# Manual Técnico do Sistema de Almoxarifado

## Documentação Completa com Análise Linha por Linha

**Desenvolvedor:** Lourenço

**Instituição:** Fundação Liberato

**Destinado a:** Alunos e Funcionários

**Versão:** Final

---

# Índice



---

# 1. Introdução

Este manual técnico fornece uma visão detalhada do Sistema de Almoxarifado, uma aplicação de desktop desenvolvida em Python com a biblioteca PyQt6 para a interface gráfica e MySQL como banco de dados. O sistema foi projetado para gerenciar o empréstimo e a devolução de componentes eletrônicos aos alunos da Fundação Liberato, oferecendo uma interface intuitiva para pesquisa de alunos, consulta de componentes e registro de movimentações.

## 1.1. Componentes do Sistema

O sistema é composto por:

- **Aplicação Principal (`SistemaALMOX.txt`):** Interface gráfica, lógica de negócio e Painel Administrativo.
- **Módulo de Configuração (`Config.txt`):** Gerenciamento de conexão com banco de dados.
- **Módulo de Login (`Senha.txt`):** Janela de autenticação para acesso administrativo.
- **Script de Processamento de Alunos (`Atualizaralunos.txt`):** Processamento de CSV e atualização de dados de alunos no banco.
- **Script de Atualização de Fotos (`atualizar_fotos.py`):** Importação automática de fotos (mantido da versão anterior).

---

# 2. Arquitetura do Sistema

A arquitetura do sistema é modular e clara.

**Interface do Usuário (Frontend):** PyQt6.

**Lógica de Negócio (Backend):** Python.

**Banco de Dados (Persistência):** MySQL.

---

# 3. Estrutura do Banco de Dados

A estrutura das tabelas (`alunos`, `componentes`, `emprestimos`) permanece a mesma, mas as credenciais de acesso ao banco foram centralizadas e simplificadas.

---

# 4. Guia de Instalação e Configuração

## 4.1. Instalação das Ferramentas

(Mantido o mesmo: Python 3.11, VS Code, DBeaver)

## 4.2. Configuração do Ambiente

**Passo 1: Instalar Dependências Python**

(Mantido o mesmo: `pip install mysql-connector-python PyQt6 pytz pandas requests`)

**Passo 2: Configurar a Conexão no DBeaver**

- Abra o DBeaver

- Clique em "Nova Conexão" e selecione "MySQL"
- Insira as novas credenciais conforme o arquivo `Config.txt`:
  - Host: `10.233.43.213`
  - Porta: `3308`
  - Usuário: `troadmin`
  - Senha: `eleTROn1c_flop3#W!w`
  - Banco de dados: `almox_tro`
- Teste a conexão antes de salvar

**Passo 3: Criar e Configurar as Tabelas**

(Mantido o mesmo SQL, garantindo que `id` seja **PRIMARY KEY** e **AUTO_INCREMENT**).

### 4.3. Executar o Sistema

(Mantido o mesmo: Abrir no VS Code e executar o arquivo principal, agora `SistemaALMOX.txt`).

---

## 5. Análise Detalhada do Código-Fonte

### 5.1. Arquivo `Config.txt` (Configuração do Banco) - Análise Linha por Linha

O arquivo `Config.txt` é responsável por gerenciar a conexão com o banco de dados MySQL.

