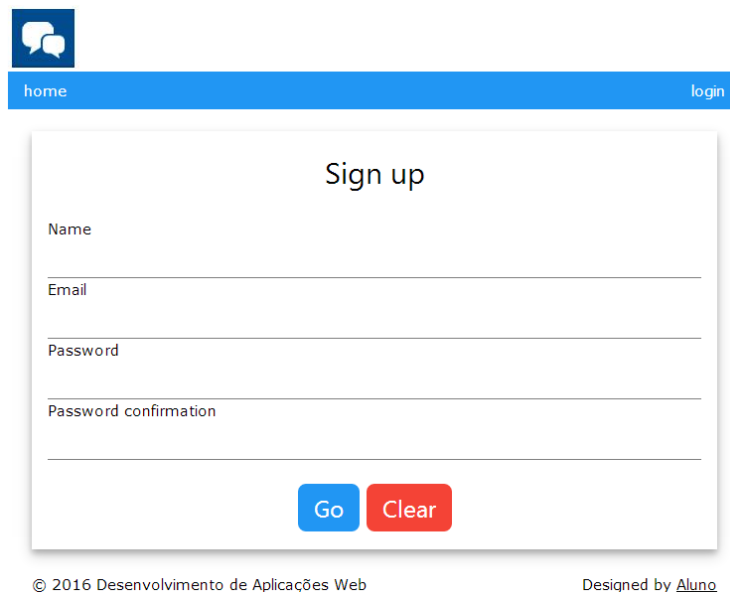


LAB 10 – Programação com o framework Symfony [parte 2]

O objectivo deste laboratório é repetir a funcionalidade do site desenhado nos LAB5, LAB6, e LAB7¹ mas agora construído com o framework **Symfony** 4.4 e a *template engine Twig*.

Assume-se aqui que já realizou com sucesso o LAB9.

1. Construa o template `register_template.html.twig` adaptando o template `register_template.tpl` realizado no LAB5



Sign up

Name

Email

Password

Password confirmation

Go Clear

© 2016 Desenvolvimento de Aplicações Web

Designed by [Aluno](#)

O template deve ser colocado na pasta

`public_html/LAB9_10/templates/blog`

O código PHP do controlador responsável pela página de registo deverá encontrar-se na função `register()` e `register_action()` dentro do ficheiro que define a classe do controlador

`(public_html/LAB9_10/src/Controller/BlogController.php)`

```
/**
 * @Route("/blog/register", name="register")
 */
```

¹ A funcionalidade do LAB8 é opcional

```

public function register()
{

}

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

}

```

Adicione o código necessário para:

- O controlador da acção valida os dados introduzidos. Em caso de insucesso o formulário é retornado com os campos correctos já preenchidos
- Em caso de sucesso
 - o controlador da acção utiliza a função `register_user($username, $email, $password)` para actualizar a base de dados
 - o controlador da acção redirecciona para o controlador `message()` e este actualiza o template `message_template.html.twig` com a mensagem "Registration successful. Welcome {{user}}!", colocando na variável Twig o nome do utilizador que se registou no site com sucesso.

O template `message_template.html.twig` contém uma meta-tag que redirecciona automaticamente para `/blog` passados 5 segundos:

```
<meta http-equiv="refresh" content="5; url= {{ path(blog) }} " />
```

2. O controlador trabalha em colaboração com a classe responsável pelo acesso à base de dados

(`public_html/LAB9_10/src/Controller/Blog_modelController.php`).

Construa a função `register_user($username, $email, $password)` responsável pela query à base de dados

```

public function register_user($username, $email, $password)
{

```

```
}
```

3. Utilizando o template `login_template.html.twig` pretende-se construir a página de login no site. Adapte o template `login_template.tpl` realizado no LAB6.



home register

Login

Email

Password

Go

© 2016 Desenvolvimento de Aplicações Web Designed by [Aluno](#)

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

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

4. O código PHP do controlador responsável pela acção de login deverá encontrar-se na função `login_action()`

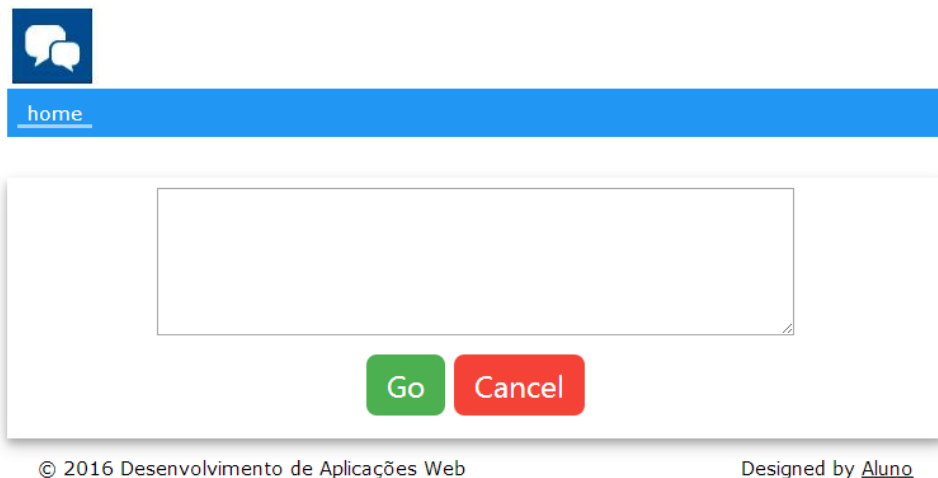
```
/**
 * @Route("/blog/login_action", name="login_action")
 */

public function login_action()
{
}
```

