

ASP.NET MVC e Web API

por: Giordano Lins



DETRAN-PE





O que é MVC?

O padrão arquitetural **MVC** (**Model-View-Controller**) separa a aplicação em três componentes principais:

Model

Responsável pelo domínio, negócio e acesso à dados da aplicação.

View

Responsável pela apresentação da aplicação.

Controller

Responsável por gerenciar as requisições, ações e rotas da aplicação, e por integrar e gerenciar a comunicação dos outros componentes.



DETRAN-PE

IVIA®

O ASP.NET MVC é um framework leve e altamente testável que integra com os recursos existentes do ASP.NET, como por exemplo *master pages* e autenticação baseada em membros.

Aplicações ASP.NET podem utilizar o framework MVC, outras continuar usando o padrão tradicional (baseado em *Web Forms* e *postbacks*), ou combinar as duas maneiras, já que nenhuma delas é exclusiva à outra.



O padrão MVC ajuda na criação de aplicações que separam os diferentes aspectos da estrutura da aplicação (lógica de entrada, lógica de negócio, lógica de interface de usuário), ao mesmo tempo em que provê o desacoplamento (*loose coupling*) desses elementos.

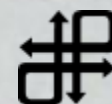
- O padrão especifica onde cada tipo de lógica deve ser colocado na aplicação.
- Gerenciamento da complexidade da aplicação.
- Foco um aspecto da aplicação por vez.
- Promoção do desenvolvimento paralelo

Suporte ao TDD (Test-Driven Development)

Além de promover o gerenciamento de complexidade, o padrão **MVC** torna mais fácil de testar as aplicações. Mais fácil inclusive do que testar aplicações baseadas em **ASP.NET Web Forms**.

Web Forms: Inicialização da classe Page, controles filhos e classes dependentes adicionais, requer até um web server.

MVC: o desacoplamento através do uso de interfaces torna possível testar componentes individualmente, em isolamento até do resto do framework.



DETRAN-PE



Vantagens de uma aplicação baseada em MVC

- ✓ Melhor gerenciamento da complexidade através da separação dos componentes.
- ✓ Não usa *forms* baseados em *view state*, o que permite maior controle sobre o comportamento da aplicação.
- ✓ Usa o padrão *Front Controller*, que processa as requisições através de um controlador centralizado, oferecendo uma infraestrutura de rotas rica.
- ✓ Recomendado para aplicações web que são mantidas por times grandes



DETRAN-PE



Vantagens de uma aplicação baseada em Web Forms.

Suporta um modelo de eventos que preserva o estado sobre o protocolo HTTP, o que beneficia o desenvolvimento de aplicações voltadas ao negócio.

Usa o padrão **Page Controller**, que adiciona funcionalidades individuais às páginas

Usa formulários de servidor baseados em *view state*, que facilita o gerenciamento da informação.

Recomendado para equipes pequenas, para tomar vantagem do grande número de componentes disponíveis e para o desenvolvimento rápido de aplicações.

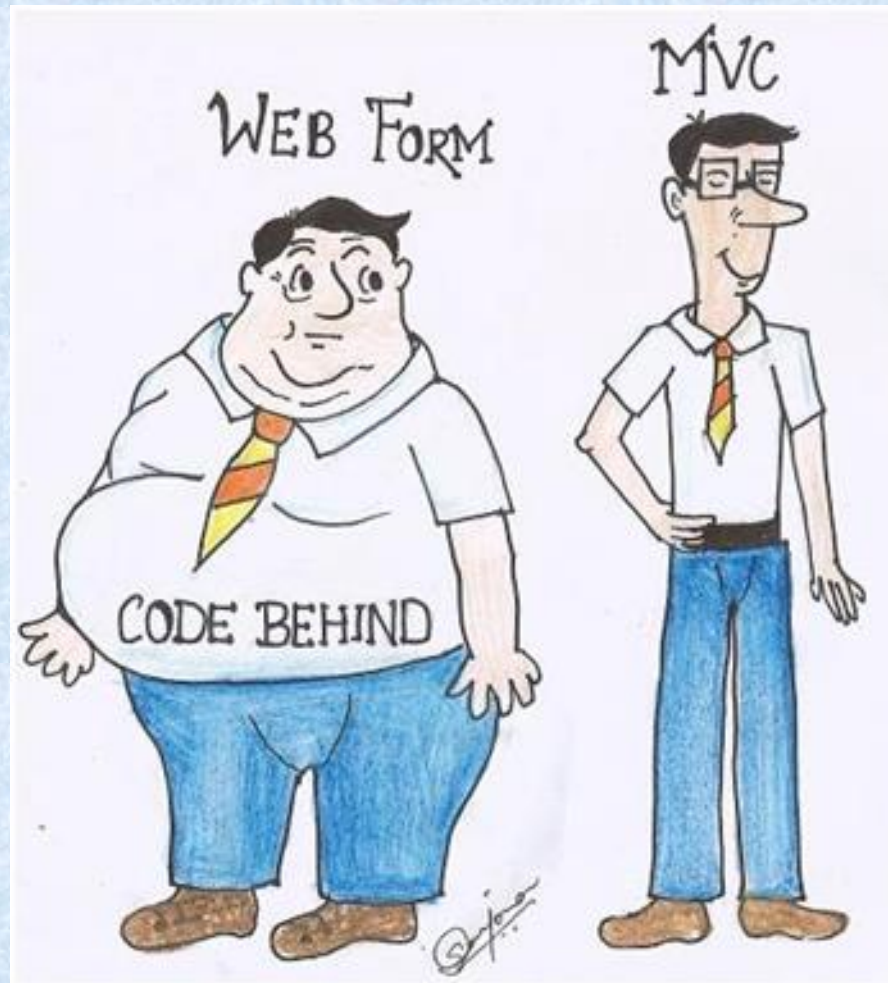
Em geral é menos complexo para o desenvolvimento de aplicações porque os componentes são integrados e requerem menos código do que o modelo MVC.



