universidade de aveiro

deti departamento de eletrónica, telecomunicações e informática

Routine View

Base de dados de uma aplicação de gestão de rotina gamificada

Universidade de Aveiro

Licenciatura em Engenharia de Computadores e Informática

Base de Dados P10G6

Regente: Prof. Carlos Costa (carlos.costa@ua.pt)

Professor Orientador: Prof. Joaquim Sousa Pinto (jsp@ua.pt)

Miguel Soares Francisco, 108304- 50%

Rafael Kauati, 105925 - 50%

Índice

Introdução

No âmbito da unidade curricular de Base de Dados, da Licenciatura em Engenharia de Computadores e Informática foi-nos proposto a criação de um projeto que fizesse a gestão de um sistema funcional e com complexidade razoável aplicável ao mundo real. O seguinte relatório irá descrever sucintamente o nosso projeto dando ênfase nas partes que achamos mais fulcrais para o funcionamento do mesmo.

Neste projeto, escolhemos apresentar uma aplicação de gestão de rotina gamificada, pois as pessoas podem ter uma rotina muito ocupada e querer organizá-la melhor e o uso de um sistema de gestão de rotina gamificado pode ajudá-la a se organizar e se manter motivada.

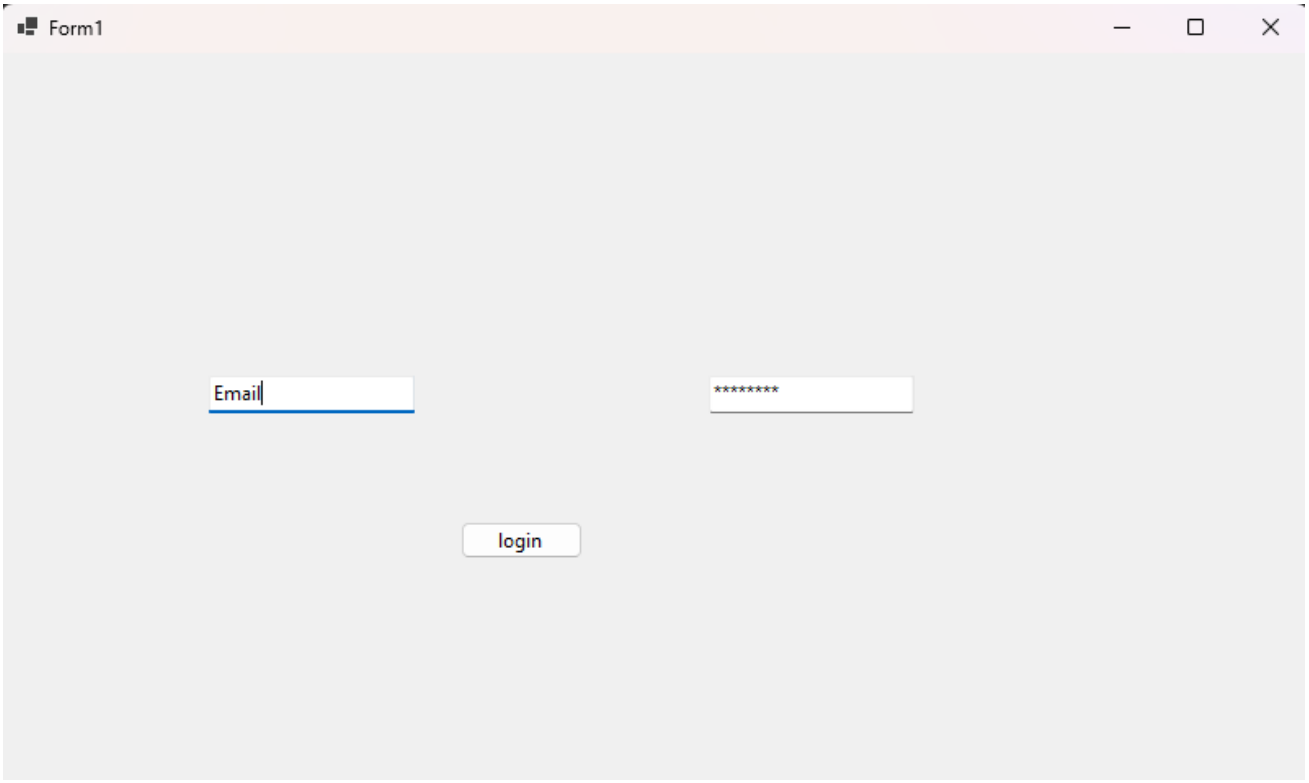
Importante : Os **scripts SQL** se encontram na pasta **SQL/** do projeto e aplicação feita em **Windows Forms** na pasta **App/.** Para abrir a aplicação WF, recomenda-se utilizar a opção **"Abrir um projeto ou uma solução"**,na página inicial do **"Visual Studio na página inicial do Visual Studio Enterprise e Navegar até a pasta App e selecionar a solução Routine-View-Forms.sln"**

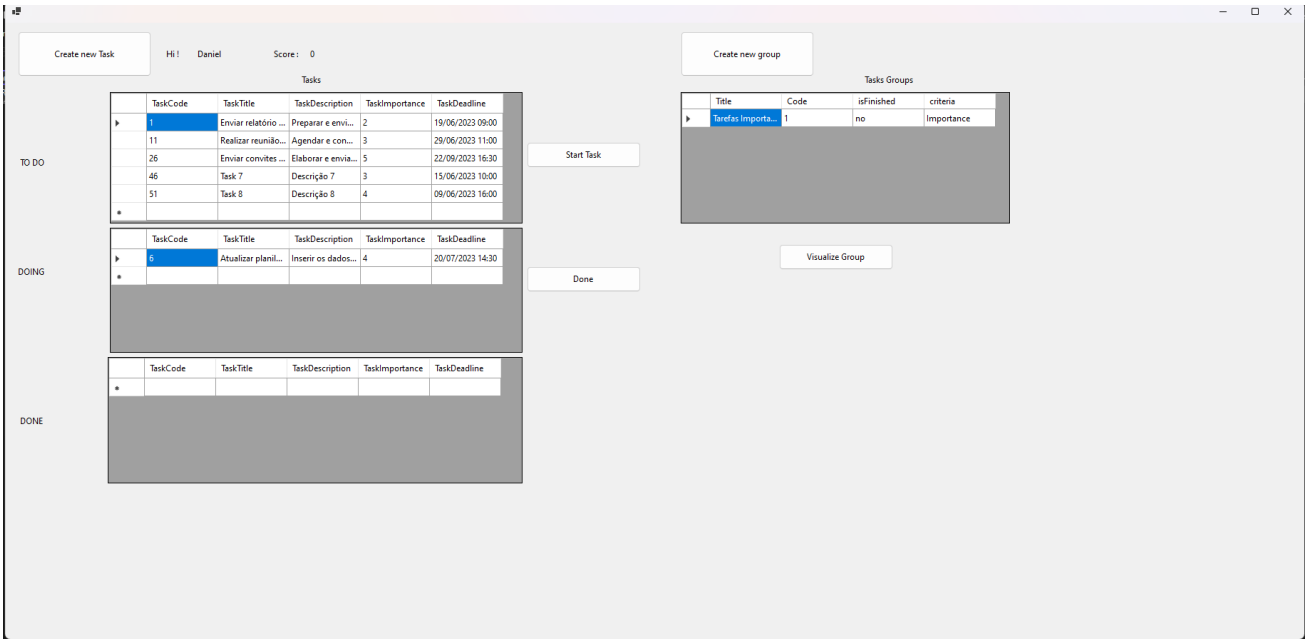
No ficheiro **ConnectionStringHelper.cs** mudar na variável estática "**connectionString"** os campos ID e Password para os dos respectivos utilizadores.

