
ASP.NET MVC e Web API
por: Giordano Lins
Microsoft CERTIFIED Technology Specialist
Microsoft CERTIFIED Professional
DETRAN-PE
IVIA

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.

Controler

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.

DETRAN-PE IVIA

O padrão MVC ajuda na criação de aplicações que separam os diferentes aspectos da estrutura da aplicação (logica 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

DETRAN-PE

IVIA®

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 idividualmente, em isolamento até do resto do framework.

DETRAN-PE IVIA

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 IVIA

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

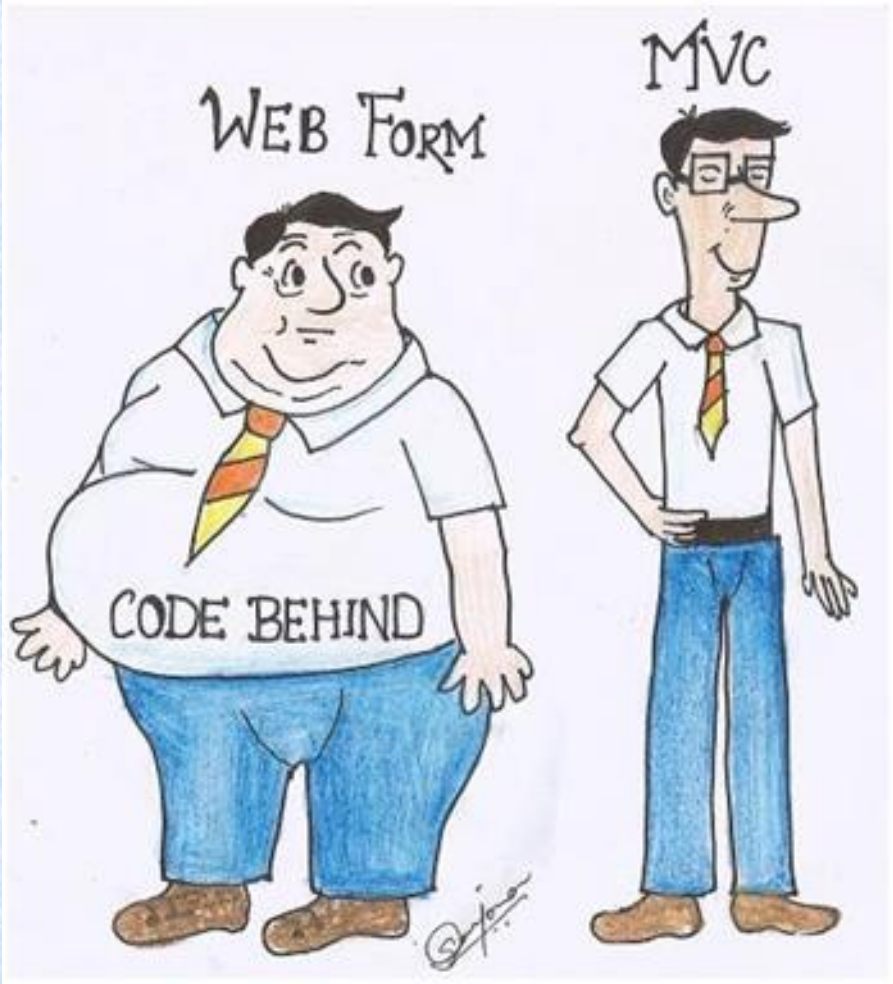
Usa formulários de servidor baseados em ***view state***, que facilita o gerenciamento da informação.

