

Webdoc

Validation d’acquis

Stephane Piriou

| Développement Web et Projet d’Intégration de Développement |

Septembre 2019

Code Source :

<https://github.com/stephanepiriou/webdoc>

Code Source du projet Java :

<https://github.com/ScanDocOrg/ScanDocRepo>

# Contexte

Le projet est une version web de Scandoc, un logiciel Java de classification de documents scolaires centré autour de la gestion d’une base de données.

Le client, le directeur d’une école, souhaitait pouvoir gérer les documents administratifs concernant tant les élèves que les professeurs de son école.

Ces documents devaient pouvoir être classifiés et retrouvés selon un workflow qui se voulait fluide et simple et centralisés sur une station administrée comme un Terminal Server sur Windows Server 2008 auquel était raccordé des terminaux sur lequel travaillait le personnel administratif, si possible sans session utilisateur, celles-ci étant déjà assurée par Windows.

Le système de classification se voulait très simple.

Bien que j’aurait dû travailler en groupe sur un autre projet, je m’était vu attribuer ce projet à la suite d’un désaccord avec mes condisciples. C’est donc seul que j’ai travaillé sur le projet en question.

J’avais choisi pour bibliothèque d’interface graphique SWT, du fait de sa rapidité d’exécution, n’ayant pas conscience au moment de mon choix de son caractère désuet et du peu de ressources documentaire en ligne entourant le framework. Il se trouve que le modèle de programmation requis par le framework était aussi très éloigné du modèle MVC actuel qui a cours maintenant dans la programmation web et desktop.

# Intentions

Pour la présente version web, une approche totalement différente a été prise. La programmation par classe, notamment pour chaque composant du modèle M(Model) V(View)C(Controller) a été adoptée, résultant une architecture très différente.

N’ayant jamais abordé ce modèle d’architecture auparavant, je suis partit d’un cours vidéo Udemy sur Internet sur la création d’un framework MVC en PHP.

Ce « framework », constitué de 5 fichiers (classes), constitue le boilerplate nécessaire à une structure d’application en MVC.

J’ai ensuite continué à construire l’entièreté de l’application sur les bases constituées par le « framework » construit lors du cours.

Le but était de se familiariser avec une méthode de programmation maintenant largement adoptée en PHP, notamment avec des frameworks tels que Symfony, Laravel ou CodeIgnitor.

# Intervenants

Je me réfèrerais au projet java, ayant été en roue libre pour son adaptation en PHP.

## Le client

Le directeur d’école Secondaire provinciale. Il n’est pas l’utilisateur finale, qui sont en réalité les membres du secrétariat de l’école. Toutefois, c’est lui qui fait la commande du projet.

Malgré mes questions répétées, ses exigences étaient peu précises, probablement du fait qu’il n’avait qu’une idée approximative du workflow auquel les utilisateurs finaux étaient soumis.

## Utilisateurs FinAux

Je n’ai malheureusement pas eu l’occasion de les rencontrer dans le cadre du projet Java. Ce qui est une faute. Dans l’idéal, j’aurais dû les rencontrer au minimum lors de la créations de l’interface, lors d’une première implémentation, au cours d’une phase de test, et lors de la livraison finale.

A default, j’ai toutefois eu l’occasion de revoir le directeur de l’école, malheureusement le projet était déjà trop avancé et il ne semblait pas satisfait de l’interface malgré que j’avais suivi ses consignes.

Au final, je n’ai jamais eu l’occasion de les rencontrer.

J’ai considéré qu’il y avait 3 types d’utilisateur (indépendamment de l’utilisateur non-enregistré dans la base de données) :

* Le simple utilisateur, qui ne peut consulter que les données relatives aux individus encodés, télécharger les documents ainsi que les imprimer.
* L’encodeur, dont le rôle consiste à encoder les données relatives à l’application. Il est aussi amené à les consulter.
* L’administrateur, dont l’unique fonction est d’administrer les utilisateurs de l’application et les enregistrant d base de données, les modifiant, les effaçant de cette dernière.

## Testeurs

Ca aurait dû être les utilisateurs finaux. A default, Fabian du secrétariat de l’ECI à jouer ce rôle avant la remise finale du projet. Ca été l’occasion de plusieurs aller-retour au secrétariat et de la correction de certains bugs.

## Développeur

Un unique développeur, moi-même, était en charge de l’analyse et de l’implémentation du projet.

## Chef de Projet

J’ai porté la casquette de Chef de Projet.

# DOSSIER TECHNIQUE

## Languages UTILISéS

L’application en client WEB a été développée en :

* PHP 7.2
* HTML 5
* Javascript

## Framework Utilisés

* jQuery 2.2.4
* jqWidgets 8.3.2
* jQuery Validation Plugin 1.19.1

## OS de developpement

* MacOS Mojave 10.14
* MacOS Catalina 10.15

## MAChine de developpement

* Macbook 12’ 2016
* Macbook Pro 13’ 2015

## Suite Logiciel DE DEVELOPPEMENT

Analyse DB :

* Visual Paradigm 14.2

Analyse UML :

* Booml 7.9 (version Windows)

Stack de développement :

* PHP 7.2.23
* PhpMyAdmin 4.9.1
* Apache 2.4.41
* MariaDB 10.4.8

IDE :

* PhpStorm 2019 rolling, licence étudiant.

## Platform de PRODUCTION

* Serveur VPS loué chez OVH

OS DE PRODUCTION

* Ubuntu Server 18.04.02

LOGICIELS DE PRODUCTION

* PHP 7.2.1
* MariaDB 10.4
* Apache 2.4.34
* PhpMaAdmin 4.9.1

DOCUMENTATION

* Phpdoc (phpdocumentor), version 2.
* Word 2019 (version MacOS).

## contrôle de version

* Client git MacOS : Tower 2.4 sur MacOS
* Client git Ubuntu Server : Gitlazy 0.8.1 sur Ubuntu Serveur
* Serveur git assuré par [www.github.com](http://www.github.com) (voir page de garde du rapport).

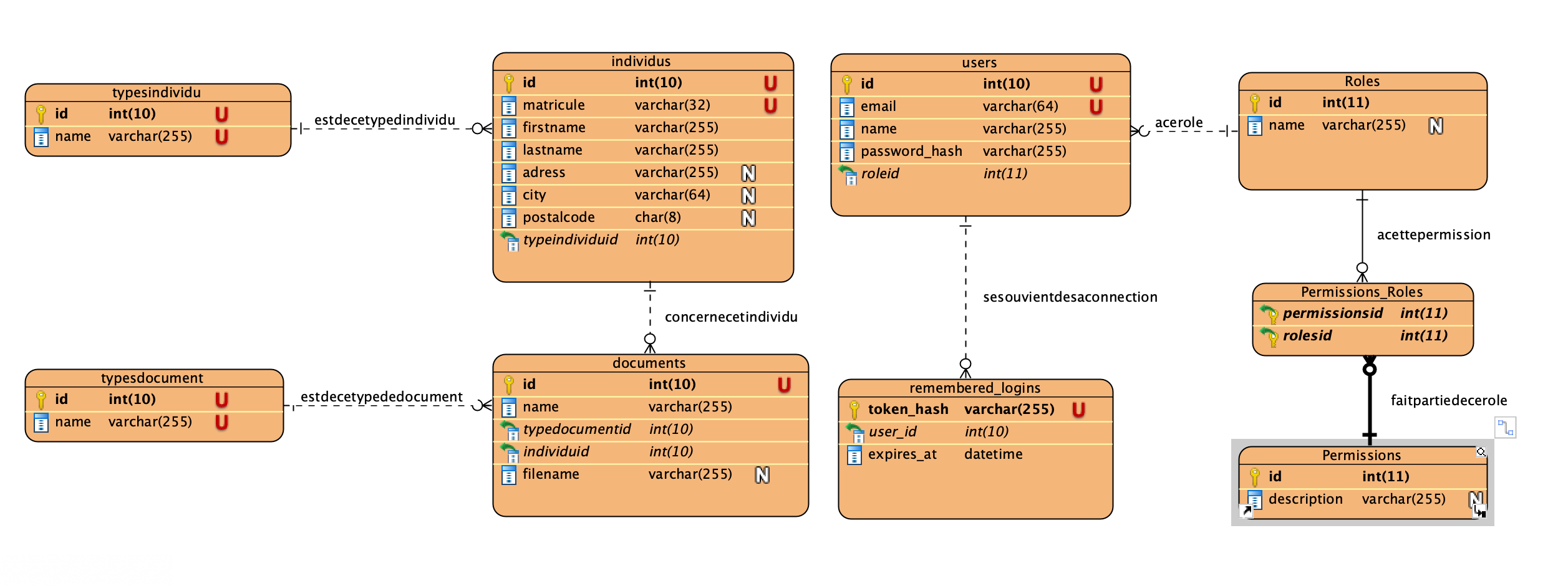
# Fonctionnalités et caractéristiques de l’application

