INSTITUTO FEDERAL
Paraná
Campus Paranaguá

Ministério da Educação

TECNÓLOGO EM ANÁLISE E DESENVOLVIMENTO
Disciplina de Desenvolvimento Web II
Aula 11: Framework Laravel - CSR e Auditing

Gil Eduardo de Andrade

Conceitos Preliminares

(https://laravel-auditing.com)

O Padrão Arquitetural CSR (*Controller-Service-Repository*)

No desenvolvimento de aplicações web modernas, a manutenibilidade, a testabilidade e a separação de conceitos (Separation of Concerns) são pilares fundamentais. O padrão CSR é uma evolução natural aplicada sobre o tradicional MVC (*Model-View-Controller*), organizando a camada de negócio que frequentemente "engorda" e se torna complexa dentro dos Modelos (*Models*) ou Controladores (Controllers).

1. Camada Controller (Controlador)

Controllers são a porta de entrada das requisições, onde a única responsabilidade é mudar o estado do HTTP. Ele recebe a requisição (Request), valida os dados de entrada (geralmente via *Form Requests* no Laravel) e delega toda a lógica de negócio para a camada de serviço.

- É sua função: capturar parâmetros, chamar o serviço adequado e retornar uma resposta (JSON, View, Redirect).
- NÃO É sua função: efetuar consultas ao banco de dados ou fazer cálculos de regras de negócio.

2. Camada Service (Serviço)

Nas *Services* residem as regras de negócio da aplicação. Esta camada é agnóstica em relação à infraestrutura (ela não sabe se a requisição veio da Web, de uma API ou de um comando de console).

- É sua função: orquestrar fluxos de trabalho, aplicar validações de negócio, disparar eventos, efetuar integração com APIs externas e solicitar dados à camada Repository.
- NÃO É sua função: manipular diretamente requisições HTTP ($request->all()) ou executar queries SQL puras.

3. Camada Repository (Repositório)

A camada que encapsula o acesso aos dados. Ela serve como um mediador entre a camada de negócio (Service) e a fonte de dados (Banco de dados, cache, arquivos).

- É sua função: Abstrair o ORM (como o Eloquent). Se futuramente a forma de obtenção de dados mudar de um formato SQL para uma API GraphQL, apenas a camada Repository precisará ser alterada.
- NÃO É sua função: Tomar decisões sobre regras de negócio.

Abaixo, o diagrama ilustra o fluxo de dados e a divisão de responsabilidades neste padrão:

