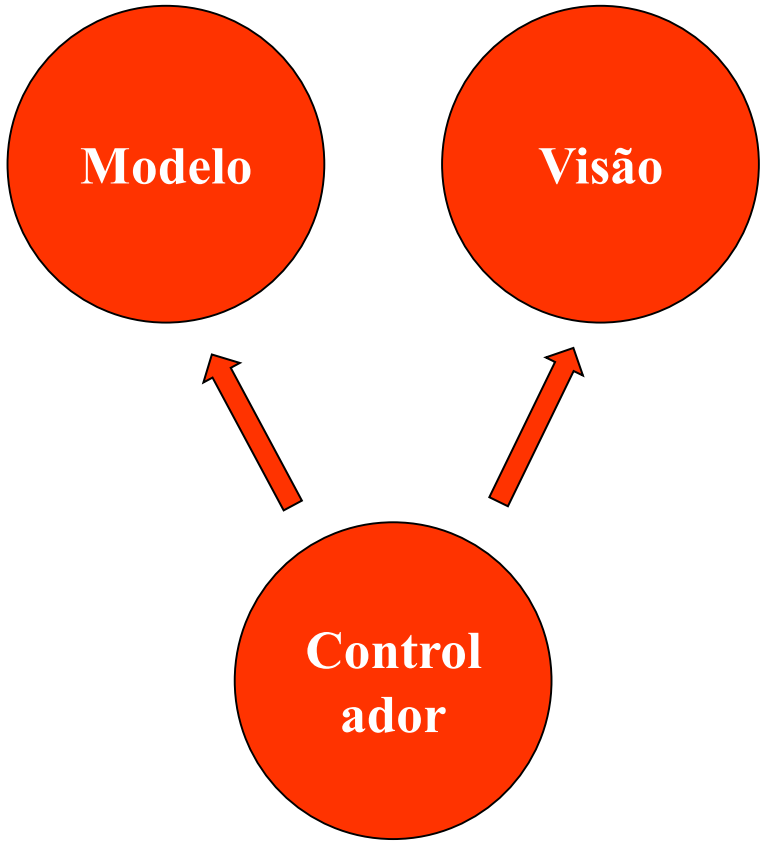
Desenvolvendo o Backend com a plataforma .NET

Tecnologia de Internet

Prof Vitor Almeida dos Santos

O Modelo MVC

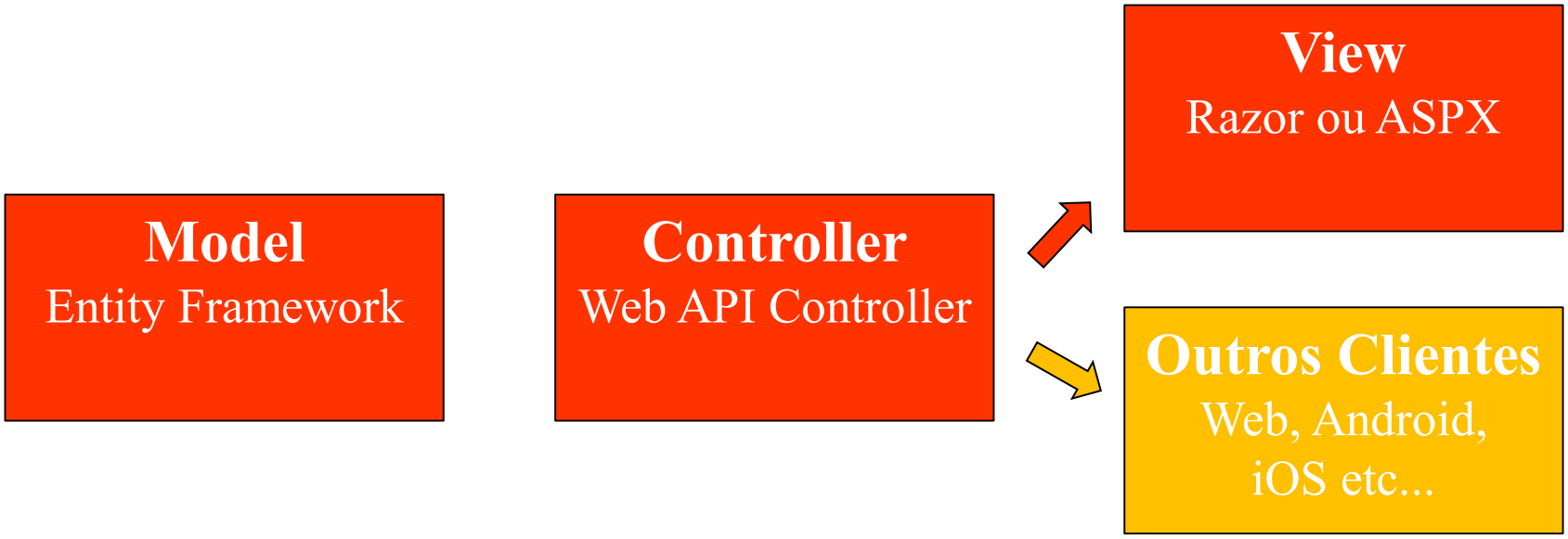
- Modelo
  - Representação do domínio de dados
  - Lógica de Negócios
  - Persistência
- Visão
  - Interface de usuário
  - Visualização do modelo
- Controlador
  - Intermediário
  - Recebe requisições, vincula dados à visão.



