

Trabalho final
Programação Orientada de Objetos I
UFSC - Florianópolis

Reference smile

SEU SORRISO
É a nossa alegria

Pamela Santos Monteiro - 18204975
Thabata John Wiggers dos Santos - 19100038

INTRODUÇÃO



Neste projeto, desenvolvemos uma aplicação de uma agenda dentista “**Reference Smile**” que opera como um sistema oferece um método prático e intuitivo para gerenciar a agenda entre dentista e pacientes, reduzindo a sobrecarga e os atrasos nas marcações.

A tela de inicial é o primeiro contato do usuário com a aplicação. O usuário vai identificar como cliente ou como dentista. Caso ele se identifica como dentista ele vai se encaminhado para pagina do dentista. Caso contrario ele vai se encaminhado para pagina do cliente para agendar as consultas.

A segunda tela depende se ele é **cliente** ou **dentista**. Caso seja cliente, o usuário terá dois botões com as suas possíveis ações:

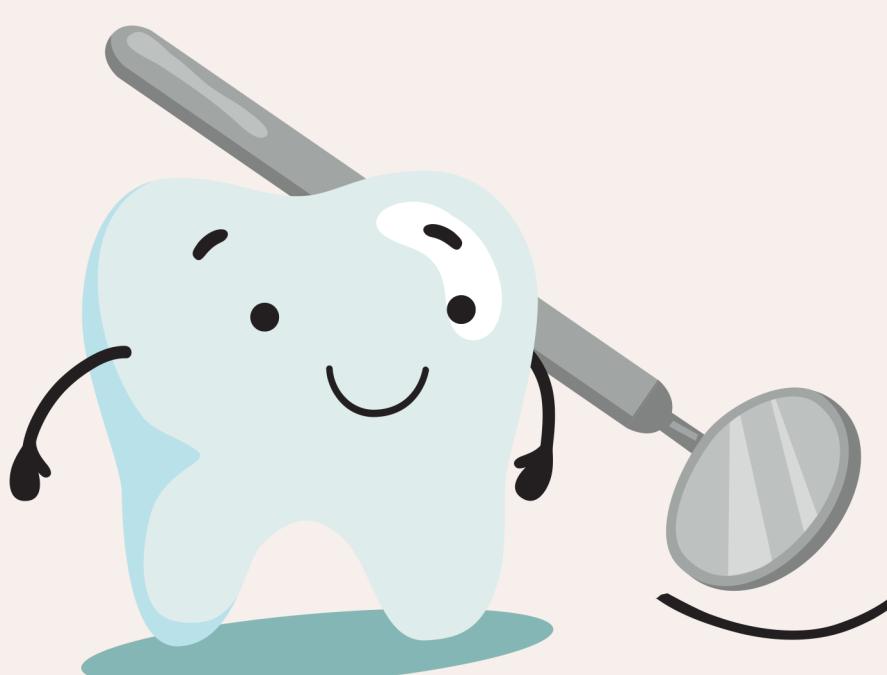
- **Adicionar Cliente** - Fazer o Cadastro do cliente.
- **Agendar consulta** - Só depois de realizar o cadastro.

Caso seja **Dentista**, o usuário terá dois botões com as suas possíveis ações:

- **Adicionar Dentista** - Fazer o Cadastro do Dentista e seu Horário de atendimento.
- **Verificar consulta** - Visualizar os horários de atendimentos dos pacientes (Clientes).

Tecnologias Utilizadas:

- Lógica de Programação: Python e Programação Orientada de Objetos.
- Interface Gráfica: Tkinter para criação das telas de interação com o usuário.



