

**Ano Letivo 2019/2020**

**Projeto de Base de Dados**

**Cantina**

Luís Alves de Sousa Rêgo, LEI, 68826

João Eduardo Santos Alcatrão, CM, 76763

**Introdução**

Como objetivo deste relatório pretende-se apresentar e expor o nosso tema para o projeto de Base de Dados, mostrando o que foi feito de modo a realizar este trabalho e os procedimentos que foram seguidos para cumprir os requisitos estabelecidos. Para este efeito vão ser apresentadas ao longo deste documento as várias facetas que constituem o nosso projeto, incluindo a análise de requisitos, diagrama de entidade-relação e esquema relacional, comandos SQL usados para criar as tabelas da base de dados, bem como outros aspetos (como stored procedures) relevantes que foram implementados. Também serão mostrados excertos de código C# que demonstram como implementámos certas funcionalidades que achámos apropriadas para o bom funcionamento da camada de interação, que esperamos que ajudem a elucidar o raciocínio por trás do projeto.

**Cantina: Análise de Requisitos**

A nossa proposta baseia-se na modelação de uma base de dados funcional que permita gerir de modo eficiente uma cantina. De tal modo, identificamos aqui as funcionalidades e aspectos fundamentais que assumimos que o sistema deva possuir para conseguirmos cumprir o nosso objetivo.

 É importante definir as caraterísticas e requisitos essenciais que o nosso projeto apresenta, pois estes guiam-nos ao encontro de uma solução satisfatória e agradável; e portanto, considerámos as seguintes como sendo imprescindíveis no funcionamento de uma cantina:

**Requisitos**:

*Gestão de refeições*

*Gestão de menu*

*Gestão de stocks (Ingredientes)*

*Gestão de funcionários*

*Gestão de clientes*

**Gestão de refeições**

O objetivo principal de uma cantina passa por servir refeições aos seus clientes. Deste modo, é importante ter uma funcionalidade que nos permita detalhar a venda de refeições (cuja composição depende do menu), atualizando informações pertinentes como o momento em que cada refeição é vendida, o histórico de refeições vendidas, o (tipo de) cliente que a comprou, e o(s) funcionário(s) que as serviram.

**Gestão de menu**

Todos os dias existe um menu novo que é composto por um conjunto de pratos. Cada prato é caracterizado pelo seu nome, tipo e pelos ingredientes que o compõem. Os ingredientes são caracterizados pelo seu nome, alergénios (caso tenham) e valor nutritivo. Pretende-se também saber e atualizar a quantidade de ingredientes em qualquer momento, para que se possa fazer uma gestão de stocks precisa e apropriada.

**Gestão de funcionários**

Cada funcionário é identificado pelo seu nome, código interno, salário e email. Cada funcionário tem um cargo atribuído e múltiplos turnos de trabalho. Cada turno pode ter vários funcionários em simultâneo.

**Gestão de stocks (Ingredientes)**

Os ingredientes têm de ser guardados numa dispensa, e cada dispensa tem uma capacidade(teórica) limite de ingredientes. Isto permite saber o estado de abastecimento dos ingredientes que estão ao dispôr da cantina. Se o stock de cada ingrediente for corretamente atualizado na base de dados, então é possível saber exatamente o nº de unidades de cada ingrediente disponíveis.

Quanto à atualização da quantidade de cada ingrediente, é da nossa filosofia que não é realístico esperar que sejam pesados com grande precisão todos os ingredientes servidos em cada prato. Nem que sejam contadas todas as unidades de um ingrediente que são colocadas num prato; uma cenoura colocada num prato pode pesar mais que duas cenouras mais pequenas noutro prato. Mesmo se isso acontecesse, grande parte da comida preparada não é comida/vendida. De modo geral, cantinas tendem a produzir sempre comida a mais, de modo a conseguirem atender qualquer cliente rapidamente durante o serviço; um prato não é “cozinhado” e preparado quando um cliente o deseja – o prato já está efetivamente pronto para ser vendido quando o cliente o pede. Assim, os ingredientes são usados, mas nem sempre fazem parte de algum prato que é vendido. Para preparar a comida, já é plausível expectar que sejam abertos/utilizados pacotes/latas (unidades desta natureza) desses ingredientes. Por exemplo, para preparar a comida do almoço de um certo dia, são abertos 100 pacotes de arroz. Talvez nem metade deste arroz chegue a um prato para ser vendido. Mas se forem utilizados 100 pacotes de arroz para preparar a comida, então não se pode utilizar mais estes 100 pacotes, quer o arroz tenha chegado a um prato ou não. A ideia portanto seria, à medida que se fossem utilizando pacotes de arroz, subtrair-se a quantidade de pacotes utilizado à quantidade de arroz disponível (sendo assim a quantidade de arroz representada por pacotes, em vez de unidades mais comuns, como gramas). Alternativamente, e talvez mais praticamente, no final de cada dia poderia-se subtrair a quantidade de pacotes usada à quantidade disponível. Ambos procedimentos permitiriam manter uma representação precisa e verídica da quantidade real de arroz na dispensa. Além disso, em vez de todos os ingredientes terem as suas quantidades representadas em quilos ou gramas, a sua representação é dinâmica e depende de cada ingrediente; por exemplo, a quantidade de arroz é representada em pacotes, a de cenouras em unidades individuais, a de água em garrafões.

