ISEL
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Trabalho Pratico II

Sistema De Gestão De Facturas FSolv

44600 Vladyslav Vorobyov

45371 Gonçalo Costa

Docentes: Afonso Remédio

Nuno Datia

Relatório parte II realizado no âmbito de Sistemas de Informação II,
do curso de licenciatura em Engenharia Informática e de Computadores
Semestre de Inverno 2020/2021

Janeiro de 2021

Resumo

Este projeto consiste na realização no âmbito da disciplina de um sistema de gestão de faturas para a empresa fictícia Fsolv.

O projeto tem como requisito a gestão de informação relacional do cliente, nesta fase será criada uma aplicação em C#, que implemente através do uso da plataforma .NET, as funcionalidades de objetos conectados (*ADO*.net) e EF (*EntityFramework*) na interação com a base de dados.

Índice

1. Introdução

Com a realização do projeto pretende-se desenvolver as capacidades do grupo na área das bases de dados. Nomeadamente na criação de modelos relacionais, controlo transacional, procedimentos armazenados, gatilhos e funções, adicionalmente também no desenvolvimento de aplicações C# que utilizem a *.NET framework*.

1.1 Ferramentas

- Microsoft SQL Server
- SQL Management Studio
- Visual Studio 2019

Como sistema de gestão de bases de dados foi utilizado Microsoft SQL Server.

Este dispõe de toda a infraestrutura que permite o alojamento e controlo da base de dados implementada.

Ambientes de desenvolvimento como *SQL Management Studio* disponibilizam de um conjunto de ferramentas que facilitam o acesso e execução de scripts na base de dados.

Nesta segunda fase é necessário um ambiente de desenvolvimento C# que é especialmente dedicado ao *.NET Framework* (entre outros).

1.2 Organização do documento

O relatório está organizado da seguinte forma:

- No capítulo 2, esta o enunciado do problema em questão
- No capítulo 3, está apresentamos a implementação
- No capítulo 4, explicamos como cumprimos as funcionalidades requeridas
- Por último, é elaborada uma conclusão do projeto.

2. Formulação do Problema

Nesta fase do trabalho pretende-se que os alunos criem uma aplicação que use diferentes *frameworks* de acesso a dados. A aplicação deve ter como meta a reutilização de código, fácil manutenção e eficiência, e ser independente do modo de acesso a dados.

É importante saber demonstrar os resultados de todos os processos envolvidos na segunda fase.

3. Implementação de aplicação .NET

Para a implementação da aplicação descrita na segunda parte do enunciado foi necessário dividir o processo em 3 partes:

- Implementar uma estrutura de acesso a dados que utilize objetos conectados do *ADO.NET*.
- Adicionar uma implementação de acesso a dados que utilize *Entity Framework*, tendo em conta que o objetivo é manter o máximo de código idêntico.
- Criar interfaces genéricas (*DAL*) que permite um controlo genérico sobre ambas as estruturas.

Havendo determinado os pontos importantes do trabalho pratico iniciamos um período de introdução às tecnologias e suas implementações no contexto atual.

3.1 *ADO.NET*

Sendo uma parte integrante da infraestrutura *.NET*, é definida por um grupo de classes que pode ser utilizado para aceder aos dados de uma base de dados. Esta tecnologia permite, de forma totalmente remota e desconectada do repositório, interagir com os dados presentes. Podemos ainda dividir as classes em duas secções: conectados e desconectados.

No âmbito deste projeto só necessitamos de objetos conectados como definidos pelo enunciado.

A estrutura é a seguinte:

- ***ADOContext***
  Este objeto inicia uma instância de repositório para cada tipo de objeto para temporariamente ser ocupado durante o contexto. Irá ter a sua conexão dedicada, através de um *SQLConnection* obj, aquando criado, que pode ser fechado quando a sua utilidade acaba. Assim permite ao programa aceder às funcionalidades da base de dados, no entanto num nível de abstração ainda elevado.

- ***Repository***
  Este objeto é responsável por submeter alterações do objeto ao nível inferior (*Mapper* correspondente), isto acontece durante a chamada de *SaveChanges*(), portanto quando algum item é criado/eliminado/alterado a mudança só ocorre quando é submetido. (Por implementar)

- ***Mapper***
  Os diferentes *Mappers* servem para fazer interação direta com a BD, a um nível inferior e mais próximo que o *context*, sendo que este utiliza as propriedades dos objetos *Mappers* para a interação que não seja *Lazy Load*. É neste nível que se encontram as operações *CRUD* executados diretamente através da conexão.

- ***Proxy***
  Os objetos Proxys servem para permitir a função Lazy Load de objetos Models para serem depois utilizados.

- ***Model***
  Os objetos *Model* são a representação das tabelas presentes na base de dados, é com as suas características que podemos com certeza afetar e ler os dados.

3.2 Entity Framework

Esta é uma *Framework* de mapeamento orientado por objetos, especifica para *ADO.NET*, o seu propósito é utilizar a mesma tecnologia que os objetos conectados, mas apresentando uma arquitetura mais compreensiva e intuitiva de utilizar. Assim oferece aos *developers* a oportunidade de não se preocuparem com as tabelas e colunas subjacentes ao software que estão a escrever/trabalhar com, desta forma apenas têm de conhecer os objetos específicos e as suas propriedades, em vez de saber tudo sobre onde e como os dados deles são guardados. Em suma, é uma tecnologia que oferece a possibilidade de subir um nível de abstração e agilizar o processo de criação de uma aplicação cliente.

Algumas anomalias notadas durante o desenvolvimento foram: o código gerado automaticamente pode gerar tipos anormais como *nullable*, quando na origem esse valor nunca seria *null*.