Python

```python
import mysql.connector

class ConfigBanco:
    DB_NAME = "almox_tro"
    DB_USER = "troadmin"
    DB_PASSWORD = "eleTROn1c_flop3#W!w"
    DB_HOST = "10.233.43.213"
    DB_PORT = 3308

    @staticmethod
    def obter_conexao():
        try:
            conexao = mysql.connector.connect(
                host=ConfigBanco.DB_HOST,
                user=ConfigBanco.DB_USER,
```

```
16                  password=ConfigBanco.DB_PASSWORD,
17                  database=ConfigBanco.DB_NAME,
18                  port=ConfigBanco.DB_PORT
19              )
20              return conexao
21          except mysql.connector.Error as erro:
22              print(f"Erro ao conectar ao banco de dados: {erro}")
23              return None
24
25      @staticmethod
26      def banco_atual():
27          return f"{ConfigBanco.DB_USER}@{ConfigBanco.DB_HOST}"
```

| Linha | Código | Descrição |
|---|---|---|
| 1-2 | `import mysql.connector` | Importa o módulo para conexão com MySQL. |
| 3 | `class ConfigBanco:` | Define a classe de configuração. |
| 4-8 | `DB_NAME = "almox_tro"`, `DB_USER = "troadmin"`, `DB_PASSWORD = "eleTROn1c_flop3#W!w"`, `DB_HOST = "10.233.43.213"`, `DB_PORT = 3308` | Define as constantes de conexão (nome do banco, usuário, senha, host e porta). |
| 10-11 | `@staticmethod`, `def obter_conexao():` | Método estático para obter a conexão. |
| 12-20 | `try: ... return conexao` | Tenta estabelecer a conexão usando as constantes definidas na classe. |
| 21-23 | `except mysql.connector.Error as erro: ... return None` | Captura erros de conexão e retorna `None`. |
| 25-27 | `@staticmethod`, `def banco_atual(): ... return f"{ConfigBanco.DB_USER}@{ConfigBanco.DB_HOST}"` | Método que retorna uma string formatada com o usuário e host da conexão, útil para exibição na interface. |

## 5.2. Arquivo `Senha.txt` (Login Administrativo) - Análise Linha por

## Linha

Este módulo implementa uma janela de login para proteger o acesso às funções administrativas.

Python

```
1   from PyQt6.QtWidgets import (
2       QWidget, QVBoxLayout, QLineEdit, QPushButton, QLabel, QMessageBox,
QHBoxLayout, QFrame
3   )
4   from PyQt6.QtCore import Qt
5   from PyQt6.QtGui import QFont
6
7   class LoginAdmin(QWidget):
8       def __init__(self, parent=None):
9           super().__init__(parent)
10          self.setWindowTitle("Login Administrativo")
11          self.resize(500, 300)
12          self.setStyleSheet("background-color: #ecf0f1;")
13
14          # --- Container central com borda arredondada ---
15          container = QFrame()
16          container.setFixedSize(350, 220)
17          container.setStyleSheet("""
18              QFrame {
19                  background-color: white;
20                  border: 2px solid #bdc3c7;
21                  border-radius: 15px;
22              }
23          """)
24
25          layout_container = QVBoxLayout()
26          layout_container.setAlignment(Qt.AlignmentFlag.AlignCenter)
27
28          # --- Título ---
29          titulo = QLabel("🔐 Área Administrativa")
30          titulo.setFont(QFont("Arial", 16, QFont.Weight.Bold))
31          titulo.setAlignment(Qt.AlignmentFlag.AlignCenter)
32          titulo.setStyleSheet("color: #2c3e50; margin-bottom: 10px;")
33          layout_container.addWidget(titulo)
34
35          # --- Campo usuário ---
36          self.usuario_input = QLineEdit()
37          self.usuario_input.setPlaceholderText("Usuário")
38          self.usuario_input.setFixedWidth(250)
39          self.usuario_input.setStyleSheet("padding: 6px; border-radius:
5px; border: 1px solid #ccc;")
```

```
40          layout_container.addWidget(self.usuario_input,
alignment=Qt.AlignmentFlag.AlignCenter)
41
42          # --- Campo senha ---
43          self.senha_input = QLineEdit()
44          self.senha_input.setPlaceholderText("Senha")
45          self.senha_input.setEchoMode(QLineEdit.EchoMode.Password)
46          self.senha_input.setFixedWidth(250)
47          self.senha_input.setStyleSheet("padding: 6px; border-radius:
5px; border: 1px solid #ccc;")
48          layout_container.addWidget(self.senha_input,
alignment=Qt.AlignmentFlag.AlignCenter)
49
50          # --- Botões ---
51          botoes_layout = QHBoxLayout()
52          botoes_layout.setAlignment(Qt.AlignmentFlag.AlignCenter)
53
54          botao_login = QPushButton("Entrar")
55          botao_login.setStyleSheet("padding: 8px; background-color:
#3498db; color: white; border-radius: 5px;")
56          botao_login.clicked.connect(self.verificar_login)
57          botoes_layout.addWidget(botao_login)
58
59          botao_cancelar = QPushButton("Cancelar")
60          botao_cancelar.setStyleSheet("padding: 8px; background-color:
#e74c3c; color: white; border-radius: 5px;")
61          botao_cancelar.clicked.connect(self.close)
62          botoes_layout.addWidget(botao_cancelar)
63
64          layout_container.addLayout(botoes_layout)
65          container.setLayout(layout_container)
66
67          # --- Layout principal centralizado ---
68          layout_principal = QVBoxLayout()
69          layout_principal.setAlignment(Qt.AlignmentFlag.AlignCenter)
70          layout_principal.addWidget(container)
71          self.setLayout(layout_principal)
72
73      def verificar_login(self):
74          usuario = self.usuario_input.text()
75          senha = self.senha_input.text()
76
77          if usuario == "admin" and senha == "cachorro":
78              self.accept_login()
79          else:
80              QMessageBox.warning(self, "Erro", "Usuário ou senha
incorretos!")
81
```

```
82      def accept_login(self):
83          self.close()
84          if self.parent():
85              self.parent().abrir_administrativo_real()
```

