Manual

Para poder criar uma rotina no sistema será necessário no mínimo criar uma *migration* e um *model*. Se for necessária uma tela que fuja do padrão, uma *view* deverá ser criada também. Também deverão ser criados os registros de rotina e os registros de menu.

Para criar a rotina no seeder da rotina, basta referenciar o nome, o prefixo(nome do sistema só para acesso rápido), slug (AppHelper::getClassName faz o slug perfeitamente referenciando a classe). Classes de Model e Controller, além de referenciar o id do sistema.

```php
Rotina::create([
    'nome'          => 'Grupo',
    'prefixo'       => Sistema::ESTRUTURA,
    'slug'          => AppHelper::getClassName(Grupo::class, true),
    'model'         => \App\Models\Estrutura\Grupo::class,
    'controller'    => \App\Http\Controllers\Estrutura\CrudController::class,
    'sistema_id'    => Sistema::ESTRUTURA_ID
]);
```

Para criar o menu, no seeder da rotina, basta referenciar o nome, o ícone, a ordem em que irá aparecer, o id do sistema, a rota que o menu irá redirecionar, a ação executada, qual a rotina que o menu utiliza e qual o menu pai desse menu.

```php
$grupo = Menu::create([
    'nome' => 'Grupo',
    'icone' => 'fa-solid fa-screwdriver-wrench',
    'ordem' => 5,
    'sistema_id' => Sistema::ESTRUTURA_ID,
    'route' => Sistema::ESTRUTURA.'.'.AppHelper::getClassName(Grupo::class, true).'.index',
    'acao_id' => Acao::CONSULTAR,
    'rotina_id' => Rotina::where('slug', 'grupo')->first()->id,
    'pai_id' => $estrutura->id
]);
```

Outras features podem ser realizadas criando um controller que extenderá a classe de ControllerPadrao, criação de Seeder e até mesmo rotas personalizadas. Vale salientar que os métodos serão apresentados, porém a descrição bem detalhada constará nos próprios arquivos do sistema, logo acima dos métodos.

1. MIGRATION

Para criar uma migration é necessário executar o comando no terminal:

- ❖ php artisan make:migration CreateNomeTabelaTable

A nova *migrations* será criada no diretório database\migrations. Cada nome de arquivo de migração contém um timestamp que permite ao Laravel determinar a ordem das *migrations*.

Cada *migration* possui dois métodos: *up* e *down*. O primeiro é utilizado para criar a tabela e o segundo para dropá-la.

```php
public function up()
{
    Schema::create(Sistema::ESTRUTURA.'.sistemas', function (Blueprint $table) {
        $table->id();
        $table->string('nome', 255);
        $table->string('slug', 255);
        $table->boolean('ativo')->default(true);
        $table->smallInteger('ordem')->default(1);
        $table->timestamps();
    });
}
```

Para criar a tabela utiliza-se o método estático “create” da classe Schema. O primeiro parâmetro é o nome da tabela e o segundo parâmetro é uma função onde será adicionado os campos a tabela.

O nome da tabela por padrão é criado com: nome_schema.nome_tabela.

Para resetar a base de dados:

- ❖ php artisan migration:reset

1.1 MÉTODOS

- $table->id() // Para criar um campo como chave primária.
- $table->string('nome do campo', tamanho) // Para criar um campo Varchar.
- $table->boolean('nome do campo') // Para criar um campo booleano.
- $table->integer('nome do campo')
- $table->smallInteger('nome do campo')
- $table->bigInteger('nome do campo')
- $table->timestamp('nome do campo') // Para criar um campo timestamp
- $table->date('nome do campo') // Para criar um campo do tipo data
- $table->string('campo')->default('valor') // Para criar um campo com valor default.

  $table->timestamps() // para criar colunas de created_at e updated_at.

Para criar uma chave estrangeira:

- $table->foreign(nome_fk')->references('pk')->on('tabela_estrangeira');

1.2 DOCUMENTAÇÃO LARAVEL

https://laravel.com/docs/8.x/migrations#available-column-types

2. SEEDERS

Os seeders servem unicamente para popular a base de dados rapidamente, sem a necessidade de ter que criar novos registros manualmente sempre.

