

FIA P GRADUAÇÃO

ENTERPRISE APPLICATION DEVELOPMENT

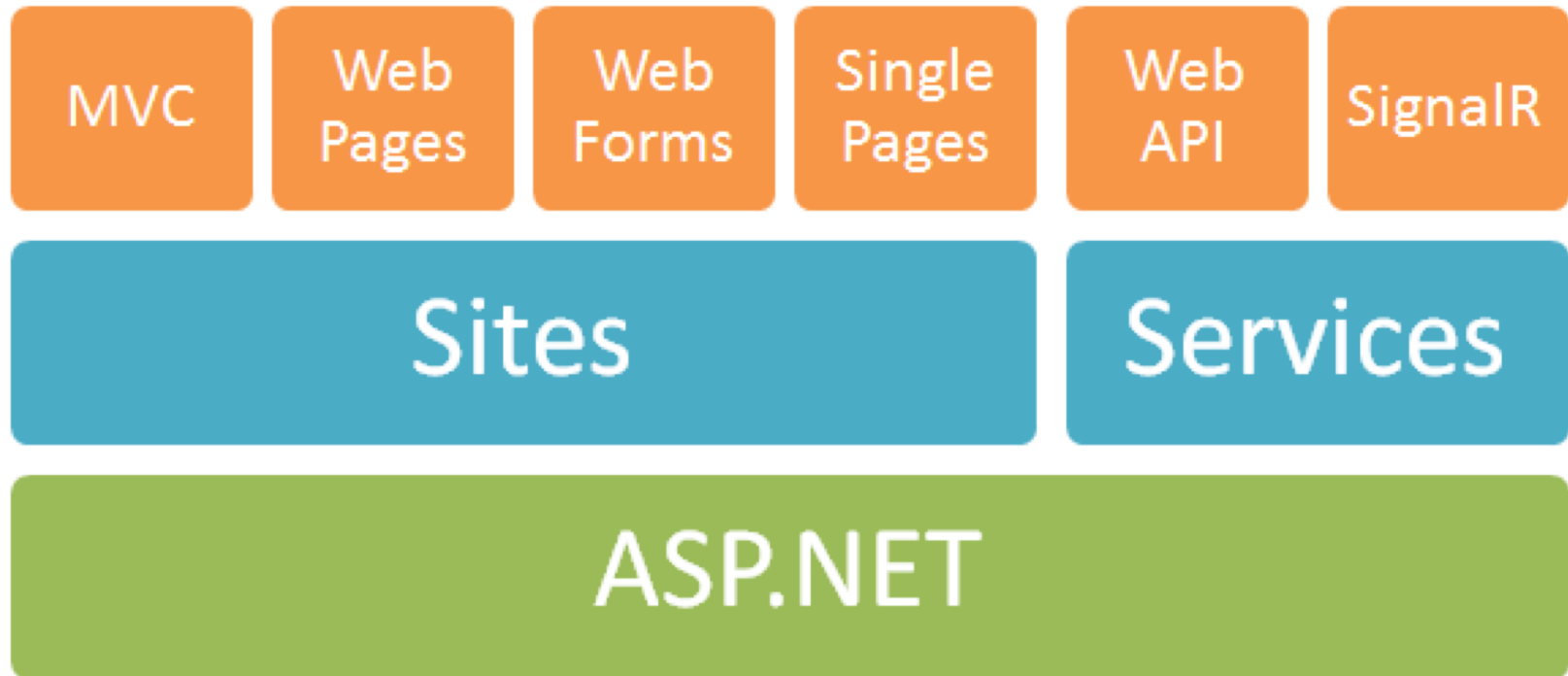
Prof. Me. Thiago T. I. Yamamoto

#03 – ASP.NET MVC - ROTAS E CONTROLER

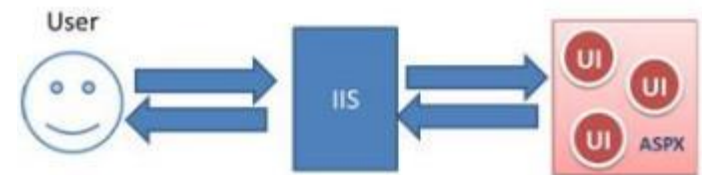


#03 – ROTAS E CONTROLER

- ASP.NET Web Forms
- O que é ASP.NET MVC?
- ASP.NET MVC Introdução
- Prática! Criando um projeto
- Estrutura do Projeto
- Controllers e Rotas
- ActionResult
- Parâmetros
- ViewBag e Strongly Typed Views



