Curso PHP

Do XLSX ao CMS (aula 16)

Autenticação

Autenticação HTTP

O HTTP fornece uma estrutura geral para controle de acesso e autenticação.

A autenticação HTTP mais comum é fundamentada no esquema "Basic".

A RFC 7235 define a estrutura de autenticação HTTP que pode ser usada por um servidor para definir uma solicitação ("challenge") do cliente e para um cliente fornecer informações de autenticação.

(MDN)

Autenticação HTTP com PHP

Pode-se utilizar a função **header()** para enviar uma mensagem de *Authentication Required* para o navegador cliente, causando o aparecimento de uma janela para a entrada de Nome de Usuário/Senha.

Uma vez que o usuário preencha um nome de usuário e uma senha, a URL contendo o script PHP será chamada mais uma vez com as variáveis predefinidas **PHP_AUTH_USER**, **PHP_AUTH_PW**, e **AUTH_TYPE** para determinar o nome de usuário, senha e tipo da autenticação, respectivamente.

Estas variáveis predefinidas são encontradas no array **$_SERVER**.

Somente o métodos de autenticação *Basic* é suportado.

(php.net)

Autenticação baseada em sessões

A autenticação baseada em sessão envolve a criação e manutenção de uma sessão para cada usuário após o login bem-sucedido.

As sessões PHP armazenam os dados do usuário no servidor e emitem um ID de sessão para o cliente (geralmente armazenado em um cookie).

Este método é amplamente utilizado por sua simplicidade e armazenamento do lado do servidor dos dados da sessão.

(Adnan taşdemir)

Testes

Definindo testes

Uma vez que definimos nossas áreas restritas, precisamos realizar o processo de login antes de acessar esta área de fato.

Antes de cada teste, iremos recriar a base de dados e realizar login.

Isto não irá afetar o comportamento dos testes, pois como usamos o modo cookies habilitados e também *singleton*, as demais requisições irão reusar a mesma sessão gerada durante o login.

```
beforeEach(function () {
    refreshDatabase();

    $senha = faker()->password(8);
    $email = Usuario::factory()->create(compact('senha'))->email;
    $_csrf_token = Http::getCsrfToken('/login');

    Http::post('/login', [
        'form_params' => compact('senha', 'email', '_csrf_token'),
    ]);
});
```

Mostra/esconde

Baseado nos casos de uso, o sistema deve permitir que os atores “visitante” e “administrador” tenham comportamentos exclusivos.

Como a nomenclatura sugere, o administrador irá gerenciar o sistema, realizando as ações de CRUD.

Para isso, precisamos saber se foi definido um valor na sessão para o usuário atual.

Com isto, podemos mostrar ou ocultar informações para os visitantes (usuário não conectados).

```php
class ConfigureView
{
    public function __invoke(Request $request): bool
    {
        View::addFunction('attr', attr(...));
        View::addFunction('url', url(...));
        View::addGlobals([
            'APP_LOCALE' => str_replace('_', '-', env('APP_LOCALE')),
            'CSRF_TOKEN' => session()->get('csrf_token'),
            'CURRENT_URI' => $request->getPathInfo(),
            'ERRORS' => flash()->get('err'),
            'OLD' => flash()->get('old'),
            'USUARIO' => session()->get('usuario'),
        ]);

        return true;
    }
}
```

```twig
{% if (USUARIO) %}
<a href="{{ url('/agendamentos/{id}/editar', {id: agendamento.id}) }}">
    <small>&#9998; Editar</small>
</a>
{% endif %}
```

Banco de dados

Model

Precisamos de um model para gerenciar nossos usuários.

Observe que neste model temos um **auto relacionamento**.

Essa relação permite que usuários dependam de outros. Isto é, um usuário pode ter mais poder de acesso que outro, que será mais explorado quando entrarmos no **tópico ACL**.

```php
DB::schema()->create('usuarios', function (Blueprint $table) {
    $table->id();
    $table->foreignIdFor(Usuario::class)->nullable();
    $table->string('nome');
    $table->string('email')->unique();
    $table->string('senha');
    $table->timestamps();
});
```

Requisição

Attempted URI

Uma vez que o usuário tente acessar uma área restrita, precisamos interceptar isto, envia-lo para o login e, finalmente, depois de devidamente autenticado, permitir que ele volte a página inicialmente requerida.

Para isto aplicamos o conceito *attempted URI*. Este conceito simplesmente salvar a URI requisitada pelo visitante na sessão antes de o redirecionar para a página de login.

```php
class AcessoRestrito
{
    public function __invoke(Request $request) : bool|RedirectResponse
    {
        if (session()->get('usuario')) {
            return true;
        }

        session()->set('attempted_uri', $request->getRequestUri());

        return redirect('/login');
    }
}
```

Middlewares

```php
// página inicial
Router::redirect('/', '/agendamentos');

// autenticação
Router::get('/login', [AuthController::class, 'index'], [Visitante::class]);
Router::post('/login', [AuthController::class, 'login'], [Visitante::class]);
Router::get('/logout', [AuthController::class, 'logout']);

// usuários
Router::get('/usuarios', [UsuarioController::class, 'index'], [AcessoRestrito::class]);
Router::get('/usuarios/cadastrar', [UsuarioController::class, 'cadastrar'], [Visitante::class]);
Router::post('/usuarios', [UsuarioController::class, 'salvar'], [Visitante::class]);
Router::get('/usuarios/editar', [UsuarioController::class, 'editar'], [AcessoRestrito::class]);
Router::put('/usuarios/editar', [UsuarioController::class, 'atualizar'], [AcessoRestrito::class]);
```

Aplicando middlewares às rotas

AuthController

Login

Após a validação do envio do formulário de login, verificamos se o usuário está efetivamente apto a acessar o sistema.

