



Créer une application à l'aide de Framework Yii

**YII 2,0**



# PRINCIPES FONDAMENTAUX



# Principes fondamentaux

- ◉ Qu'est ce que Yii
- ◉ Configuration nécessaire,
- ◉ Modelés des applications
  - (Application Template)
- ◉ Installation du Template Basic,
- ◉ Création d'une page,
- ◉ Router

# Qu'est ce que Yii

- Yii est un framework PHP hautes performances à base de composants qui permet de développer rapidement des applications Web modernes.
- Yii est implémenté selon le patron de conception modèle-vue-contrôleur (MVC)





# Configuration nécessaire

- ⦿ > PHP 5.4.0.
- ⦿ SGBD pour la base de donnée
- ⦿ Connaissances de base sur la programmation objet (OOP),
- ⦿ Connaissances de base sur control de version Git, et les systèmes de gestion de packages(composer)

# Modelés des applications

- ◎ Yii2 dispose de deux modèles d'application pour le développement: basique et avancé.
  - Basic
  - Advanced
- ◎ <https://github.com/trntv/yii2-starter-kit>



# Installation du Template Basic

1. Installation du Composer(package manager),

2. Installation **Bower-to-Composer**

```
Composer global require "fxp/composer-asset-plugin:^1.2.0"
```

Installer une seule fois

3. Créez une nouvelle application dans le nouveau répertoire 'basic'

```
composer create-project --prefer-dist yiisoft/yii2-app-basic basic
```

4. Accéder à partir du navigateur

```
http://hostname/basic/web/index.php
```

# Création et affichage des pages statiques

- ◉ Ajouter une page dans le contrôleur 'SiteController':
  1. Déclarez l'action 'actionProfile' dans le SiteController
  2. Créer Vue pour cette page,
  3. Ajouter code de rendu de la vue,
  4. `index.php?r=site/profile`





# APPLICATION



# Utilisation,,,

- ◉ Structure
- ◉ Scripts d'entrée
- ◉ Cycle de vie d'une requête
- ◉ Configuration de l'application

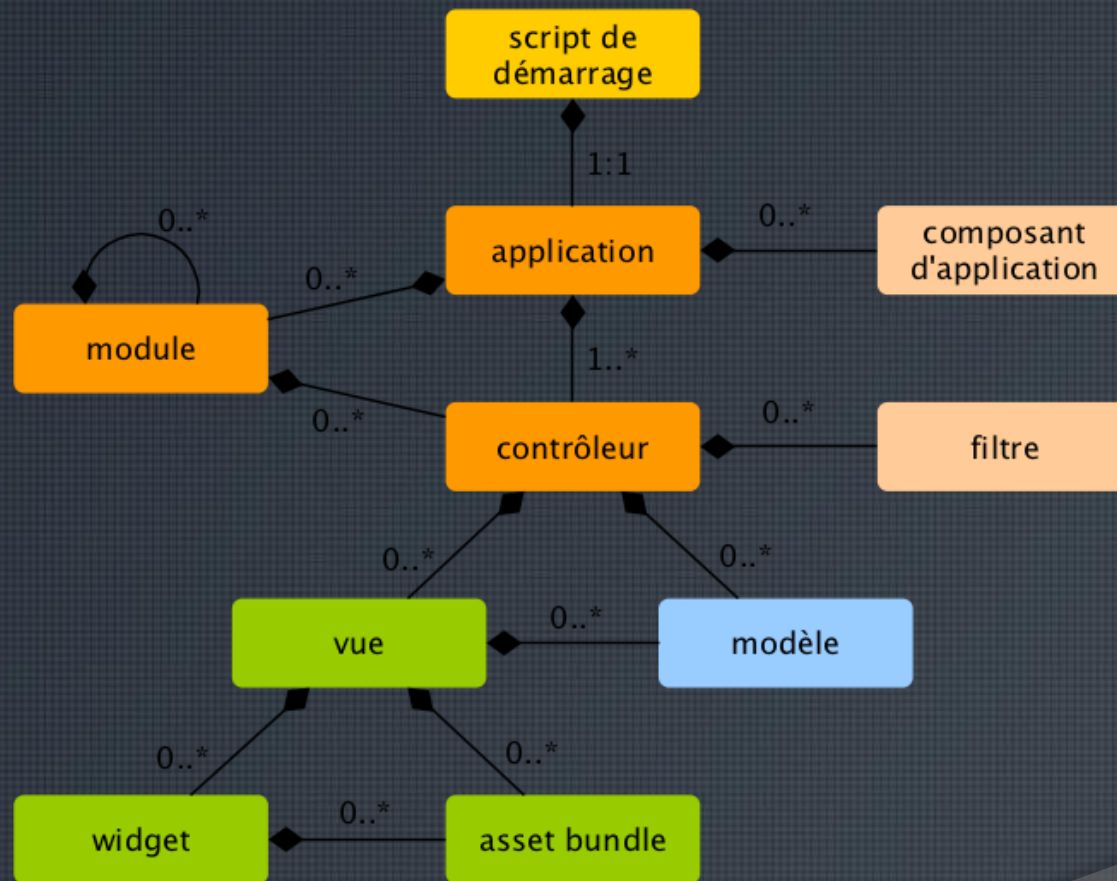


# Structure de l'Application

## ◉ Les répertoires et fichiers les plus importants de l'application

basic/	chemin de base de l'application
composer.json	utilisé par Composer, décrit les informations de paquets
config/	contient les configurations de l'application et autres
console.php	configuration de l'application console
web.php	configuration de l'application Web
commands/	contient les classes de commandes console
controllers/	contient les classes de contrôleurs
models/	contient les classes de modèles
runtime/	contient les fichiers générés par Yii au cours de l'exécution, tels que les
fichiers de logs ou de cache	and cache
vendor/	contient les paquets Composer installés, y compris le framework Yii
views/	contient les fichiers de vues
web/	racine Web de l'application, contient les fichiers accessibles via le Web
assets/	contient les fichiers assets (javascript et css) publiés par Yii
index.php	le script de démarrage (ou bootstrap) pour l'application
yii	le script d'exécution de Yii en commande console

# structure statique d'une application





# Scripts d'entrée

- Le script d'entrée est responsable du lancement d'un cycle de traitement des demandes. Ce sont des scripts PHP accessibles aux utilisateurs.

# Ce qui suit est le script d'entrée pour le modèle d'application de base

**<?php**

Définir des constantes.

```
defined('YII_DEBUG') or define('YII_DEBUG',  
defined('YII_ENV') or define('YII_ENV', 'dev
```

Enregistrez le chargeur automatique Composer.

```
require(__DIR__ . '/../vendor/autoload.php');
```

```
require(__DIR__ . '/../vendor/yiisoft/yii2/Yii.php');
```

Incluez les fichiers Yii.

```
$config = require(__DIR__ . '/../config/web.php');
```

```
(new yii\web\Application($config))->run();
```

Charger la configuration.  
Créer et configurer une instance d'application.  
Traiter la demande entrante.