DETRAN-PE



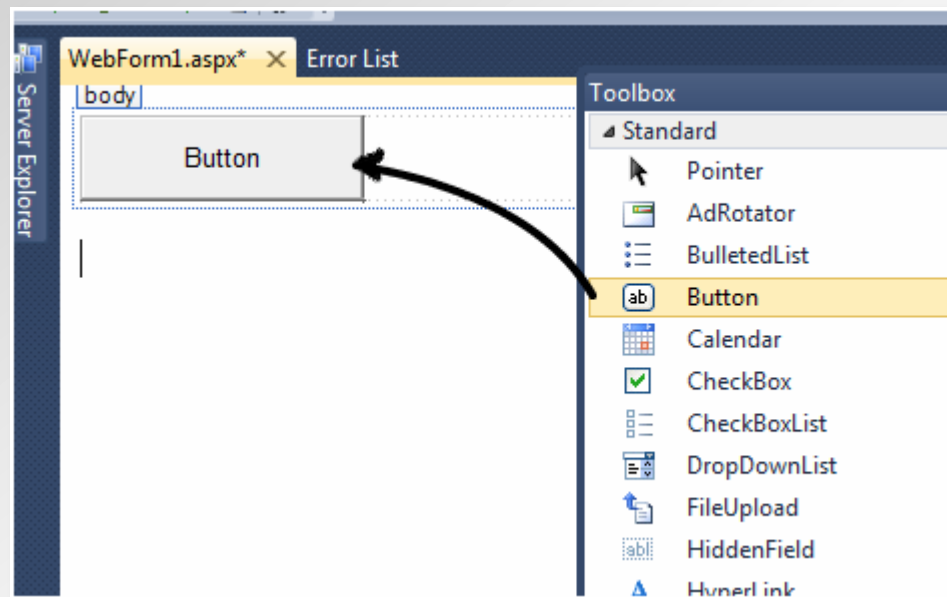
5 "*outs*" do ASP.NET Web Forms



Como o Web Forms funciona...

Se olharmos de perto a tecnologia *Web Forms*, perceberemos que ela é um "approach" **RAD / VISUAL** de desenvolver para web.

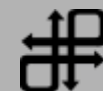
Em outras palavras, o desenvolvedor arrasta e solta os controles na ferramenta de design, e o Visual Studio codifica a lógica por trás.



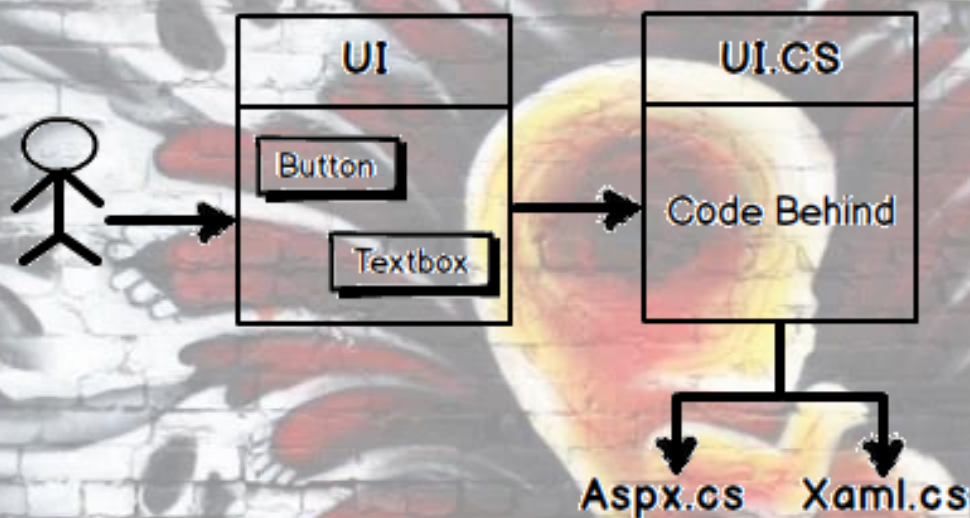
Ou seja, quando o desenvolvedor arrasta e solta um controle de botão, por exemplo, na ferramenta de design, um objeto botão é criado por trás e o desenvolvedor codifica os eventos deste objeto:

```
public partial class WebForm1 : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        // Desenvolvedor codifica aqui
    }

    protected void Button1_Click(object sender, EventArgs e)
    {
        // Desenvolvedor codifica aqui
    }
}
```



Então, quando o desenvolvedor arrasta esses elementos e codifica seus comportamentos, no "*back-end*" de forma inteligente e silenciosa a Microsoft codifica lógica em arquivos de classes parciais ASPX.CS:



Esse é o sucesso da tecnologia, já que poupa o desenvolvedor de muitos detalhes técnicos como gestão de eventos, *delegates*, protocolo HTTP (POST, GET, etc), gerenciamento de sessão, etc.

Mas devido a forma como o código (*code behind*) é disposto e invocado, há cinco problemas sérios.



Problema 1 - Solução baseada no que se vê (View), para um problema baseado no que se quer fazer (Action)

No fim das contas sites e aplicativos web são usados por usuários!

