

PHP - Sistema de Rotas

alf@esmonserrate.org



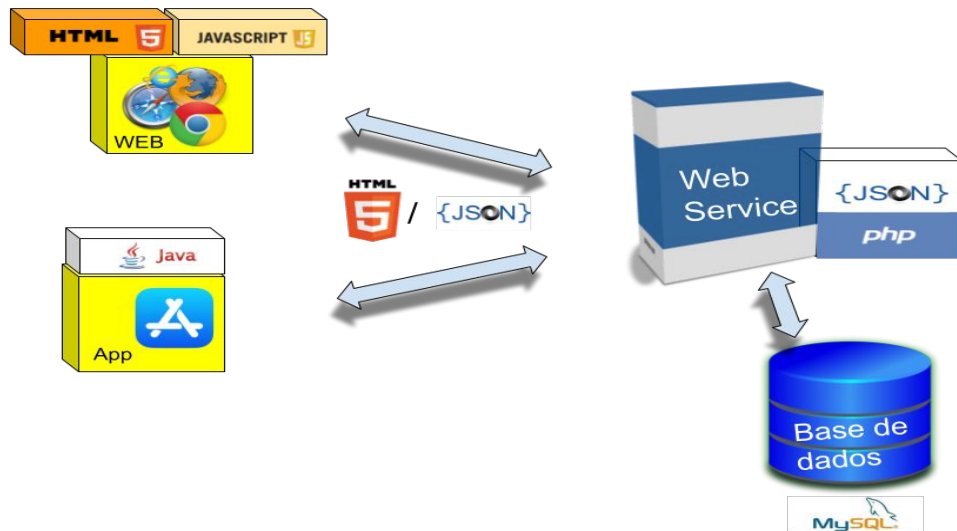
Conteúdos

- Problemas
- Objetivos
- Arquitetura do Sistema
- Web Services
- Estrutura do Site
- Segurança
- Papéis
- Rotas
- json
- Classes
- Controllers
- Classes do Sistema

Problema

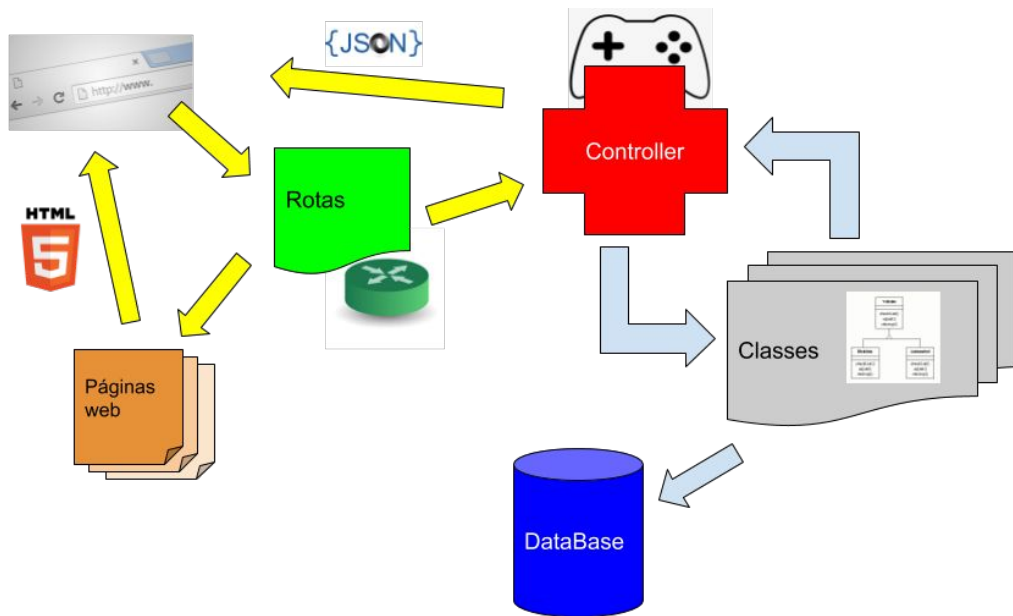
- Os mecanismos de SEO (Search Engine Optimization) valorizam url bonitas
- O acesso a todas as pastas do site pode ser indesejado

