Franck Caron
M1 GIL

Projet de Langages WEB 1

Plan

1. Présentation du projet

2. Processus de développement
   - 2.1 Modèle de données
   - 2.2 Description de l'architecture du site
   - 2.3 Réalisation des fonctionnalités
     - 2.3.1 Menu latéral
     - 2.3.2 Page d'inscription
     - 2.3.3 Interface d'administration
     - 2.3.4 Consultation des articles/catégories
     - 2.3.5 Interface d'édition

3. Manuel d'installation

1. Présentation du projet

Cette partie visera à définir le sujet et le but du projet de manière synthétique. Elle décrira également les conditions de réalisation ayant influencé le développement du site.

Le sujet consiste en l'élaboration d'une application Web, permettant à des utilisateurs d'y publier des articles à caractère scientifique.
Pour cela, l'applicatif doit fournir des fonctionnalités s'apparentant à celles d'un CMS (Content Management Software).
Il devra :

- Simplifier la saisie de nouveaux articles.
- Trier les articles dans des thématiques communes, dites « catégories »
- Faciliter la gestion des collaborateurs et des catégories présentes sur le site
- Permettre une visibilité optimale des articles publiés

Le sujet définit également un liste de contraintes qui caractérisent le comportement que devra avoir l'applicatif lors de son utilisation. Ainsi, l'applicatif doit définir des rôles utilisateurs (visiteur, rédacteur, administrateur), fournir des outils d'édition spécifiques (SVG, Images), etc.
Parmi l'ensemble des contraintes spécifiées, seules deux d'entre elles ont été ignorées lors de la réalisation :

- L'URL d'accès à une catgéorie/article :
  Ici, le site n'associe pas d'URL affichant l'arborescence des catégories à une catégorie lors de la visualisation. Elle est remplacée par un système similaire présent directement dans la page.
- La liste des références en fin d'article :
  Il n'y a pas de support particulier pour leur écriture, juste les options d'éditions fournies.

En ce qui concerne le développement du projet, la réalisation de celui-ci n'a employé aucun framework externe, et n'a utilisé que des APIs étudiées en cours (API DOM et PDO).
Le code fourni pour la production de la solution, est réparti sur les 4 langages usuels : HTML pour la structuration, CSS pour l'esthétique, JavaScript pour la programmation côté client, et PHP côté serveur.

De ce fait, l'applicatif ne requiert aucune configuration spécifique : un simple serveur PHP couplé à un SGBD, sera le strict nécessaire pour profiter des fonctionnalités du site.

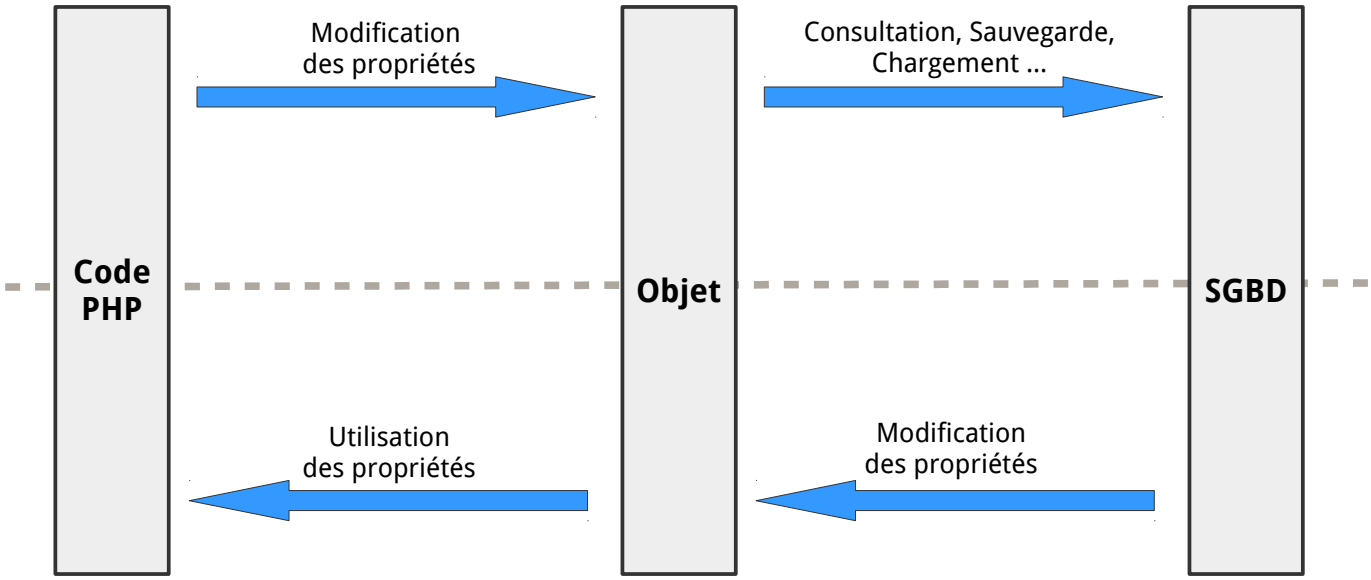
La suite de ce document, visera à expliciter la démarche de conception employée afin de réaliser l'applicatif. Le plan établi suit alors mon propre itinéraire de développement, tout en veillant à expliciter les différents mécanismes qui interviennent dans la mise en œuvre d'une fonctionnalité donnée.

