Projeto Final de Base de Dados

Gestão de Planos Alimentares e de Exercícios

Base de Dados
2016/17

Grupo P3G11:
Dzianis Bartashevich, 76537

Objetivos:

- Criação de base de dados em MySQL (projeto IHC) que permitisse fácil gestão de planos de alimentação, planos de exercícios, exercícios feitos e monitorização do peso ao longo do tempo.
- Tradução de Base de Dados em MySQL para SQL Server
- Criação de interface para verificar funcionalidade de base de dados

Pré-requisitos:

O objeto principal de base de dados é o utilizador (users) que poderá criar/alterar/apagar planos de exercícios, criar/alterar/eliminar planos de alimentação, criar/eliminar/consultar medição de peso, criar/consultar registo de exercícios feitos.

Os utilizadores registados possuem um número de utilizador único, nome completo, username, email, password e um token com duração limitada que é gerado ao fazer login e o mesmo é apagado caso utilizador faça logout.

Cada vez que utilizador faz login ou registo, é necessário registrar essa operação com data e hora.

O plano de dieta tem um único utilizador e número associado, também possui um nome, hora, avatar, data e hora de criação do plano e informações nutricionais como: proteínas, carboidratos, gorduras e calorias.

Um plano de dieta tem associado 0 ou mais alimentos.

Um alimento num plano tem associado número do plano, nome, quantidade, unidade, data e hora de criação e informações nutricionais como: proteínas, carboidratos, gorduras e calorias.

O plano de exercícios tem um único utilizador e número associado, também possui um nome, número de exercícios no plano, avatar e data e hora de criação do plano.

Um plano de exercícios tem associado 0 ou mais alimentos.

Um exercício num plano tem associado número do plano, nome, quantidade, unidade e data e hora de criação do exercício.

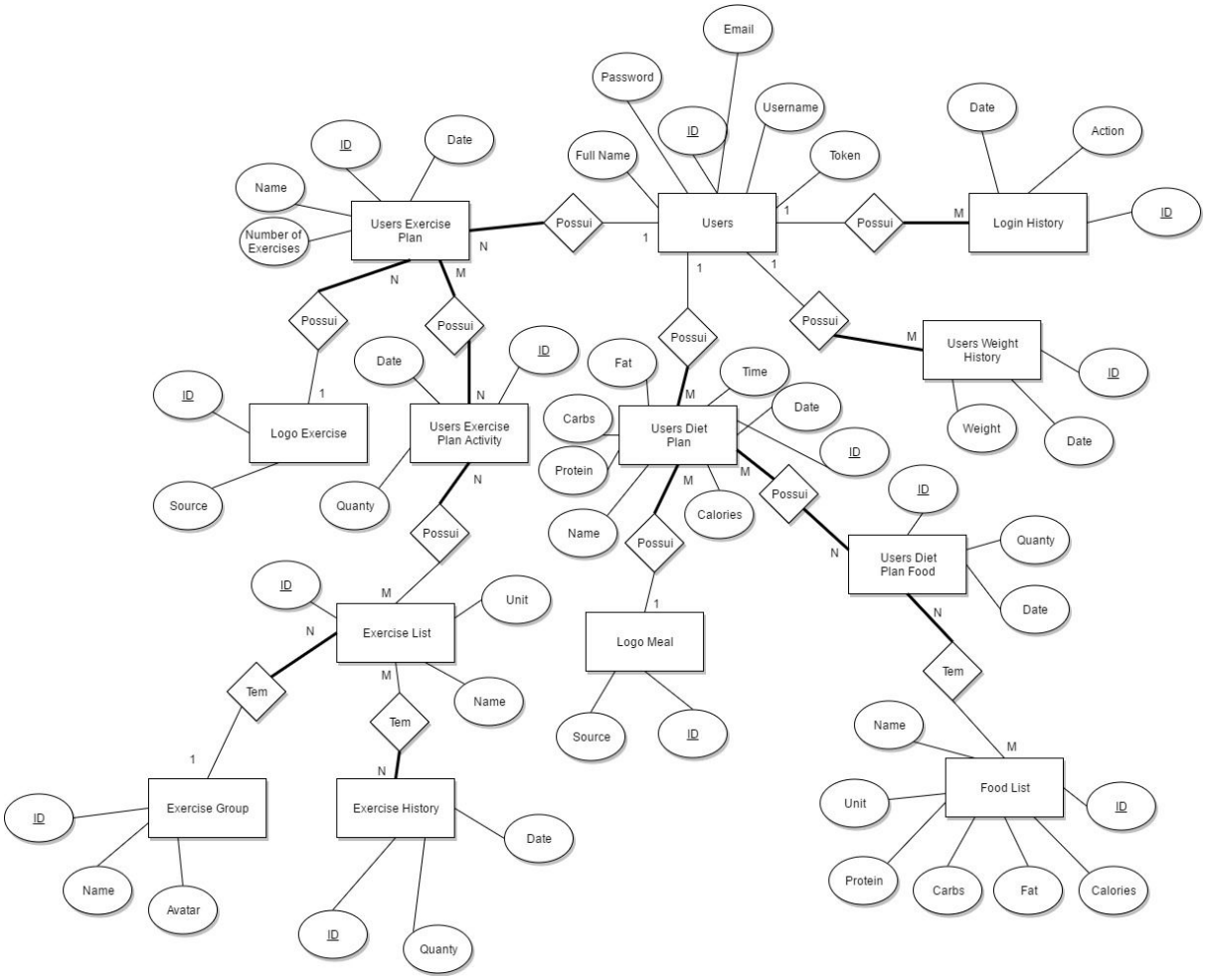
O utilizador deverá conseguir ver o peso actual e adicionar o seu peso com data de pesagem. Também deve ser possível consultar os pesos diariamente, mensalmente e todos registados.

O utilizador deverá ser capaz de consultar e usar exercícios e alimentos guardados na base de dados.

Um exercício tem associado um grupo de músculo, nome e unidade.
O histórico de exercício efetuados possui número do utilizador, nome do exercício, quantidade, unidade e data.

Deverá existir tabela com todos avatares de exercícios e outra com avatares de alimentação.