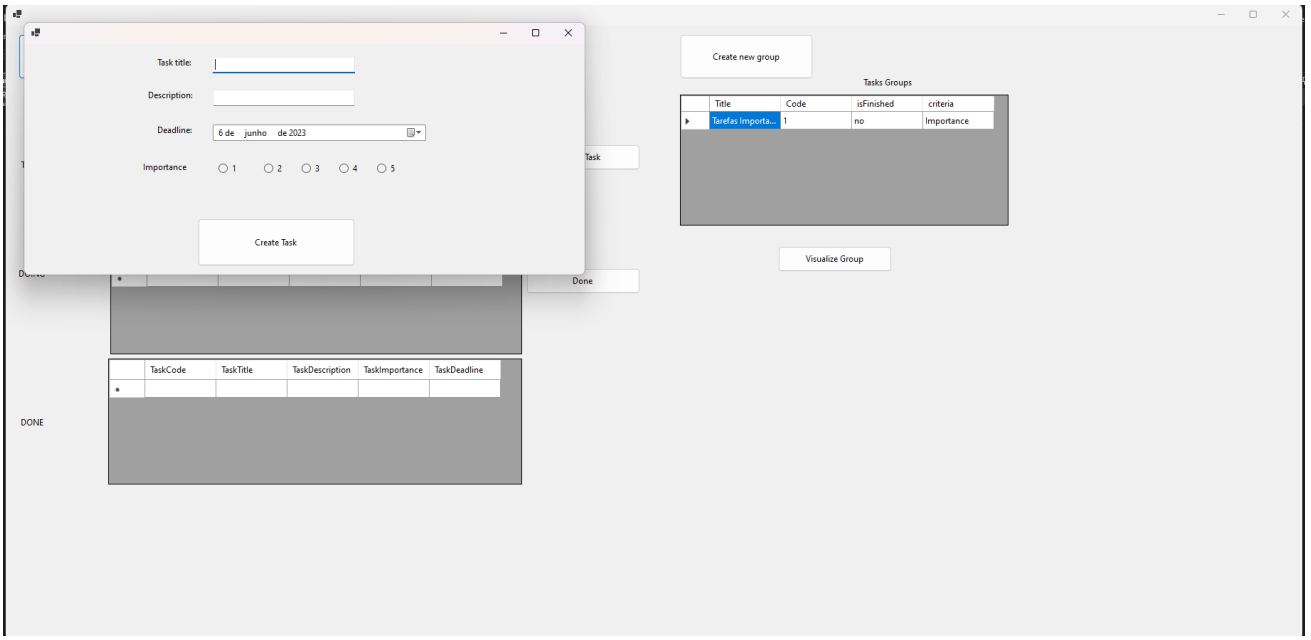
Por fim, a aplicação estará pronta para conectar com a **base de dados** das aulas e **operar** com a **camada de dados** do projeto.

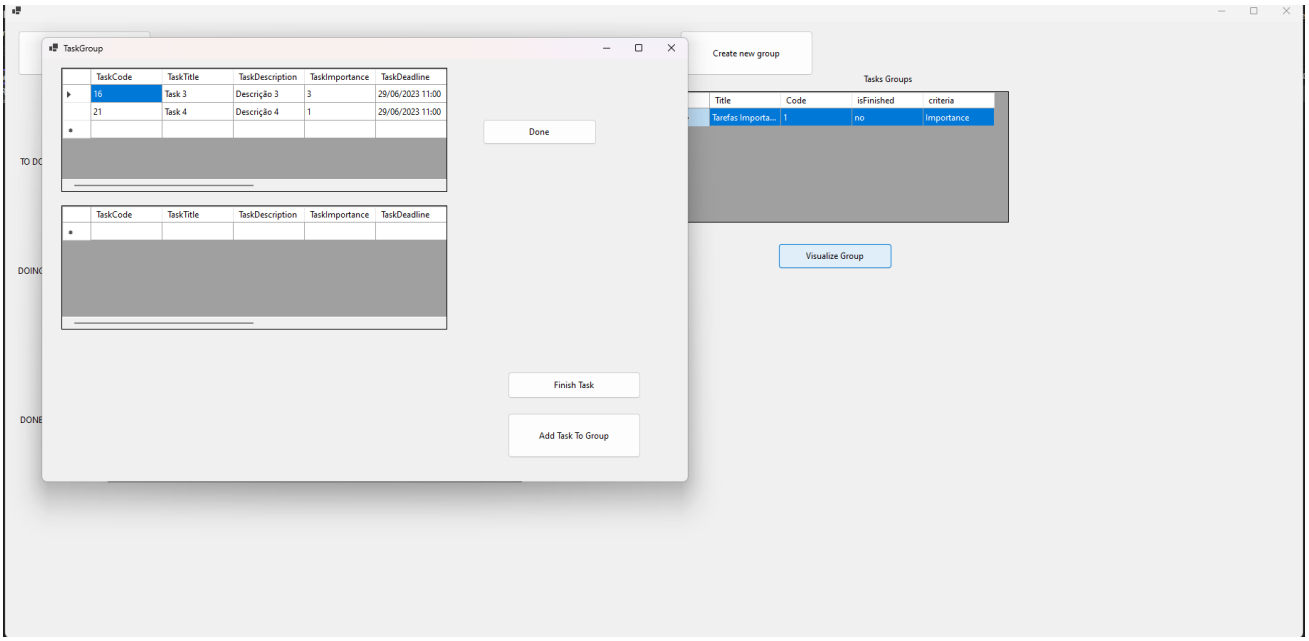
Aplicação

A aplicação desenvolvida como um **Windows Forms App** foi feito para testar as funcionalidades mais básicas e centrais, até então feitas e prontamente testadas :