* Login/Logout.
* Rétention d’un utilisateur précédemment logger grâce à l’utilisation de cookies sécurisé dans la base de données.
* Protection des pages du site derrière un formulaire de login.
* Gestion des erreurs.
* Logging des erreurs dans un fichier log.
* Version « développement » et « production » pour la gestion et le log des erreurs.
* Création automatique du premier utilisateur lors de la première visite du site.
* Page d’erreur customisée (erreur 404 & erreur 500).
* Structure selon modèle MVC.
* Gestion des utilisateurs de l’application.
* Technologie jQuery & AJAX pour la vérification des entrées.
* Interface d’application desktop, reproduisant le layout de l’application Java « Scandoc ».
* Utilisation de la bibliothèque de composants « jQwidgets ».
* Utilisation du framework CSS « Bootstrap 4 ».
* Contrôle des entrées utilisateur coté client (jQuery & Ajax) et serveur.
* Réécriture des url et routage vers méthode de contrôleur par adressage.
* Résistance aux injections SQL.
* Mot de passe stocké de manière sécurisée sous forme de hash calculé à partir d’une clé unique stockée dans un fichier de configuration.
* Stockage de cookie de connexion intersession sécurisé.
* Système de contrôle d’accès aux fonctionnalités par attribution de rôles et de permissions.

# Analyse

## Analyse de base de données

J’ai utilisé le système de notation international d’analyse de base de données au lieu du modèle Mérise (je me suis plié aux possibilités offertes par la plateforme MacOS en matière de logiciel d’analyse de base de données).



TypesIndividu (pk : id : int)

Le type d’individu sauvegardé dans la table Individus, par exemple « élève », « professeur », « éducateurs ».

name ( varchar(255) : unique)

Le nom du type d’individu.

Par exemple : « élève ».

Individus (pk : id : int)

Représente un individu pour lequel on peut sauver des documents le concernant.

matricule ( varchar(32) : unique)

Le matricule qui identifie l’individu dans le système scolaire provincial.

Par exemple : « MAT153898E »

firstname ( varchar(255) )

Le prénom de l’individu.

Par exemple : « Stéphane »

lastname ( varchar(255) )

Le nom de famille de l’individu.

Par exemple : « Piriou ».

adress ( varchar(255) )

L’adresse où habite de l’individu.

Par exemple : « 42, rue Ansiaux ».

city ( varchar(64) )

La ville ou habite l’individu.

Par exemple : « Liège ».

postalcode ( char(8) )

Le code postal du lieu où habite l’individu.

Par exemple : « 4020 ».

typeindividuid ( fk : int )

La référence numérique du type d’individu auquel l’individu appartient.

Roles (pk : id : int)

Prédéterminée au moment de la création de la base de données. Il y a 3 roles distinct pour la présente application : « utilisateur », qui permet la consultation des données et le téléchargement et l’impression des documents ; « encodeur », qui permet la consultation des données ainsi que leur modification et leur effacement ainsi le téléchargement, l’impression et l’effacement des documents ; et « administrateur », qui n’autorise uniquement que l’enregistrement, la modification et l’effacement d’utilisateur de l’application. En pratique les rôles servent uniquement à déterminer quel menu doit être affiché à l’utilisateur.

name ( varchar(255) )

Valeur possible : “administrateur”, “encodeur”, “utilisateur”.

Description ( varchar(255) )

Description des rôles existants dans l’application. Les valeurs possibles sont :

-« Un utilisateur peut consulter les données ainsi qu'afficher, imprimer et téléchargér les document » pour le rôle « utilisateur » ;

-« Un encodeur peut consulter les données, les modifier, les effacer, afficher les documents, les effacer. Il ne peut pas télécharger ces documents, ni les imprimer. » pour le rôle « encodeur » ;

-« Un administrateur s'occupe uniquement de la gestion des utilisateurs. Il ne peut ni consulter les données, ni les effacer. » pour le rôle « administrateur ».

Permissions (pk : id : int)

Prédéterminés au moment de la création de la base de données. Les permissions déterminent les accès autorisées aux méthodes des contrôleurs et à activer ou désactiver certaines actions au niveau des vews. Ils sont attribués diversement rôles au travers d’une table intermédiaire (roles\_permissions).

Il y a 4 types de permissions :

* « consultation » : qui autorise l’accès aux méthodes de contrôleurs responsable des fonctionnalités de consultation des données.
* « creation » : qui autorise l’accès aux méthodes de contrôleurs responsable des fonctionnalités de création des données.
* « modification » : qui autorise l’accès aux méthodes de contrôleurs responsable des fonctionnalités de modification des données.
* « user\_administration » : qui autorise l’accès aux méthodes de contrôleurs responsable de la création, modification, effacement d’utilisateur de l’application

name ( varchar(255) )

L’intitulé de la permission.

Rôles\_Permissions

Prédéterminé au moment de la création de la base de données. Table intermédiaire attribuant des permissions à des rôles.

Les associations sont celles-ci :

« utilisateur » : « consultation ».

« encodeur » : « consultation », « creation », « modification ».

« administrateur » : « user\_administration ».

roleid (fk : int)

Clé étrangère de l’id du rôle auquel l’enregistrement fait référence.

permissionid (fk : int)

Clé étrangère de l’id de la permission auquel l’enregistrement fait référence.

TypesDocument (pk : id : int)

Le type de document sauvegardé dans la table TypesDocument, par exemple « carte identité verso », « diplôme CESS », « certificat maladie ».

name ( varchar(255) : unique)

Le nom du type de document.

Par exemple : « CESS ».

Document (pk : id : int)

Le document sauvegardé dans la table Document, par exemple « carte identité verso », « diplôme CESS », « certificat maladie ».

name ( varchar(255) : unique )

Champ calculé. Combinaison de TypesDocument.Nom & Individus.firstname & Individu.lastname

Par exemple : « CESS de Stéphane Piriou».

filename ( varchar(255) : unique )

Champ calculé. Combinaison du hash de Documents.name + extension de fichier

Par exemple : « Uh3789jK0pPLmmU7z8iop6zZIGPFAN8PN3opMghY4.jpg».

individuid ( fk : int )

Clé étrangère. Référence numérique d’un individu.

typedocumentid ( fk : int)

Clé étrangère. Référence numérique à un type de document.

Users (pk : id : int)

Utilisateur de l’application enregistré sur le site, condition nécessaire pour la consultation, la création et la modification des données.

email ( varchar(64) – unique )

L’email de l’utilisateur, qui l’identifie ce dernier sur le système.

Par exemple : « stephane.piriou@zaclys.net »

nom ( varchar(255) )

Nom de l’utilisateur.