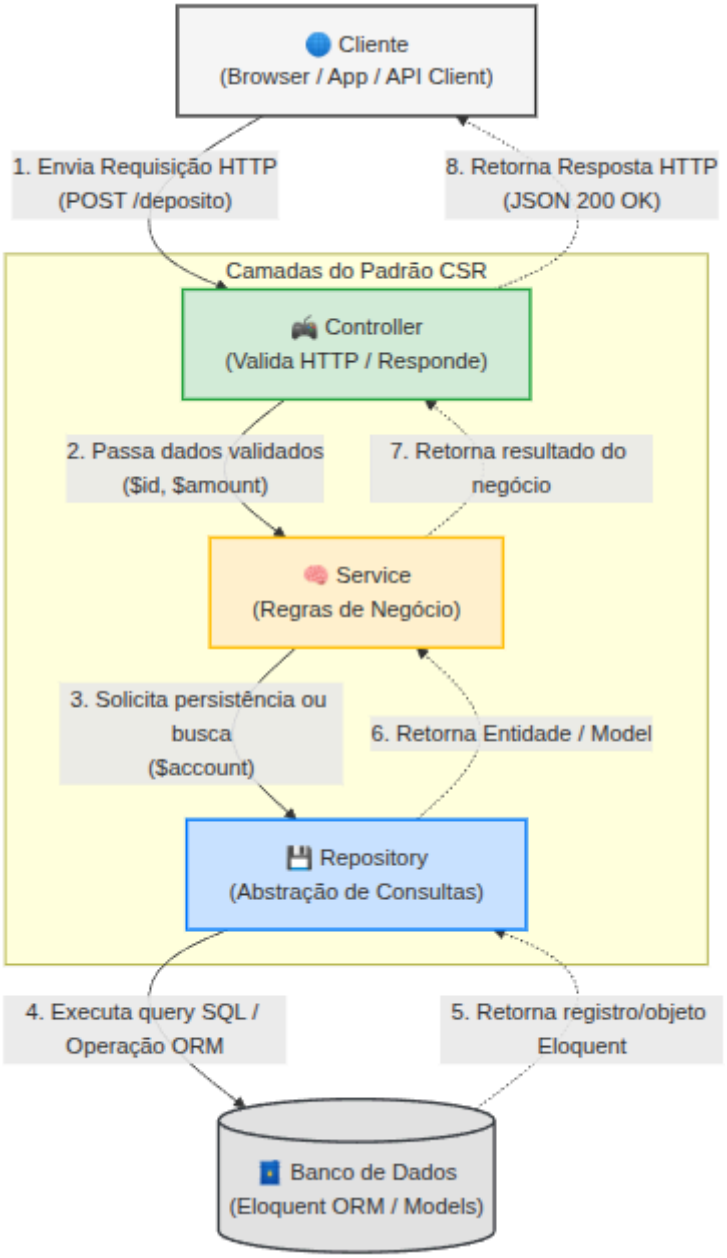


Figura 01: Digrama de Fluxo CSR | Gerada pelo autor - https://mermaid.live/

Entendendo o Fluxo do Diagrama

1. **Requisição (Ida)**:
    a. O cliente faz uma requisição HTTP.
    b. O Controller a intercepta, valida se os dados são estruturalmente corretos e os repassa "mastigados" para o Service.
    c. O Service avalia se a operação faz sentido lógico (regra de negócio) e, se tudo estiver correto, pede para o *Repository* salvar ou buscar as informações no Banco de Dados.

2. **Resposta (Volta)**:
    a. O Banco de Dados devolve a informação bruta para o Repository, que a transforma em um objeto manipulável.
    b. O Service finaliza qualquer lógica necessária e devolve o resultado limpo para o Controller.
    c. Por fim, o Controller monta a resposta HTTP (geralmente um JSON para APIs) e entrega de volta ao Cliente.

Esse fluxo garante que nenhuma camada saiba mais do que deveria sobre a outra, mantendo o código limpo e modular.

O Pacote *Laravel Auditing*

À medida que os sistemas crescem, rastrear quem alterou o quê e quando torna-se um requisito crítico de segurança e conformidade (compliance). O pacote owen-it/laravel-auditing é a solução padrão de mercado para implementar trilhas de auditoria no ecossistema Laravel.

Funcionamento

O pacote utiliza os *Eloquent Events* (como created, updated, deleted) de forma automatizada. Ao registrar um ***Trait no Model***, o pacote captura o estado anterior e posterior do registro, salvando-o em uma tabela dedicada de auditoria (audits).

Trait PHP

Um Trait em PHP é um mecanismo de reutilização de código que permite compartilhar métodos e propriedades entre diferentes classes, contornando a limitação de herança simples da linguagem. Ele atua como um bloco de construção horizontal, injetando funcionalidades comuns sem exigir uma relação hierárquica.

Principais Características

- Não pode ser instanciado: não é possível criar um objeto diretamente de um Trait.
- Composição horizontal: adiciona comportamentos diretamente às classes, como se o código estivesse escrito dentro delas.
- Múltiplos *Traits*: você pode importar quantos *Traits* precisar em uma única classe

Exemplos Simples - Conceito de Traits PHP

```php
// 1. Definindo o Trait
trait Logger {
    public function log(string $mensagem) {
        echo "[LOG] " . $mensagem . "<br>";
    }
}

// 2. Usando o Trait na classe Usuário
class Usuario {
    use Logger;

    public function criar() {
        $this->log("Usuário criado com sucesso!");
    }
}

// 3. Usando o Trait na classe Produto
class Produto {
    use Logger;

    public function deletar() {
        $this->log("Produto removido do sistema.");
    }
}

// 4. Testando
$user = new Usuario();
$user->criar(); // Saída: [LOG] Usuário criado com sucesso!
```

Principais Recursos - Laravel Auditing

- Rastreamento Automático: captura o ID do usuário autenticado, endereço IP, User-Agent e a URL da requisição de forma nativa.

