FIAP

FÁBRICA DE DESENVOLVIMENTO

#17 – Web Api - Server

FIAP

FÁBRICA DE DESENVOLVIMENTO – 2018

Prof. Thiago T. I. Yamamoto

#17 – WEB API - SERVER

thiagoyama thiagoyama@gmail.com

CONTEÚDO PROGRAMÁTICO

#17 – WEB API - SERVER

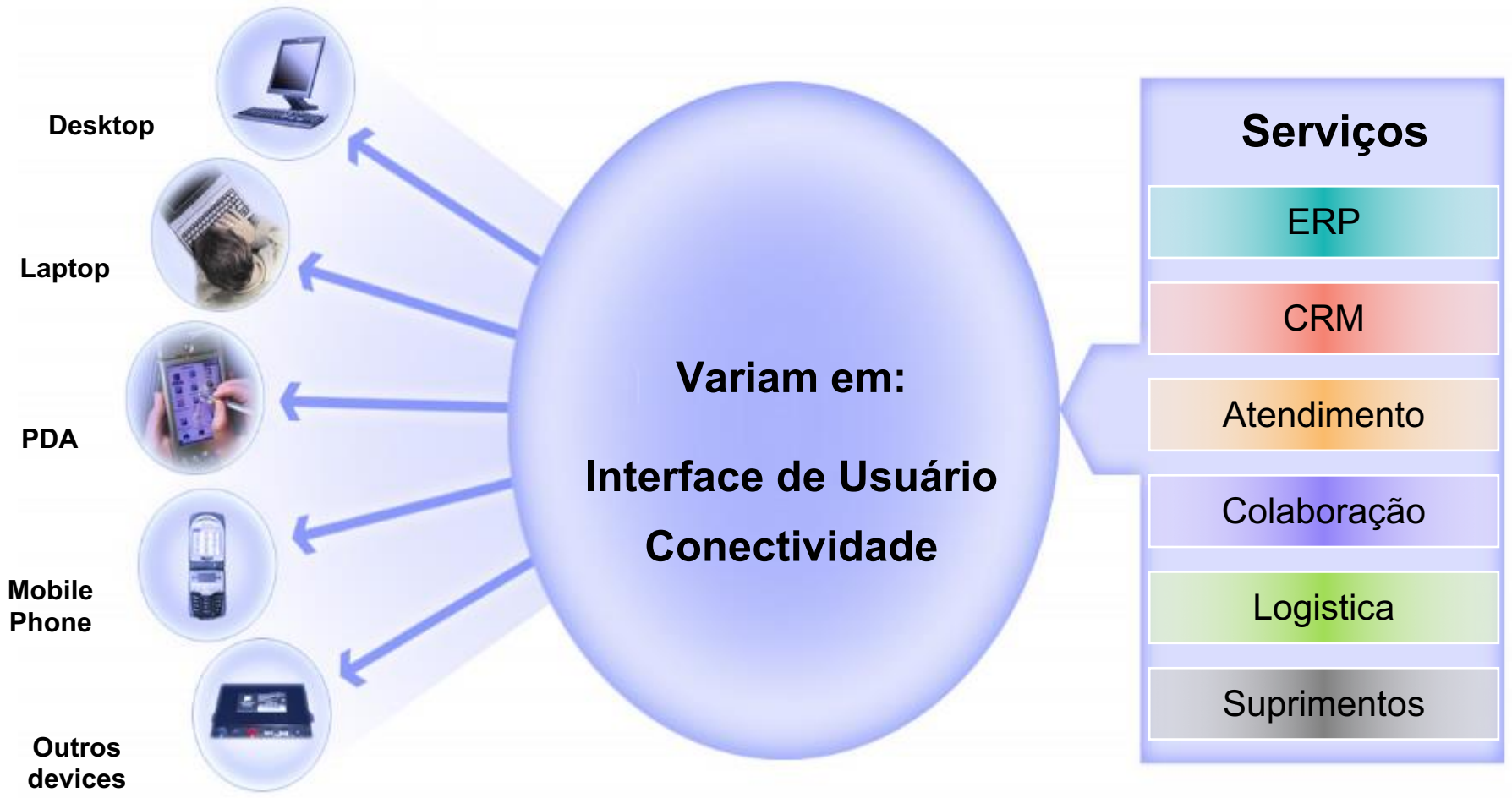
- Integração
- REST
- Web API
- DTO

FIAP

INTEGRAÇÃO E WEB SERVICE RESTFUL

WEB COMO MEIO DE COMUNICAÇÃO

ARQUITETURA MULTICANAL



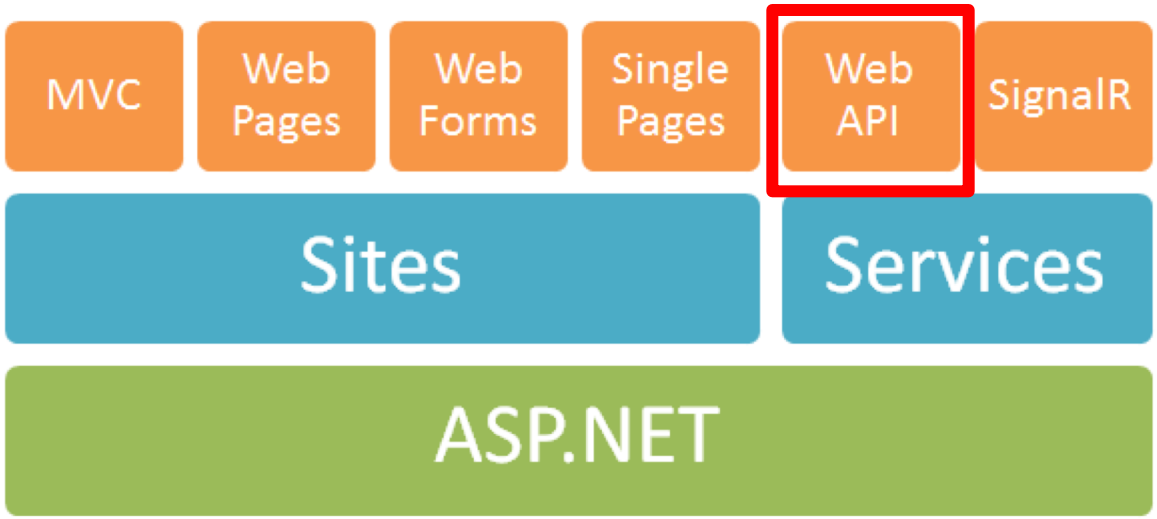
MODELO ARQUITETURAL REST

Atualmente, uma nova abordagem de construção de webservices vem sendo utilizada:

Web Services RESTFul (REpresentational State Transfer)

- Simples, leve, fácil de desenvolver e evoluir;
- Tudo é um recurso (Resource);
- Cada recurso possui um identificador (URI);
- Recursos podem ser de vários formados: html, xml, json;
- Protocolo HTTP;
- Os métodos HTTP: GET, POST, PUT, DELETE são utilizados na arquitetura REST.

ASP.NET WEB API



- Lançado em 2012;
- Framework que simplifica a construção de serviços REST;
- Pode ser utilizado em conjunto com MVC, Web Forms, etc.. ou em um projeto isolado;

MÉTODOS HTTP

FIAP