Para gerar um seeder, execute o comando make:seeder Artisan. Todos os seeders gerados pelo framework serão colocados no diretório database/seeders. Para criar um seeder usa-se o seguinte comando.

- php artisan make:seeder UserSeeder

Para executar as migrations com os seeders imediatamente, pode-se usar o comando:

- php artisan migrate --seed

2.1 DOCUMENTAÇÃO LARAVEL

https://laravel.com/docs/8.x/seeding

3. ROTAS

As rotas são os caminhos criados na url para poder direcionar o usuário a uma página ou a enviar dados. Todas as rotas possuem um método HTTP como:

- GET;
- POST;
- PUT;
- PATCH;
- DELETE;
- OPTIONS;

As rotas default do laravel são as rotas:

- INDEX;
- CREATE;
- STORE;
- EDIT;
- SHOW;
- UPDATE;
- DELETE;

3.1 MÉTODOS

Os métodos referentes a cada um desses métodos é:

- Route::get($uri, $callback);
- Route::post($uri, $callback);

- Route::put($uri, $callback);
- Route::patch($uri, $callback);
- Route::delete($uri, $callback);
- Route::options($uri, $callback);

É possível criar uma rota passando a url como primeiro parâmetro, e um array com a classe do Controller na primeira posição, e o método a se utilizar como segunda posição.

- Route::get('/sistemas', [SistemaController::class, 'index'])

Também é possível utilizar o método Route::resources('url', Class::class) para criar todas as rotas padrões em apenas uma única linha.

3.3 ROTAS NO OMNI

Para criar as rotas defaults no Omni, basta cadastrar a rotina. O sistema cria as rotas automaticamente.

Ao iniciar um novo sistema é necessário criar uma função para botar um prefixo de slug, e um prefixo no nome para todas as rotas do sistema.

Também é necessário chamar o método RouteHelper::routes(), onde o primeiro parâmetro é o ID do sistema a ser criado, e o segundo parâmetro é uma identificação se a rota será web ou api.

```
Route::prefix('/unico')->name(Sistema::UNICO . '.')->group(function () {
    RouteHelper::routes(Sistema::UNICO_ID, RouteHelper::IS_WEB);
});
```

Se for necessário, é possível adicionar novas rotas dentro desse grupo como mencionado no início desse tópico.

3.4 DOCUMENTAÇÃO LARAVEL

https://laravel.com/docs/8.x/routing

4. MODEL

Para a utilização basta estender a classe App\Models\ModelPadrao e importar a trait de App\Models\ModelAcoesPadrao.

4.1 TABELA

Para informar a tabela basta atribuir o atributo $table ao valor de schema.table

```
protected $table = Sistema::ESTRUTURA . '.rotinas';
```

4.2 ATRIBUTOS SIMPLIFICADOS

Para informar os atributos do *model* basta inicializar o *construct* e informar dentro do atributo de colunas um vetor.

```
$this->colunas = [
    new Attribute('nome', new InputOptions('Nome', Tipagem::TEXTO)),
];
```

Por padrão, é necessário instanciar a classe App\Structural\Attribute, passando como primeiro parâmetro o nome da coluna na tabela do banco de dados. O segundo parâmetro é uma instância da classe App\Structural\InputOptions onde o primeiro parâmetro é o “nome apresentável”, e o segundo parâmetro é o tipo do campo. Os tipos existentes na classe de App\Structural\Tipagem serão abordados posteriormente na sessão de Tipagens.

4.3 ATRIBUTOS COMPLETOS

```php
public function __construct(array $attributes = [])
{
    $this->colunas = [
        new Attribute(
            'nome',
            new InputOptions('Nome', Tipagem::TEXTO, Operador::LIKE, true, ['max'=> 2, 'min' => 1], 'nome'),
            new Relation('rotina', 'nome'),
            new ExhibitionOptions(true)
        ),
    ];

    parent::__construct($attributes);
}
```

Outros parâmetros também são possíveis de serem passados. Dentro da classe de *InputOptions* é possível informar 6 parâmetros, que são o “nome apresentável”, o tipo do dado, o operador padrão do filtro, sua obrigatoriedade, um array associativo de tamanho máximo e mínimo, e uma descrição.

