FREE eBook

LEARNING yii2

Free unaffiliated eBook created from
Stack Overflow contributors.

#yii2

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: yii2

It is an unofficial and free yii2 ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official yii2.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with yii2

Remarks

Yii is a generic Web programming framework, meaning that it can be used for developing all kinds of Web applications using PHP. Because of its component-based architecture and sophisticated caching support, it is especially suitable for developing large-scale applications such as portals, forums, content management systems (CMS), e-commerce projects, RESTful Web services, and so on.

Versions

Version	Release Date
2.0.12	2017-06-05
2.0.11	2017-02-01
2.0.10	2016-10-20
2.0.9	2016-07-11
2.0.8	2016-04-28
2.0.7	2016-02-14
2.0.6	2015-08-06
2.0.5	2015-07-11
2.0.4	2015-05-10
2.0.3	2015-03-01
2.0.2	2015-01-11
2.0.1	2014-12-07
2.0.0	2014-10-12

Examples

Installation or Setup

Yii2 can be installed in two ways. They are

1. Installing via Composer
2. Installing from an Archive File

Installing via Composer

Installing Composer

If you do not already have Composer installed, you may do so by following the instructions at getcomposer.org. On Linux and Mac OS X, you'll run the following commands:

```
curl -sS https://getcomposer.org/installer | php
mv composer.phar /usr/local/bin/composer
```

For windows just download and install composer-setup.exe You may have to configure the github API access token to overide Github API rate limit.

Installing Yii

With Composer installed, you can install Yii by running the following commands under a Web-accessible folder:

```
composer global require "fxp/composer-asset-plugin:^1.2.0"
composer create-project --prefer-dist yiisoft/yii2-app-basic basic
```

then run the following command to install Yii2 with basic template.

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

To install Yii2 with advanced template run

```
composer create-project --prefer-dist --stability=dev yiisoft/yii2-app-advanced advanced
cd advanced
php init
```

After that create a new database and adjust the components['db'] configuration in common/config/main-local.php accordingly. then run the following command to

```
php yii migrate
```

Installing from an Archive File

1. Download the archive file from Yii-download
2. Unpack the downloaded file to a Web-accessible folder.

3. Modify the config/web.php file by entering a secret key for the cookieValidationKey configuration item

   You can add any type of key you want:

   ```
   'cookieValidationKey' => '',

   For example : xyctuyvibonp

   'cookieValidationKey' => 'xyctuyvibonp',
   ```

```
 //insert a secret key in the following (if it is empty) - this is required by cookie
validation
'cookieValidationKey' => 'enter your secret key here',
```

Install Yii2 advanced in ubuntu

First we need to install composer. Steps to install composer Install Composer.

```
curl -sS https://getcomposer.org/installer | php
```

Now change directory:

```
sudo mv composer.phar /usr/local/bin/composer
```

Check composer working

```
composer
```

Now Composer installed.

There two ways to install Yii2 advance.

1.Installing from an Archive File

Get zip file from below link.

Unzip it into destination directory, e.g. `/var/www/html`.

https://github.com/yiisoft/yii2/releases/download/2.0.8/yii-advanced-app-2.0.8.tgz

Move inside the "advanced" folder. Move manually or type below command.

```
cd advanced
```

Run below command.

```
php init
```

2.Installing via Composer

Installing via composer require github authentication token. For token you need to sign up on GitHub.

After signup you can generate your token :

Steps to generate a token

1. In the top right corner of any page, click your profile photo, then click Settings.
2. In the user settings sidebar, click Personal access tokens.
3. Click Generate new token.
4. Give your token a descriptive name.
5. Select the scopes you wish to grant to this token.
6. Click Generate token.
7. Copy the token to your clipboard. For security reasons, after you navigate off this page, no one will be able to see the token again.

Reference : https://help.github.com/articles/creating-an-access-token-for-command-line-use/

After Generating token copy it

Change directory

```
cd /var/www/html/
```

Run below command

```
composer config -g github-oauth.github.com <AuthToken>
```

example:

```
composer config -g github-oauth.github.com f1eefb8f188c22dd6467f1883cb2615c194d1ce1
```

Install yii2

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

Move inside the "advanced" folder. Move manually or type below command.

```
cd advanced
```

Run below command.

```
php init
```

Its done!

Now you can check it.

http://localhost/advanced/frontend/web

and

http://localhost/advanced/backend/web

Read Getting started with yii2 online: https://riptutorial.com/yii2/topic/788/getting-started-with-yii2

Chapter 2: Active Record

Remarks

AR is perfect when you need to delete, update or create one or more records sequentially. Its support of dirty attributes (saving only what was really changed) results in optimized UPDATE statements which lifts the load from database significantly and reduces chances for various conflicts connected with editing same record by multiple persons at the same time.

If you don't have really complex logic in your application and therefore it doesn't require abstracting entities, AR is the best fit for deletes, updates and creates.

AR is also OK for simple queries resulting in under 100 records per page. It's not as performant as working with arrays produced by query builder or asArray() but is more pleasure to work with.

AR is not recommended for complex queries. These are usually about aggregating or transforming data so what's returned doesn't fit AR model anyway. It is preferable to use query builder in this case.

Same goes for import and export. Better to use query builder because of high amounts of data and possibly complex queries.

Examples

Find all records

```
Post::find()->all();
// SELECT * FROM post
```

or the shorthand

(Returns an active record model instance by a primary key or an array of column values.)

```
Post::findAll(condition);
```

returns an array of ActiveRecord instances.

Find All with where Condition

```
$model = User::find()
        ->where(['id' => $id])
        ->andWhere('status = :status', [':status' => $status])
        ->all();
```

Find All with orderBy

```
$model = User::find()
         ->orderBy(['id'=>SORT_DESC])
        ->all();
Or

$model = User::find()
         ->orderBy(['id'=>SORT_ASC])
        ->all();
```

Where Clause

OPERATORS

```
$postsGreaterThan = Post::find()->where(['>', 'created_at', '2016-01-25'])->all();
// SELECT * FROM post WHERE created_at > '2016-01-25'

$postsLessThan = Post::find()->where(['<', 'created_at', '2016-01-25'])->all();
// SELECT * FROM post WHERE created_at < '2016-01-25'

$postsNotEqual = Post::find()->where(['<>', 'created_at', '2016-01-25'])->all();
// SELECT * FROM post WHERE created_at <> '2016-01-25'
```

IN

```
$postsInArray = Post::find()->where(['id' => [1,2,3]])->all();
// SELECT * FROM post WHERE id IN (1,2,3)
```

BETWEEN

```
$postsInBetween = Post::find()
->where(['between', 'date', "2015-06-21", "2015-06-27" ])
->all();
```

NULL

```
$postsWithNullTitle = Post::find()->where(['title' => null]);
// SELECT * FROM post WHERE title IS NULL
```

AND

```
$postsAND = Post::find()->where(['title' => null, 'body' => null]);
// SELECT * FROM post WHERE title IS NULL AND body IS NULL
```

OR

```
$postsAND = Post::find()->where(['OR', 'title IS NULL', 'body IS NULL']);
// SELECT * FROM post WHERE title IS NULL OR body IS NULL
```

NOT

```
 $postsNotEqual = Post::find()->where(['NOT', ['created_at'=>'2016-01-25']])->all();
```

```
// SELECT * FROM post WHERE created_at IS NOT '2016-01-25'
```

NESTED CLAUSES

```
$postsNestedWhere = Post::find()->andWhere([
    'or',
    ['title' => null],
    ['body' => null]
])->orWhere([
    'and',
    ['not', ['title' => null]],
    ['body' => null]
]);
// SELECT * FROM post WHERE (title IS NULL OR body IS NULL) OR (title IS NOT NULL AND body IS
NULL)
```

LIKE OPERATOR with filterWhere activerecord methods

For example, in search filter you want to filter the post by searing post title or description posted by currently logged in user.

```
$title = 'test';
$description = 'test';
```

i) andFilterWhere()

```
$postLIKE = Post::find()->where(['user_id' => Yii::$app->user->getId()])-
>andfilterWhere(['or', ['title' => $title, 'description' => $description]])->all();
//SELECT * FROM post WHERE user_id = 2 AND ((`title` LIKE '%test%') OR (`description` LIKE
'%test%'))
```

ii) orFilterWhere()

```
$postLIKE = Post::find()->where(['user_id' => Yii::$app->user->getId()])->orFilterWhere(['or',
['title' => $title, 'description' => $description]])->all();
//SELECT * FROM post WHERE user_id = 2 OR ((`title` LIKE '%test%') OR (`description` LIKE
'%test%'))
```

iii) filterWhere()

```
$postLIKE = Post::find()->filterWhere(['AND', ['title' => $title, 'description' =>
$description]])->andWhere(['user_id' => Yii::$app->user->getId()])->all();
//SELECT * FROM post WHERE ((`title` LIKE '%test%') AND (`description` LIKE '%test%')) AND
user_id = 2
```

Note: While using filterWhere() we have to call all andwhere() or orWhere() after the filterWhere() otherwise all where conditions will get remove except filterWhere()

Create an ActiveRecord class with events based fields value

```
    <?php
```

```
    namespace models;

    use yii\db\ActiveRecord;
    use yii\behaviors\TimestampBehavior;

    class Post extends ActiveRecord
    {
        public static function tableName()
        {
            return 'post';
        }

        public function rules() {
            return [
                [['created_at', 'updated_at'], 'safe'],
            ];
        }
        public function behaviors() {
            parent::behaviors();

            return [
              'timestamp' => [
                'class' => TimestampBehavior::className(),
                'attributes' => [
                  ActiveRecord::EVENT_BEFORE_INSERT => ['created_at', 'updated_at'],
                  ActiveRecord::EVENT_BEFORE_UPDATE => ['updated_at']
                ],
                'value' => date('Y-m-d H:i:s'),
              ]
            ];
          }
    }
```

Or this can be used

```
public function beforeSave($insert)
{
    if($this->isNewRecord){
        //When create
    }else{
         //When update
    }

    return parent::beforeSave($insert);
}

  public function afterSave($insert, $changedAttributes )
{
    if($insert){
        //When create
    }else{
         //When update
    }
    return parent::afterSave($insert, $changedAttributes);
}
```

Find one record

```
$customer = Customer::findOne(10);
```

or

```
$customer = Customer::find()->where(['id' => 10])->one();
```

or

```
$customer = Customer::find()->select('name,age')->where(['id' => 10])->one();
```

or

```
$customer = Customer::findOne(['age' => 30, 'status' => 1]);
```

or

```
$customer = Customer::find()->where(['age' => 30, 'status' => 1])->one();
```

Find One Queries

Find single record based on id.

```
$model = User::findOne($id);
```

Select single column based on id.

```
$model = User::findOne($id)->name;
```

Retrieve the single record from the database based on condition.

```
$model = User::find()->one();  // give first record

$model = User::find()->where(['id' => 2])->one(); // give single record based on id
```

Select single columns record from the database based on condition.

```
$model = User::find()->select('name,email_id')->where(['id' => 1])->one();
```

OR

```
$model = User::find()->select(['id','name','email_id'])->where(['id' => 1])->one();
```

OrderBy

```
$model = User::find()->select(['id','name','email_id'])->orderBy(['id' => SORT_DESC])->one();

OR
```

```
$model = User::find()->select(['id','name','email_id'])->orderBy(['id' => SORT_ASC])->one();
```

Active records with sub queries

Example: Customers who can create a post. Each customer can create multiple posts. As soon as customer creates the post, post will be under administrators review. Now we have to fetch the customers list who have all active posts by using sub-query.