Para simplificar, toda a regra de autenticação foi atribuída ao método **Auth::attempt**.

```
public function login(Request $request): RedirectResponse
{
    if (
        $request->validate(Usuario::rules(), ['email', 'senha']) &&
        Auth::attempt($request->validated)
    ) {
        return redirect($request->attemptedUri('/agendamentos'));
    }

    return redirect('/login');
}
```

```php
class Auth
{
    public static function attempt(array $validated): ?Usuario
    {
        $usuario = Usuario::firstWhere('email', $validated['email']);
        $senhaCorreta = $usuario && password_verify($validated['senha'], $usuario->senha);

        if (! $senhaCorreta) {
            flash()->set('err', [
                'email' => 'Email/senha incorretos',
            ]);

            return null;
        }

        if (password_needs_rehash($usuario->senha, constant(env('PASSWORD_ALGO')))) {
            $usuario->forceFill($validated)->save();
            $usuario->refresh();
        }

        session()->set('usuario', $usuario);
        session()->migrate(true);

        return $usuario;
    }
}
```

Auth:attempt

Logout

O processo de logout é relativamente simples.

Consiste somente em invalidar a sessão, isto é, enviar para o navegador uma instrução de “cookie expirado”, forçando que seja gerado um novo cookie de sessão na próxima requisição.

```php
public function logout(): RedirectResponse
{
    session()->invalidate();

    return redirect('/login');
}
```

Aprimorando a UI

Menu dinâmico

Spatie Menu

O pacote de **spatie/menu** fornece uma interface fluente para criar menus de qualquer tamanho em seu aplicativo PHP.

Todas as classes fornecem uma interface legível e fluente para humanos (sem configuração de matriz).

Além disso, você pode optar por uma sintaxe mais verde e flexível ou por métodos de conveniência que cobrem a maioria dos casos de uso.

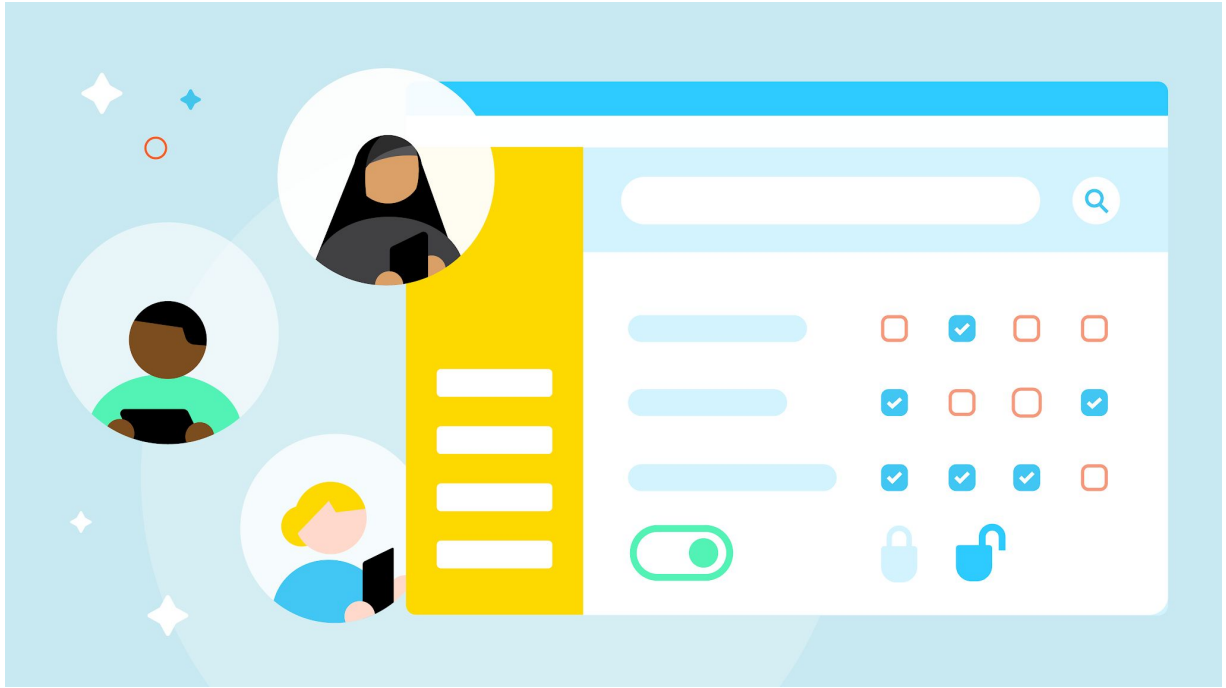
(Spatie Menu docs)

composer install spatie/menu

Instala a dependência

```php
Menu::new()
    ->link('/agendamentos', 'Agendamentos')

    // Se o usuário estiver autenticado, adiciona links específicos
    ->if(! ! $usuario, fn (Menu $menu) => $menu
        // ...
    )

    // Se o usuário NÃO estiver autenticado, exibe apenas opções limitadas
    ->if(! $usuario, fn (Menu $menu) => $menu
        // ...
    )

    // Define como ativo o link correspondente à URI atual
    ->setActive($currentUri)

    // Adiciona automaticamente uma classe CSS "active" ao link ativo
    ->setActiveClassOnLink()

    // Adiciona atributos ao link ativo
    ->each(fn (Link $link) => $link->setAttribute('aria-current', $link->isActive() ? 'true' : 'false'))

    // Renderiza o menu como HTML
    ->render();
```

Implementação do menu dinâmico



Próxima parada: estação ACL