

REST API & Implementing it in CodeIgniter

Proyecto; Api en Codeigniter 3

Juan Maria Dominguez Ortiz
IES Polígono Sur
2ºASIR

Índice

Índice	1
Apartado 0- Introducción y explicación.	2
Apartado 1 - Descargas y configuración.	3
Apartado 2 - Base de datos.	5
Apartado 3 - Controlador y Rutas.	6
Apartado 4 - Creación y lógica del Controlador “Sitios.php”.	7
Apartado 5 - Creación y lógica del Modelo “Sitios_model.php”.	8
Apartado 6 - Creación y lógica para el controlador Fiestas.php.	9
Apartado 7 - Creación y lógica para el modelo “Fiestas_model.php” .	10
Apartado 8- Conclusiones.	11
Apartado 9- Referencias	11

Apartado 0- Introducción y explicación.

Proyecto Fiestas y Sitios- Juan Maria Dominguez

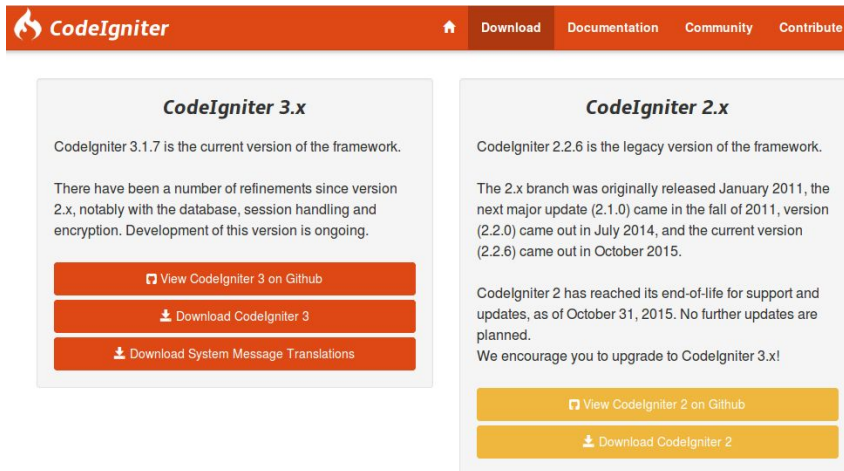
Mi API Rest - Programa(Explicación):

Con la realización de este proyecto de “API en el frameworks Codeigniter” lo que se ha querido conseguir y realizar es un programa el cual vinculado con una Base de Datos existente y configurada, pueda elegir, añadir, actualizar y/o borrar tanto sitios como fiestas y sus respectivas fechas, de manera que podamos crear una fiesta que se va a realizar en un determinado sitio u zona con una fecha concreta quedando toda la información centralizada y bien estructurada, todos estos datos podremos gestionarlos y manejarlos con peticiones vía web.

Los datos de esta API estarían almacenados en una Base de datos existente, preconfigurada y actualizable, todo configurado con y bajo el frameworks Codeigniter.

Apartado 1 - Descargas y configuración.

En primer lugar descargamos CodeIgniter 3 y el siguiente paquete de Github.



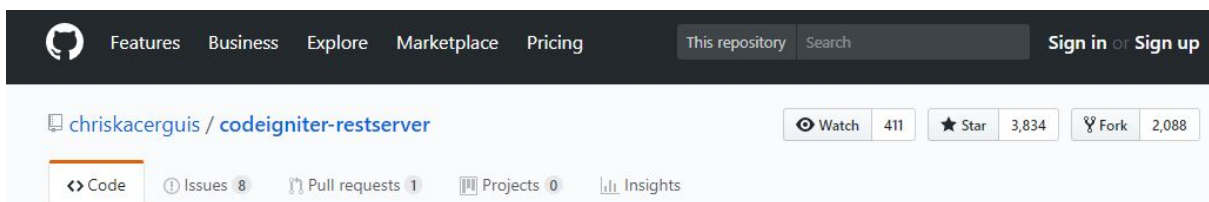
The screenshot shows the CodeIgniter website with a navigation bar at the top containing links for Home, Download, Documentation, Community, and Contribute. Below the navigation bar, there are two main sections: 'CodeIgniter 3.x' and 'CodeIgniter 2.x'. The 'CodeIgniter 3.x' section states that version 3.1.7 is the current version and provides links to 'View CodeIgniter 3 on Github', 'Download CodeIgniter 3', and 'Download System Message Translations'. The 'CodeIgniter 2.x' section states that version 2.2.6 is the legacy version and provides links to 'View CodeIgniter 2 on Github' and 'Download CodeIgniter 2'. It also mentions that CodeIgniter 2 has reached its end-of-life for support and updates as of October 31, 2015.

```
root@usuario-VirtualBox:/home/usuario# wget https://github.com/bcit-ci/CodeIgniter/archive/3.1.7.zip
```

