Recomendação de partidas

Grupo 3

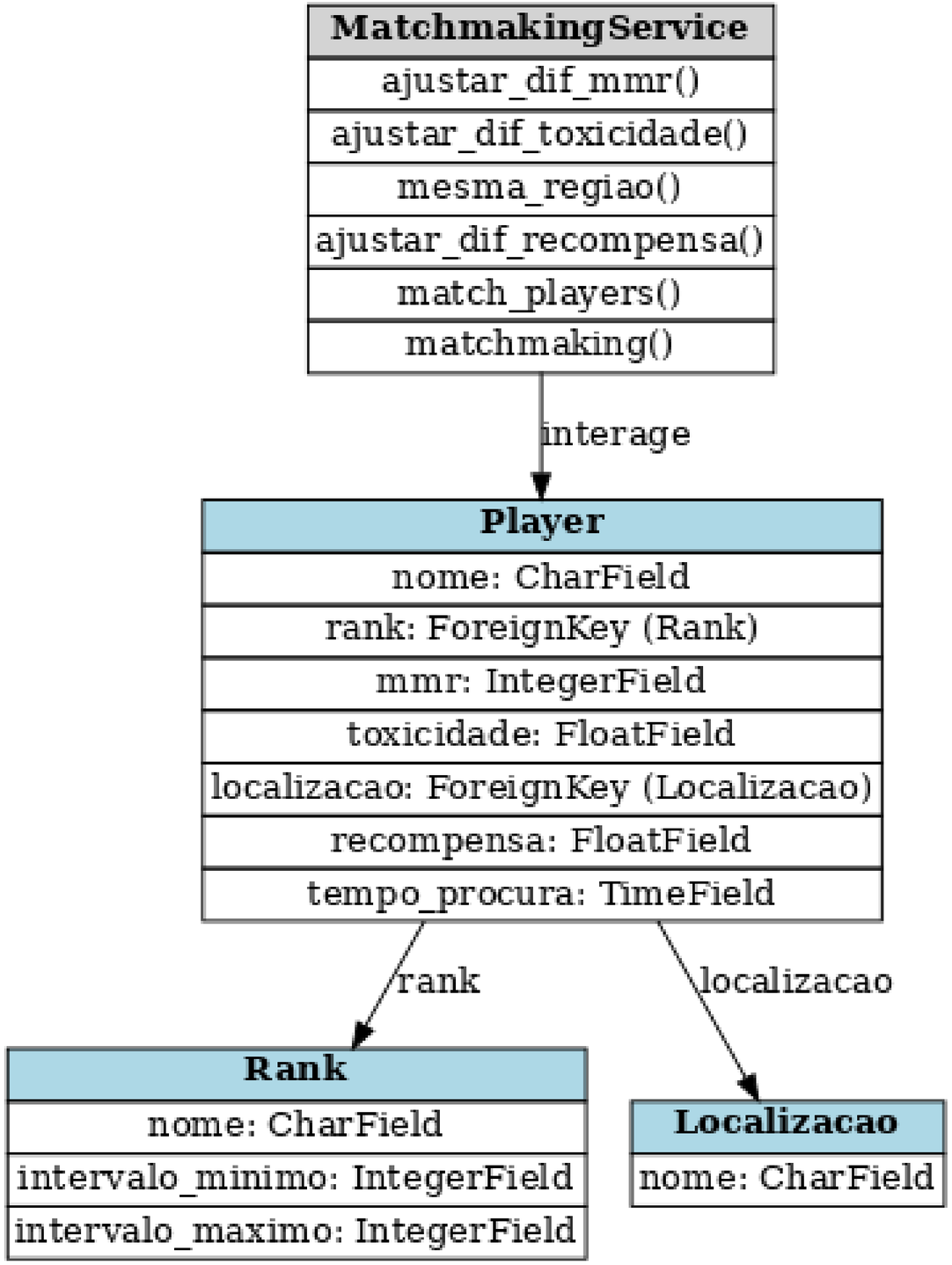