| Linha | Código | Descrição |
| --- | --- | --- |
| 1-6 | `from PyQt6...` | Importações de widgets e módulos necessários para a interface gráfica. |
| 7 | `class LoginAdmin(QWidget):` | Define a classe da janela de login, que herda de `QWidget`. |
| 8-12 | `__init__`, `setWindowTitle`, `resize`, `setStyleSheet` | Configurações iniciais da janela (título, tamanho e cor de fundo). |
| 14-23 | `container = QFrame() ...` | Cria um `QFrame` centralizado com estilo (borda arredondada) para conter os campos de login. |
| 29-33 | `titulo = QLabel(...)` | Cria o título "🔐 Área Administrativa" com fonte em negrito e centralizado. |
| 36-40 | `self.usuario_input = QLineEdit()` | Cria o campo de entrada para o nome de usuário. |
| 43-48 | `self.senha_input = QLineEdit()` | Cria o campo de entrada para a senha. `setEchoMode(QLineEdit.EchoMode.Password)` oculta a senha digitada. |
| 54-57 | `botao_login = QPushButton("Entrar")` | Cria o botão "Entrar", que chama o método `verificar_login` ao ser clicado. |
| 59-62 | `botao_cancelar = QPushButton("Cancelar")` | Cria o botão "Cancelar", que fecha a janela. |
| 73 | `def verificar_login(self):` | Método que verifica as credenciais. |
| 77 | `if usuario == "admin" and senha == "cachorro":` | **Ponto Crítico:** Verifica se o usuário é "admin" e a senha é |

| | | |
|---|---|---|
| | | **"cachorro"**. |
| 78 | `self.accept_login()` | Se as credenciais estiverem corretas, chama o método de aceitação. |
| 80 | `QMessageBox.warning(...)` | Se as credenciais estiverem incorretas, exibe uma mensagem de aviso. |
| 82 | `def accept_login(self):` | Método chamado após o login bem-sucedido. |
| 83 | `self.close()` | Fecha a janela de login. |
| 85 | `self.parent().abrir_administrativo_real()` | Chama o método `abrir_administrativo_real` na janela principal ( `AppAlmoxarifado` ) para abrir o painel administrativo. |

## 5.3. Arquivo `SistemaALMOX.txt` (Aplicação Principal) - Análise Linha por Linha

O novo arquivo principal ( `SistemaALMOX.txt` ) contém as maiores mudanças, incluindo a integração do login, o novo painel administrativo e a funcionalidade de devolução parcial.

