Considerações sobre o projeto desenvolvido

Autor: Denny Sakakibara

Inicialmente, me planejei em desenvolver três módulos para ser possível solucionar as necessidades básicas do problema. Logo, desenvolvi três projetos que foram adicionados dentro de uma *Solution, sendo eles: Business, Core e Repository.*

No projeto Core, foram declaradas as classes utilizadas em, praticamente, todo o sistema. As classes foram organizadas por pastas focando em uma organização por módulos, por exemplo, a pasta *Cadastro* fica responsável em armazenas as classes que representariam os objetos principais de uma funcionalidade. Outro exemplo, a pasta *PontuacaoCasaPopular* armazena todas as classes que envolvem na logística e controle de informação das pontuações. Enumeradores também tem o seu lugar reservado nesse projeto.

Tendo o *Core* pronto, foi desenvolvido o *Repository* responsável pelo mapeamento e persistência de dados. Foi desenvolvido um repositório genérico (*RepositorioGenerico.cs*) para ser utilizado em todos os repositórios, fazendo com que todos tenham e utilizem os mesmos métodos para manipular os dados (*IRepositorioGenerico.cs*). Por exemplo, todos os repositórios têm um método obter, inserir e atualizar. Caso tenha que ser desenvolvido algum método especifico para uma determinada tabela, é possível que seja codificada no seu específico repositório.

A Business é a camada que possui toda a regra de negócio. A organização das pastas foi feita de forma a facilitar visualizar as regras por módulo. Por exemplo, *FamiliaBusiness* é responsável por validar, inserir e atualizar de acordo com uma determinada regra sobre família. Outro exemplo, *CriterioPontuacaoBusiness* pode dar manutenção de cadastros de critérios e também guarda as regras de como é feita a avaliação de cada regra de critério.

Agora que essas três camadas foram desenvolvidas, vamos observar as soluções dos problemas. Não é viável que o cálculo das pontuações seja feito a todo momento que é requisitada a lista de aptos. Sendo assim, foi pensado em calcular e armazenar os pontos previamente. A cada inserção ou edição do registro de família é feito o cálculo das pontuações, com exceção nos casos em que a família já foi beneficiada. Realizar o cálculo em cada inserção ou edição do dado parcializa esse processamento de pontuação quando o número de famílias é gigantesco, deixando a informação pronta para ser apenas requisitada e filtrada futuramente.

O banco de dados não foi implementado, porém as classes foram implementadas imaginando o espelhamento do banco. Foram criadas as tabelas *Familia*, *PontuacaoFamilia* e *CriterioFamilia*. Família representa a própria família. A tabela *PontuacaoFamilia* foi criada para separar as responsabilidades de cadastro de família e controle da pontuação, por isso, as informações de pontuação não foram adicionadas diretamente na tabela família. *CriterioFamilia* é uma tabela de controle de critérios existentes, possui uma coluna de situação onde é possível ativar ou inativar um determinado critério sem ter que mexer diretamente na publicação da aplicação.

Para a solução do primeiro problema do desafio, foi implementado na *FamiliaBusiness* dois métodos:

public IQueryable<Familia> ObterAptasGanharCasaPopular()

e

public void ConsultarPaginacaoAptasGanharCasaPopular()

O *ObterAptasGanharCasaPopular()* retorna todos as famílias aptas a ganhar a casa, a lista é ordenada da maior para a menor pontuação, favorecendo as famílias melhores pontuadas. Esse método atenderia bem um cadastro com um volume baixo ou médio de famílias.

Para solucionar o problema de uma consulta em um cadastro gigantesco, foi desenvolvido o método *ConsultarPaginacaoAptasGanharCasaPopular()*. Esse método faz consultas paginadas, exigindo menos de processamento que o método que carrega todos de uma vez só.

A questão do controle de cálculo da pontuação é feita na *PontuacaoFamiliaBusiness*. Ela tem a responsabilidade de guardar a regra. O cálculo é feito no método *CalcularPontuacao(Familia familia)*. Esse método obtém todos os critérios ativos e verifica quais itens a família atende.

