Service Tag Manager

Documentação Técnica Completa

Gerado em: 16/08/2025 14:10

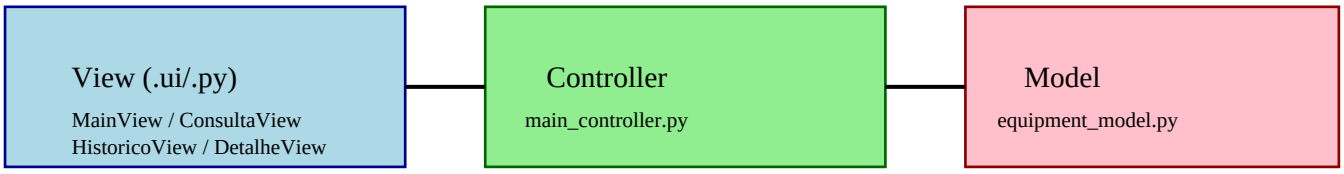
Resumo

O Service Tag Manager é um sistema de gestão de equipamentos e serviços técnicos desenvolvido em Python com PyQt5 e SQLite. Adota o padrão MVC (Model-View-Controller), utiliza um QStackedWidget como janela principal e possui telas para Cadastro, Consulta, Histórico e Edição/Detalhes. O banco de dados usa uma tabela principal 'equipamentos' com índice único na Service Tag.

Sumário

1. Arquitetura Geral

O sistema segue o padrão MVC. As Views são definidas via arquivos .ui (Qt Designer) em 'view/'. O Controller principal ('controller/main_controller.py') conecta sinais das Views aos métodos de negócio e navega entre páginas do QStackedWidget. O Model ('model/equipment_model.py') encapsula todo o acesso ao SQLite, incluindo criação/migração de tabela, inserção, busca, atualização e exclusão.



Comunicação: A View emite sinais (ex.: clique de botão, texto alterado). O Controller recebe esses sinais, executa validações e chama métodos do Model. O Model acessa o banco SQLite e retorna dados. O Controller, então, atualiza a View (tabelas, mensagens, navegação).

2. Módulos e Arquivos

Arquivo	Função Principal
main.py	Inicializa QApplication, cria QStackedWidget, instância as Views e Controller; abre o app em
controller/main_controller.py	Conecta sinais das views; implementa adicionar, consultar, histórico, editar, excluir; navegaçã
model/equipment_model.py	Cria/atualiza tabela, índices; implementa CRUD: adicionar, buscar_simples, buscar_avancad
view/ui_main.ui	Tela principal de cadastro (Service Tag, Nome, Cliente, Modelo, etc.).
view/ui_consulta.ui	Tela de consulta com busca por texto e filtros; tabela com resultados.
view/ui_historico.ui	Tela de histórico com tabela geral.
view/ui_detalhe.ui	Diálogo de Detalhes/Edição de um equipamento.

3. Fluxo de Execução (Runtime)

main.py

- Cria QApplication e QStackedWidget.
- Instancia: MainView, ConsultaView, HistoricoView.
- Adiciona as views ao QStackedWidget (índices 0, 1, 2).
- Instancia EquipmentModel (SQLite).
- Instancia MainController (passa as views, o stacked e o model).
- Define título e abre **em tela cheia** com *showFullScreen()*.
- Seta a primeira tela: índice 0 (Cadastro/Main).

Troca de Telas

A navegação é feita com *stacked_widget.setCurrentIndex(N)*. Não se chama *show()* novamente, mantendo sempre **full screen**.

