Solução:

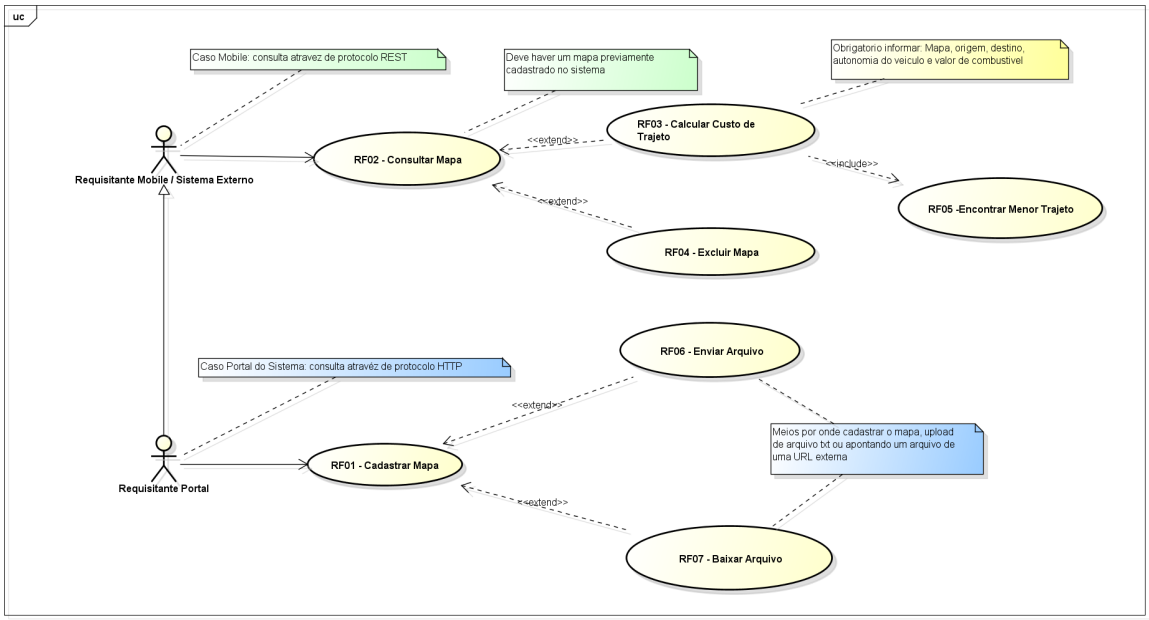
Desenvolver um sistema Web / WebService, capaz disponibilizar malhas de rotas previamente carregadas, para consultas visando, a melhor rota com menor valor de entrega possível.

O projeto teve como tecnologia escolhida a plataforma Asp.net. Mesmo sendo um teste para avaliação, existe um risco grande do domínio do problema ser maior do que o solicitado e girar em torno de diversos e complexos “Bounded Contexts” e necessidades outras integrações. Por este motivo escolhi esta plataforma, a sua capacidade de se adaptar a diferentes cenários mais pesados ou não.

Para este sistema fiz uma segregação de responsabilidades e desenvolvi com duas premissas em mente:

1- Disponibilizar um portal para carregamento de arquivos de mapas pelo administrador do sistema. Até este ponto, não coloquei segurança de usuário para não haver necessidade de mais integrações, como por exemplo questões do tipo:
Usuário é local (interno do sistema) ou relacionado ao usuário de rede de um ambiente coorporativo (login de rede), esta segunda opção seria ideal caso um certo departamento ou setor pudesse fazer alterações de mapa nos sistema enquanto outros apenas poderiam consultar ou utilizar outros módulos que viriam futuramente. Visando fugir destas questões mais complexas por não ter toda a visão do negocio, me concentrei na arquitetura do sistema e suas funcionalidades principais.
2- Disponibilizar um WebService (API) em que uma aplicação mobile ou um sistema externo pudesse ser desenvolvido para acessar este dado.

Caso de Uso



A Arquitetura

O desenvolvimento do portal segue a arquitetura em CRUD - N camadas (Onion Architecture):