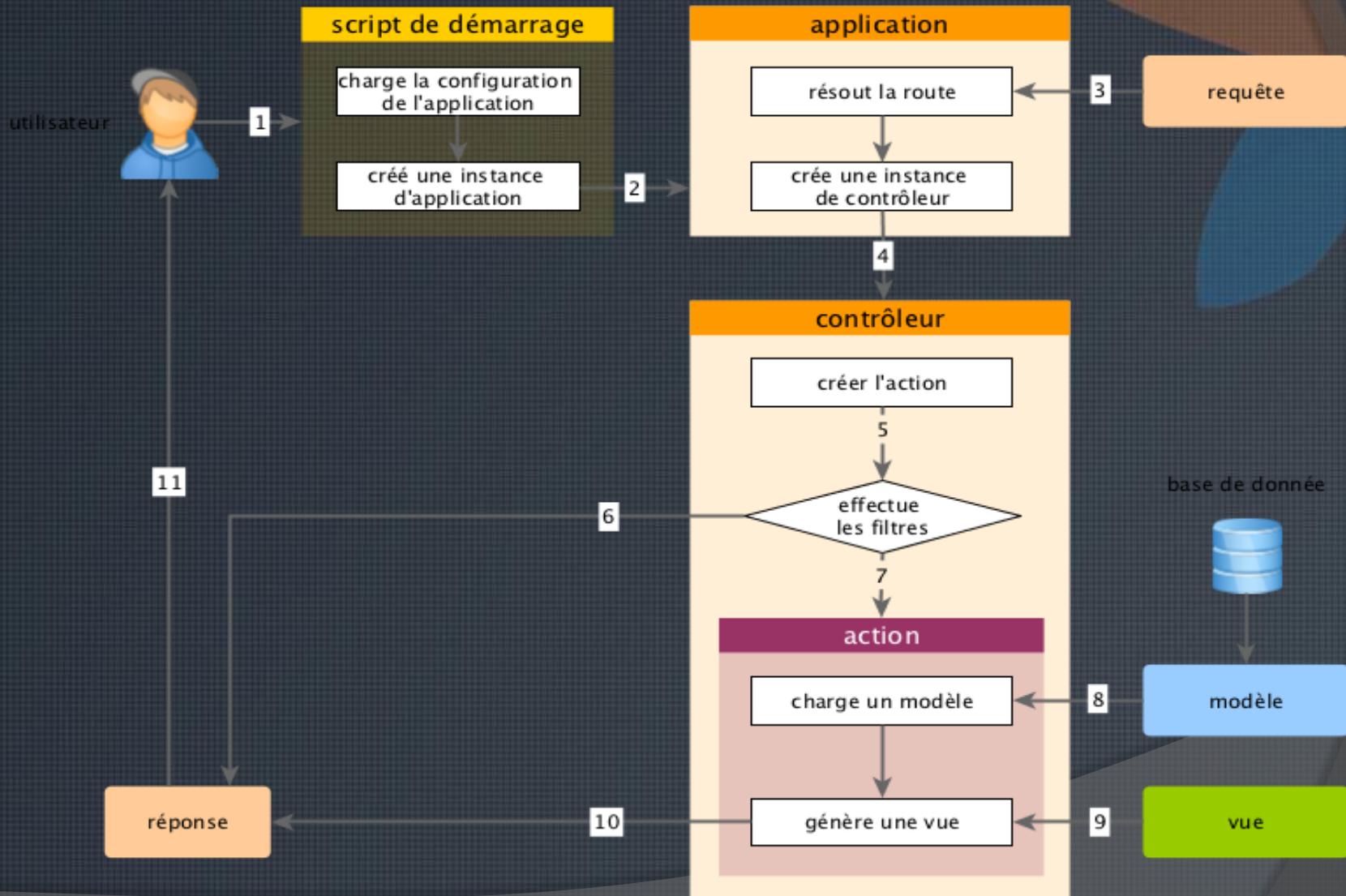


# Définir des constantes

- ◎ **YII\_DEBUG** - Définit si vous êtes en mode debug ou non. Si cette valeur est définie sur true, nous verrons plus de données de journal et d'erreur de détail.

**YII\_ENV** - Définit le mode environnement. La valeur par défaut est prod. Les valeurs disponibles sont prod, dev et test. Ils sont utilisés dans les fichiers de configuration pour définir, par exemple, une connexion DB différente (locale et distante) ou d'autres valeurs.

# Cycle de vie d'une requête





# Cycle de vie d'une requête

1. Un utilisateur fait une requête au script de démarrage web/index.php.
2. Le script de démarrage charge la configuration de l'application et crée une instance d'application pour traiter la requête.
3. L'application résout la route requise avec l'aide du composant d'application requête.
4. L'application crée une instance de contrôleur pour traiter la requête.
5. Le contrôleur crée une instance d'action et effectue les filtres pour l'action.
6. Si un filtre échoue, l'action est annulée.
7. Si tous les filtres sont validés, l'action est exécutée.
8. L'action charge un modèle de données, potentiellement depuis une base de données.
9. L'action génère une vue, lui fournissant le modèle de données.
10. Le résultat généré est renvoyé au composant d'application réponse.
11. Le composant réponse envoie le résultat généré au navigateur de l'utilisateur.

# Application

- ◉ Deux types d'applications: Web et console.
- ◉ Les applications sont des objets qui régissent la structure globale et le cycle de vie des systèmes d'application Yii. Chaque système d'application Yii contient un seul objet d'application qui est créé dans le script d'entrée et est globalement accessible via l'expression `\Yii::$app`.



# Composants de l'application

- ◉ Les composants sont des localisateurs de services. Ils hébergent un ensemble de composants dits applicatifs qui fournissent différents services pour le traitement des requêtes. Le composant db fournit des services liés à DB; etc.
- ◉ Chaque composants a une configuration spécifique.
- ◉ Chaque composant d'application possède un ID
- ◉ Accéder à un composant d'application par l'expression:  
`\Yii::$app->componentID`

Voir également

Pour plus d'informations sur DIP, reportez-vous à

[https://en.wikipedia.org/wiki/Dependency\\_inversion\\_principle](https://en.wikipedia.org/wiki/Dependency_inversion_principle)

Pour en savoir plus sur le conteneur à injection de dépendance,

<http://www.yiiframework.com/doc-2.0/guide-concept-di-container.html> - Yii2.0 - ILIMA 2016

# Composants d'application de base

- Yii définit un ensemble de composants d'application de base avec des ID fixes et des configurations par défaut.
- `assetManager`, `db`, `errorHandler`, `formatter`, `i18n`, `log`, `mailer`, `response`, `request`, `session`, `urlManager`, `user`, `view`

Voir également

Les composants de console et d'application Web :

[Http://www.yiiframework.com/doc-2.0/guide-structure-application-components.html](http://www.yiiframework.com/doc-2.0/guide-structure-application-components.html)