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 IVIA

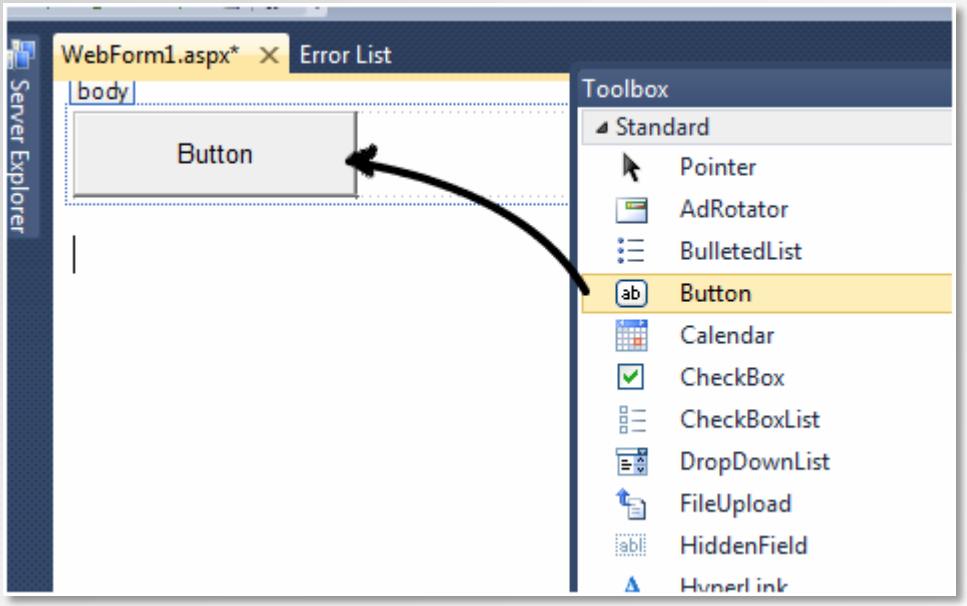
5 "*outs*" do ASP.NET Web Forms



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.

