UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Matheus Pimenta Reis

Sistema de Informação para Gerenciamento de Progressão e Promoção Funcional

Uberlândia, Brasil

2021

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Matheus Pimenta Reis

Sistema de Informação para Gerenciamento de Progressão e Promoção Funcional

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Bruno Augusto Nassif Travençolo

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2021

Matheus Pimenta Reis

Sistema de Informação para Gerenciamento de Progressão e Promoção Funcional

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Trabalho aprovado. Uberlândia, Brasil, 11 de novembro de 2021:

Bruno Augusto Nassif Travençolo
Orientador

Professor

Professor

Uberlândia, Brasil
2021

*D*edico à minha mãe, Maria Abadia Pimenta Reis, que sempre me deu o apoio e acreditou e investiu em mim. À minha vó Maria Rita Pimenta que é minha mãe de criação que sempre me inspirei e cuidou de mim. À minha noiva Isabela Fernandes Silva Simões por me acompanhar nesta longa jornada.

Agradecimentos

Agradeço ao meu professor orientador Bruno Augusto Nassif Travençolo pela paciência e por todo o apoio durante o desenvolvimento do projeto.

Deixem que o futuro diga a verdade e avalie cada um de acordo com o seu trabalho e realizações. O presente pertence a eles, mas o futuro pelo qual eu sempre trabalhei pertence a mim. Nikola Tesla

Resumo

A elaboração de relatórios para promoção/progressão de carreira dos docentes da Universidade Federal de Uberlândia (UFU) não era padronizada e não existia um sistema no qual os docentes conseguissem acompanhar de forma organizada suas informações sobre sua promoção/progressão de carreira. Na Faculdade de Computação, a partir de 2019, foi disponibilizado o sistema SCAD – Sistema de Cadastro de Atividades Docente – para apoio a essas atividades. Neste trabalho, são apresentadas melhorias e novas funcionalidades para o SCAD, que melhoraram a usabilidade do sistema e ampliam o seu uso, facilitando a elaboração do relatório a ser apresentado e avaliado. Serão apresentados os recursos e funcionalidades desenvolvidas e o aprendizado adquirido durante a análise e o desenvolvimento do projeto.

Palavras-chave: Promoção/Progressão, carreira, docentes, DDD.

Lista de abreviaturas e siglas

SCAD	Sistema de Cadastro de Atividades Docente
UFU	Universidade Federal de Uberlândia
FACOM	Faculdade de Computação
SEI	Sistema Eletrônico de Informações
CONDIR	Conselho Diretor
API	Interface de programação de aplicações (*Application Programming Interface*)
ORM	Mapeamento objeto relacional (*Object Relational Mapping*)
DDD	Design orientado a domínio (*Domain Driven Design*)
POO	Programação Orientada a Objeto
SGBD	Sistema Geranciador de Banco de Dados (*Domain Driven Design*)
HTML	Linguagem de Marcação de HiperTexto (*Hyperext Markup Language*)
XML	Linguagem de Marcação Extensiva (*Extensible Markup Language*)
CSS	Folhas de Estilo em Cascata (*Cascading Style Sheets*)

Sumário

1 Introdução

A carreira para os docentes da Universidade Federal de Uberlândia (UFU) está dividida em classes, denominações, titulações e níveis. No caso da UFU, existe uma resolução (No 03/2017 do CONDIR (Conselho Diretor)) que regulamenta esse processo. Nessa resolução, estão listados um conjunto de atividades que o servidor pode realizar, sendo que cada atividade possui uma pontuação correspondente. Após um período de interstício de pelo menos dois anos, os docentes devem relatar todas as atividades desenvolvidas nesse período e somar a pontuação referente a cada atividade. A Figura 1 mostra a pontuação necessária a ser atingida para promoção ou progressão para cada nível da carreira. Ao atingir a pontuação necessária para nível pretendido, e após análise das atividades e da documentação necessária, o docente recebe sua progressão/promoção de carreira. Quem analisa essa documentação são os docentes da mesma Unidade Acadêmica. Tudo é realizado e documentado por meio do Sistema Eletrônico de Informações (SEI) da UFU, que é o sistema usado internamente para controle de processos da Universidade.

Classe	Denominação	Titulação	Nível			
			I	II	III	IV
A	Auxiliar	G, A ou E	-	600	-	-
	Assistente A	M	-	610	-	-
	Adjunto A	D	-	630	-	-
B	Assistente	G, A ou E	620	630	-	-
	Assistente	M	630	650	-	-
	Assistente	D	650	670	-	-
C	Adjunto	G, A e E	640	650	660	670
	Adjunto	M	660	680	700	720
	Adjunto	D	700	730	760	790
D	Associado	D	840	880	920	960
E	Titular	D	1000			

Figura 1 – Pontuação de referência da carreira do magistério superior para docentes no regime de dedicação exclusiva de 40 horas (CONDIR, 2017).

Atualmente na Faculdade de Computação da UFU os docentes utilizam um sistema chamado Sistema de Cadastro de Atividades Docente (SCAD). O SCAD é um sistema de informação que fornece um lugar unificado para gerenciar todas as informações referentes a progressão/promoção de carreira, podendo efetuar o lançamento de atividades e sua respectiva documentação, acompanhar, mediante relatórios, o andamento do processo e sempre ter as informações atualizadas.

O sistema também provê uma seção para os avaliadores (que são os docentes que avaliam o relatório submetido), em que são encontrados recursos para que o mesmo possa fazer a análise e correção da documentação submetida pelos professores.

1.1 Objetivo

Implementar de novos recursos visando melhorar a experiência e usabilidade dos usuários do Sistema de Cadastro de Atividades Docente (SCAD).

1.1.1 Objetivos específicos

- Identificar quais os requisitos mais solicitados pelos docentes desde a implantação do SCAD.
- Prover melhorias no módulo do avaliador.