Create new Task
Hi ! Daniel
Score : 0
Tasks
TaskCode TaskTitle TaskDescription TaskImportance TaskDeadline
1 Enviar relatório ... Preparar e envi... 2 19/06/2023 09:00
11 Realizar reunião... Agendar e con... 3 29/06/2023 11:00
26 Enviar convites ... Elaborar e envia... 5 22/09/2023 16:30
46 Task 7 Descrição 7 3 15/06/2023 10:00
51 Task 8 Descrição 8 4 09/06/2023 16:00
TO DO
Start Task
TaskCode TaskTitle TaskDescription TaskImportance TaskDeadline
6 Atualizar planil... Inserir os dados... 4 20/07/2023 14:30
DOING
Done
TaskCode TaskTitle TaskDescription TaskImportance TaskDeadline
DONE
Create new group
Tasks Groups
Title Code isFinished criteria
Tarefas Importa... 1 no Importance
Visualize Group


Task title:
Description:
Deadline:
6 de junho de 2023
Importance
1 2 3 4 5
Create Task
Create new group
Tasks Groups
Title Code isFinished criteria
Tarefas Importa... 1 no Importance
Visualize Group
Done
DONE



Análise de Requisitos

Nesta aplicação as tarefas são organizadas em uma estrutura de stacks.

Cada task possui informações como título e descrição. Possuem ainda um estado e uma posição na stack de tasks na qual ela pertence.
É possível ainda criar grupos de tasks que são associadas baseadas em um dos critérios de : importância, deadline ou categoria.

Em um nível mais técnico, a posição de cada task na stack é determinada pelo seu grau de prioridade, determinada usando a lógica de matriz de eisenhower pelo grau de importância e deadline de cada task (dados pelo utilizador). No caso de um grupo de task, a prioridade do grupo é a média de todas as prioridades de cada task do grupo.

Também é possível atribuir uma “recompensa” para a conclusão de uma task ou um grupo de tasks, esta recompensa vem na forma de “pontos” que são somados ao total de pontos do utilizador.

Ao realizar algum número de tasks ou atingir alguma meta, o utilizador recebe uma “conquista“ (Semelhante a um achievement de um jogo), o qual possui uma descrição e está associado a uma categoria.

Em suma, a aplicação consiste em criar, editar, visualizar e deletar tarefas (ou conjuntos de tarefas), listar estas, além de listas de recompensas e conquistas.

Entidades centrais

- Task(s)
- Stack(s)
- Grupo(s) de Tasks

- Recompensas
- Conquistas

Requisitos Funcionais:

- Utilizador ser capaz criar, modificar e deletar uma task, ou um grupo de tasks
- Utilizador ser capaz de modificar dados(descrição, importância, urgência, etc) das tasks ou grupo de tasks
- Utilizador ser capaz de pesquisar uma task ou grupo de tasks por nome, grau de importância, urgência, deadline, data de criação, etc
- Utilizador se capaz de visualizar informações sobre a task ou grupo de taks
- Para cada grupo, deve ser associadas task baseadas em um dos critérios : Importância, Urgência (Deadline) ou Categoria
- Associação de uma task ou um grupo de tasks a uma recompensa
- Capacidade de visualizar as task ou de grupos de tasks já concluídas
- Capacidade de listar as recompensas(e seus dados) associadas a uma task ou a um grupo de tasks
- Visualizar todas as conquistas, ou uma única detalhada

Diagrama Entidade Relação (DER)

