

UNIVERSIDADE FEDERAL DE VIÇOSA · UFV - CAMPUS FLORESTAL

CIÊNCIA DA COMPUTAÇÃO

PROJETO INTEGRADOR 2024

**REFINAMENTO DA SPRINT 4**

CSU02: Gerar e salvar relatório individual

**v.1.0.0**

Guilherme Broedel Zorzal - 5064

Arthur Fernandes Bastos - 4679

Florestal - MG

Novembro de 2024

LISTA DE FIGURAS

[**Figura 1: Classe Relatorio 6**](#_5a1kzcrjzvr6)

[**Figura 3: Classe ControladoraRelatorio 7**](#_3bnak16as2zh)

[**Figura 4: Classe EndPointRelatorio 7**](#_hvshdxef0opl)

[**Figura 5: Classe AcessoBancoRelatorio 9**](#_bfole4tt1jci)

[**Figura 6: Tela de Resultados individuais 9**](#_ol8nmx3q4ycm)

SUMÁRIO

[**1. ATUALIZAÇÕES E CORREÇÕES 3**](#_tmeox120218u)

[**2. INTRODUÇÃO 3**](#_70quw3oh7ylw)

[**3. CASO DE USO 4**](#_ak5dt6g3marc)

[**5. BACK-END 5**](#_i3yyqqcljtlj)

[5.1 Classe Relatorio 6](#_p07vr2c5x6k8)

[5.2 Classe Frase 6](#_xi9spfhz3id4)

[5.3 ControladoraRelatorio 7](#_nfdkj2a8q2oy)

[5.4 EndPointRelatorio 7](#_anxe5q9aes00)

[5.5 AcessoBancoRelatorio 8](#_y4vp23n4kwms)

[**6 FRONT-END 9**](#_zhazk5c76q23)

[6.1 Fluxo de telas 9](#_94mscwsxnwv5)

[6.2 End-Points 10](#_yq4787kb5sfw)

[6.3 Banco de dados 10](#_udsu5n611iv)

# 1. ATUALIZAÇÕES E CORREÇÕES

No caso de uso de cadastro de professor, o back-end deve ser responsabilizado por tratar todo tipo de erro que possa ocorrer envolvendo o cadastro do professor. Isso inclui: tratar cadastros duplicados, senhas curtas demais, login errado (entrou com alguma informação errada no login), existência de caracteres não permitidos nas mensagens, dentro outros casos que possam ser percebidos durante o desenvolvimento. O padrão para esse tipo de erro deve ser o combinado: JSON com campos “field” e “message”. É importante lembrar que não devem ser retornadas informações claras demais, como por exemplo ao fazer login retornar “usuário já cadastrado” (isso indica que algum usuário já possui esse nome, expondo uma informação que não deveria ser exposta). Os caracteres para usernames podem ser apenas os descritos abaixo:

* Maiúsculas: A a Z (26 caracteres)
* Minúsculas: a a z (26 caracteres)
* Números: 0 a 9 (10 caracteres)
* Símbolos: (espaço) ( - ) ( \_ ) (3 caracteres)
* Caracteres de acentuação são permitidos

# 2. INTRODUÇÃO

O objetivo do seguinte documento consiste em apresentar especificações relacionadas ao caso de uso abordado na quarta 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 possuir getters e setters para seus atributos, a menos que tenham sido dadas outras instruções na descrição da classe presente no documento.

# 

# 3. CASO DE USO

O caso de uso abordado na Sprint atual ficou decidido como sendo o CSU02: Gerar e salvar relatório individual. Sua especificação, proposta na documentação dos casos de uso em ESOF I, pode ser conferida abaixo. Note que os casos de usos sofreram poucas alterações em relação ao proposto originalmente, ou seja, o fluxo proposto inicialmente continua sendo o mesmo. Apenas algumas palavras foram atualizadas de forma a melhorar a compreensão.

Esse caso de uso é responsável por receber os dados de cada frase finalizada, a partir dele calcular as métricas e por fim gerar o relatório individual. A documentação do caso de uso pode ser encontrada abaixo.

| CSU02: Gerar e salvar relatório individual  **Nome:** Gerar e salvar relatório individual.  **Ator Primário:** Aluno.  **Sumário:** Finalizar um jogo, salvando as estatísticas relativas a ele e gerando um relatório que descreve o desempenho do aluno no mesmo.  **Pré-condições:**   * O usuário deve ter criado seu personagem; * O texto deve ter sido escolhido pelo professor; * O texto deve ter sido completamente digitado de maneira correta pelo aluno.   **Fluxo Principal:**   1. O sistema gera e exibe o relatório. 2. O usuário pressiona o botão“Jogar novamente”. 3. O sistema dispara a ação “Jogar” com o próximo texto selecionado pelo professor. 4. Fim do caso de uso.   **Fluxo Alternativo (2): o usuário pressiona o botão “Sair”.**   1. O sistema retorna à tela inicial.   **Pós-condições:**   * O aluno pode passar imediatamente para o próximo texto selecionado pelo professor. |
| --- |

4. ESTRUTURA GERAL

Quando é iniciada cada frase, o código implementado no front-end irá fazer a contagem dos dados necessários para as métricas, passando-as através do End point para o back-end. Após o recebimento dos dados, o sistema deve ser capaz de calcular as métricas e fornecer um relatório em uma nova tela com os dados referentes ao total de tempo gasto e dos erros, a média tempo de frase medido em minutos e a média de erros por frase.

