API framework planning

Overview:

The main goal is to establish a programming methodology that must allow fast development of APIs. In the other hand, we also need to ensure the quality and maintainability of the code, so we should use a framework to reach the speed and quality goals.

Comparison between frameworks:

Yii:

- Pros:

-- Automatically generate the code skeleton through a neat interface

-- Generates the models without using a layer between the code and the database, so we don't need to create XML or YML files

-- Generates a standard interface and process for most common CRUD operations

-- Good documentation

- Cons:

-- Yii is not an API framework. The code generated by Yii includes view files, which is not the output we need for an API. Besides, the routing method and controller functions are not ready to handle the requests, so we would need to develop a plug-in to put at the top of it.

-- Setting up Yii involves the execution of command lines and running through some interfaces. Not the best thing to the junior developers, cause there is a lost of simplicity

-- The learning curve is not much low.

Frapi:

- Pros:

-- It is a framework for specific API development

-- It gives good practice guidelines to the developers

-- It provides an interface where we can set new API functions. Not much powerful though.

- Cons:

-- Poor documentation

-- Creating an advanced feature requires a deep understand of the framework.

CodeIgniter:

- Pros:

-- Easy to set up

-- Low learning curve.

-- Good documentation

-- Simple and straightforward, and still very powerful and complete

-- It gets to MVC with an easy to understand approach.

- Cons:

-- Is not an API framework. The routing method and controller functions are not ready to handle API requests.

Conclusion:

Frapi is a good API framework, but I wouldn't go with it. It has limited power and creating custom functions is not easy. Even when the team is composed by senior developers, is not a good thing to use a framework that requires a high level of understanding. That's because team changes, and so the new developers will need to learn the framework and go slow at beginning.

Yii is a good option, but we would need to develop a library to handle API requests at top of it, and by doing that we would drop off its major feature, which is to generate the CRUD interfaces and process. So it just lost its major advantage over other frameworks.

There are more frameworks we can analyze: Symfony, CakePHP, Zend Framework. But we must keep in mind we have junior developers in the team, and they will be the major work force, therefore we must choose the most simple and easy to learn framework.

I have no doubts the best option between those is CodeIgniter. I've worked with the major PHP frameworks and systems around there: Symfony, CakePHP, Zend Framework, Drupal, Magento, WordPress. But all of them requires the developer to not only have good knowledges of PHP, but also be an expert in a specific code structure. So the terms “Drupal expert”, “ZF expert”, etc. That's not a good thing in my opinion.

Some of those frameworks uses ORM, which is a database abstraction layer. I don't recommend it neither, it increases the complexity of the system, and the difficulty when doing simple database changes. It's just another element you add, another language to learn, another thing that will require your team to spend time to understand.

Besides, the major purpose of an ORM is to preserve the code layer when the whole

database structure changes. But the best practice is to first create a good database design that won't need to be drastically changed. Anyway, I never saw a system that can be preserved when the very basis of it (i.e., the database) is rot, so the whole point of using ORM is missed.

The best experiences I had was with CodeIgniter. It's easy to learn, it does not requires much knowledge beyond PHP basics. More than once I had a new develop in the team that didn't knew CodeIgniter and was able to just resume a work cause it's so simple and natural. At the same time, it's secure, has great performance, powerful. Also guides the developers to follow a development standard, so the code keeps a high quality.

Implementation Plan:

The way an API is requested determines its kind. We have two major kinds to consider: SOAP and REST. SOAP has a more formal definition, and requires the usage of a WSDL file, so each function or action that can be made, as their parameters, should be described in a specific format in the WSDL file. REST handles the requests through normal HTTP mechanism, and simplify the implementation.

This planning will consider the implementation of a REST API.

Regardless of the qualities of CodeIgniter, it's still not an API framework. How to work around?

First of, what is a RESTful API? It's just a public interface (it may require credentials) that can be accessed by requesting an URL and returns an output in the most usual formats: XML, JSON, HTML.

