UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Luiz Fernando Rosa

Desenvolvimento de sistema para controle de projeto de extensão

Uberlândia, Brasil

2023

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Luiz Fernando Rosa

Desenvolvimento de sistema para controle de projeto de extensão

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título 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

2023

Resumo

Este trabalho contempla o desenvolvimento de um sistema para auxiliar a Faculdade de Computação no registro e controle de atividades de extensão decorrentes da reforma curricular que introduziu esse tipo de atividade nos cursos de graduação da faculdade. Nele será abordado brevemente o problema em relação ao registro de atividades de extensão, em que as entidades da UFU não dispõe da ferramenta para controle dessas específicas para os cursos de graduação. Além disso, também será abordado o processo relacionado a construção de um sistema, com isso discorrendo sobre a arquitetura, tecnologia e a metodologia em cima da criação do mesmo.

Palavras-chave: Sistema de extensão, Graduação, discentes, testes.

Lista de abreviaturas e siglas

UFU	Universidade Federal de Uberlândia
FACOM	Faculdade de Computação
API	Interface de programação de aplicações (*Application Programming Interface*)
ORM	Mapeamento objeto relacional (*Object Relational Mapping*)
POO	Programação Orientada a Objeto
SGBD	Sistema Gerenciador de Banco de Dados
LINQ	Consulta Integrada à Linguagem (*Language Integrated Query*)
SIEX	Sistema de Informação de Extensão
SPA	Aplicações de Página Única (*Single Page Applications*)
DOM	*Document Object Model*

Lista de ilustrações

Sumário

1 Introdução

A extensão universitária é uma atividade importante que faz parte do tripé universitário (Ensino, Pesquisa e Extensão). Ela possibilita o compartilhamento com o público externo do conhecimento produzido por meio do ensino e da pesquisa desenvolvidas nas Universidades. Apesar da sua importância, na Faculdade de Computação (FACOM) da Universidade Federal de Uberlândia, a extensão é a que tem menor participação dentre as atividades docentes e discentes.

A partir do ano de 2023 os projetos pedagógicos dos cursos de graduação da FACOM passaram a ter uma carga de atividades extensão equivalentes a 10% do curso. Com isso, será necessário o registro individualizado de cada atividade realizada por cada aluno da faculdade. Atualmente não há ferramenta informatizada na universidade que permita fazer esse acompanhamento. Apesar de a Pró-reitoria de Extensão manter o registro de todas as atividades de extensão exercidas na universidade, o sistema atual que registra essas informações não está disponível para consultas que permitam identificar facilmente as atividades realizadas por alunos e que atendam requisitos específicos dos projetos pedagógicos dos cursos.

Dessa forma, este trabalho propõe o desenvolvimento do SEFAC – Sistema de Extensão da FACOM, de forma a facilitar o gerenciamento da atividades de extensão realizadas pelos discentes dos cursos de graduação da FACOM.

1.1 Objetivos

O objetivo deste trabalho é a criação de um sistema para registro de atividades de extensão realizados por discentes dos cursos de graduação da FACOM.

1.2 Justificativa

Como serão realizados várias atividades de extensão durante todo o curso, as entidades da UFU não estão preparadas para realizar a gestão de grande volume de informação, haja vista que as atividades contém uma carga horária demasiadamente extensa e com diversos tipos de atividades. Com isso surgiu a necessidade de criação de um sistema para acompanhamento da mesma.

2 Referencial teórico

Esta seção apresenta as tecnologias que foram utilizadas no desenvolvimento do SEFAC. O sistema foi divido em duas partes, a primeira sendo o *back end* e a segunda o *front end*. O *back end* foi construído em C#, .NET 6. O *front end* foi construindo usando o framework Angular na sua versão mais recente (versão 9). O sistema provê serviços via Interface de Programação de Aplicações (API) RESTful. A parte de banco de dados é feita pelo Posrgres, também é utilizado o conceito de *Object Relational Mapping* (ORM) utilizando *Entity Framework* para persistência. Para o controle de fontes foi utilizado git. Cada uma das tecnologias listadas neste parágrafo serão detalhadas abaixo.

2.1 C#

C# é uma linguagem de programação orientada a objetos, fortemente tipada desenvolvida pela *Microsoft*, (MICROSOFT., 2022a) para a plataforma .NET. Ela é amplamente utilizada para desenvolvimento de aplicativos para *Windows*, bem como para desenvolvimento de jogos e aplicativos para dispositivos móveis. É uma linguagem de programação moderna, com recursos como gerenciamento automático de memória e sobrecarga de operadores.

2.2 Swagger

Swagger é um conjunto de ferramentas para projetar, construir, documentar e consumir serviços da *Web RESTful*. Ele inclui um formato de especificação, chamado *OpenAPI Specification* (anteriormente conhecido como *Swagger Specification*), e várias ferramentas que suportam esse formato. O objetivo principal do *Swagger* é facilitar a comunicação e a interação com serviços da *Web RESTful*, fornecendo uma documentação clara e padronizada.

2.3 .Net

.NET é uma plataforma de desenvolvimento de aplicativos da *Microsoft* que permite a criação de aplicativos para Windows, dispositivos móveis, web e nuvem(MICROSOFT., 2022c). Ele fornece uma série de ferramentas e bibliotecas para ajudar os desenvolvedores a criar aplicativos de alta qualidade com rapidez e facilidade. A plataforma inclui a *Common Language Runtime* (CLR), que é o ambiente de execução que gerencia a memória

e as exceções, e a *Framework Class Library* (FCL), que é uma coleção de classes e tipos que fornecem funcionalidade de sistema e de negócios comuns .

