

UNIVERSIDADE FEDERAL DE VIÇOSA · UFV - CAMPUS FLORESTAL

CIÊNCIA DA COMPUTAÇÃO

PROJETO INTEGRADOR 2024

**REFINAMENTO DA SPRINT 5**

CSU09: CRUD de texto

**v.1.0.0**

Guilherme Broedel Zorzal - 5064

Arthur Fernandes Bastos - 4679

Florestal - MG

Novembro de 2024

LISTA DE FIGURAS

[**Figura 1: Classes Dificuldade e Texto 6**](#_zcdajxu3dvox)

[**Figura 2: Classe do acesso ao banco de dados 7**](#_yi6eo7yafmoa)

[**Figura 3: Classe ControladoraCrudTexto 8**](#_4swjd16mf01w)

[**Figura 4: Classe EndPointTexto atualizada 9**](#_1s1q95jxl1xk)

[**Figura 5: Tela da área do professor 12**](#_bbn0mgnbfyh9)

[**Figura 6: Tela do cadastro de texto 12**](#_cnbpfzi65ed7)

[**Figura 7: Tela da central de conteúdos 13**](#_kp6j3v5tjre3)

[**Figura 8: Tela da seleção de texto 13**](#_qect9xovlfi9)

[**Figura 9: Tela da edição de texto 14**](#_hy0jijx5ih7b)

[**Figura 10: Tela de deletar texto 14**](#_ol8nmx3q4ycm)

SUMÁRIO

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

[**2 INTRODUÇÃO 3**](#_70quw3oh7ylw)

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

[**5 BACK-END 6**](#_i3yyqqcljtlj)

[5.1 Classe Texto e Classe Dificuldade 6](#_p07vr2c5x6k8)

[5.3 Classe AcessoBancoTexto 7](#_y3jv26ngueey)

[5.4 Classe ControladoraCrudTexto 8](#_q9i8i0guxsxz)

[5.5 EndPointTexto 9](#_xpqg8downqr7)

[5.5.1 ENDPOINTS 9](#_havvsfs20gga)

[IMPORTANTE 11](#_pgohnri6sus4)

[**6 FRONT-END 11**](#_zhazk5c76q23)

[6.1 Fluxo de telas 14](#_94mscwsxnwv5)

[6.2 End-Points 15](#_yq4787kb5sfw)

[**7 BANCO DE DADOS 15**](#_udsu5n611iv)

# 1. ATUALIZAÇÕES E CORREÇÕES

No front-end, a tela de Área do Professor deve ser atualizada para contemplar as demais funcionalidades pensadas e desenvolvidas. Segue no figma e também na seção relacionada ao front-end, a nova tela.

# 2 INTRODUÇÃO

O objetivo do seguinte documento consiste em apresentar especificações relacionadas ao caso de uso abordado na quinta 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 CSU09: Crud de texto Sua especificação pode ser conferida abaixo. O Caso de CRUD de texto foi proposto posteriormente devido ao fato de ser uma melhoria para o sistema, por isso não existir sua especificação nos casos de uso original.

