

# BANCO DE DADOS COM LARAVEL (PARTE 3)



**Douglas Nassif Roma Junior**

 /douglasjunior

 /in/douglasjunior

 douglasjunior.me

 nassifrroma@gmail.com

Slides: <https://git.io/vAd6S>

# AGENDA

- Introdução a segurança com Laravel.
- Criando o modelo
- Preparando o controller
  - Create
  - Login
  - User
  - Logout
- Json Web Token (JWT)
- Relacionamento entre tabelas

# INTRODUÇÃO

- O Laravel possui um mecanismo integrado de autenticação, que facilita o controle de usuário e regras de acesso para a maioria das aplicações.
  - Autenticação Web: <https://laravel.com/docs/5.6/authentication>
  - Autenticação de API: <https://laravel.com/docs/5.6/passport>
- Porém a **Autenticação de API** fornecida pelo Laravel é complexa e tem o objetivo de trabalhar com aplicações que necessitam ser autenticadas por terceiros. Em muitas vezes, você precisa autenticar apenas os clientes da sua própria aplicação.

# INTRODUÇÃO

- Utilizando o conteúdo visto até agora, podemos criar um mecanismo de autenticação simples, seguro e que se adapte a realidade da aplicação em questão.
- **Controllers:** Irão tratar das requisições de Cadastro, Login, Logout de usuários.
- **Validators:** Auxiliam na filtragem dos dados recebidos no cadastro e login.
- **Middlewares:** Protegem as rotas que exigem autenticação.

# CRIANDO O MODELO

- O primeiro passo é criar o modelo que irá representar o Usuário autenticado.

- Para isso utilize o comando:

```
php artisan make:model Usuario
```

- Caso seja preciso, passe o parâmetro `--migration` para criar o arquivo de migração.

# CRIANDO O MODELO

- Modifique seu modelo de Usuário para se adaptar a tabela de usuários de sua aplicação:

```
<?php

class Usuario extends Authenticatable
{
    use Notifiable;

    protected $fillable = [
        'nome', 'email', 'senha',
    ];

    protected $hidden = [
        'senha', 'remember_token',
    ];

    public function getAuthPassword() {
        return $this->senha;
    }
}
```

- Perceba que o modelo do usuário deve herdar da classe **User**, que faz com que este modelo representa uma entidade de autenticação para o Laravel.

# CRIANDO O MODELO

- Personalize também o arquivo de migração:

```
class CreateUsuariosTable extends Migration
{
    public function up()
    {
        Schema::create('usuarios', function (Blueprint $table) {
            $table->increments('id');
            $table->timestamps();
            $table->string('nome');
            $table->string('email')->unique();
            $table->string('senha');
            $table->rememberToken();
        });
    }

    public function down()
    {
        Schema::dropIfExists('usuarios');
    }
}
```

# PREPARANDO O CONTROLLER

- Crie também o controller que irá tratar as rotas relacionadas ao Usuário, como cadastro, login e logout.

```
php artisan make:controller UsuarioController
```

- E então, defina as rotas necessárias no arquivo `routes/api.php`

```
Route::group(['prefix' => 'usuarios'], function () {  
  
    Route::post('/', 'UsuarioController@create');  
    Route::post('/login', 'UsuarioController@login');  
  
    Route::group(['middleware' => 'jwt.auth'], function () {  
        Route::get('/', 'UsuarioController@user');  
        Route::post('/logout', 'UsuarioController@logout');  
    });  
});
```



# IMPLEMENTANDO CREATE

- Implemente a função `UsuarioController.create` para validar o nome, email e senha do usuário e criar um novo registro no banco de dados.
- Para fazer o **hash** da senha, você pode utilizar a função `bcrypt`.
  - Saiba mais sobre **hash** de senhas em PHP:  
<http://php.net/manual/en/function.password-hash.php>

