Profiling an application with Yii

If all of the best practices for deploying a Yii application are applied and you still do not have the  
performance you want, then most probably there are some bottlenecks with the application itself. The  
main principle while dealing with these bottlenecks is that you should never assume anything and always  
test and profile the code before trying to optimize it.

In this recipe, we will try to find bottlenecks in the Yii2 mini application.

Getting ready

Create a new yii2-app-basic application using the Composer package manager, as described in the  
official guide at [http://www.yiiframework.com/doc-2.0/gurde-start-installation.html](http://www.yiiframework.com/doc-2.0/guide-start-installation.html).

1. Set up your database connection and apply the following migration:

<?php

use yii\db\Migration;

class m160308\_093233\_create\_example\_tables extends Migration  
{

public function up()

{

$tableOptions = null;

if ($this->db->driverName === 'mysql') {

$tableOptions = 'CHARACTER SET utf8 COLLATE utf8\_general\_ci  
ENGINE=InnoDB';

}

$this->createTable('{{%category}} ', [

'id' => $this->primaryKey(),

'name' => $this->string()->notNull(),

], $tableOptions);

$this->createTable('{{%article}}', [

'id' => $this->primaryKey(),

'category\_id' => $this->integer()->notNull(),

'title' => $this->string()->notNull(),

'text' => $this->text()->notNull(),

], $tableOptions);

$this->createIndex('idx-article-category\_id', '{{%article}}',

' category\_id');

$this->addForeignKey('fk-article-category\_id', '{{%article}}',

'category\_id', '{{%category}}', 'id');

}

public function down()

{

$this->dropTable('{{%article}}');

$this->dropTable('{{%category}}');

}

}

1. Generate models for each table in Yii.
2. Write the following console command:

<?php

namespace app\commands;

use app\models\Article;  
use app\models\Category;  
use Faker\Factory;  
use yii\console\Controller;

class DataController extends Controller  
{

public function actionInit()

{

$db = \Yii::$app->db;

$faker = Factory::create();

$transaction = $db->beginTransaction();  
try {

$categories = [];

for ($id = 1; $id <= 100; $id++) {

$categories[] = [

'id' => $id,

'name' => $faker->name,

];

}

$db->createCommand()->batchInsert(Category::tableName(), ['id',

'name'], $categories)->execute();

$articles = [];

for ($id = 1; $id <= 100; $id++) {

$articles[] = [

'id' => $id,

'category\_id' => $faker->numberBetween(1, 100),

'title' => $faker->text($maxNbChars = 100),

'text' => $faker->text($maxNbChars = 200),

];

}

$db->createCommand()

->batchInsert(Article::tableName(), ['id', 'category\_id', 'title',  
'text'], $articles)->execute();

$transaction->commit();

} catch (\Exception $e) {

$transaction->rollBack();  
throw $e;

}

}

}

And execute it:

./yii data/init

4. Add the ArticleController class as follows:

<?php

namespace app\controllers;  
use Yii;

use app\models\Article;

use yii\data\ActiveDataProvider;

use yii\web\Controller;

class ArticleController extends Controller  
{

public function actionIndex()

{

$query = Article::find();

$dataProvider = new ActiveDataProvider([

'query' => $query,

]);

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

'dataProvider' => $dataProvider,

]);

}

}

5. Add the views/article/index . php view as follows:

<?php

use yii\helpers\Html;  
use yii\widgets\ListView;

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

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

$this->title = 'Articles';

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

?>

<div class="article-index">

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

<?= ListView::widget([

'dataProvider' => $dataProvider,

'itemOptions' => ['class' => 'item'],

'itemView' => '\_item',

]) ?>

/div>

Then add views/article/\_item.php:

<?php

use yii\helpers\Html;

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

/\* @var $model app\models\Article \*/

?>

<div class="panel panel-default">

<div class="panel-heading"><?= Html::encode($model->title); ?></div>  
<div class="panel-body">