Arquitectura do Sistema

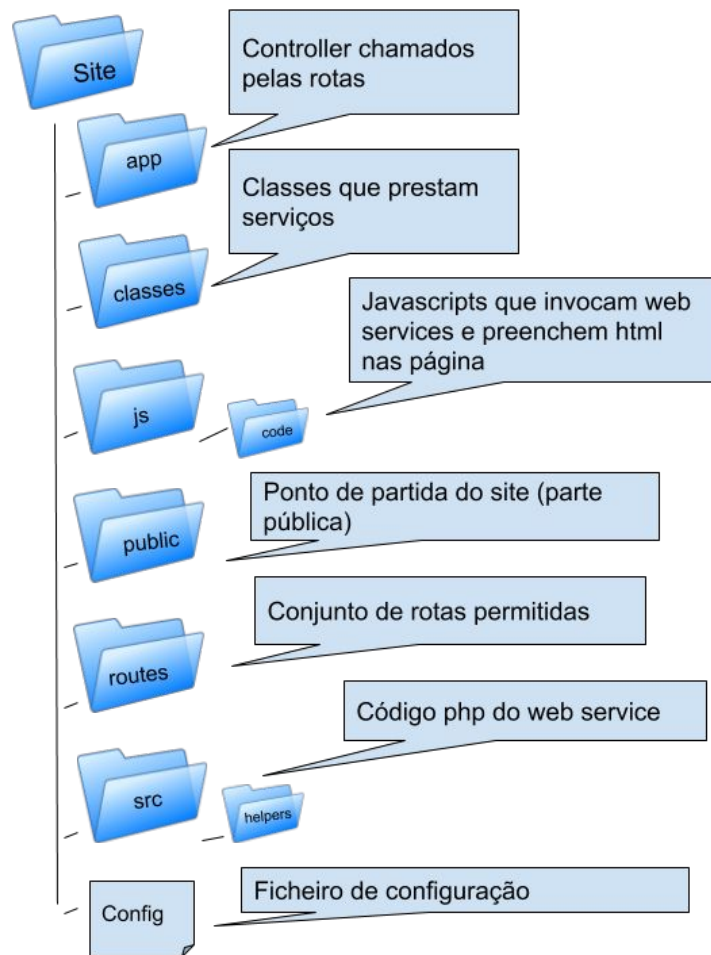


Web Services

“Web Service é uma solução utilizada na integração de sistemas e na comunicação entre aplicações diferentes” [1]



Estrutura do Site



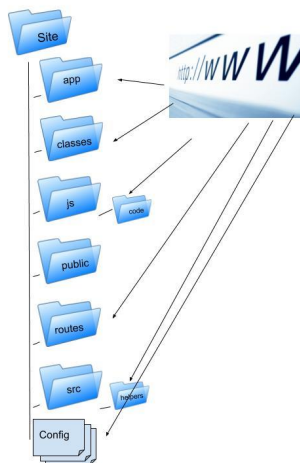
Segurança do site

Ao contrário da situação tradicional usando esta estrutura só a pasta public fica exposta ao exterior.

Tradicional

URL
`http://www.nossosite.pt`

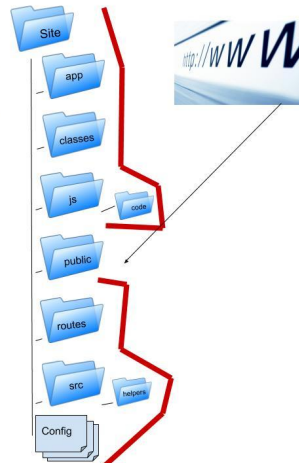
Path
`/var/www/nossosite`



Com rotas

URL
`http://www.nossosite.pt`

Path
`/var/www/nossosite/public`



Criar Rotas (Arquiteto, (GWS))

ficheiro routes.php em routes

```
<?php
use src\Route as Route;

Route::get('/', function(){
    echo "Página inicial";
});

Route::get('/adm', function(){
    require _PATH_TEMPLATE . "backoffice.php";
});

Route::get(['set' => '/clientes', 'as' => 'clientes.listaClientes'], 'ControllerClientes@listaClientes');
Route::get(['set' => '/cliente/{id}/show', 'as' => 'clientes.show'], 'ControllerClientes@show');
Route::put(['set' => '/cliente/delete', 'as' => 'clientes.delete'], 'ControllerClientes@delete');
Route::put(['set' => '/cliente/update', 'as' => 'clientes.update'], 'ControllerClientes@update');

Route::get(['set' => '/servicos', 'as' => 'servicos.listaServicos'], 'ControllerServicos@listaServicos');
Route::get(['set' => '/servico/{id}/show', 'as' => 'servicos.show'], 'ControllerServicos@show');
Route::put(['set' => '/servico/delete', 'as' => 'servicos.delete'], 'ControllerServicos@delete');
```

Envio de uma mensagem em texto

responde a <http://site.com/adm>
(../public/) + adm
envia o ficheiro backoffice.php

responde a <http://site.com/clientes>
(../public/) + clientes
chama o controllerClientes o método
lista clientes

galeria.esmonsserrate.org/finalista: x +

← → ↻ ⓘ Inseguro | galeria.esmonsserrate.org/finalista/2019/PAPsistemaTickets/public/fornecedores

```
[{"id_F": "5", "Nome": "Atendimento aos Alunos", "Balcao": "Mesa4", "id_LS": "5", "Ativo": "1", "numElements": 6}, {"id_F": "2", "Nome": "Contabilidade", "Balcao": "Mesa2", "id_LS": "2", "Ativo": "1"}, {"id_F": "3", "Nome": "Serviço Pes  
Docente", "Balcao": "Mesa3", "id_LS": "3", "Ativo": "1"}, {"id_F": "6", "Nome": "Atendimento Geral", "Balcao": "Mesa5",  
{"id_F": "1", "Nome": "A aguardar atribuição", "Balcao": "Mesa1", "id_LS": "1", "Ativo": "1"}, {"id_F": "4", "Nome": "Serviço Pe  
Docente", "Balcao": "Mesa1", "id_LS": "3", "Ativo": "1"}]
```

