Rate limiting

To prevent abuse, you should consider adding rate limiting to your APIs. For example, you may want to  
limit the API usage of each user to be, at most, five API calls within a period of one minute. If too many  
requests are received from a user within the stated period of time, a response with the status code 429  
(Too Many Requests) should be returned.

Getting ready

Repeat all the steps from the Creating a REST server recipe’s Getting ready and How to do it...  
sections.

1. Create a migration for creating a user allowance table with the following command:

./yii migrate/create create\_user\_allowance\_table

1. Then, update the just-created migration method, up, with the following code:

public function up()

{

$tableOptions = null;

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

$tableOptions = 'CHARACTER SET utf8 COLLATEutf8\_general\_ci  
ENGINE=InnoDB';

}

$this->createTable('{{%user\_allowance}}', [

'user\_id' => $this->primaryKey(),

'allowed\_number\_requests' => $this->integer(10)->notNull(),  
'last\_check\_time' => $this->integer(10)->notNull()

], $tableOptions);

}

1. Update the down methodwith the following code:

public function down()

{

$this->dropTable('{{%user\_allowance}}');

}

1. Run the created create\_film\_table migration.
2. Generate the UserAllowance model with the Gii module.

How to do it...

First, you have to update @app/controllers/FilmController. php with the following code:

<?php

namespace app\controllers;

use yii\rest\ActiveController;

use yii\filters\RateLimiter;

use yii\filters\auth\QueryParamAuth;

class FilmController extends ActiveController  
{

public $modelClass = 'app\models\Film';

public function behaviors()

{

$behaviors = parent::behaviors();

$behaviors['authenticator'] = [

'class' => QueryParamAuth::className(),

];

$behaviors['rateLimiter'] = [

'class' => RateLimiter::className(),

'enableRateLimitHeaders' => true  
];

return $behaviors;

}

}

To enable rate limiting, the User model class should implement yii\filters\RateLimitInterface and  
requires the implementation of three methods: getRateLimit(), loadAllowance(), and  
saveAllowance(). You have to add them with rate\_limit\_number and rate\_limit\_reset constants:

<?php

namespace app\models;

class User extends \yii\base\Object implements \yii\web\IdentityInterface,  
\yii\filters\RateLimitInterface  
{

public $id;  
public $username;  
public $password;  
public $authKey;  
public $accessToken;

const RATE\_LIMIT\_NUMBER = 5;  
const RATE\_LIMIT\_RESET = 60;

// it means that user allowed only 5 requests per one minute  
public function getRateLimit($request, $action)

{

return [self::RATE\_LIMIT\_NUMBER,self::RATE\_LIMIT\_RESET];

}

public function loadAllowance($request, $action)

{

$userAllowance = UserAllowance::findOne($this->id);  
return $userAllowance ?

[$userAllowance->allowed\_number\_requests,$userAllowance->last\_check\_time] :  
$this->getRateLimit($request, $action);

}

public function saveAllowance($request, $action,$allowance, $timestamp)

{

$userAllowance = ($allowanceModel =UserAllowance::findOne($this->id)) ?  
$allowanceModel : new UserAllowance();

$userAllowance->user\_id = $this->id;

$userAllowance->last\_check\_time = $timestamp;  
$userAllowance->allowed\_number\_requests =$allowance;

$userAllowance->save();

}

// other User model methods

}

How it works...

Once the identity class implements the required interface, Yii will automatically use

[ [yii\filters\RateLimiter] ] configured as an action filter for [[yii\rest\controller]] to perform a  
rate limiting check. We’ve also added the 'authenticator' behavior with the QueryParamAuth class. So,  
we are now able to authenticate with just an access token passed through a query parameter. You can add  
any authentication method that is described in the official guide in the RESTful web services section.

Let’s explain our methods. They are pretty easy to understand.

getRateLimit (): This returns the maximum number of allowed requests and the time period (example,  
[100, 600] means there can be at most 100 API calls within 600 seconds)

loadAllowance(): This returns the number of remaining requests allowed and the corresponding UNIX  
timestamp when the rate limit was last checked

saveAllowance(): This saves both the number of remaining requests allowed and the current UNIX  
timestamp

We store our data in the MySQL database. For performance, you might use a NoSQL database or another  
storage system with a higher time to get and load data.

Now let’s try to check the rate limit feature. Run this in the console:

curl -i "<http://yii-book.app/films?access-token=100-token>"

You will get the following output:

HTTP/1.1 200 OK

Date: Thu, 24 Sep 2015 01:35:51 GMT  
Server: Apache  
X-Powered-By: PHP/5.5.23

Set-Cookie: PHPSESSID=495a928978cc732bee853b83f521eba2; path=/; HttpOnly  
Expires: Thu, 19 Nov 1981 08:52:00 GMT

Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0

Pragma: no-cache

X-Rate-Limit-Limit: 5

X-Rate-Limit-Remaining: 4

X-Rate-Limit-Reset: 0

X-Pagination-Total-Count: 5

X-Pagination-Page-Count: 1

X-Pagination-Current-Page: 1

X-Pagination-Per-Page: 20

Link: <http: //yii-book.app/films?access-token=100-token&page=1>; rel=self  
Content-Length: 301

Content-Type: application/json; charset=UTF-8

[{"id":1,"title":"Interstellar","release\_year":2014},{"id":2,"title":"Harry Potter and  
the Philosopher's Stone","release\_year":2001},{"id":3,"title":"Back to the  
Future","release\_year":1985},{"id":4,"title":"Blade Runner","release\_year":1982},  
{"id":5,"title":"Dallas Buyers Club","release\_year":2013}]

Let’s learn about returned headers. When rate limiting is enabled, by default every response will be sent  
with the following HTTP headers containing the current rate limiting information:

X-Rate-Limit-Limit: This is the maximum number of requests allowed within a time period

X-Rate-Limit-Remaining: This is the number of remaining requests in the current time period

X-Rate-Limit-Reset: This is the number of seconds to wait in order to get the maximum number of  
allowed requests

So, now try to exceed the limit, request the following URL more than five times per minute and you will  
see TooManyRequestsHttpExeption:

HTTP/1.1 429 Too Many Requests  
Date: Thu, 24 Sep 2015 01:37:24 GMT  
Server: Apache  
X-Powered-By: PHP/5.5.23

Set-Cookie: PHPSESSID=bb630ca8a641ef92bd210c0a936e3149; path=/; HttpOnly  
Expires: Thu, 19 Nov 1981 08:52:00 GMT

Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0

Pragma: no-cache

X-Rate-Limit-Limit: 5

X-Rate-Limit-Remaining: 0

X-Rate-Limit-Reset: 60

Content-Length: 131

Content-Type: application/json; charset=UTF-8  
{"name":"Too Many Requests","message":"Rate limit

exceeded.","code":0,"status":429,"type":"yii\\web\\TooManyRequestsHttpException"}

See also

For further information, refer to the following URLs:

* [https://en.wikipedia.org/wiki/Leaky bucket](https://en.wikipedia.org/wiki/Leaky_bucket)
* <http://www.yiiframework.com/doc-2.0/guide-rest-rate-limiting.html>
* [httpy/www.yiiframework.com/doc-2.0/yii-filters-ratelimiter.html](http://www.yiiframework.com/doc-2.0/yii-filters-ratelimiter.html)