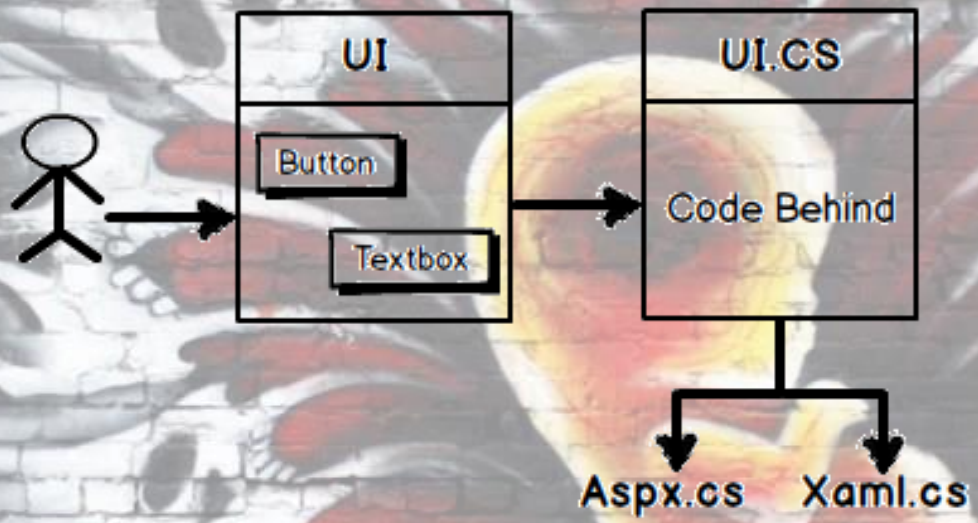


Ou seja, quando o desenvolvedor arrasta e solta um controle de botão, por exemplo, na ferramenta de design, um objeto botão é criado por trás e o desenvolvedor codifica os eventos deste objeto:

```
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 codigo (*code behind*) é disposto e invocado, há cinco problemas sérios.

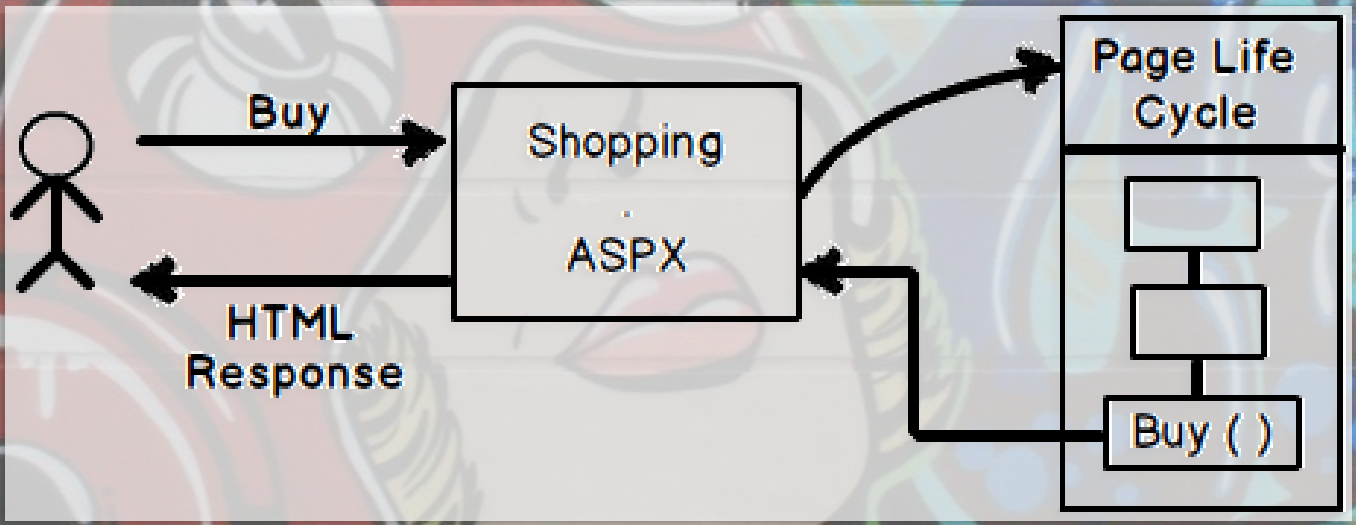
Problema 1 - Solução baseada no que se vê (View), para um problema baseado no que se quer fazer (Action)

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:

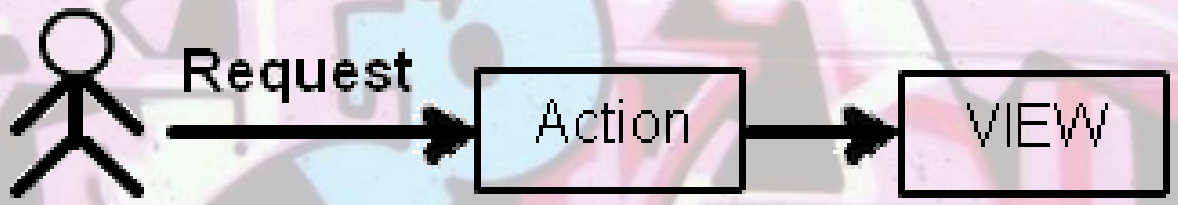
