Para usar, basta substituir:

- [Entidade] -> O nome da sua Model (ex: Produto, Tarefa, Despesa)
- [entidade] -> O nome em minúsculo, singular (ex: produto, tarefa, despesa)
- [entidades] -> O nome em minúsculo, plural (ex: produtos, tarefas, despesas)

Eu vou usar a estrutura de **Resource Controller** do Laravel, que é a mais completa e segue as convenções REST.

---

1. O Model (app/Models/[Entidade].php)

Este arquivo representa a tabela no banco de dados.

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;

class [Entidade] extends Model
{
    use HasFactory;

    /**
     * O nome da tabela no banco de dados.
     * (Opcional se o nome da tabela for o plural do nome do model, ex: 'entidades')
     */
    protected $table = '[entidades]';

    /**
     * Os atributos que podem ser preenchidos em massa.
     * (ESSENCIAL para o método 'create')
     */
    protected $fillable = [
        'nome',
        'descricao',
        'user_id', // Chave estrangeira para o dono (exemplo)
        // Adicione outros campos da sua tabela aqui
```

```
    ];

    /**
     * Os atributos que devem ser convertidos (cast).
     * (Boa prática para datas, booleanos, etc.)
     */
    protected $casts = [
        'created_at' => 'datetime',
        'updated_at' => 'datetime',
        // 'ativo' => 'boolean',
    ];

    /**
     * Exemplo de relacionamento:
     * Uma [Entidade] pertence a um Usuário.
     */
    public function user(): BelongsTo
    {
        return $this->belongsTo(User::class, 'user_id');
    }
}
```

---

2. O Repository (app/Repositories/[Entidade]Repository.php)

Este arquivo é o único que "conversa" com o Model. Sua única função é fazer queries no banco.

```
<?php

namespace App\Repositories;

use App\Models\[Entidade];
use Illuminate\Pagination\LengthAwarePaginator;

class [Entidade]Repository
{
    /**
     * Retorna uma lista paginada de entidades.
     * (Pode adicionar filtros aqui se necessário)
```

```php
    */
    public function getAllPaginated(int $perPage = 15): LengthAwarePaginator
    {
        // 'with('user')' é um exemplo de Eager Loading para carregar o relacionamento
        return [Entidade]::with('user')->orderBy('created_at', 'desc')->paginate($perPage);
    }

    /**
     * Encontra uma entidade pelo seu ID.
     * Falha (erro 404) se não encontrar.
     */
    public function findById(string $id): [Entidade]
    {
        return [Entidade]::with('user')->findOrFail($id);
    }

    /**
     * Cria um novo registro de entidade.
     */
    public function create(array $data): [Entidade]
    {
        return [Entidade]::create($data);
    }

    /**
     * Atualiza uma entidade existente pelo seu ID.
     */
    public function update(string $id, array $data): [Entidade]
    {
        $entidade = $this->findById($id);
        $entidade->update($data);
        return $entidade;
    }

    /**
     * Exclui uma entidade pelo seu ID.
     */
    public function delete(string $id): bool
    {
        $entidade = $this->findById($id);
        return $entidade->delete();
    }
}
```

---

3. O Service (app/Services/[Entidade]Service.php)

Este arquivo contém a lógica de negócio. Ele "orquestra" o repositório e outras regras.

```php
<?php

namespace App\Services;

use App\Repositories\[Entidade]Repository;
use Illuminate\Pagination\LengthAwarePaginator;
use App\Models\[Entidade];
use Illuminate\Support\Facades\Log; // Exemplo para log de erros

class [Entidade]Service
{
    protected $repository;

    public function __construct([Entidade]Repository $repository)
    {
        $this->repository = $repository;
    }

    public function getAllPaginated(int $perPage = 15): LengthAwarePaginator
    {
        return $this->repository->getAllPaginated($perPage);
    }

    public function findById(string $id): [Entidade]
    {
        return $this->repository->findById($id);
    }

    /**
     * Lógica de Negócio para CRIAR uma entidade.
     * Ex: Adicionar o ID do usuário logado.
     */
    public function create(array $data): [Entidade]
    {
        // ---- REGRA DE NEGÓCIO EXEMPLO ----
```

```
        // Adiciona o ID do usuário autenticado aos dados antes de salvar.
        $data['user_id'] = auth()->id();
        // --------------------------------

        return $this->repository->create($data);
    }

    /**
     * Lógica de Negócio para ATUALIZAR uma entidade.
     * (As checagens de permissão são feitas no Form Request ou Policy)
     */
    public function update(string $id, array $data): [Entidade]
    {
        // Se houver alguma regra de negócio (ex: disparar um evento),
        // ela viria aqui antes de chamar o repositório.