2.4 *Entity Framework*

Entity Framework (EF) é um *framework* de mapeamento objeto-relacional (ORM) para .NET desenvolvido pela Microsoft. Ele permite aos desenvolvedores trabalhar com banco de dados relacionais usando objetos do mundo real em sua aplicação, em vez de trabalhar diretamente com tabelas e consultas SQL(MICROSOFT., 2022b). Isso significa que os desenvolvedores podem escrever códigos em sua linguagem de programação de escolha, e o EF cuidará de traduzir essas operações para consultas SQL apropriadas para o banco de dados. Ele tem suporte para múltiplos bancos de dados como SQL Server, MySQL, SQLite e PostgreSQL .

2.5 Angular

Angular é um *framework* de desenvolvimento de aplicativos web desenvolvido e mantido pela *Google*. Ele é baseado em *JavaScript* e construido em cima de *Typescript* (ANGULAR., 2022) e usa um modelo de programação baseado em componentes, o que permite aos desenvolvedores criar aplicativos web de forma modular e escalável. Ele é projetado para ajudar os desenvolvedores a construir aplicativos web de alto desempenho e reativos, com suporte a rotas, componentes, diretivas, serviços e outros recursos. Além disso, oferece recursos como *data binding* bidirecional, injeção de dependência e suporte a testes automatizados. Angular é uma plataforma *JavaScript* completa para construir aplicações web complexas.

Com o Angular, temos um novo paradigma de desenvolvimento focado nos dados da aplicação. Ele não utiliza uma virtualização do DOM para manipulá-lo: ele utiliza mecanismos próprios de detecção de alterações na interface, alterações tas disparadas principalmente por uma estrutura chamada *Two-Way Data Binding* .

2.6 PostgreSql

PostgreSQL é um sistema gerenciador de banco de dados relacional de código aberto, desenvolvido na Universidade da Califórnia no Departamento de Ciências da Computação em Berkeley,. Ele permite a criação, administração e interação com bancos de dados usando a linguagem SQL (*Structured Query Language*). Ele é compatível com várias plataformas, incluindo *Windows* e *Linux*.

2.7 *Migrations*

Migrations são um conjunto de *scripts* ou comandos que facilitam a criação ou modificação de um banco de dados, com elas é possível ter um versionamento das mudanças no banco de dados, consistência entre ambientes, uma evolução do esquema do banco de dados e uma automatização. Nesse projeto foi criado *migrations* a partir das ferramentas disponibilizadas pelo o *entity framework*.

2.8 JWT *(Json Web Token)*

JWT (JSON Web Token) é um padrão aberto que define uma maneira compacta e autossuficiente para representar informações entre duas partes. Essas informações podem ser verificadas e confiáveis, pois são assinadas digitalmente (JONES; BRADLEY; SAKIMURA, 2023). *JWTs* são frequentemente utilizados para autenticação e troca segura de informações entre diferentes partes de uma aplicação web. O mesmo possui uma chave pública e privada aonde com a chave pública pode se ver oque contém dentro de um *token* mas não se pode alterar o mesmo sem a chave privada caso ao contrario o *token* deixa de ser válido

2.9 *Clean architecture*

Clean Architecture é um conjunto de princípios e diretrizes para projetar sistemas de software de forma modular e escalável. Ele foi proposto por Robert C. Martin e se baseia em três camadas principais: a camada da interface do usuário, a camada de negócios e a camada de infraestrutura. A ideia é que cada camada dependa apenas das camadas internas e não das camadas externas, o que permite que as mudanças em uma camada não afetem as outras. Isso ajuda a manter o sistema limpo, fácil de entender e fácil de testar.

2.9.1 Camada de interface

A camada de interface do usuário é a camada mais externa da arquitetura limpa e é responsável por gerenciar a interação do usuário com o sistema. Ela pode ser implementada como uma interface gráfica, uma API REST ou qualquer outra forma de interface que permita que os usuários acessem e interajam com o sistema. A camada de interface do usuário depende apenas da camada de negócios e não da camada de infraestrutura. Ela é responsável por traduzir as ações do usuário em ações que a camada de negócios entenda e por fornecer ao usuário uma resposta para suas ações.

2.9.2 Camada de negócios

A camada de negócios, também conhecida como camada de domínio, é a camada intermediária da arquitetura limpa. Ela é responsável por representar e gerenciar as regras de negócios do sistema, tais como validações, cálculos e lógica de processamento. Ela é independente da interface do usuário e da infraestrutura, o que permite que ela seja facilmente reutilizada em diferentes contextos. A camada de negócios é onde a lógica de negócios do sistema é implementada, e é responsável por decidir como os dados devem ser manipulados e armazenados. Ela depende apenas das camadas internas e não das camadas externas, o que ajuda a garantir a independência entre as camadas e a facilidade de manutenção e teste.

2.9.3 Camada de infraestrutura

A camada de infraestrutura é a camada mais interna da arquitetura limpa e é responsável por gerenciar as dependências do sistema, tais como banco de dados, arquivos e serviços externos. Ela é responsável por fornecer às camadas superiores uma forma de acessar esses recursos de forma abstrata. A camada de infraestrutura é onde as implementações específicas das tecnologias de baixo nível são realizadas, como acesso ao banco de dados, acesso ao sistema de arquivos, acesso à rede, etc. Ela é independente da camada de negócios e da interface do usuário, o que permite que ela seja facilmente reutilizada e substituída por outras implementações sem afetar as camadas superiores.

