Урок 3: Создание простейшего REST API ¶

В этом уроке мы объясним, как создать простейшее приложение, предоставляющее RESTful API с использованием различных HTTP методов:

- GET для получения и поиска данных
- POST для добавления данных
- PUT для обновления данных
- DELETE для удаления данных

Определение API ¶

Наше API содержит следующие методы:

Метод | URL | Действие		
GET	/api/robots	Возвращает всех роботов
GET	/api/robots/search/Astro	роизводит поиск роботов с “Astro” в имени
GET	/api/robots/2	Возвращает робота по его ключу (primary key)
POST	/api/robots	Добавляет нового робота
PUT	/api/robots/2	Обновляет робота по его ключу
DELETE	/api/robots/2	Удаляет робота по его ключу

Создание приложения ¶

Поскольку приложение очень простое, мы не будем включать полное MVC окружение для его разработки. В этом случае для достижения нашей цели мы будем использовать micro application.

Такой структуры файлов будет более, чем достаточно:

```
my-rest-api/
    models/
        Robots.php
    index.php
    .htaccess
```

Прежде всего, нам понадобится файл .htaccess, который содержит все правила перенаправления URI на файл index.php. Для нашего приложения он будет таким:

```
<IfModule mod_rewrite.c>
    RewriteEngine On
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteRule ^(.*)$ index.php?_url=/$1 [QSA,L]
</IfModule>
```

После этого, создаём файл index.php:

```
<?php

$app = new \Phalcon\Mvc\Micro();

// тут определяются роуты

$app->handle();
```

Теперь мы пропишем роуты, как определили выше:

```
<?php

$app = new Phalcon\Mvc\Micro();

// Получение списка всех роботов
$app->get('/api/robots', function() {

});

// Поиск роботов с $name в названии
$app->get('/api/robots/search/{name}', function($name) {

});

// Получение робота по указанному ключу
$app->get('/api/robots/{id:[0-9]+}', function($id) {

});

// Добавление нового робота
$app->post('/api/robots', function() {

});

// Обновление робота по ключу
$app->put('/api/robots/{id:[0-9]+}', function() {

});

// Удаление робота по ключу
$app->delete('/api/robots/{id:[0-9]+}', function() {

});

$app->handle();
```

Каждый роут задан с помощью метода таким же названием, что и HTTP метод. В качестве первого параметра мы передаём шаблон роута, вторым — обработчик, который, в нашем случае является анонимной функцией. Такой роут как ‘/api/robots/{id:[0-9]+}’ однозначно устанавливает, что параметр “id” должен быть числом.

Когда определено соответствие роутов запрашиваемым URI, тогда приложение выполняет соответствующие им обработчики.

Создание модели ¶

Наше API предоставляет информацию о “роботах”, хранящуюся в базе данных. Описанная ниже модель позволяет нам получить доступ к таблице объектно-

ориентированным путём. Мы реализуем немного бизнес-правил, используя встроенные валидаторы с простейшими проверками. Мы делаем это, чтобы иметь уверенность в том, что сохраняемые данные отвечают требованиям нашего приложения:

```
<?php

use Phalcon\Mvc\Model,
    Phalcon\Mvc\Model\Message,
    Phalcon\Mvc\Model\Validator\InclusionIn,
    Phalcon\Mvc\Model\Validator\Uniqueness;

class Robots extends Model
{

    public function validation()
    {
        // Тип робота должен быть: droid, mechanical или virtual
        $this->validate(new InclusionIn(
            array(
                "field"  => "type",
                "domain" => array("droid", "mechanical", "virtual")
            )
        ));

        // Имя робота должно быть уникально
        $this->validate(new Uniqueness(
            array(
                "field"   => "name",
                "message" => "The robot name must be unique"
            )
        ));

        // Год не может быть меньше нулевого
        if ($this->year < 0) {
            $this->appendMessage(new Message("The year cannot be less than zero"));
        }

        // Проверяет, были ли получены какие-либо сообщения при валидации
        if ($this->validationHasFailed() == true) {
            return false;
        }
    }

}
```

Теперь мы должны настроить соединение с базой данных, чтобы использовать его в этой модели

```
<?php

$di = new \Phalcon\DI\FactoryDefault();

// Настройка сервиса базы данных
$di->set('db', function(){
    return new \Phalcon\Db\Adapter\Pdo\Mysql(array(
        "host" => "localhost",
        "username" => "asimov",
        "password" => "zeroth",
        "dbname" => "robotics"
    ));
});

$app = new \Phalcon\Mvc\Micro($di);
```

