Leveraging HTTP caching

Instead of only server-side caching implementation you can use client-side caching via specific HTTP-  
headers.

In this recipe, we will cover full-page caching on the basis of the Last-Modified and ETag headers.

Getting ready

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

1. Create and run migration as follows:

<?php

use yii\db\Migration;

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

public function up()

{

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

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

'created\_at' => $this->integer()->unsigned()-  
>notNull(),

'updated\_at' => $this->integer()->unsigned()->notNull(),

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

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

]);

}

public function down()

{

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

}

}

1. Create an Article model as follows:

<?php

namespace app\models;  
use Yii;

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

class Article extends ActiveRecord  
{

public static function tableName()

{

return '{{%article}}';

}

public function behaviors()

{

return [

TimestampBehavior::className(),

];

}

}

1. Create a blog controller with the following actions:

<?php

namespace app\controllers;

use app\models\Article;  
use yii\web\Controller;  
use yii\web\NotFoundHttpException;  
class BlogController extends Controller  
{

public function actionIndex()

{

$articles = Article::find()->orderBy(['id' => SORT\_DESC])->all();  
return $this->render('index', array(

'articles' => $articles,

));

}

public function actionView($id)

{

$article = $this->findModel($id);  
return $this->render('view', array(

'article' => $article,

));

}

public function actionCreate()

{

$n = rand(0, 1000);

$article = new Article();

$article->title = 'Title #' . $n;

$article->text = 'Text #' . $n;

$article->save();  
echo 'OK';

}

public function actionUpdate($id)

{

$article = $this->findModel($id);

$n = rand(0, 1000);

$article->title = 'Title #' . $n;

$article->text = 'Text #' . $n;

$article->save();  
echo 'OK';

}

private function findModel($id)

{

if (($model = Article::findOne($id)) !== null) {  
return $model;

} else {

throw new NotFoundHttpException('The requested page does not exist.');

}

}

}

1. Add the views/blog/index.php view:

<?php

use yii\helpers\Html;

$this->title = 'Articles';;

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

?>

<?php foreach($articles as $article): ?>

<h3><?= Html::a(Html::encode($article->title), ['view', 'id' => $article->id])  
?></h3>

<div>Created <?= Yii::$app->formatter->asDatetime($article->created\_at) ?>  
</div>

<div>Updated <?= Yii::$app->formatter->asDatetime($article->updated\_at) ?>  
</div>

<?php endforeach ?>

5. Add the views/blog/view. php view file:

<?php

use yii\helpers\Html;

$this->title = $article->title;

$this->params['breadcrumbs'][] = ['label' => 'Articles', 'url' => ['index']];  
$this->params['breadcrumbs'][] = $this->title;