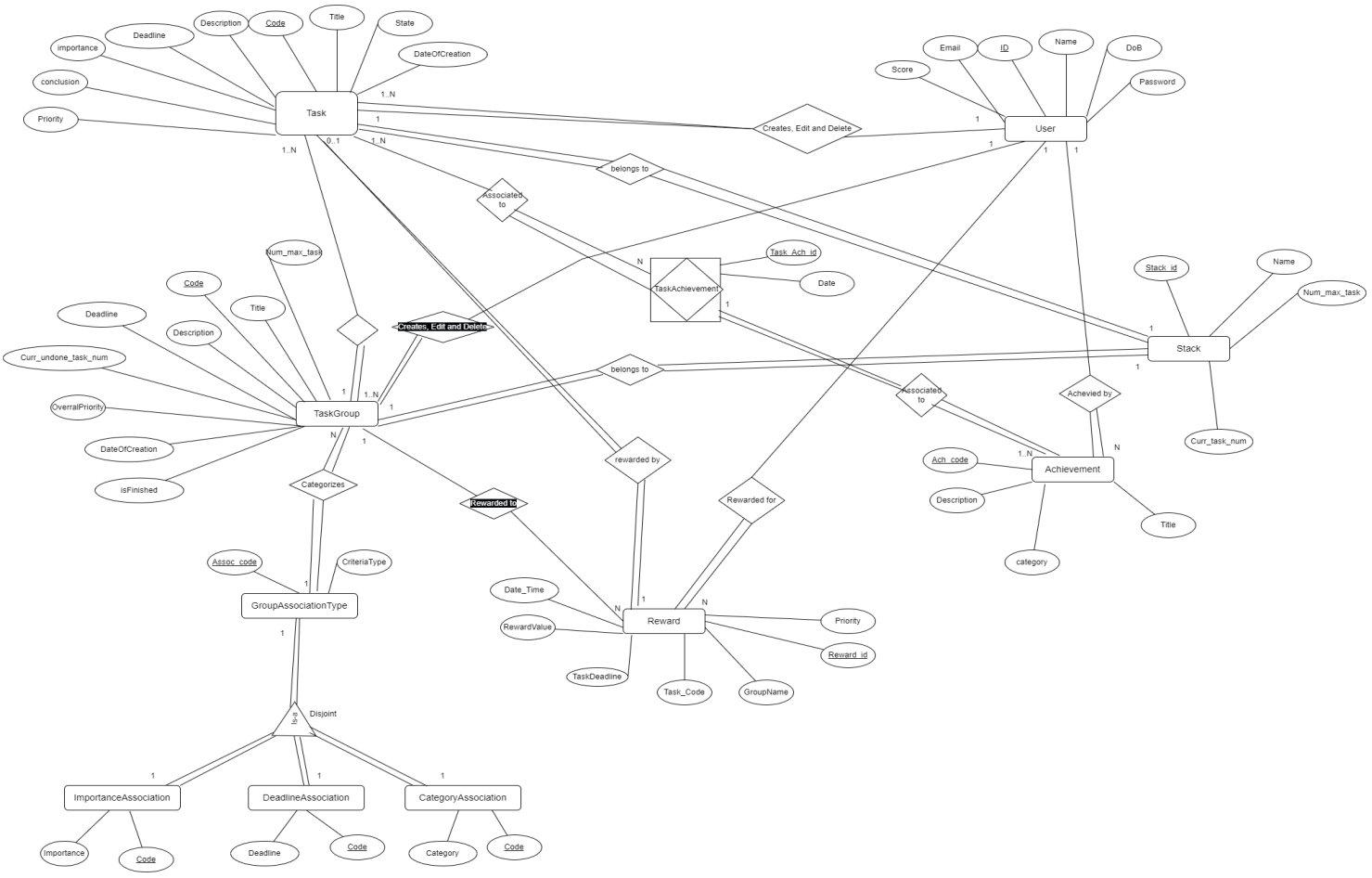
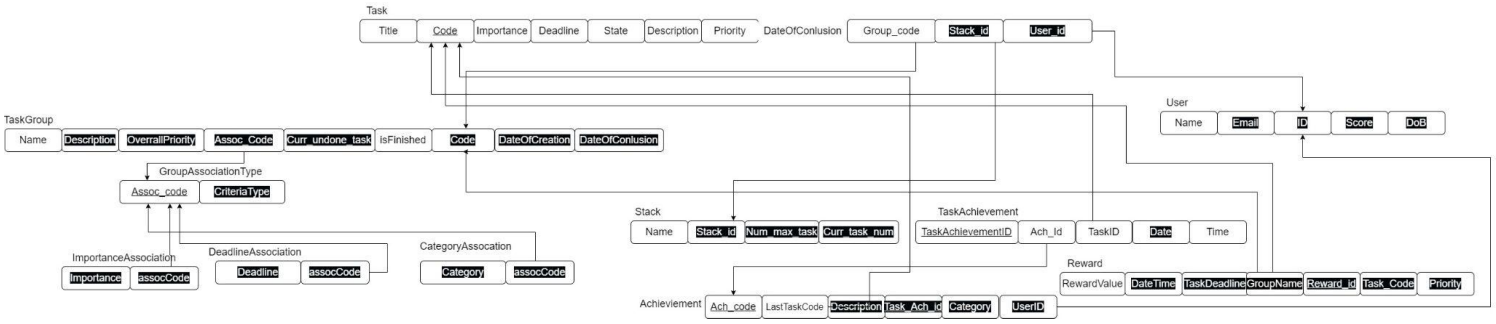


Diagrama Entidade Relação

Esquema Relacional (ER)



Esquema Relacional

Stored Procedure (SP)

Criamos também stored procedures que são utilizados como **handlers** para boa parte dos requisitos funcionais do sistema. São procedimentos que, em sua maioria, fazem múltiplas modificações em várias tabelas da base de dados, por isso quase todos possuem verificações das **transações**, com **commits** em casos das operações forem bem sucedidas e **rollbacks** em caso contrários, incluindo mensagens de erros personalizadas.

```sql
--use [Routine View]
--go

DROP PROCEDURE IF EXISTS startTask
GO

CREATE PROCEDURE startTask(@taskname varchar(100))
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;
        UPDATE [Stack]
        SET CurrMaxTasks = CurrMaxTasks + 1
        WHERE StackID = 2;


        UPDATE [Stack]
        SET CurrMaxTasks = CurrMaxTasks - 1
        WHERE StackID = 1;

        UPDATE T
        SET T.StackID = T.StackID + 1
        FROM [Task] T
        WHERE T.Title = @taskname;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            PRINT 'Error on starting task : '+@taskname;
            ROLLBACK;
    END CATCH
END
GO


DROP PROCEDURE IF EXISTS concludeTask
GO

CREATE PROCEDURE concludeTask (@taskname VARCHAR(30))
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        DECLARE @taskcode INT;
        DECLARE @taskdeadline DATETIME;
        DECLARE @taskpriority INT;

        SELECT @taskcode = Code, @taskdeadline = Deadline, @taskpriority = [Priority]
        FROM [Task]
        WHERE Title = @taskname;

        UPDATE [Task]
        SET StackID = 3, Task.Conclusion = GETDATE()
        WHERE Title = @taskname;

        UPDATE [Stack]
        SET CurrMaxTasks = CurrMaxTasks + 1
        WHERE [Stack].StackID = 3;

        UPDATE [Stack]
        SET CurrMaxTasks = CurrMaxTasks - 1
        WHERE [Stack].StackID = 2;

        INSERT INTO [Reward] ([Task_code], [Task_Deadline], [Date_Time], [Reward_Value])
        VALUES (@taskcode, ISNULL(@taskdeadline, GETDATE()), GETDATE(), @taskpriority * 2);

        COMMIT TRANSACTION;
```

```
        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
        BEGIN
            PRINT 'Error on concluding task : '+@taskname;
            ROLLBACK TRANSACTION;
        END
    END CATCH;
END
GO

DROP PROCEDURE IF EXISTS finishGroupTask
GO

CREATE PROCEDURE finishGroupTask (@taskGroup varchar(50))
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        DECLARE @unfinishedTasks int;

        SELECT @unfinishedTasks = [Curr_undone_task_num]
        FROM Task_Group
        WHERE Task_Group.Title = @taskGroup;

        IF @unfinishedTasks = 0
        BEGIN
            UPDATE Task_Group
            SET Task_Group.isFinished = 'yes'
            WHERE Task_Group.Title = @taskGroup;

            DECLARE @sumPriority int;

            DECLARE @lastTaskCode int;
            DECLARE @lastTaskDeadline datetime;

            SELECT @sumPriority = sum(TaskPriority)
            FROM getTaskGroup(@taskGroup, 'Done');

            SELECT @lastTaskCode = t.TaskCode, @lastTaskDeadline = t.TaskDeadline
            FROM dbo.getTaskGroup(@taskGroup, 'Done') t
            WHERE t.TaskConclusion = (SELECT max(TaskConclusion) FROM dbo.getTaskGroup(@taskGroup, 'Done'));

            INSERT INTO [Reward] ([Task_code], [Task_Deadline], [Date_Time], [Reward_Value])
            VALUES (@lastTaskCode, ISNULL(@lastTaskDeadline, GETDATE()), GETDATE(), @sumPriority * 2);
        END
        ELSE
        BEGIN
            UPDATE Task_Group
            SET Task_Group.isFinished = 'no'
            WHERE Task_Group.Title = @taskGroup;
        END

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
        BEGIN
            PRINT 'Error on finishing task group : '+@taskGroup;
            ROLLBACK TRANSACTION;
        END
    END CATCH;
END
GO

DROP PROCEDURE IF EXISTS checkLogIn;
```

```sql
DROP PROCEDURE IF EXISTS checkLogIn;
GO

CREATE PROCEDURE checkLogIn
(
    @email varchar(40),
    @password varchar(40),
    @confirmation int OUTPUT
)
AS
BEGIN
    DECLARE @DecryptedPassword varbinary(8000);
    DECLARE @userid int;

    SELECT @DecryptedPassword = DecryptByPassPhrase('ThePassphrase', [Password])
    FROM [User]
    WHERE [Email] = @email;

    IF CONVERT(varchar(40), @DecryptedPassword) = @password
    BEGIN
        --DECLARE @username varchar(50);
        SELECT @userid = [User].ID--, @username = [User].[Name]
        FROM [User]
        WHERE [Email] = @email;
        --EXEC sp_addlogin @username, @DecryptedPassword, [Routine View];

        SET @confirmation = @userid;
    END
    ELSE
    BEGIN
        SET @confirmation = 0;
    END
END;
GO



DROP PROCEDURE IF EXISTS addTaskToGroup
GO

CREATE PROCEDURE addTaskToGroup
    @TaskGroupTitle varchar(50),
    @TaskTitle varchar(100)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        DECLARE @ErrorUpdates TABLE (
            TaskTitle varchar(100),
            TaskGroupTitle varchar(50)
        );

        BEGIN TRANSACTION;

        -- Update the task with the group code
        DECLARE @TaskGroupCode int;

        SELECT @TaskGroupCode = Code
        FROM Task_Group
        WHERE Title = @TaskGroupTitle;

        UPDATE Task_Group
        SET Curr_undone_task_num = Curr_undone_task_num + 1
        WHERE Title = @TaskGroupTitle;

        IF @@ROWCOUNT = 0
        BEGIN
            INSERT INTO @ErrorUpdates (TaskTitle, TaskGroupTitle)
            VALUES (@TaskTitle, @TaskGroupTitle);
        END