2.9.3.1 Padrão *Repository*

O padrão *Repository* é um padrão de projeto de software que permite abstrair a lógica de acesso a dados de uma aplicação. Ele é comumente usado em aplicações que utilizam bancos de dados relacionais ou outras fontes de dados. Ele é implementado como uma interface que fornece métodos para realizar operações CRUD (*Create, Read, Update, Delete*) em um determinado tipo de entidade. Esses métodos podem ser implementados usando diferentes tecnologias de banco de dados ou fontes de dados, sem afetar a camada de negócios ou a camada de interface do usuário.

Em C#, o padrão *Repository* pode ser implementado como uma interface que é implementada por uma classe concreta. A interface pode ser definida com métodos como *Add, Update, Remove, Get, GetAll,* etc. e é implementada pela classe concreta usando uma tecnologia específica para acessar a fonte de dados (por exemplo, Entity Framework, Dapper, etc.). Isso permite que a camada de negócios use a interface do repositório sem se preocupar com as implementações específicas de banco de dados.

2.10 Trabalhos Correlatos

Dentre os sistema mais próximo ao proposto atualmente é o da SIEX, gerenciado pela PROEX (Pró-Reitoria de Extensão)(SIEX., 2022). O mesmo contém catálogo online, apresentação de certificados online, além de registro de ações de assuntos estudantis, ações de extensão e cultura, junto também com a validação de certificados. Apesar de ser simples, apresenta um design de UI um pouco antigo.

3 Desenvolvimento

3.1 Metodologia de desenvolvimento XP

Para realizar o desenvolvimento do sistema junto com seus requisitos funcionais foi utilizado a metodologia XP (*Extreme Programming* – Programação Extrema).

A metodologia XP busca promover a satisfação do cliente e a entrega de software de alta qualidade por meio de iterações curtas e frequentes, *feedback* constante e desenvolvimento incremental. Durante o desenvolvimento foi abordado quatro grande passos, sendo esses:

Planejamento: Durante esse passo foi enfatizada a comunicação entre o desenvolvedor e os stakeholders do projeto para garantir que todos entendam os requisitos e estejam alinhados em relação aos objetivos do projeto.

Projeto: Foi elaborado uma arquitetura, tendo em vista os requisitos funcionais e também as definições de metas e prazos.

Codificação: Foi realizado a parte de codificação utilizando de recursos como a IDE da Microsoft Visual Studio, que a mesma é configurada para os ambientes de desenvolvimento utilizando a linguagem C#, Visual Studio Code, uma ferramenta de edição de texto utilizado no desenvolvimento em relação ao restante das tecnologias Angular e PostgreSQL.

Acompanhamento e adaptação: Durante a execução da iteração foi realizado reuniões semanais de acompanhamento. Nessas reuniões curtas, foi compartilhado o que fizeram durante o desenvolvimento anterior, o que planejam fazer durante a semana e se têm algum obstáculo. Além de verificar se está alinhado com as metas e prazos definidos.

3.2 *Front end*

O desenvolvimento da seção de *front end* foi como ponto de partida o projeto SCAD (REIS, 2023) em que o mesmo utiliza Angular como *framework* de desenvolvimento, com isso foi utilizado o padrão estético foi alicerceado no mesmo, mas a tecnologia e versão foram modificadas como por exemplo a própria versão do angular aonde *SCAD* se utiliza da versão 7.11 o *SEFAC* foi utilizado a versão 14.2, mesmo com sua estética parecida o *SEFAC* utiliza tanto de lógica e como ferramentas diferentes para realização de *forms*, manipulação de texto e datas por exemplo.

3.3 *Back End*

O desenvolvimento da secção de *back end* foi criado utilizando tecnologias já citadas posteriormente como C#, *Entity framework* e com a arquitetura *Clean Architeture* e também foi usado *migrations* para facilitar na criação e modificação do banco.

3.4 Organização arquitetural

Como dito previamente, foi usado o padrão *Clean Architeture*. Nele foram criados quatro projetos separados que são responsáveis por atribuições diferentes.

3.4.1 API

A camada de API lida com a exposição de funcionalidades do sistema por meio de interfaces de programação de aplicativos (APIs), que foi consumido pelo o *front end*. Nesse caso foi usado Swagger (Figura 4) para ajudar na documentação.

No projeto foi desenvolvido classes para configuração do *Auto Mapper*, uma biblioteca do C#, configuração do *CORS* , *jwt* e *swagger*, criação de classes para os *end-points* da aplicação, e a parte em *json* para configuração de váriaveis de ambiente e *secrets* do projeto.

3.4.1.1 Configuração do CORS

O CORS desse projeto foi configurado de tal maneira que somente a *API Front End* Angular consiga acessar os métodos com a notação textitAuthorize na rota vista na Figura 2.

3.4.1.2 JWT e autorização

A aplicação se utiliza de *jwt* para a autenticação. Por ser uma maneira fácil e segura para a autenticação de uma *API*, ele foi configurado a conter as *roles* do usuário, seu identificar de e-mail, identificador único do *jwt*, a data de expiração, o emissor do *token* e seu destinatário. Um exemplo de arquivo *jwt* criado é mostrado na Figura 3.

3.4.1.3 *Swagger*

Com o projeto *back-end* foi utilizado a ferramenta *swagger* para documentação dos *end-points* veja a figura 4 do resultado final.

