Universidade Federal de Mato Grosso do Sul – Campus Três Lagoas
Sistemas de Informação
Disciplina: Laboratório de Programação para Web
Professores: Humberto Lidio Antonelli

Nota:

Alunos: _____ RGA: _____

Trabalho Prático: Sistema de Gerenciamento de Eventos

Imagine um grande evento acadêmico, como uma Semana de Sistemas de Informação, onde estudantes, professores e profissionais se reúnem para compartilhar conhecimento e experiências. Este evento inclui palestras sobre temas relevantes e minicursos práticos que aprofundam o aprendizado em áreas específicas. Para organizar tudo isso, é necessário um sistema robusto que permita gerenciar os eventos, suas sessões e os participantes de forma eficiente.

Sua equipe foi contratada para desenvolver o Sistema de Gerenciamento de Eventos Acadêmicos (SGEA), que será utilizado por organizadores para criar eventos, cadastrar palestras e minicursos, gerenciar inscrições e emitir certificados ao final. O sistema também permitirá que os participantes se inscrevam nos eventos e automaticamente em todas as palestras, além de selecionar os minicursos desejados, respeitando a capacidade máxima de cada sessão.

O SGEA deve ser simples de usar, mas poderoso o suficiente para lidar com eventos complexos. Ele será desenvolvido em NestJS, utilizando um banco de dados relacional (PostgreSQL) para armazenar informações sobre usuários, eventos e sessões.

Objetivo

O objetivo do trabalho é desenvolver o Sistema de Gerenciamento de Eventos Acadêmicos (SGEA), uma aplicação *backend* que permita organizar e gerenciar eventos acadêmicos de forma eficiente. O sistema deve possibilitar que organizadores criem eventos com palestras e minicursos, gerenciem sessões e inscrições, controlem capacidades máximas e emitam certificados digitais para participantes e palestrantes.

Por meio deste trabalho, os estudantes terão a oportunidade de aplicar conceitos fundamentais de desenvolvimento web, como modelagem de banco de dados relacional, implementação de APIs RESTful com NestJS e boas práticas de organização de código, enquanto constroem uma solução prática para um cenário realista.

Requisitos gerais

1. A API deve ser desenvolvida utilizando o framework NestJS.
2. O banco de dados utilizado deverá ser o PostgreSQL.
3. Deve ser utilizado o conceito de herança no banco de dados para representar diferentes tipos de usuários (organizador, participante e palestrante).
4. A API deve seguir os princípios RESTful, com:
   - *Endpoints* bem definidos e uso adequado dos métodos HTTP (GET, POST, PUT, DELETE).
   - Implementação de paginação e filtros em todos os *endpoints* de listagem utilizando parâmetros de query como `?page=1&limit=50`.
5. A documentação da API deve ser gerada automaticamente utilizando o Swagger (ou outra ferramenta similar).

- A documentação deve incluir exemplos claros para requisições e respostas, incluindo parâmetros de paginação e possíveis erros.

6. Cada grupo deve entregar o código-fonte do projeto em um repositório do GitHub, com:
   - Estrutura modular clara.
   - README.md contendo instruções para instalação e execução do projeto.

Detalhes sobre a paginação

Todos os *endpoints* que retornam listas (ex.: /users, /events, /sessions) devem suportar os seguintes parâmetros de query:

- **page**: Número da página (inicia em 1 por padrão).
- **limit**: Número máximo de itens por página (20 registros por padrão).
- **sort**: Campo pelo qual os resultados devem ser ordenados (opcional).

Exemplo:

```
GET /events?page=2&limit=10&sort=name
```

Retorna a segunda página com até 10 eventos ordenados pelo nome.

Exemplo:

```
GET /events?sort=name:desc
```

Retorna a primeira página com até 20 eventos ordenados pelo nome em ordem decrescente.

O sistema deve retornar no cabeçalho ou no corpo da resposta informações sobre a paginação, como o total de itens, o total de páginas e a página atual.