```

```
211
212         UPDATE Task
213         SET TaskGroupCode = @TaskGroupCode, Task.[State] = 'ToDo'
214         WHERE Title = @TaskTitle;
215
216         IF @@ROWCOUNT = 0
217         BEGIN
218             INSERT INTO @ErrorUpdates (TaskTitle, TaskGroupTitle)
219             VALUES (@TaskTitle, @TaskGroupTitle);
220         END
221
222         -- Update the overall priority of the group task
223         DECLARE @overallPrior int;
224
225         SELECT @overallPrior = AVG([Priority])
226         FROM [Task_Group] g
227         JOIN [Task] t ON g.Code = t.TaskGroupCode
228         WHERE g.Title = @TaskGroupTitle;
229
230         UPDATE Task_Group
231         SET Task_Group.OveralPriority = @overallPrior
232         WHERE Task_Group.Title = @TaskGroupTitle;
233
234         COMMIT TRANSACTION;
235     END TRY
236     BEGIN CATCH
237         IF @@TRANCOUNT > 0
238         BEGIN
239             PRINT 'Error on adding task ' + @TaskTitle + ' to group ' + @TaskGroupTitle;
240             ROLLBACK TRANSACTION;
241         END
242     END CATCH;
243 END
244 GO
245
246
247 DROP PROCEDURE IF EXISTS concludeTaskOfTheGroup
248 GO
249
250
251 CREATE PROCEDURE concludeTaskOfTheGroup
252     @TaskTitle varchar(100)
253 AS
254 BEGIN
255     SET NOCOUNT ON;
256
257     BEGIN TRY
258         BEGIN TRANSACTION;
259
260         DECLARE @TaskGroupCode int;
261
262         SELECT @TaskGroupCode = [TaskGroupCode]
263         FROM Task
264         WHERE Task.Title = @TaskTitle;
265
266         UPDATE Task_Group
267         SET Curr_undone_task_num = Curr_undone_task_num - 1
268         WHERE Code = @TaskGroupCode;
269
270         UPDATE Task
271         SET Task.[State] = 'Done', Task.Conclusion = GETDATE()
272         WHERE Title = @TaskTitle;
273
274         COMMIT TRANSACTION;
275     END TRY
276     BEGIN CATCH
277         IF @@TRANCOUNT > 0
278             PRINT 'Error on concluding task '+@TaskTitle+' from task group : '+str(@TaskGroupCode);
279             ROLLBACK TRANSACTION;
```

```
        IF @@TRANCOUNT > 0
            PRINT 'Error on concluding task '+@TaskTitle+' from task group : '+str(@TaskGroupCode);
            ROLLBACK TRANSACTION;

    END CATCH;
END
GO


DROP PROCEDURE IF EXISTS createTaskGroup
GO

CREATE PROCEDURE createTaskGroup
    @title varchar(50),
    @description varchar(100),
    @assoc_type varchar(20),
    @userid int
AS
BEGIN
    declare @assoc_code int;

    select @assoc_code = tg.Assoc_Code
    from [Task_Group_Assoc] tg
    where tg.CriteriaType = @assoc_type;

    insert into Task_Group ([Title],[Description],[Assoc_code], [DateOfCreation] , [userID])
    values (@title,@description,@assoc_code, GETDATE(),@userid  );

END
```

Views

Criamos algumas views básicas que facilitam a visualização de tarefas baseadas no grau de suas prioridades, assim, caso necessário, para visualizar tarefas sem precisar criar uma consulta nova para isso.

```
--use [Routine View]
--go

drop view if exists high_Priority_Tasks
go

create view high_Priority_Tasks
as

    select t.Title, t.Description, t.Deadline,t.Priority, t.Conclusion, t.TaskGroupCode ,s.[Name] as Stack
    from [Task] t
    join [Stack] s on t.StackID = s.StackID
    where t.Priority >= 3;
go


drop view if exists low_Priority_Tasks
go

create view low_Priority_Tasks
as

    select t.Title, t.Description, t.Deadline,t.Priority, t.Conclusion, t.TaskGroupCode ,s.[Name] as Stack
    from [Task] t
    join [Stack] s on t.StackID = s.StackID
    where t.Priority < 3;
go

