

Arquivo: main.py

```
from PyQt6.QtWidgets import *
from PyQt6.QtGui import *
from PyQt6.QtCore import *
from paths import *
from modules.utils.icon_loader import load_icons
from assets.styles.styles import get_menu_button_style, get_menu_button_activated_style, get_full_dark_theme
from modules.widgets import *
from modules.config.config_widget import ConfigManager

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        print("Projeto Hasta360 - Versão 2.7.2")
        self.icons = load_icons()
        self.buttons = {}
        self.active_button = None
        self.inicio_widget = None
        self.setup_ui()
        self.open_initial_page()

    # ====== SETUP DA INTERFACE ======
    def setup_ui(self):
        """Configura a interface principal da aplicação."""
        self.configure_window()
        self.setup_central_widget()
        self.setup_menu()
        self.setup_content_area()

    def configure_window(self):
        """Configurações básicas da janela principal."""
        self.setWindowTitle("Lição 360")
        self.setWindowIcon(self.icons["brasil"])

        # Posiciona a janela no canto superior esquerdo
        screen_geometry = self.screen().geometry()
        self.move(screen_geometry.left(), screen_geometry.top())

    def setup_central_widget(self):
        """Define o widget central e layout principal."""
        self.central_widget = QWidget(self)
        self.setCentralWidget(self.central_widget)
        self.central_layout = QVBoxLayout(self.central_widget)
        self.central_layout.setSpacing(0)
        self.central_layout.setContentsMargins(0, 0, 0, 0)

    def setup_menu(self):
        """Configura o menu lateral com botões de ícone que mudam de cor ao hover e adiciona tooltips personalizados."""
        pass
```

```

self.menu_layout = QVBoxLayout()
self.menu_layout.setSpacing(0)
self.menu_layout.setContentsMargins(0, 0, 0, 0)
self.menu_layout.setAlignment(Qt.AlignmentFlag.AlignTop)

# Tooltip personalizado
self.tooltip_label = QLabel("", self)
self.tooltip_label.setStyleSheet("""
    background-color: #13141F;
    color: white;
    border: 1px solid #8AB4F7;
    padding: 4px;
    border-radius: 4px;
""")
self.tooltip_label.setFont(QFont("Arial", 12))
self.tooltip_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
self.tooltip_label.setVisible(False) # Inicialmente oculto

# Definindo os botões do menu e seus contextos
self.menu_buttons = [
    ("init", "init_hover", "Sobre o Projeto", self.show_inicio),
    ("pdf_button_blue", "pdf_button", "Atas (PDF)", self.show_atas),
    ("api_azul", "api", "Atas (API)", self.show_atas_api),
    ("plan_dispensa_blue", "plan_dispensa", "Dispensa Eletrônica", self.show_dispensa),
    #("contrato_blue", "contrato", "Contratos", self.show_contratos),
    #("contrato_blue", "contrato", "Planejamento", self.show_planejamento),
    ("statistics_azul", "statistics", "Indicadores", self.show_indicadores),

    ("config", "config_hover", "Configurações", self.show_config),
]

# Criando os botões e adicionando-os ao layout do menu
for icon_key, hover_icon_key, tooltip_text, callback in self.menu_buttons:
    button = self.create_icon_button(icon_key, hover_icon_key, icon_key)
    button.clicked.connect(callback)
    button.installEventFilter(self) # Instala um filtro de evento para gerenciar o
tooltip
    button.setProperty("tooltipText", tooltip_text) # Define o texto do tooltip como
propriedade
    self.menu_layout.addWidget(button)
    self.buttons[icon_key] = button

# Adiciona um espaço flexível para empurrar o ícone para o final
    self.menu_layout.addSpacerItem(QSpacerItem(20, 20, QSizePolicy.Policy.Minimum,
QSizePolicy.Policy.Expanding))

# Adiciona o ícone 360-degrees.png na parte inferior
icon_label = QLabel(self)
icon_label.setPixmap(self.icons["360-degrees"].pixmap(40, 40)) # Define o ícone com
tamanho 50x50
icon_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
self.menu_layout.addWidget(icon_label)