### 5.3.1. Importações e Classe `ItemEmprestimo` (Mudanças)

Python

```
1   import sys
2   import os
3   import mysql.connector
4   import urllib.request
5   from mysql.connector import Error
...
16  from Config import ConfigBanco
17  from senha import LoginAdmin
18  from processar_alunos import processar_alunos
19  from emprestimos import importar_emprestimos
...
42          self.checkbox = QCheckBox()
43          self.checkbox.setEnabled(self.status == "Emprestado")
44          layout.addWidget(self.checkbox)
45          self.checkbox.setStyleSheet("""
```

```
...
57          """)
58
59
60          self.campo_devolucao = QLineEdit()
61          self.campo_devolucao.setPlaceholderText("Qtd")
62          self.campo_devolucao.setFixedWidth(50)
63          self.campo_devolucao.setEnabled(self.status == "Emprestado")
64          self.campo_devolucao.setStyleSheet("background-color: #FFFFFF;
color: #333333;")
65          layout.addWidget(self.campo_devolucao)
```

| Linha | Código | Descrição |
|---|---|---|
| 2 | `import os` | Importa o módulo `os` para operações de sistema (usado para salvar arquivos no Desktop). |
| 4 | `import urllib.request` | Importa para baixar fotos da intranet. |
| 17 | `from senha import LoginAdmin` | Importa a classe de login administrativo. |
| 18 | `from processar_alunos import processar_alunos` | Importa a função do script auxiliar de processamento de alunos. |
| 19 | `from emprestimos import importar_emprestimos` | Assume-se que esta função importa dados de empréstimos (ex: de um CSV). |
| 45-57 | `self.checkbox.setStyleSheet(...)` | Estilo do checkbox foi alterado para usar uma imagem de marca de seleção ( `check-mark.png` ). |
| 60-65 | `self.campo_devolucao = QLineEdit()` ... | Adiciona um campo de texto ( `QLineEdit` ) para que o usuário possa digitar a **quantidade a ser devolvida** (devolução parcial). |

## 5.3.2. Classe `AppAlmoxarifado` - Painel Administrativo (Novidade)

Python

```
155         # Botão Administrativo na barra de ferramentas
156         self.botao_admin = QPushButton("Abrir Administrativo")
157         self.botao_admin.clicked.connect(self.abrir_administrativo)
158         barra_ferramentas.addWidget(self.botao_admin)
...
164     def abrir_administrativo(self):
165         self.login_dialog = LoginAdmin(self)
166         self.login_dialog.show()
...
168     def abrir_administrativo_real(self):
169         self.janela_admin = QWidget()
...
191         botao_ranking = QPushButton("📊 Gerar Ranking")
192         botao_ranking.setStyleSheet(...)
193         botao_ranking.clicked.connect(self.mostrar_ranking)
...
196         botao_emails = QPushButton("📧 Gerar Emails Emprestados")
197         botao_emails.setStyleSheet(...)
198         botao_emails.clicked.connect(self.gerar_emails_pendentes)
...
203         botao_processar = QPushButton("👨‍🎓 Processar Alunos")
204         botao_processar.setStyleSheet(...)
205         botao_processar.clicked.connect(self.executar_processar_alunos)
...
208         botao_importar = QPushButton("📥 Importar Empréstimos")
209         botao_importar.setStyleSheet(...)
210
botao_importar.clicked.connect(self.executar_importar_emprestimos)
```

