Projeto de Programação Orientada a Objetos.

Observação inicial: Devido a fins de incapacidade técnica, e falta de um computador funcional com ambiente Java o projeto não será entregue de forma funcional, mas ainda de modo a buscar chegar o mais próximo possível do aprendizado proposto.

SAOO - Sistema de Auxílio em Orientação Orquestral

Alunos:

Jean Gabriel da Fonseca
RA: 202109408

Descrição do projeto:

- O SAOO (Sistema de Auxílio em Orientação Orquestral) é a idealização de um software capaz de informatizar a prática de orientação orquestral, reunindo conceitos de Programação Orientada a Objetos, Banco de Dados e REST API. Atualmente, determinada instituição religiosa tem como característica o fato de estar relacionada a maior orquestra do mundo, com cerca de 600 mil músicos no Brasil. Até alguns anos atrás, a instituição se relacionava a música unicamente com propósito sacro, tendo a prática musical como uma das suas principais características. Porém, no ano de 2017 determinada instituição adotou a prática de orientação orquestral, através da publicação de um documento oficial que esquematizou e sistematizou o modo como a orquestra é agrupada e dividida entre diferentes localizações. Atualmente, a prática da orientação orquestral (nesse projeto, o termo se refere a prática de organização sistematizada de um modelo pré definido de orquestra) é, de forma simplificada, explicada no modelo a seguir:

- Existe, em média, uma igreja por bairro. Cada igreja possuirá sua própria orquestra e maestro oficial (apesar de músicos de outra localização também poderem participar ativamente do ato musical), a quantidade de músicos máxima de uma igreja local é definida por seu tamanho (número de pessoas que possam estar simultaneamente no ambiente), e a orquestra possuirá uma organização de instrumentos baseada em percentual (por exemplo, 40% dos instrumentos da orquestra deverão pertencer ao grupo de cordas). O percentual é definido tanto para os grupos musicais (cordas, madeiras, metais etc.) quanto para instrumentos específicos (violino, viola, flauta, oboé etc.). O propósito desse modelo é sanar as necessidades legislativas referentes a emissão sonora, e também melhorar características relacionadas à teoria musical, como harmonia, timbre, ritmo, afinação etc.

Desse modo, o sistema se propõe a ser possibilidade de informatizar e automatizar o máximo possível desta boa prática, de modo a economizar tempo dos encarregados orquestrais e também atingir resultados mais concisos e otimizados.

O sistema deve, para atingir tal propósito, possuir uma base de dados que permita cadastrar músicos e suas informações, assim como as localizações das igrejas e o número de músicos máximo em cada igreja. O sistema deve então realizar atividades de automação para relacionar e otimizar a orientação orquestral, alocando os músicos da melhor forma possível, mantendo a proporção de instrumentos e, preferivelmente, mantendo os músicos em suas localizações originais. O sistema deve dar sugestões dos instrumentos mais apropriados para a entrada na orquestra. Esses resultados serão atingidos através da manipulação de classes, objetos, e métodos matemáticos.

TECNOLOGIA EMPREGADA:

Para fins demonstrativos, as possibilidade de desenvolvimento será descrita a seguir com o auxílio de inteligência artificial (CHAT GPT - 5)

TECNOLOGIA EMPREGADA:

Para fins demonstrativos, as possibilidade de desenvolvimento será descrita a seguir com o auxílio de inteligência artificial (CHAT GPT - 5)

Claro, aqui está um texto descritivo sobre as vantagens de utilizar Java e o framework Spring Boot para desenvolver uma API para Orientação Orquestral, juntamente com o banco de dados MongoDB:

Vantagens de usar Java e Spring Boot para desenvolver uma API para Orientação Orquestral

Java, como uma linguagem de programação amplamente utilizada, oferece uma variedade de benefícios para o desenvolvimento de aplicações externas para a Orientação Orquestral. Sua robustez, portabilidade e ampla comunidade de desenvolvimento tornam-no uma escolha sólida para garantir a confiabilidade e a escalabilidade permitida em um projeto de alta complexidade como este.

Além disso, o Spring Boot, um framework baseado em Java, oferece uma estrutura simplificada e eficiente para criar APIs RESTful. Sua configuração mínima e o foco em convenções sobre configurações permitem um desenvolvimento mais ágil, suavizando a complexidade do código e acelerando o processo de implementação.

