SERVIÇO PÚBLICO FEDERAL · MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE VIÇOSA · UFV
CAMPUS FLORESTAL

Design CSU02, CSU03 e CSU05

Bytecraft - 5º ano

Florestal - MG
2025

Sumário

Refatoração

Devido a uma escolha de padrão de design, o Facade, algumas alterações devem ser feitas para que o padrão do projeto seja seguido; a Figura 1 mostra o novo diagrama de classes. A seguinte classe existente foi alterada:

- **AlunoController**: essa classe de controle, e a partir de agora, qualquer classe de controle, não terá como atributo uma classe do pacote **repository**. Portanto, esse atributo deve ser retirado e, nesse caso, será realocado para a classe **AlunoService**, que deve servir como uma classe Facade e deve reunir toda a lógica relacionada à entidade **Aluno** dentro de si. Dessa maneira, **AlunoController** e outras classes de controle devem utilizar apenas métodos das classes do pacote **service** para interagir com o banco de dados.

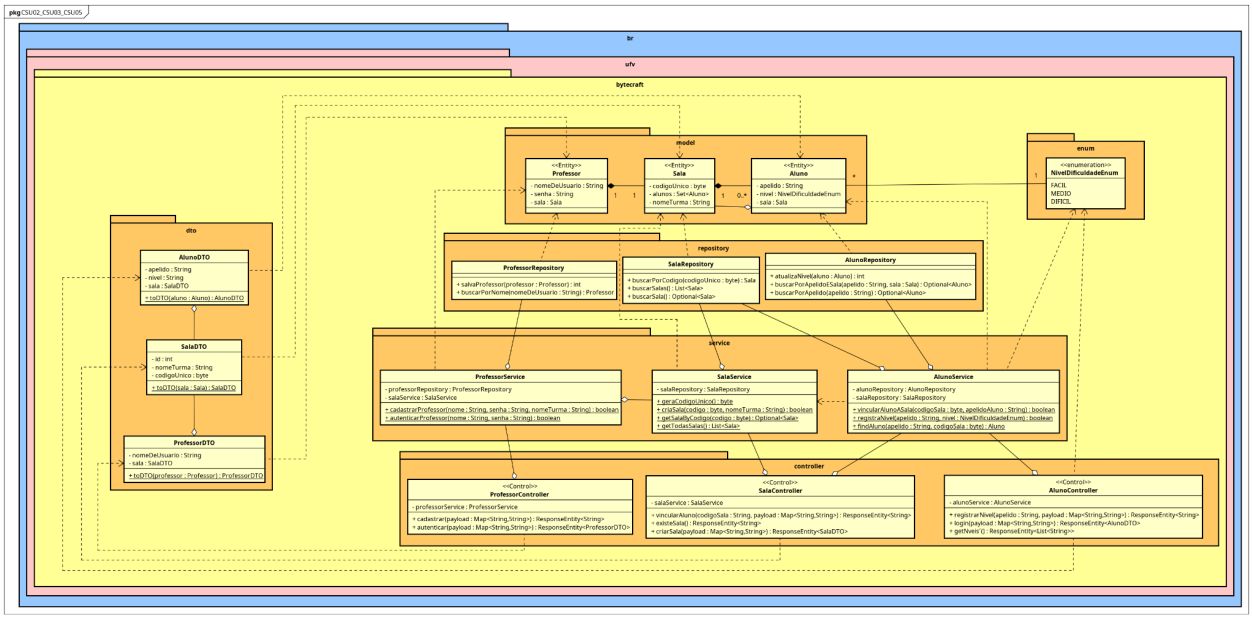

Figura 1: Diagrama de classes com CSU02, CSU03 e CSU05.

Backend CSU02, CSU03 e CSU05

O modelo de classes pode ser visto na Figura 1. Para estes casos de uso foram criadas as classes **Professor**, **ProfessorController**, **ProfessorService**, **ProfessorRepository**, **SalaController**, **SalaService**, **SalaRepository**, **AlunoService, AlunoDTO**, **SalaDTO** e **ProfessorDTO**. O novo pacote criado, **service**, reunirá toda a lógica de negócio e fornecerá métodos estáticos para serem chamados pelas classes de controle. Já o novo pacote **dto** (Data Transfer Object) é responsável por definir as estruturas de dados que transitam entre o cliente e o servidor através da API. A utilização deste pacote é uma boa prática de design que visa desacoplar a representação dos dados da API das entidades de persistência do pacote model, garantindo que apenas as informações necessárias sejam expostas.

