**ANTONIO MENEGHETTI FACULDADE**
**CURSO DE SISTEMAS DE INFORMAÇÃO**

**RENAN ESTEVAN ROOS**

# INTEGRAÇÃO DE APLICAÇÕES UTILIZANDO A TECNOLOGIA WEB SERVICE REST

RESTINGA SÊCA/RS

**2017**

**ANTONIO MENEGHETTI FACULDADE - AMF**

**CURSO DE SISTEMAS DE INFORMAÇÃO**

**RENAN ESTEVAN ROOS**

# INTEGRAÇÃO DE APLICAÇÕES UTILIZANDO A TECNOLOGIA WEB SERVICE REST

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação na Faculdade Antonio Meneghetti-AMF.

Orientador: Prof°. Ms. Fábio Sarturi Prass

RESTINGA SÊCA/RS

**2017**

**ANTONIO MENEGHETTI FACULDADE - AMF**

**Renan Estevan Roos**

# INTEGRAÇÃO DE APLICAÇÕES UTILIZANDO A TECNOLOGIA WEB SERVICE REST

Trabalho de Conclusão de Curso apresentando como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação, Curso de Graduação em Sistemas de Informação na Faculdade Antonio Meneghetti-AMF.

Orientador: Prof°. Ms. Fábio Sarturi Prass

____________________________________

Prof. Ms. Fábio Sarturi Prass

Orientador do Trabalho de Conclusão de Curso

Antonio Meneghetti Faculdade

____________________________________

Prof. Dr. Jonas Bullegon Gassen

Membro da Banca Examinadora

Antonio Meneghetti Faculdade

____________________________________

Prof. Ms. Samuel Vizzotto

Membro da Banca Examinadora

Antonio Meneghetti Faculdade

**Restinga Sêca, RS, Novembro de 2017.**

## AGRADECIMENTOS

Foram quatro anos de intenso trabalho e de intenso estudo. Muitas vezes, é necessário fazer mais do que esperamos fazer. Contudo, existem pessoas e instituições que acreditam na nossa força, e na capacidade de sermos mais do que já somos.

Dentre essas pessoas, gostaria de agradecer primeiramente o apoio incondicional de minha família, principalmente de minha irmã, que foi a maior apoiadora que tive antes e durante esse período.

Além disso, agradeço imensamente os professores do curso de Sistemas de Informação, com destaque para meu orientador, Fábio Prass, e aqueles que se tornaram mais que mestres, se tornaram amigos: Samuel Vizzotto, João Otávio e Leonardo Guedes.

Não posso deixar de mencionar meus grandes apoiadores, que sempre me auxiliaram tecnicamente: Adilson Bordin, Matias Richa e Daniel Foletto.

Alguns colegas também foram essenciais, e hoje são como irmãos para mim. Entre eles, destaco: Bruno Viera, Camila Heinz, Estevão Souza, Jâime Stefanello e Natallya Verardi.

Por fim, agradeço imensamente a instituição Antonio Meneghetti Faculdade e a Meta, por todo suporte e apoio que nos proporcionou.

**“**Cada sonho que você deixa para trás, é um pedaço do seu futuro que deixa de existir**”.**

Steve Jobs

## RESUMO

Muitas vezes não é possível integrar sistemas que estão em diferentes domínios, ou escritos em diferentes linguagens. Para resolver isso foram criados os Web Services, que possuem como objetivo principal fornecer uma aplicação que realiza transporte de dados através de componentes que trabalham de forma independente, não importando a linguagem de programação adotada nos sistemas que o requisitam. O presente trabalho tem como escopo principal apresentar os Web Services existentes e de que forma eles podem auxiliar as empresas em troca de dados pela rede. Para tal, será criada uma aplicação em caráter experimental que explicará como funciona uma das tecnologias, o REST. O sistema será desenvolvido em ASP.NET MVC e o estudo de caso utilizado tem como objetivo permitir o usuário de meios de locomoção como Uber e Táxi economizar em sua compra.

**Palavras-chave:** SOA; Web Services; REST; SOAP.

**ABSTRACT**

Often, is not possible to integrate systems that are in different domains, or written in different languages. To solve this, the Web Services were created, whose main objective is to provide an application that carries data transport through components that work independently, regardless of the programming language adopted in the systems that request it. In this paper, the main objective is to present the existing Web Services and how they can help companies in exchange for data over the network. For such, an application will be created on an experimental basis that will explain how one of the REST technologies works. The system will be developed in ASP.NET MVC and the case study used aims to allow the user of means of locomotion as Uber and Taxi save on their purchase.

**Keywords:** SOA; Web Services; REST; SOAP.

# LISTA DE ABREVIAÇÕES

**SOAP:** Simple Object Access Protocol.
**REST:** Representational State Transfer.
**HTTP:** Hypertext Transfer Protocol.
**XML:** Extensible Markup Language.
**JSON:** JavaScript Object Notation.
**SOA:** Service-Oriented Architecture.
**WSDL:** Web Services Description Language.
**UDDI:** Universal Description Discovery and Integration.
**MVC:** Model-View-Controller.

## LISTA DE ILUSTRAÇÕES

**Figura 1.** Modelo de arquitetura orientada a serviços. ................................................ 15
**Figura 2.** Estrutura de um documento WSDL. ............................................................ 16
**Figura 3.** Exemplo de documento WSDL. ................................................................... 17
**Figura 4.** Estrutura de um documento no padrão UDDI. ............................................. 19
**Figura 5.** Estrutura do protocolo SOAP. ..................................................................... 20
**Figura 6.** Dinâmica da troca de mensagens através de SOAP. .................................... 20
**Figura 7.** Modelo de mensagem do protocolo SOAP. ................................................. 20
**Figura 8.** Modelo de arquivo JSON. ............................................................................ 23
**Figura 9.** Estrutura de um XML. ................................................................................. 24
**Figura 10.** Elementos XML com atributos. ................................................................ 24
**Figura 11.** Modelo da arquitetura MVC. .................................................................... 25
**Figura 12.** Tela Inicial de desenvolvimento da aplicação ASP.NET MVC. ................ 28
**Figura 13.** View "*Index.cshtml*" (Tela Inicial do Site). ............................................... 29
**Figura 14.** Função JavaScript que inicializa o mapa. ................................................... 30
**Figura 15.** Model "Pesquisa.cs". .................................................................................. 31
**Figura 16.** Troca de valores através da Model "Pesquisa". .......................................... 31
**Figura 17.** Chamada do método "BuscarPrecos". ......................................................... 32
**Figura 18.** Requisição GET que retorna um JSON com a geolocalização de um endereço. ........................................................................................................................ 33
**Figura 19.** XML de configuração das requisições. ...................................................... 35
**Figura 20.** Requisição uniforme para todas as opções cadastradas no XML de configuração. ................................................................................................................. 36
**Figura 21.** Extração da quantidade de preços que foram retornados pela requisição.... 36
**Figura 22.** Método que ordena a lista de preços com expressão lambda. .................... 37
**Figura 23.** Código que exibe a tabela na página após realização da busca de preços. .. 38
**Figura 24.** Pesquisa finalizada apresentando rota e estimativas na tela. ...................... 38

# SUMÁRIO

**1. INTRODUÇÃO** ........ 11
**2. OBJETIVOS** ........ 12
**2.1 Objetivo Geral** ........ 12
**2.2 Objetivos Específicos** ........ 12
**3. JUSTIFICATIVA** ........ 13
**4. ABORDAGEM TEÓRICA** ........ 14
**4.1. Arquitetura Orientada a Serviços (SOA)** ........ 14
**4.2. Web Services** ........ 15
4.2.1. WSDL ........ 16
4.2.2. UDDI ........ 18
**4.3. SOAP** ........ 19
**4.4. REST** ........ 21
**4.5. JSON** ........ 22
**4.6. XML** ........ 23
**4.7. ASP.NET MVC** ........ 25
**5. METODOLOGIA** ........ 25
**6. ESTUDO DE CASO** ........ 28
**7. CONSIDERAÇÕES FINAIS** ........ 39
**8. REFERÊNCIAS** ........ 40

# 1. INTRODUÇÃO

Com o passar do tempo e, com a evolução da tecnologia, verificou-se a necessidade de um sistema utilizar a informação de outro sistema, a fim de serem realizadas tarefas importantes dentro de uma organização.

Porém, na maioria das vezes, as duas aplicações não são escritas na mesma linguagem, ou até mesmo estão localizadas em ambientes diferentes, como por exemplo, a tecnologia em nuvem.

Com o objetivo de resolver esse problema, foi criado o termo Web Service, definido como “serviço exposto na rede”, que, com a possibilidade de acesso e integração do mesmo, realiza a troca de dados entre dispositivos eletrônicos que também estejam conectados à Internet.