**Gestão de clientes**

Cada cliente é identificado pelo seu nif, nome, email e um tipo que indica qual o preçário a aplicar. Os clientes podem comprar refeições das quais se pretende guardar a referência da fatura. Um cliente só se torna cliente após efetuar uma compra. Assim, quando é efetuada uma compra, o Nif do cliente deve ser adicionado à base de dados caso não exista.

**Diagrama Entidade-Relação e Esquema Relacional**

Atendendo aos pontos anteriormente referidos, procedemos à criação destes modelos para ilustrar as entidades e as suas relações que temos de ter em conta na elaboração da base de dados, bem como a organização e estrutura que esta deve apresentar.

Diagrama Entidade-Relação - entidade-relacao.png

Esquema Relacional - esquema-relacional.png

**Criação da Base de Dados**

Com o auxílio dos modelos acima, seguimos com a criação da Base de Dados. Tendo em conta o que aprendemos durante a unidade curricular, tentámos sempre fazer as coisas corretamente. Criámos tabelas para cada entidade, e em casos de relações M:N, criámos tabelas adicionais para refletir essas relações. Usámos chaves primárias apropriadas, e tivémos cuidados para ter as referências das chaves secundárias certas, criando constraints onde necessário.

CantinaCreate.sql

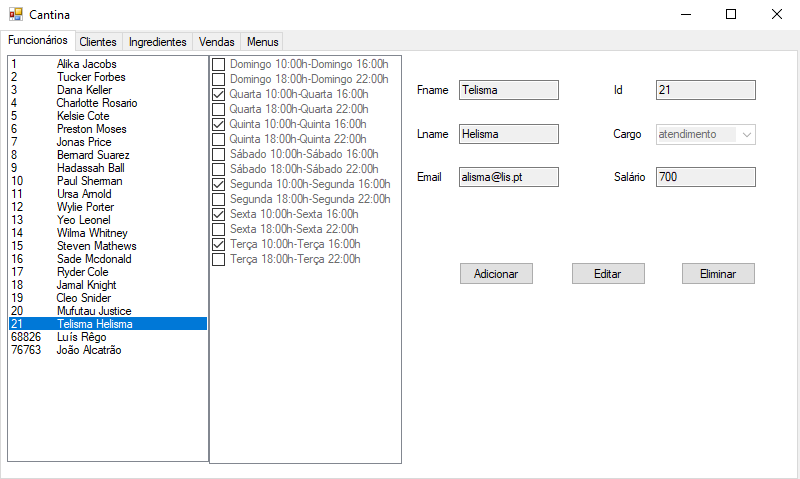
Após criadas as tabelas, fazendo uso da ferramenta generatedata.com populámos a base de dados com informação relevante ao nosso projecto. A base de dados possui ainda outros dados, provenientes das nossas alterações levadas a cargo a partir da interface gráfica.

CantinaInsert.sql

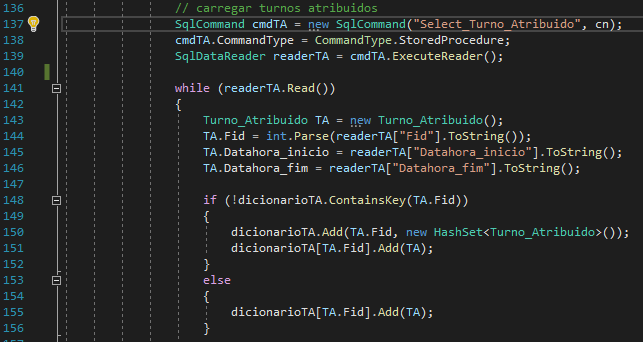
**Interface Gráfica**

Nesta secção vamos mostrar funcionalidade da interface gráfica, que envolvem todo o trabalho efetuado anteriormente e utilizam funções e comandos SQL que foram desenvolvidos de maneira a interagir com a base de dados de forma correta, tendo sempre em prioridade a segurança e consistência dos dados, bem como o desempenho da base de dados.

Começando pela janela a que o utilizador será introduzido quando corre a aplicação, esta tab é dedicada à gestão dos funcionários.