| CSU09: CRUD de Texto  **Nome:** CRUD de Texto  **Ator Primário:** Professor  **Sumário:** Permitir que o professor crie, leia, atualize e delete textos que serão utilizados pelos alunos durante o jogo.  **Pré-condições:**   * O professor deve estar autenticado no sistema. * O sistema deve estar conectado a um banco de dados operacional.   **Fluxo Principal:**   1. **Criação (Create):**    1. O professor seleciona a opção "Cadastrar novo texto".    2. O sistema exibe uma interface para inserção do texto.    3. O professor insere o texto e confirma a ação.    4. O sistema salva o texto no banco de dados.    5. O sistema confirma que o texto foi cadastrado com sucesso 2. **Leitura (Read):**    1. O professor seleciona a opção "Visualizar textos".    2. O sistema exibe a lista de textos cadastrados.    3. O professor pode selecionar um texto para visualizar seu conteúdo. 3. **Atualização (Update):**    1. O professor seleciona a opção "Editar texto".    2. O sistema exibe os textos disponíveis para edição.    3. O professor escolhe um texto e edita seu conteúdo.    4. O professor confirma as alterações.    5. O sistema atualiza o texto no banco de dados e exibe uma mensagem de sucesso. 4. **Exclusão (Delete):**    1. O professor seleciona a opção "Excluir texto".    2. O sistema exibe a lista de textos cadastrados.    3. O professor seleciona o texto que deseja excluir.    4. O sistema solicita confirmação.    5. O texto é removido do banco de dados, e uma mensagem de confirmação é exibida.   **Fluxos Alternativos:**   * **Erro de conexão com o banco de dados:**   + O sistema informa ao usuário que não foi possível completar a ação devido a uma falha de conexão e fornece a opção de tentar novamente ou retornar ao menu principal**.** * **Texto não encontrado (Read ou Delete):**   + O sistema informa que o texto selecionado não está disponível e retorna à lista de textos.   **Pós-condições:**   * O texto criado, atualizado ou excluído é refletido no banco de dados. * Alterações estão disponíveis para o uso no sistema pelos alunos. |
| --- |

4 ESTRUTURA GERAL

Como estrutura geral, nessa sprint deverá ser desenvolvido a interface e a lógica por trás da criação, visualização, atualização e remoção de texto dentro do sistema, o back-end deverá tratar da utilização dos textos de forma a atender todas as funcionalidades necessárias e o front-end deverá disponibilizar e possibilitar para o usuário, o uso das funcionalidades implementadas.

# 

# 5 BACK-END

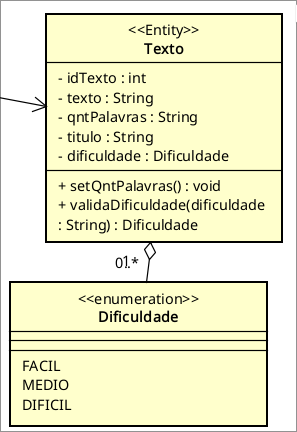
Na sprint atual, o back-end será responsável por implementar algumas classes e suas relações, conforme o diagrama de classes disponível no arquivo DOCX na mesma pasta. O coração da sprint atual é a classe ControladoraCrudTexto, que possuirá boa parte das funcionalidades relacionadas ao uso dos textos. Ela substituirá a classe ControladoraTextoAux assim que for realizado o refinamento do caso de uso SelecionarTexto (próxima sprint).

## 

## 5.1 Classe Texto e Classe Dificuldade

A classe Texto será a classe responsável por guardar o texto escolhido pelo professor, e que será usada posteriormente para possibilitar o uso do jogo. Ela deve possuir uma instância do Enum dificuldade.

O enum dificuldade, por sua vez, será responsável por guardar o nível de dificuldade do jogo, que será escolhido pelo professor durante a criação do jogo. Ele deve possuir as seguintes dificuldades: FACIL, MEDIO e DIFICIL.



###### Figura 1: Classes Dificuldade e Texto

Um ponto importante: o setter de Dificuldade receberá um enum dificuldade (tornando simples sua implementação). Porém a classe Texto terá um novo método chamado validaDificuldade. Esse método receberá uma string contendo o nome da dificuldade (fácil, médio e difícil). Atentem-se ao tratamento de maiúsculas e minúsculas, acentos, dentre outras possibilidades de erro (deve-se aceitar apenas as versões acentuadas sem capitalização, ou seja, tudo minúsculo). No fim validaDificuldade retorna um ENUM Dificuldade.

Note que setQntPalavras está como privado: ele deve ser chamado junto ao setter de texto, calculando o total de palavras presentes no texto e atribuindo o resultado ao atributo qntPalavras.

Além disso, seu construtor deve inicializar o idTexto como -1.