2. Processus de développement

2.1. Modèle de données (PHP)

Le premier axe de travail sur ce projet a été d'établir un modèle de données efficace, pour pouvoir simplifier toutes les démarches des différentes parties de l'applicatif.
En effet, la majorité des fonctionnalités requises repose sur des opérations de manipulation de données plutôt simples, tels que l'ajout, la suppression et la modification de données quelconques.

On peut donc modéliser un objet PHP servant de passerelle entre une ligne de donnée, et des variables de formulaires/sessions. Le but de cet objet serait alors de simplifier les démarches auprès du SGBD distant :



Chaque objet dispose alors d'attributs, ce qui correspond réciproquement à une ligne et à une case de la table de données qui lui est associée.

On définit ainsi l'interface DBObject (php/model/db/DBObject.php) qui modélise l'ensemble des objets PHP pouvant interagir avec une base de données distante. On retrouve dans le même fichier la classe abstraite AbstractDBObject, qui factorise le comportement commun à tous les objets d'interaction.

Chaque objet peut alors disposer d'un numéro d'identification commun avec celui enregistré comme clé primaire dans la BDD. Si cette clé est définie pour l'objet, on dira que le lien est établi, les opérations possibles sont alors celles de consultation, de mise-à-jour et de suppression. Sinon le lien est à établir, ce qui requiert de l'ajouter dans la base de données, avant de pouvoir l'exploiter entièrement.

L'intérêt de cette modélisation étant de pouvoir uniformiser les comportements des types de données requis (utilisateur, catégorie, article).

Fonctionnalités de classe :

- **Exécution d'une requête SQL** (requestTable)
  Cette fonction permet de lancer une requête sur une table donnée à partir d'un tableau de critères et d'un suffixe de requête. Un tableau de critères correspond à un tableau de valeurs devant être vérifiées par tous les objets lus, alors que le suffixe permet de rajouter des conditions supplémentaires sur l'extraction (ordre, etc.).
  Elle renvoie l'instance PDO à la suite de l'exécution de la requête, ce qui lui permet d'être la base de toutes les fonctions plus complexes...

- **Chargement depuis le SGBD :** (extractFromTable)
  Permet de charger des lignes de tables (extractFromTable) depuis la table spécifiée. C'est la fonction de chargement principale qui permet de récolter les informations du SGBD pour pouvoir construire ultérieurement les instances.
  Pour les classes filles, on définit la fonction (load) qui ajoutera le traitement supplémentaire d'instancier les objets en fonction des lignes lues.

- **Présence dans le SGBD :** (inTable)
  Indique si une valeur existe déjà dans une base de donnée, cela correspond à tester l'existance du numéro d'identification de l'objet dans la table.

Fonctionnalités d'un objet instancié :

- **Sauvegarde/Inscription :** (saveChanges/register)
  Permet d'écrire l'état actuel de l'objet au sein de la base de données. L'inscription n'a lieu que si le lien de l'objet n'est pas établi.

- **Suppression dans la base :** (unregister)
  Supprime la ligne correspondant à cette objet dans la base de données.

- **Test de validité :** (isUsable)
  Détermine si l'état actuel de l'objet peut-être écrit dans la base de données. Il s'assure de la validité des données, tant en cohérence qu'en forme, avant la sauvegarde.

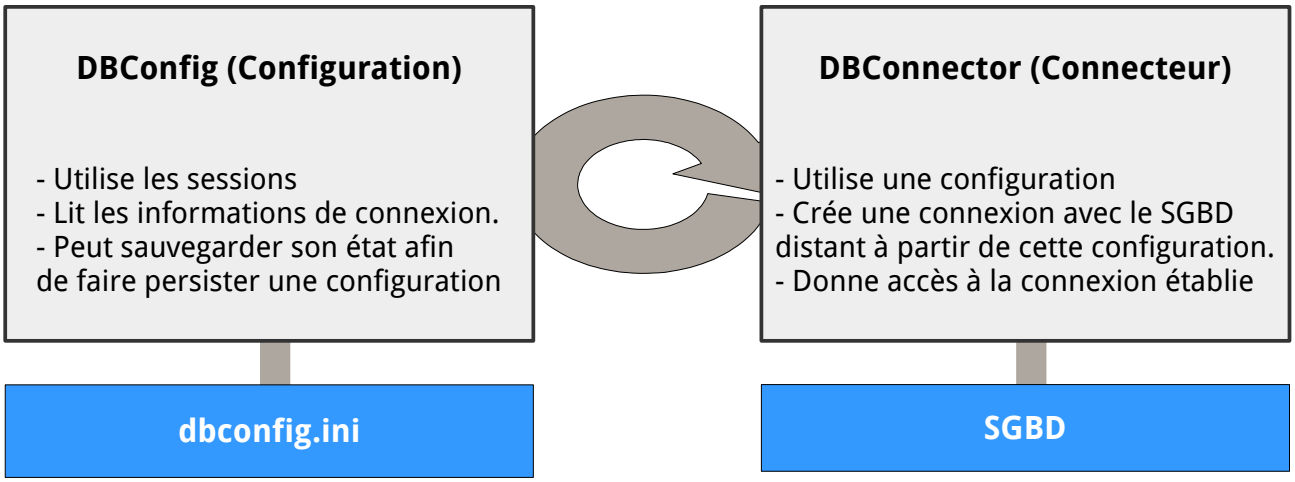
- **Accesseurs :** (get, set, getAll, setAll)
  Permet de modifier/consulter les valeurs de l'objet. Les varaintes « All » permettent d'intéragir à l'aide de tableaux d'attributs.