DESENVOLVIMENTO



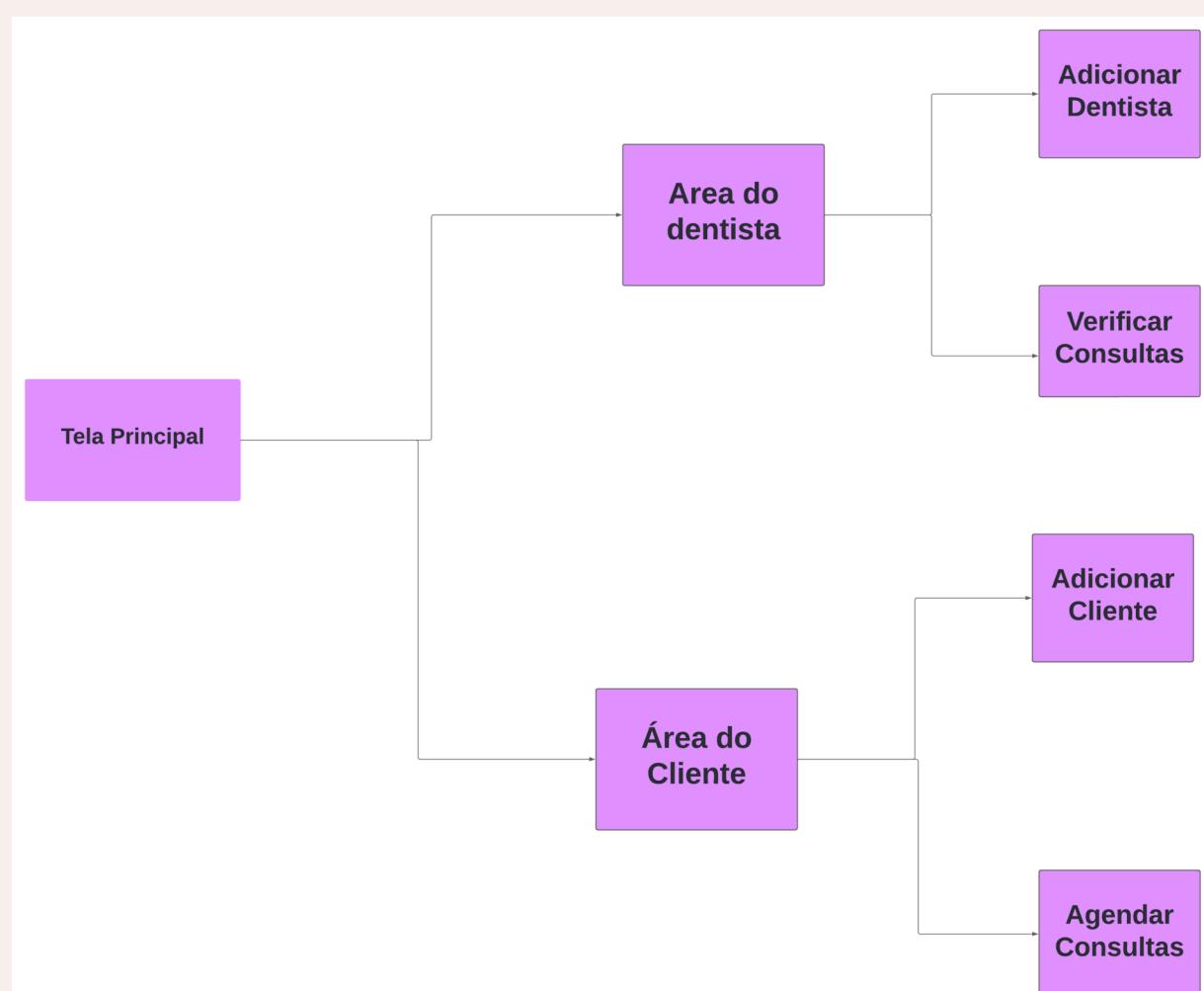
Antes de iniciar o desenvolvimento do código, foi crucial estabelecer as ideias sobre as telas e suas finalidades, bem como o funcionamento detalhado da agenda e suas marcações de horários.

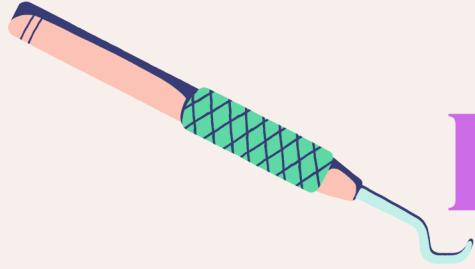
PROTOTIPO TELAS - LAYOUT



Foram feitos alguns crocs na ferramenta Canva para visualização melhor e ter ideias das telas desenvolvida nesse projeto.

FLUXO DE TELAS





DESENVOLVIMENTO

Para as entidades de interesse da nossa aplicação, que são os clientes e os dentistas, nós utilizamos herança entre classes para modelá-las. Primeiramente foi criada uma classe chamada User, que representa todos os usuários do sistema.

Essa classe tem os seguintes atributos: nome e tipo

```
model > user.py > User > get_type
1  class User:
2      def __init__(self, name):
3          self.name = name
4          self.type = ""
5
6      def get_name(self):
7          return self.name
8
9      def get_type(self):
10         return self.type
```

A **classe User** é a classe pai das classes Client, que representa os clientes da nossa aplicação, e da classe Dentist, que representa os dentistas da nossa aplicação.

A class Client é relativamente simples, com somente um atributo extra, chamado cpf. Além disso, os únicos métodos adicionais desta classe são o get_cpf e set_cpf. Já para a classe Dentist, nós temos mais atributos e métodos. Os atributos adicionais para a classe Dentist são: CRO, horário de início, horário de fim, e agenda do dentista, que é um objeto da classe Schedule.

