Institut Universitaire de Technologie
Aix-Marseille Université

Introduction à CodeIgniter

BUT Informatique à Arles

Ricardo Uribe Lobello

CodeIgniter

CodeIgniter est un framework pour le développement d'applications web.

Ses principales caractéristiques sont :

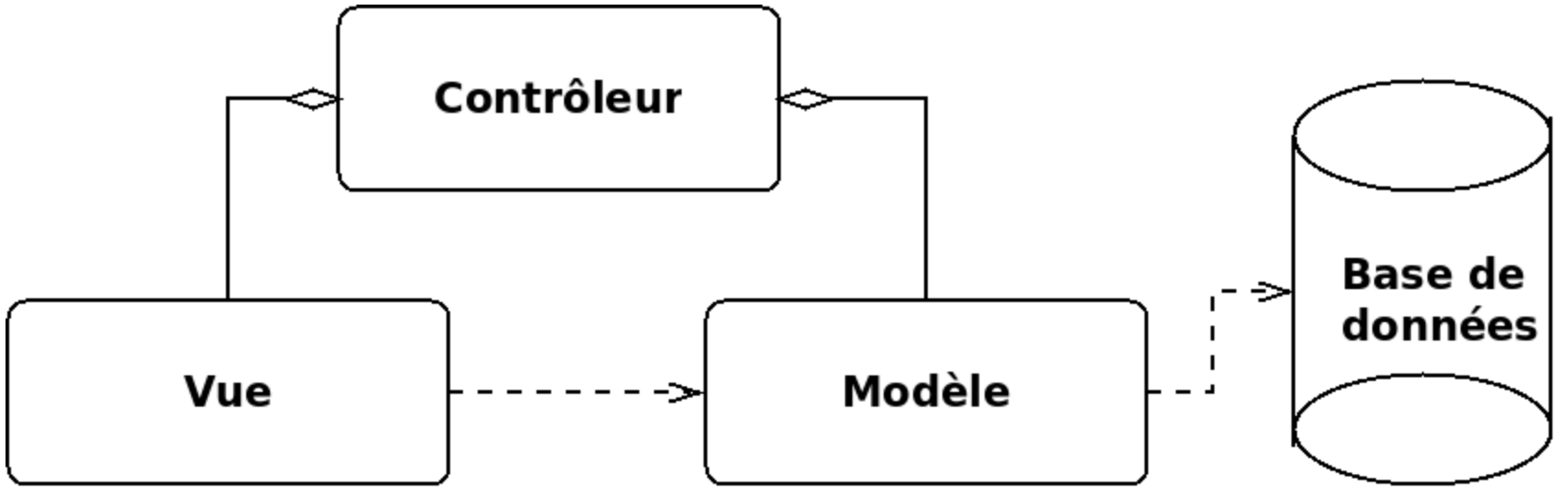
- Il est écris en PHP ;
- Il est structuré en suivant le pattern Modèle Vue Contrôleur ;
- Il fourni des librairies contenant des fonctionnalités qui sont nécessaires dans la plus grande partie de développements Web ;
- Il possède une couche de données permettant de faciliter la manipulation des données issues des bases de données relationnelles ;
- Il peut utiliser des moteurs de template permettant de mieux structurer la vue dans une application ;

Avantages de CodeIgniter

- Il est rapide et plutôt compacte (en quantité de fichiers nécessaires) ;
- La configuration nécessaire pour le faire fonctionner est minimale ;
- Il n’est pas nécessaire d’utiliser l’invité de commandes ;
- Il est souple et n’oblige pas le programmeur à utiliser des règles de programmation trop strictes ;
- Il n’est pas obligatoire d’utiliser un moteur de templates mais il est toujours possible ;
- Il est très bien documenté.

Le pattern MVC

La structure générale du pattern Modèle Vue Contrôleur est présentée ci-bas :



Le pattern MVC en codeIgniter

Les différents composant du pattern Modèle Vue Contrôleur en codeIgniter sont :

- **Views :** Ce sont des fichiers contenant surtout du HTML/CSS/JavaScript et très peu de PHP. Le PHP doit surtout être utilisé pour l'affichage des variables ou pour parcourir des ensembles de données ;
- **Models :** Ce sont des classes permettant d'accéder aux données. Il doit avoir une classe pour chaque type de donnée (table dans la base de données). Elles sont surtout utilisées pour retrouver et appliquer des contraintes aux données ;
  - Le modèle peut être complété par des **Entities** qui sont des classes correspondantes au modèles et qui implémentent la couche de logique de l'application.
- **Contrôleurs :** Ce sont des classes pour recevoir et rediriger les requêtes de l'utilisateur. Elles sont aussi responsables d'envoyer les réponses au navigateur.
  - Elles peuvent aussi gérer la sécurité, l'encodage, l'authentification, etc.

