

# **DISCIPLINA DE DESENVOLVIMENTO WEB COM ASP.NET MVC**



## SUMÁRIO

Apresentação.....	5
UNIDADE 1.....	7
1. Configurando o ambiente.....	7
2. Conhecendo uma aplicação ASP.NET MVC .....	15
2.1 Padrão Arquitetural MVC .....	15
3. Criando nosso primeiro projeto .....	18
4. Fundamentos do ASP.NET MVC – Parte 1 .....	24
4.1 Controllers (Controles) .....	24
4.2 Action Results .....	26
4.3 Views (Visão).....	27
4.4 Action Parameters .....	29
4.5 Routes (Rotas).....	30
5. Fundamentos do ASP.NET MVC – Parte 2 .....	33
5.1 View Models (Modelos).....	33
5.2 Passing Data to Views.....	35
5.3 Razor Syntax .....	38
5.4 Partial Views .....	39
6. Considerações finais .....	41
UNIDADE 2.....	42
1. Conhecendo o contexto do nosso projeto .....	42
1.1 Requisitos para o projeto do Blog pessoal .....	43
1.2 Modelagem e implementação do banco de dados .....	43
2. Desenvolvendo a regra de negócio da aplicação (back-end) .....	46
2.1 Criando o projeto do Blog pessoal .....	46
2.2 GitHub.....	47
2.3 Acessando o repositório GitHub pelo Visual Studio .....	48
2.4 Desenvolvimento.....	49
3. Desenvolvendo a apresentação do nosso site (front-end) .....	54
3.1 jQuery .....	54
3.2 Bootstrap .....	54
3.3 Layouts e Sections .....	55
4. Outros recursos .....	60
4.1 Validação de Model .....	60
4.2 Action Filters .....	63

4.3 Escolher uma tema .....	64
4.4 Scaffolding .....	65
5. Considerações finais .....	67
UNIDADE 3 .....	68
1. Detalhando recursos do ASP.NET MVC .....	68
1.1 Trabalhando com Partial View.....	68
1.2 ASP.NET e SignalR .....	70
2. Organização do projeto .....	71
2.1 Camada de negócio (Business Logic Layer) .....	71
2.2 Camada de acesso a dados (Data Access Layer) .....	78
2.3 Camada de apresentação (User interface/Presentation) .....	82
2.4 ValidateAntiForgeryToken.....	92
2.5 Configurações no Web.config.....	93
2.6 Controle de transação por meio do Unit Of Work .....	94
2.7 Outros assuntos de segurança .....	94
3. Desempenho da aplicação.....	96
3.1 Compactação da requisição com a compressão GZIP e Deflate .....	96
3.2 Filtro para remoção de espaços .....	98
3.3 Bundle e Minification.....	100
3.4 Cache .....	104
3.5 Cookie .....	106
3.6 Chrome Developer Tool.....	107
3.7 Desempenho em consultas .....	108
3.8 Content Delivery/Distribution Network (CDN).....	113
4. Considerações finais .....	114
UNIDADE 4 .....	115
1. Internet Information Services (IIS) .....	115
1.1 Gerenciador do Serviços de Informação da Internet (IIS) .....	116
2. Microsoft Azure .....	121
2.1 Application Insights .....	126
3. Qualidade do projeto e outros assuntos importantes .....	128
3.1 Rollbar.....	128
3.2 Search Engine Optimization (SEO).....	129
3.3 ASP.NET Web API.....	132
4. Certificações Microsoft.....	134

5. Dicas e materiais.....	136
6. Considerações finais .....	138

# Apresentação

Olá, pessoal. Tudo bem? Espero que sim, meu nome é João Vitor Ferrari da Silva e serei o professor da disciplina Desenvolvimento Web com ASP.NET MVC - Módulo I, da Especialização EAD em Arquitetura de Soluções em Plataforma.Net (C#). Sou tecnólogo em Análise de Sistemas, possuo uma especialização em Engenharia de Software e sou mestre em Informática na área de Inteligência Computacional, além de ser um Profissional Certificado da Microsoft<sup>1</sup> (MCP).

Iniciei na área de desenvolvimento de software em 2008 e até hoje trabalho utilizando o ambiente de desenvolvimento da Microsoft juntamente com a Linguagem de programação C# (C-Sharp). São mais de 10 anos de experiência adquirida por meio da formação acadêmica, do trabalho em algumas empresas que já passei, de dedicação e principalmente da troca de conhecimento entre outros desenvolvedores.

No começo trabalhei com codificação de aplicações desktop, conhecido por Windows Forms<sup>2</sup>, porém não demorou muito para iniciar o desenvolvimento de aplicações web, no início com framework ASP.NET WebForms<sup>3</sup> até chegar ao framework ASP.NET MVC<sup>4</sup>, tecnologia que trabalho desde 2012.

Na disciplina Desenvolvimento Web com ASP.NET MVC - Módulo I aprenderemos como desenvolver sistemas na estrutura MVC utilizando todos os recursos disponíveis pelo framework. Mostraremos passo a passo a codificação do projeto, iremos desde a definição da estrutura até a publicação do nosso sistema, sempre apresentando as boas práticas para o desenvolvimento. Trabalharemos com banco de dados SQL Server, utilizaremos Entity Framework e Dapper para consultas, conheceremos testes de unidade, padrões de projeto e ferramentas que nos auxiliam em cada processo do desenvolvimento. Ao final da disciplina você estará apto a desenvolver um sistema web com conexão ao banco de dados do zero utilizando ASP.NET MVC.

Nosso material será organizado em 5 módulos. No primeiro módulo abordaremos a configuração do ambiente de desenvolvimento e as ferramentas que utilizaremos durante toda a disciplina, além de construirmos nossa primeira aplicação com ASP.NET MVC. Já no segundo módulo iniciaremos o trabalho com dados em nossa aplicação MVC. No terceiro módulo aprenderemos como organizar melhor nosso projeto e discutiremos algumas boas práticas. O módulo quatro apresenta a questão de segurança, desempenho e integrações da nossa aplicação web. Por fim, no quinto módulo aprenderemos como publicar uma aplicação ASP.NET MVC e outras dicas para que você conheça um pouco mais sobre como melhorar seu sistema web.

Durante a leitura deste livro, quando existir necessidade deixarei no rodapé da página alguma referência ou descriptivo sobre termos destacados pelos capítulos com a finalidade de detalhamento, caso tenha alguma possível dúvida ou interesse de aprofundar um pouco mais em outros assuntos.

Como pré-requisito para esta disciplina, precisaremos conhecer o básico de C# e ter muita vontade de aprender. Quero te ajudar nesta caminhada para sermos

---

<sup>1</sup> <https://www.microsoft.com/pt-br/learning/microsoft-certified-professional.aspx>

<sup>2</sup> <https://docs.microsoft.com/pt-br/dotnet/framework/winforms/>

<sup>3</sup> <https://docs.microsoft.com/pt-br/aspnet/web-forms/>

<sup>4</sup> <https://docs.microsoft.com/en-us/aspnet/mvc/>

desenvolvedores melhores. Caso queira me acompanhar, adicione-me para trocarmos algumas figurinhas e mantermos contato:

- **LinkedIn**: <https://www.linkedin.com/in/joao-vitor-ferrari-da-silva-70685526/>;
- **GitHub**: <https://github.com/poferrari>;
- **Email**: [dev.londrina@uniciv.com.br](mailto:dev.londrina@uniciv.com.br).

Até as aulas!

# UNIDADE 1

## Nossa primeira aplicação ASP.NET MVC

O desenvolvimento web se modificou e introduziu inúmeras novidades, como: novas técnicas de codificação, tags, ferramentas e revelou a ampla capacidade do que o JavaScript pode fazer.

A plataforma ASP.NET vem evoluindo para acompanhar o desenvolvimento web. Nesse contexto, a Microsoft desenvolveu um framework para desenvolvimento de aplicações Web, chamado ASP.NET MVC.

O ASP.NET MVC permite que os desenvolvedores tenham maior controle sobre código desenvolvido, é uma tecnologia mais leve e mais eficiente do que o outro framework de desenvolvimento web da plataforma ASP.NET, chamado ASP.NET Web Forms. Esta flexibilidade, proporciona ao programador uma liberdade para codificar da maneira que seja melhor para seu processo de desenvolvimento.

O framework ASP.NET MVC permite a utilização da linguagem C#.

### 1. Configurando o ambiente

As aulas serão apresentadas em um ambiente Microsoft, utilizando o sistema operacional Windows 10. Para o desenvolvimento será feito o uso da IDE da Microsoft, o **Visual Studio Community 2017**, juntamente com algumas extensões que auxiliam na codificação. A linguagem de programação abordada em nossas aulas será C#. Conceitos técnicos de programação, do framework, da ferramenta e mesmo da linguagem, sempre que forem utilizados, serão introduzidos e exemplificados.

Antes de começarmos é interessante conhecer as edições disponíveis do Visual Studio, conforme apresenta a Figura 1. O Visual Studio Community é a IDE gratuita para estudantes, por esse motivo que iremos utilizá-la em nossas aulas. Já o Visual Studio Professional e o Visual Studio Enterprise são edições pagas, o que difere entre elas é que na edição Enterprise encontra-se mais funcionalidades para auxiliar o desenvolvimento do que a edição Professional. Por fim, o Visual Studio Code é uma ferramenta gratuita, de código aberto e que roda em outros sistemas operacionais.

# Downloads do Visual Studio

Windows

macOS

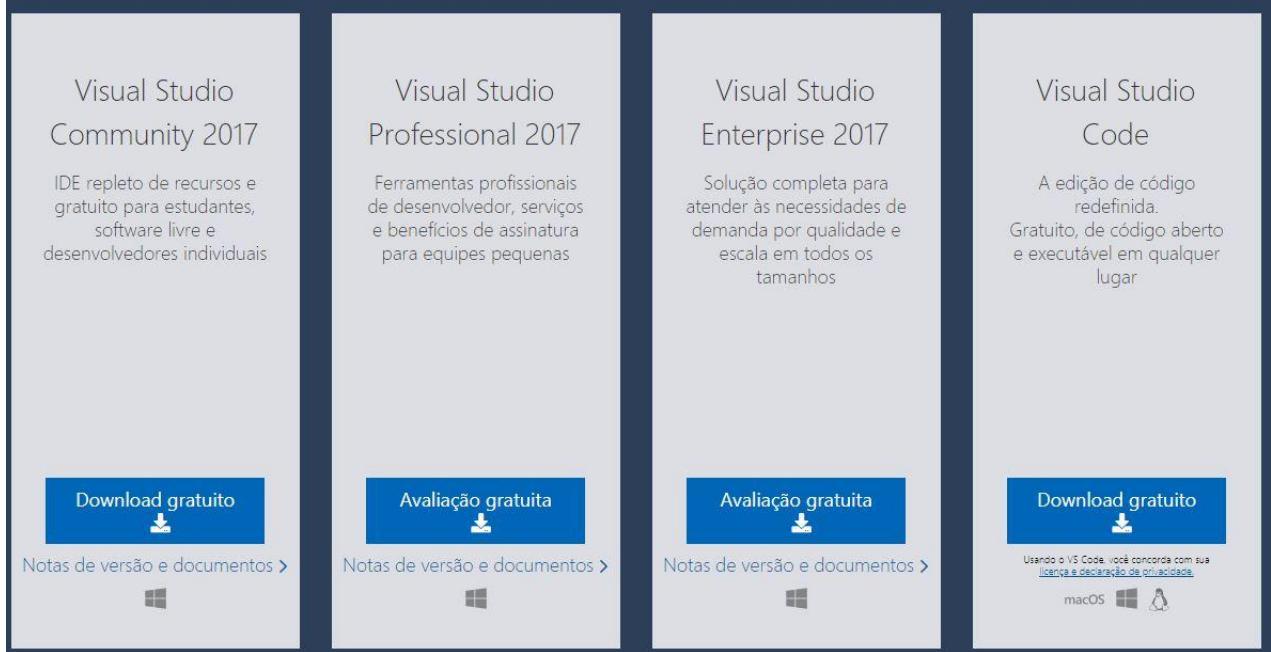


Figure 1: Diferentes edições da IDE do Visual Studio. Fonte: Adaptado de [26].

Para instalar a ferramenta **Visual Studio Community** basta acessar o seguinte endereço do próprio site da Microsoft: <https://visualstudio.microsoft.com/pt-br/vs/community/>. A instalação da IDE é simples, basta escolher os blocos das tecnologias desejadas e instalar. Recomenda-se a instalação dos seguintes blocos, representados nas Figuras 2 e 3, caso seja possível, para já deixar o ambiente preparado para as próximas disciplinas que serão estudadas e para o estudo de novas tecnologias, são eles:

- **Categoria Windows**: - .Net desktop development; - Universal Windows Platform development;
- **Categoria Web & Cloud**: - ASP.NET and web development; - Azure development; - Node.js development;
- **Categoria Mobile & Gaming**: - Mobile development with .NET;
- **Categoria Other Toolsets**: - Visual Studio extension development; - .NET Core cross-platform development.

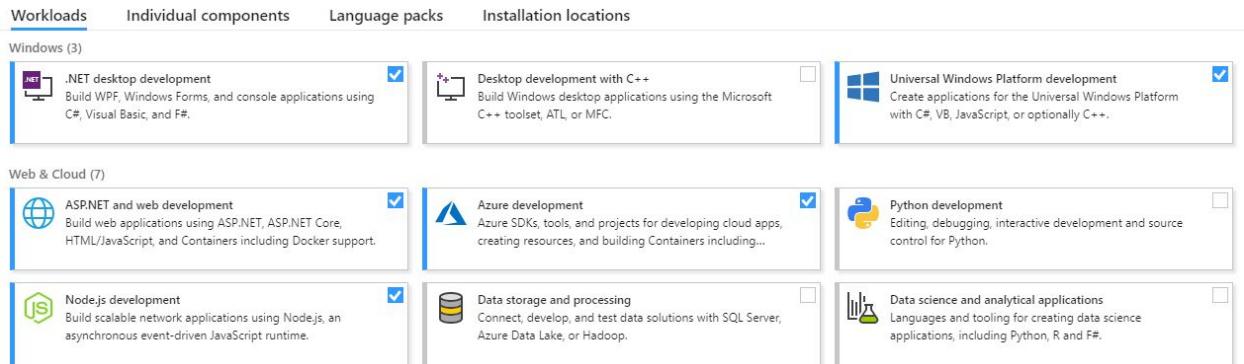


Figure 2: Opções de instalação do Visual Studio, categoria Windows e Web & Cloud.



Figure 3: Opções de instalação do Visual Studio, categoria Mobile & Gaming e Other Toolsets.

O Visual Studio notifica sobre atualizações da ferramenta e de novidades disponibilizadas pela Microsoft, conforme apresenta a Figura 4. Manter a sua IDE atualizada contribui para questão de segurança, desempenho entre outros aspectos.

Além disso, há diversas extensões disponíveis que nos auxiliam na questão de produtividade, boas práticas, integrações, qualidade do código, entre outras funcionalidades. As extensões podem ser acessadas por meio do próprio Visual Studio, basta acessar o menu principal localizado no começo da ferramenta, na opção Tools, selecionar a opção Extensions and Updates..., conforme é mostrada pela Figura 5. Outra maneira de acessar produtos e extensões do Visual Studio é por meio do seguinte endereço: <https://marketplace.visualstudio.com/>, pelo site você consegue ter mais detalhes sobre a extensão que desejar instalar, além de comentários e de avaliações dos usuários.

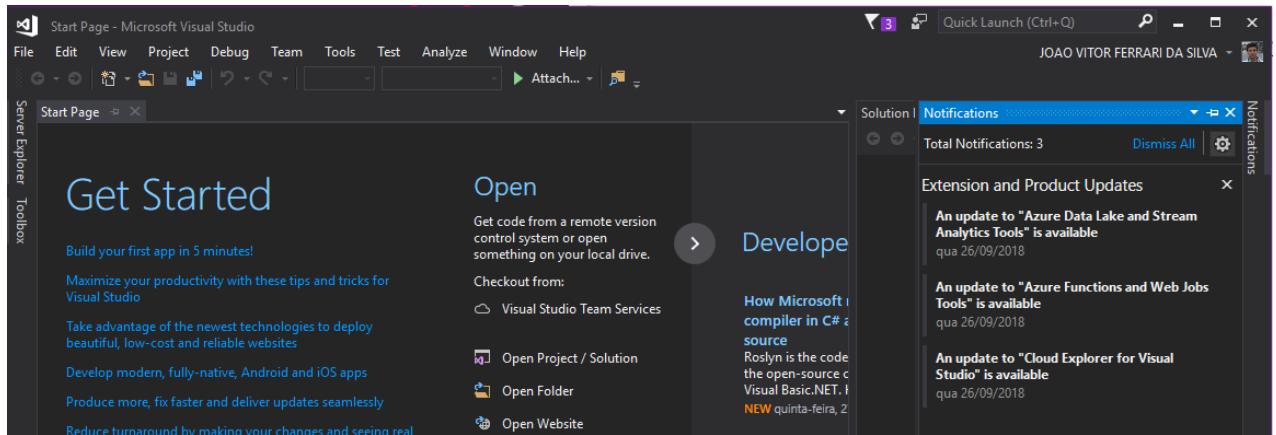


Figure 4: Notificação de atualizações do Visual Studio 2017.

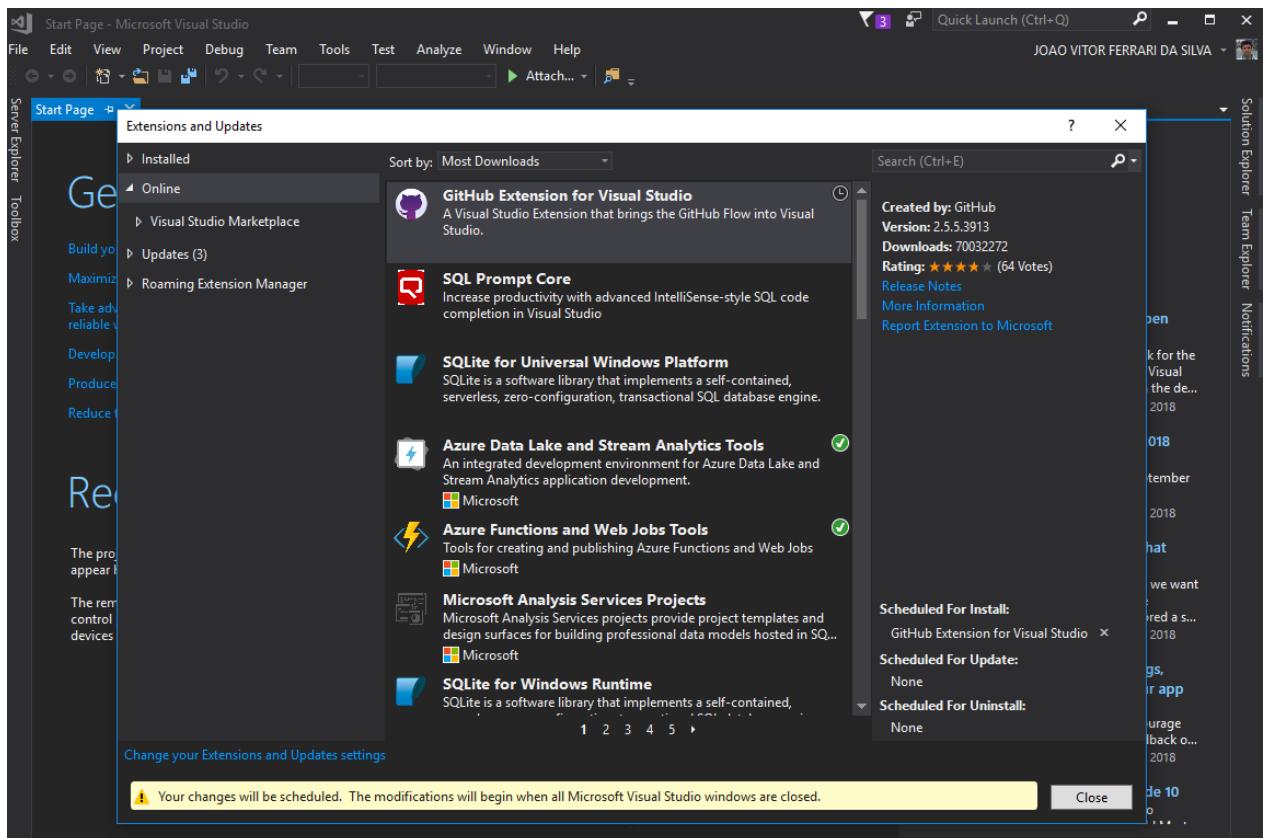


Figure 5: Tela para visualizar as extensões e atualizações disponíveis no Visual Studio 2017.

Entre tantas extensões disponíveis, umas gratuitas e outras não, há algumas que valem a pena serem citadas:

- **Web Essentials 2017:** adiciona várias ferramentas que facilitam o desenvolvimento web;
- **Productivity Power Tools 2017:** auxilia na produtividade durante o

processo de desenvolvimento;

- **Visual Studio Spell Checker**: realiza a verificação ortográfica do site;
- **Power Commands for Visual Studio**: adiciona ações que basicamente combinam várias funcionalidades que são executadas com apenas um clique;
- **SonarLint for Visual Studio 2017**: fornece aos desenvolvedores feedback em tempo real sobre a qualidade do código. Pode detectar problemas em segundos, o que melhora a produtividade;
- **ReSharper**: ferramenta que aumenta a produtividade do desenvolvedor, pois ele automatiza a maior parte do que pode ser automatizado em suas rotinas de codificação. Ela encontra erros de compilador, erros de tempo de execução, redundâncias e problemas de código à medida que você digita, sugerindo correções inteligentes para eles. É uma extensão paga, porém há disponibilidade de testar por 30 dias;
- **Roslynator 2017**: possui 400 analisadores que auxiliam nas refatorações e correções de código para o C#, é uma extensão que provavelmente chegará ao mais próximo do ReSharper de graça;
- **GitHub Extension for Visual Studio**: facilita a conexão e o trabalho com os repositórios no GitHub.

Após a instalação e configuração do Visual Studio, vamos para instalação do sistema gerenciador de banco de dados (SGBD). Utilizaremos o SQL Server, que é um SGBD relacional cliente/servidor da Microsoft e que implementa os padrões SQL (Structured Query Language).

Há várias edições do SQL Server, conforme apresenta a Figura 6. São elas:

- **SQL Server Enterprise**: é uma edição paga e mais robusta, não possui limitações de recurso;
- **SQL Server Standard**: é uma edição paga com algumas limitações de recurso;
- **SQL Server Express**: é uma edição gratuita, possui bastantes limitações, porém permite desenvolver aplicações comerciais;
- **SQL Server Developer**: é uma edição gratuita, disponibiliza vários recursos que são encontrados nas versões Standard e Enterprise, porém não permite que seja utilizado com fins comerciais, apenas para estudo.

Recursos	SQL Server 2017 Enterprise	SQL Server 2017 Standard	SQL Server 2017 Express	SQL Server 2017 Developer
Número máximo de núcleos	Ilimitado	24 núcleos	4 núcleos	Ilimitado
Memória: tamanho máximo do pool do buffer por instância	Máximo do sistema operacional	128 GB	1410 MB	Máximo do sistema operacional
Memória: cache máximo de segmentos Columnstore por instância	Máximo do sistema operacional	32 GB	352 MB	Máximo do sistema operacional
Memória: máximo de dados com otimização de memória por banco de dados	Máximo do sistema operacional	32 GB	352 MB	Máximo do sistema operacional
Tamanho máximo do banco de dados	524 PB	524 PB	10 GB	524 PB
Direitos de uso em produção	●	●	●	○
Virtualização ilimitada, um benefício de Software Assurance	●	○	○	○

Figure 6: Edições do Microsoft SQL Server 2017.

Para elaboração da aula será utilizada a versão SQL Server 2017 Express, por ser uma versão gratuita do SQL Server. No próprio site da ferramenta é descrito ser ideal para desenvolvimento e produção de aplicativos de área de trabalho, Web e pequenos servidores. O objetivo é deixar um ambiente de desenvolvimento completamente pronto para que seja possível desenvolver ferramentas que possam ser disponibilizadas para fins educacionais ou comerciais.

O download do Microsoft SQL Server 2017 Express é encontrado por meio do seguinte endereço:

<https://www.microsoft.com/en-us/download/details.aspx?id=55994>

Para criarmos nosso banco de dados utilizaremos o SQL Server Express LocalDB, recurso voltado a desenvolvedores, que é uma versão leve do Express e que possui todos os seus recursos de programação. Ele é executado no modo de usuário e tem uma instalação rápida e sem nenhuma configuração e uma lista curta de pré-requisitos (Microsoft).

Segundo a Microsoft, depois do LocalDB ser instalado, você poderá iniciar uma conexão usando uma cadeia de conexão especial. Ao conectar, a infraestrutura necessária do SQL Server é criada e iniciada automaticamente, permitindo que o aplicativo use o banco de dados sem tarefas de configuração complexas.

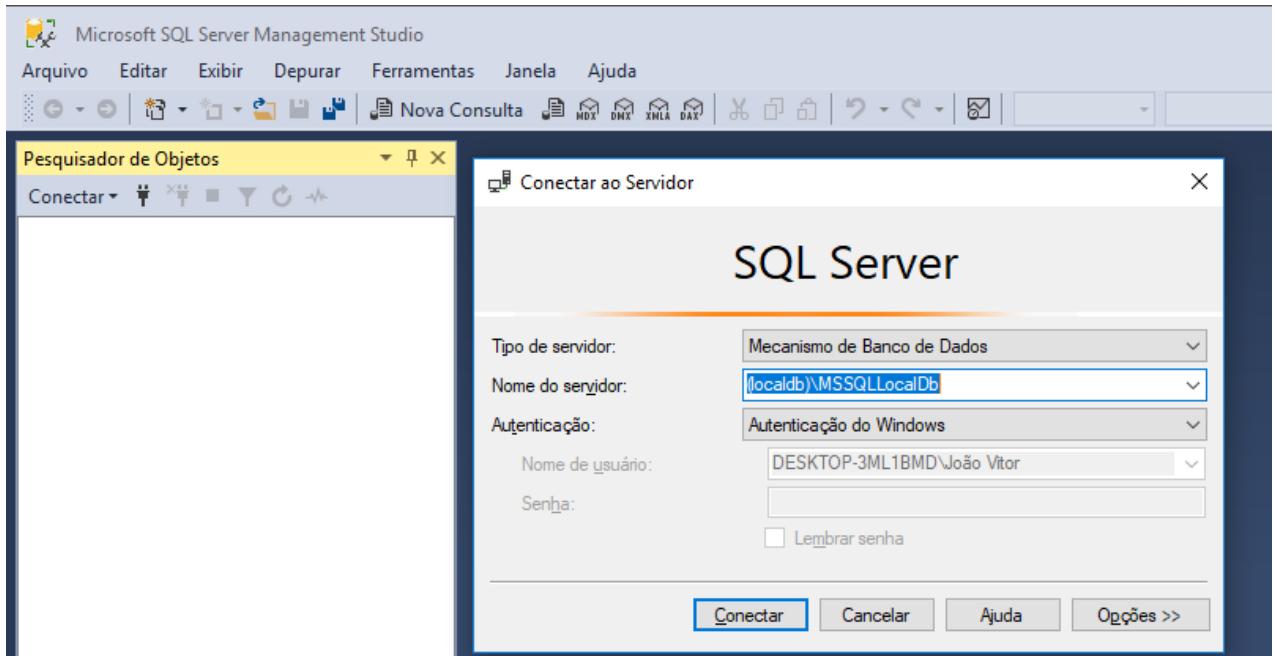


Figure 7: Acesso ao LocalDB por meio do Microsoft SQL Server Management Studio.  
Fonte: Autoria própria.

Por fim, é interessante realizar a instalação do SQL Server Management Studio (SSMS), ferramenta gratuita da Microsoft para gerenciar qualquer infraestrutura de SQL, do SQL Server para o Banco de Dados SQL do Microsoft Azure. Segundo a Microsoft, o SSMS fornece ferramentas para configurar, monitorar e administrar instâncias do SQL. Com o SSMS pode-se implantar, monitorar e atualizar os componentes da camada de dados usados pelos seus aplicativos, além de construir consultas e scripts. Detalhes sobre a instalação desta ferramenta encontra-se neste endereço: <https://docs.microsoft.com/pt-br/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-2017>.

A Figura 7 representa o acesso ao LocalDB por meio do Microsoft SQL Server Management Studio, este passo é importante para realizar a conexão ao banco de dados com sucesso.

É interessante que você tenha uma conta em algum repositório online, pois os códigos que iremos utilizar em nossas aulas serão disponibilizadas por meio de um destes repositórios. Em nossos exemplos utilizaremos o GitHub<sup>5</sup>, aplicação web que possibilita a hospedagem de repositórios Git, sistema de controle de versões gratuito para criação de repositórios públicos. O bom é que você pode disponibilizar seus códigos para que outras pessoas possam acessar e codificar contigo, além de ser um excelente portfólio para que outras pessoas vejam seu trabalho.

Para instalar o Git no Windows, basta acessar: <http://msysgit.github.io/>, fazer o download e instalar a última versão disponível. A instalação é bem simples e recomenda-se que escolha as opções padrão de configuração. Entre todos os programas que serão instalados, o que mais iremos utilizar será o Git Bash, ele permite que o Git seja executado pela linha de comando no Windows.

---

<sup>5</sup> <https://github.com/>

O Azure DevOps<sup>6</sup>, antigamente conhecido por Visual Studio Team System (VSTS), é um conjunto de aplicativos que auxiliam durante todo o ciclo de vida de desenvolvimento. Vale citar que você pode disponibilizar seu código por meio deste sistema também, pois a Microsoft libera um acesso gratuito, porém com algumas limitações, como por exemplo, a quantidade de usuários que podem ser adicionados no seu projeto.

Para criar sua conta gratuita, você precisará de uma conta da Microsoft e acessar a seguinte URL: <https://aka.ms/azdev-signin>. Iremos abordar as funcionalidades deste sistema com mais detalhes durante os próximos módulos, você aprenderá como realizar a gestão do seu projeto, a automatizar algumas rotinas de trabalho e a organização dos times por meio do Azure DevOps, além de outras funcionalidades que te auxiliarão em um fluxo de trabalho.

---

<sup>6</sup> <https://azure.microsoft.com/pt-br/services/devops/>

## 2. Conhecendo uma aplicação ASP.NET MVC

Antes de iniciarmos nossa aplicação ASP.NET MVC, precisamos conhecer a arquitetura de uma aplicação web do tipo Cliente Servidor. A Figura 8 demonstra como funciona a comunicação de uma aplicação nesta arquitetura. Pela imagem, pode-se ver o bloco com o título “User Workstation”, ele representa o acesso do usuário por meio de um navegador via requisição HTTP (HTTP request), sendo realizado este acesso por qualquer tipo de dispositivo com browser.

A requisição enviada pelo cliente é submetida ao servidor, no caso da figura representado pelo bloco do título “Web Server”. É no servidor que a requisição solicitada será processada e ao final enviará uma resposta ao cliente por meio de uma resposta via HTTP (HTTP response). É o servidor que possui a comunicação com o banco de dados e fica responsável por realizar os processos mais pesados da aplicação, ele recebe requisições de vários clientes diferentes. Já o cliente deve ter uma comunicação rápida pois tem relação com a experiência do usuário, trabalha principalmente com HTML (HyperText Markup Language), JavaScript e CSS (Cascading Style Sheets).

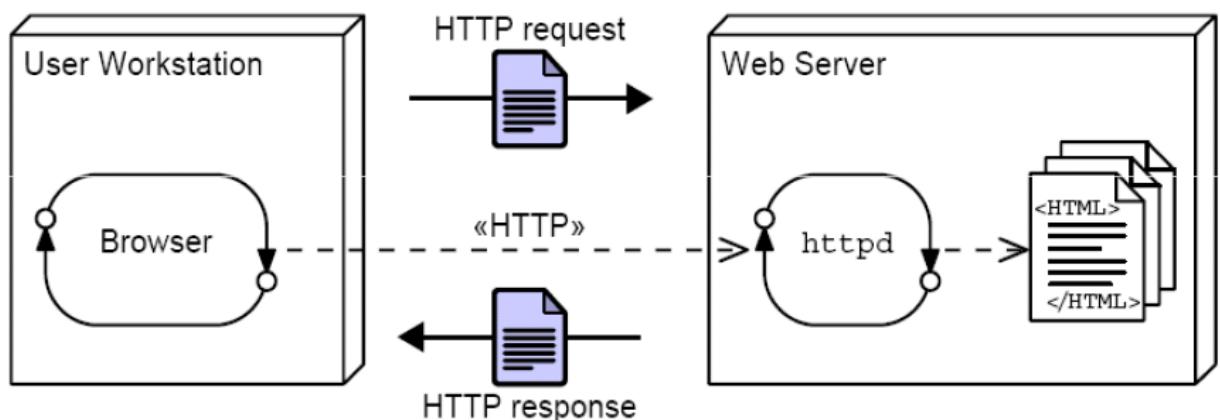


Figure 8: Diagrama de um servidor web.

### 2.1 Padrão Arquitetural MVC

O MVC (Model-View-Controller) é um padrão arquitetural na qual a aplicação é organizada em três camadas distintas, sendo os três principais componentes: modelo (Model), exibição (View) e controlador (Controller). Este padrão é independente de linguagem, há o MVC para PHP, Java, Python, entre outras tecnologias. No nosso caso aplicaremos o framework ASP.NET MVC.

O MVC visa separar as regras e lógicas do negócio da apresentação em si. Desse modo, permite um maior controle sobre a aplicação, além de facilitar a manutenção e garantir maior segurança do sistema. A estrutura MVC é organizada pelos seguintes componentes:

- Model: são as partes da aplicação que implementam a lógica para o domínio de dados da aplicação, além de retornar e armazenar o estado do modelo no banco de dados [16];
- View: são os componentes que exibem a interface do usuário (UI), em geral é criada a partir do modelo de dados [16];
- Controller: são os componentes que tratam com a interação do usuário, trabalham com o modelo e selecionam uma visão para ser exibida. Em uma aplicação MVC, a view somente exibe informação, o controller trata e retorna a entrada do usuário e a interação [16].

A Figura 9 mostra a sequência de uma solicitação no MVC. Pela figura visualiza-se que o usuário realiza uma requisição via browser, esta solicitação é interpretada por meio das rotas que são responsáveis por redirecionar esta requisição para o Controller encarregado. O controller processa a solicitação do usuário por meio do controle entre o Modelo de Dados (Model) e a apresentação da página (View), após o término do processo, ele encaminha a View como resposta ao browser.

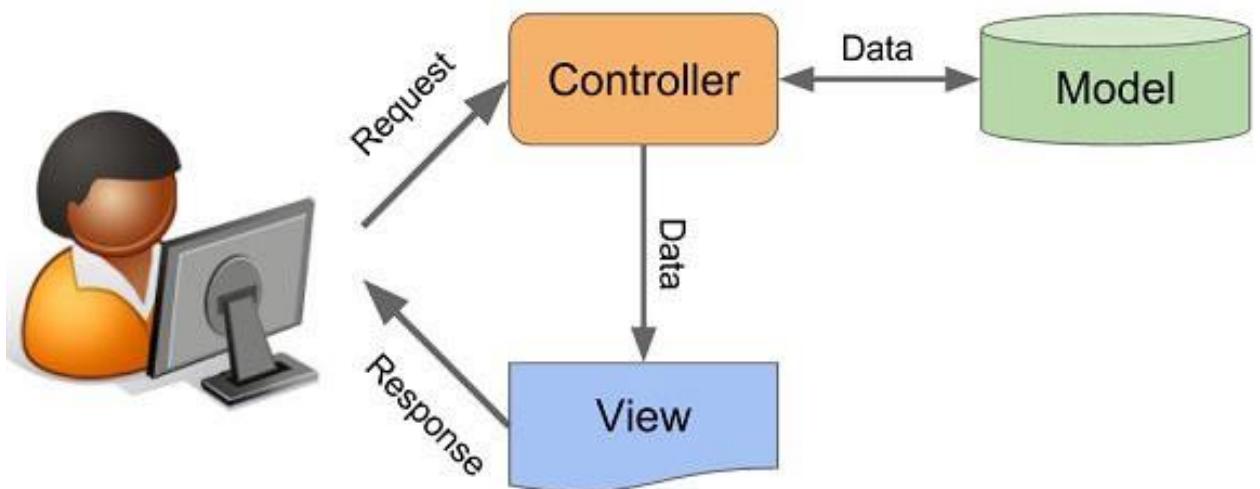


Figure 9: Representação do padrão MVC.

O ASP.NET MVC é uma estrutura de apresentação leve e altamente testável, integrada aos recursos ASP.NET existentes, como páginas mestras e autenticação baseada em associação, entre outras funcionalidades [16]. É um framework para desenvolvimento de aplicações web que funciona com base no .NET Framework e nos permite criar projetos robustos utilizando a linguagem C#.

A estrutura ASP.NET MVC fornece alguns recursos que trazem as seguintes vantagens:

- controle total sobre o HTML processado e sobre as requisições (URL);
- separação das regras e lógicas do negócio da apresentação em si, fornece separação das responsabilidades, o que viabiliza a manutenção isolada de ambas as partes;
- permite o desenvolvimento orientado a testes, chamado Test Driven

Development (TDD);

- facilidade de integração no lado do cliente com JavaScript, jQuery, JSON, entre outros;
- permite usar a arquitetura RESTful;
- torna a aplicação escalável;
- é possível ter desenvolvimento em paralelo para o model, view e controller pois são independentes;
- comunidade ativa trabalhando nesta tecnologia.

### 3. Criando nosso primeiro projeto

Após toda a instalação e configuração do ambiente, criaremos uma aplicação padrão do ASP.NET MVC. O objetivo aqui é apresentar o Visual Studio e as ferramentas disponíveis para se trabalhar com ASP.NET MVC, além de mostrar toda a estrutura do projeto, para que possamos assimiliar os conceitos e teorias estudadas até agora.

O primeiro passo será criar nosso primeiro projeto ASP.NET MVC, por meio do Visual Studio. Desse modo, pelo menu superior, acesse a seguinte sequência de opções: “File”, “New” e “Project”, após, será apresentada a tela de seleção para escolher o tipo de projeto, representada pela Figura 10. Na janela que é apresentada, na parte esquerda, que traz os templates disponíveis (modelos), selecione a linguagem Visual C# e, dentro desta categoria a opção Web (1).

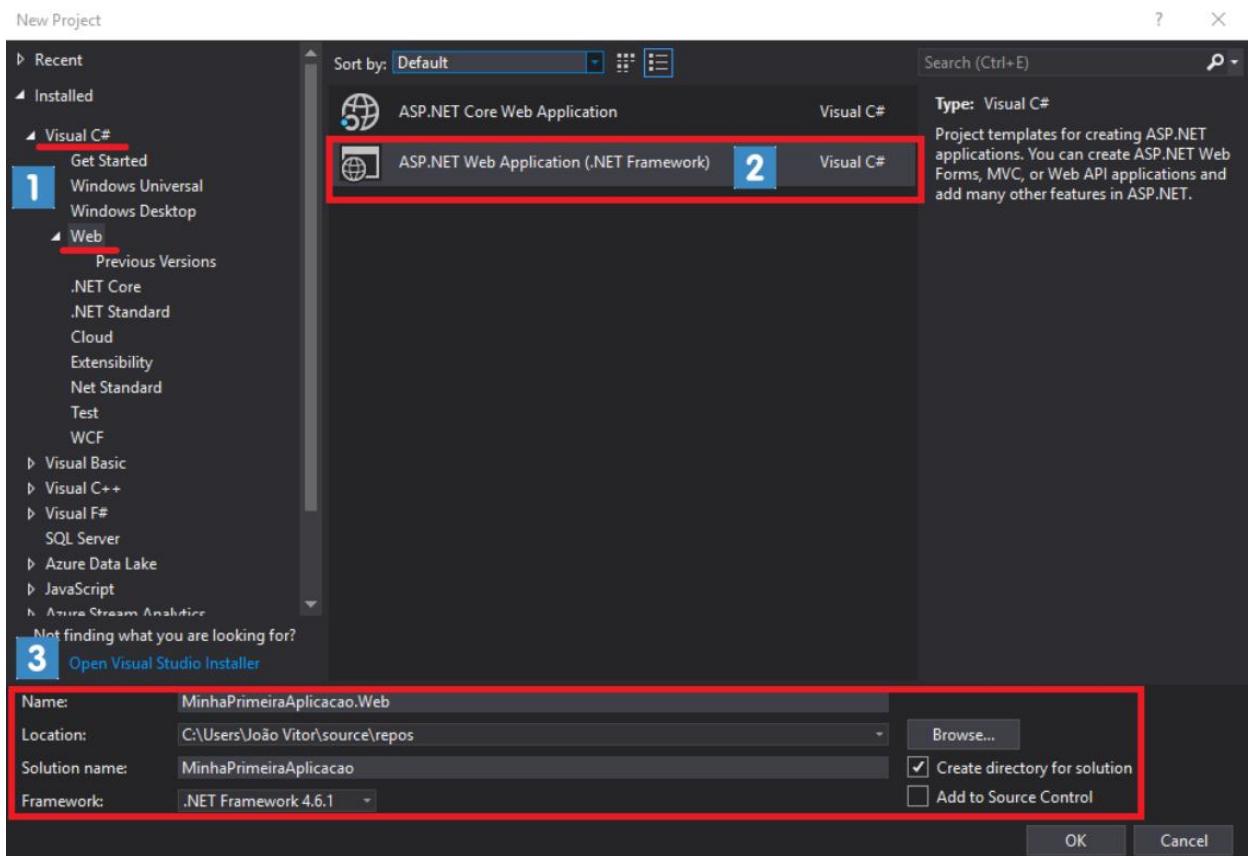


Figure 10: Janela do Visual Studio para criação de um novo projeto web ASP.NET MVC.

Seguindo pela Figura 10, na área central, marque a opção ASP.NET Web Application (.NET Framework) (2). Na parte inferior da janela (3), informe: (i) o nome do projeto, (ii) a localização em que ele ficará armazenado em sua máquina, (iii) o nome da solução que irá abranger todos os demais projetos, e (vi) a versão do .NET Framework que será utilizada, que em nossos exemplos será a versão 4.6. Caso você tenha uma conta no AzureDevops, e queria que o projeto seja adicionado automaticamente para controle de versão e uso das demais ferramentas disponíveis, basta clicar em “Add to Source Control”. Clique no botão OK para dar sequência ao

processo.

Na nova janela que se abre é preciso selecionar qual template de uma aplicação web deverá ser criado. Uma boa prática é selecionar a opção Empty, porém como é a nossa primeira aplicação, iremos selecionar a opção MVC (1), verifique que ao selecionar este template a opção MVC (2) ficou marcada automaticamente. Verifique que nesta tela há a opção para criar um projeto de teste, mantenha como padrão em seu desenvolvimento a prática de testes, desse modo, recomenda-se a seleção desta opção (3). Verifique se sua janela está conforme a Figura 11. Clique no botão OK para confirmar a seleção e o projeto será criado.

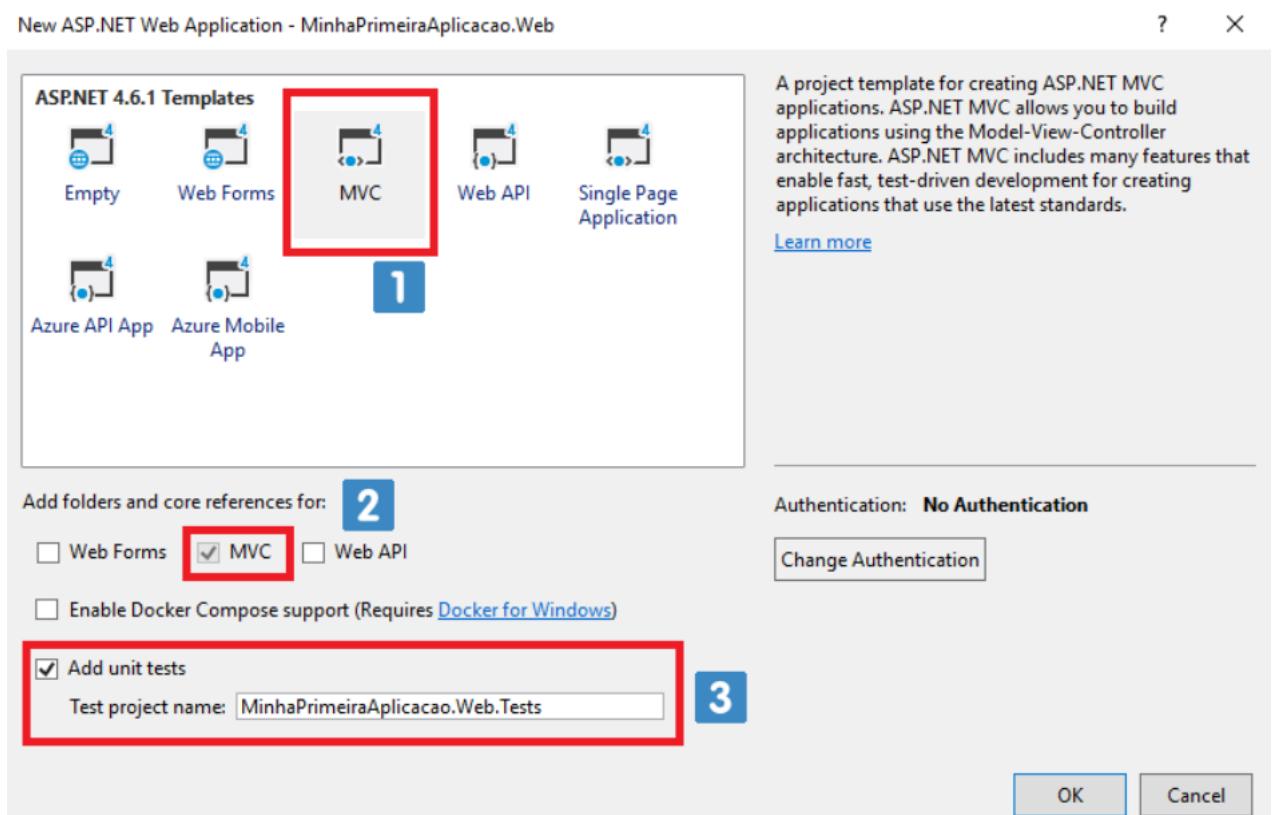


Figure 11: Selecionando o tipo de projeto ASP.NET MVC.