1.2 Justificativa

O sistema SCAD está em funcionamento desde 2019, porém, com o uso do sistema foram levantados pontos de melhoria e novos requisitos a serem implementados no sistema. Se faz necessárias melhorias funcionais, acrescentando novos recursos no sistema e melhorias na interface de forma que fique intuitivo e assim atender as necessidades dos docentes.

O módulo de avaliação dos relatórios está funcional, porém, não está sendo utilizado pelos docentes, pois é necessário desenvolver novas funcionalidades para auxiliar no processo de correção, logo se faz necessário implementar novos recursos para que de fato o sistema de torne uma ferramenta integrada de progressão/promoção de carreira para os docentes da UFU.

1.3 Organização do Trabalho

Este trabalho de conclusão de curso foi estruturado com os seguintes tópicos. No Capítulo 2 é apresentada a arquitetura e as tecnologias utilizadas no SCAD. No Capítulo 3 é apresentada a metodologia e as tarefas implementadas. No Capítulo 4 é apresentada a conclusão.

2 Referencial teórico

Este capítulo descreve o ambiente e arquitetura utilizados para o desenvolvimento do sistema.

2.1 Tecnologias

Esta seção apresenta as tecnologias que foram utilizadas no desenvolvimento do SCAD.

O sistema foi construído utilizando dois módulos, *back-end* e o *front-end*. O primeiro módulo, que é o *back-end*, foi construído em C#, .NET Core. O sistema provê serviços via Interface de programação de Aplicações (API) RESTful, já a parte de banco de dados é feita pelo PostgreSQL, também é utilizado o conceito de *Object Relational Mapping* (ORM), que utilizando a abordagem de Programação Orientada a Objetos (POO) se tem um mapeamento de classes para tabelas em banco de dados relacionais. O segundo módulo, *front-end* consome os serviços e expõe a interface para interação com o usuário é implementando utilizando Angular 2+ como *framework*. Já para a interface gráfica é utilizado *bootstrap*. Para o controle de fontes foi utilizado git, tendo como repositório remoto o BitBucket. Cada uma das tecnologias listadas neste parágrafo serão detalhadas abaixo.

2.1.1 C#

C# (C Sharp) é uma linguagem de programação orientada a objetos com forte tipagem, multiparadigma, criada pela *Microsoft* para a arquitetura .NET. Ela foi utilizada no projeto como linguagem principal para o desenvolvimento do módulo do *back-end* (MICROSOFT, 2021a).

2.1.2 .NET Core

.NET Core é uma plataforma para desenvolvimento e execução de aplicações de sistemas web, *desktop* e *mobile* com código aberto desenvolvido pela *Microsoft* e pela comunidade do *Github* (MICROSOFT, 2021c).

2.1.3 PostgreSQL

O PostqreSQL é um sistema gerenciador de banco de dados relacional (SGBD) de código aberto que é empregado para o desenvolvimento de aplicações. É popular porque é

confiável, escalonável, suporta muitos tipos diferentes de armazenamento de dados e tem uma licença de código aberto (POSTGRESQL, 2021).

2.1.4 Angular 2+

Angular é *framework* desenvolvido pela Google para o desenvolvimento de aplicações web. É voltado para a construção da interface de aplicações (*front-and*), faz uso de Linguagem de Marcação de HiperTexto (HTML), Folhas de Estilo em Cascata (CSS) e, principalmente, JavaScript (ANGULAR, 2021) (TOTVS, 2020).

2.1.5 Bootstrap

Bootstrap é um *framework* voltado para *front-end* que fornece estruturas CSS para criação de sites e aplicações responsivas de forma rápida e visando sempre a simplicidade e a facilidade. Sua lógica se baseia em criar interfaces com base em colunas (BOOTSTRAP, 2021).

2.1.6 Entity framework

O *entity framework* é um mapeador de banco de dados para C#, desenvolvido pela *Microsoft*. Básico e fácil de usar, é uma boa opção para facilitar o desenvolvimento, visto que não é necessário o programador interagir com o banco diretamente. Entre as principais vantagens deste *framework* está no suporte as consultas LINQ, controle de alterações, atualização e migrações de esquema (MICROSOFT, 2021b).

2.1.7 DDD

Design orientado a domínio (DDD),*Driven Domain Desing*, é uma abordagem de modelagem de software que possui algumas camadas explicitas para facilitar o desenvolvimento. Este tipo de estrutura foi desenvolvido lavando em consideração os princípios do *SOLID* (Ver detalhes na próxima seção) e programação orientada a objetos (EVANS, 2004).

Conforme ilustrado pela Figura 2, as quatro camadas que compõem o DDD são:

- Camada de Interface do usuário (*User Interface Layer).*

  A camada de interface é responsável pela interação com o usuário, estando aberta a várias abordagens. Por exemplo, expor uma API para outro sistema consumir.

- Camada de aplicação (*Application Layer*).

  A camada de aplicação é responsável por coordenar as atividades da aplicação.

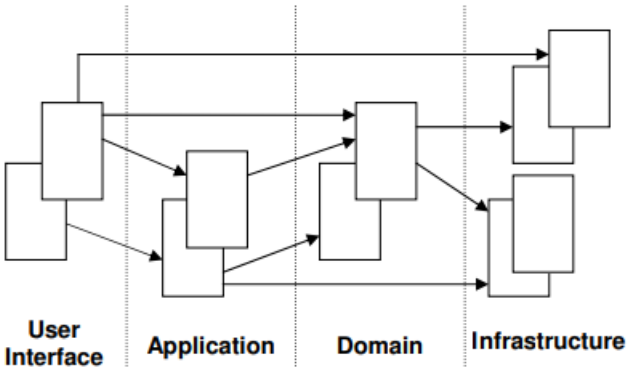

Figura 2 – DDD (MARINESCU, 2007)

- Camada de domínio (*Domain Layer*).

  A camada de domínio implementa todas as regras de negócio da aplicação.

- Camada de infraestrutura (*Infrastructure Layer*).

