Creating a REST server

In the following recipe, we use an example that illustrates how you can build and set up RESTful APIs  
with minimal coding effort. This recipe will be reused in other recipes in this chapter.

Подготовка

1. Создайте новое приложение с помощью диспетчера пакетов Composer, как описано в официальном руководстве по адресу  
   <http://www.yiiframework.com/doc-2.0/guide-start-installation.html>.   
   По русски <http://yiiframework.domain-na.me/doc/guide/2.0/ru/start-installation>.
2. Create a migration for creating an article table with the following command:

./yii migrate/create create\_film\_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 COLLATE  
utf8\_general\_ci ENGINE=InnoDB';

}

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

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

'title' => $this->string(64)->notNull(),

'release\_year' => $this->integer(4)->notNull(),

], $tableOptions);

$this->batchInsert('{{%film}}', ['id','title','release\_year'], [

[1, 'Interstellar', 2014],

[2, "Harry Potter and the Philosopher's Stone",2001],

[3, 'Back to the Future', 1985],

[4, 'Blade Runner', 1982],

[5, 'Dallas Buyers Club', 2013],

]);

}

Update the down method with the following code:

public function down()

{

$this->dropTable('film');

}

1. Run the created create\_film\_table migration.
2. Generate the Film model with the Gii module.
3. Configure your application server to use clean URLs. If you are using Apache with mod\_rewrite  
   and AllowOverride turned on, then you should add the following lines to the . htaccess file under  
   your @web directory:

Options +FollowSymLinks  
IndexIgnore \*/\*

RewriteEngine on

* if a directory or a file exists, use it directly  
  RewriteCond %{REQUEST\_FILENAME} !-f  
  RewriteCond %{REQUEST\_FILENAME} !-d
* otherwise forward it to index.php

RewriteRule . index.php

How to do it...

1. Create a controller, @app/controller/FilmController.php, with the following code:

<?php

namespace app\controllers;

use yii\rest\ActiveController;

class FilmController extends ActiveController  
{

public $modelClass = app\models\Film';

}

Update the @app/config/web. php configuration file. Add the following config of the urlManager  
component:

'urlManager' => [

'enablePrettyUrl' => true,

'enableStrictParsing' => true,

'showScriptName' => false,

'rules' => [

['class' => 'yii\rest\UrlRule', 'controller' => 'films'],

],

],

1. Reconfigure the request component in @app/config/web. php:

'request' => [

'cookieValidationKey' => 'mySecretKey',

'parsers' => [

'application/json' => 'yii\web\JsonParser',

],

]

How it works.

We extend \yii\rest\ActiveController to create our own controller, then for the created controller, the  
modelClass property was set. The \yii\rest\ActiveController class implements a common set of  
actions for supporting RESTful access to ActiveRecord.

With the above minimal amount of effort, you have already finished creating RESTful APIs for accessing  
film data.

The APIs you have created include:

* get /films: This lists all films page by page
* head /films: This shows the overview information of a film listing
* post /films: This creates a new film
* get /films/5: This returns the details of film 5
* head /films/5: This shows the overview information of film 5
* patch /films/5 and put /films/5: This updates film 5
* delete /films/5: This deletes film 5
* options /films: This shows the supported verbs regarding the /films endpoint
* options /films/5: This shows the supported verbs regarding the /films/5 endpoint

It works like this because \yii\rest\ActiveController supports the following actions:

* index: This lists the models
* view: This returns the details of a model
* create: This creates a new model
* update: This updates an existing model
* delete: This deletes an existing model
* options: This returns the allowe d HTTP methods

And there’s also a verbs() method that defines the allowed request methods for each action.

To check that our RESTful API is working correctly, let’s send several requests.