Após a criação ter sido realizada, é possível visualizar a estrutura do projeto na janela do "Solution Explorer" (1) (janela em que os arquivos do projeto são dispostos), como mostra a Figura 12. Veja a organização das pastas Controllers, Models e Views. Peço que não se preocupe agora com os detalhes da estrutura do projeto, eles serão detalhadamente explicados nos próximos capítulos, porém caso queria obter mais informações sobre o ASP.NET MVC ou sobre o Visual Studio, a área central (2) apresenta links para estudos.

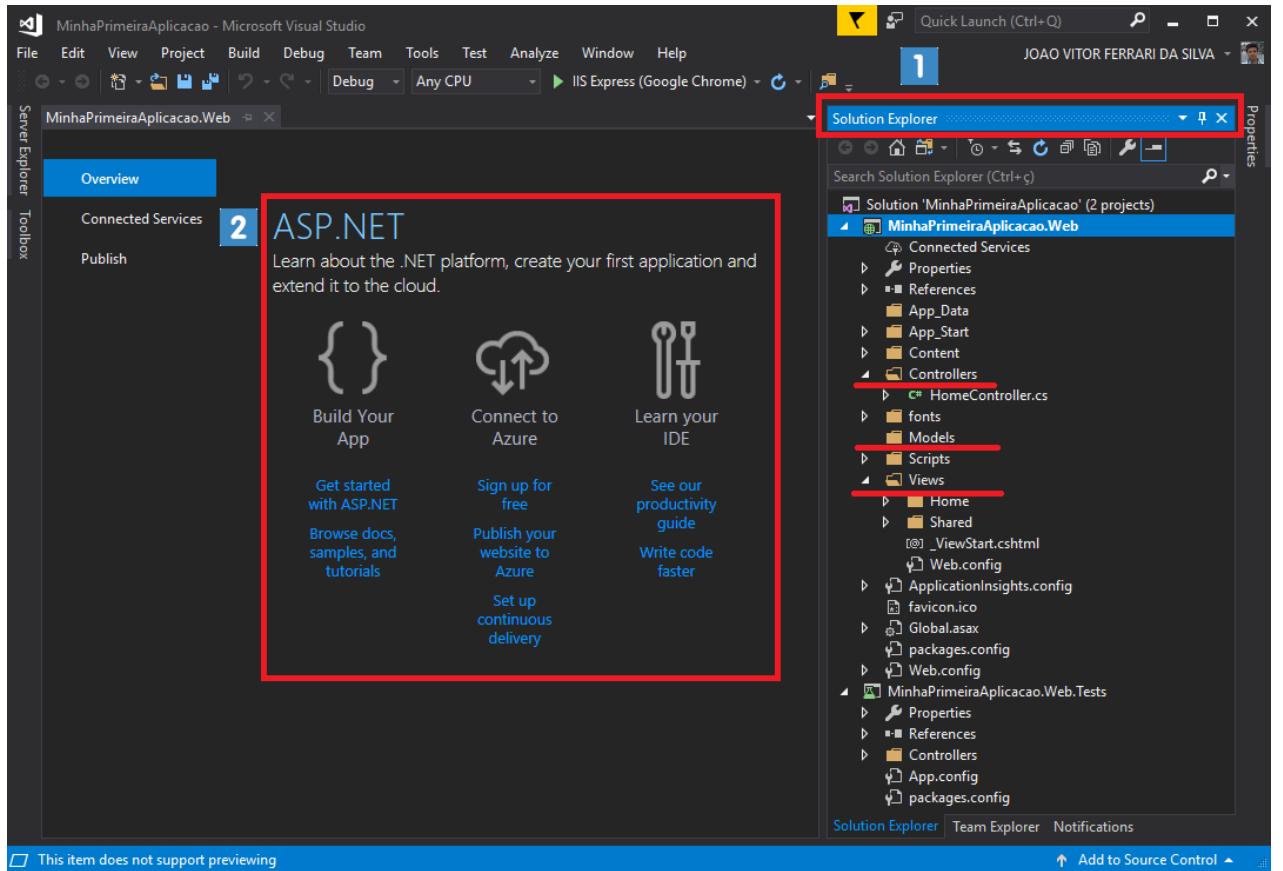


Figure 12: Estrutura inicial do projeto ASP.NET MVC no Visual Studio.

No ASP.NET MVC, uma requisição é direcionada a uma Action por meio de uma rota. A Action é um método de uma classe que representa um determinado controlador. O Controller é uma classe que representa um determinado Modelo de sua aplicação. É no Controller que são desenvolvidas as Actions [2].

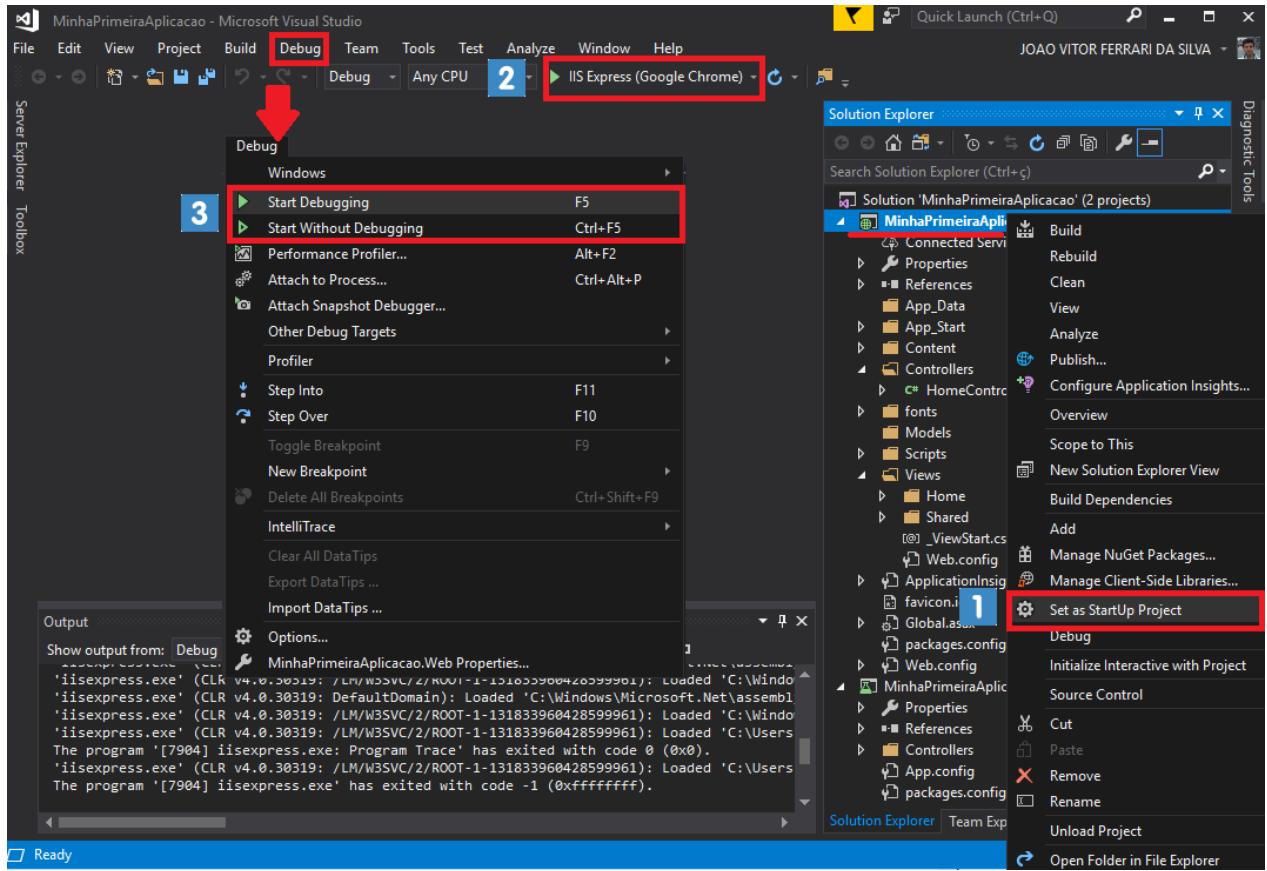


Figure 13: Copilando o projeto ASP.NET MVC no Visual Studio.

Antes de executar nosso projeto, precisamos setar nossa aplicação web como projeto que será iniciado pela solução. Para isso, basta acessar a janela do “Solution Explorer”, selecionar o projeto web, botão direito do mouse e clicar na opção “Set as StartUp Project” (1), verifique que o nome do projeto ficará em destaque, a Figura 13 demonstra a tarefa realizada.

Por fim, vamos executar o projeto, para isso podemos acessar o menu de atalhos do Visual Studio e clicar a seta verde (2) (você pode utilizar a tecla F5 como atalho para executar esta tarefa), ou pode acessar o menu superior do Visual Studio, a opção “Debug” (3) e clicar na opção “Start Debugging” (atalho F5) ou “Start Without Debugging (atalho Ctrl + F5)”. A Figura 13 apresenta as configurações para executar nosso projeto.

Uma página web carregada pelo browser deve aparecer para você. Desse modo, você acabou de criar e publicar localmente, sua primeira aplicação ASP.NET MVC. Vamos aproveitar para fazer uma pequena modificação, a fim de já conhecermos melhor nossa estrutura.

Na tela “Solution Explorer”, navegue até o arquivo “HomeController” (localizado em: MinhaPrimeiraAplicacao.Web > Controllers > HomeController). No método Index, que representa uma action. Faremos a seguinte alteração: adicionaremos uma mensagem na página inicial. Adicione o seguinte trecho de código: ViewBag.Message = “Minha primeira aplicação com ASP.NET MVC.”, conforme apresentado pela Figura 14.

```
public class HomeController : Controller
{
    1 reference | 0 requests | 0 exceptions
    public ActionResult Index()
    {
        ViewBag.Message = "Minha primeira aplicação com ASP.NET MVC.";

        return View();
    }
}
```

Figure 14: Adicionando um texto na Action Index da Controller Home.

Após precisamos adicionar nossa mensagem para apresentação ao usuário. Desse modo, navegue até a view “Index” da Controller Home (localizado em: MinhaPrimeiraAplicacao.Web > Views > Home > Index.cshtml). Adicione o seguinte trecho: @ViewBag.Message no HTML desta view, conforme mostrado pela Figura 15.

```
Index.cshtml ✘ X HomeController.cs
1  @{
2      ViewBag.Title = "Home Page";
3  }
4
5  <div class="jumbotron">
6      <h1>ASP .NET</h1>
7      <h2>@ViewBag.Message</h2>
8      <p class="lead">ASP.NET is a free web framework for building great Web sites and Web applications using
9          HTML, CSS and JavaScript.</p>
10         <p><a href="https://asp.net" class="btn btn-primary btn-lg">Learn more &raquo;</a></p>
11     </div>
```

Figure 15: Adicionando um texto na View Index da Controller Home.

Compile o projeto, conforme já aprendemos, e quando o site for carregado em seu navegador, a mensagem que acabamos de adicionar deverá ser exibida, a exemplo do que apresenta a Figura 16.

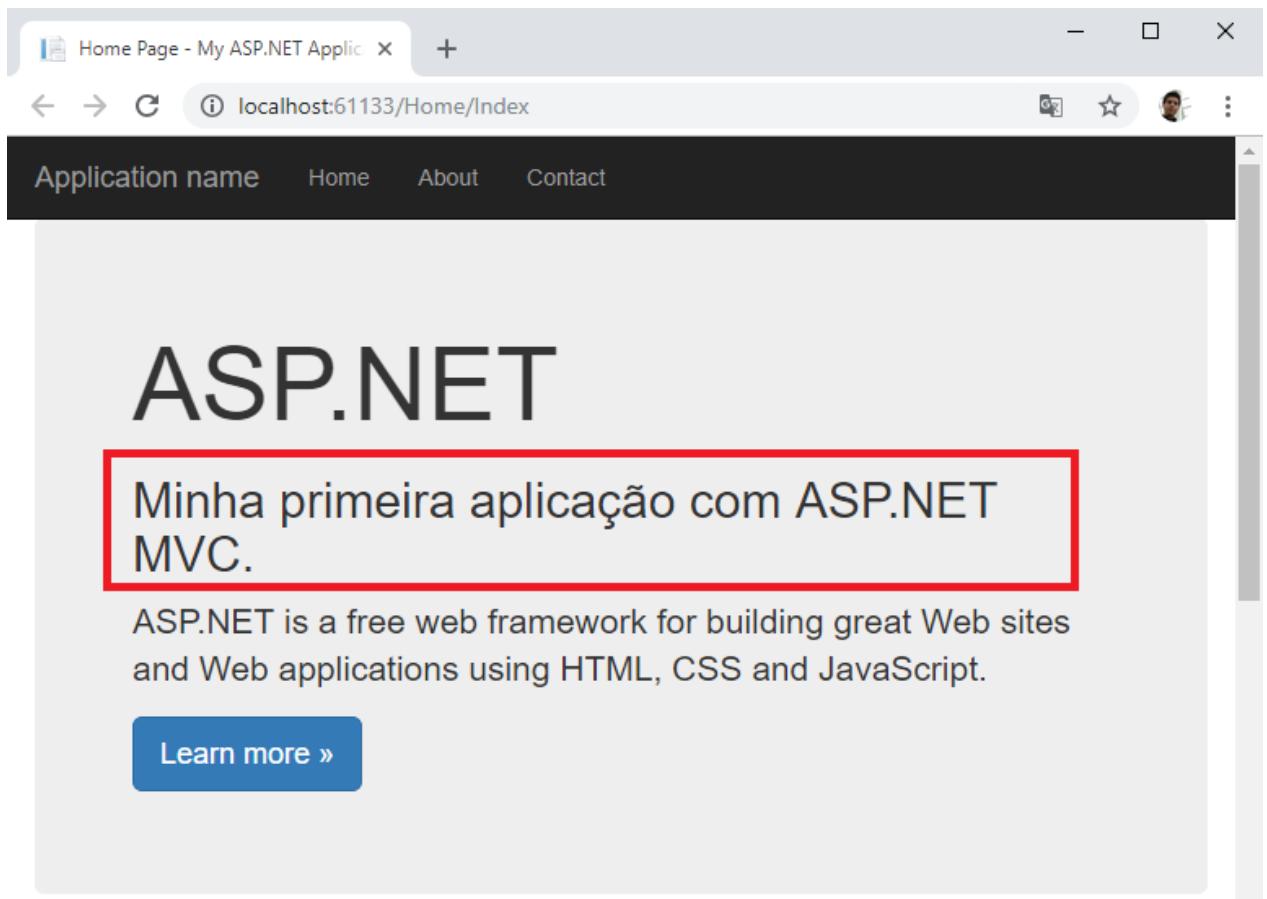


Figure 16: Primeira aplicação ASP.NET MVC funcionando e apresentando a mensagem desenvolvida.

## 4. Fundamentos do ASP.NET MVC – Parte 1

Neste capítulo, aprenderemos os principais recursos do ASP.NET MVC. Saberemos como criar uma Controller, construiremos Actions e visualizaremos os diferentes tipos de retornos que existem, implementaremos Views e conheceremos a sintaxe do Razor, além de vários outros recursos disponíveis pelo framework.

Também conheceremos algumas das convenções utilizadas pela configuração do framework ASP.NET MVC, em que estas visam simplificar e diminuir as decisões que precisam ser tomadas pelo desenvolvedor. O trabalho fica muito mais fácil e simples já que algumas decisões estruturais e de nomes são sugeridos ou até exigidos em alguns casos [23].

### 4.1 Controllers (Controles)

O Controller no padrão MVC possui a responsabilidade de responder a entrada do usuário, muitas vezes fazendo mudanças no modelo em resposta à solicitação do usuário. Desse modo, o Controller está preocupado com o fluxo do aplicativo, trabalhando com dados que chegam e fornecendo dados de saída para a visualização do usuário.

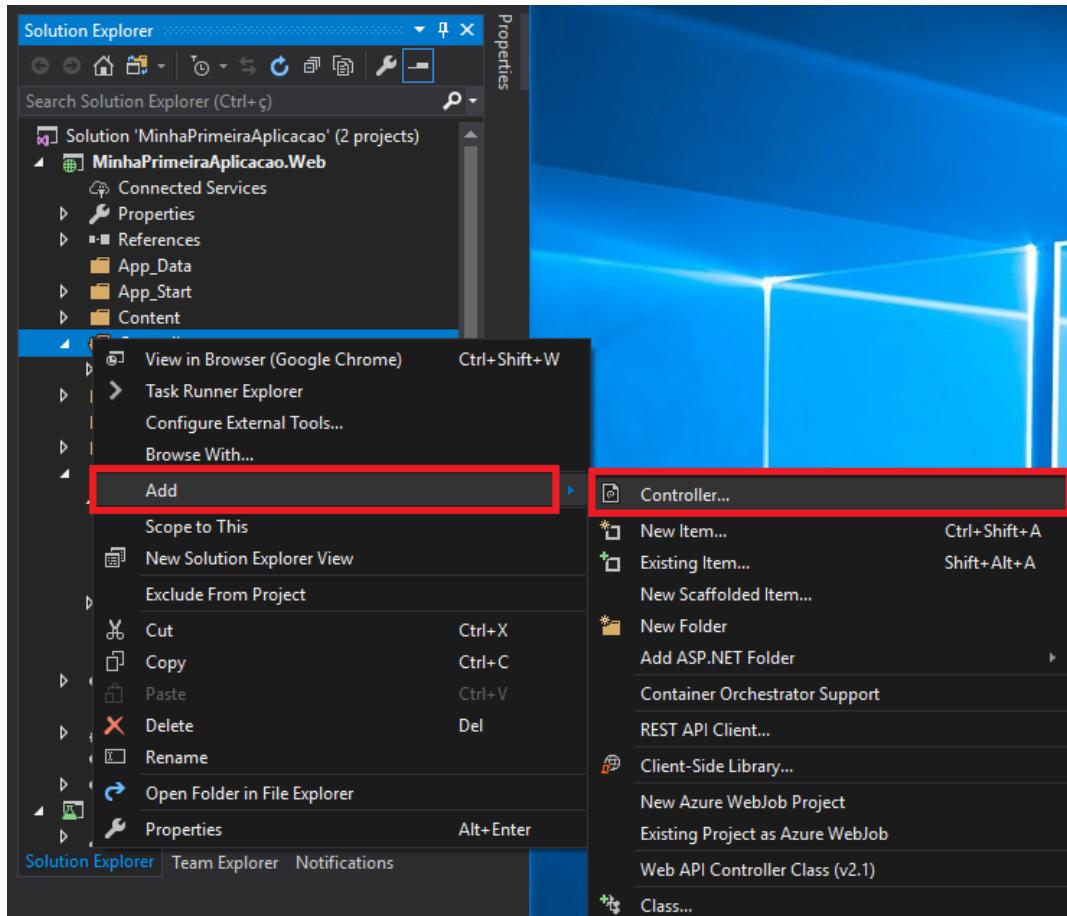


Figure 17: Opção para adicionar um Controller.

Para criar nosso Controller, basta acessar a tela “Solution Explorer”, clicar com

o botão direito do mouse sobre a pasta Controllers e clicar na opção Add > Controller, conforme demonstra a Figura 17. Na janela seguinte serão apresentados vários templates de Controllers. Pela Figura 18, visualizamos que são apresentados templates de Controller do MVC 5, que é o objeto do nosso estudo, e também de Web API 2, que será futuramente estudado em outra disciplina. No nosso caso selecionaremos o template MVC 5 Controller - Empty. Para iniciar o processo de criação do Controller, clique no botão Add.

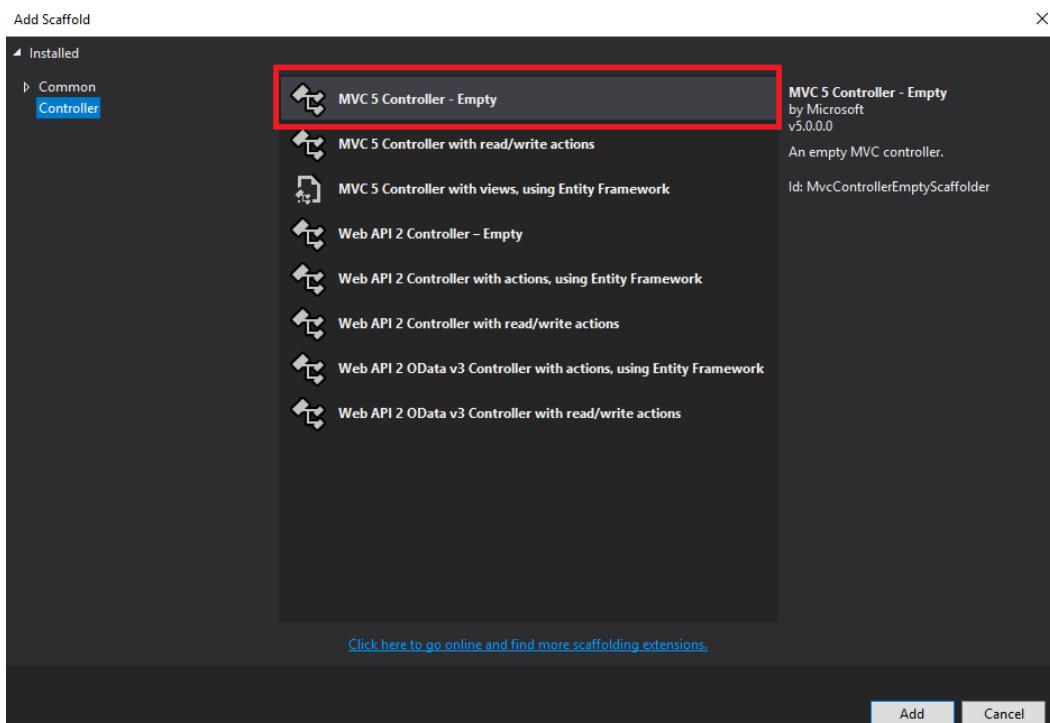


Figure 18: Selecionando template MVC 5 Controller - Empty.

Por fim, a Figura 19 é apresentada. Esta tela solicita o nome do controller, digite “CategoriasDeProduto” e mantenha o sufixo Controller, pois faz parte da convenção do ASP.NET MVC. Vale ressaltar que no caso dos Controllers há um diretório específico sugerido para a sua criação, conforme já aprendemos anteriormente, e também temos que respeitar a convenção de que todo Controller deve possuir o sufixo Controller em seu nome, além de ser uma boa prática utilizar o nome no plural, caso seja possível, como por exemplo: PessoasController, CarrosController, ContratosController etc.

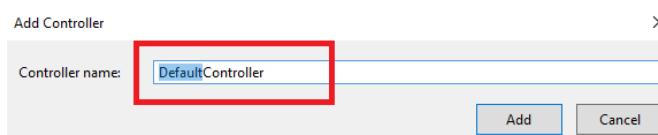


Figure 19: Atribuindo nome do Controller.

Após a conclusão de todas as etapas de criação da Controller, a CategoriasDeProdutoController deve ser exibida para você, conforme apresenta a Figura 20. Além do sufixo Controller no nome, repare que

CategoriasDeProdutoController herda da classe Controller, é necessária essa herança por características do framework. Repare na Action Index, as actions devem ser public, para que possam ser invocadas naturalmente pelo framework ASP.NET MVC, e não podem ser static, pois elas pertencerão a cada controlador instanciado.

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5  using System.Web.Mvc;
6
7  namespace MinhaPrimeiraAplicacao.Web.Controllers
8  {
9      public class CategoriasDeProdutoController : Controller
10     {
11         // GET: CategoriasDeProduto
12         public ActionResult Index()
13         {
14             return View();
15         }
16     }
17 }
```

Figure 20: Controller CategoriasDeProduto.

## 4.2 Action Results

O retorno de uma Action é representada pela classe abstrata ActionResult. Há diversas classes que estendem a ActionResult, e que representam um nível maior de especialização. A Figura 20 demostra que o retorno da Action Index do CategoriasDeProdutoController é uma View, veja o código: return View();.

Como há diferentes tipos de objetos ActionResult, deixaremos para explicar detalhadamente cada tipo conforme formos utilizar, porém a Tabela 1 informa os principais tipos de retorno, juntamente com uma breve descrição e um exemplo.

Table 1: Tabela com os principais tipos de retorno da classe ActionResult. Fonte: [12].

Classe derivada de ActionResult	Descrição	Exemplo
ViewResult	Retorna uma View	return View();
PartialViewResult	Retorna especificamente uma Partial View	return PartialView();
RedirectResult	Redireciona para uma URL específica	return Redirect("URL");
RedirectToRouteResult	Redireciona para outra Action	return RedirectToAcion("Action", "Controller");
ContentResult	Retorna texto puro (string)	return Content("Texto", "texto puro");
JsonResult	Retorna um JSON	return Json(objeto);
EmptyResult	Retorna uma resposta vazia	return EmptyResult();
FileResult	Retorna um arquivo	return File("caminho do arquivo", "application/pdf");

HttpStatusCodeResult	Retorno HTTP escolhido pelo desenvolvedor	X
----------------------	---	---

### 4.3 Views (Visão)

A View é a apresentação da solicitação para o usuário, será apresentada quando o método Action do Controller for chamado (Figura 20). Antes de criarmos uma nova View, precisamos saber que há uma convenção de configuração do framework ASP.NET MVC para Actions. Assim, a View deverá ficar em uma pasta com o nome do Controller, e a View deverá ter o nome da Action.

O Visual Studio automatiza o trabalho de criação da View, para isso, basta clicar com o botão direito do mouse sobre o código View() e navegar até a opção “Add View”. Há também a opção para navegar até uma View por meio da opção “Go To View”, caso a View já existe. A Figura 21 ilustra este processo.

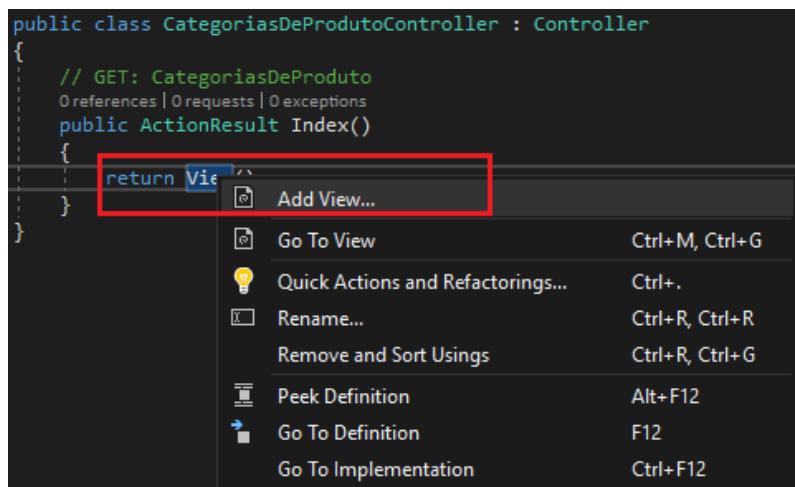


Figure 21: Adicionando uma nova View.

Logo em seguida, abrirá uma janela em que iremos informar mais detalhes sobre a View que será criada. A Figura 22 representa este processo, veja que no campo “View Name” já veio preenchido com o nome Index, mesmo nome da nossa Action criada em CategoriasDeProdutoController.

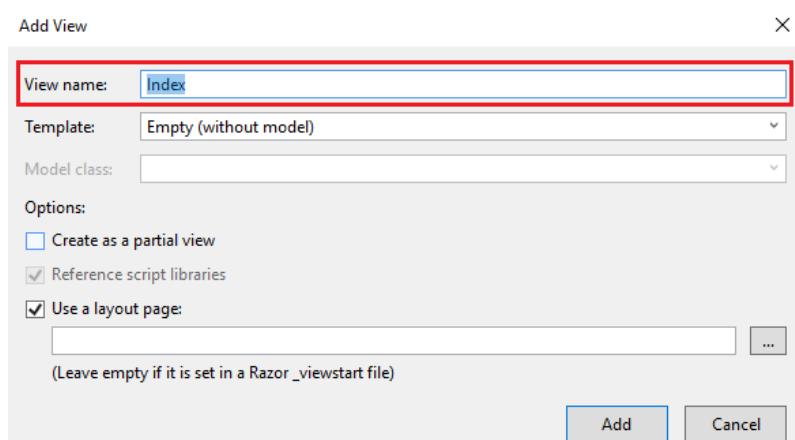


Figure 22: Detalhes para criação da View.

Nesta janela da Figura 22, há várias opções de configuração que iremos detalhar. Logo após o campo “View Name”, há a opção de seleção do template, nesta opção você automatiza a criação de uma View para as seguintes ações: Create, Delete, Details, Edit, List, Empty e Empty (without model). Repare, que logo abaixo existe o campo “Model class”, quando você seleciona algum template diferente da opção Empty (without model), este campo fica habilitado. No campo “Model class”, você seleciona qual será a Model utilizada para criação automática do template escolhido.

Por último, temos as duas opções de checagem que precisam ser apresentadas. A primeira é do campo “Create as partial view”, responsável por definir que criaremos uma Partial View. A Partial View ajuda na construção de controles visuais, é representada por funções genéricas que possam ser consumidas por vários lugares da sua aplicação (reutilização), além de poder ser utilizada para a implementação de páginas com atualizações parciais. Já, a segunda opção é o campo “Use a layout page:” com a seleção de caminho do arquivo, esta opção implementa a vinculação da View com uma página mestra. Desse modo, você criaria o conteúdo do seu site e deixaria na página mestra o cabeçalho, menu e rodapé, para quem fossem utilizados em todas Views. Repare que ao checar a opção do campo “Create as partial view”, o campo “Use a layout page” fica desabilitado, não há possibilidade de criar uma Partial View vinculado a um layout page.

Após a criação do arquivo, o Visual Studio abrirá a nossa View para edição, conforme ilustra a Figura 23.

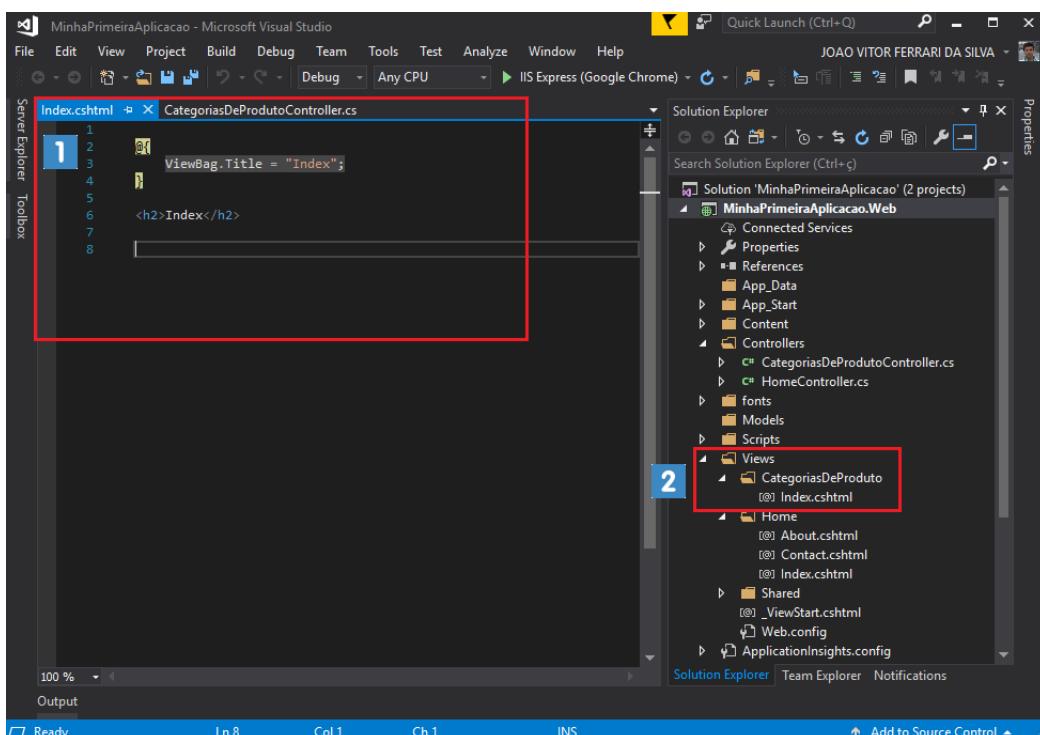


Figure 23: Código da View criada para retorno da Action Index do Controller CategoriasDeProdutoController.

Na View, você pode modificar o HTML gerado pelo Visual Studio (1), além de

utilizar todas as funcionalidades que o Razor nos proporciona (iremos detalhar esta tecnologia na Subseção 5.3). Repare pela Figura 23, que a convenção da View deve ser seguida (2), logo depois da pasta Views, temos outra pasta com o nome do Controller (CategoriasDeProdutoController) e dentro dele temos nossa View Index.

#### 4.4 Action Parameters

Os métodos das Actions serão organizados em solicitações de GET e de POST, na qual podemos representar pelos atributos [HttpGet] e [HttpPost], respectivamente. Action Parameters são maneiras que podemos utilizar para passagem de parâmetros pelas nossas Actions.

Podemos utilizar Query String para acessar os parâmetros informados na Action, conforme apresenta a Figura 24. Para realizar o cenário ilustrado pela imagem, basta você acessar o Controller CategoriasDeProdutoController e criar uma Action Teste, neste método adicione o parâmetro id, crie a View e depois acesse sua Action pela seguinte rota: <http://localhost:xxxx/CategoriasDeProduto/Teste?id=22>.

```

12
13
14 [HttpGet]
15 0 references | 0 requests | 0 exceptions
16 public ActionResult Teste(int id)
17 {
18     ViewBag.Message = id;
19     return View();
20 }

```

Figure 24: Passagem de parâmetro por Query String.

Outra maneira para trabalharmos com parâmetros é por meio do Model Class Object, na qual você pode criar uma classe responsável para informar seus valores, representada pela Figura 25, e a conversão de valores é feita de maneira automática, conforme ilustra a Figura 26. Para realizar o cenário ilustrado pela imagem, basta você acessar o Controller CategoriasDeProdutoController e alterar sua Action Teste, neste método altere o parâmetro id pela classe de Filtro, e depois acesse sua Action pela seguinte rota: <http://localhost:xxxx/CategoriasDeProduto/Teste?id=22>.

```

1 reference
public class Filtro
{
    1 reference | 0 exceptions
    public int Id { get; set; }
    0 references | 0 exceptions
    public string Nome { get; set; }
}

```

Figure 25: Classe de filtro.

```

21
22 [HttpGet]
23 0 references | 2 requests | 0 exceptions
24 public ActionResult Teste(Filtro filtro)
25 {
26     ViewBag.Message = filtro.Id;
27     return View();
28 }

```

Figure 26: Passagem de parâmetro por Model Class Object.

Podemos utilizar FormCollection para receber os parâmetros informados em um formulário, conforme exemplifica a Figura 27. Já na Action, iremos acessar os parâmetros informados no formulário da View, ilustrada pela Figura 28. Para realizar o cenário ilustrado pela imagem, basta você acessar o Controller CategoriasDeProdutoController e adicionar uma Action Index com atributo [HttpPost], neste método adicione o parâmetro FormCollection, execute sua aplicação e depois acesse sua Action pela seguinte rota: <http://localhost:xxxx/CategoriasDeProduto/Index>, preencha o formulário e envie sua requisição por meio do botão Salvar.

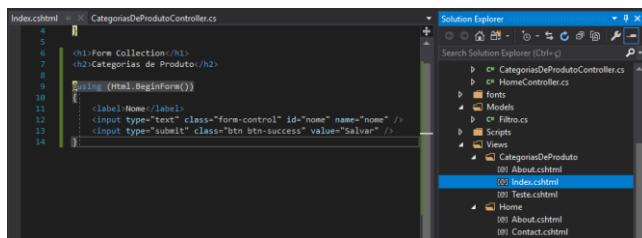


Figure 27: Acréscimo de um formulário na View Index do CategoriasDeProduto.

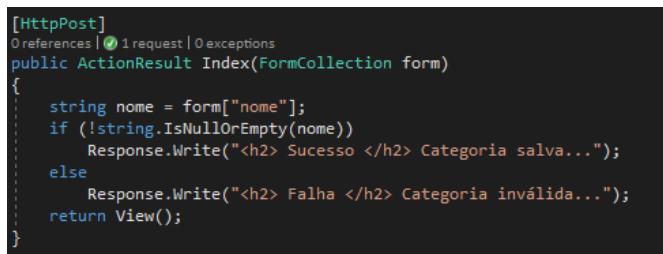


Figure 28: POST para receber os valores informados na View.

Recomenda-se que utilize o tipo formal para passagem de parâmetros, na qual os campos são informados como parâmetros no método da Action, isto quando for poucos campos, ou utilize o tipo Model Class Object, que facilita o trabalho de passagem de parâmetros e facilita a manutenção do seu código.

## 4.5 Routes (Rotas)

A navegação baseada em Rotas é uma dos aspectos fundamentais para entendermos o ASP.NET MVC. A URL em uma aplicação ASP.NET MVC é formada por segmentos, que compõem as rotas. Cada segmento possui um significado durante a requisição solicitada pela usuário. Na URL <http://localhost:xxxx/CategoriasDeProduto/Teste?id=22> requisitada via GET, são encontrados os segmentos a seguir:

1. http: representa o protocolo utilizado pela requisição;
2. localhost:xxxx: servidor e porta de comunicação;
3. CategoriasDeProduto: representa o Controller utilizado;
4. Teste: refere-se a Action utilizada;
5. ?id=22: simboliza a Query String.

As configurações das rotas são centralizadas na classe RouteConfig, localizada em: MinhaPrimeiraAplicacao.Web > App\_Start > RouteConfig.cs, é recomendado que se mantenha o padrão definido pelo ASP.NET MVC, mas há casos que precisaremos alterar as rotas para atender outras necessidades, como por exemplo, a indexação de uma página pelos motores de busca de sites de pesquisa (Google, Bing, Baidu etc.), por meio de URL amigável, mas esta parte será detalhada em outro capítulo. O código desta classe pode ser visto pela Figura 29.

```
public class RouteConfig
{
    1 reference | 0 exceptions
    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

        routes.MapRoute(
            name: "Default",
            url: "{controller}/{action}/{id}",
            defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
        );
    }
}
```

Figure 29: RouteConfig, classe responsável pela configuração de Rotas.

Um exemplo das configurações da classe RouteConfig, é a requisição das seguintes URLs <http://localhost:xxxx/CategoriasDeProduto/Index> e <http://localhost:xxxx/CategoriasDeProduto/>, perceba que o redirecionamento é realizada para mesma página, conforme demonstra a Figura 30.

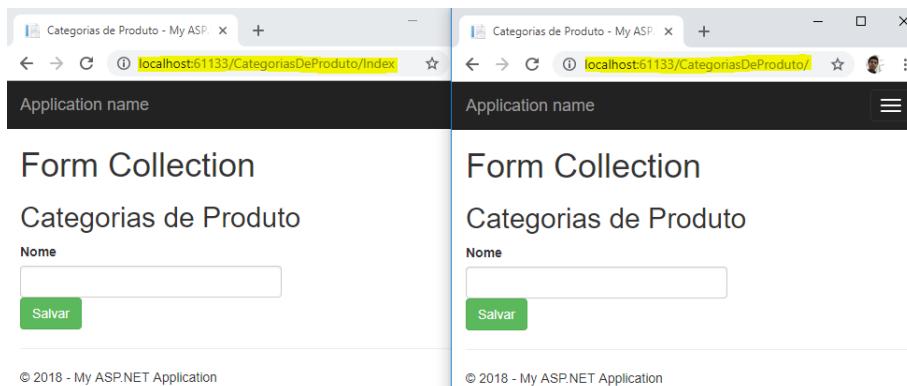


Figure 30: Navegação baseada em Rotas adicionado parâmetros.

Outro exemplo de requisições, é pelo acesso das URLs <http://localhost:xxxx/CategoriasDeProduto/Teste?id=22> e <http://localhost:xxxx/CategoriasDeProduto/Teste/22>, o acesso também é redirecionada para a mesma página, conforme apresenta a Figura 31. Isto acontece por causa do mecanismo de roteamento de requisições do ASP.NET MVC, cada chamada está fazendo referência ao mesmo recurso.

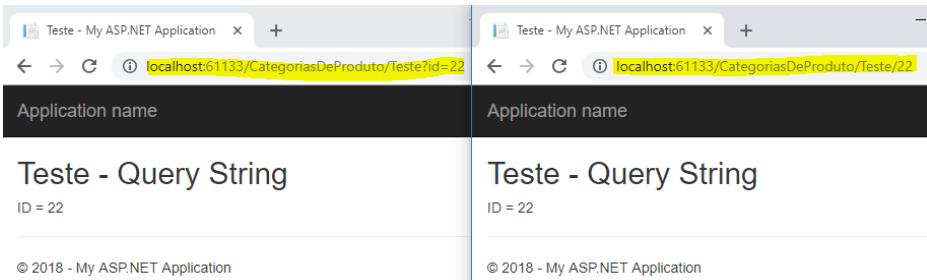


Figure 31: Navegação baseada em Rotas passando parâmetros.

Note pela Figura 29, que o redirecionamento das URLs citadas para mesma página acontece por meio da implementação `routes.MapRoute()` para a rota Default (name: "Default"), na qual temos a seguinte atribuição do valor Home para o atributo Controller (`controller = "Home"`), do valor Index para o atributo Action (`action = "Index"`), e, por fim, do valor `UrlParameter.Optional` para o atributo id (`id = UrlParameter.Optional`).

Estas configurações da classe `RouteConfig`, que possibilitam no exemplo da Figura 30, o uso da Action Index por valor default, no caso de omissão do parâmetro, e como não for enviado um terceiro argumento, que seria o id, nada é utilizado como padrão, pois ele é configurado como opcional. Já no exemplo da Figura 31, verifica-se a interpretação do parâmetro id por Query String.

## 5. Fundamentos do ASP.NET MVC – Parte 2

Neste capítulo, aprenderemos outros recursos do ASP.NET MVC. Aprenderemos como criar uma Model, como trabalhar com os dados na View e também como criar uma Partial View, entre outras funcionalidades.

### 5.1 View Models (Modelos)

O modelo representa dados específicos do domínio e lógica de negócios na arquitetura MVC. São objetos que usamos para enviar informações ao banco de dados, executar o cálculos de negócio e até renderizar uma View. Em outras palavras, esses objetos representam o domínio no qual o aplicativo se concentra e as Models são os objetos que você deseja exibir, salvar, criar, atualizar e excluir [8].

Mas para que possamos entender o modo de funcionamento do framework junto aos Models, apresentaremos algumas abordagens possíveis de forma atrelada ao nosso projeto de exemplo, o “MinhaPrimeiraAplicacao”.

Primeiramente, criaremos nossa Model para representar a Categoria de Produto, na janela “Solution Explorer”, navegue até a pasta Models do nosso projeto (localizada em: MinhaPrimeiraAplicacao.Web > Models), clique com o botão direito e navegue até a opção Add > New Item, será apresentada a tela da Figura 32.

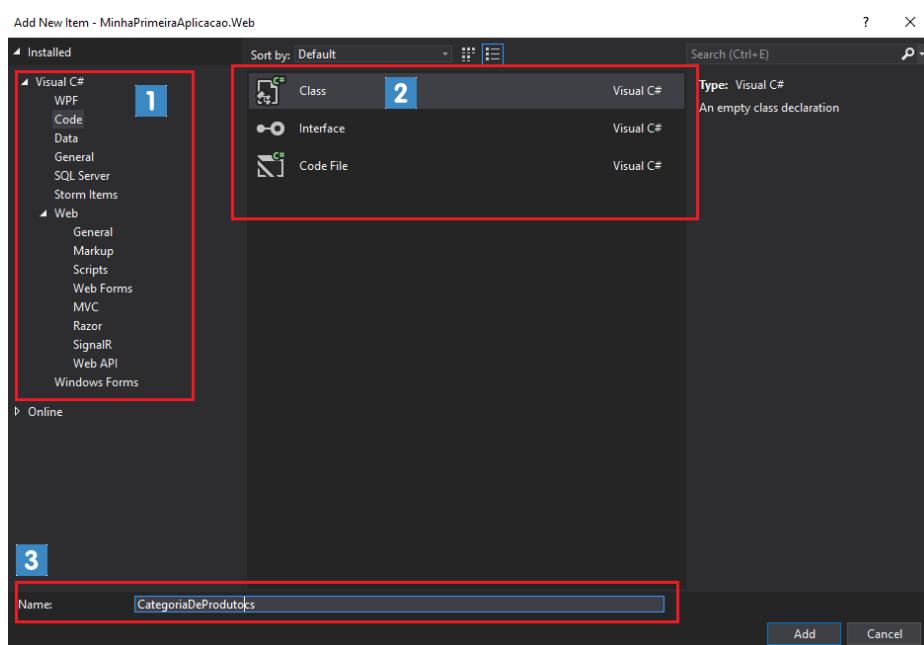


Figure 32: Criando a Model CategoriaDeProduto.

Pela Figura 32, há vários recursos que podem ser adicionados ao projeto. No nosso caso, iremos adicionar uma nova classe, para isso iremos navegar até o opção “Visual C# > Code” (1), esta ação irá filtrar os recursos disponíveis para nossa seleção, escolha opção “Class” (2), e por último, daremos o nome “CategoriaDeProduto.cs” (3) para nossa classe. Pronto, nosso modelo está criado.

Vamos adicionar as seguintes propriedades em nossa classe CategoriaDeProduto: (i) Id, (ii) Nome e (iii) Descrição. Conforme código a seguir:

```
@style=sharpc public class CategoriaDeProduto { public int Id { get; set; } public string Nome { get; set; } public string Descricao { get; set; }}
```

Faremos a seguinte alteração em nossa View Index da CategoriaDeProduto (localizada em MinhaPrimeiraAplicacao.Web > Views > CategoriasDeProduto > Index), conforme apresenta a Figura 33. Adicionaremos os inputs do HTML para informar as propriedades da classe CategoriasDeProduto.

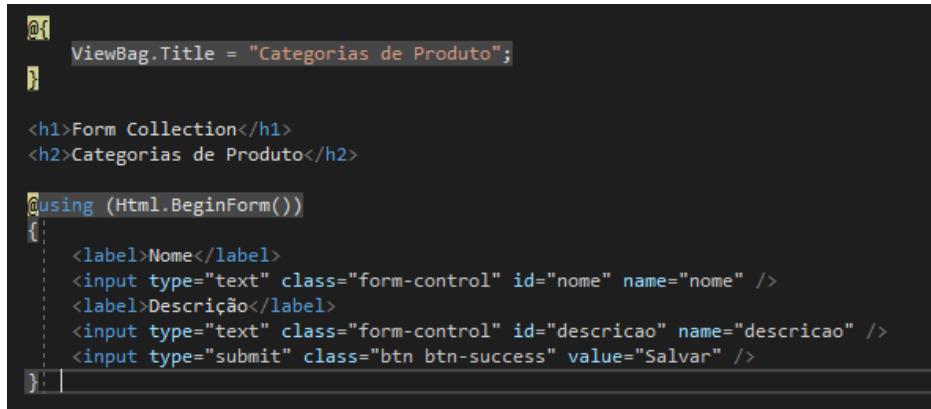


Figure 33: Formulário para enviar as informações da Model CategoriaDeProduto.

Precisaremos criar nosso método POST para receber as informações do formulário criado, basta acessar a CategoriasDeProdutoController (MinhaPrimeiraAplicacao.Web > Controllers > CategoriasDeProdutoController) e criar uma Action Index para receber os dados informados para a Model CategoriaDeProduto, conforme exemplifica a Figura 34.

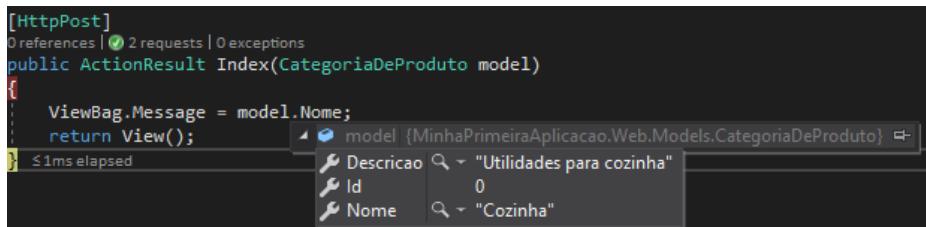


Figure 34: Recebendo os dados informados na View para Model CategoriaDeProduto.

Repare pela Figura 34 que as informações passadas por meio do formulário implementado na View são convertidas automaticamente em CategoriaDeProduto, para que isto aconteça, basta que o nome dos inputs tenham o mesmo nome das propriedades da Classe CategoriaDeProduto.

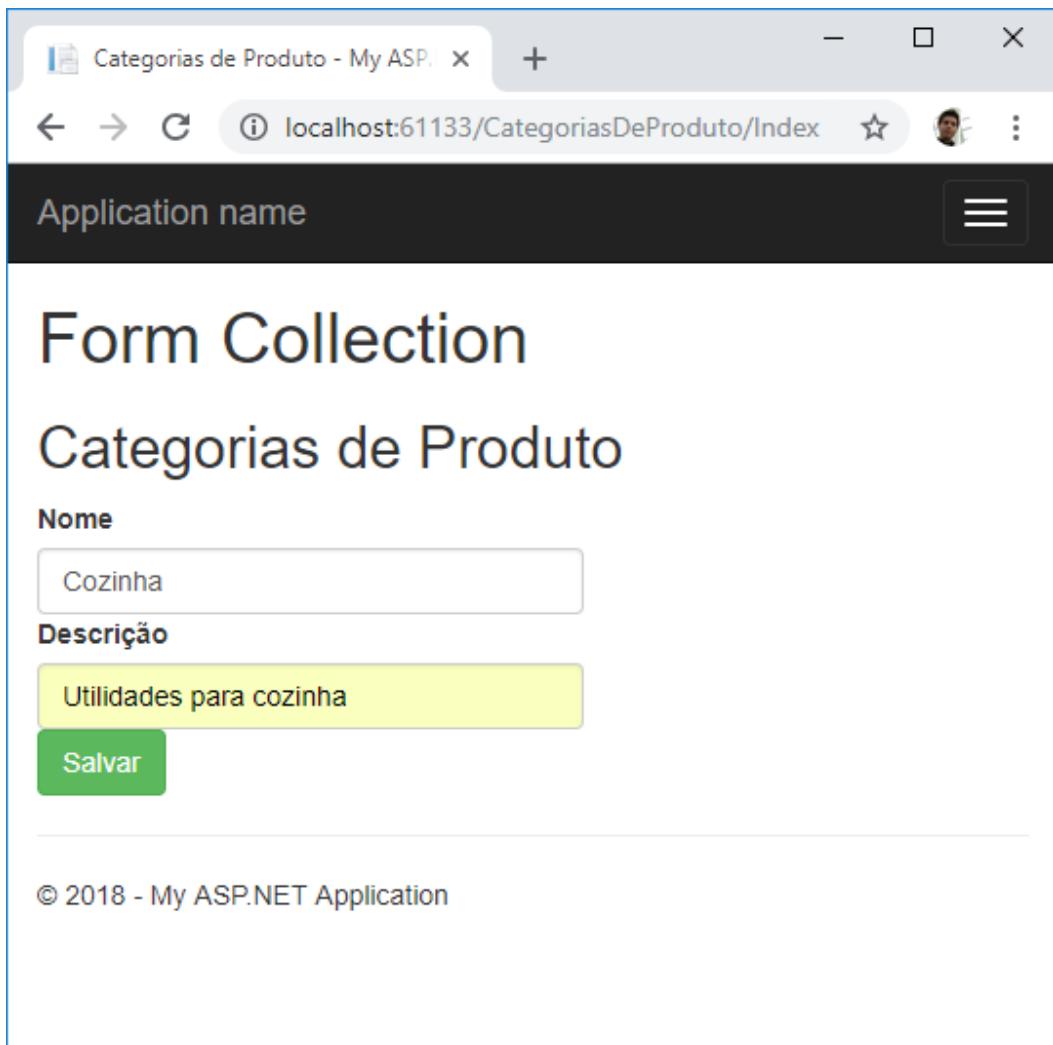


Figure 35: Apresentação da View para cadastro de CategoriaDeProduto.

Após nossa implementação, testaremos nossa aplicação executando nosso projeto. Se todo o processo for realizado com sucesso, ao acessar a URL: <http://localhost:xxxx/CategoriasDeProduto/Index> a nossa View deve ser apresentada, conforme ilustra a Figura 35.

## 5.2 Passing Data to Views

Até agora aprendemos como receber dados da View para utilizarmos em nosso Controller. Dessa forma, vamos descobrir como passamos nossos dados da Model para View por meio do Controller. Há diferentes maneiras de conseguir isso e iremos detalhar a seguir.

Primeiramente vamos conhecer ViewBag, já vimos este cara no material, mas não tivemos a oportunidade de detalhar este recurso. Ele é utilizadas para persistir dados entre a Controller e a View correspondente, possui uma duração de “tempo de vida” exclusivo entre o envio através da Controller e a exibição na View, depois disso torna-se nula novamente. Caso utilize algum Action Result de Redirect, a ViewBag também se tornará nula [19].

As características da ViewData se assemelham aos da ViewBag, na qual se

diferem da seguinte forma: enquanto a ViewData é um dicionário de objetos derivado de ViewDataDictionary e é acessível utilizando strings como chaves, além de requerer conversão quando associada a tipos complexos. A ViewBag é uma propriedade dinâmica baseada na funcionalidade "dynamic" e não há necessidade de conversão para tipos complexos [19].

A figura 36 demonstra o código de como passar dados por meio de ViewBag e de ViewData. Veja que na ViewBag você cria a propriedade Categoria dinamicamente, já na ViewData precisa criar uma chave e atribuir o valor desejado.

```
public ActionResult Create()
{
    var categoria = new CategoriaDeProduto
    {
        Id = 22,
        Nome = "Eletroportáteis",
        Descricao = "Renove seus eletros portáteis",
    };

    ViewBag.Categoria = categoria;
    ViewData["Categoria"] = categoria;

    return View();
}
```

Figure 36: Passando informações por meio de ViewBag e de ViewData.

Pela figura 37 visualizamos como obter os dados passados pela Action. Perceba que pelo recurso ViewData foi necessário realizar a conversão do objeto, já por ViewBag não há esta necessidade de conversão. Recomenda-se o uso por meio de ViewBag.

```
@using MinhaPrimeiraAplicacao.Web.Models

@{
    ViewBag.Title = "Create";
    var ViewDataVariavel = ViewData["Categoria"] as CategoriaDeProduto;
    var ViewBagVariavel = ViewBag.Categoria;
}
```

Figure 37: Obtendo informações na View por meio de ViewBag e de ViewData.

O TempData é outra forma de passarmos dados para nossa View. Ela é semelhante a uma sessão de servidor, porém possui uma duração menor. É utilizado para compartilhar informações entre Controllers, pois possui um tempo de vida maior que o ViewBag e ViewData, o TempData perdura desde sua criação até que seja chamado, ou seja, quando houver um request da informação do TempData, ele se tornará nulo novamente [6].

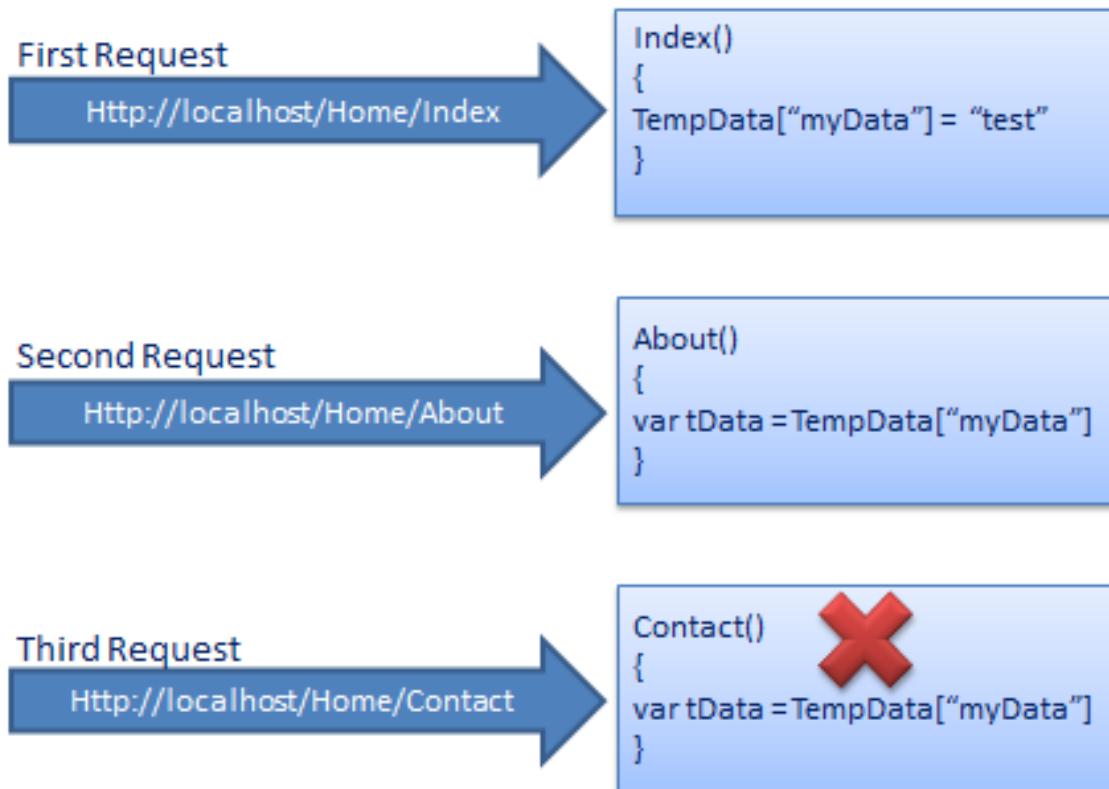


Figure 38: Ciclo de vida do TempData. Fonte: [27].

Pela figura 38 percebemos que ao adicionarmos dados de teste em TempData na primeira solicitação e na segunda solicitação subsequente acessarmos os dados de teste de TempData que armazenamos na primeira solicitação, estes dados são limpos. Dessa forma, não conseguimos obter os mesmos dados na terceira solicitação porque o TempData foi limpo após a segunda solicitação. O TempData em Views é recomendado no caso de um dado necessitar ser redirecionado entre Actions e posteriormente ser exibido numa View.

Por fim, outra maneira de passar os dados da Controller para View, é por meio de um objeto da classe de modelo para a View, conforme demonstra a figura 39, código `return View(categoria);`:

```
public ActionResult Create()
{
    var categoria = new CategoriaDeProduto
    {
        Id = 22,
        Nome = "Eletroportáteis",
        Descricao = "Renove seus eletros portáteis",
    };

    return View(categoria);
}
```

Figure 39: Passando o objeto da Model CategoriaDeProduto para View.

Na View, precisamos definir a Model que virá da Controller pela Action, conforme ilustra a figura 40. Veja que definimos a Model pelo seguinte código:

@model CategoriaDeProduto, não esqueça de referenciar a classe por meio do using. Depois de definida, podemos acessar nosso modelo por meio do: @Model, e consequentemente as propriedades da classe CategoriaDeProduto.

```
 @using MinhaPrimeiraAplicacao.Web.Models;
 @model CategoriaDeProduto

 @{
     ViewBag.Title = "Create";
 }

 <h2>Create</h2>

 <h3>Passing Data From Controller To View using Model Class Object</h3>

 <h3>Id: @Model.Id</h3>
 <h3>Nome: @Model.Nome</h3>
 <h3>Descrição: @Model.Descricao</h3>
```

Figure 40: Consumindo Model CategoriaDeProduto pela View.

### 5.3 Razor Syntax

O ASP.NET Razor é uma view engine, é uma sintaxe de script que simplifica a maneira como nós codificamos as aplicações ASP.NET. É uma ferramenta para geração de visões que possibilita a inserção de lógica da aplicação nas Views. Sua sintaxe é simplificada e tem como base a linguagem de programação C#.

Razor não é uma linguagem, mas sim um recurso do ASP.NET. Ele nos traz alguns benefícios [13], como:

1. A sintaxe Razor é limpa e concisa, o que requer um número mínimo de digitação;
2. Razor é fácil de aprender, em parte porque ele é baseado em linguagens existentes, como C#;
3. Visual Studio inclui o IntelliSense e colorização de código para a sintaxe Razor;
4. Views Razor podem ser testadas de forma unitária sem exigir que você execute o aplicativo ou abra um servidor web;
5. Sintaxe do modelo de @ para especificar o tipo que está sendo passado para a exibição.

Pela figura 41, vimos que a inserção de código deve ser precedida do caractere @ (1), e blocos de código devem estar delimitados por { (2). Dentro de um bloco, cada instrução deve ser finalizada com ; (ponto e vírgula), e é possível fazer uso de variáveis para armazenamento de valores. Veja que o @Html.EditorFor renderiza a tag input quando iniciamos nossa aplicação, outros recursos serão detalhados nos módulos a seguir, porém caso queira conhecer um pouco mais, recomendo a leitura da seguinte URL: <https://msdn.microsoft.com/pt-br/magazine/mt845651.aspx>.

```

1 1 @using MinhaPrimeiraAplicacao.Web.Models
2 2 @model CategoriaDeProduto
3
4 3 @{
5 2   ViewBag.Title = "Create";
6 }
7
8 <h2>Create</h2>
9
10 @using (Html.BeginForm())
11 {
12   <div class="form-horizontal">
13     <h4>CategoriaDeProduto</h4>
14     <hr />
15     @Html.ValidationSummary(true, "", new { @class = "text-danger" })
16     <div class="form-group">
17       @Html.LabelFor(model => model.Nome, htmlAttributes: new { @class = "control-label col-md-2" })
18       <div class="col-md-10">
19         @Html.EditorFor(model => model.Nome, new { htmlAttributes = new { @class = "form-control" } })
20         @Html.ValidationMessageFor(model => model.Nome, "", new { @class = "text-danger" })
21       </div>
22     </div>
23
24     <div class="form-group">
25       @Html.LabelFor(model => model.Descricao, htmlAttributes: new { @class = "control-label col-md-2" })
26       <div class="col-md-10">
27         @Html.EditorFor(model => model.Descricao, new { htmlAttributes = new { @class = "form-control" } })
28         @Html.ValidationMessageFor(model => model.Descricao, "", new { @class = "text-danger" })
29       </div>
30     </div>
31
32     <div class="form-group">
33       <div class="col-md-offset-2 col-md-10">
34         <input type="submit" value="Create" class="btn btn-default" />
35       </div>
36     </div>
37   </div>
38
39   <div>
40     @Html.ActionLink("Back to List", "Index")
41   </div>
42 }

```

Figure 41: Sintaxe Razor para codificação da View.

Com Razor o código da View fica mais fácil de digitar, mais simples e legível. Razor possui Helpers que são ferramentas que facilitam o desenvolvimento de alguns recursos, como Chart, que é responsável por processar um gráfico. Outros Helpers serão detalhados nos próximos Módulos.

## 5.4 Partial Views

A Partial View é uma View reutilizável, utilizada para criar componentes reaproveitáveis, evitando assim código duplicado e adquirindo o reaproveitamento e encapsulamento do código. O recurso de Partial View também é útil para separação de partes complexas ou quando uma página possui muito código (HTML e/ou Razor), facilitando a leitura e entendimento, o que torna mais fácil a manutenção do sistema, quando encontramos código menor e separado com uma única responsabilidade.

As Partial Views não possuem layouts, como as Views, e podem ser adicionadas dentro de diversas Views, como se fosse um componente ou controle.

Para criar uma Partial View precisaremos seguir o mesmo modo de cadastro de View, porém não podemos esquecer de checar a opção “Create as a partial View”, conforme é mostrado pela figura 42. Há diversas boas práticas e padrões de nomenclatura para codificação, no nosso caso utilizaremos um underscore (\_) como prefixo do nome, por exemplo “\_Menu”.

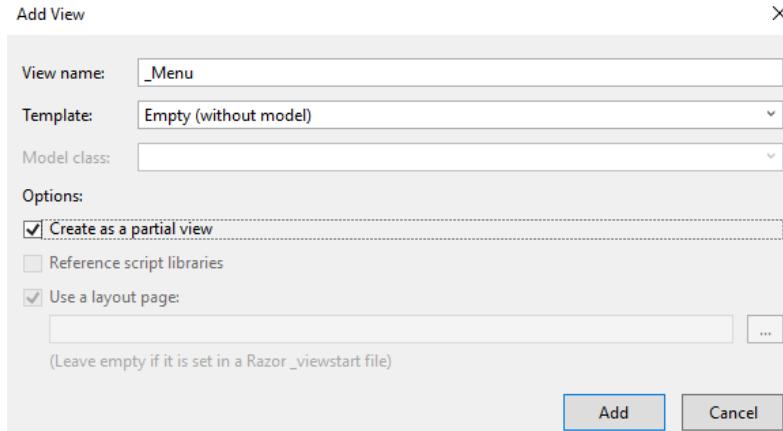


Figure 42: Criando uma Partial View para listar as opções de menu.

A implementação da Partial View `_Menu` é apresentada na figura 43. Desenvolvemos um menu simples utilizando HTML e a sintaxe Razor para construção dos links.

```

_Layout.cshtml
1 <div>
2   <p>Menu</p>
3   <ul>
4     <li>@Html.ActionLink("Home", "Index", "Home")</li>
5     <li>@Html.ActionLink("About", "About", "Home")</li>
6     <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
7   </ul>
8 </div>

```

Figure 43: Menu codificado na Partial View.

Com a implementação da Partial View concluída, precisamos fazer uso dela. Por exemplo, chamaremos ela em nossa View Index da CategoriaDeProduto, por meio do código: `@Html.Partial("_Menu")`, conforme demonstra a figura 44.

```

_Index.cshtml
1
2 @{
3   ViewBag.Title = "View";
4 }
5
6 <h2>View</h2>
7
8 <div>
9   @Html.Partial("_Menu")
10 </div>

```

Figure 44: Chamar Partial View.

## 6. Considerações finais

Neste módulo entendemos o papel do padrão MVC no framework ASP.NET MVC, em que o controller está no comando e tem a responsabilidade de decidir quais models usar e quais views transmitir de volta para o usuário. O controller está a cargo da coordenação e é o primeiro a ser executado quando a requisição web chega à aplicação. O controller é responsável por decidir qual resposta é adequada para a requisição do usuário [17].

Já a view é responsável apenas por apresentar a interface do usuário. Ao separar a lógica de domínio e desacoplar da view, o acesso aos dados e a outras chamadas a interface do usuário podem permanecer igual, mesmo quando a lógica e o acesso aos dados mudam dentro da aplicação. Por fim, o controller possui uma relação direta com a view e a model, mas a model não precisa saber nada a respeito do controller ou da view [17].

Recomendo a leitura do seguinte tutorial da Microsoft que detalhada cada recurso do ASP.NET MVC 5, acesse a URL: <https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/>.

Este módulo foi um pouco mais extenso e detalhado pois precisamos alinhar o conhecimento de todos os alunos. Há alunos que só tiveram o primeiro contato com o Visual Studio e o ASP.NET MVC agora, já há outros que conhecem toda ferramenta, mas, às vezes, não conhecia cada detalhe dos recursos que foram apresentados. Os próximos módulos serão focados mais no código que iremos desenvolver, porém todos os conceitos serão totalmente explicados e exemplificados.

# **UNIDADE 2**

## **Trabalhando com dados**

Neste módulo iniciaremos um projeto do zero, conheceremos novos recursos que ainda não foram abordados, e detalharemos os que já foram apresentados. Trabalharemos com os dados em nossa aplicação por meio do SQL Server e aprenderemos a trabalhar com GitHub no Visual Studio.

### **1. Conhecendo o contexto do nosso projeto**

Antes de iniciarmos nossa aplicação, iremos contextualizar um cenário para o desenvolvimento de um simples web site para disponibilizar artigos. Dessa forma, suponhamos que você adora escrever artigos de diversas categorias diferentes, porém está desmotivado, pois seus artigos não estão alcançando a visibilidade esperada.

Nesse contexto, você teve a ideia de desenvolver um web site para disponibilizar seus artigos pela internet. Organizar os artigos que você escreve não é uma tarefa fácil, muito menos publicá-los de maneira rápida para que os usuários accessem seu conteúdo.

Pensando no contexto proposto, o projeto do web site para blog simples tem como objetivo mostrar os passos do desenvolvimento nas etapas do back-end e front-end, assim sendo, será utilizado o ASP.NET MVC no back-end e Bootstrap no front-end da aplicação. Após separadas as responsabilidades e definidas quais tecnologias serão utilizadas, iniciaremos nosso projeto e mostraremos a integração entre o desenvolvimento front-end e back-end.

Com o desenvolvimento do projeto no framework ASP.NET MVC fica bem mais simples separar as responsabilidades dentro da aplicação utilizando o padrão MVC. Desta forma, separaremos o trabalho do front-end na camada View, enquanto o back-end nas camadas Controller, Model e View, quando for necessário.

A figura 45 ilustra as tecnologias que serão utilizadas na aplicação. No back-end da aplicação será utilizado o ASP.NET MVC com a linguagem C#. Já para o acesso ao banco de dados SQL Server será utilizado o Entity Framework. No front-end empregaremos o Bootstrap<sup>7</sup> (estilização visual e responsividade) e a jQuery<sup>8</sup> (validação e máscaras). O uso da IDE Visual Studio será muito importante neste projeto, pois ele proporcionará suporte a todo nosso desenvolvimento.

---

<sup>7</sup> <https://getbootstrap.com/>

<sup>8</sup> <https://jquery.com/>



Figure 45: Tecnologias utilizadas na aplicação. Fonte: [5]

## 1.1 Requisitos para o projeto do Blog pessoal

Antes de iniciarmos o desenvolvimento de um software, sempre há necessidade de estudarmos alguma documentação e especificações do que deve ser implementado, para que se possamos desenvolver o produto desejado. A documentação deve conter os requisitos do software a ser desenvolvido, informações sobre funcionalidades do sistema, regras de negócio do sistema, requisitos funcionais e não funcionais, além de fluxos de trabalhos, entre outras informações.

Desse modo, iremos listar alguns requisitos funcionais que devem estar presentes no blog que criaremos:

1. O autor deve ser autenticado para postar qualquer artigo e fazer as manutenções dos artigos;
2. Um artigo deve pertencer a uma categoria, desta forma, deve ter uma página para cadastro de novas categorias no blog;
3. O web site deve ter uma página para o usuário fazer login ou se cadastrar;
4. O web site deve ter uma página para manter os artigos do blog;
5. Um artigo deve apresentar o nome do autor, título, conteúdo e data de publicação;
6. O web site deve ter uma página principal em que serão listadas as postagens;
7. O web site deve ter uma página que exibirá o conteúdo de cada artigo;
8. O web site deve ter menu para acesso rápido das últimas postagens, categorias e filtros para facilitar a pesquisa de artigos.

Assim temos descritas de forma clara e objetiva as principais funcionalidades do nosso blog, além de também especificarmos algumas regras de negócio, como as informações necessárias para postar um artigo, e a obrigatoriedade de autenticação do autor para realizar o cadastro de um artigo.

Por fim, com base nos requisitos enumerados, iniciaremos o desenvolvimento da nossa aplicação. Passaremos por um ciclo de desenvolvimento, que vai desde as etapas de modelagem do domínio até a implementação da interface com o usuário, alinhado ao objetivo de atender as especificações solicitadas.

## 1.2 Modelagem e implementação do banco de dados

Após entendermos nossa necessidade, podemos modelar nosso banco de dados. Esta modelagem será a representação do domínio da nossa aplicação, no qual nos basearemos para implementação das classes de negócio no ASP.NET MVC.

Para realizar a modelagem em modo gráfico, podemos utilizar o próprio SQL Server Management Studio, conforme demonstra o Diagrama Entidade Relacionamento (DER) do nosso projeto de Blog Pessoal, representada pela figura 46.

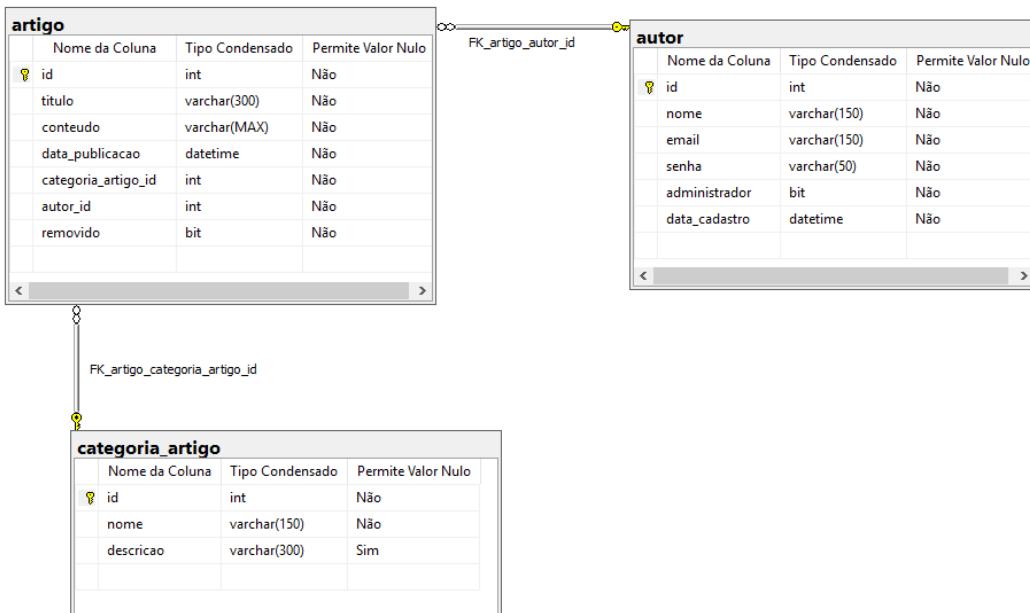


Figure 46: DER banco de dados para Blog Pessoal.

Há outras maneiras para realizar a modelagem, caso queira, podemos realizar o trabalho por meio de script. Dessa forma, precisamos criar um novo banco de dados pelo SQL Server Management Studio, chamaremos nosso banco de "BlogSimples" e em uma nova query, executaremos o script de criação das tabelas demonstrado pela listagem a seguir.

```
USE BlogSimples
CREATE TABLE dbo.[autor] ( id int IDENTITY NOT NULL, nome varchar(150) NOT NULL, email varchar(150) NOT NULL, senha varchar(50) NOT NULL, administrador bit DEFAULT(0) NOT NULL, data_cadastro datetime DEFAULT(GETDATE()) NOT NULL, CONSTRAINT PK_autor PRIMARY KEY (id), CONSTRAINT UQ_email_autor UNIQUE (email), ) GO
CREATE TABLE dbo.[categoria_artigo] ( id int IDENTITY NOT NULL, nome varchar(150) NOT NULL, descricao varchar(300) NULL, CONSTRAINT PK_categoria_artigo PRIMARY KEY (id), CONSTRAINT UQ_nome_categoria_artigo UNIQUE (nome), ) GO
CREATE TABLE dbo.[artigo] ( id int IDENTITY NOT NULL, titulo varchar(300) NOT NULL, conteudo varchar(MAX) NOT NULL, data_publicacao datetime NOT NULL, categoria_artigo_id int NOT NULL, autor_id int NOT NULL, removido bit DEFAULT(0) NOT NULL, CONSTRAINT PK_artigo PRIMARY KEY (id), CONSTRAINT FK_artigo_categoria_artigo_id FOREIGN KEY (categoria_artigo_id) REFERENCES dbo.[categoria_artigo] (id), CONSTRAINT FK_artigo_autor_id FOREIGN KEY (autor_id) REFERENCES dbo.[autor] (id) ) GO
```

## 2. Desenvolvendo a regra de negócio da aplicação (back-end)

Nesta seção, abordaremos a criação do nosso projeto ASP.NET MVC, com foco nas responsabilidades do back-end. Inciaremos nosso estudo sobre Entity Framework e aprenderemos a utilizar o GitHub como repositório do web site.

### 2.1 Criando o projeto do Blog pessoal

Iniciaremos pela criação do nosso projeto ASP.NET MVC com o Visual Studio 2017 por meio do menu: File > New > Project. Dentro da categoria Web, selecione o template ASP.NET Web Application (.NET Framework), atribua como nome do projeto “BlogPessoal.Web” e para o nome da solução “BlogPessoal”, este processo é ilustrado pela figura 47.

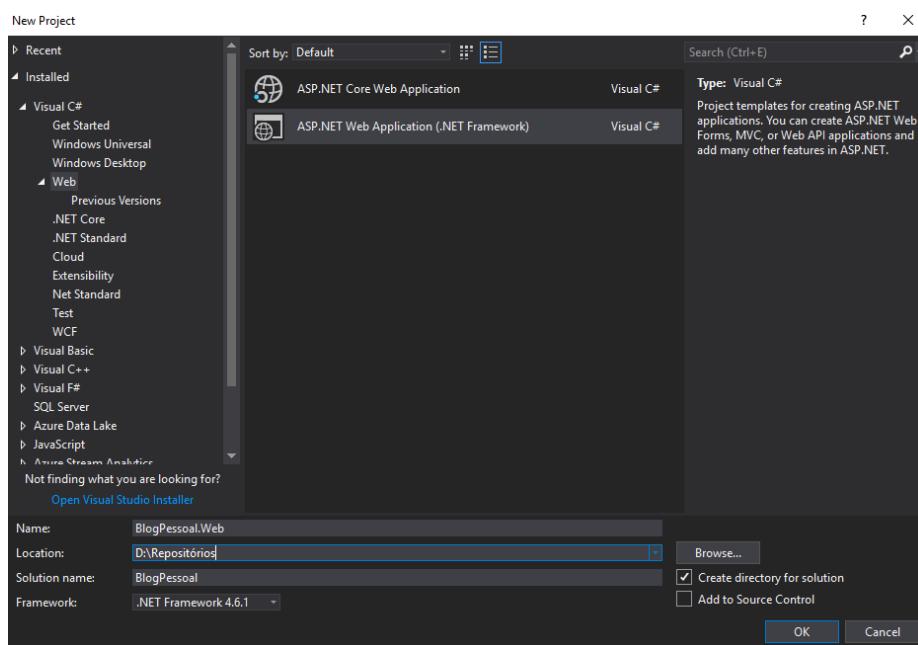


Figure 47: Selecionando o template do nosso Blog Pessoal.

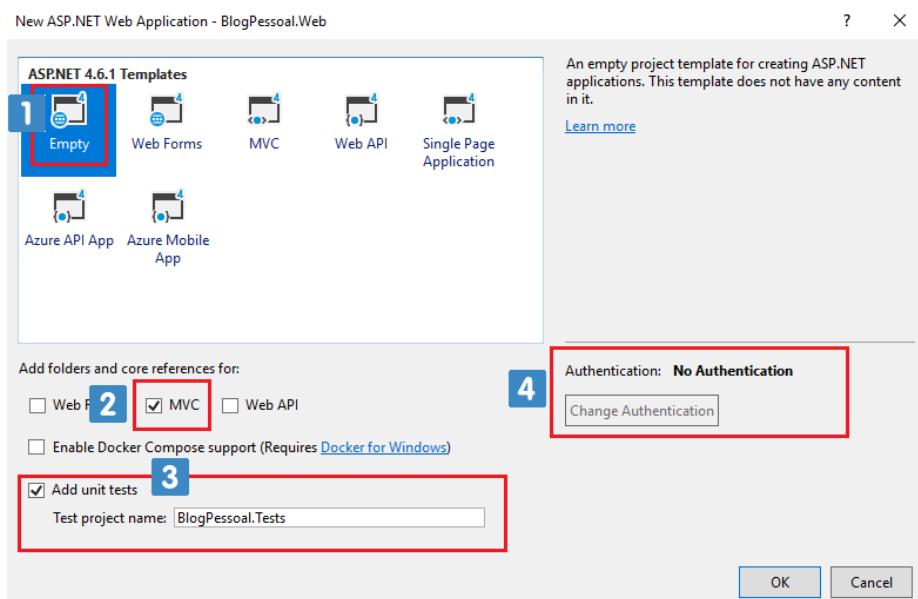


Figure 48: Configurando o template do nosso Blog Pessoal.

Na tela seguinte selecione o template Empty do ASP.NET (1), depois marque a opção MVC (2) e não esqueça de criar o projeto de teste com o nome “BlogPessoal.Test” (3), verifique se o propriedade Authentication está com a opção “No Authentication” (4), pois iremos criar nosso próprio sistema de autenticação. Por fim, clique em OK para que o Visual Studio crie o projeto. A figura 48 demonstra as configurações exemplificadas para criação do projeto.

## 2.2 GitHub

Iremos utilizar o GitHub como repositório para o nosso projeto. Caso você não tenha conta ainda, é interessante que seja criada uma conta. Para projetos de código aberto, não há custo nenhum, acesse: <https://github.com/>

Depois, criaremos um repositório remoto, que ficará disponível para todos da internet. Para isso, clique no botão “New Repository”. No Repository name, devemos preencher o nome do repositório remoto. No nosso caso, vamos preencher com “asp.net-mvc-blog-simples”. Deixe o repositório como Public, para que qualquer pessoa consiga ver o seu código. As demais opções podem ficar com os valores padrão. Finalmente, devemos clicar em Create repository. A figura 49 ilustra todo este processo.

Create a new repository

A repository contains all the files for your project, including the revision history.

---

Owner                  Repository name

 poferrari  

Great repository names are short and memorable. Need inspiration? How about [furry-bassoon](#).

Description (optional)

Blog simples desenvolvido em ASP.NET MVC com linguagem C#.

---

 Public  
Anyone can see this repository. You choose who can commit.

 Private  
You choose who can see and commit to this repository.

---

Initialize this repository with a README  
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore:  Add a license:  

---



Figure 49: Criando repositório no GitHub.

Pronto, seu repositório foi criado com sucesso. Nesse contexto, podemos trabalhar com o Git em linha de comando, porém para utilizar o modo gráfico precisaremos instalar a aplicação do GitHub para nosso ambiente Windows, faça o download pela URL: <https://gitforwindows.org/>.

O código fonte da aplicação que construiremos pode ser encontrado no GitHub, acesse: <https://github.com/poferrari/asp.net-mvc-blog-simples>. Utilize sempre que necessário para referência nos estudos e, caso queira, fique à vontade para contribuir com o código fonte, faça um fork e envie seu pull request.

### 2.3 Acessando o repositório GitHub pelo Visual Studio

Pelo Visual Studio, acesse a tela “Team Explorer - Connect”, selecione a opção GitHub. Use a opção para clonar o projeto remoto para sua máquina, escolhendo uma pasta. Após, adicione o projeto já criado.

Por meio do prompt de comandos (Git Bash), executaremos a seguinte sequência de comandos:

1. git status: para verificarmos as alterações realizadas no repositório;
2. git add .: irá realizar as alterações notificadas no repositório;
3. git commit -m "comentário qualquer": confirma alterações realizadas;
4. git push: envia as alterações realizadas para o repositório remoto;
5. git pull: atualiza o seu repositório local com as atualizações do repositório remoto.

Caso não tenha conhecimento de Git, recomenda-se a leitura do seguinte guia prático: [http://rogerdudler.github.io/git-guide/index.pt\\_BR.html](http://rogerdudler.github.io/git-guide/index.pt_BR.html).

## 2.4 Desenvolvimento

Agora que criamos a estrutura inicial do projeto e configuramos nosso repositório, iniciaremos nosso desenvolvimento. Criaremos nossos domínios, que no caso do nosso contexto, serão: Autor, Categoria de Artigo e Artigo.

Iniciaremos pelo desenvolvimento da Categoria de Artigo, criaremos a seguinte classe CategoriaDeArtigo, que ficará localizada em Models > CategoriasDeArtigo > CategoriaDeArtigo.cs, essa forma facilitará a organização do nosso código. A classe CategoriaDeArtigo deverá ter as propriedades de acordo com as colunas da tabela categoria\_artigo detalhadas no DER da figura 46 .

```
style=sharpc public class CategoriaDeArtigo public int Id get; set; public string Nome get; set; public string Descricao get; set;
```

O próximo passo refere-se ao mapeamento de nossa classe CategoriaDeArtigo para o modelo relacional, porém antes de começarmos, precisaremos conhecer o Nuget (Subseção 2.4.1) e depois o Entity Framework (Subseção 2.4.2) que serão detalhados nas subseções a seguir.

### 2.4.1 Nuget

O NuGet é um gerenciador de pacotes para desenvolvimento na plataforma Microsoft, o que inclui o desenvolvimento de aplicações ASP.NET MVC. Como é uma ferramenta que dispõe os pacotes a serem utilizados na aplicação em um servidor remoto, é oferecido também ferramentas clientes do NuGet, que permitem produzir e consumir pacotes NuGets. Ele é distribuído como uma extensão do Visual Studio, e se assemelha ao NPM (Node Package Manager), que é um gerenciador de pacotes para a linguagem de programação JavaScript muito conhecido no desenvolvimento web.

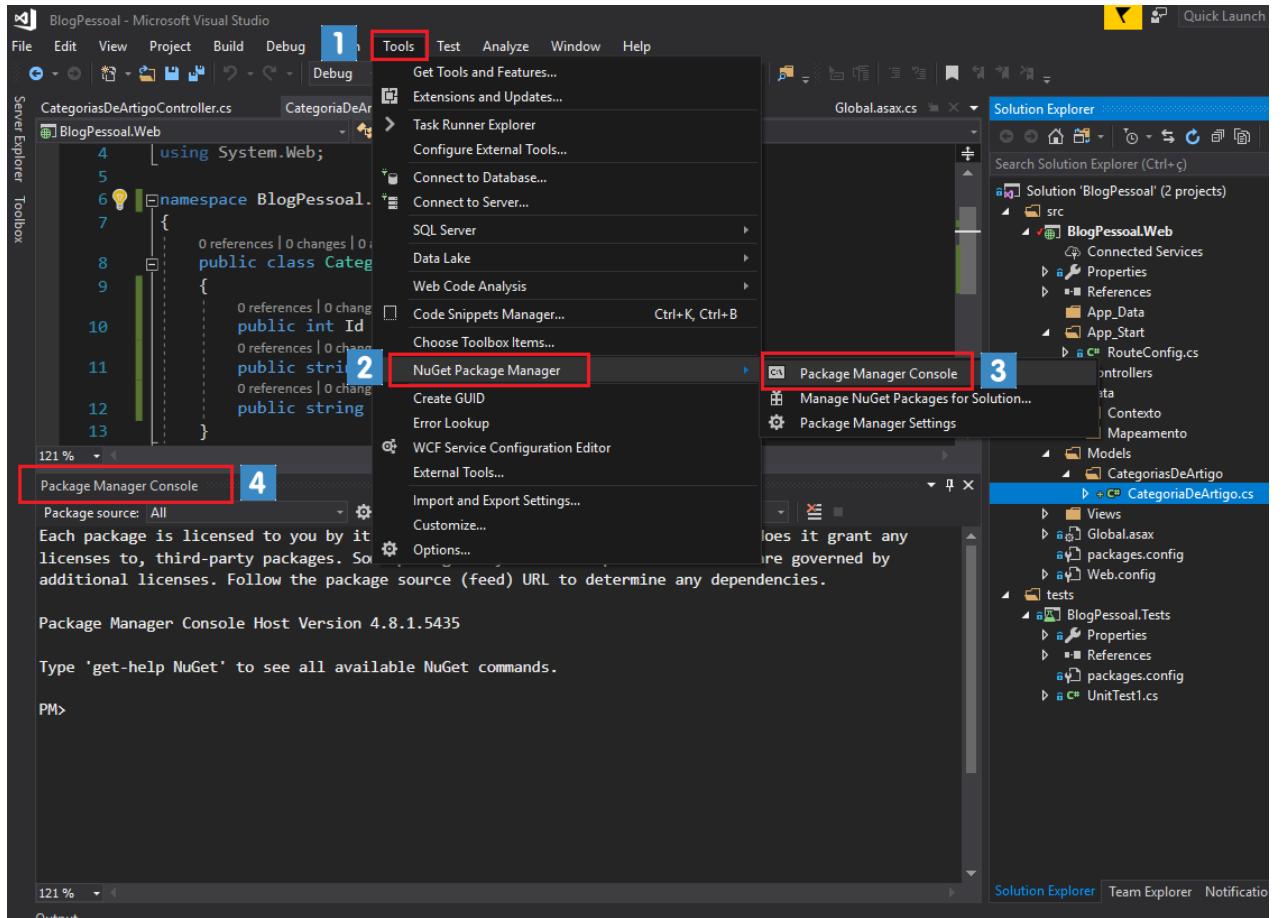


Figure 50: Apresentando o Package Manager Console.

Precisaremos do Nuget para referenciar o Entity Framework em nossa aplicação. Para abrir a tela responsável pela instalação, basta acessar o menu, opção Tools > Nuget Packager Manager > Packager Manager Console, um prompt de comando (4) deve aparecer na parte de baixo do Visual Studio, conforme demonstra a figura 50.

Para instalar o Entity Framework, basta verificar se a opção “Default project” está setado para o projeto “BlogPessoal.Web”, e digitar o seguinte comando: Install-Package EntityFramework, a figura 51 demonstra este processo. Nuget é o gerenciador de bibliotecas para plataforma .NET, e podemos acessar vários recursos por meio da URL: <https://www.nuget.org/>.

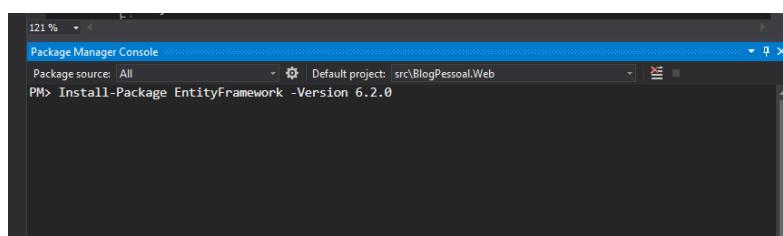


Figure 51: Instalando o Entity Framework por meio do Package Manager Console.

Há outra maneira para instalar bibliotecas, conforme ilustra a figura 52. Veja que ao selecionar o projeto e clicar com o botão direito, depois selecionar na opção

“Manage Nuget Packages”, aparecerá a tela para pesquisar pacotes, listar os já instalados e atualizações destes por meio do Nuget.

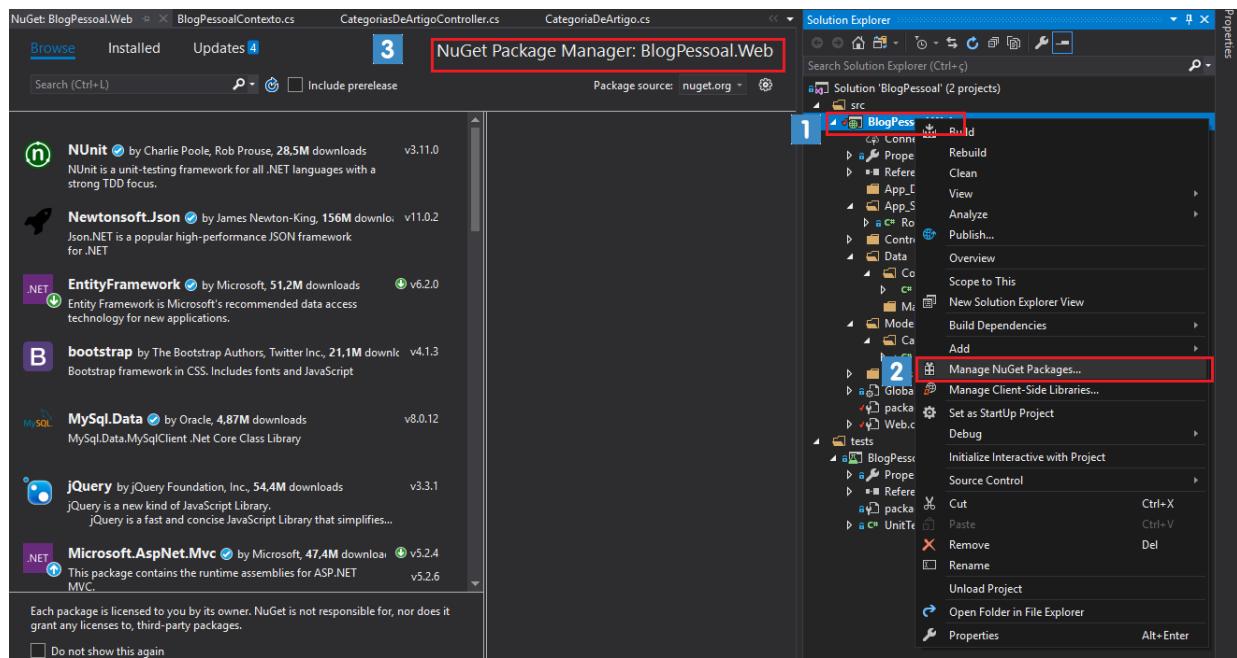


Figure 52: Instalando o Entity Framework por meio do Nuget Package Manager.

### 2.4.2 Entity Framework (EF)

O Entity Framework, ou apenas EF, é um framework para persistência de dados criado pela Microsoft, responsável pelo relacionamento objeto/relacional, conhecido como Object Relational Mapper (ORM). Ele gera objetos de negócios e entidades de acordo com as tabelas do banco de dados.

A figura 53 ilustra o funcionamento de um ORM. Ele faz o mapeamento da sua classe para o banco de dados, e assim gera o SQL, por exemplo, uma consulta referente a inserção do objeto que corresponde a uma tabela no banco de dados e realizar a operação.

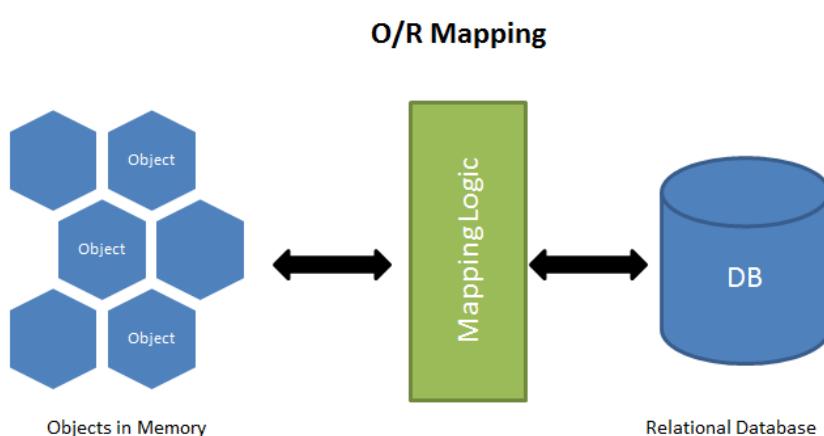


Figure 53: Mapeamento objeto relacional. Fonte: [14]

Com o uso do Entity Framework, o código fonte torna-se mais fácil de ser compreendido, facilitando a manutenção do sistema. O desenvolvimento utilizando este framework acontece de forma acelerada, ou seja ganhamos produtividade, pois não é necessário digitar comandos SQL para inserção, alteração, remoção e seleção.

Nesse contexto, será utilizado o Entity Framework para integrar os serviços criados com o banco de dados que será gerado pela nossa aplicação. Dessa forma, serão desenvolvidos serviços muito mais interessantes, complexos e elaborados.

No site da Microsoft está disponível a documentação do Entity Framework, acesse a seguinte URL <https://docs.microsoft.com/pt-br/ef/ef6/get-started>.

Para o projeto Blog Pessoal, precisaremos criar nosso Contexto. Ele é responsável por representar a conexão com o banco de dados. Em nosso projeto, criaremos a pasta Data (localizada em BlogPessoal.Web > Data), e dentro dela criaremos uma pasta chamada Contexto. Após, adicionaremos uma nova classe, nomeada como BlogPessoalContexto, o contexto é uma classe que estende System.Data.Entity.DbContext.

O DbContext dispõe de uma propriedade do tipo DbSet< TEntity > para cada classe de entidade que deverá ser representada na base de dados. Além disso, é no contexto que referenciaremos nossas classes de mapeamento da entidade.

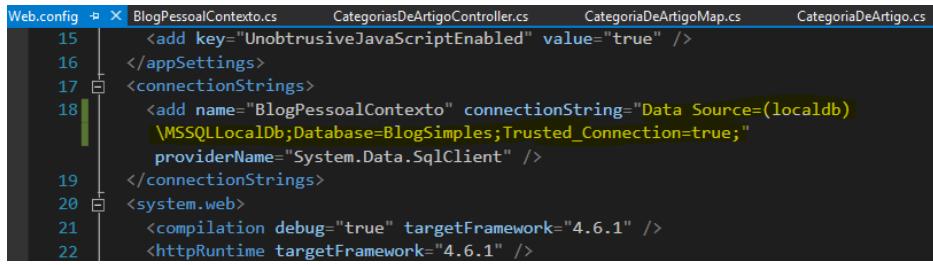
Adicionaremos a pasta Mapeamento (localizada em BlogPessoal.Web > Data > Mapeamento). O próximo passo refere-se ao mapeamento de nossa classe para o modelo relacional, para isso criaremos uma classe de mapeamento chamada CategoriaDeArtigoMap conforme demonstra a listagem.