Diagrama de classes

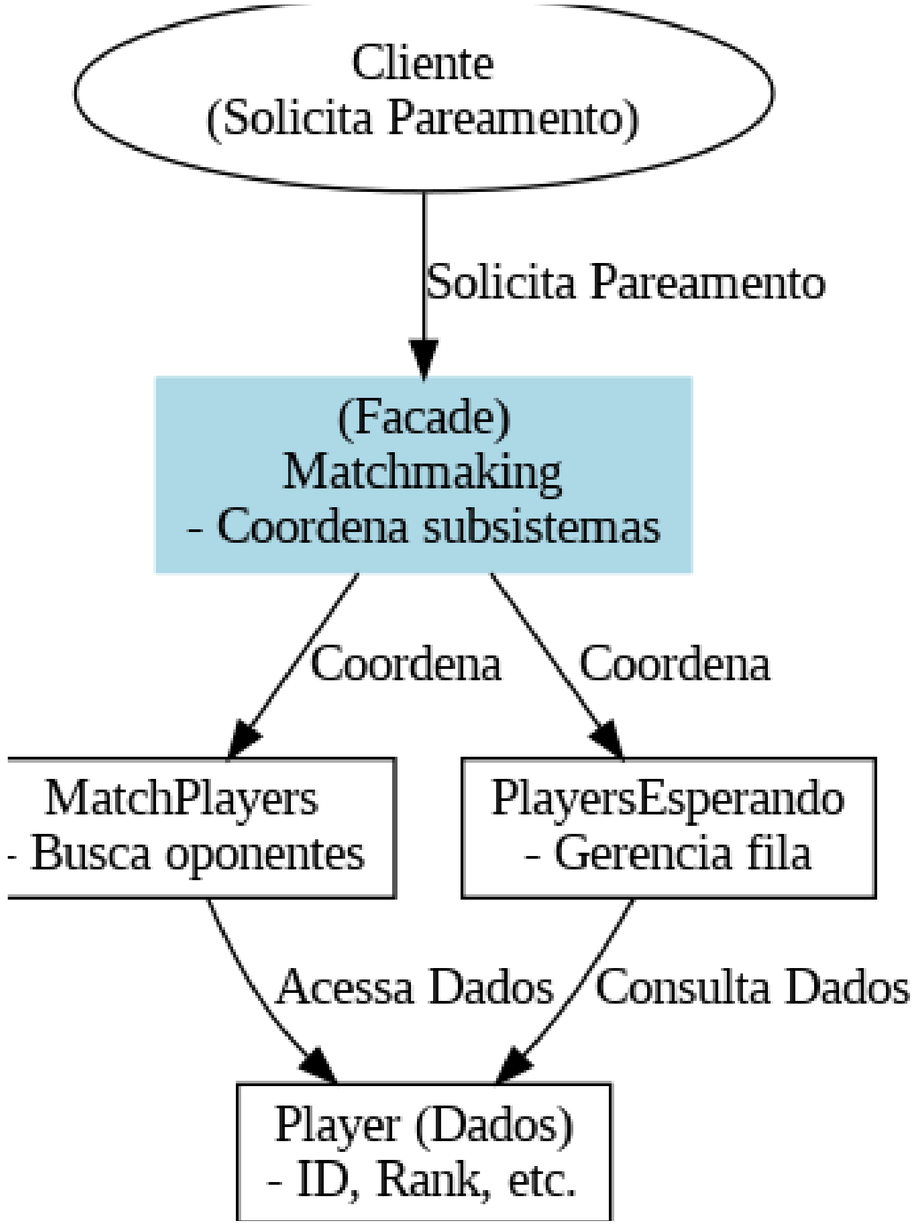