- O protocolo HTTP define oito métodos que indicam a ação a ser realizada: GET, POST, PUT, DELETE, TRACE, HEAD, OPTIONS, CONNECT.

- Os 4 métodos principais que serão utilizados nos serviços REST:
  - **GET**: recupera um recurso;
  - **POST**: cria um novo recurso;
  - **PUT**: atualiza um recurso existente;
  - **DELETE**: remove um recurso;

RESPOSTAS HTTP

- Existem 5 **tipos** de código de resposta do HTTP:
  - **1xx:** Informação – o pedido foi recebido e está sendo processada;
  - **2xx:** Sucesso – indica que a requisição foi bem sucedida;
  - **3xx:** Redirecionamento – indica a ação que deve ser tomada para completar a requisição;
  - **4xx:** Erro no Cliente – indica que foi realizada uma requisição que não pode ser atendida;
  - **5xx:** Erro no servidor – ocorreu um erro no servidor;

FORMATOS

- Recursos podem estar em vários formados: html, xml, json..
- Vamos trabalhar com **Json – JavaScript Object Notation.**
  - Formato simples e leve para transferência de dados.
  - Uma alternativa para o XML

Exemplo:

```
{
   "show": "Oasis",
   "preco": 150,
   "local": "São Paulo"
}
```