  A camada de infraestrutura é a camada responsável por prover serviços de persistência ou suporte tecnológico para as demais camadas, como a chamada de um serviço externo ou o envio de email. É a camada de mais baixo nível da aplicação.

Um facilitador, depois de entender como é seu funcionamento as alterações solicitadas foram mais intuitivas de se localizar no código, ou seja, o DDD facilitou o entendimento do negócio visto que pela separação dos domínios é fácil identificar quais as camadas reesposáveis pelo recurso a ser implementado.

2.1.8 SOLID

SOLID, é um acrônimo de 5 princípios de programação que visam a boa prática no desenvolvimento de *software* orientado a objetos, idealizado por Robert C. Martin, são eles: (PAIXAO, 2019)

- S. *Single responsability principle*. Uma classe pode ter um, e somente um, motivo para mudar.
- O. *Open/Close principle*. Objetos ou entidades devem estar abertos para extensão, mas fechados para modificação
- L. *Liskov Substituition Principle*. Uma classe derivada deve ser substituível por sua classe base
- I. *Interface Segregation Principle*. Uma classe não deve ser forçada a implementar interfaces e métodos que não irão utilizar
- D. *Dependency Inversion Principle*. Dependa de abstrações e não de implementações

2.1.9 Mapeamento objeto-relacional

O Mapeamento objeto-relacional (ORM) é uma técnica de programação que permite ao programador trabalhar com dados na forma de objetos sem ter que se preocupar em como esses dados são armazenados no banco de dados.

A melhor maneira de entender o ORM é vê-lo como uma camada de abstração entre o código-fonte e o banco de dados, permitindo que os desenvolvedores trabalhem com objetos de dados enquanto ocultam os detalhes de manipulação e consulta desses objetos em seus aplicativos (XIA; YU; TANG, 2009).

O ORM foi útil no que se tange a modificação/manutenção de código, para criar alterar, deletar ou atualizar uma tabela, basta acessar a classe que faz o mapeamento do domínio desejado, por exemplo AtividadeMap, e configurar via código, ou seja, por meio de uma linguagem de programação é possível parametrizar qual será a estrutura a ser replicada no banco de dados, como acrescentar uma coluna na tabela. Outra vantagem é a facilidade de replicar as modificações, que se executa rodando um comando via *package manager console* do .NET conhecido como *migration*.

2.1.10 Git

O git é a tecnologia utilizada para controle de versão do código. Ele garante desempenho, segurança e flexibilidade. O git já estava implantando no sistema. Utiliza como repositório remoto o BitBucket que provê ferramentas de colaboração, permitindo compartilhamento de dados, criação e implantação de códigos e automatização de testes.

O git teve um papel fundamental para o desenvolvimento do projeto, visto que foi possível ver alterações feitas por outros desenvolvedores e auxiliar no desenvolvimento dos recursos, já que o git oferece a possibilidade de acessar o histórico do código e ver com detalhes o passo a passo que foi seguindo. (ATLASSIAN, 2021)

3 Desenvolvimento

3.1 Metodologia

O sistema SCAD desenvolvido por (SILVA, 2019) foi utilizado como base para este projeto. Novos requisitos visando maior praticidade foram desenvolvidos com o intuito de agregar mais recursos e ferramentas aos docentes.

O controle das atividades foi feito por meio de um site de gerenciamento de atividades chamado Trello, em que são organizadas as atividades a se fazer, as que estão sendo feitas e as concluídas.

Antes, uma primeira etapa, foi a identificação dos principais requisitos que eram solicitados pelos usuários do sistema. Esses requisitos são descritos abaixo.

3.1.1 Novos requisitos solicitados para o SCAD)

- Módulo de gerenciamento de atividades

  - Utilizar um documento já vigente no relatório para preencher o documento da nova atividade.

  - Remover relação de todos os documentos referenciados que seja de reuso.

  - Remover *bug* da data início e fim de interstício.

  - Melhorar navegação na escolha do nível de promoção/progressão de carreira.

- Módulo de relatório

  - Adicionar recurso no relatório para mostrar a quantidade de horas lançadas na atividade.

  - Reformulação do sub-módulo de relatório.

- Módulo Gerenciamento para o avaliador

  - Fornecer meios para se consultar a resolução.

  - Melhoria na interface com a finalidade de ficar mais intuitiva.

A seguir, cada um desses requisitos são detalhados e é feita uma explicação de como foram implementados no sistema.

3.2 *Task 1* - Reuso de documento

Problema: É necessário criar um mecanismo que permite o professor fazer a reutilização de um documento comprobatório de atividade.

Esta funcionalidade se faz necessária para aprimorar a usabilidade no módulo de lançamento de atividades, pois, um mesmo documento comprobatório pode servir para mais de uma atividade. A versão atual do sistema exigia do usuário o cadastramento individual de cada documento comprobatório, mesmo em casos em que o documento serviria para comprovar dois itens distintos. Por exemplo, um diário de classe serve para pontuar uma atividade refente ao número de horas que o docente ministrou uma disciplina e também serve para comprovar o número de alunos da turma, sendo que esse número é usado em um outro item de pontuação.

Para criar a funcionalidade, foi necessário remodelar algumas tabelas do banco de dados da estrutura inicial do projeto, ou seja, alterar o mapeamento das classes atividade e documento. Também se fez necessário alterar as camadas de *controller*, que é utilizada para receber as requisições e direcionar para as camadas mais internas, camada de *service*, que é responsável pela regras de negócio, e faz a junção de várias classes com o propósito de possibilitar implementar as regras de negócio, por último a camada de *repository*, responsável pelo acesso ao banco de dados, alterando as camadas específicas da atividade e criar métodos para auxiliar na verificação e na remoção das atividades de reuso.

Para adequar o recurso de reuso dos documentos das atividades foi necessário alterar as relações das tabelas. Na classe AtividadeMap foi adicionado o relacionamento de um para muitos da tabela de documento. Na classe DocumentoMap foi retirado a relação de um para um da atividade, visto que agora pode-se ter várias atividades utilizando o mesmo documento.