Structure d'une application CodeIgniter

Il s'agit d'un répertoire racine contenant 5 sous-répertoires :

- **app :** il contient les fichiers PHP de l'application ;
- **public :** il contient tous les fichiers accessibles via un navigateur web ;
  - Cette séparation app/public permet de protéger les fichiers de code.
- **writable :** contiendra les répertoires et fichiers qui pourront être écris :
  - Les fichiers déposés sur le serveur seront initialement stockés ici.
- **System :** Il contient les fichiers implémentant les différentes parties du framework. **Attention :** Les fichiers de ce répertoire ne doivent pas être modifiés.
- **tests :** contiendra les fichiers implémentant les tests de votre application. CodeIgniter propose déjà certains fichiers de base.

Le répertoire app (1/2)

C’est le répertoire contenant les fichiers de votre application, il contient les sous-répertoires suivants :

- **Config:** contiendra les fichiers de configuration ;
- **Controllers :** contient les classes agissant comme contrôleurs de l’application ;
- **Models :** Contient les classes permettant l’accès aux bases de données ;
- **Views :** contient les templates utilisés pour générer les fichiers HTML à envoyer au navigateur de l’utilisateur ;
- **Filters :** contient des classes permettant de filtrer les requêtes de l’utilisateur ;
- **Entities :** Contient les classes faisant partie intégrale de la couche logique de votre application.

Le répertoire app (2/2)

Le répertoire app contient aussi les répertoires suivants :

- **Helpers :** contient des collections des fonctions utiles dans le développement web ;
- **Database :** contient les fichiers de migration pour des bases de données ;
- **Langage :** contient des fichiers permettant une configuration multi-langue de l'application ;
- **Librairies :** contient des classes utiles mais qui ne font pas partie d'une autre catégorie ;
- **ThirdParty :** ce sont des librairies ne faisant partie intégrante de CodeIgniter mais qui peuvent être ajoutées à l'application.

Installation de CodeIgniter

Il est possible de le faire de deux manières :

- **Composer** : en utilisant l'outil composer via la ligne de commandes :
  - **Avantages** : les dépendances sont gérées ;
  - **Problèmes** : Il faut apprendre aussi à utiliser composer et avoir la bonne version.
- **Manuellement** : il est possible de télécharger le dossier de CodeIgniter et de le placer directement dans le répertoire htdocs :
  - **Avantages** : avec un copier coller, il est possible de commencer à travailler.
  - **Inconvénients** : il faut gérer toutes les dépendances si nous voulons ajouter des nouveaux packages.

Installation avec composer

Il faut installer composer version 2 au moins pour CodeIgniter 4. Sur Ubuntu :

- Exécuter : « **curl -sS https://getComposer.org/installer -o composer-setup.php** »
- Exécuter : « **sudo php composer-setup.php --install-dir=/usr/local/bin/ --filename=composer** »
- Vérifier la version de composer avec : « **composer –version** »
- Aller dans le répertoire htdocs et saisir :
  - **composer create-project codeigniter4/appstarter <nom_du_projet>**

Configuration initiale

Avant de commencer à travailler, il faut réaliser quelques manipulations initiales :

- Changer le nom du fichier <nom du projet>/env à <nom du projet>/**.env ;**
- Éditer le fichier <nom du projet>/.env pour dé-commenter et changer la ligne :

  # CI_ENVIRONMENT = production

  Par

  CI_ENVIRONMENT = development // Cette ligne habilite l'affichage des erreurs.

- Et la ligne :

  # app.baseURL = ''

  Par le URL de base de votre projet. Ce sera l'adresse URL à utiliser dans le navigateur :

  app.baseURL = 'http://localhost/<chemin de votre répertoire public dans le projet>'

Le système de routage

Création des routes URI

Il s’agit de définir les routes URL qui seront utilisées dans le site web et de les associer aux méthodes se trouvant dans les contrôleurs.

Elles vont nous permettre de cacher partiellement la structure de fichiers de notre application web.

Il y a deux types :

- **Routes définies manuellement** : Ce sont des segments de texte dans l’url qui seront associés aux méthodes dans des contrôleurs.
- **Autorouting :** Nouvelle fonctionnalité permettant de relier un ensemble de segments dans l’URL à un namespace, une classe de type contrôleur et une méthode publique dans cette classe.

Les routes manuelles

Ce sont des segments de texte dans l'url qui seront associés aux méthodes dans des contrôleurs.

- Ces routes sont spécifiées dans le fichier **app/Config/Routes.php ;**
- Elles associent explicitement un URL avec une classe de type contrôleur et une méthode publique de cette classe ;
- Il faut spécifier le type de requête pour laquelle ils seront actives, GET, POST, PUT, DELETE, etc.
- La méthode codeIgniter add() peut être utilisée pour ajouter une route. Elle permet de recevoir tous les types de requêtes.
- Ils ont la forme suivante, pour des requêtes utilisant la méthode GET :

$routes->get('/', 'Home::index'); // L'adresse racine de notre site web exécutera la méthode index dans la classe contrôleur Home.

Création des routes

Il existe plusieurs manières de rediriger un URL vers une méthode dans un contrôleur. Par exemple :

```
$routes->get('/consulter', 'Home');
```

Exécutera la méthode par défaut, la méthode **index** dans la classe Home.