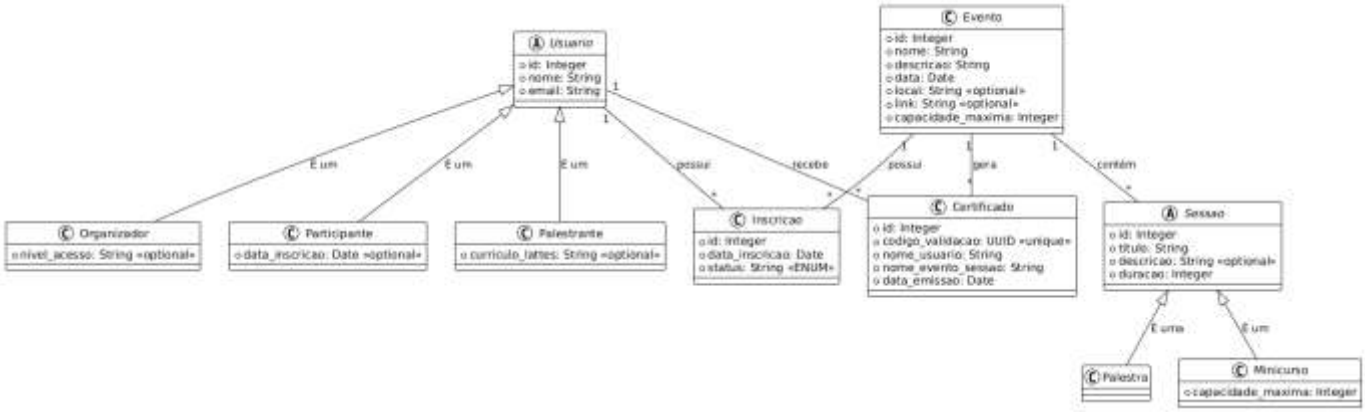
Entidades principais

1. Evento
   - Representa os eventos cadastrados no sistema.
   - Pode ser presencial ou online.
   - Deve conter informações como nome, descrição, data, local (para eventos presenciais) ou link (para eventos online), e número máximo de participantes.
2. Sessão
   - Representa as palestras, minicursos, entre outras atividades associadas a um evento.
   - Deve conter informações como título da sessão, descrição e duração em horas, se tem inscrição automática ou manual.
   - Use o conceito de herança para representar os dados específicos de cada tipo de sessão (minicursos, palestra, etc), como, por exemplo, a capacidade máxima.
3. Usuário
   - Representa uma entidade genérica para diferentes tipos de usuários.
   - Deve ser especializada em:
     - Organizador: Usuário responsável por criar e gerenciar eventos e suas sessões.
     - Participante: Usuário que pode se inscrever em eventos.
     - Palestrante: Usuário que pode ser vinculado a eventos como palestrante.
4. Inscrição
   - Representa a inscrição de um participante em um evento.

- Deve conter informações como data da inscrição e status (pendente, confirmada, cancelada).

5. Certificado
   - Representa certificados emitidos para participantes ou palestrantes após a realização do evento.
   - Deve conter informações como código de validação único, nome do participante/palestrante, nome do evento e data da emissão.

Modelo do diagrama de classe



Cronogramas e entregas

Parte 1: Estrutura inicial e CRUD básico

Nesta etapa, os grupos deverão implementar a base do sistema, incluindo a estrutura inicial do projeto, as entidades principais e as operações CRUD para eventos e usuários. O foco será na modelagem do banco de dados, organização do código e implementação dos primeiros *endpoints* RESTful.

Requisitos funcionais

1. Estrutura inicial do projeto

- Configurar o projeto NestJS com suporte ao PostgreSQL.
- Criar módulos separados para cada funcionalidade principal:
- Usuários (Organizador, Participante, Palestrante).
- Eventos.
- Sessões (Palestras, Minicursos, etc).