Par exemple : « stephanep ».

password\_hash ( varchar(255) )

Mot de passe de l’utilisateur stocké sous forme de hash calculé à partir d’un seed aléatoire stocké dans le fichier de configuration de l’application.

roleid (fk : int)

Clé étrangère représentant le rôle auquel appartient l’utilisateur.

remembered\_login ( pk : token\_hash : varchar(255) )

Stocke les cookies authentifiant l’utilisateur entre sessions.

token\_hash ( varchar(255) )

Token calculé destiné à sécurisé le cookie dans la base de donnée et d’éviter le vol de session en cas de compromission de la base de données.

user\_id ( fk : int )

Clé étrangère. Référence l’utilisateur à qui appartient le cookie.

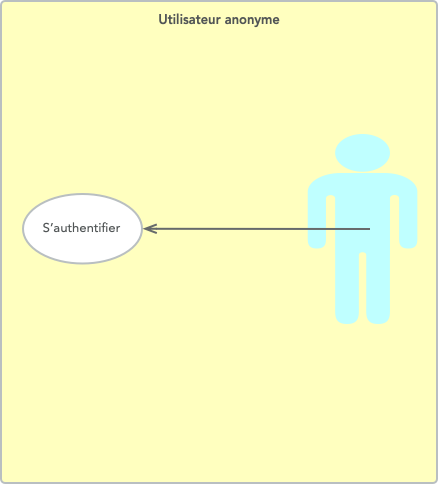
expire\_at ( datetime )

Date d’expiration du cookie.

## Analyse UML

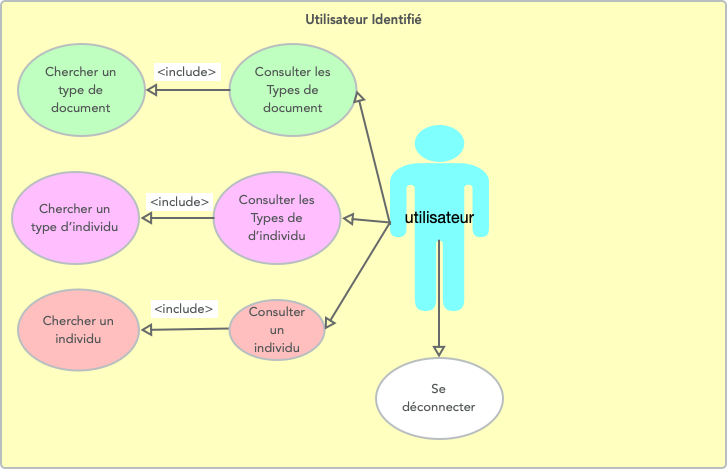
### Cas d’utilisation par type d’utilisateur

Utilisateur anonyme

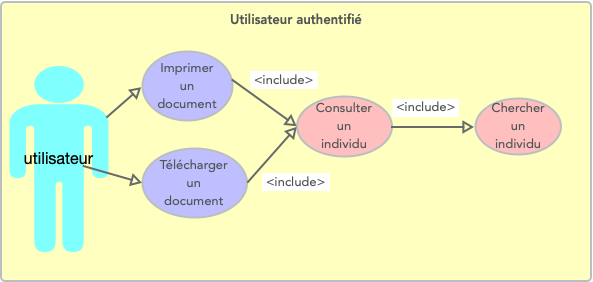


1. L’utilisateur anonyme souhaite s’authentifier (pour accéder aux fonctionnalité de l’application)

Utilisateur

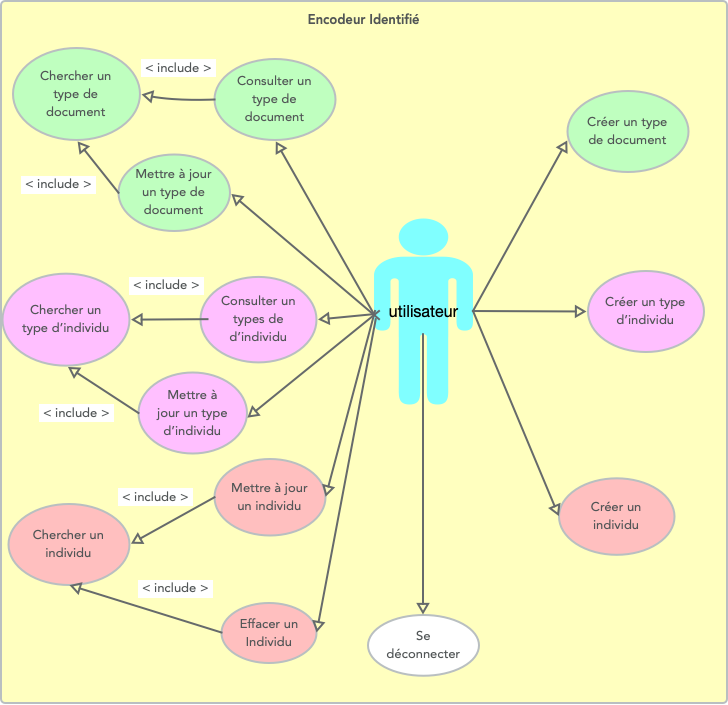


1. L’utilisateur (authentifié) souhaitant consulter un type de document implique qu’il cherche après un type de document grâce à un moteur de recherche.
2. L’utilisateur (authentifié) souhaitant consulter un type d’individu implique qu’il cherche après un type d’individu grâce à un moteur de recherche.
3. L’utilisateur (authentifié) souhaitant consulter un individu implique qu’il cherche après un individu grâce à un moteur de recherche.
4. L’utilisateur se déconnecte.

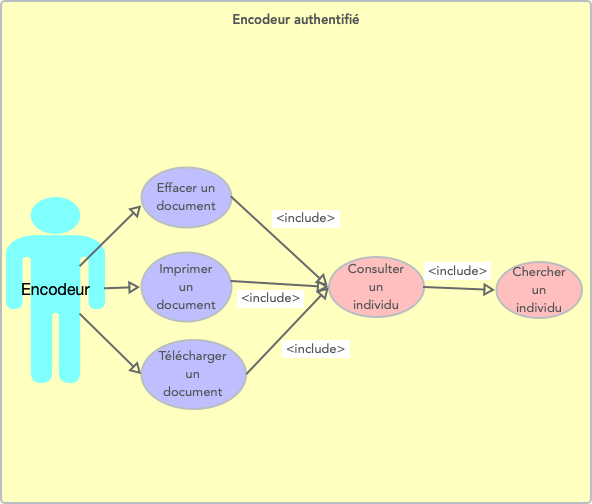


1. L’utilisateur (authentifié) souhaitant imprimer un document implique qu’il consulte un individu qui implique qu’il cherche un individu grâce au moteur de recherche.
2. L’utilisateur (authentifié) souhaitant télécharger un document implique qu’il consulte un individu qui implique qu’il cherche un individu grâce au moteur de recherche.

Encodeur

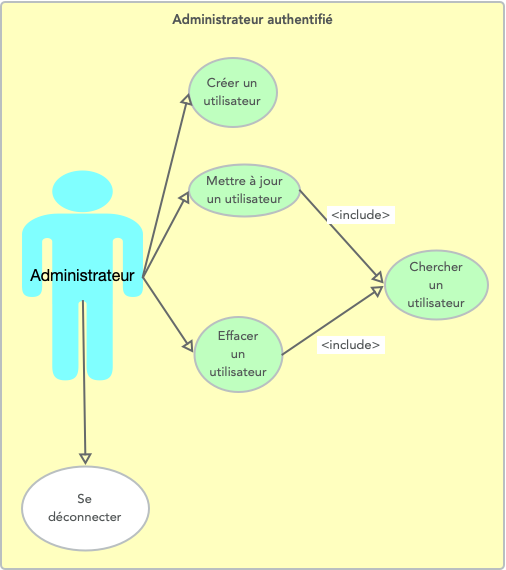


1. Encodeur crée un type de document.
2. Encodeur crée un type d’individu.
3. Encodeur crée un type un individu.
4. Encodeur souhaitant consulter un type de document implique qu’il cherche après un type de document grâce au moteur de recherche.
5. Encodeur souhaitant mettre à jour un type de document implique qu’il cherche après un type de document grâce au moteur de recherche.
6. Encodeur souhaitant consulter un type d’individu implique qu’il cherche après un type d’individu grâce au moteur de recherche.
7. Encodeur souhaitant mettre à jour un type d’individu implique qu’il cherche après un type d’individu grâce au moteur de recherche.
8. Encodeur souhaitant consulter un individu implique qu’il cherche après un individu grâce au moteur de recherche.
9. Encodeur souhaitant mettre à jour un individu implique qu’il cherche après un individu grâce au moteur de recherche.
10. Encodeur souhaitant effacer un individu implique qu’il cherche après un individu grâce au moteur de recherche.