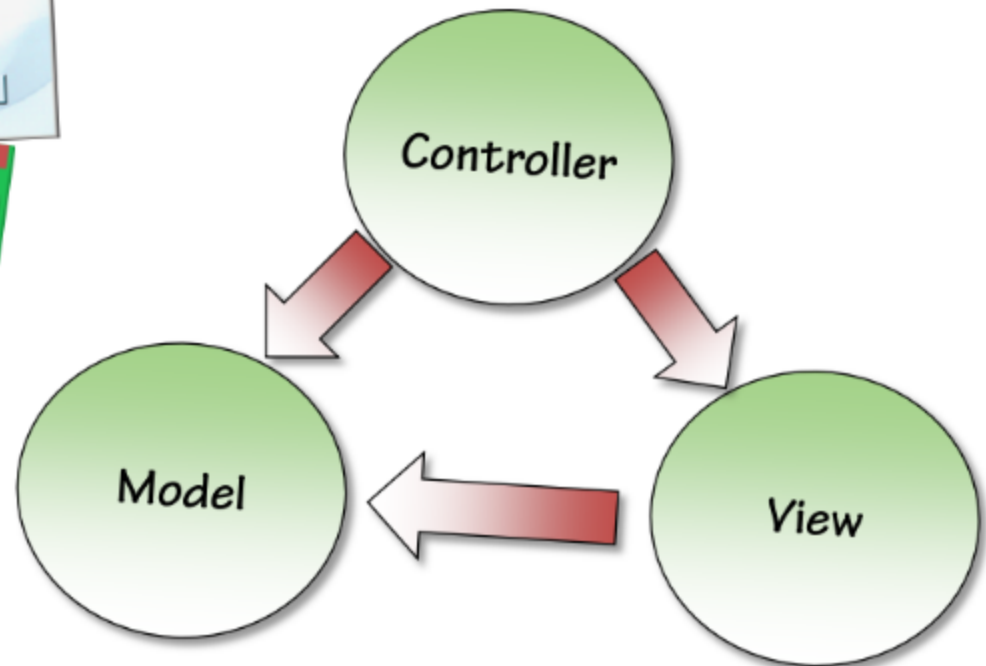
- Primeiro Framework de Desenvolvimento Web com ASP.NET;
- Existe desde 2001/2002;
- Rapidamente tomou parte do mercado, pela facilidade de desenvolvimento e similaridade com desenvolvimento desktop;
- Vantagens:
 - Rápido desenvolvimento;
 - Controles ricos;
 - Fácil migração do desenvolvimento Windows Forms para Web Forms;
- Desvantagens:
 - Difícil controle sobre o HTML gerado;
 - Difícil integração com Frameworks Javascript;
 - Dificuldade em realizar testes;



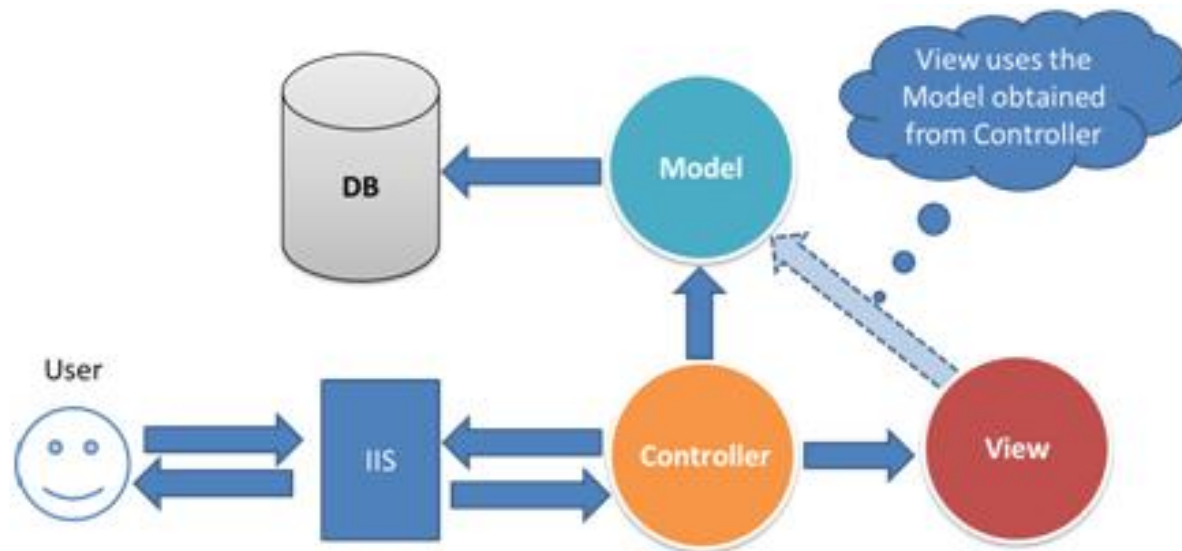
Fonte: <http://www.codeproject.com/Articles/528117/WebForms-vs-MVC>