4. Mapa de Sinais/Slots (UI → Controller → Model)

rigem (View)	Sinal	Controller (slot)	Model (quando aplicável)	Efeito
ainView.btn_add	clicked	MainController.adicionar_equipamento	EquipmentModel.adicionar_equipamento	Valida campos, insere no BD
ainView.btn_consulta	clicked	MainController.mostrar_consulta	-	Vai para tela Consulta (index
ainView.btn_historico	clicked	MainController.mostrar_historico	EquipmentModel.buscar_avancado	Carrega a tabela do Histórico
onsultaView.input_busca	textChanged/returnPressed	MainController._consulta_buscar	EquipmentModel.buscar_simples/avancado	Atualiza a tabela de resultado
onsultaView.filtro_status/tipo/prioridade	currentIndexChanged	MainController._consulta_buscar	EquipmentModel.buscar_avancado	Filtra resultados conforme se
onsultaView.btn_editar	clicked	MainController._abrir_detalhe_selecionado	EquipmentModel.obter_por_id/atualizar_basico	Abre detalhe/edição; salva al
onsultaView.btn_excluir	clicked	MainController._excluir_selecionado	EquipmentModel.excluir	Remove o registro seleciona
onsultaView.btn_voltar_consulta	clicked	MainController.voltar	-	Retorna ao Main (index 0).
istoricoView.btn_editar_h	clicked	MainController._abrir_detalhe_selecionado_hist	EquipmentModel.obter_por_id/atualizar_basico	Edição a partir do histórico.
istoricoView.btn_excluir_h	clicked	MainController._excluir_selecionado_hist	EquipmentModel.excluir	Exclusão a partir do histórico
istoricoView.btn_voltar_historico	clicked	MainController.voltar	-	Retorna ao Main (index 0).

5. Banco de Dados (Modelo Lógico + Dicionário de Dados)

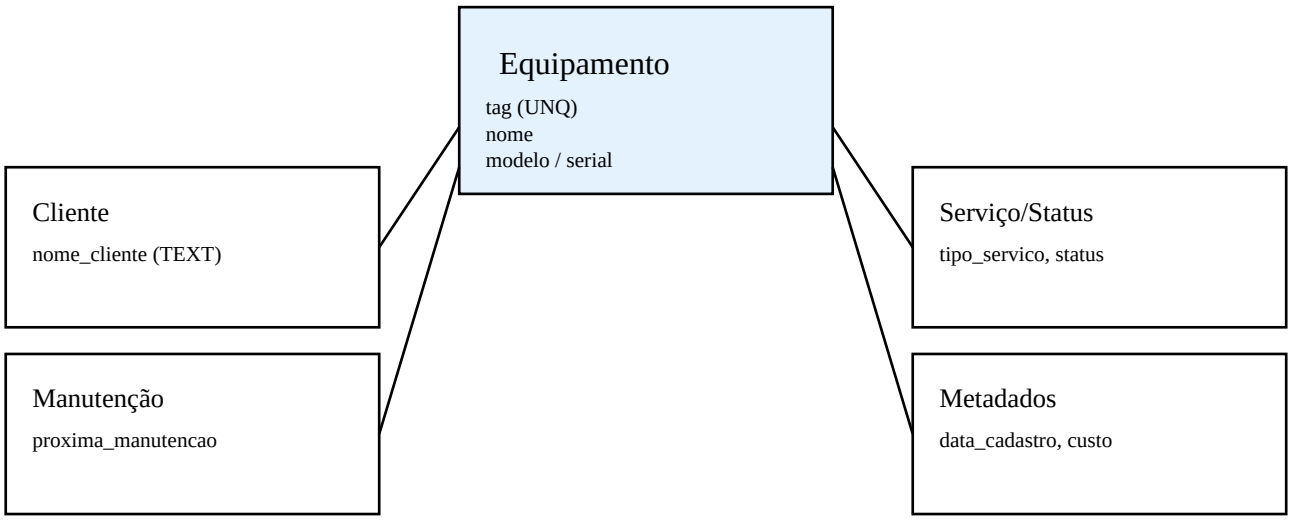
O banco usa SQLite e possui uma tabela principal **equipamentos**. A coluna *tag* tem índice único para evitar duplicidade. A *data_cadastro* é preenchida automaticamente (no código do model) ao inserir um novo registro.

Campo	Tipo	Obrigatório	Descrição
id	INTEGER (PK)	Sim	Identificador interno do registro.
tag	TEXT (UNIQUE)	Sim	Service Tag do equipamento (não pode repetir).
nome	TEXT	Sim	Nome/identificação do equipamento.
descricao	TEXT	Não	Descrição do defeito ou serviço.
data_cadastro	TEXT (ISO)	Sim (auto)	Data e hora de cadastro (datetime('now')).
cliente	TEXT	Sim	Nome do cliente responsável.
modelo	TEXT	Não	Modelo do equipamento.
serial	TEXT	Não	Número de série.
tipo_servico	TEXT	Não	Diagnóstico, Preventiva, Corretiva, etc.
status	TEXT	Não	Recebido, Em análise, Aguardando peças, Em reparo, Pronto, Entregue.
prioridade	TEXT	Não	Baixa, Média, Alta, Urgente.
proxima_manutencao	TEXT (ISO)	Não	Data prevista para próxima manutenção.
custo	REAL	Não	Custo/valor do serviço.
garantia_meses	INTEGER	Não	Garantia aplicada ao serviço, em meses.
observacoes	TEXT	Não	Observações internas.

