PDV360 · DOCUMENTAÇÃO TÉCNICA

Estrutura de Distritos e Setores

Análise, arquitetura, implementação, validações e testes da hierarquia Distrito → Setor → Lojas/Consultores no backend.

PROJETO	pdv-api (Laravel)
DATA	30 de agosto de 2026
AUTOR	Enzo Moura, com assistência de Claude Code
ESCOPO	Hierarquia organizacional, camadas de validação cross-distrito e melhorias de consistência de dados

- 17/17 testes automatizados
- 90 assertions
- Pint OK
- Sem breaking changes

SUMÁRIO

1. Índice do documento

SEÇÃO 01

Requisito original

O pedido, na íntegra (tradução livre da mensagem do usuário), definia as seguintes regras de negócio:

- O **Distrital** é responsável por criar e gerenciar os **distritos**.
- No cadastro do usuário deve existir vínculo com o termo/consultor *e* com o setor correspondente.
- Cada distrito possui seus próprios setores.
- As lojas devem ser vinculadas a setores.
- Cada consultor pertence a um único setor, alocado no momento do cadastro.
- No cadastro/edição de loja deve ser possível vincular a um setor dentro do distrito atual.
- A divisão dos dados respeita: **Distrito → Setor → Lojas / Consultores**.
- Consultas e listagens devem trazer apenas dados do distrito/setor ao qual o usuário está vinculado.
- Reaproveitar padrões existentes; não duplicar lógica se já houver estrutura de distrito/setor/loja/usuário aproveitável.
- Vínculos devem ser validados no backend, impedindo loja ou consultor associados a setor de outro distrito.

> i O usuário pediu explicitamente para analisar o backend (pdv-api) e o frontend web (pdv-web) antes de implementar, reaproveitando estruturas já existentes em vez de criar lógica paralela.

SEÇÃO 02

Análise da estrutura existente

Antes de qualquer código, o backend foi inspecionado para entender o que já existia.

2.1 Hierarquia de papéis (já existente)

PAPEL (`USERROLE`)	APELIDO NO DOMÍNIO	ESCOPO DE ACESSO
`GerenteNacional`	Nacional	Irrestrito — enxerga tudo
`GerenteDistrital`	Distrital	Gerencia um grupo de consultores via `district_manager_id`
`Dermoconsultor`	Consultor / "termo"	Vinculado a um único distrital e (agora) a um único setor

2.2 O que já existia sobre "distrito"

Um achado central da análise: **não existia entidade `Distrito` no banco de dados**. O único vínculo de "distrito" era o auto-relacionamento `users.district_manager_id` (um Dermoconsultor aponta para o `GerenteDistrital` responsável por ele). Além disso, a tabela `stores` já tinha uma coluna chamada `district` — mas essa coluna é um **texto livre de bairro/endereço**, sem nenhuma relação com hierarquia organizacional. Essa distinção foi importante para não confundir os dois conceitos durante a implementação.

2.3 Setor: já existia parcialmente

Em uma sessão anterior já havia sido implementada a entidade `Setor` (agrupando lojas e dermoconsultores), porém **de forma independente da hierarquia de distrito** — não havia `distrito_id` em lugar nenhum, e os arquivos (`SetorController`, `SaveSetorRequest`, migration, model) ainda estavam **não commitados** no repositório (apareciam como `??` no `git status`). A estrutura de Setor existia, mas faltava exatamente a camada que o pedido descreve como topo da hierarquia — o Distrito.

2.4 Padrões de código reaproveitados

- **FormRequest** por recurso, com regras condicionais via closures quando dependem do usuário autenticado.
- **Policies** para autorização por instância, além de um **Gate** global `gerenciar-cadastros`.
- **API Resources** para serialização, com `whenLoaded` / `whenCounted`.
- Autenticação **JWT própria** (não Sanctum/sessão) via `JwtService`.
- Precedente de **404 em vez de 403** para esconder existência de registros fora do escopo do ator (`StoreController::unassign()`).