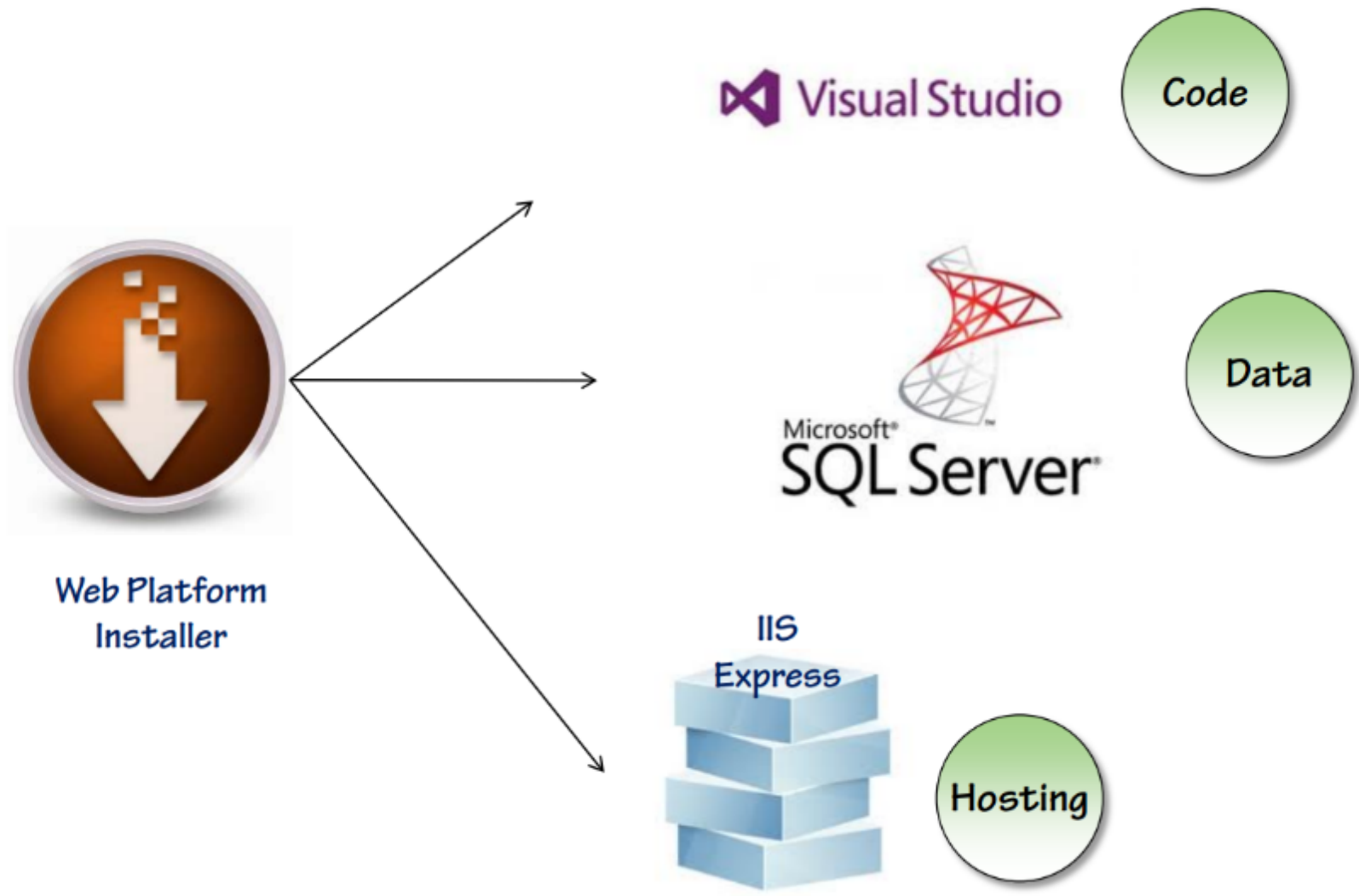
 Visual Studio



- Baseado nos padrões MVC:
 - Model: lógica da aplicação e modelo de dados;
 - View: exibe as informações para o usuário (Interface);
 - Controller: recebe as requisições do usuário e encaminha o model e/ou view;