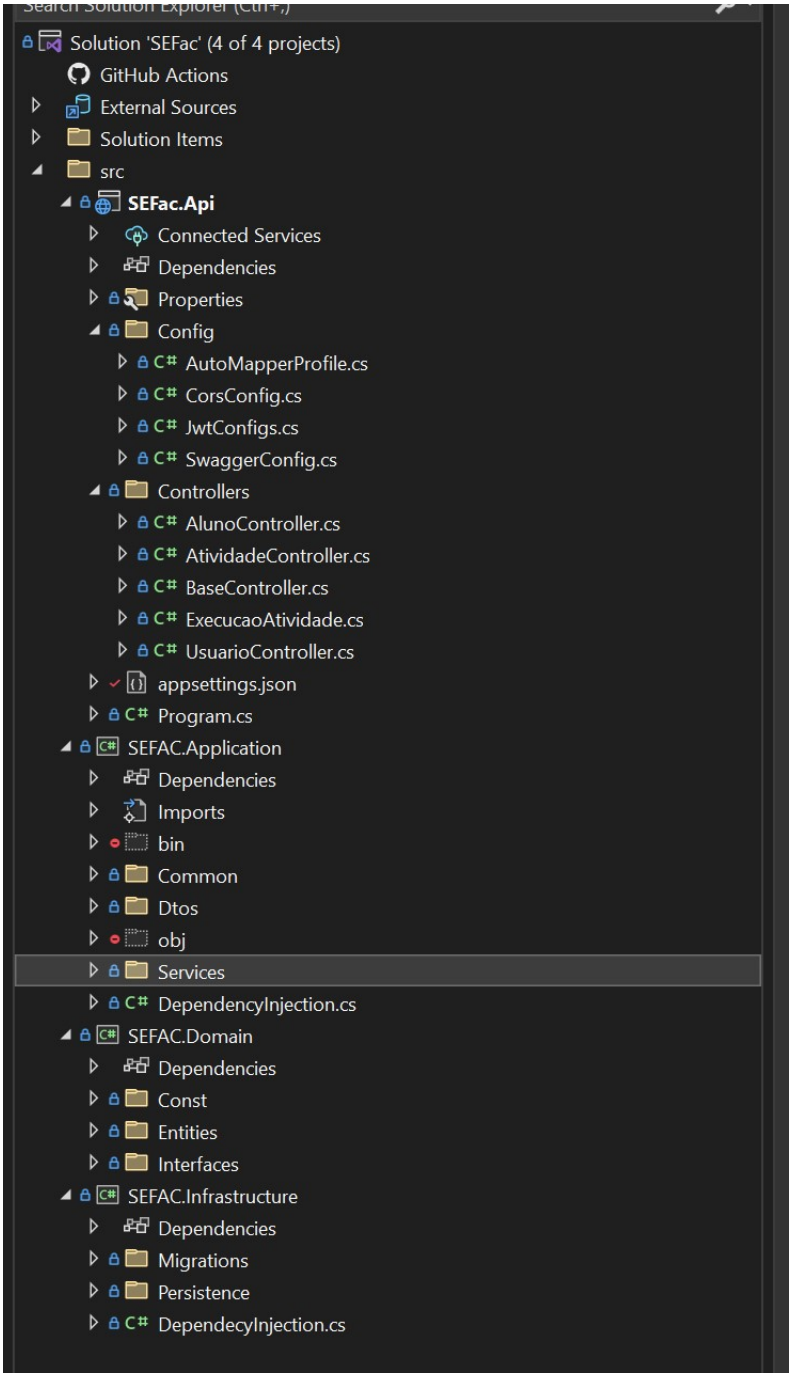

Figura 1 – Representação da arquitetura.

3.4.2 SEFAC.Application

É a camada responsável pela lógica do negócio, ela também é responsável por inverter dados da camada de interface do usuário para formas compreensíveis pela camada de domínio e vice-versa.

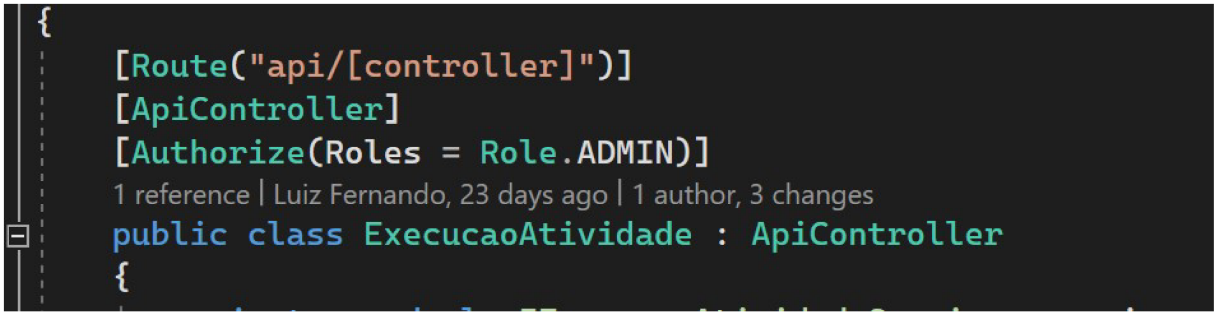

Figura 2 – Notação de autorização dada pelo *framework* .Net.

Figura 3 – Exemplo do *jwt* criado.

3.4.3 SEFAC.Domain

Ela representa as entidades centrais e as operações que são específicas do domínio da aplicação, como interfaces que a camada de *repository* deve seguir além de constantes necessárias para as *roles* de *autorization* e *policy* para o *CORS* veja a figura 5 de exemplo.