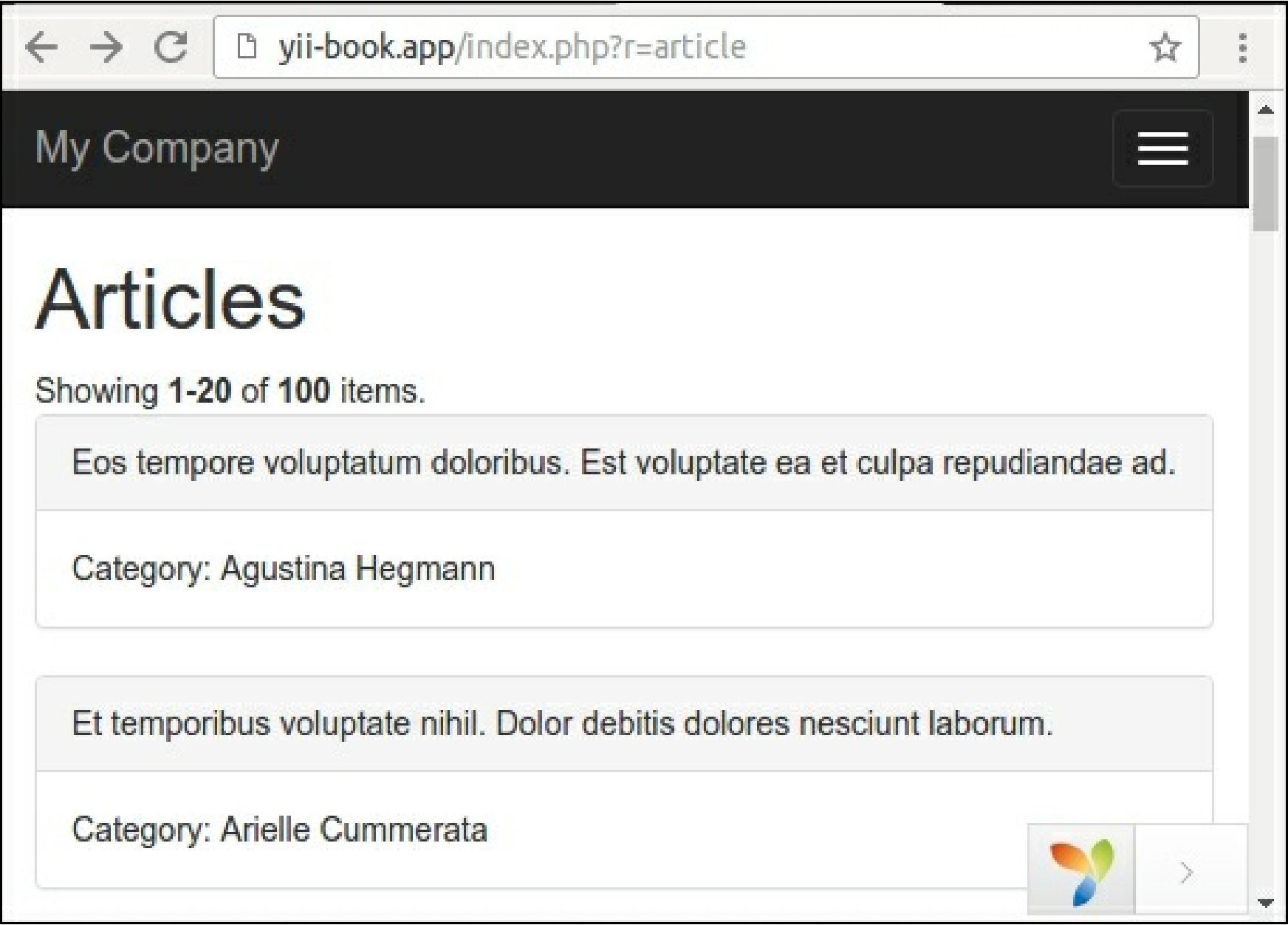
Category: <?= Html::encode($model->category->name) ?>

</div>

</div>

How to do it...

Follow these steps to profile an application with Yii:

1. Open the articles page:

2. Open the views/article/index. php file and add profiler calls before and after the Listview  
widget:

<div class="article-index">

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

<?php Yii::beginProfile('articles') ?>

<?= ListView::widget([

'dataProvider' => $dataProvider,

'itemOptions' => ['class' => 'item'],

'itemView' => '\_item',

]) ?>

<?php Yii::endProfile('articles') ?>

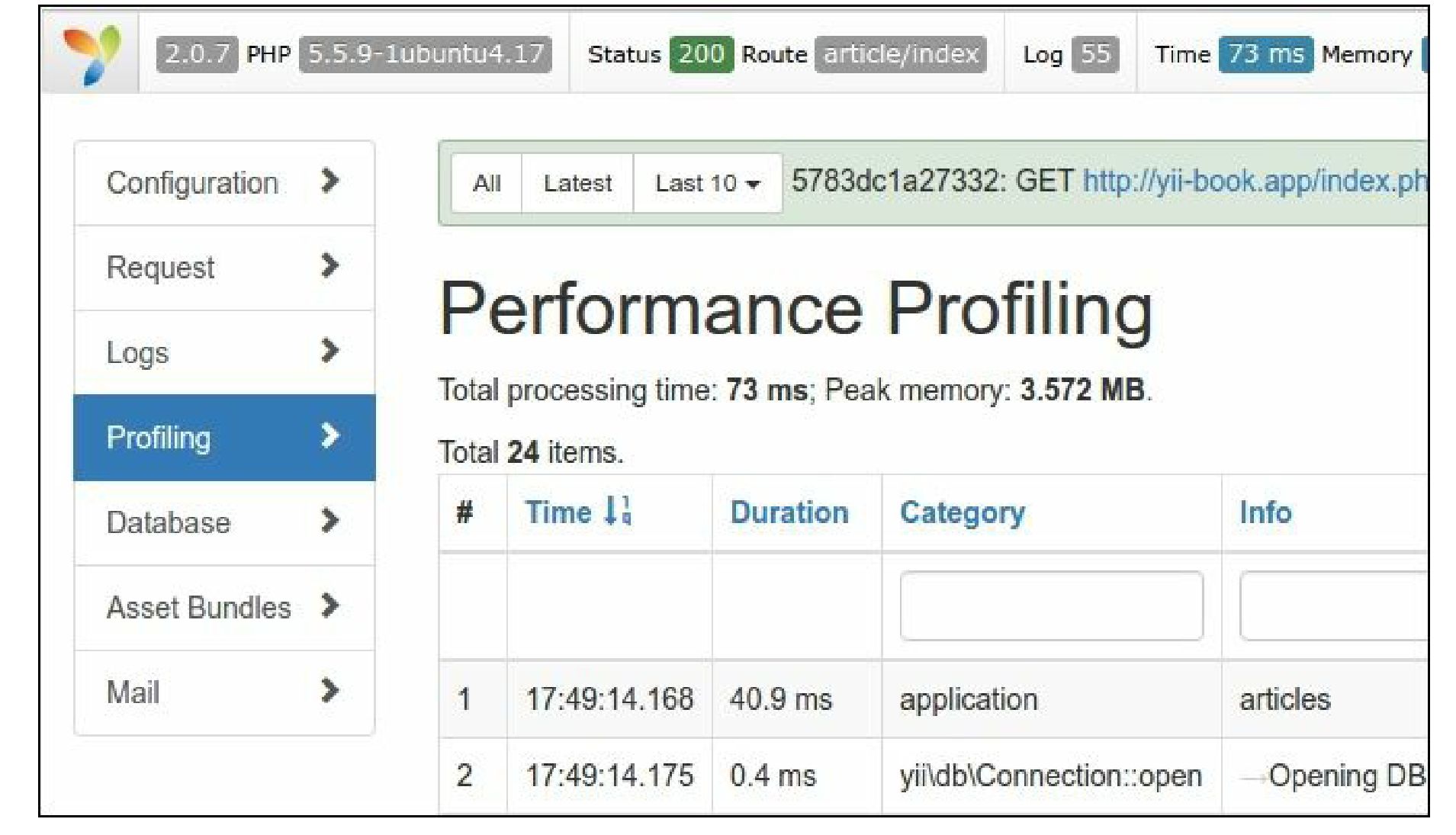
</div>

Now refresh the page.

3. Expand the debug panel at the bottom of page and click on the timing badge (73 ms in our case):

We can see that our articles block has taken close to 40 milliseconds.

Now examine the Profiling report:



1. Open our controller and add eager loading for article’s category relation as follows:

class ArticleController extends Controller  
{

public function actionIndex()

{

$query = Article::find()->with('category');

$dataProvider = new ActiveDataProvider([

'query' => $query,

]);

