Ler e escrever Dados no formato JSON com PHP

Existem diversas APIs espalhadas pela WEB onde o tráfego de informações ocorre sempre no formato JSON, além de requisições AJAX que também suportam esse formato e ajudam muito no momento de exibir dados com JavaScript.

Usaremos:

- file_get_contents()
- json_encode()
- json_decode()
- json_last_error()

Definição do formato JSON

"JSON (JavaScript Object Notation – Notação de Objetos JavaScript) é uma formatação leve de troca de dados. Para seres humanos, é fácil de ler e escrever. Para máquinas, é fácil de interpretar e gerar. Está baseado em um subconjunto da linguagem de programação JavaScript, Standard ECMA-262 3a Edição - Dezembro – 1999. JSON é em formato texto e completamente independente de linguagem, pois usa convenções que são familiares às linguagens C e familiares, incluindo C++, C#, Java, JavaScript, Perl, Python e muitas outras. Estas propriedades fazem com que JSON seja um formato ideal de troca de dados." Fonte: http://www.json.org/json-pt.html

Com essa breve definição para os leitores que ainda não tinham contato com o formato JSON já fica mais fácil entender porque esse formato se tornou muito popular para troca de informações, pois no passado o formato XML sempre foi o preferido mas atualmente alguns especialistas consideram esse formato pesado se comparado ao JSON.

Felizmente o PHP trabalha muito bem com o formato JSON, possuindo funções nativas tanto para leitura quanto para escrita. Nesse post vou demonstrar 2 exemplos de como escrever arquivos no formato JSON com PHP, depois vamos ler esses mesmos arquivos usando PHP também.

Escrever no formato JSON

1 Exemplo:- Vamos escrever dados em JSON e gravar em um arquivo "*cadastro.json*", esse dados contém código, nome e telefone fictícios, vamos usar a função nativa do PHP:

json_encode()

```php
<?php

// Array com dados
$cliente1 = array(
    'codigo'   => '001',
    'nome'     => 'William',
    'telefone' => '012 9999-6352'
);

$cliente2 = array(
    'codigo'   => '002',
    'nome'     => 'Adriano',
    'telefone' => '012 8888-4452'
);

$cliente3 = array(
    'codigo'   => '003',
    'nome'     => 'Maria',
    'telefone' => '013 3434-4444'
);

// Atribui os 3 arrays para apenas um array
$dados = array($cliente1, $cliente2, $cliente3);

// Tranforma o array $dados em JSON
$dados_json = json_encode($dados);

// Cria o arquivo cadastro.json
// O parâmetro "a" indica que o arquivo será aberto para escrita
$fp = fopen("cadastro.json", "a");
// Escreve o conteúdo JSON no arquivo
$escreve = fwrite($fp, $dados_json);
// Fecha o arquivo
fclose($fp);
?>
```

Nosso arquivo “*cadastro.json*” deve ficar com esse formato após a gravação, podemos observar que ele não fica identado, nos exemplos de leitura identei para demonstrar melhor a estrutura.

```
[{"codigo":"001","nome":"William","telefone":"012 9999-
6352"},{"codigo":"002","nome":"Adriano","telefone":"012 8888-
4452"},{"codigo":"003","nome":"Maria","telefone":"013 3434-4444"}]
```

2 Exemplo :- Agora vamos montar um arquivo JSON “*contatos.json*” com mais detalhes, inclusive contendo objetos dentro de objetos e um identificador para os dados:

```
<?php
// Array com dados
$cliente1 = array(
    'codigo'   => '001',
    'nome'     => 'William',
    'telefones' => array(
        'residencial' => '011 4125-6352',
        'celular' => '011 9999-5241'
    )
);

$cliente2 = array(
    'codigo'   => '002',
    'nome'     => 'Adriano',
    'telefones' => array(
        'residencial' => '011 4125-635',
        'celular' => '011 9999-9652'
    )
);

$cliente3 = array(
    'codigo'   => '003',
    'nome'     => 'Maria',
    'telefones' => array(
        'residencial' => '011 4444-6352',
        'celular' => '011 9999-1245'
    )
);
```

```
// Atribui os 3 arrays apenas um array
$dados = array(
        $cliente1,
        $cliente2,
        $cliente3
);

// Adiciona o identificador "Contatos" aos dados
$dados_identificador = array('Contatos' => $dados);

// Tranforma o array $dados_identificador em JSON
$dados_json = json_encode($dados_identificador);

// Abre ou cria o arquivo contato.json
// "a" indicar que o arquivo é aberto para ser escrito
$fp = fopen("contatos.json", "a");
// Escreve o conteúdo JSON no arquivo
$escreve = fwrite($fp, $dados_json);

// Fecha o arquivo
fclose($fp);
?>
```