Note: If a customer has 5 post, among 5 posts if he has at least one inactive, we have to exclude this customer from the customers list.

```
$subQuery = Post::find()->select(['customer_id'])->where(['status' => 2]); //fetch the
customers whos posts are inactive - subquery
$query = Customer::find()->where(['NOT IN', 'id', $subQuery])->all(); //Exclude the customers
whos posts are inactive by using subquery
```

Read Active Record online: https://riptutorial.com/yii2/topic/1516/active-record

Chapter 3: Advanced Project Template

Examples

Deployment in shared hosting environment

Deploying an advanced project template to shared hosting is a bit trickier than a basic one because it has two webroots, which shared hosting webservers don't support. We will need to adjust the directory structure so frontend URL will be http://site.local and backend URL will be http://site.local/admin.

Move entry scripts into single webroot

First of all we need a webroot directory. Create a new directory and name it to match your hosting webroot name, e.g., www or public_html or the like. Then create the following structure where www is the hosting webroot directory you just created:

```
www
    admin
backend
common
console
environments
frontend
...
```

> **www** will be our frontend directory so move the contents of **frontend/web** into it. Move the contents of **backend/web** into **www/admin**. In each case you will need to adjust the paths in **index.php** and **index-test.php**.

Adjust sessions and cookies

Originally the backend and frontend are intended to run at different domains. When we're moving it all to the same domain the frontend and backend will be sharing the same cookies, creating a clash. In order to fix it, adjust backend application **config backend/config/main.php** as follows:

```
'components' => [
    'request' => [
        'csrfParam' => '_csrf-backend',
        'csrfCookie' => [
            'httpOnly' => true,
            'path' => '/admin',
        ],
    ],
    'user' => [
        'identityClass' => 'common\models\User',
        'enableAutoLogin' => true,
        'identityCookie' => [
```

```
            'name' => '_identity-backend',
            'path' => '/admin',
            'httpOnly' => true,
        ],
    ],
    'session' => [
        // this is the name of the session cookie used for login on the backend
        'name' => 'advanced-backend',
        'cookieParams' => [
            'path' => '/admin',
        ],
    ],
],
```

Hope this helps for the shared hosting users to deploy advanced application.

> credits : https://github.com/yiisoft/yii2-app-advanced/blob/master/docs/guide/topic-shared-hosting.md

Sharing uploaded files between the frontend and backend using symlinks

So you've uploaded your files to a folder say `/backend/web/uploads/` and you want these uploads to be visible on the frontend too. The easiest option is to create a symlink in the frontend that links to the backend:

```
ln -s /path/to/backend/web/uploads/ /path/to/frontend/web/uploads
```

In your views you can use relative links to the files now:

```
<img src='/uploads/<?= $model->image?>' alt='My Image goes here'>
<a href='/uploads/<?= $model->filename?>' target='_blank'>Download File</a>
```

Ensure that your webserver allows symlinks to be followed.

Read Advanced Project Template online: https://riptutorial.com/yii2/topic/944/advanced-project-template

Chapter 4: Ajax Request

Examples

Submitting Ajax form

View file:

```
<?php
use yii;
use yii\bootstrap\ActiveForm;
use yii\helpers\Html;
?>

<?php
$form = ActiveForm::begin([
    'action' => ['comments/ajax-comment'],
    'options' => [
        'class' => 'comment-form'
    ]
]);
?>
    <?= $form->field($model, 'comment'); ?>

    <?= Html::submitButton("Submit", ['class' => "btn"]); ?>

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

Javascript:

```
 jQuery(document).ready(function($) {
       $(".comment-form").submit(function(event) {
            event.preventDefault(); // stopping submitting
            var data = $(this).serializeArray();
            var url = $(this).attr('action');
            $.ajax({
                url: url,
                type: 'post',
                dataType: 'json',
                data: data
            })
            .done(function(response) {
                if (response.data.success == true) {
                    alert("Wow you commented");
                }
            })
            .fail(function() {
                console.log("error");
            });

        });
    });
```

Controller Action:

```
public function actionAjaxComment()
{
    $model = new Comments();
    if (Yii::$app->request->isAjax) {
        Yii::$app->response->format = \yii\web\Response::FORMAT_JSON;

        if ($model->load(Yii::$app->requset->post()) && $model->save()) {
            return [
                'data' => [
                    'success' => true,
                    'model' => $model,
                    'message' => 'Model has been saved.',
                ],
                'code' => 0,
            ];
        } else {
            return [
                'data' => [
                    'success' => false,
                    'model' => null,
                    'message' => 'An error occured.',
                ],
                'code' => 1, // Some semantic codes that you know them for yourself
            ];
        }
    }
}
```

Render Ajax view

`Controller::renderAjax()` method can be used to respond to an Ajax request. This method is similar to renderPartial() except that it will inject into the rendering result with JS/CSS scripts and files which are registered with the view

Assume we have login form in a view file:

```
<?php
use yii\helpers\Html;
use yii\bootstrap\ActiveForm;

\yii\bootstrap\BootstrapAsset::register($this);

<div class="site-login">

    <?php $form = ActiveForm::begin(); ?>

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

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

        <?= Html::submitButton('Login',['class' => 'btn btn-primary btn-block']) ?>

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

When we use `renderPartial()` in a controller action:

```
public function actionLogin()
{
    $model = new LoginForm();
    if ($model->load(Yii::$app->request->post()) && $model->login()) {
        return $this->goBack();
    }
    return $this->renderPartial('login', [
        'model' => $model,
    ]);
}
```

Action output:

```
<div class="site-login">
    <form id="w0" action="/site/login" method="post" role="form">
        <div class="form-group field-loginform-username required">
            <label class="control-label" for="loginform-username">Имя пользователя</label>
            <input type="text" id="loginform-username" class="form-control"
name="LoginForm[username]">
        </div>
        <div class="form-group field-loginform-password required">
            <label class="control-label" for="loginform-password">Пароль</label>
            <input type="password" id="loginform-password" class="form-control"
name="LoginForm[password]">
        </div>
        <button type="submit" class="btn btn-primary btn-block">Login</button>
    </form>
</div>
```

When we use `renderAjax()` in a controller action:

```
    ...
    return $this->renderAjax('login', [
        'model' => $model,
    ]);
    ...
```

Action output (JS,CSS injected):

```
<link href="/assets/f1759119/css/bootstrap.css" rel="stylesheet">
<div class="site-login">
    <form id="w0" action="/site/login" method="post" role="form">
        <div class="form-group field-loginform-username required">
            <label class="control-label" for="loginform-username">Имя пользователя</label>
            <input type="text" id="loginform-username" class="form-control"
name="LoginForm[username]">
        </div>
        <div class="form-group field-loginform-password required">
            <label class="control-label" for="loginform-password">Пароль</label>
            <input type="password" id="loginform-password" class="form-control"
name="LoginForm[password]">
        </div>
        <button type="submit" class="btn btn-primary btn-block">Login</button>
    </form>
</div>
<script src="/assets/13aa7b5d/jquery.js"></script>
<script src="/assets/302a2946/yii.js"></script>
```

```
<script src="/assets/302a2946/yii.validation.js"></script>
<script src="/assets/302a2946/yii.activeForm.js"></script>
```

If we want to exclude some assets from view (to prevent dublicates):

```
...
Yii::$app->assetManager->bundles = [
    'yii\bootstrap\BootstrapAsset' => false,
];
return $this->renderAjax('login', [
    'model' => $model,
]);
...
```

Action output (no bootstrap.css):

```
<div class="site-login">
    <form id="w0" action="/site/login" method="post" role="form">
        <div class="form-group field-loginform-username required">
            <label class="control-label" for="loginform-username">Имя пользователя</label>
            <input type="text" id="loginform-username" class="form-control"
name="LoginForm[username]">
        </div>
        <div class="form-group field-loginform-password required">
            <label class="control-label" for="loginform-password">Пароль</label>
            <input type="password" id="loginform-password" class="form-control"
name="LoginForm[password]">
        </div>
        <button type="submit" class="btn btn-primary btn-block">Login</button>
    </form>
</div>
<script src="/assets/13aa7b5d/jquery.js"></script>
<script src="/assets/302a2946/yii.js"></script>
<script src="/assets/302a2946/yii.validation.js"></script>
<script src="/assets/302a2946/yii.activeForm.js"></script>
```

Read Ajax Request online: https://riptutorial.com/yii2/topic/2944/ajax-request

Chapter 5: Asset management

Syntax

- the depended assets will be loaded before this assets in given order
    - public $depends = ['yii\web\YiiAsset', 'yii\bootstrap\BootstrapAsset', 'yii\bootstrap\BootstrapPluginAsset', 'cinghie\fontawesome\FontAwesomeAsset',];

Remarks

this example is based on the advanced template https://github.com/yiisoft/yii2-app-advanced

The cinghie asset in this example is the asset package for adminLTE
https://github.com/cinghie/yii2-admin-lte

Examples

This is part of the layout file

```
<?php
/* this example is based on the advanced template
 * This file is located in
 * backend/views/layouts/main.php
 */

use yii\helpers\Html;

// here the asset is registered
use cinghie\adminlte\AdminLTEAsset;
AdminLTEAsset::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 class="hold-transition skin-blue sidebar-mini">

<?php $this->beginBody() ?>

<?= $content ?>

<?php $this->endBody() ?>
```

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

This is the Asset File

```
<?php

/**
* This file is the Asset Bundle File located in
*  vendor/cinghie/yii2-admin-lte/AdminLTEAsset.php
* @copyright Copyright &copy; Gogodigital Srls
* @company Gogodigital Srls - Wide ICT Solutions
* @website http://www.gogodigital.it
* @github https://github.com/cinghie/yii2-admin-lte
* @license GNU GENERAL PUBLIC LICENSE VERSION 3
* @package yii2-AdminLTE
* @version 1.3.10
*/

namespace cinghie\adminlte;

use yii\web\AssetBundle;

/**
 * Class yii2-AdminLTEAsset
 * @package cinghie\adminlte
 */
class AdminLTEAsset extends AssetBundle
{

    /**
     * @inherit
     */
    public $sourcePath = '@bower/';

    /**
     * @inherit
     */
    public $css = [
        'ionicons/css/ionicons.css',
        'admin-lte/dist/css/AdminLTE.css',
        'admin-lte/dist/css/skins/_all-skins.css'
    ];

    /**
     * @inherit
     */
    public $js = [
        'admin-lte/dist/js/app.js'
    ];

    /**
     * @inherit
     */
    public $depends = [
        'yii\web\YiiAsset',
        'yii\bootstrap\BootstrapAsset',
        'yii\bootstrap\BootstrapPluginAsset',
```

```
    'cinghie\fontawesome\FontAwesomeAsset',
    ];

}
```