En créant la route suivante :

```
$routes->get('/consulter/(:any)', 'Home::consulter');
```

Toutes les routes commençant par **consulter/** seront redirigées vers la méthode **consulter** dans le contrôleur Home.

Utilisation des namespaces dans des routes

Les contrôleurs peuvent être organisés dans des **namespaces**. Par exemple :

```
$routes->get('gestion', '\Gestion\Home::index');
```

Exécutera la méthode consulter dans la classe Home se trouvant dans le **namespace Gestion**.

La même opération peut être réalisée en utilisant la syntaxe suivante :

```
$routes->get('Gestion', 'Home::index', ['namespace' => 'Gestion']);
```

Les **namespaces** correspondent à des répertoires dans le dossier Controlers de CodeIgniter.

Passage de paramètres

Il est possible de passer des paramètres aux méthodes du contrôleur en utilisant la route, par exemple :

```
$routes→get('/consulter/(:num)', 'Home::consulter/$1');
```

Exécutera la méthode consulter dans la classe Home en passant la valeur donnée après consulter comme paramètre. Dans ce cas, la donnée devrait être un nombre.

En créant la route suivante :

```
$routes→get('/consulter/(:any)', 'Home::consulter/$1');
```

Toutes les routes commençant par consulter/ seront redirigées vers la méthode consulter dans Home et la méthode consulter recevra au moins un paramètre mais ils peuvent être plus. Ce sera à la méthode consulter de s'assurer de tous pouvoir les recevoir.

Types de segments disponibles

- **(:any) :** Corresponde à tous les segments URI jusqu'à la fin de l'URL. Ils peuvent être plusieurs segments d'URI (séparés par des « / ») ;
- **(:segment)** : Corresponde à un seul segment d'URI. Entre deux caractères « / » ;
- **(:num)** : Correspondra à un segment représentant un nombre ;
- **(:alpha)** : Corresponde à n'importe quel segment contenant des caractères alphabétiques ;
- **(:alphanum)** : Correspondra à n'importe quel segment contenant des caractères alphanumériques.

Passage de paramètres

Par exemple, la route :

```
$routes->get('Etudiant/(:num)/(:num)', [Etudiant::class, 'index']);
```

Et la route :

```
$routes->get('Etudiant/(:num)/(:num)', 'Etudiant::index/$1/$2');
```

Correspondent au même méthode qui sera appelée et qui recevra deux arguments de type numérique.

Il est possible de changer l'ordre de paramètres de la manière suivante :

```
$routes->get('Etudiant/(:num)/(:num)', [[Etudiant::class, 'index'], '$2/$1']);
```

Ou

```
$routes->get('Etudiant/(:num)/(:num)', 'Etudiant::index/$2/$1');
```

Regroupement de routes

Le mécanisme de regroupement de routes permet de préfixer un ensemble de routes avec un ou plusieurs segments URI.

Par exemple :

```php
<?php

$routes->group('admin', static function ($routes) {

    $routes->get('etudiant', 'Admin\Etudiant::index');

    $routes->get('intervenant', 'Admin\Intervenant::index');

});
```

Permet de préfixer les routes Etudiant et Intervenant de tel manière quelles seront reconnues comme admin/etudiant et admin/intervenant.

Regroupement des groupes de routes

Il est possible d'imbriquer la structuration des routes si cela est nécessaire.

Par exemple :

```php
<?php
$routes->group('Admin', static function ($routes) {
    $routes->group('Etudiant', static function ($routes) {
        $routes->get('Consulter', 'Admin\Etudiant::consulter');
    });
});
```

Ce qui donnera la route Admin/Etudiant/Consulter.

Important : le groupement de routes deviendra particulièrement important avec l'utilisation des filtres.

Création des urls à partir des routes

Reverse Routing

Ce mécanisme permet d'utiliser la méthode **url_to()** pour générer un URI correspondant à la route fournie comme paramètre.

Par exemple :

```
<?php

$routes->get('Etudiant/(:alpha)/notes/(:alphanum)', 'Etudiant::afficherNotes/$1/$2');

?>

<!-- Génération de l'URI pour accéder aux notes de la ressource R1.01 de l'étudiant Uribe -->

<a href="<?= url_to('Eudiant::afficherNotes', 'Uribe', 'R1.01') ?>">Consulter notes</a>
```

Ce qui créera l'url nécesaire pour accéder à la route spécifiée.

Création des urls à partir des routes

Utilisation des routes nommées

Ce mécanisme permet d'utiliser la méthode **url_to()** avec un **nom de route** au lieu du qualificateur (Contrôleur::methode) pour générer un URI correspondant à la route fournie comme paramètre.

Par exemple :

```
<?php
$routes->get('Etudiant/(:alpha)/notes/(:alphanum)', 'Etudiant::afficherNotes/$1/$2',
            [ 'as' => 'ConsulterNotes']);
?>
<!-- Génération de l'URI pour accéder aux notes de la ressource R1.01 de l'étudiant Uribe -->
<a href="<?= url_to('ConsulterNotes', 'Uribe', 'R1.01') ?>">Consulter notes</a>
```

Ce qui créera l'url nécesaire pour accéder à la route spécifiée.