A classe **Sala** obteve um novo atributo, o **nomeTurma**, do tipo **String**, que deve ser endereçado no banco de dados. A classe **Professor** deve ter 3 atributos privados com seus respectivos getters e setters: **nomeDeUsuario**, do tipo **String**, **senha**, do tipo **String** e **sala**, do tipo **Sala**, que representa a sala atrelada ao professor. A classe **Sala** tem uma relação de composição de 1 para 1 com **Professor**, indicando que deve existir apenas uma única sala atrelada a somente um professor e que não deve existir uma sala que não esteja relacionada a um professor.

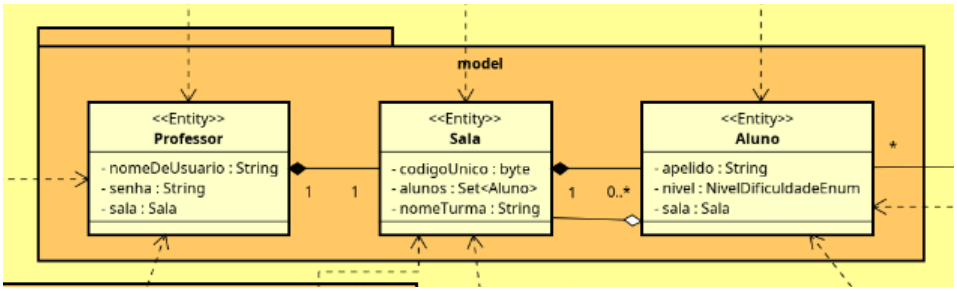

Figura 2: Classes de entidade.

A classe **ProfessorRepository** deve possuir a anotação @Repository e será responsável exclusivamente pela comunicação com a tabela de professores. Ela deve implementar o método **salvaProfessor**, que retorna um inteiro indicando o sucesso da operação e recebe como parâmetro um objeto do tipo **Professor**, sendo responsável por persistir os dados no banco. Adicionalmente, deve implementar o método **buscarPorNome**, que retorna um objeto **Professor**

e recebe como parâmetro o **nomeDeUsuario** do tipo **String**, sendo sua função recuperar um registro de professor específico a partir de seu nome.

A classe **SalaRepository** é designada para a comunicação com a tabela de salas e deve possuir a anotação @Repository. Ela deve implementar o método **salvaSala**, que retorna um **int** e recebe como parâmetro um objeto **Sala** para realizar sua persistência. Também deve implementar o método **buscarPorCodigo**, que retorna uma **Sala** e recebe o **codigoUnico** do tipo **byte** para buscar uma sala específica. Por fim, o método **buscarSalas** deve ser implementado para retornar uma **List<Sala>**, sendo responsável por consultar e retornar todos os registros de salas existentes. Além disso, permite persistir ou atualizar salas com o método **save** e localizar uma sala específica pelo nome da turma através de **buscarSala**, que retorna um **Optional**.

A classe **AlunoRepository** deve implementar o novo método **salvaAluno**, que retorna um int e recebe um objeto **Aluno**, com o objetivo de persistir um novo aluno no banco de dados. Ademais, também oferece o método **save** para persistir ou atualizar alunos, além de **buscarPorApelidoESala**, que encontra um aluno pelo apelido dentro de uma sala, e **buscarPorApelido**, que busca apenas pelo apelido.

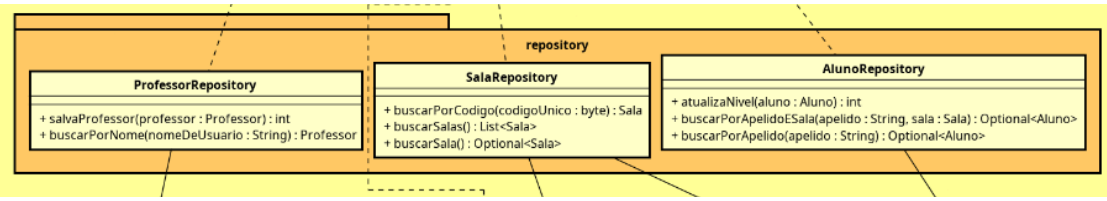

Figura 2: Classes de comunicação com o banco de dados.