- Comprar Produto
- Imprimir Fatura

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.



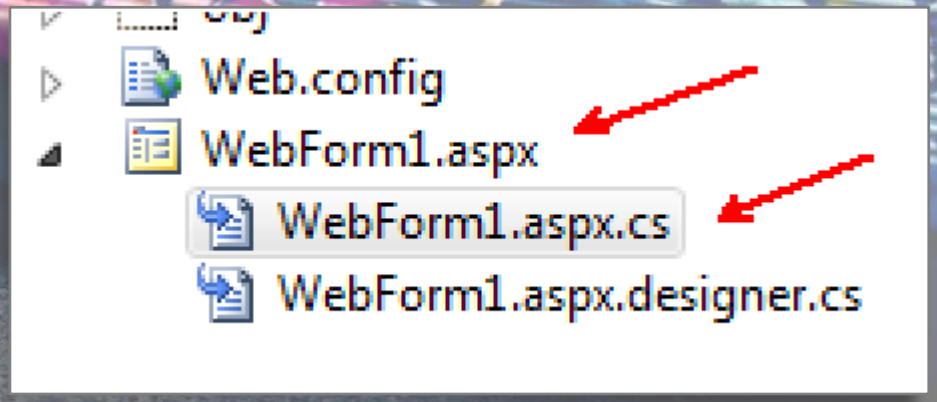
E como podemos criar uma estrutura que trate a ação primeiro em vez da *view*?

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.



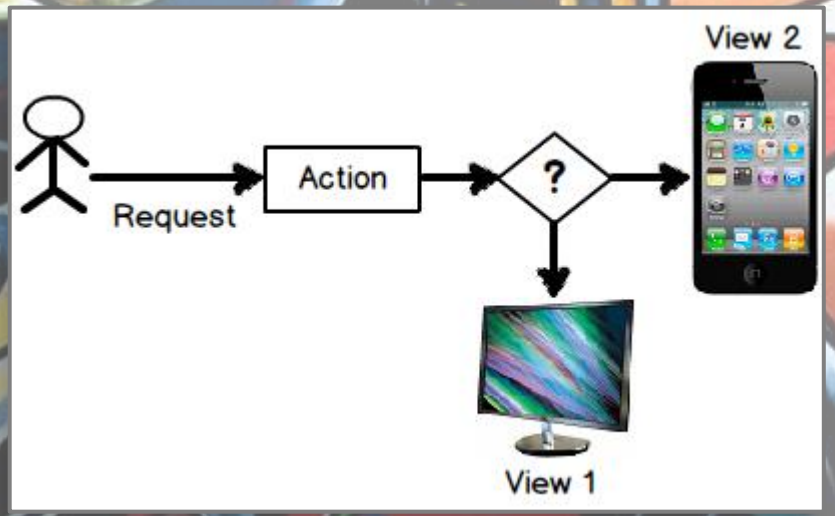
Problema 2 - Efeitos colaterais de uma má arquitetura: Alto acoplamento

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").

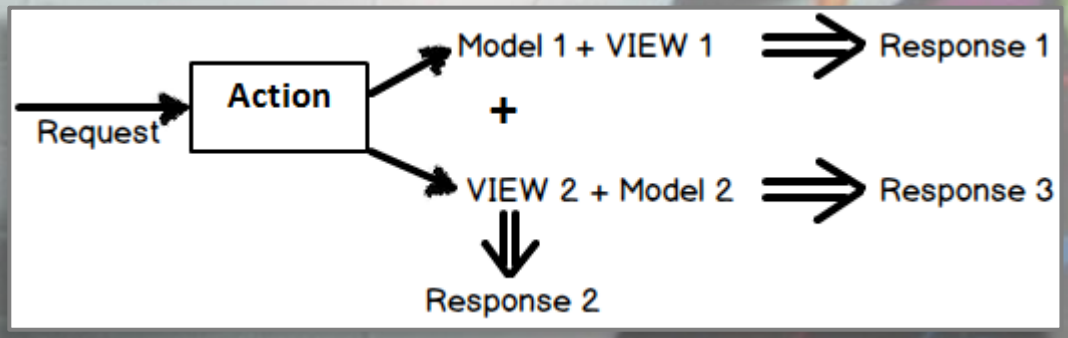
Em MVC, uma estrutura de ação pode ser criada para responder com diferentes tipos de conteúdo de maneira simples e eficiente.