?>

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

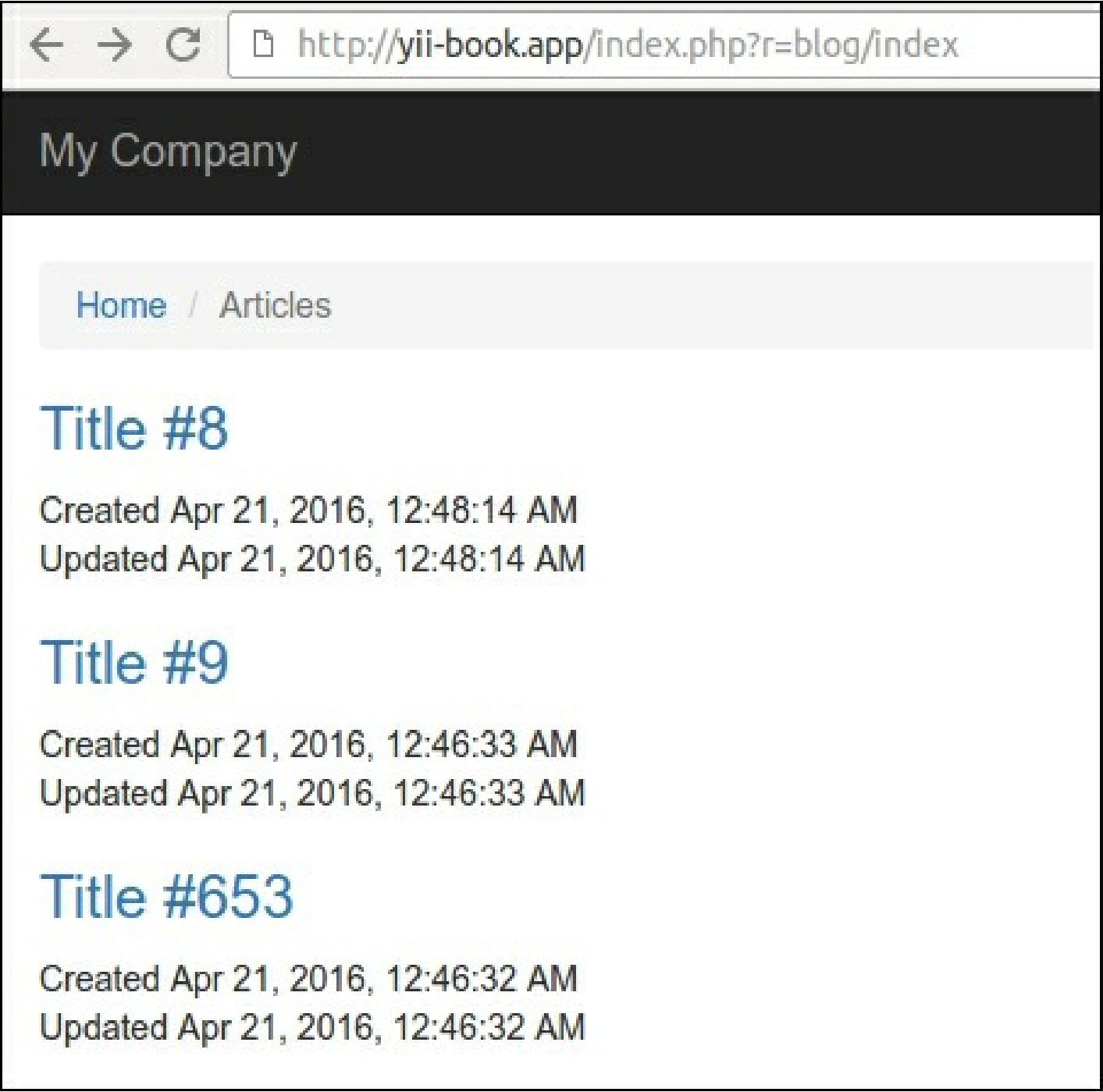
<div>Created <?= Yii::$app->formatter->asDatetime($article->created\_at) ?></div>  
<div>Updated <?= Yii::$app->formatter->asDatetime($article->updated\_at) ?></div>  
<hr />

<p><?= Yii::$app->formatter->asNtext($article->text) ?></p>

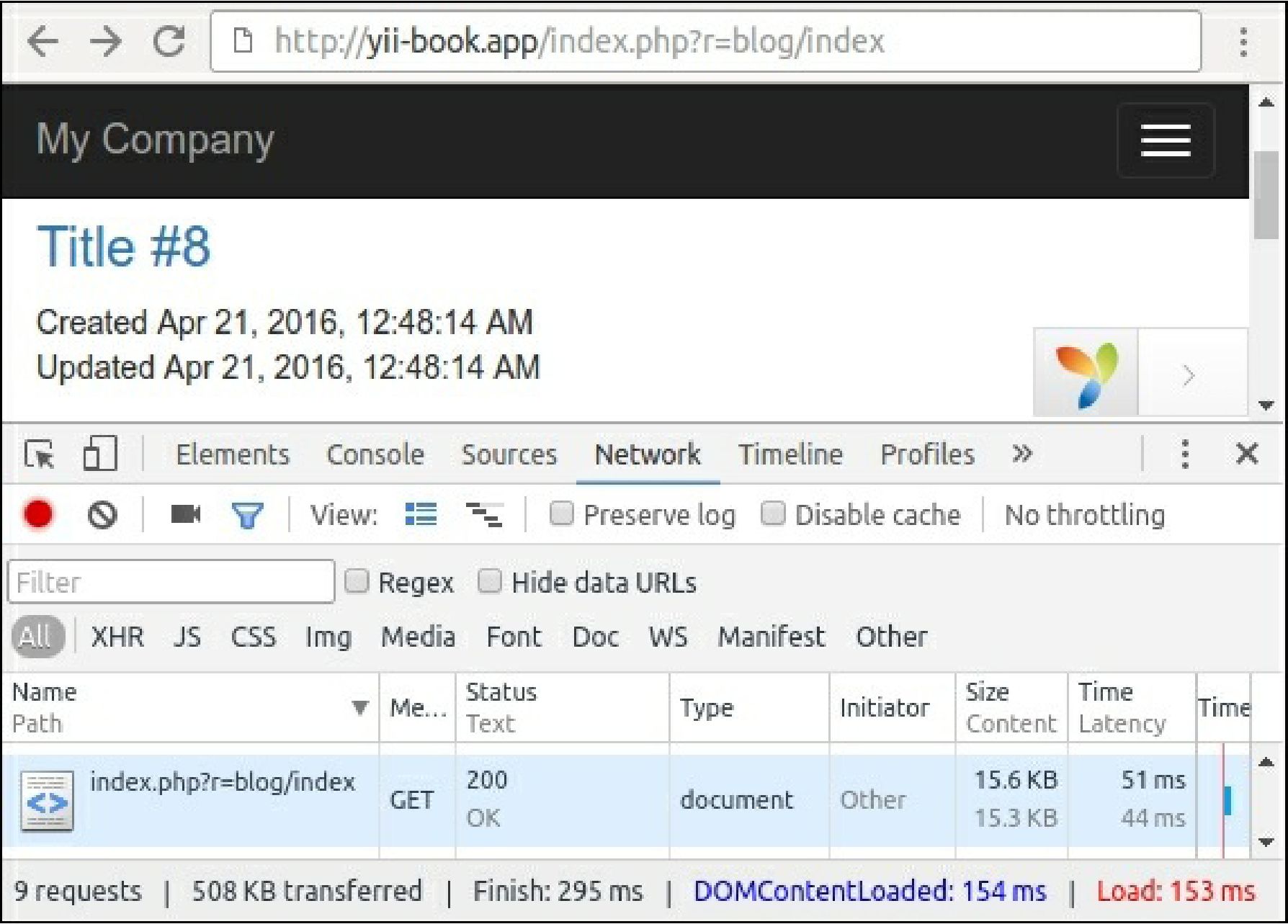
How to do it...