> ✓ Decisão: nenhuma lógica paralela foi criada. O Setor já existente foi *estendido* para pertencer a um Distrito, e o Distrito foi adicionado como camada nova acima dele — sem tocar no mecanismo `district_manager_id` já em produção.

SEÇÃO 03

Decisões de arquitetura

3.1 Distrito como entidade nova, aditiva

Como não existia representação de "distrito" como registro de banco, foi criada a tabela e o model `Distrito`. A decisão consciente foi **não substituir** `district_manager_id` por essa nova entidade: o mecanismo de hierarquia gerente→consultor já é usado em várias regras de negócio existentes (transferências, visibilidade de rotas, aprovações). Reescrevê-lo teria um raio de impacto muito maior do que o pedido exige. Em vez disso, o Distrito foi encaixado *ao lado*: o Distrital ganhou uma coluna `distrito_id` (qual distrito ele administra), e o Setor ganhou `distrito_id` (a qual distrito pertence). O `district_manager_id` continua controlando quem gerencia quem; o `distrito_id` controla a que distrito cada setor/gerente pertence.

3.2 Hipótese sobre "termo/consultor"

O pedido usa a expressão "termo/consultor". Não existe campo ou tabela "termo" em nenhum lugar do pdv-api ou do pdv-web. A hipótese assumida — e sinalizada ao usuário em vez de implementada silenciosamente — foi tratar "termo/consultor" como sinônimo do `Dermoconsultor` já existente, vinculado ao distrital via `district_manager_id`.

3.3 Setor obrigatório, Distrito do gerente opcional

Um Setor **sempre** pertence a um distrito (`distrito_id` é `NOT NULL` com `restrictOnDelete()`). Já o `distrito_id` em `users` é **nullable**: um Distrital recém-criado ainda não administra nenhum distrito até criar um (ou ser vinculado por um Nacional) — reflete o fluxo real de que o distrito nasce depois do usuário, não antes.

3.4 Quem pode criar o quê

AÇÃO	NACIONAL	DISTRITAL
Criar distrito	Sim, livre	Sim, apenas se ainda não administra nenhum (auto-vincula ao criar)
Criar setor	Sim, deve informar `distrito_id`	Sim, apenas no próprio distrito (forçado pelo controller)
Editar setor	Sim, qualquer um	Apenas os do próprio distrito; não pode mover para outro
Vincular loja a setor	Sim, qualquer setor	Apenas setor do próprio distrito
Vincular consultor a setor	Sim, qualquer setor	Apenas setor do próprio distrito

3.5 Por que 404 em vez de 403/422 em alguns pontos

Sempre que um Distrital tenta *acessar diretamente* (via `show` / `update`) um registro de outro distrito, a resposta é **404**, não **403** — retornar 403 confirmaria que o registro existe, vazando informação sobre recursos fora do escopo

do usuário. Quando o problema está no *corpo da requisição* (por exemplo, `setor_id` de outro distrito), a resposta é **422**, porque o próprio ator já sabe da tentativa — não há vazamento de informação nova.

SEÇÃO 04

Modelo de dados e hierarquia

`User.distrito_id` (nullable) → qual distrito o GerenteDistrital administra
`User.district_manager_id` (já existia) → hierarquia gerente/consultor, inalterada

4.1 Tabelas

TABELA	COLUNAS NOVAS	REGRA DE INTEGRIDADE
`distritos` (nova)	`id`, `name` (unique), `status` (default `ATIVO`), timestamps	—
`setores`	`distrito_id` (FK, NOT NULL)	`restrictOnDelete()` — não é possível apagar um distrito com setores
`users`	`distrito_id` (FK, nullable, após `district_manager_id`)	`nullOnDelete()`
`stores`	`setor_id` (da sessão anterior de Setor)	nullable — loja pode não ter setor ainda

Migration: **`database/migrations/2026_08_30_190000_create_distritos_table.php`**. O `down()` desfaz na ordem inversa para respeitar as FKs.