{JSON}

Criar Rotas (Arquiteto, (GWS))

ficheiro routes.php em routes

```
<?php
use src\Route as Route;

Route::get('/', function(){
    echo "Página inicial";
});

Route::get('/adm', function(){
    require _PATH_TEMP . "backoffice.php";
});

Route::get(['set' => '/clientes', 'as' => 'clientes.listaClientes'], 'ControllerClientes@listaClientes');
Route::get(['set' => '/cliente/{id}/show', 'as' => 'clientes.show'], 'ControllerClientes@show');
Route::put(['set' => '/cliente/delete', 'as' => 'clientes.delete'], 'ControllerClientes@delete');
Route::put(['set' => '/cliente/update', 'as' => 'clientes.update'], 'ControllerClientes@update');

Route::get(['set' => '/servicos', 'as' => 'servicos.listaServicos'], 'ControllerServicos@listaServicos');
Route::get(['set' => '/servico/{id}/show', 'as' => 'servicos.show'], 'ControllerServicos@show');
Route::put(['set' => '/servico/delete', 'as' => 'servicos.delete'], 'ControllerServicos@delete');
```

Rotas do tipo GET para obter dados

Rotas do tipo PUT para enviar dados

rotas em que são adicionados parâmetros que depois são usados no controller (id)

www.nossosite.com/servico/5/show

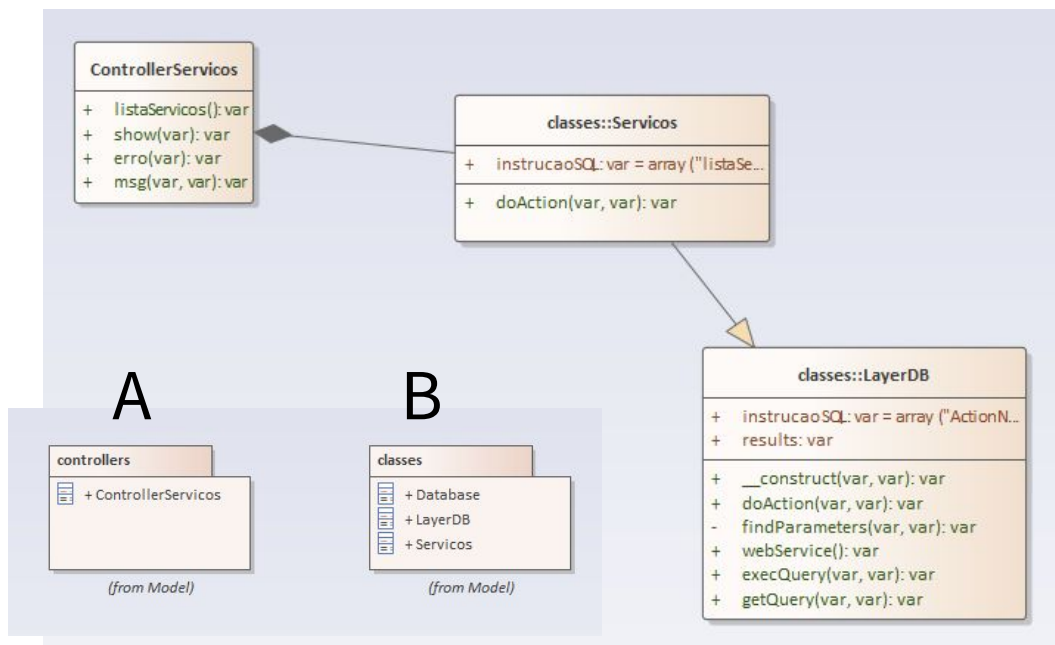
e vamos obter um json com a informação do serviço 5

Inseguro | galeria.esmonserrate.org/finalista/2019/PAPsistemaTickets/public/servico/5/show

[[{"id_LS": "5", "Nome": "Atendimento aos Alunos", "Ativo": "1", "numElements": 1}]]

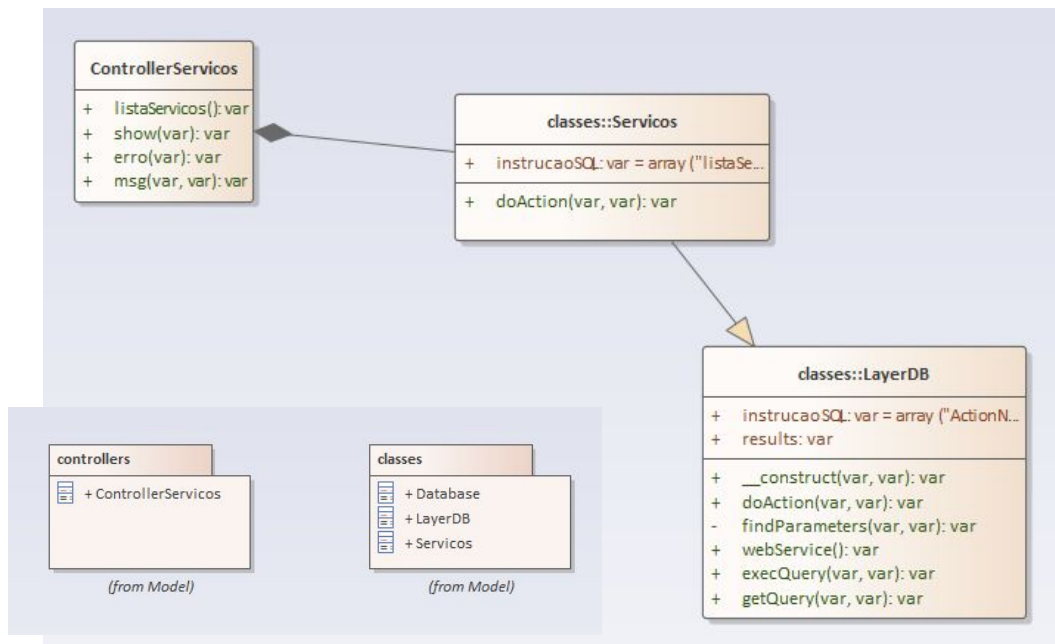
Controller (GWS)