1- Infrastructure
    a. AvenueEntrega.I18N (Internacionalização)
    b. AvenueEntrega.Infrastructure (suporte para internacionalização)
2- Domain Model
    a. AvenueEntrega.Model (Modelo de domínio baseado nos requisitos – modelo pobre sem metodos)
    b. AvenueEntrega.Rules (Regras de negocio para o domínio)
    c. AvenueEntrega.Rules.UnitTest (Testes básicos para validação apenas da lógica de encontrar a melhor rota por custo)
3- Repository
    a. AvenueEntrega.RepositoryFile (Camada de persistência de arquivo txt)
    b. AvenueEntrega.RepositoryNHibernate (Camada de persistência ORM para modelos de classes do sistema)
    c. AvenueEntrega.RepositoryNHibernate.IntegratesTest (Camada para testes com banco de dados e persistencia)
4- Services
    a. AvenueEntrega.DataContracts (camada de classes responsável por mensagerias trocadas entre o sistema do modelo e serviços)
    b. AvenueEntrega.ServiceETL (camada responsável por instanciar o repositório de arquivo e tratar o arquivo txt enviado antes de persisti-lo)
    c. AvenueEntrega.Service.ETL.IntegratedTest
    d. AvenueEntrega.ServiceContracts (Contrato de serviços utilizados pelo portal e pela API - WebServices)
    e. AvenueEntrega.Services (contém os serviços principais da aplicação)
    f. AvenueEntrega.Services.UnitTest (Teste unitário na camada de serviço e mock de dados)
5- Presentation Layer
    a. AvenueEntrega.Web.MVC (portal de interação , envio de arquivo de map e busca por melhor rota)
    b. Avenie.Entrega.Web.WCF (API para consumo por outros sistemas, sem envio de mapa)

Para o desenvolvimento desta aplicação utilizei alguns métodos e processos que fazem parte das habilidades que possuo e praticas de projeto.

Praticas de processos do DDD (Domain Driven Design) - Pensar no problema e nos relacionamentos entre as partes para desenvolver um core de sistema e ao redor disto ir levantando uma arquitetura.

Implementação da arquitetura:

1-Infrastructure Layer
Os artefatos da camada 1 Infrastructure somente servem para dar suporte se necessários para outras camadas, de forma que ele se torna uma camada que invade todas as outras. Por estas questões eu procurei generalizar o mínimo possível as outras camadas, pois geralmente o que pode ser generalizado em uma camada e servir de estrutura para outra pode ficar na infraestrutura. Porém existe o risco de outras camadas terem acesso a artefatos que não interessam a elas, então procurei generalizar pouco.

2-Domain Model Layer
Esta camada eu costumo segrega-la em 2 projetos (DomainModel e Rules)
No projeto **AvenueEntrega.Model** existem apenas POCOs (Poor Old CLR Objects), desta forma consigo construir apenas cascas de classes que irão interagir entre si e seus relacionamentos.
No projeto **AvenueEntrega.Rules** existem apenas regras de negócios entre as classes , como este teste possui poucas regras de negócio as regras básicas de cada entidade ficaram bem esplicitas em seus Specifications.

3-Repository Layer
Esta camada como o nome já diz apenas implementa os DataStores seja Banco de dados ou HardDisk. Foi utilizado persistência em disco e persistência em Banco de dados relacional atravéz de um ORM conhecido e opensource NHibernate.

4-Services Layer
Nesta camada, esta concentrado, todo o fluxo da aplicação. No projeto existem as funcionalidades principais que o sistema desempenhará. Porém toda e qualquer solicitação a estas funcionalidades seguem padrões estruturais, por exemplo todas estas funcionalidades recebem mensagens envelopadas para poder responder a elas.
Exemplo: a funcionalidade de “CalcularRota” ela aceita um tipo de mensagem “CalcularMelhorRotaRequest”, isto significa que ela recebe este objeto e dentro dele existem as especificações necessárias para esta funcionalidade desempenhar seu papel. Dentro deste envelope deve haver um objeto do tipo “ProblemaDTO”. Que seriam os requisitos para resolver o problema de rota. A resposta desta funcionalidade também se dá no formato de um envelope de mensagem respondendo a quem o solicitou com um envelope do tipo “CalcularMelhorRotaResponse”, internamente este outro tipo de mensagem possui somente o que sera de interesse a quem solicitou: “CustoTotal”, “MelhorCaminho”, “Success”, “Menssage”, e um dicionario de itens contendo resultados de validação “Rules”.
Desta forma quem invoca esta funcionalidade descrita, apenas precisa conhecer um DTO (Data Transfer Object), para hidrata-lo entrega-lo e não tem acesso ao Core do Domínio.
Os DTOs e os Envelopes todos estão no projeto **AvenueEntrega.DataContracts**.
Um outro projeto que acrescenta a este é o **AvenueEntrega.ServiceContracts** onde ele define as assinaturas das funcionalidades e serve de API para a criação de uma camada de WebService , REST (Json/XML) ou SOAP (XML).
No projeto AvenueEntrega.Service.ETL, está todo o fluxo de manipulação de Export, Transform and Load, e faz a ponte entre os repositorios (**AvenueEntrega.RepositoryFile** e

AvenueEntrega.RepositoryNHibernate), este serviço também se comunica somente por envelope de mensagens.