Modelo ER

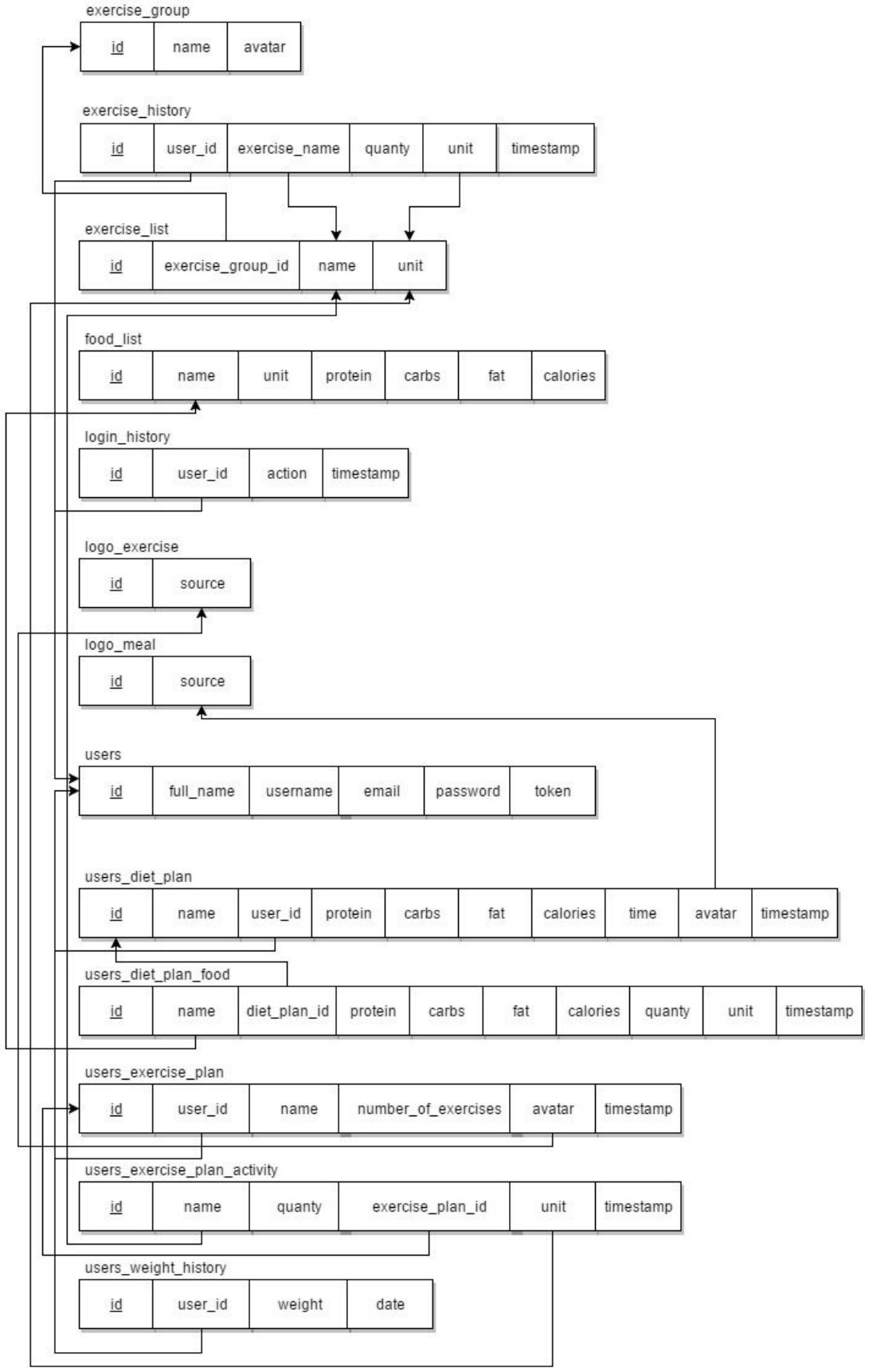


O diagrama possui treze entidades, *Users Exercise Plan, Users, Login History, Users Weight History, Logo Exercise, Users Exercise Plan Activity, Users Diet Plan, Users Diet Plan Food, Exercise List, Logo Meal, Exercise Group, Exercise History e Food List.*

A entidade principal neste diagrama é o utilizador (Users), maioria das entidades tem dependência da existência do utilizador, como por exemplo *Users Exercise Plan*, *Login History*, entre outras ...

Existem várias relações M:N tal como plano de exercício pode conter 1 ou mais exercícios e exercício pode estar contido em 1 ou mais planos.

Modelo relacional



As chaves primárias das tabelas são numéricos do tipo Identidade e auto geradas pelo SGBD.

Campos *Timestamp* são auto gerados pelo SGBD baseando na hora de INSERT/UPDATE (get_date()).

Quando utilizador faz login/registo, password é enviada para base de dados como argumento numa Stored Procedure e é encriptada dentro dela, usando algoritmo básico MD5, este algoritmos não é dos melhores, apenas serve como demonstração das funcionalidades de SQL Server.

Campo *Token* presente na tabelas *Users* serve para autenticar utilizador nas operações sobre base de dados. O *Token* é gerando na aplicação no momento de autenticação do utilizador e é enviado para base de dados.

Definição da estrutura da base de dados (SQL DDL)

Criação da Schema `trainyourlife`:

```
CREATE SCHEMA trainyourlife;
```

Criação de `exercise_group`:

```
CREATE TABLE [trainyourlife].[exercise_group](
	[id] [int] IDENTITY(1,1) NOT NULL,
	[name] [varchar](255) NULL DEFAULT (NULL),
	[avatar] [varchar](255) NULL DEFAULT (NULL),
PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

Criação de `exercise_history`:

```
CREATE TABLE [trainyourlife].[exercise_history](
	[id] [int] IDENTITY(1,1) NOT NULL,
	[user_id] [int] NOT NULL,
	[exercise_name] [varchar](255) NOT NULL,
	[quanty] [decimal](11, 2) NOT NULL,
	[unit] [char](3) NOT NULL,
	[timestamp] [datetime2](0) NOT NULL DEFAULT (getdate()),
PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

```
ALTER TABLE [trainyourlife].[exercise_history]  WITH CHECK ADD  CONSTRAINT [user_id]
FOREIGN KEY([user_id])
REFERENCES [trainyourlife].[users] ([id])
ON UPDATE CASCADE
```

```
ALTER TABLE [trainyourlife].[exercise_history] CHECK CONSTRAINT [user_id]
```

Criação de `exercise_list`:

```
CREATE TABLE [trainyourlife].[exercise_list](
	[id] [int] IDENTITY(1,1) NOT NULL,
	[exercise_group_id] [int] NOT NULL,
	[name] [varchar](255) NOT NULL,
	[unit] [char](3) NOT NULL,
PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

```
ALTER TABLE [trainyourlife].[exercise_list]  WITH CHECK ADD  CONSTRAINT [group_id]
FOREIGN KEY([exercise_group_id])
REFERENCES [trainyourlife].[exercise_group] ([id])
ON UPDATE CASCADE
ON DELETE CASCADE
```

```
ALTER TABLE [trainyourlife].[exercise_list] CHECK CONSTRAINT [group_id]
```

Criação de `food_list`:

```
CREATE TABLE [trainyourlife].[food_list](
	[id] [int] IDENTITY(1,1) NOT NULL,
	[name] [varchar](255) NOT NULL,
	[unit] [varchar](4) NOT NULL,
	[protein] [decimal](11, 2) NOT NULL,
	[carbs] [decimal](11, 2) NOT NULL,
	[fat] [decimal](11, 2) NOT NULL,
	[calories] [decimal](11, 2) NOT NULL,
PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

Criação de `login_history`:

```
CREATE TABLE [trainyourlife].[login_history](
	[id] [int] IDENTITY(1,1) NOT NULL,
	[user_id] [int] NOT NULL,
	[action] [varchar](255) NOT NULL,
	[timestamp] [datetime2](0) NOT NULL DEFAULT (getdate()),
PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

```
ALTER TABLE [trainyourlife].[login_history]  WITH CHECK ADD  CONSTRAINT
[user_id_history] FOREIGN KEY([user_id])
REFERENCES [trainyourlife].[users] ([id])
ON UPDATE CASCADE
ON DELETE CASCADE
```