A classe **ProfessorService** é uma classe de serviço que deve possuir a anotação @Service. Ela deve ter um atributo privado e final chamado **professorRepository** do tipo **ProfessorRepository**, e **salaService**, do tipo **SalaService** injetados via construtor com a anotação @Autowired. Esta classe deve implementar o método estático (**static**) **cadastrarProfessor**, que retorna um **boolean** indicando o sucesso do cadastro e recebe como parâmetros **nome**, **senha** e **nomeTurma**, todos do tipo **String**; este método é responsável por implementar a validação e o registro de um novo professor e criar e registrar a sala com seu código único usando os métodos de **SalaService**. Adicionalmente, deve implementar o método estático **autenticarProfessor**, que também retorna **boolean** e recebe os parâmetros **nome** e

senha, ambos **String**, para validar as credenciais de um professor.

A classe **SalaService** é uma classe de serviço com a anotação @Service e um atributo privado e final **salaRepository** injetado via construtor. Ela deve implementar quatro métodos estáticos: **gerarCodigoUnico**, que não recebe parâmetros e retorna um **byte** representando um código único gerado (esse método deve conferir no banco de dados se o código gerado é único antes de retornar); **criaSala**, que retorna **boolean** e recebe nome e turma para realizar a criação de uma nova sala; e **confereCodigoSala**, que retorna **boolean** e recebe um **codigo** para verificar sua existência. O método **getSalaByCodigo**, que também recebe um **codigo** do tipo **byte**, é responsável por buscar e retornar um **Optional<Sala>**, encapsulando a entidade Sala correspondente ao código fornecido ou um valor vazio caso não seja encontrada. Por fim, o método **getTodasSalas** não recebe parâmetros e retorna uma **List<Sala>**, sendo sua função consultar e retornar uma lista com todas as salas existentes no sistema.

A classe **AlunoService** é uma classe de serviço, com a anotação @Service, que possui os atributos privados e finais de **AlunoRepository** e **SalaRepository**. Ela deve implementar o método estático **vincularAlunoASala**, que retorna **boolean** e recebe **codigoSala** do tipo **byte** e **apelidoAluno** do tipo **String**, sendo responsável pela lógica de negócio de associar um aluno a uma sala. Além disso, implementa o método **registraNivel**, que retorna **boolean** e recebe **nivel** do tipo enum **NivelDificuldade** e **apelido** do tipo **String**, para persistir o nível de dificuldade escolhido pelo aluno (substitui a lógica antes realizada em **AlunoController**). O método **findAluno** recebe como parâmetros o **apelido** do tipo **String** e o **codigoSala** do tipo **byte**, e sua responsabilidade é consultar o repositório e retornar a entidade **Aluno** correspondente.

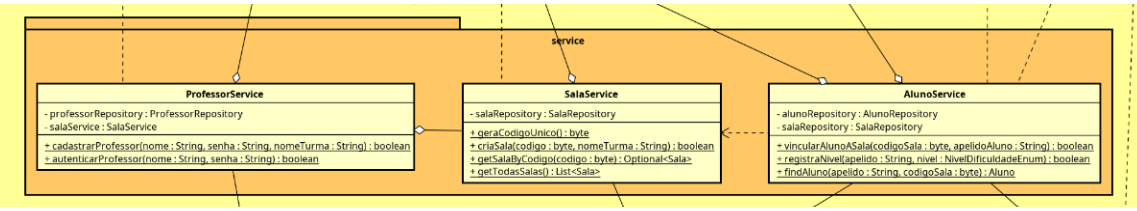

Figura 4: Classes de serviço.

A classe **ProfessorController** é uma classe de controle que deve possuir as anotações @RestController e @RequestMapping("/api/professores") e depende de **ProfessorService** (atributo privado e final injetado via @Autowired). Esta classe implementa o método **cadastrar**,

que retorna **ResponseEntity<String>**, possui a anotação @PostMapping("/cadastrar") e recebe um **payload** do tipo **Map<String, String>** com os dados do professor e da sala. Além disso, implementa o método **autenticar**, com a anotação @PostMapping("/autenticar"), que também retorna **ResponseEntity<String>** e recebe um payload com as credenciais do professor para autenticação.