Para referenciar uma *Relation*, é necessário instanciar a classe App\Structural\Relation e passar como primeiro parâmetro o nome da *relation*, e o segundo parâmetro como a coluna do *model* vinculado a relation.

Para definir as opções de visibilidade em listas, filtros e grids, pode-se instanciar a classe *ExhibitionOptions* e passar como parâmetro uma variável booleana. O primeiro parâmetro é para listas, o segundo para filtros e o terceiro para Grids.

4.4 MÉTODOS

getColunas() // Retorna as colunas do model;
getRelacoes() // Retorna as relações do model;
getColunaByCampo() // Retorna a coluna buscada pelo nome;
getSelectOptions() // Retorna as colunas já formatadas para um select;
getInfo() // Retorna as informações da Rotina armazenada no BD;
getRouteDefault($action) // retorna a url pronta para a funcionalidade, podendo ser os padrões de ‘create’, ‘index’, ‘edit’, ‘destroy’, etc;
getSlug() // Retorna a slug do model.

4.5 Exemplo

```php
final class Estado extends ModelPadrao
{
    use ModelAcoesPadrao;

    protected $table = Sistema::FROTA . ".estados";

    /**
     * @param array $attributes
     */
    public function __construct(array $attributes = [])
    {
        $this->colunas = [
            new Attribute( campo: 'title', new InputOptions( label: 'Título',  tipagem: Tipagem::TEXTO,  operador: Operador::INICIA)),
            new Attribute( campo: 'letter',
                new InputOptions( label: 'Sigla',  tipagem: Tipagem::TEXTO),
                 relation: null,
                new ExhibitionOptions( viewableList: false)
            ),
            new Attribute( campo: 'iso', new InputOptions( label: 'ISO',  tipagem: Tipagem::INTEIRO,  operador: Operador::IGUAL)),
            new Attribute( campo: 'slug', new InputOptions( label: 'SLUG',  tipagem: Tipagem::TEXTO)),
            new Attribute( campo: 'population', new InputOptions( label: 'Polupação',  tipagem: Tipagem::INTEIRO)),
        ];

        $this->relacoes = [
            new Relation( function: 'cidades',  campo: 'title',  label: 'Cidades',  isOpen: true)
        ];

        parent::__construct($attributes);
    }

    /**
     * @return HasMany
     */
    public function cidades(): HasMany
    {
        return $this->hasMany( related: Cidade::class,  foreignKey: 'state_id',  localKey: 'id')->orderBy( column: 'id');
    }
}
```

4.6 DOCUMENTAÇÃO LARAVEL

https://laravel.com/docs/8.x/eloquent

5. RELATIONS

Para criar uma *Relation* deve-se criar uma função dentro do *model*, que retorne um método *relationship* como *hasMany*, *hasOne*, *belongsTo*, onde o primeiro parâmetro é a classe relacionada. O segundo e o terceiro parâmetro podem ser nulos, mas são respectivamente a chave atual e a chave estrangeira.

```php
public function sistema()
{
    return $this->belongsTo(Sistema::class, 'sistema_id', 'id');
}
```

Para informar ao Model quais são as relações existentes utilize o $this->relacoes que recebe um array de Relations contendo as informações necessárias(descrito mais abaixo).

5.1 DOCUMENTAÇÃO LARAVEL

https://laravel.com/docs/8.x/eloquent-relationships

6. CONTROLLER

Para utilizar do *ControllerPadrao* basta estender a classe App\Http\Controllers\ControllerPadrao.

No método Construct o controller identifica qual a ação, o sistema e a rotina que estão sendo utilizadas a fim de poder manusear cada rotina adequadamente.