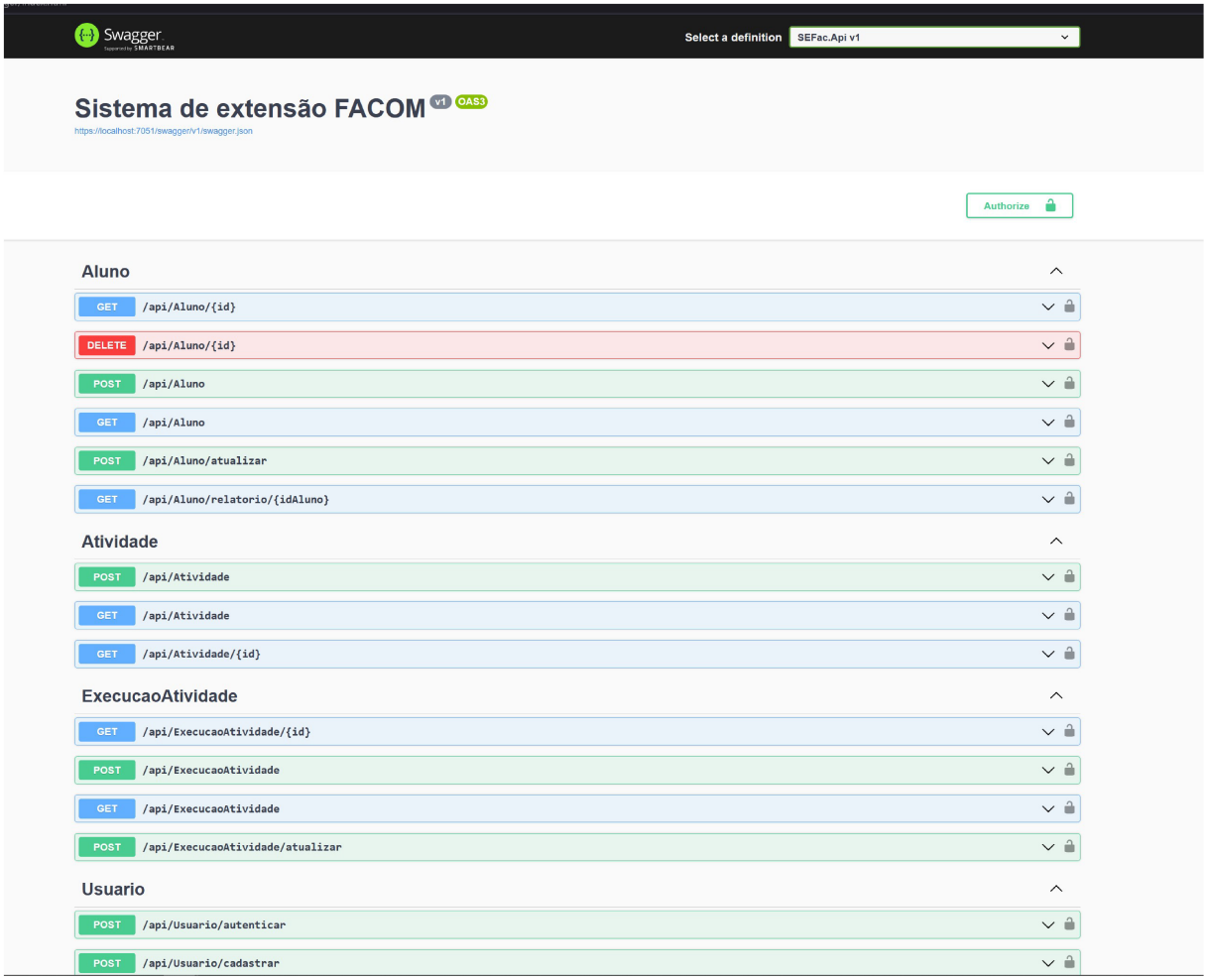

Figura 4 – Swagger

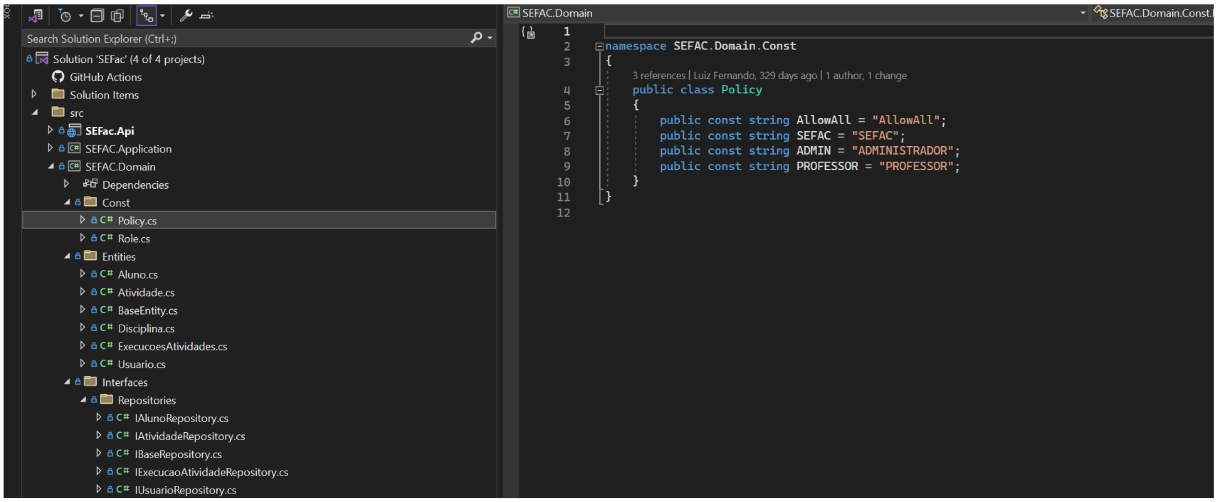

Figura 5 – Como foi realizado a estruturação da camada *domain* no projeto.

```
0 references | Luiz Fernando, 26 days ago | 1 author, 2 changes
public class AlunoConfiguration : IEntityTypeConfiguration<Aluno>
{
    0 references | Luiz Fernando, 26 days ago | 1 author, 2 changes
    public void Configure(EntityTypeBuilder<Aluno> builder)
    {
        builder.ToTable("Aluno").HasKey(x => x.Id);

        builder.HasIndex(x => x.NumeroMatricula).IsUnique();

        builder.Property(x => x.Id).ValueGeneratedOnAdd();

        builder.Property(x => x.CodigoCurso)
            .HasColumnName("Cod_Curso")
            .IsRequired();

        builder.Property(x => x.Nome)
            .HasColumnName("Nome")
            .IsRequired();

        builder.Property(x => x.NumeroMatricula)
            .HasColumnName("Nr_Matricula")
            .IsRequired();

        builder.HasMany(x => x.ExecucaoAtividades)
                .WithOne(x => x.Aluno)
                .HasForeignKey(x => x.IdAluno)
                .OnDelete(DeleteBehavior.SetNull);
    }
}
```

Figura 6 – Exemplo de configuração na camada de *repository* para auxiliar nas *migrations*

3.4.4 SEFAC.Infrastructure

É responsável por lidar com os detalhes de implementação externos e mecanismos de suporte necessários para que a aplicação funcione. No caso a parte de *framework* para a conexão no banco de dados.

No projeto ela foi pensada para que facilite a troca de um gerenciador de banco de dados de uma forma que não impacte o restante do código, que por sua vez isso foi necessário que durante o projeto foi preciso trocar de *SQL Server* para *postgreSQL* haja vista o custo da licença para subir um servidor além de ser um gerenciador já utilizados por vários alunos durante a faculdade oque permite uma maior facilidade para a manutenção de código.

Nessa camada também é responsável pelo mapeamento das entidades as quais são necessárias para a realização de *migrations*, esse mapeamento é feito de forma manual, aonde se declara o atributo e como ele deve ser mapeado, por exemplo o atributo *ID* é um valor gerado ao adicionar, um atributo *NumeroMatricula* é único e que essa mesma entidade tem relação com outra entidade *ExecucaoAtividades*, que é baseado no *IdAluno* dentro da classe *ExecucaoAtividades* veja a figura 6.

```
        /// <inheritdoc />
        0 references | Luiz Fernando, 26 days ago | 1 author, 1 change
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropForeignKey(