Desempaquetamos y ya lo tenemos.

```
root@usuario-VirtualBox:/home/usuario/proyecto# ls
CodeIgniter-3.1.7
```

Paquete Rest server



The screenshot shows the GitHub repository page for 'chriskacerguis / codeigniter-restserver'. The repository has 411 watches, 3,834 stars, and 2,088 forks. The 'Code' tab is selected, showing the repository's structure with links to Issues (8), Pull requests (1), Projects (0), and Insights.

```
root@usuario-VirtualBox:/home/usuario/proyecto# wget https://github.com/chriskacerguis/codeigniter-restserver/archive/master.zip
```

Desempaquetamos y ya lo tenemos.

```
root@usuario-VirtualBox:/home/usuario/proyecto# ls -l
total 8
drwxr-xr-x 5 root root 4096 ene 13 12:57 CodeIgniter-3.1.7
drwxr-xr-x 4 root root 4096 feb 27 22:35 codeigniter-restserver-master
```

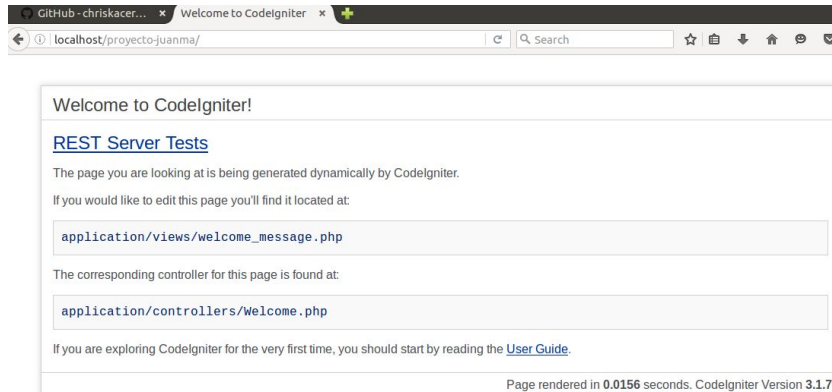
Le cambiamos el nombre al frameworks

```
usuario@usuario-VirtualBox:~/proyecto$ ls -l
total 8
drwxr-xr-x 5 root root 4096 ene 13 12:57 CodeIgniter-3.1.7
drwxr-xr-x 4 root root 4096 feb 27 22:35 codeigniter-restserver-master
usuario@usuario-VirtualBox:~/proyecto$ sudo mv CodeIgniter-3.1.7/ Proyecto-Juanma
```

Lo primero que vamos a hacer es mover nuestro proyecto a /var/www/html

```
usuario@usuario-VirtualBox:~/proyecto$ sudo mv proyecto-Juanma/ /var/www/html/
```

Probamos una petición web



Destacar que anteriormente para probarlo necesitamos instalar paquetes Php.

Por consiguiente vamos a copiar todo el contenido de la carpeta application del proyecto Github a nuestro frameworks codeigniter.

```
usuario@usuario-VirtualBox:~/proyecto/codeigniter-restserver-master/application$ sudo cp -r * /var/www/html/proyecto-juanma/application/
```

Apartado 2 - Base de datos.