- Modificações Detalhadas: armazena os campos modificados nos formatos old_values e new_values.

- Customização: Permite ignorar colunas sensíveis (como senhas), limitar a quantidade de registros guardados (*User Redundancy*) e customizar o driver de armazenamento.

Instalação - Laravel Auditing
Para instalar a versão estável para o Laravel 13, execute o comando:

```Shell
composer require owen-it/laravel-auditing
```

Publicando as Configurações e Migrations
Após baixar o pacote, é preciso publicar os arquivos de configuração e a migration que criará a tabela onde os logs de auditoria serão salvos:

```Shell
php artisan vendor:publish
--provider="OwenIt\Auditing\AuditingServiceProvider"
```

O comando anterior criará os seguintes arquivos:

- config/audit.php: arquivo de configuração do pacote instalado;
- database/migrations/…_create_audits_table.php: arquivo de migração para criação da tabela de logs.

Em seguida, vamos executar as migrations para criar a tabela audits no banco:

```Shell
php artisan migrate:fresh
```

Configurando os *Models* (Onde a mágica acontece)
Para definir qual conjunto de tabelas (Models) deve ser monitorado, só é preciso implementar uma Interface e usar um *Trait* nos *Models* desejados. Vamos supor que deseja-se monitorar o *Model Product* e o *Model Order*. Veja como ficaria a configuração:

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use OwenIt\Auditing\Contracts\Auditable; // Importa a Interface
use OwenIt\Auditing\Auditable as AuditableTrait; // Importa o Trait

class Product extends Model implements Auditable
{
    use AuditableTrait; // Habilita o rastreamento automático

    protected $fillable = ['name', 'price', 'stock'];
}
```

A partir do momento em que o *Trait* é adicionado, qualquer operação de Criação, Edição ou Remoção realizada, via *Eloquent,* irá disparar um registro na ***tabela audits***.

Ajustes Finos Opcionais por *Model*
É possível customizar o comportamento do *Laravel Auditing* para alguma tabela em específico, através da inclusão de algumas propriedades na *Model*:

- Ignorar colunas sensíveis (ex: senhas, chaves de API, etc):

```php
protected $auditExclude = ['password', 'remember_token'];
```

- Limitar a quantidade de registros por registro (Threshold): caso seja necessário limitar o tamanho do histórico, para que o mesmo não cresça infinitamente, é possível, por exemplo, manter apenas as últimas 50 modificações efetuadas para um determinado recurso.

```php
protected $auditThreshold = 50;
```

Como o pacote captura o Usuário?

O *Laravel Auditing* detecta de forma nativa e automática o usuário autenticado na requisição através do Guard padrão do Laravel (*Auth::user()*). Ele salvará duas informações cruciais na tabela de auditoria:

- user_id: ID do usuário que fez a ação.
- user_type: namespace do *Model* do usuário (ex: App\Models\User), permitindo o funcionamento com múltiplos tipos de usuários (como Administradores e Clientes).

Como Consultar os Dados de Auditoria (Uso)
O pacote cria um relacionamento dinâmico com o *Model* auditado. Você pode listar as auditorias facilmente na sua camada de controle ou exibição.

Listando o histórico de um registro específico:

```php
use App\Models\Product;

$product = Product::findOrFail(1);