The generated HTML with automaticaly loaded assets

```
<!DOCTYPE html>
<html lang="en-EN">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="csrf-param" content="_csrf">
    <meta name="csrf-token"
content="M01tTVZLdlBlBQEqGyYcYHc5PwI1CRknfB1bBiAaPTNBfyk0Ehg8EQ==">
    <title>Profil</title>
    <link href="/assets/f3e48cde/css/bootstrap.css?v=1473788138" rel="stylesheet">
<link href="/assets/24e44190/css/font-awesome.css?v=1473866258" rel="stylesheet">
<link href="/assets/fa4335a5/ionicons/css/ionicons.css?v=1473866258" rel="stylesheet">
<link href="/assets/fa4335a5/admin-lte/dist/css/AdminLTE.css?v=1473866258" rel="stylesheet">
<link href="/assets/fa4335a5/admin-lte/dist/css/skins/_all-skins.css?v=1473866258"
rel="stylesheet"></head>
<body class="hold-transition skin-blue sidebar-mini">

....

</script><script src="/assets/69b4ffbe/jquery.js?v=1473788138"></script>
<script src="/assets/6aa8a809/yii.js?v=1473788138"></script>
<script src="/assets/f3e48cde/js/bootstrap.js?v=1473788138"></script>
<script src="/assets/fa4335a5/admin-lte/dist/js/app.js?v=1473866258"></script>

</body>
</html>
```

Read Asset management online: https://riptutorial.com/yii2/topic/6900/asset-management

Chapter 6: Components

Examples

Creating and using application components

Steps to Create component:

- Create a folder named `components` in your project root folder
- Create your component inside components folder e.g.: `MyComponent.php`

```
namespace app\components;

use Yii;
use yii\base\Component;
use yii\base\InvalidConfigException;


class MyComponent extends Component
{
    public function demo()
    {
        return "welcome";
    }
}
```

- Register your component inside the `config/web.php` file

```
components' => [
    'mycomponent' => [
        'class' => 'app\components\MyComponent',
    ],
]
```

Now you can use your component method:

```
namespace app\controllers;

use Yii;

class DemoController extends \yii\web\Controller
{
    public function actionTest()
    {
        echo Yii::$app->mycomponent->demo();
    }
}
```

Dropdown List using component function

Create function in MyComponent.php

```
    namespace app\components;

    use Yii;
    use yii\base\Component;
    use yii\base\InvalidConfigException;
    use yii\helpers\Url;
    use yii\helpers\ArrayHelper;

    use app\models\User;

        class MyComponent extends Component
        {

          // Function return list of  id & user Names,used for dropdownlist
          public function getUserID()
          {
            $code = User::find()->select('id,name')
            ->where(['is_deleted'=>'n'])
            ->all();

            $result = ArrayHelper::map($code, 'id', 'name');
            if($result)
                return $result;
            else
                return ["null"=>"No User"];
          }
        }
```

-> Register Component in web.php

```
components' => [
        'mycomponent' => [
             'class' => 'app\components\MyComponent',
            ],
         ]
```

-> use it in your view

```
<?= $form->field($model, 'user_id')->dropDownList(Yii::$app->mycomponent->getUserID())?>
```

Read Components online: https://riptutorial.com/yii2/topic/2217/components

Chapter 7: Cookies

Remarks

Cookies are part of HTTP request so it's a good idea to do both in controller which responsibility is exactly dealing with request and response.

Examples

Setting a cookie

To set a cookie i.e. to create it and schedule for sending to the browser you need to create new `\yii\web\Cookie` class instance and add it to response cookies collection:

```
$cookie = new Cookie([
    'name' => 'cookie_monster',
    'value' => 'Me want cookie!',
    'expire' => time() + 86400 * 365,
]);
\Yii::$app->getResponse()->getCookies()->add($cookie);
```

In the above we're passing parameters to cookie class constructor. These basically the same as used with native PHP setcookie function:

- `name` - name of the cookie.
- `value` - value of the cookie. Make sure it's a string. Browsers typically aren't happy about binary data in cookies.
- `domain` - domain you're setting the cookie for.
- `expire` - unix timestamp indicating time when the cookie should be automatically deleted.
- `path` - the path on the server in which the cookie will be available on.
- `secure` - if `true`, cookie will be set only if HTTPS is used.
- `httpOnly` - if `true`, cookie will not be available via JavaScript.

Reading a cookie

In order to read a cookie use the following code:

```
$value = \Yii::$app->getRequest()->getCookies()->getValue('my_cookie');
```

Note: this code allows read cookies that has been set using cookie component (because it signs all cookies by default). Therefore if you add/update cookie using JS code, you can't read it using this method (by default).

Cookies for subdomains

Because of security reasons, by default cookies are accessible only on the same domain from

which they were set. For example, if you have set a cookie on domain `example.com`, you cannot get it on domain `www.example.com`. So if you're planning to use subdomains (i.e. admin.example.com, profile.example.com), you need to set `domain` explicitly:

```
$cookie = new Cookie([
    'name' => 'cookie_monster',
    'value' => 'Me want cookie everywhere!',
    'expire' => time() + 86400 * 365,
    'domain' => '.example.com' // <<<=== HERE
]);
\Yii::$app->getResponse()->getCookies()->add($cookie);
```

Now cookie can be read from all subdomains of `example.com`.

Cross-subdomain authentication and identity cookies

In case of autologin or "remember me" cookie, the same quirks as in case of subdomain cookies are applying. But this time you need to configure user component, setting `identityCookie` array to desired cookie config.

Open you application config file and add `identityCookie` parameters to user component configuration:

```
$config = [
    // ...
    'components' => [
        // ...
        'user' => [
            'class' => 'yii\web\User',
            'identityClass' => 'app\models\User',
            'enableAutoLogin' => true,
            'loginUrl' => '/user/login',
            'identityCookie' => [ // <---- here!
                'name' => '_identity',
                'httpOnly' => true,
                'domain' => '.example.com',
            ],
        ],
        'request' => [
            'cookieValidationKey' => 'your_validation_key'
        ],
        'session' => [
            'cookieParams' => [
                'domain' => '.example.com',
                'httpOnly' => true,
            ],
        ],

    ],
];
```

Note that `cookieValidationKey` should be the same for all sub-domains.

Note that you have to configure the `session::cookieParams` property to have the samedomain as your `user::identityCookie` to ensure the `login` and `logout` work for all subdomains. This behavior is

better explained on the next section.

Session cookie parameters

Session cookies parameters are important both if you have a need to maintain session while getting from one subdomain to another or when, in contrary, you host backend app under `/admin` URL and want handle session separately.

```
$config = [
    // ...
    'components' => [
        // ...
        'session' => [
            'name' => 'admin_session',
            'cookieParams' => [
                'httpOnly' => true,
                'path' => '/admin',
            ],
        ],
    ],
];
```

Read Cookies online: https://riptutorial.com/yii2/topic/2945/cookies

Chapter 8: Custom Validations

Introduction

Yii2 has some built-in validators which can be used while solving programming related or general puprose validations. When you need to create a new business logic validation you need to create your own validators.

Examples

Types of Validations

Let's initially understand basic types of custom validators:

1. Inline Validator
2. Standalone Validator

Inline Validator: It is the type of the validator we create inside the class which is basically a method we define just like other methods but with extra parameters which is passed in by Yii2.

```
....
public function ValidateMyBusiness($attr, $params){
    // adding an error here means our validation is failed.
    if ($this->{$attr} > 1100) {
        $this->addError($attr, "Some error occured");
    }
}
...
// calling above validator is simple as below:
public function rules(){
  return [
     ['money', 'validateMyBusiness', 'params' => ['targetAccount' => $this->account]];
  ]
}

# params array will be passed to our inline parameter as a second argument.
```

Read Custom Validations online: https://riptutorial.com/yii2/topic/9187/custom-validations

Chapter 9: Database Migrations

Examples

Creating Migrations

```
yii migrate/create <name>
```

The required name argument gives a brief description about the new migration. For example, if the migration is about creating a new table named news, you may use the name create_news_table and run the following command

```
yii migrate/create create_news_table
```

Migration File Example

```
<?php

use yii\db\Migration;

class m150101_185401_create_news_table extends Migration
{
public function up()
{

}

public function down()
{
    echo "m101129_185401_create_news_table cannot be reverted.\n";

    return false;
}

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

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

Drop Table

```
public function up()
{
    $this->dropTable('post');
```

```
}
```

Create table fields right away

```
yii migrate/create create_post_table --fields="title:string,body:text"
```

Generates:

```
/**
 * Handles the creation for table `post`.
 */
class m150811_220037_create_post_table extends Migration
{
/**
 * @inheritdoc
 */
public function up()
{
    $this->createTable('post', [
        'id' => $this->primaryKey(),
        'title' => $this->string(),
        'body' => $this->text(),
    ]);
}

/**
 * @inheritdoc
 */
public function down()
{
    $this->dropTable('post');
}
}
```

Create Table

```
public function up()
{
    $this->createTable('post', [
        'id' => $this->primaryKey()
    ]);
}
```

Drop / Rename / Alter Column

```
public function up()
{
    $this->dropColumn('post', 'position');

    $this->renameColumn('post', 'owner_id', 'user_id');

    $this->alterColumn('post', 'updated', $this->timestamp()->notNull()->defaultValue('0000-
00-00 00:00:00'));
}
```

Add Column

```
public function up()
{
    $this->addColumn('post', 'position', $this->integer());
}
```

Reverting Migrations

```
yii migrate/down     # revert the most recently applied migration
yii migrate/down 3   # revert the most 3 recently applied migrations
```

Transactional Migrations

```
public function safeUp()
{
    $this->createTable('news', [
        'id' => $this->primaryKey(),
        'title' => $this->string()->notNull(),
        'content' => $this->text(),
    ]);

    $this->insert('news', [
        'title' => 'test 1',
        'content' => 'content 1',
    ]);
}

public function safeDown()
{
    $this->delete('news', ['id' => 1]);
    $this->dropTable('news');
}
```

An even easier way of implementing transactional migrations is to put migration code in the `safeUp()` and `safeDown()` methods. These two methods differ from `up()` and `down()` in that they are enclosed implicitly in a transaction. As a result, if any operation in these methods fails, all prior operations will be rolled back automatically.

Migrating Multiple Databases

By default, migrations are applied to the same database specified by the db application component. If you want them to be applied to a different database, you may specify the db command-line option like shown below:

```
yii migrate --db=db2
```

Redoing Migrations

```
yii migrate/redo        # redo the last applied migration
```

```
yii migrate/redo 3      # redo the last 3 applied migrations
```

Listing Migrations

```
yii migrate/history     # showing the last 10 applied migrations
yii migrate/history 5   # showing the last 5 applied migrations
yii migrate/history all # showing all applied migrations

yii migrate/new         # showing the first 10 new migrations
yii migrate/new 5       # showing the first 5 new migrations
yii migrate/new all     # showing all new migrations
```

Modifying Migration History

```
yii migrate/mark 150101_185401                      # using timestamp to specify the migration
yii migrate/mark "2015-01-01 18:54:01"              # using a string that can be parsed by
strtotime()
yii migrate/mark m150101_185401_create_news_table   # using full name
yii migrate/mark 1392853618                         # using UNIX timestamp
```

Applying Migrations

```
yii migrate
```

This command will list all migrations that have not been applied so far. If you confirm that you want to apply these migrations, it will run the up() or safeUp() method in every new migration class, one after another, in the order of their timestamp values. If any of the migrations fails, the command will quit without applying the rest of the migrations.