```
ALTER TABLE [trainyourlife].[login_history] CHECK CONSTRAINT [user_id_history]
```

Criação de `logo_exercise`:

```
CREATE TABLE [trainyourlife].[logo_exercise](
	[id] [int] IDENTITY(1,1) NOT NULL,
	[source] [varchar](255) NOT NULL,
PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

Criação de `logo_meal`:

```
CREATE TABLE [trainyourlife].[logo_meal](
	[id] [int] IDENTITY(1,1) NOT NULL,
	[source] [varchar](255) NOT NULL,
PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

Criação de `users`:

```
CREATE TABLE [trainyourlife].[users](
	[id] [int] IDENTITY(1,1) NOT NULL,
	[full_name] [varchar](255) NOT NULL,
	[username] [varchar](255) NOT NULL,
	[email] [varchar](255) NOT NULL,
	[password] [varchar](255) NOT NULL,
	[token] [varchar](255) NULL DEFAULT (NULL),
PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

Criação de `users_diet_plan`:

```
CREATE TABLE [trainyourlife].[users_diet_plan](
	[id] [int] IDENTITY(1,1) NOT NULL,
	[name] [varchar](255) NOT NULL,
	[user_id] [int] NOT NULL,
	[protein] [decimal](11, 2) NOT NULL CONSTRAINT [DF__users_die__prote__41EDCAC5]
DEFAULT ((0)),
	[carbs] [decimal](11, 2) NOT NULL CONSTRAINT [DF__users_die__carbs__42E1EEFE]
DEFAULT ((0)),
	[fat] [decimal](11, 2) NOT NULL CONSTRAINT [DF__users_diet___fat__43D61337]
DEFAULT ((0)),
	[calories] [decimal](11, 2) NOT NULL CONSTRAINT [DF__users_die__calor__44CA3770]
DEFAULT ((0)),
	[time] [time](0) NOT NULL,
	[avatar] [varchar](255) NOT NULL DEFAULT ('/img/default_meal.jpg'),
	[timestamp] [datetime2](0) NOT NULL DEFAULT (getdate()),
PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

```
ALTER TABLE [trainyourlife].[users_diet_plan]  WITH CHECK ADD  CONSTRAINT [user_id_plan]
FOREIGN KEY([user_id])
REFERENCES [trainyourlife].[users] ([id])
ON UPDATE CASCADE
ON DELETE CASCADE
```

```
ALTER TABLE [trainyourlife].[users_diet_plan] CHECK CONSTRAINT [user_id_plan]
```

Criação de `users_diet_plan_food`:

```
CREATE TABLE [trainyourlife].[users_diet_plan_food](
	[id] [int] IDENTITY(1,1) NOT NULL,
	[diet_plan_id] [int] NOT NULL,
	[name] [varchar](255) NOT NULL,
	[quanty] [decimal](11, 2) NOT NULL,
	[unit] [varchar](4) NOT NULL,
	[protein] [decimal](11, 2) NOT NULL,
	[carbs] [decimal](11, 2) NOT NULL,
	[fat] [decimal](11, 2) NOT NULL,
	[calories] [decimal](11, 2) NOT NULL,
	[timestamp] [datetime2](7) NULL DEFAULT (getdate()),
PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
```

```
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

```
ALTER TABLE [trainyourlife].[users_diet_plan_food]  WITH NOCHECK ADD  CONSTRAINT
[diet_plan_id] FOREIGN KEY([diet_plan_id])
REFERENCES [trainyourlife].[users_diet_plan] ([id])
ON UPDATE CASCADE
ON DELETE CASCADE
```

```
ALTER TABLE [trainyourlife].[users_diet_plan_food] CHECK CONSTRAINT [diet_plan_id]
```

Criação de `users_exercise_plan`:

```
CREATE TABLE [trainyourlife].[users_exercise_plan](
	[id] [int] IDENTITY(1,1) NOT NULL,
	[user_id] [int] NOT NULL,
	[name] [varchar](255) NOT NULL,
	[number_of_exercises] [int] NOT NULL DEFAULT ((0)),
	[avatar] [varchar](255) NOT NULL DEFAULT ('/img/default_meal.jpg'),
	[timestamp] [datetime2](0) NOT NULL DEFAULT (getdate()),
PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

```
ALTER TABLE [trainyourlife].[users_exercise_plan]  WITH NOCHECK ADD  CONSTRAINT
[user_id_exercise_plan] FOREIGN KEY([user_id])
REFERENCES [trainyourlife].[users] ([id])
ON UPDATE CASCADE
ON DELETE CASCADE
```

```
ALTER TABLE [trainyourlife].[users_exercise_plan] CHECK CONSTRAINT
[user_id_exercise_plan]
```

Criação de `users_exercise_plan_activity`:

```
CREATE TABLE [trainyourlife].[users_exercise_plan_activity](
	[id] [int] IDENTITY(1,1) NOT NULL,
	[exercise_plan_id] [int] NOT NULL,
	[name] [varchar](255) NOT NULL,
	[quanty] [decimal](11, 2) NOT NULL CONSTRAINT [DF__users_exe__quant__6442E2C9]
DEFAULT ((0)),
	[unit] [varchar](4) NOT NULL,
	[timestamp] [datetime2](0) NOT NULL DEFAULT (getdate()),
PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

```
ALTER TABLE [trainyourlife].[users_exercise_plan_activity]  WITH NOCHECK ADD  CONSTRAINT
[exercise_plan_id] FOREIGN KEY([exercise_plan_id])
REFERENCES [trainyourlife].[users_exercise_plan] ([id])
ON UPDATE CASCADE
ON DELETE CASCADE
```

```
ALTER TABLE [trainyourlife].[users_exercise_plan_activity] CHECK CONSTRAINT
[exercise_plan_id]
```

Criação de `users_weight_history`:

```
CREATE TABLE [trainyourlife].[users_weight_history](
	[id] [int] IDENTITY(1,1) NOT NULL,
	[user_id] [int] NOT NULL,
	[weight] [decimal](11, 2) NOT NULL DEFAULT ((0)),
	[date] [date] NOT NULL,
PRIMARY KEY CLUSTERED
(
	[id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

```
ALTER TABLE [trainyourlife].[users_weight_history]  WITH CHECK ADD  CONSTRAINT
[user_id_weight] FOREIGN KEY([user_id])
REFERENCES [trainyourlife].[users] ([id])
ON UPDATE CASCADE
ON DELETE CASCADE
```

```
ALTER TABLE [trainyourlife].[users_weight_history] CHECK CONSTRAINT [user_id_weight]
```

SQL DML

Neste projeto decidi não usar queries ad hoc, devido ao seu baixo nível de segurança contra SQL Injection e tempo útil (compilação + execução), todo acesso a base de dados é feito através de Stored Procedure ou User Defined Functions. Para aumentar o nível de segurança ainda era possível bloquear o acesso directo as tabelas.

A entidade que é mais protegida, como no registo ou como na sua atividade na aplicação é o utilizador, nunca divulgamos o seu número único na base de dados e cada vez que é feito login é criado um token único que é usado ao longo da sua sessão.

User Defined Functions:

Obter número do utilizador (esta função só é utilizada dentro de SP ou UDF):