Desenvolvendo um Serviço Web REST com WEB API

O que é Web API?

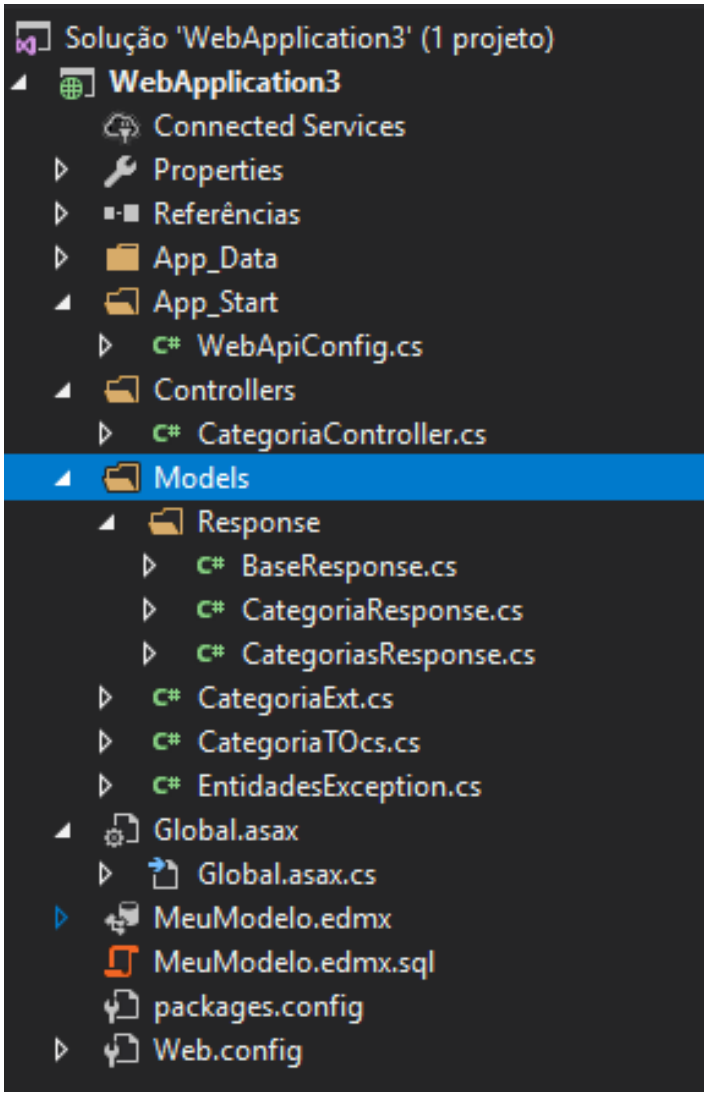
- **Framework Microsoft .NET.**
- **Frequentemente usado dentro de um modelo MVC.**