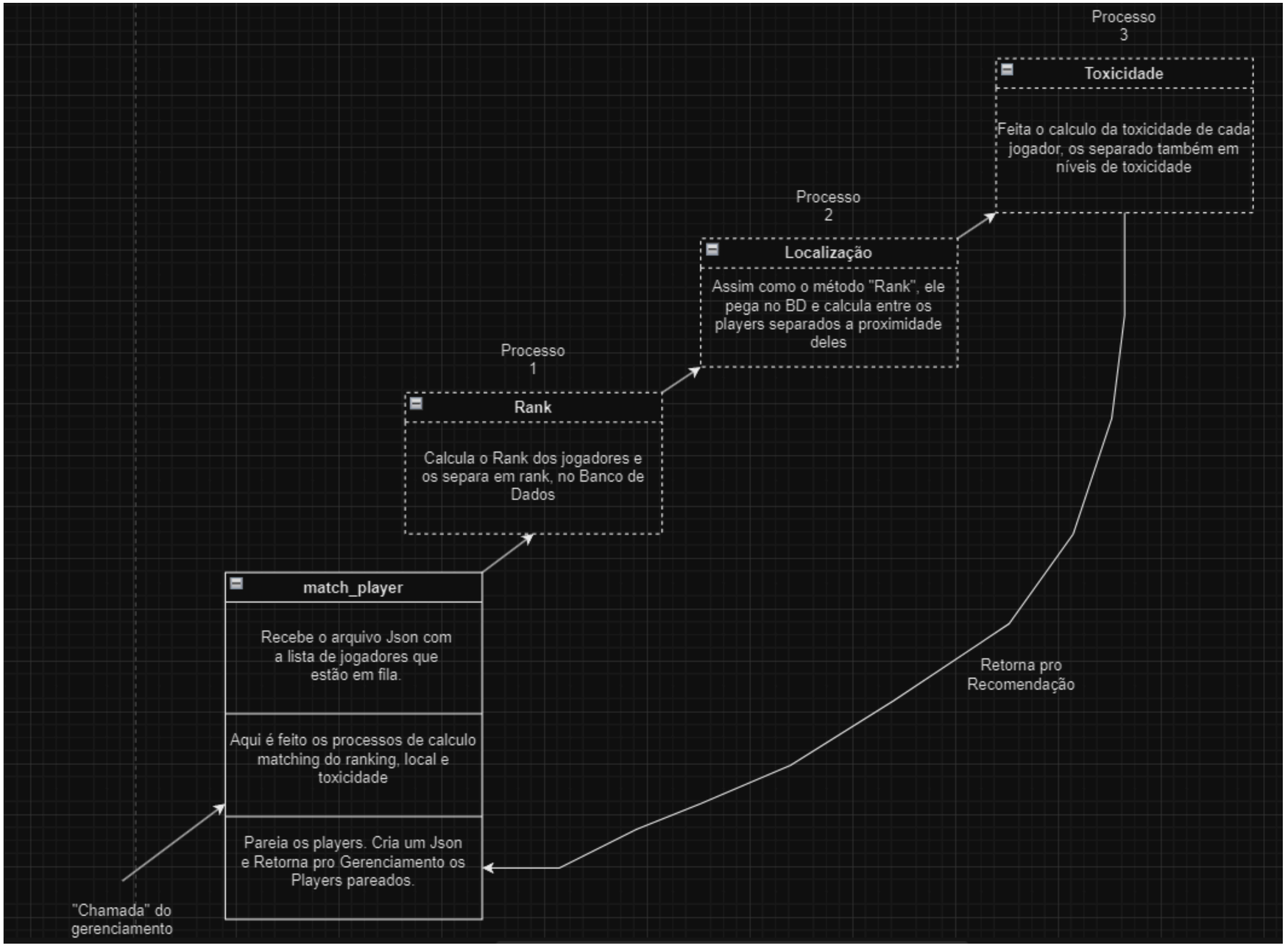