# IMPLEMENTANDO LOGIN

- Implemente a função `UsuarioController.login` para receber o email e senha do usuário, validar se as informações estão corretas no banco de dados e em seguida retornar um **Json Web Token (JWT)** como garantia de autenticação.
- Para trabalhar com JWT no Laravel você pode utilizar a biblioteca **jwt-auth**: <https://github.com/tymondesigns/jwt-auth>

# JSON WEB TOKEN

- JSON Web Token (JWT) é um padrão industrial aberto RFC 7519 para representar reivindicações de forma segura entre duas partes.
- O JWT é formado de três partes: **Header**, **Payload** e **Signature**.
- Exemplo:

**eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjE0NjQ5OTI0MD0.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ**

- Encoded

PASTE A TOKEN HERE

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoiYnRydWV9.TjVA950rM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ

Decoded

EDIT THE PAYLOAD AND SECRET (ONLY HS256 SUPPORTED)

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret
)
```

☐ secret base64 encoded

# JSON WEB TOKEN

- Para trabalhar com JWT no Laravel, basta instalar a dependência:

```
compose require tymon/jwt-auth
```

- Você pode conferir estes link para as instruções de instalação:
  - <https://arjunphp.com/laravel-5-6-rest-api-jwt-authentication/>
  - <https://appdividend.com/2018/02/24/laravel-jwt-authentication-tutorial/>

# IMPLEMENTANDO GET USER

- Implemente a função `UsuarioController.user` para retornar o usuário logado.
- Lembre-se que esta rota deve exigir autenticação, então é preciso enviar o cabeçalho http:
- `Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9...`

# IMPLEMENTANDO LOGOUT

- Implemente a função `UsuarioController.logout` para invalidar token do usuário.
- Esta rota também exige autenticação, então é preciso enviar o cabeçalho http:
- `Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9...`

# RELACIONAMENTO ENTRE TABELAS

- Em muitos casos suas tabelas do banco de dados precisam se relacionar umas com as outras.
- Como por exemplo a relação entre o Usuário e suas Tarefas.
- O Eloquent também facilita a criação e gerenciamento desses relacionamentos, e suporta diferentes tipos de relacionamentos:
  - One To One
  - One To Many
  - Many To Many
  - Has Many Through
  - Polymorphic Relations
  - Many To Many Polymorphic Relations



# RELACIONAMENTO ENTRE TABELAS

- Os relacionamentos do Eloquent são definidos como métodos em seus modelos (Model).
- Assim como como os modelos em si, os relacionamentos também suportam Query Builders.
- Por exemplo:

```
$usuario->tarefas()->where('ativa', 1)->get();
```

# USUÁRIOS TEM MUITAS TAREFAS

- Para criar uma relação entre usuários e tarefas, devemos definir uma ligação do tipo `hasMany` do modelo **Usuario** para **Tarefa**.

```
public function tarefas()  
{  
    return $this->hasMany(Tarefa::class, 'usuario_cod')  
        ->where('ativa', 1);  
}
```

# USUÁRIOS TEM MUITAS TAREFAS

- Também deve definir uma migração para que a chave estrangeira `usuario_cod` seja criada na tabela `tarefas`:

```
public function up() {  
    Schema::table('tarefas', function (Blueprint $table) {  
        $table->unsignedInteger('usuario_cod');  
        $table->foreign('usuario_cod')  
            ->references('cod')->on('usuarios');  
    });  
}  
public function down() {  
    Schema::table('tarefas', function (Blueprint $table) {  
        $table->dropForeign(['usuario_cod']);  
        $table->dropColumn('usuario_cod');  
    });  
}
```

# USUÁRIOS TEM MUITAS TAREFAS

- Desafio:
  - Agora, altere todas as rotas da classe **TarefaController** para trabalhar com o usuário logado.
  - Lembre-se de adicionar o middleware `jwt.auth` nas rotas de tarefas.

# DÚVIDAS?



**Douglas Nassif Roma Junior**

 /douglasjunior

 /in/douglasjunior

 douglasjunior.me

 nassifrroma@gmail.com

Slides: <https://git.io/vAd6S>