Metodologías y Tecnologías de la Integración de Sistemas

Ingeniería Informática

Arancha Ferrero Ortiz de Zárate
72850555X

Practica 2

Diseño Api Rest: Raml

4 de marzo del 2018

Descripción de la práctica

El objetivo de esta práctica es documentar un servicio mediante el RAML, sin necesidad de crear un servicio web.

Posteriormente, crearemos el servicio web basándonos en el documento RAML en NodeJS con Express, es decir, una API Rest.

La base de datos será prácticamente la misma de la práctica anterior. Para conectarnos a ella emplearemos la librería *mysql* de Node.

Finalmente, para poder comprobar visual y fácilmente dichos servicios, crearé un cliente web en PHP empleando el popular framework Laravel.

Requisitos del sistema

- ❏ Node.js 8.9.4
- ❏ PHP 7
- ❏ Composer
- ❏ XAMPP 7.1.14

Archivos de la práctica

Junto a esta memoria, adjunto los contratos RAML y sus correspondientes HTML en la carpeta *'documentos-raml'*, el servicio web en NodeJS *'api-rest-en-nodejs'*, el proyecto cliente en Laravel llamado *'cliente-web-laravel'* y el archivo SQL con la base de datos inicializada, *schema.sql*.

La ubicación del servicio web y del proyecto del cliente web es indiferente, pero es importante abrir una terminal en sus ubicaciones para poder poner sus correspondientes servidores en marcha.

Instalación del software necesario

Para el correcto funcionamiento de la práctica, resulta indiferente el SO empleado.

1. **Descargar e instalar XAMPP 7.1.14:** Para ello,dentro de su página oficial, podemos obtener links a sus instaladores para diferentes SOs:
    a. Windows
    b. Linux
    c. OS X

    Con la instalación de XAMPP, se nos instalará PHP 7, así que no hay que preocuparse por ello.

2. **Descargar e instalar Composer**: En su web oficial nos ofrecen una amplia variedad de formas de instalarlo. Para Windows, sugiero el instalador que nos proporcionan: Composer-Setup.exe. El cuál, además de instalarnos composer, nos configura la variable de entorno *composer* desde cualquier directorio.

3. **Instalar Laravel:** Una vez instalado Composer, podemos instalar Laravel fácilmente desde una terminal ejecutando:

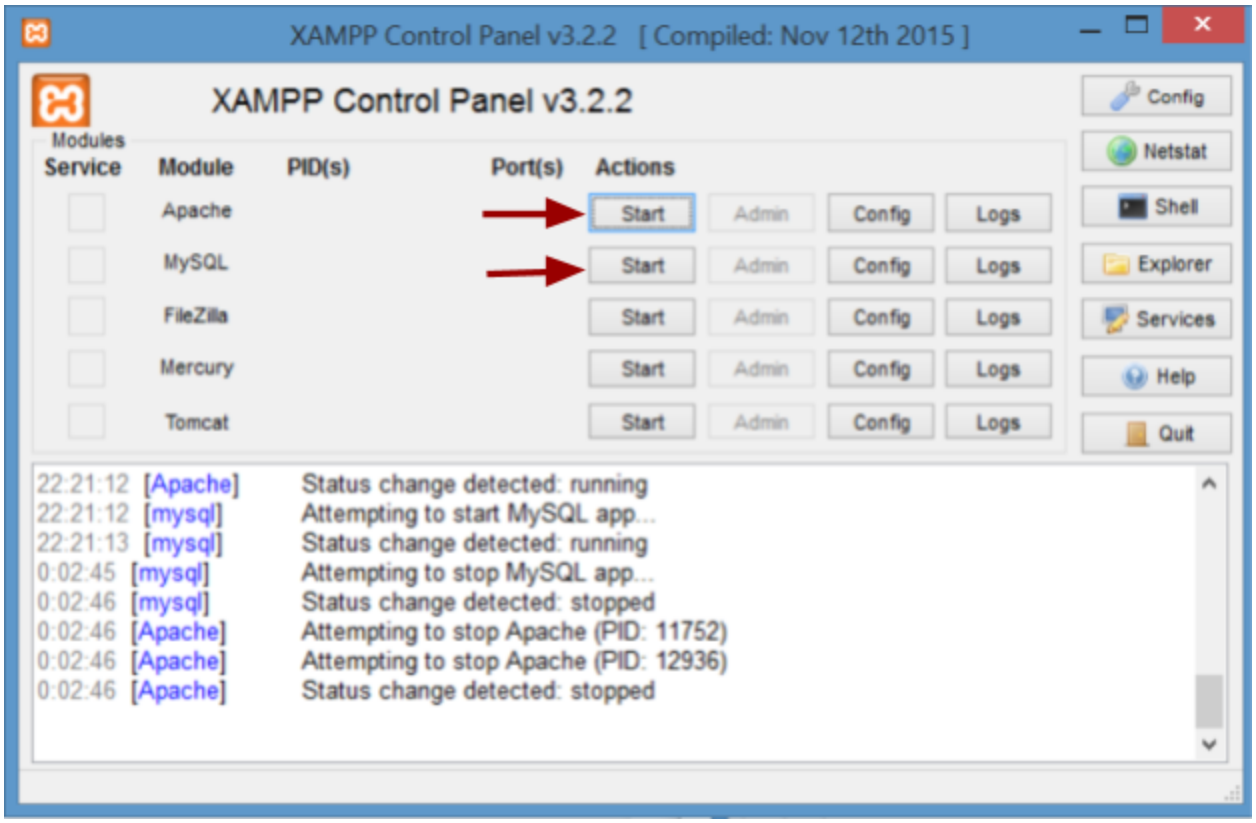
```
composer global require "laravel/installer"
```

4. **Instalar NodeJS:** En su web oficial nos proporcionan una descarga directa y personalizada para nuestro sistema operativo. En la