PHP Servicios API Rest

UNIDAD 6: Servicios Web

DWES – Desarrollo Web Entorno Servidor

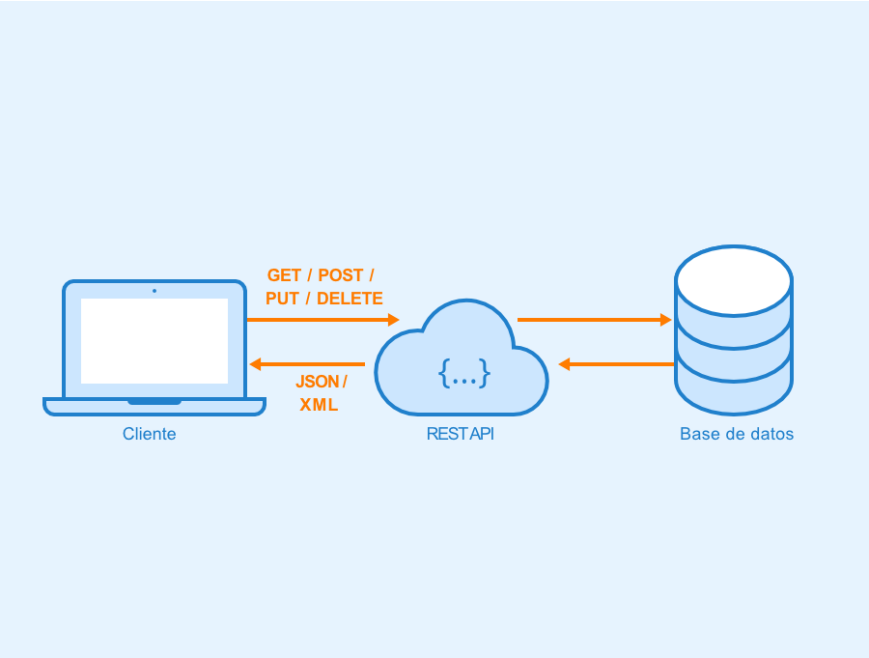
Roberto Rodríguez Ortiz

Tabla de contenidos

1. API Rest

En primer lugar, tenemos a una **API** (Interfaz de Programación de Aplicación o Application Programming Interface en inglés). Básicamente, es un tipo de software que permite a dos aplicaciones comunicarse una con otra, a través de Internet y otros dispositivos. Por ejemplo, cada vez que alguien ingresa en una app como Facebook o revisa el clima en un smartphone, se utiliza una API.

REST significa **Re**presentational **S**tate **T**ransfer y es un conjunto de principios para crear servicios web que sean escalables y fáciles de mantener. Una API REST utiliza métodos HTTP (GET, POST, PUT, DELETE, etc.) para comunicarse.



Los 4 métodos o peticiones que, usualmente, son suficientes para cubrir la mayoría de sus usos:

- **GET:** se utiliza para consultar los recursos en la API RESTful.
- **POST:** se emplea para actualizar o cambiar el estado de un recurso.
- **PUT:** se usa para crear nuevos recursos o reemplazar el contenido existente.
- **DELETE:** elimina un recurso.

1.1. Diferencias entre SOAP y REST

Es posible que muchos sistemas heredados sigan rigiéndose por SOAP, aunque REST haya surgido más tarde y se considere una alternativa más rápida en los escenarios basados en la Web. REST es un conjunto de pautas que ofrece una implementación flexible, mientras que SOAP es un protocolo con requisitos específicos, como en el caso de la mensajería XML.

Las API de REST son ligeras, así que son ideales para los contextos más nuevos, como el Internet de las cosas (IoT), el desarrollo de aplicaciones móviles y la informática sin servidor. Los servicios web de SOAP ofrecen seguridad y cumplimiento de las operaciones integrados que coinciden con muchas de las necesidades empresariales, pero que también los hacen más pesados. Asimismo, muchas API públicas, como la API de Google Maps, siguen las pautas de REST.

1.2. URLs amigables

Las URL amigables son básicamente direcciones de páginas que son más fáciles de escribir, recordar y, sobre todo, que Google interpreta de con mayor relevancia.

Por ejemplo, podríamos tener una URL como:

example.com/tienda.php?productos=zapatillas&categoria=playa

pero es una URL poco atractiva para buscadores y usuarios, aparte que muestra nuestra programación.

Sería mucho mejor una URL como esta:

example.com/productos/zapatillas/playa.

Es una dirección más concisa, fácil de escribir y que se centra en lo que realmente importa, sin mostrar el contenido de las variables que se usan a nivel de programación.

Para ello utilizaremos el archivo **.htaccess**. Primero tenemos que crear ese archivo y guardarlo generalmente en la carpeta raíz del dominio. El .htaccess es un archivo de texto plano, que tendremos que editar.

Para comenzar debemos indicar que se ponga en marcha el motor de reescritura de URL con esta línea:

```
RewriteEngine on
```

Luego tenemos que generar las redirecciones con la instrucción **RewriteRule**, indicando primero el patrón de la URL amigable y la redirección que se debe producir. Dicho patrón se debe colocar como una expresión regular y a continuación colocamos la URL a la que se debe redirigir la solicitud.

RewriteEngine On

RewriteCond %{REQUEST_FILENAME} !-f

RewriteRule ^(.)$ %{ENV:BASE}index.php [QSA,L]*

1.3. Formato JSON

La transmisión de datos se realiza preferentemente en formato JSON (ganando en popularidad a XML). JSON (JavaScript Object Notation) es un formato de texto pensado para el intercambio de datos. Su sintaxis está basada originalmente en la sintaxis de JavaScript, pero realmente es independiente de cualquier lenguaje de programación.

Las reglas sintácticas de JSON son bastante sencillas:

- En JSON existen dos tipos de elementos
  - **matrices (*arrays*)**: Las matrices son listas de valores separados por comas. Las matrices se escriben entre corchetes [ ]

    ```
    [1, "pepe", 3.14, "Pepito Conejo"]
    ```

  - **objetos (*objects*):** Los objetos son listas de parejas nombre / valor. El nombre y el valor están separados por **dos puntos :** y las parejas están separadas por comas. Los objetos se escriben entre llaves { } y los nombres de las parejas se escriben siempre entre comillas dobles.