        return $this->repository->update($id, $data);
    }

    /**
     * Lógica de Negócio para DELETAR uma entidade.
     * Ex: Checar se a entidade pode ser deletada.
     */
    public function delete(string $id): bool
    {
        // Exemplo de regra:
        // $entidade = $this->repository->findById($id);
        // if ($entidade->is_protected) {
        //     throw new \Exception('Esta entidade não pode ser excluída.');
        // }

        return $this->repository->delete($id);
    }
}
```

---

4. Os Form Requests (Os "Seguranças")

Dividimos em dois: um para criar (Store) e um para atualizar (Update), pois as regras de validação podem ser diferentes (ex: unique).

app/Http/Requests/Store[Entidade]Request.php

```php
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class Store[Entidade]Request extends FormRequest
{
    /**
     * Determina se o usuário está autorizado a fazer esta requisição.
     */
    public function authorize(): bool
    {
        // Qualquer usuário logado pode TENTAR criar.
        // Se houver regras (ex: só Admin), coloque aqui.
        return auth()->check();
    }

    /**
     * Retorna as regras de validação que se aplicam à requisição.
     *
     * @return array<string, \Illuminate\Contracts\Validation\ValidationRule|array<mixed>|string>
     */
    public function rules(): array
    {
        return [
            'nome' => 'required|string|max:255',
            'descricao' => 'nullable|string',
            // 'user_id' não é necessário aqui, pois o Service irá injetá-lo.
            // Se o usuário puder escolher o 'user_id':
            // 'user_id' => 'required|integer|exists:users,id'
        ];
    }
}
```

app/Http/Requests/Update[Entidade]Request.php

```php
<?php

namespace App\Http\Requests;

use App\Models\[Entidade];
use Illuminate\Foundation\Http\FormRequest;

class Update[Entidade]Request extends FormRequest
{
    /**
     * Determina se o usuário está autorizado a fazer esta requisição.
     * Ex: Só o dono ou um admin pode editar.
     */
    public function authorize(): bool
    {
        // 1. Pega a entidade da rota (ex: /entidades/5)
        // Isso requer que a rota use Route Model Binding.
        $entidade = $this->route('[entidade]');

        // 2. Checa a permissão
        $user = auth()->user();
        return $user->isAdmin() || $entidade->user_id == $user->id;
    }

    /**
     * Retorna as regras de validação que se aplicam à requisição.
     *
     * @return array<string, \Illuminate\Contracts\Validation\ValidationRule|array<mixed>|string>
     */
    public function rules(): array
    {
        // 'sometimes' = valide APENAS se o campo for enviado.
        // 'nullable' = o campo pode ser enviado como nulo.
        return [
            'nome' => 'sometimes|required|string|max:255',
            'descricao' => 'sometimes|nullable|string',
            // 'user_id' => 'sometimes|required|integer|exists:users,id'
        ];
    }
}
```

---

5. O Controller (app/Http/Controllers/[Entidade]Controller.php)

Este é o "Gerente", que apenas coordena o Request e o Service. Note como ele é limpo e pequeno.

```php
<?php

namespace App\Http\Controllers;

use App\Models\[Entidade];
use App\Services\[Entidade]Service;
use App\Http\Requests\Store[Entidade]Request;
use App\Http\Requests\Update[Entidade]Request;
use Illuminate\Http\JsonResponse;
use Illuminate\Http\Request; // Usado apenas no index() para filtros

class [Entidade]Controller extends Controller
{
    protected $service;

    public function __construct([Entidade]Service $service)
    {
        $this->service = $service;
    }

    /**
     * Retorna uma lista paginada de entidades.
     * GET /api/[entidades]
     */
    public function index(Request $request): JsonResponse
    {
        $perPage = $request->query('per_page', 15);
        $entidades = $this->service->getAllPaginated($perPage);
        return response()->json($entidades);
    }

    /**
     * Cria uma nova entidade.
     * POST /api/[entidades]
```

```
     */
    public function store(Store[Entidade]Request $request): JsonResponse
    {
        // $request->validated() retorna APENAS os dados validados pelo Request
        $entidade = $this->service->create($request->validated());
        return response()->json($entidade, 201); // 201 Created
    }

    /**
     * Retorna uma entidade específica.
     * GET /api/[entidades]/{entidade}
     * (Usando Route Model Binding)
     */
    public function show([Entidade] $entidade): JsonResponse
    {
        // O $entidade já foi encontrado pelo Laravel.
        // Se não for encontrado, o Laravel retorna 404 automaticamente.
        // Se precisar recarregar com relações: $entidade->load('user');
        return response()->json($entidade);
    }