// Recupera todas as auditorias ordenadas da mais recente para a mais antiga
$audits = $product->audits()->with('user')->latest()->get();

foreach ($audits as $audit) {
    echo "Ação: " . $audit->event . "<br>"; // Ex: created, updated, deleted
    echo "Feito por: " . ($audit->user->name ?? 'Sistema/Console') . "<br>";
    echo "Data: " . $audit->created_at->format('d/m/Y H:i') . "<br>";
    // Mostra o que mudou
    echo "Valores Antigos: " . json_encode($audit->old_values) . "<br>";
    echo "Valores Novos: " . json_encode($audit->new_values) . "<br><br>";
}
```

APLICANDO OS CONCEITOS: CSR / AUDITING

Continuação da Aplicação Criada na Aula Anterior

1) Repositório (Repository): Criando Diretório e Arquivos

Estrutura, dentro da pasta “**app/**” crie (manualmente) o diretório ***“Repositories/”*** com os seguintes arquivos:

- **Repositories/**
  - BaseRepository.php
  - CursoRepository.php
  - DisciplinaRepository.php

Arquivo: "Repositories/BaseRepository.php"

PHP

```php
<?php

namespace App\Repositories;

use Illuminate\Database\Eloquent\Model;

abstract class BaseRepository {

    abstract protected function getModel(): mixed;

     public function list(array $arrWith = [], array $where = [], string $orderBy = 'id') {
        if(isset($where['field']) && isset($where['value'])) {
                                                                return $this->getModel()->with($arrWith)->where($where['field'], $where['value'])
                ->orderBy($orderBy)->get();
        }
                                                                return $this->getModel()->with($arrWith)->orderBy($orderBy)->get();
    }

     public function listPaginate(array $arrWith = [], array $where = [], string $orderBy = 'id', int $limit = 6) {
        if(isset($where['field']) && isset($where['value'])) {
                                                                return $this->getModel()->with($arrWith)->where($where['field'], $where['value'])
                ->orderBy($orderBy)->paginate($limit);
        }
                                                                return $this->getModel()->with($arrWith)->orderBy($orderBy)->paginate($limit);
    }
```

```
    public function find(int|string $id, array $arrWith = []): ?Model {
        return $this->getModel()->with($arrWith)->find($id);
    }

    public function store(array $data): ?Model {
        return $this->getModel()->create($data);
    }

    public function update(array $data, int|string $id): ?Model {
        $row = $this->getModel()->findOrFail($id);
        $row->update($data);
        return $row;
    }

    public function remove(int|string $id): ?Model {
        $row = $this->getModel()->findOrFail($id);
        return $row->delete();
    }
}
```

Arquivo: “Repositories/CursoRepository.php”

```php
<?php

namespace App\Repositories;

use App\Models\Curso;

class CursoRepository extends BaseRepository {

    public function __construct(protected Curso $model) {}

    protected function getModel(): mixed {
        return $this->model;
    }
}
```

Arquivo: “Repositories/DisciplinaRepository.php”

```php
<?php

namespace App\Repositories;

use App\Models\Disciplina;

class DisciplinaRepository extends BaseRepository {

    public function __construct(protected Disciplina $model) { }

    protected function getModel(): mixed {
        return $this->model;
    }
}
```

Arquivo: "Repositories/PermissionRepository.php"

```php
<?php

namespace App\Repositories;

use App\Models\Permission;

class PermissionRepository extends BaseRepository {

    public function __construct(protected Permission $model) {}

    protected function getModel(): mixed {
        return $this->model;
    }
}
```

2) Serviços (Services): Criando Diretório e Arquivos

Estrutura, dentro da pasta "**app/**" crie (manualmente) o diretório ***"Services/"*** com os seguintes arquivos:

- **Services/**
  - BaseService.php
  - CursoService.php

- DisciplinaService.

Arquivo: “Services/BaseService.php”