Fig1. Tab dos Funcionários.png

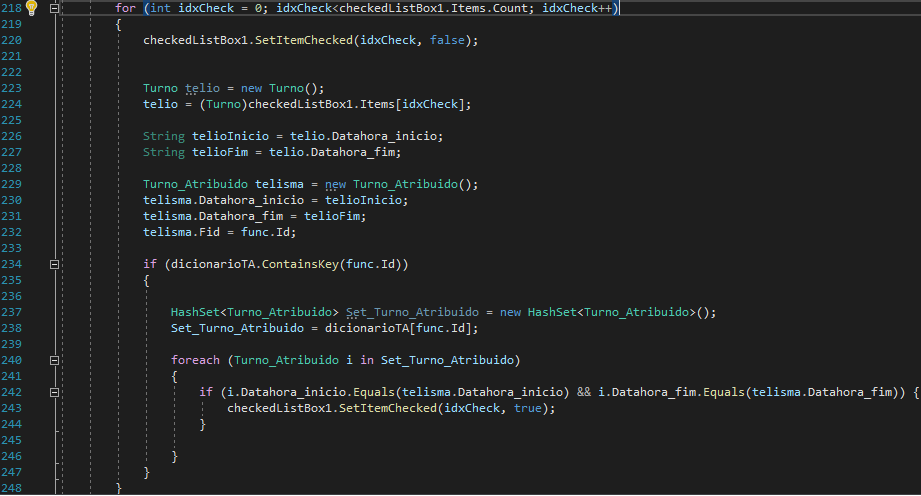
Existem 3 tabelas em foco aqui: a tabela Funcionario (onde cada Funcionário é representado na listBox da esquerda), a tabela Turno (onde cada Turno é representado na checkedListBox à direita), e a tabela Turno\_Atribuido (representada por cada caixa assinalada na checkedListBox do Turno, associada ao Funcionario selecionada na listBox da esquerda). Podemos adicionar, editar, e remover qualquer funcionário. Podemos-lhe (des)atribuir qualquer turno, atualizando assim também a tabela Turno\_Atribuido (cuja chave primária é uma chave composta pelo Id do Funcionário e pelos dois atributos que constituem um Turno, Datahora\_inicio e Datahora\_fim, que marcam o início e fim de um turno, respetivamente). Não podemos adicionar, editar ou remover um turno (da tabela Turno), porque achamos que tal não faria grande sentido neste caso: temos 14 turnos, 2 por cada dia da semana, sendo 1 para o almoço e outro para o jantar. É esta a disponibilidade de serviço que definimos para a nossa cantina.



Vejamos, por exemplo, como os dados da tabela Turno\_Atribuido são carregados:

Fig2. Carregar Turnos Atribuídos.png

Cada Turno\_Atribuido é colocado num dicionário<int, hashset<Turno\_Atribuido>>, onde a chave é o Id do funcionário, e o valor é o conjunto dos turnos atribuídos a esse funcionário. Estes objetos de Turno\_Atribuído são fabricados a partir da aplicação de um reader à Stored Procedure Select\_Turno\_Atribuido, que é uma procedure que faz Select \* From Cantina.Turno\_Atribuido.

Fig3. Excerto da função ShowFuncionario(), dedicada a apresentar no ecrã os turnos a que um funcionário está associado.

O dicionário é utilizado na função ShowFuncionario(), onde cada Turno\_Atribuido é comparado com os Turnos apresentados na checkedListBox, através dos atributos Datahora\_inicio e Datahora\_fim (que são a chave do Turno, e referenciados pela chave do Turno\_Atribuido).

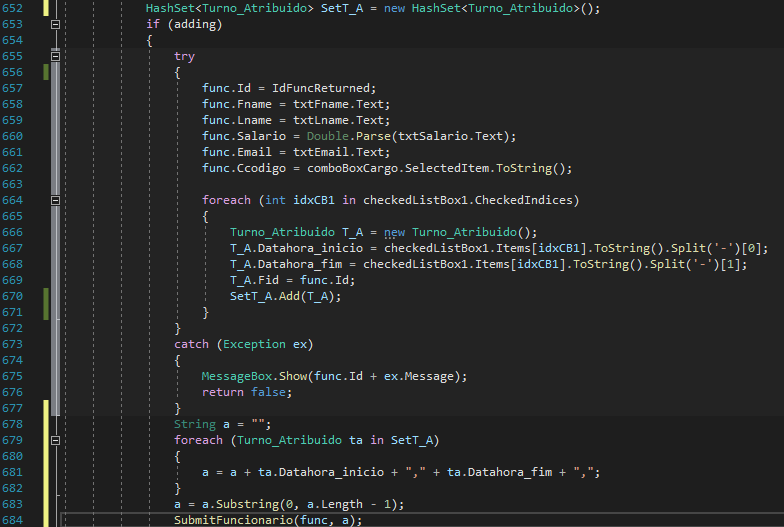
Para o caso de querermos adicionar um funcionário, são criadas as seguintes entidades, 1 Funcionario e 1 hashset<Turno\_Atribuido>;