Note que ainda há a necessidade de um getter de dificuldade. “validaDificuldade” não substitui essa função, visto que ele é um método estático. O getter de dificuldade retorna um enum Dificuldade.

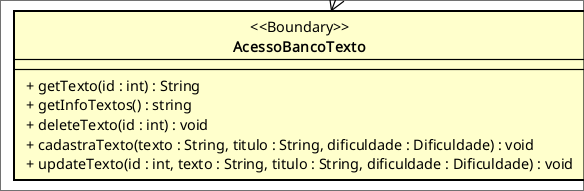
Por fim, quando o método getTexto for chamado, caso não haja nenhum texto carregado na classe (idTexto é igual a -1), esse método deve retornar um erro.

## 5.3 Classe AcessoBancoTexto

A classe AcessoBancoTexto é responsável por gerenciar o acesso ao banco de dados e armazenar as informações de cada texto. Para a função de criação essa classe possui o método cadastraTexto, recebendo o texto escrito, seu título e sua dificuldade. Seguindo para a visualização, ela possui os métodos getTexto, o qual fornece o texto escrito com base no id definido pelo banco de dados (deve retornar um objeto do tipo “Texto” contendo todas as informações do texto), e o método getInfoTodosTextos, que retorna a lista de informações do texto requisitado, nessa lista consta o título, a dificuldade e um trecho inicial do texto (até 50 palavras).

Partindo para a edição, ela possui o método updateTexto, o qual vai receber o id do texto a ser modificado e o novo texto modificado que substituirá o anterior. Além do método deleteTexto, que apaga o texto de id correspondente no banco de dados.

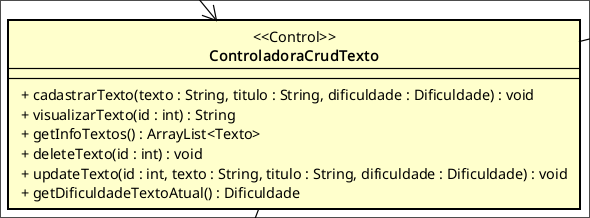
Por fim, para atualização dos textos, temos o método updateTexto(), que recebe as informações do texto e, com base no seu id, atualiza o texto.



###### Figura 2: Classe do acesso ao banco de dados

## 5.4 Classe ControladoraCrudTexto

A classe ControladoraCrudTexto é a classe responsável por realizar a maior parte das mecânicas da sprint atual relacionadas a CRUD de texto.



###### Figura 3: Classe ControladoraCrudTexto

### 

Funções:

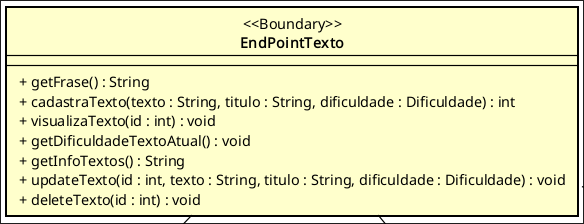
* cadastraTexto(texto: String, titulo: String, dificuldade:Dificuldade): void = Recebe as informações dos parâmetros e envia para o AcessoBancoTexto para realizar o cadastro do texto.
* visualizaTexto(id: int): String = Deve retornar o texto completo com base no id.
* getInfoTextos(): ArrayList<Texto> = retorna uma arrayList de Textos ( a arrayList será retornada do Banco de Dados).
* deleteTexto(id: int) : void = Deleta o texto com base no id através da classe AcessoBancoTexto.
* updateTexto(id: int, texto:String, dificuldade: Dificuldade): void = atualiza o texto com base nos campos fornecidos.
* getDificuldadeTextoAtual() : Dificuldade = Esse método retorna a classe Dificuldade presente na classe Texto, ou seja, o valor do enum dificuldade

Quanto aos métodos cadastraTexto, updateTexto, getInfoTextos e deleteTexto eles são responsáveis por requisitar os métodos respectivos na classe AcessoBancoTexto.

## 

## 5.5 EndPointTexto

Para a sprint atual, alguns métodos foram adicionados além do existente na sprint 3. O método getFrase permanece inalterado.



