MVC
(Model-View-Controller)

Letícia Vidal

MVC

O padrão **MVC (Model-View-Controller)** é uma das arquiteturas mais sólidas para organizar aplicações desktop, especialmente ao utilizar o **PyGtk (ou PyGObject)**. O objetivo principal é separar os dados, a interface visual e a lógica que une os dois.

Aqui está uma introdução prática de como aplicar esse conceito no seu desenvolvimento com Python e GTK

Os Três Pilares do MVC

Model (O Dado)

É o "cérebro" dos dados da sua aplicação. Ele não sabe nada sobre botões ou janelas; ele apenas gerencia a informação e as regras de negócio.

- **Em Python:** Geralmente são classes que lidam com arquivos, bancos de dados SQL ou listas de objetos.
- **Responsabilidade:** Validar dados, salvar informações e notificar quando algo mudou.

View (A Interface)

É o que o usuário vê na tela. No GTK, isso envolve seus widgets (`Gtk.Window`, `Gtk.Button`, etc.).

- **Em GTK:** Pode ser definido via código puro ou via arquivos **.ui** gerados pelo **Cambalache** ou **Glade**.
- **Responsabilidade:** Exibir dados ao usuário e capturar eventos (cliques, digitação), mas **não** processar a lógica desses eventos.

Controller (O Maestro)

É a ponte entre o Model e a View. Ele escuta as ações do usuário na View e decide o que o Model deve fazer.

- **Em Python:** É a classe que conecta os sinais do GTK (como o sinal `clicked`) às funções que alteram o Model.
- **Responsabilidade:** Receber entradas da View, atualizar o Model e atualizar a View com os novos dados.

Por que usar isso no GTK?

Usar MVC evita que seu código se torne um "espaguete" dentro de um único arquivo gigante. As principais vantagens são:

- **Testabilidade:** Você pode testar a lógica do seu Model sem precisar abrir uma janela.
- **Manutenabilidade:** Quer trocar o GTK pelo Qt ou uma interface web? Se o Model estiver separado, você só precisa refazer a View e o Controller.
- **Organização:** Facilita o trabalho em equipe (um foca no design do `.ui`, outro na lógica do banco de dados).

Exemplo Prático: Um Contador Simples

Imagine um app com um rótulo (label) e um botão de incremento.

O Model

```
class CounterModel:
    def __init__(self):
        self._value = 0

    def increment(self):
        self._value += 1

    @property
    def value(self):
        return self._value
```

A View

```
import gi
gi.require_version("Gtk", "4.0")
from gi.repository import Gtk

class CounterView(Gtk.ApplicationWindow):
    def __init__(self, app):
        super().__init__(application=app, title="Exemplo MVC")

        self.box = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=10)
        self.label = Gtk.Label(label="0")
        self.button = Gtk.Button(label="Incrementar")

        self.box.append(self.label)
        self.box.append(self.button)
        self.set_child(self.box)
```

O Controller

```
class CounterController:
    def __init__(self, model, view):
        self.model = model
        self.view = view

        # Conecta o sinal do GTK à lógica do Controller
        self.view.button.connect("clicked", self.on_button_clicked)

    def on_button_clicked(self, button):
        self.model.increment()
        # Atualiza a View com o novo valor do Model
        self.view.label.set_text(str(self.model.value))
```

Dica de Ouro para GTK Moderno

Ao trabalhar com GTK 4, considere usar o **GObject Properties** e **Data Binding**. O GTK possui mecanismos nativos para "ligar" uma propriedade do Model diretamente a um widget da View, o que automatiza parte do trabalho do Controller.

Você já costuma criar suas interfaces via código ou prefere usar ferramentas visuais como o Cambalache?