SEÇÃO 05

Implementação — arquivos novos

+ NOVO **app/Models/Distrito.php**

Distrito.php

```
class Distrito extends Model
{
    protected $fillable = ['name', 'status'];
    protected $attributes = ['status' => 'ATIVO'];

    public function setores(): HasMany
    {
        return $this->hasMany(Setor::class);
    }

    /** Gerentes distritais que administram este distrito. */
    public function managers(): HasMany
    {
        return $this->hasMany(User::class);
    }
}
```

+ NOVO **app/Http/Controllers/Api/DistritoController.php**

CRUD completo, exceto `destroy` (distritos não são removidos via API — consistente com o restante do domínio, que usa *soft status* `ATIVO` / `INATIVO` em vez de exclusão física).

DistritoController.php

```
public function index(Request $request): AnonymousResourceCollection
{
    $user = $request->user();
    $distritos = Distrito::query()
        ->withCount('setores', 'managers')
        // Distrital só enxerga o próprio distrito.
        ->when($user->isDistrict(), fn ($q) => $q->where('id', $user->distrito_id))
        ->when($request->filled('status'), fn ($q) => $q->where('status', $request->string('status')))
        ->when($request->filled('search'), fn ($q) => $q->where('name', 'like', '%'.$request->string('search').'%'))
        ->orderBy('name')->paginate($request->integer('per_page', 20))->withQueryString();

    return DistritoResource::collection($distritos);
}

/** Distrital que ainda não administra um distrito passa a administrar o que criar. */
public function store(SaveDistritoRequest $request): DistritoResource
{
    $this->authorize('gerenciar-cadastros');
    $user = $request->user();

    if ($user->isDistrict() && $user->distrito_id !== null) {
        throw ValidationException::withMessages(['distrito_id' => 'Você já gerencia um distrito.']);
    }

    $distrito = Distrito::create($request->validated());
    if ($user->isDistrict()) {
        $user->update(['distrito_id' => $distrito->id]);
    }

    return new DistritoResource($distrito);
}
```

O ponto central desse controller: um Distrital só pode criar **um** distrito (o primeiro que criar passa a ser o seu), e `show` / `update` devolvem 404 para distritos que não são o seu — mesmo padrão de escopo aplicado depois em Setor e Loja.

+ NOVO **app/Http/Requests/SaveDistritoRequest.php**

Espelha `SaveSetorRequest`: `name` obrigatório na criação / opcional na edição, máx. 120 caracteres, único ignorando o próprio registro; `status` em `ATIVO` / `INATIVO`.

+ NOVO **app/Http/Resources/DistritoResource.php**

Expõe `id`, `name`, `status`, `setores_count` e `managers_count` (via `whenCounted`, populados pelo `withCount` dos controllers).

+ COMPLETADO NESTA SESSÃO **Entidade Setor**

A entidade Setor já havia sido escrita numa sessão anterior, mas ainda não commitada e sem vínculo com distrito. Nesta sessão ela foi completada com `distrito_id` e o relacionamento `belongsTo(Distrito::class)`:

Setor.php

```
class Setor extends Model
{
    protected $table = 'setores';
    protected $fillable = ['distrito_id', 'name', 'status'];
    protected $attributes = ['status' => 'ATIVO'];

    public function distrito(): BelongsTo { return $this->belongsTo(Distrito::class); }
    public function stores(): HasMany { return $this->hasMany(Store::class); }
    public function users(): HasMany { return $this->hasMany(User::class); }
}
```

`SetorController` — regra central: Distrital tem o `distrito_id` forçado (o enviado no payload é ignorado); Nacional precisa informar qual distrito:

SetorController.php

```
public function store(SaveSetorRequest $request): SetorResource
{
    $this->authorize('gerenciar-cadastros');
    $data = $request->validated();
    $user = $request->user();

    if ($user->isDistrict()) {
        if ($user->distrito_id === null) {
            throw ValidationException::withMessages([
                'distrito_id' => 'Você precisa gerenciar um distrito antes de criar
setores.',
            ]);
        }
        $data['distrito_id'] = $user->distrito_id; // ignora valor enviado pelo cliente
    } elseif (empty($data['distrito_id'])) {
        throw ValidationException::withMessages(['distrito_id' => 'Informe o distrito do
setor.']);
    }

    return new SetorResource(Setor::create($data)->load('distrito'));
}
```

`index()` filtra por `distrito_id` do usuário quando ele é Distrital; `show()` / `update()` usam o mesmo `abort_unless(..., 404)` de escopo; `update()` remove `distrito_id` do payload quando o ator é Distrital.