No problema a ser solucionado seguinte, o de comunicar um outro módulo sobre as famílias que já foram beneficiadas, eu imaginei dois possíveis cenários. O primeiro cenário eu imaginei uma comunicação entre módulos do próprio sistema. A solução mais simples é deixar consultas preparadas para esse outro módulo consumir, sendo eles *ConsultarPaginacaoFamiliasContempladas()* e *ObterContempladosCasaPopular()*, ambos na FamiliaBusiness. O segundo cenário que eu imaginei foi uma comunicação com um módulo externo, em que uma terceira empresa precisaria dessas informações. Foi nesse segundo caso que eu imaginei em implementar um webservice/WCF para das acesso a essa informação. A empresa que tiver acesso a esse WCF poderá com apenas um link ter noção das consultas e retornos que ele pode ter. Através dos serviços e contratos do WCF, é possível visualizar a estrutura dos parâmetros e retornos. No WCF, é possível formatar e encaminhar a informação do jeito que o usuário precisa. O WCF obtém a informação na camada business e formata a informação, passando apenas aquilo que está declarado no *DataContract*, evitando que o usuário tenha acesso a informações sigilosas.

O problema de não permitir que famílias já beneficiadas recebessem novamente esse privilégio foi facilmente resolvido adicionando um filtro por status na consulta.

Caso o cliente queria adicionar mais critérios no cálculo da pontuação a equipe poderá facilmente adicionar mais um item no switch do método *AtendeCriterio(Familia familia, CriterioPontuacao criterio)* da *CriterioPontuacaoBusiness*. E caso o cliente queria inativar um critério, poderá apenas utilizar o método para inativar um critério específico. Outra solução para facilitar a manutenção dessa informação é criar uma página web que possibilitará a inativação/ ativação de um determinado critério. Para adicionar um novo critério teria que esperar uma manutenção do código para assim poder publicar futuramente.

Imaginei um outro cenário que o cliente precisasse recalcular todas as pontuações das famílias ainda não beneficiadas. Como o processamento de dados é gigantesco, imaginei que o cálculo poderia ser através de realizações parciais. Por exemplo, o processamento poderia ser realizado em vários dias executando em um horário que o servidor estaria menos requisitado, esse horário poderia ser na madrugada. Para isso, implementei um Windows Service. Esse serviço seria instalado no servidor da aplicação. Desenvolvi um controle onde podemos escolher o horário em que esse processamento será executado. Poderíamos, por exemplo, executar o processamento de 500 registros por dia às 01:00 da madrugada.

A proposta de um contexto de inscrição de famílias poderia ser feita através de um sistema web, sistema desktop ou aplicativo de celular. Todas elas poderiam já utilizar esses projetos já implementados para inicializar o desenvolvimento. Existem outas pontos importantes a ser tratado para buscar soluções, por exemplo, “seria necessário pessoas responsáveis para realizar esse cadastro ou qualquer pessoa poderia realizar o cadastro por conta?”. Para a implementação desse sistema de cadastro precisaria alterar esse banco de dados para criar identificadores melhores, por exemplo, utilização de CPF. Esse sistema deve

possibilitar que o usuário entre com login e senha, logo, um módulo de autenticação deveria ser implementado.

Por último, eu desenvolvi o projeto *Business.Tests*. Projeto responsável por testes automatizados utilizando *NUnit*. Desenvolvi dois exemplos em que apliquei na validação abordando casos se a família atende ou não um critério.

Não calculei o tempo exato que utilizei para realizar esse desenvolvimento. Utilizei apenas partes da noite dos dias 24, 25, 26 e 27. Hoje, dia 28, utilizei a parte da noite para escrever as considerações finais. Creio que foram, em média, 3,5 horas por noite. Nos cabeçalhos dos códigos, escrevi a data em que foi inicializado o desenvolvimento.