No contexto da Orientação Orquestral, onde a representação e gerenciamento de informações musicais são fundamentais, a utilização do banco de dados MongoDB pode ser vantajosa. Este banco de dados NoSQL, baseado em documentos, se alinha bem com os princípios da Programação Orientada a Objetos (POO). Sua flexibilidade de esquema e capacidade de armazenar dados complexos de forma natural e escalável se adequam às necessidades de representação de informações musicais, como instrumentos, músicos, partituras e estruturas da orquestra.

O uso conjunto de Java, Spring Boot e MongoDB oferece um ambiente de desenvolvimento poderoso para a construção de uma API específica para a Orientação Orquestral. Isso fornece não apenas uma base sólida para a manipulação e gerenciamento de dados musicais, mas também uma arquitetura flexível e eficiente para atender às demandas de um projeto baseado em Programação Orientada a Objetos.

ARQUITETURA DO SISTEMA:

Inicialmente, segue de forma descritiva o passo a passo de configuração de conexão entre o Spring Boot e o banco de dados Mongo DB

Primeiramente, você precisará adicionar algumas dependências e configurar o arquivo `application.properties` (ou `application.yml`) com os detalhes de conexão.

1. **Adicionar Dependências:**

No arquivo `pom.xml`, se estiver utilizando o Maven, adicione as dependências necessárias para o Spring Data MongoDB:

```xml
<dependencies>
    <!-- ... outras dependências ... -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-mongodb</artifactId>
    </dependency>
</dependencies>
```

2. **Configurar o arquivo `application.properties`:**

No arquivo `application.properties` (ou `application.yml`), defina as propriedades de conexão com o MongoDB:

application.properties:

```properties
spring.data.mongodb.host=localhost
spring.data.mongodb.port=27017
spring.data.mongodb.database=nomedoseubanco
```

Ou, utilizando o formato `application.yml`:

```yaml
spring:
```

```
  data:
    mongodb:
      host: localhost
      port: 27017
      database: nomedoseubanco
```

Substitua `nomedoseubanco` pelo nome do banco de dados que deseja utilizar.

3. **Configuração da Classe Principal do Spring Boot:**

Em uma classe que possui a anotação `@SpringBootApplication`, adicione a anotação `@EnableMongoRepositories` para habilitar os repositórios do Spring Data MongoDB:

```java
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.data.mongodb.repository.config.EnableMongoRepositories;

@SpringBootApplication
@EnableMongoRepositories(basePackages = "seu.pacote.repository")
public class SeuProjetoApplication {

    public static void main(String[] args) {
        SpringApplication.run(SeuProjetoApplication.class, args);
    }
}
```

Isso é tudo o que você precisa para configurar a conexão entre o Spring Boot e o MongoDB. Certifique-se de ter o MongoDB instalado e em execução localmente na porta padrão (27017) ou ajuste as configurações de host e porta de acordo com o seu ambiente.

Depois de configurar a conexão, você poderá criar repositórios para suas entidades e o Spring Boot lidará com a persistência de dados no MongoDB.

Então, será definido com a ajuda do Chat GPT 5 as classes inicialmente utilizadas. Sendo elas:

Músico:

```
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

import java.util.List;
```

```
@Document
public class Musico {
    @Id
    private String id;
    private String nome;
    private List<String> instrumentos; // Lista de IDs dos instrumentos que o músico toca
    // Outros atributos relevantes

    // Getters e Setters
}
```

Igreja:

```
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

@Document
public class Igreja {
    @Id
    private String id;
    private String nome;
    private int capacidadeMaxima;
    private String maestro;
    // Outros atributos relevantes

    // Getters e Setters
}
```

Orquestra:

```
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

import java.util.Map;

@Document
public class Orquestra {
    @Id
    private String id;
    private String idIgreja;
        private Map<String, Double> proporcaoInstrumentos; // Mapa para representar a
proporção de instrumentos
    private int quantidadeMusicos;
    // Outros atributos relevantes
```

```
    // Getters e Setters
}
```

Nesse exemplo, as classes representam as entidades Musico, Igreja e Orquestra.Como é possível vizualisar @Document indicam ao Spring Boot que essas classes devem ser mapeadas para coleções no MongoDB.

Agora, demonstrando de forma mais. completa:

Claro, vou criar um exemplo funcional com operações CRUD para as entidades `Musico`, `Igreja` e `Orquestra`, incluindo as funcionalidades de alocação de músicos, cálculo da proporção de instrumentos e sugestão de instrumentos.