```
yii migrate 3
yii migrate/to 150101_185401                      # using timestamp to specify the migration
yii migrate/to "2015-01-01 18:54:01"              # using a string that can be parsed by
strtotime()
yii migrate/to m150101_185401_create_news_table   # using full name
yii migrate/to 1392853618                         # using UNIX timestamp
```

Read Database Migrations online: https://riptutorial.com/yii2/topic/1929/database-migrations

Chapter 10: File Uploads

Examples

How to do it

Uploading Files

Uploading files in Yii is usually done with the help of [[yii\web\UploadedFile]] which encapsulates each uploaded file as an `UploadedFile` object. Combined with [[yii\widgets\ActiveForm]] and models, you can easily implement a secure file uploading mechanism.

Creating Models

Like working with plain text inputs, to upload a single file you would create a model class and use an attribute of the model to keep the uploaded file instance. You should also declare a validation rule to validate the file upload. For example,

```
namespace app\models;

use yii\base\Model;
use yii\web\UploadedFile;

class UploadForm extends Model
{
    /**
     * @var UploadedFile
     */
    public $imageFile;

    public function rules()
    {
        return [
            [['imageFile'], 'file', 'skipOnEmpty' => false, 'extensions' => 'png, jpg'],
        ];
    }

    public function upload()
    {
        if ($this->validate()) {
            $this->imageFile->saveAs('uploads/' . $this->imageFile->baseName . '.' . $this-
>imageFile->extension);
            return true;
        } else {
            return false;
        }
    }
}
```

In the code above, the `imageFile` attribute is used to keep the uploaded file instance. It is

associated with a `file` validation rule which uses [[yii\validators\FileValidator]] to ensure a file with extension name `png` or `jpg` is uploaded. The `upload()` method will perform the validation and save the uploaded file on the server.

The `file` validator allows you to check file extensions, size, MIME type, etc. Please refer to the Core Validators section for more details.

> Tip: If you are uploading an image, you may consider using the `image` validator instead. The `image` validator is implemented via [[yii\validators\ImageValidator]] which verifies if an attribute has received a valid image that can be then either saved or processed using the Imagine Extension.

Rendering File Input

Next, create a file input in a view:

```
<?php
use yii\widgets\ActiveForm;
?>

<?php $form = ActiveForm::begin(['options' => ['enctype' => 'multipart/form-data']]) ?>

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

    <button>Submit</button>

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

It is important to remember that you add the `enctype` option to the form so that the file can be properly uploaded. The `fileInput()` call will render a `<input type="file">` tag which will allow users to select a file to upload.

> Tip: since version 2.0.8, [[yii\web\widgets\ActiveField::fileInput|fileInput]] adds `enctype` option to the form automatically when file input field is used.

Wiring Up

Now in a controller action, write the code to wire up the model and the view to implement file uploading:

```
namespace app\controllers;

use Yii;
use yii\web\Controller;
use app\models\UploadForm;
use yii\web\UploadedFile;

class SiteController extends Controller
{
    public function actionUpload()
    {
```

```
        $model = new UploadForm();

        if (Yii::$app->request->isPost) {
            $model->imageFile = UploadedFile::getInstance($model, 'imageFile');
            if ($model->upload()) {
                // file is uploaded successfully
                return;
            }
        }

        return $this->render('upload', ['model' => $model]);
    }
}
```

In the above code, when the form is submitted, the [[yii\web\UploadedFile::getInstance()]] method is called to represent the uploaded file as an `UploadedFile` instance. We then rely on the model validation to make sure the uploaded file is valid and save the file on the server.

Uploading Multiple Files

You can also upload multiple files at once, with some adjustments to the code listed in the previous subsections.

First you should adjust the model class by adding the `maxFiles` option in the `file` validation rule to limit the maximum number of files allowed to upload. Setting `maxFiles` to `0` means there is no limit on the number of files that can be uploaded simultaneously. The maximum number of files allowed to be uploaded simultaneously is also limited with PHP directive `max_file_uploads`, which defaults to 20. The `upload()` method should also be updated to save the uploaded files one by one.

```
namespace app\models;

use yii\base\Model;
use yii\web\UploadedFile;

class UploadForm extends Model
{
    /**
     * @var UploadedFile[]
     */
    public $imageFiles;

    public function rules()
    {
        return [
            [['imageFiles'], 'file', 'skipOnEmpty' => false, 'extensions' => 'png, jpg',
'maxFiles' => 4],
        ];
    }

    public function upload()
    {
        if ($this->validate()) {
            foreach ($this->imageFiles as $file) {
                $file->saveAs('uploads/' . $file->baseName . '.' . $file->extension);
            }
```

```
            return true;
        } else {
            return false;
        }
    }
}
```

In the view file, you should add the `multiple` option to the `fileInput()` call so that the file upload field can receive multiple files:

```
<?php
use yii\widgets\ActiveForm;
?>

<?php $form = ActiveForm::begin(['options' => ['enctype' => 'multipart/form-data']]) ?>

    <?= $form->field($model, 'imageFiles[]')->fileInput(['multiple' => true, 'accept' =>
'image/*']) ?>

    <button>Submit</button>

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

And finally in the controller action, you should call `UploadedFile::getInstances()` instead of `UploadedFile::getInstance()` to assign an array of `UploadedFile` instances to `UploadForm::imageFiles`.

```
namespace app\controllers;

use Yii;
use yii\web\Controller;
use app\models\UploadForm;
use yii\web\UploadedFile;

class SiteController extends Controller
{
    public function actionUpload()
    {
        $model = new UploadForm();

        if (Yii::$app->request->isPost) {
            $model->imageFiles = UploadedFile::getInstances($model, 'imageFiles');
            if ($model->upload()) {
                // file is uploaded successfully
                return;
            }
        }

        return $this->render('upload', ['model' => $model]);
    }
}
```

Read File Uploads online: https://riptutorial.com/yii2/topic/2221/file-uploads

Chapter 11: Installing Extension Manually

Examples

Install Extension without Composer

> Note: it's strongly advised to use Composer. The instruction below is basically what Composer does for you.

- Download archive extension file of needed version from Github
- Open `composer.json`
- Find `PSR-4` autoload section and remember it for e.g. `kmit/select2`
- Extract files to corresponding folder in vendor folder, like `vendor/kmit/select2`
- Add following code to `vendor/composer/autoload_psr4.php`

```
'kmit\\select2\\' => array($vendorDir . '/kmit/select2'),
```

- Add following code to `vendor/yiisoft/extensions.php`:

```
'kmit/select2'(name of extension from composer.json file of extension) =>
    array (
        'name' => 'kmit/select2',
        'version' => '1.0.0.0',
        'alias' => array (
            '@vendor/kmit/select2'(path of extension folder alias) => $vendorDir .
'/kmit/select2' (path of extension folder),
        ),
    ),
```

Video Tutroial

Read Installing Extension Manually online: https://riptutorial.com/yii2/topic/2224/installing-extension-manually

Chapter 12: Pjax

Examples

Step 1 Add Structure

In views\site\form-submission.php

```
<?php Pjax::begin(['id'=>'id-pjax']); ?>
<?= Html::beginForm(['site/form-submission'], 'post', ['data-pjax' => '', 'class' => 'form-
inline']); ?>
    <?= Html::input('text', 'string', Yii::$app->request->post('string'), ['class' => 'form-
control']) ?>
    <?= Html::submitButton('Hash String', ['class' => 'btn btn-lg btn-primary', 'name' =>
'hash-button']) ?>
<?= Html::endForm() ?>
<h3><?= $stringHash ?></h3>
<?php Pjax::end(); ?>
```

Step 2 Server Side Code

```
public function actionFormSubmission()
{
    $security = new Security();
    $string = Yii::$app->request->post('string');
    $stringHash = '';
    if (!is_null($string)) {
        $stringHash = $security->generatePasswordHash($string);
    }
    return $this->render('form-submission', [
        'stringHash' => $stringHash,
    ]);
}
```

how to use pjax

Add this line at the beginning of your view.

```
<?php
use yii\widgets\Pjax;
?>
```

Add the following two lines around the content that needs partial updating.

```
<?php Pjax::begin(['id'=>'id-pjax']); ?>
Content that needs to be updated
<?php Pjax::end(); ?>
```

reload pjax

```
$.pjax.reload({container: '#id-pjax'});
```

use timeout argument in pjax

```
<?php Pjax::begin(['id'=>'id-pjax', 'timeout' => false]); ?>
```

you can specify an integer value for the timeout argument, which would be the number of milliseconds to wait (its default value is 1000). If the execution time in the server is greater than this timeout value, a full page load will be triggered.

By default pjax will submit the form using GET method. You can change the form submission method to POST like in the following example

```
<?php Pjax::begin(['id'=>'id-pjax', 'timeout' => false, 'clientOptions' => ['method' =>
'POST']]); ?>
```

Pjax advanced usage

Yii Framework 2.0 ships with built-in support for Pjax, a JavaScript library that reduces page load times. It accomplishes this by only updating the part of the page that has changed through Ajax, which can translate into substantial savings if you have many other assets on your pages. A few of our projects use this functionality and we wanted to share some lessons learned.

Problem: Page 1 is a simple static page that contains few elements. Page 2 includes an ActiveForm as well as other widgets. The ActiveForm JavaScript resources need to be loaded in order for the inline JavaScript to run, but since Page 1 did not include those assets, Page 2 ran into a JavaScript error when trying to execute the activeform line: ‘Uncaught TypeError: undefined is not a function’.

Solution: Include ActiveForm assets in a shared asset bundle that will be loaded across all pages, ensuring that any entry page will allow the correct scripts to be available.

```
class AppAsset extends AssetBundle
{
    ...
    public $depends = [
        'yii\widgets\ActiveFormAsset',
        'yii\validators\ValidationAsset',
    ];
    ...
}
```

Problem: In the same example above, Page 1 includes a few widgets (NavBar, etc.). Page 2 includes the same widgets plus a few more (ActiveForm, etc.). When loading the page via Pjax, some custom inline JavaScript was running, but the inline script placed by the ActiveForm widget didn’t seem to work, as the validation code was not working. In debug, we found that the ActiveForm init function was running, but the ‘this’ variable didn’t seem to correspond to the ActiveForm. It actually corresponded to the NavBar div. Investigating the div IDs, we saw that the

ActiveForm was expecting to have the ID of #w1, but the NavBar was already assigned that ID on the Page 1 since that was the first widget encountered on that page.

Solution: Do not rely on Yii to auto-generate the widget IDs for you. Instead, always pass in an ID when creating the widget to maintain control of those IDs.

Problem: Pjax request was getting canceled exactly 1,000 ms after the request was initiated.

Solution: Increase the Pjax timeout setting. It defaults to 1 second, which should be acceptable for production sites. However, in development, while using xdebug, our page load times are regularly over this limit.

Problem: Web application implements the Post-Redirect-Get (PRG) pattern. Pjax reloads entire page instead of just the redirection.

Solution: This is intended behavior of Pjax. The redirect doesn't serve its purpose when using Pjax, so you can determine if a request is Pjax, and if so, render the content instead of redirecting. An example may look like:

```
$endURL = "main/endpoint";
if (Yii::$app->request->isPjax) {
    return $this->run($endURL);
} else {
    return $this->redirect([$endURL]);
}
```

What has your experience been with Pjax and Yii? Comment below if you've found any gotchas or have better solutions than ours!

Read Pjax online: https://riptutorial.com/yii2/topic/4554/pjax

Chapter 13: Restful API

Examples

Start with rest api

We have a table that includes of countries so we create a model that called countrylist model

```
<?php

namespace app\models;

use Yii;

/**
 * This is the model class for table "countrylist".
 *
 * @property integer $id
 * @property string $iso
 * @property string $name
 * @property string $nicename
 * @property string $iso3
 * @property integer $numcode
 * @property integer $phonecode
 */
class Countrylist extends \yii\db\ActiveRecord
{
    /**
     * @inheritdoc
     */
    public static function tableName()
    {
        return 'countrylist';
    }

    /**
     * @inheritdoc
     */
    public function rules()
    {
        return [
            [['iso', 'name', 'nicename', 'phonecode'], 'required'],
            [['numcode', 'phonecode'], 'integer'],
            [['iso'], 'string', 'max' => 2],
            [['name', 'nicename'], 'string', 'max' => 80],
            [['iso3'], 'string', 'max' => 3]
        ];
    }

    /**
     * @inheritdoc
     */
    public function attributeLabels()
    {
        return [
            'id' => 'ID',
```

```
            'iso' => 'Iso',
            'name' => 'Name',
            'nicename' => 'Nicename',
            'iso3' => 'Iso3',
            'numcode' => 'Numcode',
            'phonecode' => 'Phonecode',
        ];
    }
}
```

and I create rest webservice for that, we create a controller for restapi and set modelClass variable for our model.