Desde então, surgiram alguns meios de implementar tal serviço e, dentre estes, estão o SOAP (*Simple Object Access Protocol*) e o REST (*Representational State Transfer*). O primeiro, tem como base um protocolo (o qual possui o mesmo nome) para realizar a comunicação de objetos e serviços. O segundo, por sua vez, procurou definir alguns padrões de uso de um protocolo já existente, o HTTP. (FIDEL, 2015)

Entre os principais benefícios da padronização através do REST, está o correto uso dos verbos HTTP (Get, Post, Put, Head, Options e Delete), que por sua vez, realizam a dinâmica de troca de informações pela rede, de qualquer lugar, de qualquer sistema.

Neste ensejo, surge o seguinte problema: como o uso da tecnologia de Web Services com o padrão REST é útil na transmissão de dados entre sistemas diferentes?

# 2. OBJETIVOS

Seguem abaixo os objetivos que visam solucionar a problemática proposta:

## 2.1 Objetivo Geral

Verificar como o uso da tecnologia de Web Services com o padrão REST é útil na transmissão de dados entre sistemas diferentes.

## 2.2 Objetivos Específicos

- Explicitar os princípios da utilização da tecnologia WebService com o padrão REST;
- Demonstrar o uso da tecnologia Web Service com padrão REST na prática, de modo que possa ser entendida a sua implementação;
- Realizar um estudo de caso que explore o uso de Web Services no padrão REST.

## 3. JUSTIFICATIVA

Muitas vezes, em diversos sistemas, é necessário buscar dados que não estão sob o domínio do mesmo, como por exemplo, consultar o endereço completo de um CEP ou buscar dados em um banco de informações localizado em outro ambiente.

Devido a isso, buscou-se automatizar a execução de tais tarefas, de modo que não seja obrigatória a intervenção manual. Sendo assim, com os Web Services a transmissão de dados tende a se tornar mais veloz, mais eficaz e mais confiável, visto que as ações humanas são reduzidas ao extremo. (MATRAKAS, 2017)

Além disso, o uso do padrão REST pode ser útil ao integrar diferentes aplicações de uma mesma organização, permitindo maior controle e otimizando processos, promovendo assim, interoperabilidade entre os sistemas.

Outro importante ponto a destacar é a diminuição de gastos em recursos de desenvolvimento, levando em conta que os Web Services permitem trocar informações entre sistemas diferentes. Por exemplo, um sistema escrito em PHP pode realizar o comando de uma tarefa para um sistema escrito em Java a partir de um Web Service escrito em C#. Com isso, não é necessário contratar uma equipe especializada em Java para a realização do desenvolvimento. Tudo isso graças às linguagens universais que são utilizadas pelo REST: XML ou JSON. Outra questão a ser levantada: E se as aplicações em PHP e em Java não estivessem localizadas na mesma rede? Seria possível a comunicação? (MATRAKAS, 2017)

Portanto, procura-se entender como essa tecnologia remota, ou seja, localizada na nuvem, tende a ser útil na transmissão de dados entre ambientes diferentes e demonstrar como implementá-la, visando apresentar conceitos que podem ser imprescindíveis para desenvolvedores de software que precisam realizar tal integração.

# 4. ABORDAGEM TEÓRICA

Com o objetivo de apresentar como os Web Services REST podem auxiliar na integração de aplicações, serão abordados os principais conceitos que permeiam esse tema. Partindo do estudo sobre Arquitetura Orientada a Serviços (SOA), serão elencados os modos, meios e tecnologias que são usados atualmente para realizar a implementação desse tipo de sistema. Sendo assim, serão explicitados alguns termos como Web Service, SOAP, REST, XML e JSON, além das ferramentas e bibliotecas que serão utilizadas para desenvolver o projeto que fará uso do Web Service.

## 4.1. Arquitetura Orientada a Serviços (SOA)

A SOA cria um modelo de arquitetura que possui por principal objetivo aprimorar a eficiência, a agilidade e a produtividade de uma empresa, de modo a tornar os serviços como os principais meios para transformar a solução de um sistema eficaz. (ERL, 2009)

Contudo, o termo “arquitetura orientada a serviços” é amplamente utilizada no *design* de estruturas computacionais que utilizam dos serviços como unidade principal da lógica em questão.

Os serviços são aplicações ou um conjunto destas que ficam expostas na rede, de modo que outro sistema possa fazer uso dos métodos e funcionalidades que o mesmo disponibiliza.

A principal razão para o uso de SOA atualmente é a necessidade de integrar departamentos e áreas que precisem de informações que estão sob domínio de outra entidade. Deste modo, criaram-se estratégias empresariais que facilitam e agilizam tais trocas. Além disso, outro motivo importante a ressaltar é que a utilização de SOA facilita mudanças, independente de tecnologia e plataformas. (ERL, 2009)

Na Figura 1 observa-se o *design* de uma arquitetura orientada a serviços, a qual é utilizada na maioria das empresas. Existem dois ambientes na estrutura, o servidor e o cliente. O servidor tem como papel publicar os serviços na web, de modo que os mesmos possam ser utilizados pelo cliente, que, por sua vez, encontra um registro de serviços, ou seja, a aplicação. Depois de encontrar a mesma, é possível realizar a busca

das informações ou enviar um comando a ser executado pela mesma no servidor. (AVELLAR E DUARTE, 2012).

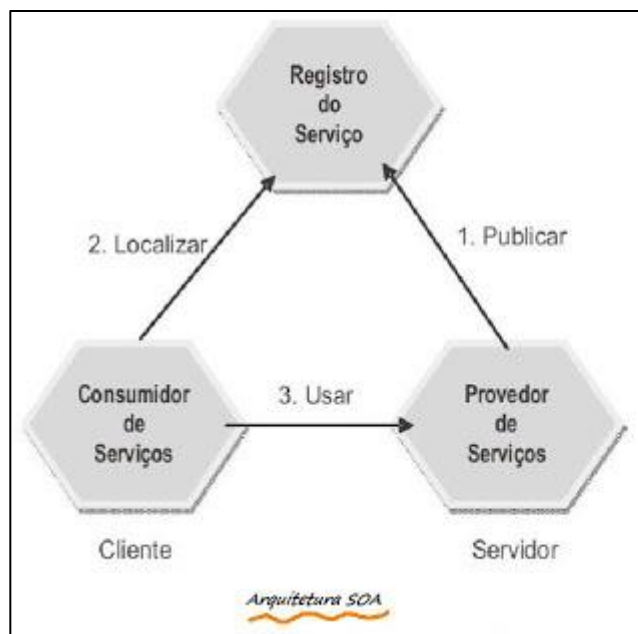


**Figura 1.** Modelo de arquitetura orientada a serviços.

Fonte: JOSUTTIS (2008).

### 4.2. Web Services

Os Web Services são serviços expostos na rede. Em outras palavras, são sistemas que podem ser encontrados através de um caminho na internet, de modo que seja possível a comunicação entre os mesmos.

De acordo com Erl (2009):

> Um Web service pode ser associado a papéis temporários, dependendo de sua utilização em runtime. Por exemplo, ele atua como fornecedor de serviços, quando recebe e responde a mensagens de solicitação, mas também pode assumir o papel do consumidor de serviço, quando for necessário o envio de mensagens de solicitação a outros Web services. (ERL, 2009, p. 39)

Sendo assim, Web Services podem ter dois tipos de utilização. O mais comum é como fornecedor de serviços, onde um sistema localizado em outro ambiente realiza uma requisição, que na linguagem de Web Services se chama *request*, e o mesmo faz a leitura desta. Após ler a requisição, o serviço realiza a ação a qual foi chamada e então

devolve o retorno, que pode ser apenas um bilhete de confirmação de realização da ação, ou também, devolver dados.

Como os Web Services são expostos na web, é necessário "achá-los" na rede, de modo que possa se saber quais métodos e chamadas podem ser realizadas. Para isso, existem dois modelos de descrição de interfaces de Web Services que se deve conhecer para criação de um serviço: WSDL e UDDI.

### 4.2.1. WSDL

Um documento WSDL (em inglês, *Web Services Description Language*), é um documento em formato XML que é usado para descrever um Web Service. Ele especifica algumas informações importantes do serviço, como seu local, seus métodos e também seus principais elementos. (SOSNOSKI, 2011).

Segundo a Sosnoski (2011), a estrutura de um arquivo WSDL é baseada em quatro principais elementos: *types*, *messages*, *portType* e *binding*. Pode-se observar a mesma abaixo, na Figura 2.