Follow these steps to leverage HTTP caching:

1. Access this URL http://yii-book.app/index.php?r=blog/create three times to generate three articles.
2. Open the following blog page:



3. Open the developer console in your browser and see the 200 ok response status for each reloading  
of the blog page:



4. Open BlogController and attach the following behaviors:

class BlogController extends Controller  
{

public function behaviors()

{

return [

[

'class' => 'yii\filters\HttpCache',

'only' => ['index'],

'lastModified' => function ($action, $params) {  
return Article::find()->max('updated\_at');

},

],

[

'class' => 'yii\filters\HttpCache',

'only' => ['view'],

'etagSeed' => function ($action, $params) {

$article = $this->findModel(\Yii::$app->request->get('id'));  
return serialize([$article->title, $article->text]);

},

],

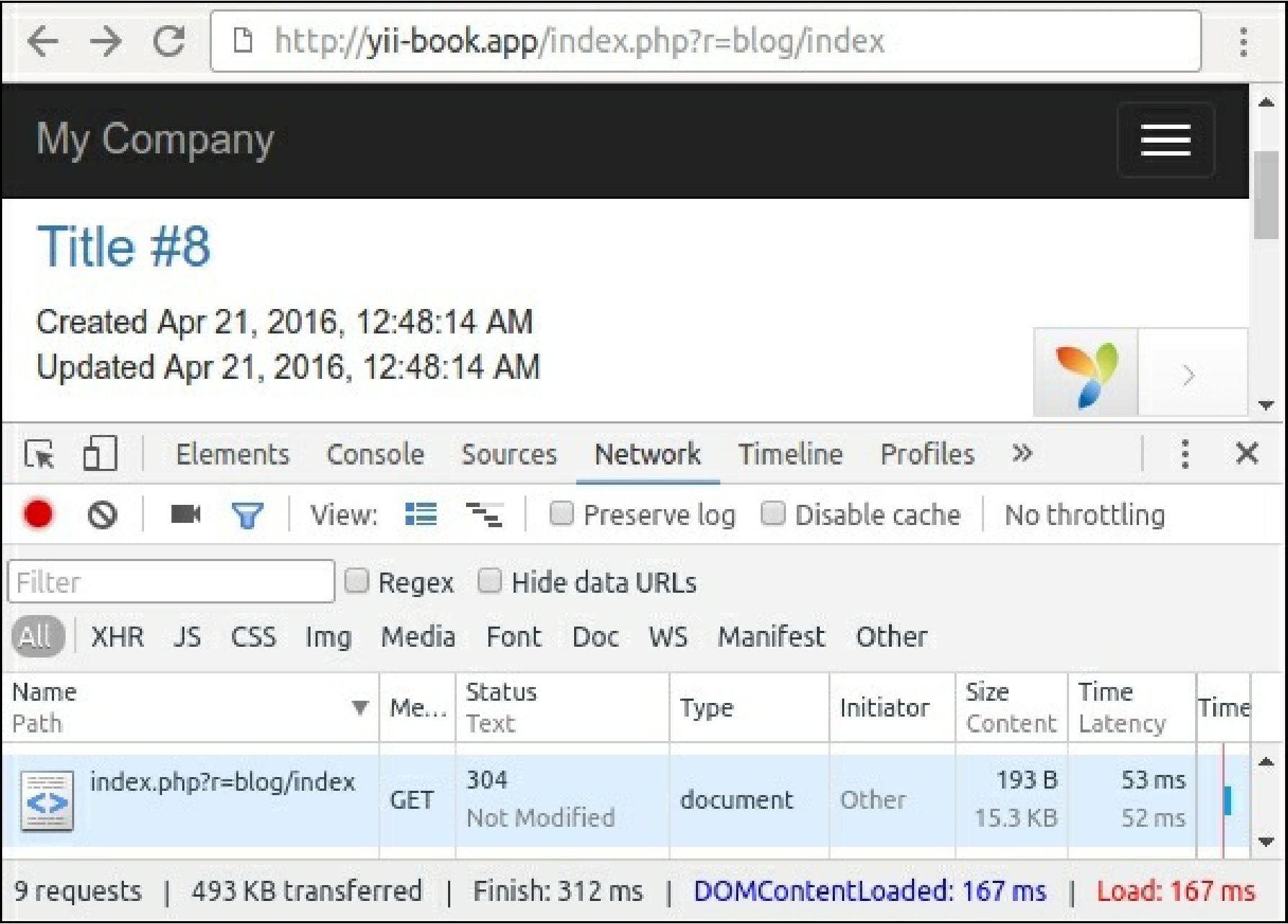
];

}

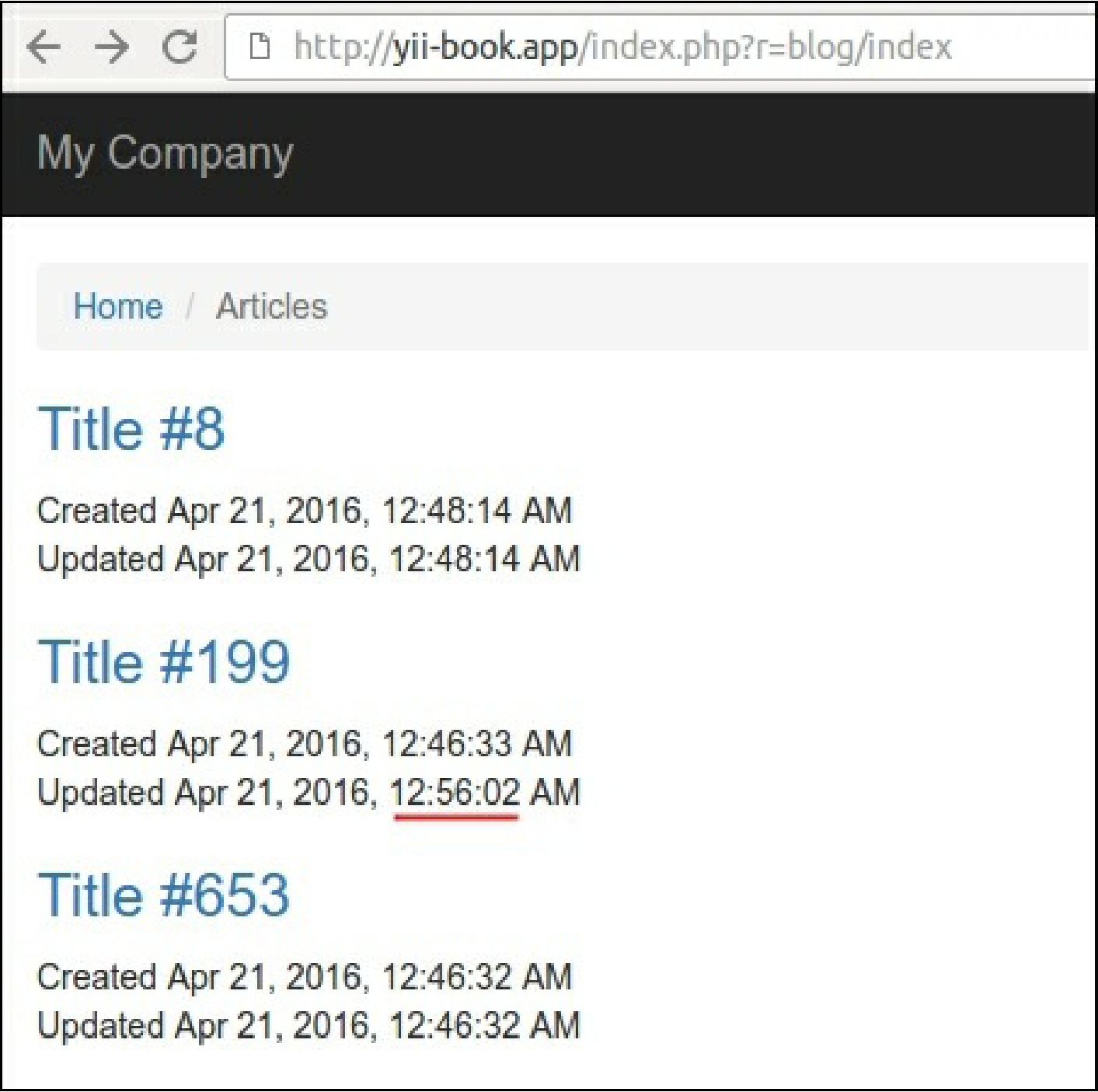
// ...