# Configurations

```
<?php
```

```
$params = require(__DIR__ . '/params.php');
```

```
$config = [  
    'id' => 'basic',  
    'basePath' => dirname(__DIR__),  
    'bootstrap' => ['log'],  
    'components' => [  
        'request' => [  
            'cookieValidationKey' => 'something',  
        ],  
        'cache' => [  
            'class' => 'yii\caching\FileCache',  
        ],  
        'user' => [  
            'identityClass' => 'app\models\User',  
            'enableAutoLogin' => true,  
        ],  
        'errorHandler' => [  
            'errorAction' => 'site/error',  
        ],  
        'mailer' => [  
            'class' => 'yii\swiftmailer\Mailer',  
            'useFileTransport' => true,  
        ],  
        'log' => [  
            'traceLevel' => YII_DEBUG ? 3 : 0,  
            'targets' => [  
                [  
                    'class' => 'yii\log\FileTarget',  
                    'levels' => ['error', 'warning'],  
                ],  
            ],  
        ],  
        'db' => require(__DIR__ . '/db.php'),  
        /*  
        'urlManager' => [  
            'enablePrettyUrl' => true,  
            'showScriptName' => false,  
            'rules' => [  
            ],  
        ],  
        */  
    ],  
    'params' => $params,  
];
```

## Configurations obligatoire

La propriété `basePath` spécifie le répertoire racine d'une application. C'est le répertoire qui contient tout le code source protégé d'un système d'application. Qui contiennent le code source correspondant au modèle MVC

## Configuration des composants



# Configuration des composants

- DB est un composant core dans le système de Yii
- Exemple de Composant de gestion d'accès à la base de données:

```
return [  
    'class' => 'yii\db\Connection',  
    'dsn' => 'mysql:host=localhost;dbname=yii2basic',  
    'username' => 'root',  
    'password' => '',  
    'charset' => 'utf8',  
];
```



# Contrôleurs, vues et routage



# Controller

- Les contrôleurs des applications Web devraient s'hériter de `yii\web\Controller` ou de ses classes enfant. Dans les applications console, elles doivent s'étendre depuis `yii\console\Controller` ou ses classes enfant.
- Pour convertir l'ID du contrôleur en nom de classe de contrôleur, procédez comme suit:
  - Prenez la première lettre de tous les mots séparés par des tirets et tournez-le en majuscules.  
Supprimez les traits d'union.  
Remplacer les barres obliques vers l'arrière par des barres vers l'arrière.  
Ajoutez le suffixe `Controller`.  
Préfixe l'espace de noms du contrôleur.
  - Ex:
  - `page` -> `app\controllers\PageController`.
  - `post-article` -> `app\controllers\PostArticleController`.
  - `user/post-article` -> `app\controllers\user\PostArticleController`.
  - `userBlogs/post-article` -> `app\controllers\userBlogs\PostArticleController`.
- Utiliser `'defaultRoute'` dans la configuration de l'application pour mettre ce contrôleur par défaut

```
'defaultRoute' => 'formation',
```



```
<?php
```

```
namespace app\controllers;
```

```
use yii\filters\AccessControl;
```

```
use yii\filters\VerbFilter;
```

```
use yii\web\Controller;
```

```
class FormationController extends Controller {
```

```
    public function actionIndex(){  
        return $this->render('index');  
    }
```

```
}
```



# Action

- Pour créer une action dans une classe de contrôleur, vous devez définir une méthode publique dont le nom commence par 'action'. Les données de retour d'une action représentent la réponse à envoyer à l'utilisateur final.
- Dans la barre d'adresse du navigateur Web.
  - Index.php?r=formation/index → actionIndex()
  - Index.php?r=formation/iafficher-formation → actionAfficherFormation()

```
class FormationController extends Controller {  
  
    public function actionIndex(){  
        return $this->render('index');  
    }  
  
    /**  
     * action avec paramètre  
     * $id c'est le même paramètre dans les paramètres d'environnement $_GET  
     * @param $id  
     *  
     * @return string  
     */  
    public function actionAfficherFormation($id){  
  
        return $this->render('view', ['id'=>$id]);  
    }  
}
```

# Créer une classe d'action autonome

- Créer une action extérieur de la classe Controller.

```
<?php

namespace app\components;

use yii\base\Action;

class ErrorAction extends Action {
    public function run() {
        return "Error";
    }
}
```

- Surcharger la méthode 'actions()' dans la class Controller

```
public function actions() {
    return [
        'error'=>ErrorAction::className()
    ];
}
```



# Répondre à la demande

- Pour répondre à la demande, le contrôleur subira le cycle de vie suivant :
  - La méthode `yii\base\Controller::init()` est appelée.
  - Le contrôleur crée une action basée sur l'ID d'action.
  - Le contrôleur appelle séquentiellement la méthode `beforeAction()` de l'application Web, du module et du contrôleur.
  - Le contrôleur exécute l'action.
  - Le contrôleur appelle séquentiellement la méthode `afterAction()` de l'application Web, du module et du contrôleur.
  - L'application attribue un résultat d'action à la réponse.

# View

- ◉ Les vues sont responsables de la présentation des données aux utilisateurs finaux. Dans les applications Web, les vues ne sont que des fichiers de script PHP contenant du code HTML et PHP.
- ◉ Ils doivent être placés dans le répertoire @app/views/ControllerID par défaut, où ControllerID fait référence à l'ID du contrôleur.
- ◉ To render a view within a controller, we may use the following methods:
  - Render () - Rendre une vue et appliquer une mise en page.
  - RenderPartial () - Rendre une vue sans mise en page.
  - RenderAjax () - Rend une vue sans mise en page, mais injecte tous les fichiers js et css enregistrés.
  - RenderFile () - Rendre une vue dans un chemin d'accès ou un alias de fichier donné.
  - RenderContent () - Rend une chaîne statique et applique une disposition.
  - Pour afficher une vue dans une autre vue, vous pouvez utiliser les méthodes suivantes:
    - Render () - Rendre une vue.
    - RenderAjax () - Rend une vue sans mise en page, mais injecte tous les fichiers js et css enregistrés.
    - RenderFile () - Rendre une vue dans un chemin d'accès ou un alias de fichier donné.



# Example

```
<?php
/* @var $this yii\web\View */
$this->title = 'APplication';
?>
<div class="site-index">
    <h1><?= \yii\helpers\Html::encode($this->title) ?></h1>
    <div class="jumbotron">
        <h1>Congratulations!</h1>
        <p class="lead">You have successfully created your Yii-powered application.</p>
        <p><a class="btn btn-lg btn-success" href="http://www.yiiframework.com">Get started with
Yii</a></p>
    </div>
</div>
```

La variable `$this` fait référence au composant de View qui gère et rend ce vue.