    ```
    {"nombre": "Pepito Conejo", "edad": 25, "carnet de conducir": true}
    ```

- Tanto en los objetos como en las matrices, el último elemento no puede ir seguido de una coma.

  ```
  {"nombre": "Pepito Conejo",}
  ["nombre": "Pepito Conejo", "edad": 25, "carnet de conducir": true, ]
  ```

- Los **espacios en blanco y los saltos de línea** no son significativos, es decir, puede haber cualquier número de espacios en blanco o saltos de línea separando cualquier elemento o símbolo del documento.

  ```
  [
   {
    "nombre": "Pepito Conejo",
    "edad": 25,
    "carnet de conducir": true
   },
   {
  ```

```
    "nombre": "Ana Barberá",
    "edad": 90,
    "carnet de conducir": false
  }
]
```

- Los valores (tanto en los objetos como en las matrices) pueden ser:
  - **números:** enteros, decimales o en notación exponencial. El separador decimal es el punto ., un número negativo empieza por el signo menos - .
  - **Cadenas:** Las cadenas se escriben entre comillas dobles.
- Los ficheros JSON no pueden contener comentarios.

En casi todos los lenguajes existen instrucciones o librerías para tratar con cadenas Json, en php tenemos:

- Json_decode
- Json_encode

Por ejemplo en el proyecto que veremos más adelante, se consulta una base de datos de usuario, para componer la respuesta se convierte el resultado de una consulta en una cadena Json con json_encode:

```
$result = $this->personGateway->findAll();
$response['status_code_header'] = 'HTTP/1.1 200 OK';
$response['body'] = json_encode($result);
```

De la misma forma los datos que recibimos podemos convertirlos en una array asociativo:

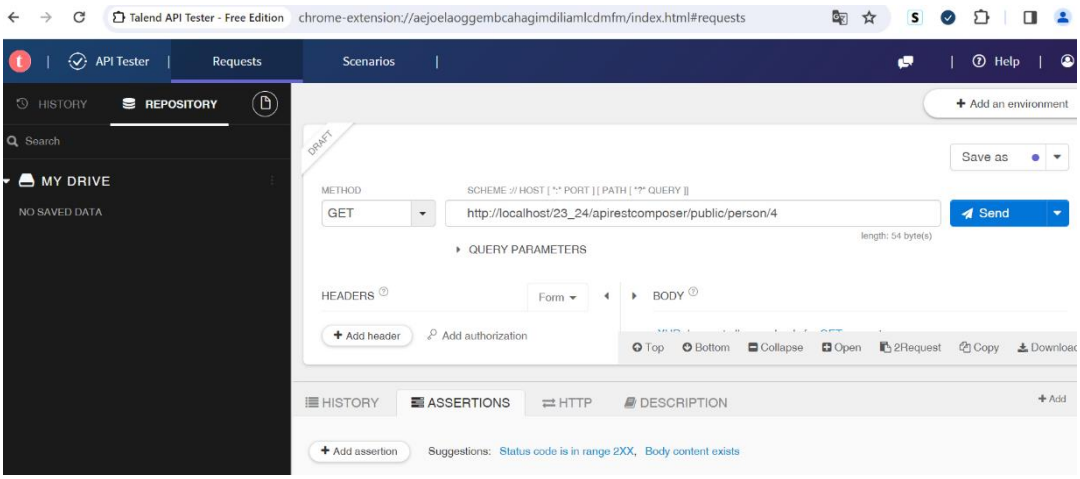
```
$input = (array) json_decode(file_get_contents('php://input'), TRUE);
```

2. Consumir el API Rest

Para comprobar que el servidor está respondiendo a las peticiones Rest que realizamos desde una página web, es conveniente tener un **cliente** aparte de la web que desarrollemos para probar la peticiones y ver que devuelve el servidor a cada una de las mismas. Existen muchas alternativas:

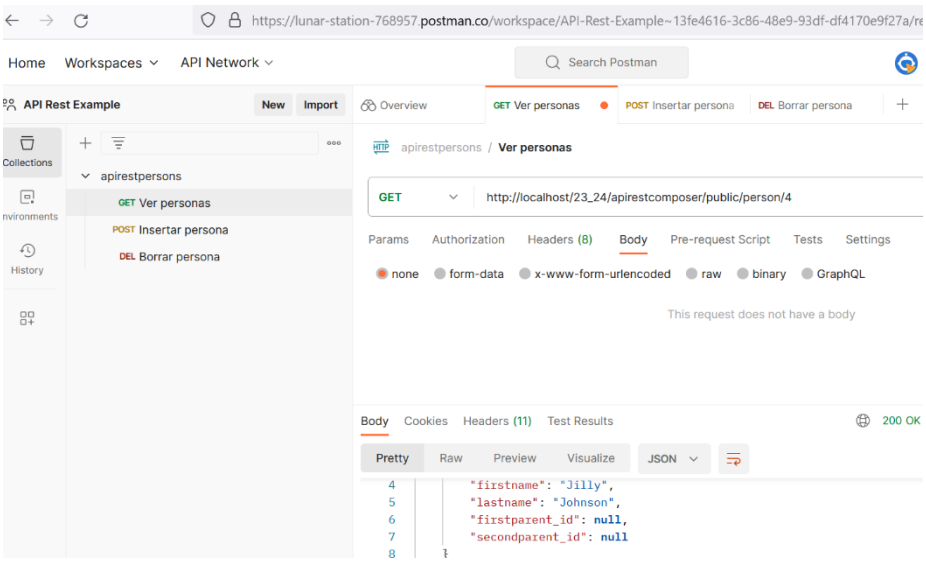
Extensión de Chrome

- **Talend** API Tester: https://chromewebstore.google.com/detail/talend-api-tester-free-ed/aejoelaoggembcahagimdiliamlcdmfm



Prueba con software

- **Postman** https://www.postman.com/ (necesario instalar Postman Agent)



2.1. Códigos de respuesta

La función *header()* envía los **headers HTTP** y asegura que estén formateados adecuadamente. Los headers han de ser lo primero en el response, no se ha de escribir nada antes de un header. A veces el HTTP server puede estar configurado para incluir otros headers además de los que se puedan especificar en el código.

Los headers contienen todo tipo de metadatos, como la codificación de texto utilizada en el mensaje del body, o el MIME type del contenido del body. En este caso estamos especificando explícitamente los códigos de respuesta HTTP. Los códigos de respuesta estandarizan una forma de informar al cliente acerca del resultado de su ***request***. Por defecto, PHP devuelve un código de respuesta 200, que significa que ha sido satisfactoria.

El servidor debe enviar el **código de respuesta HTTP más apropiado**. De esta forma el cliente puede intentar reparar sus errores, suponiendo que haya alguno.

El significado de un código de respuesta HTTP no es muy preciso, por eso HTTP es considerado algo genérico, pero se debe intentar usar el código de respuesta que más concuerde con la situación.