```
style=sharpc public class CategoriaDeArtigoMap :  
EntityTypeConfiguration<CategoriaDeArtigo> public CategoriaDeArtigoMap()  
ToTable("categoria_artigo", "dbo");  
HasKey(x => x.Id);  
Property(x => x.Nome).IsRequired()  
.HasMaxLength(150).HasColumnName("nome"); Property(x =>  
x.Descricao).IsOptional() .HasMaxLength(300).HasColumnName("descricao");
```

A classe de mapeamento CategoriaDeArtigoMap deverá ser referenciada em nosso Contexto BlogPessoalContexto, conforme apresenta listagem a seguir.

```
style=sharpc public class BlogPessoalContexto : DbContext public  
DbSet<CategoriaDeArtigo> CategoriaDeArtigo get; set;  
protected override void OnModelCreating(DbModelBuilder modelBuilder)  
modelBuilder.Configurations.Add(new CategoriaDeArtigoMap());  
base.OnModelCreating(modelBuilder);
```

Por fim, precisaremos adicionar nossa string de conexão com o banco de dados no Web.config (localizada: BlogPessoal.Web > Web.config), conforme ilustra a figura 54.



```
Web.config  + X BlogPessoalContexto.cs  CategoriasDeArtigoController.cs  CategoriaDeArtigoMap.cs  CategoriaDeArtigo.cs
  15      <add key="UnobtrusiveJavaScriptEnabled" value="true" />
  16    </appSettings>
  17  <connectionStrings>
  18    <add name="BlogPessoalContexto" connectionString="Data Source=(localdb)
  \MSSQLLocalDb;Database=BlogSimples;Trusted_Connection=true;" providerName="System.Data.SqlClient" />
  19  </connectionStrings>
  20 <system.web>
  21   <compilation debug="true" targetFramework="4.6.1" />
  22   <httpRuntime targetFramework="4.6.1" />
```

Figure 54: String de conexão no web.config.

String de conexão para banco de dados local: Data Source=(localdb)

MSSQLLocalDb;Database=BlogSimples;Trusted\_Connection=true;

Reproduza o trabalho realizado da entidade CategoriaDeArtigo para as entidades Artigo e Autor.

Lembre-se que, caso seja necessário, os códigos fontes completos da aplicação estão no GitHub: <https://github.com/poferrari/asp.net-mvc-blog-simples>. Mas é um excelente exercício que você mesmo crie todas as entidades, quanto mais código for implementado neste período de estudo, melhor ficará sua aprendizagem.

### 3. Desenvolvendo a apresentação do nosso site (front-end)

No desenvolvimento de uma aplicação é importante o uso de bibliotecas que são criadas para realizar tarefas específicas e auxiliar no desenvolvimento. Dessa forma, aprenderemos novos recursos do framework ASP.NET MVC voltados à apresentação do site, também apresentaremos um pouco de JavaScript e jQuery, bem como introduziremos o Bootstrap.

#### 3.1 jQuery

A jQuery é uma das mais famosas e funcionais bibliotecas baseadas em JavaScript existentes. Ela facilita o trabalho e manipulação dos objetos DOM (Document Object Model), chamadas Ajax, manipulação de eventos e animações, além de proporcionar a criação de plugins sobre ela.

Com jQuery conseguimos resolver algumas incompatibilidades que há entre os navegadores, também reduzimos a quantidade de código implementada para uso de algumas tarefas, pois ela facilita e simplifica o desenvolvimento. A figura 55 demonstra o codificação utilizando só JavaScript e depois com a biblioteca jQuery, quando comparado, visualizamos que com jQuery simplificamos a implementação.

The figure shows two side-by-side code snippets. The left snippet is labeled 'JavaScript' and the right is labeled 'jQuery'. Both snippets accomplish the same task: creating a table, adding it to a container, adding a header row, and adding header cells. The jQuery version is significantly shorter and more concise than the plain JavaScript version.

```
JavaScript:
// Create table.

var table = document.createElement('table');

document.getElementById("data-list").appendChild(table);

// Add the header row.

var header = table.createTHead();

var row = header.insertRow(-1);

for (var i = 0; i < columnCount; i++) {

    var cell = row.insertCell(i);

    cell.innerHTML = columnHeadings[i].toUpperCase();
}

jQuery:
// Create table.

var table = $('




```

Figure 55: Comparação de desenvolvimento utilizando só JavaScript e depois com a biblioteca jQuery. Fonte: [4].

Caso queira aprender mais sobre jQuery, segue o link de um tutorial da W3Schools: [https://www.w3schools.com/jquery/jquery\\_get\\_started.asp](https://www.w3schools.com/jquery/jquery_get_started.asp).

#### 3.2 Bootstrap

Bootstrap é uma biblioteca de CSS (Cascading Style-Sheet) e componentes jQuery que facilita o trabalho de estruturar ou criar novos layouts para aplicações. Ele amplamente difundido e utilizado em projetos web, além de ser disponibilizados gratuitamente.

Proporciona um desenvolvimento para design responsivo, ou seja, ele faz com que o site consiga responder de maneira adequada às requisições de diferentes dispositivos e navegadores. Dessa forma, sua aplicação será sempre visível e as

informações ficaram dispostas de uma forma organizada em seu layout.

No próprio site do Bootstrap é fornecido um grande conjunto de elementos HTML, há várias configurações CSS que podem ser utilizadas, além de um excelente sistema de grades para ajudar no design de páginas web. Caso você necessite de um componentes mais específico, com Bootstrap você pode construí-lo ou tentar encontrar algo já desenvolvido disponível na web, pois existe uma comunidade forte que também disponibiliza componentes implementados com Bootstrap.

Caso queira aprender mais sobre Bootstrap, segue o link de um tutorial da W3Schools: <https://www.w3schools.com/bootstrap4/>

### 3.3 Layouts e Sections

Em nosso projeto Blog Pessoal, após implementar a Controller CategoriasDeArtigo e Action Index, vamos criar a respectiva View Index, utilize o recurso já estudado no Módulo 1 (subseção 4.3) que já cria a View direto da nossa Action. Após a criação automática da View Index de CategoriasDeArtigo, percebe-se que automaticamente o Visual Studio acrescentou as bibliotecas do Bootstrap e da jQuery. Além desses recursos, também foi criado na pasta Views (localizada em BlogPessoal.Web > Views) o arquivo \_ViewStart.cshtml e uma pasta chamada Shared.

Em Shared, o arquivo \_Layout.cshtml visa auxiliar no desenvolvimento de áreas em comum que serão utilizadas por várias páginas, ou seja, não é produtivo implementarmos essas áreas comuns em cada página a ser desenvolvida. Para isso, fazemos uso de layouts.

No \_Layout.cshtml, podemos definir, como por exemplo, os menus de categorias de artigos, alguma logomarca do blog, um rodapé com as informações de contato. A figura 56 apresenta a codificação desenvolvida para o Layout do Blog Pessoal, verificamos que logo no início da página é realizada a referência para os arquivos CSS das bibliotecas adicionados no projeto (1). Em seguida, verificamos a criação de um menu superior (2), após visualizamos o local que será apresentada o conteúdo das Views (@RenderBody()) (3), todo o resto será área em comum para as Views que referenciaremos este Layout. Por fim, visualizamos um simples rodapé (4) e também a referencia dos arquivos JavaScript da jQuery e do Bootstrap (5).

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8" />
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>@ViewBag.Title - My ASP.NET Application</title>
7      <link href("~/Content/Site.css" rel="stylesheet" type="text/css" />
8      <link href("~/Content/bootstrap.min.css" rel="stylesheet" type="text/css" />
9      <script src "~/Scripts/modernizr-2.6.2.js"></script>
10 </head>
11 <body>
12     <div class="navbar navbar-inverse navbar-fixed-top">
13         <div class="container">
14             <div class="navbar-header">
15                 <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
16                     <span class="icon-bar"></span>
17                     <span class="icon-bar"></span>
18                     <span class="icon-bar"></span>
19                 </button>
20                 @Html.ActionLink("Application name", "Index", "Home", new { area = "" }, new { @class = "navbar-brand" })
21             </div>
22             <div class="navbar-collapse collapse">
23                 <ul class="nav navbar-nav">
24                 </ul>
25             </div>
26         </div>
27     </div>
28
29     <div class="container body-content">
30         @RenderBody()
31         <hr />
32         <footer>
33             <p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>
34         </footer>
35     </div>
36
37     <script src "~/Scripts/jquery-1.10.2.min.js"></script>
38     <script src "~/Scripts/bootstrap.min.js"></script>

```

Figure 56: Página Layout.

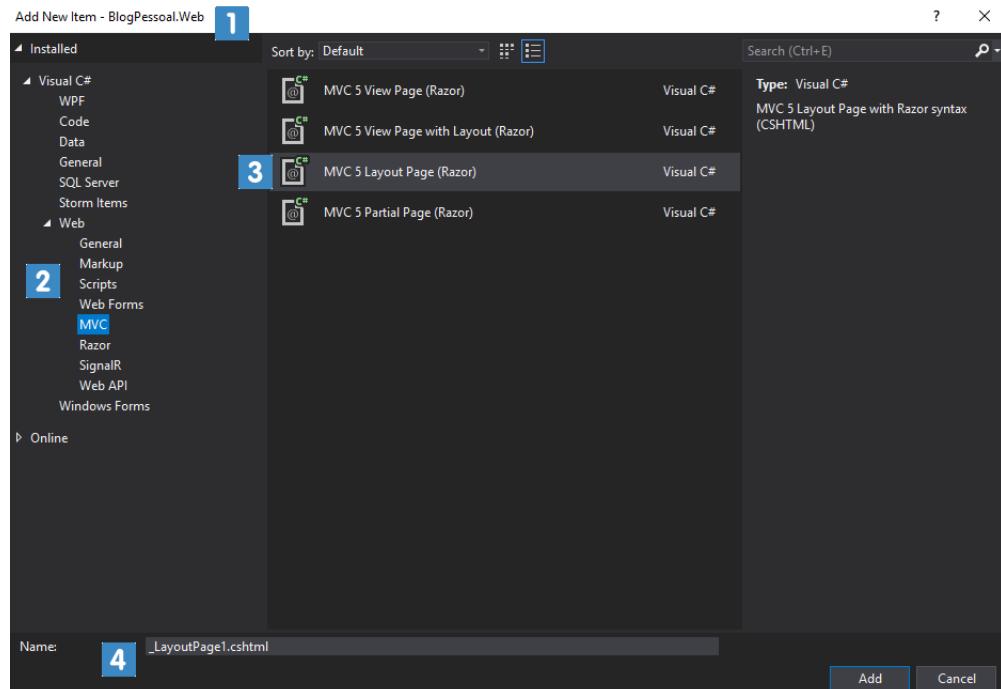


Figure 57: Criando um novo Layout.

Caso você queira construir o Layout do zero, basta acessar a pasta Shared (localizada em BlogPessoal.Web > Views), clicar com o botão direito do mouse, navegar até a opção Add > New Item (1), após filtre pela opção Web > MVC (2),

selecione a opção MVC 5 Layout Page (Razor) (3). Nomeie este Layout com o nome: \_LayoutBlog.cshtml (4), o arquivo começando com um sublinhado (“\_”), fará com que a visão não seja requisitada. Este processo está demonstrado pela figura 57.

Neste Layout desenvolvido especificamente para o Blog Pessoal, podemos ver o uso do Razo. A figura 58 apresenta a instrução Razor @RenderBody(), ela renderizará o conteúdo das Views que utilizam este layout. Outra questão, é o uso do recurso @RenderSection, ele fornece regiões específicas de código em uma View, para uma página Layout, como por exemplo, há scripts e folhas de estilo que serão utilizadas em uma View específica, desse modo não precisamos adicionar os scripts no Layout.

Para obter detalhes maiores sobre @RenderBody(), @RenderSection(), entre outros recursos do MVC, acesse <https://docs.microsoft.com/pt-br/aspnet/web-pages/overview/ui-layouts-and-themes/3-creating-a-consistent-look>.

```
1      <!DOCTYPE html>
2
3      <html lang="pt-br">
4          <head>
5              <meta charset="utf-8">
6              <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
7              <title>@string.Format("{0} - Blog Pessoal", ViewBag.Title)</title>
8              <link href("~/Content/Site.css" rel="stylesheet" type="text/css" />
9              <link href("~/Content/bootstrap.min.css" rel="stylesheet" type="text/css" />
10             <script src "~/Scripts/modernizr-2.6.2.js"></script>
11             @RenderSection("StyleContent", required: false)
12         </head>
13         <body>
14             <div>
15                 @RenderBody()
16             </div>
17             <script src "~/Scripts/jquery-1.10.2.min.js"></script>
18             <script src "~/Scripts/bootstrap.min.js"></script>
19             @RenderSection("ScriptContent", required: false)
20         </body>
21     </html>
```

Figure 58: Página Layout do Blog Pessoal.

Para fixar este conteúdo de Sections, a desenvolvedora Maria Jesusa Galapon elaborou uma resposta em um fórum que exemplifica bem o que acabamos de estudar. O conteúdo descrito por ela é apresentada pela figura 59, no exemplo desenvolvido verifica-se a criação de um Layout (1) definindo a @RenderSection("scripts"), required: false, depois a implementação de uma View Contacts (2) que utiliza o recurso @section scripts passando um arquivo JavaScript específico para esta View About (3), após a implementação de uma View (3). Por fim, visualize que na apresentação da View Contacts é acrescentado o JavaScript específico adicionado na Section, o que não ocorre para View About.

(1) you have a \_Layout.cshtml view like this

```
<html>
  <body>
    @RenderBody()

  </body>
  <script type="text/javascript" src("~/lib/layout.js")></script>
  @RenderSection("scripts", required: false)
</html>
```

(2) you have Contacts.cshtml

```
@section Scripts{
  <script type="text/javascript" src "~/lib/contacts.js"></script>

}
<div class="row">
  <div class="col-md-6 col-md-offset-3">
    <h2>    Contacts</h2>
  </div>
</div>
```

(3) you have About.cshtml

```
<div class="row">
  <div class="col-md-6 col-md-offset-3">
    <h2>    Contacts</h2>
  </div>
</div>
```

On your layout page, if required is set to false "@RenderSection("scripts", required: false)", When page renders and user is on about page, the contacts.js doesn't render.

```
<html>
  <body><div>About</div>
  </body>
  <script type="text/javascript" src "~/lib/layout.js"></script>
</html>
```

If required is set to true "@RenderSection("scripts", required: true)", When page renders and user is on ABOUT page, the contacts.js STILL gets rendered.

```
<html>
  <body><div>About</div>
  </body>
  <script type="text/javascript" src "~/lib/layout.js"></script>
  <script type="text/javascript" src "~/lib/contacts.js"></script>
</html>
```

*IN SHORT, when set to **true**, whether you need it or not on other pages, it will get rendered anyhow. If set to **false**, it will render only when the child page is rendered.*

Figure 59: Exemplo de funcionamento do RenderSection.



## 4. Outros recursos

Aprenderemos outros recursos para incrementarmos nosso Blog Pessoal. Usaremos outros templates para o projeto desenvolvido, além de conheceremos mais recursos do ASP.NET MVC e ferramentas que irão automatizar o desenvolvimento.

### 4.1 Validação de Model

Podemos realizar validações no Model por meio do uso de recursos referentes ao Data Annotations ( System.ComponentModel.DataAnnotations). Desse modo, podemos aplicar regras de validação mediante atributos para as propriedades das Models.

A figura 60 demonstra a Model CategoriaDeArtigo, nela podemos verificar o uso de alguns atributos, como por exemplo o atributo [Required], que define se tratar de uma propriedade para preenchimento é obrigatório, também há o atributo [StringLength] para validar o tamanho dos textos informados. Com o atributo [Display(Name = "\_\_\_\_")] definimos o nome da propriedade, este nome será utilizado pelas classes de validação, HTML helpers e informativos, desse modo, precisamos deixar claro e de acordo com a correta ortografia. No atributo [DataType(DataType.\_\_\_\_)] definimos o tipo da propriedade, ele auxilia o HTML helpers, especificamente o recurso Html.EditorFor() do Razor, a renderizar tag do HTML de maneira automática, por exemplo, ao definirmos o [DataType(DataType.MultilineText)] a View renderizará um text area do HTML. Ele também auxilia na validação dos campos, veja pela tabela 2 os tipos de dados disponíveis para configuração do DataType. É importante destacar que os formatos para Data e Número de Telefone estão, por padrão, no formato americano e devem ser customizados para atender nossas necessidades.

```
public class CategoriaDeArtigo
{
    4 references | JOAO VITOR FERRARI DA SILVA, 6 days ago | 1 author, 1 change
    public int Id { get; set; }

    [Required]
    [StringLength(150)]
    16 references | JOAO VITOR FERRARI DA SILVA, 6 days ago | 1 author, 1 change
    public string Nome { get; set; }

    [Required]
    [StringLength(300)]
    [Display(Name = "Descrição")]
    [DataType(DataType.MultilineText)]...
    10 references | JOAO VITOR FERRARI DA SILVA, 6 days ago | 1 author, 1 change
    public string Descricao { get; set; }

    1 reference | JOAO VITOR FERRARI DA SILVA, 1 day ago | 1 author, 1 change
    public virtual ICollection<Artigo> Artigos { get; set; }
}
```

Figure 60: Exemplo de validação do Model CategoriaDeArtigo.

Table 2: Tipos de Dados do Atributo [DataType].

Atributo	Tipo de dado
DataType	Currency
	Custom
	Date
	DateTime
	Duration
	EmailAddress
	Html
	ImageUrl
	MultilineText
	Password
	PhoneNumber
	Text
	Time
	Url

A figura 61 apresenta as implementações necessárias para validação da Model na View. O uso do recurso `@Html.ValidationSummary()`, para retornar uma mensagem padrão caso algum erro seja identificado pelo recurso `@Html.ValidationMessageFor()`, na qual definimos a propriedade da Model que queremos validar.

```

@using (Html.BeginForm())
{
    <div>

        @Html.ValidationSummary(true, "Problemas encontrados.", new { @class = "text-danger" })

        <div class="form-group">
            @Html.LabelFor(model => model.Nome)
            <div>
                @Html.EditorFor(model => model.Nome)
                @Html.ValidationMessageFor(model => model.Nome, "", new { @class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Descricao)
            <div>
                @Html.EditorFor(model => model.Descricao)
                @Html.ValidationMessageFor(model => model.Descricao, "", new { @class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            <input type="submit" value="Create" class="btn btn-default" />
        </div>
    </div>
}

```

Figure 61: Sintaxe Razor para erros da validações de Model.

Há várias validações que podem ser utilizadas, porém também podemos customizar validações específicas, como por exemplo, verificação de CPF e CNPJ. Para maiores detalhes acesse: <https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions/getting-started-with-aspnet-mvc4/adding-validation-to-the-model>.

#### 4.1.1 Validação com RegularExpressions

Por meio de expressão regular podemos implementar uma validação, para isto utilizaremos o RegularExpression. Como exemplo, podemos validar se o e-mail informado é válido, o código a seguir demonstra a implementação desta validação.

```
style=sharpc           public           class           Login
[RegularExpression(@"^([a-zA-Z0-9_-]+@[a-zA-Z0-9_-]+\.(?!\d{2,3})+)", ErrorMessage = "O e-mail errado.")]
public string Email {get; set; }

sharpcclass CustomValidationCPFAttribute : ValidationAttribute, IClientValidatable
{
    public void Validate(ModelMetadata metadata, ControllerContext context)
    {
        if (String.IsNullOrEmpty(metadata.ModelValue))
            return;
        var value = metadata.ModelValue;
        var util = new Util();
        if (!util.ValidaCPF(value))
            ModelState.AddModelError(metadata.PropertyName, "CPF inválido");
    }

    public IEnumerable GetClientValidationRules(ModelMetadata metadata, ControllerContext context)
    {
        var rule = new ModelClientValidationRule()
        {
            ValidationType = "customvalidationcpf",
            ErrorMessage = "O atributo CustomValidationCPFAttribute deve ser chamada em sua Model"
        };
        yield return rule;
    }
}

sharpcclass PessoaDto [Required]
public string Nome {get; set; }

[CustomValidationCPF(ErrorMessage = "CPF em formato invalido")]
public string CPF {get; set; }
```

The screenshot shows a browser window with the title 'Create - My ASP.NET Application'. The URL in the address bar is 'localhost:62333/Pessoas/Create'. The page itself is titled 'Create' and has a subtitle 'Pessoa'. There are two input fields: 'Nome' with the value 'João da Silva' and 'CPF' with the value '123.456.789-10'. Below the CPF field, the text 'CPF incorreto' is displayed in red, indicating an validation error. At the bottom of the form is a 'Create' button.

© 2018 - My ASP.NET Application

Figure 62: Validação com atributo customizado criado para CPF.

A figura 62 apresenta a validação com atributo customizado criado para CPF. O CPF informado não atende a validação desenvolvida pelo método `ValidaCPF` da classe de utilitários. A implementação completa deste código pode ser acessada no projeto Loja Virtual, que será apresentado no próximo módulo.

## 4.2 Action Filters

Action Filters executam lógicas de filtragem antes ou depois que uma Action é chamada. Com ela podemos incluir um filtro que será executado para adicionar um comportamento aos métodos da Controller. Há alguns tipos de filters específicos para cada tipo de necessidade, conforme listagem a seguir [18]:

- **Authorization Filter** – segurança e controle de usuários autenticados;
- **Action Filter** - injeção de comportamento na execução de um método Action;
- **Result Filter** - injeção de comportamento na execução de um Action Result;
- **Exception Filter** - captura e tratamento na ocorrência de Exceptions.

A figura 63 demonstra um exemplo de filters para autorização de usuários logados. Veja que o filtro `Authorize` definido no Controller implica que todos as Actions ali definidas só podem ser acessadas por meio de autorização válida, porém na Action

Login utilizamos o filtro AllowAnonymous, que permite acesso independente de qualquer autorização.

```
[Authorize]
public class AccountController : Controller
{
    [AllowAnonymous]
    public ActionResult Login(string returnUrl)
    {
        ViewBag.ReturnUrl = returnUrl;
        return View();
    }
}
```

Figure 63: Exemplo de filtro para autorização de usuários logados. Fonte: [18].

Existe a opção de criarmos Action Filters Customizados, pois muitas vezes necessitamos de um comportamento diferenciado para atender algum requisito solicitado, como por exemplo, uma verificação de login customizada do autor para publicação de um artigo no Blog Pessoal.

Com este recurso podemos implementar vários funcionalidades, como:

- Validação de regras para acesso dos usuários, questões de permissionamento;
- Captura de IP, endereço MAC ou outras informações do cliente;
- Tratamento e notificações de exceções;
- Logs de atividades dos usuários;
- Exibição de propagandas, produtos, entre outras funcionalidades.

Para obter detalhes maiores sobre Action Filters, acesse as seguintes URLs:

- <http://www.tutorialsteacher.com/mvc/action-filters-in-mvc>;
- <https://www.codeproject.com/Articles/426766/Custom-Filters-in-MVC-Authorization-Action-Result>;
- <https://docs.microsoft.com/pt-br/aspnet/mvc/overview/older-versions/hands-on-labs/aspnet-mvc-4-custom-action-filters>;
- [https://www.infragistics.com/community/blogs/b/dhananjay\\_kumar/posts/how-to-create-a-custom-action-filter-in-asp-net-mvc](https://www.infragistics.com/community/blogs/b/dhananjay_kumar/posts/how-to-create-a-custom-action-filter-in-asp-net-mvc).

### 4.3 Escolher uma tema

Existe a possibilidade de alterar os estilos utilizados em nosso projeto. Como por exemplo, acesse o site <https://bootswatch.com/>, nele você encontra vários temas do Bootstrap.

Em nosso projeto utilizaremos o tema Lumen, localizado em <https://bootswatch.com/lumen/>. Para usarmos este tema precisaremos apenas fazer o download do arquivo CSS (<https://bootswatch.com/4/lumen/bootstrap.css>) e adicioná-lo em nosso projeto.

## 4.4 Scaffolding

Este recurso ajuda na produtividade, ele simplifica e automatiza tarefas, como por exemplo, podemos utilizar Scaffolding para gerar código associado a funcionalidades de inclusão, leitura, alteração e exclusão de dados (CRUD).

Após configurarmos nossa entidade e mapeá-la por meio do Entity Framework, podemos utilizar este recurso para criar um CRUD completo para Artigos do projeto Blog Pessoal. A figura 64 demonstra o uso do Scaffolding por meio da criação de um Controller para gerenciar os Artigos.

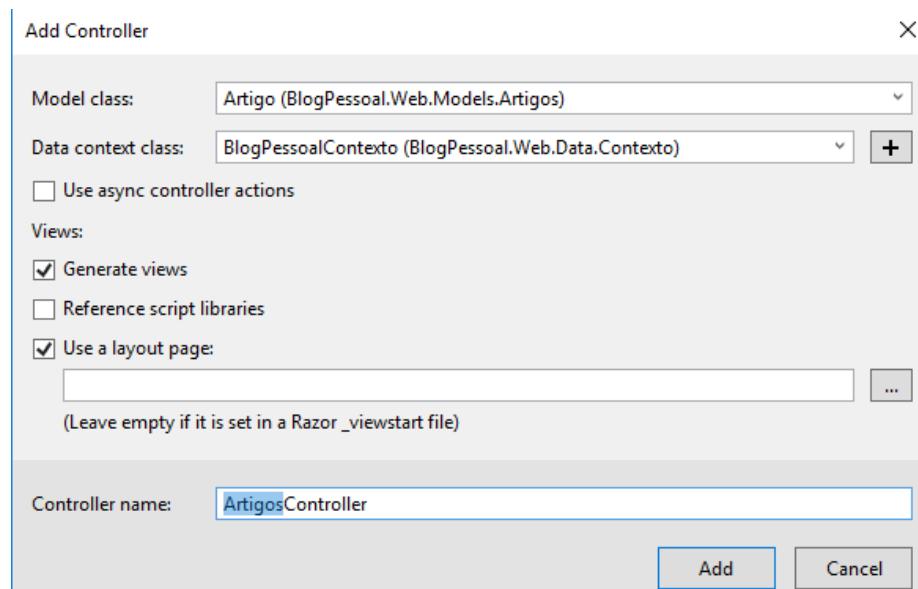


Figure 64: Scaffolding para Controller de Artigos.

Após criarmos nosso ArtigosController, percebemos que foram adicionadas várias Views (Create.cshtml, Delete.cshtml, Details.cshtml, Edit.cshtml e Index.cshtml) criadas automaticamente para atender as Actions desenvolvidas nesta Controller.

Há Scaffolding para criação de Views também, podemos definir nosso template, por exemplo, "Create" e após escolhemos o modelo utilizado, por exemplo a entidade Artigo, conforme ilustra a figura 65.

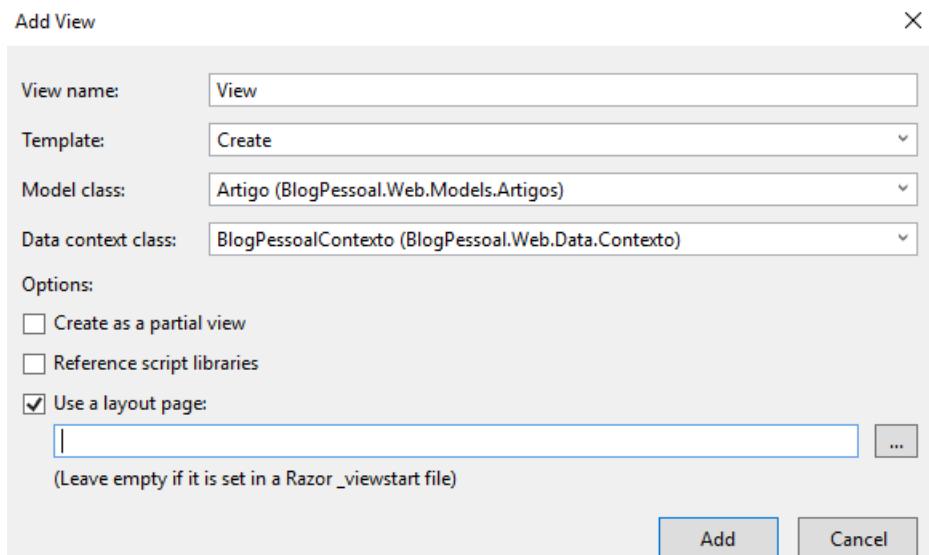


Figure 65: Scaffolding para View de criação de Artigo.

Este recurso é muito bem-vindo para ganharmos tempo de desenvolvimento, é indicado para fazermos projetos pessoais ou aplicações mais simples que não precisem de um maior controle do código gerado nas Views e nos Controllers.

## 5. Considerações finais

Neste capítulo, aprendemos como organizar o nosso trabalho. Dividimos o desenvolvimento no back-end, em que trabalhamos com as Controllers, os Models e o Entity Framework, para comunicação com o banco de dados, vimos a organização do nosso projeto Blog Pessoal.

Também iniciamos o desenvolvimento front-end com Bootstrap e jQuery, além de conhecermos outros recursos do ASP.NET MVC. Percebe-se, o quanto a IDE Visual Studio nos ajuda no desenvolvimento.

Aprendemos a automatizar as validações de Models por meio de atributos, no caso do projeto Blog Pessoal, realizamos as validações de Model nas entidades por ser um projeto simples, porém o recomendado é que seja criada a entidade para mapeamento do Entity Framework e separadamente seja criada a Model para view com as validações de atributos.

Conhecemos o Scaffolding de View para ações de CRUD baseado no modelo escolhido, vimos a criação de todo o código da View automaticamente. Também vimos o Scaffolding para acréscimo de Controller com todas as ações de CRUD desenvolvidas, tanto a parte do banck-end e quanto do front-end. É um recurso fantástico, porém é indicado para projetos simples, na qual não precisamos ter muito controle sobre o código que está sendo gerado, pois é se trata de um código padrão usando diretamente o contexto, algo que não muito indicado para projetos mais organizados.

Desse modo, após este módulo estamos aptos a criarmos site simples com ASP.NET MVC.

## UNIDADE 3

# Organizando um projeto complexo em ASP.NET MVC

Neste capítulo abordaremos a organização de um projeto complexo em ASP.NET MVC, aprenderemos alguns padrões de projeto, metodologias de trabalho, como Test Driven Development (TDD), além de outros conceitos muito discutidos entre a comunidade, como a questão da modelagem de domínios ricos, por meio da Programação Orientada a Objetos (POO) e Domain Driven Design (DDD).

Todos estes conceitos não serão detalhados a fundo, apenas serão comentados para que você tenha interesse em conhecê-los e aplicá-los em seus projetos, porém sempre que possível deixarei links interessantes sobre o assunto para que você não fique perdido. Nossa objetivo será a organização de um projeto complexo em ASP.NET MVC explorando ao máximo todos os recursos e ferramentas que sejam úteis para o seu trabalho.

Antes de iniciarmos um novo projeto mais complexo em ASP.NET MVC, vamos utilizar o projeto Blog Pessoal para entendermos recursos mais complexos do framework e atendermos os requisitos solicitados pelo projeto que foram descritos no capítulo anterior (1.1).

## 1. Detalhando recursos do ASP.NET MVC

Nesta seção, abordaremos os diferentes modos de trabalho com Partial View. Com estes recursos atenderemos a outros requisitos elencados no capítulo anterior para o desenvolvimento do Blog Pessoal.

### 1.1 Trabalhando com Partial View

Com o projeto Blog Pessoal, entendermos os diferentes modos de trabalho da Partial View. Desse modo, vamos listar as algumas das diferentes maneiras existentes no MVC [24]:

- `Html.Action`: renderiza a Partial View como um `HtmlString`, com ele precisamos criar uma Action para renderizar a Partial View, é útil quando os dados exibidos são independentes da View Model correspondente. Com ele conseguimos armazenar em uma outra variável, ou seja, primeiro ele retorna o resultado como uma string e depois renderiza o resultado para a resposta. Um exemplo no projeto Blog Pessoal seria mostrar a lista de categorias no Layout;
- `Html.RenderAction`: é semelhante ao `Html.Action`, mas a principal diferença é que ele renderiza o resultado diretamente para a resposta;
- `Html.Partial`: processa a Partial View como `HTML-encoded string`, é útil quando os dados de exibição da Partial View já estão View Model. Não há a necessidade de criar qualquer Action. Este método pode ser armazenado em uma variável, já que ele retorna um tipo `String`. Um exemplo no projeto Blog Pessoal seria

apresentar os dados de uma categoria de artigo na View de apresentação do artigo;

- `Html.RenderPartial`: é semelhante ao `Html.Partial`, mas a principal diferença é que ele renderiza o resultado diretamente para a resposta.

Vamos demonstrar os métodos `Html.Action` e `Html.RenderAction`, que são uma ótima escolha se quisermos cachejar a Partial View. Primeiramente, criaremos nossa Partial View para listar as categorias, daremos o nome “`_Categorias.cshtml`” (`BlogPessoal.Web > Views > Shared`) e descreveremos o código a seguir. Esta Partial View será responsável por construir uma listagem de categorias de artigo no HTML.

```
@using BlogPessoal.Web.Models.CategoriasDeArtigo @model  
IEnumerable<CategoriaDeArtigo>  
    <ul class="nav navbar-nav"> @foreach (var item in Model) <li> <a href="#">  
>@item.Nome</a> </li> </ul>
```

Conforme já visualizamos, os métodos `Html.Action` e `Html.RenderAction` necessitam de uma Action para seu funcionamento, conforme demonstra o código a seguir. Nesta Action nós retornamos todas as categorias de artigos e enviamos esta lista para a Partial View “`_Categorias.cshtml`”.

```
public ActionResult CategoriasDeArtigo() var lista =  
db.CategoriasDeArtigo.ToList(); return PartialView("../Shared/_Categorias", lista);
```

Nesse contexto, precisaremos chamar nossa Partial View na página desejada. Em nosso caso, adicionaremos os menu de categorias do artigo no “`_Layout.cshtml`”. Com estes métodos conseguimos criar uma página independente da View Model correspondente. Para realizar a chamada, usaremos `Html.Action`, caso queira armazenar o resultado em uma variável, ou `Html.RenderAction`, que é método “`void`”.

```
@Html.Action("CategoriasDeArtigo", "Home")  
@ Html.RenderAction("CategoriasDeArtigo", "Home");
```

Agora, iremos demonstrar os métodos `Html.Partial` e `Html.RenderPartial`. Primeiramente, criaremos nossa Partial View para apresentar detalhes da categoria de artigo na página de visualização dos artigos, daremos o nome “`_CategoriaDetalhes.cshtml`” (`BlogPessoal.Web > Views > Artigos`) e descreveremos o código a seguir. Esta Partial View será responsável por construir exibir detalhes da categoria do artigo no HTML.

```
@using BlogPessoal.Web.Models.CategoriasDeArtigo @model  
CategoriaDeArtigo <div> <fieldset> <legend>Categoria</legend> <dl class="dl-  
horizontal"> <dt> @Html.DisplayNameFor(model => model.Nome) </dt>  
<dd> @Html.DisplayFor(model => model.Nome) </dd>  
<dt> @Html.DisplayNameFor(model => model.Descricao) </dt>  
<dd> @Html.DisplayFor(model => model.Descricao) </dd> </dl> </fieldset>  
</div>
```

Conforme já aprendemos, os métodos `Html.Partial` e `Html.RenderPartial` não necessitam de uma Action para seu funcionamento. A implementação do nosso exemplo se resume na alteração da View de detalhes do Artigo (localizada em: `BlogPessoal.Web > Views > Artigos > Details.cshtml`) para realizar a chamada da Partial View que exibirá os dados da categoria, o arquivo “`_CategoriaDetalhes.cshtml`”. A listagem seguinte exemplifica o modo de utilização, lembrando que com o método `Html.Partial` podemos armazenar o retorno numa variável, já `Html.Partial` é o método “`void`”. Na última linha, apresentamos uma outra forma de chamar a Partial View por meio do caminho relativo.

```
style=sharpc @using BlogPessoal.Web.Models.Artigos @model Artigo  
@Html.Partial("_CategoriaDetalhes", Model.CategoriaDeArtigo)  
@ Html.RenderPartial("_CategoriaDetalhes", Model.CategoriaDeArtigo);  
    Html.RenderPartial("           /Views/Artigos/_CategoriaDetalhes.cshtml",  
Model.CategoriaDeArtigo);
```

Vale destacar que o método `Html.RenderPartial` e `Html.RenderAction` são mais rápidos entre todos os métodos apresentados nesta seção, pois a resposta é utilizado diretamente via Response Stream HTTP, o que deixa a solicitação mais rápida.

## 1.2 ASP.NET e SignalR

O SignalR<sup>9</sup> é uma implementação open source para facilitar a comunicação em tempo real de forma assíncrona, na qual envia pacotes que são geralmente transportados via WebSockets. Ele combina uma biblioteca ASP.NET no servidor e uma biblioteca JavaScript no cliente para manter a comunicação cliente/servidor aberta.

Podemos utilizar este recurso para desenvolvimento de aplicações para comunicação instantânea como os Chats (bate-papo), painel para controle de estoque, cotação para bolsa de valores, acompanhamento de jogos, entre outras ocasiões.

Para maior aprofundamento sobre esta tecnologia e conhecimento de exemplos práticos, recomenda-se a leitura dos seguintes links:

- <https://www.c-sharpcorner.com/UploadFile/4b0136/introducing-Asp-Net-signalr-in-mvc-5/>;
- <https://docs.microsoft.com/pt-br/aspnet/signalr/overview/getting-started/tutorial-getting-started-with-signalr-and-mvc>;
- <http://www.eduardopires.net.br/2013/04/aspnet-signalr-introducao-e-utilizacao/>;
- [http://www.macoratti.net/13/03/net\\_sign1.htm](http://www.macoratti.net/13/03/net_sign1.htm);
- <http://netcoders.com.br/signalr-e-mvc-pedidos-para-a-cozinha-em-tempo-real/>.

---

<sup>9</sup> <https://www.asp.net/signalr>

## 2. Organização do projeto

Neste seção iremos detalhar a organização de um projeto complexo em ASP.NET MVC. O que iremos aprender são alguns padrões de projetos, conceitos e técnicas de desenvolvimento.

Quando falamos de projeto de desenvolvimento, não há uma regra estabelecida e que deve ser seguida em todos os projetos que você for desenvolver. Será ensinado um modelo para aplicações maiores, porém é você que deve analisar a complexidade de seus projetos e encontrar a melhor maneira de organizar seu desenvolvimento.

Nosso projeto será organizado em três camadas, são elas:

- Camada de negócio (Business logic layer): responsável pela regra de negócio da aplicação;
- Camada de acesso a dados (Data access layer): responsável pelo acesso ao banco de dados, persistência dos objetos;
- Camada de apresentação (User interface/Presentation): responsável pelo desenvolvimento front-end, apresentação das informações para o usuário final.

O código fonte da aplicação que iremos construir pode ser encontrado no GitHub, acesse: <https://github.com/poferrari/asp.net-mvc-loja-virtual-exemplo>. Iniciaremos um projeto ASP.NET MVC de Loja Virtual para nosso estudo.

### 2.1 Camada de negócio (Business Logic Layer)

A camada de negócio, também chamada por Business Logic Layer (BLL), é responsável pela tutela de toda regra de negócio da aplicação. Nesta camada organizaremos as entidades utilizadas em nosso projeto, recomenda-se a leitura do Domain Driven Design (DDD), que é uma modelagem de software que independe de tecnologia, muito aplicada e difundida pela comunidade de desenvolvimento.

O DDD não é arquitetura em camadas, ele segue um conjunto de práticas com objetivo de facilitar a implementação de complexas regras / processos de negócios que são tratados como domínio [20]. Para o projeto Loja Virtual, o exemplo de um domínio seria Produtos, e todos os serviços relacionados a este domínio devem estar organizados no mesmo local.

Nosso projeto Loja Virtual utilizará o DDD, desse modo para conhecer maiores detalhes sobre este conteúdo, acesse o link:

<http://www.eduardopires.net.br/2016/08/ddd-nao-e-arquitetura-em-camadas/>

Não entraremos em maiores detalhes sobre DDD pois é um assunto que tem muito material para estudarmos, ficaremos apenas com a iniciação por meio do projeto Loja Virtual. Faço um pedido para que você possa aprofundar neste e demais assuntos que serão iniciados neste livro.

### 2.1.1 Testes

O teste é um processo que faz parte do desenvolvimento de software, pois visa validar se a aplicação está funcionando corretamente, ou seja, livre de falhas/bugs, e se atende aos requisitos especificados.

Nesse contexto, há diversas técnicas que podem ser aplicadas em diferentes momentos e formas para validar os aspectos principais do software. Na listagem a seguir conheceremos alguns tipos de teste de software e como aplicá-los em nosso projeto [9, 28].

- Teste de Unidade: tem um objetivo de testar unidades menores do software, ou seja, apenas um pedaço do código. Testa um componente ou classe de maneira isolada;
- Teste de Integração: é composto por diversos testes de unidade para verificar se eles estão funcionando adequadamente;
- Testes de Stress: eleva-se o software ao limite de sua potência e funcionamento até que deixe de funcionar adequadamente;
- Teste de Carga: avalia os limites do software, o quanto ele suporta em quantidade de dados, tráfego, números altos de usuários simultâneos, entre outras situações, sem que erros sejam apresentados;
- Teste de Performance: analisa se o tempo de resposta é o desejado para o momento de utilização da aplicação.