###### Figura 4: Classe EndPointTexto atualizada

### 

#### 5.5.1 ENDPOINTS

Para facilitar o entendimento das rotas, cada rota terá o nome do método associado. Isso melhora o entendimento e a interligação entre rota e método na hora de implementar (note alguns métodos utilizam o id passado pela própria URL).

* /texto
  + /create
    - POST: cadastraTexto(): Receberá um JSON com os campos texto, título e dificuldade e irá cadastrá-lo utilizando o método correspondente de ControladoraCrudTexto.
  + /visualizar/:id
    - GET: visualizaTexto(): retorna o conteúdo do texto com base em seu id.
  + /getDificuldade
    - GET: getDificuldadeTextoAtual(): Deve retornar a dificuldade do texto selecionado pelo professor.
  + /getInfoTextos
    - GET: getInfoTextos(): A rota getInfoTextos retorna um JSON contendo objetos com o seguinte padrão:

[

{

"id": 1,

"titulo": "Titulo do primeiro texto",

"dificuldade" : "facil, medio ou dificil",

"body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architecto"

},

{

"id": 2,

"titulo": "titulo do segundo texto",

"dificuldade" : "facil, medio ou dificil",

"body": "est rerum tempore vitae\nsequi sint nihil reprehenderit dolor beatae ea dolores neque\nfugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis\nqui aperiam non debitis possimus qui neque nisi nulla"

},

....

{

....

}

]



Cada objeto desses será renderizado diretamente para o professor durante as funcionalidades que lidam com o gerenciamento dos textos.

* + /update/:id
    - POST: updateTexto(): recebe um JSON contendo os campos “id”, “texto” e “dificuldade” e os repassa para a classe ControladoraCrudTexto.
  + /delete/:id
    - DELETE: Recebe um JSON contendo um id e deleta o texto com o id correspondente.

IMPORTANTE: em caso de eventuais erros ou situações adversas, o back-end deve retornar um JSON com o formato adotado até então: campos “field” e “message”.

## IMPORTANTE

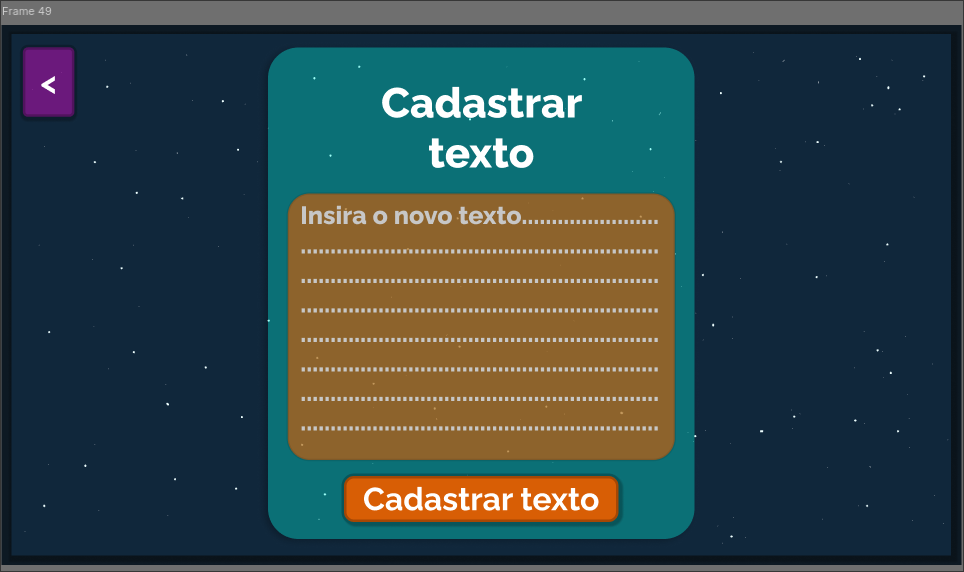
Essa é uma das sprints que mais trabalha com manipulação do banco de dados e requisição de informações e, portanto, é extremamente importante que o back-end seja desenvolvido de forma robusta. Em outras palavras, espera-se que os desenvolvedores sejam capazes de identificar falhas e problemas e tratá-los durante a implementação. Alguns exemplos: Atualizar o nome do texto para um nome já existente; cadastrar textos excessivamente grandes (acima de 5000 palavras é uma boa métrica); cadastrar textos com informações faltosas; entre outras possibilidades.

