universidade de aveiro

Base de dados
Licenciatura em Engenharia Informática
Ano Letivo 2020/2021 | 2.º Ano, 2.º Semestre

Be Your Own Manager

Sistema de Autogestão Para Apoio a Alunos

Docente:
Carlos Costa

Trabalho realizado por:
Gonçalo Leal - 98008
Ricardo Rodriguez - 9838

Índice

Introdução

Como projeto final da unidade curricular de Base de Dados decidimos desenvolver um sistema de autogestão para alunos. Este sistema seria utilizado pelo aluno para gerir as suas cadeiras, apontamentos (ou páginas), tarefas, grupos de trabalho, instituições, professores, etc. O tema do projeto foi inspirado numa ferramenta já existente e bastante conhecida, o Notion.

O objetivo principal do desenvolvimento deste trabalho é consolidar os conhecimentos adquiridos ao longo das aulas e pô-los em prática. Toda a matéria, quer teórica quer prática, foi crucial para o desenvolvimento deste trabalho, uma vez que tivemos de tomar todas as decisões sozinhos desde o pensamento e desenho do modelo de base de dados até à sua gestão e manipulação.

Entregáveis

Juntamente com este relatório, entregamos vários ficheiros. Vamos descrever os mais importantes:

- **p9g5_structure.sql**

Script com o esquema da base de dados desenvolvida para o projeto.

- **sql_code**

Pasta com o código SQL desenvolvido.

  - **/triggers.sql**: todos os triggers criados
  - **/functions.sql**: todas as UDF's criadas
  - **/storedProcedures.sql**: todos os procedimentos desenvolvidos
  - **/index.sql**: todos os índices

- **byom**

Pasta com o projeto em C#. Para testar o projeto deve ser feito login com utilizador = admin e password = admin, pois foi a conta utilizada em todas as inserções para a demonstração e, por isso, tem mais dados.
No entanto, também pode ser criada uma nova conta carregando em "Criar Conta" no formulário inicial.

- **byom.pptx**

Slides utilizados na apresentação na aula prática (foram revistos e alterados desde a apresentação)

- https://www.youtube.com/watch?v=zU7QxTbjXBI

Vídeo demonstrativo da interface terminada.

- **Repositório no GitHub:** BD_ProjetoFinal

Repositório utilizado para fazer a gestão e partilha do trabalho entre os membros do grupo. Podem ver-se as várias etapas do projeto através do histórico de commits e o esforço do grupo.

- **Requisitos.pdf:** requisitos identificados no início do projeto

Alterações feitas após a apresentação

Depois da apresentação do trabalho na aula prática, resolvemos todos os erros que tínhamos, acabamos a ligação com a base de dados das entidades apresentadas e adicionamos a lógica necessária para criar e gerir ficheiros. Do lado da base de dados não foram feitas grandes alterações.
As alterações foram demonstradas no vídeo cujo link está disponibilizado acima.