In that sense, we can just use CodeIgniter normally, with its default routing behavior. So, if we want to, lets say, list all users we have in a database, we would call an URL like

```
http://localhost/myapp/users/get
```

In CodeIgniter end, we would have a file structure like that:

```
– www
   -- myapp
   --- controllers
   ---- users.php
```

–

The URL would access the function “get” inside the users.php file. The users.php file

would look like that:

```
<?php
class User extends CI_Controller {

    function __construct()
    {
          parent::__construct();
          $this->load->model('user_model');
    }

    function get($format='json')
    {
          $users = $this->user_model->get();
          echo json_encode($users);
    }
    function add()
    {
          $data = $_POST;
          $this->user_model->add($data);
    }
?>
```

Note the “$this->user_model”. It's just a class that handles database operations, and it is simple as that:

```
<?php
class User_model extends CI_Model {

    function User_model()
     {
                parent::__construct();
     }
    function get()
    {
     // We can either use CodeIgniter's database abstraction
class with:
     $this->db->select('users.id, users.name');
     $this->db->where('status', 'active');
     $query = $this->db->get('users');

     $users = $query->result_array();

     // Or use raw queries, junior developers will probably
perfer this one:
     $sql = “SELECT users.id, users.name FROM users
```

```
                WHERE status ='active'”;
    $query = $this->db->query($sql);

    $users = $query->result_array();

    return $users;
    }
}
?>
```

We do can go simple as that. The error handling will be in the controllers functions, just simple as that:

```
<?php
    function get($format='json')
    {
    // lets say only authenticated users can access this feature
    $authenticated = $this->user_model-
>authenticate($_POST['username'], $_POST['password']);
    if (!$authenticated) {
          // this
          throw new Exception('Authentication required');
          // or that (even more simple)
          exit('Authentication required');
          return false;
    }
    $users = $this->user_model->get();
    echo json_encode($users);
    }
?>
```

But there are some other REST standard actions to be considered. RESTful services expects 4 kinds of actions: GET, POST, PUT, and DELETE. They are not actually parameters in the URL, but the HTTP request method. If we need to handle those different request methods, for sake of keeping the standard and so external developers can understand what is being done, then we need to put a library at the top of CodeIgniter.

Fortunately, there's a fine third-part library to handle that, and we can easily install it. It is available at https://github.com/philsturgeon/codeigniter-restserver.

The implementation of that library is also very simple. Using the same example, we would do:

```
<?php
require(APPPATH'.libraries/REST_Controller.php');
class User extends REST_Controller {
    function user_get()
    {
     // lets say only authenticated users can access this feature
     $authenticated = $this->user_model-
>authenticate($_POST['username'], $_POST['password']);
     if (!$authenticated) {
          $this->response(NULL, 501);
     }
     $users = $this->user_model->get();
     if ($users) {
          $this->response($users, 200);
     }
     else {
          $this->response(NULL, 404);
     }
    }
    function user_post() {
    }
    function user_put() {

    }
    function user_delete() {

    }
}
?>
```

Each request method triggers a function. The naming convention is the sufix of the function. Pretty much obvious: DELETE triggers “user_delete” and so on.

Note using this library is easy enough, and it handles error with the “$this->response” method. It also allows you to just pass an additional parameter in the URL so you can choose the output format between JSON, XML, or HTML. No need to handle it by hand in the controller side.

Besides, we have some configuration options so we can secure the API and restrict the access to the functions.

The final end point

Finally, we might need a way to do the requests to our API. External developers may use whatever they want, with any language. We can use and distribute a PHP library though: https://github.com/philsturgeon/codeigniter-restclient

It's simple to be used, like that:

```
<?php
    function users_client() {
    $this->load->library('rest', array(
          'server'=>'http://localhost/myapp/users',
          'http_user'=>'myusername',
          'http_pass'=>'mypassword'
    ));

    $user = $this->rest->get('user', 'json');

    echo $user->name;
    }
?>
```

Note the “$this->rest->get()”. Using it, our request method will be GET. For a DELETE request method, we'll do $this->rest->delete(). Very simple and obvious.

Of course, we can use cURL to trigger the requests, as we do for any RESTful services. There are CodeIgniter libraries that makes easy to use cURL. So we can either use one or another, there's no much difference. It will depend totally on the developer's preference, but both are just fine.

There is a good tutorial for the implementation of those libraries here: http://net.tutsplus.com/tutorials/php/working-with-restful-services-in-codeigniter-2/