- ***PContext***
  Semelhante no que oferece em relação à versão objetos conectados, mas tendo em conta uma realidade de entidades ligadas.
- ***DbSet***
  Similar ao repositório, mas encontra-se na integra definido e configurado dentro do contexto.
- ***Entidades***
  Desempenham a mesma função que os *models*, apenas representam as tabelas.

Como o código é maioritariamente uma adaptação das funcionalidades exigidas para um ambiente EF todos os conceitos definidos desempenham uma função substituta, mas com a mesma finalidade. Para além destas classe existem outras mas não são abertas ao exterior.

Requisitos funcionais

4.1 Segunda Fase

4.1.1 *ADO.NET*

4.2.1.1 Aceder às funcionalidades implementadas anteriormente

Através da estrutura implementada no âmbito de objetos conectados é possível interagir com os dados. Utilizando estas funcionalidades cumprimos este objetivo ao adicionar aos *Mappers* corretos as funções que permitem interagir com os níveis inferiores de maneira consecutiva até ser guardada a mudança. Por exemplo, a criação de uma factura sem items mas com um contribuinte, através da criação e configuração de uma classe *Mapper* na qual estão todos os métodos necessário à interação do programa com os dados subjacentes, permitindo a criação da entidade com `Create()`, a eliminação com `Delete(TEntity entity)`, a inserção na tabela com `Insert(TEntity entity)`, a leitura da informação com `Read(params object[] keyValues)` e a atualização dos conteúdos com `Update(TEntity entity)`. Havendo completado a implementação total referente à entidade Factura_Contribuinte, incluindo *proxy* e *model*, podemos selectivamente utilizar a funcionalidade descrita anteriormente(entre outras), isto aplica-se a todas as alíneas da primeira fase que sejam obrigatórias "traduzir" para um ambiente *.NET* através de *CRUD*. As mudanças são depois guardadas quando o *context* em que surgem seja submetido. Esta funcionalidade é acessível através do programa principal.

4.2.1.2 Implementar funcionalidades sem aceder ao trabalho anterior

Como funcionalidade sem acesso aos *Stored Procedures*, foi feito na *layer* de *services* a operação de recolha total de facturas e a comparação lógica até encontrar a mais recente, depois é recolhido o ano atual e compara-se com o ano na última factura emitida, caso seja igual, apenas adicionamos 1 ao código e apresentamos na consola, caso seja diferente, é apresentado o primeiro código para o ano atual. Esta funcionalidade é acessível através do programa principal.

4.2.1.3 Emissão de factura

Como esta funcionalidade já existe em forma de *Stored Procedure* e a restrição anterior não se aplica, simplesmente existe a possibilidade de executar a partir do contexto o comando que corre o "p_facturaEmitir" presente na *API* criada na fase anterior do projeto.

4.1.2 *EF*

4.2.2.1 Alterar aplicação para implementar funcionalidades *EF*

A estrutura *EF* foi implementada com acesso a todas as mesmas funcionalidades de forma a alterar o mínimo de código possível. Possível observar e testar a partir do menu do programa.

4.2.2.2 *Optimistic Locking*

Foram utilizadas duas instâncias de *EF* em *debug* para parar o processo apos ser feita a leitura dos valores, dessa forma durante a fase em que o *EF* guarda os dados na BD é produzido um erro numa das instâncias devido à concorrência.

4.1.3 Testes comparativos

4.2.3.1 Objetos conectados VS *Entity Framework* (1C)

Durante a alteração do código e posterior fase de testes de performance, foi possível observar duas coisas de extrema relevância quando há que decidir entre as duas abordagens. A primeira observação é, o tempo de desenvolvimento do *EF* é bastante menor face tempo gasto para os objetos conectados, assim torna o processo de criação de aplicações bastante mais ágil com código gerado automaticamente, entre outras características. A outra observação foi, a performance de *EF* é ligeiramente pior quando os objetos conectados são implementados da maneira mais correta.

4.2.3.2 Comparação de tecnologias em relação à garantia de consistência de dados.

Após o desenvolvimento total da aplicação com ambos os objetos conectados e o *EF*, a conclusão final foi: a ambos conseguem garantir a consistência de dados, no entanto utilizando os objetos conectados será preciso mais tempo a desenvolver para garantir o mesmo resultado, visto que o código gerado automaticamente encarrega-se disso no âmbito do *EF*.

5. Conclusões

Em conclusão, ambas as tecnologias têm os seus méritos e propósitos, no entanto para um processo mais ágil de desenvolvimento de aplicações de emissão de facturas será preferível a utilização de *Entity Framework*, não só pelo nível de abstração que oferece, permitindo a fácil manutenção e alteração posterior, mas também pela velocidade de processamento quando operado corretamente. O grande contratempo desta tecnologia é o cuidado que é necessário na definição de *locks* ao acesso, visto que num contexto de grande volume transacional manter registos *locked* com *pessismistic Locking* pode resultar na perda de toda a vantagem que se ganha, mas o inverso pode gerar uma situação de constante retrocesso e reinício da ação, porque a versão do registo sobre o qual a máquina opera pode não ser o mesmo que o do programa no momento da interação.

O grupo está agora confortável com todo o processo de desenhar, projetar e implementar uma aplicação de acesso a dados deste género e das diferenças no tipo de abordagem tomada referente às tecnologias de acesso ao nosso dispor.

Referências

Patterns of Enterprise Application Architecture,1ªedi., Martin Fowler, isbn-13: 978-0321127426

Pragmatic ADO.NET: Data Access for the Internet World, 1ªedi., Shawn Wildermuth, isbn-13: 978-0201745689

Microsoft .NET documentation, Entity Framework