Année Scolaire : 2018 / 2019

Nom du Projet : MySchoolUniversity

CAHIER DES SPÉCIFICATIONS TECHNIQUES

Membres du groupe :

TRAORE Amel

YILMAZ Teoman

FOUDANE Omar

Introduction

Le cahier de spécifications techniques est destiné aux développeurs, qui voudront changer, améliorer ou bloquer une fonctionnalité du site web. Afin de garantir le bon fonctionnement du site, il est conseillé de maîtriser les langages du développement web : PHP, Javascript, CSS et HTML. Ce document contient toutes les règles respectées par l'ensemble des développeurs, de plus que les aspects techniques du projet.

Sommaire:

Règles d’or

Les règles suivantes sont respectées partout dans le site :

1) Afin d’éviter le conflit entre les classes des noeuds HTML, il est obligatoire d’encapsuler tous les noeuds appartenant à un module défini dans un conteneur HTML (de préférence un div).
2) Toutes les classes principales ne contrôlent pas les erreurs. Il les ignorent et laissent la responsabilité de les gérer aux classes appelantes.
3) Dans toutes les classes, pour inclure un fichier, il faudra ajouter __DIR__ avant le chemin du fichier. Cela permet de centraliser la vérification des modules et d’éviter les erreurs relatives à l’inclusion des fichiers si l’appel a été fait hors le fichier de redirection index.php.
4) Tous les messages d’erreurs et les constantes globales se trouvent dans un seul fichier common/constants.php.
5) Tous les différents messages d’erreurs font référence aux constantes globales.
6) Toute erreur est ajoutée dans un fichier de log.
7) Les répertoires sensibles sont protégés par un fichier .htaccess.
8) Tous les fichiers css et javascript sont tous importés dans chaque visite.
9) Les attributs en base de données ont toujours un suffix qui réfèrent au nom de la table: Par exemple id_utilisateur: L’identifiant dans la table utilisateur.
10) Afin d’accepter les données envoyées depuis un formulaire, il faudra obligatoirement fournir un token. Ce dernier est envoyé au client lorsqu’il affichera le formulaire concerné.
11) Toutes les requêtes sql exécutées dans une classe doivent être déclarées dans des variables privés et statiques. Cela nous facilitera la correction des requêtes, et rendra plus lisibles notre code.
12) Il ne faut pas oublier de fermer les curseurs après avoir récupéré plusieurs lignes de données de la base de données .

PHP

Répertoires :

- ★ **API**: Ce répertoire contient un fichier de redirection principale: index.php. Ce dernier permet, en fonction des paramètres fournis, permet d'appeler le module d'API Rest correspondant. Ce répertoire est accessible depuis l'extérieur, par conséquent, il ne doit pas contenir aucun fichier ou répertoire sensible.
- ★ **Common** : Contient toutes les classes principales, les bibliothèques et les fichiers sensibles. Il comprend la logique de l'application et représente donc le coeur de l'application. Il ne doit pas être modifié sauf en cas de besoin. Ce répertoire n'est pas accessible depuis l'extérieur vu qu'il est protégé par un fichier .htaccess
- ★ **Composants:** Ce répertoire contient les menus de notre application, de plus que le module responsable de l'affichage de l'erreur. Ce répertoire ne doit pas représenter en aucun cas la logique de l'application.
- ★ **Log:** Contient les fichiers de log. Protégé par un fichier .htaccess. Le sous répertoire log contient tous les logs de l'application, résultant des erreurs rencontrées. Le sous répertoire security contient toutes les exceptions concernant une tentative d'accès à des pages dont le droit n'est pas attribué à l'utilisateur courant.
- ★ **Modules:** Contient les fichiers indispensables pour l'architecture MVC (Modèle Vue Contrôleur). Ce répertoire est à modifier si vous voulez ajouter une autre page à l'application.
- ★ **private**: Contient les fichiers à supprimer directement après l'installation de l'application. Il en éxiste 2 :
    - uploadCity.php: Permettant d'insérer les villes en BD.
    - uploadCountry.php: Permettant d'insérer les pays en BD.

- ★ **upload:** Protégé par un fichier .htaccess. Il stocke les différents fichiers chargés sur le site, il comprend:
    - Les pièces jointes des mails
    - Les cours déposés par un enseignant
    - Les dépôts des étudiants.

Les exceptions:

Les exceptions permettent d'ajouter une couche de sécurité à notre application. Elles sont déclenchées dès qu'il y'a un comportement anormal sur le site. Ils arrêtent l'exécution du programme s'ils ne sont pas gérées. Dans notre application, toutes ces erreurs sont ajouter dans un fichier de log. Les exceptions que nous avions créer pour gérer les demandes anormales des utilisateurs sont:

- `ElementIntrouvable:` Déclenchée lorsqu'on essaie d'instancier une classe principale et qu'aucun élément ne représente l'instance demandée dans la base de données. Cette exception permet d'éviter les erreurs de suppression qui sembleront réussies, alors qu'en fait elles étaient ignorées : les requêtes de suppression envoyées à la base de données comportent une condition et si celle-ci n'est pas valide, alors aucune erreur n'est déclenchée automatiquement.
- `FichierInexistant:` Déclenchée lorsque le fichier envoyé depuis un formulaire n'existe pas: une pièce jointe à titre d'exemple.
- `NonAutoriseException:` Déclenchée lorsque l'utilisateur courant essaie d'accéder aux modules dont il n'a pas le droit. Souvent déclenchée par les méthodes des classes principales en fonction des informations demandés et des droits accordés à l'utilisateur courant.
- `NonConnecterException`: Déclenchée en cas de tentative d'accès aux pages internes sans qu'aucune connexion ne soit effectuée.
- `ParametresIncorrectes:` Permet de vérifier les informations saisies par l'utilisateur. Cette exception est déclenchée lorsque les données envoyées par un formulaire ne sont pas valides.
- `ParametresInsuffisantsException:` Permet de valider les formulaires. Elle est déclenchée lorsque l'utilisateur ne fournit pas tous les paramètres requis à sa demande.
- `PasEnseignantException`: Comme NonAutoriseException, mais cette exception concerne le type de l'utilisateur connecté. Déclenchée lorsque l'action demandée ne peut être effectuée que par les enseignants : par exemple "ouvrir dépôt" dans moodle.

Les classes génériques:

Ce sont des classes qui comportent des méthodes d'instances qui peuvent être utilisée par toutes les autres classes qui ont des responsabilités similaires. Ces classes se trouvent dans le répertoire common. Il en existe 4 dans cette application :

● ClasseGenerique:

Cette classe est responsable de la vérification de l'existence de l'instance demandée à la base de données. Elle intervient au moment de la création de l'objet et déclenche l'exception ElementIntrouvable si l'instance demandée n'existe pas. Toutes les classes principales héritent de cette classe.

● ContGenerique:

Tous les contrôleurs héritent de cette classe. Cette classe définit les méthodes suivantes :

```
/*
    Exception levée, lorsqu'un des paramètres est manquants:
    @param parameter: Le nom du paramètre manquant
*/
public function pasAssezDeParametres($parameter){}

/*
    -Génère un token pour la validation des formulaires
    -@return String token: Le token généré
*/
public function genererToken(){}

/*
    -Valide un token
    -Fait appel à la méthode pasAssezDeParametres si le token
est invalide.
*/

public function validerToken(){ }
```

● VueGenerique:

Héritée par toutes les vues de l'application. Elle comporte les méthodes d'affichages répétitives dans l'application : Un tableau par exemple. Le but de cette classe est de faciliter le changement d'affichage des composants et d'éviter la redondance du code. Les méthodes de cette classes sont :

```
            /*
                -Permet de transformer un tableau d'éléments en
liste
                -@param JSON items: Le tableau d'objets à
transformer en liste
                -@param String key : La clé de l'attribut à
afficher.
                -@param String class: La classe à appliquer aux
éléments affichés
                -@param String empty_message : Le message à
afficher si le tableau est vide ou n'est pas valide
            */
            public function toListItems($items, $key, $class =
'', $empty_message = ''){}

            /*
                -Donne une représentation graphique aux
conditions en utilisant des icons
                -@param boolean cond : La condition a évaluée.
                -@param String additionalClass: La classe css à
ajouter à l'icon
            */
            public function showCond($cond, $additionalClass =
''){}

            /*
                -Génère un champs du formulaire qui contient le
token
                -@param String token : Le token qui sert à la
validation du formulaire
            */
            public function inputToken($token){}

                /*
                - Transforme un tableau d'objets en un tableau de
suppression
                - Un tableau de suppression comporte un bouton
supprimer.
```

```
                - @param Json tableau : Le tableau des objets à
afficher.
                - @param String[] cles: Les clés dans l'ordre de
l'objet à afficher.
                - @param String cle_suppression : La clé utilisée
dans le lien de suppression
                - @param String debut_lien_suppression: Le lien
qui permet de supprimer la ligne sélectionnée. A ce lien, il est
ajouté la valeur de la clé de suppression
                - @param Fonction condition_suppression: La
condition a vérifier pour afficher une lien. Si aucune condition
n'est fourni alors toutes les lignes sont affichées

                - @return String le tableau en HTML.
            */
            public function afficherTableauSuppression($tableau,
$cles, $cle_suppression, $debut_lien_suppression, $enTete = null,
$condition_suppression = null, $classe_enTete = 'thead-dark'){}


            /*
                - Renvoie le corps du tableau de suppression,
regardez la méthode ci-dessus
                - Un tableau de suppression comporte un bouton
supprimer.

                - @param Json tableau : Le tableau des objets à
afficher.
                - @param String[] cles: Les clés dans l'ordre de
l'objet à afficher.
                - @param String cle_suppression : La clé utilisé
dans le lien de suppression
                - @param String debut_lien_suppression: Le lien
qui permet de supprimer la ligne séléctionnée. A ce lien, il est
ajouté la valeur de la clé de suppression
                - @param Fonction condition_suppression: La
condition a vérifiée pour activer ou pas le bouton de
suppression.
                    Si aucune condition n'est fournie alors tous
les boutons de suppression sont activés.
```

```
                - @return String le tableau en HTML.
            */
            public function
transformerEnTableauSuppression($tableau, $cles,
$cle_suppression, $debut_lien_suppression, $condition_suppression
= null){}


            /*

                - Cette fonction permet d'afficher un tableau,

                - @param JSON $data : Un tableau des objets à
afficher dans le tableau
                - @param String[] $keys : Les clés des éléments
qui seront afficher dans le tableau
                - @param String link_first_part: Le début du lien
qui sera associé au clique sur la ligne du tableau
                - @param String link_key : La clé utlisée pour
indiquée quelle partie du tableau 'Data' est utilisé dans le lien
                - @param String header : La liste des noms de
colonnes
                - @param String header_class : La classe qui sera
appliqué à l'entête

                - @return String : Le tableau en HTML
            */


            public function afficherTableau($data, $keys,
$link_first_part = '', $link_key = null, $header = null,
$header_class = 'thead-dark', $table_id = ''){}
```