Les contrôleurs

Les contrôleurs

- Dans CodeIginiter , les contrôleurs sont les classes en charge de gérér les requêtes HTTP ;
- C’est le système de routage qui est le responsable de connecter les URIs dans l’URL avec les différentes méthodes dans les classes contrôleurs ;
- Il est possible de créer des nouvelles classes contrôleurs mais elles doivent toutes hériter de la classe **BaseController** ;
- Le nom de la classe contrôleur doit commencer impérativement par une lettre en majuscule et toutes les autres lettres doivent être en minuscule ;
- Chaque contrôleur possède un constructeur spécial **initController()** qui est appelé par CodeIgniter après l’exécution du constructeur classique de PHP, __construct().

La classe contrôleur

Chaque classe contrôleur a accès, via la classe BaseController, aux ressources suivants :

- L'objet requête via la variable $this->request ;
- L'objet réponse avec la variable $this->response ;
- Un objet log permettant d'ajouter des messages au log de l'application. Il faut utiliser la variable $this->logger ;
- Un ensemble de fichiers des classes Helpers en les déclarant comme un attribut tableau de la classe comme suit :

```
protected $helpers = ['url', 'form'] ;
```

Le contrôleur et les routes

Toutes les routes spécifiées dans le fichier app/Config/Routes.php doivent pointer vers une méthode dans un contrôleur. Par exemple :

La route :

```php
<?php

$routes->add('listerEtudiants', '\App\Controllers\
Etudiant::listerEtudiants');

?>
```

Fera référence à la méthode listerEtudiants() définie dans le contrôleur Etudiant.

Pour la méthode consulterEtudiant, il faudra créer la route :

```php
<?php

$routes→add('consulerEtudiant/(:num)', '\App\Controllers\
Etudiant::consulterEtudiant/$1');

?>
```

```php
<?php

namespace App\Controllers;


class Etudiant extends BaseController

{

    public function consulterEtudiant(string $NumEtu)

    {

        return 'Hello World!';

    }


    public function listerEtudiants()

    {

        return 'I am not flat!';

    }

}
```

Validation dans le contrôleur

La classe contrôleur a aussi accès à une méthode validate() qui permet d'implémenter une validation des informations fournies dans un requête provenant, par exemple, d'un formulaire. Par exemple :

La route :

```
<?php

$routes->post('changerEmail/(:alphanum)',
                    'Etudiant::changerEmail/$1') ;

?>
```

Exécutéra le code de la méthode changerEmail ci-contre.

NOTE : Le mécanisme de validation sera étudié plus en détail dans la section de validation de formulaires.

```
<?php
namespace App\Controllers;


class Etudiant extends BaseController
{
    public function changerEmail(string $identifiant)
    {
        if (! $this->validate(['email' => "required | is_unique[etudiant.email,id,{$identifiant}]",
                                'nom'  => 'required | alpha_numeric_spaces',
        ])) {
            return view('etudiant/changerEmail', [
                'errors' => $this->validator->getErrors(),
            ]);
        }
        // Ici vient le code de traitement du formulaire.
    }
}
```

La gestion des requêtes

- Les informations de la requête de l'utilisateur sont disponibles en utilisant la classe **IncomingRequest** de CodeIgniter. Cette classe hérite de la classe plus générale **Request**
- Dans un contrôleur, l'objet request est accessible en utilisant la propriété de la superclasse Basecontroller $this->request ;
- Si on ne se trouve pas à l'intérieur d'un contrôleur, il est possible de la récupérer en utilisant les services CodeIgniter :

```
$request = \Config\Services::request();
```

- Néanmoins, si nécessaire, il est préférable de faire passer le request à la classe concernée directement en utilisant un argument d'une fonction du type : **RequestInterface.**

Les types de requêtes

L'objet request permet d'avoir accès aux informations suivantes :

```php
<?php
//Type de requete avec la fonction is() qui retourne un boolean.
$request->is('get')
$request->is('post')
$request->is('ajax')
//A la méthode utilisé pour la requête :
$methode = $request->getMethod() ;          // Retourne 'get', 'post', etc.
```

Récupération de données

Il est possible de récupérer des informations provenant des superglobals $_SERVER, $_GET, $_POST et $_ENV.