Fig4. [SubmitFuncionario()](http://submit_func_turno.png)

Os atributos do objeto do tipo Funcionário são preenchidos segundo os valores que o utilizador passa nas textBoxs e comboBoxs. Os atributos de cada Turno\_Atribuido a ser adicionado ao hashset serão iguais aos atributos de cada Turno selecionado na checkedListBox. O Fid de cada Turno\_Atribuido vai ser igual a 0, que é um Id que nenhum funcionário tem ou vai ter. Isto é pouco relevante, porque este hashset é local, e o Turno\_Atribuido a ser adicionado ao dicionário só vai ser adicionado após a inserção do Funcionário, quando já soubermos o Id dele. O importante de retirar do hashset são os atributos Datahora\_inicio e Datahora\_fim, que são posteriormente concatenados numa string, separados apenas por vírgulas (‘,’). Essa string será depois alimentada à função SubmitFuncionário(), que dá insert na base de dados do Funcionário e dos seus Turnos Atribuídos. Como?



Fig5. [Stored\_Procedure dbo.Submit\_Funcionario](http://dbo.submit_funcionario.png)

Esta função chama a Stored Procedure dbo.Submit\_Funcionario, que trata do resto. Dentro de uma transação (para assegurar consistência entre o funcionário e os seus respectivos turnos a serem atribuídos) são executados os comandos para inserir o Funcionário e os Turnos Atribuídos (que são inseridos 1 a 1 dentro da transação), que são obtidos a partir da string anteriormente referida. Dentro desta SP, existe um loop que aplica String\_Split à string, separando-a sempre que deteta uma vírgula, e armazena o resultado numa tabela temporária. A partir dos valores dessa tabela, podemos buscar os argumentos necessários para o comando insert correspondente a cada Turno\_Atribuido. O Id do novo Funcionário, que é um valor auto-incrementado, é devolvido numa variável assim que o comando é executado, assegurando assim que os Turnos Atribuídos são atribuídos ao Funcionário correto.

E de forma resumida, é assim que implementámos esta funcionalidade de adição de Funcionário e respetivos Turnos (Atribuídos) à base de dados. Não estão aqui todos os passos descritos (que se podem encontrar na sua totalidade no ficheiro form1.cs e nas SP referidas), mas esta é apenas uma funcionalidade entre dezenas que implementámos para assegurar um funcionamento adequado, intuitivo, seguro, rápido e consistente da base de dados.

Vamos mostrar mais alguns exemplos do que podemos fazer na base de dados a partir da interface:

Tab de Vendas:

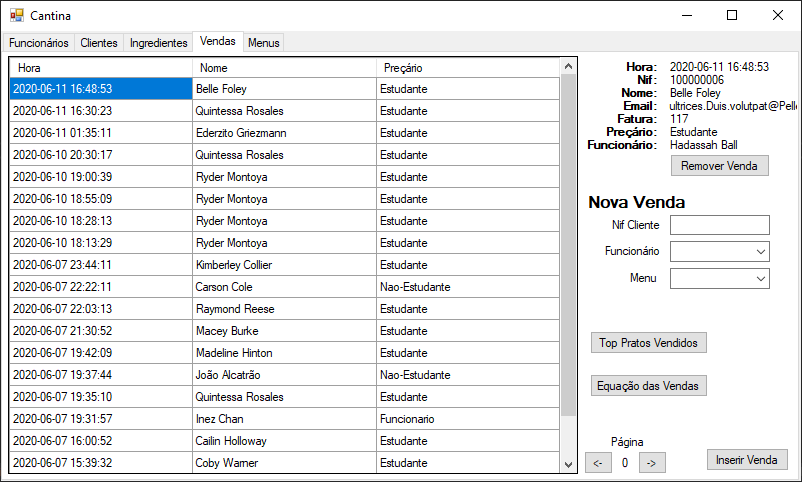


Fig6. Tab de Vendas

Nesta tab é possível ver a lista com a informação mais essencial das últimas vendas ordenadas cronologicamente. Ao fazer double click numa linha, é mostrada uma mensagem que contém informação referente aos pratos associados a essa mesma venda. Como é esperado que uma tabela de vendas tenha um número de registos muito elevado, esta lista é paginada às 20 linhas de cada vez. Esta paginação é conseguida limitando a query a 20 resultados e dando-lhe um offset de 20\*número da página. A query encontra-se na SP “Select\_Vendas”.