Vou criar uma estrutura básica que você pode expandir conforme necessário. Vamos lá:

1. **Definição das Entidades:**

```java
// Musico.java
@Entity
public class Musico {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String nome;
    // Outros atributos

    // Getters e Setters
}

// Igreja.java
@Entity
public class Igreja {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String nome;
    private int capacidadeMaxima;
    // Outros atributos

    // Getters e Setters
}
```

```java
// Orquestra.java
@Entity
public class Orquestra {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
    private Igreja igreja;

    @OneToMany
    private List<Musico> musicos;

    // Outros atributos

    // Getters e Setters
}
```

2. **Repositórios:**

```java
// MusicoRepository.java
@Repository
public interface MusicoRepository extends JpaRepository<Musico, Long> {
    // Outros métodos, se necessário
}

// IgrejaRepository.java
@Repository
public interface IgrejaRepository extends JpaRepository<Igreja, Long> {
    // Outros métodos, se necessário
}

// OrquestraRepository.java
@Repository
public interface OrquestraRepository extends JpaRepository<Orquestra, Long> {
    // Outros métodos, se necessário
}
```

3. **Serviços:**

```java
// MusicoService.java
@Service
public class MusicoService {
    private final MusicoRepository musicoRepository;
```

```
    // Injeção de dependência no construtor

    // Métodos CRUD e lógica adicional para o serviço de músicos
}

// IgrejaService.java
@Service
public class IgrejaService {
    private final IgrejaRepository igrejaRepository;

    // Injeção de dependência no construtor

    // Métodos CRUD e lógica adicional para o serviço de igrejas
}

// OrquestraService.java
@Service
public class OrquestraService {
    private final OrquestraRepository orquestraRepository;
    private final MusicoRepository musicoRepository;
    private final IgrejaRepository igrejaRepository;

    // Injeção de dependência no construtor

    // Métodos CRUD e lógica adicional para o serviço de orquestras
        // Incluir métodos para alocação de músicos, cálculo de proporções e sugestão de instrumentos
}
```

Essa estrutura básica utiliza o Spring Boot e o MongoDB para realizar operações CRUD nas entidades `Musico`, `Igreja` e `Orquestra`.

Descrições para casos de uso do sistema:

Descrição dos Casos de Uso:

1. **Gerenciar Músicos:**
    - **Descrição:** Permite ao maestro visualizar, adicionar, editar ou remover músicos do sistema.
   - **Atores Envolvidos:** Maestro
   - **Fluxo Principal:**
     - O maestro acessa a funcionalidade de gerenciamento de músicos.

- Ele pode visualizar a lista de músicos, adicionar novos, editar informações existentes ou remover músicos do sistema.

2. **Alocar Músicos na Orquestra:**
   - **Descrição:** Permite ao maestro alocar músicos na orquestra de acordo com as necessidades da apresentação musical.
   - **Atores Envolvidos:** Maestro
   - **Fluxo Principal:**
     - O maestro acessa a funcionalidade de alocação de músicos na orquestra.
     - Ele pode selecionar músicos disponíveis e alocá-los em seções específicas da orquestra para a apresentação.

3. **Calcular Proporção de Instrumentos:**
   - **Descrição:** Permite ao maestro calcular a proporção de instrumentos na orquestra para garantir a harmonia e a composição corretas.
   - **Atores Envolvidos:** Maestro
   - **Fluxo Principal:**
     - O maestro acessa a funcionalidade de cálculo de proporção de instrumentos.
     - Ele realiza cálculos para garantir que a distribuição de instrumentos na orquestra esteja de acordo com as diretrizes estabelecidas.

4. **Sugerir Instrumentos para Novos Músicos:**
   - **Descrição:** Permite ao maestro sugerir instrumentos a músicos novos ou existentes com base nas necessidades da orquestra.
   - **Atores Envolvidos:** Maestro
   - **Fluxo Principal:**
     - O maestro acessa a funcionalidade de sugestão de instrumentos.
     - Ele fornece sugestões aos músicos sobre os instrumentos mais apropriados para se juntarem à orquestra.

Consideração final: O autor é consciente da insuficiência técnica desse projeto, assim sendo, o intuito desse documento é unicamente demonstrar a possibilidade de desenvolvimento, baseado na idealização criativa e capacidade tecnológica. Muito obrigado!