2. Modelagem de entidades

Os estudantes devem modelar as entidades no banco de dados com base nos atributos mínimos fornecidos abaixo:

Usuário

- *Atributos mínimos:* id, nome, email, tipo.
- Deve permitir distinguir entre organizadores, participantes e palestrantes.

Evento

- *Atributos mínimos:* id, nome, data, capacidade_maxima.
- Deve estar relacionado a um organizador que criou o evento.

Sessão

- *Atributos mínimos:* id, titulo, duracao.
- Deve estar vinculada a um evento.
- Deve permitir distinguir entre palestras, minicursos ou qualquer outro tipo sessão possível dentro de um evento

3. Operações CRUD

Os grupos devem implementar *endpoints* RESTful para realizar as seguintes operações:

Eventos

- Criar um novo evento.
- Listar todos os eventos disponíveis.
- Buscar detalhes de um evento específico.
- Atualizar informações de um evento existente.
- Excluir um evento.

Sessões

- Criar uma nova sessão vinculada a um evento.
- Listar todas as sessões de um evento específico.
- Atualizar informações de uma sessão existente.
- Excluir uma sessão específica.

Usuários

- Criar novos usuários (organizador, participante ou palestrante).
- Listar todos os usuários cadastrados no sistema.
- Buscar detalhes de um usuário específico.

4. Documentação da API

Os grupos devem documentar todos os *endpoints* criados utilizando Swagger ou outra ferramenta similar. A documentação deve incluir:

- Descrição clara da funcionalidade de cada *endpoint*.
- Exemplos de requisição e resposta para cada operação.

Endpoints obrigatórios

Método	*Endpoint*	Descrição	Principais Status HTTP
POST	`/users`	Criar um novo usuário (organizador, participante ou palestrante).	201 Created, 400 Bad Request
GET	`/users`	Listar todos os usuários cadastrados no sistema.	200 OK
GET	`/users/{id}`	Buscar detalhes de um usuário específico pelo ID.	200 OK, 404 Not Found
PUT	`/users/{id}`	Atualizar informações de um usuário específico.	200 OK, 400 Bad Request, 404 Not Found
DELETE	`/users/{id}`	Excluir um usuário específico pelo ID.	204 No Content, 404 Not Found
POST	`/events`	Criar um novo evento.	201 Created, 400 Bad Request
GET	`/events`	Listar todos os eventos disponíveis.	200 OK

Método	*Endpoint*	Descrição	Principais Status HTTP
GET	`/events/{id}`	Buscar detalhes de um evento específico pelo ID (incluindo suas sessões).	200 OK, 404 Not Found
PUT	`/events/{id}`	Atualizar informações de um evento específico.	200 OK, 400 Bad Request, 404 Not Found
DELETE	`/events/{id}`	Excluir um evento específico pelo ID.	204 No Content, 404 Not Found
POST	`/events/{id}/sessions`	Criar uma nova vinculada a um evento.	201 Created, 400 Bad Request, 404 Not Found
GET	`/events/{id}/sessions`	Listar todas as sessões associadas a um evento.	200 OK
GET	`/sessions/{id}`	Buscar detalhes de uma sessão específica pelo ID.	200 OK, 404 Not Found
PUT	`/sessions/{id}`	Atualizar informações de uma sessão específica.	200 OK, 400 Bad Request, 404 Not Found
DELETE	`/sessions/{id}`	Excluir uma sessão específica pelo ID.	204 No Content, 404 Not Found

Entrega esperada

1. Código-fonte organizado em repositório GitHub com estrutura modular clara:
   - Módulos separados para usuários, eventos e sessões.
2. Banco de dados configurado com tabelas correspondentes às entidades principais.
3. *Endpoints* RESTful funcionando conforme descrito nos requisitos funcionais.
4. Documentação completa da API acessível via Swagger.
5. Arquivo README.md contendo:
   - Instruções claras para instalação do projeto.
   - Informações sobre variáveis de ambiente necessárias.
   - Exemplos básicos de requisições HTTP.