Configuramos el archivo que obtendrá los recursos de nuestra base de datos de nuestro proyecto.

```
root@usuario-VirtualBox: /var/www/html/proyecto-juanma/application/config
GNU nano 2.5.3 Archivo: database.php

$db['default'] = array(
    'dsn' => '',
    'hostname' => 'localhost',
    'username' => 'root',
    'password' => 'root',
    'database' => 'fiestas',
    'dbdriver' => 'mysqli',
    'dbprefix' => '',
    'pconnect' => FALSE,
    'db_debug' => (ENVIRONMENT !== 'production'),
    'cache_on' => FALSE,
    'cachedir' => '',
    'char_set' => 'utf8',
    'dbcollat' => 'utf8_general_ci',
    'swap_pre' => '',
    'encrypt' => FALSE,
    'compress' => FALSE,
    'stricton' => FALSE,
    'failover' => array(),
    'save_queries' => TRUE
);
```

Ahora toca crear la base de datos

Para ello instalamos el paquete mysql-server

Entramos y creamos la base datos con sus tablas.

```
usuario@usuario-VirtualBox:~$ mysql -u root -p
Enter password:
```

```
mysql> CREATE DATABASE fiestas;
Query OK, 1 row affected (0,00 sec)
```

```
mysql> USE fiestas;
Database changed
```

```
mysql> CREATE TABLE sitios(
  -> id INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
  -> NAME VARCHAR(100) NULL
  -> );
Query OK, 0 rows affected (0,03 sec)
```

```
mysql> CREATE TABLE fiestas(
  -> id INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
  -> fiestas VARCHAR(100) NULL,
  -> `date` DATETIME NULL,
  -> id_sitio INT NOT NULL,
  -> FOREIGN KEY (id_sitio) REFERENCES sitios(id)
  -> );
Query OK, 0 rows affected (0,21 sec)
```

Creamos las base de datos fiestas la cual va a contener;

Fiestas →

Sitios

Fiestas

Apartado 3 - Controlador y Rutas.

Una vez realizado lo anterior el paso siguiente es Crear el controlador Sitios que es el que se va a encargar de añadir editar o borrar los sitios.

```
GNU nano 2.5.3 Archivo: Sitios.php Modificad
<?php
defined('BASEPATH') OR exit('No direct script access allowed');
require_once APPPATH . '/libraries/REST_Controller.php';

class Sitios extends REST_Controller {

    public function index_get() /**Nos va a mostrar todos los sitios */
    {

    }

    public function find_get($id) /**Nos va a dar un sitio en concreto y le pasamos un id*/
    {

    }

    public function index_post() /**Se va a encargar de añadir un nuevo sitio*/
    {

    }
    public function index_put() /**Este va actualizar un registro en la BD*/
    {

    }
    public function index_delete() /**Encargado de borrar */
    {

    }

}
```

Ahora vamos a establecer las rutas para cada uno de los métodos anteriores.

```
$route['default_controller'] = 'Sitios';
$route['404_override'] = '';
$route['translate_uri_dashes'] = TRUE;

// Rutas para los sitios
$route['sitios']['get'] = 'sitios/index';
$route['sitios/(:num)']['get'] = 'sitios/find/$1';
$route['sitios']['post'] = 'sitios/index';
$route['sitios/(:num)']['put'] = 'sitios/index/$1';
$route['sitios/(:num)']['delete'] = 'sitios/index/$1';
```

Ahora toca crear nuestro modelo

```
GNU nano 2.5.3 Archivo: Sitios_model.php
<?php

class Cities_model extends CI_Model
{
    public function __construct()
    {
        parent::__construct();
    }

}
```

Una vez creamos el modelo en nuestro controlador Sitios tenemos que crear un constructor y llamar al modelo

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

Apartado 4 - Creación y lógica del Controlador “Sitios.php”.