1. Encodeur souhaitant imprimer un document implique qu’il consulte un individu qui implique qu’il cherche un individu grâce au moteur de recherche.
2. Encodeur souhaitant télécharger un document implique qu’il consulte un individu qui implique qu’il cherche un individu grâce au moteur de recherche.
3. Encodeur souhaitant effacer un document implique qu’il consulte un individu qui implique qu’il cherche un individu grâce au moteur de recherche.
4. L’encodeur se déconnecte.

Administrateur



1. L’administrateur crée un utilisateur.
2. L’administrateur qui souhaite modifier un utilisateur implique qu’il cherche un utilisateur.
3. L’administrateur qui souhaite effacer un utilisateur implique qu’il cherche un utilisateur.
4. L’administrateur se déconnecte.

## Diagramme de classe

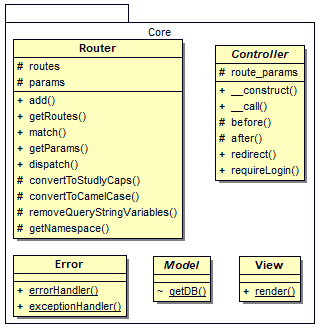
L’application est constituée de 3 strates :

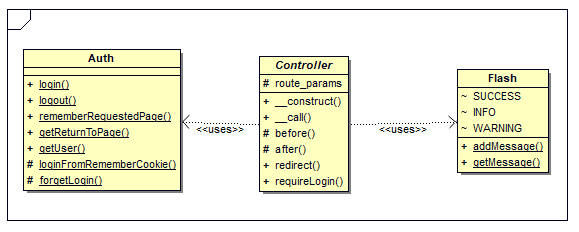
* « Core », namespace contenant les classes posant les bases d’une structure d’application selon le modèle MVC et les règles de routage ainsi que la gestion de erreurs ;
* « App », l’application proprement dite subdivisée selon la structure MVC en 3 namespaces :
  + « Controllers », contenant les classes faisant le lien entre le modèle et les vues en transmettant les données à ces dernières et contenant la logique de l’application;
  + « Model », contenant les classes représentant les données ou responsables de leur collecte, contrôlant la lecture et l’écriture dans la base de données et représentant celle-ci à travers des classes structurées selon les entités (tables) de la base de données ; et
  + « Views », contenant les vues responsables du rendu graphique de l’application et des données.
* « public », le dossier contenant les ressources nécessaires au rendu du frontend, notamment les fichiers de styles (framework CSS Bootstrap et styles personnalisés) et les fichiers de script Javascript (framework jQuery et bibliothèque de composants de bureau jqWidgets).
* Core
  + Controller
  + Model
  + View
  + Router
  + Error
* App
  + Controllers
  + Models
  + Views
* public
  + scripts
  + styles

Je ne documente ici que les classes et leur rôle général dans l’application. Pour une information sur les méthodes des classes, je suggère au lecteur d’utiliser la phpdoc fournie avec le dossier.

### Namespace « Core »

« Core » contient les classes qui rendent possible la structure en MVC, le routage et la gestion des erreurs.





Router : Classe responsable du routage et de l’appel des contrôleurs et des méthodes des contrôleurs à partir de l’URL du navigateur. Ainsi, l’URL « http://webdoc.ovh/individus/new » appellera la méthode « newAction() » du contrôleur « Individus ».

Controller : Classe abstraite contenant la base des fonctions assumées par les contrôleurs du namespace « App/Controllers ».

Model : Classe abstraite contenant la base des fonctions assumées par les classes du modèle contenues dans le namespace « App/Models ». (Celle-ci se résume à une fonction statique « getDB() ».

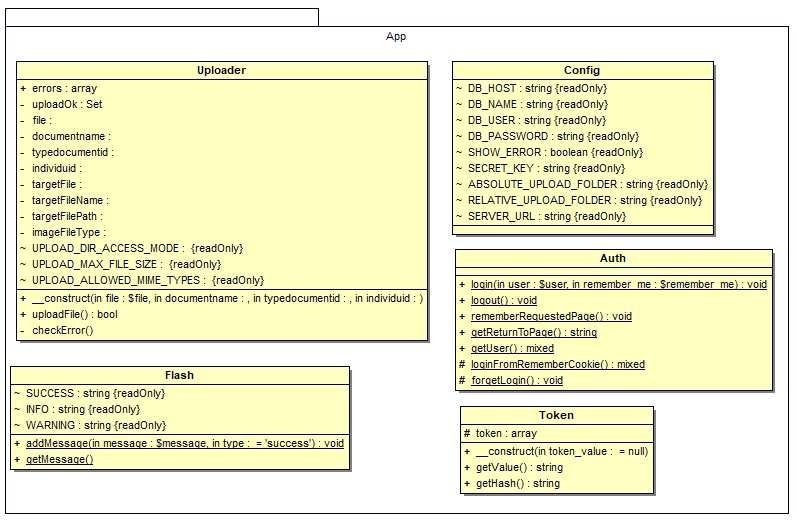
View : Classe contenant la fonction statique responsable de l’appel des fichiers de vue et de la transmission des données nécessaires à leur affichage correcte.

Error : Gestion des erreurs et de logging.

### Namespace « App »

Le namespace « App » contient la structure MVC de l’application. Il se subdivise en sous-namespace « Controllers », « Models », « Views ».

Il contient aussi les classes assistantes utilisées par les contrôleurs pour assurer leurs fonctions.



Note : Une classe, « Downloader » a été ajoutée et n’est pas présente sur le schéma.

Uploader

Classe responsable de l’upload de fichier lors de la création d’un document.

Celui-ci vérifie que certaines conditions sont bien remplies notamment afin d’éviter certains bugs, par exemple l’existence préliminaire du document dans le système de fichiers ou la base de données, l’upload de fichier non autorisés, ou d’une taille supérieure à une limite donnée.

De plus elle sécurise la classe sécurise le document en prévenant l’accès à ce dernier en encryptant le nom du fichier qui sera sauvegardé sur le serveur, rendant impossible le deviner et de télécharger le document hors du cadre du fonctionnement normal de l’application.

Flash

« Flash » sert à transmettre des messages entre les vues concernant les informations de login. Les 2 fonctions qu’elle contient permettent d’initier un message à partir d’un contrôleur et de le récupérer à partir de la vue. Comme Flash se doit d’être disponible avant et après authentification, elle est instanciée au niveau de la classe parent « Controler » du namespace « Core ».

Config

« Config » contient les réglages de l’application utilisés à travers cette dernière par son évocation. Ces derniers définissent les règles de connexion à la base de données, l’affichage des erreurs aux utilisateurs ou au contraire leur insertion dans un fichier log, le seed utilisé pour créer le hash sous la forme duquel les mots de passe sont stockés dans la base de données ainsi que le cryptage des noms de fichier sauvegardé sur le serveur, les références relative et absolue au dossier de stockage des documents sauvegardés et le nom d’hôte de l’application.