Na camada de *controller* da atividade, classe AtividadeController, foi criado um *end-point* com a implementação dos métodos GetReuseActivity e RemoveReuse que repassa para as demais camadas e prover dados para camada de *view*.

Na camada de *service* da atividade, classe AtividadeService, foram criados os métodos GetReuseActivity e RemoveAtividadeReuse. No método GetReuseActivity ele repassa para o *repository* verificando se a atividade é de reuso. No método RemoveAtividadeReuse é feita as verificações de objetos inválidos, ou seja, se alguma informação repassada pelo *front-end* é inválida. Dependendo da entrada do usuário se tem dois cenários, o primeiro é quando o usuário escolhe remover todos os documentos de todas as atividades referenciadas, ou seja, todas as atividades que incluir aquele documento. Para isso, foi necessário remover o documento pelo *id* da tabela de documento. O segundo cenário é remover somente o documento da atividade desejada, para isso o documento não é removido da tabela de documento.

Na camada *repository* da atividade, na classe AtividadeRepository, foi criado uma consulta SQL, método VerifyReuseActivity, que utiliza as tabelas de atividade e documento para verificar se uma atividade é de reuso.

Na camada do *front-end* no módulo de lançamento de atividades, foi incluso recursos para adicionar, atualizar ou remover uma atividade. Se para um relatório de atividades já possuir algum lançamento de atividade é habilitado a possibilidade do usuário selecionar alguma atividade para fazer o reuso do documento da mesma não exibindo a atividade atual. Ao atualizar ou remover uma atividade é feita uma verificação se a atividade é de reuso. Se a mesma cair na condição é mostrado ao usuário a possibilidade de atualizar ou deletar todos os documentos referenciados pela atividade, ou atualizar ou remover somente a atividade da interação. Podemos observar o resultado nas Figuras 3 e 4

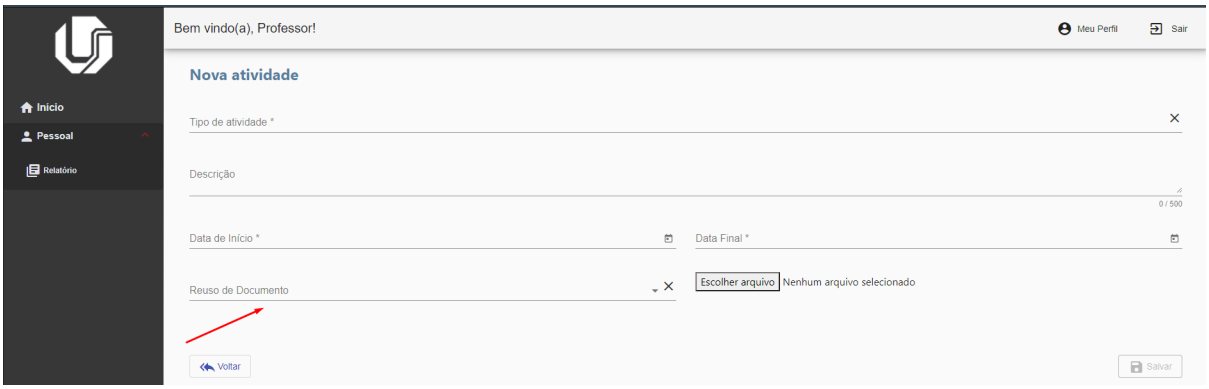

Figura 3 – Reuso de documento.

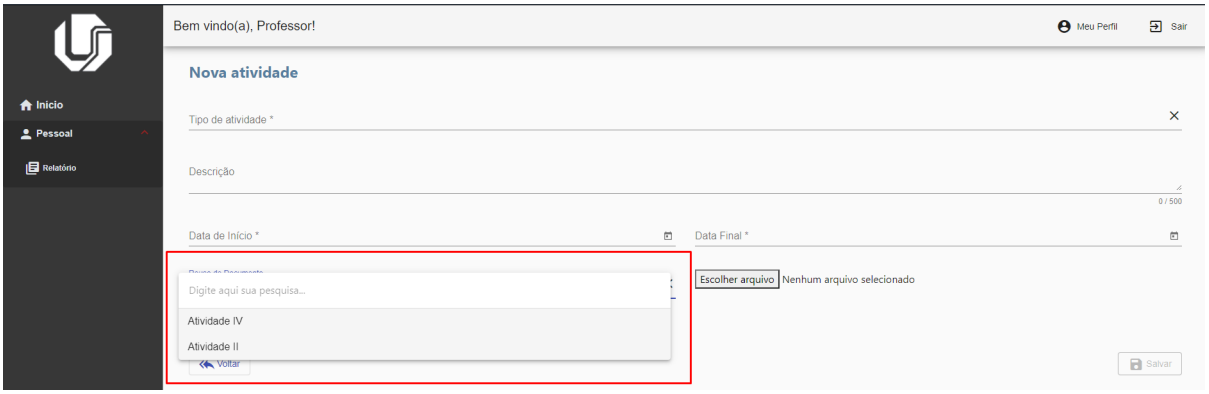

Figura 4 – Visão *dropdown* reuso de documento.

3.3 *Task 2* - Identificação Promoção/Progressão

Problema: Ao gerar o relatório final, está gerando por padrão com o título de Progressão de Carreira.

Essa funcionalidade é necessária para incrementar a assertividade do relatório já que o avaliador até então não tinha fácil a visualização da informação se relatório é para promoção ou progressão de carreira.

Conforme a Figura 1 existe uma diferença entre progressão e promoção de carreira, quando um docente progride de nível se trata de uma progressão, já quando o mesmo progride de denominação se trata promoção de carreira.