Los códigos de respuesta más comúnmente utilizados con REST son:

- **200 OK**. Satisfactoria.
- **201 Created**. Un *resource* se ha creado. Respuesta satisfactoria a un *request* POST o PUT.
- **400 Bad Request**. El *request* tiene algún error, por ejemplo cuando los datos proporcionados en POST o PUT no pasan la validación.
- **401 Unauthorized**. Es necesario identificarse primero.
- **404 Not Found**. Esta respuesta indica que el ***resource*** requerido no se puede encontrar (La URL no se corresponde con un *resource*).
- **405 Method Not Allowed**. El **método HTTP** utilizado no es soportado por este *resource*.
- **409 Conflict**. Conflicto, por ejemplo cuando se usa un PUT request para crear el mismo resource dos veces.
- **500 Internal Server Error**. Un error 500 suele ser un error inesperado en el servidor.

2.2. Procesar Requests

Existen dos aspectos fundamentales para procesar los requests de forma **REST**. El primero es iniciar un proceso diferente en función del **método HTTP**, incluso cuando las URLs son las mismas. En **PHP** está el array global **$_SERVER**, que determina que método se ha utilizado para el request.

```
$_SERVER['REQUEST_METHOD']
```

Esta variable contiene el nombre del método en un string, ya sea GET, PUT...

La otra clave es saber que URL se ha solicitado. Para ello se utiliza otra variable:

```
$_SERVER['REQUEST_URI']
```

Esta variable contiene la URL desde la primera /. Si el host es 'ejemplo.com', 'http://ejemplo.com/' devolverá '/', y 'http://ejemplo.com/prueba/' devolverá '/prueba/'.

2.3. API Rest en otros lenguajes

En nuestro caso vamos a realizar el API Rest sin utilizar ninguna librería, aunque existen muchas alternativas, por ejemplo:

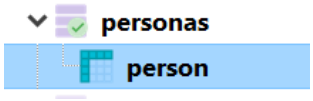
- **Slim Framework:** https://www.slimframework.com/
- **Retropie:** https://github.com/tebru/retrofit-php

también se puede realizar en Java con **Spring Boot**, también en C# , y existen herramientas como **Swagger** , https://swagger.io/ , para documentar los puntos de entrada y como consumir cada servicio, cada una con una estructura un tanto diferente, vamos a entender primero los conceptos fundamentales que nos permitirán utilizar estas soluciones de forma informada.

2.4. Ejemplo API Rest utilizando composer

Ejemplo Personas:

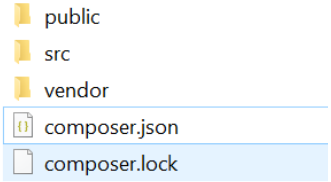
Supongamos una bbdd personas con una tabla person, que almacen el id, nombre y apellido (ignoramos el resto de campos).



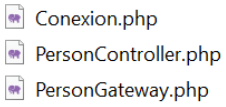
id	firstname	lastname	firstparent_id	secondparent_id
5	Jack	Davis	(NULL)	(NULL)
6	Jessica	Wilson	(NULL)	(NULL)
7	Joseph	Martinez	(NULL)	(NULL)
8	Julia	Garcia	(NULL)	(NULL)
9	Jordan	Miller	(NULL)	(NULL)
10	Jasmine	Taylor	(NULL)	(NULL)

Vamos a realizar un **API Rest** que realice un **CRUD** de dicha tabla, es decir, que nos permita realizar consultas, altas, bajas y modificaciones.

Vamos a seguir utilizando composer y su estructura de directorios:



- **Public :** contiene la vista pública index.php
- **Src :** contiene el fichero de conexión para la base de datos, el controller para… y el Gateway para…



- **Vendor :** librerías y dependencias.

Compose.json

No vamos a incluir librerías externas (require estará vacio)

```
{
    "name": "usuario/practica",
    "description": "Ejemplo API Rest",
    "type": "project",
    "require": {

    },
    "config": {
        "optimize-autoloader": true
    },
    "autoload": {
        "psr-4": {
            "Clases\\": "src"
        }
    },
   "config": {
      "platform": {
         "php": "7.4.32"
      }
   },
    "license": "GNU/GPL",
    "authors": [
        {
            "name": "usuario",
            "email": "usuario@correo.es"
        }
    ]
}
```

index.php

Punto de entrada a la aplicación.

```
<?php
require '../vendor/autoload.php';

use Clases\PersonController;

header("Access-Control-Allow-Origin: *");
header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers,
Authorization, X-Requested-With");
header("Access-Control-Allow-Methods: OPTIONS, GET, POST, PUT, DELETE");
header("Content-Type: application/json; charset=UTF-8");
header("Access-Control-Max-Age: 3600");

// Manejar la petición de verificación preliminar de CORS.
if ($_SERVER['REQUEST_METHOD'] == 'OPTIONS') {
    // No hacer nada más y finalizar el script después de enviar los encabezados de
CORS.
    exit(0);
}


//eliminamos la parte del path que no corresponde a los puntos de entrada del API
$BASE_URI = "/23_24/apirestcomposer/public";
$parsedURI = parse_url($_SERVER["REQUEST_URI"]);
$endpointName = str_replace($BASE_URI, "", $parsedURI["path"]);
```

```php
$uri = explode( '/', $endpointName);
// los endpoints comienzan con /person todo lo demás es un 404 not found
if ($uri[1] !== 'person') {
    header("HTTP/1.1 404 Not Found");
    exit();
}

// el user id es opcional y debe ser un número
$userId = null;
if (isset($uri[2])) {
    $userId = (int) $uri[2];
}

$requestMethod = $_SERVER["REQUEST_METHOD"];

//pasamos la petición y el userid al personController para que procese la petición
$controller = new PersonController($requestMethod, $userId);
$controller->processRequest();
```

Conexión.php

El fichero de conexión con la bbdd ya utilizado anteriormente.

```php
<?php

namespace Clases;

use PDO;
use PDOException;

class Conexion
{
    private $host;
    private $db;
    private $user;
    private $pass;
    private $dsn;
    protected $conexion;

    public function __construct()
    {
        $this->host = "localhost";
        $this->db = "personas";
        $this->user = "gestor";
        $this->pass = "secreto";
        $this->dsn = "mysql:host={$this->host};dbname={$this->db};charset=utf8mb4";
        $this->crearConexion();
    }

    public function crearConexion()
    {
        try {
            $this->conexion = new PDO($this->dsn, $this->user, $this->pass);
            $this->conexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        } catch (PDOException $ex) {
            die("Error en la conexión: mensaje: " . $ex->getMessage());
```

```
        }
        return $this->conexion;
    }
}
```