Pour récupérer les valeurs fournis par un formulaire, il est possible d'utiliser :

```
<?php

// Pour n'importe quel type de requête. Le filter Sanitize permet d'éliminer tous les caractères illégaux d'une adresse e-mail. Il existe d'autres filtres.

$email = $this->request->getVar('Email', FILTER_SANITIZE_EMAIL);

$nom = $this->request->getGet('Nom');          // Pour une requête avec méthode GET.

$nom = $this->request->getPost('Nom');         // Pour une requête avec méthode POST.

$nom = $this->request->getCookie('Nom');       // Pour récupérer un cookie.

$fichier = $this->request->getServer('PHP_SELF');   // Pour récupérer le nom du fichier
            courant.
```

Récupération des informations de l'URL

Il est aussi possible de récupérer des informations concernant l'url du requête courant. Par exemple, pour l'url : http://www.univ-amu.fr/etudiants/info?Prom=INFO&Id=10

```php
<?php
$uri = $request->getUri();   // Récupération de l'URI.
echo $uri->getScheme();        // http
echo $uri->getPath();          // etudiants/info
echo $uri->getQuery();         // Prom=INFO&Id=10
echo $uri->getHost();          // www.univ-amu.fr
echo $uri->getPort();          // 80
echo $uri->getSegment(1);      // etudiants
echo $uri->getTotalSegments();  // 2
```

Institut Universitaire de Technologie
Aix-Marseille Université

L’implantation du modèle et l’accès aux bases de données

Configuration de la connexion

Il faut utiliser le fichier app/Config/Database.php

Il est possible d'avoir plusieurs configurations dans un seul projet ;

Les informations de base à fournir :

- **Hostname** : nom de la machine ;
- **Username :** généralement root ;
- **Password** : A la base vide '' ;
- **Database** : Nom de la base de données

```php
<?php
  public $default = [
    'DSN'      => '',
    'hostname' => 'localhost',
    'username' => 'root',
    'password' => '',
    'database' => 'nomBDD',
    'DBDriver' => 'MySQLi',
    'DBPrefix' => '',
    'pConnect' => true,
    'DBDebug'  => true,
    'charset'  => 'utf8',
    'DBCollat' => 'utf8_general_ci',
    'swapPre'  => '',
    'encrypt'  => false,
    'compress' => false,
    'strictOn' => false,
    'failover' => [],
    'port'     => 3306,
  ];
```

Modélisation des données

Un model est une classe permettant d'interagir avec une unique table de la base de données. Elle fournie des méthodes pour :

- Se connecter à la base de données ;
- Support pour des opérations de création, mise à jour et élimination (CRUD) ;
- Validation des données ;

Ces classes doivent se trouver dans le répértoire app/Models

Elle doit hériter de la classe Model.

Elle possède aussi une méthode d'initialisation.

```php
<?php

namespace App\Models;

use CodeIgniter\Model;


class EtudiantModel extends Model

{

// La configuration de base de données à utiliser.

    protected $DBGroup = 'default';

    protected function initialize()

   {

       $this->allowedFields[] = 'Id';

   }
```

Modélisation des données

Un model est configuré via un ensemble de propriétés de la classe.

```php
<?php
    protected $table      = 'Etudiant';
    protected $primaryKey = 'id';

    protected $useAutoIncrement = true;

    protected $returnType     = 'array';
    protected $useSoftDeletes = true;
    protected $allowedFields = ['Nom',
                                'Prenom'];
```

```php
<?php
 // Enregistrement des modifications.
// Cela permettra d'implanter des softdeletes
    protected $useTimestamps = true;
    protected $dateFormat    = 'datetime';
    protected $createdField  = 'created_at';
    protected $updatedField  = 'updated_at';
    protected $deletedField  = 'deleted_at';
```

Modélisation des données

Un model est configuré via un ensemble de propriétés de la classe.

```php
<?php

// Ces tableaux contiendront les règles de

// validation pour les données de la table.

    protected $validationRules      = [];

    protected $validationMessages   = [];

    protected $skipValidation       = false;

    protected $cleanValidationRules = true;
```

```php
<?php

// Ces propriété contiendront les méthodes à

// appeler après certaines opérations.

    protected $allowCallbacks = true;

    protected $beforeInsert   = [];

    protected $afterInsert    = [];

    protected $beforeUpdate   = [];

    protected $afterUpdate    = [];

    protected $beforeFind     = [];

    protected $afterFind      = [];

    protected $beforeDelete   = [];

    protected $afterDelete    = [];
```

Utilisation d’un model

Un model peut être instancié à l’intérieur d’un contrôleur ou dans une classe Entity.

```php
<?php
// Création manuel de l’objet.
$etudiantModel = new \App\Models\EtudiantModel();

// Création d’un nouveau objet du model.
$etudiantModel = model('App\Models\EtudiantModel', false);

// Création d’une instance partagée de l’objet Model.
$etudiantModel = model('App\Models\EtudiantModel');
```

Requêtes avec un model

En utilisant un objet model, il est possible de récupérer de l'information à partir de la table référencée par le model comme suit :

```php
<?php

$etudiantModel = new \App\Models\EtudiantModel(); // Création manuel de l'objet.

$etudiant = $etudiantModel->find($id);          // Récupére l'étudiant avec l'id donné.

$etudiants = $etudiantModel->find([1, 2, 3]);  // Récupére plusieurs étudiants.

// Récupére tous les étudiants qui n'ont pas été éliminés (deleted_at == NULL).

$users = $userModel->findAll();

// Récupére tous les étudiants, même les éliminés.

$etudiants = $etudiantModel->withDeleted()->findAll();

// Récupére les étudiants remplissant le critère dans le Where.

$etudiants = $etudiantModel->where('Nom', 'Uribe')->findAll();
```