Análise de Requisitos

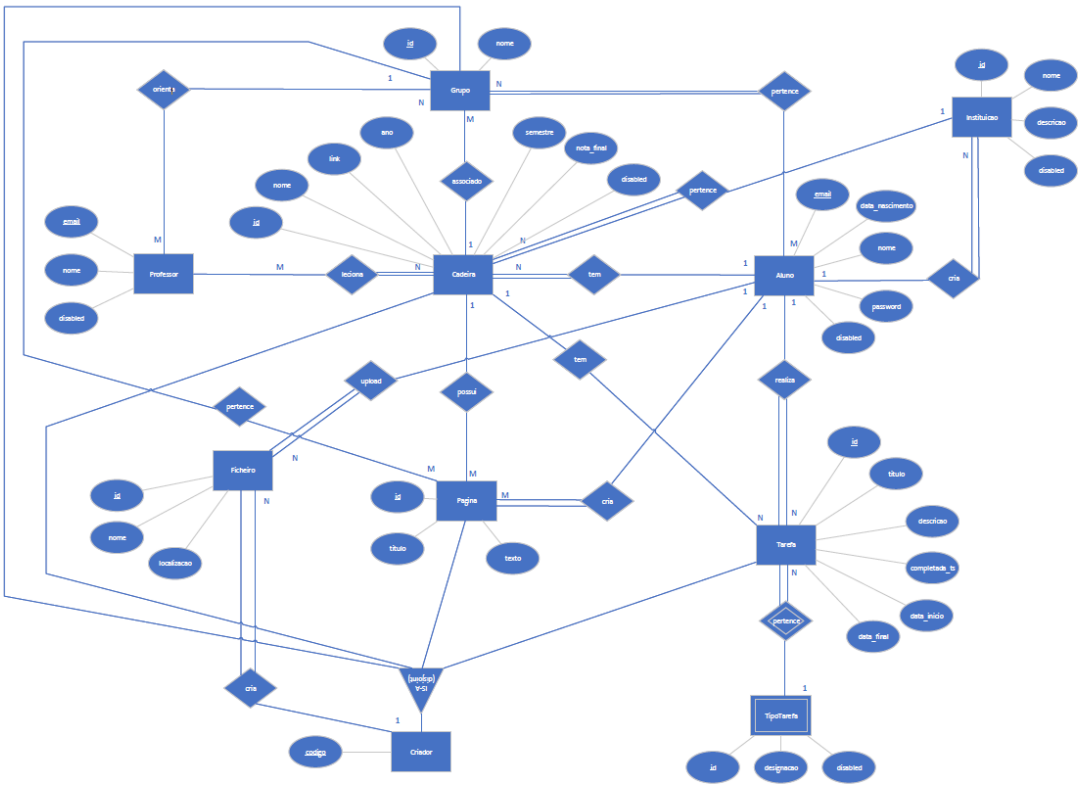
Como a plataforma que nos propomos desenvolver é direcionada para estudantes universitários e sendo que nós somos estudantes universitários, começamos o que seria para nós o mais importante numa plataforma deste género.
Concluímos que para além da entidade principal, que é o próprio aluno, devíamos ter outras entidades como: Instituições, Cadeiras, Professores, Grupos, Páginas e Tarefas.

- **Instituições:** locais onde o aluno estuda ou que frequenta, por exemplo, a Universidade de Aveiro ou o Lancaster King's School
- **Cadeiras:** as unidades curriculares que o aluno frequenta de uma determinada instituição
- **Professores:** os docentes do aluno, que lecionam cadeiras e podem ser orientadores de grupos
- **Grupos:** grupos de trabalho que o aluno queira adicionar, desde grupos para trabalhos a grupos de estudo com amigos; um grupo pode partilhar as mesmas páginas e os mesmos orientadores
- **Páginas:** esta entidade é um pouco mais livre, podendo ser o que o utilizador entender; a ideia é que possa adicionar apontamentos ou descrições das aulas de uma cadeira, partilhar informação com grupos através de páginas
- **Tarefas:** as tarefas que o aluno tem de realizar, no seguimento desta entidade criou-se a entidade TipoTarefa com vários tipos de tarefas que o utilizador pode adicionar para catalogar as suas tarefas