```
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.

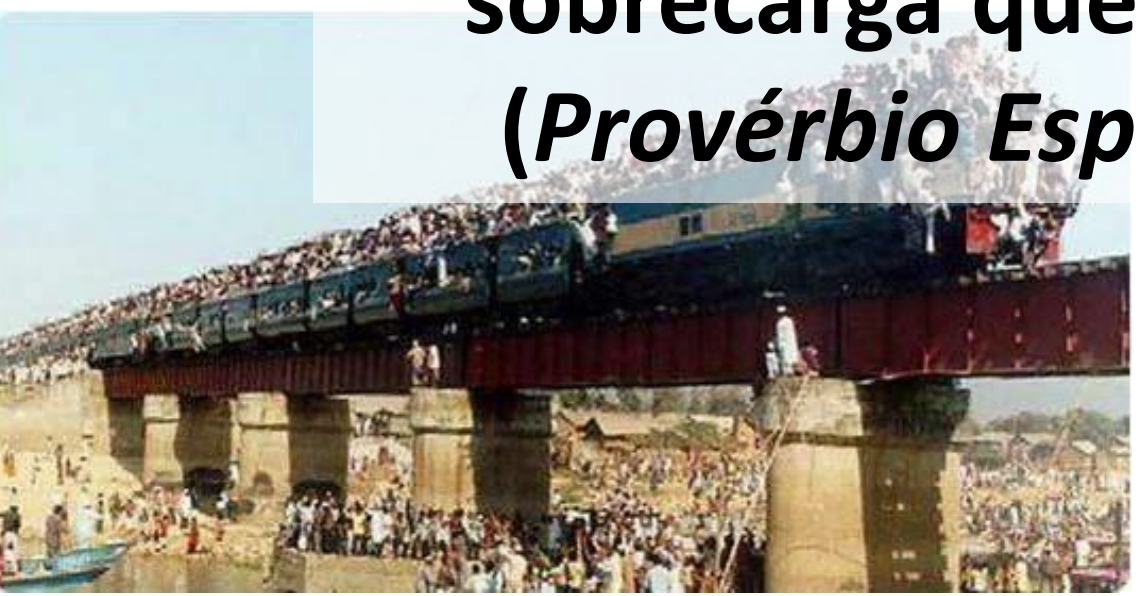


Entendendo o "S" de SRP (Single Responsability Principle)

```
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*)



DETRAN-PE IVIA

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

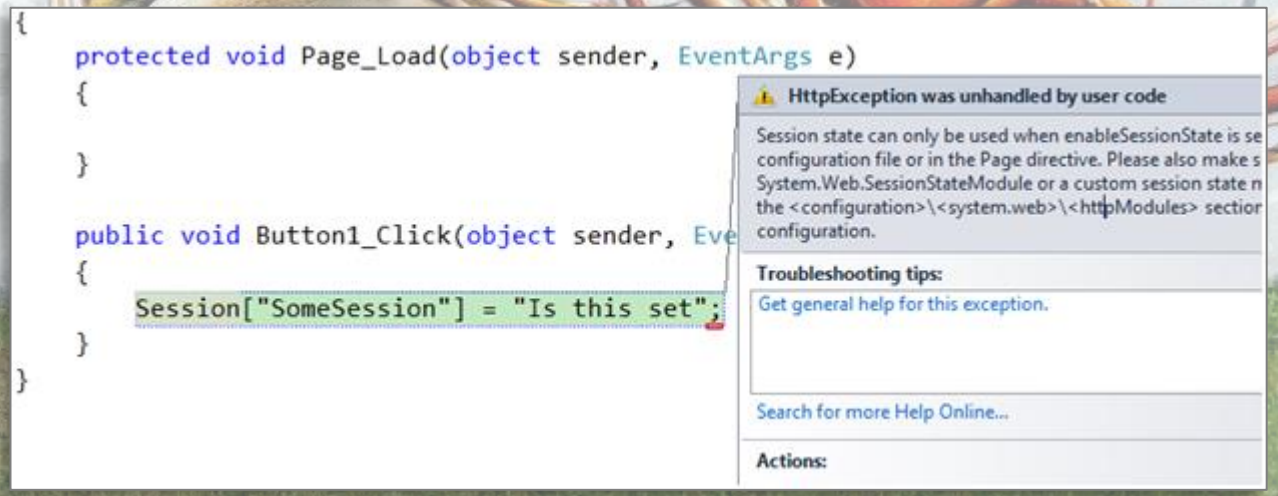
```
public ActionResult Index(string ViewName, Cliente dadosCliente)
{
    if (ViewName == "Detalhe")
    {
        return View("DetalhesCliente", dadosCliente);
    }
    else
    {
        return View("Cliente", dadosCliente);