O formato do nosso arquivo "*contatos.json*" deve ser como exibido abaixo, podemos observar novamente que ele não fica identado, nos exemplos de leitura identei para demonstrar melhor a estrutura.


```
{"Contatos":[{"codigo":"001","nome":"William","telefones":{"residencia
l":"011 4125-6352","celular":"011 9999-
5241"}},{"codigo":"002","nome":"Adriano","telefones":{"residencial":"0
11 4125-635","celular":"011 9999-
9652"}},{"codigo":"003","nome":"Maria","telefones":{"residencial":"011
4444-6352","celular":"011 9999-1245"}}]}
```

Ler formato JSON

1 Exemplo :- Temos um arquivo chamado "*importar.json*", nele temos informações como código, nome, telefone do clientes no formato JSON:

```
[
 {"codigo":"001","nome":"William","telefone":"012 9999-6352"},
 {"codigo":"002","nome":"Adriano","telefone":"012 8888-4452"},
 {"codigo":"003","nome":"Maria","telefone":"013 3434-4444"}
]
```

Agora vamos ler esse arquivo com PHP utilizando as funções:

json_decode()

file_get_contents

```
<?php

// Atribui o conteúdo do arquivo para variável $arquivo

$arquivo = file_get_contents(importar.json');

// Decodifica o formato JSON e retorna um Objeto

$json = json_decode($arquivo);

// Loop para percorrer o Objeto

foreach($json as $registro):

    echo 'Código: ' . $registro->codigo . ' - Nome: ' . $registro-
>nome . ' - Telefone: ' . $registro->telefone . '<br>';

endforeach;

?>
```

2 Exemplo :- Nesse exemplo vamos ler um arquivo "*clientes.json*" JSON mais complexo, contendo identificador e mais de um telefone por contato, o que torna a leitura mais "chata" porém nada assustador pois temos tudo o que precisamos no PHP:

```
{
  "Contatos":
      [
        {
          "codigo": "001",
          "nome": "William",
          "telefones": {
              "residencial" : "011 4125-6352",
              "celular": "011 9999-5241"
          }
        },
        {
          "codigo": "002",
          "nome": "Adriano",
          "telefones": {
              "residencial" : "011 4125-6352",
```

```
            "celular": "011 9999-9652"
          }
        },
        {
          "codigo": "003",
          "nome": "Maria",
          "telefones": {
              "residencial" : "011 4444-6352",
              "celular": "011 9999-1245"
          }
        }
      ]
}
```

Usando a mesma função json_decode() vamos percorrer o objeto de retorno mas agora vamos acessar o identificador "*Clientes*" e para cada objeto contato temos 1 objeto "*telefones*" contendo números de telefone:

```
<?php

// Atribui o conteúdo do arquivo para variável $arquivo

$arquivo = file_get_contents('contatos.json');

// Decodifica o formato JSON e retorna um Objeto

$json = json_decode($arquivo);

// Loop para percorrer o Objeto

foreach($json->Contatos as $registro):

    echo 'Código: ' . $registro->codigo . ' - Nome: ' . $registro-
>nome . '<br>';

    echo 'Telefone Residência: ' . $registro->telefones->residencial .
' - Telefone Celular: ' . $registro->telefones->celular . '<br>';

endforeach;

?>
```

E se erros ocorrerem:

Retornamos os erros que podem ocorrer com a função:

json_last_error()

Sabemos que uma das coisas mais comuns na programação são os bugs, e isso também acontece com o JSON. Se houver algum bug, podemos usar a função json_last_error() para descobrir qual problema ocorreu.

Essa função retorna um número inteiro. Podemos dizer que cada um desses números é como um "código" do erro. Eles também podem ser acessados por meio de outras constantes do PHP.

Uma lista completa das possíveis constantes podem ser encontradas na documentação do PHP.

AJAX - Enviando dados de um objeto JavaScript para PHP

Para passar as informações para o **PHP** vamos utilizar a função **Ajax** do **jQuery**.

A ideia é que o envio das informações ocorra de forma assíncrona sem a necessidade de um refresh na página.