Insertion avec un model

Il est possible de créer un nouveau enregistrement en fournissant les informations dans un tableau associatif comme suit :

```php
<?php
$data = [
    'prenom' => 'Ricardo',
    'nom'    => 'Uribe Lobello'
];
// Enregistre les informations dans la table et retourne le dernier id généré.
$dernierId = $userModel->insert($data);
// Enregistre le données et retourne true si tout s'est bien passé et false sinon.
$userModel->insert($data, false);
// Retourne l'identifiant du dernier enregistrement.
$userModel->getInsertID();
```

Actualisation avec un model

Les opérations d'actualisation sont aussi très simples à réaliser.

Attention : la clé primaire de la table ne doit pas être fournie dans le tableau data.

```php
<?php
$data = [
    'prenom' => 'Ricardo',
    'nom'    => 'Uribe Lobello'
];
// Actualise les informations d'un seul enregistrement.
$userModel->update($id, $data);
// Actualise les informations de trois enregistrements.
$userModel->update([1, 2, 3], $data);
// Actualise tous les enregistrements satisfaissant les conditions du Where.
$userModel->whereIn('id', [1, 2, 3])->set(['Role' => 'Etudiant'])->update();
```

Insertion et actualisation avec un model

La méthode **save**() permet de réaliser les opérations d'insertion et d'actualisation avec le même appel. Dans ce cas, il faudra fournir l'identifiant dans le tableau de données.

```php
<?php
// Attention, il fat bien établir la clé primaire dans le model.
$primarykey =  'Id';
$data = [
    'Id' => 1,
    'prenom' => 'Ricardo',
    'nom'    => 'Uribe Lobello'
];
// Si l'enregistrement n'existe pas, il va l'ajouter dans la table. Si l'enregistrement existe, il va actualiser ses informations en
// utlisant sa clé primaire.
$userModel->save($data);
```

Les Entités

- Une entité est une classe représentant un concept, en général, une table dans une base de données ;
- Chaque instance d'une entité représente uniquement un enregistrement (ou ligne) dans la table correspondante ;
- A différence des classes Modèles, elles ne doivent pas connaître la manière comment ses informations sont enregistrées dans la base de données, fichiers JSON ou des fichiers XML ;
- Au contraire des classes modèles, elles permettent d'implémenter les méthodes du modèle de business de l'application ;
- Elles permettent d'isoler le traitement des informations au sein de l'application de la manière comment ces informations sont rendues persistantes.

Implémentation des entités

Les classes Entités doivent être stockées dans le dossier **app/Entities**.

Les attributs sont stockés dans un tableau associatif **$attributs**.

Chaque classe Entité doit hériter de la classe **Entity** et :

- Elles peuvent contenir des setters et Getters pour faire des vérifications ou appliquer la logique de l'application.
- Les attributs sont stockés dans un tableau associatif.

Attention : Les noms des attributs doivent être écris exactement comme les colonnes correspondantes dans la base de données.

```php
<?php
// La classe entité doit être définie comme suit :
namespace App\Entities;
use CodeIgniter\Entity\Entity;
use CodeIgniter\I18n\Time;
class Etudiant extends Entity {
    public function setPassword(string $pass) {
        $this->attributes['password'] = password_hash($pass, PASSWORD_BCRYPT);
        return $this;
    }

    public function setCreatedAt(string $dateString)  {
        $this->attributes['created_at'] = new Time($dateString, 'UTC');
        return $this;
    } ...
}
```

Implémentation des entités

Chaque classe Entité doit être connectée à une classe Model.

La classe modèle correspondante à la classe Entité doit spécifier que tous les requêtes à la base de données doivent retourner des objets de la classe entity.

Si l'option useTimestamps est affectée à vrai, les colonnes spécifiées comme de telles vont être mis à jour automatiquement en dépendant de l'opération réalisée dans la base de données.

```php
<?php

namespace App\Models;

use CodeIgniter\Model;

class EtudiantModel extends Model
{
    protected $table = 'Etudiant';

    protected $allowedFields = [ ''Nom', 'Prenom', 'password'  ];

// Le type de retour des requêtes à la base de données ne sera plus constituée

// des tableux associatif mais des objets de ce type.

    protected $returnType     = \App\Entities\Etudiant::class;

// Cette ligne s'assure que les champs Created, Updated et Deleted, s'ils existent,

 soient mis à jour systématiquement.

    protected $useTimestamps = true;

}
```

