# ROTEIRO - IMPLEMENTAR AS ENTIDADES DESAFIOS E PARTIDAS

Neste roteiro apresento as atividades necessárias para que **você possa realizar** a **implementação** das **entidades desafios e partidas**.
A intenção é que este material sirva como um guia, de modo que você possa conduzir essas atividades, antes de assistir a próxima aula, onde compartilho o resultado da minha implementação.

***Importante:*** *Ao assistir a aula, onde apresento o resultado de minha implementação, você irá notar que chegamos à conclusão de que a* ***entidade rankings****, deverá ser* ***implementada*** *em um* ***momento posterior****.*

1- Começando pela **entidade Desafios**
   a. Modularize sua aplicação
      i. **Crie** o **módulo Desafios** (Module, Controller e Service). Neste momento crie apenas a estrutura padrão gerada pelo Nest
   b. **Lembre-se** de criar sua **interface** e seu respectivo **schema**:
      i. **desafio.interface.ts** - É importante notar que partida é um atributo de um Desafio

```typescript
import { Document } from 'mongoose';
import { Jogador } from 'src/jogadores/interfaces/jogador.interface';

import { DesafioStatus } from './desafio-status.enum'

export interface Desafio extends Document {

    dataHoraDesafio: Date
    status: DesafioStatus
    dataHoraSolicitacao: Date
    dataHoraResposta: Date
    solicitante: Jogador
    categoria: string
    jogadores: Array<Jogador>
    partida: Partida
}

export interface Partida extends Document{
    categoria: string
    jogadores: Array<Jogador>
    def: Jogador
    resultado: Array<Resultado>
}

export interface Resultado {
    set: string
}
```

      ii. **desafio.schema.ts**

```typescript
import * as mongoose from 'mongoose';

export const DesafioSchema = new mongoose.Schema({
    dataHoraDesafio: { type: Date },
    status: { type: String },
    dataHoraSolicitacao: { type: Date },
    dataHoraResposta: { type: Date },
    solicitante: {type: mongoose.Schema.Types.ObjectId, ref: "Jogador"},
    categoria: {type: String },
    jogadores: [{
        type: mongoose.Schema.Types.ObjectId,
        ref: "Jogador"
    }],
    partida: {
        type: mongoose.Schema.Types.ObjectId,
        ref: "Partida"
    },
}, {timestamps: true, collection: 'desafios' })

```

c. Em seu **Desafio Module** lembre-se de registrar o **DesafioSchema** no **MongooseModule**
d. Ao **criar um novo desafio**, lembre-se:
    i. Durante a **implementação** do seu **Controller**:
        1. Comece criando seu Dto **criar-desafio.dto.ts** e não se esqueça das validações:

```
import { IsNotEmpty, IsDate, IsArray, ArrayMinSize, ArrayMaxSize, IsDateString } from 'class-validator';
import { Jogador } from 'src/jogadores/interfaces/jogador.interface';

export class CriarDesafioDto {
  @IsNotEmpty()
  @IsDateString()
  dataHoraDesafio: Date;

  @IsNotEmpty()
  solicitante: Jogador;

  @IsArray()
  @ArrayMinSize(2)
  @ArrayMaxSize(2)
  jogadores: Array<Jogador>


}
```

        2. Implemente o **handle criarDesafio** - Neste método você receberá o Dto criar-desafio e o encaminhará para seu Provider
        3. Lembre-se do **ValidationPipe**
    ii. Durante a **implementação** do seu **Provider**:
        1. Lembre-se de **realizar as validações**:
            a. Os jogadores informados na requisição realmente estão cadastrados em nossa base? **Dica:** Pegue uma instância de **JogadoresService**
            b. O solicitante é um dos jogadores que fazem parte da parte do desafio?
            c. O solicitante do desafio está registrado em alguma Categoria? Lembre-se de que neste momento, conforme o Dto apresentado, nós não receberemos o id da categoria na requisição disparada pelo cliente. Deste modo, você deverá descobrir a categoria do jogador do lado do backend. **Dica:** Pegue uma instância de **CategoriasService**
        2. Antes de persistir no banco, lembre-se de incrementar a mensagem recebida com:
            a. A Categoria do jogador;
            b. A data/hora da solicitação do desafio;
            c. Com valor PENDENTE para o status do desafio. Podemos utilizar um ENUM para definir os valores possíveis para os status de um desafio:

**desafio-status.enum.ts** 164 Bytes

```
export enum DesafioStatus {
    REALIZADO = 'REALIZADO',
    PENDENTE = 'PENDENTE',
    ACEITO = 'ACEITO',
    NEGADO = 'NEGADO',
    CANCELADO = 'CANCELADO'
  }

```

e. Ao **consultar os desafios** cadastrados, lembre-se:
    i. Durante a **implementação** do seu **Controller**:

1. Implemente o **handle consultarDesafios** – Lembre-se de que este método deverá ser capaz de acionar dois diferentes métodos presentes no seu provider: **consultarDesafiosDeUmJogador** e **consultarTodosDesafios**. **Dica:** Implemente uma condição baseada no recebimento ou não do query parameter idJogador.

ii. Durante a **implementação** do seu **Provider DesafiosService**:
   1. Realize a implementação dos métodos mencionados no tópico anterior e retorne os dados, conforme solicitação do cliente.
      [Desafio] Para implementar o método consultarDesafiosDeUmJogador, será necessário descobrir como construir uma query de acesso ao mongodb, que aplique um filtro baseado no idJogador, que foi recebido na requisição.

iii. [Desafio] Ao consultar um desafio, você irá notar que a data/hora da solicitação do desafio (que você acabou de persistir com uma implementação adicional em seu backend) vai estar baseada no uso do timezone UTC, ou seja, 3 horas avançado. Tente resolver esta situação!

f. Ao **atualizar um desafio,** lembre-se:
   i. Somente a data/hora e o status de um desafio podem ser atualizados (atenção com o seu Dto)
   ii. Receba o id do desafio como um parâmetro presente em sua URL
   iii. Durante a **implementação** do seu **Controller**:
      1. Implemente o **handle atualizarDesafio** - Neste método você receberá o Dto **atualizar-desafio.dto.ts** e o encaminhará para o Provider.
      2. [Desafio] Durante a atualização de um desafio, somente devem ser aceitos os status ACEITO, NEGADO e CANCELADO. Como você implementaria esta validação, que deverá ser aplicada exclusivamente no **handle atualizarDesafio**?
   iv. Durante a implementação do **Provider DesafiosService**:
      1. Receba o Dto e o id do desafio em seu método **atualizarDesafio** e promova a atualização do desafio no banco de dados.
         **Importante**: Antes de promover a atualização, verifique se realmente o desafio está cadastrado.

g. Ao **deletar um desafio**, considere:
   i. Em seu **Controller** receber o id do desafio como um parâmetro de sua URL
   ii. Em seu **Provider** realizar uma deleção lógica do desafio, modificando seu status para CANCELADO.

2- Agora vamos para a **entidade Partidas**
Conforme mencionado anteriormente neste documento, uma **partida** deverá ser **atribuída** a um **desafio**, porém deverá possuir sua **própria Collection** no mongodb.
Deste modo, você deverá considerar a necessidade de criar um **schema específico** para **partida**.
Lembre-se de registrar este novo schema no Mongoose Module.

**partida.schema.ts** 394 Bytes

```
import * as mongoose from 'mongoose';

export const PartidaSchema = new mongoose.Schema({
    categoria: {type: String},
    jogadores: [{
        type: mongoose.Schema.Types.ObjectId,
        ref: "Jogador"
    }],
    def: { type: mongoose.Schema.Types.ObjectId, ref: "Jogador" },
    resultado: [
        { set: {type: String} }
    ]

}, {timestamps: true, collection: 'partidas' })
```

Como sugestão, a atribuição de uma partida a um desafio, poderá ficar a cargo de um Dto específico, conforme proposta abaixo:

**atribuir-desafio-partida.dto.ts** 305 Bytes

```
import { IsNotEmpty } from 'class-validator';
import { Resultado } from '../interfaces/desafio.interface';
import { Jogador } from '../../jogadores/interfaces/jogador.interface'


export class AtribuirDesafioPartidaDto {

  @IsNotEmpty()
  def: Jogador

  @IsNotEmpty()
  resultado: Array<Resultado>

}
```

Considere a possibilidade de implementar um **handle** específico em seu **DesafiosController,** que será responsável por realizar essa atribuição.
Neste handle lembre-se de receber o id do desafio como parâmetro em sua URL.
Encaminhe o Dto e o id do desafio para o **Provider** (DesafiosService) e considere a realização de algumas atividades importantes:

1- Não se esqueça de **pegar** uma **instância** de **partidaModel**
2- Validar se o desafio recebido realmente está cadastrado
3- Validar se o jogador vencedor, recebido em seu Dto, realmente faz parte do desafio
4- Baseado nas definições da interface, complemente a partida com as informações necessárias
5- Persistir a partida em sua Collection específica
6- Atualizar o status do desafio para REALIZADO
7- Atribuir a partida ao desafio
8- E por fim, persistir o desafio atualizado.

[Desafio] Em um cenário hipotético, onde conseguimos persistir a partida e não conseguimos persistir o desafio atualizado, perderemos a referência que relacionam estas duas entidades. Como você resolveria esta situação?

**3-** Em relação à entidade **Rankings**, deixaremos para um **momento posterior**. **Não deixe de acompanhar as próximas aulas!!! Vem muita coisa por aí!!!**