```

```

# Cria um widget para o menu e adiciona o layout
self.menu_widget = QWidget()
self.menu_widget.setLayout(self.menu_layout)
self.menu_widget.setStyleSheet("background-color: #13141F; ")
self.central_layout.addWidget(self.menu_widget)

def show_tooltip_with_arrow(self, text, button):
    self.tooltip_label.setText(text)
    self.tooltip_label.move(button.x() + button.width(), button.y())
    self.tooltip_label.setVisible(True)

    # Posiciona a seta
    self.tooltip_arrow.move(self.tooltip_label.x() - 10, self.tooltip_label.y() +
(self.tooltip_label.height() // 2) - 10)
    self.tooltip_arrow.setVisible(True)

def hide_tooltip(self):
    self.tooltip_label.setVisible(False)
    self.tooltip_arrow.setVisible(False)

def eventFilter(self, obj, event):
    """Filtrar eventos para exibir tooltips personalizados alinhados à direita dos botões do
menu e gerenciar ícones."""
    if isinstance(obj, QPushButton):
        # Evento de entrada do mouse no botão
        if event.type() == QEvent.Type.Enter and obj in self.buttons.values():
            tooltip_text = obj.property("tooltipText")
            if tooltip_text:
                self.tooltip_label.setText(tooltip_text)
                self.tooltip_label.adjustSize()

                # Posição do tooltip alinhada à direita do botão
                button_pos = obj.mapToGlobal(QPoint(obj.width(), 0)) # Posição global do
botão
                tooltip_x = button_pos.x() + 5 # Ajuste para a direita do botão
                tooltip_y = button_pos.y() + (obj.height() - self.tooltip_label.height()) // 2
# Centraliza verticalmente
                self.tooltip_label.move(self.mapFromGlobal(QPoint(tooltip_x, tooltip_y))) # Converte para coordenadas da janela
                self.tooltip_label.setVisible(True)

            # Altera o ícone do botão para o estado de hover, se não estiver selecionado
            if not obj.property("isSelected"):
                obj.setIcon(obj.hover_icon)

            # Evento de saída do mouse do botão
            elif event.type() == QEvent.Type.Leave and obj in self.buttons.values():
                self.tooltip_label.setVisible(False)

            # Retorna o ícone ao estado padrão, se não estiver selecionado
            if not obj.property("isSelected"):
                obj.setIcon(obj.default_icon)

```

```

# Evento de clique no botão
elif event.type() == QEvent.Type.MouseButtonPress and obj in self.buttons.values():
    # Desmarca todos os botões e reseta os ícones
    for btn in self.buttons.values():
        btn.setProperty("isSelected", False)
        btn.setIcon(btn.default_icon)

    # Marca o botão clicado como selecionado e altera o ícone
    obj.setProperty("isSelected", True)
    obj.setIcon(obj.selected_icon)

return super().eventFilter(obj, event)

def create_icon_button(self, icon_key, hover_icon_key, selected_icon_key):
    button = QPushButton()
    button.setIcon(self.icons[icon_key]) # Ícone padrão
    button.setIconSize(QSize(30, 30))
    button.setStyleSheet(get_menu_button_style())
    button.setCursor(Qt.CursorShape.PointingHandCursor)
    button.setFixedSize(50, 50)

    # Armazena os ícones padrão, de hover e de seleção
    button.default_icon = self.icons[icon_key]
    button.hover_icon = self.icons[hover_icon_key]
    button.selected_icon = self.icons[selected_icon_key]
    button.icon_key = icon_key # Armazena a chave do ícone

    # Propriedade para gerenciar o estado selecionado
    button.setProperty("isSelected", False)

    # Instala o event filter para capturar eventos de hover e selected
    button.installEventFilter(self)

return button

def set_active_button(self, button):
    """Define o botão ativo e altera o ícone para o estado hover permanente."""
    # Reseta o estilo do botão anteriormente ativo
    if self.active_button and self.active_button != button:
        self.active_button.setIcon(self.active_button.default_icon)
        self.active_button.setStyleSheet(get_menu_button_style())

    # Aplica o estilo de botão ativo
    button.setIcon(button.hover_icon)
    button.setStyleSheet(get_menu_button_activated_style())
    self.active_button = button

def show_atas(self):
    self.clear_content_area()

    # Inicializa o modelo e o caminho do banco de dados para Atas
    self.gerar_atas_model = GerarAtasModel(DATA_ATAS_PATH)