```php
<?php

namespace App\Services;

abstract class BaseService {

    abstract protected function getRepository(): mixed;

     public function all(array $arrWith = [], array $where = [], string $orderBy = 'id') {
                return  $this->getRepository()->list($arrWith,  $where, $orderBy);
    }

     public function allPaginate(array $arrWith = [], array $where = [], string $orderBy = 'id', int $limit = 6) {
          return $this->getRepository()->listPaginate($arrWith, $where, $orderBy, $limit);
    }

    public function find(int|string $id, array $with = []) {
        return $this->getRepository()->find($id, $with);
    }

    public function store(array $data) {
        return $this->getRepository()->store($data);
    }

    public function update(array $data, int|string $id) {
        return $this->getRepository()->update($data, $id);
    }

    public function remove(int|string $id) {
        return $this->getRepository()->remove($id);
    }
}
```

Arquivo: “Services/CursoService.php”

```php
<?php

namespace App\Services;

use App\Repositories\CursoRepository;

class CursoService extends BaseService {

     public function __construct(protected CursoRepository $repository)
{}

    protected function getRepository(): mixed {
        return $this->repository;
    }
}
```

Arquivo: “Services/DisciplinaService.php”

```php
<?php

namespace App\Services;

use App\Repositories\DisciplinaRepository;

class DisciplinaService extends BaseService {

          public   function   __construct(protected   DisciplinaRepository
$repository) { }

    protected function getRepository(): mixed {
        return $this->repository;
    }
}
```

Arquivo: “Services/PermissionService.php”

```php
<?php
```

```php
namespace App\Services;

use App\Repositories\PermissionRepository;

class PermissionService extends BaseService {

          public   function   __construct(protected   PermissionRepository
$repository) {}

    protected function getRepository(): mixed {
        return $this->repository;
    }

    public function loadPermissions($role) {

        $arr_permissions = Array();
             $perm  =  $this->repository->list(['resource'],  ['field'  =>
'role_id', 'value' => $role], 'resource_id');

        foreach($perm as $item) {
            $arr_permissions[$item->resource->name] = true;
        }
        // dd($arr_permissions);
        session(['user_permissions' => $arr_permissions]);
    }

    public function isAuthorized($resource) {
        $permissions = session('user_permissions');

        if(array_key_exists($resource, $permissions)) {
            return true;
        }
        return false;
    }
}
```

3) Controladoras (Controllers): Utilizando os Services

Arquivo: “Controllers/CursoController.php”

PHP

```php
<?php

namespace App\Http\Controllers;

use App\Models\Curso;
use App\Http\Requests\CursoRequest;
use Illuminate\Support\Facades\Gate;
use App\Services\CursoService;

class CursoController extends Controller {

    public function __construct(protected CursoService $service) {}

    public function index() {
        Gate::authorize('viewAny', Curso::class);
             $data  =  $this->service->all(['disciplina',  'aluno'],  [],
'nome');
        return view('curso.index', compact(['data']));
    }

    public function create() {
        Gate::authorize('create', Curso::class);
        return view('curso.create');
    }

    public function store(CursoRequest $request) {
        Gate::authorize('create', Curso::class);
        $this->service->store($request->validated());
        return redirect()->route('curso.index');
    }

    public function show(string $id) {
        $curso = $this->service->find($id);
        Gate::authorize('view', $curso);

        if(isset($curso)) {
            return view('curso.show', compact(['curso']));
        }

        return "<h1>Curso não encontrado!</h1>";
    }
```

```php
    public function edit(string $id) {
        $curso = $this->service->find($id);
        Gate::authorize('update', $curso);

        if(isset($curso)) {
            return view('curso.edit', compact(['curso']));
        }

        return "<h1>Curso não encontrado!</h1>";
    }

    public function update(CursoRequest $request, string $id) {
        $curso = $this->service->find($id);
        Gate::authorize('update', $curso);

        if(isset($curso)) {
            $this->service->update($request->validated(), $id);
            return redirect()->route('curso.index');
        }

        return "<h1>Curso não encontrado!</h1>";
    }

    public function destroy(string $id) {
        $curso = $this->service->find($id);
        Gate::authorize('delete', $curso);

        if(isset($curso)) {
            $this->service->remove($id);
            return redirect()->route('curso.index');
        }

        return "<h1>Curso não encontrado!</h1>";
    }
}
```

Arquivo: “Controllers/DisciplinaController.php”