A classe **SalaController** é uma classe de controle com as anotações @RestController e @RequestMapping("/api/salas") e depende de **SalaService** e **AlunoService** (atributos privados e finais injetados via @Autowired). Ela deve implementar o método **vincularAluno**, anotado com @PostMapping("/{codigoSala}/vincular"), que retorna **ResponseEntity<String>** e recebe um payload com os dados do aluno, além do código da sala via url. Adicionalmente, implementa o método **existeSala**, anotado com @GetMapping("/existe"), que retorna **ResponseEntity<String>** e verifica se existe pelo menos um registro de sala no banco de dados. Agora, **SalaController** expõe o endpoint **criarSala**, responsável por criar uma nova sala a partir do nome da turma informado e retornar a entidade convertida em DTO.

A classe **AlunoController** possui **alunoService**, do tipo **AlunoService** e disponibiliza o método **login** para registrar ou vincular um aluno a uma sala e retornar seus dados básicos, e o método **getNiveis**, que lista os níveis de dificuldade disponíveis no sistema.

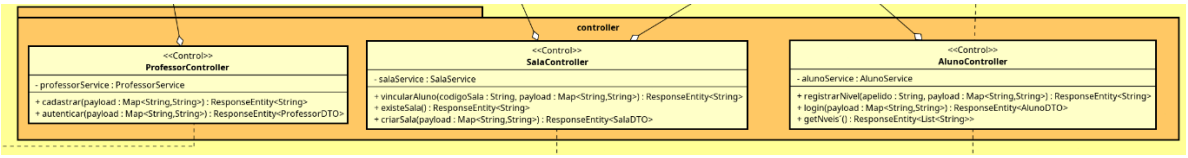

Figura 5: Classes de controle.

A classe **AlunoDTO** é um objeto de transferência de dados que representa a visão externa de um aluno. Esta classe deve possuir os atributos privados: **apelido** do tipo **String**, **nivel** do tipo **String**, e **sala** do tipo **SalaDTO**, que representa a sala à qual o aluno está vinculado. Além disso, a classe deve implementar um método público e estático chamado **toDTO**, que recebe como parâmetro um objeto da entidade **Aluno** e é responsável por realizar a conversão, retornando uma nova instância de **AlunoDTO** preenchida para ser enviada como resposta na API.

A classe **SalaDTO** representa os dados de uma sala que são seguros para exposição na API. Ela deve possuir os atributos privados: **id** do tipo **int**, **nomeTurma** do tipo **String**, e

codigoUnico do tipo **byte**. Esta classe deve implementar um método público e estático **toDTO**, que recebe um objeto da entidade **Sala** como parâmetro e retorna uma instância de **SalaDTO** correspondente. Esta classe é utilizada tanto como um objeto de retorno principal em endpoints de sala quanto como um componente dentro de **AlunoDTO** e **ProfessorDTO**.

A classe **ProfessorDTO** é um objeto de transferência de dados que encapsula a representação externa de um professor. Ela deve possuir os atributos privados **nomeDeUsuario** do tipo **String** e **sala** do tipo **SalaDTO**, indicando a sala criada e gerenciada pelo professor. A classe também deve prover um método público e estático **toDTO**, que recebe um objeto da entidade **Professor** e o converte para sua representação em **ProfessorDTO**.

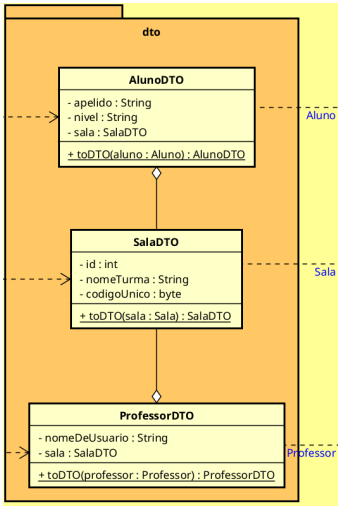

Figura 6: Classes de DTO.

CSU02

Cadastrar Professor (CSU02)

Sumário: Permite que um professor realize seu cadastro no sistema fornecendo nome, senha e turma para posterior autenticação e acesso às funcionalidades exclusivas do perfil de professor.