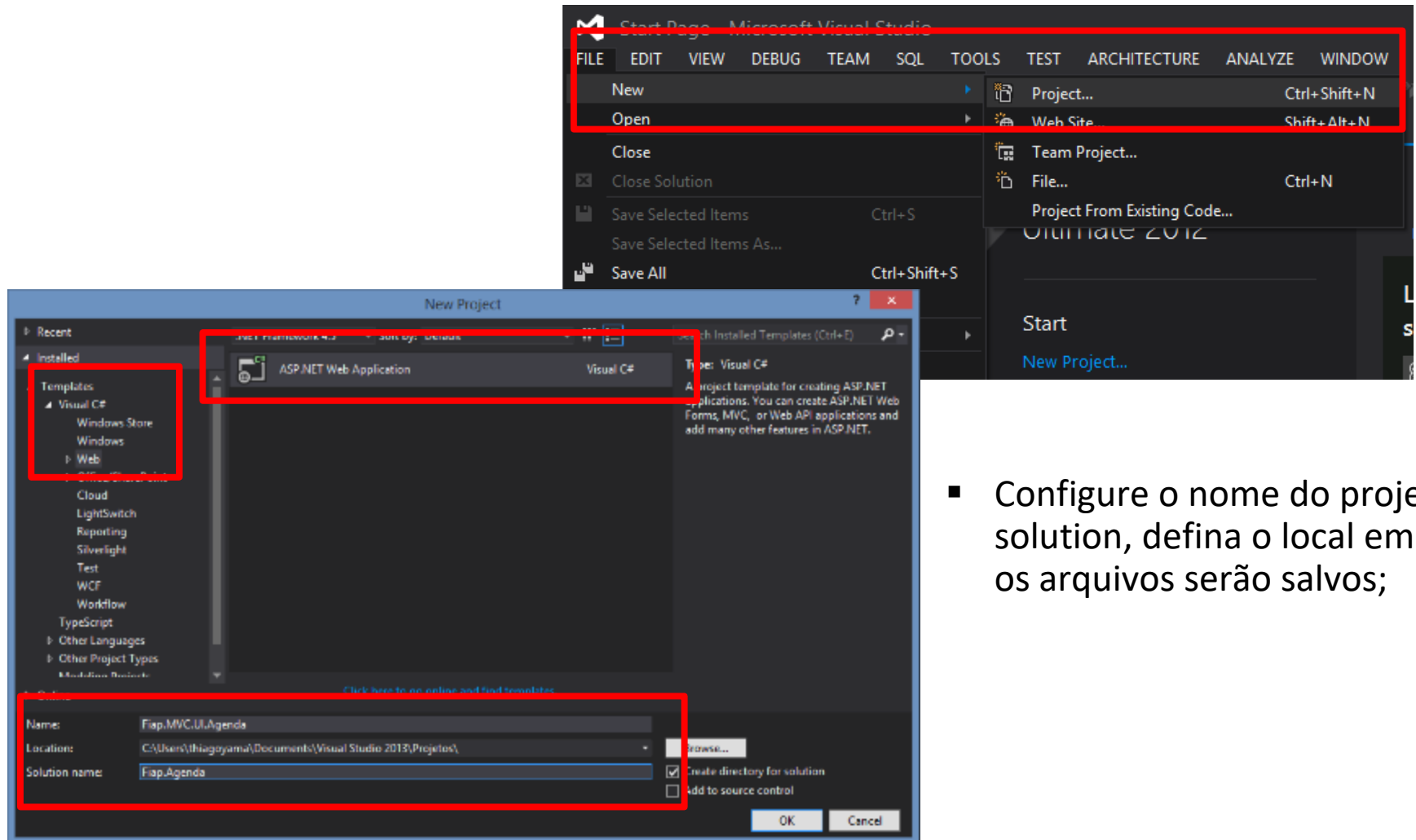
Fonte: <http://www.codeproject.com/Articles/528117/WebForms-vs-MVC>



PRIMEIRO PROJETO ASP.NET MVC

PRIMEIRO PROJETO ASP.NET MVC

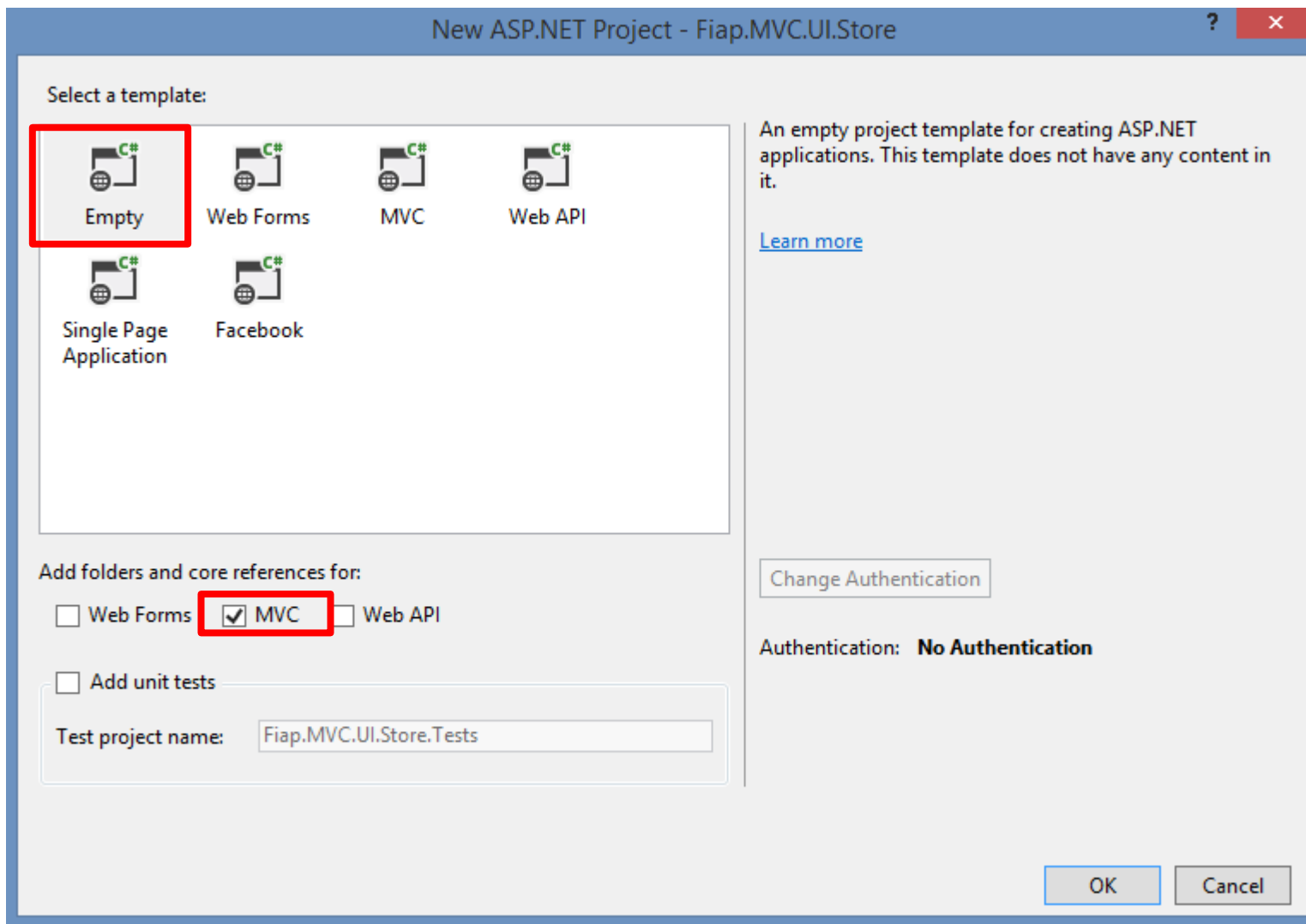
- File -> New -> Project;
- Visual C# -> Web -> ASP.NET Web Application