Auth

« Auth » se voit confier les diverses tâches liées à l’authentification de l’utilisateur et assiste le contrôleur « Login ». Il gère le cookie lié à l’authentification, la session de l’utilisateur, la déconnection. C’est aussi elle qui est respnsable de la sauvegarde de l’objet de la classe « User » symbolisant l’utilisateur courant.

Token

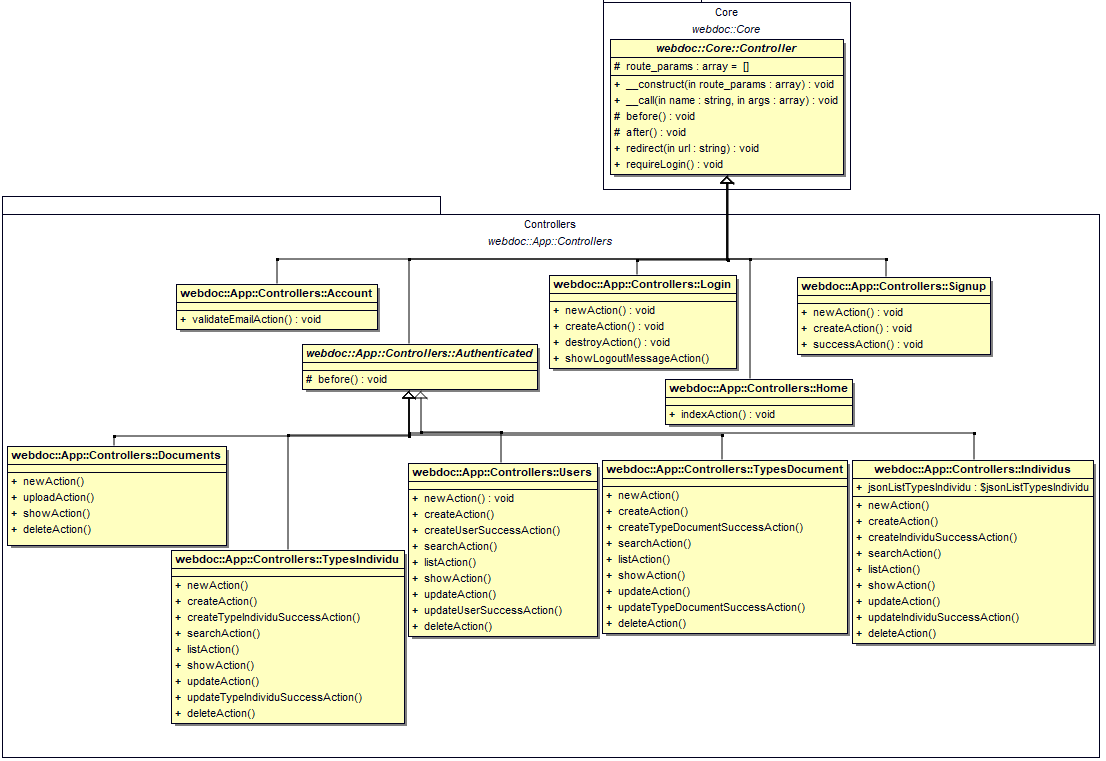
« Token » gère la création et la récupération du token sous lequel est sauvegardé le cookie dans la base de donnée.

Downloader

« Downloader » se voit la tache de gérer le processus de téléchargement de documents.

Afin de sécuriser celui-ci, la classe se charge de renommer le fichier avec l’intitulé du document dans la base de donnée et de le compresser sous un fichier zip ayant pour nom avec un hash du nom de fichier original, lui-même un hash de l’intitulé du document sauvegardé dans la base de données, et supprime enfin le zip une fois l’envoi effectué, rendant impossible son téléchargement hors du fonctionnement normal de l’application.

Namespace « App/Controllers »



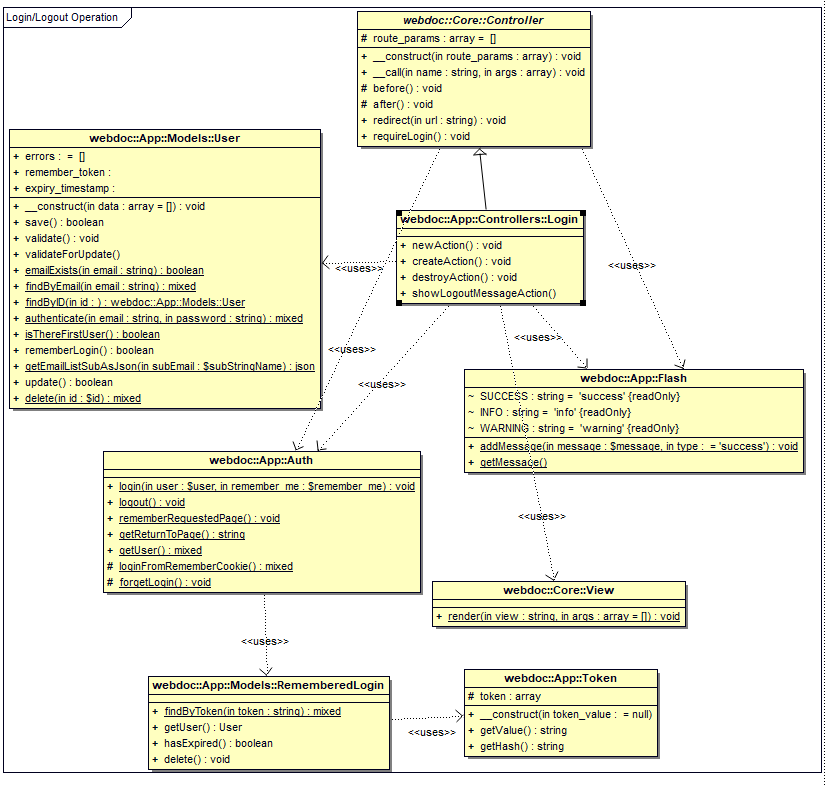
Certains contrôleurs, héritant de la classe « Authenticated », requiert que l’utilisateurs se soit logger à partir d’identifiants personnels afin d’y accéder. Ces contrôleurs sont responsables du contrôle d’accès aux vues permettant la visualisation et la modification des données de l’application.

Les contrôleurs « Account »(doit être déprécié), « Login », « Signup »(déprécié) sont responsables de l’authentification de l’utilisateur.

Le contrôleur « Home » (url : http://webdoc.local) se charge de créer un premier utilisateur au moment de la première accession à l’application. Cet utilisateur provisoire devra être effacé ultérieurement.

Contrôleur « Login » :

Contrôle l’authentification de l’utilisateur. Utilise diverses classes assistantes du namespace « App ».



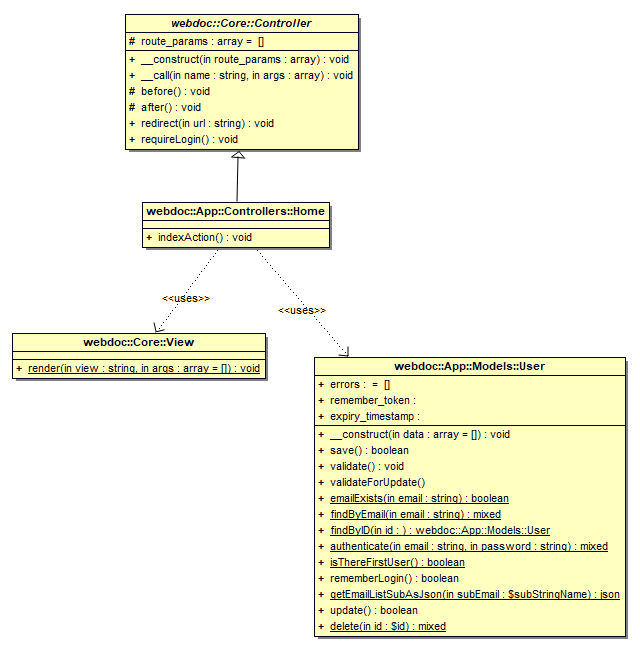
Note : Une classe « Role » a été ajoutée dans un deuxième temps et est instanciée au moment de l’instanciation d’un objet de la Classe « User » par le Contrôleur « Login » et attribuée à une propriété « $role » de la classe « User », créant une relation de type structurel entre la classe « User » et « Role ».