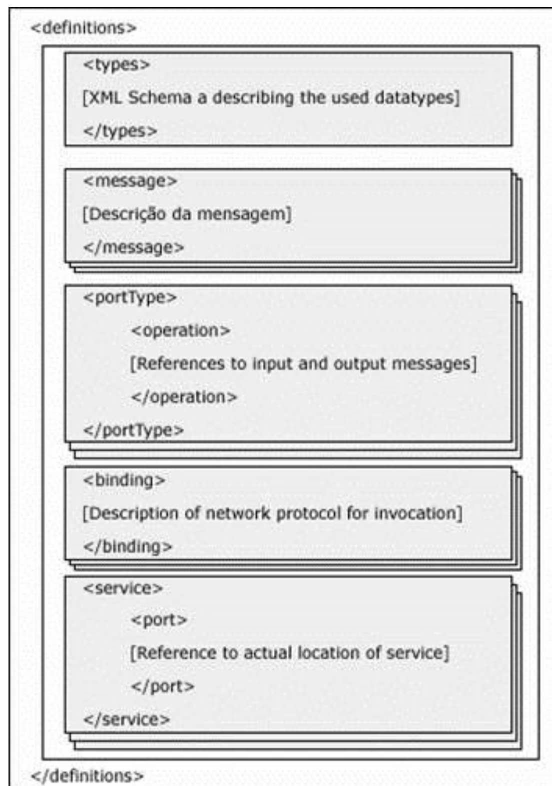


**Figura 2.** Estrutura de um documento WSDL.

Fonte: MARINHO (2013).

O campo *types* é responsável por especificar quais os tipos de dados que são usados pelo Web Service que o WSDL é referente. O segundo campo, *messages*, é o elemento que define os dados que entram e saem de cada operação fornecida pelo serviço exposto. (MARINHO, 2013).

No elemento *portType* são definidas as operações e as mensagens envolvidas. Dentro deste campo, existe outro atributo chamado *name*, que tem por objetivo nomear a operação disponibilizada pelo Web Service. Existem quatro tipos de operações que podem ser usadas: *one-way*, *request-response*, *solicit-response*, *notification*. (SOSNOSKI, 2011).

O *one-way* é um tipo de operação que apenas recebe mensagens. As operações do tipo *request-response* recebem mensagens e enviam um retorno com uma mensagem relacionada. Do tipo *solicit-response* é o contrário, enviam uma mensagem e então recebem a mensagem relacionada. O último tipo, *notification*, define uma operação que apenas envia mensagens. (SOSNOSKI, 2011).

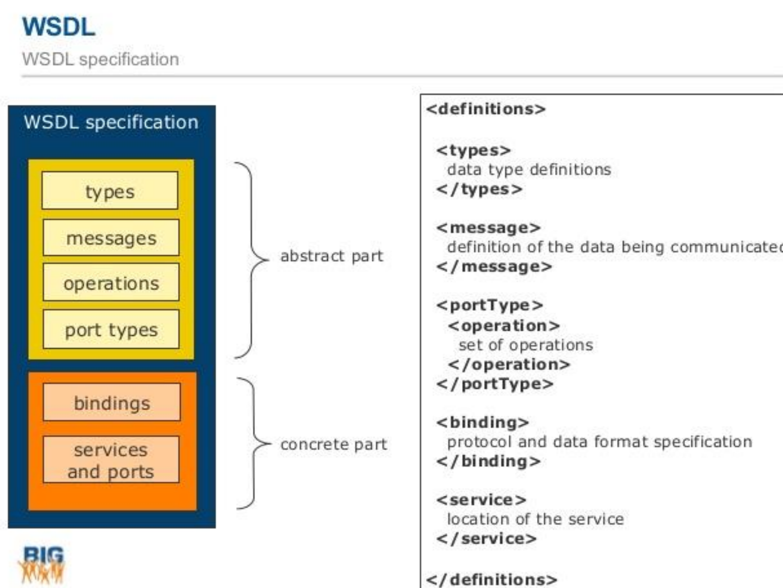


**Figura 3.** Exemplo de documento WSDL.

Fonte: Papazoglou (2008).

Por fim, existe o elemento *bindings*, que expõe qual tipo de mensagem e o protocolo usado para a operação em questão. Os atributos deste campo são *name* e *type*.

O *name* possibilita o nome do *binding* ser descrito e o *type* é responsável por referenciar o *portType* que será utilizada. (MARINHO, 2013).

Na Figura 3 é exibido um modelo de WSDL contendo todos os elementos acima. O WSDL pode ser apenas utilizado pelos Web Services que utilizam o protocolo SOAP, pois os documentos WSDL não suportam os *endpoints* (portas) escritos no padrão REST.

### 4.2.2. UDDI

O UDDI (*Universal Description Discovery and Integration*) é uma linguagem descritiva de um Web Service, utilizado para encontrar, publicar e integrar os mesmos na Internet. Segundo Gunzer (2002), o UDDI pode ser definido como um padrão desenvolvido para disponibilizar um modo de busca dos serviços, ou seja, um mediador deles, permitindo que os clientes (requisitantes do Web Service) encontrem o servidor em questão.

Atualmente na versão 3.0, o padrão UDDI foi criado em setembro de 2000 pela Microsoft, IBM e Ariba. Desde 2005, lançamento da última versão, o UDDI não recebeu mais modificações.

Na Figura 4, é possível observar a estrutura de um documento no padrão UDDI, as quais são descritas abaixo:

- *businessEntity*: esse componente tem como função descrever o provedor do Web Service. Algumas das informações apresentadas são dados de contato, serviços oferecidos, identificadores de negócio para um determinado cliente;
- *businessService*: é o elemento que vem uma hierarquia abaixo do *businessEntity*. Sua função é descrever o que realiza um serviço. Possui dois atributos, *businessKey* e *serviceKey*, que definem as categorias as quais o Web Service pertence;
- *bindingTemplate*: contém as informações técnicas do Web Service, como por exemplo, interface ou API;

  *tModels*: são referenciados pelos *bindingTemplates* e visam apontar alguns conceitos que podem ser registrados, como taxonomia, transportes e assinaturas digitais.

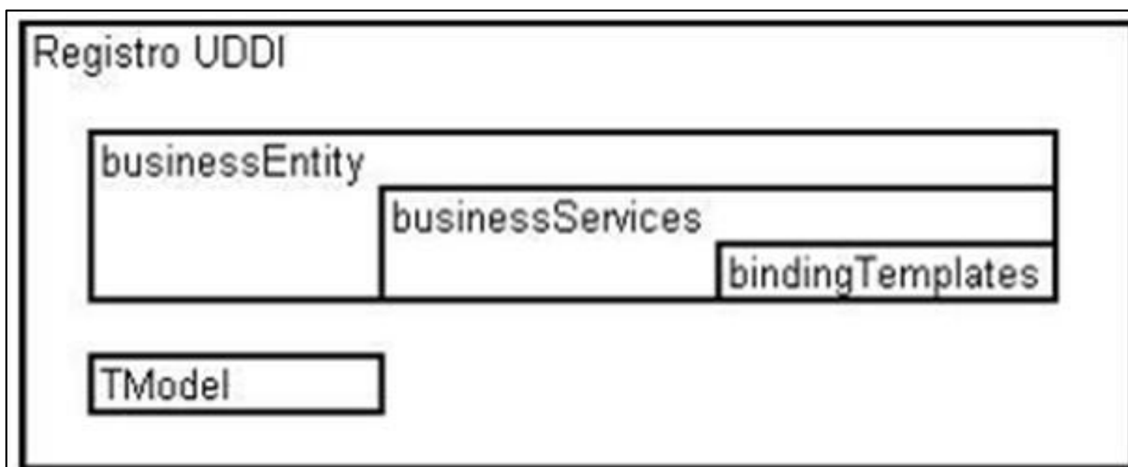


**Figura 4.** Estrutura de um documento no padrão UDDI.

Fonte: ROLIM (2014).

Depois de conhecer os serviços e saber qual a estrutura dos mesmos, necessita-se saber como realizar as chamadas para os métodos disponibilizados pelo Web Service. Para isso, foram criados um protocolo e um padrão com base em outro protocolo que podem ser usados para se fazer tais ações. Ambos são bastante difundidos entre os desenvolvedores e seu uso normalmente depende da preferência dos mesmos.

### 4.3. SOAP

SOAP, em inglês *Simple Object Access Protocol*, é um protocolo para troca estruturada de informações através do protocolo HTTP. Através das interfaces descritas pelos documentos WSDL, transfere mensagens entre os serviços instanciados.

Baseado em XML, o contexto de atuação do protocolo SOAP é descrito na Figura 5. Pode-se observar que o mesmo está em contato sobre o protocolo HTTP e utiliza as interfaces WSDL para “entender” o que o Web Service solicitou. Além disso, a dinâmica de troca das mensagens (entre o cliente e servidor) pode ser observada na Figura 6, a qual é a padrão para um serviço cliente x servidor. Existe uma *request* que faz a requisição ao servidor, o mesmo realiza a operação a qual o cliente solicitou, e devolve uma response que pode ser um conjunto de dados solicitado, ou apenas uma informação de operação realizada ou não.

