Gerenciamento alunos
Desenvolvimento Java

José Nilson Oliveira Arcanjo
Larissa Bonfim de Jesus

Salvador-BA
2025

Sumário

Salvador-BA
2025

1. Introdução

Nesse Projeto decidimos criar um gerenciamento de alunos para uma escola, facilitando as mesmas a ter um melhor controle da quantidade de alunos por turma e realizar seus cadastros, nessa API existem funções como cadastrar, atualizar, deletar e buscar por turma.

2. Camada Model

```java
@Entity
@Table(name = "tb_user")
public class Aluno {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(columnDefinition = "TEXT")
    private String nome;
    private Integer turma;

    public Aluno() {
    }

    public Aluno(Aluno aluno) {
        this.id = aluno.getId();
        this.nome = aluno.getNome();
        this.turma = aluno.getTurma();
    }

    public Aluno(Long id, String nome, Integer turma) {
        this.id = id;
        this.nome = nome;
        this.turma = turma;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public Integer getTurma() {
        return turma;
    }

    public void setTurma(Integer turma) {
        this.turma = turma;
    }
```

Na classe aluno é possível ver o banco de dados sendo criado através de anotations como @Table, além da definição de um id, por meio de @Id.
Nessa classe é definida os elementos do cadastro como nome e turma. Com eles definidos é feito os gets e sets.

3. Camada Service

```java
@Service
public class AlunoService {

    @Autowired
    AlunoRepository alunoRepository;

    @Transactional(readOnly = true)
    public ResponseEntity<?> findAllTurma(int turma){
        List<AlunoProjection> AlunoProjection = alunoRepository.searchAllAlunosTurma(turma);

        return ResponseEntity.status(HttpStatus.OK).body(AlunoProjection);
    }

    @Transactional(readOnly = true)
    public ResponseEntity<?> findAllAlunos(){
        List<Aluno> aluno = alunoRepository.findAll();

        return ResponseEntity.status(HttpStatus.OK).body(aluno);
    }

    @Transactional
    public ResponseEntity<?> saveAluno(Aluno aluno){
        alunoRepository.save(new Aluno(aluno));
        return ResponseEntity.status(HttpStatus.CREATED).body(new ResponseMessage("Aluno criado com sucesso."));
    }

    @Transactional
    public ResponseEntity<?> updateAluno(Long id, Aluno aluno){
        Optional<Aluno> alunoOptional = alunoRepository.findById(id);

        if (!alunoOptional.isPresent()) {
            return ResponseEntity.status(HttpStatus.NOT_FOUND).body(new ResponseMessage("Aluno não encontrado."));
        }

        Aluno result = alunoOptional.get();

        result.setNome(aluno.getNome());
        result.setTurma(aluno.getTurma());

        alunoRepository.save(result);
        return ResponseEntity.status(HttpStatus.ACCEPTED).body(new ResponseMessage("Aluno alterado com sucesso."));
    }

    @Transactional
    public ResponseEntity<?> deleteAluno(Long id){
        Optional<Aluno>  alunoOptional = alunoRepository.findById(id);

        if (!alunoOptional.isPresent()) {
            return ResponseEntity.status(HttpStatus.NOT_FOUND).body(new ResponseMessage("Aluno não encontrado."));
        }

        alunoRepository.deleteById(id);

        return ResponseEntity.status(HttpStatus.OK).body(new ResponseMessage("Aluno excluído com sucesso."));
    }
}
```

A classe AlunoService contém toda a lógica de negócio. Ela usa @Transactional para garantir consistência nas operações e acessa o banco através do repository. Nessa camada foram criadas algumas as funcionalidades como:
- findAllTurma: retorna lista de alunos filtrados pela turma.
- findAllAlunos: retorna todos os alunos.
- saveAluno: cria um novo aluno.
- updateAluno: atualiza informações de um aluno existente.
- deleteAluno: exclui aluno.