6.1 MÉTODOS

- **Index:** Retorna a view de listagem no sistema web, e retorna os dados de tal rotina na api.
- **Create:** Retorna a view de cadastro.
- **Store:** Recebe os dados e efetua a inserção dos dados no banco de dados.
- **Show:** Retorna a view de visualização do registro no sistema web, ou somente o registro procurado no caso da api.
- **Edit:** Retorna a view de edição do registro.
- **Update:** Recebe os dados e efetua a atualização dos dados no banco de dados.
- **Destroy:** Recebe os dados e efetua a remoção dos dados no banco de dados.

6.2 DOCUMENTAÇÃO LARAVEL

https://laravel.com/docs/8.x/controllers

7. COMPONENTES

Principais componentes disponíveis:

1. Form (dir)
    a. form
    b. help
    c. Inputs (dir)
        i. checkbox
        ii. switch
        iii. color
        iv. datalist
        v. date
        vi. externo
        vii. externo-offcanvas
        viii. file
        ix. number
        x. password

xi. radio
xii. range
xiii. select
xiv. text
xv. email

2. Grid (dir)
    a. grid
3. Table (dir)
    a. filtros
    b. listagem

Para entender melhor os atributos disponíveis acesse seus respectivos controladores em app/View/Components.

Ainda estamos criando os inputs que faltam, a lista pode estar desatualizada no momento da leitura

8. VIEW

As views só serão utilizadas em páginas customizadas. Caso não criar a respectiva view será gerado automaticamente a página com base nos colunas preenchidas no Model. Para utilizar uma view customizada, apenas precisa-se criar uma view no diretório do respectivo sistema. Lembrando que os nomes das views utilizadas pelo ControllerPadrao são a index e a create.

8.1 EXEMPLO

“Ah, quero criar uma view em Frotas!”

- Execute o comando php artisan view:make Frota/Model/Create e php artisan view:make Frota/Model/Index
- A view de create recebe a variável $editar com as informações do registro, estando assim em seu modo “update”.

9. TREE

Para poder criar uma tree, primeiro deve-se cadastrá-la na rotina de trees. Posteriormente é necessário que o model estenda a classe App\Models\ModelTreePadrao para que obtenha as colunas *parent_id*, *depth* e *ordem* que são necessárias na tree.

Se necessário, pode-se criar um controller e estender a classe App\Http\Controllers\Estrutura\TreeNodeControllerPadrao para sobrescrever os métodos.

9.2 CONFIGURAÇÕES

```
@php
    $modelTree = \App\Models\Estrutura\Tree::class;
@endphp
<x-tree.tree :id="$modelTree::TREE_CATEGORIA"
    :config="[
        $modelTree::CONFIG_NODE_LOADED_TO_EDIT => $editar->id ?? false,
        $modelTree::CONFIG_MANIPULAVEL         => true,
        $modelTree::CONFIG_SHOW_CURRENT_NODE   => true,
        $modelTree::CONFIG_SHOW_CURRENT_DEPTH  => false
    ]"
/>
```

Para chamar o componente deve-se usar o componente x-tree.tree, onde o parâmetro id recebe o ID da tree, e o parâmetro config deve ser um array associativo. Para cada configuração há uma constante criada em App\Models\Estrutura\Tree.

- CONFIG_NODE_LOADED_TO_EDIT // Se a árvore iniciará com um node específico selecionado;
- CONFIG_MANIPULAVEL // Se é possível criar, renomear e deletar os nodes
- CONFIG_SHOW_CURRENT_NODE // Se é possível de editar dados referentes ao node selecionado;
- CONFIG_SHOW_CURRENT_DEPTH // Se é possível de editar dados referentes ao nível do node selecionado;

9.3 TREE JAVASCRIPT

```
import { Tree } from '../Tree.js';

class Categoria extends Tree {

}

let tree = new Categoria($('#jstree').data('config'), {rotina: 'Servico\\Categoria'});
```

Para dimensionar uma nova tree, basta importar o modelTree e estende-lo. Ao instanciar o objeto, faz-se necessário utilizar como primeiro parâmetro a configuração (coletada via jQuery do atributo data-config), e como segundo parâmetro um objeto onde pode ser informada a rotina através do nome do Sistema/Model.