Para o desenvolvimento do recurso foi criado uma tabela no banco de dados tipo_progressão_promoção, que contém o mapeamento de progressão e promoção, foi acrescentada uma coluna na tabela de progressão. Utilizando a tabela de pontuação de referência, foi feita uma atualização no banco de dados, ou seja, foi adicionado para cada pontuação se é promoção ou progressão. Para fazer refletir a informação no relatório, a estrutura do projeto facilitou o processo, pois bastou fazer o método GetTemplateHtml receber o relatório e a partir do relatório chega-se no objeto da tabela de progressão.

Como o projeto foi construído com uma arquitetura de ORM, bastou passar o objeto relatório que já estava sendo utilizado para alimentar outras partes do relatório como parâmetro para o método. Dentro do objeto encontramos o objeto TabelaProgressao e dentro do objeto encontramos finalmente o TipoProgressaoPromocao que contém a descrição do relatório que se procura. Foi feito um tratamento se não encontrar dados colocar por padrão exibir Progressão de carreira. As Figuras 5 e 6 ilustram essa nova funcionalidade do sistema.

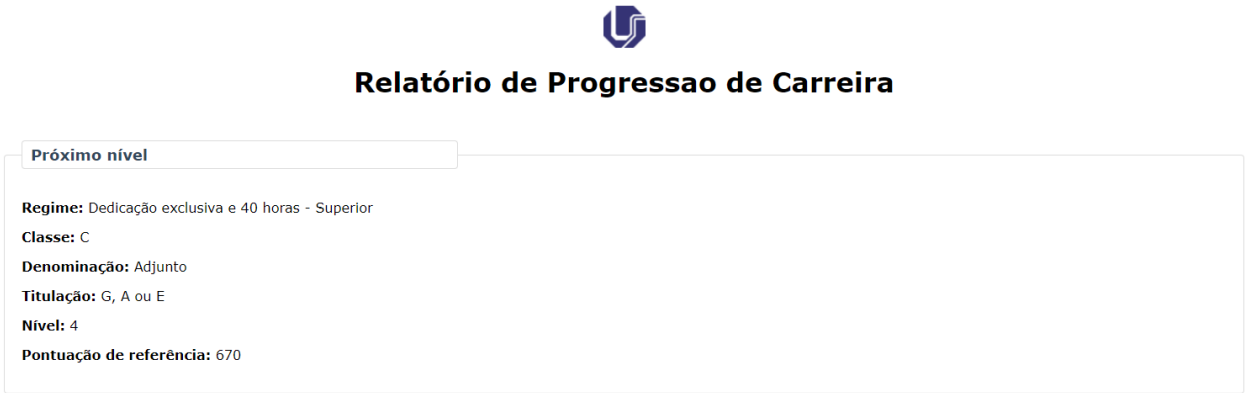

Figura 5 – Troca título relatório progressão.

Figura 6 – Troca título relatório promoção.

3.4 *Task 3* - Carregamento automático do próximo nível do novo relatório

Problema: Não era fácil preencher os campos do relatório para o nível pretendido, pois não era interativo para o usuário.

Para desenvolver a funcionalidade se fez necessário fazer uma chamada para a API para obter o objeto da tabela de progressões, já que a mesma contém todas as combinações possíveis para preencher os *dropdowns* que compõem a sessão de próximo nível.

Foi criado um método no typescript no arquivo filtro-tabela-progressao.component.ts. A estratégia abordada é a medida que o usuário for selecionando os *dropdowns* o sistema filtra as possíveis combinações para o seguinte *dropdow*, por exemplo, ao selecionar o regime, que representa a graduação do o professor, automaticamente será carregada as classes que estão disponíveis para aquele regime, e assim por diante.

Com isso, foi diminuído significativamente os possíveis erros ao preencher o relatório, já que o usuário poderá selecionar somente as opções disponíveis que na ponta tem o próximo nível pretendido, e aumenta a produtividade do usuário já que não será necessário fazer consultas adicionais para saber qual o nível pretendido. As Figuras 7 e 8 ilustram essa nova funcionalidade do sistema.

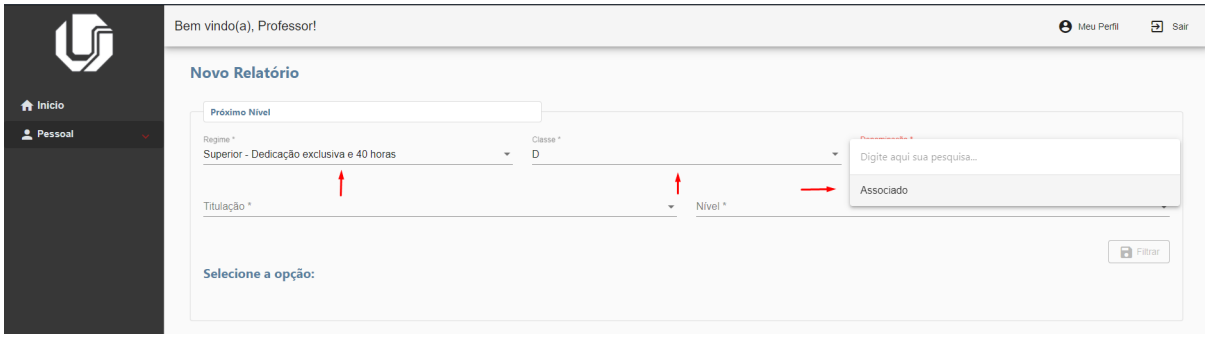

Figura 7 – Opções de denominação da carreira.