```
CREATE FUNCTION [trainyourlife].[get_user_by_token]
(@token varchar(255)) RETURNS INT
AS
BEGIN
	DECLARE @Value INT;

	SELECT
			@Value = users.id
	FROM
			users
	WHERE
			users.token = @token;

	RETURN @Value;
END
```

Verificar se existe email na base de dados (esta função só é utilizada dentro de SP ou UDF):

```
CREATE FUNCTION [trainyourlife].[email_exists]
(@email varchar(255)) RETURNS int
AS
BEGIN
	DECLARE @s INT;

	IF EXISTS (SELECT * FROM users WHERE users.email = @email) BEGIN SET @s = 1;
	END
	ELSE BEGIN SET @S = 0;

	END

	RETURN @s;
END
```

Verificar se existe username na base de dados (esta função só é utilizada dentro de SP ou UDF):

```
CREATE FUNCTION [trainyourlife].[username_exists]
(@username varchar(255)) RETURNS int
AS
BEGIN
	DECLARE @s INT;

	IF EXISTS (SELECT * FROM users WHERE users.username = @username) BEGIN SET @s =
1;
	END
	ELSE BEGIN SET @S = 0;

	END

	RETURN @s;
END
```

Obter número único do plano de alimentação (esta função só é utilizada dentro de SP ou UDF):

```
CREATE FUNCTION [trainyourlife].[get_meal_by_time]
(@meal_time time(0), @user_id int) RETURNS INT
AS
BEGIN
	DECLARE @Value INT;

	SELECT @Value = users_diet_plan.id
	FROM users_diet_plan
	WHERE users_diet_plan.time = @meal_time AND users_diet_plan.user_id = @user_id;

	RETURN @Value;
END
```

Obter número único do plano de exercícios (esta função só é utilizada dentro de SP ou UDF):

```
CREATE FUNCTION [trainyourlife].[get_plan_by_name]
(@name VARCHAR(255), @user_id INT) RETURNS int
AS
BEGIN
	DECLARE @Value INT;

	SELECT @Value = users_exercise_plan.id
	FROM users_exercise_plan
	WHERE users_exercise_plan.name = @name AND users_exercise_plan.user_id =
@user_id;

	RETURN @Value;
END
```

Stored Procedures

Adicionar exercícios ao histórico:

```
CREATE PROCEDURE [trainyourlife].[add_exercise_to_history]
	(
		@plan_id INT,
		@exercise_name VARCHAR(255),
		@quanty DECIMAL,
		@unit VARCHAR(255),
		@token VARCHAR(255),
		@result INT OUTPUT
	)
AS
BEGIN
	SET NOCOUNT ON;
	DECLARE @user_id INT;
	SET @user_id = [trainyourlife].[get_user_by_token](@token);

	SET @result = 0; -- no problem

	IF @user_id IS NULL BEGIN
		SET @result = 1; -- user doesn't exists/logged in
	END
	ELSE BEGIN
		UPDATE trainyourlife.users_exercise_plan_activity SET quanty = @quanty,
unit = @unit WHERE exercise_plan_id = @plan_id AND users_exercise_plan_activity.[name] =
@exercise_name;
		INSERT INTO trainyourlife.exercise_history
([exercise_name],[quanty],[unit],[user_id]) VALUES (@exercise_name, @quanty, @unit,
@user_id);
	END
END
```

Adicionar exercício ao plano:

```
CREATE PROCEDURE [trainyourlife].[add_exercise_to_plan](@plan_name VARCHAR(255),
@exercise_name varchar(255), @quanty decimal, @unit varchar(255), @token varchar(255),
@result int OUT)
AS
BEGIN
	SET NOCOUNT ON;
	DECLARE @user_id INT;
	DECLARE @plan_id INT;

	SET @user_id = [trainyourlife].get_user_by_token(@token);
	SET @plan_id = [trainyourlife].get_plan_by_name(@plan_name,@user_id);

	SET @result = 0; -- no problem

	IF @user_id IS NULL BEGIN SET @result = 1; -- user doesn't exists/logged in
	END
	ELSE IF @plan_id IS NULL BEGIN SET @result = 2; -- plan doesnt exists
	END
	ELSE BEGIN
				INSERT INTO users_exercise_plan_activity
([exercise_plan_id],[name],[quanty],[unit]) VALUES (@plan_id, @exercise_name, @quanty,
@unit);
	END
END
```

Adicionar alimento ao plano:

```
CREATE PROCEDURE [trainyourlife].[add_food_to_meal](@time time(0), @food_name
varchar(255), @quanty decimal, @unit varchar(255), @protein decimal, @carbs decimal,
@fat decimal, @calories decimal, @token varchar(255), @result int OUT)
AS
BEGIN
	SET NOCOUNT ON;

	DECLARE @user_id INT;
	DECLARE @meal_id INT;

	SET @user_id = trainyourlife.get_user_by_token(@token);
	SET @meal_id = trainyourlife.get_meal_by_time(@time,@user_id);

	SET @result = 0; -- no problem

	IF @user_id IS NULL BEGIN SET @result = 1; -- user doesn't exists/logged in
	END
	ELSE IF @meal_id IS NULL BEGIN SET @result = 2; -- meal doesnt exists
	END
```

```
        ELSE BEGIN
                                INSERT INTO trainyourlife.users_diet_plan_food
([diet_plan_id],[name],[quanty],[unit],[protein],[carbs],[fat],[calories]) VALUES
(@meal_id, @food_name, @quanty, @unit, @protein, @carbs, @fat, @calories);
        END
END
```

Adicionar plano de alimentação:

```
CREATE PROCEDURE [trainyourlife].[add_meal]
(@meal_name varchar(255), @time time, @avatar varchar(255), @token varchar(255), @result
int OUT)
AS
BEGIN
        SET NOCOUNT ON;
        DECLARE @user_id INT;
        SET @user_id = trainyourlife.get_user_by_token(@token);

        SET @result = 0; -- no problem

        IF @user_id IS NULL BEGIN SET @result = 1; -- user doesnt exists/logged in
        END
        ELSE IF trainyourlife.get_meal_by_time(@time,@user_id) IS NOT NULL BEGIN SET
@result = 2; -- check this line, changed during migration, meal already exists
        END
        ELSE BEGIN
                                INSERT INTO trainyourlife.users_diet_plan
([name],[time],[avatar],[user_id]) VALUES (@meal_name, @time, @avatar, @user_id);
        END
END
```

Adicionar plano de exercícios:

```
CREATE PROCEDURE [trainyourlife].[add_plan]
(@name varchar(255), @avatar varchar(255), @token varchar(255), @result int OUT)
AS
BEGIN
        SET NOCOUNT ON;
        DECLARE @user_id INT;
        SET @user_id = trainyourlife.get_user_by_token(@token);

        SET @result = 0; -- no problem

        IF @user_id IS NULL BEGIN SET @result = 1; -- user doesn't exists/logged in
        END
        ELSE IF trainyourlife.get_plan_by_name(@name,@user_id) IS NOT NULL BEGIN SET
@result = 2; -- check this line, changed during migration, meal already exists
        END
        ELSE BEGIN
                                INSERT INTO trainyourlife.users_exercise_plan
```