                name: "FK_ExecucaoAtividade_Aluno_AlunoId",
                table: "ExecucaoAtividade");

            migrationBuilder.RenameColumn(
                name: "AlunoId",
                table: "ExecucaoAtividade",
                newName: "IdAluno");

            migrationBuilder.RenameIndex(
                name: "IX_ExecucaoAtividade_AlunoId",
                table: "ExecucaoAtividade",
                newName: "IX_ExecucaoAtividade_IdAluno");

            migrationBuilder.AddForeignKey(
                name: "FK_ExecucaoAtividade_Aluno_IdAluno",
                table: "ExecucaoAtividade",
                column: "IdAluno",
                principalTable: "Aluno",
                principalColumn: "Id",
                onDelete: ReferentialAction.SetNull);
        }

        /// <inheritdoc />
        0 references | Luiz Fernando, 26 days ago | 1 author, 1 change
        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropForeignKey(
                name: "FK_ExecucaoAtividade_Aluno_IdAluno",
                table: "ExecucaoAtividade");

            migrationBuilder.RenameColumn(
                name: "IdAluno",
                table: "ExecucaoAtividade",
                newName: "AlunoId");

            migrationBuilder.RenameIndex(
                name: "IX_ExecucaoAtividade_IdAluno",
                table: "ExecucaoAtividade",
                newName: "IX_ExecucaoAtividade_AlunoId");

            migrationBuilder.AddForeignKey(
                name: "FK_ExecucaoAtividade_Aluno_AlunoId",
                table: "ExecucaoAtividade",
                column: "AlunoId",
                principalTable: "Aluno",
                principalColumn: "Id",
                onDelete: ReferentialAction.Cascade);
        }
    }
```

Figura 7 – Exemplo do código gerado pelo *framework entity framework* para a criação de *migrations*

3.4.4.1 *Migrations*

Com o uso do padrão *repository* e o *framework entity framework*, foi possível utilizar o método *code-first*, aonde que por meio de configurações e criações de classes que também são conhecidas por *entites* e o mapeamento de relações entre elas, foi possível criar *scripts* de criação, *update*, *alter table* e etc do banco de dados somente com uma linha de comando (MICROSOFT,). Com base na figura 8 é possível ver como é o histórico das *migrations* e na figura 7 podemos ver que o código gerado tem duas partes majoritárias um método *UP* e um método *DOWN*, o primeiro serve para dar persistência do código dentro do banco de dados o segundo é para casos de *rollback*