```

User Defined Function (UDF)

Temos 3 udfs utilizadas para consultas de tasks e de grupos de tasks(retornam tabelas com os campos mais importantes de cada), todas do tipo tabela (Table-Valued Function)

```
▷ Run on active connection | ═ Select block
--use [Routine View]
--go

drop function if exists GetAllTasksInStack
go

drop function if exists getTaskGroup
go


drop function if exists getAllTaskGroups
go

create function getAllTaskGroups
    (@userid int)
RETURNS TABLE
AS
RETURN(
    select distinct g.Code, g.[Title], g.[isFinished],a.CriteriaType
    from  [Task_Group] g
    join [Task_Group_Assoc] a on g.Assoc_code = a.Assoc_Code
    where g.[userID] = @userid
);
go


create function getTaskGroup
    (@TaskGroupTitle varchar(50), @state varchar(10))
RETURNS TABLE
AS
RETURN(
    select
        t.Code AS TaskCode,
        t.Title AS TaskTitle,
        t.Description AS TaskDescription,
        t.Importance AS TaskImportance,
        t.Deadline AS TaskDeadline,
        t.State AS TaskState,
        t.Priority AS TaskPriority,
        t.StackID,
        t.Conclusion AS TaskConclusion,
        t.UserID
    from
        [Task_Group] g
        join [Task] t on g.Code =  t.TaskGroupCode
    where
        g.Title = @TaskGroupTitle AND t.[State] = @state
);
go
```

```
48  );
49  go
50
51  drop function if exists GetAllTasksInStack
52  go
53
54  CREATE FUNCTION GetAllTasksInStack
55      (@stackName VARCHAR(40), @userid int)
56  RETURNS TABLE
57  AS
58  RETURN
59  (
60      SELECT
61          t.Code AS TaskCode,
62          t.Title AS TaskTitle,
63          t.TaskGroupCode AS TaskGroup,
64          t.Description AS TaskDescription,
65          t.Importance AS TaskImportance,
66          t.Deadline AS TaskDeadline,
67          t.State AS TaskState,
68          t.Priority AS TaskPriority,
69          t.StackID,
70          t.Conclusion AS TaskConclusion,
71          t.UserID
72      FROM
73          [Stack] s
74          JOIN [Task] t ON s.StackID = t.StackID
75      WHERE
76          s.[Name] = @stackName
77          AND t.TaskGroupCode IS NULL AND t.UserID = @userid
78  );
79  GO
```

Triggers

Temos também triggers(gatilhos) **after** e **instead of** que funcionam como **handlers** para inserção e atualização das entidades, em especial vale destacar o **update_priority** que implementa a lógica de atribuição da prioridade para uma **task** baseado nos valores das colunas **deadline** e **importance.**

```sql
--use [Routine View]
--go

DROP TRIGGER IF EXISTS updateTask;
GO

CREATE TRIGGER updateTask ON [Task]
INSTEAD OF UPDATE
AS
BEGIN
    DECLARE @Title VARCHAR(100),
            @Description VARCHAR(100),
            @Importance INT,
            @Priority INT,
            @StackID INT,
            @State varchar(10),
            @TaskGroupCode int,
            @Deadline DATETIME;

    DECLARE cursorTasks CURSOR FOR
        SELECT Title, [Description], Importance,[Priority], StackID ,State, TaskGroupCode,Deadline
        FROM inserted;

    OPEN cursorTasks;

    FETCH NEXT FROM cursorTasks INTO @Title, @Description, @Importance, @Priority,@StackID, @State,@TaskGroupCode ,@Deadline;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        UPDATE t
        SET t.Title = ISNULL(i.Title, t.Title),
            t.[Description] = ISNULL(i.[Description], t.[Description]),
            t.Importance = ISNULL(i.Importance, t.Importance),
            t.Priority = ISNULL(i.Priority, t.Priority),
            t.StackID = ISNULL(i.StackID, t.StackID),
            t.State = ISNULL(i.State, t.State),
            t.[TaskGroupCode] = ISNULL(i.[TaskGroupCode], t.[TaskGroupCode]),
            t.Deadline = ISNULL(i.Deadline, t.Deadline)
        FROM [Task] t
        INNER JOIN inserted i ON t.[Code] = i.[Code];

        FETCH NEXT FROM cursorTasks INTO @Title, @Description, @Importance, @Priority,@StackID, @State,@TaskGroupCode ,@Deadline;
    END;

    CLOSE cursorTasks;
    DEALLOCATE cursorTasks;
END;
GO



DROP TRIGGER IF EXISTS update_priority
GO

CREATE TRIGGER update_priority ON Task
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    IF @@TRANCOUNT > 0
    BEGIN
        IF XACT_STATE() = 1
        BEGIN
            UPDATE t
            SET t.[Priority] = CASE
                                    WHEN DATEDIFF(MINUTE, GETDATE(), i.[Deadline]) / 60 BETWEEN 0 AND 24 THEN
                                        CASE
                                            WHEN i.[Importance] >= 1 AND i.[Importance] < 3 THEN 3
                                            WHEN i.[Importance] >= 3 AND i.[Importance] <= 5 THEN 5
                                        END
```

```
BEGIN
    SET NOCOUNT ON;

    IF @@TRANCOUNT > 0
    BEGIN
        IF XACT_STATE() = 1
        BEGIN
            UPDATE t
            SET t.[Priority] = CASE
                                   WHEN DATEDIFF(MINUTE, GETDATE(), i.[Deadline]) / 60 BETWEEN 0 AND 24 THEN
                                       CASE
                                           WHEN i.[Importance] >= 1 AND i.[Importance] < 3 THEN 3
                                           WHEN i.[Importance] >= 3 AND i.[Importance] <= 5 THEN 5
                                       END
                                   WHEN DATEDIFF(MINUTE, GETDATE(), i.[Deadline]) / 60 BETWEEN 24 AND 48 THEN
                                       CASE
                                           WHEN i.[Importance] >= 1 AND i.[Importance] < 3 THEN 2
                                           WHEN i.[Importance] >= 3 AND i.[Importance] <= 5 THEN 4
                                       END
                                   WHEN DATEDIFF(MINUTE, GETDATE(), i.[Deadline]) / 60 >= 48 THEN
                                       CASE
                                           WHEN i.[Importance] >= 1 AND i.[Importance] < 3 THEN 1
                                           WHEN i.[Importance] >= 3 AND i.[Importance] <= 5 THEN 3
                                       END
                               END
            FROM Task t
            INNER JOIN inserted i ON t.Code = i.Code;

            IF @@ERROR <> 0
            BEGIN
                ROLLBACK;
                RETURN;
            END
        END
        ELSE
        BEGIN
            RAISERROR('A transa��o em andamento  impl�cita. O trigger "update_priority" requer uma transa��o expl�cita.', 16, 1);
            RETURN;
        END
    END
    ELSE
    BEGIN
        BEGIN TRANSACTION;

        BEGIN TRY
            UPDATE t
            SET t.[Priority] = CASE
                                   WHEN DATEDIFF(MINUTE, GETDATE(), i.[Deadline]) / 60 BETWEEN 0 AND 24 THEN
                                       CASE
                                           WHEN i.[Importance] >= 1 AND i.[Importance] < 3 THEN 3
                                           WHEN i.[Importance] >= 3 AND i.[Importance] <= 5 THEN 5
                                       END
                                   WHEN DATEDIFF(MINUTE, GETDATE(), i.[Deadline]) / 60 BETWEEN 24 AND 48 THEN
                                       CASE
                                           WHEN i.[Importance] >= 1 AND i.[Importance] < 3 THEN 2
                                           WHEN i.[Importance] >= 3 AND i.[Importance] <= 5 THEN 4
                                       END
                                   WHEN DATEDIFF(MINUTE, GETDATE(), i.[Deadline]) / 60 >= 48 THEN
                                       CASE
                                           WHEN i.[Importance] >= 1 AND i.[Importance] < 3 THEN 1
                                           WHEN i.[Importance] >= 3 AND i.[Importance] <= 5 THEN 3
                                       END
                               END
            FROM Task t
            INNER JOIN inserted i ON t.Code = i.Code;

            COMMIT;
        END TRY
        BEGIN CATCH
            ROLLBACK;
```

```
            COMMIT;
        END TRY
        BEGIN CATCH
            ROLLBACK;
            THROW;
        END CATCH
    END
END
GO

drop trigger if exists encryptPassword
go

create trigger encryptPassword on [User]
after insert
as
begin
    DECLARE @UserID INT
    DECLARE @Password CHAR(40)

    DECLARE UserCursor CURSOR FOR
    SELECT ID, [Password] FROM inserted

    OPEN UserCursor
    FETCH NEXT FROM UserCursor INTO @UserID, @Password

    WHILE @@FETCH_STATUS = 0
    BEGIN
        -- Atualiza a coluna Password com o valor criptografado
        UPDATE [User]
        SET [Password] =  CONVERT(VARBINARY(8000), EncryptByPassPhrase('ThePassphrase', @Password))
        WHERE ID = @UserID

        FETCH NEXT FROM UserCursor INTO @UserID, @Password
    END

    CLOSE UserCursor
    DEALLOCATE UserCursor
end
go

drop trigger if exists createTask
go

create trigger createTask on [Task]
instead of insert
as
begin
    DECLARE @Title varchar(100),
            @Description varchar(100),
            @Importance int,
            @Deadline datetime,
            @UserID int;

    -- Definir o cursor
    DECLARE cursorTasks CURSOR FOR
    SELECT Title, [Description], Importance, Deadline, [UserID]
    FROM inserted;

    OPEN cursorTasks;

    FETCH NEXT FROM cursorTasks INTO @Title, @Description, @Importance, @Deadline, @UserID;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        INSERT INTO [Task] (Title, [Description], Importance, Deadline, StackID ,[UserID])
        VALUES (@Title, @Description, @Importance, @Deadline,1 ,@UserID);

        UPDATE [Stack]
        SET CurrMaxTasks = CurrMaxTasks + 1
        WHERE StackID = 1;