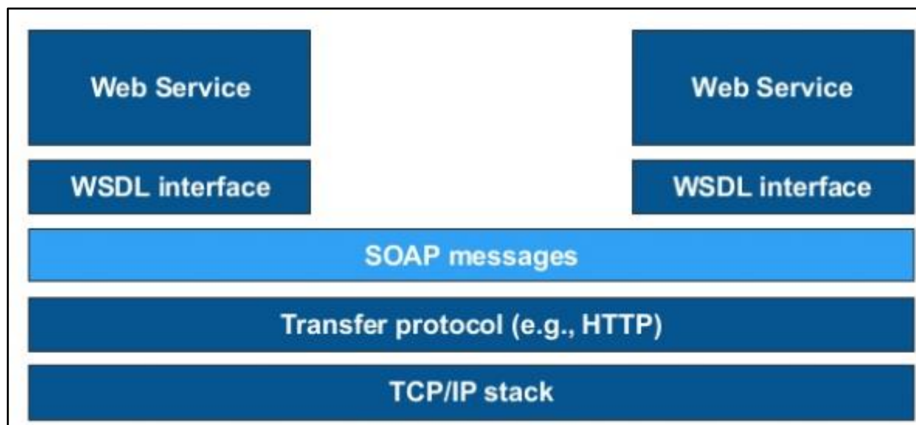


**Figura 5.** Estrutura do protocolo SOAP.

Fonte: PAPAZOGLOU (2008).

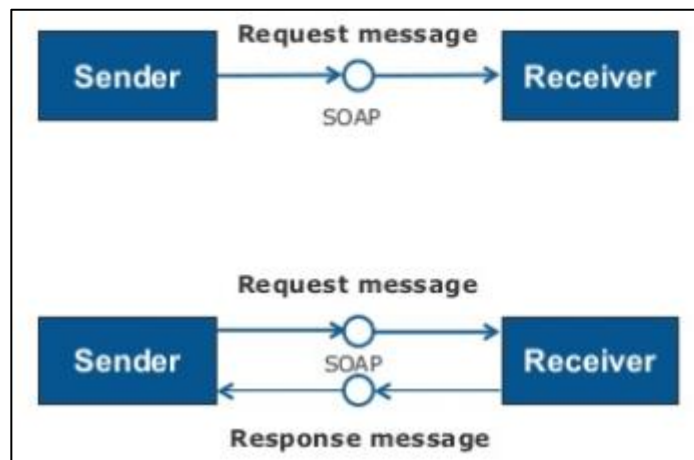


**Figura 6.** Dinâmica da troca de mensagens através de SOAP.

Fonte: PAPAZOGLOU (2008).

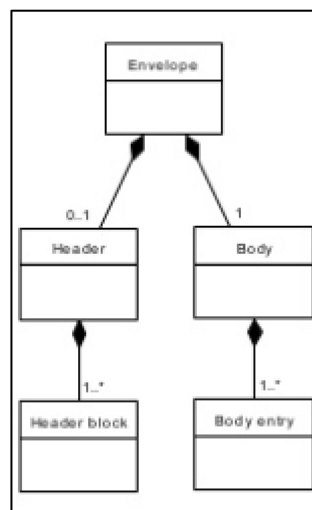


**Figura 7.** Modelo de mensagem do protocolo SOAP.

Fonte: PAPAZOGLOU (2008).

Como uma requisição SOAP é estabelecida sobre o protocolo HTTP, existe uma proximidade entre a sua estrutura com o envelope do TCP/IP, o que pode ser observado na Figura 7.

### 4.4. REST

A tecnologia REST, *Representational State Transfer*, é um padrão criado para adequar os princípios da utilização do protocolo HTTP nos Web Services. Diferente do SOAP, não é um protocolo que atua em cima do HTTP, e sim, em conjunto.

O REST foi apresentado em uma tese de doutorado no ano de 2000 por Roy Fielding, um dos mais importantes autores do protocolo HTTP e cofundador do projeto Apache HTTP Server[1].

Dentre as principais diferenças entre as implementações SOAP e REST, pode-se destacar as corretas utilizações dos principais verbos HTTP (POST, GET, PUT e DELETE), os leves pacotes que são transmitidos pela rede e a não necessidade de gerar outro documento, o envelope SOAP. (MATRAKAS, 2017).

Existem algumas situações onde o uso do REST é altamente recomendado. Dentre eles estão aquelas onde há limitação de recursos, visto que a estrutura do retorno é em qualquer formato definido pelo desenvolvedor e a maioria dos navegadores aceitam tais modelos.

A implementação correta do protocolo HTTP (verbos, *accept headers*, códigos de estado HTTP, *Content-Type*) que é base para o REST recebe uma nomenclatura própria: RESTful. O RESTful defende que as integrações utilizando o padrão REST devem seguir uma série de regras. Uma das regras mais importantes é usar cada verbo para o que foi proposto, como por exemplo: GET para buscar dados, POST para realizar inserção de dados, PUT para atualizar informações e DELETE para apagá-las. Essas são os principais que formalizam um completo sistema de realização de operações CRUD.

---

[1] O *Apache HTTP Server Project* é um esforço para desenvolver e manter um servidor HTTP de código aberto para sistemas operacionais modernos, incluindo UNIX e Windows. O objetivo deste projeto é fornecer um servidor seguro, eficiente e extensível que forneça serviços HTTP em sincronia com os padrões HTTP atuais.

Além destes, existem outros verbos que podem ser utilizados, como o HEAD e o PATCH. (MATRAKAS, 2017).

Outro aspecto importante do REST é referente às mensagens de retorno que são disponibilizadas para as chamadas realizadas. São retornadas do servidor que hospeda o Web Service as mensagens padrão do HTTP: mensagens de sucesso iniciam com o número "2" (200 para sucesso, 201 para criado, 204 para operação realizada com sucesso, porém, sem retorno para o cliente), mensagens que iniciam com o número "4" indicam problemas que ocorreram do lado do cliente (400 para requisição não compreendida, 404 para requisição não encontrada) e respostas iniciadas com o número "5" que demonstram erro do lado do servidor (500 significa *Internal Server Error*, ou seja, erro interno do servidor que impossibilitou a resposta adequada da requisição). (SILVA, 2008).

## 4.5. JSON

A fim de realizar a troca de informações entre os sistemas, foram criados alguns tipos de arquivos. Dentre os mais usados está o JSON, *JavaScript Object Notation* (sua pronúncia é *Jason*). O JSON representa alguns tipos de dados primários, como *strings*, números, booleanos e nulos, além de também poder receber tipos estruturados, como objetos e vetores. (RAMALHO, SIMÕES & CARRIÇO, 2007).

Os objetos podem ser considerados como uma coleção não ordenada de nome e valor (no formato de *string*), enquanto os vetores são considerados apenas como uma sequência ordenada de zero ou mais valores.

Devido ao JSON ser derivado do JavaScript, possui o objetivo de ser simples, entendível e textual para as aplicações que o farão uso. Pode ser observada na Figura 8 um exemplo de JSON, onde se verifica que sua estrutura é bastante simples, apenas diferindo a apresentação dos objetos e vetores para os demais tipos de dados.

Além de seu fácil entendimento, existe outra vantagem no uso de JSON devido o mesmo ser derivado do JavaScript: a facilidade na serialização e transporte dados. Como esse tipo de arquivo é serializado arbitrariamente, pode-se transformá-lo em um objeto JavaScript com uma simples *parser* presente na maioria dos navegadores de internet que são usados atualmente.

```
{
    "type": "menu",
    "value": "File",
    "items": [
        {"value": "New", "action": "CreateNewDoc"},
        {"value": "Open", "action": "OpenDoc"},
        {"value": "Close", "action": "CloseDoc"}
    ]
}
```

**Figura 8.** Modelo de arquivo JSON.

Fonte: RAMALHO, SIMÕES & CARRIÇO (2007).

## 4.6. XML

O *Extended Markup Language*, ou seja, a linguagem de marcação extensiva XML, foi criada em 1997 pela W3, com o objetivo de obter-se uma mais eficiente manipulação e transporte de dados através da Internet. Uma das características mais importantes do XML é definida em seu nome: é extensiva. Isso se deve ao fato do XML permitir a definição dos elementos de marcação de sua estrutura de acordo com sua utilização (MOREIRA, 2009).

O principal componente de um XML é chamado de elemento. O elemento é representado como um texto que possui em seu início o caractere "<" e no final, o caractere ">". Esse elemento pode ser chamado de marcador ou *tag*. Além disso, como pode ser observado na Figura 9, cada elemento de um XML pode possuir outros elementos dentro dele, texto bruto ou até uma mistura dos dois. Sendo assim, é necessário que o elemento inicie, e seja fechado. Para ser fechado, ou seja, representar que a *tag* não possuíra mais nada dentro dela, usa-se a mesma nomenclatura, porém, adicionando uma "/" após o caractere de início ("<").