Se um usuário tiver o propósito de comprar em um portal, ele vai comunicar seu objetivo com as ações de:

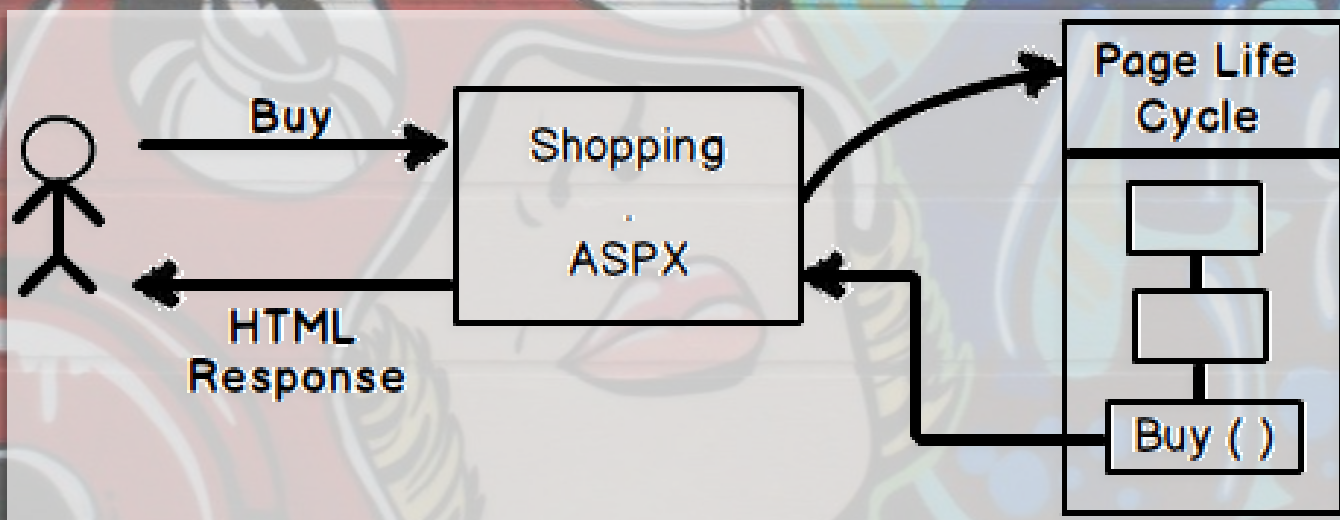
- Comprar Produto
- Imprimir Fatura



DETRAN-PE

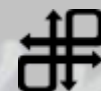


No fim das contas a aplicação acaba com uma tela como "Compras.aspx" que por sua vez executa o código contido em "Compras.aspx.cs" que executa um ciclo de vida complexo da página que por sua vez vai executar a ação desejada pelo usuário.



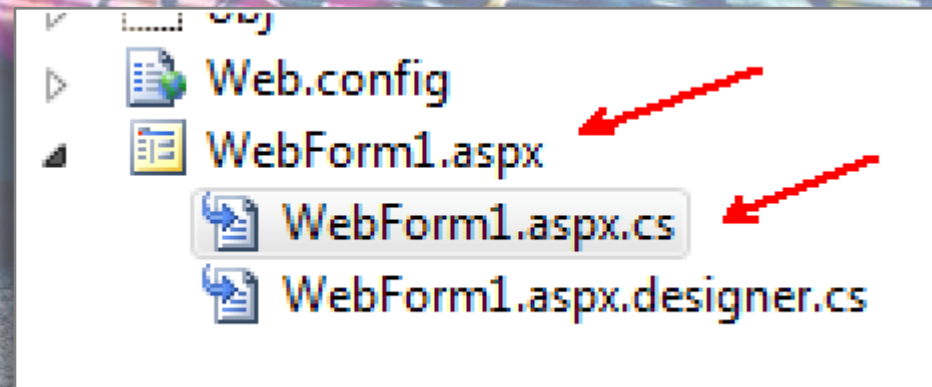
E como podemos criar uma estrutura que trate a ação primeiro em vez da *view*?

É exatamente o que o MVC faz. Primeiro ele pega a ação que pertence a um determinado controlador, e este por sua vez retorna a *view* apropriada.



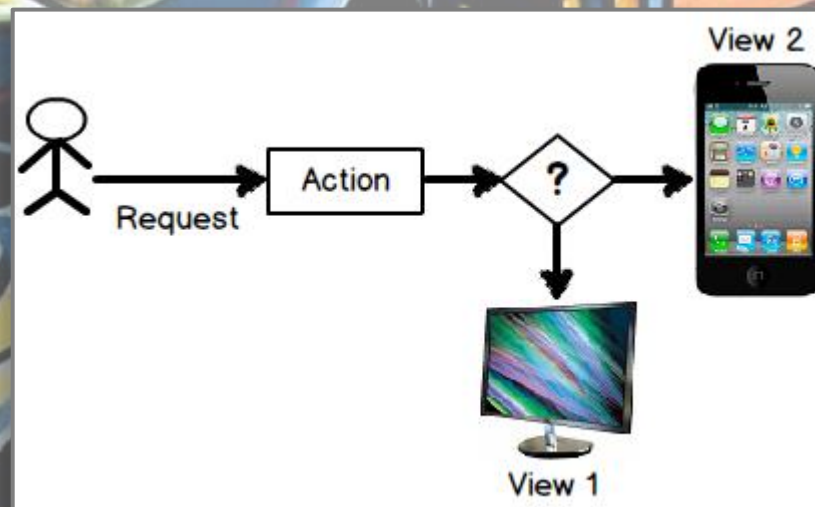
Problema 2 - Efeitos colaterais de uma má arquitetura: Alto acoplamento

Uma vez que se começa com a arquitetura errada, acaba-se realizando ajustes e no fim há sérios efeitos colaterais. O *code behind* que parece estar fisicamente separado em diferentes arquivos na verdade nunca foi desacoplado. O arquivo ASPX.CS não pode ser separado do ASPX.



