Creating modules

If you have created a complex application part and want to use it with some degree of customization in  
your next project, most probably you need to create a module. In this recipe, we will see how to create an  
application log view module.

Getting ready

Create a new yii2-app-basic application using the composer, as described in the official guide at  
<http://www.yiiframework.com/doc-2.0/guide-start-installation.html>.

How to do it...

Let’s do some planning first.

In yii2-app-basic with default configuration, all log entries are stored in the runtime/logs/app. log file.  
We can extract all messages from this file with help of regular expressions and display them on the  
GridView widget. Besides, we must allow the user to configure the path to the custom log file.

Carry out the following steps:

1. Create the modules/log directory and create the Module class with the new file option:

<?php

namespace app\modules\log;

class Module extends \yii\base\Module  
{

public $file = '@runtime/logs/app.log';

}

1. Create a simple model for transferring rows from the log file:

<?php

namespace app\modules\log\models;

use yii\base\Object;

class LogRow extends Object  
{

public $time;  
public $ip;  
public $userId;  
public $sessionId;  
public $level;  
public $category;  
public $text;

}

1. Write a log file reader class that will parse file rows, reverse its order, and return array of instances  
   of the LogRow models:

<?php

namespace app\modules\log\services;

use app\modules\log\models\LogRow;

class LogReader  
{

public function getRows($file)

{

$result = [];

$handle = @fopen($file, "r");  
if ($handle) {

while (($row = fgets($handle)) !== false) {

$pattern =

'#A' .

'(?P<time>\d{4}\-\d{2}\-\d{2} \d{2}:\d{2}:\d{2}) ' .

'\[(?P<ip>[A\]]+)\]' .

'\[(?P<userId>[A\]]+)\]' .

'\[(?P<sessionId>[A\]]+)\]' .

'\[(?P<level>[A\]]+)\]' .

'\[(?P<category>[A\]]+)\]' .

' (?P<text>.\*?)' .

'(\$\\_(GET|POST|REQUEST|COOKIE|SERVER) = \[)?' .

'$#i';

if (preg\_match($pattern, $row, $matches)) {  
if ($matches['text']) {

$result[] = new LogRow([

'time' => $matches['time'],

'ip' => $matches['ip'],

'userid' => $matches['userId'],

'sessionid' => $matches['sessionId'],

'level' => $matches['level'],

'category' => $matches['category'],

'text' => $matches['text'],

]);

}

}

}

fclose($handle);

}

return array\_reverse($result);

}

}

4. Add a helper for displaying pretty HTML-badges for the log levels:

<?php

namespace app\modules\log\helpers;

use yii\helpers\ArrayHelper;  
use yii\helpers\Html;

class LogHelper  
{

public static function levelLabel($level)

{

$classes = [

' error ' => 'danger',

'warning' => 'warning',

' info ' => 'primary',

'trace' => 'default',

'profile' => 'success',

'profile begin' => 'info',

'profile end' => 'info',

];

$class = ArrayHelper::getValue($classes, $level, 'default');

return Html::tag('span', Html::encode($level), ['class' => 'label-' .  
$class]);

}

}

1. Create a module controller that will get an array of rows from the reader and pass them into  
   ArrayDataProvider:

<?php

namespace app\modules\log\controllers;

use app\modules\log\services\LogReader;  
use yii\data\ArrayDataProvider;  
use yii\web\Controller;

class DefaultController extends Controller  
{

public function actionIndex()

{

$reader = new LogReader();

$dataProvider = new ArrayDataProvider([

'allModels' => $reader->getRows($this->getFile()),

]);

return $this->render('index', [

'dataProvider' => $dataProvider,

]);

}

private function getFile()

{

return \Yii::getAlias($this->module->file);

}

}

1. Now, create the modules/log/default/index . php view file:

<?php

use app\modules\log\helpers\LogHelper;  
use app\modules\log\models\LogRow;  
use yii\grid\GridView;  
use yii\helpers\Html;

/\* @var $this yii\web\View \*/

/\* @var $dataProvider yii\data\ArrayDataProvider \*/

$this->title = 'Application log';

$this->params['breadcrumbs'][] = $this->title;

?>

<div class="log-index">

<h1><?= Html::encode($this->title) ?></h1>

<?= GridView::widget([

'dataProvider' => $dataProvider,

'columns' => [

[

'attribute' => 'time',

'format' => 'datetime',

'contentOptions' => [

'style' => 'white-space: nowrap',

],

],

' ip:text :IP',

'userid:text:User ',

[

'attribute' => 'level',

'value' => function (LogRow $row) {

return LogHelper::levelLabel($row->level);

},

'format' => 'raw',

],

'category',

' text',

],

]) ?>

</div>

1. Attach the module to your application in the config/web. php file:

$config = [

'id' => 'basic',

'basePath' => dirname( DIR ),

'bootstrap' => ['log'],

'modules' => [

'log' => 'app\modules\log\Module',

],

'components' => [

].