```
([name],[avatar],[user_id]) VALUES (@name, @avatar, @user_id);
	END
END
```

Adicionar novo utilizador:

```
CREATE PROCEDURE [trainyourlife].[add_user]
(@full_name varchar(255), @username varchar(255), @email varchar(255), @password
varchar(255), @result int OUT)
AS
BEGIN
	DECLARE @hash_pass VARCHAR(32) = CONVERT(VARCHAR(32), HashBytes('MD5',
@password), 2);
	SET NOCOUNT ON;
	SET @result = 0;
	IF trainyourlife.username_exists(@username) = 1 -- IF EXISTS USER WITH THAT
USERNAME
			BEGIN SET @result = 1;
	END
	ELSE IF trainyourlife.email_exists(@email) = 1 -- IF EXISTS USER WITH THAT EMAIL
			BEGIN SET @result = 2;
	END
	ELSE IF @password = @email -- PASSWORD SAME AS EMAIL
			BEGIN SET @result = 3;
	END
	ELSE IF @password = @username -- PASSWORD SAME AS USERNAME
			BEGIN SET @result = 4;
	END
	ELSE IF LEN(@password) < 6   -- PASSWORD LENGHT IS LESS THAN 6 CHARS
			BEGIN SET @result = 5;
	END
	ELSE IF @email NOT LIKE '_%@_%._%' -- EMAIL FORMAT
			BEGIN SET @result = 6;
	END
	ELSE IF LEN(@username) < 3   -- USERNAME LENGHT IS LESS THAN 3 CHARS
			BEGIN SET @result = 7;
	END
	ELSE BEGIN
			INSERT INTO users ([full_name],[username],[email],[password])
VALUES (@full_name, @username, @email, @hash_pass);
	END
END
```

Adicionar peso ao histórico:

```
CREATE PROCEDURE [trainyourlife].[add_weight](@weight DECIMAL, @date date, @token
VARCHAR(255), @result INT OUT)
AS
BEGIN
	BEGIN TRANSACTION add_weight_transaction
		SET NOCOUNT ON;
		DECLARE @user_id INT;
		DECLARE @row_count INT = 0;

		SET @user_id = trainyourlife.get_user_by_token(@token);

		SET @result = 0; -- no problem

		IF @user_id IS NULL SET @result = 1; -- user doesnt exists/logged in
		ELSE IF NOT @weight > 0 BEGIN SET @result = 3; END -- weight cannot be
less than 0
		ELSE
			BEGIN
				INSERT INTO trainyourlife.users_weight_history
([weight],[date],[user_id]) VALUES (@weight, @date, @user_id);

				SELECT @row_count = count(*) FROM
trainyourlife.users_weight_history WHERE users_weight_history.date = @date AND
users_weight_history.user_id = @user_id
			END

		IF @row_count = 1
			BEGIN
				COMMIT
			END
		Else IF NOT @result = 0
			BEGIN
				ROLLBACK
			END
		ELSE
			BEGIN
				SET @result = 2; -- cannot add weight do existant date
				ROLLBACK
			END
END
```

Eliminar exercício do plano de exercício:

```
CREATE PROCEDURE [trainyourlife].[del_exercise_from_plan]( @exercise_id INT, @token
VARCHAR(255), @result int OUT)
AS
BEGIN
	SET NOCOUNT ON;
	DECLARE @user_id INT;

	SET @user_id = trainyourlife.get_user_by_token(@token);

	IF EXISTS (SELECT users_exercise_plan_activity.id FROM
trainyourlife.users_exercise_plan_activity INNER JOIN trainyourlife.users_exercise_plan
ON users_exercise_plan_activity.exercise_plan_id = users_exercise_plan.id WHERE
users_exercise_plan_activity.id = @exercise_id AND users_exercise_plan.user_id =
@user_id) BEGIN
				DELETE FROM trainyourlife.users_exercise_plan_activity
WHERE users_exercise_plan_activity.id = @exercise_id;
				SET @result = 0;
	END
	ELSE BEGIN
				SET @result = 1;
	END
END
```

Eliminar alimento do plano de alimentação:

```
CREATE PROCEDURE [trainyourlife].[del_food_from_meal]( @food_id INT, @token
VARCHAR(255), @result int OUT)
AS
BEGIN
	SET NOCOUNT ON;
	DECLARE @user_id INT;

	SET @user_id = trainyourlife.get_user_by_token(@token);

	IF EXISTS (SELECT users_diet_plan_food.id FROM trainyourlife.users_diet_plan_food
INNER JOIN trainyourlife.users_diet_plan ON users_diet_plan_food.diet_plan_id =
users_diet_plan.id WHERE users_diet_plan_food.id = @food_id AND users_diet_plan.user_id
= @user_id) BEGIN
				DELETE FROM trainyourlife.users_diet_plan_food WHERE
users_diet_plan_food.id = @food_id;
				SET @result = 0;
	END
	ELSE BEGIN
				SET @result = 1;
	END
```

```
END
```

Eliminar plano de alimentação:

```
CREATE PROCEDURE [trainyourlife].[del_meal]( @time time(0), @token VARCHAR(255), @result
int OUT)
AS
BEGIN
	SET NOCOUNT ON;
	DECLARE @user_id INT;
	DECLARE @meal_id INT;

	SET @user_id = trainyourlife.get_user_by_token(@token);
	SET @meal_id = trainyourlife.get_meal_by_time(@time,@user_id);

	SET @result = 0; -- no problem

	IF @user_id IS NULL BEGIN SET @result = 1; -- user doesn't exists/logged in
	END
	ELSE IF @meal_id IS NULL BEGIN SET @result = 2; -- meal doesnt exists
	END
	ELSE BEGIN
				DELETE FROM users_diet_plan WHERE id = @meal_id;
	END
END
```

Eliminar plano de exercícios:

```
CREATE PROCEDURE [trainyourlife].[del_plan]( @plan_name VARCHAR(255), @token
VARCHAR(255), @result int OUT)
AS
BEGIN
	SET NOCOUNT ON;
	DECLARE @user_id INT;
	DECLARE @plan_id INT;

	SET @user_id = trainyourlife.get_user_by_token(@token);
	SET @plan_id = trainyourlife.get_plan_by_name(@plan_name,@user_id);

	SET @result = 0; -- no problem

	IF @user_id IS NULL BEGIN SET @result = 1; -- user doesn't exists/logged in
	END
	ELSE IF @plan_id IS NULL BEGIN SET @result = 2; -- plan doesnt exists
	END
	ELSE BEGIN
				DELETE FROM trainyourlife.users_exercise_plan WHERE id =
@plan_id;
```

```
	END
END
```

Obter o peso atual:

```
CREATE PROCEDURE [trainyourlife].[get_current_weight]( @token VARCHAR(255))
AS
BEGIN
	SET NOCOUNT ON;
	DECLARE @user_id INT;
	SET @user_id = trainyourlife.get_user_by_token(@token);

	SELECT TOP 1
		users_weight_history.weight
	FROM
		trainyourlife.users_weight_history
	WHERE
		users_weight_history.user_id = @user_id
	ORDER BY
		users_weight_history.date DESC;
END
```

Obter nutrição do plano completo:

```
CREATE PROCEDURE [trainyourlife].[get_daily_nutrition] ( @token VARCHAR(255))
AS
BEGIN
	SET NOCOUNT ON;
	DECLARE @user_id INT;
	SET @user_id = trainyourlife.get_user_by_token(@token);

	SELECT
		Sum(users_diet_plan.protein) AS protein_sum,
		Sum(users_diet_plan.carbs) AS carbs_sum,
		Sum(users_diet_plan.fat) AS fat_sum,
		Sum(users_diet_plan.calories) AS calories_sum
	FROM
		users_diet_plan
	WHERE
		users_diet_plan.user_id = @user_id;
END
```

Obter planos de dieta do utilizador:

```
CREATE PROCEDURE [trainyourlife].[get_diet_plans]( @token varchar(255))
AS
BEGIN
	SET NOCOUNT ON;
	DECLARE @user_id INT;
	SET @user_id = trainyourlife.get_user_by_token(@token);

	SELECT users_diet_plan.id, users_diet_plan.name, users_diet_plan.protein,
users_diet_plan.carbs, users_diet_plan.fat, users_diet_plan.calories,
users_diet_plan.time, users_diet_plan.avatar FROM trainyourlife.users_diet_plan WHERE
users_diet_plan.user_id = @user_id ORDER BY users_diet_plan.time ASC;
END
```

Obter exercicios para um grupo de musculo:

```
CREATE PROCEDURE [trainyourlife].[get_exercise_by_group]( @exercise_group INT)
AS
BEGIN
	SET NOCOUNT ON;
	IF @exercise_group = 0 BEGIN
		SELECT
			exercise_list.exercise_group_id,
			exercise_list.name,
			exercise_list.id,
			exercise_list.unit
		FROM
			exercise_list;
	END
	ELSE BEGIN
		SELECT
			exercise_list.exercise_group_id,
			exercise_list.name,
			exercise_list.id,
			exercise_list.unit
		FROM
			exercise_list
		WHERE
			exercise_list.exercise_group_id = @exercise_group;
	END
END
```

Obter exercício pelo ID:

```
CREATE PROCEDURE [trainyourlife].[get_exercise_by_id]( @exercise_id INT)
AS
BEGIN
	SET NOCOUNT ON;
	SELECT
		users_exercise_plan_activity.name,
		users_exercise_plan_activity.quanty,
		users_exercise_plan_activity.unit,
		users_exercise_plan_activity.exercise_plan_id
	FROM
		users_exercise_plan_activity
	WHERE
		users_exercise_plan_activity.id = @exercise_id;
END
```

Obter exercicio de base de dados pelo ID:

```
CREATE PROCEDURE [trainyourlife].[get_exercise_from_list_by_id]( @exercise_id INT)
AS
BEGIN
	SELECT
		exercise_list.name,
		exercise_list.unit,
		exercise_list.exercise_group_id
	FROM
		exercise_list
	WHERE
		exercise_list.id = @exercise_id;
END
```

Obter lista de músculos:

```
CREATE PROCEDURE [trainyourlife].[get_exercise_group_list]
AS
BEGIN
	SET NOCOUNT ON;
	SELECT
		exercise_group.name,
		exercise_group.id,
		exercise_group.avatar
```

```
	FROM
		exercise_group
	ORDER BY
		exercise_group.name;
END
```

Obter histórico de exercícios:

```
CREATE PROCEDURE [trainyourlife].[get_exercise_history_by_name]( @exercise_name
VARCHAR(255), @token VARCHAR(255))
AS
BEGIN
	SET NOCOUNT ON;
	DECLARE @user_id INT;
	SET @user_id = trainyourlife.get_user_by_token(@token);

	SELECT TOP 10
		exercise_history.id,
		exercise_history.quanty,
		exercise_history.unit,
		exercise_history.timestamp,
		exercise_history.exercise_name
	FROM
		exercise_history
	WHERE
		exercise_history.exercise_name = @exercise_name AND
		exercise_history.user_id = @user_id
	ORDER BY timestamp DESC;
END
```

Obter exercícios num plano:

```
CREATE PROCEDURE [trainyourlife].[get_exercise_in_plan]( @plan_id INT)
AS
BEGIN
	SET NOCOUNT ON;

	SELECT users_exercise_plan_activity.id, users_exercise_plan_activity.name,
users_exercise_plan_activity.quanty, users_exercise_plan_activity.unit FROM
trainyourlife.users_exercise_plan_activity WHERE
users_exercise_plan_activity.exercise_plan_id = @plan_id;
END
```

Obter logos dos exercícios:

```
CREATE PROCEDURE [trainyourlife].[get_exercise_logo]
AS
```

```
BEGIN
	SET NOCOUNT ON;
	SELECT logo_exercise.source, logo_exercise.id FROM logo_exercise;
END
```

Obter exercícios num plano:

```
CREATE PROCEDURE [trainyourlife].[get_exercise_plans]( @token VARCHAR(255))
AS
BEGIN
	SET NOCOUNT ON;
	DECLARE @user_id INT;
	SET @user_id = trainyourlife.get_user_by_token(@token);

	SELECT users_exercise_plan.id, users_exercise_plan.name,
users_exercise_plan.number_of_exercises, users_exercise_plan.avatar FROM
trainyourlife.users_exercise_plan WHERE users_exercise_plan.user_id = @user_id;
END
```

Obter alimento da lista de alimentos:

```
CREATE PROCEDURE [trainyourlife].[get_food_by_id]( @food_id INT)
AS
BEGIN
	SET NOCOUNT ON;
	SELECT
		food_list.id,
		food_list.name,
		food_list.unit,
		food_list.protein,
		food_list.carbs,
		food_list.fat,
		food_list.calories
	FROM
		food_list
	WHERE
		food_list.id = @food_id;
END
```

Obter alimentos dentro de um plano:

```
CREATE PROCEDURE [trainyourlife].[get_food_in_meal]( @meal_id int)
AS
BEGIN
	SET NOCOUNT ON;
```

```
	SELECT users_diet_plan_food.id, users_diet_plan_food.name,
users_diet_plan_food.quanty, users_diet_plan_food.unit, users_diet_plan_food.protein,
users_diet_plan_food.carbs, users_diet_plan_food.fat, users_diet_plan_food.calories FROM
trainyourlife.users_diet_plan_food WHERE users_diet_plan_food.diet_plan_id = @meal_id;
END
```

Obter lista de alimento de base de dados:

```
CREATE PROCEDURE [trainyourlife].[get_food_list]
AS
BEGIN
	SET NOCOUNT ON;
	SELECT
		food_list.id,
		food_list.name,
		food_list.unit,
		food_list.protein,
		food_list.carbs,
		food_list.fat,
		food_list.calories
	FROM
		food_list;
END
```

Obter logo dos alimentos:

```
CREATE PROCEDURE [trainyourlife].[get_meal_logo]
AS
BEGIN
	SET NOCOUNT ON;
	SELECT logo_meal.source, logo_meal.id FROM logo_meal;
END
```

Obter lista de exercício do utilizador:

```
CREATE PROCEDURE [trainyourlife].[get_users_exercises]( @token VARCHAR(255))
AS
BEGIN
	SET NOCOUNT ON;
	DECLARE @user_id INT;
	SET @user_id = trainyourlife.get_user_by_token(@token);

	SELECT DISTINCT
		users_exercise_plan_activity.name
	FROM
		trainyourlife.users_exercise_plan_activity
```

```
	INNER JOIN trainyourlife.users_exercise_plan ON
users_exercise_plan_activity.exercise_plan_id = users_exercise_plan.id
	WHERE
		users_exercise_plan.user_id = @user_id;
END
```

Obter plano de alimentação completo do utilizador:

```
CREATE PROCEDURE [trainyourlife].[get_users_full_diet_plan]( @token VARCHAR(255))
AS
BEGIN
	SET NOCOUNT ON;
	DECLARE @user_id INT;
	SET @user_id = trainyourlife.get_user_by_token(@token);

	SELECT
		trainyourlife.users_diet_plan.protein,
		trainyourlife.users_diet_plan.carbs,
		trainyourlife.users_diet_plan.fat,
		trainyourlife.users_diet_plan.calories,
		trainyourlife.users_diet_plan.time,
		trainyourlife.users_diet_plan.name,
		trainyourlife.users_diet_plan_food.name AS food_name,
		trainyourlife.users_diet_plan_food.quanty,
		trainyourlife.users_diet_plan_food.unit,
		trainyourlife.users_diet_plan_food.protein AS food_protein,
		trainyourlife.users_diet_plan_food.carbs AS food_carbs,
		trainyourlife.users_diet_plan_food.fat AS food_fat,
		trainyourlife.users_diet_plan_food.calories AS food_calories
	FROM
		trainyourlife.users_diet_plan
	LEFT JOIN trainyourlife.users_diet_plan_food ON
trainyourlife.users_diet_plan_food.diet_plan_id = trainyourlife.users_diet_plan.id
	WHERE
		trainyourlife.users_diet_plan.user_id = @user_id
	ORDER BY
		trainyourlife.users_diet_plan.time
END
```

Obter plano de exercícios completo do utilizador:

```
CREATE PROCEDURE [trainyourlife].[get_users_full_exercise_plan]( @token VARCHAR(255))
AS
BEGIN
	SET NOCOUNT ON;
	DECLARE @user_id INT;
	SET @user_id = trainyourlife.get_user_by_token(@token);
```

```
	SELECT
		trainyourlife.users_exercise_plan.name,
		trainyourlife.users_exercise_plan.number_of_exercises,
		trainyourlife.users_exercise_plan.avatar,
		trainyourlife.users_exercise_plan_activity.name AS exercise_name,
		trainyourlife.users_exercise_plan_activity.quanty,
		trainyourlife.users_exercise_plan_activity.unit
	FROM
		trainyourlife.users_exercise_plan
	LEFT JOIN trainyourlife.users_exercise_plan_activity ON
trainyourlife.users_exercise_plan_activity.exercise_plan_id =
trainyourlife.users_exercise_plan.id
	WHERE
		trainyourlife.users_exercise_plan.user_id = @user_id
END
```

Obter todas medições de peso do utilizador:

```
CREATE PROCEDURE [trainyourlife].[get_users_weight]( @token VARCHAR(255))
AS
BEGIN
	SET NOCOUNT ON;
	DECLARE @user_id INT;
	SET @user_id = trainyourlife.get_user_by_token(@token);

	SELECT
		users_weight_history.weight,
		users_weight_history.date
	FROM
		users_weight_history
	WHERE
		users_weight_history.user_id = @user_id
	ORDER BY date DESC;
END
```

Obter medições de peso do utilizador de um mês:

```
CREATE PROCEDURE [trainyourlife].[get_weight_statistics_daily]( @token VARCHAR(255),
@def_year INT, @def_month INT)
AS
BEGIN
	SET NOCOUNT ON;
	DECLARE @user_id INT;
	SET @user_id = trainyourlife.get_user_by_token(@token);

	SELECT
		weight,
		DAY([date]) as day
```

```
	FROM
		trainyourlife.users_weight_history
	WHERE
		users_weight_history.user_id = @user_id AND YEAR([date]) = @def_year AND
MONTH([date]) = @def_month
	GROUP BY
		YEAR([date]), MONTH([date]), DAY([date]), weight;
END
```

Obter medições mínimas, médias e máximas de peso do utilizador de um ano:

```
CREATE PROCEDURE [trainyourlife].[get_weight_statistics_monthly]( @token VARCHAR(255),
@def_year INT)
AS
BEGIN
	SET NOCOUNT ON;
	DECLARE @user_id INT;
	SET @user_id = trainyourlife.get_user_by_token(@token);

	SELECT
		min(weight) as min_weight,
		max(weight) as max_weight,
		avg(weight) as avg_weight,
		MONTH([date]) as month
	FROM
		users_weight_history
	WHERE
		users_weight_history.user_id = @user_id AND YEAR([date]) = @def_year
	GROUP BY
		YEAR([date]), MONTH([date]);
END
```

Efetuar login:

```
CREATE PROCEDURE [trainyourlife].[login_user]( @login varchar(255), @password
varchar(255), @token varchar(255), @result int OUT)
AS
BEGIN
		DECLARE @hash_pass VARCHAR(32) = CONVERT(VARCHAR(32), HashBytes('MD5',
@password), 2);
	SET NOCOUNT ON;
	SET @result = 1;
	IF @login LIKE '_%@_%._%' -- EMAIL FORMAT
			BEGIN
				IF trainyourlife.email_exists(@login) = 1
					BEGIN
						IF EXISTS (SELECT users.id FROM users WHERE
users.password = @hash_pass AND users.email = @login)
							BEGIN SET @result = 0; UPDATE users
SET users.token = @token WHERE users.email = @login;
```

```
                                        END
                                END
        END
        ELSE IF trainyourlife.username_exists(@login) IS NOT NULL -- IF EXISTS USER
                BEGIN
                        IF EXISTS (SELECT users.id FROM users WHERE users.password =
@hash_pass AND users.username = @login)
                                BEGIN SET @result = 0; UPDATE users SET users.token =
@token WHERE users.username = @login;
                                END
                END
END
```

Efetuar logout:

```
CREATE PROCEDURE [trainyourlife].[logout_user]( @token varchar(255), @result int OUT)
AS
BEGIN
        DECLARE @id INT;
        SET @id = trainyourlife.get_user_by_token(@token);

        SET @result = 1;

        IF @id IS NOT NULL
                BEGIN
                        UPDATE users SET token = NULL WHERE users.id = @id; SET @result =
0;
        END
END
```

Procurar exercícios:

```
CREATE PROCEDURE [trainyourlife].[search_exercise](@search VARCHAR(255))
AS
BEGIN
        SELECT * FROM exercise_list WHERE exercise_list.name LIKE '%' + @search + '%'
END
```

Atualizar informação nutricional do plano de alimentação:

```
CREATE PROCEDURE [trainyourlife].[update_meal_nutrition]( @meal_id INT)
AS
BEGIN
	DECLARE @protein_sum DECIMAL;
	DECLARE @carbs_sum DECIMAL;
	DECLARE @fat_sum DECIMAL;
	DECLARE @calories_sum DECIMAL;