Desenvolvendo um back-end (modelo e controlador)

Criação do o projeto

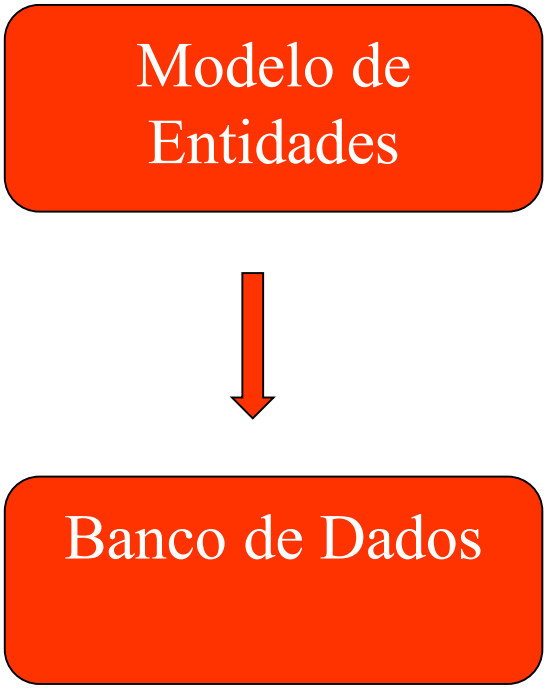
- Projeto ASP.NET vazio com suporte a Web API
- Verifique a existência das pastas “Models” e “Controllers”



Desenvolvendo um modelo com Entity Framework

Mapeamento Objeto-Relacional (ORM)

- Vantagens
  - Sua modelagem (diagrama de classes) se transforma em seu banco de dados;
  - Diversos recursos automatizados;
  - Não é necessário misturar SQL com seu código.
- Desvantagens
  - Tempo de aprendizado;
  - Configuração;
  - Controle de desempenho mais difícil: como estão sendo feitas as consultas SQL? Carregamento antecipando ou atrasado?

