MVC - Model View Controller

**Model** – Application data and behaviour in terms of its problem domain, and independente of the UI

Projeto:

Domain Model:

Movie

Customer

Rental

Transaction

Estas classes possuem propriedades/atributos e métodos que representam puramente o estado do aplicativo e as regras, não estão vinculados a interface.

O que significa que podemos pegar essas classes e usa-las em diferentes aplicativos

São POCOS (Plain Old CLR Objects)

**View** – HTML mark-up que será mostrado ao usuário

**Controller** – Responsavel pelo tratamento dos requests HTTP

Por exemplo

Mandamos uma solicitação htt://vidly.com/movies – um controlador será selecionado para tratar esse request.

Ele vai obter todos os filmes do banco de dados e coloca-los numa view e colocá-lo para o cliente ou navegador.

**ROUTER (Extra) –** Seleciona o controlador ideal para tratar o request

Vidly.com/movies -> Router -> MoviesController

**Uma ação de um controller é responsável por lidar com um request**

**Route Config:**

url: {controller}/{action}/{id}

selecionamos o controlador, a ação e o id em que a ação irá atuar

/movies/popular

Movies -> MoviesController

Popular -> Popular()

/movies/edit/1

Edit/1 -> Edit(int 1)

/movies -> MoviesController.Index()

**Controllers:**

AccountControllers: SignUp – Login -Logout

HomeControllers: Homepage

ManageController: Ações para user profile, change password, 2 factor auth

**WebConfig:**

Se atentar para connectionStrings => Informações do banco de dados

E para appSettings => configurações da aplicação

**Create View:**

**-**Create partial view => Cria um widget que pode ser usado em mais paginas

<https://bootswatch.com/>

Pagina de templates

O que devo saber ao final:

ASP.NET MVC Fundamentals

Entity Framework (Code-first)

Forms

Validation

Build RESTful Services

Clients-side Development

Auth and Authorization

Performance Optimization

Building a Feature Systematically

**ActionResult:**

Para o MoviesController:

Alterar o ActionResult para ViewResult é uma boa prática, especialmente para Unit Testing

Type ------------Helper Method

ViewResult -> View()

PartialViewResult -> PartialView()

ContentResult -> Content() ->Simple Text

RedirectResult -> Redirect() -> Redireciona o usuário para uma url

RedirectToRouteResult -> RedirectToAction() -> Redireciona para uma ação ao invés de uma URL

JsonResult -> Json() -> Retorna a serialized trace object file

FileResult -> File() -> Retorna um arquivo

HttpNotFoundResult -> HttpNotFound() -> Not Found or 404 error

EmptyResult -> Quando uma ação não precisa retornar nenhum valor, como void

Ctrl + Shift + B -> Recompila o projeto sem precisar fechá-lo

**Action Parameters:**  
Parameters Sources:

Url: /movies/edit/1

In the query string: /movie/edit?id=1

In the form data: id =1

**View Model:**

View model é um modelo construído especificamente para uma View, inclui dados e regras especificas para aquela view

**View models deve ser criada em uma nova pasta do projeto**

**Razor Sintax:**

@foreach...

A engine de MVC chamada de Razor view engine sabe como dividir a sintaxe em C# ou HTML

**Partial Views:**

Partial views são pequenas views que podemos reusar em diferentes views.

NavBar é uma opção de deixar na PartialView

**Entity Framework:**

É uma ferramenta que usamos para acessar um banco de dados, object relational mapper O/RM, mapea dados de um banco de dados em objetos das aplicações

Nos fornecee uma classe chamada DBContext (gateway para nosso banco de dados)

Cada DBContext contem um ou mais DBSets, que representam tabelas no nosso banco de dados

Usamos **LINQ** para consultar os DBSets e com entity framework traduzimos nosso LINQ query para SQL queries at runtime, abre uma conexão com o banco de dados, lê os dados, mapeia para objetos e adiciona eles nos DBSets no nosso DBContext

ADD/Modify/Delete -> DBSet – pedimos para persistir as mudanças, gera automaticamente comandos SQLs e os executa em nosso banco de dados

**Workflows:**

**DBFirst**

Design Database tables primeiro e depois ter entity framework para gerar as domain classes correspondentes

Database -> Entity Framework -> Domain Classes

**CodeFirst**

Direção contrária ao DBFirst

Criamos nossas Domain Classes e deixamos o Entity Framework criar nossas tabelas do banco de dados

Domain Classes -> Entity Framework -> Database

Mais produtivo

Versionamento completo do banco de dados

Muito mais fácil criar um integration test database, com apenas 3 linhas de código

É possível usar em aplicações com banco de dados já construídos, e ainda consegue o versionamento completo dos bancos de dados legado

Te dá completo controle sobre o banco de dados

Code First Migration:

Console Package Manager -> Tools -> NuGet Package Manager

Para usarmos migrations precisamos digitar: enable-migrations

Criará a pasta Migrations no projeto

Criar migration: add-migration NameModel

Criar migration para classes novas criadas: Em Identity Models, DbContext, adicionar um DBSet<NomeClasse> Nome para referenciar a classe

Depois no console de package, criar a migration: add-migration NameModel –force (para override)

Para criar o DataBase depois das migrações : update-database

Para ver o database, em Solution explorer, clicar em show all files e o banco estará em App Data

Não fazer big bang migration, faça pequenas migrações, criar uma pequena e gerar no database

**Seeding the database:**

Não se deve popular a tabela de banco de dados inserindo registros diretamente nela, todos os registros devem vir por Migration

Add-migration Populate”Table”

Add SQL statement no método UP do migration criado

Sql("INSERT INTO MembershipTypes (Id, SinUpFee,DurationInMonths, DiscountRate) VALUES (1, 0, 0, 0)");

Ctrl + E. Ctrl + V duplica linha

**Override regras do banco de dados:**

Na classe do objeto, adicionar [Required] para dizer que o campo é obrigatório, não sera mais nullable

[StringLength()] para definir o tamanho máximo de caracteres

Data Annotations.