Sendo assim, se mudarmos a arquitetura para uma baseada em ações, poderemos reutilizar a mesma ação com diferentes apresentações.

Por exemplo, se um usuário acionar a ação "Mostrar", ela pode invocar diferentes apresentações, como "MostrarDesktop.aspx" ou "MostrarMobile.aspx" dependendo do tipo de dispositivo.



Problema 3 - HTML não é único tipo de resposta!

Por conta do alto acoplamento do *code behind* o tipo de resposta é fixo no *web forms* (o retorno é em HTML).

Para mudar este comportamento o esforço é gigantesco e complexo (manipulando Content-Types e os métodos de "Response.End").



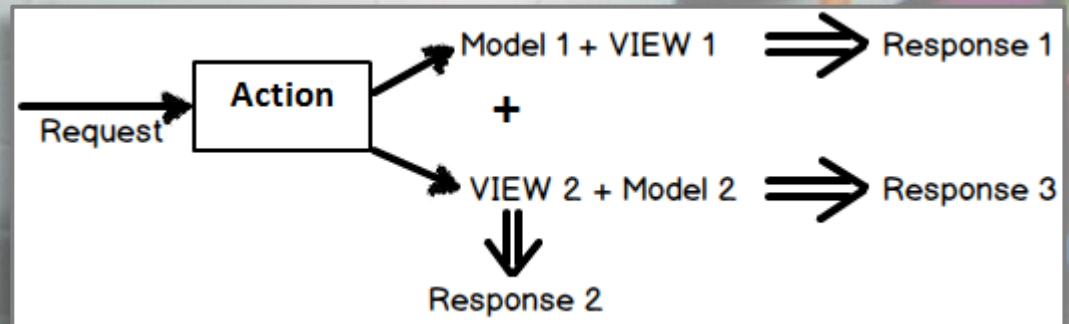
Em MVC, uma estrutura de ação pode ser criada para responder com diferentes tipos de conteúdo de maneira simples e eficiente.

```
public ActionResult Index(string viewType)
{
    if (viewType == "JSON")
    {
        return Json(new Customer(), JsonRequestBehavior.AllowGet);
    }
    else
    {
        return View("DisplayCustomer", new Customer());
    }
}
```


Problema 4 - Combinação inflexível da apresentação com os dados

Quando o **Web Forms** processa um resultado, é a *view* que toma as decisões de como manipular os dados. Isso claramente viola o SRP (dos princípios SOLID).

Mas, em uma arquitetura baseada em ação, a tomada de decisão é feita pela própria ação, e a combinação da apresentação com os dados é feita de forma mais eficiente e apropriada.

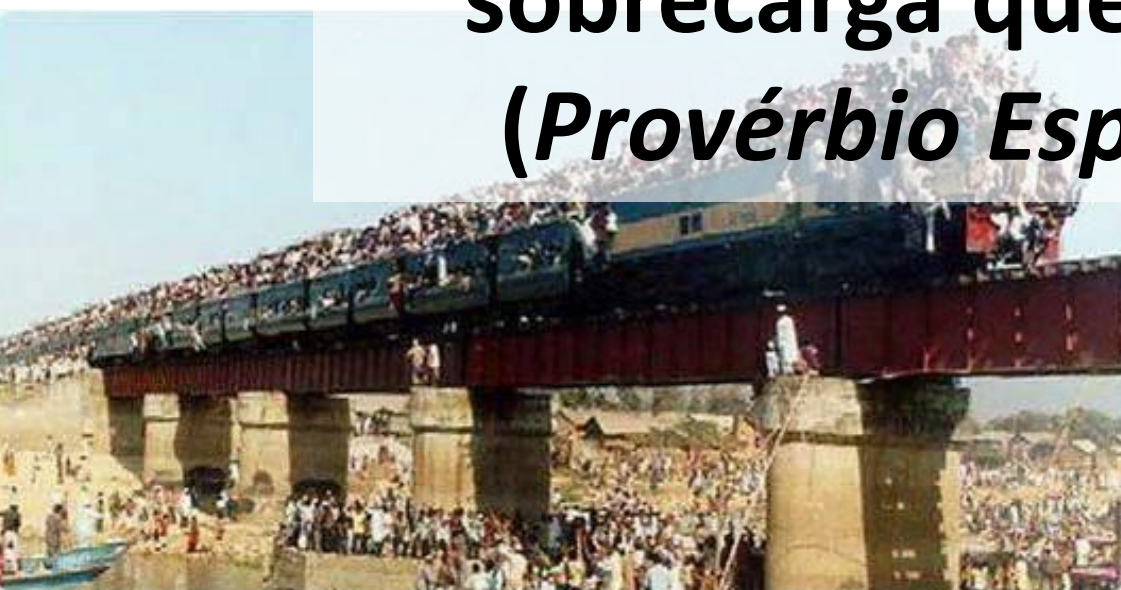


Entendendo o "S" de SRP (Single Responsibility Principle)

```
class Cliente
{
    public void Add()
    {
        try
        {
            // Acesso a dados aqui
        }
        catch (Exception ex)
        {
            System.IO.File
                .WriteAllText(@"c:\Erro.txt", ex.ToString());
        }
    }
}
```




**“Não é a carga, mas sim a
sobrecarga que mata.”
(*Provérbio Espanhol*)**



Em MVC é possível criar códigos como a seguir, onde o mesmo *Model* é anexado a diferentes *views*.

```
public ActionResult Index(string ViewName, Cliente dadosCliente)
{
    if (ViewName == "Detalhe")
    {
        return View("DetalhesCliente", dadosCliente);
    }
    else
    {
        return View("Cliente", dadosCliente);
    }
}
```


Problema 5 - Instanciar o *code behind* como uma classe normal para testes unitários é praticamente impossível

O código por trás de um *web form* é tipicamente carregado e dividido em classes parciais, e não pode ser instanciado como uma classe normal C#.