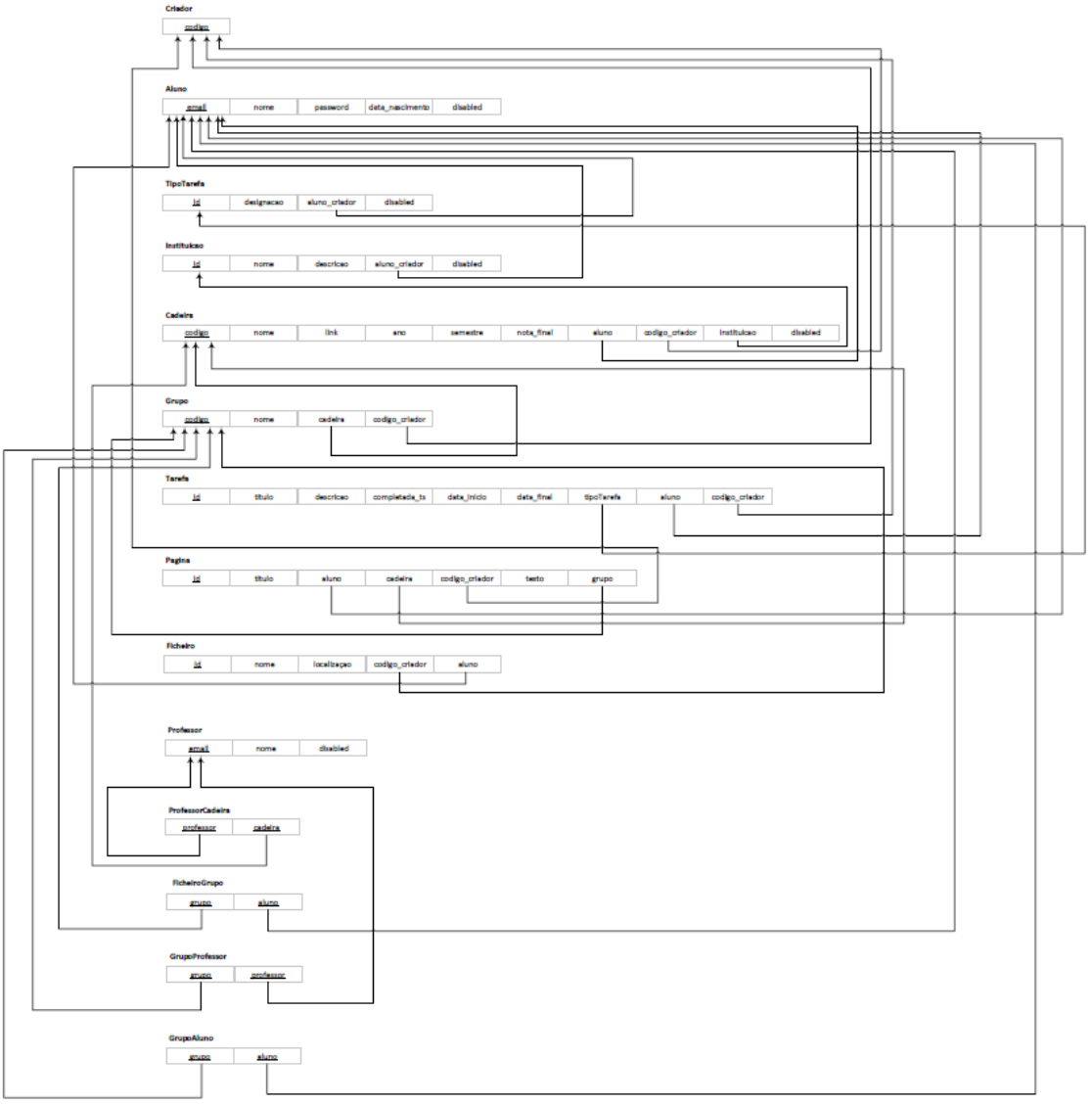
Entendemos que poderia haver a necessidade de o utilizador adicionar Ficheiros na aplicação. A aplicação permite adicionar Ficheiros a Páginas, Grupos, Tarefas e Cadeiras. Assim, surgiu a necessidade de criar uma entidade Criador para relacionar os ficheiros com a entidade onde foi inserido. Todas as entidades onde um ficheiro pode ser inserido são descendentes de Criador.