```php
<?php
```

```php
namespace App\Http\Controllers;

use App\Models\Disciplina;
use App\Http\Requests\DisciplinaRequest;
use App\Services\CursoService;
use App\Services\DisciplinaService;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Support\Facades\Gate;


class DisciplinaController extends Controller {

    public function __construct(
        protected DisciplinaService $service,
        protected CursoService $cursoService
    ) {}

    public function index() {
        Gate::authorize('viewAny', Disciplina::class);
        $data = $this->service->all(['curso'], [], 'nome');
        return view('disciplina.index', compact(['data']));
    }

    public function create() {
        Gate::authorize('create', Disciplina::class);
        $cursos = $this->cursoService->all([], [], 'nome');
        return view('disciplina.create', compact(['cursos']));
    }

    public function store(DisciplinaRequest $request) {
        Gate::authorize('create', Disciplina::class);
        $this->service->store($request->validated());
        return redirect()->route('disciplina.index');
    }

    public function show(string $id) {
        $disciplina = $this->service->find($id, ['curso']);
        Gate::authorize('view', $disciplina);

        if(isset($disciplina)) {
            return view('disciplina.show', compact(['disciplina']));
        }
```

```
        return "<h1>Disciplina não encontrada!</h1>";
    }

    public function edit(string $id) {
        $disciplina = $this->service->find($id, ['curso']);
        Gate::authorize('update', $disciplina);
        $cursos = $this->cursoService->all([], [], 'nome');

        if(isset($disciplina)) {
                  return view('disciplina.edit', compact(['disciplina',
'cursos']));
        }

        return "<h1>Disciplina não encontrada!</h1>";
    }

    public function update(DisciplinaRequest $request, string $id) {
        $disciplina = $this->service->find($id);
        Gate::authorize('update', $disciplina);

        if(isset($disciplina)) {
            $this->service->update($request->validated(), $id);
            return redirect()->route('disciplina.index');
        }

        return "<h1>Disciplina não encontrada!</h1>";
    }

    public function destroy(string $id) {

        $disciplina = $this->service->find($id);
        Gate::authorize('delete', $disciplina);

        if(isset($disciplina)) {
            $this->service->remove($id);
            return redirect()->route('disciplina.index');
        }

        return "<h1>Disciplina não encontrada!</h1>";
    }
}
```

4) Alterando o Listener de Permissões (Trocando Controller por Service)

Arquivo: "Listeners/AuthenticationListener.php"

```php
<?php

namespace App\Listeners;

use App\Events\AuthenticationEvent;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Queue\InteractsWithQueue;
use App\Services\PermissionService;

class AuthenticationListener
{
    /**
     * Create the event listener.
     */
    public function __construct(protected PermissionService $service)
    {
        //
    }

    /**
     * Handle the event.
     */
    public function handle(AuthenticationEvent $event): void
    {
        $this->service->loadPermissions($event->data);
    }
}
```

5) Alterando as Permissões nas Policies (Curso e Disciplina)

Arquivo: "Policies/CursoPolicy.php"

```php
<?php

namespace App\Policies;
```

```php
use App\Models\Curso;
use App\Models\User;
use App\Services\PermissionService;

class CursoPolicy {

     public function __construct(protected PermissionService $service)
{}

    public function viewAny(User $user): bool {
        return $this->service->isAuthorized('curso.index');
    }

    public function view(User $user, Curso $curso): bool {
        return $this->service->isAuthorized('curso.show');
    }

    public function create(User $user): bool {
         return $this->service->isAuthorized('curso.create');
    }

    public function update(User $user, Curso $curso): bool {
        return $this->service->isAuthorized('curso.edit');

    }

    public function delete(User $user, Curso $curso): bool {
        return $this->service->isAuthorized('curso.delete');