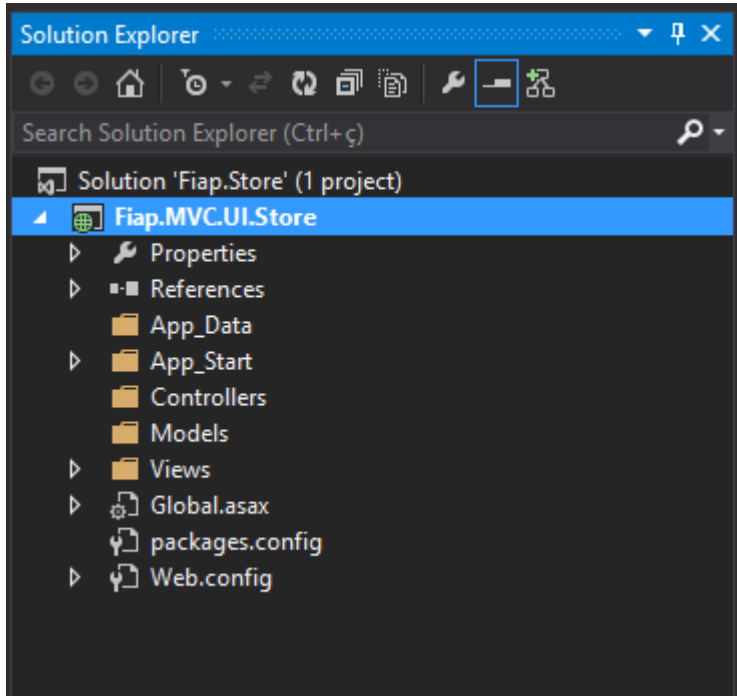


- Configure o nome do projeto e solution, defina o local em que os arquivos serão salvos;

PRIMEIRO PROJETO ASP.NET MVC

- Selecione a opção Empty e marque para adicionar as pastas do MVC:





- **Controllers:** classes de controller, é necessário que os arquivos terminem com sufixo Controller, ex: LoginController;
- **Models:** classes que representa o modelo da aplicação e classes para acesso ao banco de dados;
- **Views:** arquivos para view da aplicação;

ROTAS E CONTROLER

http://localhost:40392/home/index

- As urls são mapeadas Para métodos da classe controller;

- Controller:

- Responsabilidades básicas:

- » Recuperar dados enviados pelo usuário;
 - » Interagir com a camada model;
 - » Acionar a camada de apresentação para enviar a resposta ao usuário;

- Regras:

- » Nome da classe deve ter o sufixo “**Controller**”;
 - » Deve estender a classe **System.Web.Mvc.Controller**

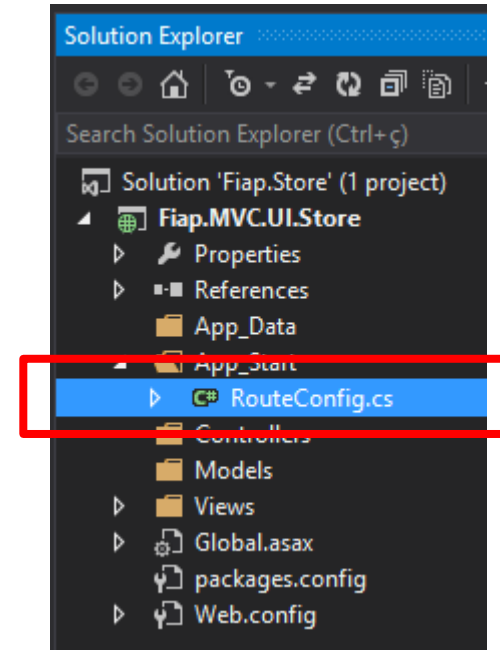
```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        //Código...
    }
}
```