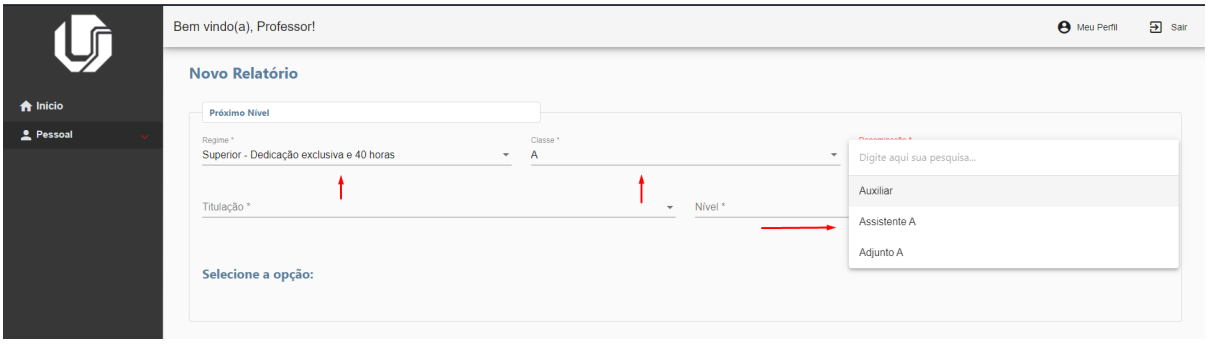

Figura 8 – Opções de denominação da carreira.

3.5 *Task 4* - Correção adicionar e editar atividade avaliador

Problema: Ao tentar alterar uma atividade no módulo de correção do avaliador apresentava um erro.

O requisito tem duas partes, um no *front-end* e outra no *back-end*, foram necessárias alterações em ambas partes.

No *back-end* a alteração necessária para o correto funcionamento, foi na função ImpedirInserirAtividades, que tem como finalidade dizer se uma pessoa pode ou não alterar uma atividade dependendo do status da atividade, podendo ser encerrado, em avaliação ou avaliado, como a atividade é em avaliação e utiliza outra tabela para persistir os dados foi acrescentado uma validação que se foi o avaliador que editou a atividade não cair na condição, evitando assim o erro no *back-end*.

No *front-end*, o primeiro ponto que foi corrigido foi na hora de decidir qual formulário preencher, pois existe o formulário de atividades e o de atividade avaliador, no qual ele não estava fazendo distinção. Foi alterado no método getById() uma condição que não deixava trazer o formulário correto.

Ao inicializar a tela de edição de atividade avaliador corretamente, depois da correção anterior, o construtor da atividade em avaliação estava incorreto, pois, não previa uma atividade correção nova, logo foi criado uma estrutura condicional nova que separa a inicialização, preparando as informações para o serviço de criar uma atividade avaliação e na outra condição uma estrutura que obtém as informações já existente de uma atividade em avaliação que se deseja editar novamente. Foi incluso em ambos os casos a inicialização da propriedade atividade.atividade_id que estava causando problemas quando chegava no *back-end*. O resultado é mostrado na Figura 9.

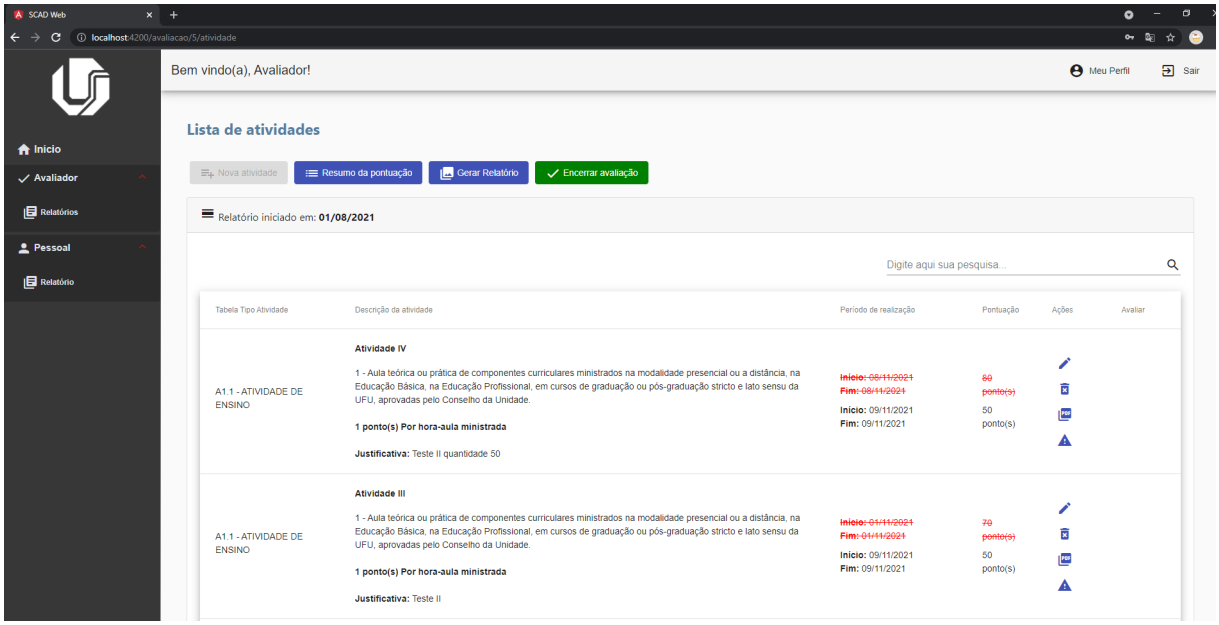

Figura 9 – Exemplo editar avaliador.

3.6 *Task 5* - Melhorias de funcionalidades

Nesta sessão serão abordadas alterações para atender pequenas funcionalidades para facilitar na usabilidade do sistema e ganhar mais agilidade ao criar os relatórios.

3.6.1 Função *auto complete* em data

Problema: Corrigir navegabilidade na tela de lançamento de atividades ao mudar entre a data início e a data final.

Levando em consideração a usabilidade de utilizar a tecla TAB para navegar entre os campos e preencher os próximos campos do formulário, o problema surgia quando se utilizava a tecla TAB da data inicial para a data final, em que a mesma perdia a formatação padrão de data e apresentava uma data inválida. Existe uma funcionalidade já implementada que é uma função no *typescript* que tem um *listener*, que é um ouvinte, uma estrutura que ouve os eventos previamente programados. O *listener* fica esperando alteração do campo da data inicial e copia a informação para o campo da data final, já que para a maioria das atividades essa data é sempre a mesma, ou seja, a atividade começa em um dia e termina no mesmo dia.