PersonController.php

Clase encargada de **recibir las peticiones** a través del método ProcessRequest, se comprueba tipo de petición y se dará respuesta utilizando la clase PersonGateway.

```php
<?php
namespace Clases;

use Clases\PersonGateway;

class PersonController {

    private $requestMethod;
    private $userId;
    private $personGateway;

    public function __construct($requestMethod, $userId)
    {
        $this->requestMethod = $requestMethod;
        $this->userId = $userId;
        $this->personGateway = new PersonGateway();
    }

    public function processRequest()
    {
        switch ($this->requestMethod) {
            case 'GET':
                if ($this->userId) {
                    $response = $this->getUser($this->userId);
                } else {
                    $response = $this->getAllUsers();
                };
                break;
            case 'POST':
                $response = $this->createUserFromRequest();
                break;
            case 'PUT':
                $response = $this->updateUserFromRequest($this->userId);
                break;
            case 'DELETE':
                $response = $this->deleteUser($this->userId);
                break;
            case 'OPTIONS':
                $response['status_code_header'] = 'HTTP/1.1 200 OK';
            default:
                $response = $this->notFoundResponse();
                break;
        }
        header($response['status_code_header']);
        if ($response['body']) {
            echo $response['body'];
        }
    }

    private function getAllUsers()
    {
        $result = $this->personGateway->findAll();
        $response['status_code_header'] = 'HTTP/1.1 200 OK';
        $response['body'] = json_encode($result);
```

```
        return $response;
    }

    private function getUser($id)
    {
        $result = $this->personGateway->find($id);
        if (! $result) {
            return $this->notFoundResponse();
        }
        $response['status_code_header'] = 'HTTP/1.1 200 OK';
        $response['body'] = json_encode($result);
        return $response;
    }

    private function createUserFromRequest()
    {
        $input = (array) json_decode(file_get_contents('php://input'), TRUE);
        if (! $this->validatePerson($input)) {
            return $this->unprocessableEntityResponse();
        }
        $this->personGateway->insert($input);
        $response['status_code_header'] = 'HTTP/1.1 201 Created';
        $response['body'] = "Usuario creado con éxito";
        return $response;
    }

    private function updateUserFromRequest($id)
    {
        $result = $this->personGateway->find($id);
        if (! $result) {
            return $this->notFoundResponse();
        }
        //coge los datos JSON enviados con la petición POST,
        // los lee como un string crudo, los decodifica a una estructura de
        // array asociativa en PHP y luego asigna esta estructura a la variable
$input
        $input = (array) json_decode(file_get_contents('php://input'), TRUE);
        if (! $this->validatePerson($input)) {
            return $this->unprocessableEntityResponse();
        }
        $this->personGateway->update($id, $input);
        $response['status_code_header'] = 'HTTP/1.1 200 OK';
        $response['body'] = null;
        return $response;
    }

    private function deleteUser($id)
    {
        $result = $this->personGateway->find($id);
        if (! $result) {
            return $this->notFoundResponse();
        }
        $this->personGateway->delete($id);
        $response['status_code_header'] = 'HTTP/1.1 200 OK';
        $response['body'] = null;
        return $response;
    }

    private function validatePerson($input)
    {
        if (! isset($input['firstname'])) {
            return false;
        }
        if (! isset($input['lastname'])) {
            return false;
        }
```

```
        return true;
    }

    private function unprocessableEntityResponse()
    {
        $response['status_code_header'] = 'HTTP/1.1 422 Unprocessable Entity';
        $response['body'] = json_encode([
            'error' => 'Invalid input'
        ]);
        return $response;
    }

    private function notFoundResponse()
    {
        $response['status_code_header'] = 'HTTP/1.1 404 Not Found';
        $response['body'] = null;
        return $response;
    }
}
```

PersonGateway.php

Esta clase contiene los métodos que **acceden a la base de datos** para realizar las operaciones con la tabla personas, consulta, inserción, actualización…

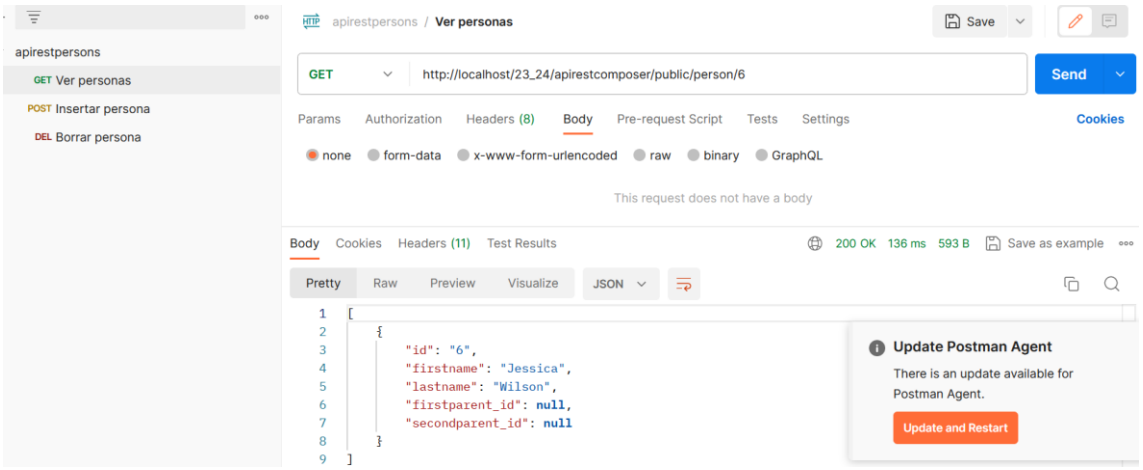
```
<?php
namespace Clases;


class PersonGateway extends conexion{

    public function __construct()
    {
         parent::__construct();
    }

    public function findAll()
    {
        $statement = "
            SELECT
                id, firstname, lastname, firstparent_id, secondparent_id
            FROM
                person;
        ";

        try {
            $statement = $this->conexion->query($statement);
            $result = $statement->fetchAll(\PDO::FETCH_ASSOC);
            return $result;
        } catch (\PDOException $e) {
            exit($e->getMessage());
        }
    }

    public function find($id)
    {
        $statement = "
            SELECT
                id, firstname, lastname, firstparent_id, secondparent_id
            FROM
                person
```

```
            WHERE id = ?;
        ";

        try {
            $statement = $this->conexion->prepare($statement);
            $statement->execute(array($id));
            $result = $statement->fetchAll(\PDO::FETCH_ASSOC);
            return $result;
        } catch (\PDOException $e) {
            exit($e->getMessage());
        }
    }

    public function insert(Array $input)
    {
        $statement = "
            INSERT INTO person
                (firstname, lastname, firstparent_id, secondparent_id)
            VALUES
                (:firstname, :lastname, :firstparent_id, :secondparent_id);
        ";

        try {
            $statement = $this->conexion->prepare($statement);
            $statement->execute(array(
                'firstname' => $input['firstname'],
                'lastname'  => $input['lastname'],
                'firstparent_id' => $input['firstparent_id'] ?? null,
                'secondparent_id' => $input['secondparent_id'] ?? null,
            ));
            return $statement->rowCount();
        } catch (\PDOException $e) {
            exit($e->getMessage());
        }
    }

    public function update($id, Array $input)
    {
        $statement = "
            UPDATE person
            SET
                firstname = :firstname,
                lastname  = :lastname,
                firstparent_id = :firstparent_id,
                secondparent_id = :secondparent_id
            WHERE id = :id;
        ";

        try {
            $statement = $this->conexion->prepare($statement);
            $statement->execute(array(
                'id' => (int) $id,
                'firstname' => $input['firstname'],
                'lastname'  => $input['lastname'],
                'firstparent_id' => $input['firstparent_id'] ?? null,
                'secondparent_id' => $input['secondparent_id'] ?? null,
            ));
            return $statement->rowCount();
        } catch (\PDOException $e) {
            exit($e->getMessage());
        }
    }

    public function delete($id)
    {
        $statement = "
```