cuál, de las 2 opciones que nos proporcionan para descargar, recomiendo la versión *8.9.4 LTS - Recomendado para la mayoría*.

Puesta en marcha

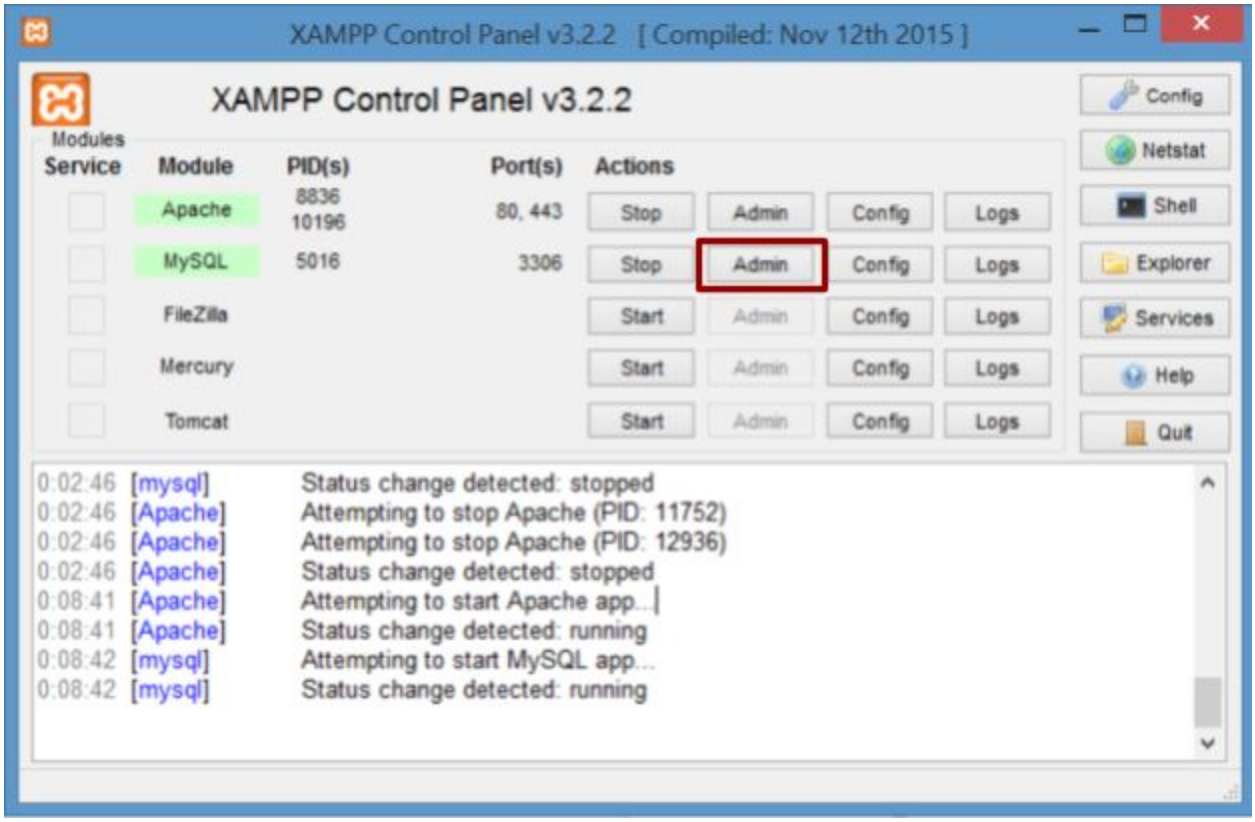
1. Iniciar servidores Apache y MySql en XAMPP