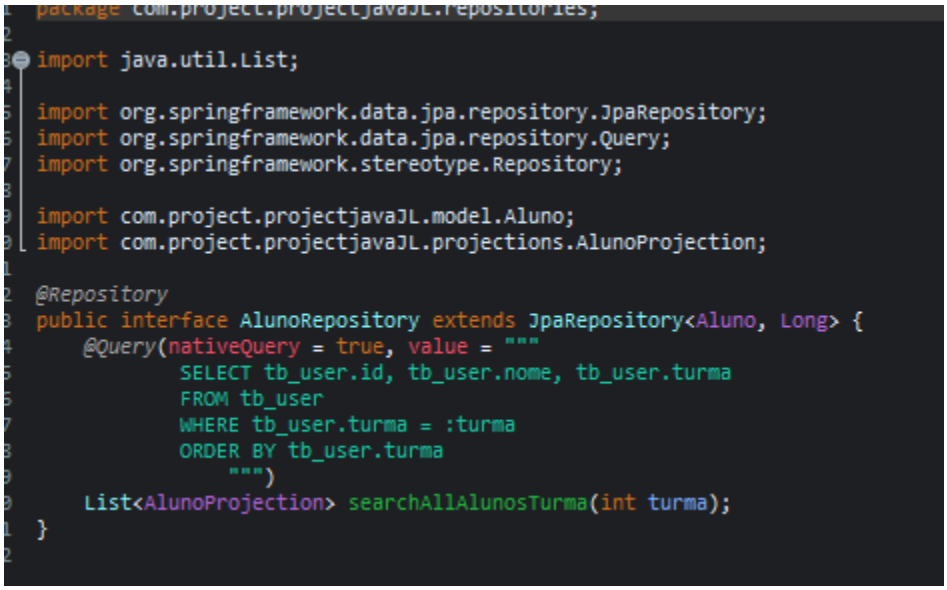
4. Camada Controller

```java
@RestController
@RequestMapping(value = "/users")
public class AlunoController {

    @Autowired
    AlunoService alunoService;

    @GetMapping
    public ResponseEntity<?> findAllAlunos(){
        return alunoService.findAllAlunos();
    }

    @GetMapping(value = "/turma/{turma}")
    public ResponseEntity<?> findAllTurma(@PathVariable int turma){
        return alunoService.findAllTurma(turma);
    }

    @PostMapping
    public ResponseEntity<?> saveAluno(@RequestBody Aluno aluno){
        return alunoService.saveAluno(aluno);
    }

    @PutMapping(value = "/{id}")
    public ResponseEntity<?> updateAluno(@PathVariable Long id, @RequestBody Aluno aluno){
        return alunoService.updateAluno(id, aluno);
    }

    @DeleteMapping(value = "/{id}")
    public ResponseEntity<?> deleteAluno(@PathVariable Long id){
        return alunoService.deleteAluno(id);
    }
}
```

A camada Controller é responsável por receber as requisições do cliente, funcionando como porta de entrada e direcionando as requisições a camada service.

Na camada controller podemos encontrar anotations ligadas ao banco de dados
- RequestMapping/users: endpoint base do controller.
- GET /turma/{turma}: lista alunos filtrados pela turma.
- PostMapping: endpoint para salvar novo aluno.
- PutMapping /{id}: endpoint para atualiza aluno.
- DeleteMapping/{id}: endpoint para exclui aluno.

5. Camada Repository



```java
package com.project.projectjavaJL.repositories;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;

import com.project.projectjavaJL.model.Aluno;
import com.project.projectjavaJL.projections.AlunoProjection;

@Repository
public interface AlunoRepository extends JpaRepository<Aluno, Long> {
    @Query(nativeQuery = true, value = """
            SELECT tb_user.id, tb_user.nome, tb_user.turma
            FROM tb_user
            WHERE tb_user.turma = :turma
            ORDER BY tb_user.turma
                """)
    List<AlunoProjection> searchAllAlunosTurma(int turma);
}
```

A camada Repository é responsável por fazer a comunicação com o banco de dados. Ela é quem realmente faz:
consultas (SELECT), inserções (INSERT), atualizações (UPDATE), exclusões (DELETE).

6. HTML

```html
<div class="container">
    <h1>Sistema de Gerenciamento de Alunos</h1>

    <div id="message-container" class="message-box" style="display: none;"></div>

    <h2>🔎 Pesquisar Alunos por Turma</h2>
    <div class="form-group" style="display: flex; gap: 10px;">
        <input type="number" id="searchTurma" placeholder="Digite o número da Turma" style="flex-grow: 1;">
        <button class="btn-info" onclick="searchAlunosByTurma()">Buscar Turma</button>
    </div>
    <div id="searchResult" style="margin-top: 10px;"></div>

    <hr>

    <h2 id="form-title">+ Cadastrar Novo Aluno</h2>
    <form id="alunoForm">
        <input type="hidden" id="alunoId">
        <div class="form-group">
            <label for="nome">Nome:</label>
            <input type="text" id="nome" required>
        </div>
        <div class="form-group">
            <label for="turma">Turma:</label>
            <input type="number" id="turma" required>
        </div>
        <button type="submit" id="submitBtn" class="btn-primary">Cadastrar</button>
        <button type="button" id="cancelEditBtn" class="btn-warning" style="display: none;" onclick="resetForm()">Cancelar Edição</button>
    </form>

    <hr>

    <h2>📚 Alunos Existentes</h2>
    <table>
        <thead>
            <tr>
                <th>ID</th>
                <th>Nome</th>
                <th>Turma</th>
                <th>Ações</th>
            </tr>
        </thead>
        <tbody id="alunosTableBody">
            </tbody>
    </table>
</div>
```