Por meio da IDE Visual Studio e com o projeto Loja Virtual de teste, vamos criar testes de unidade para as regras desenvolvidas. Esta ferramenta permite automatizar nossos testes, pois após finalizar a criação de um novo teste podemos executá-lo quantas vezes quisermos.

Nesse contexto, precisamos conhecer o Test Driven Development (TDD), ou Desenvolvimento Guiado pelos Testes, na qual prega que o desenvolvedor deve começar a implementação pelo teste e, deve o tempo todo, fazer de tudo para que seu código fique simples e com qualidade [1].

O importante que devemos saber sobre teste, é que o custo de correção de falhas/bugs pode ser aumentado em até 100x quando corrigidos nas fases finais de desenvolvimento, ao compararmos com o custo de corrigir os mesmos problemas em fases iniciais do projeto.

O TDD torna o processo mais confiável, além de reduzir custos, pois desenvolvemos e já sabemos o erro, pois como os testes são criados antes do processo de desenvolvimento, conseguimos testar constantemente. Outro ponto é que se os testes foram criados, isso quer dizer que foram entendidas as regras de negócio durante a fase de desenvolvimento dos testes unitários [22].

O ciclo do TDD é simples, primeiramente criamos um teste, depois fazemos a codificação para passar no teste implementado, por fim, refatoramos nosso código. A figura 66 apresenta o ciclo descrito.

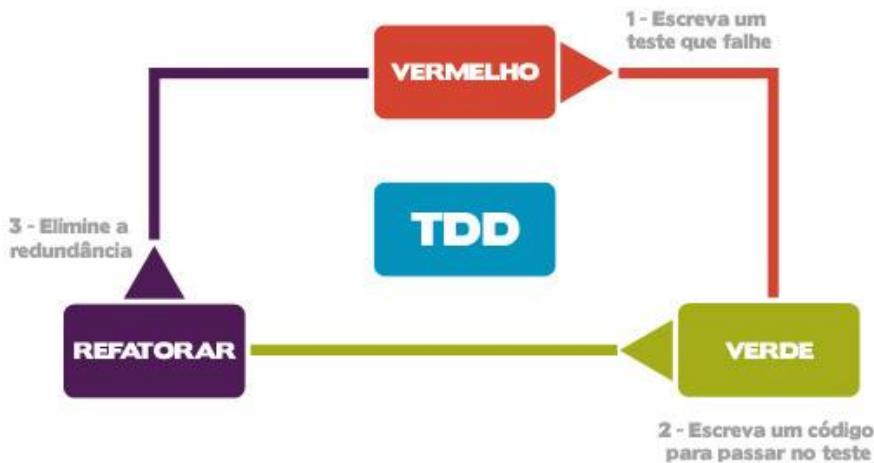


Figure 66: Ciclo do TDD. Fonte: [22].

Por estes motivos, muitas empresas e desenvolvedores optam pelo uso do TDD durante o processo de desenvolvimento. O assunto TDD está bastante comentado pela comunidade, dessa forma, recomenda-se o estudo deste método de desenvolvimento, já que neste livro apenas iniciaremos o estudo dele voltado ao nosso projeto.

Para começarmos a desenvolver nossos primeiros testes, pode ser facilitada por meio de algumas bibliotecas que adicionaremos ao nosso projeto. Existem diversas ferramentas que possibilitam esta prática, vamos a algumas:

1. NBuilder<sup>10</sup>: cria dados reais para teste, de forma fluente e extensível. As propriedades e campos públicos são preenchidos com valores automaticamente, porém permite substituí-los manualmente;
2. Faker.Net<sup>11</sup>: cria de forma fácil dados fictícios de endereços, nomes, telefones, cidades, entre outros dados;
3. Bogus<sup>12</sup>: gerador de dados simples para preencher objetos que suportam diferentes localidades;
4. Moq<sup>13</sup>: imitam objetos reais para realização de testes. Simulam a utilização de objetos que levam tempo para ser inicializados ou são muito complexos, como por exemplo, banco de dados;
5. xUnit<sup>14</sup>: simplificar e modernizar a forma de escrever testes.

Todas as ferramentas apresentadas são open-source e gratuitas.

<sup>10</sup> <https://www.nuget.org/packages/NBuilder/>

<sup>11</sup> <https://www.nuget.org/packages/Faker.Net/>

<sup>12</sup> <https://www.nuget.org/packages/Bogus/>

<sup>13</sup> <https://www.nuget.org/packages/moq/>

<sup>14</sup> <https://www.nuget.org/packages/xunit>

Implementaremos um código fonte para exemplificar o uso de cada ferramenta listada anteriormente e também o processo de desenvolvimento seguindo o TDD. Porém antes de iniciarmos, vamos aprender o padrão AAA (Arrange, Act, Assert) para criação dos nossos testes, cada termo tem o seguinte significado:

1. Arrange: organizar seção de um método de teste de unidade, inicializa os objetos e define o valor dos dados que são passados para o método que será testado;
2. Act: invoca o método testado com os parâmetros que foram organizados;
3. Assert: verifica se a ação do método em teste se comporta conforme o esperado.

Há vários tipos de Asserts que podem ser utilizados para verificar o resultado dos nossos testes, leia a documentação encontrada no site da MSDN:

<https://msdn.microsoft.com/pt-br/library/ms182530.aspx>.

Vamos para o nosso exemplo, acesse o projeto de teste, caso você ainda não tenha criado, basta adicionar um projeto do tipo “Unit Test Project (.NET Framework)” e não esquecer de nomear o projeto da seguinte maneira, por exemplo, LojaVirtual.Tests, por padrão utilize o nome da solução acrescido do “.Tests”.

Agora, iremos adicionar uma classe de teste, pelo Visual Studio acessaremos a opção para adicionar novos itens, encontre a categoria “Test”, e escolha a opção “Unit Test”, atribua o nome do domínio que será testado acrescentado do termo “Test”, por exemplo CalculadoraTest.

Em nosso exemplo de teste, simularemos o trabalho de uma Calculadora. Seguindo o TDD, vamos implementar nosso teste que verificará a soma de dois valores. A listagem a seguir apresenta o teste implementado, veja que é importante nomear claramente o nosso método, como forma didática apresentamos o teste juntamente com o padrão AAA, porém nos outros testes organizaremos este padrão pulando linhas, sem comentários os AAA.

```
style=sharpc [TestClass] public class CalculadoraTest [TestMethod] public void Somar_valores_com_sucesso() //Arrange var valor1 = 1; var valor2 = 2; var valorEsperado = 3; //Act var calculadora = new Calculadora(); var resultado = calculadora.Somar(valor1, valor2); //Assert Assert.AreEqual(valorEsperado, resultado);
```

Como já era esperado o primeiro erro é o de compilação, pois a classe Calculadora ainda não existe no projeto. Veja que o Assert é usado para afirmações, no exemplo ele afirma que o valor esperado seja igual ao valor obtido por meio do método somar da classe Calculadora.

Em nosso projeto, criaremos a classe Calculadora, implementaremos a função Somar(valor1, valor2) que retorna a soma dos dois números. Primeiro precisaremos fazer o nosso teste falhar e depois refatoraremos nosso código para que o teste seja executado com sucesso.

Por fim, nossa implementação ficará conforme as seguintes listagens.

```
style=sharpc [TestMethod] public void Somar_valores_com_sucesso() var  
valor1 = 1; var valor2 = 2; var valorEsperado = 3;  
var calculadora = new Calculadora(); var resultado = calculadora.Somar(valor1,  
valor2);  
Assert.AreEqual(valorEsperado, resultado);  
style=sharpc public class Calculadora public int Somar(int valor1, int valor2)  
return valor1 + valor2;
```

Este assunto pode ser muito mais explorado, e é importante que você aprofunde o conhecimento em testes, ele tem sido o diferencial entre os desenvolvedores. A prática e estudos levam a aprendizagem de como testar de maneira efetiva e o que realmente precisa ser testado.

### 2.1.2 Princípios SOLID

Os princípios SOLID são a base para vários padrões de projetos (GoF). São considerados uma boa prática de programação orientada a objetos que visam diminuir o acoplamento entre classes e separar responsabilidades como forma de melhorar o código da aplicação desenvolvida, além de contribuir para manutenção do software.

SOLID é um acrônimo que representa os 5 princípios da programação orientada a objetos, que são:

- **Single responsibility principle (SSRP)**: princípio da Responsabilidade Única, na qual uma classe deve ter um, e somente um, motivo para mudar;
- **Open/closed principle (OOPC)**: princípio Aberto/Fechado, na qual devemos ser capaz de estender um comportamento de uma classe sem a necessidade de modificá-lo;
- **Liskov substitution principle (LLSP)**: princípio da substituição de Liskov, na qual as classes derivadas devem ser substituíveis por suas classes bases;
- **Interface segregation principle (IISP)**: princípio da segregação de interfaces, na qual muitas interfaces específicas são melhores do que uma interface única geral;
- **Dependency inversion principle (DDIP)**: princípio da inversão de dependência, na qual precisamos depender de abstrações e não de implementações.

A utilização destes princípios tem o objetivo de evitar erros, falhas e defeitos, além de impossibilitar a implementação de estrutura de código ruim. Esses princípios tornam o software mais evolutivo, de fácil manutenção e facilita mudanças necessárias após a conclusão do projeto, não impactando em outras áreas do programa.

É importante o conhecimento do SOLID para qualquer desenvolvedor, independente do nível de classificação que se enquadre, seja júnior, pleno ou sênior. Há uma boa parte dos programadores que não aplicam esses princípios aos sistemas

desenvolvidos, é importante que estes princípios estejam enraizados em nossa codificação.

Recomenda-se o estudo e a prática dos princípios SOLID, pois estes conhecimentos te ajudarão a ser um profissional mais qualificado. Deixo alguns links para que você possa iniciar seus estudos e depois aprofundá-los, o bom de aprender padrões de projeto, princípios SOLID, é que eles são aplicados independente da linguagem, por isso é importante a compreensão do que é proposto para que você possa aplicar em suas necessidades.

- O desenvolvedor de software, Diego Neves disponibilizou uma série de artigos apresentando cada princípio SOLID com exemplos em linguagem C#, acesse:  
<http://netcoders.com.br/aplicando-solid-com-c-srp/>;  
<http://netcoders.com.br/aplicando-solid-com-c-ocp/>;  
<http://netcoders.com.br/aplicando-solid-com-c-lsp-liskov-substitution-principle/>;  
<http://netcoders.com.br/aplicando-solid-com-c-isp-interface-segregation-principle/>;  
<http://netcoders.com.br/aplicando-solid-com-c-dip-dependency-inversion-principle/>;
- No site Code Project, há um artigo com conceitos e exemplos em Inglês, acesse: <https://www.codeproject.com/Articles/703634/SOLID-architecture-principles-using-simple-Csharp>;
- No site C# Corner, há um outro artigo com conceitos e exemplos em Inglês, acesse: <https://www.c-sharpcorner.com/UploadFile/damubetha/solid-principles-in-C-Sharp/>;
- Livro: Orientação a Objetos e SOLID para Ninjas - Projetando classes flexíveis (<https://www.casadocodigo.com.br/products/livro-oo-solid>).

Agora que iniciamos nossos estudos sobre os princípios SOLID, melhoraremos o que é proposto no princípio Dependency Inversion Principle por meio de contêineres de injeção de dependência.

A Injeção de Dependência é um padrão de projeto que visa o controle de uma classe que seja inserida como dependência de uma outra classe, ou seja, procura remover dependências desnecessárias entre as classes. Desta forma, lembramos que o princípio da Inversão de Controle rege que nossas classes dependam de uma abstração e não de uma implementação. Veja um exemplo de código muito acoplado pela figura 67.

```
public class PedidosController : Controller
{
    public ActionResult Index()
    {
        var pedidoRepositorio = new PedidoRepository();
        var pedidos = pedidoRepositorio.ObterTodos();
        return View(pedidos);
    }
}
```

Figure 67: Exemplo de código muito acoplado.

Podemos evitar este alto acoplamento por meio da Injeção de Dependência. Em nosso projeto ASP.NET MVC, utilizaremos as bibliotecas do Unity, precisaremos instalar via Nuget as seguintes referências:

1. Unity: <https://www.nuget.org/packages/Unity/>;
2. Unity.Mvc: <https://www.nuget.org/packages/Unity.Mvc/>.

Após a instalação dos pacotes do Unity, vejamos que duas classes são adicionadas em nosso projeto. A classe UnityMvcActivator configura a injeção de dependência em nosso projeto, esta classe precisa ser inicializada por meio do método Start(), faremos esta inicialização na classe Global.asax, por meio do método Application\_Start() adicionaremos o código: UnityMvcActivator.Start();.

A outra classe adicionada, é a UnityConfig, ela é responsável por configurar os objetos que serão controlados pelo Unity, ou seja, os objetos que serão injetados em nossas classes. Desse modo, não precisaremos tomar cuidado para instanciar o Contexto do EF, carregar o repositório e todo o resto, o próprio Unity ficará responsável por gerenciar nossos objetos.

A listagem a seguir apresenta o modo de referenciar classes para injeção.

```
style=sharpc    public static class UnityConfig    public static void
RegisterTypes(IUnityContainer                                     container)
container.RegisterType<LojaVirtualContexto,                      LojaVirtualContexto>(new
HierarchicalLifetimeManager());           container.RegisterType<IPessoaRepositorio,
PessoaRepositorio>(new HierarchicalLifetimeManager());
```

```
public class PessoasController : Controller
{
    private readonly IPessoaRepositorio _pessoaRepositorio;

    0 references | 0 changes | 0 authors, 0 changes
    public PessoasController(IPessoaRepositorio pessoaRepositorio)
    {
        _pessoaRepositorio = pessoaRepositorio;
    }

    0 references | 0 changes | 0 authors, 0 changes
    public ActionResult Index()
    {
        var pessoas = _pessoaRepositorio.GetAll();
        return View();
    }

    [HttpPost]
    0 references | 0 changes | 0 authors, 0 changes
    public ActionResult Create(Pessoa pessoa)
    {
        _pessoaRepositorio.Adicionar(pessoa);
        _pessoaRepositorio.SalvarTodos();
        return View("Index");
    }
}
```

Figure 68: Exemplo de código utilizando repositório de Pessoa por meio da injeção de dependência via configuração do Unity.

Após finalizada todas as configurações necessárias, poderemos carregar nosso repositório injetado no construtor da nossa Controller e utilizá-lo em cada Action, conforme apresenta a figura 68. Em nenhum momento precisamos instanciar nosso contexto, o repositório padrão e nem o nosso próprio repositório de Pessoas, desse modo, deixamos nosso código menos acoplado, reduzimos a quantidade de código, facilitamos a manutenção e ganhamos tempo de desenvolvimento.

O uso do padrão repositório aplicado em nossas entidades será detalhado na seção 2.2.2, porém aproveitamos o assunto de Injeção de Dependência para aplicar em um exemplo que será utilizado em nosso projeto Loja Virtual.

Para maior aprofundamento sobre este padrão de projeto, recomenda-se a leitura dos seguintes links:

- [http://www.macoratti.net/11/07/ioc\\_di1.htm](http://www.macoratti.net/11/07/ioc_di1.htm);
- <https://www.devmedia.com.br/design-patterns-injecao-de-dependencia-com-csharp/23671>;
- <https://martinfowler.com/articles/injection.html>.

## 2.2 Camada de acesso a dados (Data Access Layer)

A camada de acesso a dados, também chamada por Data Access Layer (DAL), é responsável pela persistência dos dados, ela permite separar regras de negócio das regras de acesso a banco de dados. Em nossa aplicação, ficarão organizadas nesta camada todas as funcionalidades de banco de dados, como toda a estrutura utilizada pelo Entity Framework, ou seja, classes de mapeamento da entidade, contexto e atualizações do banco, além da classe para obter as conexões ao banco ou para executar comandos SQL, e repositórios de dados.

No Entity Framework, há três métodos de trabalho que valem a pena ser apresentadas:

- Database First<sup>15</sup>: neste modo, você já possui um banco de dados e a partir dele é gerado as entidades, tem auxílio de um recurso visual por meio do Visual Studio, utilizamos o arquivo .EDMX;
- Model First<sup>16</sup>: modelamos nossa base de dados de forma visual por meio dos recursos da própria IDE Visual Studio, utilizamos o arquivo .EDMX integrado;
- Code First<sup>17</sup>: primeiramente, desenvolvemos as entidades, ou seja, nossas

---

<sup>15</sup> <https://docs.microsoft.com/pt-br/aspnet/mvc/overview/getting-started/database-first-development/setting-up-database>

<sup>16</sup> <https://www.c-sharpcorner.com/UploadFile/4b0136/model-first-approach-in-Asp-Net-mvc-5/>

<sup>17</sup> <https://docs.microsoft.com/pt-br/aspnet/mvc/overview/getting-started/getting-started-with-ef-using-mvc/creating-an-entity>

classes de domínio, e o contexto, depois criamos nosso banco por meio de Migrations.

Veja algumas dicas para você escolher qual o melhor modelo de trabalho para suas necessidades, acesse:

<http://www.entityframeworktutorial.net/choosing-development-approach-with-entity-framework.aspx>.

Quando trabalhamos com Code First, há também modos de trabalharmos com o mapeamento de nossas entidades, são elas:

- Data Annotations<sup>18</sup>: é usado atributos simples chamados DataAnnotations, as configurações são feitas na própria entidade, atribuindo validações em suas propriedades. DataAnnotations também são compreendidas por um número de aplicativos .NET, como o ASP.NET MVC, que permite aos aplicativos aproveitar as mesmas anotações para validações do lado do cliente;
- Fluent API<sup>19</sup>: utilizado para configurar as classes de domínio para substituir as convenções. Desse modo, deixamos nossa entidade mais limpa, diferentemente do modo DataAnnotations.

Recomendo a leitura dos seguintes links, eles detalham as configurações das entidades por meio de Data Annotation e por Fluent API, respectivamente:

- <http://www.entityframeworktutorial.net/code-first/dataannotation-in-code-first.aspx>
- <http://www.entityframeworktutorial.net/code-first/fluent-api-in-code-first.aspx>

Em nosso projeto, utilizaremos o modo Code First do EF e as configurações das entidades serão realizadas por meio de Fluent API. Utilizaremos Migrations para trabalhar com o banco de dados, este assunto será detalhado na seção seguinte.

### 2.2.1 Migrations

Com Entity Framework trabalhando no modelo Code First Migration, temos a possibilidade do versionamento da estrutura do banco de dados, com este recurso é possível visualizar todas as atualizações realizadas na base.

Pelo Migration podemos realizar qualquer alteração em nossas entidades, como por exemplo, criar uma tabela, criar associações entre outras alterações, e com esta técnica conseguimos refletir estas atualizações no banco de dados sem perder os registros e sem ter que remover o banco.

Para usarmos Migrations, precisaremos executar os seguintes comandos “Package Manager Console” do Visual Studio, lembre-se de definir como projeto inicial

---

framework-data-model-for-an-asp-net-mvc-application

<sup>18</sup> <https://docs.microsoft.com/pt-br/ef/ef6/modeling/code-first/data-annotations>

<sup>19</sup> <https://docs.microsoft.com/pt-br/ef/ef6/modeling/code-first/fluent/types-and-properties>

a aplicação que estiver configurada com a conexão do banco de dados, que no caso do nosso exemplo será o projeto web “LojaVirtual.Web” (acesse as propriedades e clique em “Set as StartUp Project” (1)), e no “Package Manager Console” lembre-se de setar como “Default Project” (2) o projeto responsável por acesso ao banco de dados, que no caso será a biblioteca “LojaVirtual.DAL”, a figura 69 apresenta este passo inicial para usarmos a técnica de Migrations.

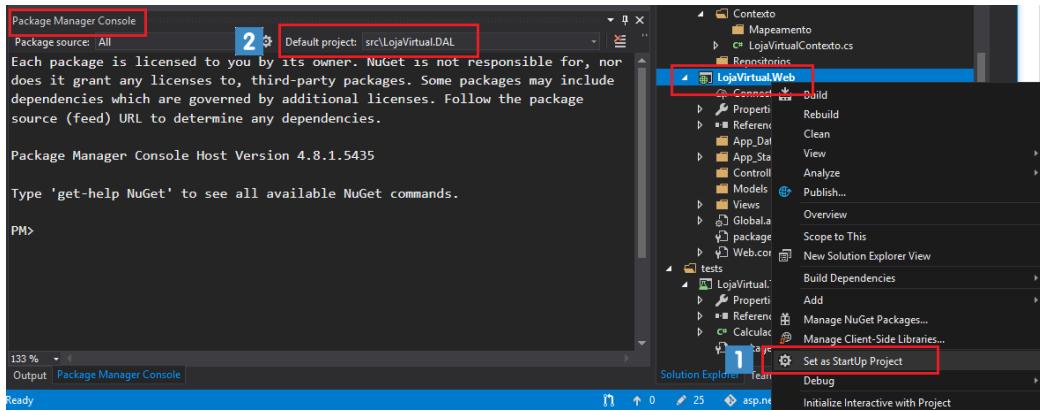


Figure 69: Executar comandos para Migrations.

No “Package Manager Console” utilizaremos os seguintes comandos para trabalharmos com Migrations em nosso projeto Loja Virtual:

- Enable-Migrations: permite o uso de Migrations no projeto, cria automaticamente uma classe de configuração (LojaVirtual.DAL > Migrations > Configuration.cs);
- Add-Migration: cria uma classe de Migrations, aqui ficam registradas as atualizações que serão realizadas no banco de dados, por meio dos métodos Up() e Down(). Todos estas classes estão localizadas em: LojaVirtual.DAL > Migrations, com elas temos um histórico de todas as alterações já realizadas. Neste comando precisamos passar um nome para a classe de Migrations, por exemplo, add-migration InicializarBancoDeDados;
- Update-Database: executa as classes de Migrations criadas pelo comando Add-Migration e aplica as alterações no banco de dados.

O Migrations permite que nós possamos voltar a uma atualização específica de Migration já realizada, é por este motivo que quando criamos nossa classe de Migration, ela constrói os métodos Up() e Down(), veja que o método Up() apresenta as atualizações que serão realizadas no banco de dados, enquanto que o método Down() desfaz todas as atualizações feitas naquele Migration. Desse modo, conseguimos voltar versões do banco de dados, este histórico também é importante para sabermos o momento que cada alteração foi realizada.

No site da Microsoft encontra-se um artigo que apresenta como utilizar Migration em uma aplicação ASP.NET MVC 5 usando o Entity Framework 6 Code First e o Visual Studio, acesse:

<https://docs.microsoft.com/pt-br/aspnet/mvc/overview/getting-started/introduction/>

[started/getting-started-with-ef-using-mvc/migrations-and-deployment-with-the-entity-framework-in-an-asp-net-mvc-application.](http://www.asp.net/mvc/tutorials/getting-started-with-ef-using-mvc/migrations-and-deployment-with-the-entity-framework-in-an-asp-net-mvc-application)

Além da documentação oficial encontrada no site da Microsoft, recomenda-se a leitura rápida do seguinte tutorial:

<http://www.entityframeworktutorial.net/code-first/code-based-migration-in-code-first.aspx>.

### 2.2.2 Padrão repositório (Repository Pattern)

No projeto Blog Pessoal, toda a regra de negócio e a camada de acesso ao banco de dados estavam no Controller, esse jeito de trabalhar não é muito interessante, pois não há a separação de responsabilidades das classes. Desse modo, fica difícil de realizarmos testes de unidades eficazes.

Nesse contexto, iremos utilizar o padrão repositório para nos auxiliar no desenvolvimento. Criaremos uma interface com métodos genéricos que serão utilizados por qualquer repositório de entidade (LojaVirtual.BLL > \_Base > IRepositoryBase.cs), após implementaremos esta interface em uma classe padrão abstrata (LojaVirtual.DAL > \_Base > RepositorioBase.cs) que será herdada para os repositórios mais específicos, acompanhe a listagem do código fonte a seguir.

```
style=sharpc public interface IRepositoryBase<TEntity> where TEntity : class
IQueryable<TEntity> GetAll(); IQueryable<TEntity> Get(Func<TEntity, bool> predicate);
TEntity Find(params object[] key); void Atualizar(TEntity obj); void SalvarTodos(); void
Adicionar(TEntity obj); void Excluir(Func<TEntity, bool> predicate);

style=sharpc public abstract class RepositorioBase<TEntity> : IDisposable,
IRepositoryBase<TEntity> where TEntity : class private readonly LojaVirtualContexto
_ctx = new LojaVirtualContexto();
public IQueryable<TEntity> GetAll() return _ctx.Set<TEntity>();
public IQueryable<TEntity> Get(Func<TEntity, bool> predicate) return
GetAll().Where(predicate).AsQueryable();
public TEntity Find(params object[] key) return _ctx.Set<TEntity>().Find(key);
public void Atualizar(TEntity obj) _ctx.Entry(obj).State = EntityState.Modified;
public void SalvarTodos() _ctx.SaveChanges();
public void Adicionar(TEntity obj) _ctx.Set<TEntity>().Add(obj);
public void Excluir(Func<TEntity, bool> predicate) _ctx.Set<TEntity>()
.Where(predicate).ToList() .ForEach(del => _ctx.Set<TEntity>().Remove(del));
public void Dispose() _ctx.Dispose();
```

Após criarmos nosso repositório padrão, iremos implementar um repositório específico para o nosso domínio Cliente. Primeiramente criaremos uma interface para este repositório de Cliente (LojaVirtual.BLL > Clientes > IClienteRepositorio.cs), após implementaremos nossa classe de repositório de Cliente (LojaVirtual.DAL > Repositorios > ClienteRepositorio.cs), não podemos esquecer de utilizar a herança das classes de padrão IRepositoryBase e RepositorioBase, respectivamente, e informar o domínio que será trabalhado. Veja listagem a seguir que demonstra a implementação deste repositório para Cliente, além de adicionar um método específico para o Clientes apenas.

```
style=sharpc public interface IClienteRepositorio : IRepositoryBase<Cliente>
void AuditarAcesso();
style=sharpc public class ClienteRepositorio : RepositoryBase<Cliente>,
IClienteRepositorio public void AuditarAcesso() Debug.WriteLine("Algo de Cliente");
```

Há várias maneiras de implementar um repositório padrão, você pode adicionar ou remover métodos desta classe, o importante é entendermos o tanto que esta classe no auxilia no desenvolvimento, além de organizar nosso código, facilitando manutenção e contribuindo com o reaproveitamento do código.

Por fim, fica disponível um artigo do site MSDN da Microsoft que explica a criação do padrão repositório com o Entity Framework:

<https://msdn.microsoft.com/pt-br/library/dn630213.aspx>

## 2.3 Camada de apresentação (User interface/Presentation)

A camada de apresentação, ou chamada apenas de interface, é a camada que interage diretamente com o usuário, por meio dela que são feitas as requisições como consultas, por exemplo.

### 2.3.1 Componentização

Como já vimos o ASP.NET MVC possibilita o trabalho com Partial Views, este recurso é muito interessante, conforme já vimos, porém há um modelo de trabalho em que este recurso pode ser potencializado.

A componentização de software pode ser descrita como uma unidade de software que pode ser unida a outras unidades de software para formar um sistema de maior tamanho [21]. Nesse contexto, os componentes são úteis pela capacidade de sua reutilização em outras páginas/aplicações.

Assim, recomenda-se que quando formos desenvolver nossas telas, caso exista a possibilidade, devemos dividi-lá em várias Partial Views que sejam de acordo com a separação de responsabilidades, desse modo, podemos ter um baixo acoplamento e aumentar o uso de componentes já desenvolvidos.

Um exemplo prático que pode ser visualizado no projeto, seria a criação de uma Partial View para visualização dos dados do endereço. Esta Partial View poderá ser utilizada no cadastro de Clientes, para exibição do endereços dele, e também poderá ser reutilizada no fluxo de fechamento do pedido de compra da nossa loja, apresentando as informações do endereço de entrega.

### 2.3.2 Listagem de registros

Há um Helper, chamado WebGrid, que nos auxilia na construção de tabelas. Ele já incorpora recursos de paginação, ordenação e alternar cor das linhas da tabela. Por meio da sintaxe Razor construiremos uma tabela com WebGrid em nosso projeto BlogPessoal.

Como exemplo, listaremos as categorias de artigo. Primeiramente, acessaremos nossa View Index (localizada em BlogPessoal.Web > Views > CategoriasDeArtigo > Index.cshtml), instanciaremos nossa classe WebGrid e carregaremos nossa lista de categorias por meio do método Bind(), conforme listagem

a seguir.

```
style=sharpc @ var grid = new WebGrid(); grid.Bind(Model);
```

Após, realizaremos as configurações do objeto WebGrid e adicionaremos as colunas da nossa tabela. O WebGrid apresenta várias configurações, entre elas a questão de aplicarmos estilos em nossa tabela. O código completo da listagem de categorias por meio do WebGrid é apresentado a seguir.

```
style=sharpc @grid.GetHtml( tableStyle: "webgrid", headerStyle: "webgrid-header", footerStyle: "webgrid-footer", alternatingRowStyle: "webgrid-alternating-row", selectedRowStyle: "webgrid-selected-row", rowStyle: "webgrid-row-style", mode: WebGridPagerModes.All, firstText: "<< Primeiro", previousText: "< Anterior", nextText: "Próximo >", lastText: "Último >>", columns: grid.Columns( grid.Column("Nome", header: "Nome"), grid.Column("Descrição", header: "Descrição"), grid.Column(style: "td-acao", header: "Ações", format: @<text>@Html.ActionLink("Editar", "Edit", new id = item.Id) <span>|</span>@Html.ActionLink("Remover", "Delete", new id = item.Id) </text>) ) )
```

A figura70 apresenta o resultado de uma listagem utilizando o WebGrid. Este recurso nos auxilia no desenvolvimento de tabelas mais elaboradas.

Para aprofundamento do assunto e conhecimento de outros recursos que o WebGrid nos possibilita, recomenda-se a leitura dos seguintes links:

- <https://docs.microsoft.com/en-us/dotnet/api/system.web.helpers.webgrid?view=aspnet-webpages-3.2>;
- <https://www.aspsnippet.com/Articles/WebGrid-Step-By-Step-Tutorial-with-example-in-ASPNet-MVC.aspx>.

Nome	Descrição	Ações
Doações	Doações	Editar   Remover
Eletroportáteis	Eletroportáteis	Editar   Remover
Ferramentas	Ferramentas	Editar   Remover
Floricultura	Floricultura	Editar   Remover
Informática	Informática	Editar   Remover
Livros	Livros	Editar   Remover
Natal	Natal	Editar   Remover
Papelaria	Papelaria	Editar   Remover
Páscoa	Páscoa	Editar   Remover
Prêmios Bônus Clube	Prêmios Bônus Clube	Editar   Remover

< Anterior 1 2 3 Próximo >

Figure 70: Listagem paginada de categorias utilizando o WebGrid.

### 2.3.3 Paginação de registros

A paginação de registros também é uma tarefa muito comum e necessária para aplicações web. Na seção anterior vimos a paginação de registro por meio do

WebGrid, porém vimos que a paginação acontece apenas na View para apresentação ao usuário. Dessa forma, não é indicado para uma tabela que tenha muitos registros, pois para estas tabelas maiores, devemos paginar os registros na Controller e retornar apenas os itens de uma determinada página.

Neste caso, podemos utilizar a biblioteca PagedList, disponível via Nuget pelo link <https://www.nuget.org/packages/PagedList/>. Para demonstrar este recurso, utilizaremos o projeto BlogPessoal, criaremos uma nova listagem de categorias que será desenvolvida por meio do PagedList.

Em nosso Controller CategoriasDeArtigoController, criaremos uma nova Action para listagem chamada de IndexPagedList e que aceitará como parâmetro opcional a página desejada para apresentação de registros, confira a implementação desta Action a seguir.

```
public class CategoriasDeArtigoController : Controller
{
    private const int TotalPorPagina = 5;

    public ActionResult IndexPagedList(int? pagina)
    {
        var lista = db.CategoriasDeArtigo
            .OrderBy(t => t.Nome)
            .ToPagedList(pagina ?? 1, TotalPorPagina);
        return View(lista);
    }
}
```

Após adicionarmos os parâmetros necessários para o controle PagedList, verificamos que o retorno da nossa Action foi modificado pelo método ToPagedList(paginaAtual, TotalPorPagina), este é o método responsável por implementar efetivamente a paginação dos registros.

Criaremos a View IndexPagedList que apresentará os registros paginados. Declararemos a classe Model derivada da Interface IPagedList, conforme apresenta o exemplo a seguir.

```
@using BlogPessoal.Web.Models.CategoriasDeArtigo
@model PagedList.IPagedList<CategoriaDeArtigo>
```

A construção da tabela pode ser elaborada igualmente já fizemos no início de nossas aulas, conforme exemplo a seguir.

```
<table class="table">
    <tr>
        <th>Nome</th>
        <th>Descrição</th>
        <th>Opções</th>
    </tr>
    @foreach (var item in Model) {
        <tr>
            <td>@item.Nome</td>
            <td>
                @Html.DisplayFor(t => item.Descrição)
                @Html.ActionLink("Editar", "Edit", new { id = item.Id })
                @Html.ActionLink("Remover", "Delete", new { id = item.Id })
            </td>
        </tr>
    }
</table>
```

O uso do PagedList nos dá propriedades para trabalharmos a paginação em nossa View. Com ele podemos verificar se há itens a serem apresentados em um botão próximo ou em um botão anterior, além de exibir informações como a quantidade de registros, quantidade de páginas por registro, página atual, entre outros. Algumas funcionalidades desta biblioteca são apresentadas em uma parte do código da View IndexPagedList, vista a seguir.

```
<div class="paginacao-centro" >
    @if (Model.HasPreviousPage)
        @Html.ActionLink("<< Primeiro", "IndexPagedList", new { pagina = 1 }, new { @class = "botao" })
    ...
    @if (Model.HasNextPage)
        @Html.ActionLink("Último >>", "IndexPagedList", new { pagina = Model.TotalPages }, new { @class = "botao" })
    ...
</div>
```

```

"espaco-botao" )
    @Html.ActionLink("< Anterior", "IndexPagedList", new pagina =
Model.PageNumber - 1 , new @class = "espaco-botao" )
    @if (Model.HasNextPage) @Html.ActionLink("Proximo >", "IndexPagedList",
new pagina = Model.PageNumber + 1 , new @class = "espaco-botao" )
    @Html.ActionLink("Ultimo >>", "IndexPagedList", new pagina =
Model.PageCount , new @class = "espaco-botao" ) </div>
<fieldset> <p> Registros encontrados: @Model.TotalItemCount </p> <p>
Pagina @(Model.PageCount < Model.PageNumber ? 0 : Model.PageNumber) de
@Model.PageCount </p> </fieldset>

```

A figura 71 apresenta o resultado da alteração realizada. Veja que logo abaixo da listagem de categorias, é apresentada a navegação entre páginas. No final, a página exibe a quantidade total de categorias em nosso banco de dados, e também informa a página atual e o total de páginas disponíveis para navegação.

Nome	Descrição	Opções
Cozinha	Utilidades de cozinha	<a href="#">Editar</a> <a href="#">Remover</a>
Doações	Doações	<a href="#">Editar</a> <a href="#">Remover</a>
Eletroportáteis	Eletroportáteis	<a href="#">Editar</a> <a href="#">Remover</a>
Entretenimento	Cinema	<a href="#">Editar</a> <a href="#">Remover</a>
Ferramentas	Ferramentas	<a href="#">Editar</a> <a href="#">Remover</a>

Figure 71: Listagem paginada de categorias utilizando o PagedList.

Com PagedList ganhamos tempo de desenvolvimento, pois ele implementa vários recursos automaticamente que podemos utilizar na implementação de nossas Views, além disso, nossa aplicação torna-se mais rápida, pois ela passa a retornar uma parte de registros ao invés de trazer a quantidade total. Por meio das propriedades retornadas, temos a possibilidade de customizar componentes sem qualquer limitação.

Para maiores detalhes, acesse o projeto do PagedList no GitHub, <https://github.com/troygoode/PagedList>.

### 2.3.4 DropDownList em Cascata com jQuery

Filtros em cascata é um requisito comum em aplicações, na qual refere-se a capacidade de filtrarmos os registros apresentados em um componente DropDownList com base no item selecionado pelo usuário em outro componente.

Por exemplo, no projeto Loja Virtual, o cadastro de endereço terá o seguinte comportamento: quando o usuário selecionar a Unidade Federativa (UF) desejada por meio do DropDownList, a aplicação deverá carregar os municípios associados àquela UF selecionada em outro DropDownList.

Vamos testar este recurso, as alterações em nosso projeto serão bem simples, precisaremos de algumas linhas de código em alguns Controllers e Views, além de um pouco de jQuery e JSON.

Primeiramente, aprenderemos como usar o atributo UIHint em aplicações ASP.NET MVC. Ele especifica o template que o Dynamic Data vai usar para exibir um campo de dados. Quando o atribuímos em uma propriedade da Model e ao usar o EditorFor do Razor em nossas Views, o framework ASP.NET MVC vai procurar o modelo especificado que definimos no atributo UIHint. A listagem a seguir demonstra o uso deste atributo em uma Model.

```
... [Display(Name = "UF", Description = "Unidade Federativa (UF)")] [UIHint("UFDropDownList")] public int UfId { get; set; } [Display(Name = "Município")] [UIHint("MunicipioDropDownList")] public int MunicipioId { get; set; } ...
```

Conforme demonstra o código, o framework ASP.NET MVC irá procurar os templates UFDropDownList e MunicipioDropDownList, estas Partial View deverão ser criadas em /Views/Shared/EditorTemplates. Algo importante a saber sobre UIHint, é que em Views poderemos utilizar os recursos EditorFor e/ou DisplayFor, para seu funcionamento correto, devemos saber os diretórios corretos que o framework irá procurar, são eles [11]:

- **EditorFor:**
  - /Views/Shared/EditorTemplates
  - /Views/Controller\_Name/EditorTemplates
- **DisplayFor:**
  - /Views/Shared/DisplayTemplates
  - /Views/Controller\_Name/DisplayTemplates

A seguir visualizaremos as implementações dos templates UFDropDownList para listagem das Unidades Federativas, e MunicipioDropDownList para listagem de Municípios, respectivamente.

```
@using LojaVirtual.BLL.Municipios.Dtos  
@ List<UFDto> ufs = ViewBag.Ufs; var selectUf = new SelectList(ufs, "Id", "Sigla");  
@Html.DropDownListFor(model => model.UfId, selectUf, "[Selecione]", new { @class = "form-control" })  
@using LojaVirtual.BLL.Municipios.Dtos
```

```

@ var municipios = ViewBag.Municipios as List<MunicipioDto>; var
selectMunicipio = new SelectList(municipios, "Id", "Nome");
@Html.DropDownListFor(model => model.selectMunicipio, "[Selecione]", new
@class = "form-control")

```

Podemos usar este recurso para definir que determinadas propriedades da nossa Model se comportará como um controle desenvolvido, na qual toda a sua lógica ficará definida em um único local, como por exemplo os DropDownList para Unidades Federativas e Municípios.

A implementação da View para utilizarmos este recurso, ficará conforme trecho de código a seguir.

```

style=sharpc      @using LojaVirtual.BLL.Pessoas.Enderecos.Dto  @model
EnderecoDto ... <div class="form-group"> @Html.LabelFor(model => model.UFId,
htmlAttributes: new  @class = "control-label col-md-2" ) <div class="col-md-10">
@Html.EditorFor(model => model.UFId, new htmlAttributes = new  @class = "form-
control" ) @Html.ValidationMessageFor(model => model.UFId, "", new  @class =
"text-danger" ) </div> </div> <div class="form-group"> @Html.LabelFor(model =>
model.MunicipioId, htmlAttributes: new  @class = "control-label col-md-2" ) <div
class="col-md-10">    @Html.EditorFor(model   => model.MunicipioId, new
htmlAttributes = new  @class = "form-control" ) @Html.ValidationMessageFor(model
=> model.MunicipioId, "", new  @class = "text-danger" )</div></div>

```

Precisaremos implementar a Action que irá receber o código identificador da Unidade Federativa e retornará as cidades pertencentes daquela UF. Esta Action será invocada a partir de uma função jQuery, e seu retorno será JSON por meio do JsonResult. Confira o código completo da Action a seguir.

```

style=sharpc      [HttpPost]  [AllowAnonymous]  public  JsonResult
ObterMunicipios(int ufId)  var lst = RetornarMunicipios(ufId); return Json(lst,
JsonRequestBehavior.AllowGet);

```

Por fim, implementaremos a função jQuery que fará a carga em cascata do DropDownList MunicipioID. Iremos implementá-la em nossa View no momento em que a página for carregada. O código implementado é bem simples e apresenta o básico de jQuery, a listagem a seguir demonstra esta função.

```

style=sharpc          (function()('#UFId').change(function () {
.ajax(method: 'POST', url: '/Pessoas/ObterMunicipios/
', dataType: 'json', data: ufId:(this).val(), cache: false, async: true, success: function
(data)      ('#MunicipioId').html('');('#MunicipioId').append('<option><
/option>').val(null).html("[Selecione]");.each(data,      function (index)
('#MunicipioId').append('<option></option>');
.val(data[index].Id).html(data[index].Nome)); ); );
}); 

```

Por fim, a figura 72 demonstra o funcionamento dos nossos controles. O DropDownList Cascade desenvolvido para seleção de UF que carrega os municípios.

# Create

Endereço

The screenshot shows a form with three dropdown menus. The first dropdown, labeled 'UF', has 'PR' selected. The second dropdown, labeled 'Município', has 'Londrina' selected from a list that includes '[Selezione]' and 'Maringá'. The third dropdown, labeled 'Nome', has 'Maringá' listed.

Figure 72: Apresentação do DropDown Cascade.

### 2.3.5 Máscaras de entrada de dados com jQuery

Uma necessidade comum no desenvolvimento de aplicações é o uso de máscaras para entrada de dados, desse modo, auxiliamos o usuário a imputar corretamente as informações no padrão que esperamos.

Aprenderemos a mascarar os campos dos nosso formulários por meio do uso de scripts jQuery para aplicar as máscaras de entrada. Para esta tarefa utilizaremos o plugin jQuery Masked Input, disponível pelo seguinte endereço:

<https://plugins.jquery.com/maskedinput/>

No projeto Loja Virtual, precisaremos referenciar este script em nossas páginas. Sem muito esforço, conseguiremos adicionar as máscaras nos campos desejados. Na listagem a seguir vimos a aplicação de máscara nos seguintes campos: CPF, Celular, CEP e Data de Nascimento.

```
<script type="text/javascript">
(function()('#CPF').mask('999.999.999.99');      ('#Celular').mask('(99)9999 - 9999? 9');('#CEP').mask('99999-999'); ", ErrorMessage = "O e-mail com formato errado.")] public string Email get; set;
[Display(Name = "Senha")] [Required] [StringLength(55, MinimumLength = 3)] [DataType(DataType.Password)] public string Senha get; set;
```

Implementaremos nossa Controller de Acesso e nesta classe adicionaremos nossa Action para autenticação do usuário. Esta Action deve ser enviado via POST, atributo [HttpPost], e com permissão de acesso anônimo, atributo [AllowAnonymous]. Primeiro devemos verificar se a Model está válida, após esta verificação pesquisaremos o autor que possua o e-mail e senha informados na View de Login. Caso encontre o autor, utilizaremos o recurso FormsAuthentication e setaremos as informações do autor logado via comando .SetAuthCookie(autorLogado.Email, true). Acompanhe o trecho de código a seguir.

```
[HttpPost] [AllowAnonymous] public ActionResult Entrar(Login usuario) if (!ModelState.IsValid) return View("Index");
```

```

var autorLogado = db.Autores.Where(t => t.Email.Equals(usuario.Email,
StringComparison.OrdinalIgnoreCase)
t.Senha.Equals(usuario.Senha)).FirstOrDefault(); if (autorLogado != null)
FormsAuthentication.SetAuthCookie(autorLogado.Email, true); return
RedirectToLocal(); return View("Index");

private ActionResult RedirectToLocal() if (TempData["ReturnUrl"] != null) var
returnUrl = TempData["ReturnUrl"] as string; if (Url.IsLocalUrl(returnUrl)) return
Redirect(returnUrl); return RedirectToAction("Index", "Home");

style=sharpc [AllowAnonymous] public ActionResult Index(string returnUrl) if
(Request.IsAuthenticated) return RedirectToAction("Index", "Home"); if
(!string.IsNullOrEmpty(returnUrl)) TempData["ReturnUrl"] = returnUrl; return View();

```

Configuraremos a autenticação no Web.config por meio do código a seguir. Esta configuração faz com que ao acessarmos páginas que necessitem de autorização, o usuário seja redirecionado a nossa View de Login. Além disso, ela atribui a rota que tentamos acessar por meio da query string returnUrl. Utilizaremos o recurso TempData para armazenar este valor ao acessar a Action Index.

```

style=sharpc ... <system.web> <authentication mode="Forms"> <forms
name=".ADAAuthCookie" loginUrl=" /Acesso/" timeout="45" slidingExpiration="false"
protection="All" cookieless="UseCookies" /></authentication> ...

```

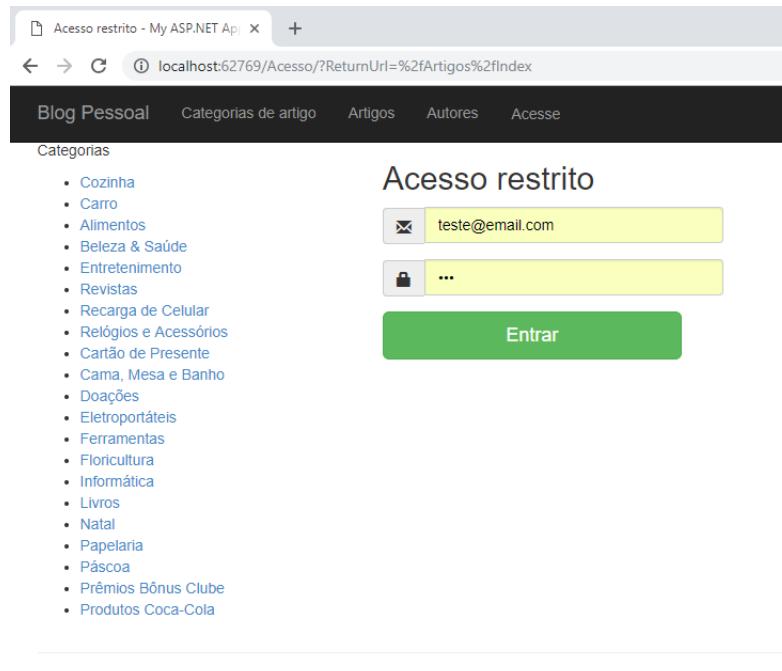


Figure 73: Tela de Login.