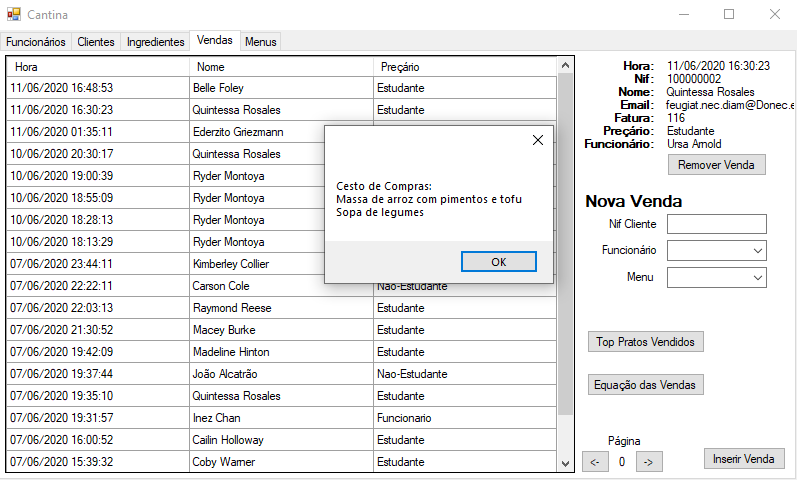


Fig7. Funcionalidade de DoubleClick numa venda (mostra o Cesto de Compras dessa venda).

Do lado direito do ecrã existe mais informação sobre a linha seleccionada e ainda a opção de a remover. Também é possível consultar dados estatísticos relativos aos pratos mais escolhidos pelos clientes assim como os tipos de clientes e as suas compras (esta última funcionalidade apresenta uma equação a partir da qual se pode obter o dinheiro recebido a partir de todas a vendas efetuadas. Basta atribuir um valor monetário aos preçários mencionados [que correspondem ao tipo de cliente]).

Para inserir um novo registo basta introduzir o nif do cliente, selecionar o funcionário que fez a venda, selecionar o menu e por fim escolher os pratos que o cliente leva. Caso o nif inserido não esteja associado a nenhum cliente na base de dados, é mostrada uma janela para completar esse registo. Tanto a inserção como a remoção de uma venda exigem inserir/remover múltiplas linhas em 2 tabelas, e portanto são feitas em transações dentro das SPs “Insert\_Compra” e “Remove\_Compra”.

Tab dos Clientes:

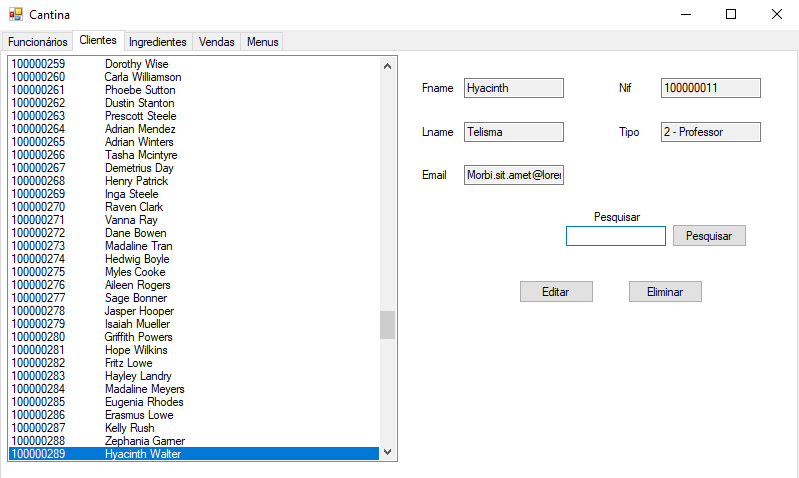


Fig8. Tab dos Clientes (mostrando o resultado de uma pesquisa por “Hyacinth”, que seleciona todos os Clientes cujo método ToString() contenha a expressão inserida, e viaja automaticamente para o último resultado encontrado. Tentar editar todos estes é impossível; apenas o primeiro Cliente selecionado pode ser editado).

Sempre que é efetuada uma venda, com um Nif que não está ainda na BD, é inserida nesta um novo cliente, com esse Nif, que pode ser consultado nesta Tab. Como já referido anteriormente, caso o Nif não esteja associado a nenhum cliente, é mostrada uma janela para efetuar um registo rápido desse cliente. Campos como o primeiro e último nome são opcionais, mas uma Venda precisa obrigatoriamente de um Nif e de um tipo de cliente, para que se possa saber o preço que o cliente vai pagar pela sua compra. À semelhança com o que acontece nas cantinas que nós pessoalmente conhecemos (nomeadamente as cantinas da UA), o preço que um cliente paga por uma compra depende do “estatuto” que lhe é atribuído pela cantina, e não pelos “pratos” que são incluídos na venda (por exemplo, um aluno que leve um prato de Feijoada e uma Sopa, paga exatamente o mesmo que um aluno que leve apenas o prato de Feijoada. No entanto, indivíduos que não são estudantes e que não têm qualquer afiliação com a UA pagam mais pela mesma compra). No caso das cantinas da UA, um aluno pode pagar diretamente com dinheiro ao efetuar a compra, ou pode comprar previamente uma (ou mais) senhas numa das máquinas disponíveis no campus; o preço é constante. No nosso caso, decidimos fazer o mesmo, no entanto, continuamos a mostrar em cada compra exatamente o que cada cliente compra (ao fazer double click numa venda; uma funcionalidade que pode ser útil para determinar os pratos mais vendidos/populares, o que permite saber o que o cliente geralmente gosta e permite a cantina hipotética do nosso projeto preparar os stocks de ingredientes à luz desta informação).