SEÇÃO 06

Implementação — arquivos modificados

Resumo do que mudou em cada arquivo já existente, e por quê.

~ MODIFICADO **app/Models/User.php**

- Adicionado `distrito_id` e `setor_id` a `$fillable`.
- Nova relação `distrito(): BelongsTo` — "distrito que este gerente administra".
- Relação `setor(): BelongsTo` — setor do consultor.

~ MODIFICADO **app/Models/Store.php**

- `setor_id` adicionado a `$fillable`.
- Nova relação `setor(): BelongsTo`.

~ MODIFICADO **SaveStoreRequest.php · SaveUserRequest.php**

Ambos ganharam a **mesma** regra de validação (closure) para `setor_id`:

SaveStoreRequest.php / SaveUserRequest.php

```
'setor_id' => ['nullable', 'integer', 'exists:setores,id', function ($attribute, $value,
$fail) {
    $user = $this->user();
    if ($value && $user?->isDistrict() && Setor::find($value)?->distrito_id !== $user-
>distrito_id) {
        $fail('Este setor não pertence ao seu distrito.');
    }
}],
```

`SaveUserRequest` ganhou também `distrito_id` (nullable, `exists:distritos,id`) — usado quando o Nacional cadastra/edita um Distrital diretamente.

~ MODIFICADO **app/Policies/StorePolicy.php**

Antes, qualquer gerente podia ver/editar/atribuir qualquer loja. Agora, um Distrital só pode fazer isso dentro do próprio distrito:

StorePolicy.php

```
public function view(User $actor, Store $store): bool
{
    if ($actor->role->isManager()) { return $this->withinOwnDistrito($actor, $store); }
    return $store->activeAssignments()->where('user_id', $actor->id)->exists();
}

/** Distrital só administra lojas de setores do próprio distrito. */
private function withinOwnDistrito(User $actor, Store $store): bool
{
    return ! $actor->isDistrict() || $store->setor?->distrito_id === $actor->distrito_id;
}
```

Nacional não é afetado: `isDistrict()` é `false` para ele, logo a checagem sempre retorna `true` no primeiro termo do `||`.

~ MODIFICADO StoreController.php

- `index()` restringe lojas às do distrito do Distrital via `whereHas('setor', ...)`.
- Novos filtros de query string: `?setor_id=` e `?distrito_id=`.
- `show` / `store` / `update` passam a carregar a relação `setor`.

~ MODIFICADO UserController.php

- Novos filtros: `?setor_id=` e `?distrito_id=` (cobre tanto o Distrital que administra aquele distrito quanto o consultor cujo setor pertence a ele).
- Todos os pontos que retornam `UserResource` passam a carregar `setor` e `distrito`.

~ MODIFICADO StoreResource.php · UserResource.php

Ambos ganharam blocos `setor_id` / `setor` (via `whenLoaded`); `UserResource` ganhou também `distrito_id` / `distrito`.

~ MODIFICADO database/seeders/DemoSeeder.php

Passa a criar um `Distrito` ("Distrito Fortaleza") antes do usuário Distrital, vincula o Distrital a ele, cria um `Setor` ("Setor Centro-Aldeota") dentro desse distrito, e vincula o Dermoconsultor e as lojas de exemplo a esse setor — o ambiente de demonstração já nasce com a hierarquia completa preenchida.

