**Sistema de Gerenciamento de Reservas em Restaurantes - Documentação Técnica**

**1. Identificação**

-Nome do sistema: Sistema de Gerenciamento de Reservas em Restaurantes

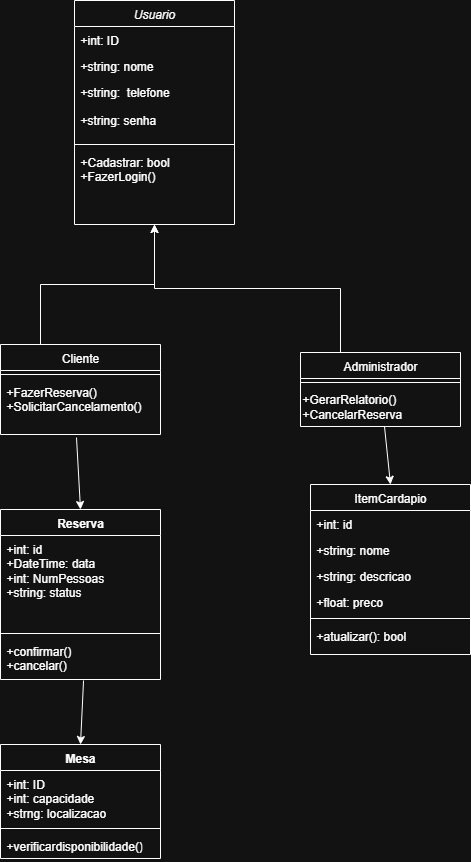
- Equipe:

- Erison Oliveira - Desenvolvedor Backend/Banco de Dados

- Marco Rocha - Desenvolvedor Frontend

- Victor Gabriel - Desenvolvedor Full Stack

**2. Diagrama UML de Classes (Detalhado)**



Relações:

- Herança: Cliente, Funcionario e Administrador herdam de Usuario

- Associação: Reserva associada a Mesa e Cliente

- Agregação: Relatório associado a Reserva

**3. Diagramas de Interação**

Diagrama de Sequência - Fazer Reserva

1. Cliente acessa sistema e faz login

2. Sistema autentica usuário

3. Cliente navega para "Fazer Reserva"

4. Sistema exibe formulário com campos: data, hora, número de pessoas, local preferência

5. Cliente preenche dados e submete

6. Sistema verifica disponibilidade de mesas

7. Sistema cria reserva no banco de dados

8. Sistema envia confirmação por email

9. Sistema exibe confirmação para o cliente

Diagrama de Atividade - Processo de Reserva

[Início] → Cliente solicita reserva → Verificar disponibilidade →

[Disponível?] → Sim → Criar reserva → Enviar confirmação → [Fim]

|

Não → Sugerir alternativas → [Fim]

**4. Protótipo de Código (Skeleton Classes)**

Estrutura de pastas:

restaurante\_reservas/

├── manage.py

├── restaurante\_reservas/

│ ├── \_\_init\_\_.py

│ ├── configuracoes.py

│ ├── urls.py

│ └── wsgi.py

├── core/

│ ├── \_\_init\_\_.py

│ ├── modelos.py

│ ├── visualizacoes.py

│ ├── admin.py

│ └── excecoes.py

├── reservas/

│ ├── \_\_init\_\_.py

│ ├── modelos.py

│ ├── visualizacoes.py

│ ├── urls.py

│ └── templates/

├── cardapio/

│ ├── \_\_init\_\_.py

│ ├── modelos.py

│ ├── visualizacoes.py

│ └── templates/

└── relatorios/

├── \_\_init\_\_.py

├── visualizacoes.py

└── templates/

```python

from django.db import models

from django.contrib.auth.models import AbstractBaseUser, BaseUserManager

**class UsuarioManager(BaseUserManager):**

def create\_user(self, email, nome, telefone, password=None):

# Implementar criação de usuário

pass

def create\_superuser(self, email, nome, telefone, password):

# Implementar criação de superusuário

pass

**class Usuario(AbstractBaseUser):**

TIPOS\_USUARIO = (

('C', 'Cliente'),

('F', 'Funcionario'),

('A', 'Administrador'),

)

email = models.EmailField(unique=True)

nome = models.CharField(max\_length=100)

telefone = models.CharField(max\_length=15)

tipo = models.CharField(max\_length=1, choices=TIPOS\_USUARIO)

USERNAME\_FIELD = 'email'

REQUIRED\_FIELDS = ['nome', 'telefone']

objects = UsuarioManager()

def \_\_str\_\_(self):

return self.nome

**class Cliente(Usuario):**

cpf = models.CharField(max\_length=11, unique=True)

def fazer\_reserva(self, data, hora, qtd\_pessoas, local\_pref):

# TODO: Implementar lógica de reserva

pass

def cancelar\_reserva(self, reserva\_id):

# TODO: Implementar cancelamento de reserva

pass

def editar\_reserva(self, reserva\_id, novos\_dados):

# TODO: Implementar edição de reserva

pass

**class Funcionario(Usuario):**

matricula = models.CharField(max\_length=20, unique=True)

def visualizar\_reservas(self, data):

# TODO: Implementar visualização de reservas

pass

def gerar\_relatorio(self, tipo, data\_inicio, data\_fim):

# TODO: Implementar geração de relatórios

pass

class Administrador(Usuario):

nivel\_acesso = models.IntegerField(default=1)

def gerar\_relatorios(self, tipo, parametros):

# TODO: Implementar geração de relatórios administrativos

pass

def gerenciar\_usuarios(self, acao, usuario\_id):

# TODO: Implementar gerenciamento de usuários

pass

```

**models.py (reservas)**

```python

from django.db import models

from core.models import Cliente

class Mesa(models.Model):

STATUS\_CHOICES = (

('D', 'Disponível'),

('O', 'Ocupada'),

('R', 'Reservada'),

('M', 'Manutenção'),

)

LOCAL\_CHOICES = (

('I', 'Interna'),

('E', 'Externa'),

('V', 'Varanda'),

)

numero = models.IntegerField(unique=True)

capacidade = models.IntegerField()

status = models.CharField(max\_length=1, choices=STATUS\_CHOICES, default='D')

local = models.CharField(max\_length=1, choices=LOCAL\_CHOICES, default='I')

def disponivel(self, data, hora):

# TODO: Implementar verificação de disponibilidade

pass

def \_\_str\_\_(self):

return f"Mesa {self.numero} ({self.get\_local\_display()})"

class Reserva(models.Model):

STATUS\_CHOICES = (

('P', 'Pendente'),

('C', 'Confirmada'),

('F', 'Finalizada'),

('X', 'Cancelada'),

)

cliente = models.ForeignKey(Cliente, on\_delete=models.CASCADE)

mesa = models.ForeignKey(Mesa, on\_delete=models.CASCADE)

data = models.DateField()

hora = models.TimeField()

status = models.CharField(max\_length=1, choices=STATUS\_CHOICES, default='P')

qtd\_pessoas = models.IntegerField()

local\_pref = models.CharField(max\_length=1, choices=Mesa.LOCAL\_CHOICES)

observacoes = models.TextField(blank=True)

data\_criacao = models.DateTimeField(auto\_now\_add=True)

data\_atualizacao = models.DateTimeField(auto\_now=True)

def criar(self):

# TODO: Implementar criação de reserva

pass

def cancelar(self):

# TODO: Implementar cancelamento de reserva

pass

def editar(self, novos\_dados):

# TODO: Implementar edição de reserva

pass

def \_\_str\_\_(self):

return f"Reserva {self.id} - {self.cliente.nome} - {self.data} {self.hora}"

```

**models.py (cardapio)**

```python

from django.db import models

class ItemCardapio(models.Model):

CATEGORIAS = (

('E', 'Entrada'),

('P', 'Prato Principal'),

('S', 'Sobremesa'),

('B', 'Bebida'),

)

nome = models.CharField(max\_length=100)

descricao = models.TextField()

preco = models.DecimalField(max\_digits=10, decimal\_places=2)

categoria = models.CharField(max\_length=1, choices=CATEGORIAS)

disponivel = models.BooleanField(default=True)

imagem = models.ImageField(upload\_to='cardapio/', blank=True, null=True)

def \_\_str\_\_(self):

return self.nome