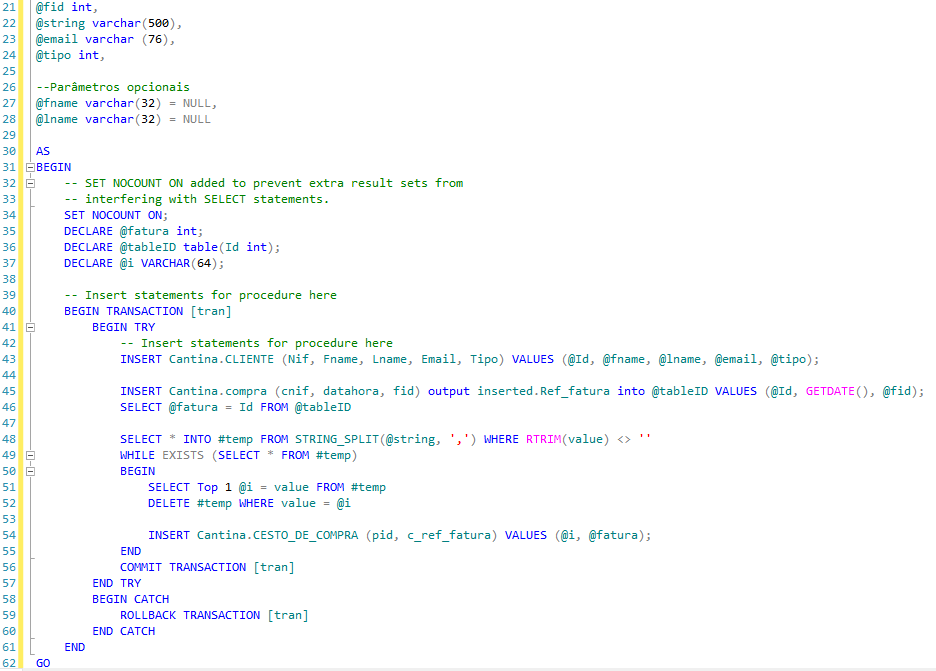
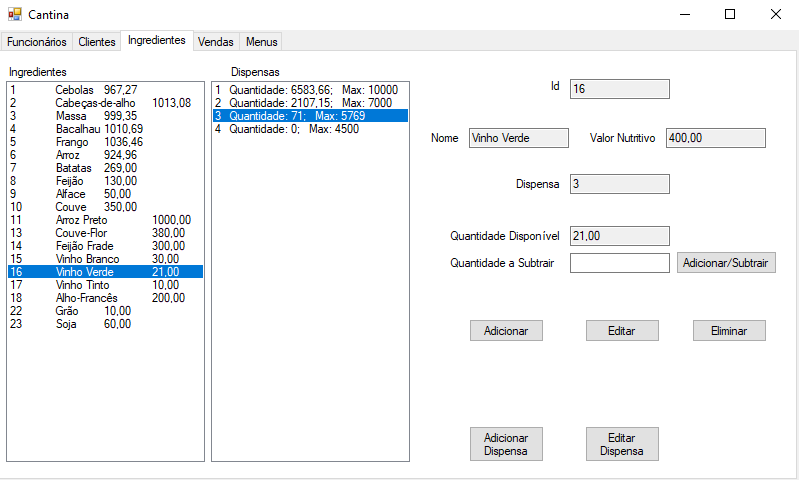
Dito isto, a tab dos clientes na nossa aplicação mostra apenas informação sobre os clientes. É possível alterar qualquer campo deles (exceto o Nif), mas não é possível adicioná-los manualmente; eles são adicionados automaticamente ao efetuar uma venda, caso já não estejam na BD. Este “check” é feito na função buttonVendasInserir\_Click. É feito um select de todos os clientes na BD, e os seus nifs são guardados num hashset. Se o Nif inserido na venda não estiver contido nesse hashset, somos levados a uma nova forma, cujo preenchimento e submissão, se for efetuado com sucesso, executa a SP Insert\_Compra\_E\_Cliente, cujo código se mostra abaixo:

Fig9. Excerto de código da SP dbo.Insert\_Compra\_E\_Cliente. Numa transação, pode-se inserir na BD um Cliente, uma Compra, e todos os itens incluídos nessa Compra (cada um desses itens corresponde a uma row na tabela Cesto\_de\_Compra, que têm uma chave primária composta por 2 atributos que são referências à chave primária “Id” de Prato e à chave primária “Ref\_fatura” de Compra, respetivamente).