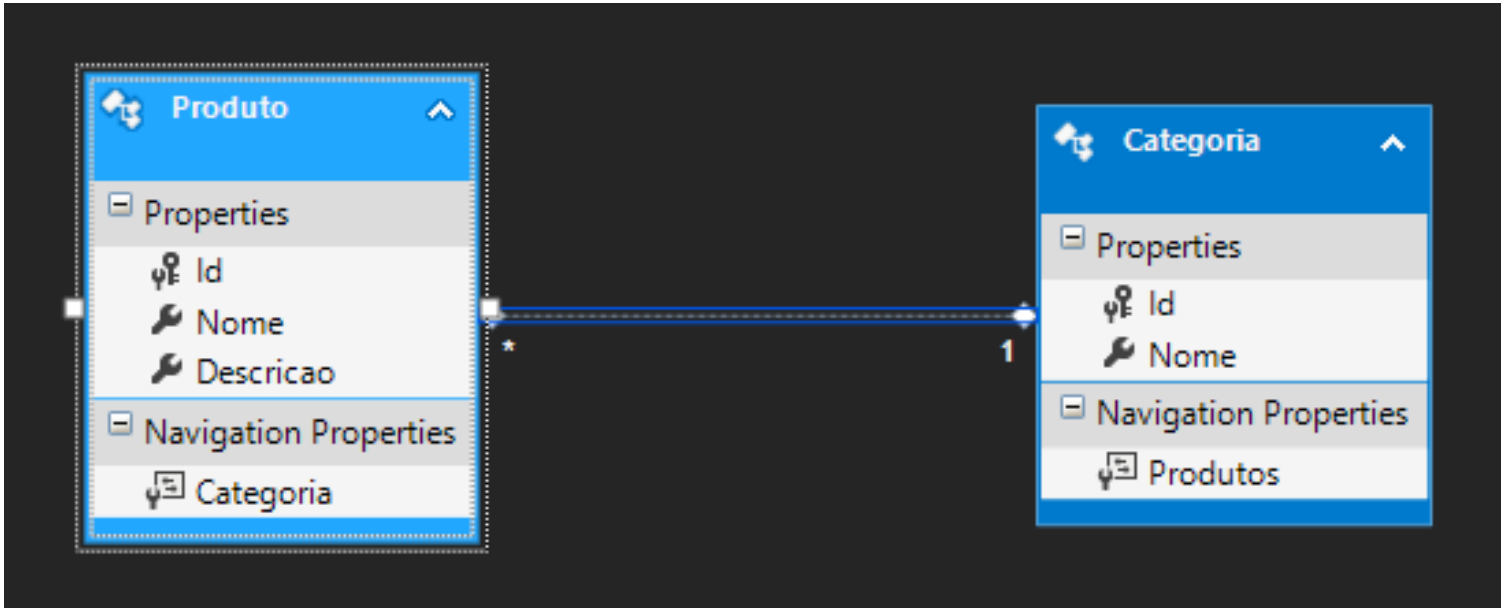


Mapeamento Objeto-Relacional

- Mapeamento 1 x 1
  - Uma classe == Uma tabela
  - Dentro de cada classe há uma instância da outra

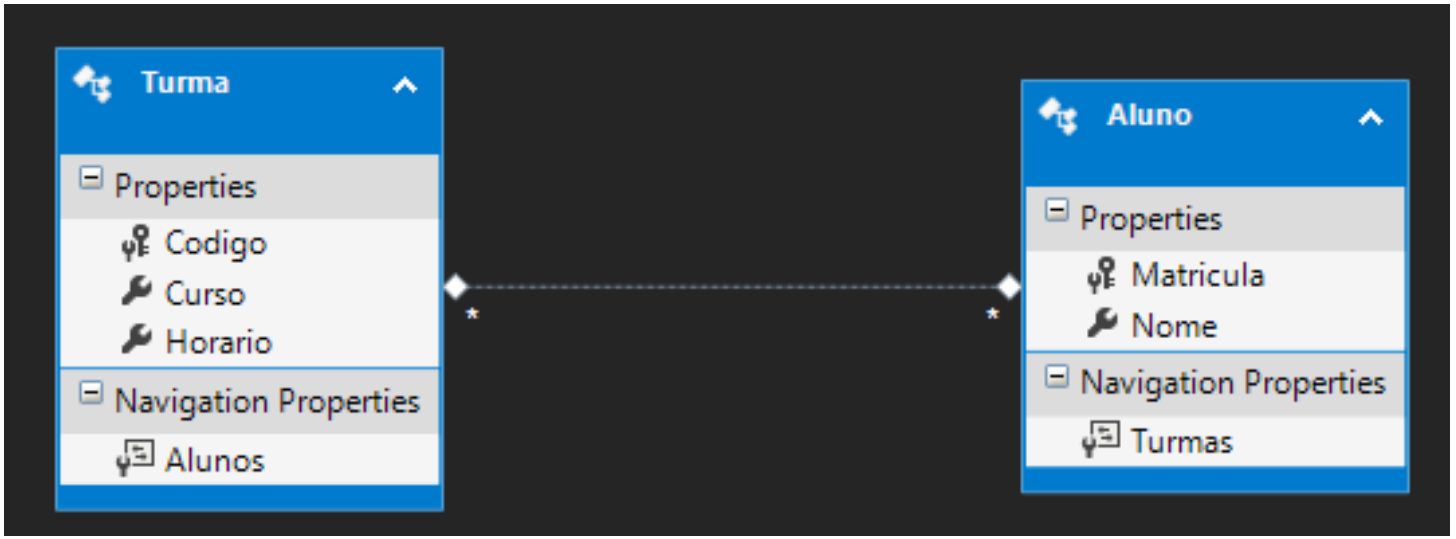
Mapeamento Objeto-Relacional

- Mapeamento 1 x n
  - Uma classe == Uma tabela
  - Dentro da classe “1” → Coleção da classe “n”
  - Dentro da classe “n” → Coleção da classe “1”