Contrôleur « Account » (doit être déprécier) :

Contient la méthode de réponse à la méthode d’appel AJAX de la vue située /App/Views/Signup/new.php. Doit être transférée dans le contrôleur « Users ».

Contrôleur « Home » :

Contrôleur de la page <http://webdoc.ovh>. Crée automatiquement un administrateur si aucun utilisateur n’existe déjà dans la base de donnée, rendant possible l’accès au contrôler responsable de la gestion des utilisateurs ou sinon, redirige vers <http://webdoc.ovh/login/new>

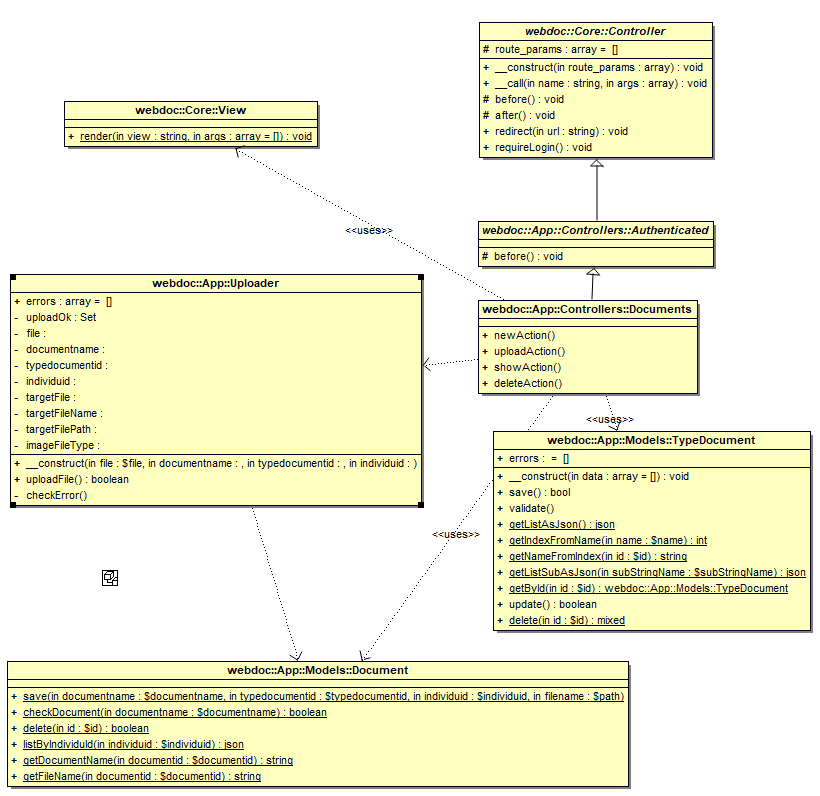


Contrôleur « Authenticated » :

Classe abstraite, héritant de la classe « Controller » et de sa méthode « before() » responsable de vérifier que l’utilisateur est authentifié, et gérant le contrôle d’accès aux contrôleurs et permettant l’accès, la création et la modification des données qui héritent de cette classe, à savoir « Documents », « TypesDocument », « Individus », « TypesIndividu », « Users ».

Contrôleur « Documents » :

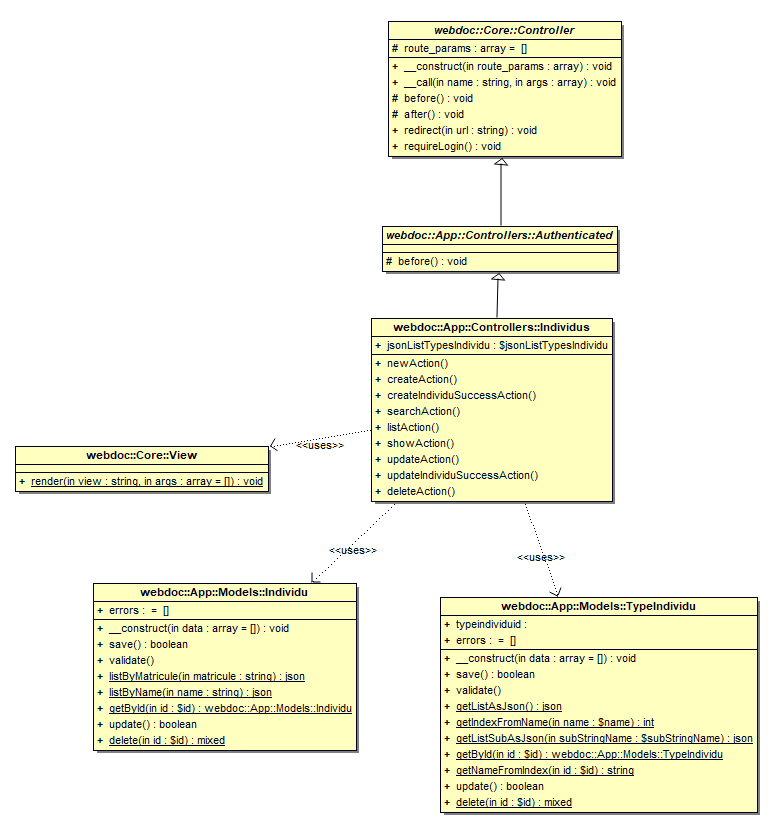
Gère l’accès aux vues du namespace « App/Views » et aux données distribuées par la classe « Document » et « TypeDocument » du namespace « App/Model ». Hérite de « Authenticated ».



Note : La classe « Downloader » a été ajoutée dans un deuxième temps. Tout comme avec la classe « Uploader », le contrôleur « Documents » dépend de la classe « Downloader » au travers d’une méthode additionnelle « downloadAction() ».

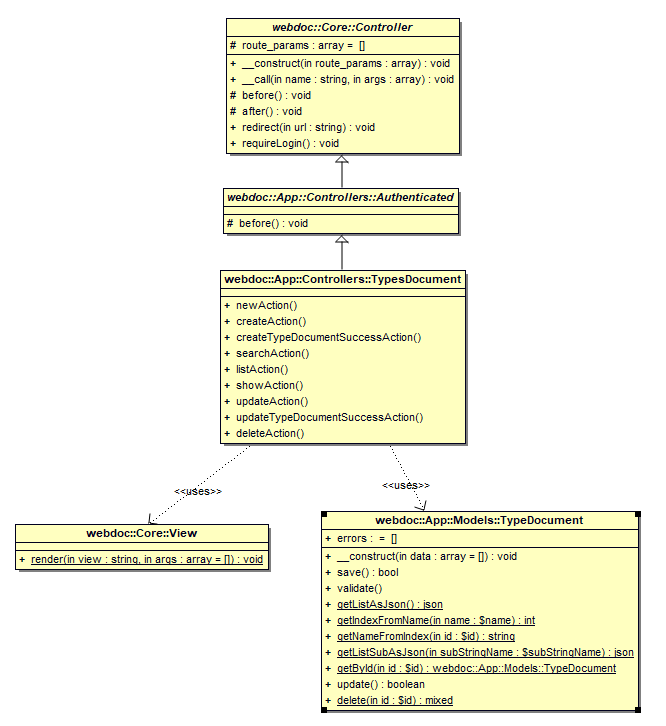
Contrôleur « Individus » :

Gère l’accès aux vues du namespace « App/Views  » et aux données distribuées par la classe « Individu » et « TypeIndividu » du namespace « App/Model ». Hérite de « Authenticated ».