Permite-se também, no XML, o uso da associação de atributos. Estes podem ser considerados como propriedades ou características de um elemento, e são definidas em pares, como no JSON: nome e valor. O nome significa qual a característica que será apresentada para o elemento correspondente, e o valor é sua informação, como pode ser visto na Figura 10.

```xml
<?xml version="1.0"?>
<person>
  <first_name>Jonathan</first_name>
  <last_name>Freeman</last_name>
  <login_count>4</login_count>
  <is_writer>true</is_writer>
  <works_with_entities>
    <works_with>Spantree Technology Group</works_with>
    <works_with>InfoWorld</works_with>
  </works_with_entities>
  <pets>
    <pet>
      <name>Lilly</name>
      <type>Raccoon</type>
    </pet>
  </pets>
</person>
```

**Figura 9.** Estrutura de um XML.

Fonte: Loenert (2017).

```xml
<marc:datafield tag="100" ind1="1" ind2=" ">
   <marc:subfield code="a">Brown, Dan,</marc:subfield>
   <marc:subfield code="d">1964-</marc:subfield>
</marc:datafield>
<marc:datafield tag="245" ind1="1" ind2="4">
   <marc:subfield code="a">The lost symbol :</marc:subfield>
   <marc:subfield code="b">a novel /</marc:subfield>
   <marc:subfield code="c">Dan Brown.</marc:subfield>
</marc:datafield>
<marc:datafield tag="250" ind1=" " ind2=" ">
   <marc:subfield code="a">1st ed.</marc:subfield>
</marc:datafield>
<marc:datafield tag="260" ind1=" " ind2=" ">
   <marc:subfield code="a">New York :</marc:subfield>
   <marc:subfield code="b">Doubleday,</marc:subfield>
   <marc:subfield code="c">c2009.</marc:subfield>
</marc:datafield>
<marc:datafield tag="300" ind1=" " ind2=" ">
   <marc:subfield code="a">509 p. :</marc:subfield>
   <marc:subfield code="b">ill. ;</marc:subfield>
   <marc:subfield code="c">25 cm.</marc:subfield>
</marc:datafield>
```

**Figura 10.** Elementos XML com atributos.

Fonte: Assumpção (2015).

Não é recorrente encontrar um estudo sobre qual linguagem de marcação é a melhor: JSON ou XML. Hoje em dia, normalmente, os desenvolvedores procuram

desenvolver a aplicação desejada com comunicação entre as partes envolvendo a linguagem de marcação que mais a agrada, pois ambas possuem a mesma estrutura, diferenciando apenas suas sintaxes.

## 4.7. ASP.NET MVC

Com o objetivo de disponibilizar um produto que permite o desenvolvimento de Web Sites, a Microsoft criou uma biblioteca chamada ASP.NET, que juntamente com a arquitetura MVC, permite a criação de sites utilizando a plataforma .NET.

De acordo com Ritter (2011):

> O ASP.NET MVC é um framework da plataforma .NET que fornece um desenvolvimento dividido em camadas, promovendo total controle da aplicação e, juntamente com o ADO.NET Entity Framework, é possível realizar de forma transparente e rápida o acesso ao banco de dados, além de simplificar o código das classes responsáveis pelas regras de negócio da aplicação. (RITTER, 2011, p. 5)

A primeira versão do ASP.NET MVC foi disponibilizada em março de 2009, visando "fugir" do ASP.NET Web Forms, arquitetura de desenvolvimento de sites da plataforma .NET utilizado até então.

O padrão MVC (*Model-View-Controller*) visa à separação dos conceitos em modelos, visões e controladores, conforme Figura 11:

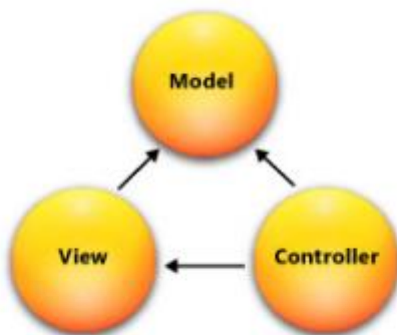


**Figura 11.** Modelo da arquitetura MVC.

Fonte: HANSELMAN (2010).

Os modelos armazenam as regras de negócio (BLL ou *business*) da aplicação, bem como toda parte de comunicação com o banco de dados utilizado. As visões são os componentes que possuem a interface com o usuário, onde são baseados em um objeto modelo. Por fim, os controladores são todos os componentes que fazem a interação com

o usuário, selecionando a página a qual o mesmo está utilizando, ou seja, necessita trabalhar juntamente com o modelo.

Dentre as principais vantagens do uso do ASP.NET MVC, pode-se destacar o fácil gerenciamento, resultado da divisão de conceitos, o processamento de requisições através de um único controlador, o *Front Controller* e o alto controle em seu *front-end*, permitindo que os *web designers* saibam qual o comportamento do aplicativo (HANSELMAN, 2010).

Outro ponto importante a ser destacado é que o MVC utiliza uma URL (*Uniform Resource Location*) para encontrar o arquivo que está sendo manipulado pelo usuário. Esse arquivo normalmente possui a extensão *.aspx*, e nele, se encontram os códigos e marcações que são utilizadas pela página que está requisitando algo. Esse processo é controlado por um objeto chamado *UrlRoutingModule*, que analisa a situação que está sendo realizada e realiza a busca através do caminho disponibilizado na URL.

# 5. METODOLOGIA

Segundo Lazar, Feng e Hochheiser (2010), a pesquisa experimental tem como objetivo testar hipóteses que dizem respeito à convicção do pesquisador. Desta forma, este trabalho tem caráter experimental no que diz respeito à utilização das tecnologias descritas nos capítulos anteriores, visando verificar as possibilidades que existem para criação de um determinado sistema.

Uma pesquisa com resultados qualitativos não apresenta números como retorno. Entretanto, ajuda a identificar possíveis soluções de um problema, bem como a descoberta de comportamentos e tendências da mesma. Com base nisso, pode se dizer que o experimento traz resultados qualitativos visto que ao testar as tecnologias, são expostas as características encontradas no desenvolvimento. (MINDMINERS, 2016)

Além disso, procura-se demonstrar como as tecnologias devem ser utilizadas e de que modo elas trazem bons resultados para uma determinada situação.

## 5.1. Metodologia de trabalho

A fim de explicitar os conceitos que foram abrangidos na abordagem teórica, têm-se dois serviços e uma aplicação, que possuem por objetivo principal o transporte de informações entre os mesmos, sendo que eles estão em domínios diferentes, e serão encontradas apenas pela rede.

A primeira aplicação será o site que fará o consumo dos outros dois serviços, que estão disponíveis na rede gratuitamente. Esse *website*, desenvolvido em ASP.NET MVC, realizará o consumo dos outros dois serviços através da tecnologia de Web Services REST.

Os outros dois serviços, por sua vez, fornecem informações referentes à estimação de preços para contratação de meios de locomoção, através de requisições do tipo GET, com retorno em JSON. São eles: Uber API e Taxicode API.

O modelo de negócio que será utilizado para tal, é um site que será aberto ao público, que, ao informar dois endereços (um de saída e um de destino), buscará dos dois serviços os melhores preços para realização do trajeto. Serão apresentados na tela os preços em ordem crescente, além da rota que será realizada, no mapa.

## 6. ESTUDO DE CASO

Este item do trabalho fornecerá o estudo de caso selecionado com base na metodologia escolhida, ou seja, o desenvolvimento do *site* com o consumo dos dois Web Services selecionados, de modo que no final seja exposto o objetivo final: coletar os dados, tratá-los e apresentá-los para o usuário.

A tecnologia escolhida para a criação do site foi ASP.NET MVC versão 5. A Figura 12, abaixo, mostra como o Visual Studio apresenta a tela inicial ao selecionar a tecnologia citada. Dentre os principais itens em um projeto MVC, podem ser destacados:

- *Content*: abriga todo tipo de conteúdo visual ou estilos (CSS) que serão utilizados na página;
- *Controllers*: responsável por salvar todas as páginas controladoras de conteúdo das *Views*;
- *Models*: classes modelos que visam criar um estrutura de negócio (*business*) para os objetos que serão utilizados nas *Controllers e Views*;
- *Scripts*: todo conteúdo *Javascript* que as páginas farão uso;
- *Views*: páginas do site na extensão *cshtml*. A inicial se chama *Index.cshtml*, enquanto que a página *Master*, conhecida por ser o “esqueleto” de todas as outras é chamada de *_Layout.cshtml*.