Após estas alterações, vimos que, por exemplo, ao tentar acessar a rota Artigos/Index do ArtigosController que possui o atributo [Authorize], recebemos como resposta a tela de Login, conforme apresenta a figura 73. Repare que a rota Artigos/Index está disponível por meio da query string returnUrl.

Após realizar a autenticação corretamente por meio do e-mail e senha de um autor disponível no banco de dados do projeto Blog Pessoal, a página para listagem de

Artigos é carregada. No menu, apresentamos o e-mail do autor logado no sistema e um botão para sair da aplicação. A Action responsável por sair do sistema segue o trecho de código a seguir, praticamente precisaremos chamar o comando FormsAuthentication.SignOut().

```
public ActionResult Sair() { FormsAuthentication.SignOut(); return RedirectToAction("Index");}
```

O processo de autenticação apresenta uma propriedade que pode ser utilizada quando precisamos verificar se o usuário está autenticado, o Request.IsAuthenticated retorna verdadeiro caso o usuário esteja autenticado na aplicação. Com este código podemos fazer várias validações internas e de controles, como por exemplo, apresentar o e-mail do usuário logado, conforme demonstra o trecho de código a seguir utilizado no menu do projeto Blog Pessoal.

O código @User.Identity.GetUserName() apresenta o login utilizado do autor que se autenticou no site, esta informação foi preenchida por meio do comando FormsAuthentication.SetAuthCookie(autorLogado.Email, true); durante o login da aplicação.

```
@using Microsoft.AspNet.Identity
@if (Request.IsAuthenticated)
<a class="page-scroll btn btn-outline" href="#" title="Acesso restrito"> Oi, @User.Identity.GetUserName(). </a> <a class="page-scroll btn btn-outline" href="@Url.Action("Sair", "Acesso")"> <i class="fa fa-power-off"></i> Sair </a> else
<a class="page-scroll btn btn-outline" href="@Url.Action("Index", "Acesso")" title="Cadastre-se em nosso site">Acesse </a>
```

Recomendo a seguinte leitura:

<https://www.codeproject.com/Articles/578374/AplusBeginner-splusTutorialplusonplusCustomplusF>

Este artigo detalha a implementação de uma autenticação de formulários personalizados em uma aplicação ASP.NET MVC.

Uma dica muito importante, caso tenha necessidade de salvar alguma informação sigilosa do usuário no sistema ou em algum banco de dados, é imprescindível que estas informações sejam criptografadas. Esta ação contribui muito com a segurança da informação em sua aplicação. É interessante que dados utilizados apenas pelo usuário, como a senha cadastrada para acesso ao sistema, devem ser utilizada criptografia de mão única, que seja difícil de desfazer.

### 2.3.6 Roles

O processo de autorização visto anteriormente é básico, pois basta que o usuário esteja autenticado para ter direito de acesso ao recurso controlado. O que precisamos agora é refinar esta autorização: limitar o acesso para um grupo de usuários. Essa limitação pode ser feita por meio de papéis (roles), que criamos e inserimos neles um grupo de usuários [2].

Há várias implementações de roles disponíveis para o ASP.NET MVC, desde mais simples que veremos a seguir, até aplicações mais complexas.

Primeiramente, no projeto Blog Pessoal, alteraremos nosso método de login em AcessoController para informarmos os respectivos roles do usuário logado, confira o trecho de código a seguir que demonstra nossa alteração.

```
style=sharpc [Authorize] public class AcessoController : Controller ...  
[HttpPost] [AllowAnonymous] [ValidateAntiForgeryToken] public ActionResult Entrar(Login usuario) if (!ModelState.IsValid) return View("Index");  
    var autorLogado = db.Autores.Where(t => t.Email.Equals(usuario.Email,  
StringComparison.OrdinalIgnoreCase)  
t.Senha.Equals(usuario.Senha)).FirstOrDefault(); if (autorLogado != null) var roles =  
autorLogado.Administrador ? "Admin" : "User";  
    var authTicket = new FormsAuthenticationTicket( 1, autorLogado.Email,  
DateTime.Now, DateTime.Now.AddDays(1), false, roles, "/"); var cookie = new  
HttpCookie(FormsAuthentication.FormsCookieName,  
FormsAuthentication.Encrypt(authTicket)); Response.Cookies.Add(cookie);  
    return RedirectToAction(); return View("Index"); ...
```

A segunda etapa é implementarmos, no arquivo Global.asax, a lógica para recriarmos o usuário com as roles, com base no conteúdo do cookie de autenticação. O código deve ser inserido no método Application\_AuthenticateRequest(), como pode ser visto na listagem adiante.

```
style=sharpc protected void Application_AuthenticateRequest(object sender,  
EventArgs e) var authCookie = Context.Request.Cookies[  
FormsAuthentication.FormsCookieName]; if (authCookie == null) return; var authTicket =  
FormsAuthentication.Decrypt(authCookie.Value); var roles =  
authTicket.UserData.Split(",").ToArray(); var userPrincipal = new  
GenericPrincipal(new GenericIdentity( authTicket.Name), roles); Context.User =  
userPrincipal;
```

Por fim, no projeto Blog Pessoal, permitiremos que apenas autores que possuem a flag administrador ativa poderão acessar a gestão de Autores, para realizar esta configuração precisaremos acrescentar os Roles solicitados pelo atributo Authorize em AutoresController, o comando que deve ser chamado é: [Authorize(Roles = "Admin")], acompanhe o trecho de código a seguir.

```
style=sharpc [Authorize(Roles = "Admin")] public class AutoresController :  
Controller ...
```

Desse modo, apenas usuários com Roles de Admin poderão acessar as Actions do Controller de Autores. Realize estas alterações e teste sua implementação, autentique usuário admin e depois tente acessar com um autor que não possua a flag administrador ativa.

O ASP.NET Identity é um sistema de associação para a construção de aplicações web, para smartphones, aplicações híbridas, etc., usando as identidades sociais para autenticação e autorização. Desse modo, poderemos utilizar o Windows Live (Hotmail, por exemplo), Gmail, Facebook e Twitter para autenticação antes que o usuário comece a usar nossa aplicação web. Recomendo a leitura do seguinte artigo para

aprofundamento sobre este assunto:

[http://www.macoratti.net/15/10/mvc\\_roles1.htm](http://www.macoratti.net/15/10/mvc_roles1.htm)

Quando formos trabalhar com esta tecnologia, precisaremos referenciar as bibliotecas:

- Microsoft.AspNet.Identity;
- Microsoft.AspNet.Identity.Owin;
- Microsoft.AspNet.Identity.EntityFramework;
- Microsoft.Owin.Host.SystemWeb.

Recomendo a leitura da documentação oficial da Microsoft pelo seguinte endereço:

<https://docs.microsoft.com/pt-br/aspnet/identity/overview/getting-started/introduction-to-aspnet-identity>

Ainda falando de ASP.NET Identity, que é o novo componente de Membership da Microsoft, recomendo a visualização deste tutorial bem completo sobre o assunto, acesse por meio do link [https://www.youtube.com/watch?v=PKop7c\\_OdR0](https://www.youtube.com/watch?v=PKop7c_OdR0), este vídeo foi elaborado pelo MVP Eduardo Pires<sup>20</sup>.

## 2.4 ValidateAntiForgeryToken

Nossa aplicação web pode sofrer com ataques de solicitação intersite forjada, ou também conhecido como Cross Site Request Forgery (CSRF). O CSRF é um problema de segurança relacionado com a possibilidade de que outras aplicações possam realizar requisições em nosso site, dessa forma, ficamos expostos a problemas como ataque de um clique (one-click attack) ou sequestro de sessão (session riding).

Com a finalidade de evitar este problema, utilizaremos o `Html.AntiForgeryToken()` em nossos formulários. Ele protege nossa aplicação da falha relacionada ao CSRF, pois este recurso gera um cookie com os códigos de AntiForgeryToken e insere o código mais recente como campo oculto na View.

As alterações em nosso código são muito simples, basta adicionarmos o seguinte atributo `[ValidateAntiForgeryToken]` na Action que receberá uma requisição HTTP por meio do método POST. O trecho de código a seguir demonstra a implementação deste recurso para realizar o login no projeto Blog Pessoal.

```
style=sharpc [HttpPost] [AllowAnonymous] [ValidateAntiForgeryToken] public  
ActionResult Entrar(Login usuario) if (!ModelState.IsValid) return View("Index"); ...
```

Depois faremos o uso do `Html.AntiForgeryToken()` em nossa View de login, conforme demonstra listagem a seguir.

```
style=sharpc @using (Html.BeginForm("Entrar", "Acesso", new { ReturnUrl =  
ViewBag.ReturnUrl }, FormMethod.Post, new { id = "frmLogin", data_toggle =
```

---

<sup>20</sup> <http://www.eduardopires.net.br/2014/08/asp-net-identity-tutorial-completo/>

```
"validator", role = "form", defaultbutton = "entrar" )) @Html.AntiForgeryToken()  
@Html.ValidationSummary(true) <div class="form-group"> <div class="input-group">  
...
```

Desse modo, acabamos de proteger nossa requisição de realizar login do projeto Blog Pessoal, com esta implementação garantimos que o login será feito pelo formulário disponível em nosso site, e não por uma requisição enviada por outra aplicação.

Há momentos que nós não precisaremos adicionar este recurso de segurança, por exemplo, quando precisarmos que outras aplicações façam requisição em nossa aplicação, assim, recomendamos que este recurso seja implementado em seus formulários sempre que possível.

Para maiores detalhes e aprofundamento sobre o assunto, acesse a documentação oficial deste recurso no próprio site da Microsoft:

<https://docs.microsoft.com/en-us/dotnet/api/system.web.mvc.htmlhelper.antiforgerytoken?view=aspnet-mvc-5.2>

## 2.5 Configurações no Web.config

O arquivo Web.config localizado na raiz do projeto Web é responsável pela configuração de vários recursos disponíveis na plataforma ASP.NET, sendo eles: métodos de autenticação variados, monitoramento de performance e escalabilidade, além de podermos configurar nossos próprios recursos criados.

O Web.config é uma orientação para determinada aplicação ASP.NET ser configurada. Não se engane, a extensão .config é apenas uma nomenclatura, e ele é somente um arquivo XML que pode ser alterado em qualquer editor de textos, bastando que se conheça as tags a serem usadas [3].

Dessa modo, podemos realizar algumas recomendações, atualmente é importante sabermos do uso de Hyper Text Transfer Protocol Secure (HTTPS), que é a versão segura do HTTP, em nossas aplicações, ainda mais quando trabalhamos com dados sensíveis de usuários, compras e vendas pela internet, isto que um site rodando em HTTPS ajuda na indexação do seu site na pesquisa orgânica dos diversos buscadores.

HTTPS é o protocolo sobre qual os dados são enviados entre um navegador e o site que está conectado, isto significa que todas as comunicações entre eles são criptografadas por meio de Secure Sockets Layer (SSL), ou pelo formulário mais evoluído, Transport Layer Security (TLS).

Usar HTTPS nas aplicação depende da aquisição de um certificado HTTPS e a configuração deste no servidor onde será hospedado a aplicação web. Desse modo, não iremos entrar em maiores detalhes sobre este assunto neste capítulo, porém deixarei um artigo interessante que fala sobre a importância HTTPS: <https://belodigital.com/seguranca/importancia-do-https/>.

Em nossa aplicação, após configurarmos o HTTPS no servidor, não devemos permitir que o acesso seja realizado sem HTTPS ou pelo menos devemos garantir que páginas mais sensíveis sejam acessadas via HTTPS, como tela de login, página de compras, páginas de informações do usuário entre outras. Para deixar que o site seja acessado via HTTPS apenas, há uma configuração que podemos informar no arquivo

Web.config da aplicação web, conforme listagem a seguir.

```
style=sharpc ... <system.webServer> <rewrite> <rules> <rule name="Redirect to https" stopProcessing="true"> <match url="(.*)" /> <conditions> <add input="HTTPS" pattern="off" ignoreCase="true" /> </conditions> <action type="Redirect" url="https://HTTP_HOSTREQUEST_URI" redirectType="Permanent" appendQueryString="false" /> </rule> ...
```

Por meio desta simples configuração todas as requisições que não estiverem em HTTPS são redirecionadas, lembrando que esta configuração depende de um recurso para URL Rewrite instalado no servidor da aplicação e também a aquisição de um certificado SSL. Deixarei alguns links listados a seguir para aprofundamento do assunto:

- <https://configr.com/blog/a-importancia-de-se-ter-um-site-com-https-ssl/>;
- <https://www.pedrodias.net/webmaster/migrar-para-https>.

## 2.6 Controle de transação por meio do Unit Of Work

O Unit Of Work ou Unidade de Trabalho é um padrão de projeto responsável por controlar o que fazemos durante uma transação, geralmente é executada sobre uma regra de negócio e coordena as alterações para a camada de acesso a banco de dados.

```
style=sharpc public interface IUnitOfWork<TContext> : IDisposable where TContext : DbContext void AbrirTransacao(); void Commit(); void Rollback();
```

```
style=sharpc     public class UnitOfWork<TContext> : IDisposable, IUnitOfWork<TContext> where TContext : DbContext private readonly TContext _contexto; private DbContextTransaction _transaction;
```

```
public UnitOfWork(TContext contexto) _contexto = contexto;
```

```
public void AbrirTransacao() _transaction = _contexto.Database.BeginTransaction();
```

```
public void Commit() _transaction.Commit();
```

```
public void Rollback() _transaction.Rollback();
```

```
public void Dispose() _contexto.Dispose();
```

Importante dizer que utilizando EF evitamos problemas com SQL Injection.

## 2.7 Outros assuntos de segurança

Há muitas outras técnicas de segurança que devemos implementar em nossa aplicação.

Recomendo a leitura deste link: <https://bit.ly/2SR9Eon>, neste artigo são listados 10 pontos de segurança para uma aplicação ASP.NET MVC. Entre elas, recomendo que você implementem o código que remove informações do servidor enviadas no cabeçalho das requisições, dessa forma evitamos que pessoas mal intencionadas

tenham acesso a informações do servidor.

Continuando neste ponto de não exibir informações, é imprescindível que seja configurado o tratamento de erro, pois devemos evitar que página de erro do servidor seja apresentada ao usuário, além de que aquelas informações podem expor certos dados que na mão de pessoas mal intencionadas, sejam prejudiciais a nossa aplicação.

### 3. Desempenho da aplicação

Atualmente, nós queremos acessar as informações o mais rápido possível, por vezes não temos muita paciência para esperar o carregamento de uma página que está demorando muito, salvo se for algo muito específico e exclusivo para gente.

Neste cenário, precisamos criar aplicações que tenham desempenho alto e que consumam o menor recurso possível, pois vejamos que o acesso de aplicação via mobile está aumentando cada vez mais, desse modo, além de desenvolvemos um site que seja responsivo, ele precisa apresentar informações de maneira rápida.

Assim, as próximas seções apresentarão técnicas, tecnologias e ferramentas que nos ajudarão a melhorar o desempenho de nossas aplicações.

#### 3.1 Compactação da requisição com a compressão GZIP e Deflate

Reducir o tamanho de uma requisição HTTP do servidor para o navegador do cliente está relacionado com desempenho da nossa aplicação.

Uma maneira eficaz e muito simples de fazer isso é usando a compressão pelo componente GZIP que está disponível não só para ASP.NET, mas em várias outras linguagens. Esta tecnologia é a mais fácil de aplicar para reduzir o peso das páginas e a que tem o maior impacto, chegando a comprimir o tamanho da resposta em até 85% [25].

Um exemplo de implementação desta compactação será realizada no projeto Blog Pessoal. Primeiramente acessaremos a classe Global.asax, implementaremos o método Application\_BeginRequest<sup>21</sup>, este método consiste em manipulador de eventos, ele é executado em todas as solicitações tratadas em tempo de execução do ASP.NET.

Como o Application\_BeginRequest é usado automaticamente quando uma solicitação é recebida, implementaremos a compactação da requisição por meio da compressão GZIP e Deflate no ASP.NET MVC. Acompanhe o trecho de código a seguir na qual demonstra a compactação configurada para todos os Controllers.

```
style=sharpc public class MvcApplication : System.Web.HttpApplication ...  
protected void Application_BeginRequest(object sender, EventArgs e) {  
    HttpApplication app = (HttpApplication)sender;  
  
    string encodings = app.Request.Headers.Get("Accept-Encoding"); if (encodings != null) encodings = encodings.ToLower(); if (encodings.Contains("deflate")) app.Response.Filter = new DeflateStream(app.Response.Filter, CompressionMode.Compress); app.Response.AppendHeader("Content-Encoding", "deflate"); else if (encodings.Contains("gzip")) app.Response.Filter = new GZipStream(app.Response.Filter, CompressionMode.Compress); app.Response.AppendHeader("Content-Encoding", "gzip");}
```

---

<sup>21</sup> <https://www.dotnetperls.com/application-beginrequest>

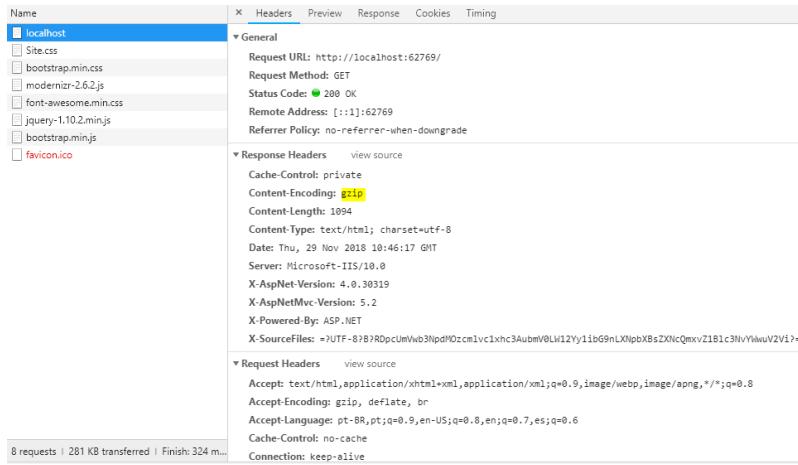


Figure 74: Detalhes da requisição compactada com compressão GZIP.

Após esta alteração conseguimos analisar algumas informações. Primeiramente houve mudança no cabeçalho de resposta da nossa requisição, o atributo Content-Encoding: gzip indica que os arquivos foram comprimidos, confira detalhes pela figura 74.

Aprendemos a configurar a compactação de todas as requisições do projeto, porém podemos optar pela compactação de uma Controller ou de uma Action em específico, desse modo criaremos um atributo chamado CompressFilterAttribute. Confira a implementação deste filtro pela listagem a seguir.

```
style=sharpc public class CompressFilterAttribute : ActionFilterAttribute public
override void OnActionExecuting(ActionExecutingContext filterContext)
HttpRequestBase request = filterContext.HttpContext.Request;
string acceptEncoding = request.Headers["Accept-Encoding"];
if (string.IsNullOrEmpty(acceptEncoding)) return;
acceptEncoding = acceptEncoding.ToUpperInvariant();
HttpResponseBase response = filterContext.HttpContext.Response;
if (acceptEncoding.Contains("GZIP")) response.AppendHeader("Content-
encoding", "gzip"); response.Filter = new GZipStream(response.Filter,
CompressionMode.Compress); else if (acceptEncoding.Contains("DEFLATE"))
response.AppendHeader("Content-encoding", "deflate"); response.Filter = new
DeflateStream(response.Filter, CompressionMode.Compress);
```

Para realizar a chamada deste atributo, basta adicionarmos o [CompressFilter] na Action ou no Controller para fazermos a compactação da requisição de maneira automática, simples e fácil. Ao implementá-lo no Controller, não há necessidade que este atributo seja replicado nas Actions daquela classe. Configura um exemplo a seguir encontrado no projeto Blog Pessoal.

```
style=sharpc [CompressFilter] public ActionResult Index() return View();
```

Por fim, vale apresentar uma comparação realizada entre o carregamento de uma página sem compressão e outra com a compactação GZIP configurada. Acompanhe esta comparação por meio das figuras 75 e 76.

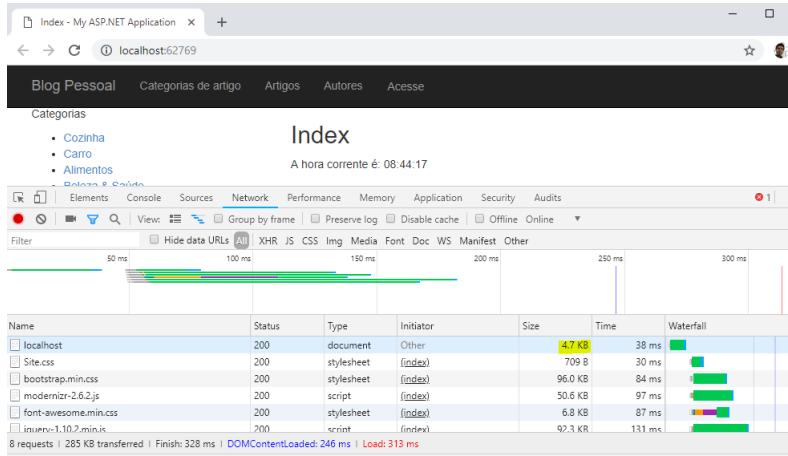


Figure 75: Tamanho da requisição sem compressão.

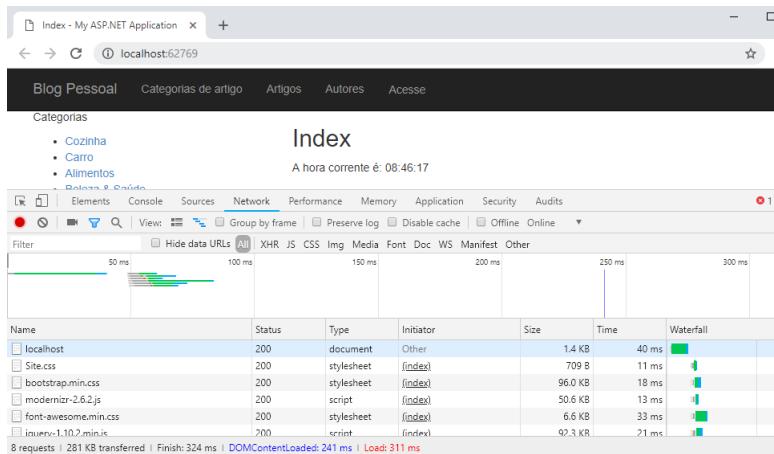


Figure 76: Tamanho da requisição com compressão GZIP configurada.

A figura 75 demonstra uma requisição realizada sem compressão, vejamos que o tamanho total retornado foi de 4.7 KB. Já, a figura 76 que apresenta uma requisição com compressão GZIP configurada, retornou o tamanho total de 1.4 KB, comprimindo o tamanho da resposta em aproximadamente 70%.

Existem diversas maneiras de configurar um site para que as páginas sejam enviadas comprimidas, assim como outros arquivos externos também. Vimos apenas algumas formas via código-fonte, há configurações que devem ser feitas nos servidores da aplicação.

Para uma leitura mais aprofundada, recomendo ler o seguinte artigo:  
<https://developers.google.com/speed/articles/gzip>.

### 3.2 Filtro para remoção de espaços

Com a finalidade de melhorar o desempenho de nossa aplicação, vamos conhecer o filtro para remoção de espaços do HTML. Primeiramente implementaremos a seguinte classe `WhitespaceFilterAttribute` no projeto Blog Pessoal, acompanhe o trecho a seguir.

```
style=sharpc public class WhitespaceFilterAttribute : ActionFilterAttribute
public override void OnActionExecuted (ActionExecutedContext filterContext) var
```

```

response = filterContext.HttpContext.Response;
if (filterContext.HttpContext.Request.RawUrl == "/sitemap.xml") return;
if (response.ContentType != "text/html" || response.Filter == null) return;
response.Filter = new HelperClass(response.Filter); ...

```

Os detalhes da classe HelperClass estão disponíveis no repositório do projeto Blog Pessoal no GitHub.

Utilizaremos o atributo [WhitespaceFilter] na Action Index do HomeController para realizarmos nossos testes, conforme listagem a seguir.

```
style=sharpc [WhitespaceFilter] public ActionResult Index() return View();
```

A aplicação deste filtro pode não ser o principal método que faça nossa aplicação ter um desempenho melhorado, porém é bom sabermos que todos os espaços gerados no HTML estão ocupando largura da banda.

Por fim, vale apresentar uma comparação realizada entre o carregamento de uma página sem o filtro de remoção de espaços e outra com o filtro configurado. Acompanhe esta comparação por meio das figuras 77 e 78.

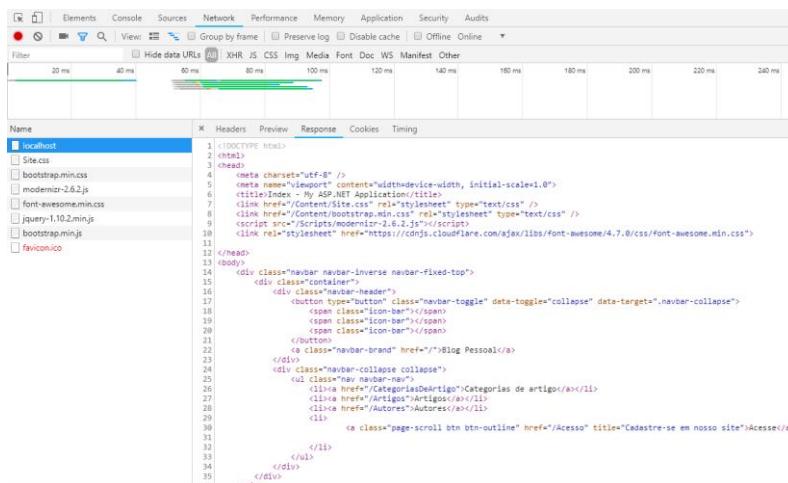


Figure 77: Exemplo de requisição sem filtro de remoção de espaços.

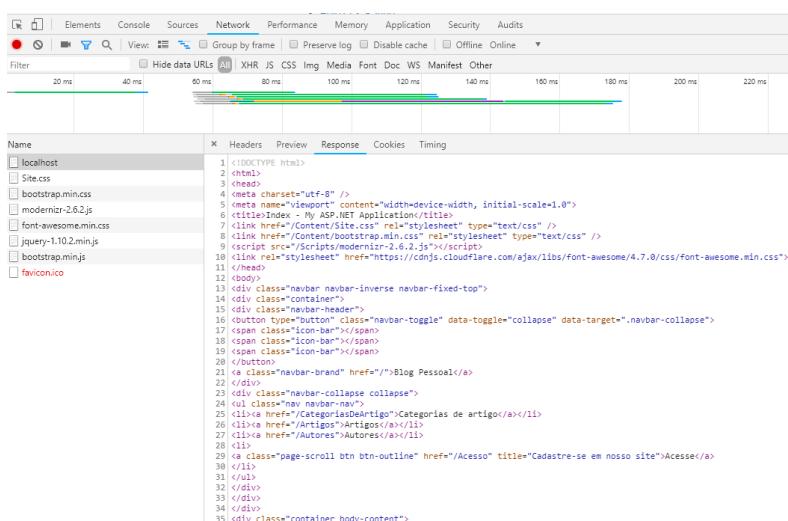


Figure 78: Exemplo de requisição com filtro de remoção de espaços configurado.

A figura 77 demonstra uma requisição realizada sem filtro de remoção de espaços, vejamos que o tamanho total retornado foi de 4.7 KB. Já, a figura 78 que apresenta uma requisição com filtro de remoção de espaços configurado, retornou o tamanho total de 3 KB, comprimindo o tamanho da resposta em aproximadamente quase 25%.

Uma dica especificamente sobre este filtro, quando você for aplicá-lo verifique se suas páginas funcionarão corretamente, pois já vi alguns caso que a aplicação de atributo inviabilizou a ação de plugin que era inicializado e também acarretou um problema num componente desenvolvido. Fora este ponto de atenção, recomendo que você sempre analise seu cenário e identifique o uso deste ou qualquer outro filtro, até mesmo a combinação de vários atributos para que nossa aplicação tenha um desempenho melhorado.

Qualquer filtro que formos utilizar é essencial que nós os testemos, a fim de verificar se o funcionamento da tela continua correto.

### 3.3 Bundle e Minification

Antes de iniciarmos, sabemos que há anos toda a parte de design dos sites ocorre por meio de arquivos CSS. Não basta ter um HTML perfeito, com tudo o que esta versão nos oferece, é fundamental um bom CSS para fazer o show a parte. Então, é comum num projeto de tamanho médio ter diversos arquivos CSS, a fim de customizar as páginas do site. Outro fator que todo desenvolvedor WEB deve saber, é que o uso de JavaScript, é cada vez mais usado. Aliás, é praticamente impossível hoje em dia uma aplicação WEB não usar JavaScript. Isto tudo porque o processamento do código será realizado no lado do cliente [10].

Neste sentido, imaginemos a quantidade de bytes trafegados entre o servidor e o cliente a cada solicitação? Será que há como diminuir esta quantidade, ou melhor ainda, colocar parte dos dados no cache do servidor? É neste cenário que iremos aplicar o recurso de empacotamento e minimização disponível no ASP.NET.

Primeiramente realizaremos a configuração do Bundle no projeto Blog Pessoal, criaremos a classe BundleConfig dentro na pasta App\_Start, implementaremos um método estático chamado RegisterBundles que recebe como parâmetro uma coleção de Bundles (BundleCollection), confira listagem a seguir. Importante lembrar que esta classe faz referência ao namespace System.Web.Optimization, encontrada via Nuget:

<https://www.nuget.org/packages/Microsoft.AspNet.Web.Optimization/>

```
style=sharpc    public    class    BundleConfig    public    static    void
RegisterBundles(BundleCollection    bundles)    bundles.Add(new    ScriptBundle(""
/bundles/jquery").Include(    "/Scripts/jquery-version.js"));
    bundles.Add(new    ScriptBundle("    /bundles/jqueryval").Include(    ""
/Scripts/jquery.validate*"));
    bundles.Add(new    ScriptBundle("    /bundles/modernizr").Include(    ""
/Scripts/modernizr-*"));
    bundles.Add(new    ScriptBundle("    /bundles/bootstrap").Include(    ""
/Scripts/bootstrap.js"));
    bundles.Add(new    StyleBundle("    /Content/css").Include(    ""
/Content/bootstrap.css", " /Content/site.css"));
```

Note que a cada linha é adicionado um elemento à coleção (bundle.Add) onde você pode referenciar um ScriptBundle ou um StyleBundle.

Nesta notação é preciso informar o nome virtual (" /bundles/jquery") que será criado em tempo de execução numa pasta chamada bundles abaixo do diretório raiz, seguido do método Include, onde forneceremos toda a lista de arquivos separados por vírgula. Esta estrutura vale tanto para script quanto style.

Outra notação a ser aprendida é o version, que significa qualquer versão. Por exemplo, em " /Scripts/jquery-version.js" significa que qualquer arquivo chamado jquery-1.10.2.js, jquery-1.10.2.intellisense.js ou jquery-1.10.2.min.js serão adicionados ao mesmo.

Precisaremos referenciar a classe BundleConfig no arquivo Global.asax.cs, iremos invocá-lo dentro do método chamado Application\_Start que é executado toda vez que a aplicação é iniciada. Confira listagem a seguir que há uma chamada para a classe BundleConfig.RegisterBundles passando a lista de BundleTables.Bundles.

```
style=sharpc public class MvcApplication : System.Web.HttpApplication
protected void Application_Start()           AreaRegistration.RegisterAllAreas();
FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
RouteConfig.RegisterRoutes(RouteTable.Routes);
BundleConfig.RegisterBundles(BundleTable.Bundles); ...
```

Agora iremos para execução do Bundle, como exemplo, alteraremos o arquivo \_Layout.cshtml (localizado em: BlogPessoal.Web > Views > Shared), substituiremos as referências das tags script e link, pelos Bundles configurados na classe BundleConfig. A figura 79 apresenta a substituição das tags (1, 3) para uso dos Bundles (2, 4) criados, por motivos educacionais deixamos as tags comentadas para vermos a representação de cada Bundle.

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8" />
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>@ViewBag.Title - Blog Pessoal</title>
7
8      1 @*<link href="~/Content/Site.css" rel="stylesheet" type="text/css" />
9      <link href="~/Content/bootstrap.min.css" rel="stylesheet" type="text/css" />
10     <script src="~/Scripts/modernizr-2.6.2.js"></script>*@
11
12     2 @Styles.Render("~/Content/css")
13     @Scripts.Render("~/bundles/modernizr")
14
15     <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/
16         awesome.min.css">
17     @RenderSection("StyleContent", required: false)
18 </head>
19 <body>
20     <div class="navbar navbar-inverse navbar-fixed-top">...</div>
21
22     <div class="container body-content">...</div>
23
24     3 @*<script src="~/Scripts/jquery-1.10.2.min.js"></script>
25     <script src="~/Scripts/bootstrap.min.js"></script>*@
26
27     4 @Scripts.Render("~/bundles/jquery")
28     @Scripts.Render("~/bundles/bootstrap")
29
30     @RenderSection("ScriptContent", required: false)
31 </body>
32 </html>

```

Figure 79: Utilizando Bundle no arquivo \_Layout.cshtml.

Uma dica importante, caso a View \_Layout.cshtml não reconheça os comandos Styles.Render e Scripts.Render, precisaremos adicionar a referência do namespace System.Web.Optimization no arquivo web.config localizado dentro as pasta Views, adicione a seguinte linha: <add namespace="System.Web.Optimization"/> dentro da tag “<system.web.webPages.razor>”.

Ao rodar o projeto Blog Pessoal podemos analisar a figura 80, vimos que são realizadas requisições separadas para o arquivos “site.css” e “bootstrap.css”, outra questão é que não foram carregados os arquivos Minified dos arquivos CSS e Javascript, conforme demonstra a figura 81.

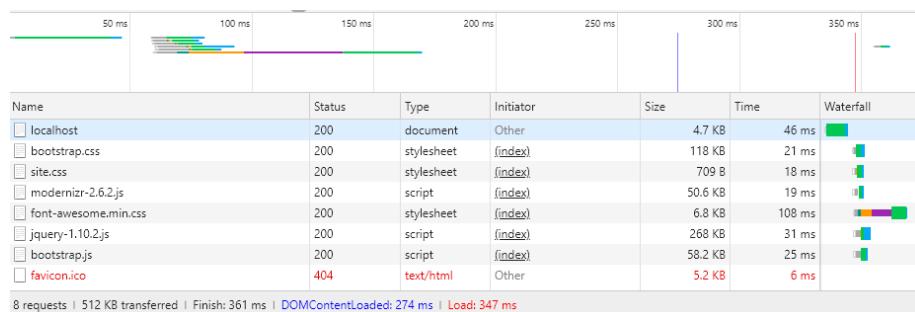


Figure 80: Requisições da página inicial do projeto Blog Pessoal.

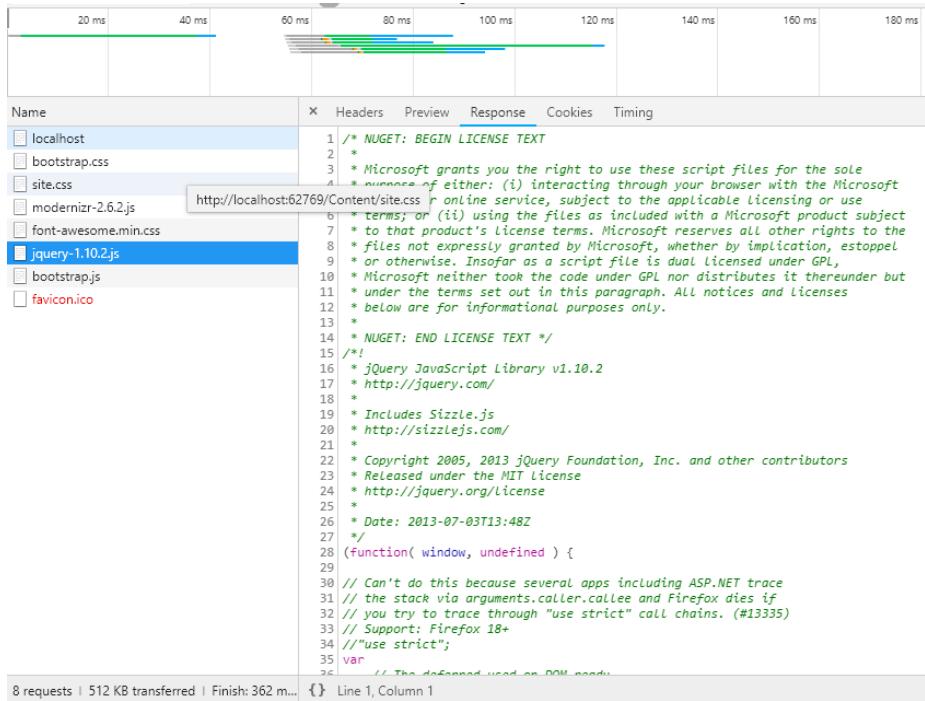


Figure 81: Arquivo jQuery obtido pela requisição da página inicial do Blog Pessoal.

Complicado carregar todos estes arquivos a cada requisição, certo? Mas infelizmente é assim, então, o que podemos fazer para melhorar a performance? Vamos ativar a otimização dos Bundles que desenvolvimentos. Primeiro, abriremos o arquivo Global.asax.cs e adicionaremos o comando `BundleTable.EnableOptimizations = true;` ao final do método `Application_Start`. Isto fará com que os arquivos sejam compactados e seja gerado um token para cada conjunto de Bundle declarado. Isto é o conceito de minification, use e abuse deste conceito [10].

```

style=sharpc public class MvcApplication : System.Web.HttpApplication
protected void Application_Start()           AreaRegistration.RegisterAllAreas();
FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
RouteConfig.RegisterRoutes(RouteTable.Routes);
BundleConfig.RegisterBundles(BundleTable.Bundles);
BundleTable.EnableOptimizations = true; ...

```

Execute novamente o projeto no navegador, use o F12 para rastrear a captura do tráfego, conforme a figura 82. Note que há uma grande diferença na quantidade de arquivos trafegados, temos uma requisição para css com os arquivos "site.css" e "bootstrap.css". Outra questão é que agora foram carregados os arquivos Minified dos arquivos CSS e Javascript, conforme demonstra a figura 83.

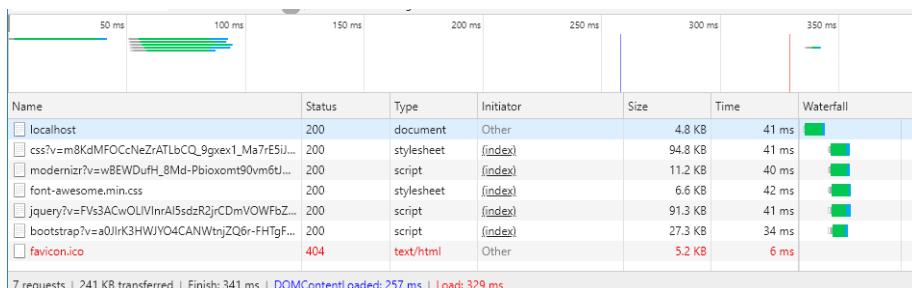


Figure 82: Requisições da página inicial do projeto Blog Pessoal com otimização ativada.

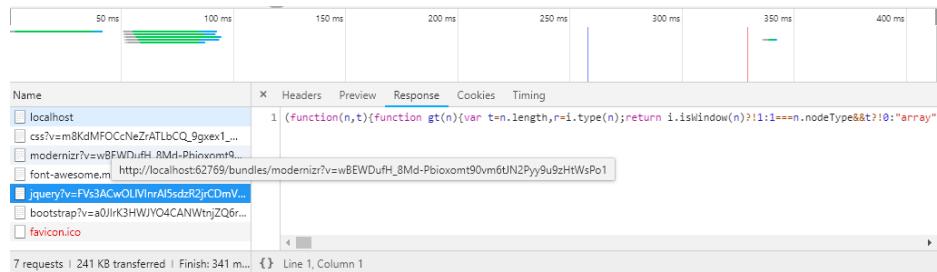


Figure 83: Arquivo jQuery Minified obtido pela requisição da página inicial do Blog Pessoal.

A principal vantagem do uso do bundle e minification é que o arquivo ficará no cache do servidor por um ano. Toda requisição que bater no servidor será comparada com o token gerado, caso tenha alguma alteração, o arquivo é enviado ao cliente [10].

Acesse a documentação oficial por meio do link:

<https://docs.microsoft.com/en-us/aspnet/mvc/overview/performance/bundling-and-minification>

### 3.4 Cache

Cache é uma otimização interna do servidor que salva algumas informações que são usadas com certa frequência pela aplicação. Ele é importante pois se soubermos configurá-lo corretamente, é possível carregarmos nosso site muito mais rápido para os clientes, além disso, gastaremos menos recurso do servidor, suportaremos muito mais visitas, evitando os picos de acesso se ficarmos off-line.

O uso de cache em nossas aplicações é muito importante, pois como aprendemos ele é responsável por melhorar o desempenho da nossa aplicação, torna ele mais rápida e faz como que a gente gaste menos recursos do servidor, porém devemos saber como usá-los, há requisições que poderemos utilizar e há outras que não conseguiremos utilizá-lo, tudo deve ser muito bem estudado e analisado para que este recurso não traga mais dores de cabeça do que solução para nossos problemas.

No ASP.NET MVC, há um atributo de filtro chamado `OutputCache`, com ele poderemos aplicar o recurso de cache de saída em formulários da web. O cache de saída permite armazenar em cache o conteúdo retornado por uma Action do Controller.

Para exemplificar o uso deste recurso, utilizaremos o projeto Blog Pessoal, assim adicionaremos cache no carregamento da página inicial do nosso site. Primeiramente configuraremos o `[OutputCache(Duration = 10, VaryByParam = "none")]` na Action Index do HomeController. O parâmetro “Duration” representa por quantos segundos o retorno da Action ficará armazenada em cache, já o parâmetro “VaryByParam” determina as variações de entrada, o valor “none” indica que o cache não irá variar com base em entradas de query strings ou formulários, já em compensação, o valor “\*” especifica que o cache irá variar com base em quaisquer valores definidos em `Request.QueryString` e `Request.Form`. Veremos exemplos desta implementação a seguir.

```
 [OutputCache(Duration = 10, VaryByParam = "none")] public  
 ActionResult Index() return View();
```

Na View Index (localizado em: View > Home) adicionaremos uma tag <div> que apresentará o horário corrente da requisição, com esta implementação conseguiremos ver o recurso de cache ação, pois depois da primeira vez que a View for carregada e nós solicitarmos novamente esta View, a requisição não será enviada para o servidor novamente caso seja feita entre os 10 segundos definidos no OutputCache como duração do cache.

Ainda no projeto Blog Pessoal, implementaremos cache na View responsável por exibir detalhes dos artigos, desse modo deixaremos nosso site mais rápido e preparado caso nossos artigos sejam acessados por muitas pessoas, evitando ao máximo que nossa aplicação fica off-line, e se caso fique pelo menos o conteúdo carregado fique disponível aos usuários durante um período.

Primeiramente, configuraremos o [OutputCache(Duration = 20, VaryByParam = "id")] na Action Details do ArtigosController. Veja que o parâmetro “VaryByParam” está com o valor “id”, ou seja, determina que serão armazenadas diferentes entradas em cache para cada combinação de valores constantes nos objetos Request.QueryString e Request.Form. Desse modo, ao requisitarmos o endereço <http://localhost:xxxx/Artigos/Details/1> ele ficará armazenador em cache durante 20 segundos, mesmo que nós acessemos outra página de detalhes do artigo e voltarmos para a primeira View de detalhe aberta, ela continuará em cache até que se passe os 20 segundos definidos.