```

```
195
196         FETCH NEXT FROM cursorTasks INTO @Title, @Description, @Importance, @Deadline, @UserID;
197     END;
198
199     CLOSE cursorTasks;
200
201     DEALLOCATE cursorTasks;
202
203 end;
204 go
205
206
207 DROP TRIGGER IF EXISTS updateUserScore;
208 GO
209
210 CREATE TRIGGER updateUserScore
211 ON [Reward]
212 AFTER INSERT
213 AS
214 BEGIN
215     DECLARE @newReward INT;
216     DECLARE @userid INT;
217
218     SELECT @newReward = [Reward_Value],
219            @userid = Task.UserID
220     FROM inserted
221     INNER JOIN Task ON inserted.Task_code = Task.Code;
222
223     UPDATE [User]
224     SET [Score] = [Score] + @newReward
225     WHERE [User].ID = @userid;
226
227 END;
228 GO
229
230
```

Indexes

Para a otimização de consultas/modificações de entidades da base dados, foram criados **indexes** para as tabelas com parâmetros de **fillfactor (configuração que determina a porcentagem de espaço disponível preenchido por dados em uma página de índice),** distintos, cujo os valores variam dependendo da frequência

(esperada) de leitura/modificação de sua determinada tabelas

```
create unique clustered index user_ID_index on [User](ID);

alter index user_ID_index on [User] rebuild with (fillfactor = 100) ;



create unique clustered index stackID_index on [Stack](StackID);
go

alter index stackID_index on [Stack] rebuild with (fillfactor = 70) ;
go




create unique clustered index task_Code_index on [Task](Code);
go

alter index task_Code_index on [Task] rebuild with (fillfactor = 70) ;
go




create unique clustered index Ach_code_index on Achieviement(Ach_code);
go
alter index Ach_code_index on [Achieviement] rebuild with (fillfactor = 70) ;
go
```