Parte 2: Inscrições e controle de capacidades

Nesta etapa, os grupos deverão implementar a funcionalidade de inscrição de participantes em eventos, incluindo a inscrição automática a seleção manual das sessões. Além disso, será necessário gerenciar as capacidades máximas do evento e das sessões, garantindo que as regras de negócio sejam respeitadas.

Requisitos Funcionais

1. Inscrição no evento

- O participante pode se inscrever em um evento desde que haja vagas disponíveis no evento geral.
- Ao se inscrever no evento:
  - O participante é automaticamente registrado em todas as sessões com inscrições automáticas que estão associadas ao evento.
    - Nenhuma ação adicional é necessária para essas sessões.
- Sessões com inscrições automáticas que forem incluídas posteriormente, devem registrar os participantes que já se inscreveram no evento.

2. Seleção manual de sessões

- Após se inscrever no evento, o participante pode selecionar outras sessões com inscrições manuais que deseja participar, desde que:
  - Haja vagas disponíveis na sessão escolhida.
  - O participante ainda não esteja inscrito na mesma sessão.
- Cada participante pode se inscrever em múltiplas sessões dentro do mesmo evento, respeitando a capacidade máxima de cada um.

3. Controle de capacidades

- O sistema deve verificar a disponibilidade de vagas antes de permitir qualquer inscrição:
  - No evento geral (capacidade máxima do evento).
  - Em cada sessão (capacidade máxima individual).
- Caso não haja vagas disponíveis, o sistema deve retornar uma mensagem clara indicando o problema.

Endpoints obrigatórios

Método	*Endpoint*	Descrição	Principais Status HTTP
POST	`/events/{eventId}/enrollments`	Inscrever um participante em um evento (inscrição automática em palestras).	201 Created, 400 Bad Request, 404 Not Found, 409 Conflict
POST	`/enrollments/{enrollmentId}/workshops`	Adicionar sessões selecionados à inscrição do participante.	201 Created, 400 Bad Request, 404 Not Found, 409 Conflict
GET	`/events/{eventId}/participants`	Listar todos os participantes inscritos em um evento.	200 OK, 404 Not Found
GET	`/sessions/{sessionId}/participants`	Listar todos os participantes inscritos em uma sessão.	200 OK, 404 Not Found
DELETE	`/enrollments/{enrollmentId}`	Cancelar a inscrição de um participante no evento.	204 No Content, 404 Not Found

Entrega esperada

1. Código-fonte atualizado com os módulos e serviços necessários para gerenciar inscrições e sessões.
2. Banco de dados atualizado com tabelas e relacionamentos necessários para armazenar informações sobre inscrições e sessões.
3. *Endpoints* RESTful funcionando conforme descrito nos requisitos funcionais.
4. Documentação completa da API acessível via Swagger.
5. Arquivo README.md contendo:
   - Instruções claras para instalação do projeto.
   - Exemplos básicos de requisições HTTP para inscrição em eventos e seleção de sessões.

Parte 3: Funcionalidades avançadas e finalização do sistema

Nesta etapa, os grupos deverão implementar as funcionalidades avançadas do sistema, incluindo autenticação e autorização, emissão e validação de certificados digitais e relatórios administrativos para organizadores. O objetivo é consolidar o sistema como uma solução completa para o gerenciamento de eventos acadêmicos, garantindo segurança, controle e usabilidade para todos os tipos de usuários.

Requisitos Funcionais

1. Autenticação e Autorização

- Implementar autenticação baseada em JWT (JSON Web Token).
- Diferenciar permissões por tipo de usuário:
  - Organizador: Pode criar, editar e excluir eventos e sessões; acessar relatórios administrativos.
  - Participante: Pode se inscrever em eventos, selecionar sessões e acessar seus certificados.
  - Palestrante: Pode visualizar as sessões onde está vinculado e acessar seus certificados.
- **Regras de negócio:**
  - Apenas usuários autenticados podem acessar endpoints protegidos.
  - Organizador pode gerenciar apenas os eventos que criou.
  - Participantes só podem visualizar suas próprias inscrições e certificados.