ROTAS - CONFIGURAÇÃO

- Configuração de Rotas fica definido dentro da pasta App_Start -> RouteConfig.cs
- Podemos adicionar mais configurações de rotas;

MapRoute:

- name: nome da rota;
- url: definição da url;
- defaults: valores padrões caso uma parte da url não seja informada;



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

- O método (Action) é chamado, ele executa o processamento e depois deve devolver um ActionResult, um valor que indica o que deve ser executado depois da ação;

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        return View();
    }
}
```

Com esse retorno, o ASP.NET MVC irá determinar qual arquivo será utilizado para a resposta para o usuário:

1. Views\Nome do Controller\Acao.cshtml
2. Views\Shared\Acao.cshtml

RETORNO – ACTION RESULT

- Se o servidor não encontrar a view correspondente, um erro será exibido no browser:

Erro de Servidor no Aplicativo '/'. ---

The view 'Index' or its master was not found or no view engine supports the searched locations. The following locations were searched:

- ~/Views/Home/Index.aspx*
- ~/Views/Home/Index.ascx*
- ~/Views/Shared/Index.aspx*
- ~/Views/Shared/Index.ascx*
- ~/Views/Home/Index.cshtml*
- ~/Views/Home/Index.vbhtml*
- ~/Views/Shared/Index.cshtml*
- ~/Views/Shared/Index.vbhtml*

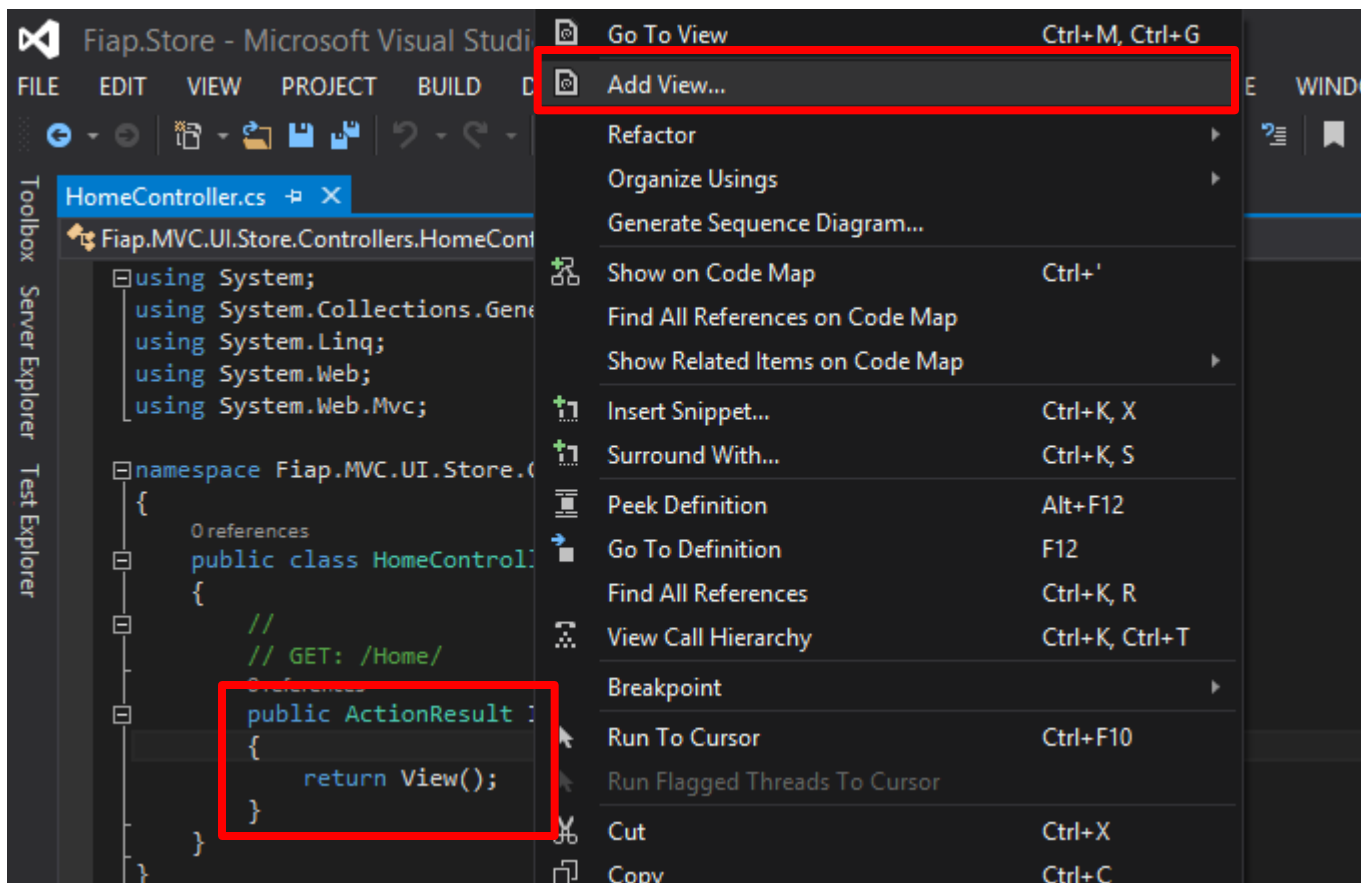
Descrição: Ocorreu uma exceção sem tratamento durante a execução da atual solicitação da Web. Examine o rastreamento de pilha para obter mais informações sobre o erro.