   Una vez abierto XAMPP, iniciamos los servidores Apache y MySQL, pulsamos en Start:

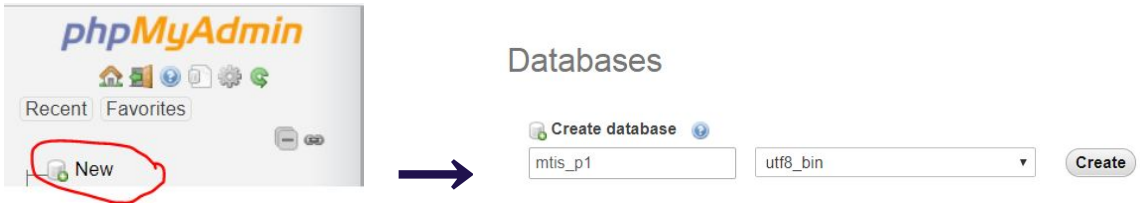


2. Crear base de datos MySql

Crearemos la base de datos con ayuda de phpMyAdmin. Para ello, en XAMPP, pulsaremos sobre el botón de Admin habilitado al iniciar el servidor MySql:

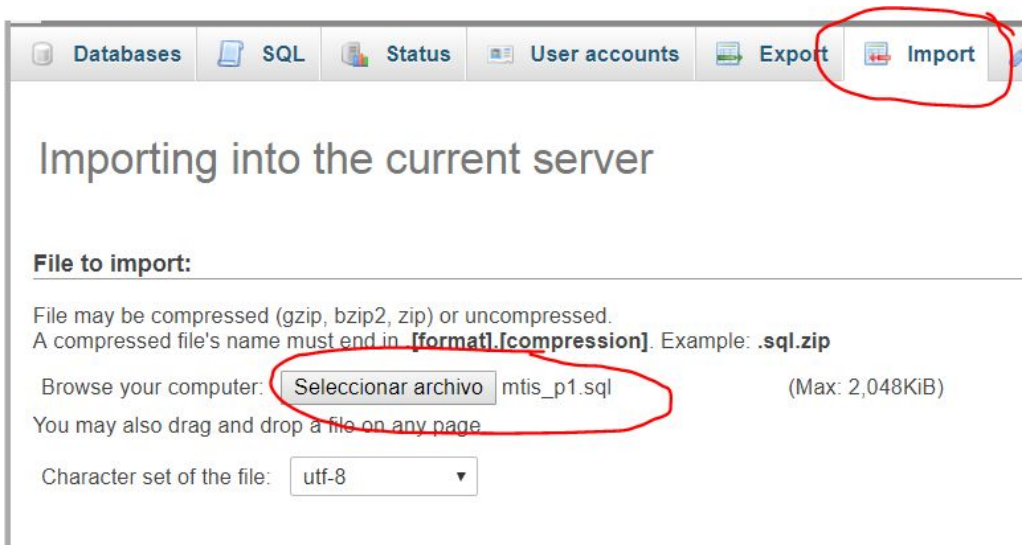


Ahora, en el panel de phpMyAdmin, crearemos una nueva base de datos con el nombre *'mtis_p1'* y codificación *utf8_bin*:

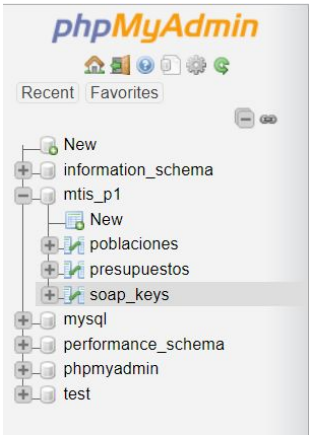


Para inicializar la Base de Datos emplearemos el archivo schema.sql adjuntado, con la base de datos exportada.

En el panel de phpMyAdmin, dentro de la pestaña de Import, podemos seleccionar nuestro archivo schema.sql, y la codificación utf-8, e importarlo, pulsando (más abajo) sobre el botón Go:



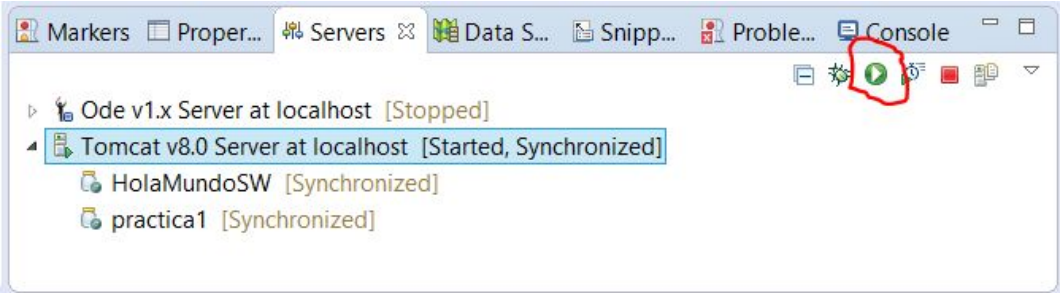
Finalmente, obtendremos el siguiente resultado con nuestras tablas creadas y con datos:



P.D.: En la tabla 'soap_keys', se pueden consultar cuáles existen actualmente.

3. Iniciar servidor Tomcat

Para iniciar el servidor Tomcat con el servicio web, necesitamos abrir el proyecto *'practica1'* con Eclipse. Una vez abierto, en el menú de abajo, en la pestaña Servers, seleccionamos Tomcat y pulsamos sobre el botón de "play":



4. Iniciar servidor Laravel

A la hora de iniciar el servidor en Laravel con el cliente, necesitamos abrir una terminal en la ruta donde se halla el proyecto *cliente-web-laravel* adjuntado en la práctica. En él ejecutamos (sólo la primera vez):

```
composer update
```

Y a continuación, iniciamos el servidor con:

```
php artisan serve
```

5. Descargar dependencias e iniciar API Rest en Node

Para descargar las dependencias, necesarias para iniciar la API Rest en Node, necesitamos abrir una terminal en la ruta donde se halla el proyecto *api-rest-en-node* adjuntado en la práctica. En él ejecutamos (sólo la primera vez):

```
npm install
```