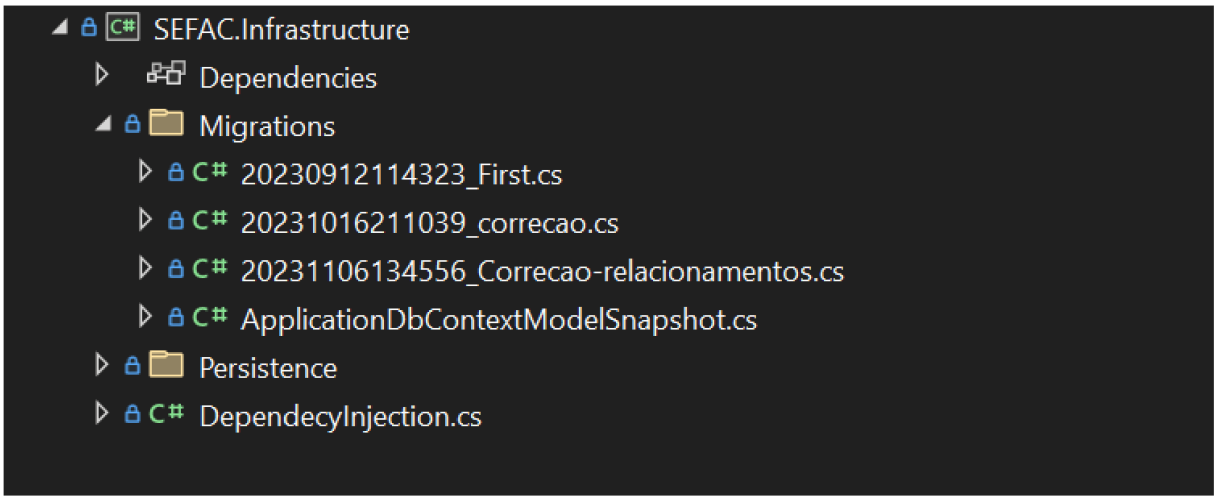

Figura 8 – Estruta de histórico realizado pelas *migrations*

4 Resultados Obtidos

4.1 Tela inicial

4.1.1 Tela de login

Foi criado um sistema de login baseado em usuário e senha criado pelo administrador, futuramente será integrado com o sistema de login da UFU.

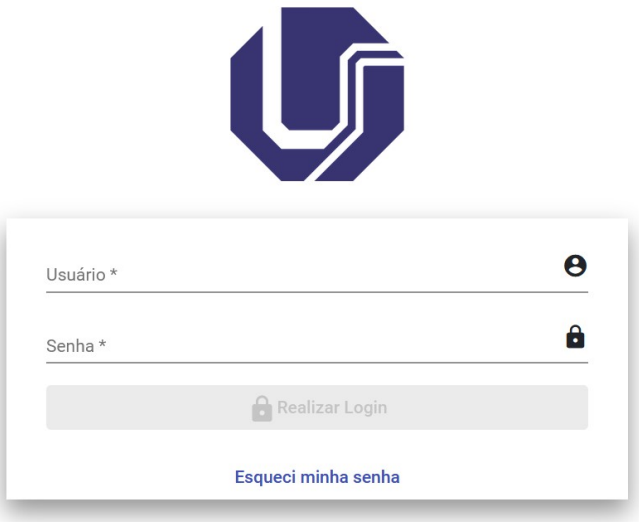

Figura 9 – Tela de login

4.1.2 Tela principal

Após realizar o login, será redirecionado para uma tela principal aonde os menus ficam na lateral esquerda e opções de perfil e logout na parte superior a direta.

4.2 CRUD

4.2.1 Cadastro de aluno

Foi criado um cadastro de aluno aonde e necessários os seguintes dados:

- Nome do aluno

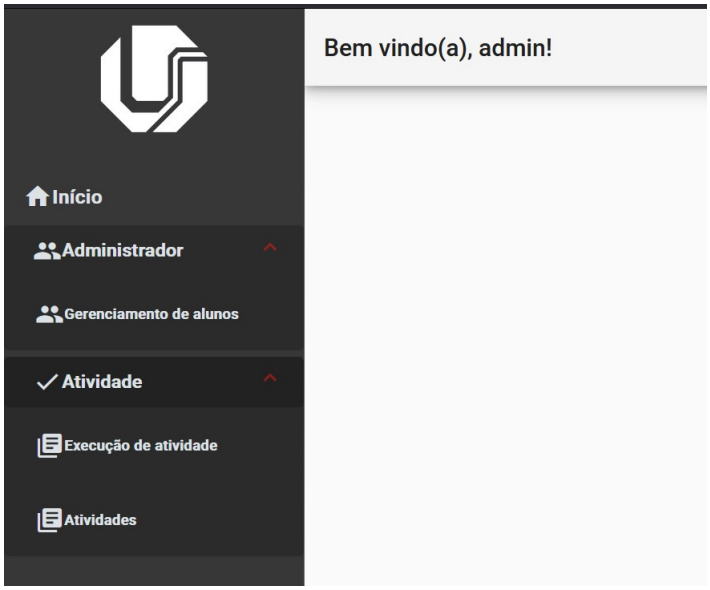

Figura 10 – Tela principal

- Número da sua matrícula
- Código curso

Figura 11 – Cadastro de aluno.