Nem o teste de interface é fácil devido as nuances da arquitetura dos *web forms*.

```
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }

    public void Button1_Click(object sender, EventArgs e)
    {
        Session["SomeSession"] = "Is this set?";
    }
}
```

HttpException was unhandled by user code

Session state can only be used when enableSessionState is set to true in the configuration file or in the Page directive. Please also make sure that you have registered System.Web.SessionStateModule or a custom session state module in the <configuration>\<system.web>\<httpModules> section of configuration.

Troubleshooting tips:

[Get general help for this exception.](#)

[Search for more Help Online...](#)

Actions:

Então MVC é a solução? O que se perde com ele?

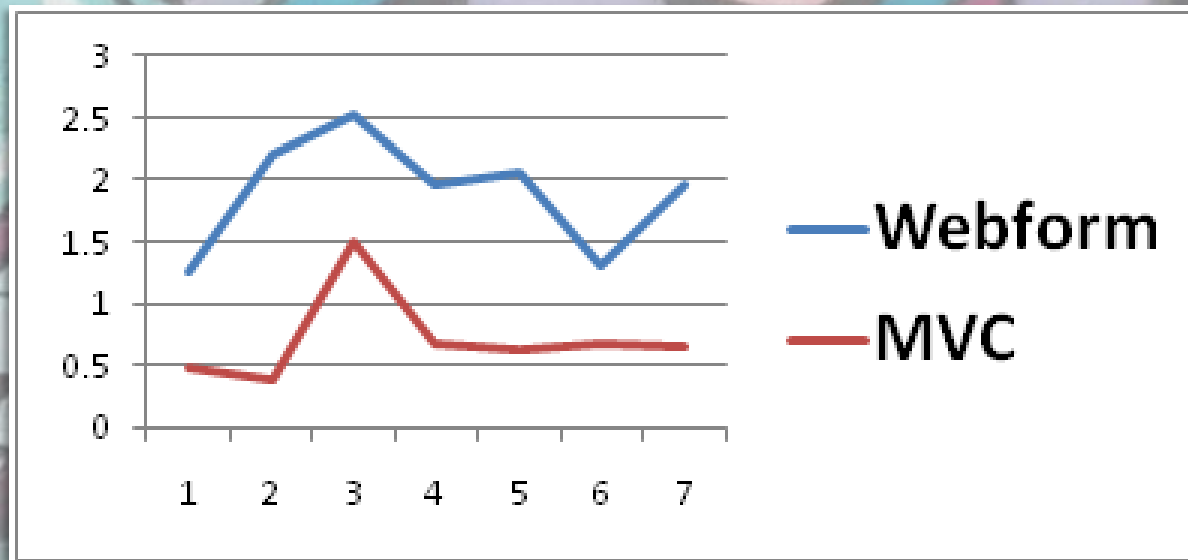
A grande vantagem dos Web Forms é a programação RAD / VISUAL

- No MVC não há ***Server Controls***
- No MVC não há ciclo de vida de página
- No MVC não há *view state*

Mas só isso mesmo :P

E o que se ganha?

Velocidade!



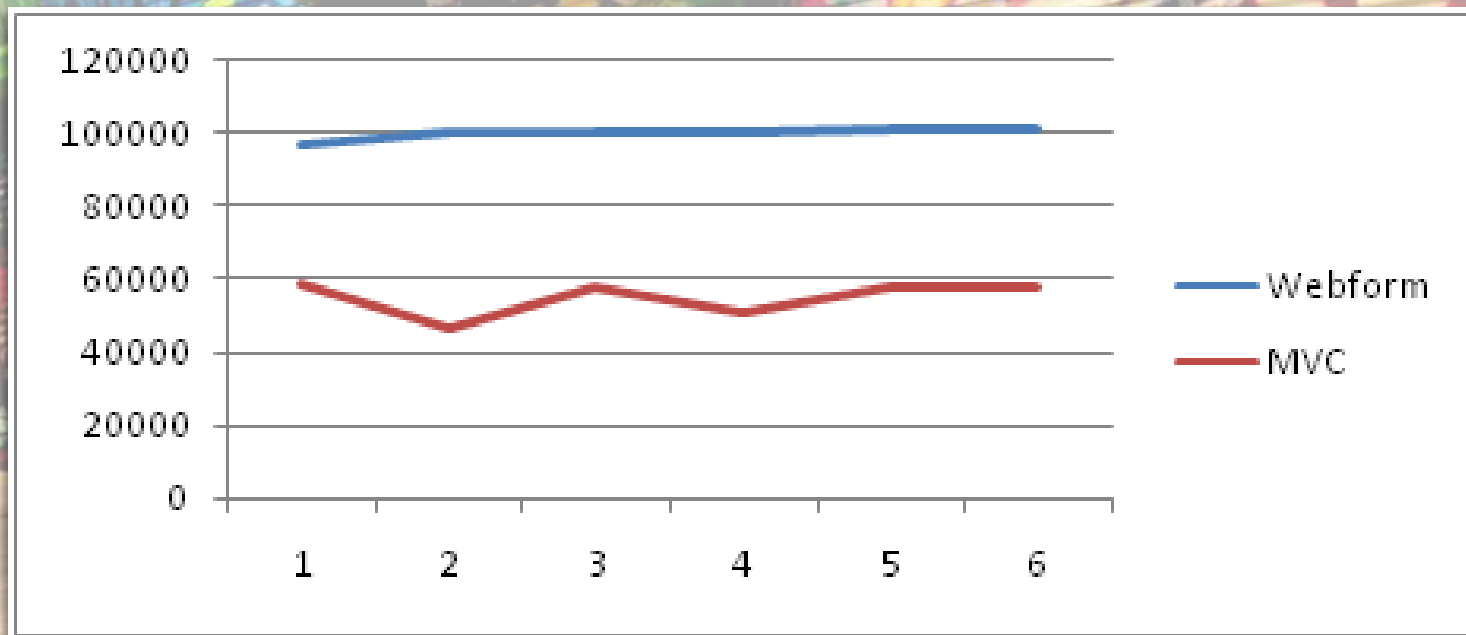
**Comparação do tempo de resposta entre as duas arquiteturas
(comparado apenas a requisição, sem os *assets* para não
influenciar os testes)**