Ator primário: Professor.

Precondições:

1. O usuário deve ter selecionado o perfil “Professor” no sistema.
2. O usuário deve ter selecionado o campo cadastro de professores

Fluxo Principal

1. O sistema exibe o formulário de cadastro com os campos obrigatórios nome

completo,senha e nome da turma.

2. O professor preenche todos os campos obrigatórios.
3. O professor clica no botão “Cadastrar”.
4. O sistema registra, no banco de dados, os dados do professor e gera o código único da sala.
5. O sistema exibe a mensagem: “Cadastro realizado com sucesso!”.
6. O sistema redireciona automaticamente para a tela de autenticação.

Fluxo de Alternativo(4): Falha no registro

1. Se ocorrer falha ao registrar os dados no banco de dados:

a. O sistema exibe a mensagem: “Erro ao cadastrar. Por favor, tente novamente.”.

b. O sistema mantém o formulário preenchido para nova tentativa.

Fluxo de Exceção (RN02)

1. Se a senha escolhida não tiver 6 caracteres:
    a. O sistema permanece na tela de cadastro e exibe a mensagem: “Senha inválida, deve ter no mínimo 6 caracteres”.
    b. O fluxo retorna ao passo 2 do fluxo principal.

Fluxo de Exceção (RN03)

1. Se o nome completo informado já estiver cadastrado no sistema:
    a. O sistema exibe a mensagem: “Este professor já está cadastrado. Já possui uma conta? Se sim, autentique-se. Não? Realize o cadastro com outro nome.”.
    b. Se o usuário clicar em “autentique-se” presente na mensagem, será redirecionado para a Tela de Autenticação e o fluxo continua no passo c. Caso contrário, ele permanecerá na tela de cadastro, sem alteração do formulário, exceto o campo de senha.
    c. O sistema mantém o formulário preenchido, exceto o campo de senha.

Pós-condições

(Sucesso): Os dados do professor são armazenados no sistema e ele é redirecionado para a tela de autenticação.

(Falha): O sistema informa o erro ocorrido e mantém o formulário para correção.

Regras de Negócio

RN02: A senha deve ter no mínimo 6 caracteres.

RN03: O nome de usuário inserido pelo professor deve ser único.

RN04: Cada sala criada pelo professor deve possuir um código único de 2 dígitos de identificação para evitar conflitos de vinculação.

Frontend

Devem ser criadas 1 página: a página de cadastro do professor. Deve ser seguida como base de implementação a figura abaixo.

Figura 7: Cadastro de professor.

O fluxo das telas deve ser o seguinte, como pode ser visualizado na Figura 7:

- **Tela Cadastro de professor**
    - Ao clicar no botão “Cadastrar”, o sistema deve registrar os dados do professor e redirecionar o professor para a tela “Autenticação”. Essa operação deve realizar uma requisição ao backend, enviando os dados para o caminho “api/professores/cadastrar”.
    - Em caso de falha deve-se exibir uma mensagem de erro de acordo com o especificado em CSU02.
- **Tela Autenticação**
    - Representa o início do fluxo após o sucesso no cadastro do professor.