- **ModAPIGenerique:**

Héritée par toutes les pages de redirection pour les APIs Rest. Comporte la même méthode `pasAssezDeParametres()` que `ContGenerique`, mais désignée pour les réponses d'API.

Classes Bibliothèques:

Ces classes ne comportent pas la logique d'application et ont pour responsabilité de faciliter le traitement de quelques actions redondantes. Elles sont réutilisables et permettent de bien séparer les responsabilités. Toutes ses classes ont été codées par nous même et ne sont pas importés depuis internet. Ces classes sont :

- ## Database

Permet d'initialiser la connexion à la base de données et d'interagir avec cette base de données, les méthodes de cette classe sont :

```
/*
    Initialise la connexion
*/
public static function initConnexion()
{}


/*
    - Renvoie l'instance de la base de données
*/
public static function getDB()
{}



/*
    -Renvoie la période maximale dans la base de données.
    -Si aucune date n'est présente, alors la période
actuelle est insérée.
*/

public static function getDBYear(){}

/*
    - Récupère l'astuce contenue dans le message d'erreur
d'une exception PDOException
    - @param PDOException exception : L'exception que l'on
veut traiter.
```

```
            - @return String Hint: L'astuce contenu dans le
message d'erreur
        */

        public static function getPDOHint($exception)
        {}
```

- **Date**

Modélise une date, la signature des méthodes de cette classe est :

```
        /*
            - Renvoie le mois courant
        */
        public static function getMonth()
        {}

        /*
            - Renvoie l'année courante
        */
        public static function getYear()
        {}

        /*
            -Renvoie la date correspondante à la période courante:
            -Les deux périodes possibles sont :
                -Premier semestre : YYYY-09-01 => (YYYY + 1)-02-01
                -Deuxième semestre: YYYY-02-01 => YYYY-07-01
            -return Json : {
                debut: date,
                fin: date
            }
        */
        public static function getPeriodeCourante()
        {}
```

- **Fichier:**

Cette classe est chargée de la gestion des fichiers. Elle comporte une seule méthode:

```
        /*
            -Permet de télécharger un fichier s'il existe
        */
        public function telecharger($filename)
        {}

       /*
            - Récupère le contenu d'un fichier csv, et le transforme en un Tableau.
            - @param String separator: Le séparateur des colonnes dans le fichier.
            - @return String[][] : Le contenu du fichier csv.
        */
        public function fichierEnTableau($separator)
        {}

        /*
            - Supprime le fichier
        */
        public function supprimer()
        {}
```

- **FileUpload**

Cette classe gère le chargement des fichiers envoyés par les formulaires. Elle s'occupe de la vérification de l'existence du fichier et la validation de son extension. La signature des méthodes de cette classe est :

```
    /*
        -Créer une instance du fichier chargé depuis un formulaire
        -@param String filename: Le nom du fichier dans le formulaire.
        -@Throws FichierInexistant: Si le fichier n'existe pas
    */
    public function __construct($fileName)
    {}
```

```
    /*
        - Copie un fichier sur le disque
        - @return boolean uploaded: Vrai si le fichier a bien été
copier sur le disque
    */

    public function copyFile()
    {}

    /*
        - Valide le fichier en vérifiant l'extension du fichier.
    */
    public function checkMimes($mimes)
    {}




    /*
        - Récupère l'éxtension du fichier chargé
    */
    public function getExtension()
    {}

    /*
        - Renvoie l'emplacement du fichier sur le disque
    */
    public function getFullPath(){}
```

- **Response**

Cette classe est utilisé par les API afin d'envoyer une réponse ou une erreur au client. La signature des méthodes de cette classe est :

```
            /*
           - Envoie une erreur au client et arrête tous les
traitements en cours sur le serveur
           - @param Integer error_code: Le code d'erreur.
           - @param String message : Le message à envoyer.
```

```
            - @param PDO db : L'instance de connexion à la base de
donnée.
            - @param Fichier file : Le fichier en cours de
traitement.
        */
        public static function send_error($error_Code, $message,
PDO $db = null, $file = null) : void
        {}

        /*
            - Envoie un réponse positive au client et arrête
l'exécution du script.
            - @param body: La réponse en Json.
        */
        public static function sendHttpBodyAndExit($body = null)
        {}
```

- **Token:**

Génère et valide le token utilisé dans les formulaire de cette application. La signature des méthodes de cette classe est :

```
    /*
        - Valide un token
        - @param token : Le token à valider
        - @return boolean : Vrai si le token est valide, faux
sinon.
    */
    public static function validateToken($token){}

    /*
        - Crée un token
        - @return String token : Le token créé
    */
    public static function createToken(){}
```

- **ErrorHandler:**

Gestionnaire des erreurs. Permet d'afficher l'erreur au client, ainsi qu'ajouter cette erreur dans les logs. La seule méthode accessible de cette classe est :

```
            - Gère une exception, puis redirige l'utilisateur
vers la page d'erreur
            - @param Exception e: L'exception à logger
            - @param String titreErreur: Le titre de l'erreur
            - @param String message: Le message d'erreur à
afficher
            - @param Json remplacements: L'objet contenant les
associations clé=>valeur.
                Les occurrences de {{clé}} dans le message
d'erreur seront remplacés par leur valeur.
        */
        public static function afficherErreur($e, $titreErreur
= DEFAULT_ERROR_TITLE, $messageErreur =
DEFAULT_ERROR_MESSAGE, $remplacements = array() ){}
```

- **ErrorHandlerAPI:**

Étend ErrorHandler, elle permet de logger les erreurs qui surviennent au niveau des API. La seule méthode de cette classe est:

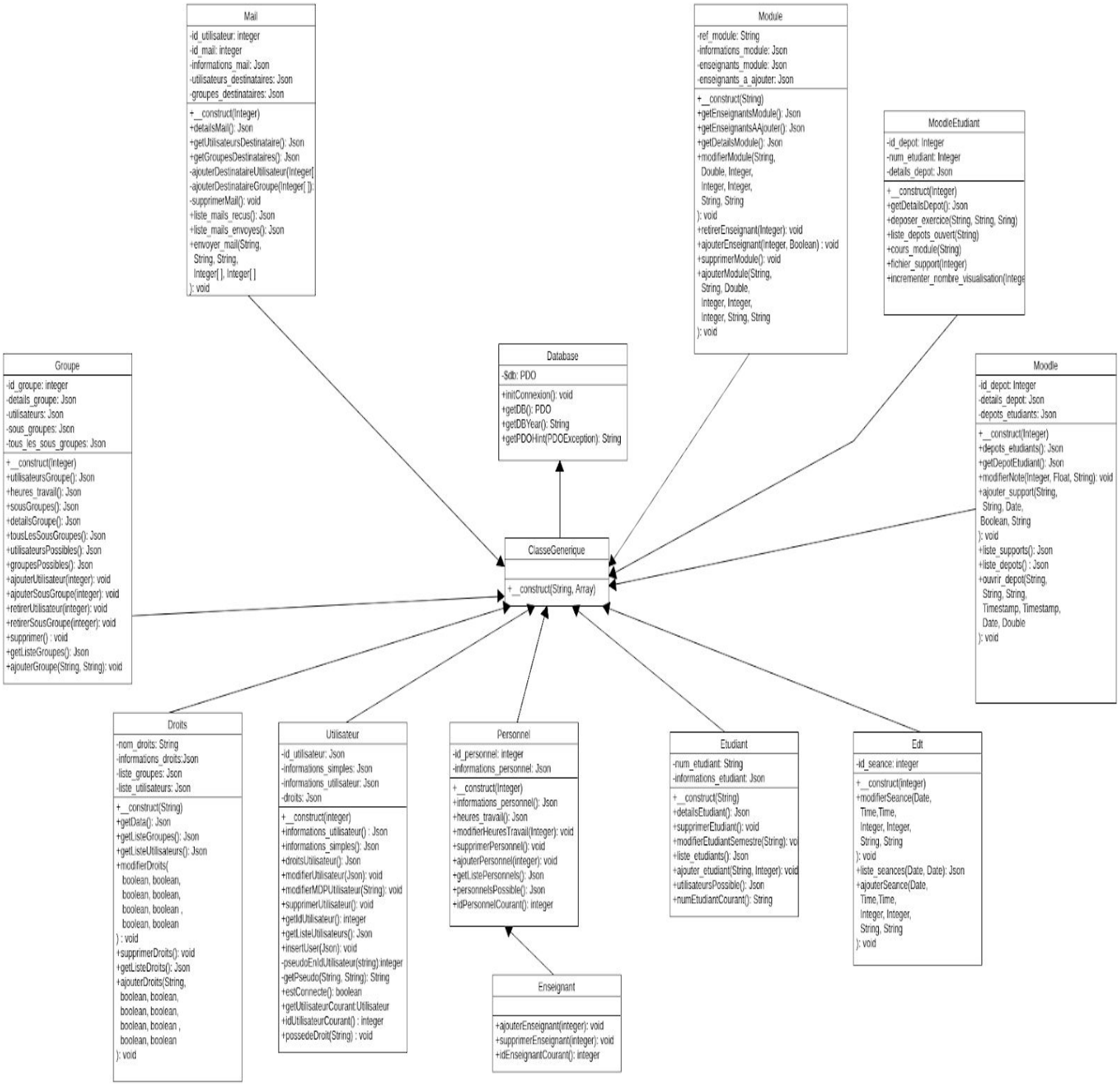
```
        /*
            - Gère une execption, puis redirige l'utilisateur
vers la page d'erreur
            - @param Exception e: L'exception à logger
            - @param String titreErreur: Le titre de l'erreur
            - @param String message: Le message d'erreur à
afficher
            - @param Json remplacements: L'objet contenant les
associations clé=>valeur.
                Les occurrences de {{clé}} dans le message
d'erreur seront remplacés par leur valeur.
            - @param Integer http_error_code: Le code d'erreur
à envoyer
        */
        public static function afficherErreur($e, $titreErreur
= DEFAULT_API_ERROR_TITLE, $messageErreur =
DEFAULT_API_ERROR_MESSAGE, $remplacements = array(),
$http_error_code = INTERNAL_SERVER_ERROR ){}
```