O código JavaScript

```
<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>
<script>
  var usuario = {
    'nome': 'João',
    'profissao': 'Engenheiro',
    'cidade': 'São Paulo'
  }
  var dados = JSON.stringify(usuario);
  $.ajax({
    url: 'recebe.php',
    type: 'POST',
    data: {data: dados},
    success: function(result){
      // Retorno se tudo ocorreu normalmente
    },
    error: function(jqXHR, textStatus, errorThrown) {
      // Retorno caso algum erro ocorra
    }
  });
</script>
```

Adicionando o jQuery

Na linha 1 do nosso código adicionamos a biblioteca **jQuery**. O **jQuery** é necessário para chamarmos a função **AJAX**.

```
<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>
```

Criando o objeto

Na linha 4 começamos a construir nosso **objeto**, é simplesmente um **array** com 3 **chaves**.

```
var usuario = {
    'nome': 'João',
    'profissao': 'Engenheiro',
    'cidade': 'São Paulo'
}
```

Convertendo o objeto para JSON

Na linha 10 fazemos a conversão do **objeto** para um formato que o **PHP** consegue manipular, esse formato é o **JSON**.

```
var dados = JSON.stringify(usuario);
```

Chamada da função Ajax do jQuery

Na função **Ajax** precisamos definir o arquivo de destino das informações, método de envio e os dados a serem transferidos.

```
$.ajax({
    url: 'recebe.php',
    type: 'POST',
    data: {data: dados},
    success: function(result){
      // Retorno se tudo ocorreu normalmente
    },
    error: function(jqXHR, textStatus, errorThrown) {
      // Retorno caso algum erro ocorra
    }
});
```

Na linha 13 informamos a url para o arquivo **PHP**, neste exemplo é um arquivo chamado “recebe.php”.

Na linha 14 informamos o método de envio das informações, neste exemplo estamos utilizando o método **POST**.

Na linha 15 informamos os dados a serem enviados, neste exemplo estamos enviando as informações armazenadas na variável “dados”.

Na linha 16 podemos obter o retorno se todo o processo de envio foi realizado com sucesso. Aqui dentro podemos construir a nossa lógica, inclusive manipulando o DOM.

Na linha 19 definimos a lógica caso algum erro tenha ocorrido durante o processo.

O código PHP

```
<?php
  $usuario = $_POST['data'];
  $dados = json_decode($usuario, true);
  var_dump($dados);
?>
```

Recebendo o objeto com PHP

Na linha 2 utilizamos a variável *$_POST* pra receber os dados enviados pelo **JavaScript**, esses dados estão sendo armazenados na variável *$usuario*.

```
$usuario = $_POST['data'];
```

Decodificando o JSON

Na linha 4 decodificamos o **JSON** com a função *json_decode* do **PHP**, as informações decodificadas são armazenadas na variável *$dados*. A variável *$dados* é um **Array** e pode ser manuseado como tal. Lembrando que se trata de um **Array Associativo**.

```
$dados = json_decode($usuario, true);
```

Exibindo os dados

A partir deste momento os dados podem ser manuseados conforme a nossa necessidade. Na linha 5 utilizamos a função *var_dump* para exibir todo o conteúdo do **Array**.

```
var_dump($dados);
```

Trabalhar com dados é muito importante em programação! Fazer com que duas linguagens que se complementam -como é o caso do JavaScript e PHP**-** compartilhem informações, se torna algo extremamente necessário.

Importar planilhas do Excel com PHP

O segredo para se trabalhar com importações de arquivos independente da extensão é conhecer a estrutura desse arquivo, ou seja, em que ordem estão dispostas as colunas que serão importadas. Nesse post vamos simular a importação de uma planilha contendo cadastros de clientes com alguns campos básicos, a ideia é ler essas linhas e inserir em uma tabela cliente no MySQL.

Layout e dados da planilha

Observem que as colunas possuem títulos, essa primeira linha terá que ser descartada no momento da importação, outro ponto importante é o código do cliente que possui "0" a esquerda do número então o campo que vai receber esse valor na tabela tem que ser do tipo VARCHAR, campos com data type INT não aceitam zeros a esquerda.

Observação: A extensão xlsx é gerada a partir de planilhas gravadas pela ferramenta Microsoft Excel, nesse exemplo abri uma planilha com a ferramenta LibreOffice Calc pois estou trabalhando com Ubuntu.