- A. Os controllers estão guardados na pasta app
- B. As classe guardadas em classes



Controller_(GWS)

- O ControllerServicos é usado nas rotas de serviços
- Tem 4 métodos
- Os 4 são públicos (+)
- Utiliza uma classe serviços que é um extensão de LayerDB



Controller_(GWS)

- A. O ControllerServicos tem 4 métodos
- B. listaServicos e show são usados nas rotas
- C. listaServicos usa uma classe Servicos

A

```
ControllerService x
1 <?php
2
3 namespace app;
4 use classes\Database;
5 use classes\tickets\Servicos;
6
7 class ControllerServicos{
8
9     public function listaServicos(){
10         $servicos=new Servicos("listaServicos");
11         echo $servicos->webService();
12     }
13
14     public function show($id) {
15
16         $p['id']=$id;
17         $servicos=new Servicos("seeServicos",$p);
18         echo $servicos->webService();
19     }
20
21     public function erro($message){
22         echo $this->msg('Error', $message);
23     }
24
25     public function msg($title, $message){
26         echo json_encode(['Title'=>$title, 'Message' => $message]);
27     }
28 }
29
30
31
32
33 ?>
```

C

B

```
Route::get(['set' => '/servicos', 'as' => 'servicos.listaServicos'], 'ControllerServicos@listaServicos');
Route::get(['set' => '/servico/{id}/show', 'as' => 'servicos.show'], 'ControllerServicos@show');
```

Controller (GWS)

- A. O objecto Servico é criado passando o valor “listaServico” que corresponde ao que queremos fazer
- B. Usamos o método webService para enviar a estrutura json com o resultado

```
ControllerService x
1 <?php
2
3 namespace app;
4 use classes\Database;
5 use classes\tickets\Services;
6
7 class ControllerServices{
8
9     public function listaServicos(){
10         $servicos=new Services("listaServicos");
11         echo $servicos->webService();
12     }
13
14
15     public function show($id) {
16
17         $p['id']=$id;
18         $servicos=new Services("seeServicos",$p);
19         echo $servicos->webService();
20     }
21
22
23     public function erro($message){
24         echo $this->msg('Error', $message);
25     }
26
27     public function msg($title, $message){
28         echo json_encode(['Title'=>$title, 'Message' => $message]);
29     }
30 }
31
32
33 ?>
```

A

B

Controller (GWS)

- A. show é usado na 2ª rota
- B. passamos um parâmetro na rota e no método (id) que corresponde ao número do serviço
- C. em show usamos novamente o objecto Servicios mas passamos dois parâmetros: um texto e um array

```
ControllerService x
1 <?php
2
3 namespace app;
4 use classes\Database;
5 use classes\tickets\Servicos;
6
7 class ControllerServicos{
8
9     public function listaServicos(){
10         $servicos=new Servicios("listaServicos");
11         echo $servicos->webService();
12     }
13
14     public function show($id) {
15
16         $p['id']=$id;
17         $servicos=new Servicios("seeServicos",$p);
18         echo $servicos->webService();
19     }
20
21     public function erro($message){
22         echo $this->msg('Error', $message);
23     }
24
25     public function msg($title, $message){
26         echo json_encode(['Title'=>$title, 'Message' => $message]);
27     }
28 }
29
30
31
32
33 ?>
```

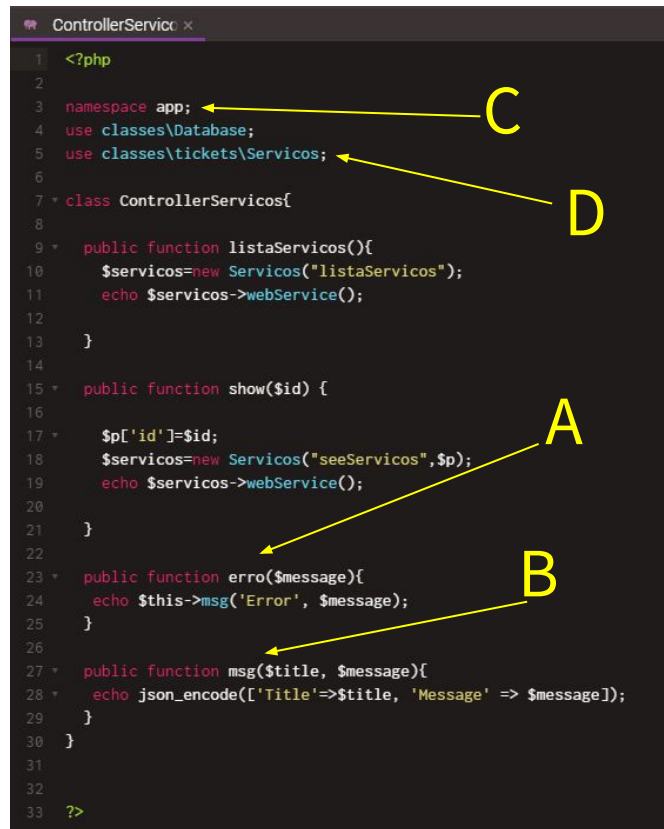
A

```
Route::get(['set' => '/servicos', 'as' => 'servicos.listaServicos'], 'ControllerServicos@listaServicos');
Route::get(['set' => '/servico/{id}/show', 'as' => 'servicos.show'], 'ControllerServicos@show');
```