Bien una vez tenemos todo esto realizado ahora vamos a pasar a crear la lógica de nuestro control de todas y cada una de las funciones que hicimos en nuestro Sitios.php

```
public function index_get() /**Nos va a mostrar todos los sitios */
{
    $sitios = $this->sitios_model->get(); /**El siguiente metodo nos devuelve todas las ciudades*/

    if (is_null($sitios)) {
        $this->response(array('response' => $sitios), 200); /**Array que contiene los sitios*/
    } else {
        $this->response(array('error' => 'No hay sitios en la base de datos...'), 404); /**Array de muestra si no encuentra datos*/
    }
}

public function find_get($id) /**Nos va a dar un sitio en concreto y le pasamos un id*/
{
    if (!$id) { /**Metodo para encontrar un solo sitio en concreto, aqui le indicamos que necesita una id, sino nos va a tirar error*/
        $this->response(null, 400);
    }

    $sitio = $this->sitios_model->get($id);

    if (is_null($sitio)) {
        $this->response(array('response' => $sitio), 200);
    } else {
        $this->response(array('error' => 'Sitio no encontrada...'), 404);
    }
}
```

```
public function index_post() /**Se va a encargar de añadir un nuevo sitio*/
{
    if (!$this->post('sitio')) { /**Metodo para comprobar que nos viene un dato sitio*/
        $this->response(null, 400);
    }

    $id = $this->sitios_model->save($this->post('sitio')); /**Si viene un dato sitio lo insertamos en la BD*/

    if (is_null($id)) {
        $this->response(array('response' => $id), 200);
    } else {
        $this->response(array('error', 'Algo va mal...'), 400);
    }
}

public function index_put() /**Este va actualizar un registro en la BD*/
{
    if (!$this->put('sitio')) { /** Con este metodo vamos a indicarle cual es el sitio en concreto que queremos editar, y no nos e
        $this->response(null, 400);
    }
}

$update = $this->sitios_model->update($this->put('sitio'));

if (is_null($update)) {
    $this->response(array('response' => 'Sitio actualizada!'), 200);
} else {
```

```
public function index_delete() /**Encargado de borrar */
{
    if (!$id) { /**En este caso es lo mismo que el anterior pero borrando el sitio en concreto, y no nos borra todos*/
        $this->response(null, 400);
    }

    $delete = $this->sitios_model->delete($id);

    if (is_null($delete)) {
        $this->response(array('response' => 'Sitio eliminado!'), 200);
    } else {
        $this->response(array('error', 'Algo va mal...'), 400);
    }
}
```


Estos 5 métodos citados anteriormente lo que nos va a realizar en resumidas cuenta es

Método 1→ Obtiene todos los sitios.

Método 2→ Obtiene un sitio en concreto.

Método 3→ Añade un nuevo sitio y lo guarda en la BD.

Método 4→ Actualiza un registro y lo guarda en la BD.

Método 5→ Borra un registro.

Apartado 5 - Creación y lógica del Modelo

“Sitios_model.php”.

Una vez tenemos realizada la lógica del controlador lo siguiente es pasar a crear la lógica de nuestro Modelo “Sitios_model.php para que se corresponda con la lógica del controlador.”

```
public function get($id = null) /**Metodo para obtener una o todas los sitios*/
{
    if (!is_null($id)) {
        $query = $this->db->select('*')->from('sitios')->where('id', $id)->get(); /**Realizamos una consulta a la BD para poder mostrarnos los datos*/
        if ($query->num_rows() === 1) {
            return $query->row_array();
        }
        return null;
    }

    $query = $this->db->select('*')->from('sitios')->get();
    if ($query->num_rows() > 0) {
        return $query->result_array();
    }
    return null;
}

public function save($sitio) /** Metodo para insertar una nueva ciudad*/
{
    $this->db->set($this->_setSitio($sitio))->insert('sitios'); /**Realizamos un insert a la BD de los nuevos datos*/

    if ($this->db->affected_rows() === 1) {
        return $this->db->insert_id();
    }

    return null;
}

public function update($sitio) /**Metodo para actualizar un sitio ya existente*/
{
    $id = $sitio['id'];

    $this->db->set($this->_setSitio($sitio))->where('id', $id)->update('sitios'); /**Realizamos una consulta y update a la BD*/

    if ($this->db->affected_rows() === 1) {
        return true;
    }

    return null;
}

public function delete($id) /**Metodo para borrar un sitio*/
{
    $this->db->where('id', $id)->delete('sitios'); /**Realizamos un delete del dato a la BD*/

    if ($this->db->affected_rows() === 1) {
        return true;
    }

    return null;
}

private function _setSitio($sitio) /**Metodo para ahorrarnos código con el save y update */
{
    return array(
        'name' => $sitio['name']
    );
}
```

Apartado 6 - Creación y lógica para el controlador Fiestas.php.