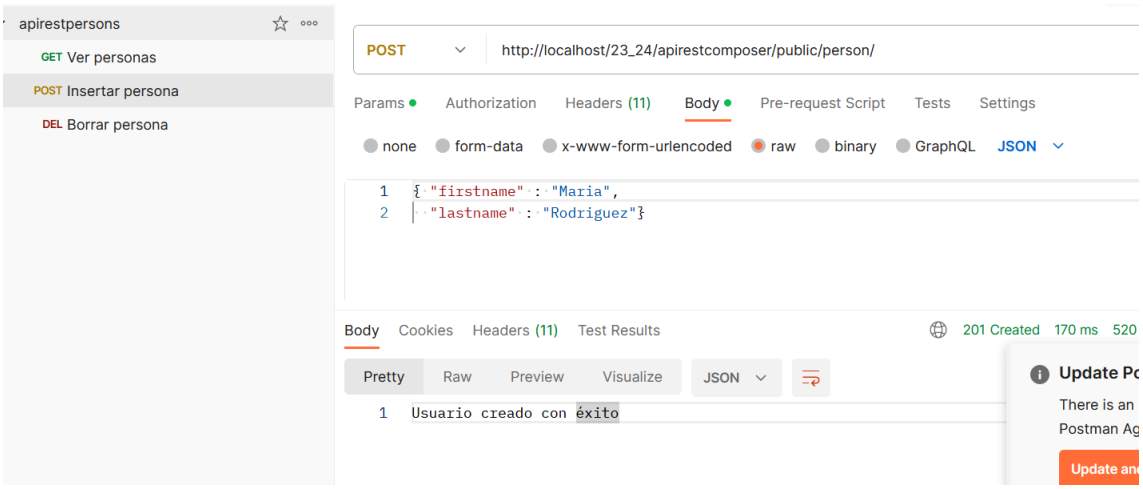
```
            DELETE FROM person
            WHERE id = :id;
        ";

        try {
            $statement = $this->conexion->prepare($statement);
            $statement->execute(array('id' => $id));
            return $statement->rowCount();
        } catch (\PDOException $e) {
            exit($e->getMessage());
        }
    }
}
```

Pruebas

Utilizando Postman comprueba que el servicio responde a una consulta, en este caso un Get donde le paso el userid=6:



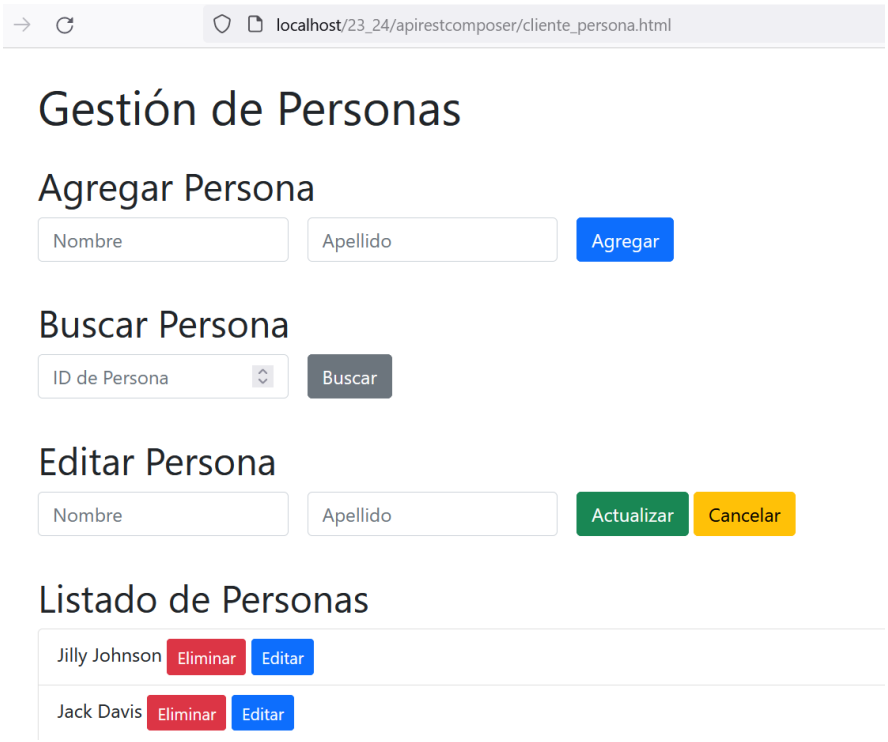
Otro ejemplo es la inserción de una persona utilizando POST y pasando el Json con los datos en el Body (raw):



Cliente

Vamos a realizar un fichero HTML que consuma los servicios creados, utilizaremos javacript para realizar las peticiones.