Desenho da Base de Dados

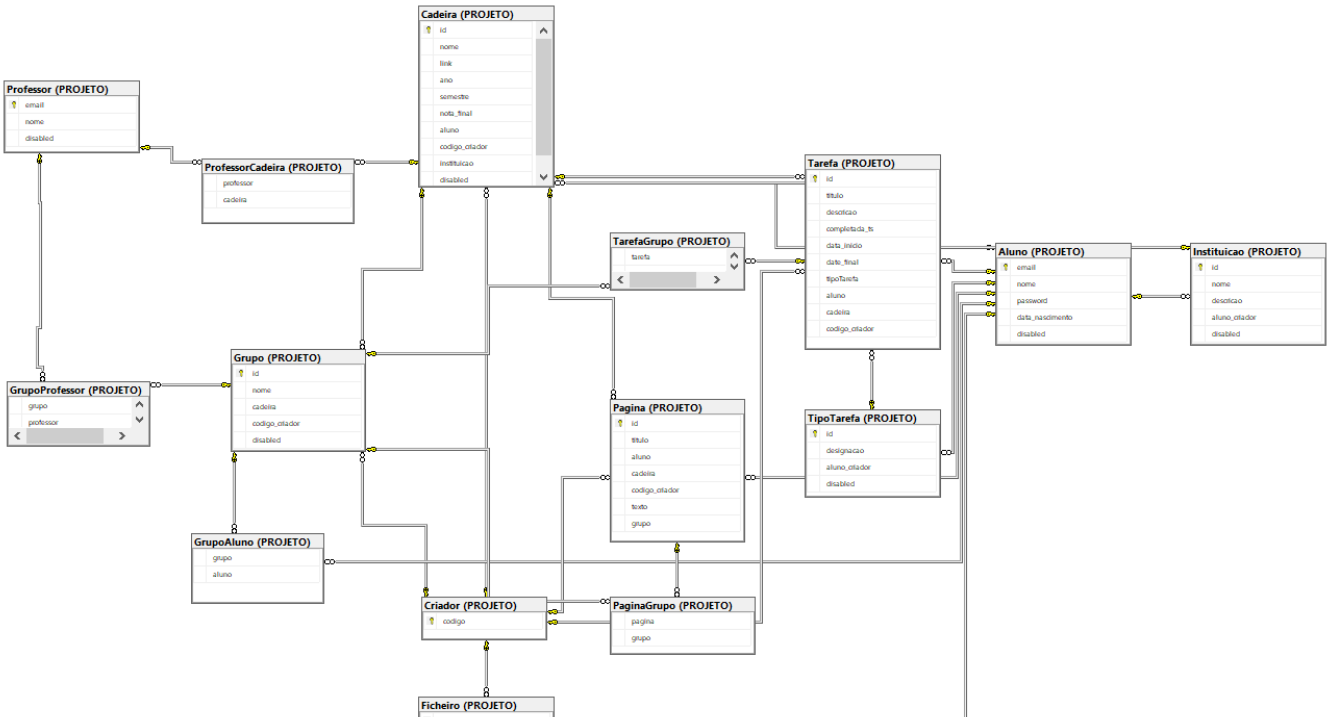
O desenho final da base de dados é muito semelhante ao desenho apresentado ao professor no início do desenvolvimento do trabalho.
Após a apresentação apenas adicionamos a entidade Criador que representa as entidades Página, Grupo, Tarefa e Cadeira enquanto criadores de ficheiros, sendo que a relação é um IS-A disjoint.
Durante o desenvolvimento da interface do projeto e da programação da base de dados apenas foram adicionados alguns atributos a algumas entidades, mas tudo o resto se manteve intacto.



Apresentamos agora o esquema entidade-relação que se encontra em conformidade com o diagrama entidade-relação (DER).