Lo primero antes de nada es crearnos el controlador fiestas, vamos a establecer las rutas que serían la misma que con la de sitios.

```
// Rutas para las fiestas
$route['fiestas']['get'] = 'fiestas/index';
$route['fiestas/(:num)']['get'] = 'fiestas/find/$1';
$route['fiestas']['post'] = 'fiestas/index';
$route['fiestas/(:num)']['put'] = 'fiestas/index/$1';
$route['fiestas/(:num)']['delete'] = 'fiestas/index/$1';
```

Ahora si creamos en nuestro controlador el “Fiestas.php” que va a contener las mismas funciones que las de Sitios.php pero cambiando las nomenclaturas, la programación es la misma que la anterior.

```
GNU nano 2.5.3 Archivo: Fiestas.php

<?php
defined('BASEPATH') OR exit('No direct script access allowed');

require_once APPPATH . '/libraries/REST_Controller.php';

class Fiestas extends REST_Controller {

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

    public function index_get() /**Nos va a mostrar todas las fiestas */
    {
        $fiestas = $this->fiestas_model->get(); /**El siguiente metodo nos devuelve todas las fiestas*/

        if (is_null($fiestas)) {
            $this->response(array('response' => $fiestas), 200); /**Array que contiene las fiestas*/
        } else {
            $this->response(array('error' => 'No hay fiestas en la base de datos...'), 404); /**Array de muestra si no encuentra datos*/
        }
    }

    public function find_get($id) /**Nos va a dar una fiesta en concreto y le pasamos un id*/
    {
        if (!$id) { /**Metodo para encontrar una sola fiesta en concreto, aqui le indicamos que necesita una id, sino nos va a tirar error*/
            $this->response(null, 400);
        }

        $fiesta = $this->fiestas_model->get($id);

        if (is_null($fiesta)) {
            $this->response(array('response' => $fiesta), 200);
        } else {
            $this->response(array('error' => 'Fiesta no encontrada...'), 404);
        }
    }

    public function index_post() /**Se va a encargar de añadir una nueva fiesta*/
    {
        if (!$this->post('fiesta')) { /**Metodo para comprobar que nos viene un dato fiesta*/
            $this->response(null, 400);
        }

        $id = $this->fiestas_model->save($this->post('fiesta')); /**Si viene un dato fiesta lo insertamos en la BD*/

        if (is_null($id)) {
            $this->response(array('response' => $id), 200);
        } else {
            $this->response(array('error', 'Algo va mal...'), 400);
        }
    }
}
```

```

public function index_put() /**Este va actualizar un registro en la BD*/
{
    if (!$this->put('fiesta')) { /** Con este metodo vamos a indicarle cual es la fiesta en concreto que queremos editar, y no nos edita todos*/
        $this->response(null, 400);
    }
    $update = $this->fiestas_model->update($this->put('fiesta'));

    if (!is_null($update)) {
        $this->response(array('response' => 'fiesta actualizada!'), 200);
    } else {
        $this->response(array('error', 'Algo va mal...'), 400);
    }
}

public function index_delete() /**Encargado de borrar */
{
    if (!$id) { /**En este caso es lo mismo que el anterior pero borrando la fiesta en concreto, y no nos borra todas*/
        $this->response(null, 400);
    }

    $delete = $this->fiestas_model->delete($id);

    if (!is_null($delete)) {
        $this->response(array('response' => 'Fiesta eliminado!'), 200);
    } else {
        $this->response(array('error', 'Algo va mal...'), 400);
    }
}

```

Estos 5 métodos citados anteriormente lo que nos va a realizar en resumidas cuenta es lo mismo que en el primer Sitios.php

Método 1→ Obtiene todos las fiestas.

Método 2→ Obtiene una fiesta en concreto.

Método 3→ Añade una nueva fiesta y lo guarda en la BD.

Método 4→ Actualiza un registro y lo guarda en la BD.

Método 5→ Borra un registro.

Apartado 7 - Creación y lógica para el modelo “Fiestas_model.php” .

Una vez tenemos nuestro controlador queda crear su respectivo modelo y logica del “Fiestas_model.php”

```

GNU nano 2.5.3 Archivo: Fiestas_model.php

<?php
defined('BASEPATH') OR exit('No direct script access allowed');

class Fiestas_model extends CI_Model
{
    public function __construct()
    {
        parent::__construct();
    }

    public function get($id_sitio = null) /**Metodo para obtner una o todas las fiestas*/
    {
        if (!is_null($id_sitio)) { /**Realizamos una cosnulta en base al id del sitio a la BD para poder mostras los datos*/
            $query = $this->db->select('*')->from('fiestas')->where('id_sitio', $id_sitio)->order_by('date', 'ASC')->get();
            if ($query->num_rows() > 0) {
                return $query->result_array();
            }
            return null;
        }

        $query = $this->db->select('*')->from('fiestas')->get();
        if ($query->num_rows() > 0) {
            return $query->result_array();
        }

        return null;
    }

    public function save($fiestas) /** Metodo para insertar una nueva fiesta*/
    {
        $this->db->set($this->_setFiestas($fiestas))->insert('fiestas'); /**Realizamos un insert a la BD de los nuevos datos*/

        if ($this->db->affected_rows() === 1) {
            return $this->db->insert_id();
        }

        return null;
    }
}