**`yii\helpers\Html::encode()`**  
Protection de XSS attaque

# Accès aux données dans les vues

- Pour accéder aux données dans une vue, vous devez passer les données comme deuxième paramètre à la méthode de rendu(render) de vue.

```
public function actionAfficherFormation( ){  
    $nom = 'Amine';  
    return $this->render('view',['nom'=>$nom]);  
}
```

Au niveau de Controller

```
<h1>This is formation de <?=$nom?></h1>
```

Au niveau de View



# Utilisation du contexte du contrôleur dans une vue

- ◉ `<?php $this->context->hello() ?></p>`

# Réutilisation des vues avec les partiels

- Yii prend en charge les partiels, donc si nous avons un bloc sans beaucoup de logique que vous voulez réutiliser

```
<?php  
echo $this->render('_part');
```



# Utilisation de blocs

- ◉ Une des fonctionnalités Yii que vous pouvez utiliser dans vos vues est les blocs
- ◉ Création :

```
<?php $this->beginBlock('block1');?>  
<h2>Exemple de block</h2>  
<?php $this->endBlock(); ?>
```

- ◉ Utilisation:

```
<?php  
echo $this->blocks['block1'];
```

# Ne pas faire cela avec des vues:

- ⦿ Contenir HTML et code PHP simple pour formater et rendre des données.
- ⦿ NE PAS traiter les demandes.
- ⦿ NE PAS modifier les propriétés du modèle.
- ⦿ NE PAS effectuer de requêtes de base de données.



# Layouts

- Layouts représentent les parties communes de vues multiples, par exemple l'en-tête de page et le pied de page. Par défaut, Layout doivent être stockées dans le dossier views/layouts.
- La variable `$content` est le résultat de rendu de vues.
- des méthodes déclenchent des événements sur le processus de rendu afin que les scripts et les tags enregistrés dans d'autres endroits puissent être correctement injectés:
  - `head()`, `beginBody()`, `endBody()`, `beginPage()`, `endPage()`:**

```

<?php
/* @var $this \yii\web\View */
/* @var $content string */

use yii\helpers\Html;
use yii\bootstrap\Nav;
use yii\bootstrap\NavBar;
use yii\widgets\Breadcrumbs;
use app\assets\AppAsset;

AppAsset::register($this);
?>
<?php $this->beginPage() ?>
<!DOCTYPE html>
<html lang="<?php Yii::$app->language ?>">
<head>
    <meta charset="<?php Yii::$app->charset ?>">
    <meta name="viewport" content="width=device-width, initial-
scale=1">
    <?php Html::csrfMetaTags() ?>
    <title><?php Html::encode($this->title) ?></title>
    <?php $this->head() ?>
</head>
<body>
<?php $this->beginBody() ?>

<div class="wrap">
    <?php
    NavBar::begin([
        'brandLabel' => 'My Company',
        'brandUrl' => Yii::$app->homeUrl,
        'options' => [
            'class' => 'navbar-inverse navbar-fixed-top',
        ],
    ]);
    echo Nav::widget([
        'options' => ['class' => 'navbar-nav navbar-right'],
        'items' => [
            ['label' => 'Home', 'url' => ['/site/index']],
            ['label' => 'About', 'url' => ['/site/about']],
            ['label' => 'Contact', 'url' => ['/site/contact']],
            Yii::$app->user->isGuest ? (
                ['label' => 'Login', 'url' => ['/site/login']]
            ) : (
                '<li>'
                . Html::beginForm(['/site/logout'], 'post')
                . Html::submitButton(
                    'Logout (' . Yii::$app->user->identity->username .
                    ')',
                    ['class' => 'btn btn-link logout']
                )
                . Html::endForm()
                . '</li>'
            ),
        ],
    ]);
    NavBar::end();
?>

    <div class="container">
        <?php Breadcrumbs::widget([
            'links' => isset($this->params['breadcrumbs']) ? $this-
>params['breadcrumbs'] : [],
            ]) ?>
        <?php $content ?>
    </div>

<footer class="footer">
    <div class="container">
        <p class="pull-left">&copy; My Company <?php date('Y') ?></p>

        <p class="pull-right"><?php Yii::powered() ?></p>
    </div>
</footer>

<?php $this->endBody() ?>
</body>
</html>
<?php $this->endPage() ?>

```



# Création(Layout)

1. Dans le répertoire views/layouts, créez un fichier appelé withoutNav.php
2. Pour appliquer layout au SiteController, ajoutez la propriété \$layout à la classe SiteController,

```
class SiteController extends Controller
{
    public $layout = 'withoutNav';
}
```

3. Si on veut appliquer just pour une action :

```
public function actionIndex()
{
    $this->layout = 'withoutNav';
    return $this->render('index');
}
```

```

<?php
/* @var $this \yii\web\View */
/* @var $content string */
use app\assets\AppAsset;
use yii\helpers\Html;

AppAsset::register( $this );
?>
<?php $this->beginPage() ?>
<!DOCTYPE html>
<html lang="<?= Yii::$app->language ?>">
<head>
    <meta charset="<?= Yii::$app->charset ?>">
    <meta name="viewport" content="width = device-width, initial-scale = 1">
    <?= Html::csrfMetaTags() ?>
    <title><?= Html::encode( $this->title ) ?></title>
    <?php $this->head() ?>
</head>

<body>
<?php $this->beginBody() ?>
<div class="wrap">
    <div class="container">
        <?= $content ?>
    </div>
</div>

<footer class="footer">
    <div class="container">
        <p class="pull-left">©My Company <?= date( 'Y' ) ?></p>
        <p class="pull-right"><?= Yii::powered() ?></p>
    </div>
</footer>
<?php $this->endBody() ?>
</body>

</html>
<?php $this->endPage() ?>

```



# Widgets

- ◉ Un widget est un code client réutilisable, qui contient HTML, CSS et JS. Ce code comprend une logique minimale et est enveloppé dans un objet `yii\base\Widget`. Nous pouvons facilement insérer et appliquer cet objet dans n'importe quelle vue.

# Utilisation des widgets

- Pour utiliser un widget dans une vue, vous devez appeler la fonction `yii\base\Widget::widget()`. Cette fonction prend un tableau de configuration pour initialiser le widget

```
<h1>This is formation de <?=$nom?></h1>
<?php
echo \yii\bootstrap\Tabs::widget(
    [
        'items' => [
            [
                'label' => 'Formation 1',
                'content' => 'Contenu',
                'active' => true
            ],
            [
                'label' => 'Coure 2',
                'content' => 'Contenu',
                'headerOptions' => [ ],
                'options' => ['id' => 'SomeID'],
            ],
        ],
    ],
);
```

Configuration

Widget pour afficher le contenu en tab



# This is formation de Amine

Formation 1

[Coure 2](#)

Contenu



# Block Widget

- Certains widgets prennent un bloc de contenu. Il doit être inclus entre les fonctions `yii\base\Widget::begin()` et `yii\base\Widget::end()`.

```
<?php
use yii\bootstrap\Modal;

Modal::begin([
    'header' => '<h2>Block widget </h2>',
    'toggleButton' => ['label' => 'Show the magic'],
]);

echo 'Rock you like a hurricane ';

Modal::end();
```

Voir également

Pour en savoir plus sur les widgets, vous pouvez utiliser les ressources suivantes:

[Http://www.yiiframework.com/doc-2.0/yii-base-widget.html](http://www.yiiframework.com/doc-2.0/yii-base-widget.html)

[Https://github.com/yiisoft/yii2-bootstrap/blob/master/docs/guide/usage-widgets.md](https://github.com/yiisoft/yii2-bootstrap/blob/master/docs/guide/usage-widgets.md)

# Assets

- ◎ **Asset** est un fichier (css, js, vidéo, audio ou image, etc.) qui peut être référencé dans une page Web. Yii gère des actifs dans des ensembles d'actifs. Le but d'un ensemble d'actifs est d'avoir un groupe de fichiers JS ou CSS liés dans la base de code et de pouvoir les enregistrer au sein d'un seul appel PHP. Les ensembles d'actifs peuvent également dépendre d'autres ensembles d'actifs.