];

1. Add a link to the controller in the views/layouts/main. php file:  
   echo Nav::widget([

'options' => ['class' => 'navbar-nav navbar-right'],  
'items' => [

['label' => 'Home', 'url' => ['/site/index']],

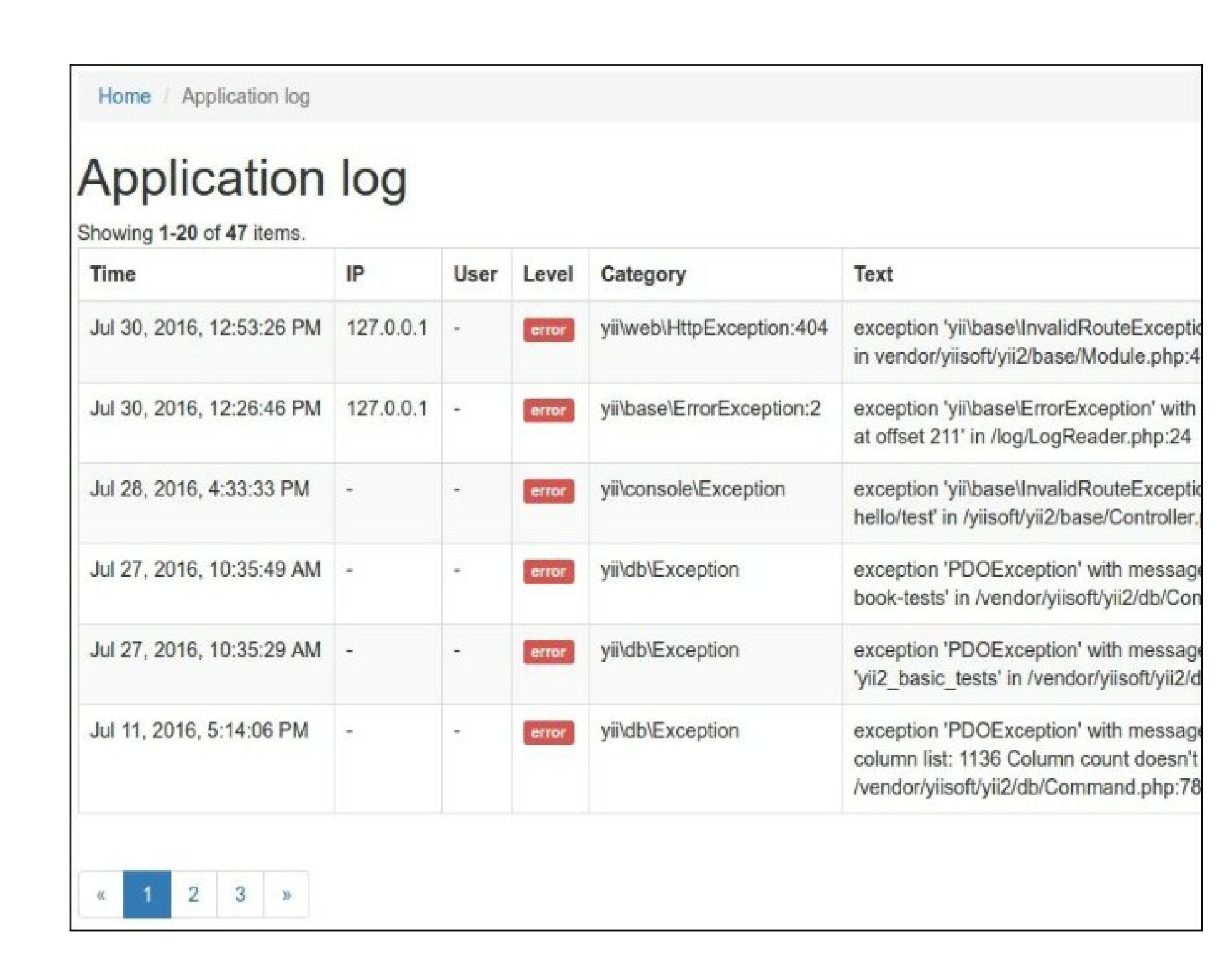
['label' => 'Log', 'url' => ['/log/default/index']],  
['label' => 'About', 'url' => ['/site/about']],  
['label' => 'Contact', 'url' => ['/site/contact']],

],

]);

NavBar::end();

1. Go to url /index. php?r=log and ensure that the module works:



How it works...

You can group your controllers, models, views, and other components by separated modules and attach  
them into your application. You can generate a module template with the help of Gii or make it manually.

Each module contains a main module class where we can define configurable properties, define change  
paths, attach controllers, and so on. By default, a module generated with Gii runs the index action of the  
default controller.

See also

* For more information about modules and about best practices, refer to  
  <http://www.yiiframework.com/doc-2.0/guide-structure-modules.html>
* The Making extensions distribution-ready recipe