**Arquitetura Funcional do Sistema de Compra de Ingressos e Produtos**

**1. Visão Geral**

Aplicação desktop desenvolvida em **Python**, permitindo:

* Seleção de ingressos (quantidade e tipo).
* Adição de produtos opcionais.
* Ações de **reserva** ou **compra**, com cálculo automático dos valores.

**2. Componentes Funcionais**

**2.1. Interface Gráfica (UI)**

* **Lib:** wxPython.
* **Descrição:** Interface limpa e intuitiva.
* **Componentes:**
  + **Seletores** para escolher quantidade de ingressos e tipos.
  + **Campos de texto** para especificar a quantidade de pedidos, caso as opções pré-definidas pelo usuário não atendam.
  + **Checkboxes** para uma fácil seleção de sabores de produtos ou tipos de compra.
  + **Botões** para realizar ações de **Reservar** ou **Comprar**.

**2.2. Modelos de Dados**

* **Ingresso:**
  + Atributos: tipo (inteira ou meia) e preço.
* **Produto:**
  + Atributos: nome, preço e lista de sabores/variações.

**2.3. Lógica de Negócio**

* Validação dos dados inseridos.
* Gerenciamento da quantidade de ingressos e produtos.
* Cálculo do valor total da compra ou reserva.
* Exibição de resumo da operação.

**3. Fluxo de Funcionamento**

1. **Seleção da quantidade de ingressos.**
2. **Escolha do tipo de ingresso (inteira, meia ou personalizado).**
3. **Seleção de produtos adicionais e quantidades, com sabores opcionais.**
4. **Ação:** reservar ou comprar.
5. **Exibição do resumo com detalhes e valor total.**

**4. Regras de Negócio**

* Ajuste dinâmico de meias e inteiras no modo **Personalizado**.
* Validação da seleção de produtos e suas quantidades.
* Cálculo total com base em ingressos e produtos selecionados.

**5. Arquitetura de Classes**

+----------------+ +----------------+

| Produto | | Ingresso |

+----------------+ +----------------+

| - nome | | - tipo |

| - preco | | - preco |

| - opcoes | +----------------+

+----------------+

▲

|

| contém

|

+------------------------------+

| MudancasPersonalizadasFrame |

+------------------------------+

| - ingressos |

| - produtos |

| - UI Components |

+------------------------------+

| +get\_quantidade() |

| +get\_tipo\_ingressos() |

| +get\_produtos() |

| +exibir\_resultado() |

| +on\_comprar() |

| +on\_reservar() |

| +on\_total\_changed() |

+------------------------------+

**6. Possíveis Integrações Futuras com AWS**

**6.1. Serviços AWS que podem ser integrados**

| **Serviço AWS** | **Função** |
| --- | --- |
| **Amazon S3** | Armazenamento de comprovantes, ingressos em PDF ou imagens. |
| **Amazon DynamoDB** | Banco NoSQL para armazenar compras, reservas, histórico. |
| **AWS Lambda** | Processamento serverless para gerar recibos, notificações. |
| **Amazon API Gateway** | Criação de APIs para comunicação entre o app e backend. |
| **Amazon SNS ou SES** | Envio de notificações por SMS ou e-mail (confirmações). |
| **Amazon Cognito** | Autenticação de usuários (login, cadastro, segurança). |
| **AWS AppConfig/SSM** | Gerenciamento remoto de configurações do app. |
| **AWS CloudWatch Logs** | Monitoramento de erros, logs e métricas do aplicativo. |

**6.2. Como o Código Está Apto para Essas Integrações?**

* ✅ **Uso de Programação Orientada a Objetos (POO):**  
  Classes como Produto e Ingresso encapsulam bem os dados, facilitando expansão.
* ✅ **Separação Clara de Responsabilidades:**  
  A interface gráfica está bem separada da lógica de negócios. Isso permite transformar partes da lógica em APIs, funções em nuvem ou microserviços.
* ✅ **Dados Estruturados e Serializáveis:**  
  Informações como produtos e ingressos podem ser convertidas facilmente para JSON, permitindo comunicação com APIs (API Gateway) e bancos como DynamoDB.
* ✅ **Eventos Prontos para Integração:**  
  As ações disparadas pelos botões **"Comprar"** e **"Reservar"** são pontos naturais para enviar dados para funções Lambda, onde essas transações, após serem efetuadas gerariam comprovantes que seriam armazenados no S3 e enviariam confirmações por e-mail ou SMS.

**7. Exemplo de Arquitetura Expandida com AWS**

[Usuário - App wxPython]

↓

[API Gateway] ---> [AWS Lambda] ---> [DynamoDB] (Pedidos)

↳ [S3] (Comprovantes PDFs)

↳ [SES/SNS] (E-mail/SMS confirmação)

**8. Benefícios da Arquitetura Atual para a Nuvem**

* 🟩 **Extensível:** Fácil criar um backend na nuvem sem alterar a interface desktop.
* 🟩 **Escalável:** Pode ser migrado para a nuvem mantendo a lógica atual.
* 🟩 **Baixo custo:** Utilizando serviços serverless como Lambda, paga-se apenas pelo uso.
* 🟩 **Alta disponibilidade:** Armazenamento, autenticação e APIs gerenciadas pela AWS oferecem alta confiabilidade.