# Model

- ◉ Les modèles sont des objets représentant la logique métier et les règles. Pour créer un modèle, vous devez étendre la classe `yii\base\Model` ou ses sous-classes.
- ◉ La classe étend `[[yii\base\Model]]`, une classe de base fournie par Yii, communément utilisée pour représenter des données de formulaire.

# Creation(Model)

- Nous pouvons créer des classes de modèle en étendant `yii\base\Model` ou ses sous-classes
- Plusieurs options:
  - Attributs
  - Étiquettes d'attribut (Attribute labels)
  - Règles de validation
  - Exportation de données

```
<?php

namespace app\models;

use yii\base\Model;

class ProductForm extends Model {

}
```



# Les attributs

- Les attributs représentent les données. Ils peuvent être accédés comme des éléments de tableau ou des propriétés d'objet. Chaque attribut est une propriété accessible au public d'un modèle.

```
$etudiant = new Etudiant();  
//ecriture  
$etudiant->preNom = 'Amine';  
//lecture d'attribue  
echo $etudiant->preNom;
```



# Étiquettes d'attributs

- Nous avons souvent besoin d'afficher les étiquettes associées aux attributs. Par défaut, les étiquettes d'attributs sont générées automatiquement par la méthode `yii \ base \ Model :: generateAttributeLabel ()`. Pour déclarer manuellement les étiquettes d'attributs, nous devons surcharger la méthode `yii \ base \ Model :: attributeLabels ()`.

```
<?php
```

```
namespace app\models;
```

```
use yii\base\Model;
```

```
class Etudiant extends Model {
```

```
    public $id;  
    public $preNom;  
    public $nom;
```

```
    /**  
     * @return array customized attribute labels  
     */
```

```
    public function attributeLabels() {  
        return [  
            'id'      => 'ID',  
            'preNom'  => 'Prénom',  
            'nom'     => 'Nom',  
        ];  
    }
```

```
    /**  
     * @return array the validation rules.  
     */
```

```
    public function rules() {  
        return [  
            [ [ 'id', 'nom' ], 'required' ],  
            [ 'id', 'integer' ],  
            [ 'preNom', 'string', 'length' => [4, 24]]  
        ];  
    }
```

Les attributs

Étiquettes d'attributs

Les Règles de validation



# Travailler avec les formulaires





# Formulaires

- ◉ Sans Model
- ◉ Avec Model
- ◉ Utilisation les regèles de validation
- ◉ Création notre propre regèles de validation
- ◉ Personnaliser le message d'erreur

# Sans Model (Old style)

## ⦿ Dans le contrôleur

```
public function actionForm(){  
    if(isset($_POST['nom'])){  
        //traitement  
    }  
    return $this->render('form');  
}
```

## ⦿ Dans la vue(form.php)

```
<form action="<?= \yii\helpers\Url::toRoute(['product/form'])?>" method="post">  
    <label>Nom</label>  
    <input type="text" name="nom">  
    <button type="submit">Envoyer</button>  
</form>
```



# Créer un formulaire

## Avec Model

- Lorsqu'un formulaire est basé sur un modèle, la façon habituelle de créer ce formulaire dans Yii est via la widget `yii\widgets\ActiveForm`. Dans la plupart des cas, un formulaire a un modèle correspondant qui est utilisé pour la validation des données. Si le modèle représente des données d'une base de données, le modèle doit être dérivé de la classe `ActiveRecord`. Si le modèle capture une entrée arbitraire, il doit être dérivé de la classe `yii\base\Model`.





# Formulaire

1. Créer un Modèle
2. Créer une Action
3. Créer des Vues
4. Essayer

# Créer un Modèle

```
<?php
namespace app\models;
use yii\base\Model;
class Etudiant extends Model {

    public $preNom;
    public $nom;

    /**
     * @return array the validation rules.
     */
    public function rules() {
        return [
            [ [ 'id', 'nom' ], 'required' ],
            [ 'id', 'integer' ],
            [ 'preNom', 'string', 'length' => [4, 24]]
        ];
    }

    /**
     * @return array customized attribute labels
     */
    public function attributeLabels() {
        return [
            'id' => 'ID',
            'preNom' => 'Prénom',
            'nom' => 'Nom',
        ];
    }
}
```

# 1. Créer une Action

```
public function actionAjouterEtudiant(){  
    $model = new Etudiant();  
    //le cas de l'utilisateur envoyé le formulaire  
    if($model->load(Yii::$app->request->post())){  
        //traitement  
  
        if($model->validate()){  
            //passer a une autre vue apres la validation OK  
            return $this->render('etudiant-ajout',['model'=>$model]);  
        }  
    }  
    return $this->render('form',['model'=>$model]);  
}
```

Diagram illustrating the steps of the `actionAjouterEtudiant()` function:

1. `$model = new Etudiant();`
2. `if($model->load(Yii::$app->request->post())){`
3. `if($model->validate()){`
4. `return $this->render('etudiant-ajout',['model'=>$model]);`
5. `return $this->render('form',['model'=>$model]);`



# Créer une Action

1. L'action commence par créer un objet `Etudiant`.
2. elle tente de peupler le modèle avec les données de `$_POST`, fournies dans yii par `[[yii\web\Request::post()]]`.
  1. Si le modèle est peuplé avec succès (c'est à dire, si l'utilisateur a soumis le formulaire HTML),
3. l'action appellera `[[yii\base\Model::validate() | validate()]]` pour s'assurer de la validité de toutes les valeurs.
4. Render Vue dans le cas de succès
5. Render Vue dans les cas normaux

# Créer des Vues

## Form.php

```
<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;
?>
<?php $form = ActiveForm::begin(); ?>

<?= $form->field($model, 'preNom') ?>

<?= $form->field($model, 'nom') ?>

<div class="form-group">
    <?= Html::submitButton('Soumettre', ['class' => 'btn btn-primary']) ?>
</div>

<?php ActiveForm::end(); ?>
```

## etudiant-ajout.php

```
<?php
use yii\helpers\Html;
?>
<p>Vous avez entré les informations suivantes :</p>

<ul>
    <li><label>Pre nom</label>: <?= Html::encode($model->preNom) ?></li>
    <li><label>Nom</label>: <?= Html::encode($model->nom) ?></li>
</ul>
```


# Creation(Form)

Nom

Prénom

Submit





# Play with it

- ◎ TP: Créer différent types de input

# Validation

- ◉ Nous ne devrions jamais faire confiance aux données reçues des utilisateurs. Pour valider un modèle avec des entrées utilisateur, vous devez appeler la méthode `yii\base\Model::validate()`. Il renvoie une valeur booléenne si la validation réussit. S'il ya des erreurs, vous pouvez les obtenir à partir de la propriété `yii\base\Model::$errors`.