Classes Principales:

Les classes principales sont responsables de la logique de l'application. Ces classes représentent les objets en base de données. Une vérification des droits est faites avant de fournir les données aux méthodes appelantes. Si les droits requises pour l'action demandée ne sont pas accordés, alors l'exception NonAutoriseException est déclenchée. Toutes les classes principales étendent ClasseGenerique, par conséquent, une vérification préalable de l'existence de l'objet demandé est occasionné avant l'instanciation. Si cette instance n'existe pas en base de données, alors l'exception ElementIntrouvable est levée. Sauf quelques exceptions. Ces classes sont les seuls à opérer avec la base de données.

Ces classes aident à mieux architecturer l'application. En effet en centralisant la logique d'application, les API et le modèle de l'architecture MVC pourront accéder aux même ressources sans devoir à dupliquer le code.

Diagramme de classes



Vous pouvez télécharger cette image sur le lien : https://ibb.co/w7MGqWT

➔ Utilisateur:

```
        /*
         - Crée une instance d'un utilisateur déjà éxistant
         - @param id_utilisateur: L'identifiant de l'utilisateur
éxistant
         - @param pseudo_utilisateur: Le pseudo de l'utilisateur
éxistant
         - @throws PDOException : Si un problème se produit lors de
l'éxecution de la requête
         - @throws ElementIntrouvable: Si l'utilisateur n'éxiste pas
        */
        public function __construct($id_utilisateur = "",
$pseudo_utilisateur = ""){}


            /*
             - Récupère les informations de l'utilisateur
             - Les informations de l'utilisateurs sont stockés dans une
variable d'instance
               pour qu'il soit possible de récupèrer l'éxception par le
déclencheur
               et ne plus avoir à requeté une deuxième fois. Cela
évitera d'avoir des excéptions non gérées

             - @throws PDOException: Si les informations_utilisateur n'a
jamais été lancée auparavant
               et qu'une erreur se produise lors de l'éxecution de la
requête.
             - @throws NonAutoriseException: Si l'utilisateur actif n'a
pas le droit de création des utilisateurs
               et s'il veut consulter les droits d'un utilisateur autre
que lui-même

             - Attention cette fonction permet de récupérer l'ensemble
des informations relatives à l'utilisateur
```

```
            Il faudra filter les données après l'éxecution de cette
requête.
            */
            public function informations_utilisateur(){}

            /*
                -Récupère les informations non sensibles de
l'utilisateur
                -@returns data: {
                    id_utilisateur: integer,
                    nom_utilisateur: string,
                    pseudo_utilisateur: string,
                    prenom_utilisateur: string,
                    semestre: string,
                    est_enseignant: boolean,
                    est_personnel : boolean,
                    est_etudiant : boolean
                }
                -@Throws PDOException: A la première éxecution
                -@Throws NonAutoriseException : Si l'utilisateur courant
ne possède pas le droit de création des utilisateur
                    et s'on cherche les informations d'un autre
utilisateur.
            */
            public function informations_simples(){}

            /*
             - Récupère les droits de cet utilisateur. Ces droits inclus
aussi
               les droits du sous groupes auxquels il appartient.
             - @throws PDOException: Si une erreur se produit lors de
l'éxecution de la requête
             - @throws NonAutoriseException: Si l'utilisateur actif n'a
pas le droit de création des utilisateurs
               et s'il veut consulter les droits d'un utilisateur autre
que lui-même
             */
            public function droitsUtilisateur(){}
```

```
            /*
             - Modifie un utilisateur donnée.
             - @param data: Un tableau associative contenant l'ensemble
des informations relatives à l'utilisateur
             data : {
                 'email'            : string,
                 'nom'              : string,
                 'prenom'           : string,
                 'tel'              : string,
                 'adresse'          : string,
                 'est_homme'        : boolean,
                 'date_naissance'   : date,
                 'droits'           : string (le nom du droits
associés),
                 'pays_naissance'   : string,
                 'code_postal'      : string
             }

             - @throws PDOException : Si la modification de
l'utilisateur a échouée
             - @throws NonAutoriseException: Si l'utilisateur ne possède
pas le droit de création utilisateurs
            */
            public function modifierUtilisateur($data)
            {}


            /*
             - Modifie le mot de passe de l'utilisateur
             - @param mdp: Le nouveau mot de passe de l'utilisateur
             - @throws PDOException
             - @throws NonAutoriseException: Si l'utilisateur ne possède
pas le droit de création utilisateurs
               et s'il veut changer un mot de passe d'un autre
utilisateur autre que lui-même

            */

            public function modifierMDPUtilisateur($mdp)
            {}
```

```
            /*
             - Supprime cet utilisateur
             - @throws NonAutoriseException: Si l'utilisateur ne possède
pas le droit de création utilisateurs
             - @throws PDOException: Si la suppression de cet
utilisateur est impossible.
                Cela peut être dû à plusieurs raisons tels que :
                -L'identifiant de l'utilisateur est déjà une clé
étrangère dans une autre table
            */
            public function supprimerUtilisateur()
            {}


            /*
            GETTERS
            */
            public function getIdUtilisateur(){}









            /*
             - Renvoie la liste de tous les utilisateurs éxistants dans
la base de données
             - Cette liste contient:
                    --L'identifiant de l'utilisateur
                    --Pseudo de l'utilisateur
                    --Nom de l'utilisateur
                    --Prenom de l'utilisateur
                    --N°Tel de l'utilisateur
                    --Mail d'inscription de l'utilisateur
                    --Date de naissance de l'utilisateur
```

```
             - @Throws PDOException

            */

            public static function getListeUtilisateurs()
            { }




            /*
             - Permet la création d'un nouveau utilisateur
             - @param
                data : {
                    'email'            : string,
                    'nom'              : string,
                    'prenom'           : string,
                    'tel'              : string,
                    'adresse'          : string,
                    'est_homme'        : boolean,
                    'date_naissance'   : date,
                    'droits'           : string (le nom du droits
associés),
                    'pays_naissance'   : string,
                    'code_postal'      : string
                }
             - @throws PDOException
             - @throws NonAutoriseException: Si l'utilisateur ne possède
pas le droit de création utilisateurs

            */
            public static function insertUser($data)
            {}

            /*
             - Renvoie l'identifiant d'utilisateur correspondant au
pseudo passé en paramètre
             - @param pseudo: Le pseudo de l'utilisateur
             - @throws PDOException
```

```
            */
            private static function pseudoEnIdUtilisateur($pseudo)
            {}


            /*
             - Forme le pseudo de l'utilisateur
             - Le pseudo de l'utilisateur est soit constitué des
premières lettre du prenom suivi du nom
                Soit on y ajouter un nombre aléatoire entre 0 et 400.
             - @param nom: Le nom de l'utilisateur
             - @param prenom: Le prenom de l'utilisateur
             - @throws PDOException

            */
            private static function getPseudo($nom, $prenom)
            {}


            /*
             - Renvoie vrai si l'utilisateur est connecté
            */
            public static function estConnecte()
            {}

            /*
             - Renvoie l'instance de l'utilisateur actif
            */
            public static function getUtilisateurCourant(){}


            /*
                -Renvoie l'identifiant de l'utilisateur connecter
                -@Throws NonConnecterException : Si aucun utilisateur
n'est connecter
            */
            public static function idUtilisateurCourant(){}

            /*
             - Vérifie si l'utilisateur actif possède le droit nom_droit
```

```
            - @param nom_droit : Le nom du droit que l'on souhaite
vérifier
            - @throws NonAutoriseException : Si le droit n'est pas
accordé
           */
           public static function possedeDroit($nom_droit){}
```