Tab dos Ingredientes:



Nesta tab podemos ver os stocks ingredientes à disposição dos cozinheiros da nossa cantina hipotética. Qualquer Prato que possa ser servido têm que ser obrigatoriamente (pelo menos em respeito à BD) constituído por um ou mais dos Ingredientes exibidos na listBox da esquerda. Isto permite obter algum nível de consistência entre o que há disponível e o que se pretende cozinhar e vender.

Todos os Ingredientes têm um Id, gerado automaticamente (e que portanto é a sua chave primária) e vários atributos que podem oferecer informação útil sobre os mesmos (destaca-se aqui a Quantidade Disponível). Cada Ingrediente também está atribuído a uma Dispensa, onde supostamente as suas unidades são armazenadas. Cada Dispensa possui um Id, uma Capacidade Máxima, e uma Capacidade Atual. Sempre que aumentamos o stock de um Ingrediente que está associado a uma dispensa, também aumentamos a Capacidade Atual dessa Dispensa. Se a Capacidade Atual exceder a Capacidade Máxima, aparecerá uma mensagem de aviso, mas a operação é legal; estes atributos servem apenas para dar uma ideia do quão cheia ou vazia se encontra uma Dispensa. Tendo em conta ainda o facto de a Quantidade de cada Ingrediente ser medida em unidades dinâmicas, que podem ser aquilo que o utilizador pretender (bagos de arroz, pacotes de arroz, quilos de arroz, etc.), e que podem ser diferentes das unidades de outros Ingredientes, não conseguimos justificar o acto de fazer o atributo Capacidade Máxima um atributo limitador definitivo da Dispensa. Serve apenas para guiar os funcionários da cantina no bom caminho, com informações relevantes e que dêem uma ideia realista do estado de vários aspetos da cantina, tal como grande parte das funcionalidades que implementámos na interface.

É possível editar a Quantidade Disponível de cada Ingrediente de 2 maneiras: ao editar o próprio Ingrediente, e redefinir uma nova Quantidade Disponível, ou ao inserir um valor numérico na caixa de texto ao lado do botão Adicionar/Subtrair para adicionar ou subtrair esse valor à Quantidade Disponível. É do nosso entender que este último método possa ser mais útil e que “dê mais jeito” no dia-a-dia, para atualizar a Quantidade de cada Ingrediente quando esse Ingrediente é utilizado para fazer comida, quer esta seja posteriormente vendida ou não (cantinas tendem a fazer sempre comer a mais, porque é tipicamente preferível nos interesses de uma cantina que haja comida a mais do que a menos e ter potenciais clientes sem comida). O método da Edição pode ser mais conveniente se houver uma contagem exata por parte dos trabalhadores das unidades de Ingredientes, ou quando são feitos restocks em massa.

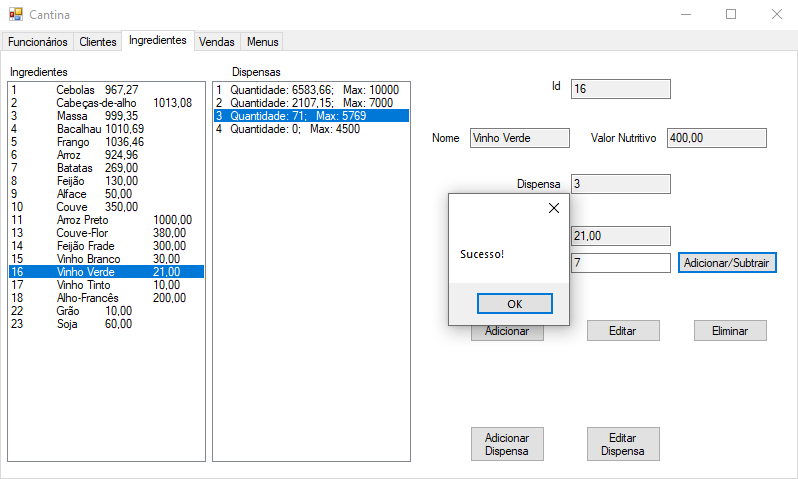
Em termos de implementação desta funcionalidade, em termos simples, ao clicar no butão é chamada a função AdicionarSubtrair\_Click, é calculado o valor aritmético da adição do valor inserido (que pode ser negativo) ao valor da Quantidade Disponível, e o output é passado para a função UpdateIngredienteQuantidade, juntamente com o Ingrediente a ser modificado. Dentro desta função é chamada a SP Update\_Ingrediente\_Quantidade. Caso tudo corra como pressuposto (o input tem de ser um valor numérico, e não podemos tentar Quantidades negativas), o Ingrediente passará a possuir a Quantidade Disponível pretendida.