```

```

# Configura a visualização (View) e o controlador (Controller)
self.gerar_atas_view = GerarAtasView(
    icons=self.icons,
    model=self.gerar_atas_model.setup_model("controle_atas", editable=True),
    database_path=DATA_ATAS_PATH,
    parent=self
)
self.gerar_atas_controller = GerarAtasController(
    icons=self.icons,
    view=self.gerar_atas_view,
    model=self.gerar_atas_model
)

# Adiciona a view de Atas à área de conteúdo
self.content_layout.addWidget(self.gerar_atas_view)

# Define o botão "ata" como ativo
self.set_active_button(self.buttons["pdf_button_blue"])

def show_atas_api(self):
    self.clear_content_area()

    # Inicializa o modelo e o caminho do banco de dados para Atas
    self.gerar_atas_api_model = GerarAtasApiModel(DATA_ATAS_API_PATH)

    # Configura a visualização (View) e o controlador (Controller)
    self.gerar_atas_api_view = GerarAtasApiView(
        icons=self.icons,
        model=self.gerar_atas_api_model.setup_model("controle_atas_api", editable=True),
        database_path=DATA_ATAS_API_PATH,
        parent=self
    )
    self.gerar_atas_controller = GerarAtasApiController(
        icons=self.icons,
        view=self.gerar_atas_api_view,
        model=self.gerar_atas_api_model
    )

    # Adiciona a view de Atas à área de conteúdo
    self.content_layout.addWidget(self.gerar_atas_api_view)

    # Define o botão "ata" como ativo
    self.set_active_button(self.buttons["api_azul"])

def show_indicadores(self):
    self.clear_content_area()

    self.db_manager = DatabaseManager(DATA_ATAS_PATH)

    # Instanciar a nova view
    indicadores_view = IndicadoresView(
        icons = self.icons,
        db_manager=self.db_manager,

```

```

        data_atas_path=DATA_ATAS_PATH,
        data_atas_api_path=DATA_ATAS_API_PATH,
        parent=self
    )

    # Adicionar a view ao layout de conteúdo
    self.content_layout.addWidget(indicadores_view)

    # Define o botão correspondente como ativo
    self.set_active_button(self.buttons["statistics_azul"])

def show_dispensa(self):
    self.clear_content_area()

    # Instancia o modelo de Dispensa Eletrônica com o caminho do banco de dados
    self.dispensa_model = DispensaEletronicaModel(DATA_DISPENSA_ELETRONICA_PATH)

    # Configura o modelo SQL
    sql_model = self.dispensa_model.setup_model("controle_dispensas", editable=True)

    # Cria o widget de Dispensa Eletrônica e passa o modelo SQL e o caminho do banco de dados
    self.dispensa_widget = DispensaEletronicaWidget(self.icons, sql_model,
self.dispensa_model.database_manager.db_path)

    # Cria o controlador e passa o widget e o modelo
    self.controller = DispensaEletronicaController(self.icons, self.dispensa_widget,
self.dispensa_model)

    # Adiciona o widget de Dispensa Eletrônica na área de conteúdo
    self.content_layout.addWidget(self.dispensa_widget)
    self.set_active_button(self.buttons["plan_dispensa_blue"])

def show_planejamento(self):
    self.clear_content_area()

    # Instancia o modelo de Dispensa Eletrônica com o caminho do banco de dados
    self.planejamento_model = PlanejamentoModel(DATA_PLANEJAMENTO_PATH)

    # Configura o modelo SQL
    sql_model = self.planejamento_model.setup_model("controle_planejamento", editable=True)

    # Cria o widget de Dispensa Eletrônica e passa o modelo SQL e o caminho do banco de dados
    self.planejamento_view = PlanejamentoView(self.icons, sql_model,
self.planejamento_model.database_manager.db_path)

    # Cria o controlador e passa o widget e o modelo
    self.controller_planejamento = PlanejamentoController(self.icons, self.planejamento_view,
self.planejamento_model)

    # Adiciona o widget de Dispensa Eletrônica na área de conteúdo
    self.content_layout.addWidget(self.planejamento_view)
    self.set_active_button(self.buttons["contrato_blue"])