➔ Personnel

```
           /*
            - Crée une instance d'un personnel en se basant sur son
      identifiant
            - @param id_personnel: L'identifiant du personnel qu'on
      veut créer
            - @Throws PDOException : Si l'éxecution de la requête à
      échouée.
           */
           public function __construct($id_personnel){}


           /*
            - Récupère les informations relatives au personnel
      courant

            - @Return data = {
                  id_utilisateur      : integer,
                  id_personnel        : integer,
                  id_enseignant       : integer or null,
                  nom_utilisateur     : string,
                  prenom_utilisateur  : string,
                  pseudo_utilisateur  : string,
                  annee_courante      : string, "départ => fin"
                  heures_travail      : integer (Total des heures de
      travail pour le semestre courant )
            };

            - @Throws NonAutoriseException: Si l'on essaie de
      récupérer les détails d'un autre personnel
               et qu'on a pas le droit création_utilisateurs
```

```
         - @Throws PDOException : Si l'éxecution de la requête à
échouée.
        */

        public function informations_personnel()
        {}


        /*
         - Récupère l'ensemble des heures de travail du personnel
         - @Throws NonAutoriseException: Si l'on essaie de
récupérer les détails d'un autre personnel
            et qu'on a pas le droit création_utilisateurs
         - @Throws PDOException : Si l'éxecution de la requête à
échouée.
        */

        public function heures_travail()
        {}


        /*
            - Modifie les heures de travail du personnel pendant
la période courante.

            - @Throws PDOException : Si l'exécution de la requête
a échoué
            - @Throws NonAutoriseException : Si l'utilisateur ne
possède pas le droit_modification_heures de travail,
                en plus que le droit creation utilisateurs
        */
        public function modifierHeuresTravail($heures_travail)
        {}

        /*
            - Supprime un personnel
            - @Throws PDOException.
            - @Throws NonAutoriseException : Si l'utilisateur ne
possède pas le droit "création_utilisateurs"
        */
```

```
        public function supprimerPersonnel()
        {}



        /*
            - Inscrit un personnel dans la base de données
            - @Throws PDOException.
            - @Throws NonAutoriseException : Si l'utilisateur ne
possède pas le droit "création_utilisateurs"
        */

        public static function ajouterPersonnel($id_utilisateur)
        {}



        /*
            - Récupère la liste de tous les personnels
            - Chaque ligne du tableau est du type : {
                id_utilisateur      : integer,
                id_personnel        : integer,
                id_enseignant       : integer or null,
                nom_utilisateur     : string,
                prenom_utilisateur  : string,
                pseudo_utilisateur  : string,
                heures_travail      : integer (Total des heures de
travail pour ce personnel )

            }
            - @Throws PDOException.
            - @Throws NonAutoriseException : Si l'utilisateur ne
possède pas le droit "création_utilisateurs"
        */

        public static function getListePersonnels()
        {}



        /*
            -Renvoie la liste du personnel possible
```

```
                -@return {
                    id_utilisateur: integer,
                    pseudo_utilisateur: string,
                    nom_utilisateur: string,
                    prenom_utilisateur:string
                }
                -@Throws PDOException
                -@Throws NonAutoriseException: Si l'utilisateur
    courant ne possède pas le droit de création du personnel
            */
            public static function personnelsPossible(){}



            /*
             - Renvoie l'identifiant du personnel courant
             - Si l'utilisateur courant n'existe pas ou n'est pas un
    personnel,
                alors null est retourné
            */
            public static function idPersonnelCourant(){}
```

➔ Enseignant

```
        /*
         - Ajoute un personnel dans la table enseignant
         - @param id_personnel : L'identifiant d'un personnel

         - @Throws PDOException.
         - @Throws NonAutoriseException: Si le droit de création
utilisateur n'est pas accordé

        */
        public static function ajouterEnseignant($id_personnel)
        {}

        /*
         - Supprime un enseignant existant.
```

```
        - @param id_personnel: L'identifiant du personnel associé à cet
enseignant.

        - @Throws NonAutoriseException: Si le droit de création
utilisateur n'est pas accordé.
        - Attention: Aucune erreur ne sera déclenchée si l'enseignant à
supprimer n'existe pas.
       */
       public static function supprimerEnseignant($id_personnel)
       {}



       /*
           - Renvoie l'identifiant de l'enseignant connecter
           - @Throws PasEnseignantException : Si l'utilisateur connecté
n'est pas un enseignant
       */

       public static function idEnseignantCourant(){}
```

➔ Droits

```
       /*
        - Instancie un droit identifié par son nom
        - @param nom_droits: Le nom du droit à instancié
        - @Throws PDOException : Si une erreur se produit au niveau de
la base de données
        - @Throws ElementIntrouvable: Si le droit est inéxistant
       */
       public function __construct($nom_droits)
       {}

       /*
        - Renvoie les données relatives à ce droit
        - @return data : {
            nom_droits                          : string,
            droit_creation_utilisateurs         : boolean,
```

```
            droits_creation_modules            : boolean,
            droit_creation_cours               : boolean,
            droit_creation_groupes             : boolean,
            droit_modification_absences        : boolean,
            droit_modification_droits          : boolean,
            droit_modification_heures_travail  : boolean,
            droit_visualisation_statistique    : boolean
         }
         - @Throws PDOException : Si l'exécution de la requête a échoué
        */
        public function getData(){}



        /*
         - Renvoie la liste des groupes qui ont ce droit
         - @return data : {
            nom_groupe : string
          }
         - @Throws PDOException : Si l'exécution de la requête a échoué
         - @Throws NonAutoriseException: Si l'utilisateur ne possède pas
le droit de modification droits
        */

        public function getListeGroupes()
        {}

        /*
         - Renvoie la liste des utilisateurs qui ont ce droit
         - @return data : {
            nom_groupe : string
          }
         - @Throws PDOException : Si l'exécution de la requête a échoué
         - @Throws NonAutoriseException: Si l'utilisateur ne possède pas
le droit de modifications des droits
        */

        public function getListeUtilisateurs()
        {}
```

```
        /*
         - Modifie ce droit

         - @param creation_utilisateur, ..., statistiques: Vrai si
accordée, faux sinon

         - @Throws PDOException.
         - @Throws NonAutoriseException : Si l'utilisateur courant n'a
pas le droit de modification des droits
        */

        public function modifierDroits(
            $creation_utilisateurs,
            $creation_modules,
            $creation_cours,
            $creation_groupes,
            $modification_absences,
            $modification_droits,
            $modification_heures_travail,
            $statistiques
        ) {}

        /*
         - Supprime ce droit

         - @Throws PDOException: Si ce droit est déjà attribué à un
groupe ou un utilisateur
         - @Throws NonAutoriseException : Si l'utilisateur courant n'a
pas le droit de modification droits
        */

        public function supprimerDroits()
        {}
```

```
        /*
         - Renvoie la liste de tous les droits
         - @Throws PDOException.
         - @Throws NonAutoriseException : Si l'utilisateur courant n'a
pas le droit de création utilisateurs ou le droit de création des
groupes
        */

        public static function getListeDroits()
        {}



        /*
         - Ajoute un droit dans la base de données
         - @param nom_droits : Le nom du droit à créer
         - @param creation_utilisateur, ..., statistiques: Vrai si
accordée, faux sinon

         - @Throws PDOException.
         - @Throws NonAutoriseException : Si l'utilisateur courant n'a
pas le droit de modifications des droits
        */

        public static function ajouterDroits(
            $nom_droits,
            $creation_utilisateurs,
            $creation_modules,
            $creation_cours,
            $creation_groupes,
            $modification_absences,
            $modification_droits,
            $modification_heures_travail,
            $statistiques
        ) {}
```

➔ Edt:

Représente une séance. Une séance est valide si :

1. Sa durée en minutes est un multiple de 10.
2. Le jour de la séance n'est ni samedi ni dimanche
3. La séance se déroule entre 8h00 et 19h00
4. L'enseignant du module n'est pas déjà occupé avec un autre module
5. Une autre séance n'est pas planifiée dans la même salle

```
        /*
            - Instancie une nouvelle séance
            - @param id_seance: L'identifiant de la séance
            - @throws PDOException : Si la requête de vérification
a échouée
            - @throws ElementIntrouvable: Si aucune séance ne porte
l'identifiant du séance passé en paramètre
        */

        public function __construct($id_seance){}


        /*
            - Permet de modifier cette séance

            - @param date_seance:  La nouvelle date de début
(YYYY-MM-DD)
            - @param heure_depart : L'heure dans laquelle va
commencé la séance
            - @param duree : La durée de séance 'hh:mm:ss'
            - @param groupe: l'identifiant du groupe
            - @param enseignant: L'identifiant d'enseignant qui
s'occupera de cette séance
            - @param module : La référence du module enseigné dans
cette séance
            - @param salle : Le nom de la salle dans laquelle va se
dérouler cette séance

            -@return : L'identifiant de la séance modifiée
```

```
            - @Throws PDOException: Si la séance n'est pas valide
(Si elle ne respecte pas les conditions déclarées ci-dessus)
            - @Throws NonAutoriseException: Si l'utilisateur
courant ne possède pas le droit droit_creation_cours
        */
        public function modifierSeance($date_seance, $heure_depart,
$duree, $groupe, $enseignant, $module, $salle){}


        /*
            - Renvoie la liste des séances entre deux dates

            - @param debut  : La date du début de la séance
            - @param fin    : La date maximale des séances
recherchées

            -@return data   : {
                id_seance: integer,
                id_groupe: integer,
                id_enseignant: integer,
                ref_module: string,
                ref_semestre: string,
                nom_groupe: string,
                nom_salle: string,
                date_seance : date 'YYYY-MM-DD',
                heure_depart_seance: time 'hh:mm:ss',
                duree_seance : time,
                nom_groupe: string,
                nom_enseignant: string,
                prenom_enseignant: string,
                nom_module: string,
                couleur_module: string,
                abreviation_module: string
            }

            - @Throws PDOException: Si un problème survient au
niveau de la base de données
        */
        public static function liste_seances($debut, $fin){}
```

```
        /*
            -Ajoute une séance dans la base de données
            -@param : Comme pour modifierSeance
            -@return : L'identifiant de la nouvelle séance
            -@Throws PDOException: Si la séance n'est pas valide,
c'est-à-dire qu'elle ne respecte pas les conditions de validité
d'une séance
            -@Throws NonAutoriseException: Si l'utilisateur courant
ne possède pas le droit droit_creation_cours
        */
        public static function ajouterSeance($date_seance,
$heure_depart, $duree, $groupe, $enseignant, $module, $salle){}
```