```
 [AllowAnonymous] [OutputCache(Duration = 20, VaryByParam =  
 "id")] public ActionResult Details(int? id) if (id == null) return new  
 HttpStatusCodeResult(HttpStatusCode.BadRequest); Artigo artigo =  
 db.Artigos.Find(id); if (artigo == null) return HttpNotFound(); return View(artigo);
```

Vimos que o cache está ação quando exploramos detalhes das nossas requisições e vemos no cabeçalho de resposta o atributo Cache-Control: public, max-age=20, conforme exemplifica a figura 84.

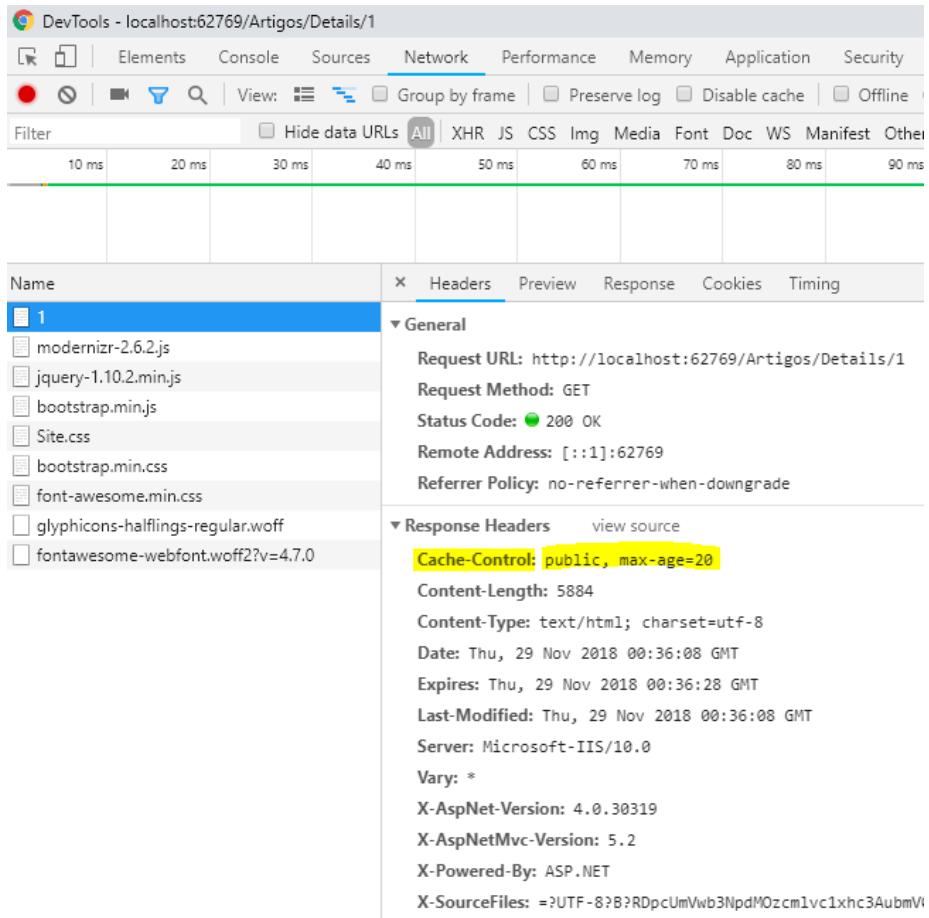


Figure 84: Detalhes da requisição com controle de cache.

Por fim, recomendo as seguinte leitura para maior aprofundamento sobre o assunto abordado nesta seção:

- [https://www.tutorialspoint.com/asp.net\\_mvc/asp.net\\_mvc\\_caching.htm](https://www.tutorialspoint.com/asp.net_mvc/asp.net_mvc_caching.htm);
- <https://www.devmedia.com.br/efetuando-o-cache-de-views-no-asp-net-mvc/27628>;
- <https://docs.microsoft.com/pt-br/aspnet/mvc/overview/older-versions-1/controllers-and-routing/improving-performance-with-output-caching-cs>.

### 3.5 Cookie

Cookies fornecem um recurso em aplicativos da Web para armazenar informações específicas de usuário, como preferências do usuário ou histórico. O cookie é um pequeno pedaço de texto que acompanha solicitações e respostas como se eles fossem entre o servidor da Web e o cliente. Cookie contém informações que o aplicativo da Web pode ler sempre que o usuário visitar seu site [15].

O Cookie trabalha com a informação do tipo chave - valor no formato de texto. Com este recurso podemos criar algumas funcionalidades interessantes ao nosso

usuário, como por exemplo, no projeto Loja Virtual podemos armazenar os itens adicionados ao “carrinho de compra” em um cookie, desse modo, caso o usuário não finalize a compra, os itens adicionados continuarão no carrinho quando voltar ao site, ou posteriormente enviaremos e-mail referente a aqueles produtos. Ele possui algumas limitações de tamanho, quantidade por browser, entre outras.

Implementaremos um simples exemplo do uso de Cookie no projeto Blog Pessoal, salvaremos o e-mail utilizado ao logar no site para que na próxima vez que o autor acessar nosso site, o e-mail dele já estará preenchido. Para salvar o e-mail no cookie implementaremos o trecho de código a seguir em AcessoController (localizado em: BlogPessoal.Web > Controllers > AcessoController).

```
    private void AdicionarCookieLogin(string email)    HttpCookie  
cookie = Response.Cookies["usuarioBlogPessoal"]; cookie.Value = email; cookie.Expires  
= DateTime.Today.AddDays(7);
```

O método AdicionarCookieLogin deve ser chamado após a autenticação com sucesso do usuário. Com as informações do autor logado enviaremos seu e-mail para o Cookie “usuarioBlogPessoal”. Depois, precisaremos realizar uma pequena alteração em nossa Action que carrega a página de login, nesta Action iremos recuperar o valor do Cookie “usuarioBlogPessoal” e preenchê-lo no input de e-mail da tela de acesso restrito.

```
 [AllowAnonymous] public ActionResult Index(string returnUrl) if  
(Request.IsAuthenticated)    return    RedirectToAction("Index",    "Home");    if  
(!string.IsNullOrEmpty(returnUrl)) TempData["ReturnUrl"] = returnUrl; var cookie =  
Request.Cookies["usuarioBlogPessoal"];    if    (cookie    !=    null  
!string.IsNullOrEmpty(cookie.Value)) return View(new Login Email = cookie.Value );  
return View();
```

Recomendo a leitura dos seguintes artigos para aprofundamento no assunto:

- [https://msdn.microsoft.com/pt-br/library/ms178194\(v=vs.100\).aspx](https://msdn.microsoft.com/pt-br/library/ms178194(v=vs.100).aspx);
- <https://www.devmedia.com.br/gravacao-leitura-e-remocao-de-cookies-no-asp-net/32835>.

### 3.6 Chrome Developer Tool

O Chrome DevTools é um conjunto de ferramentas de autoria e depuração de Web incorporado ao Google Chrome. Use o DevTools para iterar, depurar e criar o perfil do seu site.

Acesse esta ferramenta clicando F12 do Chrome. Durante todo o desenvolvimento web, esta ferramenta te dará suporte e te auxiliará durante todo o processo de implementação, além de te ajudar na análise de desempenho das suas requisições.

### 3.7 Desempenho em consultas

Até agora, falamos muito de desempenho voltado para a aplicação web. Conhecer as tecnologias que nos ajudarão a performar nosso site é muito importante, porém precisamos entender que há outras questões que poderão deixar nossa aplicação mais lenta.

Conforme já apresentamos, utilizamos como ORM o Entity Framework, aprendemos o quanto está ferramenta nos ajuda no desenvolvimento de nossas soluções, porém há algumas atenções que devemos levantar em relação a esta tecnologia.

Primeiramente, o EF e qualquer outro ORM são tecnologias que geram de maneira dinâmica o SQL com base nas ações realizadas via código. Desse modo, o SQL gerado não é viabilizado para ser o mais performático possível, mas sim que o SQL seja criado corretamente. Sendo assim, a maneira que construímos nossas consultas pelo EF tem forte influencia sobre o SQL que será gerado, a falta de conhecimento de muitos desenvolvedores sobre o EF faz com que muitas profissionais, e principalmente o DBA, que analise cada consulta que é enviada ao banco de dados, odeiem qualquer ORM.

Para mudarmos esta ideia, é importante conhecermos maneiras de estruturarmos melhor nossas consultas do EF e também conhecer ferramentas que possam nos ajudar nesta tarefa, pois sabemos que com o EF ganhamos produtividade de um lado vs. performance.

Um primeiro cuidado com o EF é que nossa consulta só será enviada ao banco de dados após a invocação de algum destes métodos, que são: .ToList(), .First(), .Count(), .Max(), .Min() entre outros. Então, devemos realizar os filtros (Where()) antes de chamá-los, precisamos prestar bastante atenção na ordem dos métodos em nossas consultas.

Outro cuidado relacionado ao EF, é que durante o desenvolvimento precisamos deixar habilitado o Log do EF, com este recurso conseguimos ver todos os comandos que o EF está executando durante o Debug da aplicação. Para habilitar este recurso, basta acessar o contexto do EF, que por exemplo, é o LojaVirtualContexto do projeto Loja Virtual e adicionar o seguinte comando em suas configurações: Database.Log = t => Debug.WriteLine(t);

A figura 85 apresenta como habilitar a configuração de Log do EF, repare que o comando Database.Log = t => Debug.WriteLine(t); está entre os comentários #if DEBUG e #endif, isto significa que somente ao rodar a aplicação em modo de debug as instruções dispostas entre esses comentários serão executadas.

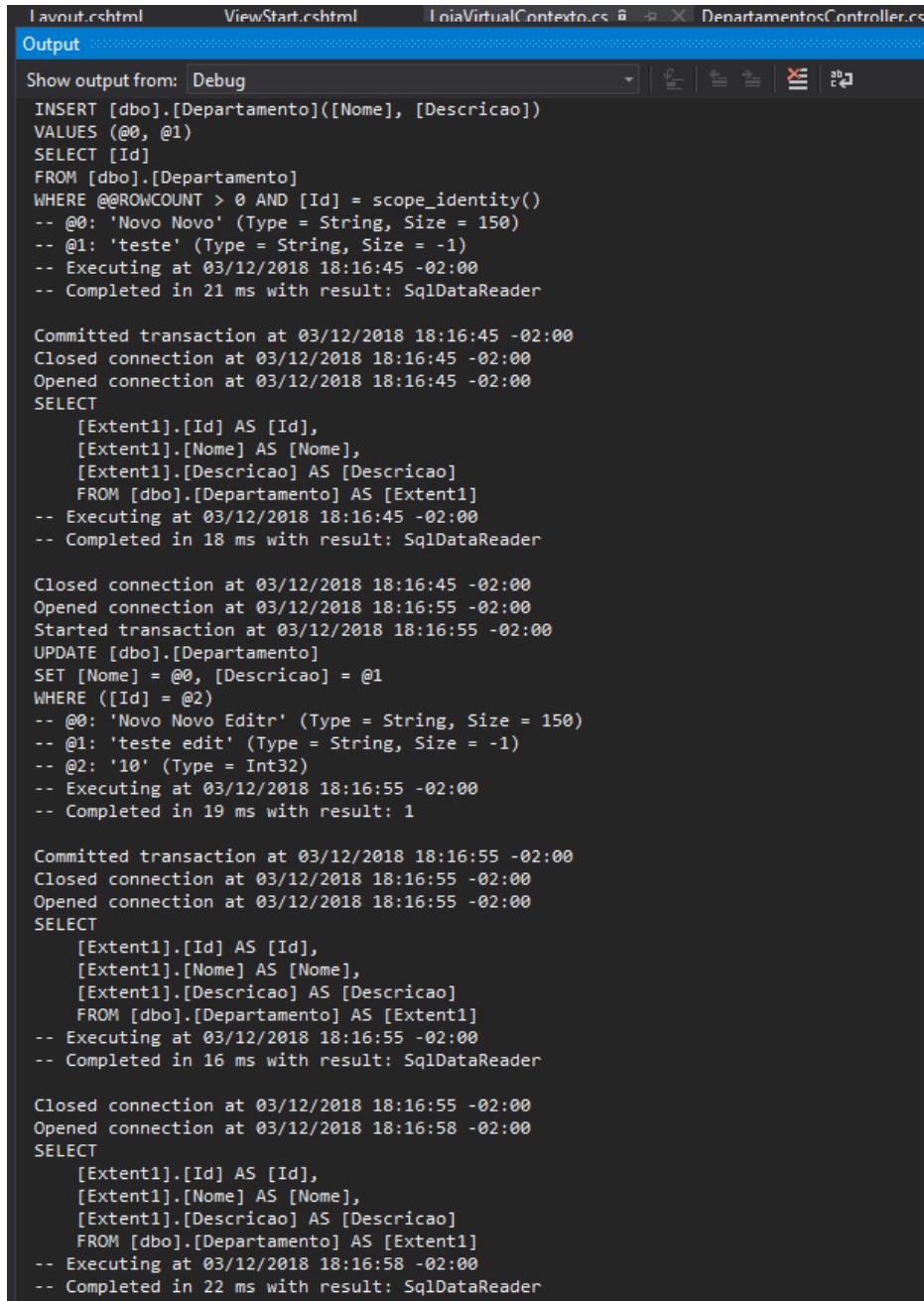
```
public class LojaVirtualContexto : DbContext
{
    1 reference | JOAO VITOR FERRARI DA SILVA, 2 hours ago | 1 author, 1 change
    public LojaVirtualContexto()
        : base(typeof(LojaVirtualContexto).Name)
    {
        IniciarContexto();
    }

    1 reference | JOAO VITOR FERRARI DA SILVA, 2 hours ago | 1 author, 1 change
    private void IniciarContexto()
    {
        ...
        Database.SetInitializer(new CreateDatabaseIfNotExists<LojaVirtualContexto>());
        Database.SetInitializer(new MigrateDatabaseToLatestVersion<LojaVirtualContexto, Configuration>());

        Configuration.ValidateOnSaveEnabled = false;
    #if DEBUG
        Database.Log = t => Debug.WriteLine(t);
    #endif
    }
}
```

Figure 85: Habilitar Log do EF.

Por fim, a figura 86 apresenta Log dos comandos gerados pelo EF durante execução em debug da aplicação Loja Virtual. Na imagem foram executados comandos realizados à manutenção do domínio Departamento.



The screenshot shows the Visual Studio Output window with the title bar "Iavout.cshtml", "ViewStart.cshtml", "IovaVirtualContexto.cs", "DenpartamentosController.cs", and "Output". The "Output" tab is selected. The log displays the following SQL commands:

```

INSERT [dbo].[Departamento]([Nome], [Descricao])
VALUES (@0, @1)
SELECT [Id]
FROM [dbo].[Departamento]
WHERE @@ROWCOUNT > 0 AND [Id] = scope_identity()
-- @0: 'Novo Novo' (Type = String, Size = 150)
-- @1: 'teste' (Type = String, Size = -1)
-- Executing at 03/12/2018 18:16:45 -02:00
-- Completed in 21 ms with result: SqlDataReader

Committed transaction at 03/12/2018 18:16:45 -02:00
Closed connection at 03/12/2018 18:16:45 -02:00
Opened connection at 03/12/2018 18:16:45 -02:00
SELECT
    [Extent1].[Id] AS [Id],
    [Extent1].[Nome] AS [Nome],
    [Extent1].[Descricao] AS [Descricao]
    FROM [dbo].[Departamento] AS [Extent1]
-- Executing at 03/12/2018 18:16:45 -02:00
-- Completed in 18 ms with result: SqlDataReader

Closed connection at 03/12/2018 18:16:45 -02:00
Opened connection at 03/12/2018 18:16:55 -02:00
Started transaction at 03/12/2018 18:16:55 -02:00
UPDATE [dbo].[Departamento]
SET [Nome] = @0, [Descricao] = @1
WHERE ([Id] = @2)
-- @0: 'Novo Novo Editr' (Type = String, Size = 150)
-- @1: 'teste edit' (Type = String, Size = -1)
-- @2: '10' (Type = Int32)
-- Executing at 03/12/2018 18:16:55 -02:00
-- Completed in 19 ms with result: 1

Committed transaction at 03/12/2018 18:16:55 -02:00
Closed connection at 03/12/2018 18:16:55 -02:00
Opened connection at 03/12/2018 18:16:55 -02:00
SELECT
    [Extent1].[Id] AS [Id],
    [Extent1].[Nome] AS [Nome],
    [Extent1].[Descricao] AS [Descricao]
    FROM [dbo].[Departamento] AS [Extent1]
-- Executing at 03/12/2018 18:16:55 -02:00
-- Completed in 16 ms with result: SqlDataReader

Closed connection at 03/12/2018 18:16:55 -02:00
Opened connection at 03/12/2018 18:16:58 -02:00
SELECT
    [Extent1].[Id] AS [Id],
    [Extent1].[Nome] AS [Nome],
    [Extent1].[Descricao] AS [Descricao]
    FROM [dbo].[Departamento] AS [Extent1]
-- Executing at 03/12/2018 18:16:58 -02:00
-- Completed in 22 ms with result: SqlDataReader

```

Figure 86: Log dos comandos gerados pelo EF.

Nas seções a seguir, apresentaremos algumas ferramentas e tecnologias que nos ajudaram no trabalho com dados.

### 3.7.1 Profiler SQL Server

O Profiler SQL Server é uma ferramenta para rastrear, recriar e solucionar problemas no MS SQL Server. Ele permite que os desenvolvedores e administradores de banco de dados (DBAs) criem e manipulem rastreamentos, reproduzam e analisem resultados de rastreamento. Em suma, é como um painel que mostra a saúde de uma instância do MS SQL Server.

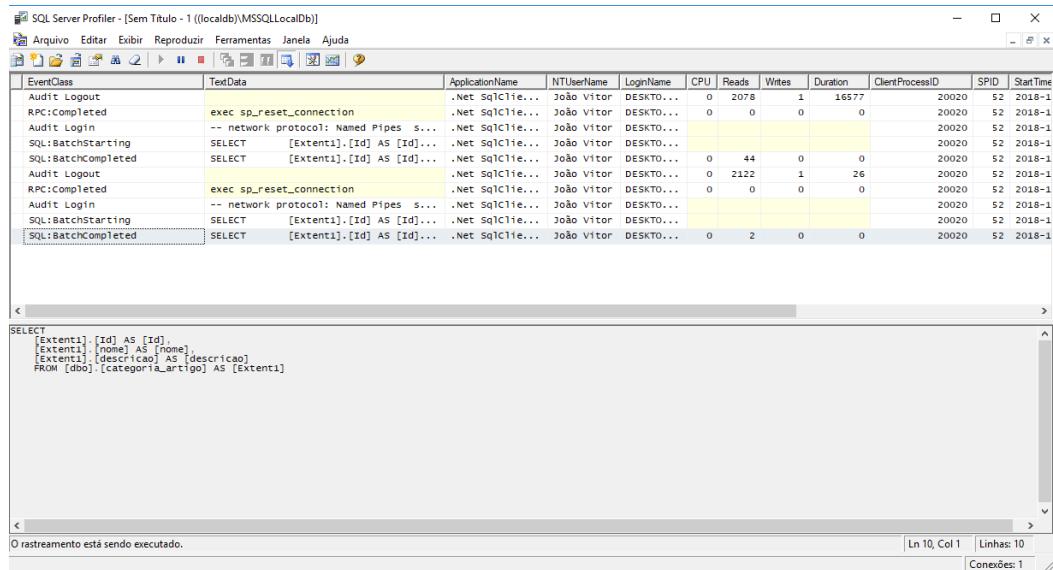


Figure 87: Ferramenta Profiler SQL Server.

A figura 87 apresenta detalhes da ferramenta Profiler SQL Server. Com ela poderemos analisar as consultas que são enviadas ao banco de dados. Uma dica bacana para melhorarmos nossas consultas é analisarmos o plano de execução gerado.

Uma dica muito importante sobre esta ferramenta, é que depois de analisar seus rastreamentos nunca esqueça de parar o rastreamento.

Acompanhe a documentação oficial desta ferramenta pelo link:

<https://docs.microsoft.com/pt-br/sql/tools/sql-server-profiler/sql-server-profiler?view=sql-server-2017>

Por fim, recomendo a leitura do seguinte artigo:

<http://netcoders.com.br/monitoramento-no-microsoft-sql-server-2012-utilizando-a-ferramenta-sql-server-profiler/>

Sobre plano de execução das consultas, recomendo a leitura dos seguintes links:

- <https://www.devmedia.com.br/sql-server-query-analise-do-plano-de-execucao/30024;>
- <https://imasters.com.br/banco-de-dados/5-coisas-que-voce-deve-saber-sobre-o-plano-de-execucao-das-suas-queries.>

### 3.7.2 LINQPad

O LINQPad é uma ferramenta que permite realizar consultas a bancos de dados com facilidade usando a linguagem LINQ. Ela é uma alternativa prática e leve do SQL Management Studio, além de ser uma excelente ferramenta para que você possa aprender mais sobre LINQ, pois nela constam vários exemplos para sua prática.

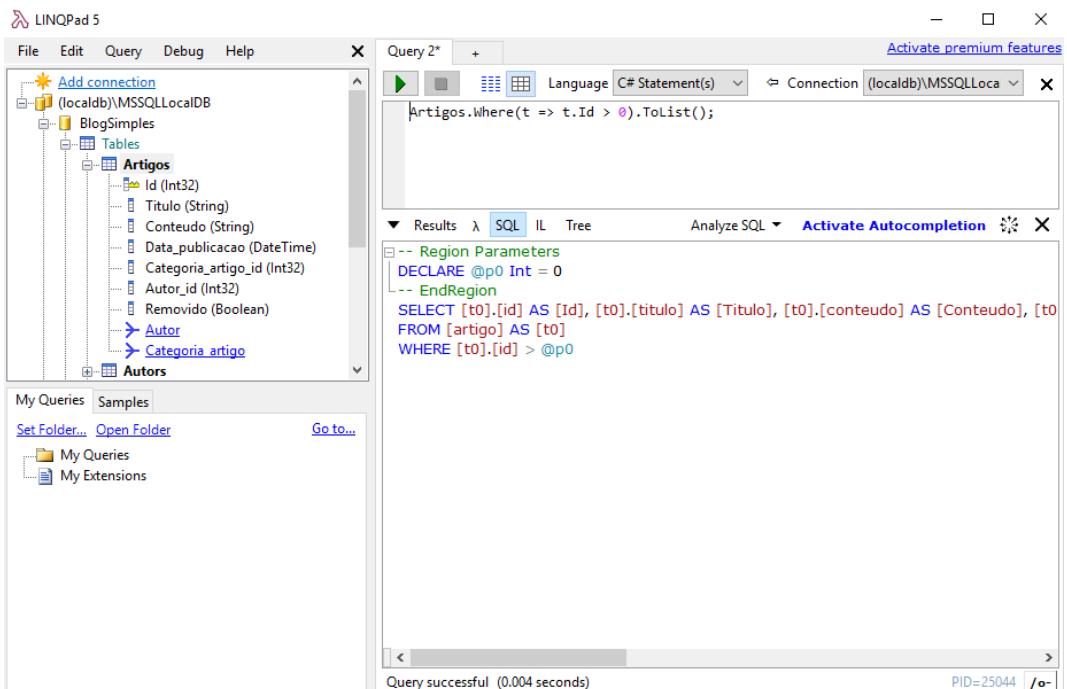


Figure 88: Ferramenta LINQPAD.

A figura 88 apresenta detalhes da ferramenta LINQPad, ela possui suporte a LINQ to SQL e Entity Framework, o que nos possibilita construir e testar nossas consultas nesta ferramenta antes de adicionarmos em nossa aplicação.

Acesse o site oficial da ferramenta: <https://www.linqpad.net/>

### 3.7.3 Dapper

Vimos que os ORMs são úteis em grande número de situações, porém eles podem apresentar limitações em cenários mais específicos, como por exemplo, consultas que envolvem tabelas com muitos registros são demoradas, pois os comandos SQL gerados por meio de uma solução ORM nem sempre serão executadas da forma mais performática possível.

Uma solução possível para este cenário, seria o uso de recursos básicos do ADO.NET, de modo a utilizarmos instruções SQL escritas de forma a se beneficiarmos de índices que otimizem o processamento, porém este recurso pode não ser tão produtivo em termos de desenvolvimento.

É neste cenário que utilizaremos o micro-ORM Dapper, que não possui as funcionalidades igual de um ORM, mas disponibiliza Extension Methods que simplificam o trabalho com os objetos de conexão do ADO.NET.

Para utilizá-lo em nossos projetos, precisaremos instalar o Dapper via Nuget. Como ele trabalha com métodos de extensão sobre a conexão, precisaremos criá-la para começar a utilizá-lo.

É interessante que para relatórios, gráficos e consultas complexas/performáticas utilizarmos o Dapper.

Recomendo o acesso à página oficial do projeto Dapper, acesse: <https://github.com/StackExchange/Dapper>.

Também recomendo conhecer o Dapper.Contrib que é uma extensão do Dapper que facilita a implementação do CRUD (inclusão, alteração, exclusão e leitura

de registros).

Uma boa prática é evitar escrever consultas direto no código do projeto, prefira chamar Views do banco de dados.

### **3.8 Content Delivery/Distribution Network (CDN)**

O CDN é uma rede de servidores que armazenam réplicas do conteúdo de outros sites na memória (cache) e depois os entrega aos usuários, baseando-se na localização geográfica para conectá-los ao servidor mais próximo e mais rápido, reduzindo o tempo de transferência dos dados (latência).

Utilizar a referência das bibliotecas por meio do CDN, ao invés da referência local, podem aumentar o desempenho do site.

No projeto Blog Pessoal, podemos referenciar o CDN do Bootstrap, da jQuery, da Font Awesome, entre outras bibliotecas. Também podemos contratar um serviço para criar um CDN dos scripts que desenvolvemos.

## **4. Considerações finais**

Criptografia é uma alinhada que você deve ter durante o desenvolvimento. Pode ser aplicada em vários lugares em nossa aplicação, por exemplo, criptografar a string de conexão do banco no Web.config e as informações sensíveis do usuário. Falando de criptografia e Web.config, vamos lembrar de configurar o certificado SSL em nossa aplicação, vimos os benefícios desta tecnologia, sabemos que só ela não irá garantir toda a segurança em nossa aplicação, mas com certeza já será de grande valia neste assunto, além de ajudar a indexação do nosso site pelos buscadores de site.

Vale destacar que páginas lentas não contribuem com a indexação do site pelos tradicionais buscadores do mercado. Desse modo, devemos sempre buscar o bom desempenho de nossas aplicações, precisamos utilizar, sempre que possível e quando viável, todas as técnicas que aprendemos neste capítulo, como a compactação da requisição, uso de cache, cookie, entre outras tecnologias.

Jamais esqueça de desenvolver sua aplicação com o máximo de segurança possível, garanta a proteção de quem está acessando sua aplicação. Além da segurança, se preocupe com o desempenho de seu projeto, sabemos que a maioria dos usuários esperam pouco tempo para que o site seja carregado, caso ele demore muito, perderemos usuários.

# UNIDADE 4

## Publicando nossos projetos

Neste capítulo, aprenderemos como disponibilizar nossa aplicação para os clientes. Conheceremos as ferramentas utilizadas para publicação na nuvem por meio do Azure e manualmente pelo Internet Information Services (IIS).

Serão apresentadas técnicas de indexação para que sua aplicação seja bem ranqueada pelos diversos sites de busca.

Por fim, apresentaremos algumas dicas, boas práticas e assuntos que nós devemos colocar no nosso radar de conhecimento para sermos desenvolvedores melhores.

### 1. Internet Information Services (IIS)

O IIS é um servidor web que oferece recursos para hospedagem de sites, serviços e aplicativos, na qual suporta a tecnologia ASP.NET.

Aprenderemos a ativar o recurso do IIS em nosso ambiente Windows para montarmos um servidor HTTP, configurar nosso site e disponibilizarmos o projeto Blog Pessoal.

Primeiramente, acessaremos o “Executar” (combinação de teclas: Windows + R), digitaremos “appwiz.cpl” e Enter para iniciar. Isto abrirá a parte do Programa e Recursos do Painel de Controle, no lado esquerdo, clique no link “Ativar ou desativar recursos do Windows”.

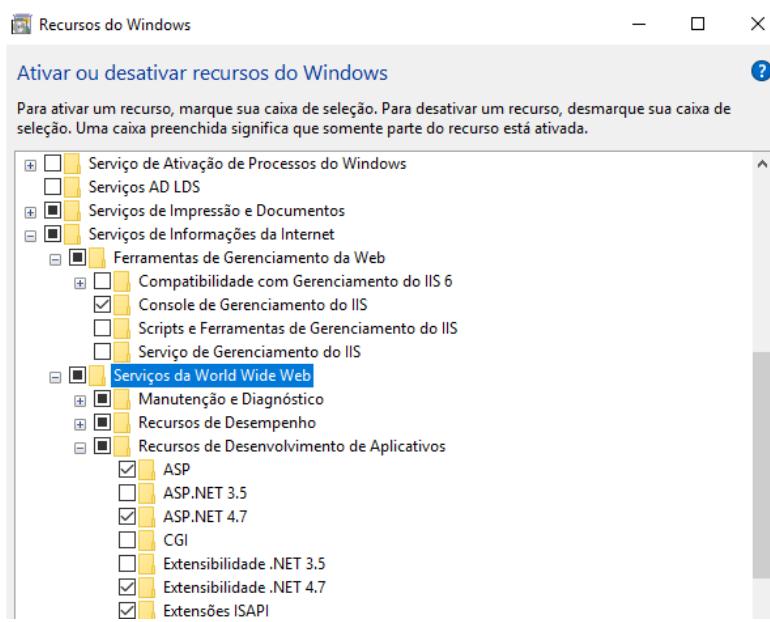


Figure 89: Instalação do IIS.

Após, abrirá uma caixa de seleção, selecione “Serviços de Informações da Internet”. A figura 89 ilustra este processo. Por padrão, ele instala todas as coisas necessárias para hospedar um site, porém nós provavelmente precisaremos também de alguns dos componentes mais centrados no desenvolvimento.

Para verificarmos se ocorreu tudo certo com nossa instalação, basta acessarmos o navegador com a seguinte url: <http://localhost/>, deve ser carregado a tela padrão do IIS, conforme apresenta a figura 90.

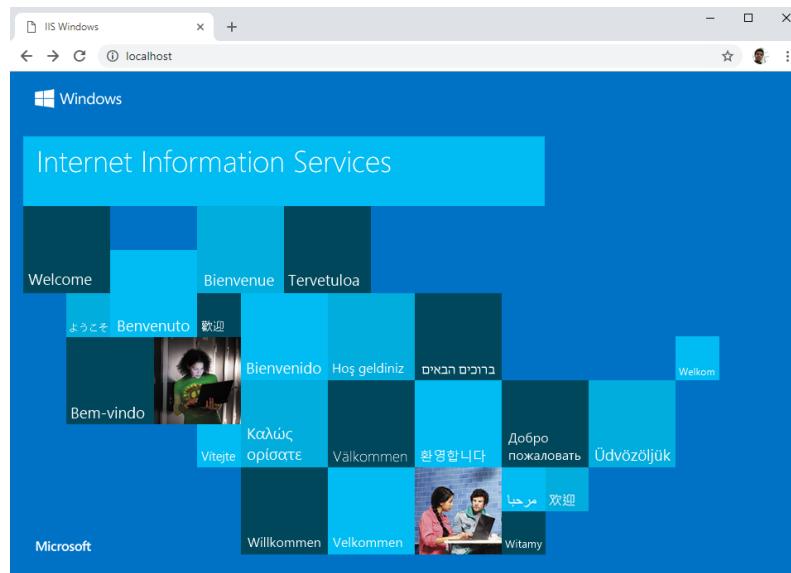


Figure 90: Página padrão do IIS.

Acesse o site oficial do IIS: <https://www.iis.net/>.

## 1.1 Gerenciador do Serviços de Informação da Internet (IIS)

O Gerenciador do IIS permite configurarmos Serviços de Informações da Internet (IIS). O Gerenciador do IIS depende da versão do IIS e o sistema operacional.

Antes de publicarmos nossa aplicação, conhceremos o Web Platform Installer<sup>22</sup>, que é uma ferramenta gratuita que facilita o download, a instalação e a atualização dos componentes mais recentes da Microsoft Web Platform, incluindo o IIS, o SQL Server Express, o .NET Framework e o Visual Studio. Também facilita a instalação e a execução dos aplicativos da Web gratuitos mais populares para blogs, gerenciamento de conteúdo e muito mais com a galeria de aplicativos da Web incorporada do Windows. A figura 91 apresenta a tela inicial do Web Platform Installer.

---

<sup>22</sup> <https://www.microsoft.com/web/downloads/platform.aspx>

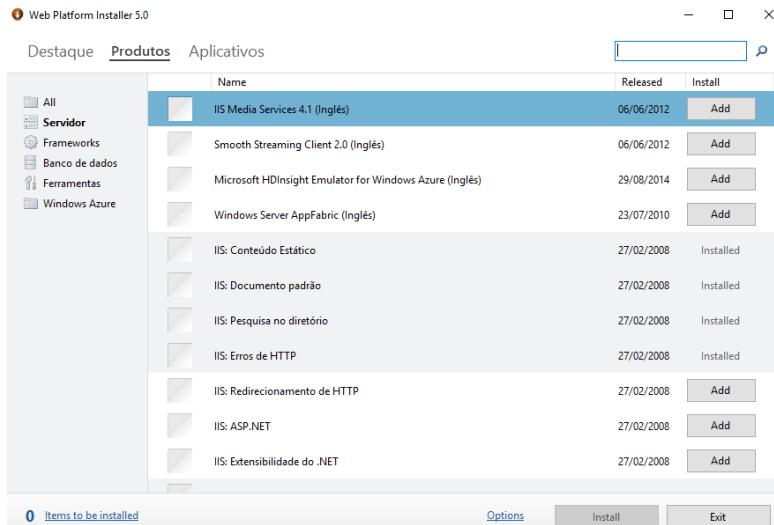


Figure 91: Apresentação Web Plataform Installer.

No Web Platform Installer, acesse a opção “Servidor”, caso os recursos “IIS: ASP.NET”, “URL Rewrite”, “.NET Framework 4.5”, “ASP.NET MVC 4” e “ASP.NET MVC 3” não estejam instalados, peço que você os adicione e depois clique em instalar. Quando precisar de qualquer outro recurso ou necessitar atualizá-lo, utilize o Web Plataform Installer.

Com nosso serviço de hospedagem configurado e atualizado, iremos publicar o projeto Blog Pessoal. No projeto BlogPessoal.Web acesse a opção “Publish”, conforme ilustra a figura 92.

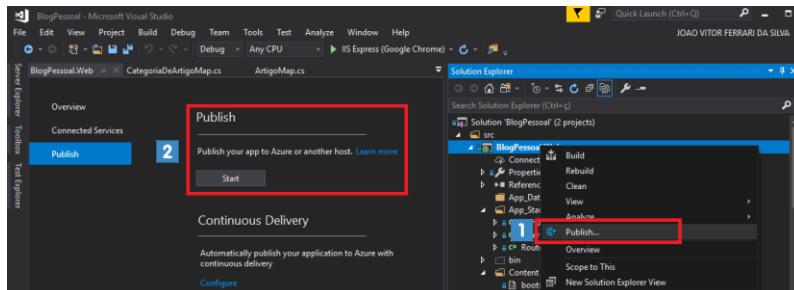


Figure 92: Opção “Publish” do projeto BlogPessoal.Web.

Depois escolha a opção de publicação “Folder” em “Pick a publish target”, escolha o caminho desejado para que os arquivos da aplicação sejam disponibilizados. Por fim, basta publicarmos nosso projeto conforme apresenta a figura 93.

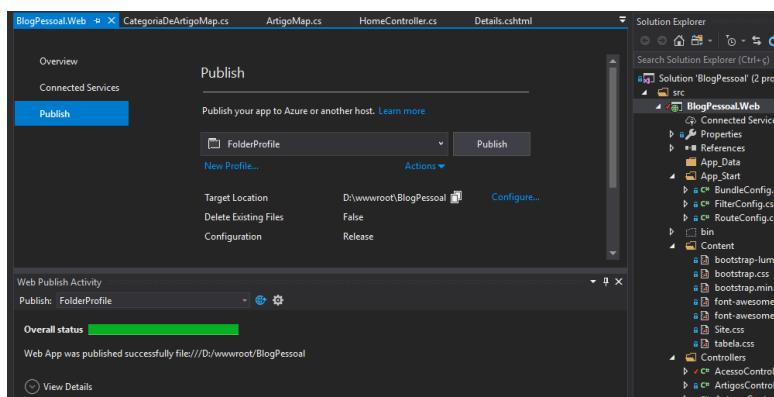


Figure 93: Publicação com sucesso do projeto BlogPessoal.Web.

Agora vamos configurar nossa aplicação no Gerenciador do IIS. Primeiramente adicionaremos um Pool de Aplicativos específico para o Blog Pessoal. Com o Pool de Aplicativos (1) definimos um grupo de um ou mais processo de trabalho, fazendo com que as aplicações sejam um processo único dentro do servidor web. A figura 94 demonstra este processo.

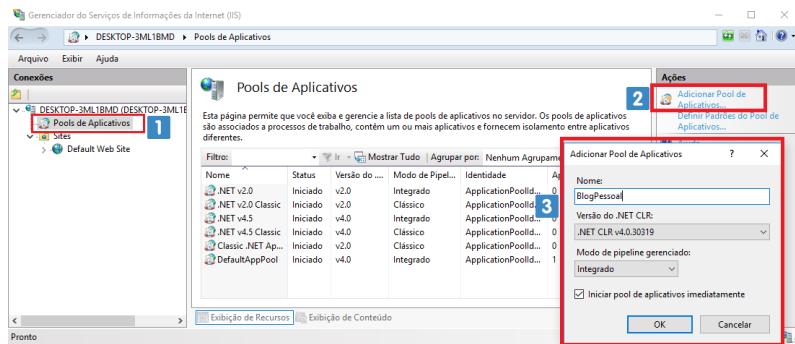


Figure 94: Adicionando Pool de Aplicativos para o Blog Pessoal.

Por fim, iremos adicionar nosso site no Gerenciador do IIS. Na ação “Sites” clique em “Adicionar Sites...”, a tela representada pela figura 95 deverá ser apresentada. O nome do nosso site será “BlogPessoal”, depois selecionaremos o Pool de Aplicativos criado na etapa anterior, informaremos o caminho onde publicamos os arquivos do site pelo Visual Studio, e por fim adicionaremos uma porta que nossa aplicação será iniciada, muito cuidado para não informar uma porta que já está sendo utilizada para qualquer outro serviço.

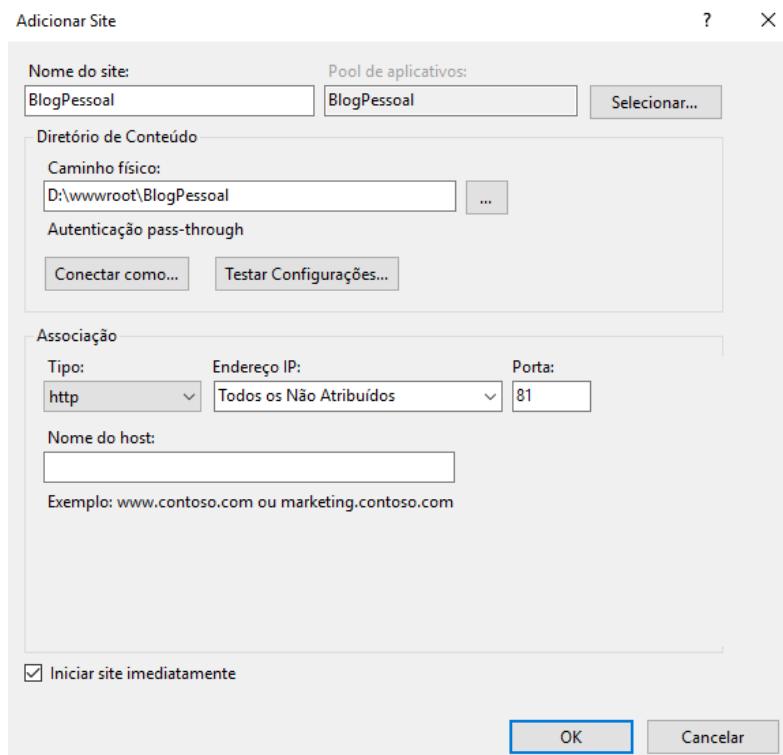


Figure 95: Criando o site Blog Pessoal no Gerenciador do IIS.

Por fim, basta acessarmos localhost na porta informada, o site Blog Pessoal deve ser carregado, conforme apresentada a figura 96. No exemplo apresentado, o projeto utiliza uma conexão com banco de dados do SQL no Azure, você pode utilizar o banco de dados configurado na seu computador, porém precisará configurá-lo, caso tenha interesse pesquise sobre acesso remoto do SQL Server<sup>23</sup>.

Nome	Descrição	Opções
Alimentos	Alimentos	<a href="#">Editar</a> <a href="#">Remover</a>
Beleza & Saúde	Beleza & Saúde	<a href="#">Editar</a> <a href="#">Remover</a>
Cama, Mesa e Banho	Importante para casa	<a href="#">Editar</a> <a href="#">Remover</a>
Carro	Teste	<a href="#">Editar</a> <a href="#">Remover</a>
Cartão de Presente	Presente	<a href="#">Editar</a> <a href="#">Remover</a>
Cozinha	Utilidades de cozinha	<a href="#">Editar</a> <a href="#">Remover</a>
Doações	Doações	<a href="#">Editar</a> <a href="#">Remover</a>

Figure 96: Acessando site Blog Pessoal publicado no IIS.

Aprenderemos um recurso que nos auxiliará na publicação das aplicações, o nome dele é Web Deploy. Primeiramente, por meio do Web Platform Installer, adicionaremos os recursos relacionados ao Web Deploy<sup>24</sup>, que são: “Web Deploy 3.6” e “Web Deploy 3.6 for Hosting Server”, siga o exemplo demonstrado pela figura 97.

Figure 97: Adicionando recurso Web Deploy.

Execute o Visual Studio como administrador e abra o projeto Blog Pessoal, adicionaremos um novo perfil de publicação. No projeto BlogPessoal.Web acesse a opção “Publish”, depois em “Pick a publish target” escolha a opção “IIS, FTP, etc”, deverá ser apresentada a tela demonstrada pela figura 98.

<sup>23</sup> <https://www.sqlfromhell.com/habilitando-a-conexao-remota-no-sql-server/>

<sup>24</sup> <https://www.iis.net/downloads/microsoft/web-deploy>

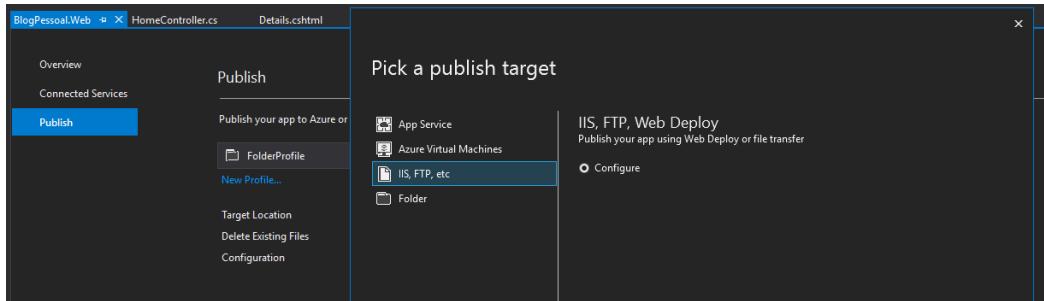


Figure 98: Configurando perfil de publicação para Web Deploy.

Após clique no botão “Publish” para carregar a tela de configurações, conforme apresenta a figura 99. Nesta tela selecionamos em “Publish method” a opção “Web Deploy”, em “Server” informaremos a opção “localhost”, por exemplo. No campo “Site name” informaremos o nome do nosso site, atente-se que este nome deve ser igual ao nome configurado no IIS. Neste exemplo local não precisaremos informar o “User name” e nem o “Password”, por último, clique no botão “Validate Connection” para verificar se as configurações estão corretas.

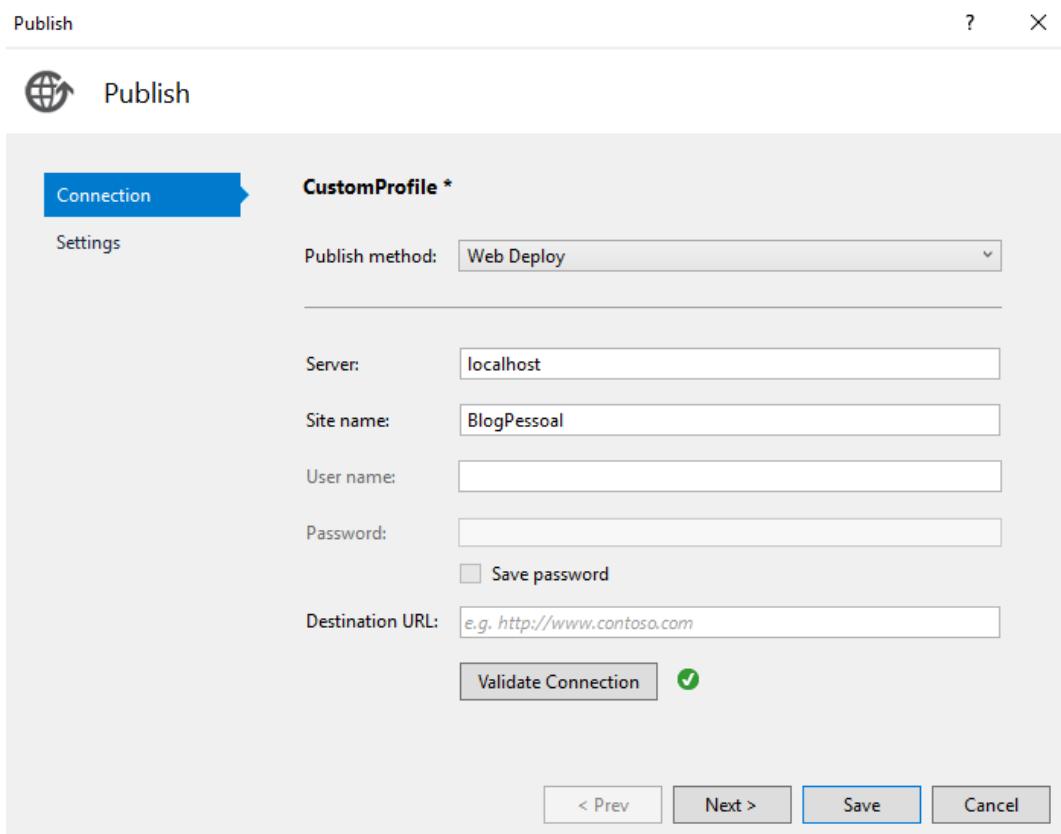


Figure 99: Configurações do Web Deploy.

Após configurado, o deploy se torna bem mais simples e sem a necessidade de acesso ao servidor. Importante conhecermos este recurso pois o utilizaremos para publicarmos nossa aplicação no Azure, acompanhe as próximas seções.

## 2. Microsoft Azure

Iniciaremos um estudo básico sobre o Microsoft Azure<sup>25</sup>, que é uma plataforma para a implementação de computação em nuvem (cloud computing) que oferece um conjunto específico de serviços para desenvolvedores e engloba desde websites até active directory, passando por banco de dados SQL, machine learning entre outros recursos.

O Microsoft Azure pode ser usado tanto como IaaS (Infrastructure-as-a-Service) quanto como PaaS (Platform-as-a-Service), é flexível (qualquer linguagem de programação e qualquer sistema operacional) além da possibilidade de escalar facilmente.

Há muitas vantagens ao adotar o Azure, como a questão de você não se preocupar diretamente com a infraestrutura, backup, segurança, redundância, além de outros importantes assuntos. Com esta tecnologia podemos tornar nossa aplicação escalável por meio de poucos cliques, imagina realizar a mesma ação em uma infraestrutura local? Daria bem mais trabalho.

Você pode estar se perguntando, será que eu como desenvolvedor devo me preocupar com infraestrutura? Será que preciso entender destes assuntos também? A resposta simplesmente é sim, nós como desenvolvedores devemos entender de infraestrutura também, ainda mais se nosso objetivo é sermos cada dia mais desenvolvedores melhores. Precisamos conhecer sobre a infraestrutura que nossa aplicação será executada, nesse sentido, conhecer a estrutura do Microsoft Azure nos ajuda a rodarmos nossos projetos com maior desempenho possível e utilizando menos recurso, afinal quando falamos de cloud computing seremos cobrados por diferentes critérios, como horas de uso, processamento, tráfego entre outros.

Os principais serviços do Microsoft Azure que poderemos utilizar em nossas aplicações são:

- Máquinas virtuais: são sistemas operacionais que funcionam como computadores completos, mas com todos seus recursos de armazenamento e computação originados da nuvem. Possui a flexibilidade de configuração e ferramentas específicas para cada caso;
- Aplicações em nuvem: é simular um computador inteiro dentro da nuvem, porém não permitem a flexibilidade de alteração das Máquinas virtuais. Estas aplicações aumentam a produtividade, simplificam a aquisição e permitem disponibilização rápida;
- Bancos de dados SQL: é um serviço de banco de dados relacional na nuvem. É um serviço inteligente e totalmente gerenciado, com alta compatibilidade e portabilidade para migrar bancos sem alterar os aplicativos;
- Armazenamento e Backup: o backup dos dados é algo muito importante, e o Azure faz este processo de maneira simplificada e automatizada.

---

<sup>25</sup> <https://azure.microsoft.com/pt-br/>

O Azure permite a criação de uma conta gratuita que oferece 12 meses de serviços gratuitos, caso você ainda não tenha uma conta, acesse: <https://azure.microsoft.com/pt-br/free/>.

Agora, é a vez de publicarmos o projeto Blog Pessoal no Azure. Poderemos optar por disponibilizar nossa aplicação via Máquina virtual, assim, precisaremos criar uma VM Windows Server, instalarmos e configurarmos todo o computador, como o IIS, que aprendemos na seção anterior, juntamente com os recursos do .NET Framework e do ASP.NET.

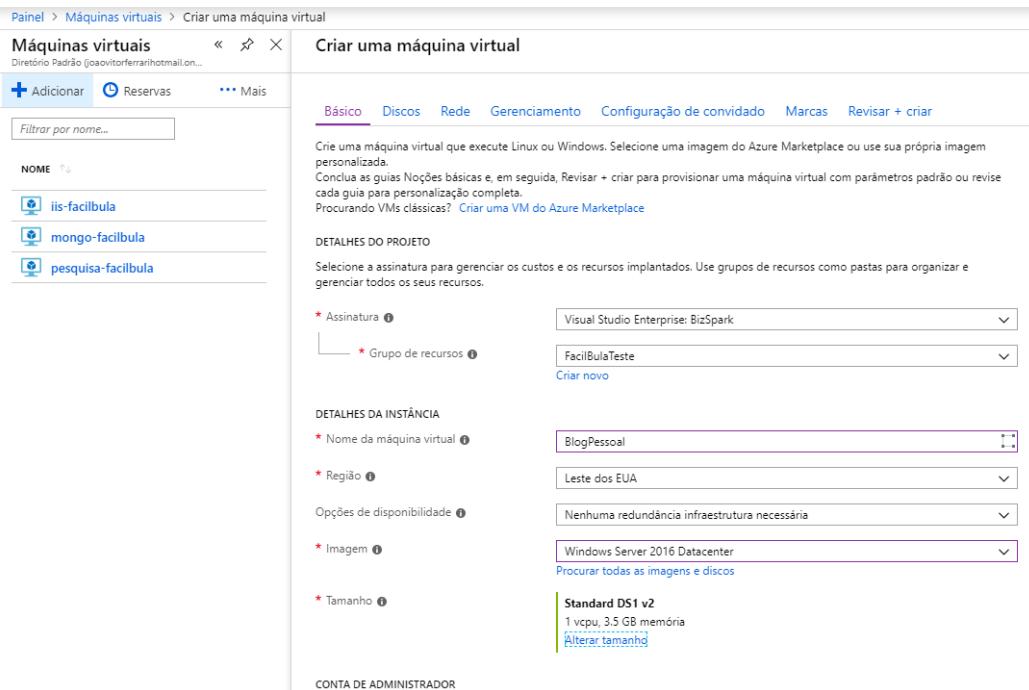


Figure 100: Criar uma máquina virtual no Microsoft Azure.

A figura 100 apresenta a criação de uma máquina virtual no Microsoft Azure, nesta tela informamos as questões sobre assinatura para cobrança, depois preenchemos o nome da máquina virtual e selecionamos a imagem que será utilizada, neste caso foi escolhido Window Server 2016 Datacenter, depois precisará definir o tamanho da máquina, escolher as configurações, cada configuração tem um preço diferente, fique atento aos valores.

Outra maneira de disponibilizar o projeto Blog Pessoal será por Serviços de Aplicativos<sup>26</sup>. Escolheremos a opção “Aplicativo Web”, com este recurso podemos criar e implantar sites em segundos, tão poderosos quanto necessitarmos.

<sup>26</sup> <https://docs.microsoft.com/pt-br/azure/app-service/app-service-web-get-started-dotnet>

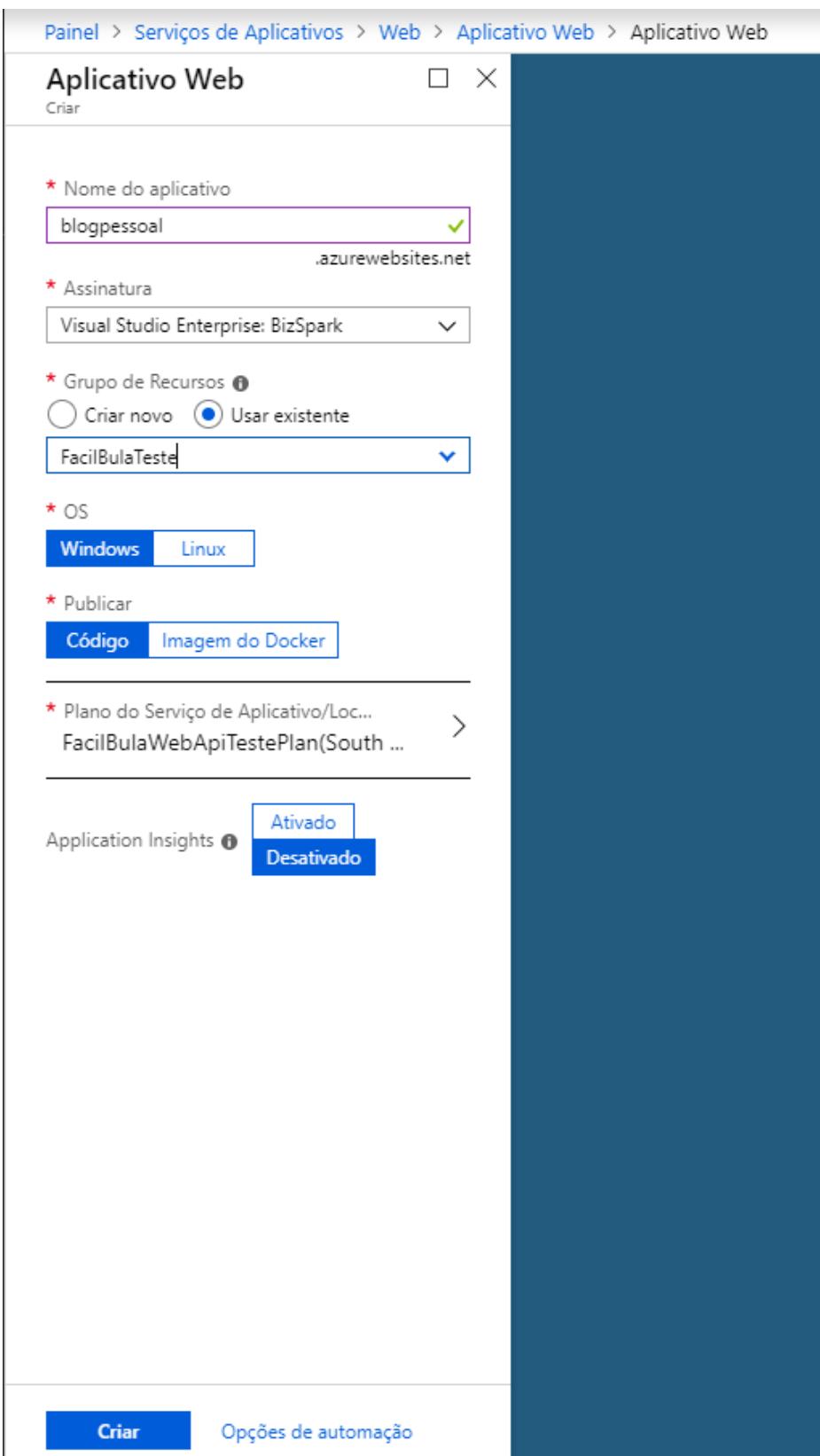


Figure 101: Criar um aplicativo da web do Blog Pessoal no Microsoft Azure.

A figura 101 demonstra o processo de criação de um novo aplicativo da web no Microsoft Azure. Primeiramente, informaremos o nome do nosso aplicativo, que no exemplo será o Blog Pessoal. Depois precisaremos configurar a questão de assinatura e

cobrança, escolher o sistema operacional e por último selecionar o plano de serviço do aplicativo, há vários planos que podem ser configurados.

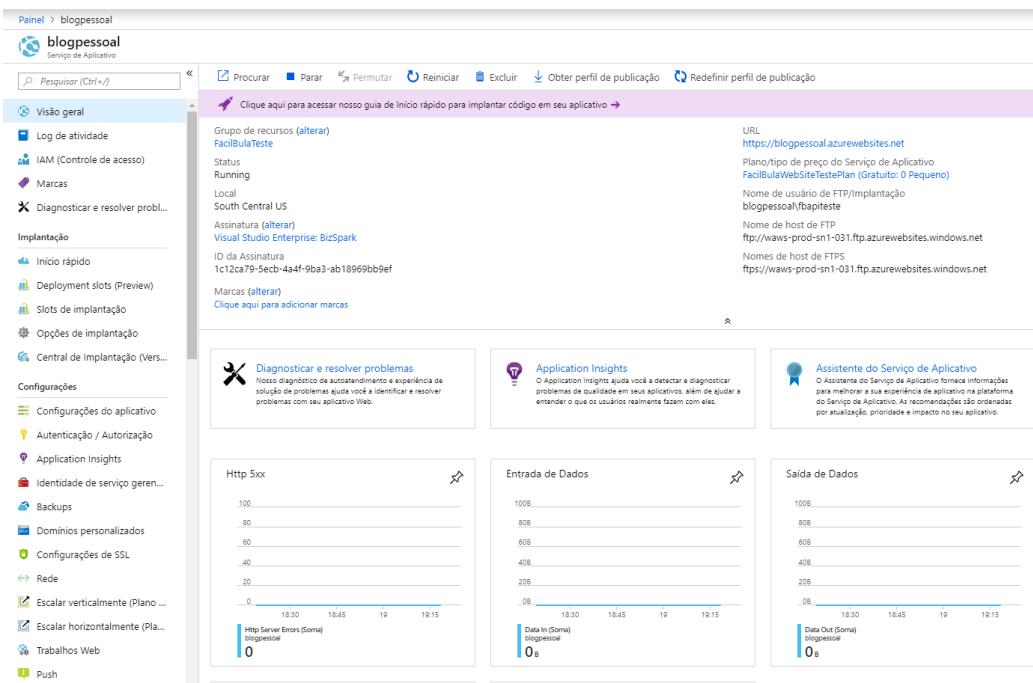


Figure 102: Visão geral do aplicativo da web do Blog Pessoal no Microsoft Azure.

Após a criação do aplicativo da web, será carregada a página de gestão e monitoramento do serviço. Nesta tela de Visão Geral, visualizamos detalhes das requisições realizadas em nossa aplicação. Há também um menu que apresenta variados recursos, como log de atividade, configuração do aplicativo, escalonamento horizontal e vertical, entre outros. A figura 102 apresenta esta tela juntamente com o menu de recursos disponíveis.

Agora, vamos para o Visual Studio. Abriremos o projeto Blog Pessoal e adicionaremos a opção “Publish...”, após adicionaremos um novo perfil de publicação, depois em “Pick a publish target” escolha a opção “App Service”, deverá ser apresentada a tela demonstrada pela figura 103.

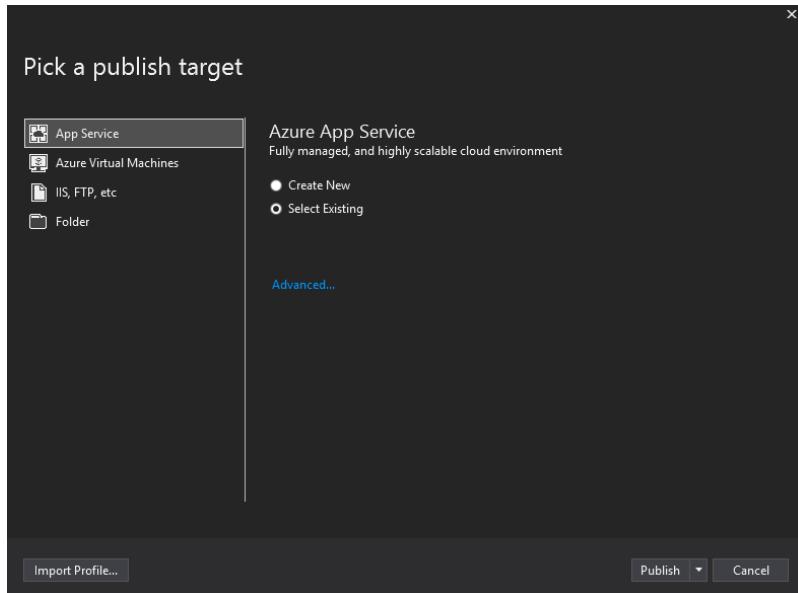


Figure 103: Configurando perfil de publicação para App Service.

Após clique no botão “Publish” para carregar a tela de configurações, conforme apresenta a figura 104. Nesta tela selecionamos a assinatura do Azure que será utilizada, e logo após serão listados os aplicativo da Web, selecione a opção do Blog Pessoal.

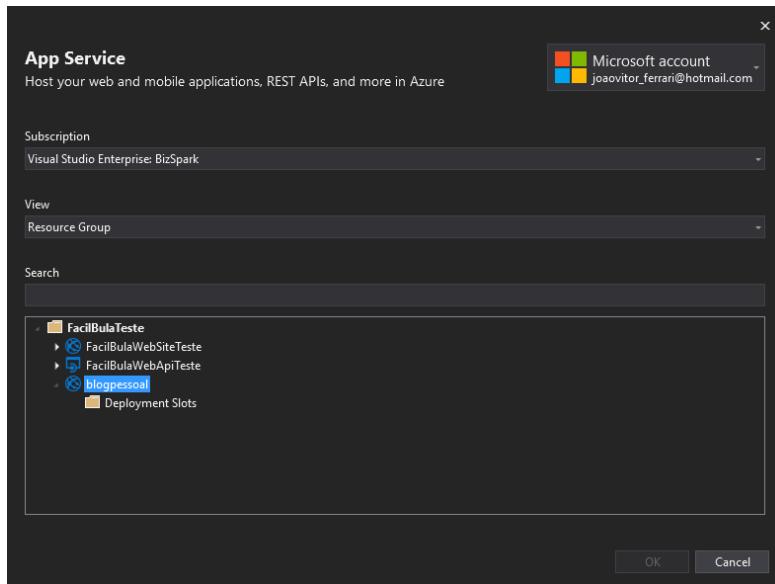


Figure 104: Selecionando o aplicativo da web do Blog Pessoal.

Por fim, basta clicar no botão “OK”, e publicar o projeto, ao final do processo basta acessarmos a url: <http://blogpessoal.azurewebsites.net/> e o site será carregado, conforme demonstra a figura 105.

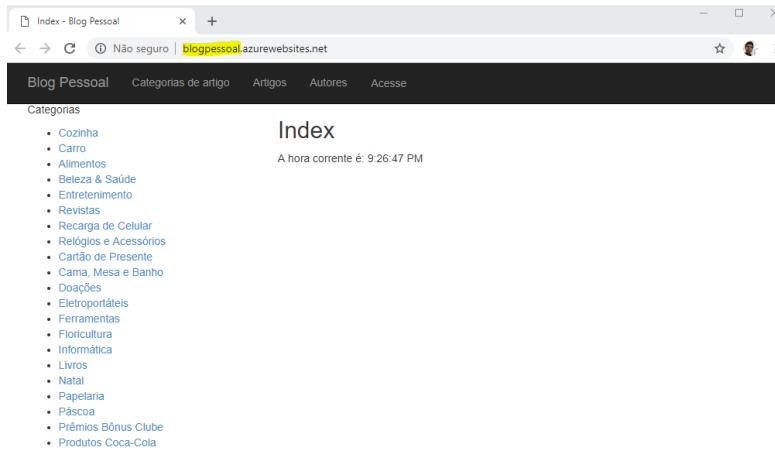


Figure 105: Selecionando o aplicativo da web do Blog Pessoal.

Com o serviço de aplicativo da web, conseguimos aproveitar as ferramentas existentes para criar e implantar aplicativos sem a complicaçāo de termos que gerenciar a infraestrutura. Os sites do Microsoft Azure oferecem desenvolvimento flexível e seguro, implantação e opções de colocação em escala para aplicativos Web de qualquer tamanho.

Segundo o Azure, com aplicativo da web conseguimos ter algumas vantagens, como:

- a maneira mais rápida para criar na nuvem;
- provisionamento e implantação de maneira mais rápida;
- plataforma segura que escala automaticamente;
- ótima experiência para os desenvolvedores do Visual Studio;
- aberto e flexível para todos;
- monitoramento, alerta e autoescala (visualização).

No próprio site é oferecido o guia do desenvolvedor para o Azure por meio do link:

<https://azure.microsoft.com/pt-br/campaigns/developer-guide/>

Com ele aprenderemos como desenvolver no Azure desde o primeiro dia usando cenários comuns de design de aplicativos.

Você pode experimentar os recursos oferecidos pelo Microsoft Azure, leia o seguinte artigo e verifique alguns programas disponíveis:

<https://www.tshooter.com.br/2016/02/15/voc-desenvolvedor-ento-tem-azure-de-graa-para-voc/>

## 2.1 Application Insights

Com o Application Insights coletamos dados de monitoramento dos aplicativos por meio de uma técnica conhecida como Telemetria.

Ele possibilita o monitoramento de diversos aspectos das aplicações, como: quantidade de acessos, tempo de resposta, tempo de acesso a recursos externos, como banco de dados, APIs externas, quantidade de falhas, usuários simultâneos e outras métricas. Há também a opção de registrar métricas customizadas, ou seja, podemos criar monitoramento para regras de negócio, como quantos usuários de um tipo específico acessam a aplicação, quantas requisições retornaram com um determinado tipo de informação, por exemplo.

Recomendo a seguinte leitura: <https://imasters.com.br/apis-microsservicos/monitorando-apis-com-application-insights-parte-01>. Neste artigo, são detalhados cada recurso disponível no Application Insights.

Leia a documentação oficial pelo link: <https://docs.microsoft.com/pt-br/azure/application-insights/app-insights-overview>.

### 3. Qualidade do projeto e outros assuntos importantes

Nas próximas subseções, serão apresentados alguns assuntos importantes que nós como desenvolvedores devemos conhecer. Alguns destes temas serão mais detalhados que outros, porém recomendo que você aprofunde seus estudos e em cada assunto, nem que seja pelo menos para conhecê-los melhor, lembre-se que o conhecimento é a nossa maior riqueza.

#### 3.1 Rollbar

Muitas vezes quando nossa aplicação apresenta algum erro, a primeira coisa que nós, desenvolvedores, fazemos é solucioná-lo o mais rápido possível, ainda mais se for um problema apresentado em ambiente de produção. O recomendável, é que nós analisemos o erro e encontremos a causa raiz, para solucionarmos o problema de uma vez por todas e termos cuidado de não gerar outros problemas devido as alterações realizadas.

Para isso é muito importante estarmos atentos aos erros que acontecem nas nossas aplicações web, principalmente quando elas estão em produção. Neste cenário, recomendo que você conheça o Rollbar<sup>27</sup>.

O Rollbar facilita a captura dos erros da sua aplicação e as organiza de uma forma muito simples. É uma ferramenta paga, porém ela oferece um plano gratuito que aceita até 5.000 registros de erro por mês.

A figura 106 apresenta as capturas de um projeto de exemplo deles, você pode acessá-lo pelo seguinte endereço: <https://rollbar.com/demo/demo/>. Navegue e conheça todos os recursos oferecidos pela ferramenta.

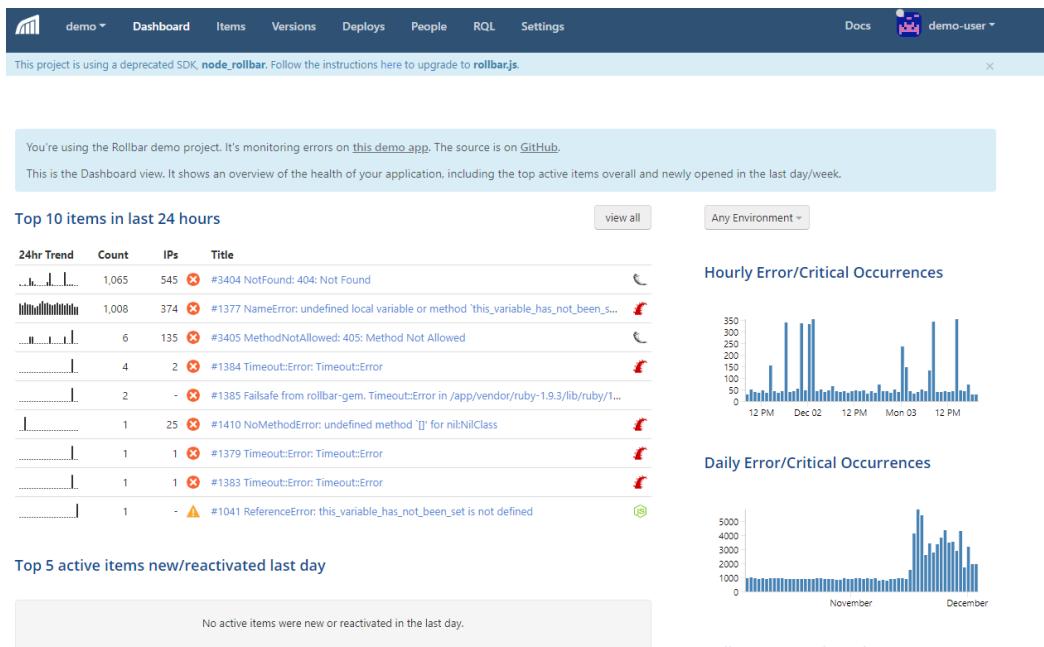


Figure 106: Apresentação da ferramenta Rollbar.

<sup>27</sup> <https://rollbar.com/>

Entre todos os recursos disponíveis, vale a pena citar que podemos utilizar o Rollbar em nossas aplicações ASP.NET por meio de uma biblioteca via Nuget, veja <https://www.nuget.org/packages/Rollbar/>.

Recomendo que você faça a seguinte leitura: <https://medium.com/albertomonteiro/capturando-erros-em-producao-no-asp-net-com-rollbar-2ec9f6fe12e4>.

### 3.2 Search Engine Optimization (SEO)

Um dos maiores objetivos das instituições que investem em Marketing Digital é alcançar as primeiras posições nos diversos mecanismos de busca existentes, e conseguir cliques gratuitos, além de mais visitantes em nosso site.

No entanto, publicar nosso site na internet não é o suficiente para garantir que nosso site conquiste boas posições na busca orgânica da página do Google, por exemplo. Precisaremos investir em uma estratégia específica para otimizar os resultados e garantir que nosso site alcance um bom posicionamento.

Há várias otimizações que podemos fazer em nosso site, visando melhorar a experiência dos usuários e torná-lo aderente aos olhos dos motores de busca. Estas otimizações são chamadas de Search Engine Optimization (SEO), ou simplesmente, otimização para mecanismos de busca.

Antes de iniciarmos é muito importante que nós tenhamos fixado toda questão de segurança da aplicação e também de desempenho, pois se nosso site não estiver protegido com HTTPS e se tivermos páginas lentas já são critérios ruim para SEO.

Aprenderemos algumas técnicas utilizadas que contribuíram com o posicionamento de um site real desenvolvido em ASP.NET MVC.

Começaremos com a implementação de URL amigáveis, juntamente com a definição de títulos diferenciados, a fim de que palavras-chave possam constar no título, na URL e também no conteúdo da página.

Como exemplo utilizaremos o projeto Blog Pessoal, criaremos uma rota para URL amigável dos detalhes de artigo, basicamente nossa URL ficará organizada pelo ano, mês e dia de publicação, acréscimo do título e no final o seu respectivo código identificador. Veja a configuração desta nova rota realizada no arquivo RouteConfig pela figura 107. Importante esclarecer que a configuração de rota “Default” deve ficar na última posição de suas configurações, conforme exemplifica a figura.