DETRAN-PE

IVIA[®]

Mais velocidade ainda! E menos carga.



**Comparação do tamanho do conteúdo de resposta entra as
duas arquiteturas**

Quando visualizamos o código fonte da página, encontramos muitos dados gerados pelo web forms que não existem no MVC. Isso significa que menos largura de banda é consumida na navegação.

```
<body>
  <form method="post" action="WebForm1.aspx" id="form1">
    <div class="aspNetHidden">
      <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
        value="wFRayhq09cK735FA/bhsy7ds1d0NoQ8BnfUj1i3twCvnpME6Q0LVRlh/Rx10E0epzEz6XyqChdzCj5BNF9LRFQzZd+SAH3KzF3vLxp92Hn0KzATU8Na3y0VlaXgPDuBvRLu61xskXkQ6q817f71
        Gkx9L8DGOvzpU35c1PD81XgCUG0ra7NbdwL3HT8QsLM3Zf1EehAXet8YUTAEbuRHRiKoactcnVdy8c8UABVjnn0zo9oHwU11x3/G3E0Nan/BTXsM4jrbzLoQONcv7yQQsshIVD1KGP6/Q17LesvbNcPe2
        cmQOg+FX0CK7LLX5KzIzbx+HwUyeykqH80CzX0E0UQ66XZSSU1qnpOILA8BacE/66mLQ1VP0uPQCFSNDCHV8uLH5C1YV7Su9ywl30H1Z3JQ5gE1twx004CnoD1Pvq+H4IEId20TCf3xH84zR9aX1zqUmw4
        7Kqz0pDiPMV87GNH+p4kSGqCEHjDg8z1s0r3Q1d9qJUXupUx0NAvcIeSut+P1010yq1d0tHTMRnOrbd5bWmJeo8myTnDbHC06z08/3y20kYw9z8EipVM898y2m0ldz1w1YPw1hThZvrd+EARlfxywk3w1xN7:
        rSYH4navPLD2nPlly9bW63PjnhUvVh13DdfHjSySRIAG0u4yf8Hcah6HcdtrSC3IKZ6XZfnfmoPPE1f5z3bh0Rpd6y0GK3H+uQ0y3Y0trBARRPySash80xpgH4Ax7cK5YmLNIzqnrsx04vHQ2pZUT1lyf
        Q0uH7t1cxhd046E6fV3QcSaIKKc6F3H2IZNyNaSVkoDn+ssQ/p6IgtLyaQf24HjrcvFNDP6y5XntucP8RiE1NbHgrhplnRwQVfn7+/6jse8X0SoRRRevH5S6my2oJYHkXF/s66vXthChc27Q87gvZ1/0v
        P2h3o0906Ra3LURdsEgk579WmCYk0+svQ00EKA/EmAaTa+11Ac+MwXwrtY3NT3Yz5QFb1aBefqJ4K7JXz7D5z5XKSZCmudKps5fv311CRTQ65ktXYCKHukMhA7EpTX0pkWp4S2AD0E0ZwV73jDXXQ+fur
        hA0yC60Vzrh90bDen8Xgg9wH57773Tg8ing7QqPbh8TVAtwHwHY/7aDKrvq6rK9G0gbPZvc4wIaaC75ZnXfL8w2Bq1eaZ3CA3CIu+3xpFEns1zC6GRY1HbSdFanXc0e0TC1gl0fuAtazR5bGK79KXK8t
        1KDMWY0A1hwa69066fk04VkhQV2eq/Umeom8pcDeeC4qHs56t8n0G5UWPshTRPMV1rtaHvc0yc8Peh1n10ov13gdI00oPn/wb9t305FidF51AXNg9ESUuA/IodAjf+/nD1ZguLvx46puCR9d0Vh208X1bori
        mwKuk1WmrI92aH4d8m3sojH+seIcsjfucxQ8806uYh5K48KeIdJUsdCnh57CPc9t1+b07U0Decy8iFXiYfK10vYxx09T35Fj0frj7HafzZx6kZqsdt11QmbsnCqU+YtDWG0ZTP9Qcs/8s77Xj0vJc08si
        yHQ8ArPmF9oGxXJ00uKuaKqH+8K9svH0DQazQMB0s28Hxk+2VBNZJPjSEV95ChkQ0hEIZKK2+YHnXPKFus290SIRn1Lex8d5/8/zbfw82TwAkRyRmu3uNV67n2NuhDf8/VLSbe+Z9oq4zb821V30jG51F4c