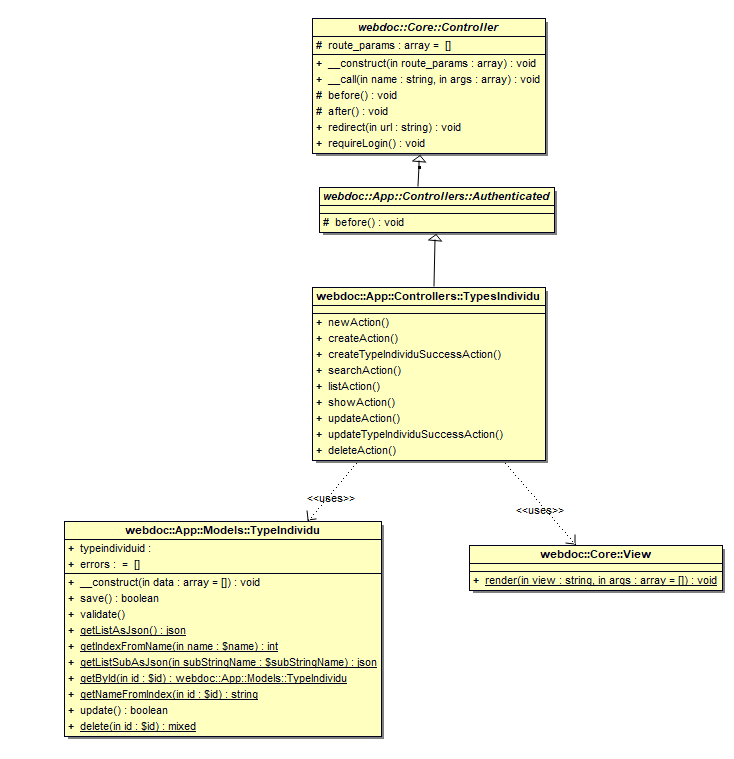
Contrôleur « TypesDocument » :

Gère l’accès aux vues du namespace « App/Views » et aux données distribuées par la classe « TypeDocument » du namespace « App/Model ». Hérite de « Authenticated ».



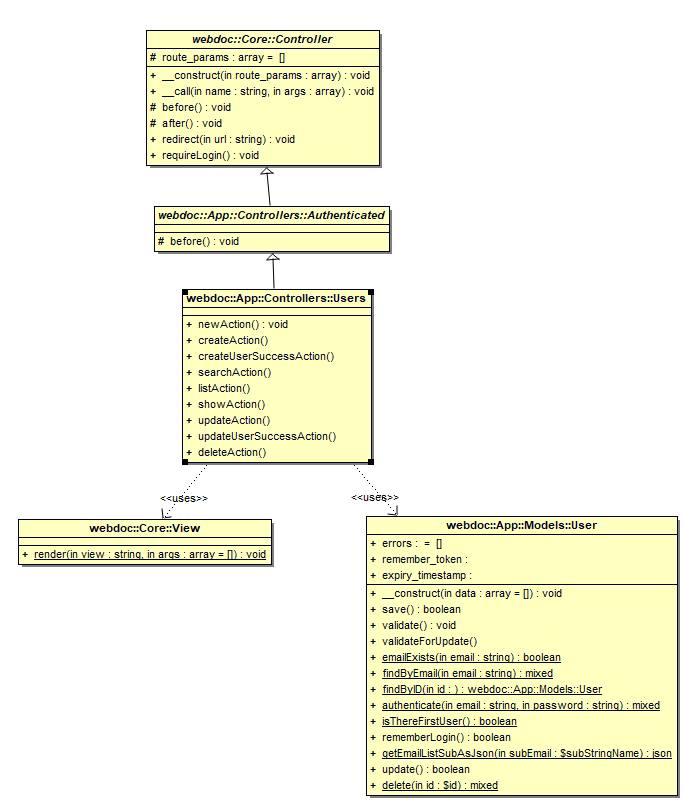
Contrôleur « TypesIndividu» :

Gère l’accès aux vues du namespace « App/Views » et aux données distribuées par la classe « TypeIndividu » du namespace « App/Model ». Hérite de « Authenticated ».



Contrôleur « Users » :

Gère l’accès aux vues du namespace « App/Views » et aux données distribuées par la classe « Users » du namespace « App/Model ». Il permet la création et la modification d’utilisateur. Hérite de « Authenticated ».

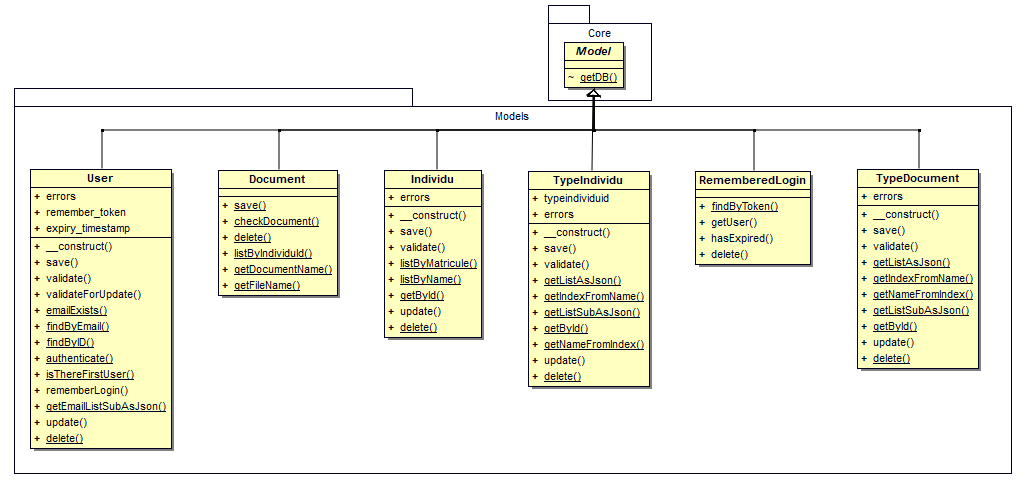


Note 1 : la classe « Role », ajoutée dans un deuxième temps et associée (lien d’association) à la classe « User » à travers une nouvelle propriété « $role », n’est pas présente sur le diagramme. (Voir Model/User & Model/Role pour plus de détail).

Note 2 : Une propriété supplémentaire, « $role », a été ajouté ajoutée à la classe « User ». Cette propriété est un objet instancié de la classe « Role ».

Namespace « App/Models » :

Les classes de ce namespace représente le modèle des données de l’application et assure l’accès à la base de données. Elles dérivent de la classe abstraite « Model » du namespace « Core ».



Note 1 : Une classe, « Rôle », a été ajoutée dans un second temps et n’est pas présente sur le schéma.

Note 2 : Une propriété supplémentaire à la classe « User », « $role », a été ajoutée dans un second temps. Il s’agit d’un objet instancié à partir de la classe « Role ».

Classe « User » du modèle :

Contrôle l’accès, la création et la modification des données de la table « Users » de la base de données. La classe est aussi responsable de la validation des données.

La classe à une deuxième fonction : représenter l’utilisateur courant de l’application à travers un objet « $user » sauvé comme une variable de session, « $\_SESSION[‘current\_user’] », et détruite au moment où l’utilisateur se déconnecte de l’application. Cet objet sert à contrôler l’accès aux fonctionnalités de l’application à travers la vérification des valeurs présentent dans une de ses propriétés : « $role ».

Si l’utilisateur ferme et ré-ouvre son navigateur, l’objet $user et la variable de session est recréée à partir du cookie de connexion créé lors de l’identification de l’utilisateur sur l’application.

Classe « Role » du modèle :

Un objet de la classe « Role » est créé par l’entremise de la création d’un object de la classe « User » lors de l’authentification de l’utilisateur courant par formulaire de connexion ou par cookie d’identification.