Mapeamento Objeto-Relacional

- Mapeamento n x n
  - Uma classe == Uma tabela
  - Tabela extra para guarda as relações;
  - Dentro de cada classe há uma coleção da outra

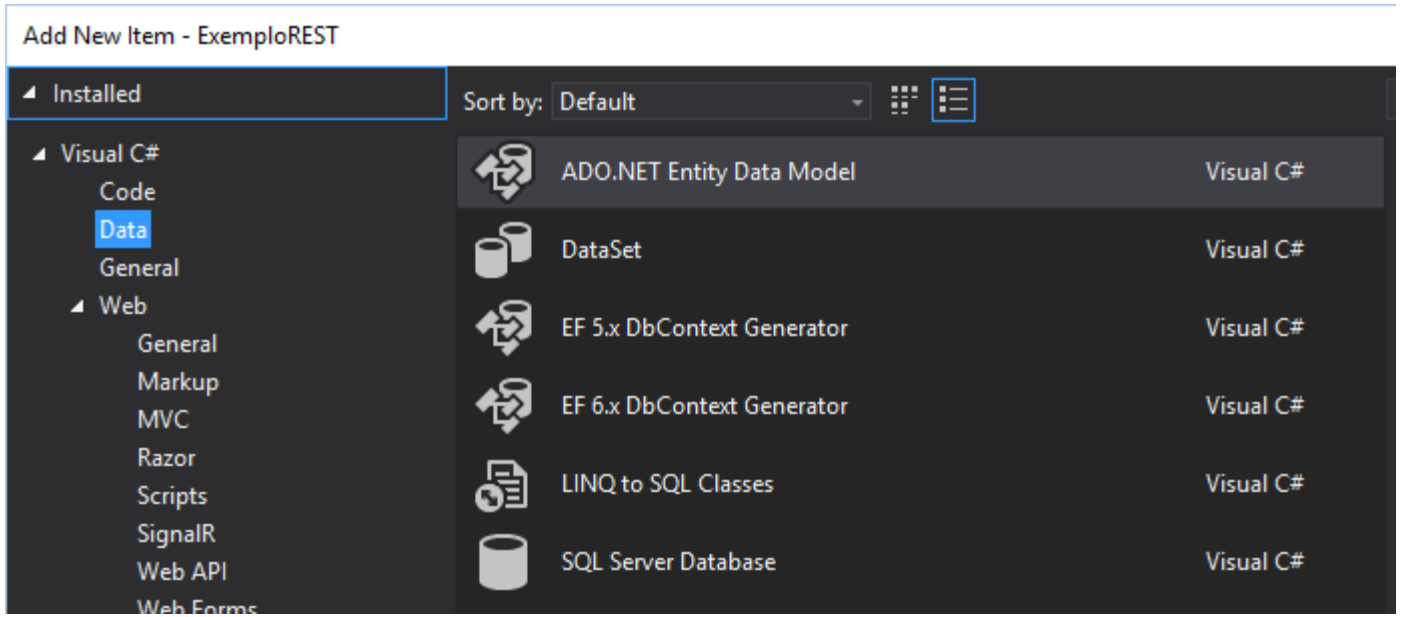


Passos para desenvolvimento do modelo

1. Criar um BD;
2. Criar modelo de entidades com Entity Data Model (EDM);
3. Configurar de um BD a partir do EDM;
4. Criar os métodos de manipulação de dados (inserir, consultar, atualizar e remover)