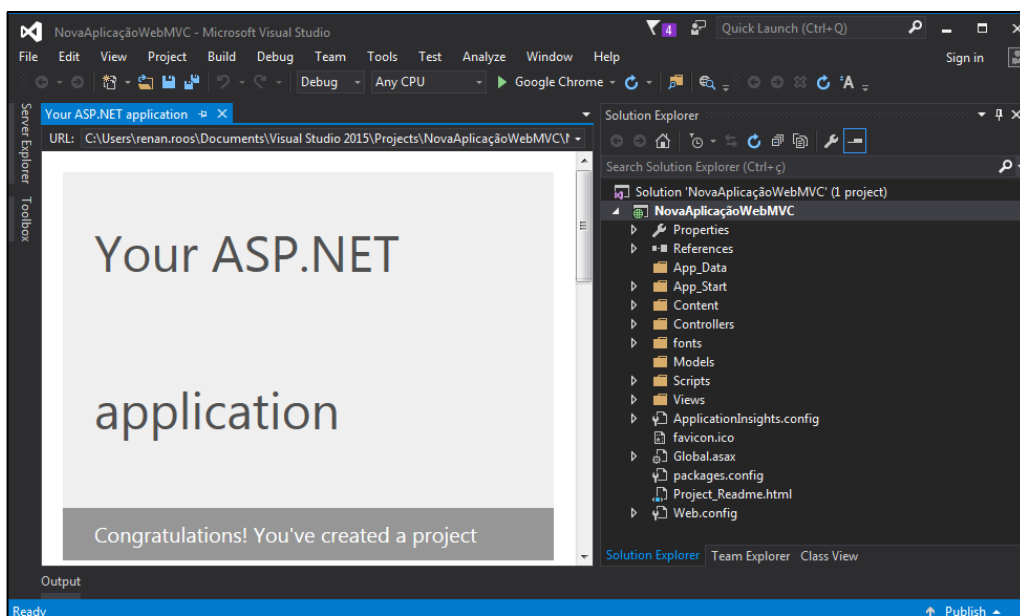


**Figura 12.** Tela Inicial de desenvolvimento da aplicação ASP.NET MVC.

Para melhor integração e separação de tarefas, foram criados dois principais projetos:

- "PrecosTrajeto.csproj": Projeto MVC;
- "EstimaTrajeto.csproj": *ClassLibrary* [2] que fará o consumo dos Web Services e retornará uma lista de preços para determinado trajeto informado.

Se baseando no conceito de *Model-View-Controller*, começou-se pela estruturação da *View*, ou seja, a tela principal do *site*. Como será uma tela única, foi escolhido o arquivo *Index.cshtml* (página inicial), e alguns elementos visuais foram inseridos - outros removidos -, os quais podem ser observados na Figura 13.

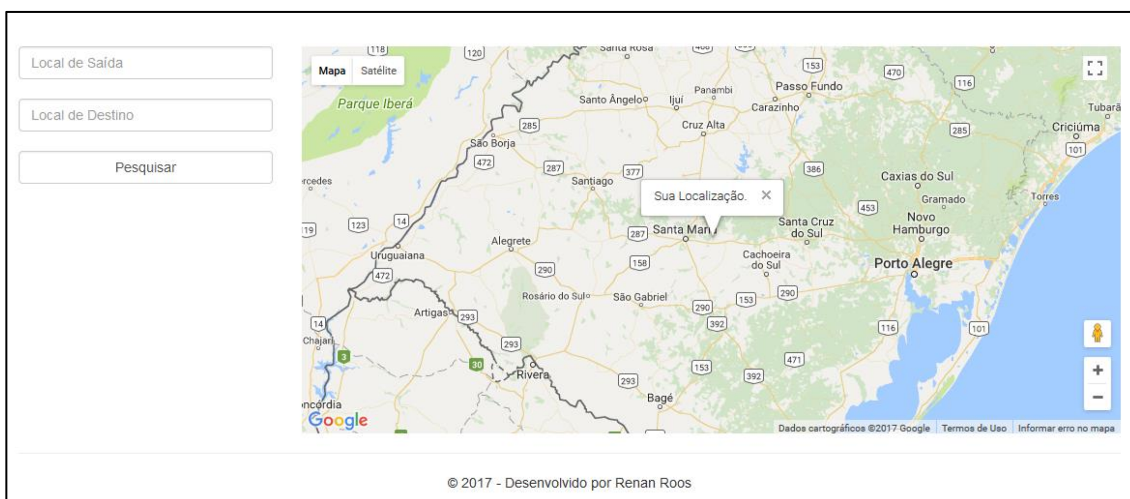


**Figura 13.** View "*Index.cshtml*" (Tela Inicial do Site).

Para exibição do mapa, foi utilizada a API do Google Maps, que disponibiliza diversos serviços, como busca de endereços a partir da geolocalização (ou reverso), estabelecimento de rotas entre dois pontos, ou também a busca da localização atual. Todos estes foram utilizados no site. Na Figura 14 pode ser observada a função que inicia o mapa quando a tela inicial é carregada.

[2] *Class Library* é um conjunto de classes que podem ser utilizadas em comum para qualquer tecnologia do .NET Framework, como ASP.NET MVC, Windows Forms, etc. (AECE, 2017)

```
function iniciarMapa() {
    var map = new google.maps.Map(document.getElementById('map'), {
        center: { lat: -34.397, lng: 150.644 },
        zoom: 7
    });
    var infoWindow = new google.maps.InfoWindow({ map: map });

    // Try HTML5 geolocation.
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(function (position) {
            var pos = {
                lat: position.coords.latitude,
                lng: position.coords.longitude
            };

            infoWindow.setPosition(pos);
            infoWindow.setContent('Sua Localização.');
            map.setCenter(pos);
        }, function () {
            handleLocationError(true, infoWindow, map.getCenter());
        });
    } else {
        // Browser doesn't support Geolocation
        handleLocationError(false, infoWindow, map.getCenter());
    }
```

**Figura 14.** Função JavaScript que inicializa o mapa.

Realizado o carregamento da página, o usuário pode inserir o local de onde irá fazer a solicitação do transporte, e qual será o seu destino. Após incluir tais informações, pode solicitar a pesquisa através do botão “Pesquisar”, que, por sua vez, começará o processo de elaboração da lista de preços na *Controller*.

Para realizar o intercâmbio de informações entre a *View* e a *Controller*, foi criada uma classe modelo (*Model*), chamada “Pesquisa.cs”, que salva os dados que precisam ser trocados entre a tela e o *back-end*. Os atributos desta classe podem ser observados na Figura 15.

Além dos endereços de saída e de destino, a *Model* também irá receber os quatro pontos que precisam ser utilizados pelo mapa para o estabelecimento das rotas e uma lista do objeto “Preco”, que contém o nome da opção (Uber ou Táxi), além de outro objeto, chamado “Retorno”. Esse objeto, por sua vez, possui o nome do carro e o valor do trajeto.

```
5 references
public class Pesquisa
{
    #region Atributos

    private string saida;
    private string destino;
    private string latitudeSaida;
    private string longitudeSaida;
    private string latitudeDestino;
    private string longitudeDestino;
    private List<Preco> listPrecos;

    #endregion Atributos
```

**Figura 15.** Model “Pesquisa.cs”.

Na Figura 16, pode ser verificado como a *View* permite referenciar a *Model* diretamente nela (linha 2), evitando a necessidade de comunicar um parâmetro por vez, ao invés de somente um objeto (neste caso, a própria *Model* “Pesquisa”, linhas 27 a 31).

```
1   <!-- Referência da Model Pesquisa -->
2   @model PrecosTrajeto.Models.Pesquisa
3
4   <!DOCTYPE html>
5   <html>
6   <head>
7       <style>
8           #map {
9               height: 400px;
10              width: 100%;
11          }
12      </style>
13  </head>
14  <body onload="iniciarMapa()">
15      <div class="row">
16          <br />
17      </div>
18      <div class="row">
19          <div class="col-md-3 form-group">
20
21              @using (Html.BeginForm())
22              {
23                  @Html.AntiForgeryToken()
24                  <!-- Inclusão dos valores da tela no objeto model -->
25                  @Html.EditorFor(model => model.Saida, new { htmlAttributes = new { @class = "form-control", {
26                  <br />
27                  @Html.EditorFor(model => model.Destino, new { htmlAttributes = new { @class = "form-control"
28                  @Html.HiddenFor(model => model.LatitudeSaida)
29                  @Html.HiddenFor(model => model.LongitudeSaida)
30                  @Html.HiddenFor(model => model.LatitudeDestino)
31                  @Html.HiddenFor(model => model.LongitudeDestino)
32                  <br />
```

**Figura 16.** Troca de valores através da Model “Pesquisa”.

Finalizada a passagem de valores para o objeto, assim que o *form* for submetido, é criada automaticamente pelo ASP.NET MVC uma chamada no formato POST (envio de informações), a qual será recebida pela *Controller*.

