4 - ApiService

Iagro CLP Local — ApiService

Cliente HTTP síncrono (`requests`) para o backend IAgro. Centraliza autenticação por headers, parsing defensivo de JSON e rotas de negócio usadas por monitors e CommandProcessor.

Código: `src/services/api_service.py`
Contrato: `docs/api/backend-openapi.yaml` · **Página 12** — mapa operationId

> **Info**
> `parcel_id` nos métodos PATCH é o **ID de `work_order_recipe_plc_sync`** , não o id da parcela da OS — legado de nomenclatura no desktop.

📋 Resumo

- Instância única por sessão; `update_headers` após refresh token (`AuthService` + `main._session_refresh_loop`).
- Base URL: construtor / env `API_BASE_URL` .
- Timeouts: **10s** (maioria GET/PATCH); **5s** (`get_active_view_variables`).

🎯 API pública — inventário completo

Método	HTTP	Query / path	Retorno	Consumidor
`get_branches`	GET `/branches`	—	`list[dict]`	UI login
`get_smart_caldas_by_branch`	GET `/smart-caldas`	`branchId` , paginação	dict com `items`	ContainerManager
`get_smart_calda_by_id`	GET `/smart-caldas/{id}`	`X-Branch-ID`	dict + `configurationJson`	CLPContainer, PM, CP reset

`get_active_view_variables`	GET `…/active-view-variables`	`x-supervisor-key`	`list[int]`	SequentialCoordinator
`get_work_order_recipe_sync`	GET `/work-order-recipe-plc-sync`	`smartCaldaId`, `status`, `limit`	`list`	RecipeMonitor
`get_recipe_sync_by_id`	GET `/work-order-recipe-plc-sync/{id}`	`X-Branch-ID`	`dict \\| None`	`sanitize_orphan_create_recipes`
`get_active_recipe`	GET (3×) mesmo path	`status` ∈ ativos	`dict \\| None`	RecipeProcessor, gates
`update_parcel_status`	PATCH `…/{id}`	body `{ status, … }`	`bool`	CP, PM
`update_parcel_process_sequence`	PATCH `…/process-sequence`	sequência	`bool`	PM
`update_headers`	—	Bearer, tenant	mutação in-place	Session refresh

📁 Headers e escopo multi-tenant

Header	Obrigatório	Rotas
`Authorization: Bearer {token}`	Sim (autenticado)	Todas exceto login
`X-Tenant-ID`	Sim	Todas autenticadas

`X-Branch-ID`	SmartCalda / receitas / PATCH sync	Quando `branch_id` passado no método
`x-supervisor-key`	active-view	Supervisório local

Falha de token → exceção propagada; monitors logam e pulam ciclo.

📁 Contratos de dados

Resposta genérica

`_extract_list` / helpers aceitam envelope: `data`, `items`, `results`, `branches`, ou lista na raiz — tolerância a versões de API.

`update_parcel_status(parcel_id, status, branch_id?, productsUsed?)`

Campo body	Uso
`status`	Enum ERP (string)
`productsUsed`	Opcional em cancelamento supervisório

Chamadas críticas:

Chamador	Status	Momento
CommandProcessor	`NA_MAQUINA`	Após writes iniciais todos `completed`
CommandProcessor	`BLOQUEADO`	`_fail_recipe_write`
ProductionMonitorService	`EM_PRODUCAO`, `PAUSADO`, `PROCESSADA`, `CANCELADO`, …	Gatilhos OPC mapeados

`get_active_recipe(smart_calda_id, branch_id?)`

Três requisições sequenciais com `status`:

1. `NA_MAQUINA`

2. `EM_PRODUCAO`
3. `PAUSADO`

Retorna **primeira** não vazia. Usado para impedir segunda receita enquanto uma está ativa na calda.

`get_recipe_sync_by_id(sync_id, branch_id?)`

Campos lidos pelo sanitizer: `status`, `parcelStatus` (fallback). Terminais: `CANCELADO`, `PROCESSADA`.

`get_work_order_recipe_sync`

Filtro típico: `status=NA_FILA`, `limit=50` (thread) ou `1` (coordenador). Ordenação **no cliente** por `createdAt` ASC.

🔄 Fluxos ponta a ponta

A — Captura de fila

```
RecipeMonitor (gates OK)
  → get_work_order_recipe_sync(NA_FILA)
  → RecipeProcessor.process_recipe
  → (sem HTTP até CP) update_parcel_status NA_MAQUINA
```

B — Bloqueio por receita ativa

```
RecipeProcessor.process_recipe
  → get_active_recipe
  → se recipe_sync.id != active.id: return False
```

C — Supervisório (backend-only no HTTP de confirmação)

```
Frontend → Backend confirm-action
  → Backend PATCH ERP terminal
  → Backend INSERT recipe-finalize (PostgreSQL CLP — não ApiService desk
Desktop sanitizer → get_recipe_sync_by_id (somente leitura)
```

D — Reset / config

```
CommandProcessor / PM
  → get_smart_calda_by_id → configurationJson
  → mapeamento tags reset e recipe_mapping
```

🎯 Integração por componente

Componente	Métodos
`main.py`	Cria ApiService após login
`ContainerManager`	Repassa para cada container

`RecipeMonitorService`	`get_work_order_recipe_sync`
`RecipeProcessor`	`get_active_recipe`
`CommandProcessor`	`update_parcel_status`, sanitizer `get_recipe_sync_by_id`
`ProductionMonitorService`	Status produção + `update_parcel_process_sequence`
`SequentialCoordinator`	`get_active_view_variables`

⚙️ Configuração e resiliência

Parâmetro	Valor
Timeout padrão	10 s
Timeout active-view	5 s
404 lista receitas	`[]` (não exceção)
404 sync by id	`None`
404 SmartCalda	exceção (reset impossível)
Erro rede	`Exception("Erro de conexão…")`

⚠️ Armadilhas de integração

Armadilha	Realidade no código
Achar que RP muda status ERP	Só CP após OPC
`parcel_id` = parcela OS	É sync id na PATCH
Desktop enfileira finalize	Backend PostgreSQL CLP
`get_active_recipe` uma chamada	Três GET por status

🔗 OpenAPI (operationId)

Python	operationId
`get_branches`	`getBranches`
`get_smart_caldas_by_branch`	`getSmartCaldasByBranch`
`get_smart_calda_by_id`	`getSmartCaldaById`
`get_active_view_variables`	`getActiveViewVariables`
`get_work_order_recipe_sync`	`getWorkOrderRecipeSync`
`get_active_recipe`	`getActiveRecipe`
`get_recipe_sync_by_id`	GET by id (mesmo recurso PATCH)
`update_parcel_status`	`updateParcelStatus`
`update_parcel_process_sequence`	`updateParcelProcessSequence`

🔗 Referências

- **3 - AuthService · 12 - API — contrato OpenAPI**
- **5–8 · 7** (NA_MAQUINA / BLOQUEADO / sanitizer)