~ MODIFICADO routes/api.php

routes/api.php

```
// Distritos: nível organizacional acima do setor (Distrito -> Setor ->
Lojas/Consultores).
Route::apiResource('distritos', DistritoController::class)->except('destroy')
    ->parameters(['distritos' => 'distrito']);

// Setores: agrupam lojas e dermoconsultores dentro de um distrito.
Route::apiResource('setores', SetorController::class)->except('destroy')
    ->parameters(['setores' => 'setor']);
```

SEÇÃO 07

Camadas de validação (defense in depth)

A regra "garanta que os vínculos sejam validados no backend para evitar que uma loja ou consultor seja associado a um setor de outro distrito" foi implementada em **quatro camadas independentes**, cada uma cobrindo um ponto de entrada diferente. Mesmo que uma camada seja contornada ou removida por engano no futuro, as outras ainda protegem a invariante.

CAMADA	ONDE	O QUE IMPEDE	HTTP
1 · FormRequest	`SaveStoreRequest`, `SaveUserRequest`	Distrital enviando `setor_id` de um setor fora do seu distrito.	422
2 · Controller	`SetorController`, `DistritoController`	Distrital forçando `distrito_id` de outro distrito (ignorado); acesso direto a setor/distrito de outro distrito.	201 forçado / 404
3 · Policy	`StorePolicy::view/update/assign`	Distrital acessando, editando ou atribuindo consultor a loja de outro distrito.	404
4 · Domain Service	`UserService::guardSetorDistrito()`	Inconsistência entre `setor_id` do consultor e `distrito_id` do gerente — cobre também o Nacional, que atravessa distritos livremente.	422

! As camadas 1–3 protegem contra um **ator Distrital** malicioso ou descuidado. A camada 4 protege contra uma **inconsistência de dados** que pode ocorrer mesmo com um ator Nacional legítimo — que tem permissão para atravessar distritos, mas não deveria conseguir criar um estado inconsistente.

SEÇÃO 08

Melhorias adicionais (segunda rodada)

Após a implementação inicial atender ponto a ponto o pedido original, foi feita uma revisão adicional por iniciativa própria, focada em fechar uma lacuna de **integridade de dados** (não apenas de autorização) que as camadas 1–3 não cobriam.

8.1 Lacuna encontrada: transferência de consultor não revalidava o setor

`UserService::transfer()` move um Dermoconsultor de um Distrital para outro. As validações em `SaveStoreRequest` / `SaveUserRequest` só entram em ação quando o próprio Distrital está editando; a transferência, porém, é operação exclusiva do **Nacional**. Antes da correção, era possível transferir um consultor para um gerente cujo distrito não tinha relação nenhuma com o `setor_id` atual do consultor.

8.2 Correção: guard compartilhado entre create/update/transfer

UserService.php

```
/** Consultor e gerente distrital precisam pertencer ao mesmo distrito. */
private function guardSetorDistrito(?int $setorId, ?int $managerId): void
{
    if ($setorId === null || $managerId === null) { return; }

    $setorDistritoId = Setor::find($setorId)?->distrito_id;
    $managerDistritoId = User::find($managerId)?->distrito_id;

    if ($setorDistritoId !== null && $managerDistritoId !== null && $setorDistritoId !==
$managerDistritoId) {
        throw ValidationException::withMessages([
            'setor_id' => 'O setor informado não pertence ao distrito do gerente
distrital.',
        ]);
    }
}
```

Reutilizado nos três pontos de entrada do serviço de usuários:

- `create()` — antes de `User::create($data)`.
- `update()` — antes de aplicar as mudanças, usando o valor novo ou o atual como fallback.
- `transfer()` — antes de mover o consultor, usando seu `setor_id` atual contra o `distrito_id` do gerente de destino.

Um guard irmão, `guardDistrito()`, também foi adicionado para impedir que `distrito_id` seja setado em qualquer papel que não seja `GerenteDistrital`.

8.3 Filtros de consulta por distrito

Adicionado `?distrito_id=` em `GET /api/stores` e `GET /api/users`, permitindo que um Nacional filtre a listagem por distrito sem precisar descobrir os `setor_id` daquele distrito manualmente.

Nenhuma dessas melhorias altera contratos de API existentes: são filtros novos, opcionais, e uma validação adicional que só bloqueia estados já logicamente inválidos — nenhum teste pré-existente quebrou.

