LAB 12 – Programação em JavaScript, JQuery, e AJAX

Assume-se aqui que já realizou com sucesso o LAB10.

FUNCIONALIDADE REPLY TO POST

1. Adicione a tabela "`replies`" à sua base de dados

```
a12345@daw2:~$mysql -u a12345 -p -h 10.10.23.183 db_a12345
mysql> CREATE TABLE `replies` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `content` text CHARACTER SET utf8 COLLATE utf8_bin,
  `user_id` int(11) DEFAULT NULL,
  `micropost_id` int(11) DEFAULT NULL,
  `created_at` datetime NOT NULL,
  PRIMARY KEY (`id`),
  CONSTRAINT FOREIGN KEY (`user_id`) REFERENCES `users` (`id`),
  CONSTRAINT FOREIGN KEY (`micropost_id`) REFERENCES `microposts` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

Nota: se desejar pode realizar o comando utilizando a interface web em

http://daw.deei.fct.ualg.pt/phpMyAdmin

2. Actualize o template `index_template.html.twig` que foi realizado no LAB8, de forma a ter

1. o texto "`update post`" se o utilizador que fez *log in* é o autor do post (já existente)

```
<a href="{{path('post')}}{{$blog.id}}">update post</a>
```

2. o texto "`reply to post`" se o post *não pertence* ao utilizador (novidade)

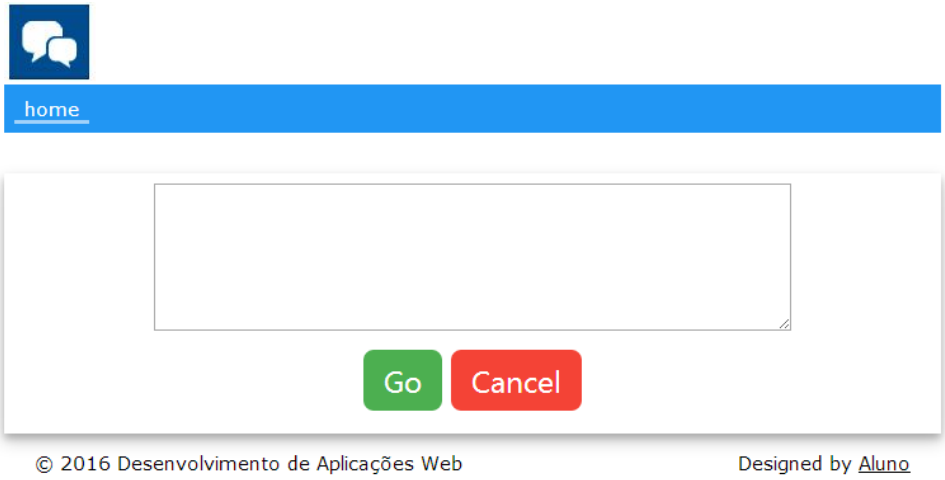
```
<a href="{{path('post')}}{{$blog.id}}">reply to post</a>
```

Actualize o método `index()` no ficheiro que define a classe do controlador (`public_html/LAB8_10/src/Controller/BlogController.php`)

de forma a garantir que em ambos os casos `href` seja o mesmo:

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

O template associado ao método "post" é `blog_template.html.twig` (já existente) mostra-se aqui novamente:



Utilize as funções `asset()` ou `path()` para garantir que o seu site é portável.

3. Actualize o método "`post_action`" dentro do ficheiro que define a classe do controlador (`public_html/LAB8_10/src/Controller/BlogController.php`) Para:

- Chamar o método `Blog_modelController::new_blog()` caso nenhum `blog_id` seja transmitido no URL (já existente)

- Chamar o método `Blog_modelController::update_blog($blog_id)` caso o utilizador que fez login seja o autor do post (já existente)
- Chamar o método `Blog_modelController::new_reply($blog_id)` caso o utilizador que fez login *não seja* o autor do post (**novidade**)

4. Construa o método `new_reply($blog_id)` dentro do ficheiro que define os métodos de acesso à base de dados
   (`public_html/LAB8_10/src/Controller/Blog_modelController.php`)

```
public function new_reply($user_id, $blog_id, $reply)
{

}
```

5. Construa o método `get_replies($id)` em `Blog_modelController.php`

```
public function get_replies($id)
{

}
```

Utilize a nova funcionalidade do site para introduzir "comentários" em posts de outros utilizadores (em alternativa coloque manualmente algumas "comentários" acedendo directamente à base de dados)

6. Actualize novamente o template `index_template.html.twig` e o método `index()` para que em todos os posts em que tenha havido "replies" estas apareçam imediatamente a seguir ao post respectivo:

Esta funcionalidade exige a realização de dois "nested loops": um loop exterior ("posts"), e um loop interior ("replies") para receber os dados enviados nos arrays `$blogs` e `$replies`

O resultado deve ser o seguinte screenshot (ou equivalente):

FUNCIONALIDADE SHOW/HIDE REPLIES COM JAVASCRIPT/JQUERY

7. Actualize novamente o template `index_template.html.twig` e introduza um “botão” no template

```
<button id="b{{$blog.id}}" onclick="myToggle('{{$blog.id}}')">Show</button>
```

e uma divisão “escondida” que contem as “replies”...