```
 <?php
    namespace app\controllers;
    use yii\rest\ActiveController;
    use Yii;
    class CountrylistController extends ActiveController
    {
      public $modelClass='app\models\Countrylist';
    }
?>
```

for using restapi we need pretty urls and
we add this rule for pretty url

```
'urlManager' => [
   'class' => 'yii\web\UrlManager',
   'enablePrettyUrl' => true,
   'showScriptName' => false,
   'rules' => [
       ['class'=>'yii\rest\UrlRule','controller'=>'countrylist']
       ],
    ],
```

after that we access to can test our rest api as an example

http://localhost/countrylist gives us list of counties.

How to override default actions of rest api Yii2

As an example you want to disable pagination in your default index action and get all results in index. How can you do that? It's simple. You should override the index action in your controller like this:

```
public function actions() {
    $actions = parent::actions();
    unset($actions['index']);
    return $actions;
}

public function actionIndex() {
    $activeData = new ActiveDataProvider([
        'query' => \common\models\Yourmodel::find(),
```

```
        'pagination' => false
    ]);
    return $activeData;
}
```

Override Content-Type for specific action

Use case: just one action which should return a plain (text) content as-is:

```
public function actionAsXML()
{
    $this->layout = false;
    Yii::$app->response->format = Response::FORMAT_XML;

    return ['aaa' => [1, 2, 3, 4]];;
}
```

Pre-defined response formats are:

- FORMAT_HTML
- FORMAT_XML
- FORMAT_JSON
- FORMAT_JSONP
- FORMAT_RAW

There is no mime type for `text/plain` out of the box, use this instead:

```
public function actionPlainText()
{
    $this->layout = false;
    Yii::$app->response->format = Response::FORMAT_RAW;
    Yii::$app->response->headers->add('Content-Type', 'text/plain');

    return $this->render('plain-text'); // outputs template as plain text
}
```

Read Restful API online: https://riptutorial.com/yii2/topic/6102/restful-api

Chapter 14: Routing and URLs

Remarks

All URLs should be created via helper `yii\helpers\Url` it helps you to much if you decide to change url rules in urlManager.

Examples

Creating URLs

Helper yii\helpers\Url provides a set of static methods for managing URLs. This helper may used in views/controllers code.

URL to a route:

```
echo Url::to(['post/index']);
```

URL to a route with parameters:

```
echo Url::to(['post/view', 'id' => 100]);
```

anchored URL:

```
echo Url::to(['post/view', 'id' => 100, '#' => 'content']);
```

absolute URL:

```
echo Url::to(['post/index'], true);
```

absolute URL using the https scheme:

```
echo Url::to(['post/index'], 'https');
```

Note: The route passed to the `Url::to()` method is context sensitive. It may use current module and current controller. For example, assume the current module is `admin` and the current controller is `post`:

relative route with action ID only (contains no slashes at all):

```
echo Url::to(['index']);    // -->> '/index.php?r=admin%2Fpost%2Findex'
```

relative route (has no leading slash):

```
echo Url::to(['post/index']);    // -->> '/index.php?r=admin%2Fpost%2Findex'
```

absolute route (starts with slash):

```
echo Url::to(['/post/index']);    // -->> '/index.php?r=post%2Findex'
```

current requested URL:

```
echo Url::to();
echo Url::to(['']);
```

To create URL based on the **current route** and the **GET parameters** use Url::current().

Assume `$_GET = ['id' => 10, 'page' => 7]`, current route is `post/view`.

current URL:

```
echo Url::current();    // -->> '/index.php?r=post%2Fview&id=10&page=7'
```

current URL without `page` parameter:

```
echo Url::current(['page' => null]);  // -->> '/index.php?r=post%2Fview&id=10'
```

current URL with changed `page` parameter:

```
echo Url::current(['page' => 12]);    // -->> '/index.php?r=post%2Fview&id=10&page=12'
```

Read Routing and URLs online: https://riptutorial.com/yii2/topic/5510/routing-and-urls

Chapter 15: Session

Examples

Session in yii2

import Session Class

```
use  yii\web\Session;
```

Create a session

```
$session = Yii::$app->session;
$session->open(); // open a session
$session->close();  // close a session
```

Store the value in session variable.

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

$session->set('name', 'stack');
OR
$session['name'] = 'stack';
OR
$_SESSION['name'] = 'stack';
```

Get the value from the session variable.

```
$name = $session->get('name');
OR
$name = $session['name'];
```

Remove the session variable

```
$session->remove('name');
OR
unset($session['name']);
OR
unset($_SESSION['name']);

$session->destroy(); // destroy all session
```

Remove all session variables

```
$session->removeAll();
```

Check Session variable

```
$session->has('name')
OR
isset($session['name'])
//both function return boolean value [true or false]
```

Session Flash

Set session flash

```
$session = Yii::$app->session;
$session->setFlash('error', 'Error in login');
```

Get session flash

```
echo $session->getFlash('error');
```

Check session flash

```
$result = $session->hasFlash('error');
```

Remove session flash

```
$session->removeFlash('error');
```

Remove all session flash variables

```
$session->removeAllFlashes();
```

Directly use session variable

Set and get session variable

```
\Yii::$app->session->set('name','stack');
\Yii::$app->session->get('name');
```

Session flash

```
\Yii::$app->getSession()->setFlash('flash_msg','Message');
\Yii::$app->getSession()->getFlash('flash_msg');
```

Creating and editing session variables that are arrays

Save the session variable as a variable.

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

$sess = $session['keys'];
```

Then create or update the array value you want

```
$sess['first'] = 'abc';
```

And finally save to the session variable

```
$session['keys'] = $sess
```

Remember URL to revisit later

Use case: remember the current URL to return to after adding a new record in a different (related) controller, for instance create a new contact to add to an invoice being edited.

InvoiceController / actionUpdate:

```
Url::remember(Url::current(), 'returnInvoice');
```

ContactController / actionCreate:

```
if ($model->save()) {
    $return = Url::previous('returnInvoice');
    if ($return) {
        return $this->redirect($return);
    }
    // ...
}
```

You can reset the remembered URL once you're done:

InvoiceController / actionUpdate:

```
if ($model->save()) {
    Url::remember(null, 'returnInvoice');
    // ...
}
```

The key name - `returnInvoice` in this example - is optional.

Read Session online: https://riptutorial.com/yii2/topic/3584/session

Chapter 16: Testing

Examples

Set up testing environment

Install Codeception:

```
composer global status
composer global require "codeception/codeception=~2.0.0" "codeception/specify=*"
"codeception/verify=*"
```

Install Faker:

```
cd /var/www/yii                                    // Path to your application
composer require --dev yiisoft/yii2-faker:*
```

Create a database, which will be used only for the tests. You can duplicate the existing database or apply migrations:

```
cd tests
codeception/bin/yii migrate
```

Adjust the `components['db']` configuration in `tests/codeception/config/config-local.php`.

Add the directory `/var/www/yii/vendor/bin` to your path.

Review all configuration and `.yml` files.

Start the webserver, e.g.:

```
php -S localhost:8080
```

Run the tests:

```
codecept run
```

More information:

- http://www.yiiframework.com/doc-2.0/guide-test-environment-setup.html
- http://codeception.com/install
- https://github.com/yiisoft/yii2-app-basic/tree/master/tests
- https://github.com/yiisoft/yii2-app-advanced/tree/master/tests

Note: These instructions are valid for the Yii2 version 2.0.9. In the version 2.0.10 will be according to Sam Dark the testing part re-factored (and the instructions have to be updated). The version 2.0.10 should be released on 11. September 2016: https://github.com/yiisoft/yii2/milestones

How to mock ActiveRecord

If you want to mock AR that doesn't try to connect to database you can do it in the following way (if using PHPUnit):

```
$post = $this->getMockBuilder('\app\model\Post')
    ->setMethods(['save', 'attributes'])
    ->getMock();
$post->method('save')->willReturn(true);
$post->method('attributes')->willReturn([
    'id',
    'status',
    'title',
    'description',
    'text'
]);
```

The catch is that we need to override attributes() method since ActiveRecord by default is getting attributes list from database schema which we're trying to avoid.

Read Testing online: https://riptutorial.com/yii2/topic/2226/testing

Chapter 17: Validation

Examples

Validate unique value from database in Yii2

Few people have issue regarding error message not displaying if existing value is being entered in textbox.

So, For Example I'm *not allowing* user to enter *existing email*.

signup.php

(Page where you wanted to sign up new user without existing email id)

1. Remove `use yii\bootstrap\ActiveForm;` (if present)
2. Add `use yii\widgets\ActiveForm;`
3. Add `'enableAjaxValidation' => true` (In that field Where you want to stop user for entering existing email id.)

```
<?php
use yii\bootstrap\ActiveForm;
use yii\widgets\ActiveForm;
?>

<?= $form->field($modelUser, 'email',['enableAjaxValidation' => true])
    ->textInput(['class'=>'form-control','placeholder'=>'Email']); ?>
```

Controller

Add these lines on top `use yii\web\Response;use yii\widgets\ActiveForm;`

```
<?php
use yii\web\Response;
use yii\widgets\ActiveForm;

  .
  .// Your code
  .

  public function actionSignup() {

    $modelUser = new User();

    //Add This For Ajax Email Exist Validation
    if(Yii::$app->request->isAjax && $modelUser->load(Yii::$app->request->post())){
      Yii::$app->response->format = Response::FORMAT_JSON;
      return ActiveForm::validate($modelUser);
    }
    else if ($model->load(Yii::$app->request->post())) {

    }
```

```
  }
?>
```

Model

```
[['email'],'unique','message'=>'Email already exist. Please try another one.'],
```

Validating Unique Value From Database : Unique Validation

Some people have issues regarding error messages not being displayed if an existing value is being entered. For example I'm not allowing a user signup with an existing email.

View

```
<?php
.....................