Criando o banco de dados e a tabela cliente no MySQL

```
CREATE DATABASE blog;

USE blog;

CREATE TABLE cliente(
	id int auto_increment primary key,
	codigo varchar(10),
	nome varchar(100),
	cpf varchar(15),
	email varchar(100),
	celular varchar(15)
);
```

Script PHP para importar planilhas

Um pré-requisito para conseguirmos escrever essa importação é a utilização de uma classe PHP que não é de minha autoria mas o leitor deve baixar o arquivo SimpleXLSX.class.php, essa classe possui todos os métodos necessários para leitura de planilhas do excel com a extensão xlsx.
Vamos apenas escrever outra classe "*ImportaPlanilha*" mas simples que vai funcionar como um "wrapper" dos métodos para ler os valores das células e utilizando uma conexão

PDO vamos criar outro método que vai servir para gravar os valores no banco de dados, desse modo conseguimos importar planilhas com PHP.

```
<?php
ini_set('max_execution_time','-1');
require_once "SimpleXLSX.class.php";

class ImportaPlanilha{

        // Atributo recebe a instância da conexão PDO
        private $conexao  = null;

     // Atributo recebe uma instância da classe SimpleXLSX
        private $planilha = null;

        // Atributo recebe a quantidade de linhas da planilha
        private $linhas   = null;

        // Atributo recebe a quantidade de colunas da planilha
        private $colunas  = null;

        /*
         * Método Construtor da classe
         * @param $path - Caminho e nome da planilha do Excel xlsx
         * @param $conexao - Instância da conexão PDO
         */
        public function __construct($path=null, $conexao=null){

                if(!empty($path) && file_exists($path)):
                        $this->planilha = new SimpleXLSX($path);
                        list($this->colunas, $this->linhas) = $this-
>planilha->dimension();
                else:
                        echo 'Arquivo não encontrado!';
                        exit();
                endif;

                if(!empty($conexao)):
                        $this->conexao = $conexao;
                else:
                        echo 'Conexão não informada!';
                        exit();
                endif;

        }

        /*
         * Método que retorna o valor do atributo $linhas
         * @return Valor inteiro contendo a quantidade de linhas na
planilha
         */
        public function getQtdeLinhas(){
                return $this->linhas;
        }

        /*
         * Método que retorna o valor do atributo $colunas
         * @return Valor inteiro contendo a quantidade de colunas na
planilha
         */
```

```
        public function getQtdeColunas(){
                return $this->colunas;
        }

        /*
         * Método que verifica se o registro CPF da planilha já existe na
tabela cliente
         * @param $cpf - CPF do cliente que está sendo lido na planilha
         * @return Valor Booleano TRUE para duplicado e FALSE caso não
         */
        private function isRegistroDuplicado($cpf=null){
                $retorno = false;
                try{
                        if(!empty($cpf)):
                                $sql = 'SELECT id FROM cliente WHERE cpf =
?';
                                $stm = $this->conexao->prepare($sql);
                                $stm->bindValue(1, $cpf);
                                $stm->execute();
                                $dados = $stm->fetchAll();

                                if(!empty($dados)):
                                        $retorno = true;
                                else:
                                        $retorno = false;
                                endif;
                        endif;


                }catch(Exception $erro){
                        echo 'Erro: ' . $erro->getMessage();
                        $retorno = false;
                }

                return $retorno;
        }

        /*
         * Método para ler os dados da planilha e inserir no banco de
dados
         * @return Valor Inteiro contendo a quantidade de linhas
importadas
         */
        public function insertDados(){

                try{
                        $sql = 'INSERT INTO cliente (codigo, nome, cpf,
email, celular)VALUES(?, ?, ?, ?, ?)';
                        $stm = $this->conexao->prepare($sql);

                        $linha = 0;
                        foreach($this->planilha->rows() as $chave =>
$valor):
                                if ($chave >= 1 && !$this-
>isRegistroDuplicado(trim($valor[2]))):
                                        $codigo  = trim($valor[0]);
                                        $nome    = trim($valor[1]);
                                        $cpf     = trim($valor[2]);
                                        $email   = trim($valor[3]);
                                        $celular = trim($valor[4]);
```

```
                                        $stm->bindValue(1, $codigo);
                                        $stm->bindValue(2, $nome);
                                        $stm->bindValue(3, $cpf);
                                        $stm->bindValue(4, $email);
                                        $stm->bindValue(5, $celular);
                                        $retorno = $stm->execute();

                                        if($retorno == true) $linha++;
                                endif;
                        endforeach;

                        return $linha;
                }catch(Exception $erro){
                        echo 'Erro: ' . $erro->getMessage();
                }

        }
}
```

Observações sobre os principais métodos da classe "*ImportaPlanilha*":

1 – No método construtor recebo e executo uma validação básica dos 2 parâmetros $path e $conexao, instancio um objeto da classe SimpleXLSX, logo abaixo chamo o método dimension() que pertence a classe SimpleXLSX e capturo o retorno desse método atribuindo os valores de $linha e $coluna, com isso já temos quantidade de linhas e colunas da planilha passada como parâmetro.

```
 public function __construct($path=null, $conexao=null){

        if(!empty($path) && file_exists($path)):
                $this->planilha = new SimpleXLSX($path);
                list($this->colunas, $this->linhas)= $this->planilha-
>dimension();
        else:
                echo 'Arquivo não encontrado!';
                exit();
        endif;

        if(!empty($conexao)):
                $this->conexao = $conexao;
        else:
                echo 'Conexão não informada!';
                exit();
        endif;

}
```