Au final, nous disposons de 3 types d'objets à modéliser. On crée alors les classes User (php/model/db/user.php), Article (php/model/db/article.php) et Category (php/model/db/category.php), en tant que classes filles d'AbstractDBObject, qui modéliseront les objets PHP que nous devront manipuler dans les applicatifs.

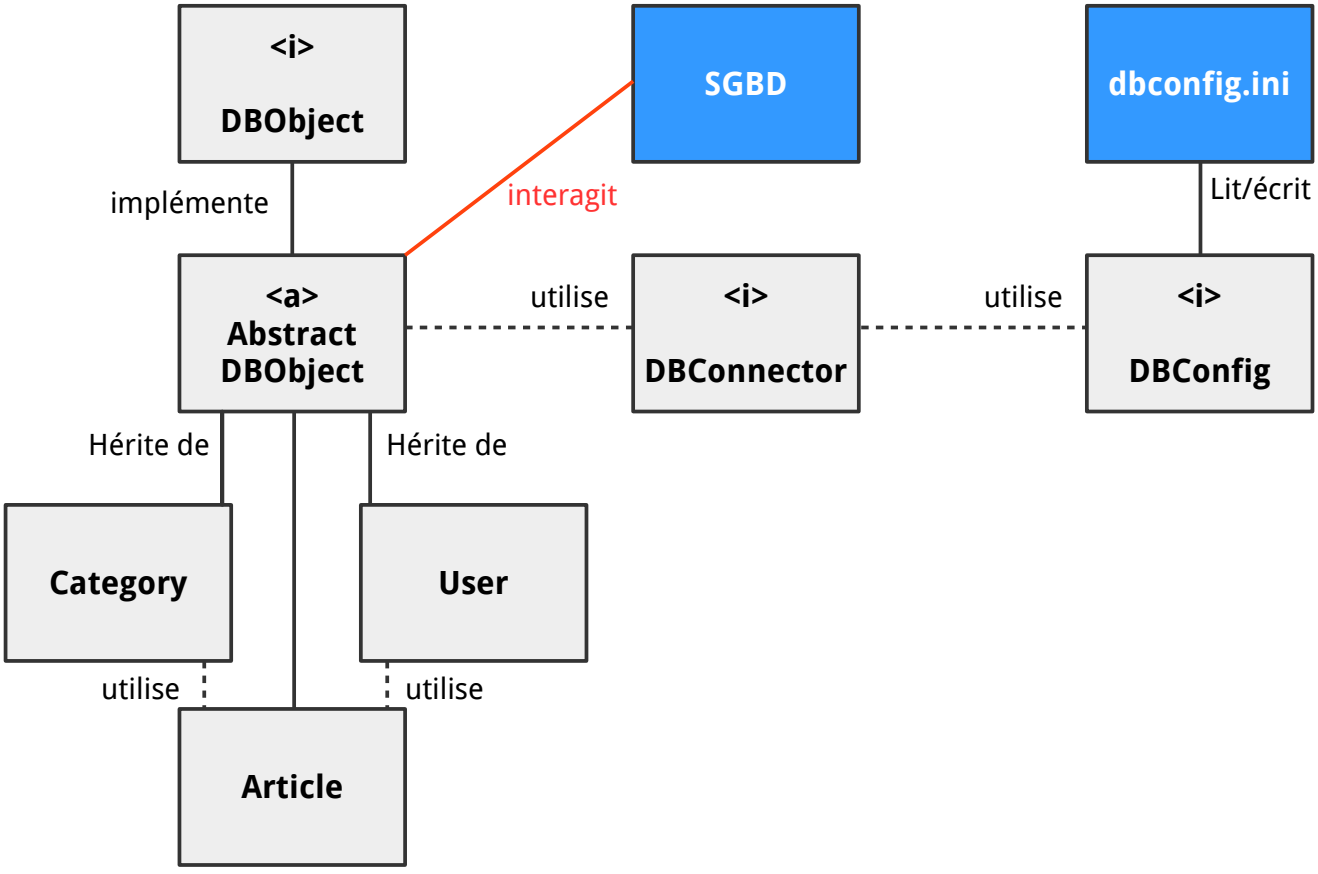
Cependant, une contrainte de taille réside dans la possibilité de réaliser la liaison avec le SGBD distant. En effet, vu qu'une des contraintes principales est de proposer une configuration simple du site. Il est indispensable de stocker les informations de connexion dans un fichier accessible (« en dehors du code » donc).
Le problème étant que le chemin d'accès au fichier dépendra de l'emplacement de la page en cours d'uitlisation.

A cet effet, nous allons utiliser le mécanisme de session afin d'enregistrer le chemin d'accès au fichier de configuration d'une page quelconque. Mais il faudra toujours une couche objet pour simplifier l'extraction de ses informations et la création d'une connexion avec le SGBD distant.

D'où l'intérêt des classes de configuration (DBConfig) et de connecteurs (DBConnector) :



Ainsi, on obtient le modèle de données suivant :



2.2. Description de l'architecture du site (PHP/HTML/CSS)

Une fois le modèle de données établi, on peut commencer à construire la structure du site en définissant le type d'architecture à utiliser, et la structure générale d'une page.
La manipulation de données étant omniprésente dans cette application, la majorité des pages consisteront à mettre en forme des formulaires de saisie, que ce soit pour l'édition (saisie de texte), l'inscription (saisie d'informations), ou le panneau d'administration (modification via tableaux de valeurs).