2. Emissão Automática de Certificados

- Emitir automaticamente certificados para todos os participantes inscritos e palestrantes vinculados às sessões após o término do evento.
- A disponibilização dos certificados aos usuários deve ser feita retornando um documento PDF com todas as informações necessárias.
- Cada certificado deve conter:
  - Nome do participante ou palestrante.
  - Nome do evento ou sessão.
  - Data do evento/sessão.
  - Um código único de validação (ex.: UUID).
- **Regras de negócio:**
  - Certificados só podem ser emitidos para participantes inscritos no evento ou palestrantes vinculados às sessões.

3. Validação Pública de Certificados

- Criar um *endpoint* público que permita validar a autenticidade de um certificado com base em seu código único.
- O *endpoint* deve retornar:
  - Dados básicos do certificado (nome, evento/sessão, data).
  - Status do certificado (Válido, Revogado).

4. Relatórios Administrativos

- Criar um painel administrativo para organizadores com métricas detalhadas sobre os eventos que organizaram:
  - Número total de participantes inscritos no evento geral.
  - Ocupação por sessão.

- Taxa de ocupação geral (%).

Endpoints obrigatórios

Método	*Endpoint*	Descrição	Principais Status HTTP
POST	`/auth/login`	Realizar login e gerar token JWT para autenticação.	200 OK, 401 Unauthorized
GET	`/auth/me`	Retornar informações do usuário autenticado.	200 OK, 401 Unauthorized
GET	`/certificates/participant/{id}`	Listar todos os certificados emitidos para um participante.	200 OK, 404 Not Found
GET	`/certificates/speaker/{id}`	Listar todos os certificados emitidos para um palestrante.	200 OK, 404 Not Found
GET	`/certificates/validate/{code}`	Validar a autenticidade de um certificado com base em seu código único.	200 OK, 404 Not Found, 410 Gone
PATCH	`/certificates/validate/{id}/revoke`	Revogar um certificado emitido.	200 OK, 404 Not Found, 410 Gone
GET	`/admin/events/{eventId}/report`	Gerar relatório detalhado sobre um evento organizado.	200 OK, 403 Forbidden, 404 Not Found

Entrega esperada

1. Código-fonte atualizado com as funcionalidades avançadas implementadas.
2. Banco de dados atualizado com tabelas para armazenar informações sobre certificados.
3. *Endpoints* RESTful funcionando conforme descrito nos requisitos funcionais.
4. Documentação completa da API acessível via Swagger, incluindo exemplos detalhados dos novos *endpoints*.
5. Arquivo README.md contendo:
   - Instruções claras para instalação do projeto.
   - Informações sobre variáveis de ambiente necessárias (ex.: chave secreta JWT).
   - Exemplos básicos de requisições HTTP para login, emissão/validação de certificados e relatórios.

Critérios de avaliação

Critério	Descrição	Peso (%)
Etapa 1: Estrutura Inicial e CRUD Básico		
Estrutura do Projeto	Organização modular clara no NestJS; uso correto de *controllers*, *services* e repositórios.	**25%**
Modelagem do Banco de Dados	Implementação correta das tabelas no PostgreSQL; uso adequado de herança e relacionamentos.	**20%**
Operações CRUD	*Endpoints* RESTful funcionando corretamente com verbos HTTP apropriados.	**20%**
Tratamento de Erros	Uso de códigos HTTP apropriados (ex.: 400 para dados inválidos, 404 para recursos não encontrados) e mensagens de erro padronizadas (ex.: formato RFC 7807) para conflitos e validações.	**15%**