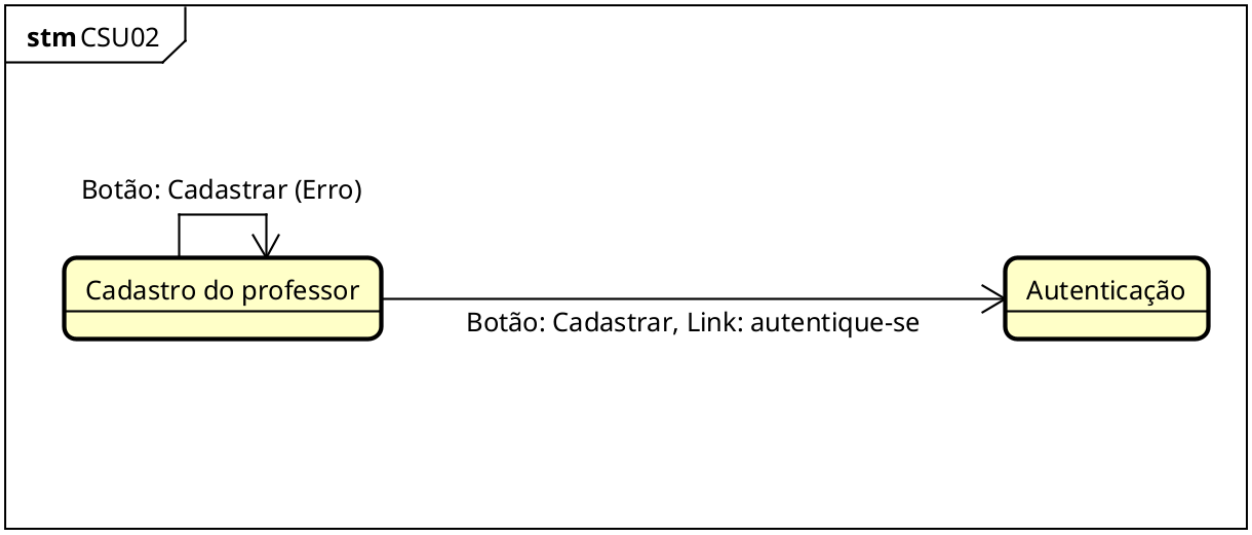

Figura 8: Máquina de estados representando o fluxo de telas de CSU02.

Espera-se que o sistema seja capaz de registrar corretamente o cadastro do professor, redirecionar para o fluxo correspondente e tratar erros de navegação de forma clara, garantindo que o usuário possa realizar uma nova tentativa de cadastro caso ocorra alguma falha.

CSU03

Autenticar Professor (CSU03)

Sumário: Permite que o professor realize autenticação no sistema utilizando o nome e senha cadastrado no sistema.

Ator primário: Professor

Precondições:

1. O professor já deve ter realizado o cadastro no sistema.

Fluxo Principal

1. O sistema exibe para o professor os campos “nome” e “senha”, obrigatórios para autenticação no sistema.
2. O professor realiza a entrada do nome e senha cadastrado.
3. O sistema realiza a validação das credenciais com os dados cadastrados no banco de dados.
4. O sistema redireciona o professor para a tela da sala criada no cadastro do professor e o caso de uso termina.

Fluxo Alternativo (3): Usuário não cadastrado

1. Se não for possível validar as informações preenchidas no banco de dados, o sistema exibe a mensagem “Usuário e/ou senha inválido(s)”.
    a. O sistema permanece na tela de autenticação, permitindo que o professor tente realizar a autenticação novamente e o caso de uso retorna ao passo 2.
    b. O professor pode escolher a opção “Cadastrar” para realizar o cadastro no sistema, iniciando o caso de uso “Cadastrar Professor (CSU02)”.

Pós-condições:

(Sucesso): O sistema exibe a sala criada para o professor, permitindo a realização das ações dentro da mesma.

(Falha): O sistema exibe a mensagem informando que o professor não possui cadastro.

Regras de Negócio:

-

Frontend

Devem ser criadas 2 páginas: página de autenticação e página da sala. Deve ser seguida como base de implementação a figura abaixo.