}

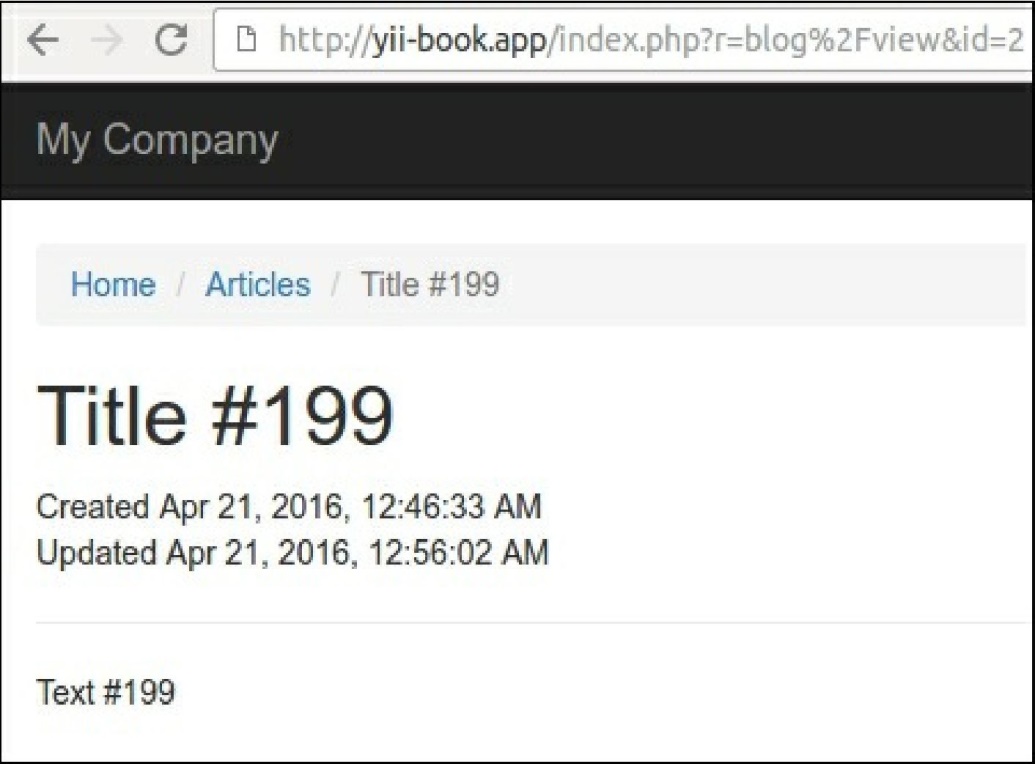
5. Next, reload the page a few times and check that the server returns the 304 Not Modified status  
instead of 200 ok:



1. Open the relevant page using the following URL to update random articles: http: //yii-  
   book.app/index.php?r=blog/update.
2. After updating the blog page, check that the server returns 200 ok the first time and 304 Not  
   Modified thereafter, and verify that you see the new updated time on the page:



8. Open any page from our article, as follows:



Verify that the server returns 200 ok the first time and 304 Not Modified on subsequent requests.