    }
}
```

DETRAN-PE IVIA

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*.



DETRAN-PE IVIA

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

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

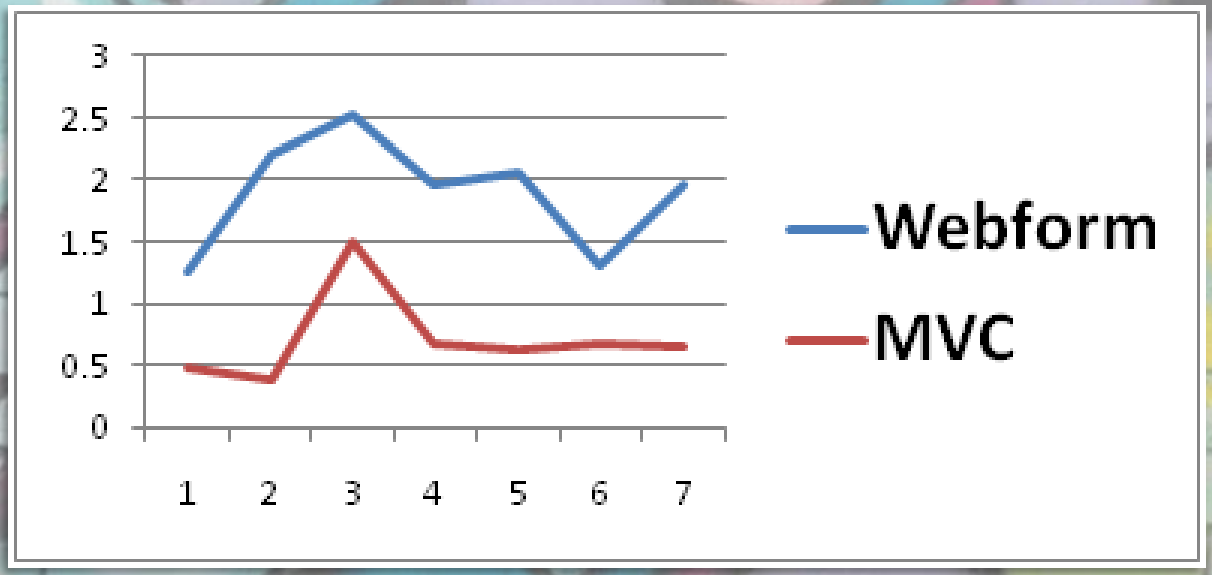
- No MVC não há ***Server Controls***
- No MVC não há ciclo de vida de página
- No MVC não há *view state*

Mas só isso mesmo :P

DETRAN-PE IVIA

E o que se ganha?

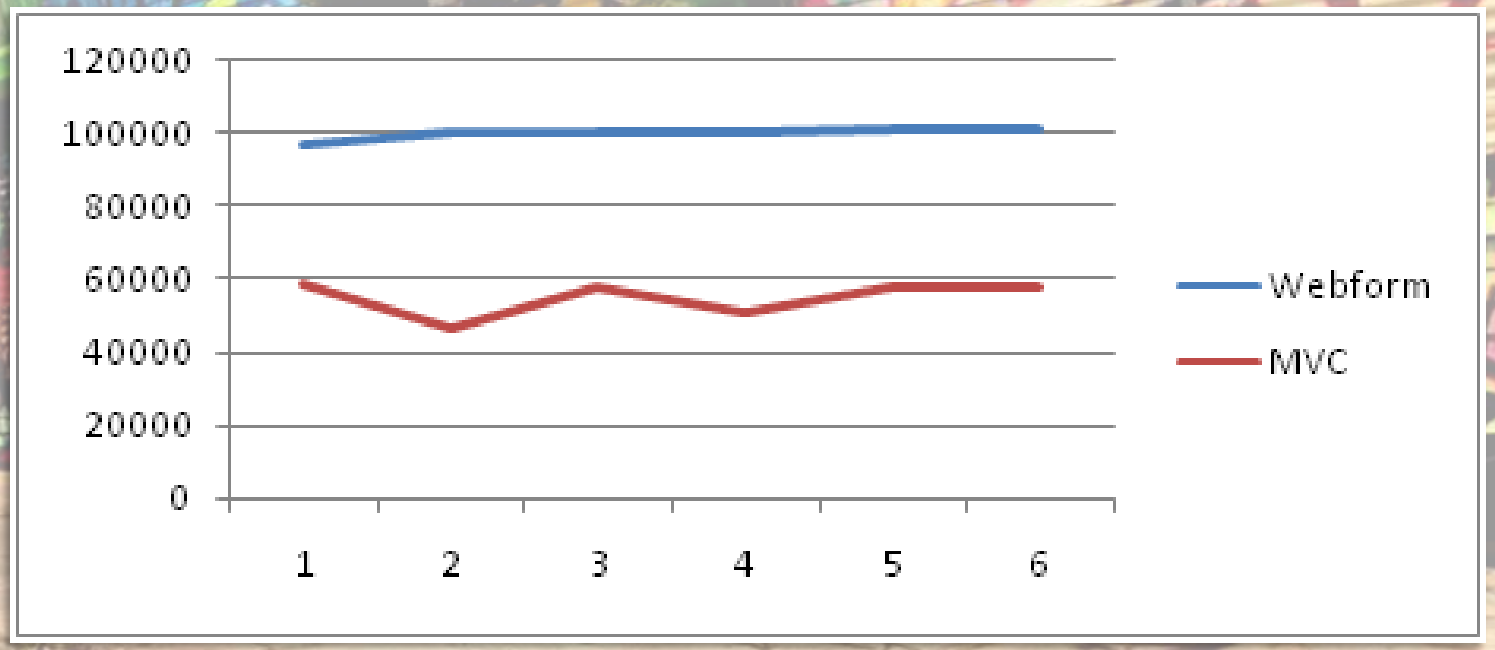
Velocidade!



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.