O resultado foi (esquema gerado pelo nosso SGBD):



Código SQL

Triggers

Para permitir a abstração do que acontece na base de dados foram utilizados triggers nos casos em que era possível. Através da utilização de triggers podemos realizar tarefas em vez de outras ou após instruções como inserts ou deletes e o programador da interface não se apercebe de que não é a instrução dele que está a ser executada.

```
CREATE TRIGGER createCriadorTarefa
ON PROJETO.Tarefa
INSTEAD OF INSERT
AS
BEGIN
    DECLARE @code VARCHAR(15);
    DECLARE @titulo VARCHAR(250), @descricao VARCHAR(250), @completada_ts DATETIME,
        @data_inicio DATE, @date_final DATE, @tipoTarefa INT, @aluno VARCHAR(250), @cadeira INT;

    BEGIN TRANSACTION
        SELECT @code = PROJETO.getUniqueCode('T');

        INSERT INTO PROJETO.Criador VALUES(@code);

        SELECT @titulo = titulo, @descricao = descricao, @completada_ts = completada_ts, @data_inicio = data_inicio,
            @date_final = date_final, @aluno = aluno, @tipoTarefa = tipoTarefa, @cadeira = cadeira
        FROM INSERTED;

        INSERT INTO PROJETO.Tarefa VALUES(@titulo, @descricao, @completada_ts, @data_inicio,
            @date_final, @tipoTarefa, @aluno, @cadeira, @code);
    COMMIT

END
```

O trigger apresentado na figura acima substitui a instrução de insert na tabela Tarefa. O trigger foi criado, porque Tarefa deriva de Criador, mas o código do criador é gerado a partir de uma função e toda esta lógica não tem de ser conhecida pelo programador da interface. Assim, em vez de ser logo realizado o insert é primeiro gerado o código de Criador e a entidade Criador e só depois é feito o insert na tabela Tarefa.

```sql
CREATE TRIGGER deleteAluno
ON PROJETO.Aluno
INSTEAD OF DELETE
AS
BEGIN
    DECLARE @email VARCHAR(250);

    BEGIN TRANSACTION
        SELECT @email = email FROM DELETED;

        UPDATE PROJETO.Aluno SET disabled = 1 WHERE email = @email;
    COMMIT

END
```

Outra das formas como os triggers foram utilizados no projeto foi para realizar o soft delete em vez do delete para não haver perda de informação.

Index

Os índices foram usados para melhorar a velocidade de execução da pesquisa por nome/título. Algumas das entidades podem estar disabled (soft delete) e essa caraterística foi tida em consideração aquando da criação dos índices.

```sql
CREATE INDEX IxTituloTarefa ON PROJETO.Tarefa(titulo);
CREATE INDEX IxNomeGrupo ON PROJETO.Grupo(nome);
CREATE INDEX IxTituloPagina ON PROJETO.Pagina(titulo);
CREATE INDEX IxNomeInstituicao ON PROJETO.Instituicao(nome) WHERE disabled = 0;
CREATE INDEX IxNomeCadeira ON PROJETO.Cadeira(nome) WHERE disabled = 0;
```

UDF’s

```sql
CREATE FUNCTION PROJETO.getUniqueCode (@char VARCHAR(2))
RETURNS VARCHAR(15)
AS
BEGIN
    DECLARE @time VARCHAR(15), @code VARCHAR(15)
    SELECT @time = CAST(DATEDIFF_BIG(MILLISECOND, '1970-01-01', GETDATE()) AS VARCHAR(15));
    SET @code = CONCAT(@char, @time)
    RETURN @code
END
```