| Linha | Código | Descrição |
| --- | --- | --- |
| 156-158 | `self.botao_admin = QPushButton(...)` | Adiciona o botão "Abrir Administrativo" na barra de ferramentas. |
| 164-166 | `def abrir_administrativo(...)` | Abre a janela de login ( `LoginAdmin` ) antes de acessar o painel. |
| 168-222 | `def abrir_administrativo_real(...)` | Cria o Painel Administrativo, que só é acessado após o login. |
| 191-212 | `botao_ranking` , `botao_emails` , `botao_processar` , | Botões para as funções administrativas: Gerar Ranking, Gerar E-mails |

| | | |
|---|---|---|
| | botao_importar | Pendentes, Processar Alunos (via script auxiliar) e Importar Empréstimos. |

### 5.3.3. Métodos Administrativos (Novidade)

Python

```
1235        def mostrar_ranking(self):
1247                    WHERE e.data_emp >= DATE_SUB(CURDATE(), INTERVAL 1
YEAR)
...
1264                desktop = os.path.join(os.path.expanduser("~"),
"Desktop")
1265                caminho_arquivo = os.path.join(desktop,
"ranking_materiais.txt")
...
1296        def gerar_emails_pendentes(self):
1309                      AND e.data_emp >= DATE_SUB(CURDATE(), INTERVAL 1
YEAR)
...
1325                caminho_arquivo = os.path.join(desktop,
"emails_emprestados.txt")
...
1274        def executar_processar_alunos(self):
1276                sucesso = processar_alunos()  # função importada
...
1284        def executar_importar_emprestimos(self):
1288                importar_emprestimos(caminho_csv)
```

| Método | Descrição | |---|---|---| | `mostrar_ranking()` | Consulta o banco para listar os componentes mais emprestados no **último ano** e salva o resultado em um arquivo `ranking_materiais.txt` na Área de Trabalho. | | `gerar_emails_pendentes()` | Consulta o banco para listar os e-mails dos alunos com empréstimos ativos no **último ano** e salva em `emails_emprestados.txt` na Área de Trabalho. | | `executar_processar_alunos()` | Chama a função `processar_alunos()` (do script auxiliar) para atualizar o cadastro de alunos. | | `executar_importar_emprestimos()` | Chama a função `importar_emprestimos()` para importar dados de empréstimos (assumidamente do arquivo `emficha.csv`). |},{find:

### 5.3.4. Devolução Parcial (Mudança Crítica)

Python

```
1112                for item in itens_selecionados:
1113                    texto_qtd = item.campo_devolucao.text().strip()
1114
1115                    if texto_qtd == "":
1116                        qtd_devolver = item.quantidade
1117                    else:
1118                        try:
1119                            qtd_devolver = int(texto_qtd)
1120                        except ValueError:
1121                            QMessageBox.warning(self, "Valor inválido",
f"Quantidade inválida para {item.componente}")
1122                            continue  # ✅ AGORA ESTÁ DENTRO DO LOOP
1123
1124                        if qtd_devolver <= 0 or qtd_devolver >
item.quantidade:
1125                            QMessageBox.warning(self, "Quantidade
inválida", f"Quantidade fora do limite para {item.componente}")
1126                            continue
1127
1128
1129                    restante = item.quantidade - qtd_devolver
1130
1131                    if restante > 0:
1132                        # Atualiza o registro original com o restante
1133                        cursor.execute(
1134                            "UPDATE emprestimos SET quantidade = %s
WHERE id = %s",
1135                            (restante, item.id_emprestimo)
1136                        )
1137
1138                        # Cria novo registro apenas da parte devolvida
1139                        cursor.execute(
1140                            """INSERT INTO emprestimos (aluno_id,
componente_id, quantidade, data_emp, data_dev, status)
1141                               SELECT a.id, c.id, %s, %s, %s, 'Devolvido'
1142                               FROM alunos a
1143                               JOIN componentes c ON c.nome = %s
1144                               WHERE a.matricula = %s
1145                            """,
1146                            (qtd_devolver, data_emp_formatada, agora,
item.componente, self.matricula_selecionada)
1147                        )
1148                    else:
1149                        # Se devolveu tudo, só marca como devolvido
1150                        cursor.execute(
1151                            "UPDATE emprestimos SET data_dev = %s,
status = 'Devolvido' WHERE id = %s",
```

```
1152                            (agora, item.id_emprestimo)
1153                        )
```