        4g4Xa/OlVfD0ck1d8vcB1lpBkdxCsHFjYX35Y0Lrfs0o2L4uyU0jy+B+khv1hwbs1ghXckX3Aab8YtdYp+63A/9etzxmeEuJR/dh5kUeg4ta7X+q6KDuQ8k5gpDmIW0Y9AZv25FP9/Z3Q0d5G4sqLD5mf
        3611c4QpGT11ztya+QZm6C6W0DhNEHrJ55z2m3sLLV0ZOSrJef/aAYCr6Gc5UfRf93zoara3PQ20oC4NM/zfucJt203NaD7Hr5vLC0nc4m1CnByanRjPg8Xtq+4NakX17VRYw154EzH941j7L37A140rC
        jK2nRLf1d3Q1d8DkAp681q2t9I1asw20qcX00e6XtFcTq5Efi1EHRUXVsp1jkkfoto770/Knj1oiMUSQ5Z0GZdTckKBEpVe2He1gtR4cxk7AF1yyH2f5srfSU7110VgqQPEIdciw68cITQ1F2TK5Y94Dq8M
        EX08Nv0d0Z108tegdUSREthmLvdV+Ytb2Z7s8xryQ0Q+A51Pb6/PmQXaunrveA13xUPk1oX110q1/+1y512Bk1+0uoJ+X2sAht0Y7x8KsPDBuAmu5EHf1RzbmHf0JcatuJ140018e0e+/JL7t9/zgX01f
        tLx0+dtbu71k1j70oTKF429AX/GHdTK6X09sXcTmwXsSroQ7P8TnIVBj69ydx+eHXLVukTTy0Zq56V550F8M23Q1qoHqTKLH6Grzj+anuJjsjgZ6CyhQ9213LYB157Xf091bH7TgdGr5wqePz9Nyq9CquU
        I6W4N4Z08N454un3FskX/pj4e56iPHiYk03PqTs1tuloUwhs57/qMALU88Xqu2Fz+RLN59E2GP1DDc4dNhtsYc+q0k0NP42bkzn8x5Z0c2FckR1HsRsk90WGD0g8MebrTeRaL126bu7jGnKf1g7q88
        CMp1gjs1xe85o/c2HWvcfc7X48Q7P1Pks0ty+0m/q4C6LHdGLXtLnkTISHC0THu4ovvHlJmnsE3H/v0R0D8zPgr16b2VIQ/iCsp+BP0Qd257jP7MR1H0sssc2RL0H157P5u9Y4e19Xr01mjhaoj2Poi
        KIS/POCobyBF3dxs1H7xngD09v0jcs3EE09NgikUG871ChIP7a0L7cb/ssn/7T+v2wahpdgm1e0pp/bf4P/PF65+mc2bdfCW6FVQECuLFXtRnR0H2RQXS11IhtK0GwF5Tht7eHGxdNT1t8Rk/bjMrsILi
        VHMdIeyxpt2opemfhrJ5R1tHKECgk16bk0L3u7JQZ0h0Q1c4jEY26m12DyrcPUCh1BaYagQ8Hq8c28G/R4EH+BwcQmc5J6d0zqnd5d0+3kUrcI5u0R2Vg7Uh1ES/WUPzhWfK0BuWv/+3gVcXBpWz8
        tMvYhDoyymvua8IurF8t/VzRw0lyq3BhAK5638AeuQ3Au7hyydRrcUuHf2g+eqPwadyLOArVufZ/Qq7Uzheok1b3j0omDsw4Uv4gNAousff04Xs2f4yoxtdfc8BIVxcUUD0SqmUJZsK/z3oNQCIE+0Dq
        vG8bsn8SHs1L0w78yVHTYfy1skXJ3uRzISGQpKottVWdkV7ID6FQ7nbZnJaM7VpNwAdFP3q2PaQRM2VPA7Nhab9zkelMRVR55/zMEIdeBmZERR0VfpGZt91bd1wzWAFrXPaxjyjpNuecsdq15kdZaa
        /nxtNF+74CNPfHbPx5+E1ych3XwLBurIukcRPNcd19GdHwQdXexF092Kz5D0PC7Rw111huo7FDnodp4A/Hd/n1wEFHdIPQLa7t1gYB5uSnkgfBCn67YpQot7zthJ0HsxEx24HQ01+4NabH5KvP+C4gd
        xm2f650C440qj6dU6j25s0mPwU11KMRK0M2n9v3j5eFURKpVUE08rfVXc+q2vQdHt60s+fz5pYrYkADW19Kde4xtLOEL6UkqBQ9M92M7meP97wPmYx5tdpBv9C8G7Zc14Pdadu0U8Hh/dF19Ag
        xp1FMANc1IvgJ1a77YH0bnTXuZeZTh0b/7h57zJk3fd7Q6x9T/mY38fucmKCKX0uacTckZHSu5SaxKcAja+RaaT8hkU495U9R0tnxpy44jRxtgdvDmwHf8oTxHkR9qxhVklLahMvNvEzjfn7HY9FRP09pK1Z
        X68N80e82U0feTxCec75evvDwD0KLL0MDH2+c+0RssLVV1VY713T0FFvBeGe5VvGcG30EsFLs2X9tE11R1w13aeeLs2ERX1vX7305Zcdo8vFAJKd+hFR0T+7anV90NCEHTKBFdwSc1Nf080F/a8A
```