Esta função é utilizada para a geração do código da entidade Criador através da concatenação de uma letra com o tempo em milissegundos que se passou desde 01-01-1970 até à data atual. A letra concatenada depende da entidade que se irá relacionar com o Criador, por exemplo, se estivermos a criar uma Tarefa concatenamos um “T”.

```sql
CREATE FUNCTION PROJETO.tarefasSemanais ()
RETURNS @t TABLE (
    id INT,
    titulo VARCHAR(250),
    data_tarefa DATE
)
AS
BEGIN
    DECLARE @addDays INT, @sutractDays INT, @weekday VARCHAR(10), @initDate DATE, @finDate DATE;
    SELECT @weekday = DATENAME(WEEKDAY, GETDATE());

    IF @weekday = 'Monday'
        BEGIN
            SET @sutractDays = 0;
            SET @addDays = 6;
        END
    ELSE IF @weekday = 'Tuesday'
        BEGIN
            SET @sutractDays = -1;
            SET @addDays = 5;
        END
    ELSE IF @weekday = 'Wednesday'
        BEGIN
            SET @sutractDays = -2;
            SET @addDays = 4;
        END
    ELSE IF @weekday = 'Thursday'
        BEGIN
            SET @sutractDays = -3;
            SET @addDays = 3;
        END
```

```
    ELSE IF @weekday = 'Friday'
        BEGIN
            SET @sutractDays = -4;
            SET @addDays = 2;
        END
    ELSE IF @weekday = 'Saturday'
        BEGIN
            SET @sutractDays = -5;
            SET @addDays = 1;
        END
    ELSE IF @weekday = 'Sunday'
        BEGIN
            SET @sutractDays = -6;
            SET @addDays = 0
        END

    SELECT @initDate = CAST(DATEADD(DAY, @sutractDays, CAST(GETDATE() AS DATE)) AS DATE);
    SELECT @finDate = CAST(DATEADD(DAY, @addDays, CAST(GETDATE() AS DATE)) AS DATE);

    INSERT INTO @t
        SELECT id, titulo, data_inicio as data_tarefa FROM PROJETO.Tarefa WHERE data_inicio >= @initDate AND data_inicio <= @finDate
        UNION
        SELECT id, titulo, date_final as data_tarefa FROM PROJETO.Tarefa WHERE date_final >= @initDate AND date_final <= @finDate;

    RETURN
END
```

Esta função retorna todas as tarefas da semana atual. Não é usada diretamente na interface, mas sim através de outra função que será apresentada de seguida.

```
CREATE FUNCTION PROJETO.getTarefasSemanaByDia (@dia VARCHAR(10))
RETURNS @table TABLE (
    id INT,
    titulo VARCHAR(250),
    descricao VARCHAR(250),
    completada_ts DATETIME,
    data_inicio DATE,
    date_final DATE,
    tipoTarefa INT,
    aluno VARCHAR(250),
    cadeira INT,
    codigo_criador VARCHAR(20)
)
AS
BEGIN
    DECLARE @t TABLE (id INT, titulo VARCHAR(250), data DATE)
    INSERT INTO @t SELECT * FROM PROJETO.tarefasSemanais() as tarefas

    DECLARE tarefas_cursor CURSOR FOR
        SELECT * FROM @t

    DECLARE @id INT, @titulo VARCHAR(250), @data DATE -- para o cursor
    DECLARE @descricao VARCHAR(250), @completada_ts DATETIME, @data_inicio DATE, @date_final DATE, @tipoTarefa INT, @aluno VARCHAR(250), @cadeira INT, @codigo_criador VARCHAR(20)

    OPEN tarefas_cursor
    FETCH NEXT FROM tarefas_cursor INTO @id, @titulo, @data

    WHILE @@FETCH_STATUS = 0
    BEGIN
        DECLARE @weekday VARCHAR(10);
        SELECT @weekday = DATENAME(WEEKDAY, @data);
        IF @weekday = @dia
        BEGIN
            SELECT @descricao = descricao, @completada_ts = completada_ts, @data_inicio = data_inicio, @date_final = date_final, @tipoTarefa = tipoTarefa, @aluno = aluno, @cadeira = cadeira, @codigo_criador = codigo_criador
            FROM PROJETO.Tarefa
            WHERE id = @id

            INSERT INTO @table VALUES(@id, @titulo, @descricao, @completada_ts, @data_inicio, @date_final, @tipoTarefa, @aluno, @cadeira, @codigo_criador)
        END

        FETCH NEXT FROM tarefas_cursor INTO @id, @titulo, @data
    END

    RETURN
END

SELECT * FROM PROJETO.tarefasSemanais() as tarefas
SELECT * FROM PROJETO.getTarefasSemanaByDia('Tuesday') WHERE completada_ts IS NULL ORDER BY data_inicio DESC
```

Esta função devolve uma tabela com todas as tarefas de um dia da semana atual. O dia da semana é um parâmetro de entrada da função e através da função anterior são selecionadas apenas as tarefas que começam, terminam ou estão em curso nesse dia.