| Linha | Descrição |
| --- | --- |
| 1113-1129 | O sistema verifica o campo `campo_devolucao` (o `QLineEdit` na interface). Se estiver vazio, assume que a quantidade total (`item.quantidade`) será devolvida. Se preenchido, valida se o valor é um número válido e se não excede a quantidade emprestada. |
| 1131-1147 | Se a devolução for parcial (`restante > 0`): o registro original é **atualizado** com a quantidade restante, e um **novo registro** é criado na tabela `emprestimos` para a parte devolvida, com o status 'Devolvido'. |
| 1148-1153 | Se a devolução for total (`restante <= 0`), o registro original é apenas atualizado para 'Devolvido', como na versão anterior. |

## 6. Scripts Auxiliares

### 6.1. Arquivo `Atualizaralunos.txt` (Processamento e Atualização de Alunos) - Análise Linha por Linha

Este script é responsável por processar o arquivo CSV de alunos e se conectar ao banco de dados para realizar a atualização dos alunos (INSERT/UPDATE). |},{find:

Python

```
1   import pandas as pd
2   import os
3   import glob
4   import mysql.connector
5   from mysql.connector import Error
...
64          # 🔗 Conexão com banco MySQL
65          try:
66              conexao = mysql.connector.connect(
67                  host='10.233.43.213',
68                  user='troadmin',
```

```
69                  password='eleTROn1c_flop3#W!w',
70                  database='almox_tro',
71                  port=3308
72              )
...
76              for _, aluno in df_alunos.iterrows():
...
84                      # Verifica se matrícula já existe
85                      cursor.execute("SELECT id FROM alunos WHERE
matricula = %s", (matricula,))
86                      resultado = cursor.fetchone()
87
88                      if resultado:
89                          # Atualiza registro existente
90                          cursor.execute("""
91                              UPDATE alunos
92                              SET nome = %s, turma = %s, email = %s
93                              WHERE matricula = %s
94                          """, (nome, turma, email, matricula))
95                      else:
96                          # Insere novo registro
97                          cursor.execute("""
98                              INSERT INTO alunos (matricula, nome, turma,
email)
99                              VALUES (%s, %s, %s, %s)
100                         """, (matricula, nome, turma, email))
...
107             conexao.commit()
```

| Linha | Descrição |
| --- | --- |
| 4-5 | Importa `mysql.connector` e `Error` para operações de banco de dados. |
| 64-72 | Estabelece conexão direta com o banco de dados usando as credenciais do `Config.txt`. |
| 76-100 | Itera sobre o DataFrame processado e executa a lógica de **UPSERT** (UPDATE ou INSERT): |
| 84-86 | Verifica se a matrícula já existe no banco. |
| 88-94 | Se existe, executa um `UPDATE` para atualizar `nome`, `turma` e `email`. |
| 95-100 | Se não existe, executa um `INSERT` para adicionar o novo aluno. |

| | |
|---|---|
| 107 | `conexao.commit()` : Confirma as alterações no banco. |

### 6.2. Arquivo `atualizar_fotos.py` (Importação de Fotos)

(Manter a documentação anterior, mas garantir que as credenciais de conexão sejam as novas: Host `10.233.43.213` , Porta `3308` , Banco `almox_tro` ).

---

## 7. Conclusão

(Atualizar a conclusão para refletir as novas funcionalidades de segurança e manutenção).

---

## 8. Apêndices

(Atualizar o Apêndice B com o novo SQL e o Apêndice C com os novos requisitos de biblioteca).

---

**Fim do Manual Técnico**