```

```
public function update($id, $fiestas) /**Metodo para actualizar una fiesta ya existente**/
{
    $this->db->set($this->_setFiestas($fiestas))->where('id', $id)->update('fiestas'); /**Realizamos una consulta y update a la BD**/
    if ($this->db->affected_rows() === 1) {
        return true;
    }
    return null;
}

public function delete($id) /**Metodo para borrar una fiesta**/
{
    $this->db->where('id', $id)->delete('fiestas'); /**Realizamos un delete del dato a la BD**/
    if ($this->db->affected_rows() === 1) {
        return true;
    }
    return false;
}

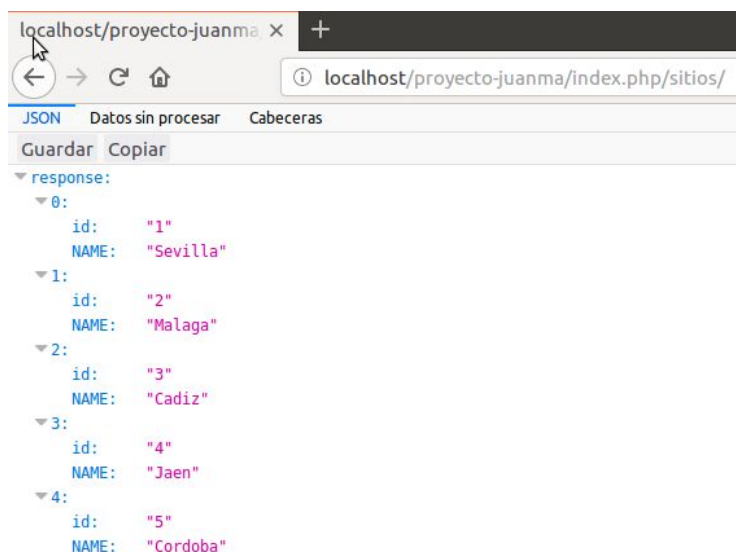
private function _setFiestas($fiestas) /**Metodo para ahorrarnos codigo con el save y update **/
{
    return array(
        'fiestas' => $fiestas['fiestas'],
        'date' => $fiestas['date'],
        'id_sitio' => $fiestas['id_sitio']
    );
}
```

Apartado 8- Pruebas y Conclusiones.

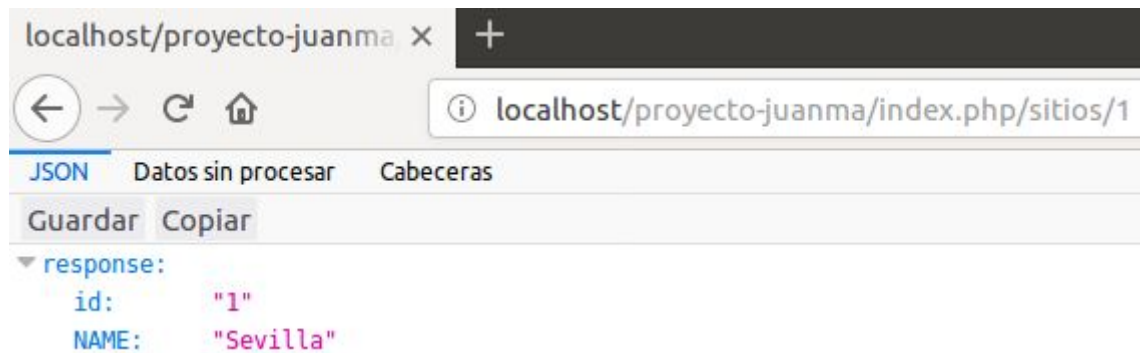
Hasta aquí sería todo en base a la configuración de nuestra API.

Lo siguiente que vamos a realizar es peticiones web de nuestra API la cual nos devolverá datos de la base de datos sincronizada con el programa la cual previamente tiene que tener unos datos o introducirlos.