Funções para pesquisar, cadastrar e visualizar alunos existentes.

```javascript
/**
 * Exibe uma mensagem de feedback (sucesso ou erro) para o usuário.
 */
function showMessage(message, isError = false) {
    messageContainer.style.display = 'block';
    messageContainer.textContent = message;
    messageContainer.className = 'message-box ' + (isError ? 'error' : 'success');
    setTimeout(() => {
        messageContainer.style.display = 'none';
    }, 5000);
}

/**
 * Limpa e reseta o formulário para o modo de Cadastro.
 */
function resetForm() {
    form.reset();
    alunoIdInput.value = '';
    formTitle.textContent = '+ Cadastrar Novo Aluno';
    submitBtn.textContent = 'Cadastrar';
    submitBtn.className = 'btn-primary';
    cancelEditBtn.style.display = 'none';
}
```

Avisa ao usuário que o cadastro foi concluído ou não e limpa o campo para um novo cadastro.

```javascript
/**
 *Listar Todos os Alunos (GET /users)
 */
async function fetchAlunos() {
    try {
        const response = await fetch(BASE_URL);
        if (!response.ok) {
            throw new Error('Falha ao buscar alunos: ' + response.status);
        }
        const alunos = await response.json();
        renderTable(alunos);
    } catch (error) {
        console.error('Erro ao buscar alunos:', error);
        showMessage('Não foi possível carregar a lista de alunos.', true);
    }
}

/**
 *Cadastrar Novo Aluno (POST /users)
 */
async function createAluno(alunoData) {
    try {
        const response = await fetch(BASE_URL, {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify(alunoData)
        });

        const data = await response.json();
        if (!response.ok) {
            throw new Error(data.message || 'Erro ao cadastrar.');
        }

        showMessage(data.message || 'Aluno criado com sucesso!', false);
        resetForm();
        fetchAlunos(); // Atualiza a tabela
    } catch (error) {
        console.error('Erro na criação:', error);
        showMessage(error.message, true);
    }
}
```

Funções para criar e atualizar alunos.

```
/**
 *Excluir Aluno (DELETE /users/{id})
 */
async function deleteAluno(id) {
    if (!confirm(`Tem certeza que deseja excluir o aluno com ID ${id}?`)) {
        return;
    }

    try {
        const response = await fetch(`${BASE_URL}/${id}`, {
            method: 'DELETE'
        });

        const data = await response.json();
        if (!response.ok) {
            throw new Error(data.message || 'Erro ao excluir.');
        }

        showMessage(data.message || 'Aluno excluído com sucesso!', false);
        fetchAlunos(); // Atualiza a tabela
    } catch (error) {
        console.error('Erro na exclusão:', error);
        showMessage(error.message, true);
    }
}

/**
 * Pesquisar Alunos por Turma (GET /users/turma/{turma})
 */
async function searchAlunosByTurma() {
    const turma = searchTurmaInput.value;
    searchResultDiv.innerHTML = ''; // Limpa resultados anteriores

    if (!turma) {
        searchResultDiv.innerHTML = '<p class="error">Por favor, digite o número de uma turma.</p>';
        return;
    }

    try {
        // Chamada para o endpoint do Controller: GET /users/turma/{turma}
        const response = await fetch(`${BASE_URL}/turma/${turma}`);

        if (!response.ok) {
            // Se houver um erro no backend (ex: 500), mostra erro genérico.
            throw new Error(`Erro ao buscar alunos da turma: ${response.status}`);
        }

        const alunos = await response.json();

        if (alunos.length === 0) {
            searchResultDiv.innerHTML = `<p class="error">Nenhum aluno encontrado na Turma ${turma}.</p>`;
            return;
        }

        // Se encontrado, exibe os detalhes em formato de lista (ou tabela)
        let htmlContent = `<p class="success">🔍 Alunos encontrados na Turma ${turma}º:</p><ul>`;

        // Loop para listar os alunos encontrados
        alunos.forEach(aluno => {
            htmlContent += `<li><strong>ID:</strong> ${aluno.id} | <strong>Nome:</strong> ${aluno.nome}</li>`;
        });

        htmlContent += '</ul>';
        searchResultDiv.innerHTML = htmlContent;

    } catch (error) {
        console.error('Erro na pesquisa por turma:', error);
        searchResultDiv.innerHTML = `<p class="error">Falha na comunicação com a API ou erro interno.</p>`;
    }
}
```

Funções para deletar e listar os alunos pela sua turma.

7. Site

Acessado por: http://localhost:8088/aluno.html