# 5. BACK-END

Na sprint atual, o back-end será responsável por implementar algumas classes e suas relações, conforme o diagrama de classes disponível no arquivo DOCX na mesma pasta. Entre as tarefas, destaca-se o desenvolvimento de uma classe Relatório, que incluirá os métodos getters e setters para cada métrica, e de uma classe Frase, destinada a armazenar as informações de cada frase enviada. Além disso, será criada uma classe para gerenciar o acesso ao banco de dados de relatórios, uma classe controladora e, por fim, uma classe de endpoints responsável por integrar o back-end ao front-end.

## 

## 5.1 Classe Relatorio



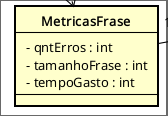
###### Figura 1: Classe Relatorio

A classe Relatorio possui getters e setters referentes a cada tipo de métrica calculada, onde os setters devem ser privados, com exceção do setFrase o qual deve ser capaz de armazenar em uma lista todas as frases enviadas pela classe ControladoraRelatorio, contendo os dados referentes a cada uma (a lista é determinada pelo losango do diagrama UML).

Dessa forma, o método calculaMetricas ao final do processamento de frases irá acessar essa lista e, a partir dos dados de cada campo da classe Frase, gerar as métricas referentes àquele texto (atualizando os campos da classe Relatorio com os setters privados).

LEMBRANDO: O losango fechando da classe relatório implica que a classe deve conter uma lista para guardar as MetricasFrase.

## 5.2 Classe MetricasFrase

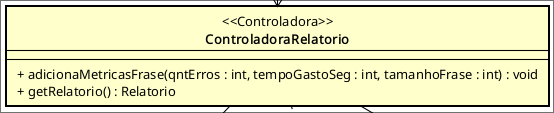


**Figura 2:** Classe Frase

A classe Frase é responsável por armazenar os dados referentes a cada frase finalizada, sendo assim ela deve ser capaz de guardar a quantidade de erros, o tamanho da frase e o tempo gasto. Deve possuir os devidos getters e setters.

## 

## 5.3 ControladoraRelatorio

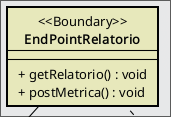


###### Figura 3: Classe ControladoraRelatorio

A classe ControladoraRelatorio é a classe responsável por gerenciar o envio e uso das frases. Seus métodos são:

* adicionaMetricasFrase(qntErros: int, tempoGastoSeg: int, tamanhoFrase: int) : void = Recebe as métricas passadas pela classe EndPointRelatorio, transforma-as em um objeto do tipo Frase, e o envia para a classe Relatorio através do método setFrase() dessa classe.
* getRelatorio(): Relatorio = Retorna o objeto Relatorio que possui instanciado dentro de si para a classe EndPointRelatorio. Além disso, esse método também é responsável por guardar o relatório no banco de dados, através da classe AcessoBancoRelatorio.

## 5.4 EndPointRelatorio



###### Figura 4: Classe EndPointRelatorio

A classe EndPointRelatorio ficará responsável por receber as requisições realizadas pelo front-end. Ela deve escutar as seguintes rotas:

* relatorio
  + GET: Deve retornar um JSON com um *field* correspondendo a um campo da classe Relatorio, como mostrado abaixo. Os *fields* devem ser preenchidos com as informações presentes no objeto Relatorio recebido da classe ControladoraRelatorio.

{

totalErros: --- ;

totalTempo: --- ;

mediaFraseMinuto: ---;

mediaErrosFrase: ---;

totalCaracteresDigitados: ---;

frasesPerfeitas: ---;

}



* relatorio/metrica
  + POST : Recebe um JSON com os campos abaixo e passa-os para a classe ControladoraRelatorio.

{

qntErros: ---;

tempoGastoSeg: ---;

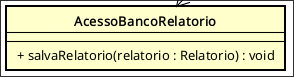
tamanhoFrase: ---;

}



## 5.5 AcessoBancoRelatorio

A classe AcessoBancoRelatorio é responsável por guardar o Relatorio final no banco de dados. Ele possui apenas um método, salvaRelatorio, que recebe um objeto do tipo Relatorio e guarda suas informações no banco de dados (cada atributo para sua coluna específica do banco).



###### Figura 5: Classe AcessoBancoRelatorio

# 

# 

# 6 FRONT-END

Quanto ao que deve ser implementado pelo front-end, serão criadas as telas desenvolvidas no figma correspondentes a “Relatório”. Segue a figura abaixo.



###### Figura 6: Tela de Resultados individuais

## 6.1 Fluxo de telas

Nessa sprint, o Front-end não realiza fluxos de tela significativos. A única coisa que deve ocorrer é o retorno para a página principal ao clicar no botão de retorno (em roxo).

## 

## 6.2 End-Points

Os endpoints a serem utilizados pelo front podem ser encontrados dentro da área “endpoints” do capítulo “Backend” deste documento.

## 6.3 Banco de dados

Para a sprint atual, a tarefa do banco de dados será o armazenamento dos relatórios individuais dos alunos através da tabela ‘RelatorioIndividual’, contendo as métricas que foram calculadas no back-end. As colunas do banco de dados deverão ser as seguintes:

* id
* totalTempo
* totalErros
* totalErros
* mediaFraseSegundo
* mediaErrosFrase
* totalCaracteresDigitados
* frasesPerfeitas

Onde id é a chave primária.