Crearemos un formulario similar al siguiente, que nos permita realizar un CRUD de la tabla personas:




```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Gestión de Personas</title>
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet">
  <style>
    .container {
      margin-top: 20px;
    }
  </style>
</head>
<body>
<div class="container">
  <h1 class="my-4">Gestión de Personas</h1>

  <div class="mb-3">
    <h2>Agregar Persona</h2>
    <form id="addPersonForm" class="row g-3">
      <div class="col-auto">
        <input type="text" id="firstname" class="form-control" placeholder="Nombre"
required>
```

```
      </div>
      <div class="col-auto">
        <input type="text" id="lastname" class="form-control" placeholder="Apellido"
required>
      </div>
      <div class="col-auto">
        <button type="submit" class="btn btn-primary mb-3">Agregar</button>
      </div>
    </form>
  </div>

  <div class="mb-3">
    <h2>Buscar Persona</h2>
    <form id="searchPersonForm" class="row g-3">
      <div class="col-auto">
        <input type="number" id="userId" class="form-control" placeholder="ID de
Persona" required>
      </div>
      <div class="col-auto">
        <button type="submit" class="btn btn-secondary mb-3">Buscar</button>
      </div>
    </form>
  </div>

  <div class="mb-3" style="display:none;" id="editPersonSection">
    <h2>Editar Persona</h2>
    <form id="editPersonForm" class="row g-3">
      <input type="hidden" id="editUserId">
      <div class="col-auto">
        <input type="text" id="editFirstname" class="form-control"
placeholder="Nombre" required>
      </div>
      <div class="col-auto">
        <input type="text" id="editLastname" class="form-control"
placeholder="Apellido" required>
      </div>
      <div class="col-auto">
        <button type="submit" class="btn btn-success mb-3">Actualizar</button>
        <button type="button" class="btn btn-warning mb-3"
onclick="cancelEdit()">Cancelar</button>
      </div>
    </form>
  </div>

  <h2>Listado de Personas</h2>
  <div id="personsList" class="list-group"></div>
</div>

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"></
script>
<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>

<script>
  const apiURL = "http://localhost/23_24/apirestcomposer/public/person";

  function addPerson(event) {
    event.preventDefault();
    const firstname = document.getElementById('firstname').value;
    const lastname = document.getElementById('lastname').value;
    const data = { firstname, lastname };

    fetch(apiURL, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(data)
```

```
  })
          .then(response => {
            if (response.ok) {
              alert('Persona agregada correctamente');
              loadPersons();
            } else {
              alert('Error al agregar persona:');
            }
          })
          .catch(error => console.error('Error:', error));
  }

function editPerson(event) {
  event.preventDefault();
  const userId = document.getElementById('editUserId').value;
  const firstname = document.getElementById('editFirstname').value;
  const lastname = document.getElementById('editLastname').value;
  const data = { firstname, lastname };

  fetch(`${apiURL}/${userId}`, {
    method: 'PUT',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(data)
  })
          .then(response => {
            if (response.ok) {
              alert('Persona actualizada correctamente');
              loadPersons();
              cancelEdit();
            } else {
              alert('Error al actualizar persona:');
            }
          })
          .catch(error => console.error('Error:', error));

}

function searchPerson(event) {
  event.preventDefault();
  const userId = document.getElementById('userId').value;

  fetch(`${apiURL}/${userId}`)
          .then(response => response.json())
          .then(data => {
            const personData = `<pre>${JSON.stringify(data, null, 2)}</pre>`;
            document.getElementById('personsList').innerHTML = personData;
          })
          .catch(error => console.error('Error al buscar persona:', error));
}

function deletePerson(id) {
  if (!confirm('¿Estás seguro de que quieres eliminar esta persona?')) return;

  fetch(`${apiURL}/${id}`, {
    method: 'DELETE'
  })
          .then(response => {
            if (response.ok) {
              alert('Persona eliminada correctamente');
              loadPersons();
            } else {
              alert('Hubo un problema al eliminar la persona');
            }
          })
          .catch(error => console.error('Error:', error));
}
```

```
  function loadPersons() {
    fetch(apiURL)
            .then(response => response.json())
            .then(data => {
              const listItems = data.map(person => `
                <div class="list-group-item">
                    <span>${person.firstname} ${person.lastname}</span>
                    <button class="btn btn-danger btn-sm"
onclick="deletePerson(${person.id})">Eliminar</button>
                    <button class="btn btn-primary btn-sm"
onclick="loadPersonForEdit(${person.id})">Editar</button>
                </div>
            `).join('');
              document.getElementById('personsList').innerHTML = listItems;
            })
            .catch(error => console.error('Error:', error));
  }

  function loadPersonForEdit(id) {
    fetch(`${apiURL}/${id}`)
            .then(response => {
              if (!response.ok) {
                throw new Error(`HTTP status ${response.status}`);
              }
              return response.json();
            })
            .then(data => {
              const person = data[0];
              document.getElementById('editUserId').value = person.id || '';
              document.getElementById('editFirstname').value = person.firstname ||
'';
              document.getElementById('editLastname').value = person.lastname || '';
              document.getElementById('editPersonSection').style.display = 'block';
            })
            .catch(error => {
              console.error('Error al cargar la persona para editar:', error);
              alert('Hubo un error al cargar los datos para editar.');
            });
  }

  function cancelEdit() {
    document.getElementById('editPersonSection').style.display = 'none';
  }

  document.getElementById('addPersonForm').addEventListener('submit', addPerson);
  document.getElementById('editPersonForm').addEventListener('submit', editPerson);
  document.getElementById('searchPersonForm').addEventListener('submit',
searchPerson);

  loadPersons();
</script>
</body>
</html>
```

Otros recursos:

- https://github.com/daveh/php-rest-api
- https://github.com/denisdoblev/api_taller_1/tree/master (Curso OpenWebinars)
    - https://openwebinars.net/academia/aprende/crear-api-rest-php-sin-frameworks
- https://developer.okta.com/blog/2019/03/08/simple-rest-api-php

3. Autenticación JWT- Json Web Token

JSON Web Token o JWT es un **estándar abierto** (RFC 7519) que define una forma compacta y autónoma de transmitir información de forma segura entre partes como un **objeto JSON**. Esta información se puede verificar y confiar porque está firmada digitalmente. Los JWT se pueden firmar mediante un secreto (con el algoritmo HMAC) o un documento público.

JWT ayuda a resolver el problema de autorización de múltiples servidores al almacenar la información del usuario autorizado en el lado del cliente, no en el servidor. JWT se envía en cada solicitud HTTP y puede validarse (mediante código) de cualquier servidor utilizado por la aplicación.

Un **JSON Web Token o JWT** parece una cadena con tres partes separadas por puntos. El siguiente es un ejemplo de JWT.