O que foi feito é alterar a função que contém o *listener*, criando uma propriedade de data do *typescript* que tem a formatação desejada e atribuindo ao campo da data final. Foi incluso também uma máscara nos campos de datas, que é um texto guia para ajudar o usuário a colocar a data no formato desejado, evitando assim entradas inválidas.

3.6.2 Acréscimo de informação no relatório final campo de quantidade horas da atividade

Problema: No relatório final não consta qual a quantidade de horas ou unidade de medida para a atividade, o que dificulta a correção do mesmo já que se tem que pesquisar por outros meios a informação.

Para atender o recurso foi necessário alterar o Linguagem de Marcação Extensiva (XML) que gera descrições, que tem todas as descrições genéricas que são utilizadas em todo o projeto. O mesmo se encontra no arquivo Endpoints.resx que faz uso de uma biblioteca de classes .NET que facilita trabalhar com XML. No arquivo foi adicionado um novo parâmetro contendo a informação da quantidade. Já na classe PrintService.cs que monta a estrutura do relatório e faz uso da estrutura do XML, foi adicionado o parâmetro utilizado que está dentro do objeto já utilizado atividade, que contém a quantidade que é a informação procurada conforme a Figura 10.

Tabela A1.1 - ATIVIDADE DE ENSINO:

Descrição da Atividade	Período de Realização	Pontuação	Documento
Atividade I 1 - Aula teórica ou prática de componentes curriculares ministrados na modalidade presencial ou a distância, na Educação Básica, na Educação Profissional, em cursos de graduação ou pós-graduação stricto e lato sensu da UFU, aprovadas pelo Conselho da Unidade. **1 ponto(s) Por hora-aula ministrada**	**Início:** 08/03/2021 **Fim:** 08/03/2021 Quantidade: 30	30,00 ponto(s)	tabA1.1_doc-1.pdf

Figura 10 – Resultado quantidade no relatório final.

3.6.3 Visualização de atividades por área

Problema: No módulo de lançamento de atividades não é possível visualizar qual é a área da atividade.

Na listagem de atividades não era possível ver qual era a área da atividade lançada no módulo. Para aumentar a qualidade das informações apresentadas ao usuário, foi criado uma coluna no *datatable* na tela listagem de atividades.

A alteração envolveu somente o *front-end*, visto que a informação necessária já se encontrava em uma classe existente. Para fazer a alteração foi necessário alteração no *framework* do Angular, já que ele está utilizando o *material*, que é uma biblioteca de componentes, bibliotecas, bibliotecas CSS e outros, que provê o componente *datatable*, utilizado para exibir as tabelas. Como se trata de um componente, a configuração foi simples, bastou acrescentar a nova propriedade e configurar o tamanho que a propriedade ocuparia no *grid* de exibição, conforme Figura 11.

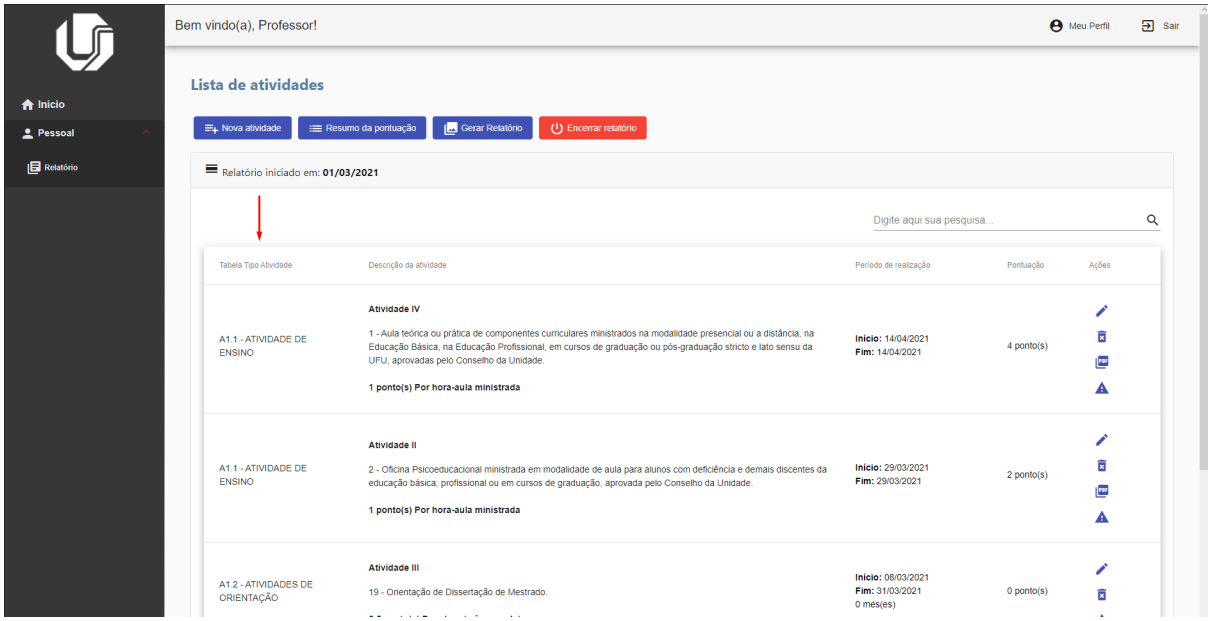

Figura 11 – Nova coluna tipo atividade

3.6.4 Criar campo *timestamp* na tabela de atividades e documentos

Problema: Não existe forma de consultar quando o usuário criou uma atividade ou adicionou um documento.

Para acompanhamento e solução dos problemas relatados pelos usuários do sistema, se fez necessário acrescentar na tabela de atividades/documentos uma coluna adicional que guarda a data e hora que foi feito o lançamento das atividades.

