# LAB 8 – Programação com o framework Symfony [parte 1]

O objectivo deste laboratório é repetir a funcionalidade do portal desenhado no LAB4, mas agora construído com o *framework* **Symfony 4.4** e a *template engine* **Twig**.

## 0. Preliminares

**IMPORTANTE: VAMOS UTILIZAR O SERVIDOR daw2: IP 10.10.23.184**

Faça login no servidor daw2[1] (IP 10.10.23.184)

**Download  o framework para dentro da pasta "public_html/LAB8_10"**

```
a12345@daw2:~$ cd public_html

a12345@daw2:~/public_html$
git clone git://github.com/jmatbastos/LAB8_10.git
```

Complete a instalação com as livrarias standard.

```
a12345@daw2:~/public_html$ cd LAB8_10
a12345@daw2:~/public_html/LAB8_10$ composer install
```

**Faça uma cópia da base de dados que tem no servidor daw (IP 10.10.23.183) para este servidor (substitua "a12345" pelo seu login)** [2]

```
a12345@daw2:~$ mysqldump -ua12345 -p****** -h10.10.23.183 db_a12345 > myDB.SQL
a12345@daw2:~$ /usr/local/bin/mysql-db
a12345@daw2:~$ mysql -ua12345 -p****** db_a12345 < myDB.SQL
```

**Abra o ficheiro ".env" para utilizar as credenciais da sua base de dados.**

```
a12345@daw2:~/public_html/LAB8_10$ nano .env
```

atualize a linha

```
DATABASE_URL=mysql://a12345:******@127.0.0.1:3306/db_a12345?serverVersion=15.1
```

**Substitua "a12345" pelo seu login e "******" pela password de acesso à sua nova base de dados no servidor daw2 dada pelo comando** `mysql-db`

[1] O framework Symfony NÃO funciona no servidor daw (IP 10.10.23.183)
[2] Quer continuar a utilizar a base de dados que tem no servidor daw? Neste caso a linha é
`DATABASE_URL=mysql://a12345:******@10.10.23.183:3306/db_a12345`

### Crie o controlador “BlogController”

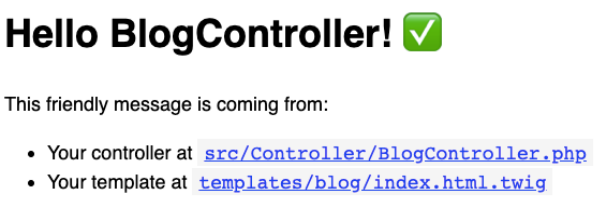
```
a12345@daw2:~/public_html/LAB8_10$ php bin/console make:controller
Choose a name for your controller class (e.g. GrumpyChefController):
 > Blog
```

### Teste a instalação do framework.

A partir do seu browser preferido vá ao seguinte URL

**`http://daw.deei.fct.ualg.pt/~a12345/LAB8_10/public/index.php/blog`**

Deverá receber uma página web de boas-vindas




---

### OBSERVAÇÕES

- Sempre que receber uma mensagem de erro informando o acesso negado à pasta "**`LAB8_10/var/cache/dev`** ",

Warning: file_put_contents(/users/a12345/public_html/lixo/var/cache/dev/srcApp_KernelDevDebugContainerDeprecations.log): failed to open stream: Operation not permitted

terá que limpar o "cache" com o comando

```
a12345@daw2:~/public_html/LAB8_10$ rm -rf var/cache
```

OU (em alternativa) com o comando

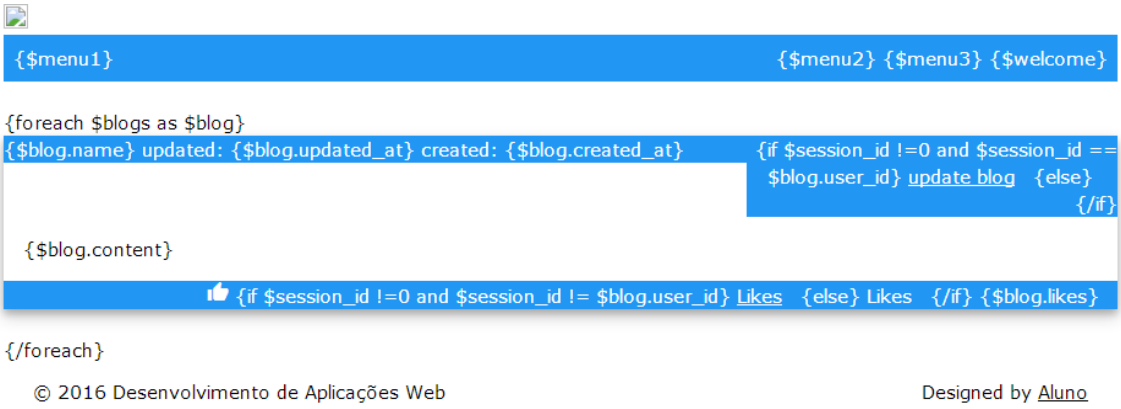
```
a12345@daw2:~/public_html/LAB8_10$ php bin/console cache:clear
```

- Se instalar um componente Symfony utilizando o "composer", poderá ter (nem sempre é necessário) que refazer novamente as permissões na pasta "vendor" com os comandos

```
a12345@daw2:~/public_html$ find LAB8_10/vendor  -type d -exec chmod 755 {} \;
a12345@daw2:~/public_html$ find LAB8_10/vendor  -type f -exec chmod 644 {} \;
```

# 1. Desenho do template index_template.html.twig

Adapte o template `index_template.tpl` para o template `index_template.html.twig`




que vai ser utilizado para construir a página de rosto do site.