SEÇÃO 09

Endpoints da API

MÉTODO	ROTA	DESCRIÇÃO	STATUS
GET	`/api/distritos`	Lista distritos (Distrital só vê o próprio)	Novo
GET	`/api/distritos/{distrito}`	Detalhe (404 se fora de escopo)	Novo
POST	`/api/distritos`	Cria distrito; auto-vincula o Distrital criador	Novo
PUT	`/api/distritos/{distrito}`	Atualiza (404 se fora de escopo)	Novo
GET	`/api/setores`	Lista setores; filtros `distrito_id`, `status`, `search`	Completado
GET	`/api/setores/{setor}`	Detalhe (404 se fora de escopo)	Completado
POST	`/api/setores`	Cria setor; `distrito_id` forçado para Distrital	Completado
PUT	`/api/setores/{setor}`	Atualiza (Distrital não move de distrito)	Completado
GET	`/api/stores?setor_id=&distrito_id=`	Filtros novos + escopo automático por distrito	Alterado
GET	`/api/users?setor_id=&distrito_id=`	Filtros novos	Alterado
POST	`/api/users/{user}/transfer`	Agora valida consistência setor/distrito no destino	Alterado

SEÇÃO 10

Testes e verificação

10.1 Suíte automatizada

`tests/Feature/Pdv360FlowTest.php` ganhou dois novos testes ponta a ponta (via HTTP real, autenticando com JWT emitido por `JwtService`):

- `test_distrito_scopes_setores_and_blocks_cross_district_links()` — confirma que o Distrital só lista/vê setores do próprio distrito (404 no alheio); que forçar `distrito_id` de outro distrito é ignorado; e 422 ao vincular loja/consultor a setor de outro distrito.
- `test_transfer_blocks_setor_from_a_different_distrito()` — transferência para gerente de outro distrito é bloqueada com 422 em `setor_id`.

```
php artisan test
```


```
{"tool":"phpunit","result":"passed","tests":17,"passed":17,"assertions":90,"duration_ms":419}
```


Progressão ao longo da sessão: 15 testes (linha de base) → 16 → 17. Todos verdes em cada etapa.

10.2 Estilo de código

`./vendor/bin/pint --dirty --test` apontou violações de `fully_qualified_strict_types` e `ordered_imports` em seis arquivos. Corrigido com auto-fix; suíte re-executada e 100% verde depois da formatação.

10.3 Verificação manual

Com o servidor local rodando e o banco SQLite resetado (`migrate:fresh --seed` , seguro por ser um banco de demonstração reproduzível pelo `DemoSeeder`), foram exercitados via `curl` : login Nacional/Distrital, escopo de `/api/setores` e `/api/distritos` , forçar `distrito_id` de outro distrito na criação (ignorado), vincular loja a setor de outro distrito (422), acesso direto a setor/distrito fora de escopo (404). Todos os comportamentos bateram com o desenho da Seção 07.

SEÇÃO 11

Lacunas conhecidas e próximos passos

Frontend (pdv-web) não foi alterado. Todo o trabalho desta sessão foi restrito ao backend, por decisão deliberada: o pedido de melhorias/documentação não pediu explicitamente a implementação da tela. O front ainda não tem nenhuma tela, rota ou terminologia de Distrito/Setor. Recomendação: telas de CRUD de Distrito e de Setor, campos `setor_id` nos formulários de loja/usuário, e filtro por distrito/setor nas listagens — reaproveitando os componentes já usados nas telas existentes.

- **Migração de dados legados:** lojas e consultores criados antes desta mudança têm `setor_id` nulo; hoje só é possível vincular registro a registro pelo PUT existente.
- **Exclusão de distrito/setor:** propositalmente fora do escopo do CRUD, consistente com o padrão do domínio de usar `status` em vez de exclusão física.
- **Migração de lojas ao transferir consultor:** pendência de negócio pré-existente no próprio código — não foi criada nem resolvida nesta sessão, apenas preservada.