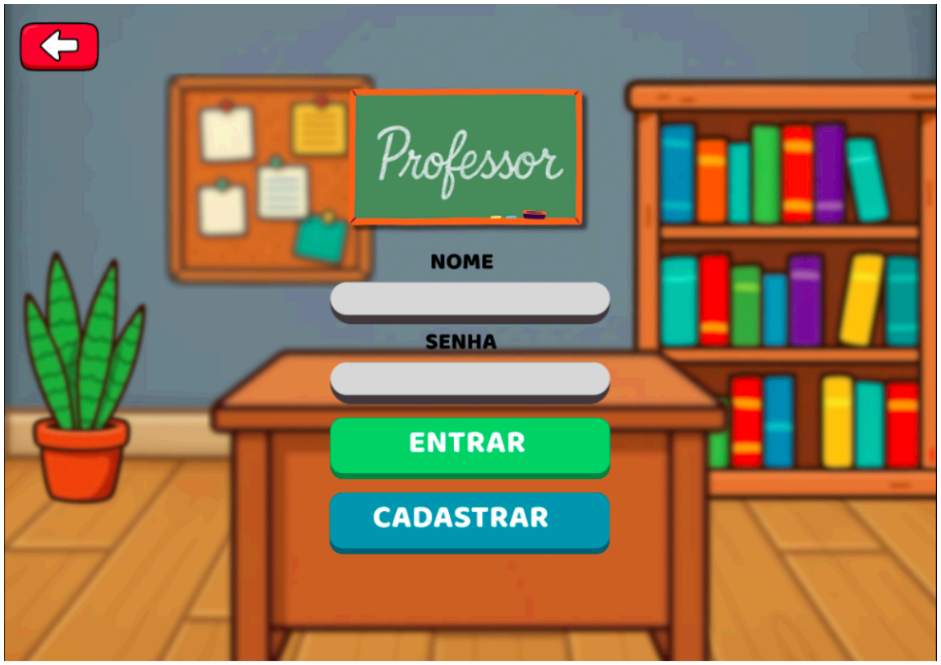

Figura 9: Autenticação.

O fluxo das telas deve ser o seguinte, como pode ser visualizado na Figura 9:

- **Tela de Níveis**
  - Ao clicar no botão “Entrar”, o sistema deve registrar os dados do professor e redirecionar o professor para a “Tela da Sala”. Essa operação deve realizar uma requisição ao backend, enviando os dados para o caminho “api/professores/autenticar”.
  - Em caso de falha deve-se seguir o especificado em CSU03.
- **Tela da Sala**
  - Representa o início do fluxo correto de autenticação.
  - Não é necessário detalhar funcionalidades adicionais neste caso de uso, apenas garantir que o acesso tenha ocorrido corretamente.
- **Tela de Cadastro do professor**
  - Representa o início do fluxo caso o professor clique em “Cadastrar”.

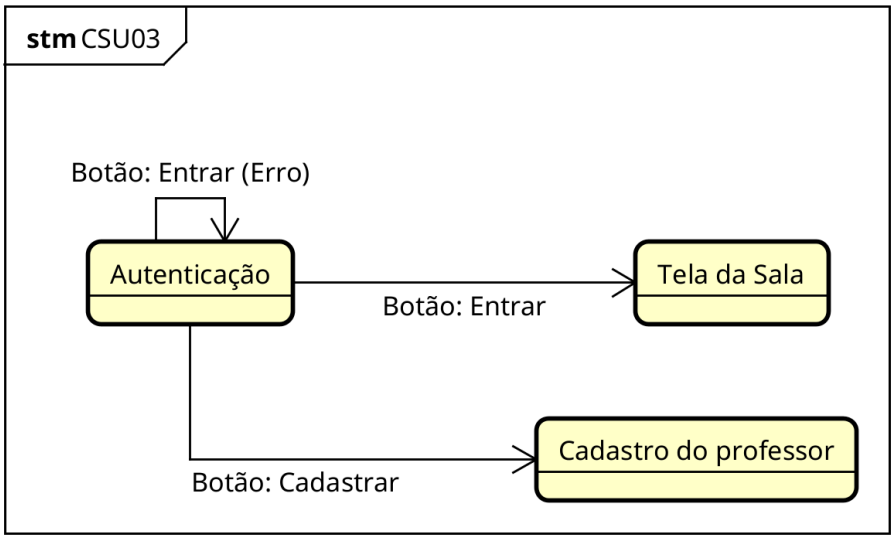

Figura 10: Máquina de estados representando o fluxo de telas de CSU03.

Espera-se que o sistema seja capaz de realizar a autenticação do professor, direcioná-lo para a tela da Sala e lidar com erros de forma clara, permitindo que o usuário tente autenticar-se novamente em caso de falha.

CSU05

Vincular aluno à sala (CSU05)

Sumário: Permite que o aluno se vincule a uma sala criada pelo professor através da inserção do código de 2 dígitos gerado pelo sistema.

Ator primário: Aluno

Precondições:

1. O aluno deve ter selecionado o perfil "Aluno" no sistema.
2. Deve existir pelo menos uma sala criada por um professor no sistema.
3. O aluno deve possuir o código de acesso da sala fornecido pelo professor.

Fluxo Principal

1. O sistema exibe para o aluno um campo obrigatório "Código da Sala" ,“Nome Completo” e um botão "Entrar na Sala".
2. O aluno insere o código de 2 dígitos fornecido pelo professor.
3. O sistema valida o código inserido verificando sua existência no banco de dados.
4. O sistema registra a vinculação do aluno à sala correspondente ao código.
5. O sistema redireciona automaticamente o aluno para a tela de fases e o caso de uso termina.

Fluxo Alternativo (3): Código inválido

1. Se o código inserido não corresponder a nenhuma sala existente no sistema:
    a. O sistema exibe a mensagem "Código da sala inválido. Verifique o código e tente novamente".
    b. O fluxo retorna ao passo 2 do fluxo principal.

Fluxo de Exceção (RN07)

1. Se ocorrer erro ao registrar a vinculação do aluno à sala no banco de dados.
    a. O sistema exibe a mensagem "Erro ao entrar na sala. Tente novamente".
    b. O sistema permanece na tela de vinculação, permitindo nova tentativa.

Pós-condições

(Sucesso): O aluno é vinculado com sucesso à sala selecionada, seus dados ficam associados à turma para cálculo de ranking, e ele é redirecionado para a tela de fases onde poderá iniciar os jogos.

(Falha): O aluno não é vinculado a nenhuma sala, permanece na tela de vinculação e deve tentar novamente com um código válido ou aguardar resolução de problemas técnicos.

Regras de Negócio

RN06: Um aluno só pode estar vinculado a uma sala por vez durante sua sessão no sistema.

RN07: A vinculação do aluno à sala deve ser registrada no banco de dados para permitir o cálculo correto do ranking da turma e manter o histórico de participação.

Frontend

Devem ser criadas 2 páginas: página de entrada do aluno e página de seleção de níveis. Devem ser seguidas como base de implementação as figuras abaixo.

Figura 11: Entrada do aluno.

Figura 12: Seleção de níveis.

O fluxo das telas deve ser o seguinte, como pode ser visualizado na Figura 12:

- **Tela de Entrada do aluno**
    - Ao clicar no botão “Entrar na Sala”, o sistema deve registrar os dados do aluno e redirecionar para a “Tela de Níveis”. Essa operação deve realizar uma requisição ao backend, enviando os dados para o caminho “api/salas/{codigoSala}/vincular”.
    - Em caso de falha, deve-se seguir a especificação em CSU05.
- **Tela de Níveis**
    - Representa o início do fluxo específico após o sucesso na vinculação do aluno.

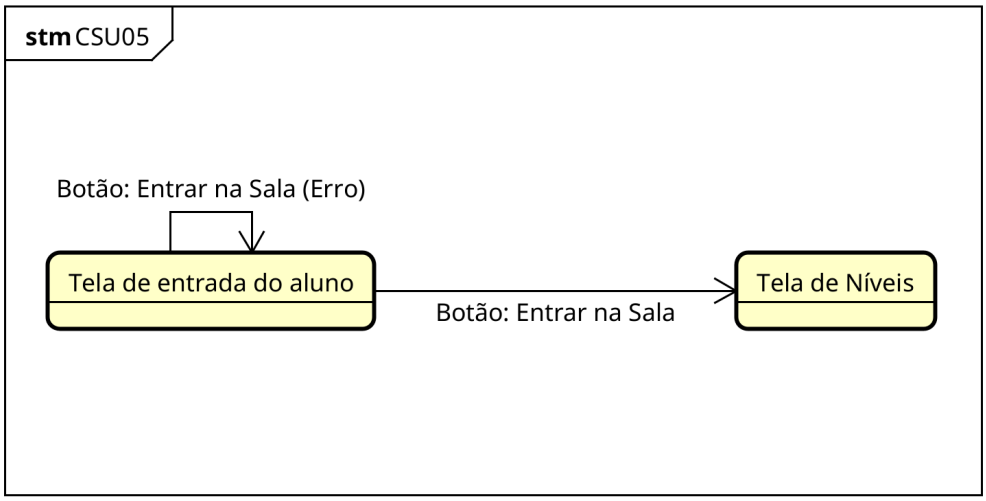

Figura 13: Máquina de estados representando o fluxo de telas de CSU06.

Espera-se que o sistema seja capaz de registrar a o aluno, vinculá-lo à sala e lidar com erros de navegação de forma clara, permitindo que o aluno tente novamente em caso de falha.