    <?= $form->field($modelUser, 'email')->textInput(['class'=>'form-
control','placeholder'=>'Email']) ?>
.....................
```

Controller

```
<?php
use yii\web\Response; // important lines
use yii\widgets\ActiveForm; // important lines

.
.// Your code
.

public function actionSignup()
{

    $modelUser = new User();

    //Add This For Ajax Validation
    if(Yii::$app->request->isAjax && $modelUser->load(Yii::$app->request->post())){
        Yii::$app->response->format = Response::FORMAT_JSON;
        return ActiveForm::validate($modelUser);
    }
    if ($modelUser->load(Yii::$app->request->post()) && $modelUser->save()) {
        return $this->redirect(['someplace nice']);
    }
    return $this->render('update', [
        'modelUser' => $modelUser,
    ]);
}
```

Model

```
public function rules()
{
```

```
    return [
        ..............
        ['email', 'unique', 'message'=>'Email already exist. Please try another one.'],
        ...........
    ]
}
```

Disable Validation Error Message On Focus / Key Up

By default error message appears below `textbox` in `<div class="help-block"></div>` on *keyUp* or *after pressing submit button* if any validation constraints aren't met.

Sometimes we want a message on submit only i.e. no validation at `onKeyup` event.

Let's check `yii2/widgets/ActiveForm.php` file:

```
<?php

namespace yii\widgets;

use Yii;
use yii\base\InvalidCallException;
use yii\base\Widget;
use yii\base\Model;
use yii\helpers\ArrayHelper;
use yii\helpers\Url;
use yii\helpers\Html;
use yii\helpers\Json;

class ActiveForm extends Widget
{
  public $action = '';
  public $method = 'post';
  public $options = [];
  .
  .
  .
  public $validateOnSubmit = true;
  public $validateOnChange = true;
  public $validateOnBlur = true;
  public $validateOnType = false;

  .
  .
  .

}
```

There we see that `$validateOnBlur` is set to `true` by default. Changing framework files is a very bad thing to do so we need to override it when using the form:

```
<?php $form = ActiveForm::begin([ 'id' => 'register-form','validateOnBlur' => false]); ?>
```

Scenario in Validation

Using scenario you can perform validation on different situation

Define scenario in model class

```
    class User extends \yii\db\ActiveRecord
    {
      public static function tableName()
      {
          return 'user_master';
      }

    // define validation in rule() function
  public function rules()
  {
    return [
      [['email_id'],'email'],
      [['first_name',],'required','on'=>['create','update']],  // create scenario
      [['email_id',],'required','on'=> ['admin','create','update','forgotpassword']],
      [['mobile',],'required','on'=>['admin','create','update']],
    ];
  }
}
```

Use Scenario in Controller

```
public function actionCreate()
{
  $model = new User();
  $model->scenario="create";  // use create scenario, create scenario validaion applied in
this model


}
public function actionUpdate()
{
    $model = new User();
    $model->scenario="update";  // use update scenario, update scenario validaion applied in
this model
}
```

Validate array

Since Yii2 version 2.0.4 there is the EachValidator used to validate each item in an array.

```
[
    // ... other rules
    ['userIDs', 'each', 'rule' => ['integer']],
]
```

The `['integer']` part can be every other validator object that Yii2 offers and can hold the specific arguments for the validator. Like: `['integer', 'min' => 1337]`. If the userIDs doesn't contain an array the rule validation will fail.

If you just want to see if an attribute contains an array without validating the contents you can write your own validator.

```
[
    ['myAttr', function($attribute, $params) {
        if (!is_array($this->$attribute)) {
            $this->addError($attribute, "$attribute isn't an array!");
        }
    }]
]
```

Read Validation online: https://riptutorial.com/yii2/topic/839/validation

Chapter 18: Working with Databases

Examples

Using Yii2 query builder

Yii2 provides efficient ways to retrieve data from the database.Consider an example of a simple employee table having fields ***emp_id, emp_name and emp_salary***. In order to retrieve the employee names and their salaries, we use the query.

```
select emp_name,emp_salary from employee
```

To generate the above query in Yii2, there are a lot of methods.One of the method is to use a ***yii\db\Query*** object.

```
//creates a new \yii\db\Query() object
$query=new \yii\db\Query();

$rows=$query->select(['emp_name','emp_salary']) //specify required columns in an array
             ->from('employee') //specify table name
             ->all(); //returns an array of rows with each row being an associative array of
name-value pairs.
```

We can make use of a foreach loop to loop through each name-value pair in the ***$rows*** array.

```
foreach ($rows as $row) {
    echo "Employee Name: ".$row['emp_name'].",Employee Salary: ".$row['emp_salary']."<br>";
}
```

This will output

Employee Name: Kiran,Employee Salary: 25000

Employee Name: Midhun,Employee Salary: 50000

Employee Name: Jishnu,Employee Salary: 20000

Employee Name: Ajith,Employee Salary: 25000

Employee Name: Akshay,Employee Salary: 750000

More Examples

Suppose we need to find the name of employees whose salary is equal to 25000.We can write the query in sql as

```
select emp_name from employee where salary=25000
```

In Yii2, the code for generating the above query

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

$rows=$query->select(['emp_name'])
            ->from('employee')
            ->where(['emp_salary'=>25000]) //specify the condition as an associative array
where key is column name
            ->all();
```

If we need to find employee names whose salary is greater than 25000,We can write the code in Yii2 as

```
  $rows=$query->select(['emp_name'])
        ->from('employee')
        ->where(['>','emp_salary', 25000])
//Here first element in the above array specify relational operator used, second element
specify the table name and third the value itself.
        ->all();
```

More condition checking using where()

Multiple conditions can be written using **where()** method as given below.

```
// Creates a new \yii\db\Query() object
$query = new \yii\db\Query();
$rows = $query->select(['emp_name','emp_salary'])
        ->from('employee')
        ->where(['emp_name' => 'Kiran', 'emp_salary' => 25000]) // Specify multiple conditions
        ->one(); // Returns the first row of the result
```

The above code will fetch an employee having the name **kiran** and salary **25000**. If multiple employees are satisfying the above condition, the call ***one()*** makes sure that only the first result is fetched. To fetch all results you should use ***all()***.

Note that if you use ***all()*** the result will always be an array; Even if there is only one or zero results. This array contains all results as arrays or is empty when no records match. The call ***one()*** will return the resulting array directly or false if the query doesn't return anything.

The equivalent code in sql is given below.

```
select emp_name, emp_salary from employee where emp_name = 'Kiran' and emp_salary = 25000
limit 1;
```

An alternative way of writing the above query in Yii2 is given below.

```
$rows = $query->select(['emp_name', 'emp_salary'])
    ->from('employee')
    ->where(['emp_name' => 'Kiran'])
    ->andWhere(['emp_salary' => 25000])
    ->one();
```

Additional set of conditions can be specified using **andWhere**. This will be useful if we need to add additional condition checking to the query later.

Yet another way to specify multiple conditions is by making use of **operator format of where()** method.The above query can also be written as given below.

```
 $rows = $query->select(['emp_name','emp_salary'])
    ->from('employee')
    ->where(['and', 'emp_name="kiran"', 'emp_salary=25000'])
    ->one();
```

Here we specify the operator '**and**' as the first element in the array. Similarly we can also use '**or**', '**between**', '**not between**', '**in**', '**not in**', '**like**', '**or like**', '**not like**', '**or not like**', '**exists**', '**not exists**', '**>**', '**<=**' etc as operators.

Examples of using 'in' and 'like'

Suppose we need to find the employees having salaries **20000, 25000 and 50000**. In normal sql we would write the query as

```
select * from employee where salary in (20000,25000,50000)
```

In Yii2 we can write this as given below.

```
$rows = $query->from('employee')
        ->where(['emp_salary' => [20000,25000,50000]])
        ->all();
```

Another way of specifying the same condition is

```
$rows = $query->from('employee')
    ->where(['in', 'emp_salary', [20000,25000,50000]]) // Making use of operator format of
where() method
    ->all();
```

Similarly '**not in**' can be specified instead of '**in**' if we want to get all employees not having salaries 20000, 25000 and 50000.

Now let us see some examples of using '**like**' inside where() condition. Suppose we need to find all employees having the string '**gopal**' in their name. The names can be venugopal, rajagopal, gopalakrishnan etc. The sql query is given below.

```
select * from employee where emp_name like '%gopal%'
```

In Yii2 we will write this as

```
 $rows = $query->from('employee')
       ->where(['like', 'emp_name', 'gopal']) // Making use of operator format of where()
method
        ->all();
```

If we need to find all employees having the string '**gopal**' and '**nair**' in their name. We can write as

```
   $rows = $query->from('employee')
        ->where(['like', 'emp_name', ['gopal','nair']]) // Making use of operator format of
where() method
        ->all();
```

This would evaluate as

select * from employee where emp_name like '%gopal%' and '%nair%'

Similarly we can use '**not like**' to indicate all employees not having the string '**gopal**' and '**nair**' in their names.

Using orderBy()

The orderBy() method specifies the ORDER BY fragment of a SQL query.For example consider our employee table having fields **emp_id, emp_first_name, emp_last_name and emp_salary** .Suppose we need to order the result by increasing order of employee salaries.We can do it in sql as given below.

```
Select * from employee order by emp_salary
```

In yii2, we can build the query as given below

```
 //creates a new \yii\db\Query() object
    $query=new \yii\db\Query();

    $rows= $query->from('employee')->orderBy([
    'emp_salary' => SORT_ASC //specify sort order ASC for ascending DESC for descending
    ])->all();
```

If we need to order the employees with their first name in ascending order and then their salaries in descending order, we can write it in plain sql as follows.

```
Select * from employee order by emp_first_name ASC, emp_salary DESC
```

The equivalent sql can be build using yii2 as follows

```
//creates a new \yii\db\Query() object
    $query=new \yii\db\Query();

    $rows= $query->from('employee')->orderBy([
    'emp_first_name' => SORT_ASC
    'emp_salary' => SORT_DESC
    ])->all();
```

You can also specify ORDER BY using a string, just like you do when writing raw SQL statements. For example ,the above query can also be generated as given below.

```
//creates a new \yii\db\Query() object
$query=new \yii\db\Query();
$rows=$query->from('employee')->orderBy('emp_first_name ASC, emp_salary DESC')->all();
```

You can call addOrderBy() to add additional columns to the ORDER BY fragment. For example

```
//creates a new \yii\db\Query() object
$query=new \yii\db\Query();
$rows=$query->from('employee')->orderBy('emp_first_name ASC')
    ->addOrderBy('emp_salary DESC')->all();
```

Read Working with Databases online: https://riptutorial.com/yii2/topic/4167/working-with-databases

Chapter 19: Yii2 ActiveForm

Examples

Form Fields In Yii2

Showing Basic Example of the View Page in Yii2 For New Learners

These are basic classes you must to add to create form using yii2 ActiveForm

```
<?php

Use yii\helpers\Html;
Use yii\widgets\ActiveForm;
```

The Below line will start the form tag for our form below showing example shows that how to specify id for the form and how to apply any classes for the form..

```
$form =ActiveForm::begin([   'id'=> 'login-form',   'options'=> ['class' => 'form-
horizontal'],]) ?>
```

Here $model Specify which database table field we want to bind with form that model object stored here in this variable which has been passed from the relevant controller.

```
   <?= $form->field($model, 'username') ?>
   <?= $form->field($model, 'password')->passwordInput() ?>
```

'username' and 'password' is the name of the table field with which our value will be bound.