Controller _(GWS)

- A. erro - serve para mandar mensagens de erro no formato json
- B. msg server para enviar mensagens em formato json
- C. As classes pertencem a um namespace (pasta)
- D. As classe são incluídas para serem usadas

```
ControllerService x
1  <?php
2
3  namespace app; ← C
4  use classes\Database;
5  use classes\tickets\Servicos; ← D
6
7  class ControllerServicos[
8
9  public function listaServicos(){
10     $servicos=new Servicos("listaServicos");
11     echo $servicos->webService();
12 }
13
14 public function show($id) {
15
16     $p['id']=$id;
17     $servicos=new Servicos("seeServicos",$p);
18     echo $servicos->webService();
19 }
20
21 }
22
23 public function erro($message){
24     echo $this->msg('Error', $message);
25 }
26
27 public function msg($title, $message){
28     echo json_encode(['Title'=>$title, 'Message' => $message]);
29 }
30 }
31
32
33 ?>
```



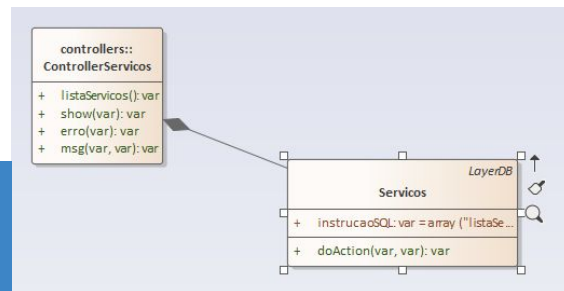
Classes do sistema (pPHP)

- Todas as classes do sistema usam o código já desenvolvido para acesso à base de dados
- Desta forma se decidirmos alterar o gestor de base de dados todo o sistema se mantém a funcionar, bastando trocar a classe de acesso à base de dados
- O mesmo acontece de usarmos ficheiros de texto ou xml

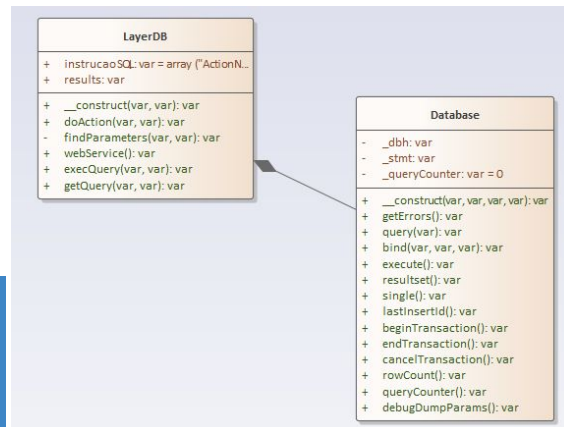
Classes do sistema (pPHP)

- Conseguimos isolar as duas camadas
- Podemos incorporar no nosso sistema outras classe que prestam serviços
 - por exemplo uma classe para criar pdf, ou imprimir, ou mandar sms...

Camada de Negócio



Camada de Dados



Classes do sistema (pPHP)

A. A classe Servicos estende LayerBD

- a. não precisamos de criar métodos de acesso a base de dados
- b. nem de criação de json
- c. esse código já foi implementado em LayerBD



```
<?php

namespace classes\tickets;
use classes\db\Database;
use classes\db\LayerDB;

ini_set("error_reporting", E_ALL);

class Servicos extends LayerDB{

    public $instrucaoSQL = array ("listaServicos" => 'SELECT `id_LS`, `Nome`, `Ativo` FROM `TicketsListaServicos` order by `Nome`',
                                "seeServicos" => 'SELECT `id_LS`, `Nome`, `Ativo` FROM `TicketsListaServicos` WHERE `id_LS`=id;',
                                "addServicos" => 'INSERT INTO `TicketsListaServicos` (`id_LS`, `Nome`, `Ativo`)
                                                VALUES (:id_LS, :Nome, 1 );');

    public function doAction($saccao, $parameters=""){
        switch ($saccao){
            case "addServicos":
                $this->execQuery($saccao, $parameters);
                break;
            case "listaServicos":
            case "seeServicos":
                $this->getQuery($saccao, $parameters);
                break;
            default:
                break;
        }
    }
}
```

Classes do sistema (pPHP)