Fig10. Subtrair 7 unidades de Vinho Verde às 21 unidades disponíveis. Sucesso!

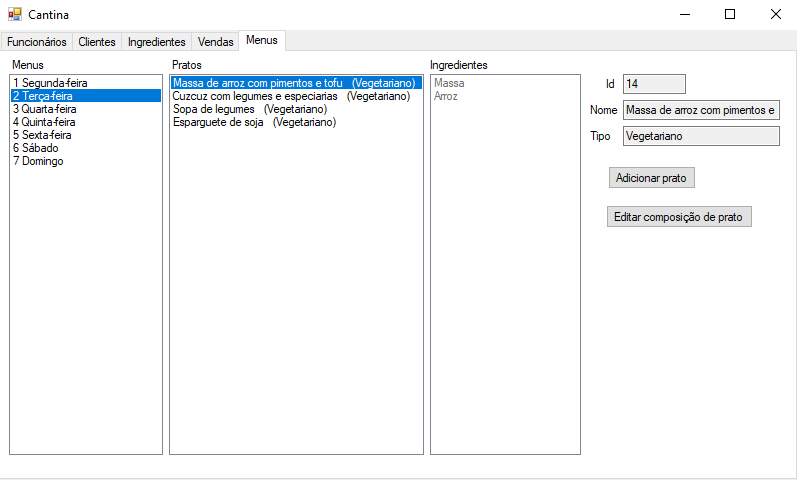
Tab dos Menus:

Fig11. Menus disponíveis, e o que há para comer em cada Menu. O Menu tem uma Composição, que corresponde aos Pratos que são preparados para esse Menu. Embora não seja usual mostrar a Composição de cada Prato, e sem dúvida que os cozinheiros sabem preparar a refeição sem consultar a sua Composição (até porque é necessário muito mais do que a composição de um prato para o preparar propriamente), decidimos mostrar os Ingredientes que compõem cada Prato. Assim, após adicionar um Prato (cuja atribuição de Id é feita automaticamente), é possível editar a sua Composição (como referido anteriormente, apenas com Ingrediente que estejam ao dispor da Cantina). Mais uma vez, estas funcionalidades têm valor na medida que ajudam a orientar os funcionários e a disponibilizar informação atualizada (e facilmente atualizável) sobre o que se vende na Cantina em cada dia.

**Conclusão**

A maior parte das funcionalidades não foi aqui explorada ou sequer mencionada; são demasiadas para tal tarefa. No entanto, ao usar a aplicação, pode-se explorar em maior detalhe aquilo que a Interface oferece. As funcionalidades mais relevantes foram aqui apresentadas, e os seus objetivos expostos e as suas implementações brevemente resumidas. As SP que criámos também não foram todas apresentadas neste relatório, nem as suas implementações mostradas, mas todas estão disponíveis noutra parte deste trabalho, tal como pedido. Não sentimos necessidade de utilizar alguns dos métodos que nos foram ensinados tal como cursores (preferimos usar While statements, porque nos são mais intuitivos, mais rápidos e conseguem satisfazer o mesmo propósito), UDFs (cada caso foi um caso, até mesmo nas SP que fazem processamento e split de strings, a maneira e o número de comandos a executar por cada split eram diferentes, e portanto não encontrámos justificação para usar estas ferramentas) ou triggers (novamente, nas situações que se justificaria o seu uso, conseguimos o efeito desejado de outras maneiras que nos pareceram igualmente válidas). Usámos transações sempre que foi necessário executar mais do que um comando SQL num mesmo procedimento, de modo a garantir a consistência e dos dados da BD. Usámos preferencialmente os comandos de transação inerentes ao SQL Server (Begin Transaction, Rollback, etc.), mas em situações em que eram chamadas diferentes SP como um bloco único a ser executado, utilizámos várias vezes a classe TransactionScope(), disponível no Windows Forms, que apresenta o mesmo propósito que os métodos de transação SQL apresentam: garantir atomicidade, consistência, isolamento e durabilidade das modificações efetuadas na BD. Da mesma maneira, todo o código SQL está contido dentro de SPs, de modo a garantir segurança e melhor desempenho na BD. Usámos os métodos AddWithValue disponibilizados no Windows Forms para passar parâmetros para o código SQL a ser executado nas SPs, oferecendo assim uma camada de proteção contra SQL Injections.

No nosso entender, utilizámos o que precisámos e que era adequado para criar uma BD e uma interface funcionais e seguras, e conseguimos cumprir com os requisitos impostos para o projeto e seguimos as boas práticas lecionadas na unidade curricular ao longo do semestre. Objetivo cumprido.