11 - UI — IAgroCLPAppCTk

Iagro CLP Local — IAgroCLPAppCTk

Shell **CustomTkinter** do operador: autenticação, seleção de filial/SmartCalda, conexão OPC, monitoramento, variáveis ao vivo, logs de atividade e indicadores de progresso (receita / discover).

Código: `src/main.py` — classe `IAgroCLPAppCTk(ctk.CTk)`
Loading: `src/ui/log_loading.py` · **Containers:** `src/ui/container_manager.py`

📋 Resumo

Camada	Componente
Sessão	`SessionManager` + **3 - AuthService**
HTTP negócio	**4 - ApiService** via `ContainerManager`
Runtime CLP	**9 - CLPContainer** (N instâncias por calda da API)
Agendamento UI	`ui_scheduler` — marshalling thread-safe para `after()`

🔄 Bootstrap (`__init__`)

```
1. ContainerManager(self)
2. ui_scheduler = UiScheduler(root.after)
3. SupervisorKeyServer (porta local — x-supervisor-key para active-view)
4. SessionManager.load ou LoginWindow
5. _build_ui / _wire_events
6. after(1000): _update_indicators_loop
7. after(90000): _session_refresh_loop
```

Sem login válido: apenas fluxo de autenticação; containers não carregam da API.

🔄 Autenticação e sessão

```
LoginWindow
  → AuthService.login
  → SessionManager.save_session
  → ApiService(access_token, tenant_id)
  → ContainerManager.set_api_service (todos containers)
  → UI: BranchSelector
  → GET smart-caldas → cards sidebar (smart_calda_id obrigatório)
```

```
8
9  _session_refresh_loop (90s):
10   refresh_token → SessionManager.update
11   → ApiService.update_headers
12   → _sync_session_tokens_to_api_service()
```

Logout: limpa sessão, desconecta containers, volta ao login.

🎯 ContainerManager e sidebar

- Lista derivada **somente** do backend (`smart_calda_id` UUID).
- Busca/filtro local nos cards.
- Seleção: `_on_container_selected` → painel detalhe + variáveis.
- Eventos propagados:
  - `_on_container_log` — fila de mensagens por calda
  - `_on_container_loading` — spinner / barra receita
  - `_on_containers_updated` — refresh lista

Cada `CLPContainer` registra callbacks na construção/atualização da instância UI.

🔄 Conexão OPC (async na UI)

Passo	Método
Operador clica conectar	`toggle_connection`
Worker thread	`_start_opc_connect_async` → `container.connect()`
Volta UI	`_apply_opc_connect_result`
Busy state	`_opc_connect_set_busy_ui` (desabilita botões)

Erro OPC: mensagem no log do container + status `error` / `disconnected` no card.

🔄 Monitoramento

Ação	Efeito
`start_monitoring`	`SequentialCoordinator.start()` + `_sync_command_processor_thread`

`stop_monitoring`	Para coordenador; pode religar thread CP
`interval_spinbox`	`container.interval_ms` → leitura OPC
`on_row_monitor_clicked`	Usa `container.is_monitoring()` — não assume thread CP ativa

Indicador "monitor ON" na sidebar reflete `is_monitoring()`, não apenas `is_connected()`.

🔄 Loading e logs (`log_loading`)

Origem: `CLPContainer.emit_loading` ← `CommandProcessor._emit_container_loading`.

Evento	`kind` típico	`persist`
Início escrita receita	`recipe_write`	True
Progresso N/M	`recipe_write` + current/total	True
Discover	`discover`	variável
Fim	`stop` / mensagem final	—

Logs traduzidos via `src/utils/i18n.py` (`tr("log.loading_recipe", …)`).

🔄 Fluxo operador vs supervisório

No desktop Python:

1. Login → branch → selecionar calda
2. Conectar CLP
3. Ligar monitor
4. Observar captura `NA_FILA` → loading receita → status ERP via API (indireto)

Cancel / complete de receita:

- UI web supervisório → backend `confirm-action`

- Desktop **não** implementa o modal; efeito visível via logs CP (`recipe-finalize`) e eventual liberação de nova `NA_FILA`

⚙️ Configuração UI

Variável	Uso
`API_BASE_URL`	Host backend
`UI_VARS_REFRESH_MIN_INTERVAL_S`	Debounce tabela variáveis
Tema CTk	`ctk.set_appearance_mode` em `main`

⚠️ Threading

Origem thread	Destino UI
OPC read worker	`ui_scheduler.schedule` → `after`
CommandProcessor	logs + loading via scheduler
Connect async	`_apply_opc_connect_result` no main thread

Nunca atualizar widgets CTk diretamente de threads de serviço.

🔗 Referências

- **9 - CLPContainer** · **10 - SequentialCoordinator**
- **3 - AuthService** · **4 - ApiService** · **7 - CommandProcessor** (loading)