➔ Etudiant

```
      /*
       - Instancie un étudiant.
       - @Throws PDOException: Si l'éxecution de la requête a
échouée
       - @Throws ElementIntrouvable : Si l'étudiant n'existe pas.
      */
      public function __construct($num_etudiant)
      {}
```

```
      /*
        - Récupère la moyenne de cet étudiant pendant tous les
semestres qu'il a passé
        - @return data = {
            moyenne : float,
            ref_semestre: string,
            date_debut: date,
            date_fin : date
        }
        -@Throws PDOException.
```

```
       -@Throws NonAutoriseException : Si on essaie d'accéder aux
détails d'un autre étudiant
       et qu'on a pas le droit creation_utilisateurs
      */
     public function detailsEtudiant()
     {}


     /*
       -Supprime cet étudiant
       -@Throws PDOException : Si le num étudiant est associé à
d'autres informations dans la BD.
       -@Throws NonAutoriseException: Si l'utilisateur courant n'a
pas le droit de création utilisateurs
     */

   public function supprimerEtudiant() {}



   /*
    - Modifie le semestre de l'étudiant
    - @param semestre string: Le nouveau semestre de l'étudiant.
    - @Throws PDOException : Si la modification a échouée
    - @Throws NonAutoriseException : Si l'utilisateur courant n'a
pas le droit de création utilisateurs
   */

   public function modifierEtudiantSemestre($semestre)
   {}

   /*
     - Récupère la liste des étudiants.
     - Les informations retournées pour chaque étudiant sont:
         -num_etudiant       : string,
         -pseudo_utilisateur : string,
         -nom_utilisateur    : string,
         -prenom_utilisateur : string,
         -points_ets         : integer
     -@Throws PDOException.
```

```
    */
    public static function liste_etudiants() {}


    /*
     - Ajoute un étudiant dans la base de données
     - @param num string: Le numéro d'étudiant.
     - @param id_utilisateur int: L'identifiant de l'utilisateur
que l'on souhaite rendre un étudiant.
     - @Throws PDOException : Si l'insertion a échouée
     - @Throws NonAutoriseException : Si l'utilisateur courant n'a
pas le droit de création utilisateurs
    */
    public static function ajouter_etudiant($num, $id_utilisateur)
    {}



    /*
      - Renvoie la liste des utilisateur pouvant être des étudiants
      - @return tableau de : {
        id_utilisateur: integer,
        pseudo_utilisateur: string
      }
      -@Throws PDOException .
      -@Throws NonAutoriseException : Si l'utilisateur courant ne
possède pas le droit création utilisateurs
    */

    public static function utilisateursPossible(){}

    /*
      - Renvoie le numéro de l'étudiant courant
      - Si l'utilisateur courant n'existe pas ou n'est pas un
étudiant,
        alors null est retourné
    */
    public static function numEtudiantCourant(){}
```

➔ Groupe

```
        /*
            - Instancie un groupe
            - @Throws PDOException
            - @Throws ElementIntrouvable: Si aucun groupe ne porte
l'id_groupe passé en paramètre
        */
        public function __construct($id_groupe)
        {}


        /*
            - Récupère tous les utilisateurs appartenant
directement au groupe et non pas à l'un des sous groupes.
            - @return tableau de data : {
                    id_utilisateur: integer,
                    pseudo_utilisateur: string,
                    id_personnel: integer,
                    id_enseignant: integer,
                    num_etudiant: integer,
                    periode : String date_debut || ' => ' ||
date_fin
                }
            - @Throws PDOException.
        */
        public function utilisateursGroupe()
        {}



        /*
         - Retourne la liste des sous-groupes directs de ce groupe.
         - @return tableau de data: {
                id_groupe : integer,
                nom_groupe: string,
                nombre_utilisateurs: integer
         }
         - @Throws PDOException
```

```
        */
        public function sousGroupes()
        {}


        /*
            - Récupère les détails du groupe
            - @return data : {
                - id_groupe : integer,
                - nom_groupe: string,
                - nom_droits: string
            }
            -@Throws PDOException
            -@Throws NonAutoriseException : Si l'utilisateur
courant n'a pas le droit de droit_creation_groupes
        */
        public function detailsGroupe()
        {}


        /*
            - Récupère tous les sous-groupes directs ou indirects
d'un groupe
            - Le groupe courant sera aussi inscrit dans la liste
des groupes.
            - @return tableau de data : {
                id_groupe: integer,
                nom_groupe: integer
            }
            - @Throws PDOException.
        */
        public function tousLesSousGroupes(){}

        /*
            -Renvoie la liste des utilisateurs qui ne sont pas dans
ce groupe
            -@return tableau de :{
                id_utilisateur: integer,
                pseudo_utilisateur: string,
                nom_utilisateur: string,
```

```
                prenom_utilisateur: string
            }
            -@Throws PDOException
            -@Throws NonAutoriseException : Si l'utilisateur
courant ne possède pas
            le droit de création des groupes
        */

        public function utilisateursPossibles(){ }



         /*
            -Renvoie la liste des groupes qui ne font pas partie
des sous-groupes directes de ce groupe
            -@return tableau de :{
                id_utilisateur  : integer,
                nom_groupe      : string
            }
            -@Throws PDOException
            -@Throws NonAutoriseException : Si l'utilisateur
courant ne possède pas le droit de création des groupes
        */

        public function groupesPossibles(){ }

        /*
            - Ajoute un utilisateur au groupe
            - @param id_utilisateur: l'identifiant de l'utilisateur
à ajouter
            - @Throws PDOException : Si l'insertion a échoué
            - @Throws NonAutoriseException: Si l'utilisateur
courant ne possède pas le droit de création groupes
        */
        public function ajouterUtilisateur($id_utilisateur)
        {}


        /*
            - Ajoute un sous-groupe au groupe
```

```
            - @param sous-groupe: l'identifiant du groupe à ajouter
            - @Throws PDOException : Si l'insertion a échoué ou si
on essaie d'ajouter un sous-groupe dont il est déjà enfant
            - @Throws NonAutoriseException: Si l'utilisateur
courant ne possède pas le droit de création des groupes
        */
        public function ajouterSousGroupe($sous_groupe)
        {}


        /*
            - Retire un utilisateur de la liste des utilisateurs
directs du groupe
            - Attention: Cette méthode ne retire pas l'utilisateur
des sous-groupes.
            - @param id_utilisateur: l'identifiant de l'utilisateur
à ajouter
            - @Throws PDOException : Si la suppression a échoué
            - @Throws NonAutoriseException: Si l'utilisateur
courant ne possède pas le droit de création des groupes
        */

        public function retirerUtilisateur($id_utilisateur)
        {}


        /*
            - Retire un sous_groupe au groupe
            - @param sous_groupe: l'identifiant du groupe à ajouter
            - @Throws PDOException : Si la supression a échouée
            - @Throws NonAutoriseException: Si l'utilisateur
courant ne possède pas le droit création groupes
        */

        public function retirerSousGroupe($sous_groupe)
        {}


        /*
            - Supprime ce groupe
```

```
            - Supprime aussi tous les sous groupes et les
utilisateurs appartenant à ce groupe
            - @Throws PDOException : Si la suppression a échoué
            - @Throws NonAutoriseException: Si l'utilisateur
courant ne possède pas le droit de création des groupes
        */

        public function supprimer()
        {}




        /*
            - Renvoie la liste de tous les groupes éxistants:
            - @returns tableau de : {
                id_groupe           : integer,
                nom_groupe          : integer,
                nom_droits          : string,
                nombre_sous_groupes : integer (Les sous_groupes des
sous groupes sont inclus),
                nombre_utilisateurs : integer (Les utilisateurs des
sous groupes sont inclus)
            }
            - @Throws PDOException
        */

        public static function getListeGroupes()
        {}


        /*
            -Crée un groupe
            -@param nom_groupe: Le nom du nouveau groupe
            -@param nom_droits: Le nom du droit associé

            -@returns null
            -@Throws PDOException: Si l'insertion a échouée
```

```
            -@Throws NonAutoriseException: Si l'utilisateur courant
ne possède pas le droit de création des groupes
        */

        public static function ajouterGroupe($nom_groupe,
$nom_droits)
        {}
```