Получение данных ¶

Сначала мы реализуем обработчик, который отвечает на GET-запрос и возвращает всех доступных роботов. Для выполнения этой задачи будем использовать PHQL,

который будет возвращать результат выполнения простого запроса в формате JSON:

```
<?php

// Получение всех роботов
$app->get('/api/robots', function() use ($app) {

    $phql = "SELECT * FROM Robots ORDER BY name";
    $robots = $app->modelsManager->executeQuery($phql);

    $data = array();
    foreach( $robots as $robot){
        $data[] = array(
            'id' => $robot->id,
            'name' => $robot->name,
        );
    }

    echo json_encode($data);
});
```

PHQL позволяет нам писать запросы с помощью высокоуровневого, объектно-ориентированного SQL-диалекта, которые внутри него будут переведён в правильные SQL-операторы в зависимости от используемой СУБД. Условие “use” при определении анонимной функции позволяет нам легко передать некоторые переменные из глобальной области видимости в локальную.

Обработчик поиска по названию будет выглядеть следующим образом:

```
<?php

// Поиск роботов, в названии которых содержится $name
$app->get('/api/robots/search/{name}', function($name) use ($app) {

    $phql = "SELECT * FROM Robots WHERE name LIKE :name: ORDER BY name";
    $robots = $app->modelsManager->executeQuery($phql, array(
        'name' => '%' . $name . '%'
    ));

    $data = array();

    foreach ($robots as $robot){
        $data[] = array(
            'id' => $robot->id,
            'name' => $robot->name,
        );
    }

    echo json_encode($data);

});
```

В нашем случае поиск по полю “id” очень похож, кроме того, мы сообщаем, найден робот или нет:

```
<?php

// Получение робота по ключу
$app->get('/api/robots/{id:[0-9]+}', function($id) use ($app) {

    $phql = "SELECT * FROM Robots WHERE id = :id:";
    $robot = $app->modelsManager->executeQuery($phql, array(
        'id' => $id
    ))->getFirst();

    //Create a response
    $response = new Phalcon\Http\Response();
```

```
    if ($robot == false) {
        $response->setJsonContent(array('status' => 'NOT-FOUND'));
    } else {
        $response->setJsonContent(array(
            'status' => 'FOUND',
            'data' => array(
                'id' => $robot->id,
                'name' => $robot->name
            )
        ));
    }

    return $response;
});
```

Вставка данных

Получая данные в виде JSON-строки, вставленной в тело запроса, мы точно так же используем PHQL для вставки:

```
<?php

// Добавление нового робота
$app->post('/api/robots', function() use ($app) {

    $robot = $app->request->getJsonRawBody();

    $phql = "INSERT INTO Robots (name, type, year) VALUES (:name:, :type:, :year:)";

    $status = $app->modelsManager->executeQuery($phql, array(
        'name' => $robot->name,
        'type' => $robot->type,
        'year' => $robot->year
    ));

    // Формируем ответ
    $response = new Phalcon\Http\Response();

    //Проверка, что вставка произведена успешно
    if ($status->success() == true) {

        // Изменение HTML статуса
        $response->setStatusCode(201, "Created");

        $robot->id = $status->getModel()->id;

        $response->setJsonContent(array('status' => 'OK', 'data' => $robot));

    } else {

        // Изменение HTML статуса
        $response->setStatusCode(409, "Conflict");

        //Отправляем сообщение об ошибке клиенту
        $errors = array();
        foreach ($status->getMessages() as $message) {
            $errors[] = $message->getMessage();
        }

        $response->setJsonContent(array('status' => 'ERROR', 'messages' => $errors));
    }

    return $response;
});
```

Обновление данных

Обновление данных аналогично их вставке. Полученный параметр “id” сообщает о том, информацию о каком роботе необходимо обновить:

```php
<?php

// Обновление робота по ключу
$app->put('/api/robots/{id:[0-9]+}', function($id) use($app) {

    $robot = $app->request->getJsonRawBody();

    $phql = "UPDATE Robots SET name = :name:, type = :type:, year = :year: WHERE id = :
id:";
    $status = $app->modelsManager->executeQuery($phql, array(
        'id' => $id,
        'name' => $robot->name,
        'type' => $robot->type,
        'year' => $robot->year
    ));

    // Формируем ответ
    $response = new Phalcon\Http\Response();

    // Проверка, что обновление произведено успешно
    if ($status->success() == true) {
        $response->setJsonContent(array('status' => 'OK'));
    } else {

        //Изменение HTML статуса
        $response->setStatusCode(409, "Conflict");

        $errors = array();
        foreach ($status->getMessages() as $message) {
            $errors[] = $message->getMessage();
        }

        $response->setJsonContent(array('status' => 'ERROR', 'messages' => $errors));
    }

    return $response;
});
```