	SELECT
		@protein_sum = SUM(case when unit = 'g' then  (quanty*(protein/100)) else
(quanty*protein) end),
		@carbs_sum = SUM(case when unit = 'g' then  (quanty*(carbs/100)) else
(quanty*carbs) end),
		@fat_sum = SUM(case when unit = 'g' then  (quanty*(fat/100)) else
(quanty*fat) end),
		@calories_sum = SUM(case when unit = 'g' then  (quanty*(calories/100))
else  (quanty*calories) end)
	FROM
		users_diet_plan_food
	WHERE
		users_diet_plan_food.diet_plan_id = @meal_id;

	IF @protein_sum IS NULL BEGIN SET @protein_sum = 0;END
	IF @carbs_sum IS NULL BEGIN SET @carbs_sum = 0;END
	IF @fat_sum IS NULL BEGIN SET @fat_sum = 0;END
	IF @calories_sum IS NULL BEGIN SET @calories_sum = 0;END

	UPDATE users_diet_plan
		SET
			users_diet_plan.protein = @protein_sum,
			users_diet_plan.carbs = @carbs_sum,
			users_diet_plan.fat = @fat_sum,
			users_diet_plan.calories = @calories_sum
	WHERE users_diet_plan.id = @meal_id;
END
```

Triggers

Adicionar o log quando utilizador faz login:

```
CREATE TRIGGER [trainyourlife].[update_on_login]
ON [trainyourlife].[users]
WITH EXECUTE AS CALLER
AFTER UPDATE
AS
BEGIN
	DECLARE
		@user_id INT

		SELECT @user_id = id FROM INSERTED
   INSERT INTO p3g11.trainyourlife.login_history (user_id,action) VALUES
(@user_id,'login');
END
```

Adicionar o log quando utilizador faz registo:

```
CREATE TRIGGER [trainyourlife].[update_on_register]
ON [trainyourlife].[users]
WITH EXECUTE AS CALLER
AFTER INSERT
AS
BEGIN
	DECLARE
		@user_id INT

		SELECT @user_id = id FROM INSERTED
   INSERT INTO p3g11.trainyourlife.login_history (user_id,action) VALUES
(@user_id,'register');
END
```

Atualizar informação nutricional do plano de alimentação quando é feito insert ou update:

```
CREATE TRIGGER [trainyourlife].[update_nutrition_on_insert]
```

```
ON [trainyourlife].[users_diet_plan_food]
WITH EXECUTE AS CALLER
AFTER INSERT, UPDATE
AS
BEGIN
	DECLARE @meal_id INT;
	SELECT @meal_id = diet_plan_id FROM INSERTED;
	EXEC [trainyourlife].[update_meal_nutrition] @meal_id;
END
```

Atualizar informação nutricional do plano de alimentação quando é feito delete:

```
CREATE TRIGGER [trainyourlife].[update_nutrition_on_delete]
ON [trainyourlife].[users_diet_plan_food]
WITH EXECUTE AS CALLER
AFTER DELETE
AS
BEGIN
	DECLARE @meal_id INT;
	SELECT @meal_id = diet_plan_id FROM DELETED;
	EXEC [trainyourlife].[update_meal_nutrition] @meal_id;
END
```

Atualizar número de exercícios contidos num plano quando é feito delete:

```
CREATE TRIGGER [trainyourlife].[update_exercise_numero_on_delete]
ON [trainyourlife].[users_exercise_plan_activity]
WITH EXECUTE AS CALLER
AFTER DELETE
AS
BEGIN
   DECLARE @exer_num INT;

   SELECT @exer_num = count(*) FROM users_exercise_plan_activity WHERE exercise_plan_id
= (SELECT exercise_plan_id FROM DELETED);

   UPDATE users_exercise_plan SET number_of_exercises = @exer_num WHERE id = (SELECT
exercise_plan_id FROM DELETED);
END
```

Atualizar número de exercícios contidos num plano quando é feito insert ou update:

```
CREATE TRIGGER [trainyourlife].[update_exercise_numero_on_insert_update]
ON [trainyourlife].[users_exercise_plan_activity]
WITH EXECUTE AS CALLER
AFTER INSERT, UPDATE
AS
BEGIN
```

```
   DECLARE @exer_num INT;

   SELECT @exer_num = count(*) FROM users_exercise_plan_activity WHERE exercise_plan_id
= (SELECT exercise_plan_id FROM INSERTED);

   UPDATE users_exercise_plan SET number_of_exercises = @exer_num WHERE id = (SELECT
exercise_plan_id FROM INSERTED);
END
```

Indexação

Depois de uma análise efetuada as queries, foi verificado que na maioria dos caso user_id é o parâmetro de pesquisa mais usado.

```
[exercise_history]

CREATE NONCLUSTERED INDEX [user_id]
ON [trainyourlife].[exercise_history] (
  [user_id] ASC
)

[exercise_list]

CREATE NONCLUSTERED INDEX [group_id]
ON [trainyourlife].[exercise_list] (
  [exercise_group_id] ASC
)

[login_history]

CREATE NONCLUSTERED INDEX [user_id]
ON [trainyourlife].[login_history] (
  [user_id] ASC
)

[users_exercise_plan]

CREATE NONCLUSTERED INDEX [user_id_exercise_plan]
ON [trainyourlife].[users_exercise_plan] (
  [user_id] ASC
)

[users_exercise_plan_activity]

CREATE NONCLUSTERED INDEX [exercise_plan_id]
ON [trainyourlife].[users_exercise_plan_activity] (
  [exercise_plan_id] ASC
)
```

```
[users_weight_history]

CREATE NONCLUSTERED INDEX [user_id_weight]
ON [trainyourlife].[users_weight_history] (
  [user_id] ASC
)
```

Considerações finais

Do meu ponto de vista a base de dados criada corresponde aos requisitos propostos e mais do que isso, em termos de segurança.

Neste projeto foram usados vários aspectos aprendidos nas aulas, como criação de tabelas, chaves primárias e estrangeiras, triggers, criar índices *clustered* e *nonclustered* para melhorar o desempenho dos pedidos, stored procedures, user defined functions também para melhorar o desempenho e segurança, encriptação de palavras, no entanto decidi não usar views porque não me pareceu trazer vantagens. Também foram usadas as transações, no entanto não tínhamos nenhum bom caso de aplicação.

Em relação a interface criada em Visual Basic, não houve muito tempo para aprender e desenvolver uma aplicação com boa interface gráfica como outros grupos poderão ter, devido ao facto na disciplina IHC, foi desenvolvida a mesma interface gráfica mas em HTML/JS.

Como a disciplina é focada principalmente na base de dados e não na forma como o resultado é apresentado visualmente, fiz uma interface minimalista funcional.

Como ao nível de base de dados ou interface gráfica desenvolvida, foram feitos testes com vários utilizadores e não houve reclamações. O projeto continuará a ser desenvolvido por mim e em breve estará online, neste momento está em fase de teste, só funciona na versão mobile, ou seja, só funciona num telemóvel, está no seguinte link (http://trainyourlife.tk) Essa foi a razão de eu ter preferido fazer interface em HTML/JS, devido a possibilidade de ter um projeto funcional e ser utilizado pelas pessoas.