```
<div id="d{{$blog.id}}" style="display:none;">


</div>
```

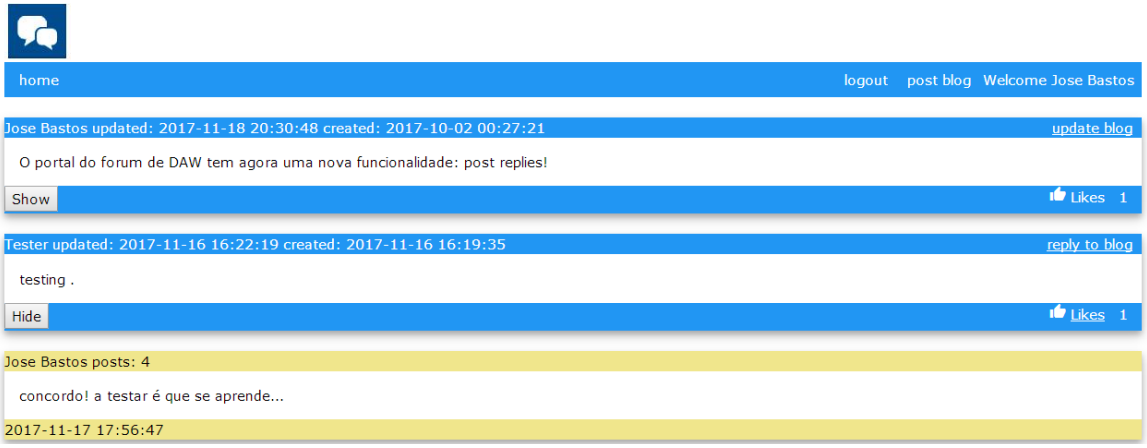
Utilizando JavaScript ou a livraria JQuery construa a função `myToggle(id)`

```
<script>
function myToggle(id) {
    .
    .
    .
    }
</script>
```

que alternando clicar no botão mostra ou esconde as “replies” de um determinado “post”, como se mostra em seguida

- “Replies” escondidas:

- “Replies” visíveis



Teste o funcionamento do botão. Considere esta secção do laboratório concluída quando obtiver a mesma funcionalidade do site

http://daw.deei.fct.ualg.pt/~a999993/LV_exame2/blog

FUNCIONALIDADE SHOW/HIDE REPLIES COM AJAX

8. Faça um backup dos ficheiros `BlogController.php` e `index_template.blade.php`

9. No método `index()` **remova** a funcionalidade de acesso à tabela "queries"

10. No template `index_template.html.twig` **remova** o loop interior

11. Construa o método `replies($id)` em `BlogController.php`

```
/**
 * @Route("/blog/replies/{blog_id?}", name="replies")
 */
public function replies($blog_id){
 $replies = json_encode($this->blog_model->get_replies($blog_id));
 echo $replies;
}
```

12. Utilizando JavaScript ou a livraria JQuery construa a função `myToggle(id)`

```
<script>
        function myToggle(id) {
    .
    .
    .
    }
 </script>
```

Esta função é uma função **AJAX** que recebe dados no formato **JSON**:

Se o botão de um determinado post disser "Show"

- O botão passa a dizer "Hide"
- É feito um GET ao URL "blog/replies/blog_id". Nota: este GET é apenas feito uma vez para um determinado blog_id!
- A que se encontra escondida (style="display:none;") passa a ser visível (style="display:block;") e o seu conteúdo é preenchido com os dados da resposta ao GET.

Se o botão de um determinado post disser "Hide"

- O botão passa a dizer "Show"

- A que se encontra visível (style="display:block;") passa a estar escondida (style="display:none;")

Teste o funcionamento do botão. Considere o laboratório concluído quando obtiver a mesma funcionalidade do site

http://daw.deei.fct.ualg.pt/~a999993/LV_exame2/blogJS

http://daw.deei.fct.ualg.pt/~a999993/LV_exame2/blogJQ

http://daw.deei.fct.ualg.pt/~a999993/LV_exame2/blogNG

REFERÊNCIAS:

- https://www.w3schools.com/js/default.asp
- https://www.w3schools.com/jquery/default.asp
- https://www.w3schools.com/js/js_ajax_intro.asp
- https://www.w3schools.com/jquery/jquery_ajax_intro.asp

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,
  PRIMARY KEY  (`id`),
  CONSTRAINT FOREIGN KEY (`user_id`) REFERENCES `users` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `replies` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `content` text CHARACTER SET utf8 COLLATE utf8_bin,
  `user_id` int(11) DEFAULT NULL,
  `micropost_id` int(11) DEFAULT NULL,
  `created_at` datetime NOT NULL,
  PRIMARY KEY (`id`),
  CONSTRAINT FOREIGN KEY (`user_id`) REFERENCES `users`
(`id`),
  CONSTRAINT FOREIGN KEY (`micropost_id`) REFERENCES
`microposts` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```