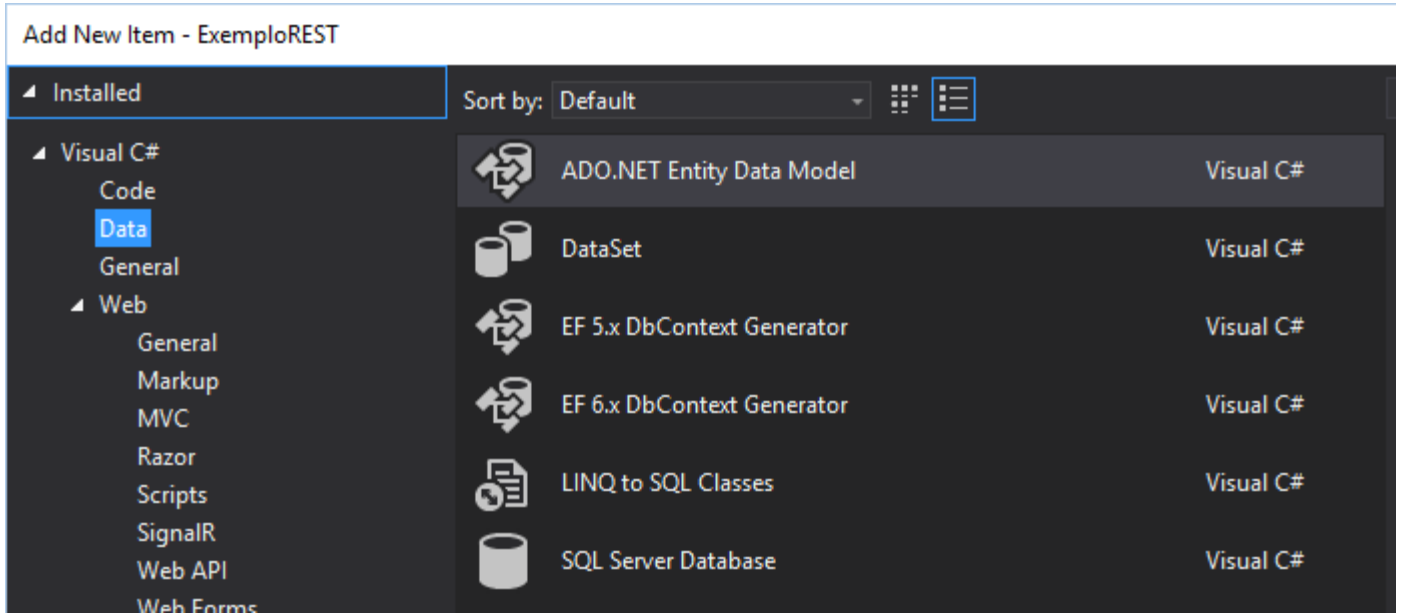
Passo 1: Construindo o BD

- Botão direito sobre o projeto → Add new item → SQL Server Database



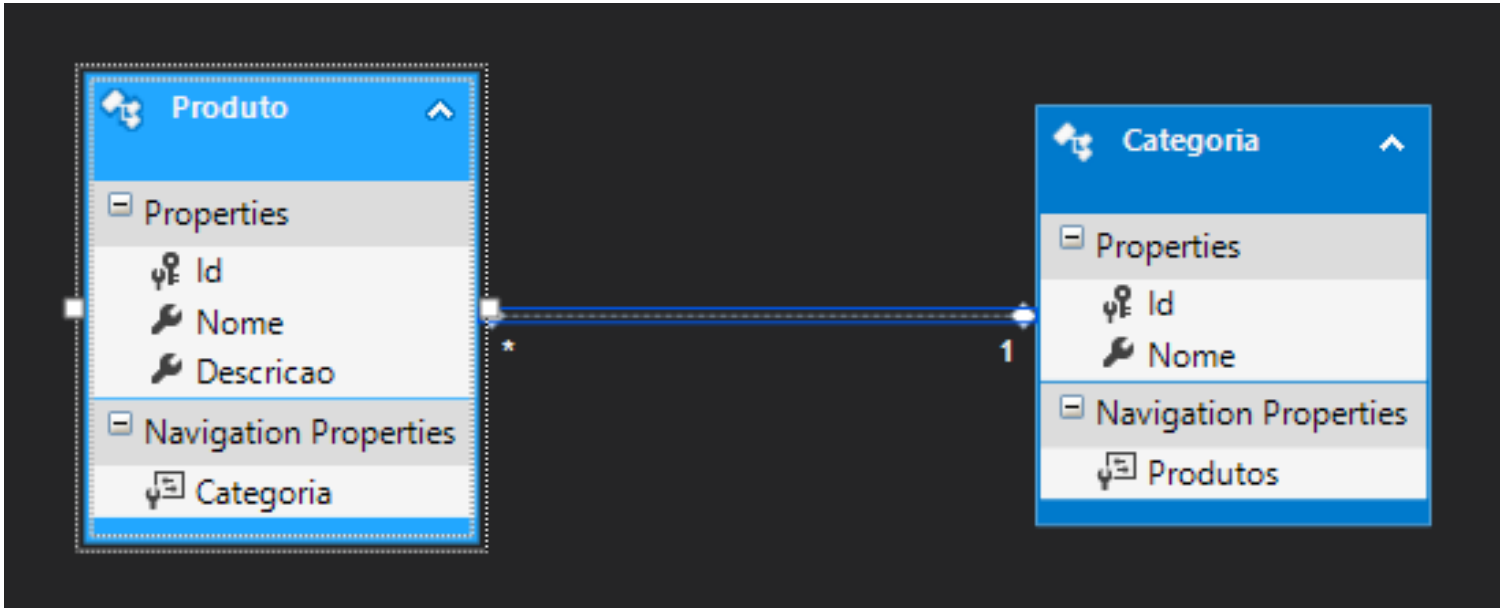
Passo 2: Entity Data Model

- Botão direito sobre o projeto → Add new item → ADO.NET Entity Data Model



Passo 2: Entity Data Model

- Defina suas entidades e relacionamentos.



Passo 3: Criando o BD a partir do modelo EDM

- Botão direito sobre o modelo EDM → Generate Database from Model;
- Salve o arquivo .sql;
- Duplo clique no BD para visualizá-lo na janela Server Explorer;
- Botão direito sobre o BD → New Query;
- Copie o código script do BD e clique em “Execute”.

Passo 4: Manipulando dados

- Defina classes com os métodos para manipulação de dados;
- As classes podem ter os mesmos nomes das entidades; utilize a palavra **partial** para isso.
- Exemplo:

```
public partial class Categoria
{
     public static List<Categoria> Listar(){...}
     public static Categoria Consultar(int id){...}
     public static int Inserir(string nome){...}
}
```

Passo 4: Manipulando dados

- Consultar.

```
public static Categoria Consultar(int id)
{
  Categoria categoria = null;
  using (MeBDEntities context = new MeBDEntities())
  {
    var categoria_ = from Categpria c in context.Categorias
                     where c.Id == id
                     select c;

    if (categoria_.Count() > 0)
    {
      categoria = categoria_.First();
    }
  }
  return categoria.;
}
```

Passo 4: Manipulando dados

- Listar.

```
public static List<Categoria> Listar()
{
  List<Categoria> categorias = new List<Categoria>();
  using (MeBDEntities context = new MeBDEntities())
  {
     categorias.AddRange(context.Categorias);
  }
  return categorias;
}
```

Passo 4: Manipulando dados

- Inserir.

```
public static void inserir(string nome)
{
  using (MeBDEntities context = new MeBDEntities())
  {
    Categoria c = new Categoria();
    c.Nome = nome;

    context.Categorias.Add(c);
    context.SaveChanges();
  }
}
```