Ainsi, il semble pertinent d'employer une architecture MVC :

- La partie vue consistera à afficher une structure de page.
- La partie contrôleur fournira le formulaire/interface, adapté à la situation. Chaque page possédera un ou plusieurs contrôleur(s), selon le type d'action à mener (menu de redirection/ édition).
- La partie modèle traitera les demandes en arrière-plan au moyen de requêtes GET ou POST. Cela s'effectuera via une page .php spécifique.

La mise en place physique est réalisée en découpant le répertoire php en 3 sous-répertoires : view, model et controller.

Nous allons maintenant raisonner en termes de composants, et spécifier les besoins pour chacune de ces 3 parties : En premier lieu, la vue doit avant tout définir l'allure du site. Elle sera donc structurée, de manière à suivre le même design pour toutes les pages. Ainsi, on factorisera les composants communs de la vue : le haut de page et le menu de navigation.

On peut alors scinder une page en 3 parties :

- Le haut de page : Associé à la vue
- Le menu : Associé au contrôleur
- Le contenu : Défini par chaque page de manière spécifique (Contrôleur ?, Vue ?)

Le haut de page ne correspond pas seulement qu'à la trame de fond. Il contient des fenêtres d'affichage permettant de consulter des notifications à destination de l'utilisateur, ou des messages d'erreur en cas d'échec de l'applicatif. Afin de mettre son mécanisme en place, on utilise des variables de session spécifiques, qui contiennent une notification/erreur à afficher ?

Celle-ci s'affichera alors dans le haut de page, lors de l'ouverture de la même page.

Par exemple, en cas d'échec d'authentification :

Quant au menu, il permet de naviguer entre les différentes pages du site, et contient également l'encart de connexion permettant de s'authentifier ou de s'inscrire. Il est relié à plusieurs contrôleurs, dont l'étude sera faite dans la partie suivante.

Enfin, le bloc de contenu est propre à chaque page et lui permet d'assurer son utilité. Autrement dit, c'est ce composant qui contiendra toutes les sous-fonctionnalités attendues, avec tous les sous-composants nécessaires.

2.3. Réalisation des fonctionnalités (PHP/HTML/CSS/JS)

Cette partie va détailler la façon dont chaque grande fonctionnalité est réalisée. Chaque fonctionnalité reposant en partie sur le mécanisme de session Web, voici une description des variable de sessions PHP, utilisées par l'applicatif :

Nom	Description	Utilité
$_SESSION ['user']	Contient les informations de l'utilisateur courant. Permet de définir si : • L'utilisateur courant n'est qu'un visiteur (la variable n'est pas défnie). • L'utilisateur est un contributeur (varaible définie) • L'utilisateur est un administrateur (attribut isAdmin).	Contrôler l'accès à certaines parties du site. Renseigner les informations de l'utilisateur courant.
$_SESSION ['dbconfig']	Contient le chemin vers le fichier de configuration utilisé pour la connexion avec la base de données.	Permet d'activer les fonctionnalités des objets DBObject.
$_SESSION ['note']	Contient le message de notification à afficher. Se supprime automatiquement après le 1er affichage.	Permet de mieux informer l'utilisateur.
$_SESSION ['error']	Contient le message d'erreur à afficher. Se supprime automatiquement après le 1er affichage.	Permet de signaler une erreur critique.

$_SESSION ['page']	La dernière page du site consultée	Utilisée pour les retours à la page, dans les formulaires de traitement.
$_SESSION ['admin_edit']	Les informations sur le type d'objet en cours d'édition dans l'interface d'administration	Permet d'utiliser l'interface d'administration pour un type précis d'objet à éditer (cf 2.3.3)

2.3.1. Menu latéral

Le menu latéral est divisé en 3 parties :

La première partie correspond à la barre de connexion. Son rôle est d'authentifier l'utilisateur afin de passer ses droits de visiteur à utilisateur, lui accordant alors le droit de créer et de publier de articles. Cette barre renvoie également vers une page d'inscription, si le visiteur ne dispose pas de compte valide.