2 – Esse método tem como objetivo verificar se o CPF passado como parâmetro já existe na tabela, se existir ele retorna TRUE senão é retornado FALSE, com esse método conseguimos impedir que sejam duplicados dados na tabela cliente usando como parâmetro o valor do CPF.

```
 private function isRegistroDuplicado($cpf=null){
        $retorno = false;

        try{
                if(!empty($cpf)):
                        $sql = 'SELECT id FROM cliente WHERE cpf = ?';
                        $stm = $this->conexao->prepare($sql);
                        $stm->bindValue(1, $cpf);
                        $stm->execute();
                        $dados = $stm->fetchAll();

                        if(!empty($dados)):
                                $retorno = true;
                        else:
                                $retorno = false;
                        endif;
                endif;


        }catch(Exception $erro){
                echo 'Erro: ' . $erro->getMessage();
                $retorno = false;
        }

        return $retorno;
}
```

3 – Para finalizar temos o método que realmente importa os dados da planilha para o banco de dados, observem que utilizamos o objeto SimpleXLSX que está no atributo $planilha para percorrer as linhas e as colunas da planilha, a leitura dos valores é feita passando um índice. Antes de inserir verifico se a linha que está sendo lida é a primeira, nesse caso sabemos que essa linha contém apenas os títulos das colunas então não serão inseridos esses valores e também chamamos o método “isRegistroDuplicado()” para verificar se o CPF já foi cadastrado.

```
 public function insertDados(){

try{

$sql = 'INSERT INTO cliente (codigo, nome, cpf, email, celular) VALUES
(?, ?, ?, ?, ?)';
        $stm = $this->conexao->prepare($sql);

        $linha = 0;
        foreach($this->planilha->rows() as $chave => $valor):
        if ($chave >= 1 && !$this->isRegistroDuplicado(trim($valor[2]))):

                $codigo  = trim($valor[0]);
                $nome    = trim($valor[1]);
                $cpf     = trim($valor[2]);
                $email   = trim($valor[3]);
                $celular = trim($valor[4]);

                $stm->bindValue(1, $codigo);
                $stm->bindValue(2, $nome);
```

```
                $stm->bindValue(3, $cpf);
                $stm->bindValue(4, $email);
                $stm->bindValue(5, $celular);
                $retorno = $stm->execute();

                if($retorno == true) $linha++;
        endif;
endforeach;

return $linha;

}catch(Exception $erro){
                echo 'Erro: ' . $erro->getMessage();
        }

}
```

Importando dados da Planilha

Abaixo temos um exemplo de como usar a classe “ImportaPlanilha.class.php” para importar planilhas, observem que estou configurando uma conexão PDO para ser passada como parâmetro no momento que instanciarmos a classe, além do caminho e nome da planilha xlsx que nesse exemplo esta no mesmo diretório que os scripts.

```
<?php

/* Seta configuração para não dar timeout */
ini_set('max_execution_time','-1');

/* Require com a classe de importação construída */
require 'ImportaPlanilha.class.php';

/* Instância conexão PDO com o banco de dados */
$pdo = new PDO('mysql:host=localhost;dbname=blog', 'root', '123456');

/* Instância o objeto importação e passa como parâmetro o caminho da
planilha e a conexão PDO */
$obj = new ImportaPlanilha('./clientes.xlsx', $pdo);

/* Chama o método que retorna a quantidade de linhas */
echo 'Quantidade de Linhas na Planilha ' , $obj->getQtdeLinhas(), '<br>';

/* Chama o método que retorna a quantidade de colunas */
echo 'Quantidade de Colunas na Planilha ' , $obj->getQtdeColunas(),
'<br>';

/* Chama o método que inseri os dados e captura a quantidade linhas
importadas */
$linhasImportadas = $obj->insertDados();

/* Imprime a quantidade de linhas importadas */
echo 'Foram importadas ', $linhasImportadas, ' linhas';
?>
```

Se tudo correr bem após executar o script acima no navegador, devemos obter o resultado como exibido na imagem abaixo, onde temos a quantidade de linhas e colunas existentes na planilha, quantidade de registros que foi importada para o banco de dados.

Ao executarmos novamente esse script, a quantidade de registros importada será “0” zero, pois estamos verificando no momento do INSERT se o CPF do cliente já está gravado na tabela, nesse caso nenhum registro será inserido.