Passo 4: Manipulando dados

- Atualizar.

```
public static void atualizar(int id, string nome)
{
  using (MeBDEntities context = new MeBDEntities())
  {
    var categoria_ = from Categpria c in context.Categorias
                     where c.Id == id
                     select c;

    if (categoria_.Count() > 0)
    {
      Categoria c = categoria_.First();
      c.Nome = nome;
      context.SaveChanges();
    }
  }
}
```

Passo 4: Manipulando dados

- Remover.

```
public static bool remover(int id)
{
  using (MeBDEntities context = new MeBDEntities())
  {
    var categoria_ = from Cateagoria c in context.Categorias
                     where c.Id == id
                     select c;

    if (categorias_.Count() > 0)
    {
      context.Categorias.Remove(categoria_.First());
      context.SaveChanges();
      return true;
    }
    else
      return false;
  }
}
```

Passo 4: Manipulando dados

- Defina exceções específicas para suas regras de negócios;
- Exemplo:

```
public static int Inserir(string nome)
{
    int retorno = 0;
    if (nome.Length == 0)
        throw new EntidadesException("Nome Vazio", 100);

    using (MeuBDEntities context = new MeuBDEntities())
    {
        var categoria_ = from Categoria cat in context.Categorias
                         where cat.Nome == nome
                         select cat;

        if (categoria_.Count() > 0)
            throw new EntidadesException("Categoria já existe", 101);

        Categoria c = new Categoria();
        c.Nome = nome;
        context.Categorias.Add(c);
        context.SaveChanges();
        retorno = c.Id;
    }
    return retorno;
}
```

Desenvolvendo um controlador com Web API

Passos para desenvolvimento do controlador

1. Criar os tipos para transferência de dados (tipos Transfer Objects);
2. Criar os tipos para retornos dos métodos dos controllers (tipos Response);
3. Desenvolver os métodos dos controllers;
4. Configurar controller (CORS e JSON)

Passo 1 – Tipos para transferência de dados

- Defina tipos segundo o padrão “Transfer Object”; são objetos para transferência de dados entre camadas;

- Exemplo:

```
public class CategoriaTO
{
    public int Id { get; set; }
    public string Nome { get; set; }
}
```

Passo 2 – Retorno (response) dos métodos do controller

- Defina tipos “Response” específicos para retorno dos métodos do controller;
- Cada Response possui os valores a serem retornados pela requisição, o status da requisição (sucesso ou erro) e mensagens de status correspondets.
- Defina uma classe BaseResponse para ser herdada pelos demais tipos Response;

```
public class BaseResponse
{
    public int Status { get; set; }
    public string Detalhes { get; set; }
}
```

Passo 2 – Retorno (response) dos métodos do controller

- Defina tipos que estendam “BaseResponse”, específicos para cada controller;

```
public class CategoriasResponse : BaseResponse
{
    public List<CategoriaTO> Categorias { get; set; }
}
```

Passo 3 – Implemente os métodos do controller

- Faça as chamadas aos métodos do modelo e retorne os tipos Response apropriados:

```
public IHttpActionResult Get()
{
    CategoriasResponse cResponse = new CategoriasResponse();
    try
    {
        cResponse.Categorias = Categoria.Consultar();
    }
    catch (EntidadesException eex)
    {
        cResponse.Status = eex.Codigo;
        cResponse.Detalhes = eex.Message;
    }
    catch (Exception ex)
    {
        cResponse.Status = -1;
        cResponse.Detalhes = ex.Message;
    }
    return Ok(cResponse);
}
```

Passo 4 – Configuração do controller

- Configurações Gerais
  - Em WebApiConfig acrescente:

```
config.Formatters.JsonFormatter.SupportedMediaTypes.Add(new MediaTypeHeaderValue("text/html"));
```

Passo 4 – Configuração do controller

- Baixe o seguinte pacote via NuGet:
  Microsoft.AspNet.WebApi.Cors

- Inclua o seguinte comando no método Register do seu WebAPIConfig:

```
config.EnableCors();
```

- Inclua o seguinte atributo sobre seu controller:

```
[EnableCors(origins: "*",  headers: "*",
methods: "*")]
```