# Validation

```
$model = new Etudiant();  
//Remplissez les attributs du modèle avec les entrées utilisateur  
if($model->load(\Yii::$app->request->post())){  
    //traitement  
  
    if($model->validate()){  
        //passer a une autre vue apres la validation OK  
    }else{  
        //validation no ok  
    }  
}  
return $this->render('form', ['model'=>$model]);
```



# Déclaration de règles de validation

- Pour que la fonction `validate()` fonctionne, vous devez surcharger la méthode `yii\base\Model::rules()`.
- Pour chaque règle, vous devez définir au moins les attributs auxquels s'applique la règle et le type de règle appliquée.
- Les règles de validation de base sont : **boolean, captcha, compare, date, default, double, each, email, exist, file, filter, image, ip, in, integer, match, number, required, safe, string, trim, unique, url**

```
public function rules() {  
    return [  
        [ [ 'id', 'nom' ], 'required' ],  
        [ 'id', 'integer', 'max'=>5 ],  
        [ 'preNom', 'string', 'length' => [4, 24]]  
    ];  
}
```

# Exemple d'une règle

```
[  
  // obligatoire, spécifie quels attributs doivent être validés par cette règle.  
  // Pour un seul attribut, vous pouvez utiliser le nom d'attribut directement  
  // sans l'avoir dans un tableau  
  ['Attribut1', 'attribut2', ...],  
  
  // requis, spécifie le type de cette règle.  
  // Il peut s'agir d'un nom de classe, d'un alias de validateur ou d'un nom de méthode de validation  
  'Valideur',  
  
  // facultatif, indique dans quel (s) scénario (s) cette règle doit être appliquée  
  // si non donné, cela signifie que la règle s'applique à tous les scénarios  
  // Vous pouvez également configurer l'option "except" si vous souhaitez appliquer la règle  
  // à tous les scénarios sauf ceux énumérés  
  'On' => ['scénario1', 'scénario2', ...],  
  
  // facultatif, spécifie des configurations supplémentaires pour l'objet validateur  
  'Propriété1' => 'valeur1', 'propriété2' => 'valeur2', ...  
]
```

# personnaliser le message d'erreur

- Pour personnaliser le message d'erreur de la propriété 'nom', modifiez la méthode rules () de Etudiant de la manière suivante.

```
public function rules() {  
    return [  
        [ [ 'id', 'nom' ], 'required', 'message' => 'Obligatoire monsieur ' ],  
        [ 'id', 'integer', 'max' ],  
        [ 'preNom', 'string', 'length' => [4, 24] ]  
    ];  
}
```





La plupart des applications utilisent des bases de données aujourd'hui

**Base de donnée!**

# Base de donnée

- Yii introduit trois façons de vous permettre de travailler avec des bases de données. Ils sont comme suit:
  1. SQL via DAO
  2. Query Builder
  3. Active Record



# Utilisation des bases de données

- ◎ Préparation du DB
  - Création du base de donnée
  - Configuration d'accès
  - Création des tables
    - Méthode ordinaire
    - Par Migration
- ◎ **Query Builder**
- ◎ **Active Record**
  - Créer application CRUD




# Préparation du DB

- Création du base de donnée
  - Par l'outis de 'PHPmyAdmin'

Bases de données

Créer une base de données ?

BD Interclassement **Créer**

Base de données	Action
<input type="checkbox"/> cours	 Vérifier les privilèges

# Yii - Accès à la base de données

- Yii prend en charge les bases de données suivantes -  
:
  - MySQL
  - MSSQL
  - SQLite
  - MariaDB
  - PostgreSQL
  - ORACLE
  - CUBRID
- Pour accéder à la connexion DB, nous utilisons cette expression:

```
\Yii::$app->db
```

# Préparation du DB

- ⦿ Configuration d'accès
  - Configurez la connexion à la base de données dans le fichier config/db.php.
  - La configuration suivante est pour le système utilisé actuellement.

```
<?php  
  
return [  
    'class' => 'yii\db\Connection',  
    'dsn' => 'mysql:host=localhost;dbname=yii2basic',  
    'username' => 'root',  
    'password' => 'password',  
    'charset' => 'utf8',  
];
```



# Préparation du DB

- Pour configurer une connexion DB, vous devez spécifier son DSN (Data Source Name) via la propriété dsn. Le format DSN varie pour différentes bases de données :

# Configuration du SGBD

- ◉ **MySQL, MariaDB** – mysql:host = localhost;dbname = mydb
- ◉ **PostgreSQL** – pgsql:host = localhost;port = 5432;dbname = mydb
- ◉ **SQLite** – sqlite:/path/to/db/file
- ◉ **MS SQL Server (via sqlsrv driver)** – sqlsrv:Server = localhost;Database = mydb
- ◉ **MS SQL Server (via mssql driver)** – mssql:host = localhost;dbname = mydb
- ◉ **MS SQL Server (via dblib driver)** – dblib:host = localhost;dbname = mydb
- ◉ **CUBRID** – cubrid:dbname = mydb;host = localhost;port = 33000
- ◉ **Oracle** – oci:dbname = //localhost:1521/mydb



# Préparation du DB

- Création des tables
  - Méthode ordinaire (phpmyadmin) le cas de SGBD MYSQL

The screenshot shows the 'Structure' tab in phpMyAdmin for creating a new table. The table name is 'table' and it has 1 column. The columns table is as follows:

Nom	Type	Taille/Valeurs*	Défaut	Interclassement	Attributs	Null	Index	A_I	Commentaires	Type MIM
	INT		Aucune			<input type="checkbox"/>	---	<input type="checkbox"/>		
	INT		Aucune			<input type="checkbox"/>	---	<input type="checkbox"/>		
	INT		Aucune			<input type="checkbox"/>	---	<input type="checkbox"/>		

Below the table, there are sections for 'Commentaires sur la table:', 'Moteur de stockage:' (set to InnoDB), 'Interclassement:', and 'Définition de PARTITION:'. A 'Sauvegarder' button is at the bottom right.

- Utiliser le système de migration offert par Yii2 de manière transparente



# Yii - Migration de base de données

- Lors du développement d'une application basée sur la base de données, la structure de la base de données évolue avec le code source. Yii fournit la fonctionnalité de migration de base de données qui vous permet de suivre les modifications de la base de données.