Detalhes da Exceção: System.InvalidOperationException: The view 'Index' or its master was not found or no view engine supports the searched locations. The following locations were searched:

- ~/Views/Home/Index.aspx*
- ~/Views/Home/Index.ascx*

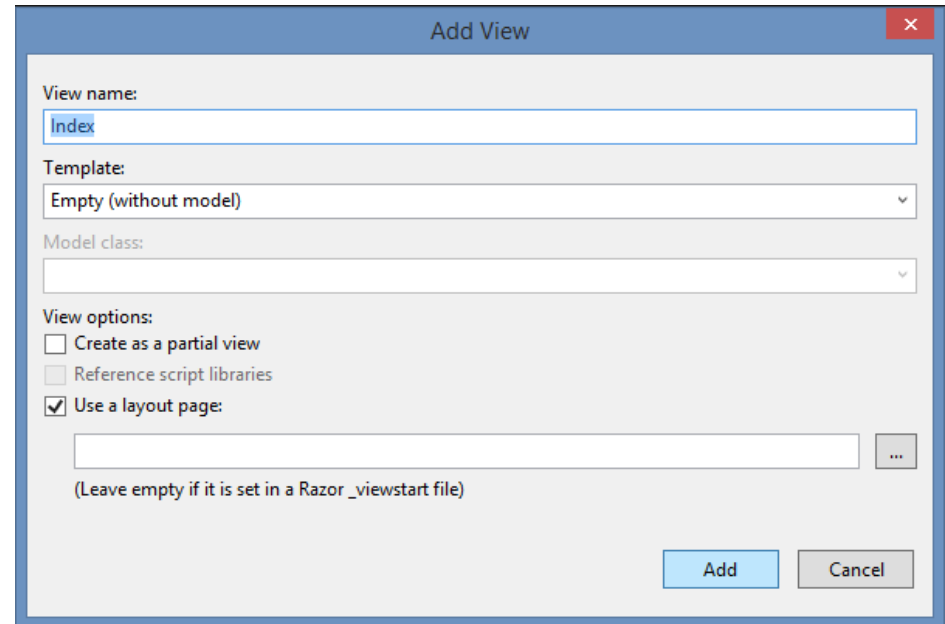
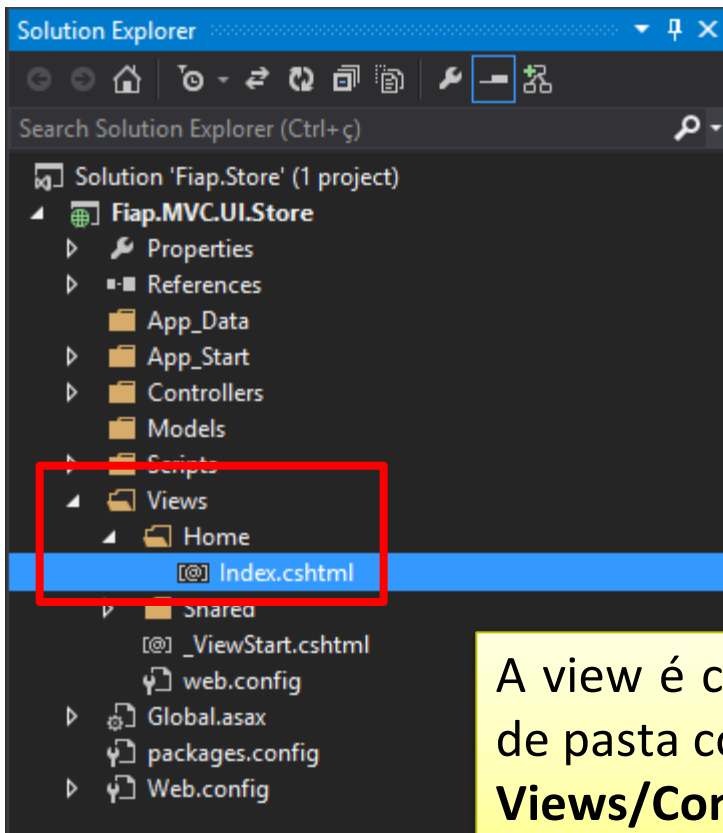
ADICIONANDO UMA VIEW

- Para adicionar a view de uma **Ação (Método)**, clique com o botão direito dentro do método e escolha **Add View...**



ADICIONANDO UMA VIEW

- O nome da **View** já vem preechida de acordo com o nome da action.
- Clique em **Add**



A view é criada com o nome e a estrutura de pasta correta:
Views/Controller/Action.cshtml

- Nem sempre precisamos devolver uma View (Página) como resposta para o usuário..
- É possível redirecionar o usuário para uma URL específica, direcioná-lo para uma outra **Action**, etc..

Action Result	Helper Method	Description
ViewResult	View	Renderiza a view para usuário
PartialViewResult	PartialView	Determina o arquivo que deve ser utilizado para construir a página parcial de resposta
RedirectResult	Redirect	Redireciona para uma URL específica;
RedirectToRouteResult	RedirectToAction RedirectToRoute	Redireciona para outra ação da camada de controle
ContentResult	Content	Devolve um tipo definido pelo desenvolvedor, por exemplo texto
JsonResult	Json	Devolve um objeto no formato JSON.
JavaScriptResult	JavaScript	Devolve código javascript
FileResult	File	Devolve valores binários para escrever no response
EmptyResult	(None)	Devolve uma resposta vazia