Stored Procedures

No projeto a maioria dos procedimentos são utilizados para criar ou eliminar (ou soft delete) entidades. A criação podia ser feita através de triggers, mas para o desenvolvimento da interface foi vantajoso o uso de procedimentos.
Por exemplo:

```sql
CREATE PROCEDURE PROJETO.createGroup
    @nome VARCHAR(250), @cadeira INT, @aluno VARCHAR(250)
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @code VARCHAR(15), @id INT;
    BEGIN TRANSACTION
        SELECT @code = PROJETO.getUniqueCode('G');

        INSERT INTO PROJETO.Criador VALUES(@code);

        INSERT INTO PROJETO.Grupo(nome, cadeira, codigo_criador) VALUES(@nome, @cadeira, @code);

        SELECT @id = SCOPE_IDENTITY();

        INSERT INTO PROJETO.GrupoAluno VALUES(@id, @aluno);
    COMMIT

    SELECT @id AS id, @code AS code
END
```

```sql
CREATE PROCEDURE PROJETO.deleteCadeira
    @id INT
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION
        DECLARE @code VARCHAR(20);
        SELECT @code = codigo_criador FROM PROJETO.Cadeira WHERE id = @id;

        IF EXISTS (SELECT * FROM PROJETO.Grupo WHERE cadeira = @id)
            BEGIN
                UPDATE PROJETO.Cadeira SET disabled = 1 WHERE id = @id;
            END
        ELSE
            BEGIN
                IF EXISTS (SELECT * FROM PROJETO.ProfessorCadeira WHERE cadeira = @id)
                    BEGIN
                        DELETE FROM PROJETO.ProfessorCadeira WHERE cadeira = @id;
                    END

                IF EXISTS (SELECT * FROM PROJETO.Tarefa WHERE cadeira = @id)
                    BEGIN
                        DELETE FROM PROJETO.Tarefa WHERE cadeira = @id;
                    END

                DELETE FROM PROJETO.Ficheiro WHERE codigo_criador = @code;
                DELETE FROM PROJETO.Cadeira WHERE id = @id;
                DELETE FROM PROJETO.Criador WHERE codigo = @code;

            END

        COMMIT

END
```

Para a verificação das credenciais de login também foi utilizada uma Stored Procedure.

```
269  CREATE PROCEDURE PROJETO.login
270          @email VARCHAR(250), @password VARCHAR(100)
271  AS
272  BEGIN
273      SET NOCOUNT ON;
274      IF EXISTS (SELECT * FROM PROJETO.Aluno WHERE email = @email AND password = @password AND disabled = 0)
275          SELECT 1
276      ELSE
277          SELECT 0
278  END
```

Programação e interface

Apesar de não ser um ponto fulcral para este projeto, achamos importante desenhar uma interface simples e cativante para o utilizador, bem como adicionar várias funcionalidades ao nosso programa.

Inicialmente, achámos pertinente a criação de dois formulários: o de Log-in (*Login.cs*), para utilizadores já existentes; e o de criar conta (*CriarConta.cs*), para alunos que ainda não têm nenhuma conta associada. Ambos os formulários têm as devidas validações para ver, por exemplo, se existe ou não uma conta com os parâmetros inseridos no login ou se já existe uma conta com o email inserido ao criar uma nova conta. Para segurança do utilizador, decidimos armazenar as *passwords* com o resultado de um algoritmo de *hashing*.

Após entrar no programa, o utilizador é movido para a página inicial *(BYOM.cs*). Esta apresenta todas as tarefas semanais, cada uma destas posicionada no(s) respetivo(s) dia(s) da semana. O utilizador pode, também, ordenar as tarefas da semana consoante a data de início e a data de final de forma crescente e decrescente. Este pode também inserir uma nova tarefa ao clicar no botão “Adicionar tarefa”.