La classe « Role » reprend partiellement les données des tables « Roles » et « Permissions » à travers ses propriétés « $name » et « $permissions ». Le but de cet classe est d’identifier le rôle et les permissions attribuées à l’utilisateur courant et lui permettre ou nom de cette manière à accéder aux différentes fonctionnalités de l’application.

Classe « TypeDocument » du modèle :

Contrôle l’accès, la création et la modification des données de la table « TypesDocument » de la base de données. La classe est aussi responsable de la validation des données.

Classe « Document » du modèle :

Contrôle l’accès et la création des données de la table « Documents » de la base de données. Ainsi qu’à la table issue de la jonction entre les tables « Documents » et « TypesDocument ». La classe est aussi responsable de la validation des données.

Classe « TypeIndividu » du modèle :

Contrôle l’accès, la création et la modification des données de la table « TypesIndividu » de la base de données. La classe est aussi responsable de la validation des données.

Classe « Individu » du modèle :

Contrôle l’accès, la création et la modification des données de la table « Individus » de la base de données. Ainsi que la table issue de la jonction entre les tables « Individus » et « TypesIndividu ». La classe est aussi responsable de la validation des données.

Classe « RememberLogin » du modèle :

Contrôle l’accès aux données de l’entité RememberLogin de la base de données. Elle est responsable du stockage des données du cookie responsable de l’authentification automatique de l’utilisateur entre plusieurs sessions de travail.

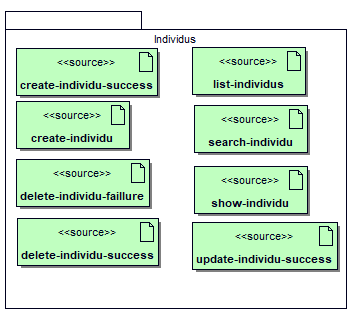
Namespace « App/Views »

Les vues ne sont pas composées de classes mais de simples fichier PHP contenant du code de html reprenant l’interface graphique ainsi que du code PHP de contrôle d’affichage des données.

Les vues sont distribuées au navigateur par la méthode statique render() de la classe « View » du namespace « App/Core ». Cette méthode permet aussi la transmission des données des contrôleurs aux vues sous forme d’array.

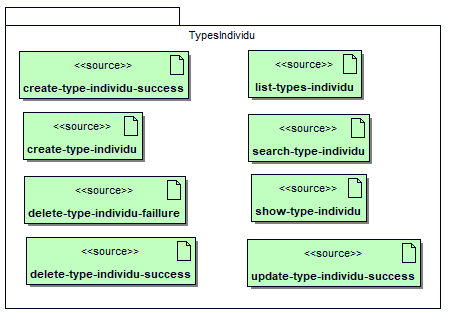
Vues « Individus » :

Contient les vues du namespace « App/Views/Individus »



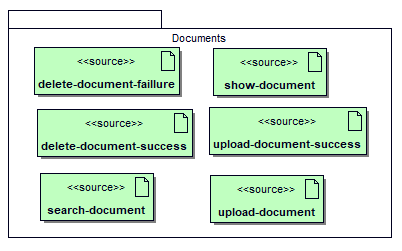
Vues « TypesIndividu » :

Contient les vues du namespace « App/Views/TypesIndividu ».



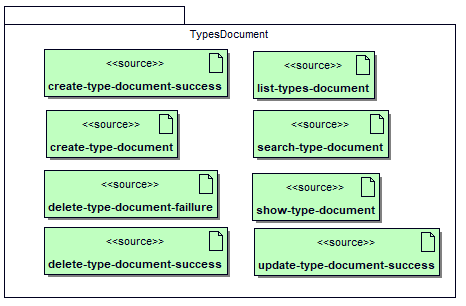
Views « Documents » :

Contient les vues du namespace « App/Views/Documents ».



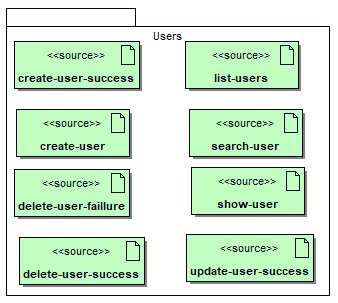
Views « TypesDocument » :

Contient les vues du namespace « App/Views/TypesDocument ».



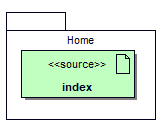
Views « Users » :

Contient les vues du namespace « App/Views/Users ».



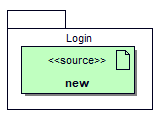
Views « Home » :

Contient les vues du namespace « App/Views/Home ».



Views « Login »:

Contient les vues du namespace « App/Views/Login ».



# Securité

## Serveur :

SSH : Le serveur est accessible au travers d’un serveur ssh par clé privée sécurisée par une passphrase. L’authentification au serveur par mot de passe a été désactivée. Les clés privées sont conservée dans la Keychain de mon Macbook.

SSL : Les connexions par browser sont sécurisées et cryptée par un certificat SSL émit par une autorité de certification publique, Let’s Encrypt.

Permission du système de fichier : Les permissions ont été réglées pour n’autoriser l’écriture que dans le dossier d’upload du dossier public.

## PHP

Les plugins non utilisés ont été désactivés.

APACHE :

Le dossier de l’application (root directory), inaccessible au public à l’exception du dossier public.

.htaccess régulant l’accès dans les deux principaux dossiers exposés au public.

MYSQL :

MYSQL accessible que depuis l’hôte localhost.

Utilisateur MYSQL différent de root (stephane) avec droit d’accès aux données.

Utilisation de méthodes PHP d’accès à la base de données sécurisant les requêtes SQL (injection SQL)

WEB :

Protection XSS par vérification des entrées coté serveur à l’aide de règle d’inclusion regex (système de liste blanche/whitelist).

J’ai fait le choix de ne pas automatiquement déconnecter l’utilisateur de sa session et de lui permettre de retrouver sa connexion entre sessions à l’aide d’un cookie survivant pendant 30 jours dans le browser, considérant que l’accès à application n’était possible en théorie d’accès qu’à partir d’un sous-réseau intranet de l’institution en faisant l’utilisation et m’inspirant aussi des pratiques actuelles de développement web. L’utilisateur a toutefois la possibilité de se déconnecter explicitement grâce à un lien du menu s’il le souhaite.

# Conclusion

La structure en MVC offre l’avantage d’une structure de travail solide et répétitive ainsi que d’un code très ordonné. Elle permet aussi, lors du travail d’équipe, de répartir plus facilement les tâches en poste de responsabilité indépendant les uns des autres.

Toutefois, elle n’est pas sans désavantage. Le premier venant à l’esprit étant une complexification de l’application à travers les moyens mis en œuvre pour établir la charpente (le « framework », ici « Core ») autour de laquelle peut ensuite s’épanouir sans heurt le code de l’application (les classes des namespace « App/Controller », « App/Models », « App/Views »).

De véritables frameworks, comme symphony, Laravel, CodeIgniter, permettent de ne pas devoir s’inquiéter des très nombreux détails techniques qu’il faut résoudre afin d’apporter les fonctionnalités nécessaires à une application moderne et à ne s’occuper que de la logique métier du domaine d’application que l’on veut implémenter.

J’ai au départ voulu m’orienter vers de telle solutions. La simplicité, les fonctionnalités et l’élégance d’un framework comme Laravel ma séduit dès le moment où j’ai envisagé le travail. La seule raison qui m’en a empêché à été la conviction que celui-ci ne serait pas accepté.

Toutefois, la création d’un mini framework de toute pièce, bien qu’inspirée d’un cours en ligne, m’a fait entrevoir le genre de prouesses techniques que requiert le développement de tels frameworks. Ceci en dépit de la simplicité du projet qui m’a été confié.