Diagrama estrutural

Facade



Behavioral

Chain of Responsibility

Creational Pattern

```python
from django.db import models
from django.utils.timezone import now
from datetime import datetime, timedelta

class Rank(models.Model):
    nome = models.CharField(max_length=20)
    intervalo_minimo = models.IntegerField()
    intervalo_maximo = models.IntegerField()

    def __str__(self):
        return f"{self.nome} ({self.intervalo_minimo}-{self.intervalo_maximo})"


class Localizacao(models.Model):
    nome = models.CharField(max_length=20)

    def __str__(self):
        return self.nome


class Player(models.Model):
    nome = models.CharField(max_length=15, default="Usuario")
    rank = models.ForeignKey(Rank, on_delete=models.CASCADE)
    mmr = models.IntegerField(default=0, verbose_name="MMR")  # Valor do rank do player
    toxicidade = models.FloatField(default=10.0, verbose_name="Toxicidade")  # Quanto mais baixo, mais tóxico
    localizacao = models.ForeignKey(Localizacao, on_delete=models.CASCADE)
    recompensa = models.FloatField(default=0.0, verbose_name="Recompensa")  # Recompensa extra ao vencer
    tempo_procura = models.TimeField(verbose_name="Tempo de Procura")  # Tempo em que começou a buscar partida

    def __str__(self):
        return f"{self.nome} (ID: {self.id}, Rank: {self.rank}, MMR: {self.mmr})"
```

```python
def tempo_espera(self) -> float:
    """
    Calcula o tempo de espera do jogador em minutos.
    """
    now_time = now().time()
    tempo_inicio = datetime.combine(datetime.today(), self.tempo_procura)
    tempo_atual = datetime.combine(datetime.today(), now_time)

    if tempo_atual < tempo_inicio:
        tempo_atual += timedelta(days=1)  # Ajuste para jogadores que passaram da meia-noite

    waiting_time = tempo_atual - tempo_inicio
    return waiting_time.total_seconds() / 60  # Retorna o tempo em minutos
```

```python
# Função para ajustar a diferença de mmr, considerando o tempo de espera
def ajustar_dif_mmr(jogador):
    return 100 + (jogador.tempoEspera() * 5)

# Função para ajustar a diferença de toxicidade, considerando o tempo de espera
def ajustar_dif_toxicidade(jogador):
    return 1.0 + (jogador.tempoEspera() * 0.1)

# Função para verificar se os jogadores estão na mesma região
def mesma_regiao(jogador, outro_jogador):
    if jogador.tempoEspera() < 10:
        return jogador.localizacao == outro_jogador.localizacao
    return True  # Ignora a localização após 10 minutos

def ajustar_dif_recompensa(jogador, outro_jogador):
    dif_recompensa = abs(jogador.mmr - outro_jogador.mmr) / 10
    if jogador.mmr < outro_jogador.mmr:
        jogador.recompensa = max(jogador.recompensa - dif_recompensa, 0)  # Evitar valores negativos
        outro_jogador.recompensa = min(outro_jogador.recompensa + dif_recompensa, 100)  # Limitar max
    else:
        jogador.recompensa = min(jogador.recompensa + dif_recompensa, 100)
        outro_jogador.recompensa = max(outro_jogador.recompensa - dif_recompensa, 0)


def match_players(players):
    prontos = []
    procurando = list(players)
    atual_jogador = procurando.pop(0)
    oponente = None

    dif_mmr = ajustar_dif_mmr(atual_jogador)
    dif_toxicidade = ajustar_dif_toxicidade(atual_jogador)

    for other_player in procurando:
        if (
            abs(atual_jogador.mmr - other_player.mmr) <= dif_mmr and
            abs(atual_jogador.toxicidade - other_player.toxicidade) <= dif_toxicidade and
            mesma_regiao(atual_jogador, other_player)
        ):

            # Found a suitable match
            oponente = other_player
            ajustar_dif_recompensa(atual_jogador, oponente)
            break

    if oponente:
        prontos.append((atual_jogador, oponente))
        procurando.remove(oponente)
    else:
        procurando.append(atual_jogador)  # Requeue procurando player for next iteration

    return prontos

def matchmaking():
    waiting_players = player.objects.all()

    if not waiting_players:
        print("sem jogadores procurando partida")
        return []

    # Sort by queue time (longest waiting players first)
    waiting_players = sorted(
        waiting_players,
        key=lambda p: p.tempoEspera(),
        reverse=True,
    )

    return match_players(waiting_players)
```