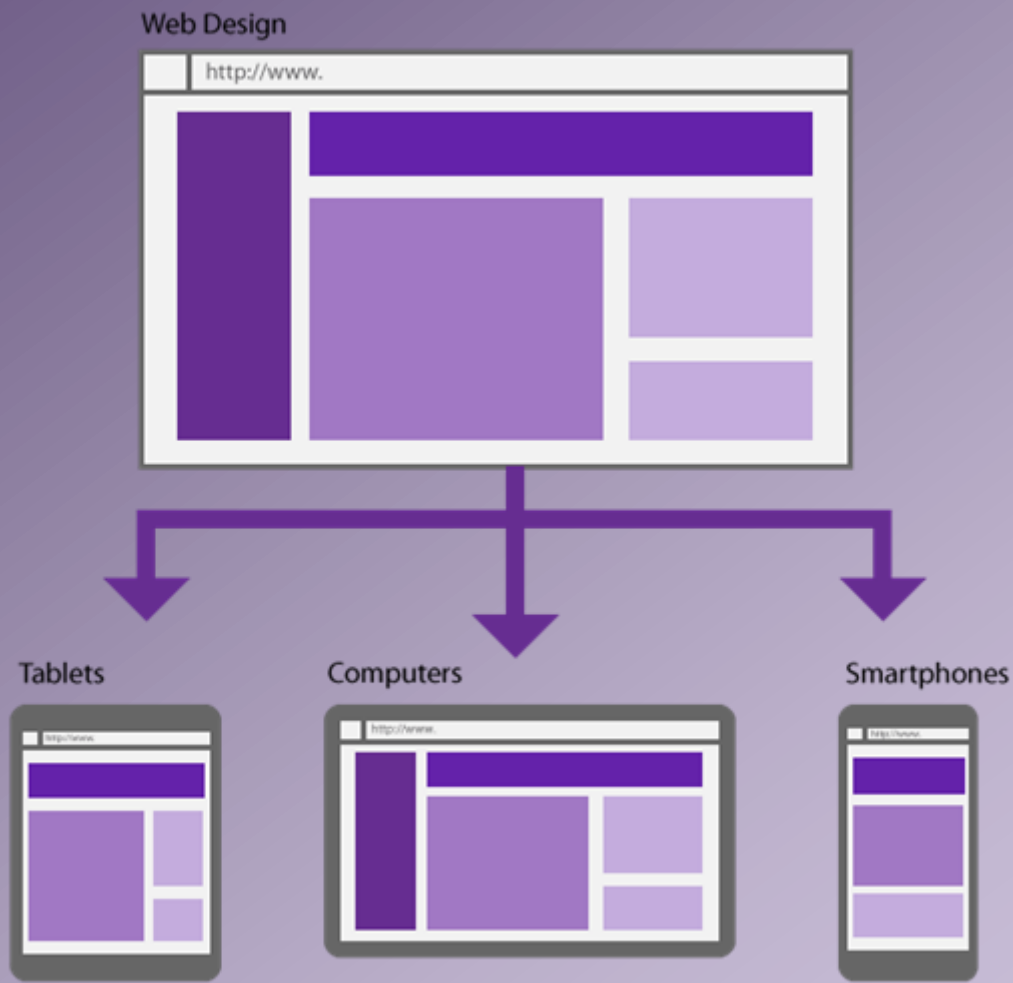


DETRAN-PE



E como é o ASP.NET MVC na prática?

ASP.NET Web API



DETRAN-PE

IVIA[®]

O que é uma API?

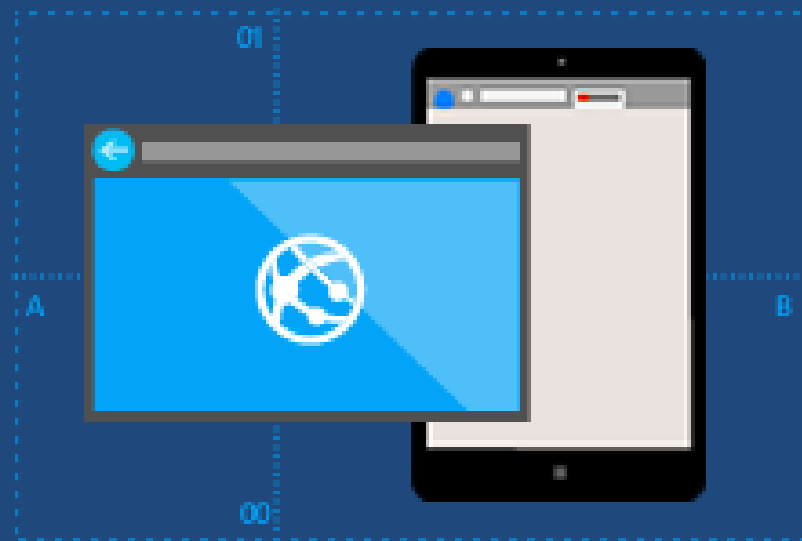
Uma API (*Application Programming Interface*) é um conjunto de instruções e padrões para acessar aplicações baseadas em web.

Uma API é uma interface *software-to-software*, não uma interface de usuário. As aplicações conversam entre si sem a intervenção ou conhecimento do usuário.



ASP.NET Web API é um framework que facilita a construção de serviços HTTP que alcançam uma gama de clientes, incluindo navegadores e dispositivos móveis.

ASP.NET Web API é uma plataforma ideal para a construção de aplicações *RESTful* com o .NET Framework.



REST

(Representational State Transfer)

É um protocolo baseado na comunicação *stateless*, *client-server* e *cacheable*, funcionando em cima da arquitetura do protocolo HTTP.

A idéia é que em vez de usar mecanismos complexos como CORBA, RPC ou SOAP para conectar dois sistemas utiliza-se chamadas HTTP.

REST comparado com SOAP

Requisição SOAP:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-
encoding">
  <soap:body pb="http://www.acme.com/phonebook">
    <pb:GetUserDetails>
      <pb:UserID>12345</pb:UserID>
    </pb:GetUserDetails>
  </soap:Body>
</soap:Envelope>
```

Requisição REST

<http://www.acme.com/phonebook/UserDetails/12345>



E como é o ASP.NET Web API na prática?



Van Gogh é arte.

Grafite é arte.

Programar é arte.

Curta arte.

Faça arte!

;)