Além disso, a **classe Dentist** tem métodos para validar o horário de início e de fim, pois o horário de ínicio não pode ser igual ou maior que o horário de fim

Por fim, a **classe Schedule** é a agenda do dentista. Ela contém os **atributos schedule**, que é um **dicionário** cuja chave é um dia do ano e os valores são o horário de inicio, o horário de fim da consulta e o Cliente marcado para aquela consulta.

```
> schedule.py > ...
class Schedule:
    def set_schedule(self, day: date, start_time: int, end_time: int, dentist_start_time: int, dentist_end_time: int, client: Client) -> None:
        is_schedule_valid = self.validate_schedule_time(start_time, end_time, dentist_start_time, dentist_end_time)
        is_overlapping = self.is_overlap(day, start_time, end_time)
        if is_schedule_valid and not is_overlapping:
            self.schedule[day].append((start_time, end_time, client))
            return True
        else:
            return False

    def get_all_schedule(self) -> defaultdict:
        return self.schedule

    def get_schedule(self, day: date) -> list:
        return self.schedule.get(day, ["Não há compromissos nesse dia."])

    def get_schedule_as_text(self, day: date) -> str:
        schedule = self.get_schedule(day)
        if schedule == "Não há compromissos nesse dia.":
            return [schedule]
        schedule_text = []
        for appointment in schedule:
            schedule_text.append(f'{appointment[0]}:00 - {appointment[1]}:00 - {appointment[2].get_name()}\n')
        return schedule_text

    def validate_schedule_time(self, start_time: int, end_time: int, dentist_start_time: int, dentist_end_time: int) -> bool:
        if start_time < end_time and start_time >= dentist_start_time and end_time <= dentist_end_time:
            return True
        return False

    def is_overlap(self, day: date, start_time: int, end_time: int) -> bool:
        for schedule in self.schedule[day]:
            if start_time < schedule[1] and end_time > schedule[0]:
                return True
        return False
```

Outro exemplo é a Classe Telas:

Foram usados métodos **create_screen()**, é específico para cada telas e neles são adicionados elementos que compõem as apresentações básicas de cada telas. Por exemplo nesse método é adicionado o plano de fundos, maioria dos texto e as figuras.

```
❸ initial_screen.py > ...
1  from tkinter import *
2  from tkinter import ttk
3  from client_area import ClientArea
4  from dentist_area import DentistArea
5  from constants import *
6
7  class InitialScreen:
8
9      def __init__(self, root):
10         self.root = root
11         self.dentists = []
12         self.clients = []
13
14         self.root.title("Reference Smile - Sistema de Agendamento de consultas dentárias")
15         self.root.geometry(f"{WIDTH}x{HEIGHT}")
16         self.root.resizable(False, False)
17
18
19         # Frame principal com conteúdo
20         mainframe = ttk.Frame(self.root, padding="3 3 12 12")
21         mainframe.grid(column=0, row=0, sticky=(N, W, E, S))
22         self.root.columnconfigure(0, weight=1)
23         self.root.rowconfigure(0, weight=1)
24
25         self.create_screen()
26
27         for child in mainframe.winfo_children():
28             child.grid_configure(padx=5, pady=5)
29
30         self.root.protocol("WM_DELETE_WINDOW", self.on_close)
31
32     def dentist_area(self):
33         dentists = DentistArea(self.root, self.dentists)
34         self.dentists = dentists.dentists
35
36     def client_area(self):
37         clients = ClientArea(self.root, self.clients, self.dentists)
38         self.clients = clients.clients
```

E também foram utilizados **create_buttons()**, para criar os botões das telas e **create_input()** para criar as entradas.



SCREENSHOTS



TELA PRINCIPAL

REFERENCE SMILE



Área do Dentista
Área do Cliente



A equipe da Reference Smile está pronta para atendê-lo.

ÁREA DO DENTISTA



Adicionar Dentista
Verificar Consulta

←

ADICIONAR DENTISTA

Nome:

CRO:

Inicio do expediente (Hora):

Fim do expediente (Hora):



Adicionar

ADICIONAR CLIENTE

Nome:

CPF:



Adicionar

AGENDAR CONSULTAS

ESCOLHA SEU DENTISTA

CPF DO CLIENTE

NOME DO CLIENTE

DIA DA CONSULTA:

HORÁRIO DE INÍCIO

HORÁRIO DE FIM

Adicionar

VERIFICAR CONSULTAS

Dentista

DIA DA CONSULTA:

CONSULTAS:



Adicionar

←

Papel de cada membro:

O design da aplicação foi feito em conjunto entre **Pamela** e **Thabata**. O fluxo de telas foi feita pela aluna **Thabata**. Além disso ela desenvolveu as telas de adicionar cliente e dentista. O desenvolvimento das classes e demais telas foram feitas pela aluna **Pamela**.