Validador de formato Json: http://jsonlint.com/

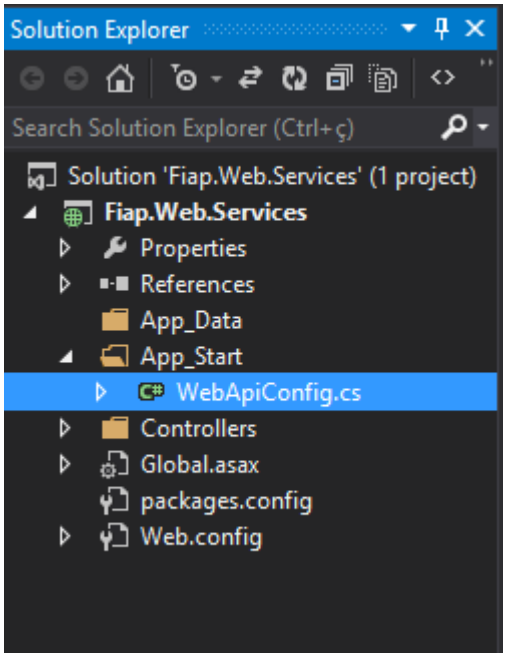
FIAP

IMPLEMENTAÇÃO

CONTROLLER E CONFIGURAÇÕES

```
public class ClienteController : ApiController
{
}
```

- Controller deve estender de **ApiController;**
- Configuração de Rotas fica definido dentro da pasta App_Start -> WebApiConfig.cs;
- Podemos adicionar mais configurações de Rotas;




```
config.Routes.MapHttpRoute(
    name: "DefaultApi",
    routeTemplate: "api/{controller}/{id}",
    defaults: new { id = RouteParameter.Optional }
);
```

- Configuração no Global.asax:

```
GlobalConfiguration.Configure(WebApiConfig.Register);
```

MÉTODOS

- Pode começar com o nome do método (Get, Put, Delete, Post) ou ser anotado com o método ([HttpGet], [HttpPost],[HttpPut],[HttpDelete]);
- O tipo de retorno depende do método:
  - Pode ser uma Entidade, uma coleção; void; IHttpActionResult...

- Alguns retornos possíveis para **IHttpActionResult**:
  - Ok
  - Created
  - NotFound
  - InternaErrorServer
  - Unauthorized
  - BadRequest
  - Conflict
  - Redirect

```
public IHttpActionResult Put(Cliente cliente)
{
      return Ok();
}
```

GET - BUSCA

Listar: *Uri: /api/cliente*

```
public IEnumerable<Cliente> Get()
{
    return _unit.ClienteRepository.List();
}
```

Buscar Por Id: *Uri: /api/cliente/2*

```
public Cliente Get(int id)
{
    return _unit.ClienteRepository.SearchById(id);
}
```

Buscar Por Nome: *Uri: /api/cliente?name=Thiago*

```
public IEnumerable<Cliente> Get(string name)
{
    return _unit.ClienteRepository.SearchByName(name);
}
```

POST - CADASTRO

Uri: /api/cliente

```
public IHttpActionResult Post(Cliente cliente)
{
      if (ModelState.IsValid)
      {
         _unit.ClienteRepository.Add(cliente);
         _unit.Save();
         var uri = Url.Link("DefaultApi", new { id = cliente.ClienteId });
         return Created<Cliente>(new Uri(uri), cliente);
      }
      else
      {
         return BadRequest(ModelState);
      }
}
```

PUT - ATUALIZAÇÃO

Uri: /api/cliente/3

```
public IHttpActionResult Put(int id, Cliente cliente)
{
      if (ModelState.IsValid)
      {
            cliente.ClienteId = id;
            _unit.ClienteRepository.Update(cliente);
            _unit.Save();
            return Ok(cliente);
      }
      else
      {
            return BadRequest(ModelState);
      }
}
```

DELETE - REMOÇÃO

Uri: /api/cliente/3

```
public void Delete(int id)
{
    _unit.ClienteRepository.Delete(id);
    _unit.Save();
}
```

DESIGN PATTERN - DTO

- Não é uma boa prática utilizar as Entidades de Banco de Dados para enviar informações no Web Service;
- Assim como os ViewModels, podemos criar classes específicas para a transmissão de informações entre aplicações;
- **Data Transfer Objects (DTOs)**
  - Remove as referencias circulares;
  - Desacopla a camada de serviços da camada de banco de dados;
  - Agrupa somente as propriedades relevantes ao serviço, omitindo informações desnecessárias;

Se estiver utilizando as Entidades com relacionamentos bidirecionais, é necessário uma configuração para ignorar referencias circulares (WebApiConfig.cs):

```
config.Formatters.JsonFormatter.SerializerSettings.ReferenceLoopHandling =
Newtonsoft.Json.ReferenceLoopHandling.Ignore;
```

FIAP