4.2.1.1 Lista de alunos

Foi criado uma página para mostrar todos os alunos já cadastrados, podendo assim também alterar certos dados do mesmo ou excluir um aluno cadastrado

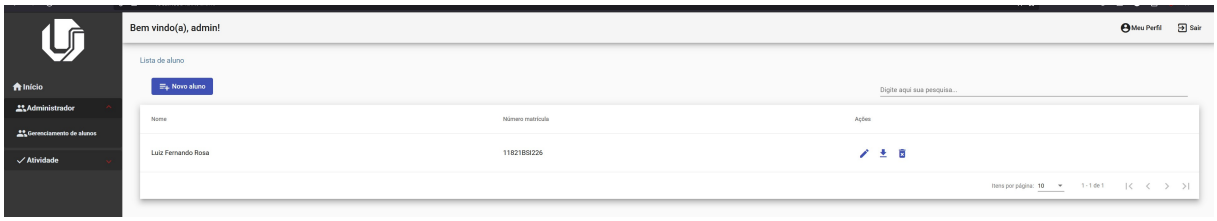

Figura 12 – Lista de alunos.

4.3 Atividade

4.3.1 Cadastro de atividade

Foi criado um cadastro de atividade sendo preciso preencher os seguintes dados

- Código Siex
- Descrição da atividade (opcional)
- Upload de arquivo (opcional)

Figura 13 – Cadastro de atividade.

4.3.2 Lista de atividades

Também foi criado uma lista para citar as atividades já cadastradas.

4.4 Execução de atividades

4.4.1 Cadastro de execução de atividade

Foi criado uma cadastro para atrelar as atividades a suas devidas execuções, nele é necessário os seguintes dados:

- O nome da execução
- A data início e data fim
- O aluno (já devidamente cadastrado no portal)
- Carga horária
- Duração
- Atividade (já devidamente cadastrada no portal)

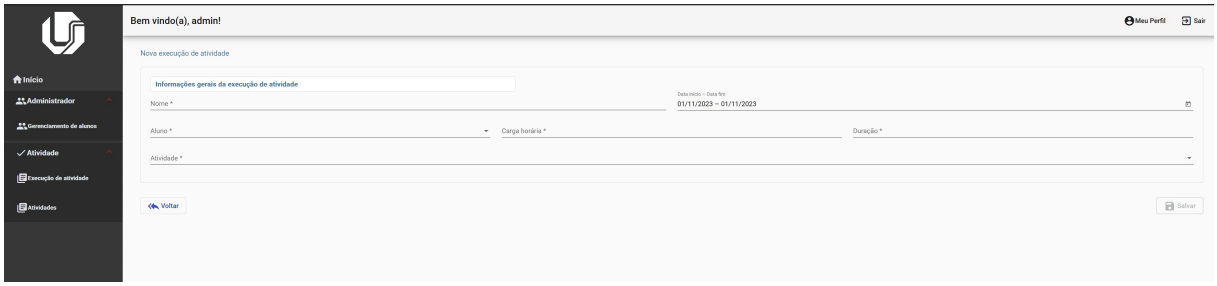

Figura 14 – Cadastro de execução de atividade

5 Conclusão

Neste trabalho, foi desenvolvido um sistema com objetivo de facilitar o cadastro de projetos de extensão, haja vista que o novo currículo do curso de Graduação em Sistemas de Informação do campus Santa Mônica, tem uma quantidade de horas obrigatórias, com isso o sistema tem como principal metas armazenar de forma fácil fornecendo uma interface simples e intuitiva ao usuário.

O desenvolvimento do sistema teve como maior prioridade os alicerces do mesmo, a parte arquitetural tanto *front-end* como *back-end*, ele permite o cadastro de usuários, atividades, e execuções das mesmas e também um cadastro de disciplinas.

Para trabalhos futuros, visando a melhoria do sistema foram levantados alguns requisitos:

- **Sistema de *login* unificado com a UFU:** Atualmente o sistema de login e feito e gerenciado pela própria aplicação, futuramente deve-se integrar com o sistema de login da UFU para um melhor ambiente integrado.
- ***DevOps:*** Atualmente para subir uma versão no ar é necessário utilizar os servidores da UFU e subir o código de forma manualmente, com isso além de ser necessário um grande tempo para fazer isso, também pode ocorrer de erros humanos. Com um processo automatizado além de evitar isso, pode se mostrar um *MVP* de forma muito mais rápida.

Referências

ANGULAR. **Documentação do Angular**. 2022. Disponível em : <https://angular.io/guide/what-is-angular>. Citado na página 9.

JONES, M. B.; BRADLEY, J.; SAKIMURA, N. **RFC 7519 JSON WEB TOKEN**. 2023. Disponível em: <https://datatracker.ietf.org/doc/html/rfc7519>. Citado na página 10.

MICROSOFT. **Migrations Overview.** Disponível em : <https://learn.microsoft.com/en-us/ef/core/managing-schemas/migrations/?tabs=dotnet-core-cli>. Citado na página 19.

MICROSOFT. **Documentação do C#.** 2022. Disponível em : <https://learn.microsoft.com/pt-br/dotnet/csharp/tour-of-csharp/>. Citado na página 8.

_____. **Documentação do Entity Framework**. 2022. Disponível em : <https://docs.microsoft.com/pt-br/ef/>. Citado na página 9.

_____. **Documentação do .Net.** 2022. Disponível em : <https://dotnet.microsoft.com/pt-br/learn/dotnet/what-is-dotnet>. Citado na página 8.

REIS, M. P. **Sistema de Informação para Gerenciamento de Progressão e Promoção Funcional**. 2023. Disponível em: <https://repositorio.ufu.br/bitstream/123456789/33841/1/SistemaInforma%c3%a7%c3%a3oPara.pdf>. Citado na página 13.

SIEX. **Sistema de informação de extensão**. 2022. Disponível em : <http://www.siex.proex.ufu.br/>. Citado na página 12.