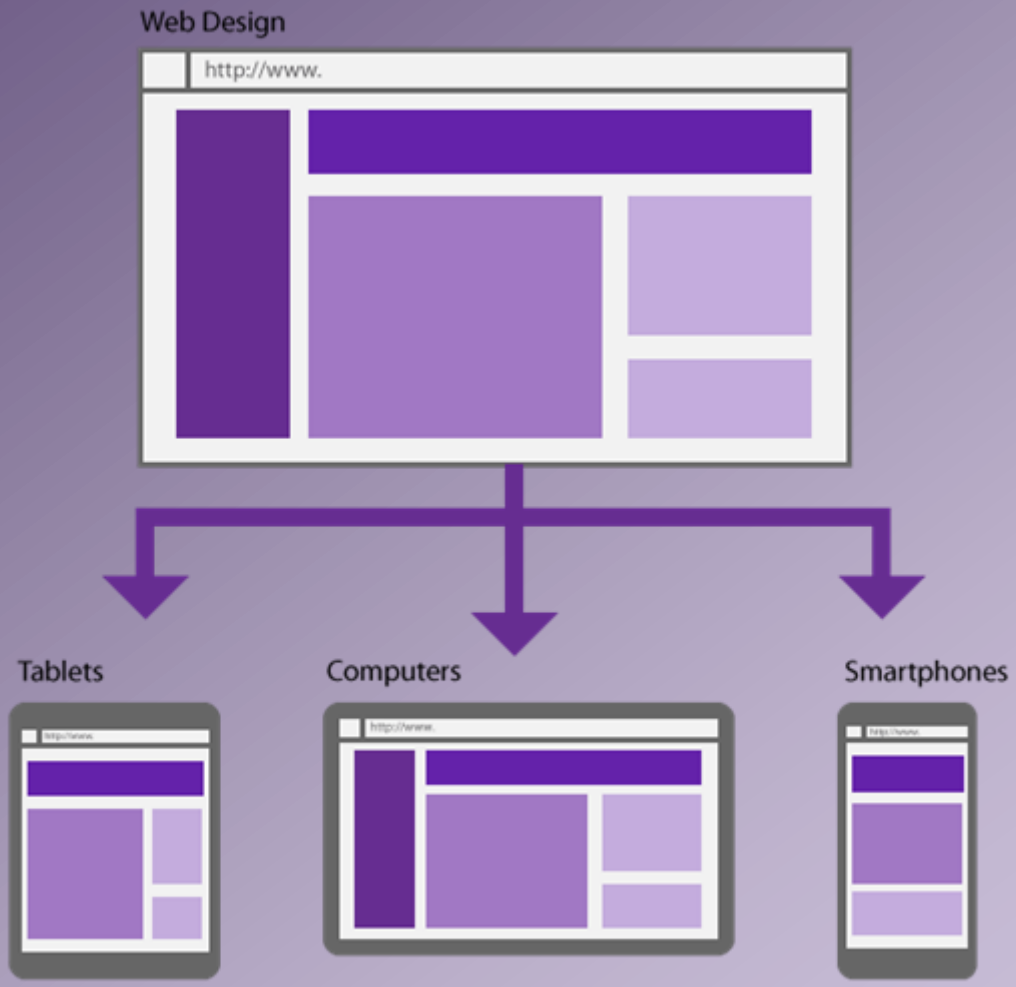
Comparação do tamanho do conteúdo de resposta entra as duas arquiteturas

DETRAN-PE IVIA

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.

E como é o ASP.NET MVC na prática?

ASP.NET Web API



DETRAN-PE IVIA

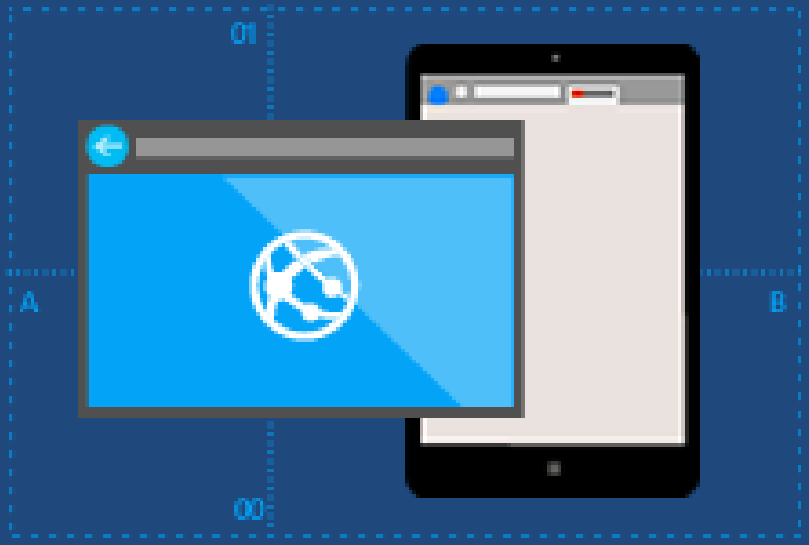
O que é uma API?

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.

DETRAN-PE IVIA

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.

DETRAN-PE

IVIA®

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.

DETRAN-PE IVIA

REST comparado com SOAP