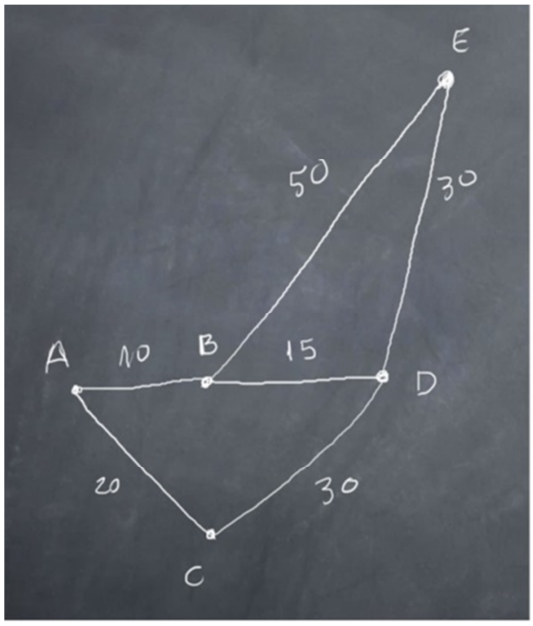
5.Presentation Layer
Esta camada é formada por 2 projetos **AvenueEntrega.Web.WCF** que conhece o projeto ServiceContracts para criar a API de serviço no padrão (SOAP / Json ou XML) e **AvenueEntrega.Web.MVC** que contem toda a interface de usuário. Este projeto utiliza um padrão MVC já bastante difundido atualmente. Uma particularidade na minha implementação deste padrão é que utilizo um padrão MVVM (ModelViewViewModel) que já ficará claro o motivo.

Fluxo da aplicação

Cada view possui um modelo ViewModel. Praticamente tudo o que vai para a interface de usuário, vai como um ViewModel ou retorna para a Controller como um ViewModel. Na Controller existe uma conversão sendo feita automaticamente, de ViewModels para DTOs, antes de transmiti-los dentro de um envelope de mensagem ao Serviço. No serviço este envelope é aberto e de acordo com as informações contidas, ele se comporta de maneira diferente, solicita o repositório que interessar e efetua a tarefa de dados que necessitar.
Esta tarefa terá um sucesso ou falha, que será percebida pelo Serviço e devolvido à Controller no formato de um novo envelope de mensagem. A Controller ao receber esta informação converte de volta em um ViewModel e alimenta as validações (em caso de formulário) ou notifica o usuário de um processo bem sucedido.

O problema - Uniform Cost Seach Algorithm (Chipest First):

Para a solução fundamental para este problema, imaginei a utilização de um algoritmo de busca chamado Uniform Cost Search (Chipest First).
Comparado a esta imagem, onde os pontos seriam origens ou destinos e os seguimentos de retas (não necessariamente retas), seriam as rotas entre cada deslocamento contendo seu espaço em km (quilômetros).



Cada rota possui um custo de trajeto, porém podemos ter mais de uma única forma para se sair de um ponto e chegar a outro ponto. Ex: Como dado como exemplo no enunciado, do problema. No caso onde seja necessário transitar do ponto de Origem A para ponto de Destino D e apenas olhando para a figura abaixo, poderemos verificar de imediato que existem 3 formas de sair do ponto A e chegar ao ponto D, porém cada um possui valores de custos diferentes:

a. A->B->D =25 km
b. A->C->D =50 km
c. A->B->E->D=90 km

O algoritmo Chipest First, conforme ele avança de um ponto A ele identifica suas fronteiras (possibilidades de rotas), ele prioriza o avanço para aquele que possuem o menor custo de deslocamento. Uma vez que o algoritmo, tenha chegado no destino, ele não termina ainda, pois ele precisa analisar as rotas possíveis que ele começou a vasculhar anteriormente, e definir se conseguem chegar no destino com menor custo ou não.

Cálculo do menor Custo

Durante a consulta de menor trajeto, o sistema deverá processar o seguinte calculo abaixo:
Onde "**CustoTotalDeslocamento**", é a somatória de todos os trechos que definem o menor caminho possível calculado pelo algoritmo e sua unidade é Km (quilometros).
A variável "**AutonomiaDoVeiculo**" é informado no momento da pesquisa, assim como a variável "**ValorDoLitroDeCombustivel**".

$$CustoTotalDeDeslocamento = \left(\frac{\left(\min \sum CustoDeslocamento \right)}{AutonomiaDoVeiculo} \right) * ValorDoLitroDeCombustivel$$

Verificando o exemplo de caso informado no arquivo de **1.Descrição do problema**, temos:

$$CustoTotalDeDeslocamento = \left(\frac{25km}{\frac{10km}{L}} \right) * \frac{2,50reais}{L} = \left(\frac{25 \cancel{km} \cancel{/L}}{10 \cancel{km}} \right) * \frac{2,50}{\cancel{L}} = 2,5 * 2,50 = 6,25$$

Testes

Como teste utilizei um estudo de caso, desenvolvido em um curso de inteligência artificial.
Mapa: Romenia , deslocamento de ‘Arad’ até ‘Bucharest‘.