Apesar das semelhanças entre a página referida anteriormente, a página Tarefas (Tarefas.cs) engloba todas as tarefas criadas pelo utilizador. Esta possui a mesma funcionalidade de ordenação, de adicionar/remover tarefas mas permite, também, pesquisar tarefas por título, por cadeira, por tipo de tarefa e ou se esta está, ou não, realizada.

Quando o utilizador clica duas vezes numa tarefa, abre-se uma nova página (CriarTarefa.cs) que apresenta toda a informação relativa à mesma: título, descrição, data de início, data final, a cadeira associada (opcional), o tipo de tarefa, bem como a opção de realizada. A partir desta, podemos também criar uma nova cadeira e tipo de tarefa (que depois preenchem os respetivos campos da tarefa), visualizar/adicionar ficheiros à tarefa e, como é evidente, atualizar ou apagar a tarefa.

Na página Cadeiras (Cadeiras.cs), temos uma lista de todas as cadeiras associadas ao utilizador, uma opção de criar uma nova cadeira e, por último, a filtragem de cadeiras através do nome, ano, semestre e instituição.

Podemos criar/visualizar uma cadeira na página CriarCadeira.cs onde podemos introduzir/encontrar os parâmetros associados à mesma (nome, link, nota, instituição,etc..) bem como quatro listas associadas à cadeira: lista de páginas, lista de tarefas, lista de professores e, por último, a lista de ficheiros. Podemos criar ou visualizar cada uma destas entidades específicas associadas à cadeira com as respetivas validações. O utilizador pode apagar, atualizar ou criar uma cadeira, o que se verifica também na maior parte das entidades presentes na nossa base de dados.

Nas Páginas (Paginas.cs), são alistadas todas as páginas criadas pelo aluno, podendo visualizar as já existentes ou criar novas páginas. Esta permite, também, filtrar páginas por nome e por cadeira.

Ao criar uma nova página (CriarPagina.cs), é necessário introduzir um título e uma cadeira (opcional), podendo depois introduzir texto e ficheiros à pagina.

Um aluno pode ver os seus Grupos (Grupos.cs) apresentados em lista, podendo filtrá-los por nome, cadeira e ainda por orientador do mesmo. É possível também adicionar um novo grupo ou visualizar um já existente.

A página CriarGrupo.cs pede um nome e a cadeira associada ao mesmo aquando da criação de um novo grupo mas, caso esteja a ser visualizado um grupo já existente, é possível ver/adicionar/remover colegas de grupo, professores orientadores, páginas do grupo e ficheiros associados ao mesmo. As operações CRUD são também exequíveis.

Por último, um utilizador pode também adicionar/visualizar as suas Instituições (Instituicoes.cs) e filtrar as mesmas consoante o nome e a descrição.

Ao criar uma nova instituição, é obrigatório inserir os parâmetros de nome e de descrição, sendo que a lista das cadeiras associadas à instituição, bem como a filtragens destas por nome, só é visível quando estamos a ver informação de uma instituição já criada na base de dados. Esta entidade suporta, igualmente, as operações CRUD.

Um Professor já existente pode ser adicionado a um grupo na página ModalProfessor.cs, podendo ser criado em CriarProfessor.cs. Da mesma forma, um Ficheiro pode ser adicionado na página CriarFicheiro.cs.

Para poder armazenar informação no programa, guardamos os diferentes tipos de entidades em listas/dicionários. Para isso, tivemos de criar classes que representassem cada uma das entidades, todas estas nomeadas com o prefixo “Classe” (exemplo: ClasseCadeira.cs), que facilitam a apresentação dos dados ao utilizador.

Toda a filtragem das listas referidas anteriormente é realizada através da concatenação de valores na condição *where*’s e a ordenação é feita através de *order by*’s. Isto torna o programa mais lento por ter de consultar constantemente a base de dados, mas sendo este um projeto da unidade curricular de Base de Dados, achamos que seria a melhor opção para pôr em prática o que aprendemos ao longo do semestre.