Nome do Índice	Coluna	Tipo
idx_equip_tag	tag	UNIQUE
idx_equip_nome	nome	INDEX
idx_equip_cliente	cliente	INDEX
idx_equip_status	status	INDEX

6. Modelo Conceitual (ER)

O modelo atual é centrado em uma entidade principal **Equipamento**. Campos como Cliente e Técnico são armazenados como texto (sem tabelas separadas). Abaixo, uma visão conceitual simplificada alinhada ao que existe hoje.



Observação: Um modelo normalizado futuro poderia separar entidades como *Cliente*, *Técnico* e *Serviço* em tabelas próprias, criando relacionamentos 1:N e N:N. O sistema atual, entretanto, é intencionalmente simples e armazena essas informações na mesma tabela.

7. Principais Fluxos

7.1 Fluxo de Cadastro

1) Usuário preenche o formulário na MainView e clica em **Adicionar Equipamento**.
2) Sinal *clicked* chama **MainController.adicionar_equipamento()**.
3) Controller valida campos (tag, nome, cliente) e chama **EquipmentModel.adicionar_equipamento()**.
4) Model executa INSERT no SQLite e seta *data_cadastro* automaticamente (datetime('now')).
5) Controller mostra QMessageBox de sucesso/erro e limpa os campos.

7.2 Fluxo de Consulta

1) Usuário digita no campo de busca e/ou escolhe filtros (status, tipo, prioridade).
2) Sinais *textChanged* e *currentIndexChanged* chamam **MainController._consulta_buscar()**.
3) O Controller decide entre *buscar_simples* (texto) e *buscar_avancado* (filtros).
4) Recebe os registros do Model e preenche a QTableWidget, atualizando o contador.
5) Botões *Editar/Excluir* operam sobre a linha selecionada.

7.3 Fluxo de Histórico

1) Ao abrir a tela, o Controller chama **EquipmentModel.buscar_avancado()** com termo vazio.
2) Popula a tabela com os resultados e exibe o total.
3) Ações de *Editar/Excluir* funcionam similar à Consulta.

7.4 Fluxo de Edição/Detalhes

1) A partir de Consulta ou Histórico, o usuário seleciona uma linha e clica em **Editar** (ou duplo clique).
2) Controller chama **EquipmentModel.obter_por_id()** e abre o diálogo **DetalheView** carregando os dados.
3) Ao salvar, chama **EquipmentModel.atualizar_basico()** para status, prioridade, próxima manutenção e observações.
4) Atualiza a tela atual após o sucesso.

8. Boas Práticas e Extensões Futuras

- Manter validações de campos obrigatórios no Controller e restrições (UNIQUE) no Banco.
- Não chamar show() nas trocas de tela; use apenas setCurrentIndex no QStackedWidget.
- Centralizar acesso ao DB no Model para facilitar manutenção/testes.
- Usar índices em colunas consultadas com frequência (tag, nome, cliente, status).
- Futuro: separar Cliente/Técnico em tabelas próprias; adicionar autenticação e perfis; relatórios PDF.

9. Perguntas Frequentes (FAQ)

Pergunta	Resposta
Como o sistema evita Service Tags duplicadas?	Há um índice UNIQUE na coluna 'tag' e o Controller trata o erro, exibindo mensagem amigável.
Onde a data de cadastro é definida?	No Model, no INSERT, usando datetime('now') (fuso local ou padrão conforme a configuração).
Como a consulta é feita ao digitar?	Sinais textChanged/returnPressed chamam MainController._consulta_buscar(), que aciona o Mode
Como o app fica sempre em tela cheia?	No main.py, a janela principal (QStackedWidget) é aberta com showFullScreen().
Como excluir um registro?	Selecione na tabela e use o botão Excluir; o Controller confirma e chama EquipmentModel.excluir()