Assim que o objeto “Pesquisa”, ou seja, a *Model*, chegar à *Controller*, inicia o processo de busca e criação da lista de preços com base nos parâmetros informados. Como o método de comunicação com os Web Services externos fica no projeto “EstimaTrajeto.csproj”, a classe “Trajeto.cs” deste projeto é instanciada e os parâmetros enviados para o método “BuscarPrecos”, conforme Figura 17.

```
[HttpPost]
0 references | 3 requests | 0 exceptions
public ActionResult Index(Pesquisa pesquisa)
{
    #region Declaração

    //Instanciação da classe Trajeto
    Trajeto trajeto = new Trajeto();
    //Lista que receberá o retorno do método BuscarPrecos da classe Trajeto
    List<Preco> listPrecos = new List<Preco>();

    var qtdPrecos = 0;
    JObject json = new JObject();

    #endregion

    #region Implementação

    //Chamada do método BuscarPrecos com os endereços vindos da Model
    listPrecos = trajeto.BuscarPrecos(pesquisa.Saida, pesquisa.Destino);
```

**Figura 17.** Chamada do método “BuscarPrecos”.

Após chamar o projeto “EstimaTrajeto”, é iniciada a elaboração da lista de preços. A primeira tarefa é buscar a latitude e longitude dos pontos informados, pois estes são os parâmetros obrigatórios, tanto do Web Service do Uber (UBER, 2017), quanto do Web Service Taxicode (TAXICODE, 2017).

O método que realiza a busca dessas informações retorna um objeto JSON, retornado pela API do Google Maps através de uma requisição do tipo GET, a qual pode ser observada na Figura 18.

```
        4 references | 0 exceptions
195     public JObject BuscarLatitudeLongitude(string endereco)
196     {
197         #region Declaração
198
199         string finalUrl = string.Empty;
200         JObject retorno = new JObject();
201
202         #endregion
203
204         #region Implementação
205
206         //URL
207         finalUrl = string.Format("https://maps.google.com/maps/api/geocode/json?address
208
209         //Request
210         var webRequest = WebRequest.Create(finalUrl);
211         if (webRequest != null)
212         {
213             webRequest.Method = "GET";
214             webRequest.Timeout = 12000;
215             webRequest.ContentType = "application/json";
216
217             //Response
218             using (Stream s = webRequest.GetResponse().GetResponseStream())
219             {
220                 using (StreamReader sr = new StreamReader(s))
221                 {
222                     var jsonResponse = sr.ReadToEnd();
223                     retorno = JObject.Parse(jsonResponse);
224                     return retorno;
225                 }
226             }
227         }
228         else
229         {
230             return null;
231         }
232
233         #endregion
234     }
235
```

**Figura 18.** Requisição GET que retorna um JSON com a geolocalização de um endereço.

A lógica de uma requisição feita a um Web Service de padrão REST através do .NET é bastante simples. Primeiramente, na linha 207, é criada uma URL, também chamada de URI (*Uniform Resource Identifier),* a qual receberá a localização do serviço, bem como seus parâmetros. Na linha 210, é realizada uma *WebRequest*, que é uma *request* a URI que foi criada anteriormente. Alguns atributos são incluídos ao *WebRequest*, onde os mais importantes e utilizados são:

- *Method*: Verbo da requisição (GET, POST, DELETE, PUT, etc.);
- *Timeout*: Tempo máximo que a requisição pode utilizar;
- *ContentType*: O tipo de conteúdo que será retornado.

Entre as linhas 218 e 226, realiza-se o processo de busca do conteúdo solicitado pela *request* e conversão para um objeto JSON. Nesta fase, dois métodos são obrigatórios: *GetResponse* e *GetResponseStream*. O primeiro, busca o retorno da requisição, e a transforma em uma *WebResponse*, enquanto o segundo, transforam essa *WebResponse* em um *Stream*, o qual permite a conversão através de um simples comando *parse*.

Tendo em vista que a estrutura de uma requisição é a mesma para todos Web Services no padrão REST, na Figura 19 há parte do XML de configurações, que conterá todos os dados necessários para a realização de uma chamada, como URL, verbo a ser utilizado, tipo de retorno, *timeout*, entre outros. A criação desse XML teve por objetivo, além de facilitar na criação das chamadas para diferentes Web Services, auxiliar na coleta mais rápida das informações a serem utilizadas, informando quais itens do retorno devem ser extraídos.

Portanto, na Figura 20, é apresentada a requisição padrão, para qualquer um dos Web Services cadastrados no XML de configuração. Em comparação com a requisição da Figura 18, existe apenas uma diferença: a possibilidade de inclusão de cabeçalho na *request*. Muitos Web Services não permitem o retorno da informação sem a adição de alguns dados de acesso (normalmente um *token*). Nos casos que estamos utilizando, por exemplo, a API do Uber necessita de *token*, enquanto que o Taxicode não solicita. Sendo assim é necessário incluir um *header* na *WebRequest* (linhas 419 à 423).

```xml
<?xml version="1.0" encoding="utf-8" ?>
<options>
  <option>
    <name>Uber</name>
    <key>
      <name>Authorization</name>
      <value>Token jbABdC8-0mMxXdvc8otT_oeeyBz2cunyhY-FW06E</value>
    </key>
    <url>https://api.uber.com/v1.2/</url>
    <method>estimates/price</method>
    <rest>GET</rest>
    <type>application/json</type>
    <timeout>12000</timeout>
    <parameters>
      <parameter optional="false">start_latitude</parameter>
      <parameter optional="false">start_longitude</parameter>
      <parameter optional="false">end_latitude</parameter>
      <parameter optional="false">end_latitude</parameter>
      <parameter optional="true">seat_count</parameter>
    </parameters>
    <returns>
      <return>
        <name>display_name</name>
        <translate>Carro</translate>
        <status>1</status>
      </return>
      <return>
        <name>price</name>
        <translate>Preço</translate>
        <status>1</status>
      </return>
    </returns>
    <status>1</status>
  </option>
  <option>
    <name>Taxicode</name>
    <key>
      <name></name>
      <value></value>
    </key>
    <url>https://api.taxicode.com/</url>
    <method>booking/quote</method>
    <rest>GET</rest>
    <type>application/json</type>
    <timeout>12000</timeout>
    <parametros>
      <parameter optional="false">pickup</parameter>
      <parameter optional="false">destination</parameter>
      <parameter optional="false">date</parameter>
    </parametros>
```

**Figura 19.** XML de configuração das requisições.

```
412     var webRequest = WebRequest.Create(finalUrl);
413
414     if (webRequest != null)
415     {
416         webRequest.Method = rest;
417         webRequest.Timeout = Convert.ToInt32(timeout);
418         webRequest.ContentType = type;
419         if (!(token == "" || token == null))
420         {
421             //Adição do cabeçalho (token de autenticação)
422             webRequest.Headers.Add(nameToken, string.Concat(token));
423         }
424
425         using (Stream s = webRequest.GetResponse().GetResponseStream())
426         {
427             using (StreamReader sr = new StreamReader(s))
428             {
429                 var jsonResponse = sr.ReadToEnd();
430                 requisicao.resultadoRequisicao = JObject.Parse(jsonResponse);
431                 return requisicao;
432             }
433         }
434     }
435     else
436     {
437         throw new Exception("error");
438     }
```

**Figura 20.** Requisição uniforme para todas as opções cadastradas no XML de configuração.

Finalizada a requisição e retornado o objeto JSON com o conteúdo solicitado, é realizada a inclusão do mesmo em uma lista. Tal lista é tratada, de modo que os dados necessários para apresentação na tela sejam extraídos do objeto. Como o retorno de ambas API's é JSON, foram utilizados métodos nativos do .NET para extração das informações apontadas no XML de configuração. Um exemplo do uso destes métodos pode ser observado na Figura 21, onde se busca a quantidade de nós filhos existentes em um determinado *array* (existente no JSON).

```
var qtdPrecos = resultado.resultadoRequisicao.SelectTokens(array).Children().Count();
```

**Figura 21.** Extração da quantidade de preços que foram retornados pela requisição.

Com a lista pronta, é possível retorná-la para a *Controller*. Esta precisa ordenar a lista de forma que os preços mais baratos sejam apresentados antes, a fim de facilitar a

visualização dos dados por parte do usuário. Para executar esta tarefa, foi criado um método chamado "OrdenarPrecos", que, com a utilização de expressões lambdas, permite a ordenação com uma só linha de código (linha 98), vide Figura 22.

```
88   /// <summary>
89   /// Método que ordena a lista de preços com base no valor
90   /// </summary>
91   /// <param name="listPrecos">Lista de Preços retornados</param>
92   /// <returns>Retorna a lista de preços ordenada</returns>
     1 reference | 0 requests | 0 exceptions
93   public List<Preco> OrdenarPrecos(List<Preco> listPrecos)
94   {
95       try
96       {
97           //Expressão lambda que ordena a lista através de seu valor
98           return listPrecos.OrderBy(x => x.retorno.valor).ToList();
99       }
100      catch (Exception ex)
101      {
102          throw;
103      }
104  }
105
```

**Figura 22.** Método que ordena a lista de preços com expressão lambda.

Para finalizar o processo, é necessário apenas enviar os dados para a *View* através da *Model* "Pesquisa". Os dados que precisam ser enviados para que o Google Maps API crie a rota no mapa são:

- "LatitudeSaida": latitude do endereço onde irá partir a viagem;
- "LongitudeSaida": longitude do endereço onde irá partir a viagem;
- "LatitudeDestino": latitude do endereço destino da viagem;
- "LongitudeDestino": longitude do endereço destino da viagem.