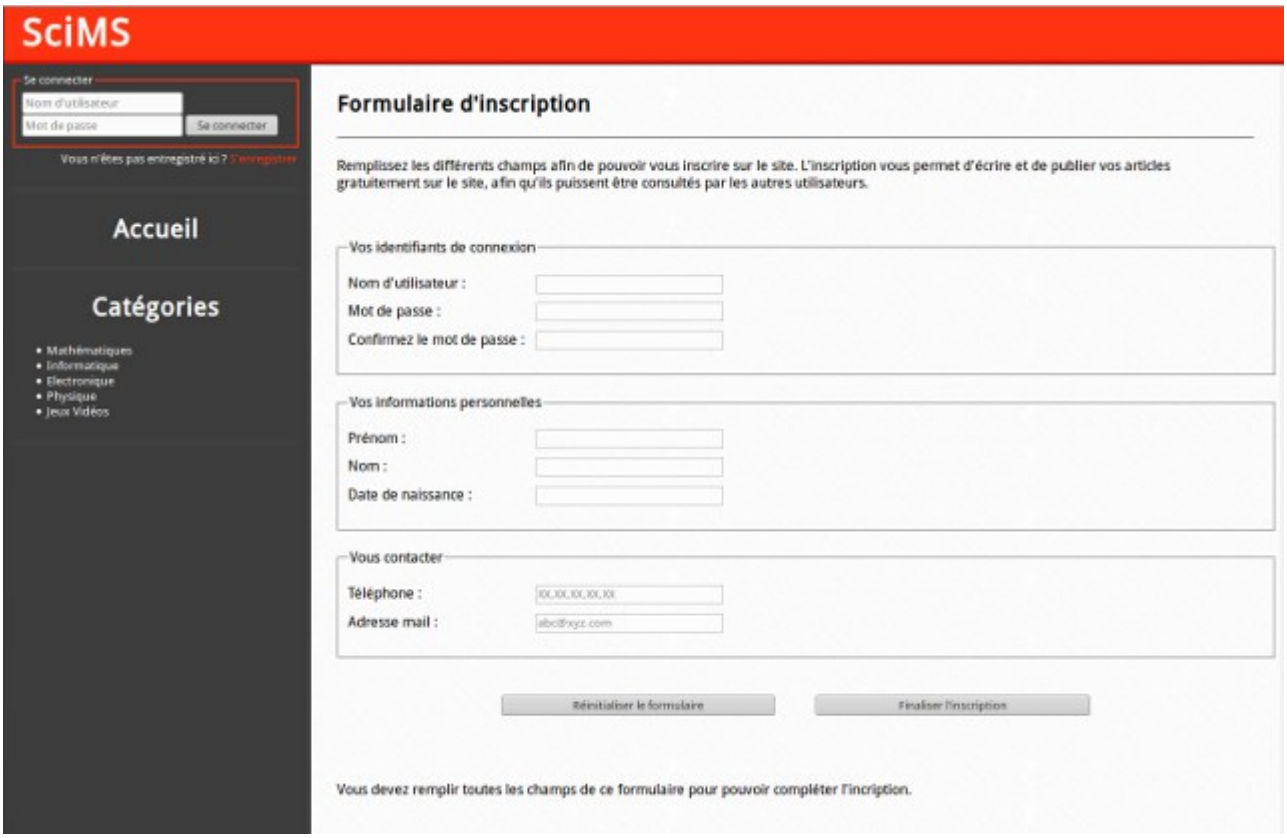
Fonctionnement :
Son mécanisme repose sur un simple formulaire qui va charger un profil d'utilisateur avec les données entrées, les attributs du profil chargé seront alors stockés dans la variable de session associée.

La seconde partie correspond aux liens intermédiaires situé entre le bloc des catégories, et l'encart de connexion. Elle sert à rediriger l'utilisateur vers l'accueil, l'éditeur ou l'interface d'administration, selon les droits dont il dispose.

La dernière partie correspond à la liste des catégories d'article consultables. Elle permet de renvoyer un lien direct vers la catégorie choisie afin d'y consulter

Fonctionnement :
Le menu va charger l'ensemble des catégories racines, (i.e les catégories qui n'ont pas de catégorie « au-dessus d'eux »). pour chaque catégorie, le site va la mettre en valeur en lui associant un lien dans la barre de menu. Ce lien renvoie sur la page de visualisation de contenu (cf 2.3.4), avec un numéro d'identification correspondant à la catégorie choisie.

2.3.2 Page d'inscription (PHP/CSS/JS)



Cette page est constituée d'un unique formulaire, permettant d'enregistrer un nouveau collaborateur sur le site. Son fonctionnement se limite à envoyer les données remplies et à authentifier l'utilisateur avec son nouveau compte, après inscription.

Si JavaScript est activé, un script de contrôle des données entrées s'activera, afin de s'assurer de la conformité des informations avant interprétation par le serveur. Dans le cas, où l'utilisateur se trompe dans le format d'une saisie, ce dernier est averti par l'affichage d'un message d'erreur.
Si l'information est correcte, le texte sera mis en valeur avec une couleur verte.

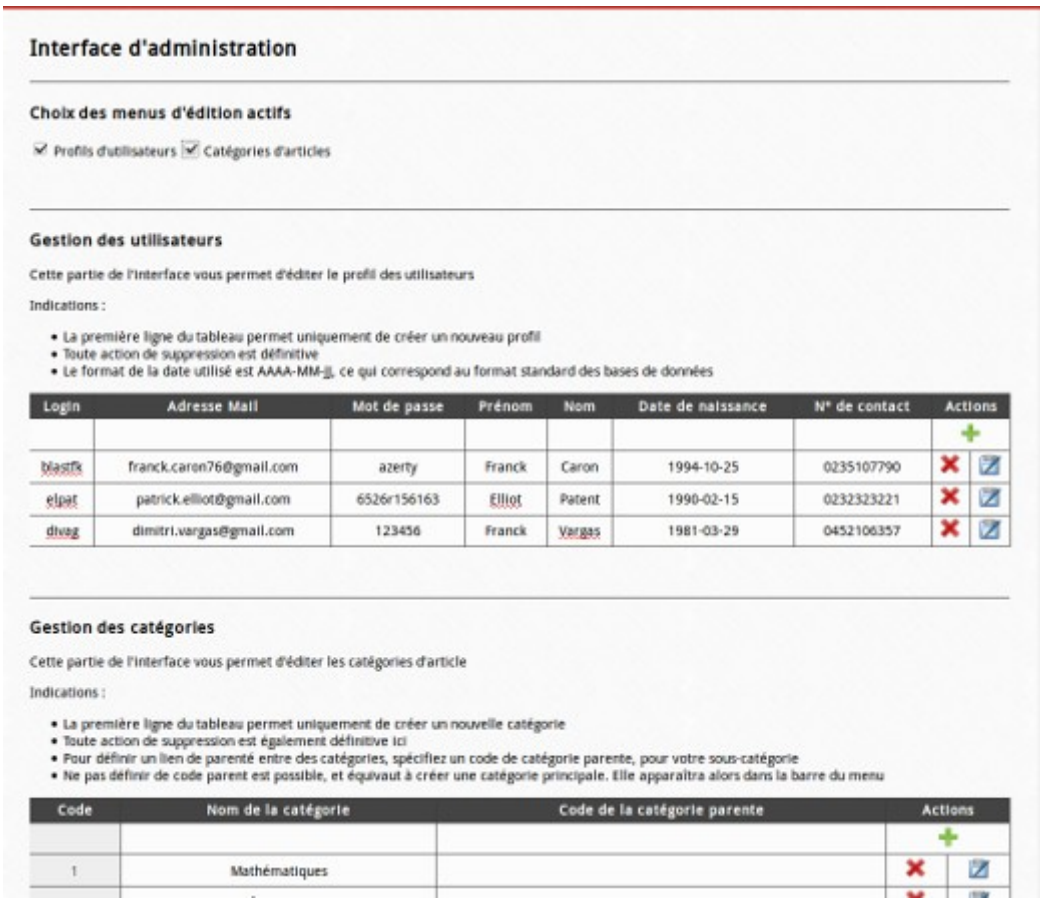
Sous-composants associés :