# Yii - Migration de base de données

- ◎ Yii fournit les outils migration suivants (accès par Console):
  - Créer de nouvelles migrations
  - Rétablir les migrations
  - Appliquer les migrations
  - Réappliquer les migrations
  - Afficher le statut et l'historique de la migration



# Création d'une migration

- À l'intérieur de la racine de projet du modèle d'application de base, ouvrez la fenêtre de console et exécutez.

```
D:\wamp\www\coures.dev\basic>yii migrate/create table_product
```

```
vagrant@local:/var/www/coures.dev/basic$ ./yii migrate/create table_product
Yii Migration Tool (based on Yii v2.0.10)

Create new migration '/var/www/coures.dev/basic/migrations/m170106_223146_table_product.php'? (yes|no)
[no]:yes
New migration created successfully.
vagrant@local:/var/www/coures.dev/basic$
```



# Création d'une migration

```
<?php

use yii\db\Migration;

class m170106_223146_table_product extends Migration
{
    public function up()
    {

    }

    public function down()
    {
        echo "m170106_223146_table_product cannot be reverted.\n";

        return false;
    }

    /**
     * Use safeUp/safeDown to run migration code within a transaction
     */
    public function safeUp()
    {
    }

    public function safeDown()
    {
    }
}

*/
```

# Création d'une migration

## ⦿ Ajouter des définitions des tables

```
public function up() {  
    $columns = [  
        'id'      => $this->primaryKey(),  
        'preNom' => $this->string( 100 ),  
        'nom'     => $this->string( 100 )  
    ];  
    $this->createTable( 'product', $columns );  
}
```

## ⦿ Pour mettre à niveau une base de données, exécutez cette commande.

**Yii migrate**

Voir également

Pour en savoir plus sur le système de migration, vous pouvez utiliser les ressources suivantes:

<http://www.yiiframework.com/doc-2.0/guide-db-migrations.html>

# SQL via DAO

• Pour exécuter une requête SQL, nous devons suivre ces étapes

1. Créez `yii\db\Command` avec une requête SQL.
2. Paramètres de liaison(**Binding**) (non requis)
3. Exécutez la commande.

```
$query = \Yii::$app->db->createCommand( 'SELECT * FROM product WHERE id=:id' );
```

```
$query->bindValue(':id',1);
```

```
$products = $query->queryAll();
```

```
var_dump( $products );
```

1

2

3



# Différent commandes pour récupérer les données

```
// retourne un ensemble de lignes. Chaque ligne est un tableau associatif de noms de colonnes et de valeurs.  
// un tableau vide est renvoyé si la requête ne renvoie aucun résultat  
$query      = \Yii::$app->db->createCommand( 'SELECT * FROM product' );  
$products   = $query->queryAll();  
var_dump( $products );  
  
// renvoie une seule ligne (la première ligne)  
// false est renvoyé si la requête n'a aucun résultat  
$product    = \Yii::$app->db->createCommand( 'SELECT * FROM product WHERE id=1' )->queryOne();  
var_dump( $product );  
// retourne une seule colonne (la première colonne)  
// un tableau vide est renvoyé si la requête ne renvoie aucun résultat  
$productName = \Yii::$app->db->createCommand( 'SELECT name FROM product' )->queryColumn();  
var_dump( $productName );  
// retourne une valeur scalaire  
// false est renvoyé si la requête n'a aucun résultat  
$count      = \Yii::$app->db->createCommand( 'SELECT COUNT(*) FROM product' )->queryScalar();  
var_dump( $count );
```

# Autre façon d'envoyer les paramètres de la requête

```
$params = [  
    ':name' => 'La',  
    ':qty'  => 5  
];  
$query = \Yii::$app->db->createCommand( "SELECT * FROM product WHERE name LIKE ':%name%' AND stock >:qty", $params  
);  
  
$products = $query->queryAll();  
var_dump( $products );
```

```
$query = \Yii::$app->db->createCommand( "SELECT * FROM product WHERE name LIKE ':%name%' AND stock >:qty" );  
$query->bindValues( [  
    ':name' => 'La',  
    ':qty'  => 5  
]);  
$products = $query->queryAll();
```

# Requêtes INSERT, UPDATE et DELETE

- Pour les requêtes INSERT, UPDATE et DELETE, vous pouvez appeler les méthodes insert(), update() et delete() de l'objet yii\db\Command

```
// INSERT (table name, column values)
Yii::$app->db
    ->createCommand()
    ->insert('product', [
        'name' => 'Lait',
    ])->execute();
// UPDATE (table name, column values, condition)
Yii::$app->db
    ->createCommand()
    ->update('product', ['name' => 'Parfum'], 'name = "Lait"')
    ->execute();
// DELETE (table name, condition)
Yii::$app->db->createCommand()
    ->delete('product', 'name = "Parfum"')
    ->execute();
```



# Interroger la base de donne avec 'Query Builder'

- ◉ Le générateur de requêtes(Query Builder) nous permet de créer des requêtes SQL de manière programmatique.
- ◉ Le générateur de requêtes nous aide à écrire du code SQL plus lisible.
- ◉ Pour utiliser le générateur de requêtes, suivez ces étapes:
  1. Construire un objet `yii\db\Query`.
  2. Construire la requête.
  3. Exécuter une méthode de requête.

# Interroger la base de donne avec 'Query Builder'

```
//générer "SELECT id, name, email FROM user WHERE name = 'User10';
```

```
$query = ( new \yii\db\Query() );
```

1

```
$query->select( [ 'id', 'name', 'email' ] )  
->from( 'user' )  
->where( [ 'name' => 'User10' ] );
```

2

```
$user = $query->one();
```

3



# QueryBuilder (Where() )

- La fonction where () définit le fragment WHERE d'une requête. Pour spécifier une condition WHERE, vous pouvez utiliser trois formats:
  - Format 'string'- 'name = User10'
  - Format tableau- ['name' => 'User10', 'email' => user10@gmail.com']
  - Format opérateur - ['like', 'name', 'User']



# QueryBuilder (Where() )

## Example

```
//Format 'string'
$product = (new \yii\db\Query())
    ->select(['id', 'name', 'qty'])
    ->from('product')
    ->where('name = :name', [':name' => 'Lait'])
    ->one();
var_dump($product);
//Format tableau
$product = (new \yii\db\Query())
    ->select(['id', 'name', 'qty'])
    ->from('product')
    ->where(['qty'=>'5'])
    ->one();
var_dump($product);
//Format opérateur
$product = (new \yii\db\Query())
    ->select(['id', 'name', 'qty'])
    ->from('product')
    ->where(['like', 'name'=>'Vélo'])
    ->one();
var_dump($product);
```

# Format opérateur

- Le format opérateur nous permet de définir des conditions arbitraires dans le format suivant:

[operator, operand1, operand2]

- L'opérateur peut être:

# L'opérateur peut être:

- ⦿ 'and' - ['and', 'id = 1', 'id = 2'] générera "id = 1 AND id = 2"
- ⦿ 'or' - similaire à l'opérateur 'and'
- ⦿ 'between' - ['between', 'id', 1, 15] générera "id BETWEEN 1 AND 15"
- ⦿ 'not between' - similaire à l'opérateur between, mais BETWEEN est remplacé par NOT BETWEEN
- ⦿ 'in' - ['in', 'id', [5, 10, 15]] générera "id IN (5, 10, 15)"
- ⦿ 'not in' - similaire à l'opérateur in, mais IN est remplacé par NOT IN
- ⦿ 'like' - ['like', 'name', 'user'] générera "name LIKE '%user%'"
- ⦿ 'or like' - similaire à l'opérateur 'like', mais OR est utilisé pour diviser les prédicats LIKE
- ⦿ 'not like' - similaire à l'opérateur 'like', mais LIKE est remplacé par NOT LIKE
- ⦿ 'or not like' - semblable à l'opérateur 'not like', mais OR est utilisé pour concaténer les prédicats NOT LIKE
- ⦿ 'exists' - requiert un opérande qui doit être une instance de la classe yii\db\Query(Sous requete)
- ⦿ 'not exists' - similaire à l'opérateur 'exists', mais construit une expression NOT EXISTS (Sous requete)
- ⦿ '<, <=, >, > =, Ou tout autre opérateur DB: ['<', 'id', 10] générera "id <10"



# Plusieurs conditions

```
$product = (new \yii\db\Query())  
->select(['id', 'name', 'qty'])  
->from('product')  
->where('name = :name', [':name' => 'Lait'])  
->andWhere('qty>2')  
->all();
```

# Format de récupération des données

- La classe `yii\db\Query` fournit un ensemble de méthodes :
- `All ()` - Renvoie un tableau de rangées de paires nom-valeur.
- `One ()` - Renvoie la première ligne.
- `Column ()` - Retourne la première colonne.
- `Scalar ()` - Retourne une valeur scalaire de la première ligne et de la première colonne du résultat.
- `Exists ()` - Retourne une valeur indiquant si la requête contient un résultat
- `Count ()` Renvoie le résultat d'une COUNT requête
- ....





# Active Record

- ◉ Active Record fournit une API orientée objet pour accéder aux données.
- ◉ Une classe Active Record est associée à une table de base de données.



# Déclaration de classes 'ActiveRecord'

- ⦿ Déclarer une classe Active Record en héritant yii\db\ActiveRecord.
- ⦿ Définition d'un nom de table

```
<?php

namespace app\models;

use yii\db\ActiveRecord;

class Product extends ActiveRecord {
    public static function tableName() {
        return 'product';
    }
}
```

# Accès aux données

- ◉ Les données ramenées de la base de données sont remplies en instances Active Record et chaque ligne du résultat de la requête correspond à une seule instance Active Record.
- ◉ Nous pouvons accéder aux valeurs des colonnes en accédant aux attributs des instances Active Record,

```
<?php

namespace app\controllers;

use app\models\Product;
use yii\web\Controller;

class ProductController extends Controller {
    public function actionQuery(){
        //obtenir le produit id = 5
        $product= Product::findOne(5);
        echo $product->nom;
    }
}
```



# Enregistrer les données dans la base de données

- Pour enregistrer des données dans la base de données, il faut appeler la méthode `yii\db\ActiveRecord::save()`.

```
$product = new Product();  
$product->nom = 'Lait';  
if($product->save()){  
    echo 'Validation et l'enregistrement des données OK';  
}else{  
    echo 'Erreur';  
    var_dump($product->getErrors());  
}
```

- Mettre à jour de données existante

```
$product = Product::findOne(1);  
  
if($product){  
    $product->nom = 'Parfum';  
    if($product->save()){  
        echo 'Validation et l'enregistrement des données OK';  
    }else{  
        echo 'Erreur';  
        var_dump($product->getErrors());  
    }  
}
```

# Supprimer

- Pour supprimer il faut :
  1. Récupérer l'instance Active Record
  2. Appelez la méthode « `yii\db\ActiveRecord::delete()` »

```
$product = Product::findOne(1);
if($product){
    try{
        if($product->delete()){
            echo 'supprimé avec succès';
        }else{
            echo 'Erreur';
            var_dump($product->getErrors());
        }
    }catch (\yii\db\Exception $e){
        echo 'Erreur generé par SGBD';
    }
}
```

- Supprimer plusieurs lignes de données:

```
Product::deleteAll(['qty<5']);
```

# Sessions

- La gestion de session offerte par composant « session »

```
$session = \Yii::$app->session;
```

- Opérations:

```
$session = \Yii::$app->session;

// Définir une variable de session
$session->set('var', 'value');

// Obtenir une variable de session
$language = $session->get('var');
var_dump($language);

// Supprimer une variable de session
$session->remove('var');

// Vérifier si une variable de session existe
if (!$session->has('var')) echo "var no existe";
```



# Authentification

- ⦿ Le processus de vérification de l'identité d'un utilisateur est appelé « authentification ».
- ⦿ Il utilise généralement un nom d'utilisateur et un mot de passe pour juger si l'utilisateur est celui qu'il prétend comme.
- ⦿ Pour utiliser le framework d'authentification Yii, nous devons:
  - Configurez le composant d'application utilisateur.
  - Implémentez l'interface `yii\web\IdentityInterface`.

# Autorisation (Filtre de contrôle d'accès)

- ◉ Le processus de vérification que l'utilisateur a suffisamment d'autorisation pour faire quelque chose,
- ◉ Yii fournit un ACF (Access Control Filter), une méthode d'autorisation implémentée sous `yii\filters\AccessControl`

# Exemple

```
public function behaviors() {  
    return [  
        'access' => [  
            'class' => AccessControl::className(),  
            'only' => ['about', 'contact'],  
            'rules' => [  
                [  
                    'allow' => true,  
                    'actions' => ['about'],  
                    'roles' => ['?'],  
                ],  
                [  
                    'allow' => true,  
                    'actions' => ['contact', 'about'],  
                    'roles' => ['@'],  
                ],  
            ],  
        ],  
    ];  
}
```



# Les règles d'accès

- Les règles d'accès prennent en charge de nombreuses options :
  - allow - Définit s'il s'agit d'une règle "allow" ou "deny".
  - actions - Définit les actions auxquelles correspond cette règle.
  - controllers - Définit les contrôleurs auxquels correspond cette règle.
  - roles - Définit les rôles utilisateur auxquels correspond cette règle. Deux rôles spéciaux sont reconnus -
    - ? - correspond à un utilisateur invité.
    - @ - correspond à un utilisateur authentifié.
  - ips - Définit les adresses IP auxquelles correspond cette règle.
  - verbs - Définit la méthode de requête (POST, GET, PUT, etc.) que cette règle correspond.
  - matchCallback - Définit une fonction callable PHP qui doit être appelée pour vérifier si cette règle doit être appliquée.
  - denyCallback - Définit une fonction callable PHP qui doit être appelée lorsque cette règle refuse l'accès.