```
public ActionResult Index()
{
    return Content("Olá Mundo");
}
```

Retorna o Texto: **Olá Mundo**

```
public ActionResult Index()
{
    return View("Home");
}
```

Retorna para a página **Home**

```
public ActionResult Index()
{
    return RedirectToAction("Action", "Controller");
}
```

Retorna para outra **Action** em outro **Controller**

ACTION SELECTORS – HTTPGET E HTTPPOST

- Podemos definir qual tipo de requisição a Action do Controller irá processar:
 - GET
 - POST

Processa requisições **GET**

```
[HttpGet]
public ActionResult Index()
{
    return Action();
}
```

Processa requisições **POST**

```
[HttpPost]
public ActionResult Index()
{
    return Action();
}
```

RECEBENDO PARÂMETROS

- No controller, podemos receber os parâmetros enviados pela view (request);

Formulário HTML

```
<form action="/Home/Cadastrar">
  <label>Nome
    <input type="text" name="nome" />
  </label>
  <label>Sobrenome
    <input type="text" name="sobrenome" />
  </label>
  <input type="submit" value="Enviar" />
</form>
```

- Uma forma é recuperar as informações através do objeto Request utilizando o name dos input do formulário:

```
public ActionResult Cadastrar()
{
    string nome = Request["Nome"];
    string sobrenome = Request["Sobrenome"];
    return View();
}
```

- Outra forma é receber as informações como parâmetro da action. Assim, basta definir o mesmo nome para os campos do formulário e parâmetros da action:

```
public ActionResult Cadastrar(string nome, string sobrenome)
{
    return View();
}
```

- Podemos ainda utilizar uma entidade como parâmetro, assim o ASP.NET MVC irá criar o objeto com os valores preenchidos no formulário:

```
public class Cliente
{
    public string Nome { get; set; }
    public string Sobrenome { get; set; }
}
```

- As propriedades da entidade devem possuir o mesmo nome dos parâmetros HTTP.

```
public ActionResult Cadastrar(Cliente cliente)
{
    string nome = cliente.Nome;
    string sobrenome = cliente.Sobrenome;
    return View();
}
```

ENVIANDO VALORES PARA A TELA

1. ViewBag
2. TempData
3. Strongly Types Views

- Podemos adicionar dinamicamente valores à ViewBag:

Camada Controller

```
public ActionResult Index()
{
    ViewBag.TituloFormulario = "Cadastro Cliente";
    return View();
}
```

Camada View

```
<h3>@ViewBag.tituloformulario</h3>
```


- Para informações temporárias, podemos utilizar o TempData:

Camada Controller

```
public ActionResult Cadastrar()  
{  
    TempData["mensagem"] = "Cadastrado com Sucesso!";  
    return View();  
}
```

Camada View

```
<h3>@TempData["mensagem"]</h3>
```

- Podemos enviar um objeto para a camada view, passando o objeto como parâmetro do retorno da action:

Camada Controller

```
public ActionResult Cadastrar(Cliente cliente)
{
    return View(cliente);
}
```

Camada View

```
@model FIAP.Models.Cliente

<p>
    @Model.Nome, @Model.Sobrenome
</p>
```

Definimos qual tipo de objeto será recebido

Acessamos as propriedades do objeto

EXERCÍCIO

- Prática!
- Crie um formulário para receber informações de um usuário:
 - Nome
 - Idade
 - Email
- Depois de enviar as informações, o usuário deve receber uma mensagem de sucesso e as informações do usuário enviadas.

Copyright © 2013 – 2018 - Prof. Me. Thiago T. I. Yamamoto

Todos direitos reservados. Reprodução ou divulgação total ou parcial deste documento é expressamente proibido sem o consentimento formal, por escrito, do Professor (autor).

“Descobri que quanto mais eu estudo, mais sorte eu pareço ter nas provas”