



Sistemas de Informação 2

2015/16 Inverno

Trabalho Prático - fase 2

Grupo 19

Henrique Calhó nº38245

Tiago Magalhães nº28593

Docente

Nuno Datia

Índice

[Índice](#)

[Introdução](#)

[Alterações ao modelo](#)

[Alteradas as seguintes tabelas:](#)

[Criados os seguintes procedimentos:](#)

[Aplicações .net](#)

[a \) Listar os tickets não fechados existentes no sistema](#)

[b \) Atribuir um técnico responsável ao ticket mais recente](#)

[c \) Inserir uma acção associada a um ticket](#)

[d \) Remover um ticket](#)

[e \) Exportar a informação de um ticket](#)

[Aplicação - objectos connected ado.net](#)

[Comparação entre Disconnected e Connected](#)

Introdução

Esta fase do trabalho tem como propósito criar aplicações .net que simulem um sistema de tickets usado em balcões de apoio ao cliente, aproveitando a base de dados criada na primeira fase.

Neste relatório pretendemos apresentar a nossa implementação e as explicações necessárias das soluções adoptadas.

Como conclusão serão analisados os conhecimentos adquiridos.

Alterações ao modelo

Decidimos fazer pequenas alterações ao modelo que nos facilitaram a implementação das aplicações .net e garantiram que a informação dos tickets estava coerente com o xml schema.

Alteradas as seguintes tabelas:

● Ticket_User

- adicionado um atributo id, incrementado com identity (1,1), de forma a que seja única para cada user.
- a chave primária permace o email

● Ticket_Action

- adicionado um atributo orderNumber, que passa a ser chave em vez de beginDate. Indica o índice da ação dentro do ticket

● Request

- adicionado um atributo orderNumber, que passa a ser chave em vez de requestDate. Indica o índice da ação dentro do ticket

Criados os seguintes procedimentos:

● Proc_Assign_Technician

- recebe dois parâmetros:
- um de entrada - @cod - o qual indica o código do ticket a remover
- um de saída - @res - o qual indica se a operação foi bem sucedida (true(1) ou false(0))
- o código começa por verificar se o técnico existe na base de dados. Caso não exista retorna @cod = ''.
- se existir, obtém-se o código do ticket mais recente que ainda não tem nenhum ticket associado, sendo esse código armazenado em @cod, e é feito um update desse ticket para ter o @tech_num do técnico associado
- caso não exista um ticket sem técnico associado, @cod será null, e a informação será apanhada como situação de erro

● Proc_Remove_Ticket

- recebe dois parâmetros:
- um de entrada - @cod - o qual indica o código do ticket a remover
- um de saída - @res - o qual indica se a operação foi bem sucedida
- começa por verificar se o ticket pretendido existe e se não foi removido previamente.
Se uma dessas condições não se realizar, retorna-se @res = 0
- em caso afirmativo, chama-se o procedimento sp_Delete_Ticket e coloca-se @res = 1, para indicar que a operação foi bem sucedida

● Proc_Get_Ticket_Info

- recebe um parâmetro de entrada - @cod - o qual indica o código do ticket pretendido
- apresenta toda a informação necessária para se criar um ficheiro xml
- a utilização deste procedimento é opcional. No código para o evento, a escolha de usar este procedimento foi para não ser preciso criar uma query string comprida, mas como é óbvio este procedimento limita a informação obtida
- caso se pretendesse outras peças de informação, seria necessário alterar o procedimento, o que não é recomendável, pelo que seria preferível alterar uma query string do sqlCommand

Foi necessário uma aplicação para cada pergunta pois o acesso a dados é feito de maneira diferente mas ambas apresentam a mesma estrutura.

Estas são sustentadas num Form_Main, a qual apresenta cinco opções ao utilizador, cada uma associada a um diferente Form. A escolha de uma opção abre uma nova janela, onde se pode realizar a operação pretendida. Caso se escolha uma opção quando outra já está a ser utilizada, a janela desta última fecha-se para abrir uma para a nota (só pode ser feita uma opção de cada vez).

a) Listar os tickets não fechados existentes no sistema

Form_ListTickets apresenta num DataGridView, todos os tickets não fechados.

b) Atribuir um técnico responsável ao ticket mais recente

Form_AssignTechnician apresenta uma textBox para inserção do valor do técnico que se quer associar ao ticket mais recente que não tem nenhum técnico associado.

Apresenta o resultado da operação numa MessageBox.

c) Inserir uma acção associada a um ticket

Form_InsertAction apresenta uma textBox para se introduzir o código do ticket que se pretende inserir uma acção. Se o ticket existir e tiver um ticket_type, todos os steps associados a esse ticket_type são apresentados numa DataGridView, assim como existem três textBox para inserção de:

- número do técnico que vai realizar a acção

- id do step sobre o qual se pretende realizar a acção
- nota sobre a acção a realizar

O resultado da operação é apresentada numa MessageBox.

d) Remover um ticket

Form_RemoveTicket apresenta uma textBox para inserção do código do ticket a remover, sendo o resultado da operação apresentado numa MessageBox.

e) Exportar a informação de um ticket

Form_ExportTicketToXML apresenta duas textBox para inserção de:

- código do ticket sobre o qual se pretende a informação
- nome do ficheiro xml que se pretende criar

O resultado da operação é apresentada numa MessageBox, e em caso de criação o ficheiro estará armazenado no local por omissão da aplicação.

Aplicação - objectos connected ado.net

Para o uso dos objetos conectados do ado.net, é necessária uma connection string. Esta é armazenada nas propriedades do projecto (ou no ficheiro App.config), a qual pode ser alterada pelos diferentes utilizadores.