Além destes dados, é necessário enviar o atributo "ListPrecos" da *Model*, a fim de que a *View* consiga exibir a estimativa de preço e a respectiva companhia na página. Feito isso, é possível atualizar a página com base no objeto finalizado.

No momento de atualização da página, a *View* busca os dados da *Model*, faz um laço de repetição, e cria uma tabela com a lista. Esse processo está na Figura 23, enquanto que a tela finalizada tanto com rota, quanto com a tabela de retorno, estão na Figura 24.

```
36                  if (Model.ListPrecos != null && Model.ListPrecos.Count > 0)
37                  {
38                      <div class="table-responsive">
39                          <table class="table">
40                              <tr>
41                                  <th><label class="label-info">#</label></th>
42                                  <th>Companhia</th>
43                                  <th>Carro</th>
44                                  <th>Valor</th>
45                              </tr>
46                              @{foreach (var item in Model.ListPrecos)
47                                  {
48                                      <tr>
49                                          <td><input type="radio" class="radio" /></td>
50                                          <td>@item.name</td>
51                                          <td>@item.retorno.nomeCarro</td>
52                                          <td>£@item.retorno.valor.ToString()</td>
53                                      </tr>
54                                  }
55
56                              }
57                          </table>
58                      </div>
59                      <br />
60                      <input type="button" class="btn btn-default" value="Comprar" />
61                  }
```

**Figura 23.** Código que exibe a tabela na página após realização da busca de preços.

Stamford Bridge

Londres

Pesquisar

| # | Companhia | Carro | Valor |
|---|---|---|---|
| ○ | Uber | uberPOOL | £10 |
| ○ | Uber | uberX | £10 |
| ○ | Uber | uberASSIST | £10 |
| ○ | Uber | ACCESS | £10 |
| ○ | Uber | uberXL | £14 |
| ○ | Uber | UberEXEC | £19 |
| ○ | Uber | UberLUX | £29 |
| ○ | Taxicode | Diamond Cars | £29 |
| ○ | Taxicode | Green London Cars | £175 |
| ○ | Taxicode | Plaza Chauffeur Cars | £185 |

Comprar

**Figura 24.** Pesquisa finalizada apresentando rota e estimativas na tela.

## 7. CONSIDERAÇÕES FINAIS

Este trabalho teve por finalidade apresentar os principais conceitos do uso de tecnologias REST, principalmente com a biblioteca .NET Framework, através de uma abrangente pesquisa bibliográfica.

Além disso, o trabalho utilizou um estudo de caso para exemplificar os estudos realizados na pesquisa. Este, se baseou no desenvolvimento de um site ASP.NET MVC, que teve por objetivo realizar o consumo de dois Web Services externos, onde os ambos são disponibilizados no padrão REST.

Os Web Services usados foram de dois meios de locomoção amplamente difundidos: Uber e Taxicode (para busca de táxis). Visando disponibilizar o melhor preço para um determinado trajeto, os retornos em formato JSON foram tratados e apresentados na tela em forma de tabela, além da rota completa poder ser visualizada em um mapa.

Portanto, é possível constatar que os Web Services são uma excelente forma de unir duas aplicações que ficam em locais diferentes, ou que estão escritas em distintas tecnologias, porém, os dados de uma é essencial na outra. No estudo de caso, por exemplo, o benefício adquirido com o uso de Web Services foi à economia na realização de curtas viagens.

Algumas sugestões para melhoria desse exemplo seria a criação de uma aplicação *mobile* com o mesmo *core* do site, que ajude ainda mais o usuário nessas situações. Também, a inclusão de mais dados, vindos de outras companhias, para que mais opções sejam obtidas.

# 8. REFERÊNCIAS


AÉCE, Israel. **Por dentro da Base Classe Library - Introdução.** 2017. <http://www.linhadecodigo.com.br/artigo/2051/por-dentro-da-base-classe-library-introducao.aspx>. Acesso em: 22 nov. 2017.

APACHE HTTP SERVER PROJECT (Estados Unidos). **Apache HTTP Server Project.** 1995. Disponível em: <http://httpd.apache.org/>. Acesso em: 14 jun. 2017.

ASSUMPÇÃO, Fabrício. **Representação no domínio bibliográfico: um olhar sobre os Formatos MARC 21**. 2015. Disponível em: < http://fabricioassumpcao.com/tag/marcxml>. Acesso em 23 nov. 2017.

AVELLAR E DUARTE. **SOA**. 2012. < http://www.avellareduarte.com.br/projeto/conceitos/webservices/webservicesa.htm>. Acesso em: 22 nov. 2017.

ERL, Thomas. **Introdução às tecnologias Web Services: SOA, SOAP, WSDL e UDDI - Parte1.** 2009. Disponível em: <http://www.devmedia.com.br/introducao-as-tecnologias-web-services-soa-soap-wsdl-e-uddi-parte1/2873>. Acesso em: 14 jun. 2017.

HANSELMAN, Scott. **ASP.NET MVC 2 Basics**. 2010. Disponível em: <http://channel9.msdn.com/blogs/matthijs/aspnet-mvc-2-basics-introduction-by-scotthanselman>. Acesso em: 14 jun. 2017.

JOSUTTIS, Nicolai M. **SOA na Prática**, Editora Alta Books, Edição 1, 2008, ISBN: 9788576081845.

LAZAR, Jonathan; FENG, Jinjuan Heidi; HOCHHEISER, Harry. **Research Methods in Human Computer**. Interaction, Wiley, 2010. ISBN 0-470-72337-8, 978-0-470-72337-1.

LOENERT, Laura. **O que é JSON? JavaScript Object Notation desvendado**. 2017. Disponível em: <http://blog.rivendel.com.br/2017/09/12/o-que-e-json-javascript-object-notation-desvendado/>. Acesso em 23 nov. 2017.

MARINHO, Paulo R. **Web Services Description Language, WSDL**. 2013. Disponível em: <http://www.webcodefree.com.br/blog/?p=972>. Acesso em 23 nov. 2017.

MATRAKAS, Miguel. **Introdução a web services RESTful**. 2017. Disponível em: <https://www.devmedia.com.br/introducao-a-web-services-restful/37387>. Acesso em 22 nov. 2017.

MINDMINERS. **Pesquisa quantitativa e qualitativa: qual é a melhor opção?** 2016. Disponível em: <https://mindminers.com/pesquisas/pesquisa-qualitativa-quantitativa>. Acesso em: 23 nov. 2017.

ORGANIZATION FOR THE ADVANCEMENT OF STRUCTURED INFORMATION STANDARDS (Estados Unidos). Oasis (Org.). **Introduction to UDDI: Important Features and Functional Concepts.** 2004. Disponível em: <http://uddi.xml.org/files/uddi-tech-wp.pdf>. Acesso em: 14 jun. 2017.

PAPAZOGLOU, Michael P. **Web Services: Principles and Technology**, INFOLAB/CRISM, Tilburg University, The Netherlands, 2008, ISBN: 978-0-321-15555-9.

RAMALHO, José Carlos ; SIMÕES, Alberto ; CARRIÇO, Luís, ed. lit. - "XATA2007 : **XML : aplicações e tecnologias associadas** : actas da Conferência Nacional, 5, Lisboa, 2007." [S.l. : s.n.], 2007, ISBN 978-972-99166-4-9. p. 33–46.

RITTER, Felipe. **Desenvolvimento em ASP.NET MVC utilizando Entity Framework**. 2011. 58 f. Trabalho de Conclusão de Curso (Graduação) - Universidade Tecnológica Federal do Paraná, Medianeira, 2011.

ROLIM, Carlos Oberdan. **Tópicos de Sistemas de Informação A**. 2014. Disponível em: <http://slideplayer.com.br/slide/1265748/>. Acesso em: 23 nov. 2017.

SILVA, Bruno Luiz Pereira da. **Web Services REST**. 2008. Disponível em: <http://www.linhadecodigo.com.br/artigo/2059/web-services-rest.aspx>. Acesso em 23 nov. 2017.

SOSNOSKI, Dennis. **Entendendo e Modelando o WSDL 1.1**. 2011. IBM Corporation. Disponível em: <https://www.ibm.com/developerworks/br/library/j-jws20/j-jws20-pdf.pdf>. Acesso em 23 nov. 2017.

TAXICODE (Org.). **Taxicode API Documentation**. 2017. <https://api.taxicode.com/#booking/quote>. Acesso em 8 nov. 2017.

UBER (Org). **GET /estimates/price**. 2017. <https://developer.uber.com/docs/riders/references/api/v1.2/estimates-price-get>. Acesso em 21 set. 2017.