Cursores

Em alguns **triggers** foram implementados alguns **cursores** para iterar sobre linhas inseridas no **insert/updates** de entidades(na sua maioria, **tasks**) e utilizador o valor de suas colunas

```
DROP TRIGGER IF EXISTS updateTask;
GO

CREATE TRIGGER updateTask ON [Task]
INSTEAD OF UPDATE
AS
BEGIN
    DECLARE @Title VARCHAR(100),
            @Description VARCHAR(100),
            @Importance INT,
            @Priority INT,
            @StackID INT,
            @State varchar(10),
            @TaskGroupCode int,
            @Deadline DATETIME;

    DECLARE cursorTasks CURSOR FOR
        SELECT Title, [Description], Importance,[Priority], StackID ,State, TaskGroupCode,Deadline
        FROM inserted;

    OPEN cursorTasks;

    FETCH NEXT FROM cursorTasks INTO @Title, @Description, @Importance, @Priority,@StackID, @State,@TaskGroupCode ,@Deadline;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        UPDATE t
        SET t.Title = ISNULL(i.Title, t.Title),
            t.[Description] = ISNULL(i.[Description], t.[Description]),
            t.Importance = ISNULL(i.Importance, t.Importance),
            t.Priority = ISNULL(i.Priority, t.Priority),
            t.StackID = ISNULL(i.StackID, t.StackID),
            t.State = ISNULL(i.State, t.State),
            t.[TaskGroupCode] = ISNULL(i.[TaskGroupCode], t.[TaskGroupCode]),
            t.Deadline = ISNULL(i.Deadline, t.Deadline)
        FROM [Task] t
        INNER JOIN inserted i ON t.[Code] = i.[Code];

        FETCH NEXT FROM cursorTasks INTO @Title, @Description, @Importance, @Priority,@StackID, @State,@TaskGroupCode ,@Deadline;
    END;

    CLOSE cursorTasks;
    DEALLOCATE cursorTasks;
END;
GO
```

```sql
create trigger encryptPassword on [User]
after insert
as
begin
    DECLARE @UserID INT
    DECLARE @Password CHAR(40)

    DECLARE UserCursor CURSOR FOR
    SELECT ID, [Password] FROM inserted

    OPEN UserCursor
    FETCH NEXT FROM UserCursor INTO @UserID, @Password

    WHILE @@FETCH_STATUS = 0
    BEGIN
        -- Atualiza a coluna Password com o valor criptografado
        UPDATE [User]
        SET [Password] =  CONVERT(VARBINARY(8000), EncryptByPassPhrase('ThePassphrase', @Password))
        WHERE ID = @UserID

        FETCH NEXT FROM UserCursor INTO @UserID, @Password
    END

    CLOSE UserCursor
    DEALLOCATE UserCursor
end
go
```

```sql
create trigger createTask on [Task]
instead of insert
as
begin
    DECLARE @Title varchar(100),
            @Description varchar(100),
            @Importance int,
            @Deadline datetime,
            @UserID int;

    -- Definir o cursor
    DECLARE cursorTasks CURSOR FOR
    SELECT Title, [Description], Importance, Deadline, [UserID]
    FROM inserted;

    OPEN cursorTasks;

    FETCH NEXT FROM cursorTasks INTO @Title, @Description, @Importance, @Deadline, @UserID;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        INSERT INTO [Task] (Title, [Description], Importance, Deadline, StackID ,[UserID])
        VALUES (@Title, @Description, @Importance, @Deadline,1 ,@UserID);

        UPDATE [Stack]
        SET CurrMaxTasks = CurrMaxTasks + 1
        WHERE StackID = 1;

        FETCH NEXT FROM cursorTasks INTO @Title, @Description, @Importance, @Deadline, @UserID;
    END;

    CLOSE cursorTasks;

    DEALLOCATE cursorTasks;

end;
go
```

Segurança

Login e autenticação : Foi desenvolvido um procedimento responsável por validar/autenticar a entrada de usuário (especificamente os utilizadores da tabela User criada para o projeto) na aplicação e, por consequência, a visualização e manipulação de dados da DB através da **GUI**.

Esta funcionalidade ainda não está perfeitamente alinhada com os **Users** da DB (os quais possuem suas próprias credenciais de acesso), mas em trabalho futuro pretende-se conectar estas duas funcionalidades.

Sql-injection : no que diz respeito a questões de vulnerabilidade, tentamos fazer o melhor para obstruir (possíveis) pontos de vulnerabilidade da aplicação feita no Windows Forms, utilizando dados que são previamente validados para assegurar **insert** e **updates** de valores (os quais também passam por outras validações baseadas no estado da entidade a ser atualizada na BD), assim evitando que possíveis acessos inadequados a BD.

Além disso, vale destacar que boa parte das funcionalidades de leitura ou modificação de entidades da base de dados são realizados(em sua maioria) a partir de **UDFs/SPs/Triggers instead of** que possuem suas próprias lógicas de validação dos dados inseridos, de forma que não são realizadas operações sobre a camada de dados com “valores inadequados”.

```sql
DROP PROCEDURE IF EXISTS checkLogIn;
GO

CREATE PROCEDURE checkLogIn
(
    @email varchar(40),
    @password varchar(40),
    @confirmation int OUTPUT
)
AS
BEGIN
    DECLARE @DecryptedPassword varbinary(8000);
    DECLARE @userid int;

    SELECT @DecryptedPassword = DecryptByPassPhrase('ThePassphrase', [Password])
    FROM [User]
    WHERE [Email] = @email;

    IF CONVERT(varchar(40), @DecryptedPassword) = @password
    BEGIN
        SELECT @userid = [User].ID
        FROM [User]
        WHERE [Email] = @email;

        SET @confirmation = @userid;
    END
    ELSE
    BEGIN
        SET @confirmation = 0;
    END
END;
GO
```

```
private int VerifyLogin(string email, string password)
{
    int confirmation = 0;
    string query = "checkLogIn"; // Nome do procedimento armazenado
    using (SqlCommand cmd = new SqlCommand(query, cn))
    {
        cmd.CommandType = CommandType.StoredProcedure;

        // Parâmetros de entrada
        cmd.Parameters.AddWithValue("@email", email);
        cmd.Parameters.AddWithValue("@password", password);

        // Parâmetro de saída
        SqlParameter confirmationParam = new SqlParameter("@confirmation", SqlDbType.Int);
        confirmationParam.Direction = ParameterDirection.Output;
        cmd.Parameters.Add(confirmationParam);

        cmd.ExecuteNonQuery();

        // Obtém o valor de confirmação do parâmetro de saída
        confirmation = (int)confirmationParam.Value;

        // Retorna true se o login for verificado com sucesso (confirmation = 1)
    }
    return confirmation;

}
```

*O **confirmation** é um parâmetro de saída de um **SP** que valida as credenciais de um utilizador inseridas no login da aplicação, caso seja válida a entrada do utilizador, o **confirmation** recebe o **ID** do utilizador, que é utilizado em operações nas outras partes da aplicação.*

Conclusão

Ao longo do desenvolvimento deste projeto fizemos o possível para aprender e resolver as várias etapas de desenvolvimento de um sistema de gestão de base de dados que atendem as necessidades de uma empresa/serviço, desde as etapas de planejamento do domínio da base de dados até questões técnicas de segurança e utilização destes serviços conectados à base de dados.

A equipa reconhece que ainda falta aperfeiçoar várias partes do projeto, bem como o desenvolvimento de outras funcionalidades do serviço, principalmente sobre as entidades **task_groups** e **achievements**, cuja as funcionalidades na camada de dados ainda não foram devidamente implementadas, algo que certamente se tornará trabalho futuro.