A base de dados é local. Esta foi uma escolha feita por limitação de tempo, porque seria mais “correcto” utilizar-se uma base de dados hosted num site público. No entanto, estando local é mais fácil para os testes individuais, mas os dados não estão realmente a ser partilhados por diferentes utilizadores.

As ligações à base de dados são feitas com connection pooling, sendo utilizado código de try-catch-finally para apanhar potenciais erros. Caso exista um erro, a connection pool é limpa, e tenta-se novamente o acesso.

Como a maior parte dos exercícios só requerem uma interrogação à base de dados, tanto SqlConnection como SqlCommand estão contidos dentro de using(...){}. Isto implica que não é preciso ser feito dispose dos seus recursos, porque é feito automaticamente.

1. Listagem de todos tickets não fechados

Esta opção, associada ao Form_ListTickets, faz o seguinte query à base de dados:

```
- "select * from dbo.vi_Ticket where ticketState!='closed'"
```

O resultado é armazenado num SqlDataReader dr, sendo os seus valores colocados no DataGridView, enquanto existir dr.Read() for true.

2. Associar um técnico ao ticket mais recente

Esta opção, associada ao Form_AssignTechnician, cria um SqlCommand cmd associado ao stored procedure "proc_Assign_Technician", sendo preciso designar cmd dessa forma, através de cmd.CommandType = CommandType.StoredProcedure, assim como associar a cmd dois SqlParameter, um dos quais o output.

O resultado do query é armazenado no parâmetro ticket, sendo feito um teste para interpretar o resultado obtido.

3. Inserção de acção num ticket

Esta opção, associada ao Form_InsertAction, requer quatro comandos.

- O primeiro é uma interrogação à base de dados para se todos os steps associados ao ticket passado como argumento. Se o ticket não existir ou não estiver num estado de "in progress", a operação para e lança-se o respectivo aviso.
- O segundo comando é um pedido a um stored procedure, "sp_Insert_Ticket_Action", no qual se passam os diferentes parâmetros, do qual se obtêm o número de identificação da acção associada ao ticket.
- Após confirmação do fecho dessa acção (através de um OK da MessageBox) utiliza-se o um comando para utilizar o stored procedure "sp_End_Ticket_Action" para que a acção seja terminada e actualizada com endDate.
- Com a confirmação do fecho da acção, o utilizador é questionado se a acção resolveu o problema (através de um OKCancel da MessageBox). Em caso afirmativo, o sistema verifica se o técnico é o técnico responsável pelo ticket. Se não for, nada acontece. Se assim não for, utiliza-se um comando para utilizar o stored procedure "sp_Close_Ticket", de forma a encerrar o ticket, colocando-lhe o estado "Closed".
- Por fim faz-se o dispose dos recursos de SqlConnection, porque ao contrário dos restantes exercícios, neste caso não está contido dentro de um using(..).

4. Remoção de um ticket

Esta opção, associada ao Form_RemoveTicket, requer um único SqlCommand para se utilizar o stored procedure "proc_Remove_Ticket".


5. Exportação da informação de um ticket para XML

Esta opção, associada ao Form_ExportTicketToXML, pode ser dividida em três etapas:

- A primeira requer o uso de uma ligação à base de dados para se obter a informação pretendida do ticket. No código apresentado, utiliza-se um command para aceder ao stored procedure "proc_Get_Ticket_Info", mas, como já foi mencionado previamente, tem a limitação de não se poder pedir diferentes informações. Tal é resolvido utilizando uma string query para se criar a interrogação desejada. O resultado da interrogação é armazenado num SqlDataReader.
- A segunda etapa consiste em se popular um DataSet, cuja estrutura é carregada a partir de um XML Schema, fornecido no enunciado. A população do DataSet é feita através das suas DataTables, cada uma sendo preenchida com os valores correspondentes, contidos no SqlDataReader.
- A última etapa consiste em se criar o ficheiro XML a partir do DataSet, através do método WriteXML, que recebe como parâmetro o nome do ficheiro XML a criar.

Comparação entre Disconnected e Connected

Existem várias diferenças entre os dois modelos criados.



O modelo connected requer uma ligação à base de dados, pelo que para além de uma connection string, também precisa de código para estabelecer e fechar essa ligação, tendo que fazer dispose dos recursos usados.

Para a criação de ficheiros xml, o modelo Connected precisa de criar um DataSet, definir a sua estrutura e ainda adicionar-lhe os dados pretendidos para o ficheiro. Esta diferença nota-se com o modelo Disconnected, no qual os dados já estão contidos dentro do DataSet a partir do load de um ficheiro xml.

Isto implica que num modelo Connected a criação de um ficheiro xml tem mais trabalho de codificação do que num modelo Disconnected.

Uma vantagem que o modelo Connected apresenta é a possibilidade de alterações à base de dados.

Enquanto que um modelo Disconnected só se pode alterar o próprio ficheiro xml, no modelo Connected é possível de se alterar uma base de dados comum a diversos utilizadores (neste código tal não acontece, mas seria só uma situação de se alojar a base de dados num site para se demonstrar). Isto significa que enquanto que num modelo Disconnected, todas as alterações ao xml ficam contidas nesse ficheiro, podendo ocorrer erros de conflito quando se fizer a sincronização com a base de dados. No modelo Connected os conflitos tendem a ser menores.

Isto implica que o modelo Disconnected não é o mais eficaz na inserção e alteração de dados. Mas só para a leitura de dados é mais rápido que o modelo Connected.