Here in below code we are putting submit button for form submission and applying 'Login' as Button Text and basic css classes to it.

```
<div class="form-group">
      <div class="col-lg-offset-1 col-lg-11">
          <?= Html::submitButton('Login', ['class' => 'btn btn-primary']) ?>
      </div>
  </div>
```

Here in below code we are Closing out form

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

Create Password Field :

```
<?= $form->field($model, 'password')->passwordInput() ?>
```

Create TextField :

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

Create Hidden Form Field :

```
echo $form->field($model, 'hidden1')->hiddenInput()->label(false);
```

Create Dropdown :

```
<?php echo $form->field($model, 'name')
->dropdownList(
Stud::find()->select(['name'])
->indexBy('name')->column(),
['prompt'=>'Select no']); ?>
```

Dropdown list with Id And Name

```
<?= $form->field($model, 'name')->dropDownList(
            ArrayHelper::map(Stud::find()->all(), 'no', 'name'),['prompt' => 'Select Car
Name']
 ) ?>
```

Create FileUploader :

```
echo $form->field($model, 'imagepath')->fileInput();
```

Adding A Placeholder and Customized Label

```
 <?= $form->field($model, 'username')->textInput()->hint('Please enter your  name')-
>label('Name') ?>
```

ActiveForm Validations

You can enable/disable ajax and client validations in active form.

```
$form = ActiveForm::begin([
    'id' => 'signup-form',
    'enableClientValidation' => true,
    'enableAjaxValidation' => true,
    'validationUrl' => Url::to('signup'),
]);
```

1. `enableClientValidation` is by default enabled in ActiveForm. If you don't need client validation in form we can disable by assigning as false.
2. `enableAjaxValidation` is by default disabled in ActiveForm. If you want to enable it we have to add manually in ActiveForm like above.
3. `validationUrl` - if you want to keep all the validation coding in separate controller action for this form we can configure the activeform using `validationUrl`. If we didn't set this, it will take the form's action value.

The above two arguments will affect for whole form. If you want to check ajax validation only for particular field in the form you can add `enableAjaxValidation` for that particular field. It will work only for that field not whole form.

For example in registration form you want to check the username already exist validation on time of user enter in the form. you can use this `enableAjaxValidation` argument for that field.

```
echo $form->field($model, 'username', ['enableAjaxValidation' => true]);
```

Read Yii2 ActiveForm online: https://riptutorial.com/yii2/topic/6807/yii2-activeform

Chapter 20: Yii2 Jquery Calendar For Text Field

Examples

Add jquery calendar for a text field with max as current date

If we want to display a jquery calendar for end user who can pick maximum date as current date in the calendar. The below code will useful to this scenario.

```
<?php
use yii\jui\DatePicker;
use yii\widgets\ActiveForm;
?>

<?php $form = ActiveForm::begin(['id' => 'profile-form']); ?>
.....
<?= $form->field($model, 'date_of_birth')->widget(DatePicker::classname(), ['dateFormat' =>
'php:M d, Y', 'options' => ['readonly' => true], 'clientOptions' => [ 'changeMonth' => true,
'changeYear' => true, 'yearRange' => '1980:'.date('Y'), 'maxDate' => '+0d']]) ?>
.....
<?php ActiveForm::end(); ?>
```

Add jquery calendar for a text field with min date

For some forms you want display the days from future/past days and other days need to disabled, then this scenario will help.

```
<?php
use yii\jui\DatePicker;
use yii\widgets\ActiveForm;
?>

<?php $form = ActiveForm::begin(['id' => 'profile-form']); ?>
.....

<?php
$day = '+5d'; //if you want to display +5 days from current date means for future days.
#(or)
$day = '-5d'; //if you want to display -5 days from current date means older days.
?>
<?= $form->field($model, 'date_of_birth')->widget(DatePicker::classname(), ['dateFormat' =>
'php:M d, Y', 'options' => ['readonly' => true], 'clientOptions' => [ 'changeMonth' => true,
'changeYear' => true, 'yearRange' => '1980:'.date('Y'), 'maxDate' => $day]]) ?>
.....
<?php ActiveForm::end(); ?>
```

Add jquery calendar with from date and to date

If you want to have calendar for from date and to date and also to date calendar days always will

be greater than from date field, then below scenario will help.

```
<?php $form = ActiveForm::begin(['id' => 'profile-form']); ?>
.....
<?= $form->field($model, 'from_date')->widget(DatePicker::classname(), ['dateFormat' => 'php:M
d, Y', 'options' => ['readonly' => true], 'clientOptions' => [ 'changeMonth' => true,
'changeYear' => true, 'yearRange' => '1980:'.date('Y'), 'onSelect' => new
yii\web\JsExpression('function(selected) { var dt = new Date(selected);
dt.setDate(dt.getDate() + 1); $("#filter-date-to").datepicker("option", "minDate", dt); }')]])
?>

<?= $form->field($model, 'to_date')->widget(DatePicker::classname(), ['dateFormat' => 'php:M
d, Y', 'options' => ['readonly' => true, 'id' => 'filter-date-to'], 'clientOptions' => [
'changeMonth' => true, 'changeYear' => true, 'yearRange' => '1980:'.date('Y')]]) ?>
.....
<?php ActiveForm::end(); ?>
```

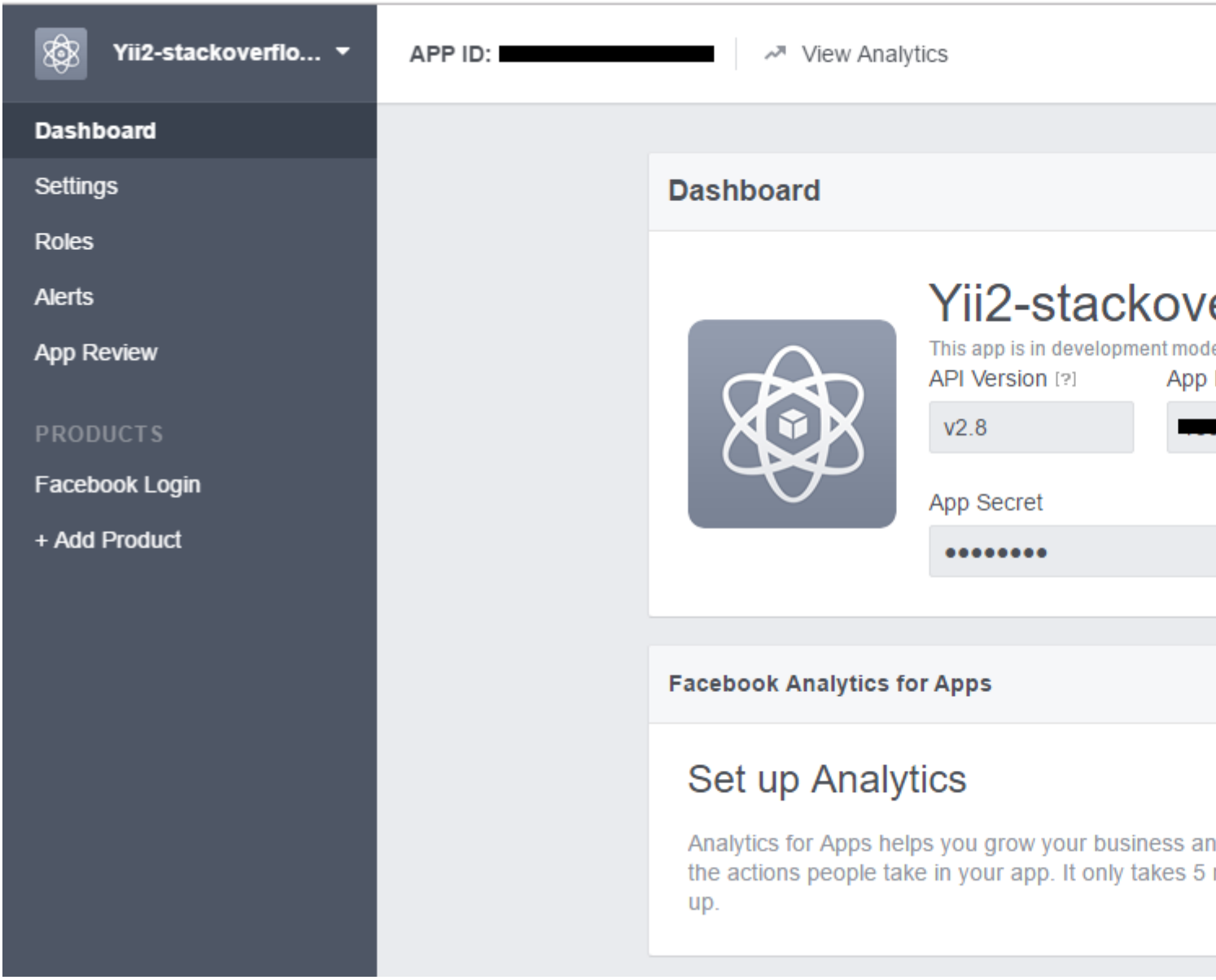
Read Yii2 Jquery Calendar For Text Field online: https://riptutorial.com/yii2/topic/6366/yii2-jquery-calendar-for-text-field

Chapter 21: Yii2 OAuth2 - Ex: consumer facebook OAuth2

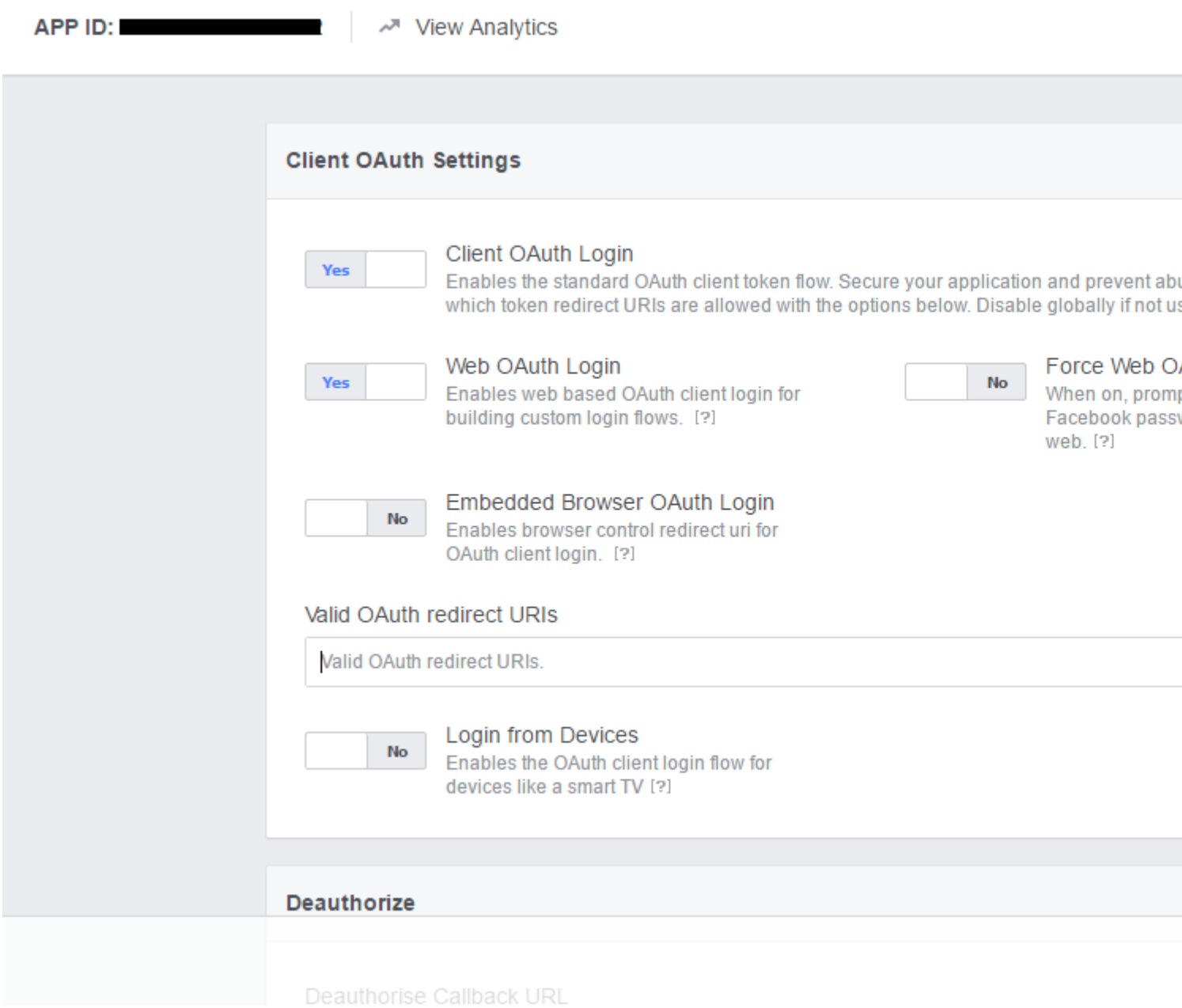
Examples

Create an app on facebook developer

Go to the https://developers.facebook.com/ and create your app.