- A. O 1º parâmetro define a acção que queremos realizar
- B. O 2º parâmetro (não é obrigatório) passa valores ou filtros

```
<?php

namespace classes\tickets;
use classes\db\Database;
use classes\db\LayerDB;

ini_set("error_reporting", E_ALL);

class Servicos extends LayerDB{

    public $instrucaoSQL = array ("listaServicos" => 'SELECT `id_LS`, `Nome`, `Ativo` FROM `TicketsListaServicos` order by `Nome`',
                                "seeServicos" => 'SELECT `id_LS`, `Nome`, `Ativo` FROM `TicketsListaServicos` WHERE `id_LS`=:id;',
                                "addServicos" => 'INSERT INTO `TicketsListaServicos` (`id_LS`, `Nome`, `Ativo`)
                                                VALUES (:id_LS, :Nome, 1 );');

    public function doAction($acao, $parameters=""){
        switch ($acao){
            case "addServicos":
                $this->execQuery($acao, $parameters);
                break;
            case "listaServicos":
            case "seeServicos":
                $this->getQuery($acao, $parameters);
                break;
            default:
                break;
        }
    }
}
```

The diagram illustrates the mapping of parameters in the `doAction` method to the list items A and B. A yellow arrow points from item A to the `$acao` parameter in the `doAction` method signature. Another yellow arrow points from item B to the `$parameters` parameter in the same signature. A third yellow arrow points from item B to the `WHERE `id_LS`=:id;` SQL query string in the `seeServicos` element of the `$instrucaoSQL` array.

Classes do sistema (pPHP)

- A. método para acções de leitura da base de dados (SELECT)
- B. método para acções de escrita na base de dados (INSERT, UPDATE, DELETE)

```
<?php

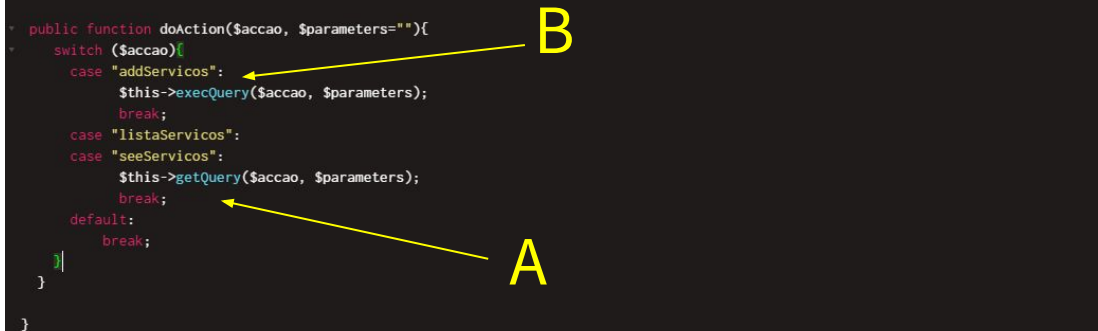
namespace classes\tickets;
use classes\db\Database;
use classes\db\LayerDB;

ini_set("error_reporting", E_ALL);

class Servicos extends LayerDB{

    public $instrucaoSQL = array ("listaServicos" => 'SELECT `id_LS`, `Nome`, `Ativo` FROM `TicketsListaServicos` order by `Nome`',
                                  "seeServicos" => 'SELECT `id_LS`, `Nome`, `Ativo` FROM `TicketsListaServicos` WHERE `id_LS`=id;',
                                  "addServicos" => 'INSERT INTO `TicketsListaServicos` (`id_LS`, `Nome`, `Ativo`)
                                                  VALUES (:id_LS, :Nome, 1 );');

    public function doAction($sacao, $parameters=""){
        switch ($sacao){
            case "addServicos":
                $this->execQuery($sacao, $parameters);
                break;
            case "listaServicos":
            case "seeServicos":
                $this->getQuery($sacao, $parameters);
                break;
            default:
                break;
        }
    }
}
```



Classes do sistema (pPHP)