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

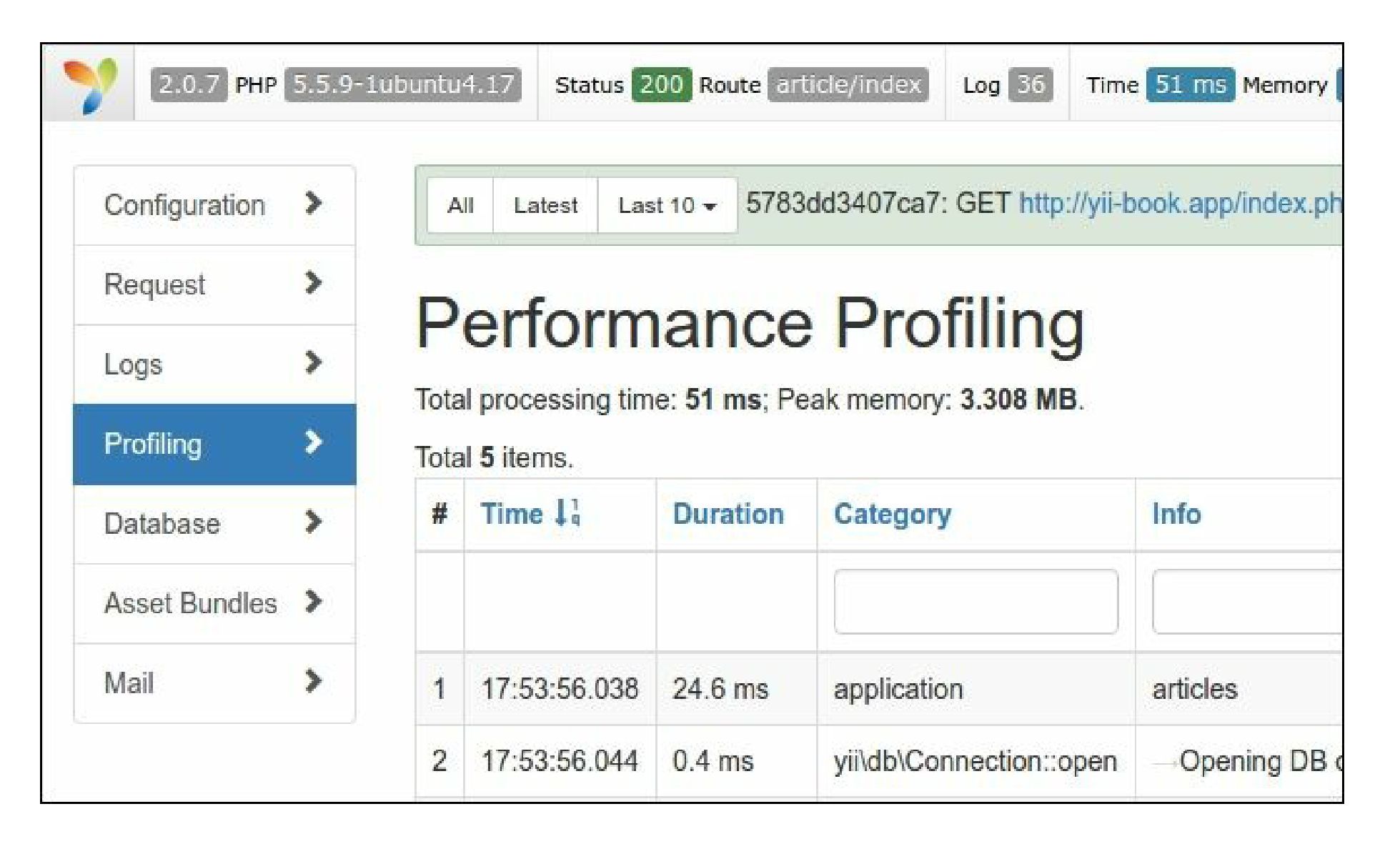
'dataProvider' => $dataProvider,

]);

}

}

1. Go back to the site, refresh the page, and open the Profiling report again:



Right now the articles listing has taken close to 25 milliseconds because the application makes fewer SQL  
queries with eager loading of related models.

How it works...

You can enclose any fragment of source code with Yii: :beginProfile and Yii: :endProfile calls:

Yii::beginProfile('articles');

// ...

Yii::endProfile('articles' );

After executing the page, you can see the report with all timings on the Profiling page of the debug  
module.

Also, you can use nested profiling calls as follows:

Yii::beginProfile('outer');

Yii::beginProfile('inner');

// ...

Yii::endProfile('inner');

Yii::endProfile('outer');

Note

Note: Take care with correct opening and closing calls in this case and correct block naming. If you the  
miss Yii: : endProfile call or switch the order of Yii: : endProfile( ' inner ' ) and  
Yii: : endProfile ('outer'), performance profiling will not work.

See also

* [For more information about logging refer to the following URL: http://www.yiiframework.com/doc-](http://www.yiiframework.com/doc-2.0/guide-runtime-logging.html%23performance-profiling)  
  2.0/gnirie-rnntime-logging.html#performance-profiling
* About tuning of the application performance refer to the following URL:  
  <http://www.yiiframework.com/doc-2.0/guide-tutorial-performance-tuning.html>