnalité d'abonnement

Nous avons donné à l'utilisateur la possibilité de pourvoir suivre d'autres utilisateurs. Pour ceci vous devez dans un premier temps vous inscrire ou vous connectez avec votre compte.

Puis si l'un des articles vous plaît en le consultant, vous pouvez suivre le rédacteur de cet article en cliquant sur s'abonner à coté du pseudo du rédacteur. Votre identifiant et celui du rédacteur seront alors ajoutés dans une entré de notre table Followers. Ceci nous permettra par la suite de vous affichez tous les articles, des rédacteurs que vous suivez, ayant été modifiés récemment. En effet, vous pouvez obtenir cet affichage en appuyant sur l'icône d'étoile dans le menu.

Pour obtenir ce résultat nous récupérons votre identificateur dans la session courante puis nous la comparons avec les FollowerId de la table Followers, on récupère ainsi les identificateurs des utilisateurs auxquels vous êtes abonné. Il ne nous reste plus qu'a trouver leurs articles et les trier par ordre décroissant de date de dernières modifications avant de vous les afficher.

Une fois abonné à un utilisateur le bouton vous ayant permis l'abonnement est changé pour vous permettre de vous désabonné d'un utilisateur.

Idées d'améliorations

Nous regroupons ici quelques idées d'amélioration de notre site web qui, avec plus de temps, aurait possiblement pu être implémenté.

- Pagination des pages ou chargement ajax.
- Notification par mail de l'actualité du site
- Ajouter un commentaire aux articles
- Ajout de la fonctionnalité : mot de passe oublié
- Liste déroulante pour l'affichage des catégories

Bibliographie

Silex

CSS3
https://developer.mozilla.org/fr/docs/Web/CSS/:root
https://developer.mozilla.org/fr/docs/Web/CSS/calc