- A. cada acção tem um nome
- B. e corresponde a uma instrução SQL
- C. O parâmetros são passados por nome se preocupação (:id) com a ordem

```
<?php

namespace classes\tickets;
use classes\db\Database;
use classes\db\LayerDB;

ini_set("error_reporting", E_ALL);

class Servicos extends LayerDB{

    public $instrucaoSQL = array ("listaServicos" => 'SELECT `id_LS`, `Nome`, `Ativo` FROM `TicketsListaServicos` order by `Nome`',
                                "seeServicos" => 'SELECT `id_LS`, `Nome`, `Ativo` FROM `TicketsListaServicos` WHERE `id_LS`=:id;',
                                "addServicos" => 'INSERT INTO `TicketsListaServicos` (`id_LS`, `Nome`, `Ativo`)
                                                VALUES (:id_LS, :Nome, 1 );');

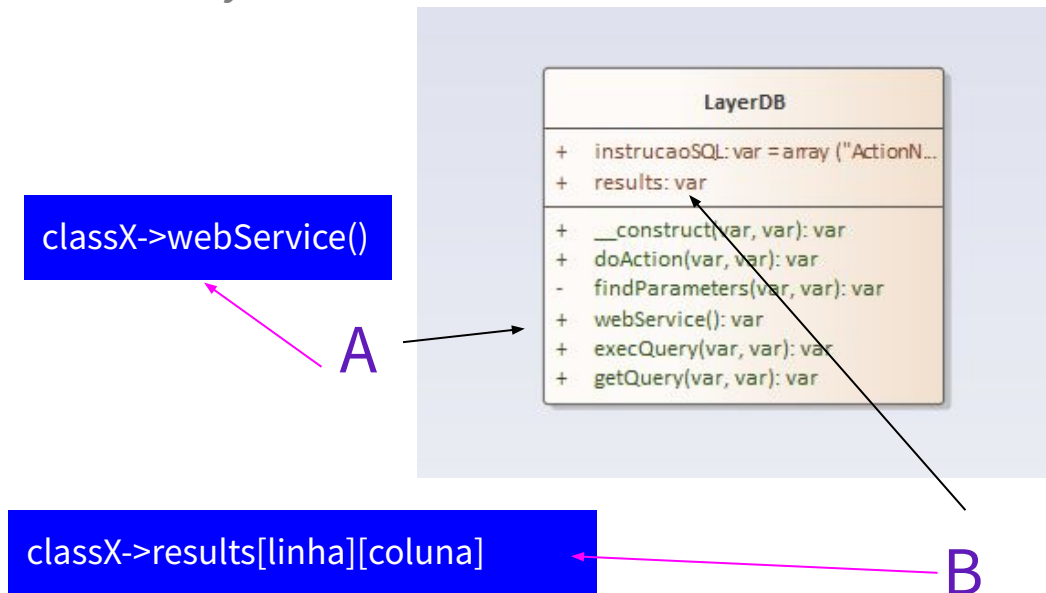
    public function doAction($acao, $parameters=""){
        switch ($acao){
            case "addServicos":
                $this->execQuery($acao, $parameters);
                break;
            case "listaServicos":
            case "seeServicos":
                $this->getQuery($acao, $parameters);
                break;
            default:
                break;
        }
    }
}
```

The diagram illustrates the mapping between the code and the requirements:

- A** points to the `"listaServicos"` key in the `$instrucaoSQL` array, corresponding to requirement A.
- B** points to the `"seeServicos"` key in the `$instrucaoSQL` array, corresponding to requirement B.
- C** points to the `"addServicos"` key in the `$instrucaoSQL` array and the `doAction` function, corresponding to requirement C.

Classes do sistema (pPHP)

- A. Os resultados podem ser obtidos como json - usando o método `webService()`
- B. Podem ser obtidos directamente com o array `results`



Classes do sistema (pPHP)

- A. Exemplo para ler web service json em PHP
- B. Escolher a rota do ws
- C. ler o resultado para o array \$data
- D. ver linha a linha

```
$json = file_get_contents("http://www.forumvianense.org/public/eventos");  
$data = json_decode($json, true);  
//var_dump($data);  
foreach($data as $linha){  
    print_r($linha);  
    echo "<br>-----<br>";  
}  
|
```

C B

D

Classes do sistema (pPHP)

- A. Passamos dois critérios
- B. Recebemos os resultados
- C. Acedemos directamente usando a linha e o campo
- D. navegamos nos resultados

A

B

D

```
private function totalYearAccount($parametros){  
  
    $p['idAccount']=72;  
    $p['year']=2020;  
    $vendas = new Clientes("total de vendas", $p);  
    $rs=$vendas->resultset();  
  
    echo $vendas->results[0]['numElements'];  
    echo $vendas->results[0]['year'];  
  
    foreach ($rs as $linha){  
        echo $rs['idAccount'];  
        echo $rs['total'];  
    }  
}
```

C

Javascript (pJS)

- Preferencialmente a ligação das páginas aos web service deve ser feito em JS
- O JS corre na máquina do cliente e por isso devemos ser cuidadosos com que revelamos nas páginas
- Password e acesso escritos em script JS ficam disponíveis do lado do cliente
- Ler os dados de uma consulta e carregá-los para a página, escondendo detalhes no html é o mesmo que os colocar à mostra

Javascript (pJS)

- Exemplo de um script para alterar uma página

A

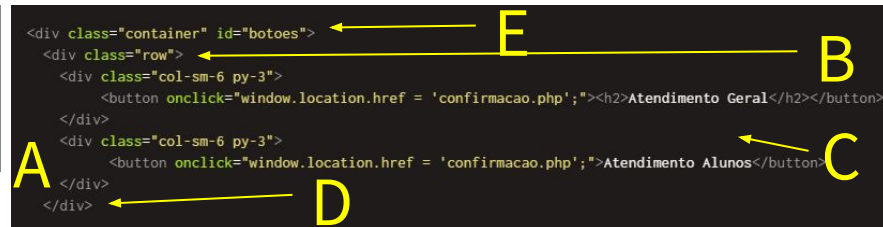
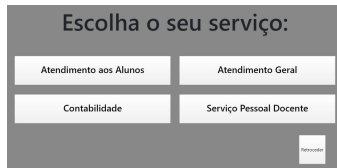
B