Requisição SOAP:

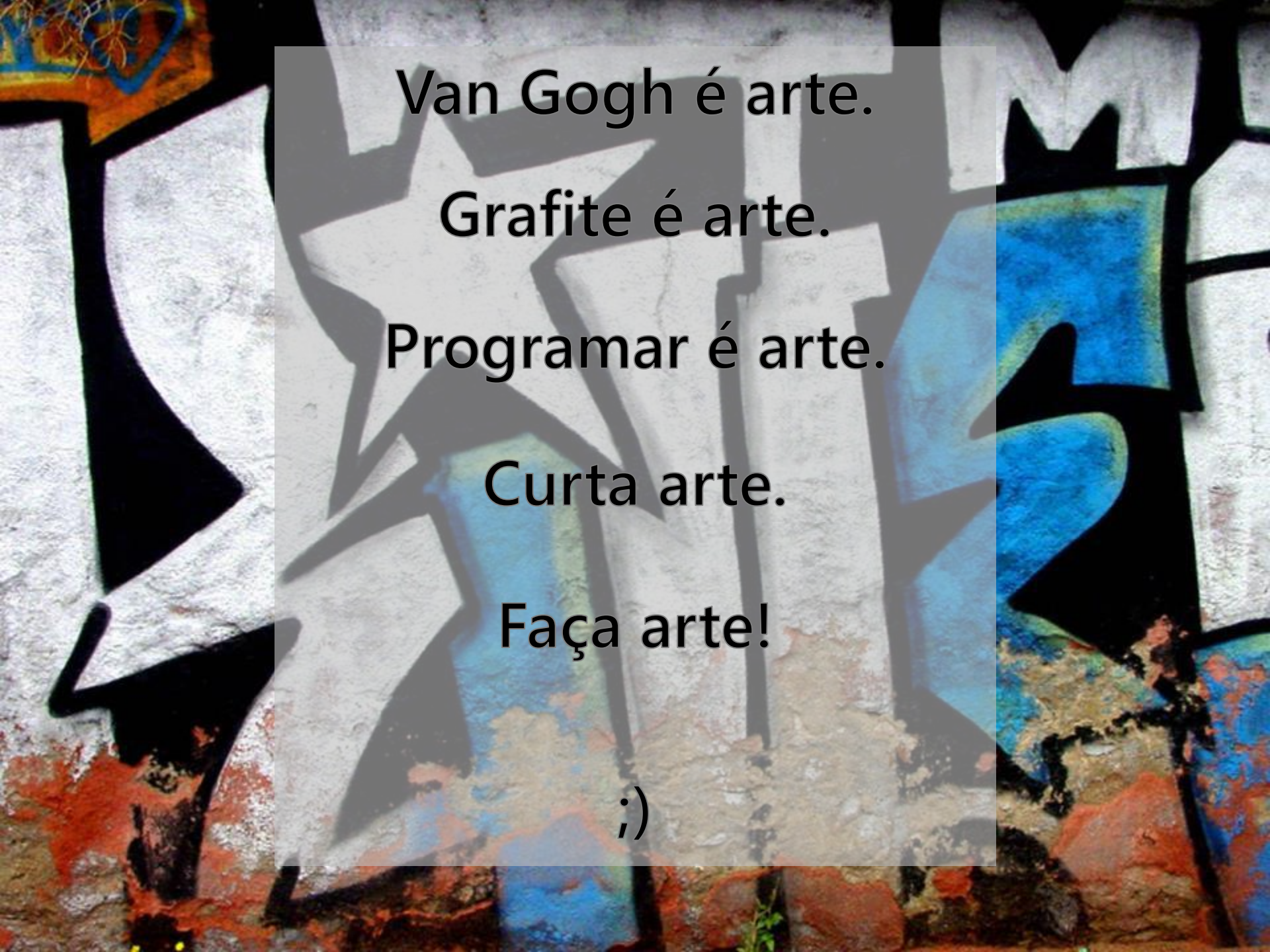
```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-
encoding">
 <soap:body pb="http://www.acme.com/phonebook">
  <pb:GetUserDetails>
   <pb:UserID>12345</pb:UserID>
  </pb:GetUserDetails>
 </soap:Body>
</soap:Envelope>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

Requisição REST

```
http://www.acme.com/phonebook/UserDetails/12345
```

DETRAN-PE IVIA

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


Van Gogh é arte.
Grafite é arte.
Programar é arte.
Curta arte.
Faça arte!
;)