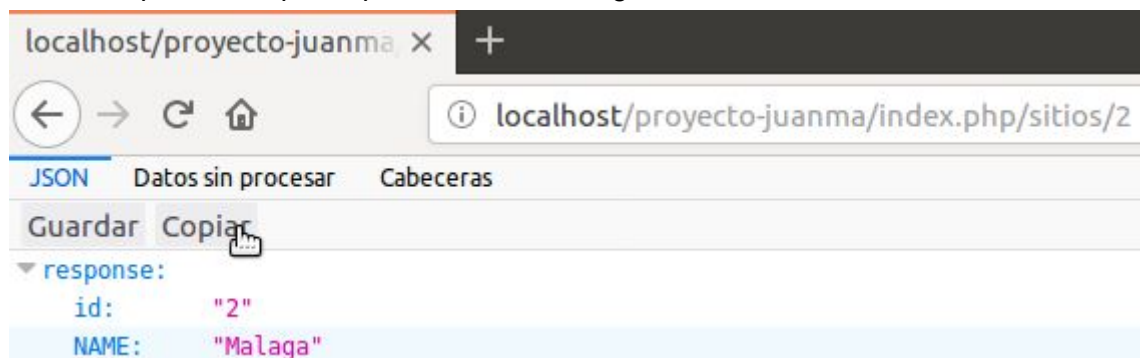
La primera petición que le enviamos a nuestra API es la de ver todos los sitios posibles, gracias a la función que creamos index_get.



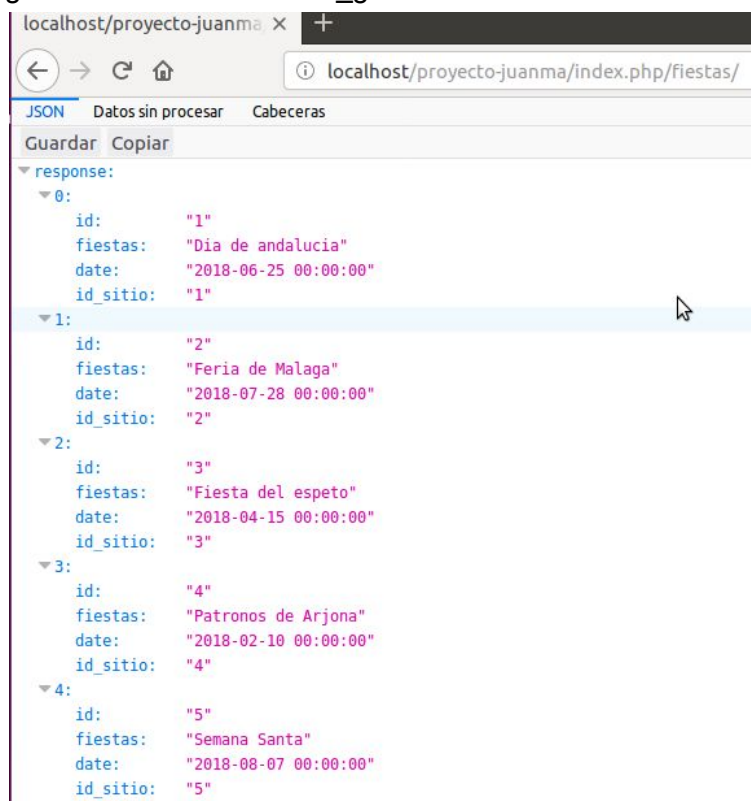
La siguiente petición que enviamos es la de individualizar por un código y que nos muestre solo esos datos con el código enviado, es decir si le pasamos 1 nos mostrará Sevilla.