How it works...

There are time-based and content-based approaches to check the availability of the cached response  
content for your browser with the help of HTTP-headers.

Last-Modified

This approach suggests that the server must return the last modification date of every document. After  
storing the date, our browser can attach it in the If-Modified-Since header for every subsequent  
request.

We must attach the action filter to our controller and specify the lastModified callback as follows:

class BlogController extends Controller  
{

public function behaviors()

{

return [

[

'class' => 'yii\filters\HttpCache',

'only' => ['index'],

'lastModified' => function ($action, $params) {  
return Article::find()->max('updated\_at');

],

// ...

];

}

// ...

}

The \yii\filters\HttpCache class calls the callback and compares the returned value with the  
$\_server['http\_if\_modified\_since'] system variable. If the document has still not changed,  
HttpCache will send a lightweight 304 response header without running the action.

However, if the document has been updated, the cache will be ignored and the server will return a full  
response.

|  |  |
| --- | --- |
| **Request** | **Response** |
| First request with full response | |
| GET /index.php?r=blog HTTP 1.1 | HTTP/1.1 200 OK  Cache-Control: public, max-age=3600 Last-Modified: Thu, 21 Apr 2016 00:56:02 GMT  < ! DOCTYPE html>  <html lang="en-US"> |
| Second request with If-Modified-Since with blank response | |
| GET /index.php?r=blog HTTP 1.1 If-Modified-Since: Thu, 21 Apr 2016 00:56:02 GMT | HTTP/1.1 304 Not Modified Cache-Control: public, max-age=3600 |
| Third request after updating the posts with a full response | |
| GET /index.php?r=blog HTTP 1.1 If-Modified-Since: Thu, 21 Apr 2016 00:56:02 GMT | HTTP/1.1 200 OK  Cache-Control: public, max-age=3600 Last-Modified: Thu, 21 Apr 2016 01:12:02 GMT  < ! DOCTYPE html>  <html lang="en-US"> |

As an alternative or an addition to the Last-Modified header variable, you can use ETag.

Entity Tag

In cases when we do not store the last modified date in our documents or pages, we can use custom  
hashes, which can be generated at the base of the document content.

For example, we can use a content title for our document to hash a specific tag:

class BlogController extends Controller  
{

public function behaviors()

{

return [

[

'class' => 'yii\filters\HttpCache',

'only' => ['view'],

'etagSeed' => function ($action, $params) {

$article = $this->findModel(\Yii::$app->request->get('id'));  
return serialize([$article->title, $article->text]);

},

],

];

}

// ...

}

The HttpCache filter will attach this tag to the server response as an ETag header variable.

After storing ETag, our browser can attach it in the If-None-Match header for every subsequent request.

If the document still has not changed, HttpCache will send a lightweight 304 response header without  
running the action.

|  |  |
| --- | --- |
| **Request** | **Response** |
| First request with full response | |
| GET index.php?r=blog/view&id=3 HTTP 1.1 | HTTP/1.1 200 OK  Cache-Control: public, max-age=3600 Etag: "VYkwdOXBzV23KhnzTTJXU"  <!DOCTYPE html>  <html lang="en-US"> |
| Second request with If-None-Match and blank response | |
| GET index.php?r=blog/view&id=3 HTTP 1.1 | HTTP/1.1 304 Not Modified |

|  |  |
| --- | --- |
| If-None-Match: | Cache-Control: public, max-age=3600 |
| "VYkwdOXBzV23KhnzTTJXU" | Etag: "VYkwdOXBzV23KhnzTTJXU" |
| Third request after updating the post with a full response | |
|  | HTTP/1.1 200 OK |
| GET index.php?r=blog/view&id=3 HTTP 1.1 | Cache-Control: public, max-age=3600Etag: "Ur4Ghd6hdYthrn82Ph44dhF" |
| If-None-Match: "VYkwdOXBzV23KhnzTT JXU" | <!DOCTYPE html> <html lang="en-US"> |

When the cache is valid, our application will send the 304 Not Modified response HTTP-headers instead  
of the page content and will not run controllers and actions repeatedly.

See also

* For more information about HTTP caching refer to  
  [https://developers.google.com/web/fundamentals/performanre/optimi7ing-content-efficiency/http-](https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/http-caching)  
  caching
* For HTTP-caching in Yii2 refer to <http://www.yiiframework.com/doc-2.0/guide-caching-http.html>