# 

# 6 FRONT-END

Quanto ao que deve ser implementado pelo front-end, serão criadas as telas desenvolvidas no figma correspondentes a “Crud de texto”. Seguem as figuras abaixo.

###### Figura 5: Tela da área do professor



###### Figura 6: Tela do cadastro de texto



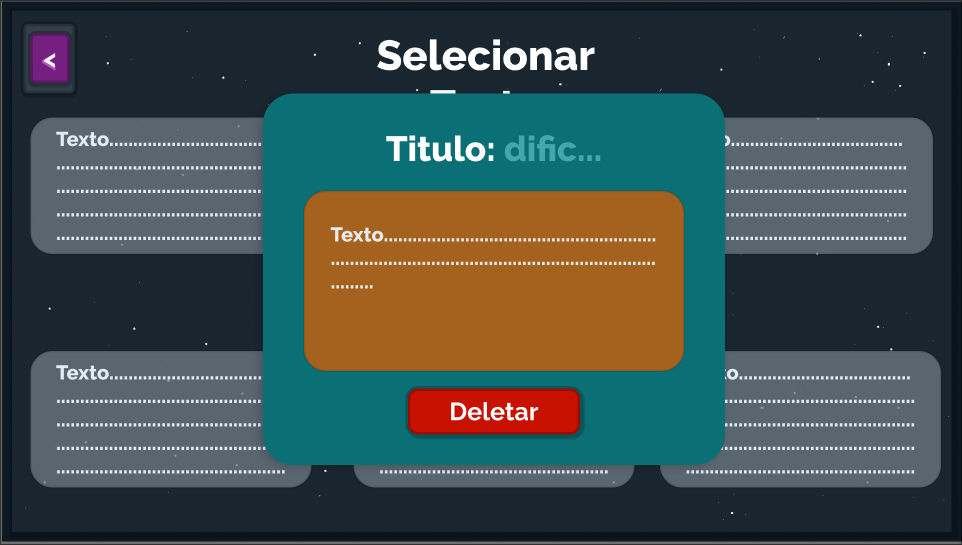
###### Figura 7: Tela da central de conteúdos



###### Figura 8: Tela da seleção de texto



###### Figura 9: Tela da edição de texto



###### Figura 10: Tela de deletar texto

## 6.1 Fluxo de telas

Nessa sprint, o Front-end deverá realizar o fluxo de telas da seguinte forma:

* Na tela de Área do Professor, cada botão deverá redirecionar para sua área correspondente.
* Para cada tela de manipulação de texto, ao selecionar o texto, o front deverá exibir a tela correspondente, de acordo com o que foi escolhido na input de seleção (input de modo)

## 

## 6.2 End-Points

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

## 

# 7 BANCO DE DADOS

O banco de dados deve conter uma tabela chamada Textos, que armazenará as informações dos textos criados. Essa tabela terá os seguintes campos: id (chave primária para identificar o texto), texto (conteúdo do texto), titulo (título do texto) e a dificuldade (nível de dificuldade). O sistema deve permitir as operações de cadastro, atualização, consulta e exclusão de textos.

Para isso, ao cadastrar um texto, será necessário inserir seu conteúdo, título e dificuldade. As atualizações devem modificar apenas os campos alterados. Além disso, deve ser possível consultar um texto específico pelo seu ID ou recuperar metadados de todos os textos, como título e dificuldade. Por fim, a exclusão deve permitir remover textos com base no ID fornecido.