

IFRN

INFOWEB – POO EM PYTHON

Streamlit – Sistema de Agendamento

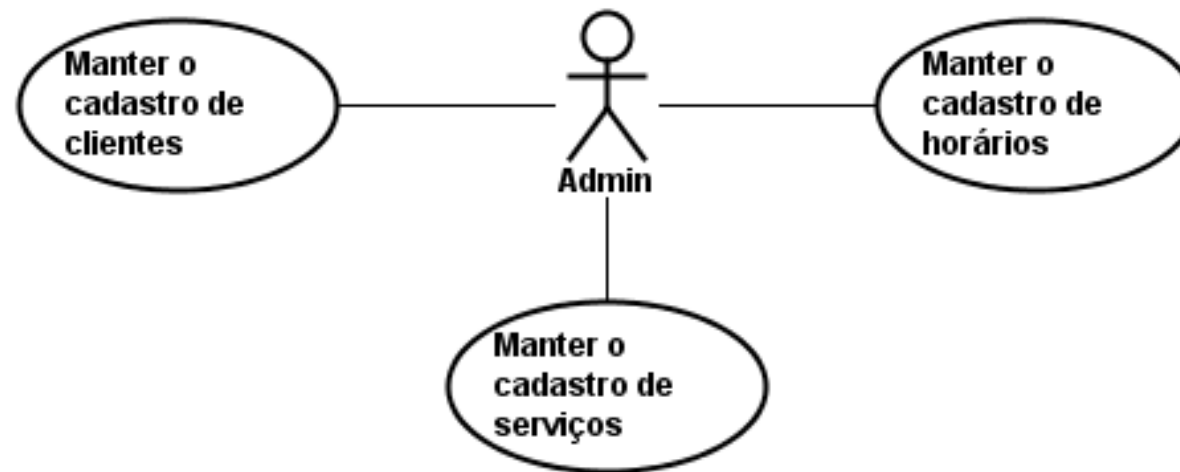
Prof. Gilbert Azevedo

Conteúdo

- Sistema de Agendamento
 - Diagramas de Casos de Uso e de Classes
- Cadastro de clientes *com Streamlit*
 - Utilização do *Streamlit* na programação em camadas
 - Funções *tabs*, *selectbox*, *dataframe* do *Streamlit*

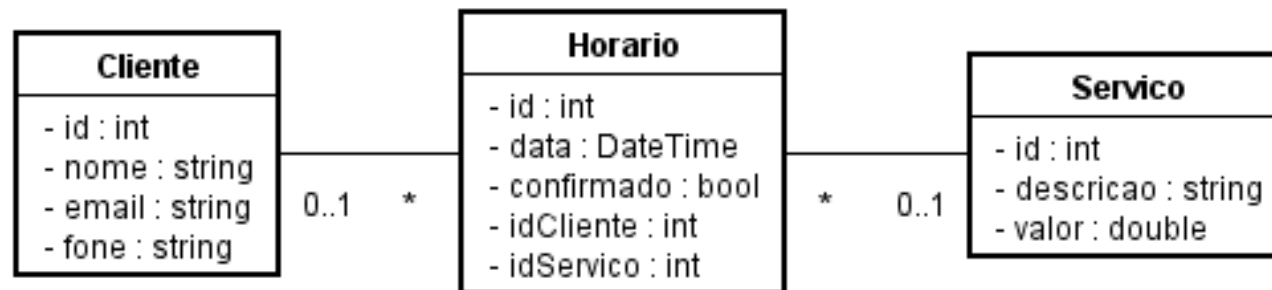
Sistema de Agendamento

- O objetivo principal do sistema é permitir que clientes agendem serviços a serem realizados em um dia e horário específicos
- Por exemplo: clínicas, oficinas mecânicas, salões de beleza, pet-shops, ...



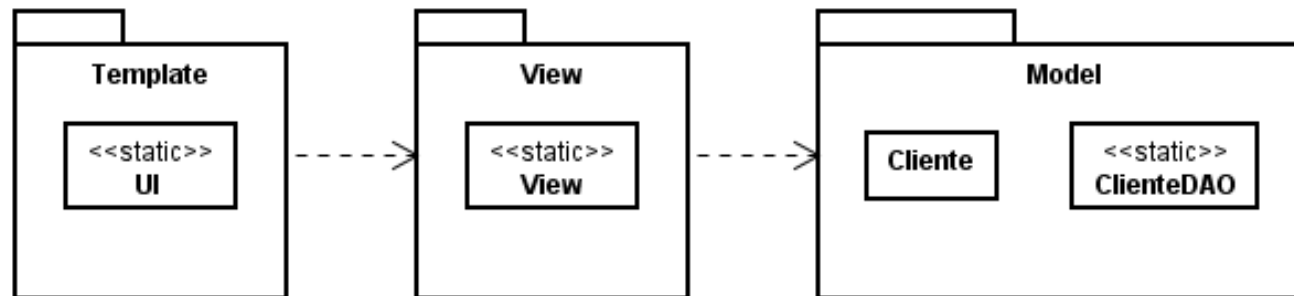
Modelo do Sistema

- As entidades Cliente, Serviço e Horário representam as entidades do sistema
- O diagrama mostra que um cliente pode agendar vários horários e, em cada horário, realizar um serviço

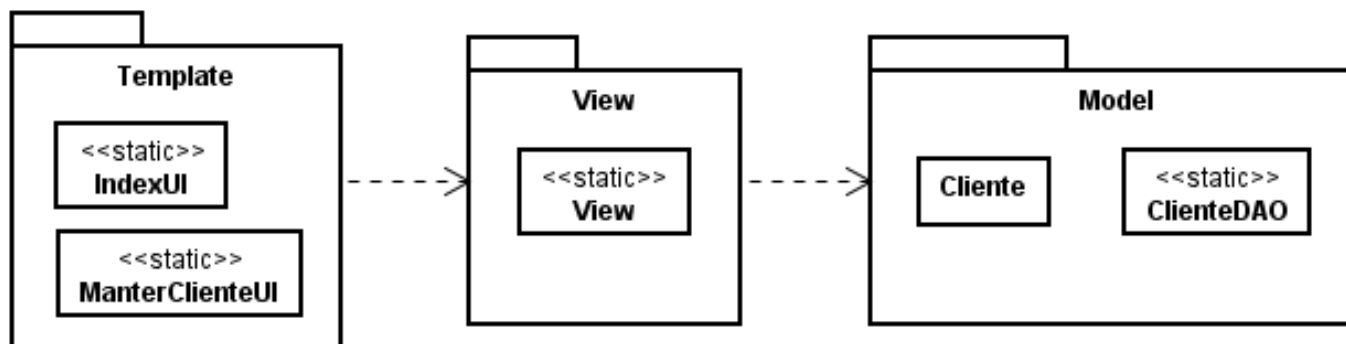


Cadastro de Clientes

- O *Streamlit* vai ser usado para construir uma camada de interface com o usuário baseada em páginas

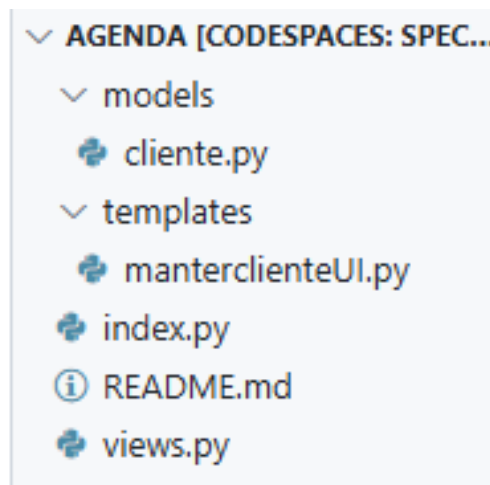


- A classe UI será substituída pelas classes IndexUI e ManterClienteUI



Passo 1. Organização do Código Fonte

- Organize a pasta do projeto Agenda de forma similar a figura abaixo:
- A pasta *models* vai conter as entidades e os DAOs
- A pasta *templates* vai conter as páginas da aplicação
- O arquivo *index.py* vai ter a classe principal que acessa as páginas em *templates*
- O arquivo *views.py* vai ter a(s) classe(s) com as operações do sistema



Passo 2. Definição da classe Cliente

- Programe a classe Cliente de acordo com o diagrama abaixo
- Insira os métodos *get* e *set* não listados no diagrama
- O método *to_json* deve retornar um dicionário com os dados de um cliente
- O método *from_json* deve retornar um Cliente com base no dicionário informado

Cliente
- id : int - nome : string - email : string - fone : string
+ __init__(id : int, n : string, e : string, f : string) + __str__() : string + to_json() : dict + from_json(dic : dict) : Cliente

Passo 2.1. Cliente.__init__ e __str__

```
class Cliente:
```

```
    def __init__(self, id, nome, email, fone):
```

```
        self.set_id(id)
```

```
        self.set_nome(nome)
```

```
        self.set_email(email)
```

```
        self.set_fone(fone)
```

```
    def __str__(self):
```

```
        return f"{self.__id} - {self.__nome} - {self.__email} -  
                {self.__fone}"
```

Cliente
- id : int - nome : string - email : string - fone : string
+ __init__(id : int, n : string, e : string, f : string) + __str__() : string + to_json() : dict + from_json(dic : dict) : Cliente

Passo 2.2. Cliente – *gets e sets*

```
def get_id(self): return self.__id
def get_nome(self): return self.__nome
def get_email(self): return self.__email
def get_fone(self): return self.__fone

def set_id(self, id): self.__id = id
def set_nome(self, nome): self.__nome = nome
def set_email(self, email): self.__email = email
def set_fone(self, fone): self.__fone = fone
```

Passo 2.3. Cliente – operações com *json*

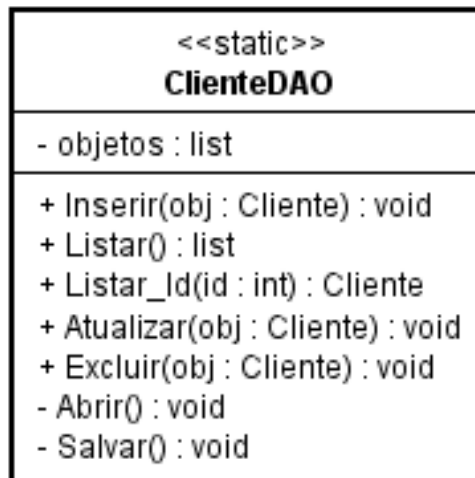
```
def to_json(self):  
    dic = {"id":self.__id, "nome":self.__nome,  
          "email":self.__email, "fone":self.__fone}  
    return dic
```

```
@staticmethod
```

```
def from_json(dic):  
    return Cliente(dic["id"], dic["nome"], dic["email"],  
                   dic["fone"])
```

Passo 3. ClienteDAO

- Programe a classe ClienteDAO de acordo com o diagrama abaixo
- A classe mantém a lista de objetos clientes do cadastro
- Inserir, Listar, Listar_Id, Atualizar e Excluir realizam as operações do CRUD
- Abrir e Salvar realizam as operações com o arquivo *json*



Passo 3.1. ClienteDAO.inserir

```
import json
```

```
class ClienteDAO:  
    __objetos = []  
    @classmethod  
    def inserir(cls, obj):  
        cls.abrir()  
        id = 0  
        for aux in cls.__objetos:  
            if aux.get_id() > id: id = aux.get_id()  
        obj.set_id(id + 1)  
        cls.__objetos.append(obj)  
        cls.salvar()
```

Passo 3.2. ClienteDAO.listar

```
@classmethod
def listar(cls):
    cls.abrir()
    return cls.__objetos
```

```
@classmethod
def listar_id(cls, id):
    cls.abrir()
    for obj in cls.__objetos:
        if obj.get_id() == id: return obj
    return None
```

Passo 3.3. ClienteDAO.atualizar

```
@classmethod
def atualizar(cls, obj):
    aux = cls.listar_id(obj.get_id())
    if aux != None:
        cls.__objetos.remove(aux)
        cls.__objetos.append(obj)
        cls.salvar()
```

Passo 3.4. ClienteDAO.excluir

```
@classmethod
def excluir(cls, obj):
    aux = cls.listar_id(obj.get_id())
    if aux != None:
        cls.__objetos.remove(aux)
        cls.salvar()
```

Passo 3.5. ClienteDAO.abrir

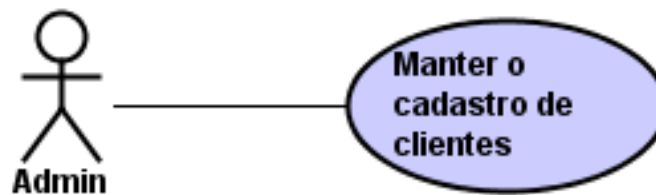
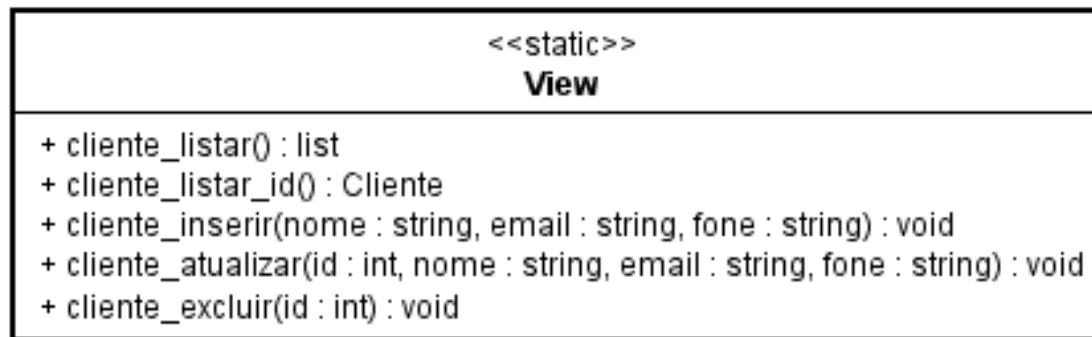
```
@classmethod
def abrir(cls):
    cls.__objetos = []
    try:
        with open("clientes.json", mode="r") as arquivo:
            list_dic = json.load(arquivo)
            for dic in list_dic:
                obj = Cliente.from_json(dic)
                cls.__objetos.append(obj)
    except FileNotFoundError:
        pass
```


Passo 3.6. ClienteDAO.salvar

```
@classmethod
def salvar(cls):
    with open("clientes.json", mode="w") as arquivo:
        json.dump(cls.__objetos, arquivo, default =
            Cliente.to_json)
```

Passo 4. View

- Programe a classe View de acordo com o diagrama abaixo
- A classe mantém as operações do sistema que serão chamadas pela interface com o usuário



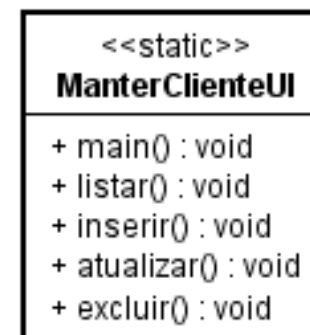
Passo 4.1. Classe View

```
from models.cliente import Cliente, ClienteDAO
class View:
    def cliente_listar():
        return ClienteDAO.listar()
    def cliente_listar_id(id):
        return ClienteDAO.listar_id(id)
    def cliente_inserir(nome, email, fone):
        cliente = Cliente(0, nome, email, fone)
        ClienteDAO.inserir(cliente)
    def cliente_atualizar(id, nome, email, fone):
        cliente = Cliente(id, nome, email, fone)
        ClienteDAO.atualizar(cliente)
    def cliente_excluir(id):
        cliente = Cliente(id, "", "", "")
        ClienteDAO.excluir(cliente)
```

Passo 5. ManterClienteUI

- A classe ManterClienteUI vai ser usada para montar uma interface com o usuário para o cadastro de clientes usando o Streamlit
- Cada uma das operações do CRUD: inserir, listar, atualizar e excluir, será realizada em um método diferente da classe, controlado pelo *main*
- Importe os módulos necessários à classe ManterClienteUI

```
import streamlit as st
import pandas as pd
import time
from views import View
```



Passo 5.1. ManterClienteUI.main

- Main utiliza a função *tabs* do Streamlit para controlar abas na página

```
class ManterClienteUI:
```

```
    def main():
        st.header("Cadastro de Clientes")
        tab1, tab2, tab3, tab4 = st.tabs(["Listar", "Inserir",
                                           "Atualizar", "Excluir"])
        with tab1: ManterClienteUI.listar()
        with tab2: ManterClienteUI.inserir()
        with tab3: ManterClienteUI.atualizar()
        with tab4: ManterClienteUI.excluir()
```

Passo 5.1. ManterClienteUI.main

- Main utiliza a função *tabs* do Streamlit para controlar de abas na página
- O tabs permite fazer “várias páginas” na mesma página
- Listar

Cadastro de Clientes

Listar Inserir Atualizar Excluir

Nenhum cliente cadastrado

- Inserir

Cadastro de Clientes

Listar Inserir Atualizar Excluir

Informe o nome

Passo 5.2. ManterClienteUI.listar

- Listar usa um *dataframe* para apresentar os clientes cadastrados obtidos a partir da operação `cliente_listar` da classe `View`
- O *dataframe* é apresentado a partir de uma lista de dicionários montada a partir da lista de clientes

```
def listar():  
    clientes = View.cliente_listar()  
    if len(clientes) == 0: st.write("Nenhum cliente cadastrado")  
    else:  
        list_dic = []  
        for obj in clientes: list_dic.append(obj.to_json())  
        df = pd.DataFrame(list_dic)  
        st.dataframe(df)
```

Passo 5.2. ManterClienteUI.listar

- Listar usa um *dataframe* para apresentar os clientes cadastrados obtidos a partir da operação cliente_listar da classe View
- O *dataframe* é apresentado a partir de uma lista de dicionários montada a partir da lista de clientes

Cadastro de Clientes					
Listar Inserir Atualizar Excluir					
	id	nome	email	fone	
0	1	Gilbert	gilbert@email.com	123456789	
1	2	Azevedo	azevedo@email.com	987654321	

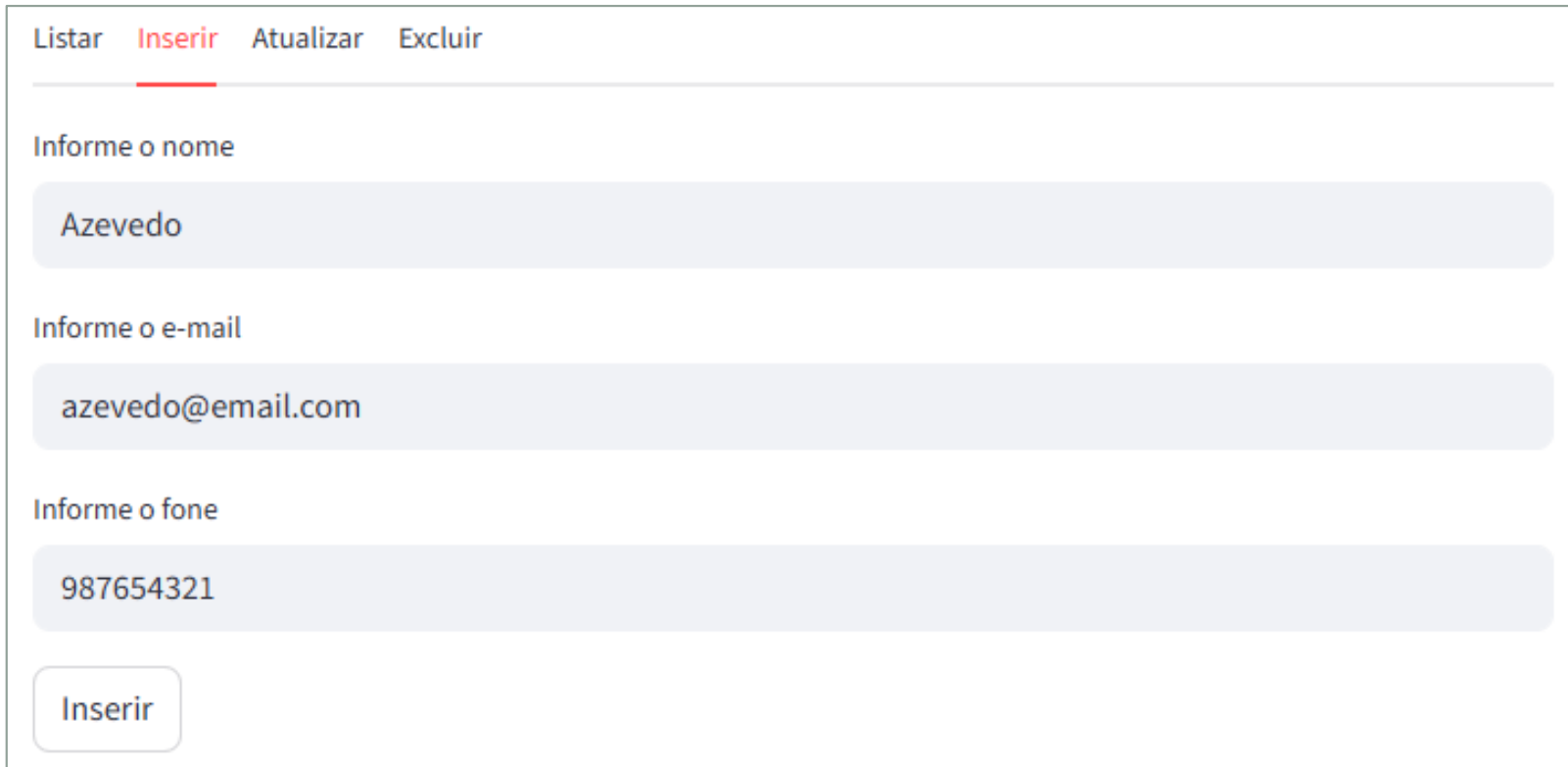
Passo 5.2. ManterClienteUI.inserir

- Inserir usa *text_inputs* para ler os dados do usuário e chama a operação de inserir da view para cadastrar o cliente

```
def inserir():  
    nome = st.text_input("Informe o nome")  
    email = st.text_input("Informe o e-mail")  
    fone = st.text_input("Informe o fone")  
    if st.button("Inserir"):  
        View.cliente_inserir(nome, email, fone)  
        st.success("Cliente inserido com sucesso")  
        time.sleep(2)  
        st.rerun()
```

Passo 5.2. ManterClienteUI.inserir

- Inserir usa *text_inputs* para ler os dados do usuário e chama a operação de inserir da view para cadastrar o cliente



The screenshot shows a web interface for inserting a client. At the top, there is a navigation bar with four links: 'Listar', 'Inserir' (highlighted in red), 'Atualizar', and 'Excluir'. Below the navigation bar, there are three text input fields. The first field is labeled 'Informe o nome' and contains the text 'Azevedo'. The second field is labeled 'Informe o e-mail' and contains the text 'azevedo@email.com'. The third field is labeled 'Informe o fone' and contains the text '987654321'. At the bottom left of the form, there is a button labeled 'Inserir'.

Passo 5.3. ManterClienteUI.atualizar

- Inserir usa *select_box* e *text_inputs* para atualizar os dados de um cliente

```
def atualizar():  
    clientes = View.cliente_listar()  
    if len(clientes) == 0: st.write("Nenhum cliente cadastrado")  
    else:  
        op = st.selectbox("Atualização de Clientes", clientes)  
        nome = st.text_input("Novo nome", op.get_nome())  
        email = st.text_input("Novo e-mail", op.get_email())  
        fone = st.text_input("Novo fone", op.get_fone())  
        if st.button("Atualizar"):  
            id = op.get_id()  
            View.cliente_atualizar(id, nome, email, fone)  
            st.success("Cliente atualizado com sucesso")
```

Passo 5.3. ManterClienteUI.atualizar

- Inserir usa *select_box* e *text_inputs* para atualizar os dados de um cliente

Listar Inserir **Atualizar** Excluir

Atualização de Clientes

1 - Gilbert - gilbert@email.com - 123456789 ▼

Informe o novo nome

Gilbert

Informe o novo e-mail

gilbert@email.com

Informe o novo fone

123456789

Atualizar

Passo 5.4. ManterClienteUI.excluir

- Inserir usa *select_box* para excluir um cliente

```
def excluir():  
    clientes = View.cliente_listar()  
    if len(clientes) == 0: st.write("Nenhum cliente cadastrado")  
    else:  
        op = st.selectbox("Exclusão de Clientes", clientes)  
        if st.button("Excluir"):  
            id = op.get_id()  
            View.cliente_excluir(id)  
            st.success("Cliente excluído com sucesso")
```

Passo 5.4. ManterClienteUI.excluir

- Inserir usa *select_box* para excluir um cliente

Listar Inserir Atualizar **Excluir**

Exclusão de Clientes

1 - Gilbert - gilbert@email.com - 123456789



Excluir

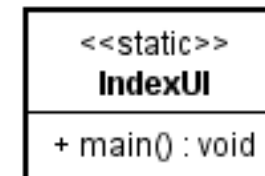
Passo 6. IndexUI

- A classe IndexUI, por enquanto, tem a tarefa única de mostrar a página do cadastro de clientes

```
from templates.manterclienteUI import ManterClienteUI
```

```
class IndexUI:  
    def main():  
        ManterClienteUI.main()
```

```
IndexUI.main()
```



Referências

- Documentação do Streamlit
 - <https://docs.streamlit.io/>
- Código completo da aplicação
 - <https://github.com/Gilbert-Silva/Agenda>