➔ Mail

```
        /*
            -Instancie un email.
            -@param id_mail : l'identifiant du mail
            -@Throws PDOException
            -@Throws ElementIntrouvable: Si aucun mail ne porte cet
identifiant
        */
        public function __construct($id_mail)
        {}


        /*
            -Récupère les détails d'un mail
            -@returns data : {
                id_utilisateur : integer,
                prenom_utilisateur: string,
                nom_utilisateur: string,
                pseudo_utilisateur: string,
                sujet_mail : string,
                message_mail: string
            }
            -@Throws PDOException : Possible seulement si c'est le
premier appel de la fonction
        */
        public function detailsMail()
        {}

        /*
```

```
            -Renvoie la liste des utilisateurs destinataires de ce
mail
            -@return liste de {
                id_utilisateur: integer,
                nom_utilisateur: string,
                prenom_utilisateur: string,
                pseudo_utilisateur: string
            }
            -@Throws PDOException : Possible seulement si c'est le
premier appel de la fonction
        */

        public function getUtilisateursDestinataire(){}


        /*
            -Renvoie la liste des groupes destinataires du mail
            -@returns tableau de  :{
                id_groupe: integer,
                nom_groupe: string
            }
            -@Throws PDOException : Possible seulement si c'est le
premier appel de la fonction
        */
        public function getGroupesDestinataires(){}

        /*
            -Ajoute les utilisateurs destinataires du mail
            -@param liste_utilisateurs: la liste des identifiants
des utilisateurs à ajouter
            -@Throws PDOException
        */

        private function
ajouterDestinataireUtilisateur($liste_utilisateurs){}


        /*
            -Ajoute les groupes destinataires du mail
```

```
            -@param liste_groupes: la liste des identifiants des
groupes à ajouter
            -@Throws PDOException
        */
        private function
ajouterDestinataireGroupe($liste_groupes){}



        /*
            -Supprime le mail courant
            -@Warning : Dès que le mail a été supprimé, tous les
destinataires ne pourront plus le consulter
            -@Throws PDOException
        */
        public function supprimerMail() {}


        /*
            -Récupère la liste des mails reçus
            -@returns tableau de : {
                id_mail : integer,
                sujet_mail: string,
                message_mail: string,
                pieces_jointe_mail: string,
                date_envoi_mail : date,
                date_lecture_mail: date,
                id_utilisateur : integer,
                nom_utilisateur: string,
                prenom_utilisateur: string,
                pseudo_utilisateur: string
            }
            -@Throws PDOException
            -@Throws NonConnecterException: Si aucun utilisateur
n'est connecté
        */

        public static function liste_mails_recus() {}

        /*
```

```
            -Récupère la liste des mails envoyés
            -@returns tableau de : {
                id_mail : integer,
                sujet_mail: string,
                message_mail: string,
                pieces_jointe_mail: string,
                date_envoi_mail : date,
                date_lecture_mail: date,
                id_utilisateur : integer,
            }
            -@Throws PDOException
            -@Throws NonConnecterException: Si aucun utilisateur
n'est connecté
        */

        public static function liste_mails_envoyes() { }

        /*
            -Envoie un mail
            -@param id_expediteur integer:  Identifiant de
l'utilisateur expediteur
            -@param sujet_mail string: Le sujet du mail
            -@param message_mail string: Le message du mail
            -@param lien_piece_jointe string: Le chemin vers la
pièce jointe
            -@param liste_utilisateurs_destinataires int[]: Tableau
des identifiants des utilisateurs destinataires
            -@param liste_groupes_destinataire int[]: Tableau des
identifiants des groupes destinataires

            -@Throws PDOException
            -@Throws NonConnecterException: Si aucun utilisateur
n'est connecté
        */


        public static function envoyer_mail( $sujet_mail,
$message_mail, $lien_piece_jointe,
$liste_utilisateurs_destinataire, $liste_groupes_destinataire)
        {}
```

➔ Module

```
        /*
            - Instancie un module
            - @param ref_module: La référence du module à
instancier
            - @Throws ElementIntrouvable: Si aucun module ne porte
cette référence
        */
        public function __construct($ref_module)
        {}


        /*
            - Retourne la liste des enseignants appartenant à ce
module
            - @returns tableau de : {
                    id_enseignant: integer,
                    pseudo_utilisateur: string,
                    nom_utilisateur: string,
                    prenom_utilisateur: string,
                    est_responsable: boolean
            }
            -@Throws PDOException
        */
        public function getEnseignantsModule()
        {}


        /*
            -Retourne la liste des enseignants qui n'enseignent pas
dans ce module
            -@returns tableau de : {
                    id_enseignant: integer,
                    pseudo_utilisateur: string,
                    nom_utilisateur: string,
                    prenom_utilisateur: string
            }
```

```
            -@Throws PDOException
            -@Throws NonAutoriseException: Si l'utilisateur ne
possède pas le droit de création des modules.
        */
        public function getEnseignantsAAjouter(){}

        /*
            -Récupère les détails relatifs à un module:
            -@return data: {
                ref_module: string,
                nom_module: string,
                coefficient_module: double,
                heures_cm_module: integer,
                heures_td_module: integer,
                heures_tp_module: integer,
                couleur_module: string,
                abreviation_module: string
            }
            -@Throws PDOException
        */

        public function getDetailsModule()
        {}


        /*
            - Modifie un module

            - @param $nom String        : Le nom du module
            - @param $coef double       : Le coefficient du module
            - @param $heures_cm integer : Le nombre des heures de
cours magistraux (CM)
            - @param $heures_tp integer : Le nombre des heures de
travaux pratiques (TP)
            - @param $heures_td integer : Le nombre des heures de
travaux dirigés (TD)
            - @param $couleur String    : La couleur du module en
hex
            - @param $abreviation String: Le nom du module
```

```
            -@Throws PDOException
            -@Throws NonAutoriseException: Si l'utilisateur courant
n'a pas le droit de création des modules
        */
        public function modifierModule($nom, $coef, $heures_cm,
$heures_tp, $heures_td, $couleur, $abreviation)
        {}


        /*
            -Retire un enseignant de la liste des enseignants
            -@param id_enseignant: L'identifiant de l'enseignant à
retirer
            -@throws PDOException
            -@Throws NonAutoriseException: Si l'utilisateur courant
n'a pas le droit de création des modules
        */

        public function retirerEnseignant($id_enseignant)
        {}


        /*
            -Ajoute un enseignant à la liste des enseignants
            -@param id_enseignant: L'identifiant de l'enseignant à
ajouter
            -@param est_responsable Boolean: Si l'enseignant est
responsable ou pas de ce module
            -@throws PDOException
            -@Throws NonAutoriseException: Si l'utilisateur courant
n'a pas le droit de création des modules
        */

        public function ajouterEnseignant($id_enseignant,
$est_responsable){}


        /*
            -Supprime ce module
```

```
            -Retire aussi tous les enseignants responsables de ce
module.
            -@throws PDOException
            -@Throws NonAutoriseException: Si l'utilisateur courant
n'a pas le droit de création des modules
        */

        public function supprimerModule(){}



        /*
            -Créer un module

            - @param $nom String         : Le nom du module
            - @param $coef double        : Le coefficient du module
            - @param $heures_cm integer : Le nombre des heures de
cours magistraux
            - @param $heures_tp integer : Le nombre des heures de
travaux dirigés
            - @param $heures_td integer : Le nombre des heures de
travaux pratiques
            - @param $couleur String     : La couleur du module en
hex
            - @param $abreviation String: Le nom du module

            -@throws PDOException         : Si l'insertion a échoué
("Si un module porte déjà la même référence").
            -@Throws NonAutoriseException: Si l'utilisateur courant
n'a pas le droit de création des modules
        */

        public static function ajouterModule($ref, $nom, $coef,
$heures_cm, $heures_tp, $heures_td, $couleur, $semestre,
$abreviation)
        {}



        /*
            -Récupère la liste des modules disponibles
```

```
            -@return tableau de data: {
                ref_module: string,
                nom_module: string,
                coefficient_module: double,
                heures_cm_module: integer,
                heures_td_module: integer,
                heures_tp_module: integer,
                couleur_module: string,
                abreviation_module: string
            }
            -@Throws PDOException

        */
        public static function listeModules()
        {}

        /*
            -Retourne l'ensemble des modules enseignés par un
enseignant
            -@param id_enseignant: L'identifiant de l'enseignant
dont on veut récupérer les modules
            -@Throws PDOException
        */

        public static function modulesEnseignant($id_enseignant){}
```