Utilisation d'une classe entité

```php
<?php

// Pour charger un objet à partir de son identifiant.

$etudiant = $etudiantModel->find($id);

// Pour afficher les informations de l'objet, des accessors sont disponibles pour chaque attribut.

echo $etudiant->Nom;

echo $etudiant->Prenom;

// Actualisation.

unset($user->Nom);

if (! isset($etudiant->Nom)) {

    $etudiant->Nom = 'Uribe Lobello';

}
```

```php
<?php

// Méthode pour mettre à jour l'enregistrement dans la table correspondante.

$userModel->save($user);


// Création d'un nouveau enregistrement dans la table Etudiant.

$nouveauEtudiant = new \App\Entities\
Etudiant();

$nouveauEtudiant->Nom = 'André';

$nouveauEtudiant->Prenom    = 'Fou';

$etudiantModel->save($nouveauEtudiant);
```

L’implantation de la vue (interface utilisateur)

Les views

- Les views sont simplement des fichiers .php contenant une page ou une partie d'une page, par exemple :
  - L'entête ;
  - Le pied de page ;
  - Le menu de navigation.
- En CodeIgniter, ces pages ne son jamais accessibles directement. Elles doivent être chargées par un contrôleur.
- Le contrôleur reçoit la requête, décide l'action à exécuter avec le modèle et construit une réponse en chargeant les views nécessaires pour la construire.
- Ainsi, une réponse peut être composée d'une ou plusieurs views.

Construction d’une réponse

- Pour envoyer un view vers le client, il faut retourner le résultat de l’appel à la function view() en passant le nom du fichier php contenant le code de la view.
- Tout ce qui est retourné par une méthode du contrôleur en passant par la méthode view() sera envoyé au navigateur.

```php
<?php

namespace App\Controllers;

use CodeIgniter\Controller;


class GestionDepIUT extends Controller
{
    public function index()
    {
        return view('index’);
    }
}.
```

Construction d'une réponse

- Il est possible d'envoyer une concaténation des views vers le navigateur.
- La fonction view() possède un deuxième argument permettant d'envoyer des données à la view dans un tableau associatif.
- Dans l'exemple ci-contre, le fichier contenu disposera d'une variable $Titre contenant le titre de la page.

```php
<?php
namespace App\Controllers;
use CodeIgniter\Controller;


class GestionDeplUT extends Controller
{
    public function index() {
        $data = [  'Titre' => 'Titre de la page' ];
        return view('Entete') . view('navigation')
                . view('contenu', $data) . view('PiedDePage');
    }
}.
```

Construction d'une réponse

Le tableau associatif à envoyer à la vue est très souple. Il permet d'envoyer :

- D'autres tableaux associatifs ;
- Des objets ;
- Des tableaux d'objets ;

Attention : pour les utiliser dans la vue, il faudra que la vue connaisse les types de données utilisées, par exemple, la classe des objets qui lui sont transmis.

```php
<?php
    public function listerEtudiants() {
        $etudiants = $etudiantModel->findAll();
        $data = [  'Titre' => 'Liste des étudiants',
                    'etudiants' => $etudiants];
        return view('Entete')
                . view('navigation')
                . view('contenu', $data)
                . view('PiedDePage');
    }
}
```

Le fichier de view

Le fichier contenu.php de la réponse aura accès aux variables passées dans le tableau associatif.

CodeIgniter fourni des formats propriétaires pour afficher la valeur d'une variable :

<?= $titre?>

et aussi pour le parcours d'un tableau avec :

foreach ($etudiants as $etudiant)

...

endforeach

```
<!-- Format pour l'affichage de la valeur en titre -->
   <h1><?= esc($titre) ?></h1>
   <ul>
<!-- Format pour la création d'une boucle foreach -->
   <?php foreach ($etudiants as $etudiant): ?>
      <li><?= esc($etudiant->Prenom) ?>
          <?= esc($etudiant->Nom) ?>
      </li>
   <?php endforeach ?>
   </ul>
```

Les filtres

Les filtres

- Un filtre est une classe qui vous permet de capturer des requêtes afin de réaliser des actions avant et après la méthode qui traite la requête dans le contrôleur ;
- Ces filtres peuvent modifier la requête ou la réponse envoyée au navigateur ;
- Ils sont utiles pour :
  - Protéger le code du contrôleur des attaques CSRF ;
  - Contrôler l'accès à certaines parties de votre code. En se basant sur le rôle de l'utilisateur ;
  - Réaliser de négociations de contenu automatique ;
  - Etc.

Implantation des filtres

- Votre filtre sera une nouvelle classe implémentant l'interface FilterInterface.
- Cette interface contient deux méthodes :
  - **before**() : recevant la requête envoyée par le navigateur. Elle sera exécutée avant la méthode dans le contrôleur ;
  - **after**() : recevant la requête originale et la réponse à transmettre au navigateur une fois finie l'exécution de la méthode dans le contrôleur.

```php
<?php
namespace App\Filters;
use CodeIgniter\Filters\FilterInterface;
use CodeIgniter\HTTP\RequestInterface;
use CodeIgniter\HTTP\ResponseInterface;

class MyFilter implements FilterInterface
{
    public function before(RequestInterface $request, $arguments = null) {
        // Code à exécuter avant le contrôleur.
    }

    public function after(RequestInterface $request,
                          ResponseInterface $response, $arguments = null)
{
        // Code à exécuter après le contrôleur.
    }
}
```

Configuration des filtres

La configuration des filtres se fait dans le fichier **app/Config/Filters.php**.