Let’s begin with the get request. Run this in the console:

curl -i -H "Accept:application/json" "<http://yii-book.app/films>"

You will get the following output:

HTTP/1.1 200 OK

Date: Wed, 23 Sep 2015 17:46:35 GMT  
Server: Apache  
X-Powered-By: PHP/5.5.23  
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?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 send a post request. Run this in the console:

curl -i -H "Accept:application/json" -X POST -d title="New film" -d release\_year=2015  
"<http://yii-book.app/films>"

You will get the following output:

HTTP/1.1 201 Created

Date: Wed, 23 Sep 2015 17:48:06 GMT

Server: Apache

X-Powered-By: PHP/5.5.23

Location: <http://yii-book.app/films/6>

Content-Length: 49

Content-Type: application/json; charset=UTF-8  
{"title":"New film","release\_year":"2015", "id":6}

Let’s get the created film. Run in this the console:

curl -i -H "Accept:application/json" "<http://yii-book.app/films/6>"

You will get the following output:

HTTP/1.1 200 OK

Date: Wed, 23 Sep 2015 17:48:36 GMT  
Server: Apache  
X-Powered-By: PHP/5.5.23  
Content-Length: 47

Content-Type: application/json; charset=UTF-8  
{"id":6, "title":"New film", "release\_year":2015}

Let’s send a delete request. Run this in the console:

curl -i -H "Accept:application/json" -X DELETE "<http://yii-book.app/films/6>"

And you will get the following output:

HTTP/1.1 204 No Content

Date: Wed, 23 Sep 2015 17:48:55 GMT

Server: Apache

X-Powered-By: PHP/5.5.23

Content-Length: 0

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

There’s more...

We will now look at content negotiation and customizing the Rest URL rule:

Content negotiation

You can also easily format your response with content negotiation behavior.

For example, you can put this code to your controller and all data will be returned in an XML format.

You should have a look at the full list of formats in the documentation.

use yii\web\Response;  
public function behaviors()

{

$behaviors = parent::behaviors();

$behaviors['contentNegotiator']['formats']['application/xml']= Response::FORMAT\_XML;  
return $behaviors;

}

Run this in the console:

curl -i -H "Accept:application/xml" "<http://yii-book.app/films>"

You will get the following output:

HTTP/1.1 200 OK

Date: Wed, 23 Sep 2015 18:02:47 GMT  
Server: Apache

X-Powered-By: PHP/5.5.23  
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?page=1>>; rel=self  
Content-Length: 516

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

<?xml version="1.0" encoding="UTF-8"?>

<response>

<item>

<id>1</id>

<title>Interstellar</title>

<release\_year>2014

</release\_year>

</item>

<item>

<id>2</id>

<title>Harry Potter and the Philosopher's Stone</title>

<release\_year>2001

</release\_year>

</item>

<item>

<id>3</id>

<title>Back to the Future</title>

<release\_year>1985

</release\_year>

</item>

<item>

<id>4</id>

<title>Blade Runner</title>

<release\_year>1982

</release\_year>

</item>

<item>

<id>5</id>

<title>Dallas Buyers Club</title>

<release\_year>2013

</release\_year>

</item>

</response>

Customizing the Rest URL rule

You have to remember a controller ID, by default, is defined in plural form. This is because  
yii\rest\UrlRule automatically pluralizes controller IDs. You can simply disable this by setting  
yii\rest\UrlRule::$pluralize to false:

'urlManager' => [

//..

'rules' => [

[

'class' => 'yii\rest\UrlRule',

'controller' => 'film'

'pluralize' => false

],

],

//..

]

If you would also like to specify how a controller ID should appear in the patterns, you are able to add a  
custom name to an array as a key value pair, where the array key is the controller ID and the array value  
is the actual controller ID. For example:

'urlManager' => [

//. .

'rules' => [

[

'class' => 'yii\rest\UrlRule',

'controller' => ['super-films' => 'film']

],

],

//..

]

See also

For further information, refer to the following URL:

* [http://www.viiframework.com/doc-2.0/guide-rest-quick-starr.hrml](http://www.yiiframework.com/doc-2.0/guide-rest-quick-start.html)
* <http://www.yiiframework.com/doc-2.0/yii-rest-urlrule.html>
* <http://www.yiiframework.com/doc-2.0/guide-rest-response-formatting.html>
* <http://budiirawan.com/setup-restful-api-yii2/>