Удаление данных ¶

Удаление очень похоже на обновление. Полученный параметр “id” сообщает о том, какого робота необходимо удалить:

```php
<?php

// Удаление робота по ключу
$app->delete('/api/robots/{id:[0-9]+}', function($id) use ($app) {

    $phql = "DELETE FROM Robots WHERE id = :id:";
    $status = $app->modelsManager->executeQuery($phql, array(
        'id' => $id
    ));

    // Формируем ответ
    $response = new Phalcon\Http\Response();

    if ($status->success() == true) {
        $response->setJsonContent(array('status' => 'OK'));
    } else {

        // Изменение HTTP статуса
        $response->setStatusCode(409, "Conflict");

        $errors = array();
        foreach ($status->getMessages() as $message) {
            $errors[] = $message->getMessage();
        }

        $response->setJsonContent(array('status' => 'ERROR', 'messages' => $errors));

    }
```

```
    return $response;
});
```

Тестирование приложения ¶

Используя curl мы протестируем все роуты нашего приложения для проверки правильности его функционирования:

Получение всех роботов:

```
curl -i -X GET http://localhost/my-rest-api/api/robots

HTTP/1.1 200 OK
Date: Wed, 12 Sep 2012 07:05:13 GMT
Server: Apache/2.2.22 (Unix) DAV/2
Content-Length: 117
Content-Type: text/html; charset=UTF-8

[{"id":"1","name":"Robotina"},{"id":"2","name":"Astro Boy"},{"id":"3","name":"Terminato
r"}]
```

Поиск робота по имени:

```
curl -i -X GET http://localhost/my-rest-api/api/robots/search/Astro

HTTP/1.1 200 OK
Date: Wed, 12 Sep 2012 07:09:23 GMT
Server: Apache/2.2.22 (Unix) DAV/2
Content-Length: 31
Content-Type: text/html; charset=UTF-8

[{"id":"2","name":"Astro Boy"}]
```

Получение робота по id:

```
curl -i -X GET http://localhost/my-rest-api/api/robots/3

HTTP/1.1 200 OK
Date: Wed, 12 Sep 2012 07:12:18 GMT
Server: Apache/2.2.22 (Unix) DAV/2
Content-Length: 56
Content-Type: text/html; charset=UTF-8

{"status":"FOUND","data":{"id":"3","name":"Terminator"}}
```

Добавление робота:

```
curl -i -X POST -d '{"name":"C-3PO","type":"droid","year":1977}'
    http://localhost/my-rest-api/api/robots

HTTP/1.1 201 Created
Date: Wed, 12 Sep 2012 07:15:09 GMT
Server: Apache/2.2.22 (Unix) DAV/2
Content-Length: 75
Content-Type: text/html; charset=UTF-8

{"status":"OK","data":{"name":"C-3PO","type":"droid","year":1977,"id":"4"}}
```

Попытка добавить робота с уже существующим именем:

```
curl -i -X POST -d '{"name":"C-3PO","type":"droid","year":1977}'
    http://localhost/my-rest-api/api/robots

HTTP/1.1 409 Conflict
Date: Wed, 12 Sep 2012 07:18:28 GMT
```

```
Server: Apache/2.2.22 (Unix) DAV/2
Content-Length: 63
Content-Type: text/html; charset=UTF-8

{"status":"ERROR","messages":["The robot name must be unique"]}
```

Или обновление робота с непонятным типом:

```
curl -i -X PUT -d '{"name":"ASIMO","type":"humanoid","year":2000}'
    http://localhost/my-rest-api/api/robots/4

HTTP/1.1 409 Conflict
Date: Wed, 12 Sep 2012 08:48:01 GMT
Server: Apache/2.2.22 (Unix) DAV/2
Content-Length: 104
Content-Type: text/html; charset=UTF-8

{"status":"ERROR","messages":["Value of field 'type' must be part of
    list: droid, mechanical, virtual"]}
```

И, наконец, удаление робота:

```
curl -i -X DELETE http://localhost/my-rest-api/api/robots/4

HTTP/1.1 200 OK
Date: Wed, 12 Sep 2012 08:49:29 GMT
Server: Apache/2.2.22 (Unix) DAV/2
Content-Length: 15
Content-Type: text/html; charset=UTF-8

{"status":"OK"}
```

Заключение ¶

Как видно, с помощью Phalcon легко разработать RESTful API. Позже мы подробно объясним в документации как использовать микро-приложения и язык PHQL.

назад вперёд указатель