- php/controller/register_controller.php : Le formulaire d'inscription
- php/model/register_model.php : Le modèle associé au formulaire, afin de procéder à l'inscription de l'utilisateur auprès de la base de données
- js/register.js : Le script associé au dynamisme du formulaire.

2.3.3 Interface d'administration (PHP/JS)
Attention : Cette page requiert l'utilisation de JavaScript

Cette page est constituée de deux tableaux d'administration. Le premier permet d'éditer les différents profils d'utilisateurs, tandis que le deuxième permet d'éditer les catégories. Leur affichage simultané ou non, se gère au moyen d'une petite barre de choix, activé par JavaScript.

L'interface d'administration se présente sous la:



Le mécanisme d'édition par tableau requiert l'emploi d'un seul type de contrôleur (defini dans php/controller/admin_controller.php), réutilisable qui se définit au moyen de la variable de session $_SESSION['admin_edit'] :

- $_SESSION['admin_edit']['keys'] : L'ensemble des attributs de l'objet à afficher dans la table.
- $_SESSION['admin_edit']['uneditableKeys'] : Les attributs non éditables de l'objet.
- $_SESSION['admin_edit']['tableName'] : Le nom de la table du type d'objet.
- $_SESSION['admin_edit']['desc'] : Un tableau associatif permettant d'associer une description à un nom d'attribut.

Ce fichier va alors définir un tableau qui sera rempli du contenu présent dans la table spécifiée par $_SESSION['admin_edit']['tableName'].

La dernière colonne de chaque tableau, représente les actions possibles sur l'objet de la même ligne. Elles correspondent à des actions de suppression/ajout/modification, comme exigé par les contraintes du sujet. Ces dernières sont rendues actives au moyen du javascript.
En effet, le script admin_controller.js associe à chaque action, une requête GET qui permettra au tableau d'envoyer la ligne correspondante en tant que couples (clé, valeur) dans la requête.

Chaque tableau est associé à un modèle de traitement spécifique (php/model/admin_user_controller.php pour les utilisateurs et php/model/admin_category_controller.php pour les catégories) qui va s'assurer de faire correspondre l'état du tableau avec celle de la table de données, en modifiant la ligne impliquée.

2.3.4 Consultation des articles/catégories (PHP/CSS)

La page php/view/viewer.php permet d'afficher le contenu d'une catégorie ou d'un article donné. Il n'a pas été fait distinction de la consultation entre les deux types afin de simuler le comportement d'URL arborescente, et de mettre en valeur le chemin complet de la racine, jusqu'à l'article ou la catégorie concernée.

Afin de différencier le comportement en fonction de ces 2 cas d'utilisation, l'accès à la page peut se faire à l'aide d'une requête GET pour spécifier le numéro de catégorie, ou d'article que l'on souhaite utiliser :