➔ Moodle

```
    /*
        - Instancie un dépôt
        - @param id_depot: L'identifiant du dépôt à instancier
        - @Throws ElementIntrouvable : Si le dépôt n'existe pas ou
s'il appartient à un autre enseignant
        - @Throws PDOException
    */
    public function __construct($id_depot){}

    /*
        -Récupère l'ensemble des dépôts des étudiants
```

```
        -@return tableau de : {
            pseudo_utilisateur      : string,
            num_etudiant            : string,
            commentaire_etudiant    : string,
            commentaire_enseignant  : string,
            id_depot_exercice       : integer,
            note_depot              : float
        }
        -@Throws PDOException: Si la récupération de la liste a
échoué.
    */

    public function depots_etudiants(){}


    /*
        -Récupère le dépôt de l'étudiant
        -@param id_etudiant: L'identifiant de l'étudiant
        -@return tableau de : {
            commentaire_depot_etudiant : string,
            date_depot_etudiant        : string,
            lien_depot_etudiant        : string,
            num_etudiant               : string,
            id_depot_exercice          : integer
        }

        -@Throws PDOException
    */
    public function getDepotEtudiant($id_etudiant){}

    /*
        -Modifie le commentaire ou la note d'un dépôt étudiant
        -@param id_etudiant : L'identifiant de l'étudiant voulu.
        -@param note integer: La nouvelle note de l'étudiant.
        -@param commentaire string: Le commentaire de l'enseignant
sur le dépôt étudiant
        -Cette fonction ne produit aucun résultat si le dépôt de
l'étudiant n'existe pas
        -@Throws PDOException
    */
```

```
   public function modifierNote($id_etudiant, $note,
$commentaire){}

   /*
       -Ajoute un support de cours (Lors d'un dépôt de cours).
       -@param nom_support: Le nom qui apparaîtra sur le support.
       -@param lien: le lien vers le dépôt sur le disque
       -@param date_ouverture: La date de début d'ouverture du
dépôt
       -@param est_caché: Si oui, alors le dépôt ne sera pas
téléchargeable pour les étudiants
       -@param module: Le module concerné par le dépôt

       -@Throws PDOException: Si l'enseignant n'est pas
responsable du module
       -@Throws PasEnseignantException: Si l'utilisateur courant
n'est pas un enseignant
   */
   public static function ajouter_support($nom_support, $lien,
$date_ouverture, $est_cache, $module){}


   /*
       - Renvoie la liste des supports pédagogiques déposés par
l'enseignant courant
       - @Throws PDOException.
       - @Throws PasEnseignantException: Si l'utilisateur actuel
n'est pas un enseignant.
   */

   public static function liste_supports(){}


   /*
       -Renvoie la liste des dépôts ouvert par l'utilisateur
actuel
       -@return tableau de data: {
               id_groupe                       : integer,
               id_support                      : integer,
```

```
                ref_module                      : string,
                groupe_depot                    : integer,
                nom_groupe                      : string,
                lien_fichier_support            : string,
                date_depot_support              : date,
                date_ouverture_support          : date,
                support_est_cachee              : boolean,
                nb_consultation_support         : integer,
                date_debut_depot_exercice       : timestamp,
                date_fermeture_depot_exercice   : timestamp,
                coefficient_depot               : double
        }
        -@Throws PDOException
        -@Throws PasEnseignantException: Si l'utilisateur courant
n'est pas un enseignant
    */
    public static function liste_depots(){}


    /*
        -Permet l'ouverture d'un dépôt

        -@param string nom_depot        : Le nom que portera le
dépôt, ainsi que le support lors de son téléchargement.
        -@param string module           : La référence du module
        -@param integer groupe_depot    : L'identifiant du groupe
auquel est destiné ce dépôt.
        -@param path    lient_depot     : Le chemin vers le support
d'exercice sur le disque
        -@param timestamp date_debut    : La date où le date sera
ouvert pour que les étudiants puissent déposer.
        -@param timestamp date_fermeture: La date après laquelle
aucun étudiant ne pourra encore déposer.
        -@param date    date_ouverture  : La date après laquelle le
dépôt sera visible.
        -@param double  coefficient     : Le coefficient du dépôt,
0 si le dépôt n'est pas noté

        -@throws PDOException: Si l'ouverture du dépôt a échoué.
```

```
       -@throws PasEnseignantException: Si l'utilisateur courant
n'est pas un enseignant.
   */

   public static function
ouvrir_depot($nom_depot,$module,$groupe_depot, $lien_depot,
$date_debut, $date_fermeture, $date_ouverture, $coefficient){}
```

➔ Salle

```
     /*
       - Instancie une salle
       - @param nom_salle: le nom de la salle
       - @Throws PDOException
       - @Throws ElementIntrouvable : Si aucune salle ne porte ce
nom
     */
     public function __construct($nom_salle)
     {}



     /*
       - Renvoie les détails de la salle
       - @return {
              nom_salle : string,
              nombre_ordinateurs_salle: integer,
              nombre_places_salle     : integer,
              contient_projecteur_salle:boolean
       }
       -@Throws PDOException
     */
      public function detailsSalle()
     {}


       /*
           - Supprime une salle
           - @Throws PDOException
```

```
            - @Throws NonAutoriseException : Si l'utilisateur
courant ne possède pas le droit création cours
     */
     public function supprimerSalle() { }

   /*
       -Modifie la salle courante

       -@param nb_pc: nombre d'ordinateurs dans la salle
       -@param nb_places: nombre de tables dans la salle
       -@param contient_projecteur: Vrai si la salle contient un
vidéo-projecteur

       -@Throws PDOException
       -@Throws NonAutoriseException : Si l'utilisateur ne possède
pas le droit de création des cours

   */
   public function modifierSalle($nb_pc, $nb_places,
$contient_projecteur)
     { }


     /*
       -Renvoie la liste des salles
       -returns tableau de : {
           nom_salle: string,
           nombre_ordinateurs_salle: integer,
           nombre_places_salle     : integer,
           contient_projecteur_salle:boolean
       }
       -@Throws PDOException
     */

     public static function liste_salles() {}


   /*
       -Ajoute une salle
```

```
        -@param nom: Le nom de la salle
        -@param nb_pc: nombre d'ordinateurs dans la salle
        -@param nb_places: nombre de tables dans la salle
        -@param contient_projecteur: Vrai si la salle contient un
vidéo-projecteur

        -@Throws PDOException
        -@Throws NonAutoriseException : Si l'utilisateur ne possède
pas le droit de création des cours

    */
    public static function ajouter_salle($nom, $nb_pc, $nb_places,
$contient_projecteur)
    {}
```

➔ Semestre

```
    /*
        - Instancie un semestre
        - @param ref_semestre: le semestre que l'on veut instancier
        - @Throws PDOException
        - @Throws ElementIntrouvable: Si aucun semestre ne porte
cette référence
    */
    public function __construct($ref_semestre)
    {}


    /*
        - Renvoie les détails du semestre
        - @return data: {
            ref_semestre        : string,
            nom_semestre        : string,
            points_ets_semestre : integer,
            periode_semestre    : 1 ou 2, (Sa période dans l'année)
        }
        - @Throws PDOException
    */
    public function detailsSemestre()
```

```
    {}

    /*
      - Renvoie la liste des années où s'est déroulé ce semestre.
      - @returns tableau de : {
            annee           : string (date_debut => date_fin),
            moyenne         : float,
            nombre_reussite : integer,
            nombre_echecs   : integer,
            taux_reussite   : double,
            taux_echecs     : double
          }
       - @Throws PDOException
       - @Throws NonAutoriseException : Si l'utilisateur ne
possède pas le droit creation modules
    */
    public function anneesSemestre()
    {}

  /*
      - Renvoie la liste des étudiants qui ont passé ce semestre.
      - @returns tableau de : {
            annee               : string (date_debut =>
date_fin),
            num_etudiant        : integer,
            nom_utilisateur     : string,
            prenom_utilisateur  : string,
            moyenne             : double
          }
       - @Throws PDOException
       - @Throws NonAutoriseException : Si l'utilisateur ne
possède le droit visualisation statistique
  */
    public function etudiantsSemestre()
    {}

    /*
      - Retourne la liste des modules associés à ce semestre
      - @returns tableau de :{
          ref_module: string,
```

```
            ref_semestre: string,
            nom_module: string,
            couleur_module: string,
            abreviation_module: string
        }
        -@Throws PDOException
    */

    public function modulesSemestre(){}

    /*
        - Renvoie la liste des groupes associés à ce semestre.
        - @returns un tableau de :{
            id_semestre : L'identifiant du groupe ayant le même nom
que la référence du semestre,
            id_groupe   : L'identifiant du groupe fils,
            nom_groupe  : Le nom du groupe fils
        }
        - @Throws PDOException
    */

    public function groupes_semestre(){}




    /*
        - Modifie un semestre
        - On ne peut modifier que le nom ou les points ets du
semestre.
        - @param nom : string
        - @param pts_ets integer : Les points ets du semestre
        - @Throws PDOException : Si la modification a été refusée
        - @Throws NonAutoriseException : Si l'utilisateur courant
ne possède pas le droit droits_creation_modules
    */
    public function modifierSemestre($nom, $pts_ets)
    {}
```

```
    /*
      - Retire un étudiant de la liste des étudiants du semestre
      - On ne peut retirer que les étudiants de l'année courante
      - @param num_etudiant integer: Le numéro de l'étudiant à
retirer
      - @Throws PDOException : Si la modification a été refusée
      - @Throws NonAutoriseException : Si l'utilisateur courant
ne possède pas le droit droits_creation_modules
    */

    public function retirerEtudiant($num_etudiant)
    {}


    /*
      - Supprime ce semestre
      - @Throws PDOException : Si la suppression a été refusée en
raison de contraintes d'intégrité
      - @Throws NonAutoriseException : Si l'utilisateur courant
ne possède pas le droit droits_creation_modules
    */

    public function supprimerSemestre()
    {}

    /*
      - Renvoie la liste des semestres
      - Attention, cette function retourne plus de détails que
prévu
      - @returns {
            detailsSemestre : Comme détails semestre,
            nombre_reussite : integer,
            nombre_echoue   : integer
        }
      - @Throws PDOException
    */

    public static function liste_semestres()
    {}
```

```
    /*
      - Ajoute un semestre
      - @param ref: La référence du nouveau semestre
      - @param nom : string
      - @param pts_ets integer : Les points ets du semestre
      - @param periode         : La période de l'année dans
laquelle va se dérouler ce semestre
      - @Throws PDOException : Si l'insertion a échoué pour des
raisons d'invalidité du semestre
      - @Throws NonAutoriseException : Si l'utilisateur courant
ne possède pas le droit droits_creation_modules
      - @Throws ParametresIncorrectes: Si la période est
différente de 1 et 2.
    */


    public static function ajouter_semestre($ref, $nom,
$points_ets, $periode)
    {}
```

Architecture MVC :

L’architecture MVC (Modèle-Vue-Contrôleur) permet de bien architecturer le site web basé sur des interactions avec les utilisateurs. La logique du site web, l’affichage et la gestion des actions utilisateurs sont tous bien séparés et interagissent entre eux.