Click `Add product` and choose `Facebook Login`



Install yii2-authclient

Before install this extension, you must install yii2-app. In this example, I use yii2-basic template. Guide for installation in here.

Run

```
composer require --prefer-dist yiisoft/yii2-authclient
```

or add

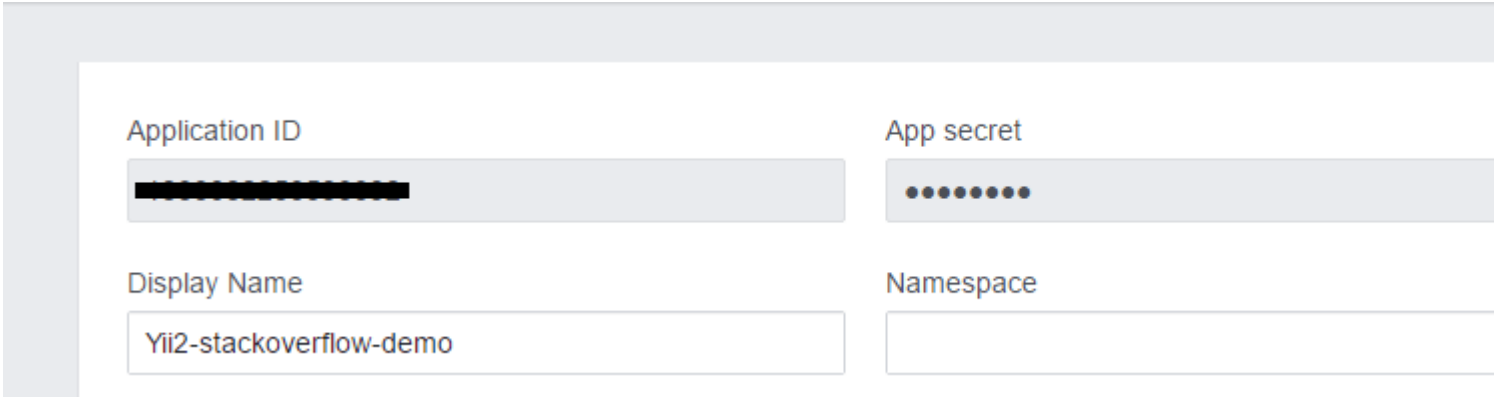
```
"yiisoft/yii2-authclient": "~2.1.0"
```

to the `require` section of your `composer.json`.

Add config `authClientCollection` to your config `components`:

```
return [
    'components' => [
        'authClientCollection' => [
            'class' => 'yii\authclient\Collection',
            'clients' => [
                'facebook' => [
                    'class' => 'yii\authclient\clients\Facebook',
                    'clientId' => 'facebook_client_id',
                    'clientSecret' => 'facebook_client_secret',
                ],
            ],
        ]
    ],
    // ...
];
```

`facebook_client_id` is application id and `facebook_client_secret` is app secret.



Add auth action and set up callback

1. Add button `Login as facebook account` to your login view:

Edit `site/login.php` in views folder, add theses line to content of page login:

```
<?= yii\authclient\widgets\AuthChoice::widget([
     'baseAuthUrl' => ['site/auth'],
     'popupMode' => false,
]) ?>
```

Above, we set that `auth` action in `SiteController` will handler OAuth2 flow.

Now we create it.

```
class SiteController extends Controller
{
    public function actions()
    {
        return [
            'auth' => [
                'class' => 'yii\authclient\AuthAction',
                'successCallback' => [$this, 'onAuthSuccess'],
            ],
```

```
        ];
    }

    public function onAuthSuccess($client)
    {
        // do many stuff here, save user info to your app database
    }
}
```

We use `yii\authclient\AuthAction` for create url and redirect to facebook login page.

Function `onAuthSuccess` used to get user info, login to your app.

Add redirect_url to facebook app setting

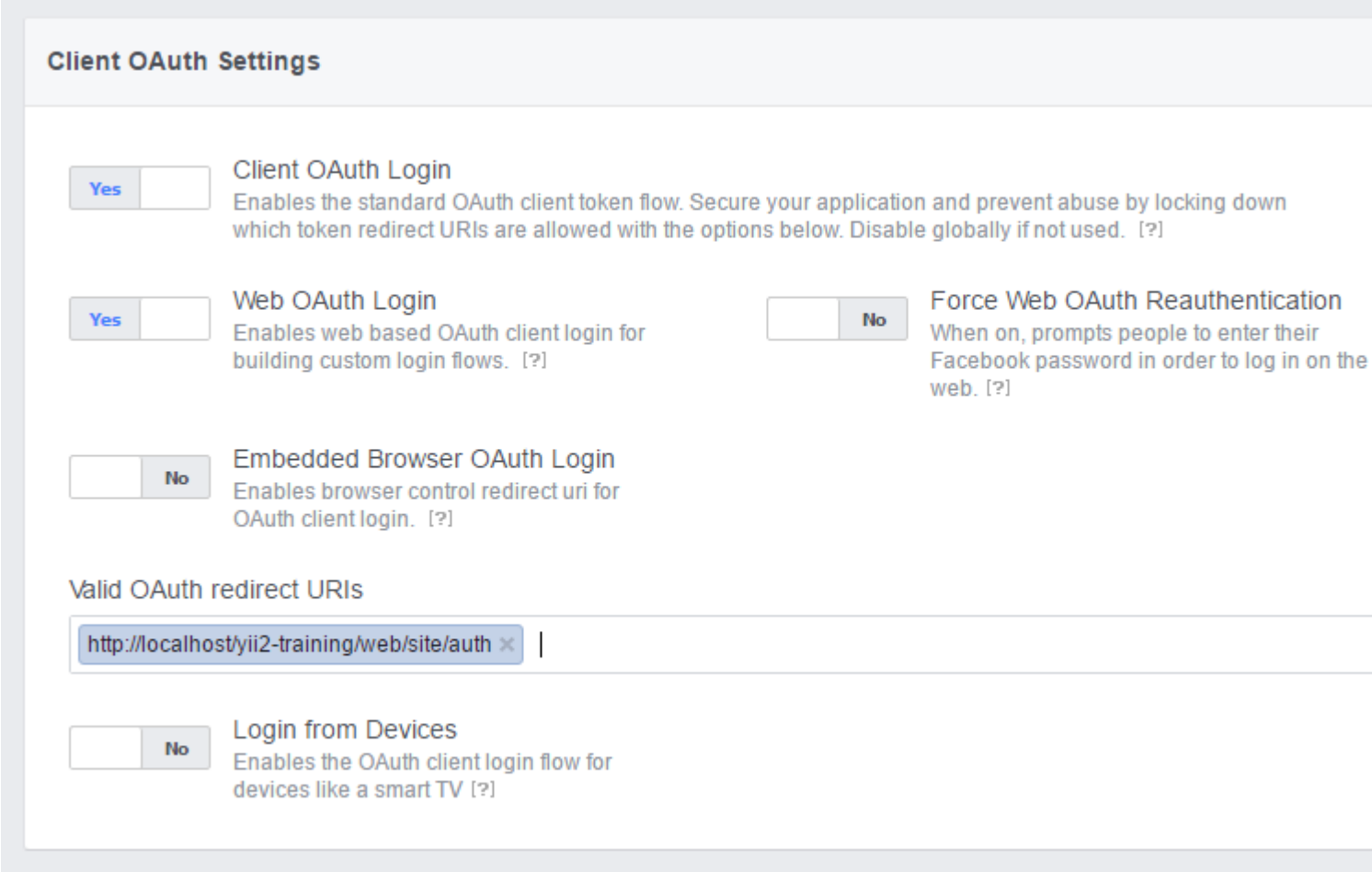
If you enable prettyUrl in your yii2-app, your redirect_uri will be:

```
http://<base_url>/web/site/auth
```

And disable pretty url:

```
http://<base_url>/web/index.php?r=site%2Fauth
```

Example:



Example for onAuthSuccess function

```
/**
 * @param $client ClientInterface
 */
public function onAuthSuccess($client)
{
    //Get user info
    /** @var array $attributes */
    $attributes = $client->getUserAttributes();
    $email = ArrayHelper::getValue($attributes, 'email'); //email info
    $id = ArrayHelper::getValue($attributes, 'id'); // id facebook user
    $name = ArrayHelper::getValue($attributes, 'name'); // name facebook account

    //Login user
    //For demo, I will login with admin/admin default account
    $admin = User::findByUsername('admin');
    Yii::$app->user->login($admin);
}
```

Read Yii2 OAuth2 - Ex: consumer facebook OAuth2 online:
https://riptutorial.com/yii2/topic/7428/yii2-oauth2---ex--consumer-facebook-oauth2

Credits

S. No	Chapters	Contributors
1	Getting started with yii2	Bibek Lekhak, Community, Farcaller, jagsler, Mohan Rex, Muhammad Shahzad, Pasha Rumkin, Sam Dark, urmaul, Yasar Arafath, Yasin Patel, Yatin Mistry
2	Active Record	Insane Skull, Manikandan S, Michael St Clair, Mike Artemiev, particleflux, saada, Sam Dark, Yasar Arafath, Yasin Patel
3	Advanced Project Template	mnoronha, Mohan Rex, Salem Ouerdani, Sam Dark, topher
4	Ajax Request	Anton Rybalko, Ilyas karim, meysam
5	Asset management	Thomas Rohde
6	Components	Sam Dark, Yasin Patel
7	Cookies	IStranger, Sam Dark
8	Custom Validations	Ejaz Karim
9	Database Migrations	Ali MasudianPour, jlapoutre, Sam Dark
10	File Uploads	Sam Dark
11	Installing Extension Manually	Insane Skull, mnoronha, Sam Dark, vishuB
12	Pjax	gmc, Manikandan S, Muaaz Rafi, Shaig Khaligli, yafater, Yasin Patel
13	Restful API	jagsler, jlapoutre, yafater, Yasin Patel
14	Routing and URLs	IStranger, Ярослав Гойса
15	Session	Brett, Goke Obasa, jlapoutre, MikelG, Yasin Patel
16	Testing	Antonín Slejška, Bizley, Sam Dark, XzAeRo
17	Validation	jagsler, Mihai P., Nana Partykar, Sam Dark, Yasin Patel
18	Working with Databases	jagsler, Kiran Muralee
19	Yii2 ActiveForm	Hina Vaja, Manikandan S, particleflux

20	Yii2 Jquery Calendar For Text Field	Manikandan S
21	Yii2 OAuth2 - Ex: consumer facebook OAuth2	ThanhPV