Critério	Descrição	Peso (%)
Documentação da API	Swagger completo com exemplos claros e bem estruturados para todos *endpoints* implementados.	**15%**
Qualidade Geral	Clareza no código; boas práticas aplicadas; organização geral do projeto.	**5%**
Etapa 2: Inscrições e Controle de Capacidades		
Estrutura do Projeto	Organização modular clara no NestJS; uso correto de *controllers*, *services* e repositórios.	**14%**
Modelagem do Banco de Dados	Relacionamentos Many-to-Many entre inscrições, participantes e sessões.	**19%**
Funcionalidade de Inscrição	Implementação correta da inscrição automática e manual em sessões.	**23%**
Controle de Capacidades	Verificação adequada das vagas disponíveis para eventos e sessões.	**20%**
Tratamento de Erros	Uso de códigos HTTP apropriados (ex.: 400 para dados inválidos, 404 para recursos não encontrados) e mensagens de erro padronizadas (ex.: formato RFC 7807) para conflitos e validações.	**10%**
Documentação da API	Swagger atualizado com exemplos de erros (ex.: evento lotado).	**9%**
Qualidade Geral	Clareza no código; boas práticas aplicadas; organização geral do projeto.	**5%**
Etapa 3: Funcionalidades Avançadas		
Estrutura do Projeto	Organização modular clara no NestJS; uso correto de *controllers*, *services* e repositórios.	**8%**
Modelagem do Banco de Dados	Implementação correta das tabelas no PostgreSQL; uso adequado de herança e relacionamentos.	**8%**
Autenticação e Autorização	Implementação correta com JWT; diferenciação clara por tipo de usuário.	**20%**
Emissão Automática de Certificados	Certificados gerados corretamente após o término do evento ou sessão.	**16%**
Validação Pública de Certificados	*Endpoint* de validação de certificados com respostas claras (válido/revogado).	**14%**
Relatórios Administrativos	Métricas detalhadas apresentadas corretamente para organizadores.	**12%**
Tratamento de Erros	Uso de códigos HTTP apropriados (ex.: 400 para dados inválidos, 404 para recursos não encontrados, 401 para usuários não autenticados, 403 para usuários sem permissão) e mensagens de erro padronizadas (ex.: formato RFC 7807) para autenticação, conflitos e validações.	**10%**
Documentação da API	Swagger completo com exemplos claros e bem estruturados para todos *endpoints* implementados, tanto para o caso de sucesso quanto para os erros.	**8%**
Qualidade Geral	Clareza no código; boas práticas aplicadas; organização geral do projeto.	**4%**

Observações e Recomendações:

- Consulte sempre o CONSOLE para verificar os possíveis erros no código em execução.
    - Qualquer erro implicará em pontos a menos na correção do trabalho.
- Tente pensar em todos os possíveis erros que o usuário pode cometer e EVITE que eles ocorram, como por exemplo, a validação de número e datas.
    - A parte lógica de como o sistema funciona também faz parte da avaliação!
- NÃO deixe para fazer o trabalho na última semana!
- Cada grupo deve criar um repositório privado no GitHub e adicionar o professor como colaborador até o final da primeira semana do trabalho prático.
- As entregas devem ser feitas via GitHub até o prazo estipulado no cronograma.
- Durante a revisão parcial, será avaliada a estrutura inicial do projeto, a modelagem das entidades e o funcionamento dos primeiros *endpoints*.
- **Em trabalhos que forem identificadas cópias, a será divida entre os grupos!**

Este trabalho é uma oportunidade para aplicar os conceitos aprendidos em aula e desenvolver habilidades práticas em programação para *backend*. Aproveitem para explorar a criatividade e o pensamento sistêmico, de modo a criar uma aplicação que vá além dos requisitos mínimos exigidos nesse documento.

Em caso de dúvidas, SEMPRE pergunte ao professor, seja durante as aulas, horário de atendimento ou com agendamento de horário específico.

Bom trabalho!