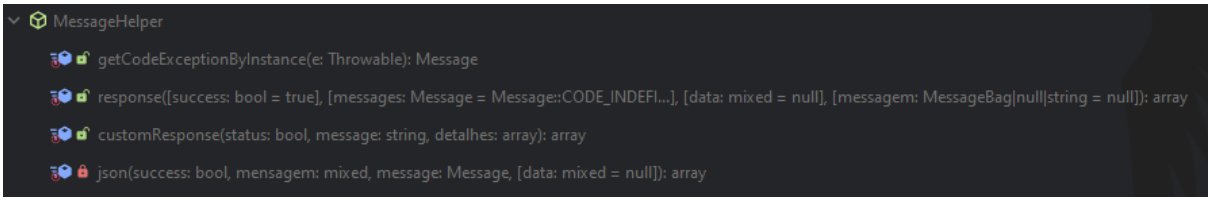
9.4 DOCUMENTAÇÃO JSTREE

https://www.jstree.com/

10. MENSAGERIA

10.1 CLASSE DE MESSAGE HELPER

Utilizar a classe MessageHelper para disparar as mensagens.



10.2 MÉTODOS

getCodeExceptionByInstance() // Obtém uma Messege de acordo com a exception passada por parâmetro;
response() // Método para disparar uma resposta padrão;
customResponse() // Mensagens customizadas, obedecendo o mínimo da estrutura de respostas.

10.3 ENUM MESSAGE

Os textos podem ser obtidos pelo enum de Message (app/Exceptions/Message.php):

Utilizando o método de message() você obtém o texto cadastrado em app/Exceptions/CodeList.json ou no local alterado via env *(conforme descrito nas variáveis do env)*.

10.4 EXEMPLO

```
try{
   //something
}catch (Exception $e){
        return MessageHelper::response(false, $this->messegeByException($e));
}

MessageHelper::response(false, Message::ERRO_GERAL);

MessageHelper::response(false, Message::ERRO_GERAL, $dados, $mensagem_customizada);
```

11. API

Para utilizar a API é necessário estar autenticado. Essa autenticação é realizada através de um Bearer Token, obtido após a autenticação feita pela rota POST http://localhost/api/login onde se é enviado um email e uma senha através dos atributos “email” e “password”.

Para acesso a qualquer funcionalidade ou qualquer rotina, deve-se acessar a rota onde o primeiro diretório será “/api”. O resto se mantém igual a parte WEB do sistema.

11.2 EXEMPLO

11.2.1 Rota Web

http://localhost/estrutura/sistema

11.2.2 Rota Api

http://localhost/api/estrutura/sistema

12. ENV

Além das variáveis necessárias para o laravel trabalhar, temos mais algumas opções:

12.1 APP_DEBUG (BOOLEAN)

Permite estourar exceptions.

12.2 API_DEBUG (BOOLEAN)

Permite estourar exceptions na api.

12.3 ENABLE_ALL_PERMISSIONS (BOOLEAN)

Libera todas as permissões no sistema.

12.4 DEBUGBAR_ENABLED (BOOLEAN)

Exibe a barra de informações no final da página.

12.5 APP_CLIENTE (STRING)

Nome do cliente que está instalado, um por host.

12.6 SESSION_LIFETIME (INT EM MINUTOS)

Tempo da sessão no php. Padrão é 120 minutos (2h).

12.7 JWT_SECRET (STRING, 64 BITS OU MAIS)

Chave para criptografia do token de login na API.

12.8 CODE_LIST (string, diretorio)

Possibilita trocar o diretório do arquivo de mensagens de erro.

13. EXTRAS

Documentação a ser estudada para melhor entendimento do ecossistema:

- https://laravel.com/docs/8.x
- https://getbootstrap.com/docs/5.1/getting-started/introduction/