## Autenticação

1. Para criar as rotas e controllers relacionados ao login, execute o comando
php artisan make:auth

**Utilizando um model customizado**
1. Adicione os campos “password” e “remember_token” ao migration do seu model, e torne único (→unique()) o atributo que fará o papel do login:

```
        Schema::create('funcionarios', function (Blueprint $table) {
            $table->increments('id');
            $table->string('nome');
            $table->string('apelido')->unique();
            $table->string('password');
            $table->rememberToken();
```

2. Modifique o arquivo /config/auth.php para informar o model que deverá fazer o papel do usuário:

```
    'providers' => [
        'users' => [
            'driver' => 'eloquent',
            'model' => OrgTabajara\Funcionario::class,
        ],
```

3. Modifique o formulário de login (resources/view/auth/login.blade.php) modificando o nome do campo do e-mail para o atributo escolhido.

4. Modifique o Model

```
<?php

namespace OrgTabajara;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Foundation\Auth\User as Authenticatable;

class Funcionario extends Authenticatable
{
    protected $fillable = ['nome', 'apelido', 'password'];
    protected $hidden = ['password', 'remember_token'];
```

## Criar as Views

1. Blade Template
2. Location

**Autorização**

**Bloqueando acesso para usuários não logados**

1. Na rota

```
Route::get('funcionario/create', 'FuncionarioController@create')
->middleware('auth');
```

2. Na rota, agrupando rotas

```
Route::middleware('auth')->group(function() {
   Route::get('funcionario/create', 'FuncionarioController@create');
   Route::post('funcionario/store', 'FuncionarioController@store');
});
```

3. No controller

```
   public function __construct(Funcionario $funcionario) {
      $this->funcionario = $funcionario;
      $this->middleware('auth');
   }
```

4. No controller, restringindo os métodos

```
   public function __construct(Funcionario $funcionario) {
      $this->funcionario = $funcionario;
      $this->middleware('auth', ['only' => ['create', 'store']]);
   }
```

**O middleware auth verifica apenas se o usuário está logado ou não. Para regras mais complexas, podemos utilizar um middleware customizado ou definir políticas de acesso**

**Autorização com um middleware customizado:**

1. Crie o middleware

php artisan make:middleware AutorizacaoMiddleware

2. Edite o método handle (o arquivo do middleware fica no diretório /app/Http/Middleware

```
    public function handle($request, Closure $next)
    {
      //$request->route()->uri <- retorna a rota
      if($request->is("funcionario/create")) {
         if(\Auth::guest() || \Auth::user()->departamento->descricao != "rh")
            return redirect("login");
      }

      return $next($request);
    }
```

3. Em seguida, é necessário registrar este middleware no kernel (/app/Http/Kernel.php)

```
protected $routeMiddleware = [
    'auth' => \Illuminate\Auth\Middleware\Authenticate::class,
    'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
    'bindings' => \Illuminate\Routing\Middleware\SubstituteBindings::class,
    'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,
    'can' => \Illuminate\Auth\Middleware\Authorize::class,
    'guest' => \OrgTabajara\Http\Middleware\RedirectIfAuthenticated::class,
    'signed' => \Illuminate\Routing\Middleware\ValidateSignature::class,
    'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
    'autorizacao' => \OrgTabajara\Http\Middleware\AutorizacaoMiddleware::class,
];
```

4. Por fim, associe o middleware customizado (‘autorização’) na rota ou no controller

**Autorização com Políticas**

1. Crie o arquivo de Política
php artisan make:policy FuncionarioPolicy --model Funcionario

2. Edite o arquivo, que fica localizado no diretório /app/Policies

```
    public function create(Funcionario $user)
    {
        return $user->departamento->descricao == "rh";
    }
```

3. Registre o arquivo de política, adicionando a classe ao arquivo
app/Providers/AuthServiceProvider.php

4. Acrescente a chamada da autorização via políticas no action do controller (ou na view)

```
  public function create() {
     $this->authorize('create', \OrgTabajara\Funcionario::class);

     $departamentos = \OrgTabajara\Departamento::all();
     return view('funcionario.cadastrar', [
        'departamentos' => $departamentos
          ]);
  }
```

Se o usuário logado (ou guest) não tiver autorização para realizar a ação, será levantada uma exceção do tipo Symfony\Component\ HttpKernel\Exception\AccessDeniedHttpException, que deve ser tratada da formam convencional.

**Obs: Também é possível autorizar na View:**

```
@can('create', \OrgTabajara\Funcionario::class)
<a class="dropdown-item" href="/funcionario/create">
    {{ __('funcionario.cadastrar') }}
</a>
@endcan
```