Y a continuación, iniciamos el servidor con:

```
node app.js
```

6. Abrir URL del cliente en el Navegador

Finalmente, podemos comprobar el funcionamiento de nuestra práctica desde la URL: localhost:8000

Pasos seguidos para las elaboración de la práctica

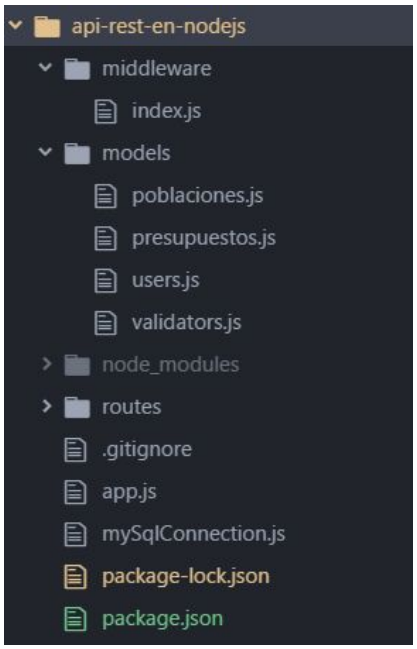
El primer paso ha sido definir en los 3 contratos RAML las cuatro operaciones especificadas en el enunciado de la práctica con sus entradas y salidas: *validarNIF, validarIBAN, consultaCódigoPostal y generaPresupuesto*.

```
#%RAML 1.0
title: MTIS - Practica 2 - Documento 1
mediaType: application/json
baseUri: http://localhost:3000
version: 1.0

types:
/validarNIF:
  post:
    description: Comprueba, según el sistema español, la validez de un NIF
    body:
      application/json:
        type: object
        properties:
          RestKey: RestKey
          nif: NIF
    responses:
      200:
        body:
          application/json:
            type: object
            properties:
              isValid: boolean
            example:
              "isValid": true
      500:
        body:
          application/json:
            type: object
            properties:
              isValid: boolean
            example:
              "isValid": false
```

Ejemplo de definición de tipos en la operación generaPresupuesto:

```
types:
  Presupuesto:
    type: object
    properties:
      RestKey: string
      fechaPresupuesto: date-only
      idCliente: number
      referenciaProducto: string
      cantidadProducto: number
```

Seguidamente, he creado el servicio web en Express con NodeJS, con cierta estructura lógica de carpetas y clases:



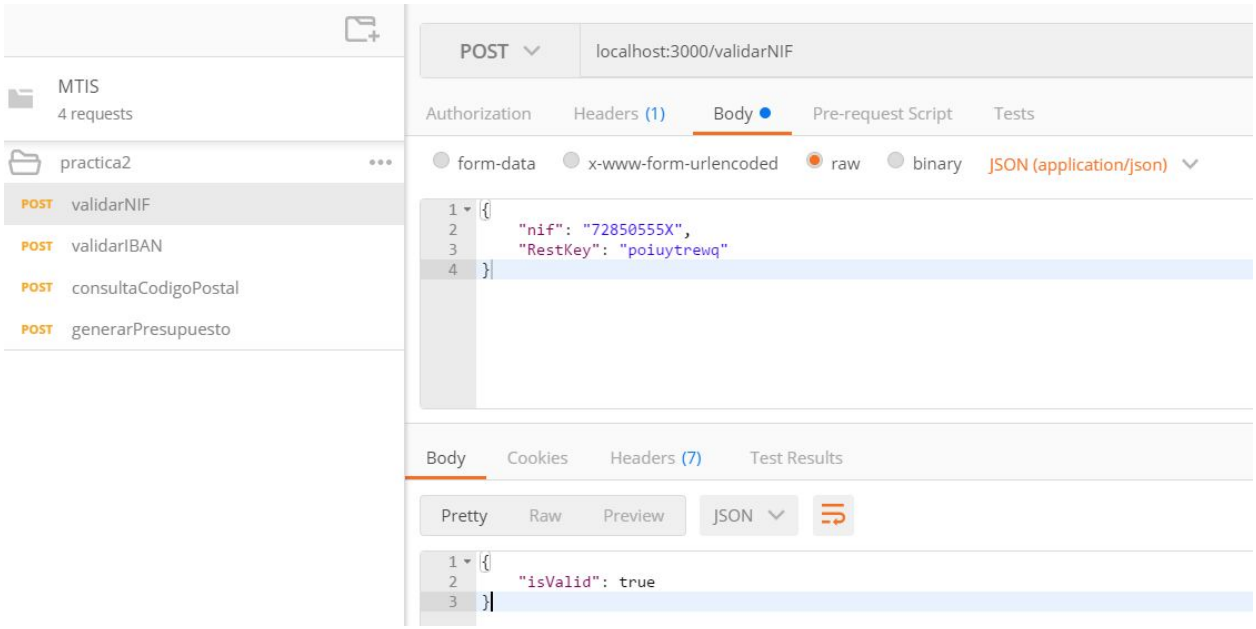
En los archivos del directorio */models*, se encuentra la lógica de cada operación. Por ejemplo, en el caso de */validarNIF*:

```
var validatorModel = {};

validatorModel.checkNif = function(nif, callback)
{
  if(nif) {
    var lockup = 'TRWAGMYFPDXBNJZSQVHLCKE';
    var valueDni = nif.substr(0, nif.length-1);
    var letra = nif.substr(nif.length-1, 1).toUpperCase();

    var isValid = (lockup.charAt(valueDni % 23) === letra)

    callback(null, { isValid: isValid });
  } else {
    var error = "Field \'nif\' is required"
    throw error;
  }
}
```

Y en */routes/index.js*, se halla la respuesta para cada endpoint. Por ejemplo, */validarNIF*:

```
// Obtiene un NIF y devuelve si es válido
app.post("/validarNIF", function(req,res,next)
{
  var nif = req.body.nif

  ValidatorsModel.checkNif(nif, function(error, data)
  {
    if(data)
    {
      res.send({isValid: data.isValid})
      res.status(200)
      res.end()
      next()
    }
    else
    {
      res.status(500)
      res.end()
      next()
    }
  });
});
```

A continuación, con ayuda de Postman, se comprueba el correcto funcionamiento de cada operación (previamente a esto, se crea e inicializa la base de datos):



Finalmente, cree el proyecto cliente web con Laravel:

```
laravel new cliente-web-laravel
```

Me serví de la librería de Guzzle para Laravel: Guzzle\Client, realiza la conexión del cliente con el servicio web cada vez que se realiza una operación:

```php
$client = new Client(['base_uri' => $this->base_uri]);

try {
  $response = $client->request('POST', 'generarPresupuesto', [
      'json' => [
          'RestKey' => $restKey,
          'fechaPresupuesto' => $fechaPre,
          'idCliente' => $idCliente,
          'referenciaProducto' => $referenciaProd,
          'cantidadProducto' => $cantidadProd
      ]
  ]);
```

Lo cuál me devolvía un objeto $response, que en el caso de la validación del NIF, para obtener el elemento respuesta *isValid,* devuelto por HTTP. Para acceder a dicho elemento, previamente necesitamos descodificar a un objeto JSON:

```php
$body = $response->getBody();
$json = json_decode($body);

$nifValid = $json->isValid;
```

Para la conexión con la base de datos MySql, me serví de una clase, *mySqlConnection.js*, en la cual configuramos los datos de la BD y la exportamos para poderla emplear en los distintos endpoints de la aPI Rest.

mySqlConnection.js

```js
//llamamos al paquete mysql que hemos instalado
var mysql = require('mysql'),
//creamos la conexion a nuestra base de datos con los datos de acceso de cada uno
connection = mysql.createConnection(
  {
    host: 'localhost',
    user: 'arancha',
    password: '12345',
    database: 'mtis_p2'
  }
);

module.exports = connection;
```