```
public class RouteConfig
{
    1 reference | João Vitor Ferrari, 44 days ago | 1 author, 1 change
    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

        routes.MapRoute(
            name: "UrlAmigavelDetalhesArtigo",
            url: "Artigo/{ano}/{mes}/{dia}/{nome}/{id}",
            defaults: new { controller = "Artigos", action = "Detalhes", nome = (string)null },
            constraints: new { id = @"\d+" });

        routes.MapRoute(
            name: "Default",
            url: "{controller}/{action}/{id}",
            defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional })
    }
}
```

Figure 107: Configuração de URL amigável no arquivo RouteConfig.

Agora iremos testar nossa configuração, adicionaremos uma listagem dos artigos mais recentes na página inicial (localizado em: BlogPessoal.Web > Views > Home > Index) do projeto Blog Pessoal. Para construir a URL amigável para detalhes do artigo, podemos utilizar o `@Html.ActionLink`, conforme listagem a seguir. Foi desenvolvida uma classe para tratamento do nome, ela faz tratamentos sobre o texto informado, como deixar tudo em letra minúscula, remover espaços desnecessários e adicionar hífen no lugar, remover caracteres especiais entre outras alterações, e o converte para que seja melhor apresentado na URL.

```
style=sharpc ... <ul> @ List<Artigo> ultimosArtigos = ViewBag.UltimosArtigos;
foreach (var artigo in ultimosArtigos) <li>
@artigo.DataPublicacao.ToShortDateString() <span> - </span>
@Html.ActionLink(artigo.Titulo, "Detalhes", "Artigos", new { ano =
artigo.DataPublicacao.Year, mes = artigo.DataPublicacao.Month, dia =
artigo.DataPublicacao.Day, nome = TrataNome.ReescreverNome(artigo.Titulo), id =
artigo.Id, null}) </li> </ul> ...
```

Finalmente, ao acessarmos a URL no padrão que configuramos, a página de detalhes do artigo deve ser carregada com sucesso, conforme apresenta a figura 108.

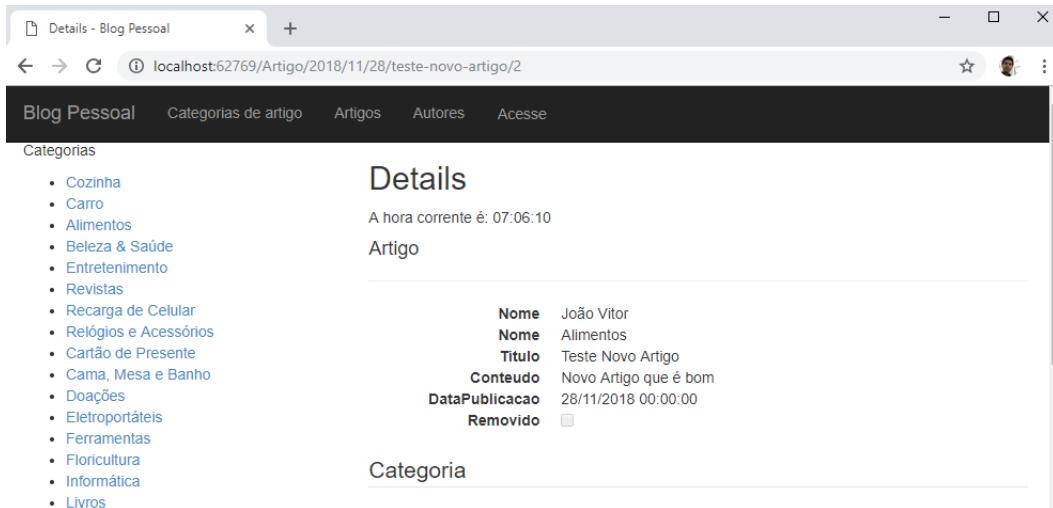


Figure 108: URL amigável na página de detalhes do artigo.

Há outra forma de configurar URL amigável por meio do roteamento baseado em atributo, utilizando o comando `routes.MapMvcAttributeRoutes()`; no `RouteConfig`. Depois precisaremos configurar a rota em nossa Action, por meio do atributo `[Route]`, conforme listagem a seguir.

```
style=sharpc [AllowAnonymous] [OutputCache(Duration = 20, VaryByParam =
"id")] [Route("Artigos/ano/mes/dia/nome/id")] public ActionResult Detalhes(int ano,
int mes, int dia, string nome, int? id) if (id == null) return new
HttpStatusCodeResult(HttpStatusCode.BadRequest); Artigo artigo =
db.Artigos.Find(id); if (artigo == null) return HttpNotFound(); return View(artigo);
```

O artigo disponível no endereço abaixo detalha o uso de roteamento baseado em atributo.

<https://blogs.msdn.microsoft.com/webdev/2013/10/17/attribute-routing-in-asp-net-mvc-5/>

Outra questão que devemos abordar é o uso semântico das tags, para isto, recomendo a leitura do seguinte artigo: <https://neilpatel.com/br/blog/tags-htm/>. Neste artigo são abordadas algumas ações que devemos saber durante o desenvolvimento das nossas Views.

Juntamente com toda a questão de tags e boas práticas, recomendo que você deixe estruturado os metadados de suas páginas. Conheceremos um pouco sobre o Schema.org<sup>28</sup>, que é uma iniciativa dos três grandes buscadores (Google, Yahoo e Bing) para oferecer um conjunto de tipos padronizados a serem utilizados com o formato microdata, como propriedades específicas para artigos. O objetivo do Schema é se tornar a principal referência na utilização de marcações de conteúdo no HTML.

Como exemplo, estruturaremos nossa página de detalhes dos artigos no projeto Blog Pessoal seguindo as propriedades que o Schema.org informa por meio deste link: <https://schema.org/Article>. Confira a seguir as alterações realizadas na View de detalhes do Artigo.

```
style=sharpc ... <div itemscope itemtype="http://schema.org/Article">
    <h1 itemprop="name"> @Html.DisplayFor(model => model.Titulo) </h1>
    <h4>   <time itemprop="datePublished">    @Html.DisplayFor(model =>
model.DataPublicacao)      </time>      Por      <span      itemprop="author">
@Html.DisplayFor(model => model.Autor.Nome) </span> </h4>
    <hr />
    <div      itemprop="articleBody">      <p>      @Html.DisplayFor(model =>
model.Conteudo)</p> </div> </div> ...
```

Use a ferramenta “Structured Data Testing Tool” da Google para validar se os vocabulários do Schema.org foram aplicados corretamente em sua página da web.

Há outras técnicas que nos ajudarão neste trabalho de melhorar o posicionamento de um site nos mecanismos de buscas globais, como:

- Redirecionamento WWW: direciona, por exemplo, [www.SITE.com.br](http://www.SITE.com.br) e [SITE.com.br](http://SITE.com.br) para a mesma URL. Redirecionar solicitações de um domínio não preferencial é importante, pois os motores de busca verificam a URL com e sem “www” como dois sites diferentes;
- Canonicalização de IP: encaminha o IP do site para o nome do domínio, por exemplo, [SITE.com.br](http://SITE.com.br). É considerável que o IP do site não seja indexado pelos robôs de pesquisa;
- Arquivo robots.txt: impedir os acessos dos robôs dos motores de busca a diretórios e páginas específicas;

---

<sup>28</sup> <https://schema.org/>

- XML Sitemap: listagem de URLs disponíveis para rastreamento e acréscimo de informações adicionais, como última atualização do site, a frequência de mudanças e sua importância.

O uso dos arquivos robots.txt e XML Sitemap permite aos motores de busca rastrear o site de forma mais inteligente.

Use as ferramentas de webmaster para acompanhamento e melhoria de performance do seu site, como “Google Search Console” e “Bing Webmaster”. Também utilize a ferramenta “PageSpeed Insights” para verificar a velocidade das nossas páginas.

Por fim, recomendo que você estude e aplique cada técnica apresentada, busque também por outras técnicas de SEO que não foram apresentadas neste material.

### 3.2.1 Google Analytics

A ferramenta Google Analytics<sup>29</sup> é responsável por colher informações relacionadas a navegação dos usuários, termos de pesquisa utilizados, local, tempo e forma de acesso, tecnologia usada, entre outras, sobre o site ou aplicativo associado.

As métricas geradas pelo Google Analytics são importantes para o desenvolvimento das páginas de um site que precisa ser indexado, durante toda a programação do site as técnicas de SEO devem ser analisadas e aplicadas, como *Uniform Resource Locator* (URL) amigável, estruturação do *HTML*, uso de *meta tags*, entre outras práticas que contribuem para a elevação do número de acessos relacionados com a pesquisa orgânica dos motores de buscas, como Google, Bing, Baidu entre outros.

Para configurar o acompanhamento do Google Analytics em nossa aplicação podemos utilizar o seguinte endereço:

<https://support.google.com/analytics/answer/1008080?hl=pt-BR>.

### 3.2.2 Redes Sociais

Não esqueça de implementar as diversas integrações existentes entre seu site e as redes sociais disponíveis. Mantenha as redes sociais de seu site ativas, comunique-se com seus seguidores, e deixe os canais abertos para comunicação com eles.

## 3.3 ASP.NET Web API

O ASP.NET Web API<sup>30</sup> é um framework da plataforma ASP.NET que facilita a construção de serviços na arquitetura REST [7], pois aplica o conceito de servir aplicações em rede, além de ser aplicável a construção de API. Com REST HTTP conseguimos alcançar uma grande variedade de clientes, como aplicações mobile e browsers. É a plataforma ideal para construção de serviços REST baseados em .Net.

A vantagem de utilizar uma API (Application Programming Interface) é que a

---

<sup>29</sup> <https://analytics.google.com/analytics/web/>

<sup>30</sup> <https://docs.microsoft.com/pt-br/aspnet/web-api/overview/getting-started-with-aspnet-web-api/tutorial-your-first-web-api>

comunicação entre os dados é realizado de forma simples, por vezes via Json, com o uso de métodos GET, POST ou PUT dependendo da requisição solicitada. Centralizar o acesso às informações, contribui para uso compartilhado dos dados, que ocorre via acesso ao site e aos aplicativos, desse modo, ambas aplicações realizam as consultas por REST consumindo da API desenvolvida para retorno dos dados.

Caso você tenha alguma dúvida sobre esta plataforma, recomendo a leitura do seguinte artigo:

[http://www.macoratti.net/16/05/net\\_mvcwapi.htm](http://www.macoratti.net/16/05/net_mvcwapi.htm)

Para maior aprofundamento sobre esta tecnologia e conhecimento de exemplos práticos, recomenda-se a leitura dos seguintes links:

- [https://www.tutorialspoint.com/asp.net\\_mvc/asp.net\\_mvc\\_web\\_api.htm](https://www.tutorialspoint.com/asp.net_mvc/asp.net_mvc_web_api.htm);
- <https://www.devmedia.com.br/introducao-ao-asp-net-web-api/25180>.

## 4. Certificações Microsoft

Antes de entrarmos nos detalhes da certificação da Microsoft, primeiro vamos entender o porquê de obter uma certificação. Conquistar uma certificação é afirmar que você domina determinado assunto e tem ao seu lado uma certificadora que afirma o seu conhecimento, ou seja, ela assina embaixo garantindo que você fez a prova e foi aprovado.

Atualmente, vemos o mercado de trabalho da nossa área, Tecnologia da informação, cada vez mais competitivo. Dessa forma, nos deparamos com um grande desafio: como posso ser um profissional mais requisitado e competitivo para atender às exigências do mercado de trabalho?

Nós precisamos nos destacar da concorrência, e um modo de ter algo a mais, é obter certificados que comprovam a qualidade de um profissional. Nesse contexto, as certificações Microsoft andam em alta no mercado.

A certificação da Microsoft relacionada com a disciplina estudada, é o Exame 70-486: Developing ASP.NET MVC Web Applications, acesse <https://www.microsoft.com/pt-br/learning/exam-70-486.aspx>. Esta prova aborda o ASP.NET MVC 5, porém apesar do nome não deixar claro, é cobrado muita coisa sobre o Azure nesta prova também.

O exame abrange uma série de temas diferentes, por isso, caso tenha interesse em tirar a certificação 70-486, recomendo que você estude firme, dedique-se muito a cada assunto que tratamos durante o curso e não esqueça de aprofundar cada tópico solicitado pelo exame.

Deixo alguns links importantes e materiais que podem ser utilizados para sua preparação:

- <https://www.microsoftpressstore.com/store/exam-ref-70-486-developing-asp.net-mvc-web-applications-9781509300921>;
- <https://global3.mindhub.com/70-486-developing-asppnet-mvc-web-applications/p/MU-70-486>;
- <http://failedturing.blogspot.com/2016/05/microsoft-exam-70-486-study-guide.html>;
- <https://www.lambda3.com.br/2016/01/guia-de-estudo-para-certificacao-microsoft-70-486/>;
- <https://www.youtube.com/watch?v=pCVOM6kIkLQ&list=PLwftZeDnOzt1LY4pDfrxLYQaRdgZd14hV>;
- <https://www.pluralsight.com/paths/mvc5>.

Acredito que vale a pena tirar certificações da Microsoft, pois além do conhecimento adquirido por meio do estudo, ao ser aprovado você já tem um destaque no mercado de trabalho e de quebra ganha descontos em e-books da Microsoft, descontos em produtos Dell, acesso gratuito a uma rede social privada para

profissionais certificados, além de materiais de divulgação das suas conquistas. A Microsoft tem programas de auxílio a empresas, como o Microsoft Partner Network (MPN), que reduz o custo dos produtos da Microsoft utilizados, uma das maneiras de obter benefícios, e descontos por meio deste programa é pelo número de profissionais certificados. Assim, vale a pena uma empresa investir na certificação de seus colaboradores já que também obtém vantagens, desse modo, todo mundo sai ganhando.

## 5. Dicas e materiais

Recomendo que você acompanhe os artigos dos seguintes sites:

- <https://www.codeproject.com/>;
- <https://www.asp.net/mvc>;
- <http://netcoders.com.br/>;
- <https://codigosimples.net/>;
- <https://imasters.com.br/>.

Porém não fiquei limitado apenas nesta lista, há vários outros sites que tratam do assunto ASP.NET MVC. Geralmente quando você tiver alguma dúvida e sair pesquisando por uma solução, na maioria das vezes você será redirecionado ao site <https://stackoverflow.com/>, por vezes, este site nos acompanhará em nossas dúvidas e problemas que tivermos.

Há dois canais no YouTube que seria muito interessante que você se inscrevesse, são eles: Canal dotNET<sup>31</sup> e Coding Night<sup>32</sup>. Estes canais oferecem muito conteúdo de altíssima qualidade e de forma gratuita, são vídeos elaborados por profissionais com muita experiência no mercado de trabalho.

Há alguns livros que quero te recomendar, para que você possa se aprofundar nos assuntos aqui abordados:

- ASP.NET MVC5 - Crie aplicações web na plataforma Microsoft, Casa do Código;
- Professional ASP.NET MVC 5, dos autores Brad Wilson, David Matson, Jon Galloway e K. Scott Allen;
- Pro ASP.NET MVC 5 Platform, de Adam Freeman;
- Azure - Coloque suas plataformas e serviços no cloud, Casa do Código;
- HTML5 Programming for ASP.NET Developers, de Bipin Joshi;
- Web Applications on Azure, de Rob Reagan;
- ASP.NET jQuery Cookbook, Second Edition, de Sonal Aneel Allana.

Realize o teste elaborado pela Tutorials Teacher, por meio do link:  
<http://www.tutorialsteacher.com/online-test/mvc-test>

---

<sup>31</sup> <https://www.youtube.com/channel/UCIahKJr2Q50Sprk5ztPGnVg>

<sup>32</sup> <https://www.youtube.com/channel/UCLoVnmvp0fYn-BCK7yKTxUQ>

Ao final veja as respostas de todas as questões, reforce os assuntos que ainda não obteve sucesso, para aprendermos e fixarmos todos os recursos precisaremos de muita dedicação.

Por fim, estude projetos do GitHub em ASP.NET MVC, ou qualquer outra tecnologia de seu interesse, aprenda com aquele código, e até tente contribuir, trabalhar em projetos open-source que são desenvolvidos pela comunidade fará você crescer bastante. Também é interessante acompanhar os blogs de desenvolvedores, de empresas de TI e também dos vários MVPs que disseminam o conhecimento de tecnologias da Microsoft e outras várias.

## 6. Considerações finais

Após toda a implementação realizada, todos os recursos que foram aprendidos, encerramos nossa disciplina aprendendo um pouco sobre a infra-estrutura utilizada em nossos projetos para publicarmos nossas aplicações. Desta forma, trabalhamos todo o ciclo de desenvolvimento de uma aplicação.

Vemos como trabalhar com projetos menores e também algumas práticas recomendadas para projetos maiores. Aprendemos padrões de projetos, além de outros recursos que nos ajudam no desenvolvimento. Entendemos da importância de construirmos software com qualidade e conhecemos ferramentas que nos auxiliam neste processo, e também metodologias que contribuem para alcançarmos este objetivo.

Pudemos conhecer a quantidade de recursos disponíveis pelo framework ASP.NET MVC. Este link disponibiliza os recursos disponíveis, visite sempre que tiver dúvida ou quiser aprofundar algum assunto, acesse:

<https://docs.microsoft.com/pt-br/aspnet/mvc/overview/getting-started/recommended-resources-for-mvc>

Caso você desenvolva alguma aplicação que precise de bom posicionamento na internet, não esqueça de aplicar as técnicas de SEO, como URL amigável, estruturação do HTML, uso de meta tags, entre outras práticas que contribuem para a elevação do número de acessos relacionados com a pesquisa orgânica dos motores de buscas, como Google, Bing e Yahoo. Não esqueça também de analisar as métricas geradas pelo Google Analytics, pois elas são importantes para o desenvolvimento das páginas.

Estude muito cada tópico apresentado neste livro, não se limite as linhas aqui descritas, vá além, aprofunde cada detalhe visto e pratique bastante, quando falamos de codificação, precisamos ter o seguinte norte: quanto mais estudamos e programamos, melhores profissionais nós ficaremos. Se você quiser aceitar mais um desafio, recomendo que você busque realizar a prova de certificação do Exame 70-486 Developing ASP.NET MVC Web Applications, desse modo você precisará estudar muitos assuntos que, por vezes, foram apenas iniciados ou citados nesta disciplina.

Por fim, fica registrado meu agradecimento pela oportunidade em te ajudar neste caminho de aprendizagem visando sua profissionalização. Desejo-te bons estudos e muito sucesso!

## Referências

- [1] Mauricio Aniche. *Test-Driven Development: Teste e Design no Mundo Real com .NET*. Editora Casa do Código, 2014.
- [2] Everton Coimbra de Araújo. *ASP.NET MVC 5*. Casa do Código, Maio, 2016.
- [3] Andrews Ferreira Bárbara. .net: Entendendo o arquivo web.config, nov. 2018.
- [4] Isabel Castillo. Comparison: Create a table with pure vanilla javascript versus jquery, out. 2018.
- [5] DEV MEDIA. Asp.net mvc e identity: Autorização de usuários com claims, out. 2018.
- [6] Dino Esposito. *Programming Microsoft ASP.NET MVC, 3rd Edition*. Microsoft Press; 3 edition (February 25, 2014), 2014.
- [7] R. Fielding. Representational state transfer. *Architectural Styles and the Design of Network-based Software Architecture*, pages 76–85, 2000.
- [8] Jon Galloway, Brad Wilson, K. Scott Allen, and David Matson. *Professional ASP.NET MVC 5*. John Wiley & Sons, Agosto, 2014.
- [9] Portal GSTI. O que é teste de software?, nov. 2018.
- [10] Renato Haddad. Asp.net - uso de bundle e minification no asp.net mvc 4 e asp.net web forms 4.5, nov. 2018.
- [11] José Carlos Macoratti. Asp.net MVC 5 - usando o atributo uihint - i, nov. 2018.
- [12] José Carlos Macoratti. Asp .net MVC - tipos de retorno dos métodos action, out. 2018.
- [13] José Carlos Macoratti. Asp .net - apresentando a sintaxe razor para c# e vb .net, nov. 2018.
- [14] Zoran Maksimovic. Microsoft.net o/r mapper: choose your own!, nov. 2018.
- [15] Microsoft. Como: Gravar um cookie, nov. 2018.
- [16] Microsoft Developer Network. Visão geral do asp.net mvc, set. 2018.

- [17] Jeffrey Palermo, Ben SCHEIRMAN, and Jimmy BOGARD. *ASP.NET MVC em Ação*. Editora Novatec, Janeiro, 2011.
- [18] Eduardo Pires. Asp.net MVC - action filters - entendendo e customizando, nov. 2018.
- [19] Eduardo Pires. Asp.net MVC - viewdata, viewbag e tempdata, mai. 2018.
- [20] Eduardo Pires. DDD não é arquitetura em camadas, nov. 2018.
- [21] Rafaela Pozzebom. O que é componentização de software?, nov. 2018.
- [22] Fabio Gomes Rocha. Tdd: fundamentos do desenvolvimento orientado a testes, nov. 2018.
- [23] Fabrício Sanchez and Márcio Fábio Althmann. *Desenvolvimento web com ASP.NET MVC*. Casa do Código, Novembro, 2013.
- [24] Jhonathan Soares. Diferenças entre renderpartial vs renderaction vs partial vs action no mvc, nov. 2018.
- [25] Jhonathan Soares. Habilitando compressão gzip com web.config em asp.net, nov. 2018.
- [26] Microsoft Visual Studio. Downloads visual studio 2017, ago. 2018.
- [27] Tutorials Teacher. Asp.net mvc - tempdata, ago. 2018.
- [28] Júlio Viegas. Teste de software: Introdução, conceitos básicos e tipos de testes, nov. 2018.