```

```

def show_contratos(self):
    self.clear_content_area()

    # Instancia o modelo de Dispensa Eletrônica com o caminho do banco de dados
    self.contratos_model = ContratosModel(DATA_CONTRATOS_PATH)

    # Configura o modelo SQL
    sql_model = self.contratos_model.setup_model("controle_contratos", editable=True)

    # Cria o widget de Dispensa Eletrônica e passa o modelo SQL e o caminho do banco de dados
    self.contratos_view      =      ContratosView(self.icons,      sql_model,
self.contratos_model.database_manager.db_path)

    # Cria o controlador e passa o widget e o modelo
    self.controller_contratos = ContratosController(self.icons, self.contratos_view,
self.contratos_model)

    # Adiciona o widget de Dispensa Eletrônica na área de conteúdo
    self.content_layout.addWidget(self.contratos_view)
    self.set_active_button(self.buttons["contrato_blue"])

def show_config(self):
    """Exibe o gerenciador de configurações."""
    self.clear_content_area()

    # Instanciar o ConfigManager com os ícones
    self.config_manager = ConfigManager(self.icons, self)

    # Adicionar o ConfigManager à área de conteúdo
    self.content_layout.addWidget(self.config_manager)

    # Define o botão correspondente como ativo
    self.set_active_button(self.buttons["config"])

def show_inicio(self):
    self.clear_content_area()

    self.inicio_widget = InicioWidget(self.icons)

    self.content_layout.addWidget(self.inicio_widget)
    # Define o botão "init" como o ativo (correspondente ao botão inicial)
    self.set_active_button(self.buttons["init"])

def open_initial_page(self):
    """Abre a página inicial da aplicação."""
    self.clear_content_area()

    # Verifica se gerar_atas_widget já foi criado, caso contrário, chama show_atas
    if not self.inicio_widget :
        self.show_inicio()
    else:
        self.content_layout.addWidget(self.inicio_widget )

```

```

# Define o botão "ata" como ativo
self.set_active_button(self.buttons["init"])

# ===== ÁREA DE CONTEÚDO =====

def setup_content_area(self):
    """Configura a área principal para exibição do conteúdo."""
    self.content_layout = QVBoxLayout()
    self.content_layout.setSpacing(0)
    self.content_layout.setContentsMargins(0, 0, 0, 0)
    self.content_layout.setAlignment(Qt.AlignmentFlag.AlignTop)
    self.content_image_label = QLabel(self.central_widget)
    self.content_image_label.hide()
    self.content_layout.addWidget(self.content_image_label)

    self.content_widget = QWidget()
    self.content_widget.setLayout(self.content_layout)
    self.content_widget.setMinimumSize(1050, 700)
    self.central_layout.addWidget(self.content_widget)

def clear_content_area(self, keep_image_label=False):
    """Remove todos os widgets da área de conteúdo, exceto a imagem opcional."""
    for i in reversed(range(self.content_layout.count())):
        widget = self.content_layout.itemAt(i).widget()
        if widget and (widget is not self.content_image_label or not keep_image_label):
            widget.setParent(None)

# ===== EVENTO DE FECHAMENTO DA JANELA =====

def closeEvent(self, event):
    """Solicita confirmação ao usuário antes de fechar a janela."""
    reply = QMessageBox.question(
        self, 'Confirmar Saída', "Você realmente deseja fechar o aplicativo?",
        QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No,
        QMessageBox.StandardButton.No
    )
    event.accept() if reply == QMessageBox.StandardButton.Yes else event.ignore()

if __name__ == "__main__":
    import sys

    app = QApplication(sys.argv)
    #dark_theme = get_full_dark_theme()
    #app.setStyleSheet(dark_theme)
    window = MainWindow()
    window.show()
    sys.exit(app.exec())

```