    }
    public function restore(User $user, Curso $curso): bool {
        return $this->service->isAuthorized('curso.delete');
    }

    public function forceDelete(User $user, Curso $curso): bool
    {
        return $this->service->isAuthorized('curso.delete');
    }
}
```

Arquivo: "Policies/DisciplinaPolicy.php"

PHP

```php
<?php

namespace App\Policies;

use App\Models\Disciplina;
use App\Models\User;
use App\Services\PermissionService;

class DisciplinaPolicy {

     public function __construct(protected PermissionService $service)
{}

    public function viewAny(User $user): bool {
        return $this->service->isAuthorized('disciplina.index');
    }

    public function view(User $user, Disciplina $disciplina): bool {
        return $this->service->isAuthorized('disciplina.show');
    }

    public function create(User $user): bool {
        return $this->service->isAuthorized('disciplina.create');
    }

    public function update(User $user, Disciplina $disciplina): bool {
        return $this->service->isAuthorized('disciplina.edit');
    }

    public function delete(User $user, Disciplina $disciplina): bool {
        return $this->service->isAuthorized('disciplina.delete');
    }

    public function restore(User $user, Disciplina $disciplina): bool {
        return $this->service->isAuthorized('disciplina.delete');
    }

     public function forceDelete(User $user, Disciplina $disciplina):
bool {
        return $this->service->isAuthorized('disciplina.delete');
```

```
    }
}
```

6) Testando a Aplicação / Recursos de Autenticação

Comando
(no terminal de comando - dentro da pasta da aplicação)

Subindo a Aplicação

Shell
```
docker-compose up
```

7) Auditoria: Instalando e Configurando o Laravel Auditing

Comando
(no terminal de comando - dentro da pasta da aplicação)

Instalação

Shell
```
docker-compose exec app composer require owen-it/laravel-auditing
```

Comando
(no terminal de comando - dentro da pasta da aplicação)

Configuração/Publicação

Shell
```
docker-compose exec --user $(id -u):$(id -g) app php artisan
vendor:publish
--provider="OwenIt\Auditing\AuditingServiceProvider"
```

Comando
(no terminal de comando - dentro da pasta da aplicação)

Efetuando Migração - Criando Tabela de Auditoria

```shell
docker-compose exec app php artisan migrate:fresh --seed
```

8) Aplicando Auditoria - Cursos

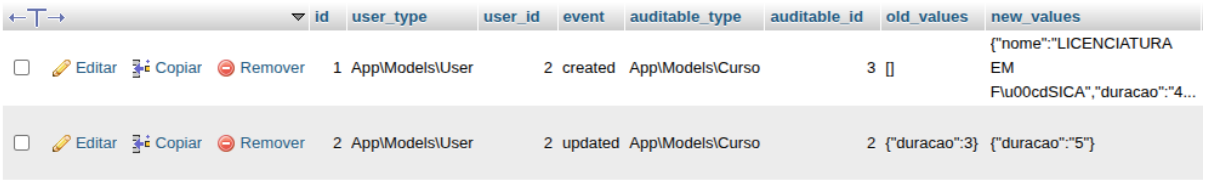
Arquivo: "Models/Curso.php"

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use \Illuminate\Database\Eloquent\SoftDeletes;
use OwenIt\Auditing\Contracts\Auditable;  // Linha Adicionada
use OwenIt\Auditing\Auditable as AuditableTrait; // Linha Adicionada

class Curso extends Model implements Auditable { // Linha Alterada

    use AuditableTrait;       // Linha Adicionada
    use SoftDeletes;

    protected $fillable = [
        'nome',
        'duracao',
    ];

    public function disciplina() {
        return $this->hasMany('\App\Models\Disciplina');
    }

    public function aluno() {
        return $this->hasMany('\App\Models\Aluno');
    }
}
```