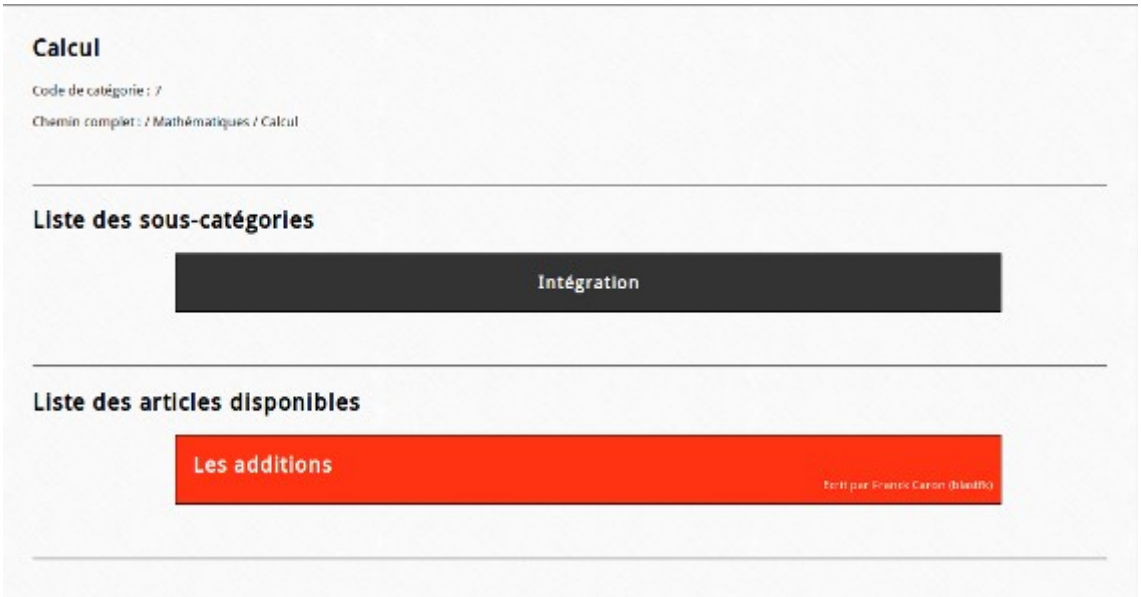
- Si la variable $_GET['idart'] est définie (ID d'un article) : On affiche l'article correspondant.
- Si la variable $_GET['idcat'] est définie (ID d'une catégorie) : On affiche la catégorie correspondante.
- Si aucune des deux n'est définie : On affiche le contenu de la catégorie racine.

De ce fait, la page fait appel à 2 composants distincts :

Consultation d'une catégorie :

La page appelle le contrôleur php/controller/viewer_controller.php, qui s'occupera de charger les informations concernant les catégories enfants, et les articles disponibles appartenant à cette catégorie, s'ils existent.

Le rendu de la page sera alors :



Les catégories sont affichées avec leur chemin complet (où chaque nom de catégorie est cliquable pour y accéder plus facilement), et leur code de catégorie, ce qui permet de les prendre en référence pour les futures recherches de l'utilisateur.

Consultation d'un article :

La page appelle le composant de vue php/view/components/viewer_article.php, qui s'occupera d'afficher le contenu d'un article, en affichant toutes ses informations (nom de l'auteur, chemin complet, date de publications, abstract, ...).

Hormis le chargement des données, le mécanisme notable pour cette partie est la réalisation du bloc sommaire, qui est rendu fonctionnel par l'emploi de code CSS, (css/components/summary.css) au moyen d'un système de compteurs spécifique.

2.3.5 Interface d'édition (PHP/JS/CSS)
Attention : Cette page requiert l'utilisation de JavaScript

La page php/view/editor.php permet de créer de nouveaux articles au moyen d'une interface de création. Cette interface est rendue vivante grâce à du code JavaScript qui possède une structure particulière :

Pour pouvoir représenter le contenu d'un article (sections, sous-sections, paragraphes, ...), on va construire un élément HTML (au moyen de l'API DOM) qui représentera le contenu de l'article. Cet élément contiendra lui-même d'autres blocs qui lui serviront de contenu. En structurant la manière dont l'on construit cette élément, on peut ainsi structurer l'édition du contenu de l'article.

Représentation d'un bloc :

Un bloc (Objet JS) contient un élément (balise HTML), lorsqu'on imbrique un bloc dans un autre, on ajoute le contenu du bloc contenu dans celui du bloc conteneur. Cela revient à représenter sous forme d'objet JS, l'imbrication naturelle des balise HTML.

Ainsi, on définit des blocs conteneurs, et des blocs objets, qui eux, ne servent qu'à fournir du contenu et non contenir d'autre blocs.

On peut alors définir 4 types de blocs :

- Le conteneur universel : Peut contenir aussi bien des conteneurs que des objets. Cela correspond à la définition d'une section d'article (contient des sous-sections, mais aussi des objets).
- Le super-conteneur : Il ne peut contenir que des conteneurs. Cela correspond à un article, qui ne peut qu'être découpé par des sections.
- Le conteneur d'objets : Il ne peut contenir que des objets. Cela correspond à une sous-section.
- Le bloc d'objets : Ne sert à définir du contenu. Correspond à tous les types de blocs éditables à implémenter (paragraphes, images, figures SVG...).

Ainsi, le fichier js/struct.js, se charge de définir de telles sortes de structures. On définira également un 5^{e} type, le type bloc sommaire, qui possède un traitement particulier.

Construction de l'éditeur :

L'éditeur repose sur un formulaire géant, dont certaines entrées sont des balises input, de type hidden. Leur contenu ne sera défini que via le javaScript. Chaque entrée correspond alors à un attribut de l'objet PHP Article.