Adicione o código necessário para:

- Em caso de sucesso guardar no array de sessão o id, nome e email do utilizador validado
- Em caso de sucesso carregar o template 'message_template.html.twig' com a mensagem "Welcome back {{user}}!")
- Em caso de insucesso o formulário de login é retornado com a mensagem de erro "Login failed: wrong email or password". Utilize uma variável de sessão para passar esta informação entre os dois controladores.

5. Utilizando o template `blog_template.html.twig` pretende-se construir a página de criar/alterar blog no site. Adapte o template `blog_template.tpl` realizado no LAB7.



home

Go Cancel

© 2016 Desenvolvimento de Aplicações Web Designed by [Aluno](#)

O código PHP do controlador responsável pela página de registo (ou actualização) de um "post" deverá encontrar-se na função `post($blog_id)` e o controlador da acção na função `post_action($blog_id = FALSE)`

```
/**
 * @Route("/blog/post/{blog_id?}", name="post")
 */
public function post($blog_id = FALSE)
{
}

/**
 * @Route("/blog/post_action/{blog_id?}", name="post_action")
 */
public function post_action($blog_id = FALSE)
{
}
```

Adicione o código necessário para:

- O controlador utiliza a função `new_blog($user_id, $blog)` para inserir na base de dados o novo post
- O controlador utiliza as funções `get_blog($user_id, $blog_id)` e `update_blog($user_id, $blog_id, $content)` para actualizar a base de dados com o texto actualizado do post
- O controlador redirecciona para a página principal do site (“blog”)

NOTA: o seu código tem que ser robusto contra tentativas de inserir posts em nome de outro utilizador, ou de alterar posts que não pertencem ao utilizador!

6. Construa as funções `new_blog()`, `get_blog()`, `update_blog()`, responsáveis pelas queries correspondentes à base de dados em “src/Controller/Blog_modelController.php”

7. Construa a função que destroi a sessão quando o utilizador faz logout. O controlador carrega o template ‘message_template’ (sugere-se “See you back soon!”), e que ao fim de 5 segundos redirecciona para a página de rosto)

```
/**
 * @Route("/blog/logout", name=" logout")
 */

public function logout()
{
    $this->session->set('userid', '');
    $this->session->set('username', '');
    return $this->redirectToRoute('blog');
}
```

8. Actualize a função `login_action()` para, em caso de sucesso no login e a “checkbox” “remember_me” tiver sido seleccionada, implementar a funcionalidade

- enviando um cookie para o browser com os dados

```
$cookie_name = 'siteAuth';
$cookie_time = (60 * 24 * 30); // 30 days
$remember_digest = substr(md5(time()),0,32);
```

- actualizando o campo “remember_digest” na tabela “users” através da função `set_remember_digest()` em “src/Controller/Blog_modelController.php”

```
public function set_remember_digest($email,$remember_digest)
{
}
```

Em “src/Controller/BlogController.php” atualize a função `index()` para, se receber um cookie “siteAuth” verificar se o seu valor existe na tabela “users” e em caso de sucesso validar imediatamente o utilizador.

A função que verifica se o conteúdo do cookie existe na tabela “users” é

```
public function check_remember_digest($remember_digest)
{
}
```

em “src/Controller/Blog_modelController.php”

9. Teste o funcionamento do site no URL *

http://daw.deei.fct.ualg.pt/~a12345/LAB9_10/public/index.php/blog

Considere o lab concluído quando obtiver a mesma funcionalidade que foi requerida nos LAB5, LAB6, e LAB7.

IMPORTANTE

- Os recursos locais devem ter URLs relativos: utilize as funções `asset()` e `path()` do Twig!
- Utilize o componente Symfony “Validator” para fazer a validação dos dados introduzidos nos formulários

REFERÊNCIAS:

- http://daw.deei.fct.ualg.pt/~a999990/SF_exam2/public/blog
- <https://symfony.com/doc/4.4/index.html>
- <https://symfonycasts.com/screencast/symfony4>
- http://intranet.deei.fct.ualg.pt/DAW/slides/SF_overview.pdf
- <http://all.deei.fct.ualg.pt/symfony/>

* NOTA: o servidor <http://all.deei.fct.ualg.pt> NÃO corre código Symfony

ANEXO 1 . Estrutura da base de dados

A estrutura da base de dados pode ser consultada em

<http://all.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,  
  PRIMARY KEY (`id`),  
  CONSTRAINT FOREIGN KEY (`user_id`) REFERENCES `users`  
  (`id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```