$aliases permettent d'affecter des noms aux ensembles des filtres afin de les exécuter un après les autres.

$globals permet d'établir quels sont les filtres qui s'exécuter avant toute exécution d'une méthode dans un contrôleur.

```
<?php
namespace Config;
use CodeIgniter\Config\BaseConfig;

class Filters extends BaseConfig {
    public $aliases = [
            'MesFiltres' => [
            \App\Filters\Autentication::class,
            \App\Filters\Verification::class
            ],
            ];
    public $globals = [
                'before' => [ 'MesFiltres' ],
                'after' => []
    ];

    // ...
}
```

Configuration des filtres

La configuration des filtres se fait dans le fichier **app/Config/Filters.php**.

Il est possible de spécifier des urls pour lesquels les filtres ne seront pas exécutés.

Ou de spécifier pour quelles méthodes, les filtres vont s'exécuter.

Attention : uniquement les méthodes before() vont s'exécuter dans ce dernier cas.

```php
<?php
namespace Config;
use CodeIgniter\Config\BaseConfig;

class Filters extends BaseConfig {
    public $aliases = [
            'MesFiltres' => [
            \App\Filters\Autentication::class,
            \App\Filters\Verification::class
            ] ];
    public $globals = [
                'before' => [ 'MesFiltres'  => ['except' => ['exit/*''] ],
                'after' => [] ];
public $methods = [
        'post' => ['filtreSubmit', 'FiltreVerif'],
        'get'  => ['filteSubmit'],
    ];
}
```

Utilisation des filtres

Dans la méthode **before()**, vous pouvez réaliser plusieurs actions :

- **Modifier l'objet request :** il faut modifier l'objet et le retourner. Cela remplacera l'ancien objet request dans le contrôleur. Qui sera de toute manière exécuté ;
- **Retourner non vide** : Si vous retournez vide, la méthode du contrôleur va aussi s'exécuter. Par contre, en retournant non vide, vous arrêtez l'exécution de la méthode dans le contrôleur ;
- **Retourner une réponse :** dans ce cas, la réponse sera envoyée au client et le contrôleur ne sera pas exécuté.

```php
<?php
namespace App\Filters;
use CodeIgniter\Filters\FilterInterface;
use CodeIgniter\HTTP\RequestInterface;
use CodeIgniter\HTTP\ResponseInterface;

class MyFilter implements FilterInterface
{
    public function before(RequestInterface $request, $arguments = null)
    {
        $auth = service('auth');

        if (! $auth->isLoggedIn()) {
            return redirect()->to(site_url('login'));
        }
    }
}
```

Institut Universitaire de Technologie
Aix-Marseille Université

La session

Implémentation de la session

La session est fournie comme un service en CodeIgniter ;

Ainsi, elle est disponible dans chaque page mais il faudra de toute manière la charger.

Nous n'utiliserons plus la session pour stocker des informations de manière semi-permanente (nous disposons pour cela de la base de données).

Nous l'utiliserons uniquement pour stocker des informations pertinentes sur l'utilisateur qui est vient de s'authentifier dans notre application.

Pour charger la session, il faut faire :

$session = \Config\Services::session($config);

Le paramètre $config est optionnel.

Utilisation de la session


```php
<?php
// Récupérer une valeur de la session.
$nom = $session->get('Nom');
$nom = $session->Nom;
// Affecter un ensemble de données à la session.
$donnees = [ 'Nom'  => 'Uribe', 'Prenom'     => 'Ricardo', 'Age' => 18 ];
$session→set($donnees);
// Changer la valeur d'une donnée.
$session->set('Nom', 'Uribe Lobello');
// Vérifier si un item existe dans la session.
$session->has('Nom');
```

Utilisation de la session


```php
<?php

// Eliminer une information de la session

$session->remove('Nom');

// Por éliminer toutes les informations de la session.

$session->destroy();

// Pour détruire la session en changeant l'ID qui est généralement affecté à la connexion.

$session->stop();
```

Les informations de configuration de la session se trouvent dans le fichier **app/Config/Session.php**. Cela inclut :

- Le temps d'expiration ;
- Le nom du cookie utilisé ;
- Le temps de mise à jour et d'autres.

Utilisation de la session

Le flash data sont des informations qui peuvent être stockées dans la session mais qui ne vont pas qu'être conservées que pendant la prochaine requête.

C'est peut être utile pour maintenir des informations sur une procédure en cours qui doit se poursuivre sur plusieurs pages.

Attention, il ne faut pas en abuser.

```php
<?php

// Indique qu'une information stockée dans la session est un flash data

$session->markAsFlashdata('Nom');//

// Créer un flash data.

$session->setFlashdata('Nom', 'Uribe');

// Obtenir un flash data.

$session->getFlashdata('Nom');

// Maintenir un flash data pour une autre requête.

$session->keepFlashdata('item');
```