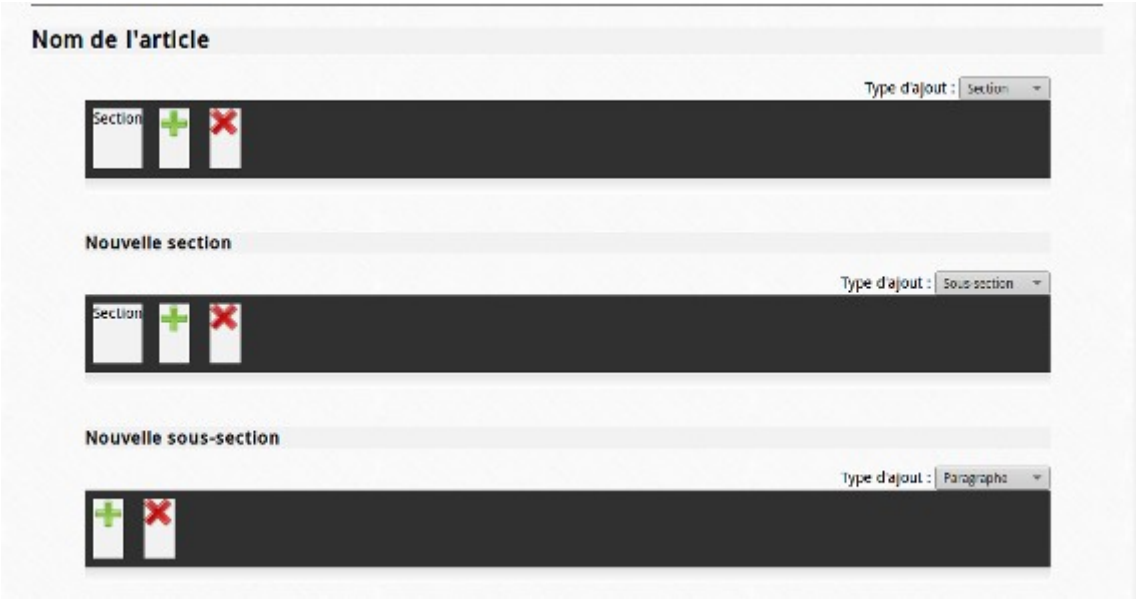
La première partie de l'éditeur permet de remplir les informations relatives à l'article. Il s'agit de champs de saisie simples qui permettent d'éditer le contenu de ces informations.

La deuxième partie de l'éditeur représente les barres de contenus. Une barre de contenu représente l'ensemble des blocs (Objets JS) contenu dans un certain conteneur.
Chaque barre de contenu permet d'éditer les éléments qui y sont placés. Si le bloc à éditer est un conteneur, la barre originelle ouvrira une nouvelle barre de contenu pour éditer l'élément.

Ainsi, trois barres sont utilisées : une pour le contenu de l'article, une autre pour le contenu d'une section, et la dernière pour le contenu d'une sous-section. Le mécanisme de sélection de l'élément en cours d'édition s'effectue au moyen de variables globales, définies dans le fichier struct.js.

Afin de représenter une barre de contenu, on représente chaque bloc par un élément HTML que l'on vient ajouter à la barre. Cet élément est écouté par des listeners qui possèdent des actions précises lorsqu'il sont déclenchés. Ces actions (supprimer un bloc, ouvrir un éditeur particulier, etc.) permettent de dynamiser le contenu de l'interface et de modifier le document en temps réel.
Chaque action se termine par un rafraîchissement de l'interface qui donne lieu à une actualisation des barres de contenu, et à la mise-à-jour du bloc de sommaire en cas de modifications sur les sections.

Rendu des barres de contenu :



Chaque barre dispose de 2 actions permettant de rajouter ou de supprimer des blocs.

Si le bloc à éditer est un bloc objet, la troisième partie de l'éditeur apparaît. Il s'agit du bloc d'édition, permettant de modifier le contenu HTML du bloc spécifié. Selon les cas, il s'agira d'un éditeur de texte ou d'un gestionnaire d'upload, afin de pouvoir gérer les différents types d'éléments souhaités : svg, images, formules et paragraphes.

Enfin, la quatrième partie de l'éditeur consiste en une fenêtre de prévisualisation du contenu édité, son contenu est actualisé en même temps que le rafraîchissement de l'interface.

Le mécanisme de toutes ces fonctionnalités est situé dans le fichier js/editor .js

Lors de la publication, un écouteur est placé sur le bloc formulaire. Ainsi, le script peut charger le contenu édité et les informations de l'article dans les champs de formulaire appropriés.
L'article peut alors être ajouté au moyen du modèle de traitement php/model/editor_model.php.

3. Manuel d'installation

Ces instructions vous permettront pas à pas, d'installer le site en quelques secondes, depuis l'interface de configuration.

Etape 1 :
Placez le dossier du projet dans un répertoire exploitable (localhost) de votre serveur PHP.

(SI vous émulez votre serveur à l'aide de XAMPP, le répertoire localhost correspond au répertoire système /opt/lampp/htdocs/)

Etape 2 :
Ouvrez votre navigateur et entrez dans la barre d'addresse : http://localhost/SciMS/
Vous aurez accès au dossier du projet et pourrez ouvrir la première page du site depuis cette fenêtre.

Etape 3 :
Depuis la fenêtre de navigation, cliquez sur la page config.php.

Etape 4 :
Dans la partie configuration du site, renseignez les champs indiqués :

- DSN : correspond au Data Source Name, i.e une chaine permettant d'atteindre le SGBD distant (dépend du type de base de données que vous employez)
- Compte : Le nom d'utilisateur utilisé pour contacter la base de données
- Mot de passe : Le mot de passe associé à ce compte

Etape 5 :
Cliquer sur configurer. Votre installation se sera alors déroulée avec succès !
Le fichier dbconfig.ini sera alors crée (ou redéfini). Ne le supprimez surtout pas.

Etape 6 :
Ouvrez la page home.php depuis le sous-dossier php/view/.

Etape 7 (Désinstallation) :
Depuis la page config.php, cliquez sur supprimer la configuration.