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOlwvXC9leGFtcGxlLm9yZyIsImF1ZC
```

Fuente: https://www.codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/

De esta forma una vez identificados con usuario contraseña, para las siguientes peticiones solo sería necesario poseer el token JWT para identificarnos.

Vamos a crear un ejemplo partiendo del ejercicio anterior. Copia el contenido en otra carpeta y creamos una base de datos (personas_jwt), similar a la anterior añadiendo dos campos, email y password:

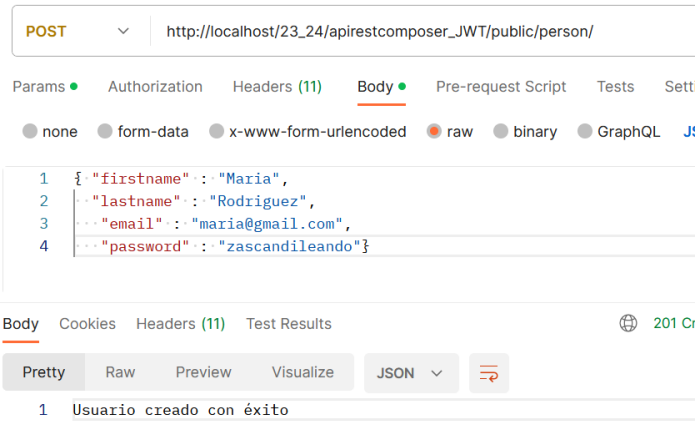
	#	Nombre	Tipo de datos	Longitud/Con...	Sin signo	Permitir NULL
🔑	1	**id**	**INT**	**10**	☐	☐
	2	firstname	VARCHAR	256	☐	☐
	3	lastname	VARCHAR	256	☐	☐
	4	email	VARCHAR	256	☐	☐
	5	password	VARCHAR	2048	☐	☐
🔑	6	firstparent_id	INT	10	☐	☑
🔑	7	secondparent_...	INT	10	☐	☑

En PersonGateway añadimos los dos campos que email y password que los deberemos recibir como parámetros en el post, la **password** la almacenaremos de forma **cifrada**:

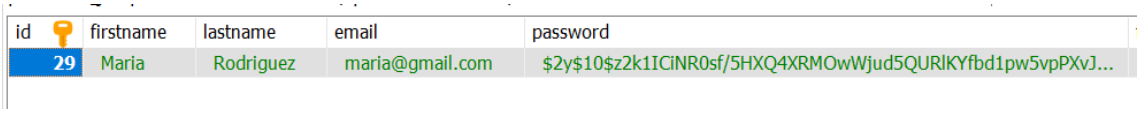
```
public function insert(Array $input)
{
    $statement = "
        INSERT INTO person
            (firstname, lastname, email, password,  firstparent_id,
secondparent_id)
        VALUES
            (:firstname, :lastname, :email, :password,
:firstparent_id, :secondparent_id);
    ";

    try {
```

```
            $stmt = $this->conexion->prepare($statement);

            // bind the values
            $stmt->bindParam(':firstname', $input['firstname']);
            $stmt->bindParam(':lastname', $input['lastname']);
            $stmt->bindParam(':email',$input['email']);
            // hash the password before saving to database
            $password_hash = password_hash($input['password'],
PASSWORD_BCRYPT);
            $stmt->bindParam(':password', $password_hash);
            $stmt->bindParam(':firstparent_id',$input['firstparent_id']);
            $stmt-
>bindParam(':secondparent_id',$input['secondparent_id']);
            // execute the query, also check if query was successful
            $stmt->execute();
            return $stmt->rowCount();

        } catch (\PDOException $e) {
            exit($e->getMessage());
        }
}
```

Asegúrate de cambiar los datos en el fichero de conexión y podemos realizar la prueba con postman:



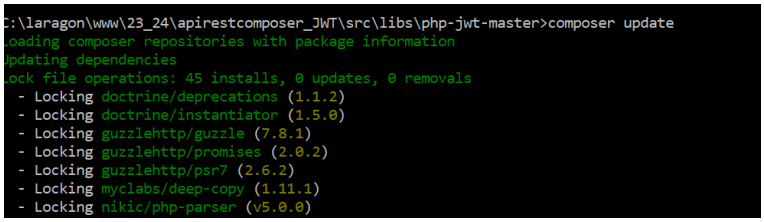
En Heidi comprobamos que se ha insertado con éxito:



id	firstname	lastname	email	password
29	Maria	Rodriguez	maria@gmail.com	$2y$10$z2k1ICiNR0sf/5HXQ4XRMOwWjud5QURlKYfbd1pw5vpPXvJ...

En el directorio src del proyecto crearemos una carpeta **libs** donde debemos descargar la siguiente **librería PHP-JWT**: https://github.com/firebase/php-jwt

Habrá que ejecutar el comando **composer update** en dicho directorio.



Otra forma más cómoda seria incorporarlo directamente a nuestro composer:

```
>composer require firebase/php-jwt

>composer update
```

Esto incorporaría la última versión a nuestro fichero composer:

```
{
    "name": "usuario/practica",
    "description": "Ejemplo API Rest",
    "type": "project",
    "require": {
        "firebase/php-jwt": "^6.10"
    },
    "config": {
        "optimize-autoloader": true
    },
    "autoload": {
        "psr-4": {
            "Clases\\": "src"
        }
    },
       "config": {
              "platform": {
                     "php": "8.1.10"
              }
       },
    "license": "GNU/GPL",
    "authors": [
        {
            "name": "usuario",
            "email": "usuario@correo.es"
        }
    ]
}
```

Recomendamos usar la **versión 8 de PHP**.

Generación de claves

Vamos a necesitar un sistema de claves pública y privada para la identificación en el servidor. Más información sobre Sistema de clave pública / clave privada o criptografía asimétrica.

Descarga de **openssl** : https://sourceforge.net/projects/openssl-for-windows/

Descomprimir y ejecutar:

```
>openssl genrsa -out id_rsa_jwt.pem 2048

>openssl rsa -in id_rsa_jwt.pem -pubout > id_rsa_jwt.pub
```

Esto nos creará los ficheros

`id_rsa_jwt.pem` -> **clave privada**

`id_rsa_jwt.pub` -> **clave pública**

Vamos a crear un fichero para realizar el login en el directorio public:

Login.php

Vamos a recibir un usuario y contraseña y generar el token JWT.

```
<?php
    namespace Clases;

    use \Firebase\JWT\JWT;

    class Login extends conexion
    {

        var $key = "-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAv9f2Z8OfTswxG+V/rMiylSOWqXX0H0Im/6prrWp0UGgtW5hL
……..
-----END RSA PRIVATE KEY-----";
        var $issued_at ;
        var $expiration_time ;
        var $issuer = "http://localhost/CodeOfaNinja/RestApiAuthLevel1/";

        var $id;
        var $firstname;
        var $lastname;
        var $email;
        var $password;

        public function __construct($requestMethod)
        {
            parent::__construct();
            $this->requestMethod = $requestMethod;
            $this->issued_at = time();
            $this->expiration_time = $this->issued_at + (60 * 60); //valid for 1 hour
        }

        public function processRequest()
        {
            switch ($this->requestMethod) {
                case 'GET':
                   break;
                case 'POST':
                    $response = $this->login();
                    break;

                case 'OPTIONS':
                    $response['status_code_header'] = 'HTTP/1.1 200 OK';
                default:
                    $response = $this->notFoundResponse();
                    break;
            }