```

**5. Planejamento de Uso dos Conceitos de POO**

### Herança

- `Cliente`, `Funcionario` e `Administrador` herdam de `Usuario`

- `Usuario` herda de `AbstractBaseUser` do Django

### Polimorfismo

- Métodos como `save()` serão sobrescritos em várias classes para adicionar validações específicas

- Método `\_\_str\_\_()` será implementado em todas as classes de modelo para representação textual consistente

### Classes abstratas

- `Usuario` é uma classe base abstrata para os diferentes tipos de usuários do sistema

### Encapsulamento

- Uso de modificadores de acesso (atributos privados com métodos públicos para acesso)

- Validações internas nos métodos setters

**6. Aplicação dos Padrões de Projeto**

### Singleton para Gerenciamento de Configurações

```python

**# core/config.py**

class ConfiguracaoRestaurante(metaclass=Singleton):

def \_\_init\_\_(self):

self.horario\_abertura = time(11, 0) # 11:00 AM

self.horario\_fechamento = time(23, 0) # 11:00 PM

self.tempo\_maximo\_reserva = 2 # horas

self.antecedencia\_minima\_cancelamento = 2 # horas

```

### Factory Method para Criação de Relatórios

```python

# relatorios/factory.py

class RelatorioFactory:

@staticmethod

def criar\_relatorio(tipo, data\_inicio, data\_fim):

if tipo == 'ocupacao':

return RelatorioOcupacao(data\_inicio, data\_fim)

elif tipo == 'reservas':

return RelatorioReservas(data\_inicio, data\_fim)

elif tipo == 'clientes':

return RelatorioClientes(data\_inicio, data\_fim)

else:

raise ValueError("Tipo de relatório não suportado")

```

Strategy para Processamento de Reservas

```python

**reservas/strategies.py**

class ReservaStrategy(ABC):

@abstractmethod

def processar(self, reserva):

pass

class ReservaNormalStrategy(ReservaStrategy):

def processar(self, reserva):

# Processamento padrão para reservas normais

pass

class ReservaGrupoStrategy(ReservaStrategy):

def processar(self, reserva):

# Processamento especial para reservas de grupo

pass

```

**7. Persistência e Tratamento de Exceções**

**exceptions.py**

python

class ReservaException(Exception):

Exceção base para erros relacionados a reservas

pass

class ReservaIndisponivelException(ReservaException):

Exceção lançada quando não há mesas disponíveis

pass

class ReservaInvalidaException(ReservaException):

Exceção lançada quando os dados da reserva são inválidos

pass

class HorarioInvalidoException(ReservaException):

Exceção lançada quando o horário está fora do funcionamento

pass

class UsuarioException(Exception):

Exceção base para erros relacionados a usuários

pass

class EmailJaCadastradoException(UsuarioException):

Exceção lançada quando o email já está cadastrado

pass

```

**### repository.py (Padrão Repository)**

```python

# reservas/repository.py

class ReservaRepository:

def \_\_init\_\_(self):

self.model = Reserva

def obter\_por\_id(self, reserva\_id):

try:

return self.model.objects.get(id=reserva\_id)

except self.model.DoesNotExist:

raise ReservaNaoEncontradaException(f"Reserva {reserva\_id} não encontrada")

def obter\_por\_cliente(self, cliente\_id):

return self.model.objects.filter(cliente\_id=cliente\_id)

def obter\_por\_data(self, data):

return self.model.objects.filter(data=data)

def salvar(self, reserva):

try:

reserva.save()

return reserva

except Exception as e:

raise ReservaException(f"Erro ao salvar reserva: {str(e)}")

def excluir(self, reserva\_id):

try:

reserva = self.obter\_por\_id(reserva\_id)

reserva.delete()

except Exception as e:

raise ReservaException(f"Erro ao excluir reserva: {str(e)}")

```

**8. Critérios de Sucesso**

1. Estrutura inicial do código reflete corretamente a arquitetura MVC definida

2. UML de classes detalhado e consistente com os requisitos do documento

3. Dois diagramas de interação (sequência e atividade) para casos de uso principais

4. Classes esqueleto organizadas em pacotes conforme arquitetura planejada

5. Uso de conceitos de POO planejado com comentários (herança, polimorfismo, encapsulamento)

6. Evidência da aplicação de padrões de projeto (Singleton, Factory Method, Strategy)