O template **`index_template.html.twig`** deve ser colocado na pasta

`public_html/LAB8_10/templates/blog/`

**NOTA:** os recursos utilizados no template (imagens, css, javascript) devem ser colocados em pastas (images, css, js, etc) dentro da pasta "`public`"

# 2. Construa o controlador `BlogController.php` na pasta

`public_html/LAB8_10/src/Controller`

```
<?php

namespace App\Controller;

use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Session\SessionInterface;
use Symfony\Component\PropertyAccess\PropertyAccess;
use Symfony\Component\Validator\Constraints as Assert;
use Symfony\Component\Validator\Validator\ValidatorInterface;
use Symfony\Component\Validator\Context\ExecutionContextInterface;
use App\Controller\Blog_modelController;

class BlogController extends AbstractController
{

      private $blog_model;
```

```
      private $session;
      private $validator;


      public function __construct(Blog_modelController $blog_model, SessionInterface
$session, ValidatorInterface $validator)
    {
        $this->blog_model = $blog_model;
        $this->session = $session;
        $this->validator = $validator;
    }
```

O código PHP do controlador responsável pela página de rosto deverá encontrar-se na função `index()`

```
   /**
    * @Route("/blog", name="blog")
    */
    public function index()
    {

    }
```

# 3. Construção da classe responsável pelo acesso à base de dados.

Construa o ficheiro `Blog_modelController.php` na pasta

`public_html/LAB8_10/src/Controller`

```
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Doctrine\DBAL\Driver\Connection;


class Blog_modelController extends AbstractController
{
        private $connection;


    public function __construct(Connection $connection)
    {
        $this->connection = $connection;
    }
```

Construa a função `get_posts()` responsável pela query à base de dados

```
    public function get_posts()
    {

    }
```

## 4. Se realizou o lab no seu computador pessoal, faça o upload dos ficheiros

- `index_template.html.twig`
- `BlogController.php`
- `Blog_modelController.php`

para os directorios "`templates/blog`" e "`src/Controller`", respectivamente na pasta "LAB8_10" no seu site web pessoal

Teste o funcionamento do site no url

http://daw.deei.fct.ualg.pt/~a12345/LAB8_10/public/index.php/blog

Considere o lab concluído quando obtiver a mesma funcionalidade que foi requerida no LAB4

IMPORTANTE

- Os recursos locais devem ter URLs relativos: utilize as funções `asset()` e `path()` do Twig!

REFERÊNCIAS:

- http://daw.deei.fct.ualg.pt/~a999990/SF_exame2/post
- https://symfony.com/doc/4.4//index.html
  http://intranet.deei.fct.ualg.pt/DAW/slides/SF_overview.pdf
- http://all.deei.fct.ualg.pt/symfony/
- https://twig.symfony.com/doc/3.x/

# ANEXO 1. Estrutura da base de dados

A estrutura da base de dados pode ser consultada em
http://daw.deei.fct.ualg.pt/phpMyAdmin

```
CREATE TABLE `users` (
  `id` int(11) NOT NULL auto_increment,
  `name` varchar(255) default NULL,
  `email` varchar(255) default NULL,
  `created_at` datetime NOT NULL,
  `updated_at` datetime NOT NULL,
  `password_digest` varchar(255) default NULL,
  `remember_digest` varchar(255) default NULL,
  `admin` tinyint(1) default NULL,
  `activation_digest` varchar(255) default NULL,
  `activated` tinyint(1) default NULL,
  `activated_at` datetime default NULL,
  `reset_digest` varchar(255) default NULL,
  `reset_sent_at` datetime default NULL,
  PRIMARY KEY  (`id`),
  UNIQUE KEY `index_users_on_email` (`email`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;


CREATE TABLE `microposts` (
  `id` int(11) NOT NULL auto_increment,
  `content` text,
  `user_id` int(11) default NULL,
  `created_at` datetime NOT NULL,
  `updated_at` datetime NOT NULL,
  `likes` int(11) NOT NULL DEFAULT 0,
  PRIMARY KEY  (`id`),
  CONSTRAINT FOREIGN KEY (`user_id`) REFERENCES `users`
(`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

NOTA:

- Se quiser fazer o site utilizando o componente **Doctrine ORM**, pode gerar uma classe para cada tabela com os comandos

```
php bin/console doctrine:mapping:import "App\Entity" annotation --path=src/Entity
php bin/console make:entity --regenerate App
```

- A tabela "users" em Apêndice tem que ser alterada adicionando a coluna "roles"

```
ALTER TABLE users ADD `roles` longtext COMMENT
'(DC2Type:json)';
```

Referência
https://symfony.com/doc/current/doctrine/reverse_engineering.html