We'd like to share our experience in developing a small and easy-to-use api-server on the Yii-Framework. We had a task to write a small backend for an android application and an api-server for it. So, we came up with an idea to combine both the API and Backend in one project. But how can we make it smart and convenient? Our team by trial and error found an effective and fast way to do it and we're glad to spotlight this research.

API on Yii Framework: step by step procedure

We will use a **MySQL database** and the **Yii Framework 1.1.14**, as of writing this article it is the latest stable version (we're expecting version 2.0 to be more stable). Let's skip the details of how to create and adjust an empty project, I think everybody who is interested in this article is aware how to do it. To begin with, we create two models – country and category- with the help of migration.

```
$this->createTable(‘`country`’,                                    array(
            ‘id’=>’pk’,
```

In each table we add two entries. For example, in the country table we add two countries – Russia and India. In the category table we add two categories – category1, category2 and assign different countries to them. We set the task – to write two api-methods of the following type:

1. http://anysite.com/api/getCountryList/ - return country list;

2. http://anysite.com/api/getCategoryList/countryId/1 - return category list of the country. If the country isn't indicated to return all categories.

Firstly, we write a **Response class** that will create the server response as JSON. We set three private properties of the class:

+44 20 7183 5820
info@magora.co.uk
sales@magora.co.uk

```
private $data; //Here we will save answer date;
private $code; //HTTP code – server's answer;
```

Let's set constants for codes and messages:

```
const MSG_OK = 'ok';
const MSG_NO_DATA = 'no data';
```

We redefine "setters" and "getters":

```
public function __set($name, $value) {
```

Then we declare the method that will set the code and messages of the server:

```
public function setCode($code, $message){
```

Finally, we set the very method that sends responses:

```
public function send(){
switch($this->code){
```

Further, we have to create a **controllerApi**, where the list of api-methods will be created and the whole logical process will take place. We set a private property – Response, where we will keep the object of the created Response class.

```
private $response;
public function init() {
```

We then write an actionCall method with a **$method parameter**. This method handles requests to the api-server. Therefore, all the requests will go through it. If there is an api-method and all the parameters are right, the actionCall will call the requested api-method, if not, it returns an error.

```
public function actionCall($method = null) {
```


+44 20 7183 5820
info@magora.co.uk
sales@magora.co.uk

```
array($param->name => Yii::app()->request->getParam($param->name));
>isDefaultValueAvailable()) {
```

In this method we use a so-called Reflection (a total sum of the classes designed to get information about language objects directly when a PHP script is being executed). We used **ReflectionMethod**, because we work with this class.

By means of a **getParameters** method we get the parameters of the requested api-method, then check each parameter with a received GET data. In case the Get parameter name corresponds to the parameter **(if (Yii::app()->request->getParam($param->name)))**, the value of this parameter is set from GET parameter.

If the Get parameter isn't set for this parameter, we check whether this parameter has value by default. If it has it (it means that this parameter isn't necessary to call this api-method), so we set this value **($param->getDefaultValue())**. In case it doesn't have a default value **(if (!$param->isDefaultValueAvailable()))**, we return errors about a wrong request of this api-method.

The next step is to adjust routing in such a way that requests like that: **http://anysite.com/api/<method>/<param>/<value>/...../<param>/<value>** will redirect to the method actionCall. For this purpose, we should configure the UrlManager:

```
'urlManager' => array(
'urlFormat' => 'path',
```

Then we can start writing the api-method that we discussed in the beginning of the article:

```
private function getCountryList() {
if ($cou
```


+44 20 7183 5820

info@magora.co.uk
sales@magora.co.uk

```
:country_id');
```

These are some examples of requests and responses:

1. **Request:**
   http://anysite.com/api/getCountryList/

   Response:

   ```
   {"code":200,"message":"ok","data":{"countries":[{"id":"1","name":"Russia"},
   "id":"2","name":"India"}]}}
   ```

2. **Request:**
   http://anysite.com/api/getCategoryList/

   Response:

   ```
   {"code":200,"message":"ok","data":{"categories":[{"id":"1","name":"category1
   "id":"2","name":"category2"}]}}
   ```

   In case this **api-method** was declared like this:
   private function getCategoryList($countryId),
   the **response** would be:

   ```
   {"code":500,"message":"wrong input data"}
   ```

   because the obligatory attribute of api-method wasn't sent.

3. **Request:**
   http://anysite.com/api/getCategoryList/countryId/1

   Response:

   ```
   {"code":200,"message":"ok","data":{"categories":[{"id":"1","name":"category1
   ```

4. **Request:** if non-existent method.

   Response:

   ```
   {"code":404,"message":"unknown method"}
   ```

Conclusion

+44 20 7183 5820
info@magora.co.uk
sales@magora.co.uk

The result of our work is a very handy tool for implementation of an api-server. We used it in an Android application, where users check all the necessary parameters, make an order and then pay through a bank or use a mobile payment system.

We hope that our tutorial will be helpful for developers, who are looking for ways to combine a server and an API for their applications.

+44 20 7183 5820

info@magora.co.uk
sales@magora.co.uk