C

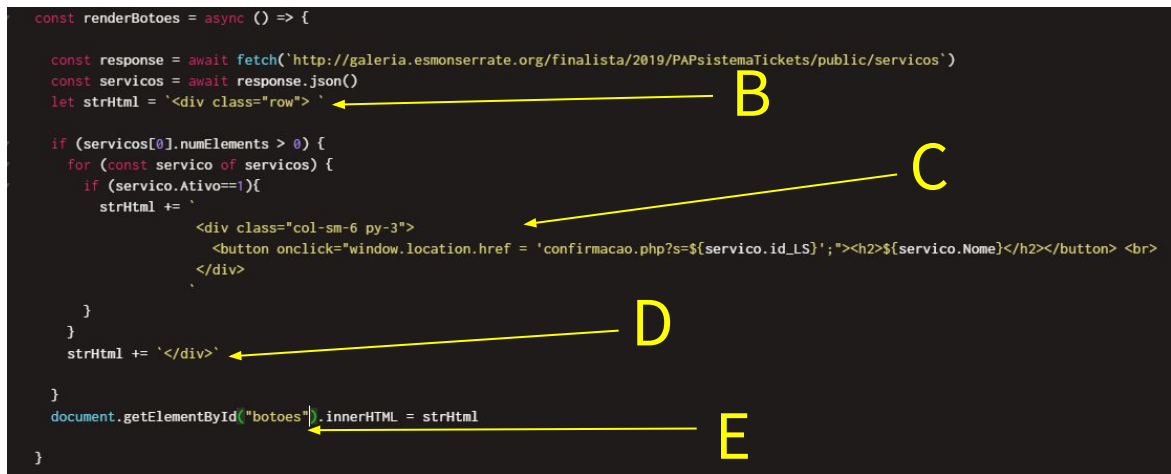
```
const renderBotoes = async () => {  
  
    const response = await fetch('http://galeria.esmonserrate.org/finalista/2019/PAPsistemaTickets/public/servicos')  
    const servicos = await response.json()  
    let strHtml = '<div class="row"> '  
  
    if (servicos[0].numElements > 0) {  
        for (const servico of servicos) {  
            if (servico.Ativo==1){  
                strHtml += '  
                    <div class="col-sm-6 py-3">  
                        <button onclick="window.location.href = \'confirmacao.php?s=${servico.id_LS}\';"><h2>${servico.Nome}</h2></button> <br>  
                    </div>  
                '  
            }  
        }  
        strHtml += '</div>'  
    }  
    document.getElementById("botoes").innerHTML = strHtml  
}
```

- A. Nome da função
- B. Indicação que deve esperar pelo WS
- C. Chamar o WS e guardar a resposta em servicos

Javascript (pJs)



- Exemplo de um script para alterar uma página



- A. O objetivo é alterar o html da página
- B. Cabeçalho do html
- C. Cada linha da tabela gera um botão
- D. Final do html
- E. Carregar no elemento

Javascript (pJS)

- Este script pode ser usado em várias página e alterar um elemento “botão”

JavaScript - post

```
const _SERVIDORt= window.location.protocol + "://" + window.location.host + "/";

const addAlbum = async (tit,descricao,pai,accao) => {

    const response = await fetch(_SERVIDORt+`public/album/add`, {
        headers: {
            "Content-Type": "application/x-www-form-urlencoded"
        },
        method: "POST",
        body: `tit=${tit}&descricao=${descricao}&pai=${pai}&accao=${accao}`
    })

    const lv = await response.text()
}
```

PHP - web service - post

No ficheiro de retas vemos qual o controller que é ativado

```
Route::get(['set' => '/albums/apagados/{limitInf}/{limitSup}/shows', 'as' => 'album.AlbunsApagados'], 'ControllerAlbum@apagarAlbum');  
Route::post(['set' => '/album/add', 'as' => 'album.adicionarAlbum'], 'ControllerAlbum@adicionarAlbum');  
Route::post(['set' => '/album/{id}/update', 'as' => 'album.atualizarAlbum'], 'ControllerAlbum@atualizarAlbum');
```

PHP - web service - post

```
public function adicionarAlbum(){
    $nomeL=$fic->limpaCaca($_POST['tit']);
    $p['descricao']=$_POST['descricao'];
    $p['id']=$_POST['pai'];
    $p['pai']=$_POST['pai'];
    $p['nome']=$nomeL;
    $p['titulo']=$_POST['tit'];

    $res= new Album("AlbumPorId",$p);

    echo $servicos->results[0]['lastId'];
    // echo $res;
}
```

Javascript

```
document.getElementById("codArtigoGRP").hidden=false
```

```
document.getElementById("txtGRP").hidden=true
```

Mostrar ou esconder elementos html

```
if (getURLPos(4)=="alterar"){  
    verTarefa(getURLPos(5))  
    verListaTarefas(getURLPos(5));  
    //alert("alterar");  
}  
else{  
    //alert(getURLPos(4));  
}
```

Ler a url e retirar um valor da posição 4

Referências

[1]. Wikipédia (2019). Web service – Wikipédia, a enciclopédia livre;

https://pt.wikipedia.org/wiki/Web_service

[2]. wpjr2's (2008). Orientação por Objetos: Vantagens e Desvantagens « wpjr2's Weblog;

<https://wpjr2.wordpress.com/2008/04/23/orientacao-por-objetos-vantagens-e-desvantagens/>