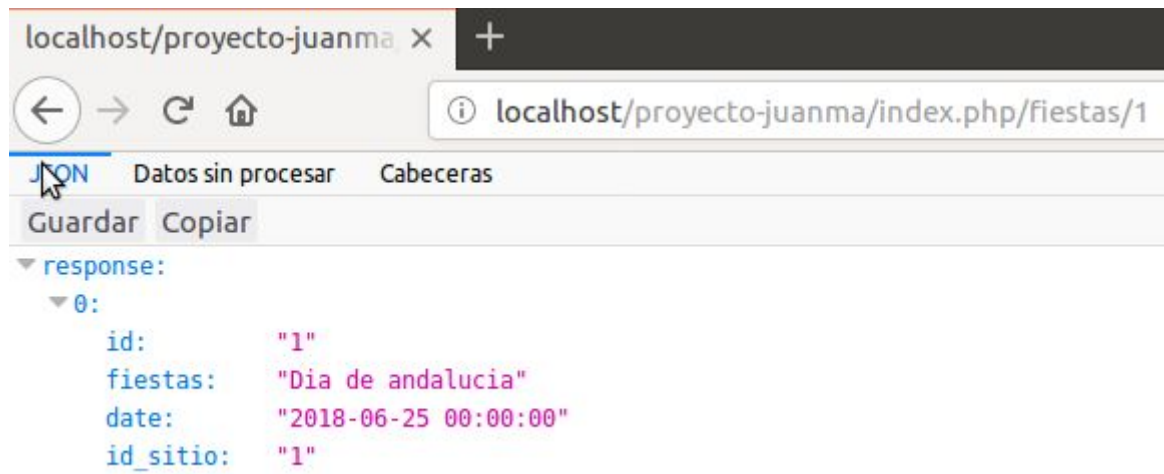
Lo mismo para otro tipo de petición con el código 2.



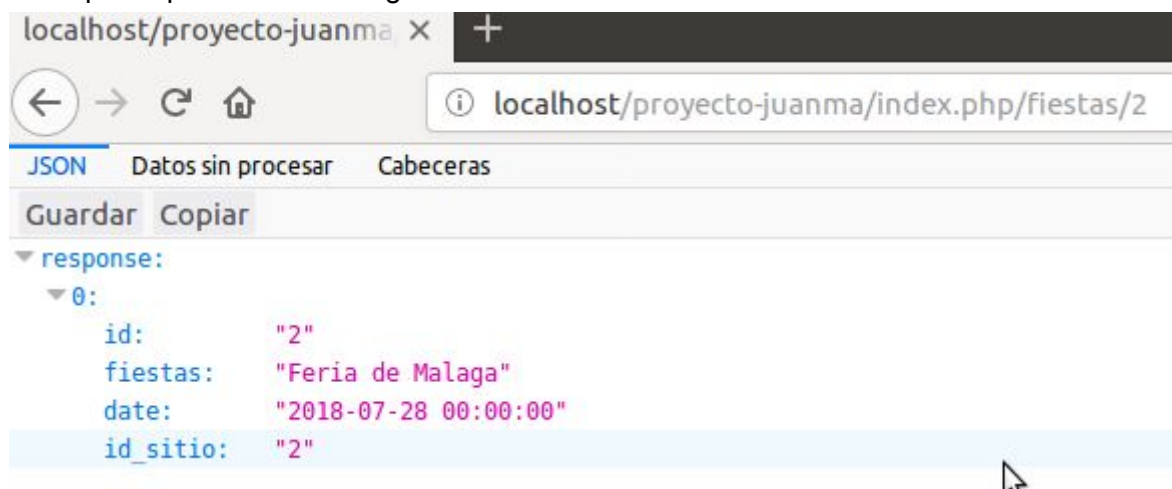
Lo siguiente es una vez tenemos los sitios, su código y nombre es mostrar las fiestas de esa Provincia realizando la siguiente petición, que logra mostrarnos todas con todas sus fiestas gracias a la función index_get.



Si lo que quisiéramos como en sitios es individualizar la petición podríamos realizarlo también realizando la petición con el código que corresponde a la sección.



Otro tipo de petición con código diferente.



Para añadir actualizar y borrar datos lo realizamos con peticiones CURL

```
root@juanma-VirtualBox: /home/juanma
root@juanma-VirtualBox:/home/juanma# curl -X POST -H 'Content-Type: application/json' -d '{"id":"6","NAME":"Almeria"}' localhost/proyecto-juanma/index.php/sitios
root@juanma-VirtualBox:/home/juanma#
```

Apartado 9- Referencias

- <https://www.juanwilde.com/api-rest-codeigniter-3-frontend-angularjs/>
- <https://github.com/JuanWilde/WeatherAPI>
- <https://www.youtube.com/watch?v=-F7FsNrxAM&list=FLbY3LfM7LtSA7KP5Rwrqbzg>
- <http://www.desarrollolibre.net/blog/tema/256/codeigniter/como-crear-una-api-rest-con-codeigniter>