➢ Modèle:

Élément qui contient les données ainsi que de la logique en rapport avec les données: validation, lecture et enregistrement. Dans ce site web, le modèle fait appel aux classes principales et gères les erreurs qui peuvent résulter lors de l’appel des méthodes de ces classes.

➢ Contrôleur:

Le contrôleur permet de récupérer les données brutes envoyées par un utilisateur. S’il manque un paramètre ou si l’utilisateur n’envoie pas de token, alors la requête est rejetée et un message d’erreur est affiché. Le contrôleur ne doit pas gérer les exceptions provenant des classes principales.

➢ Vue:

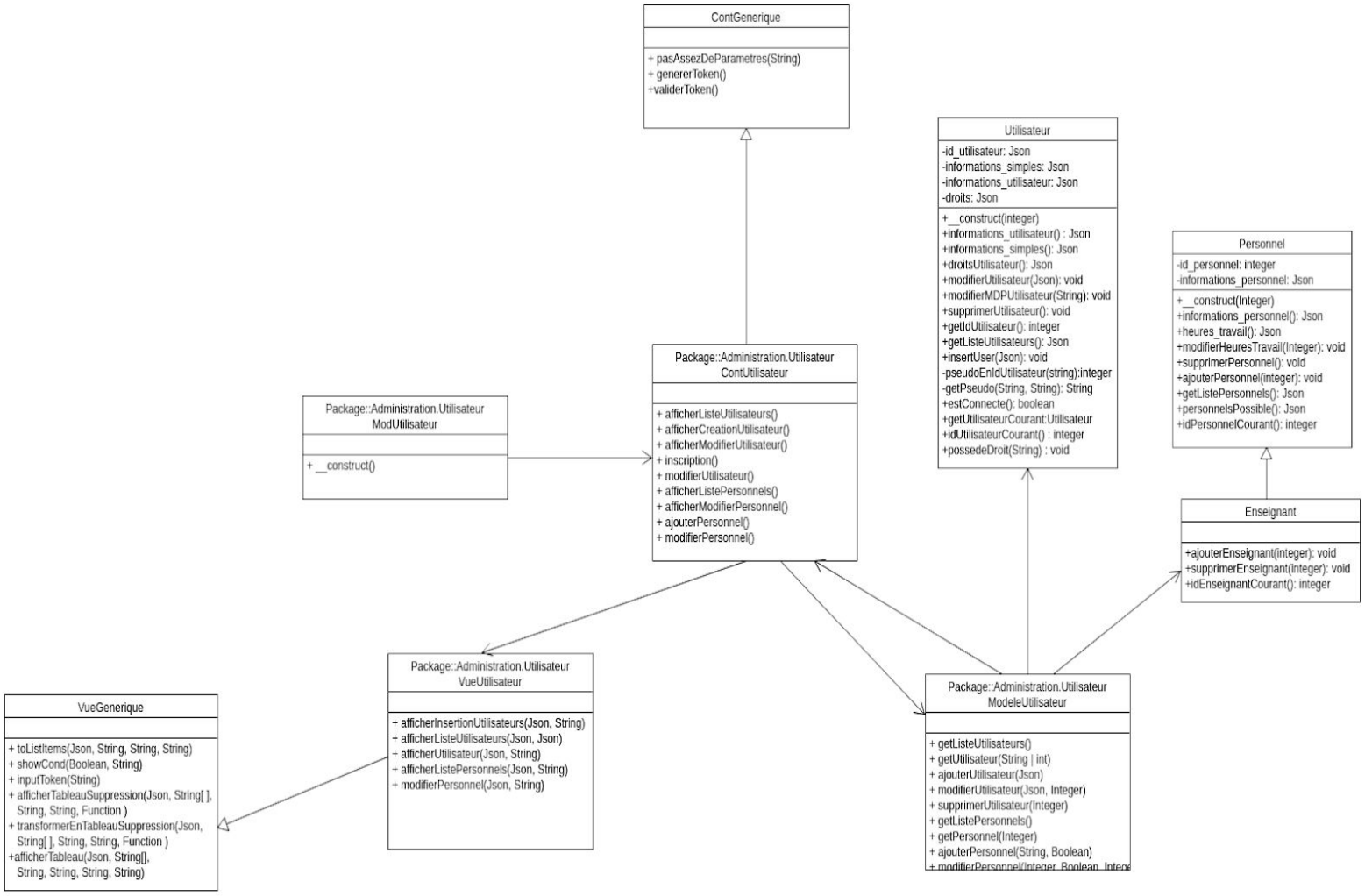
La vue représente tout ce qui est visuel. Elle ne s’occupe que de l’affichage des éléments récupérés par le contrôleur et ne gère pas les cas d’erreur relatifs à la logique d’application.

➢ Mod:

Permet d’interpréter la demande du client et appelle la méthode relative à la demande du client. C’est ici où sont indiqués les redirections par défaut si l’utilisateur exécute une action indéfinie.

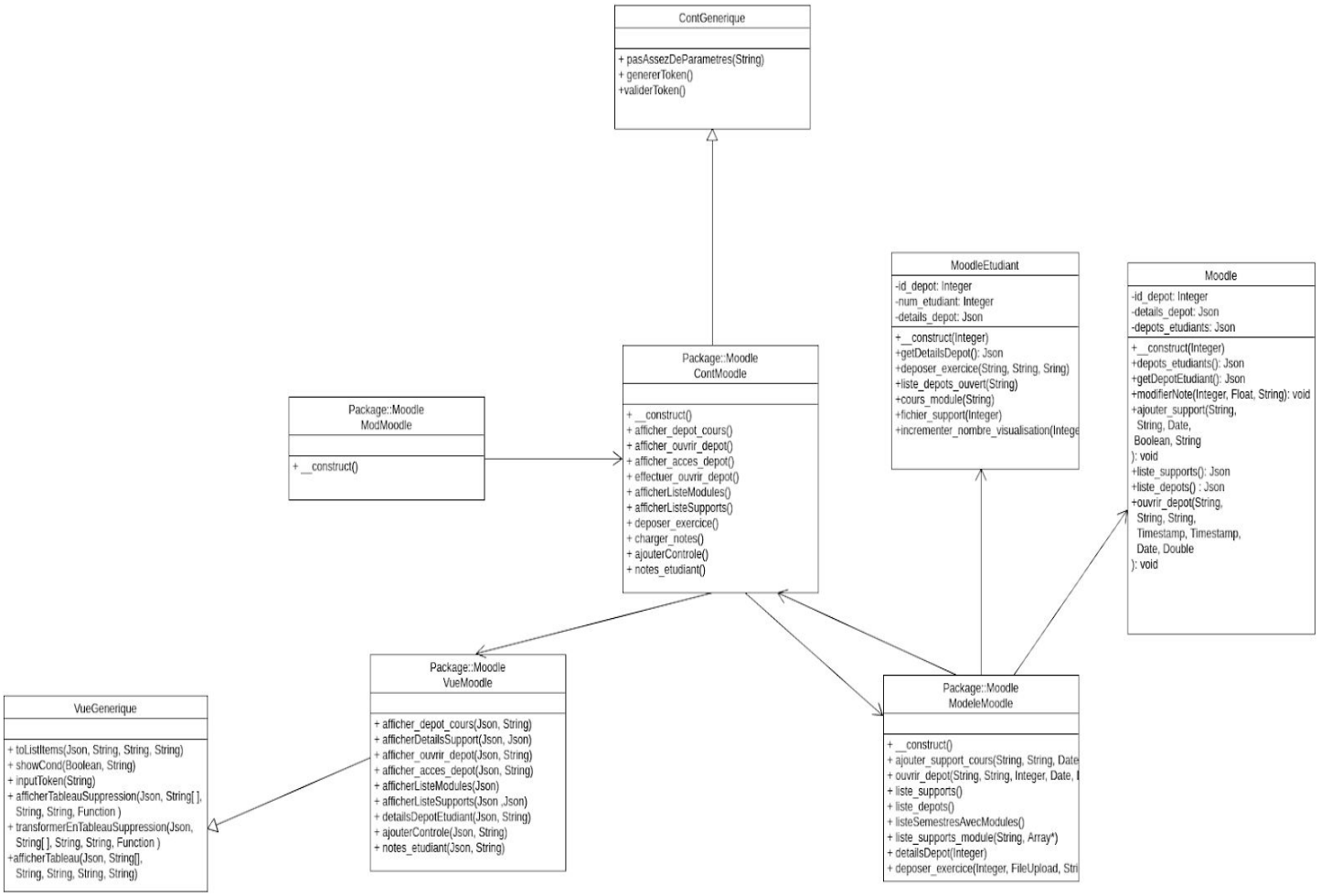
➢ Exemples de MVC: Diagrammes de classe

Utilisateur



Vous pouvez télécharger l'image sur le lien : https://ibb.co/8zwRDQJ

Moodle



Vous pouvez télécharger l'image sur le lien : https://ibb.co/TTXzxJ1

API Rest (AJAX) :

Dans ce site web, les API permettent de récupérer des données brutes sans aucune balise html. Cela facilite énormément l'extraction des données. Les API assurent plusieurs fonctionnalités telles que :

1. Le téléchargement des fichiers.
2. L'autocomplete suite à une saisie de l'utilisateur
3. Le déplacement des séances sur l'emploi de temps

Les API se trouvent dans le répertoire API et sont composés de deux parties:

1. ModAPI: Permet d'interpréter la demande de client et appelle le modèle et la méthode correspondants. Il s'assure aussi que tous les paramètres requis sont fournis.
2. Modèle API: Communique avec les classes principales. Il permet de récupérer les données de la base de données et de gérer les exceptions éventuelles.