9) Testando Auditoria

a) Cadastrando um novo curso - Licenciatura em Física
b) Alterando um Curso - TDAS, mudando duração para 5 anos

c) Verificando Dados Inseridos na Tabela ***"audits"***



	id	user_type	user_id	event	auditable_type	auditable_id	old_values	new_values
Editar Copiar Remover	1	App\Models\User	2	created	App\Models\Curso	3	[]	{"nome":"LICENCIATURA EM F\u00cdSICA","duracao":"4...
Editar Copiar Remover	2	App\Models\User	2	updated	App\Models\Curso	2	{"duracao":3}	{"duracao":"5"}

10) Exibindo as Informações de Auditoria - Curso

"routes/app.php"

```php
Route::get('/audit/curso/{id}', [CursoController::class, 'audit'])
        ->name('curso.audit')->middleware(['auth', 'verified']);
```

"app/Http/Controllers/BaseRepository.php"

```php
public function audit(int|string $id) {
    $row = $this->getModel()->findOrFail($id);

    return $row->audits()->with('user')->latest()->get()->transform(
        function ($audit) {
            $old = is_string($audit->old_values)
            ?
                json_decode($audit->old_values, true)
            :
                (array) $audit->old_values;
            $new = is_string($audit->new_values)
            ?
                json_decode($audit->new_values, true)
```

```
            :
                (array)$audit->new_values;

            // Força todos os valores internos do array a virarem strings
            $audit->old_values = array_map('strval', $old ?? []);
            $audit->new_values = array_map('strval', $new ?? []);

            return $audit;
        });
    }
```

"app/Http/Controllers/BaseService.php"

```php
public function audit(int|string $id) {
    return $this->getRepository()->audit($id);
}
```

"app/Http/Controllers/CursoController.php"

```php
public function audit(string $id) {
    $curso = $this->service->find($id);
    Gate::authorize('delete', $curso);

    if(isset($curso)) {
        $data = $this->service->audit($id);
        return view('curso.audit', compact(['data']));
    }

    return "<h1>Não encontrado!</h1>";
}
```

"views/curso/index.blade.php"

```php
// Acionado
@can('delete', $item)
   <a href="{{route('curso.audit', $item->id)}}" class="...">
      <svg ...>
          <path .../>
```

```
      </svg>
   </a>
@endcan
```

“views/curso/audit.blade.php”

```php
@extends('template/main',
    [
        'titulo'=>"Sistema Aula",
        'cabecalho' => 'Lista de Cursos',
        'rota' => '',
    ]
)
@section('conteudo')

     <table class="table align-middle caption-top table-striped">
        <thead>
            <th class="text-secondary">AÇÃO</th>
            <th class="text-secondary">AUTOR</th>
            <th class="d-none d-md-table-cell text-secondary">DATA
HORA</th>
            <th class="d-none d-md-table-cell text-secondary">MUDOU
DE</th>
            <th class="text-secondary">PARA</th>
        </thead>
        <tbody>
            @foreach ($data as $audit)
                 <tr>
                    <td>{{ strtoupper($audit->event) }}</td>
                    <td>{{ $audit->user->name ?? 'Sistema/Console'
}}</td>
                    <td class="d-none d-md-table-cell">{{
$audit->created_at->format('d/m/Y H:i') }}</td>
                    <td class="d-none d-md-table-cell">
                        @if(count($audit->old_values) == 0)
                            -
                        @else
                            @foreach ($audit->old_values as $key =>
$value)
```

```
                                <p><strong>{{ strtoupper($key)
}}:</strong> {{ $value }}</p>
                            @endforeach
                        @endif
                    </td>
                    <td>
                        @if(count($audit->new_values) == 0)
                            -
                        @else
                            @foreach ($audit->new_values as $key =>
$value)
                                <p><strong>{{ strtoupper($key)
}}:</strong> {{ $value }}</p>
                            @endforeach
                        @endif
                    </td>
                </tr>
            @endforeach
        </tbody>
    </table>

@endsection
```

11) Testando Exibição dos Dados de Auditoria