Seguindo a lógica do ORM, para o desenvolvimento da atividade foi necessário alterar código somente no *back-end* e como a estrutura do projeto utiliza o conceito de ORI, se fez necessário alterar algumas classes. Como o método utilizado é análogo tanto para documentos como atividades, será explicado para a tabela de atividades, porém o processo foi feito para as ambas tabelas.

Na classe Atividade foi acrescentada a propriedade DataCadastro que tem como tipo *DateTime*, foi utilizado um *DataAnotation DataMember* que específica que a propriedade faz parte de um contrato, podendo utilizá-lo para serializar posteriormente os dados. O próximo passo é acrescentar na classe AtividadeMap que é uma classe de mapeamento do *entity framework* a nova propriedade, para isso é utilizado uma função que associa a propriedade a um nome escolhido para ser o nome da coluna no banco de dados. No exemplo foi escolhido o nome data_cadastro.

Próximo passo é alterar a classe que efetivamente vai atribuir a propriedade a data e hora desejada, AtividadeService é a classe a ser alterada, pois quando adicionamos uma nova atividade é a classe responsável por prover o serviço de inserção no banco de dados. Atribuiremos a Data_cadastro o valor de *DateTime.Now* que é uma classe do C# que

nos retorna a data e hora no qual foi chamado a propriedade.

A estrutura do projeto facilita na hora que é necessário criar, alterar ou deletar tabelas no banco de dados, visto que é tudo feito nas classes e não se faz necessário ter contato direto com o banco de dados. Também se tem um versionamento do banco de dados, sendo simples voltar uma alteração que eventualmente não se obteve o resultado esperado. As alterações ficam intuitivas após entender a estrutura do projeto, fica fácil fazer alterações relacionadas a tabelas no banco de dados com a estrutura do projeto.

4 Conclusão

Conclui-se que os objetivos foram cumpridos. Como destaque, ganho para o usuário do SCAD, economia de tempo, qualidade das informações apresentadas e melhorias de usabilidade da interface.

A estrutura e organização do projeto, apresentou uma curva de aprendizado mediana, pois utiliza alguns conceitos de programação como DDD, ORI e faz uso de Angular, que não é simples de entender, pois, possuem uma estrutura complexa. Após o entendimento de como funciona cada parte, no *back-end* e *front-end*, o cenário fica favorável, pois, fica rápido e fácil o desenvolvimento, visto que a estrutura visa a facilidade de entendimento das regras de negócio e promove uma comunicação baseada no negócio e facilita na organização e estruturação do código.

Para trabalhos futuros, visando a melhoria do projeto foram levantandos alguns pontos e dificuldades apresentadas durante o desenvolvimento do projeto.

- **DevOps:** A necessidade foi percebida pelo modo complexo de submeter para o servidor da UFU os arquivos do projeto, sendo um trabalho manual e que pode acontecer erros humanos. Com o DevOps a intenção é automatizar todo o processo, a partir dos *commits* já fazer refletir no servidor da UFU as alterações ou um servidor de testes.
- **Testes automatizados:** Um problema que foi constatado durante o desenvolvimento é a necessidade de testar outras partes do código sem ser somente a parte específica do que está sendo alterado, visto que uma pequena alteração pode quebrar ou parar de trazer os resultados esperados quando existe a modificação de alguma parte do código. Com os testes automatizados iniciaria a cobertura dos módulos para garantir seu funcionamento antes de fazer o *deploy* do sistema, alertando o desenvolvedor que existe alguma coisa errada, não esperada após seu *commit*, evitando possíveis problemas em produção e transtorno aos usuários do SCAD.

Referências

ANGULAR. *The modern web developer's platform*. 2021. Disponível em: <https://angular.io>. Citado na página 12.

ATLASSIAN. *O que é Git*. 2021. Disponível em: <https://www.atlassian.com/br/git/tutorials/what-is-git>. Citado na página 14.

BOOTSTRAP. *Build fast, responsive sites with Bootstrap*. 2021. Disponível em: <https://getbootstrap.com>. Citado na página 12.

CONDIR. *RESOLUÇÃO No 03/2017, do Conselho Diretor da Universidade Federal de Uberlândia*. 2017. Citado na página 9.

EVANS, E. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. [S.l.]: Addison-Wesley, 2004. Citado na página 12.

MARINESCU, A. A. F. Domain-driven desing quickly. 2007. Citado na página 13.

MICROSOFT. *Documentação do C#*. 2021. Disponível em: <https://docs.microsoft.com/pt-br/dotnet/csharp/>. Citado na página 11.

MICROSOFT. *Documentação do Entity Framework*. 2021. Disponível em: <https://docs.microsoft.com/pt-br/ef/>. Citado na página 12.

MICROSOFT. *Introdução ao .NET*. 2021. Disponível em: <https://docs.microsoft.com/pt-br/dotnet/core/introduction>. Citado na página 11.

PAIXAO, J. R. da. *O que é SOLID*. 2019. Disponível em: <https://medium.com/desenvolvendo-com-paixao/o-que-é-solid-o-guia-completo-para-você-entender-os-5-princípios-da-poo-2b937b3fc530>. Citado na página 13.

POSTGRESQL. *Documentation PostgreSQL*. 2021. Disponível em: <https://www.postgresql.org/docs/>. Citado na página 12.

SILVA, W. S. *Sistema de Cadastro de Atividades Docente*. 2019. Disponível em: <https://repositorio.ufu.br/handle/123456789/28962>. Citado na página 15.

TOTVS, E. *Por que o Angular é um framework tão poderoso?* 2020. Disponível em: <https://www.totvs.com/blog/developers/angular/>. Citado na página 12.

XIA, C.; YU, G.; TANG, M. *Efficient Implement of ORM (Object/Relational Mapping) Use in J2EE Framework: Hibernate*. 2009. 1-3 p. Citado na página 14.