    /**
     * Atualiza uma entidade específica.
     * PUT/PATCH /api/[entidades]/{entidade}
     */
    public function update(Update[Entidade]Request $request, [Entidade] $entidade): JsonResponse
    {
        // A autorização e validação já ocorreram no Update[Entidade]Request
        $entidadeAtualizada = $this->service->update($entidade->id, $request->validated());
        return response()->json($entidadeAtualizada);
    }

    /**
     * Exclui uma entidade específica.
     * DELETE /api/[entidades]/{entidade}
     */
    public function destroy([Entidade] $entidade): JsonResponse
    {
        // Você pode criar um Delete[Entidade]Request para autorização,
        // ou usar uma Policy (o mais recomendado).
        // Exemplo simples de autorização:
        if (auth()->user()->cannot('delete', $entidade)) {
            abort(403, 'Ação não autorizada.');
        }
```

```
        $this->service->delete($entidade->id);
        return response()->json(null, 204); // 204 No Content
    }
}
```

---

6. As Rotas (routes/api.php)

Finalmente, conecte tudo no seu arquivo de rotas.

```
<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\[Entidade]Controller;

// ... suas outras rotas

Route::middleware('auth:sanctum')->group(function () {
    // ...

    // Esta única linha cria TODAS as 5 rotas (index, store, show, update, destroy)
    Route::apiResource('[entidades]', [Entidade]Controller::class);

    // ...
});
```

Explicação das Funções e Comandos

1. Funções Principais

- **__construct(...)**: É o "Construtor" da classe. Ele usa a **Injeção de Dependência** do Laravel para "pedir" as classes de que precisa. O Controller pede o Service, e o Service pede o Repository. O Laravel cuida de criar e entregar (injetar) essas instâncias

automaticamente.

- **$fillable (no Model):** Lista de "campos seguros" que o método Model::create() tem permissão para preencher. Se um campo não estiver aqui (como is_admin), ele será ignorado por segurança.
- **findOrFail($id) (no Repository):** Um comando Eloquent que significa "Encontre o registro com este ID. Se não encontrar, pare tudo e retorne um erro 404 Not Found".
- **with('user') (no Repository):** Chama-se "Eager Loading". Ele diz ao Laravel: "Quando você buscar a entidade, já traga junto os dados do 'usuário' relacionado a ela". Isso evita o problema de N+1 queries e é muito mais eficiente.
- **authorize() (no Form Request):** O "Segurança". Executa *antes* de tudo. Se retornar false, a requisição é barrada com um erro **403 Forbidden** (Não Autorizado).
- **rules() (no Form Request):** As "Regras de Validação". Executa *depois* do authorize(). Se as regras falharem, a requisição é barrada com um erro **422 Unprocessable Entity** (com a lista de erros).
- **$request->validated() (no Controller):** Função mágica dos Form Requests. Ela retorna um array contendo *apenas* os dados que passaram nas rules(). Isso impede que dados maliciosos (como um is_admin = true enviado pelo usuário) cheguem ao seu Service.
- **Route Model Binding (no Controller e Rota):** Quando você "pede" a Model no método do controller (ex: function show([Entidade] $entidade)), o Laravel automaticamente faz o findOrFail() para você, usando o ID que veio na URL. Isso limpa seu controller.

2. Comandos CLI do Artisan

Para criar esses arquivos, você usará o php artisan:

1. **Criar Model e Migration:**
   Bash
   `php artisan make:model [Entidade] -m`
   - make:model [Entidade]: Cria o arquivo app/Models/[Entidade].php.
   - -m: Cria *também* o arquivo de *migration* em database/migrations/ para você definir a estrutura da tabela no banco.
2. **Criar o Controller (sem ser resource):**
   Bash
   `php artisan make:controller [Entidade]Controller`
   - *Nota:* Eu **não** recomendo usar php artisan make:controller [Entidade]Controller -r --api (o que criaria um controller resource) porque os métodos gerados não viriam preparados para injetar o Service. É melhor copiar e colar o template acima.
3. **Criar os Form Requests:**

Bash
```
php artisan make:request Store[Entidade]Request
php artisan make:request Update[Entidade]Request
```

4. Arquivos Manuais (Service e Repository):
   O Laravel não tem comandos artisan nativos para Services e Repositories, pois eles são padrões de arquitetura, não componentes nativos do framework.
   - Você deve criar as pastas app/Services e app/Repositories manualmente.
   - Dentro delas, crie os arquivos ([Entidade]Service.php e [Entidade]Repository.php) e cole os templates.
5. Dica Extra (Policies para Autorização):
   A forma mais "Laravel" de fazer a autorização (em vez de usar o método authorize no Request) é com uma Policy.
   Bash
   ```
   php artisan make:policy [Entidade]Policy --model=[Entidade]
   ```

   - Isso cria um arquivo app/Policies/[Entidade]Policy.php onde você pode definir métodos como view(), create(), update(), delete(), e o Laravel os usará automaticamente.