        }

        public function emailExists($email)
        {
            // query to check if email exists
            $query = "SELECT id, firstname, lastname,email, password FROM person " .
                " WHERE email = ? LIMIT 0,1";
            // prepare the query
            $stmt = $this->conexion->prepare($query);
            // sanitize
            $email = htmlspecialchars(strip_tags($email));
            // bind given email value
            $stmt->bindParam(1, $email);
```

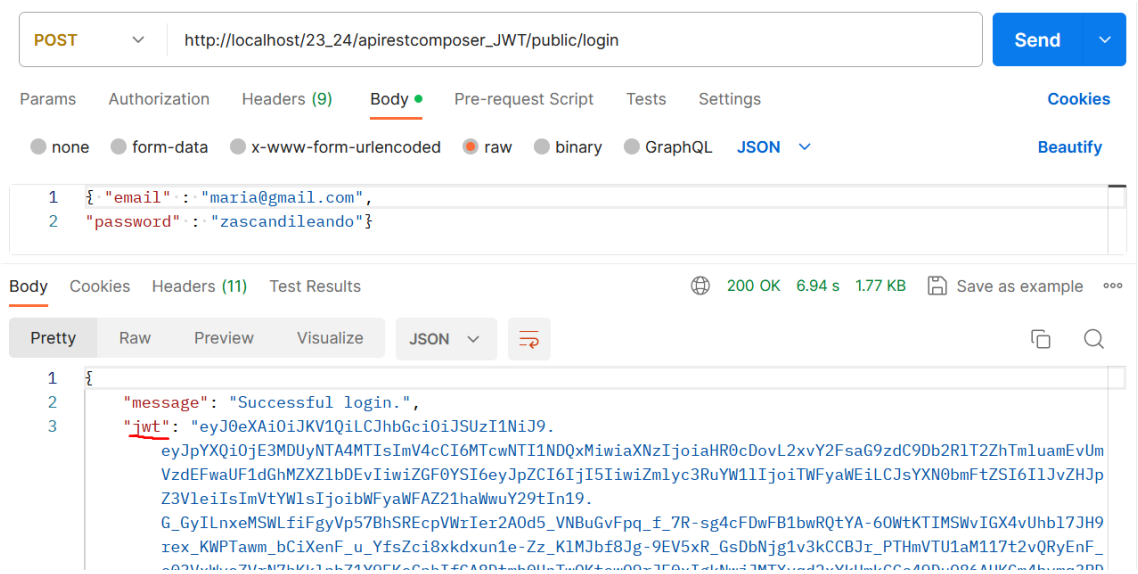
```
            // execute the query
            $stmt->execute();
            // get number of rows
            $num = $stmt->rowCount();
            // if email exists, assign values to object properties for easy access
and use for php sessions
            if ($num > 0) {
                // get record details / values
                $row = $stmt->fetch(\PDO::FETCH_ASSOC);
                // assign values to object properties
                $this->id = $row['id'];
                $this->firstname=$row['firstname'];
                $this->lastname=$row['lastname'];
                $this->email=$row['email'];
                $this->password = $row['password'];
                // return true because email exists in the database
                return true;
            }
            // return false if email does not exist in the database
            return false;
        }

        function login(){
            // get posted data
            $data = json_decode(file_get_contents("php://input"));
            $arraydata = get_object_vars($data);
            // set product property values
           // $email = $data['email'];
            $email = $arraydata['email'];

            $email_exists = $this->emailExists($email);


            // check if email exists and if password is correct
            if($email_exists && password_verify($arraydata['password'], $this-
>password)){
                $token = array(
                    "iat" => $this->issued_at,
                    "exp" => $this->expiration_time,
                    "iss" => $this->issuer,
                    "data" => array(
                        "id" => $this->id,
                        "firstname" => $this->firstname,
                        "lastname" => $this->lastname,
                        "email" => $this->email
                    )
                );
                // set response code
                http_response_code(200);
                // generate jwt
                $jwt = JWT::encode($token, $this->key, "RS256");
                echo json_encode(
                    array(
                        "message" => "Successful login.",
                        "jwt" => $jwt
                    )
                );
            }// login failed
            else{
                // set response code
                http_response_code(401);
                // tell the user login failed
                echo json_encode(array("message" => "Login failed."));
            }
        }
```

```
    }
?>
```

En caso de no haber utilizado el composer, si has descargado la bibliotecas JWT directamente, tendrás que incluir las rutas de las clases:

```
include_once 'libs/php-jwt-master/src/BeforeValidException.php';
include_once 'libs/php-jwt-master/src/ExpiredException.php';
include_once 'libs/php-jwt-master/src/SignatureInvalidException.php';
include_once 'libs/php-jwt-master/src/JWT.php';
```

Probamos el servicio con Postman, enviamos las credenciales y recibimos el Token de autenticación:



Vamos a crear un cliente para el servicio anterior en la carpeta public, vamos a necesitar la **clave pública** junto con el JWT.

Validate_token.php

```
<?php
    require '../vendor/autoload.php';
// required headers
    header("Access-Control-Allow-Origin: http://localhost/rest-api-authentication-example/");
    header("Content-Type: application/json; charset=UTF-8");
    header("Access-Control-Allow-Methods: POST");
    header("Access-Control-Max-Age: 3600");
    header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, Authorization, X-Requested-With");


    use \Firebase\JWT\JWT;
    use \Firebase\JWT\Key;

    $key = "-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAv9f2Z8OfTswxG+V/rMiy
-----END PUBLIC KEY-----";

// get posted data
    $data = json_decode(file_get_contents("php://input"));
// get jwt
    $jwt = isset($data->jwt) ? $data->jwt : "";
// if jwt is not empty
    if ($jwt) {
        // if decode succeed, show user details
        try {
            // decode jwt
            $decoded = JWT::decode($jwt, new Key($key, 'RS256'));
            // set response code
            http_response_code(200);
            // show user details
            echo json_encode(array(
                "message" => "Access granted.",
                "data" => $decoded->data
            ));
        } // if decode fails, it means jwt is invalid
        catch (Exception $e) {
            // set response code
            http_response_code(401);
            // tell the user access denied  & show error message
            echo json_encode(array(
                "message" => "Access denied.",
                "error" => $e->getMessage()
            ));
        }
    } // show error message if jwt is empty
    else {
        // set response code
        http_response_code(401);
        // tell the user access denied
        echo json_encode(array("message" => "Access denied."));
    }
?>
```

Podemos probarlo con Postman:

Por último, podríamos crear un cliente que permitiese tanto crear el usuario en el servidor como autenticarse con las credenciales. De forma similar a como realizamos el cliente para el API Rest de personas, utilizaríamos un fichero en public realizado en Html + javascript.

Creamos el fichero **client.html**: