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

CAMPUS FLORESTAL



**Backlog da Sprint 01 - CSU03**

GUILHERME BROEDEL ZORZAL - 5064

ARTHUR FERNANDES BASTOS - 4679

Florestal - MG

2024

[Introdução 3](#__RefHeading___Toc335_3906640980)

[Caso de Uso 3](#__RefHeading___Toc333_3906640980)

[CSU03: Selecionar Avatar 4](#__RefHeading___Toc337_3906640980)

[Back-end 5](#__RefHeading___Toc331_3906640980)

[Dev Juniors 5](#__RefHeading___Toc341_3906640980)

[ListaAvatar 5](#__RefHeading___Toc345_3906640980)

[Aluno 6](#__RefHeading___Toc343_3906640980)

# 

# Introdução

O objetivo do seguinte documento consiste em apresentar especificações relacionadas ao caso de uso abordado na primeira sprint, de acordo com o product backlog proposto com base no calendário.

Uma convenção importante com relação aos diagramas de classe: caso a representação da classe não possua nenhum método, deve-se entender que a classe precisa de possuir getters e setters para seus atributos, a menos que tenham sido dadas outras instruções na descrição da classe presente no documento.

# Caso de Uso

O caso de uso abordado na Sprint atual ficou decidido como sendo o CSU03: Selecionar personagem. Sua especificação, proposta na documentação dos casos de uso em ESOF I, pode ser conferida abaixo. Note que o caso de uso não sofreu alterações em relação ao proposto originalmente, ou seja, o fluxo proposto inicialmente continua sendo o mesmo.

Esse caso de uso é o caso de uso responsável por realizar a seleção do avatar do aluno. A documentação do caso de uso pode ser encontrada abaixo.

Uma das modificações que foi realizada no caso de uso em si foi sua renomeação para “Selecionar Avatar”. Foi decidido unanimemente entre os Designers de Arquitetura que essa nomenclatura seria mais clara e fácil de compreender.

|  |
| --- |
| CSU03: Selecionar Avatar **Nome:** Selecionar Avatar.  **Ator Primário:** Aluno.  **Sumário:** O aluno deve ser capaz de selecionar o avatar que o representa melhor, dentre os disponíveis no jogo.  **Pré-condições:**   * Nenhuma.   **Fluxo Principal:**   1. O sistema exibe um conjunto de avatares prontos para o usuário; 2. O usuário escolhe uma dentre as opções; 3. O sistema requisita que o usuário confirme sua escolha; 4. O usuário confirma sua escolha; 5. O avatar é alterado.   **Fluxo alternativo (2): Usuário escolhe sair do menu**   1. Se o usuário escolher sair, o sistema retorna para a tela anterior.   **Fluxo alternativo (4): Usuário cancela sua escolha**   1. Se o usuário optar por cancelar sua escolha, o caso de uso recomeça do passo 1.   **Pós-condições:**   * O avatar do aluno selecionado pelo aluno foi salvo corretamente. |

# 

# Back-end

O backend, na sprint atual, ficará responsável pela implementação de algumas classes e relações envolvendo o Aluno e o Avatar, além de criar uma lista circular para a seleção de avatares.

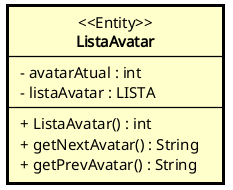
## Dev Juniors

O caso de uso atual requer que seja realizada a implementação das seguintes classes: “Aluno” e “ListaAvatar”. As especificações de cada classe serão explicadas abaixo. Para a implementação, considere que as imagens estão localizadas em: digitovsky/database/imagens/avatares

### ListaAvatar

A classe “ListaAvatar” é a classe que será o coração do funcionamento da tela “selecionar avatar”, que pode ser conferida na figura 3 da seção sobre o Front-end. A ideia dessa classe é criar um esquema de “Carrossel” das imagens de avatar: quando o aluno seleciona a próxima imagem no frontend, o backend retorna a próxima imagem guardada no repositório .

Esse processo deve se dar de forma circular, ou seja, uma vez que a lista chegue ao final, o back retorna a primeira imagem novamente.



**Figura 3 - Classe para Lista de Avatares**

Os atributos da classe ListaAvatar são os seguintes:

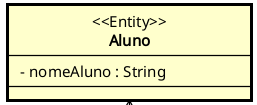
* **avatarAtual : int** = O atributo *avatarAtual* deve conter o id do avatar que foi escolhido pelo usuário. Em outras palavras, é o id da lista correspondente ao avatar atual.
* **listaAvatar : LISTA** = O atributo *listaAvatar* é a estrutura responsável por efetivamente realizar a simulação de uma lista circular. Essa é a estrutura que será responsável por guardar as imagens dos avatares. A estrutura a ser utilizada fica a critério dos dev juniors (por isso seu tipo está marcado como LISTA), ou seja, pode ser ArrayList, HashSet, LinkedList, etc. O importante é que a classe ListaAvatar se comporte como uma lista circular, a estrutura utilizada não tem importância.

Quanto aos métodos a serem criados, serão esses:

* **ListaAvatar():** esse é o construtor da classe. Quando a classe for instanciada, ela deve carregar os caminhos dos arquivos presentes na pasta de imagens (especificada nos endpoints do backend) e montar a lista circular a partir deles. Além disso, esse método deve setar o atributo *avatarAtual* como sendo 0 (o primeiro avatar da lista). Note que por caminho entende-se o caminho relativo considerando como raiz a rota da api (localhost)
* **getNextAvatar() : String** = esse método deve retornar o nome do próximo avatar, ou seja, a string presente na próxima posição da lista. Por exemplo, se o valor de avatarAtual for 3, deve retornar a string presente na posição 4 e setar *avatarAtual* para 4. Caso a lista tenha tamanho 4 (posições 0,1,2,3), deve retornar a string presente na posição 0 que representa o caminho do arquivo e setar o valor de *avatarAtual* para 0.
* **getPrevAvatar() : String** = realiza praticamente a mesma coisa que o método getNextAvatar, porém pegando o avatar anterior. Utilizando o mesmo exemplo de getNextAvatar, ela setaria o valor de avatarAtual para 2 e retornaria e string presente na posição 2
* **setAvatarAtual(nome : String) : int** = Por fim, o método setAvatarAtual receberá um caminho de arquivo (a escolha de avatar do aluno) e, com base nisso, seta o valor de avatarAtual para o valor correspondente. Em outras palavras, o método deve procurar dentro da lista de avatares qual endereço contém a string passada como parâmetro e guardar sua posição em *avatarAtual*

### Aluno

A classe “Aluno” é a classe responsável por guardar as informações referentes ao aluno. Sua implementação é extremamente simples: Ela só precisa possuir os seus respectivos getters e setters e o construtor, que deve atribuir uma string vazia ao atributo “nomeAluno”



**Figura 4: Classe Aluno**