**Querying Objects:**

Criar DBContext para acessar o banco de dados

Adicionar private ApplicationDbContext na classe

E chamar o construtor com new ApplicationDbContext();

Ao adicionarmos MembershipTypes por migration, eles são registros que fazem parte da aplicação, então devem entrar no deploy da aplicação, via migration

Ao adicionarmos Customers via hard input no banco de dados, eles não serão deployados na aplicação, pois serão para teste, então não colocamos no migration

**Eager Loading:**

Entity framework só carrega os objetos default de uma classe, não objetos relacionados

Exemplo: Customers não carrega MembershipType properties

Adicionando shortcut:

Tools -> Options -> Keyboard -> Seleciona o local de atalho e depois as teclas de atalho e Assign

**Forms**

The Markup:

Para criar um form, primeiro precisamos de uma Ação.

Usando helper methods o ASP.NET MVC automaticamente adiciona validações para o campo escolhido **TextBoxFor**

**Validation:**

Para adicionar validation, seguimos 3 passos:

-Adicionar Data Anotations nas entidades [Required], [StringLenght]...

-Usar ModelState.Valid para mudar o flow do programa, se o valid state não for valido, retorna a mesma View

-Adicionar Validation Methods ao Form

Quando temos um Id do tipo Byte, a propriedade fica implicitamente required

**Data Annotations:**

[Required] - Obrigatório

[StringLength(255)] -> Tamanho do texto

[Range(1,10)] -> Minimo e máximo valor

[Compare(“OtherProperty”)] -> Para comparar senhas ou email por exemplo

[Phone]

[EmailAddress]

[Url]

[RegularExpression(“...”)]

->todas essas anotações tem mensagens de validação próprias

Para fazer override da mensagem de erro de validação:

[Required(ErrorMessage=”Mensagem”)]]

**Refactoring Magic Numbers:**

MembershipTypeId ==1 <- Magic Number

Adicionar o tipo do membership type dentro da classe de membership types, para referenciarmos na validação

Public static readonly byte PayAsYouGo = 1; <- Exemplo

**Validation Summary:**

Como deixar a lista de erros como uma lista no topo do form

@Html.ValidationSummary() -> Logo embaixo de Begin Form, em cima do form

Para retirar a mensagem de erro na validação de ID, precisamos inicializar o Customer na Action NEW no customer controller

Customer = new Customer()

Para assim, a view setar o VALUE de Id do Customer para 0, já que é inteiro e required

**Client-side Validation:**

As validações acima são feitas no server, fazer as validações no cliente side tbm é importante por que:

Feedback imediato

Não precisa gastar recurso do server toda vez que um user cometer um erro

Usamos o server-side validation para prevenir usuários maliciosos de atravessar(by-pass) o cliente side validation.

Por padrão o client side validation não está habilidado no ASP.NET MVC

Em BundleConfig temos o bundles/jqueryval

No layout podemos ver que RenderSection scripts, está como não required

Podemos adiciona-lo na view que queremos

@section scritps{

@Scripts.Render(“~bundles/jqueryval”) para habilitarmos o cliente-side validation

}

Com o client-side validation habilitado, quando o form detectar erros de input, ele informará o problema sem nenhuma atividade de network.

O bom de data annotations é que ASP.NET MVC usa eles para ambos os lados, client-side validation e server-side validation

Client-side validation só funciona com Data Annotations padrões, na lista acima.

Validações customizadas, como o caso de saber se o cliente tem mais de 18 anos para assinar certos planos, não são aplicadas, para isso, deveríamos criar códigos jquery. Não é muito recomendável fazer isso para atributos customizados, pois se alterar a regra de negócio, também será necessário alterar a validação no client-side e server-side

**Cross-side Request Forgery**

Hacker cria um request em nossa aplicação, de um site diferente

Para prevenir isso, precisamos certificar que o nosso request vem somente do nosso Customer Form

Adicionamos ao pé do Form @Html.AntiForgeryToken()

Esse método cria um Token e colocar como um campo escondido no form e como um cookie no computador do usuário

E no Action Save, adicionamos a anotação [ValidateAntiForgeryToken]

**RESTful services with ASP.NET web API**

**Client -> Request -> Server**

**Server -> HTML -> Client**

Tem uma maneira alternativa de gerar HTML Markup, podems gerar diretamente no cliente, ao invés das ações retornarem Markups, podemos retornar RAW Data

Benefícios:

Menos recursos do server, melhora escalabilidade, por que cada cliente será responsável por gerar suas próprias views

Precisa de menos largura de banda (bandwidth), melhorando performance, os dados chegam mais rápidos ao cliente. A aplicação se torna um pouco mais responsive

Suporta uma ampla gama de clientes, como Mobile, Tablet e Web – Os apps simplesmente pegam os endpoints, pegam os dados e geram a view localmente

Chamamos esses EndPoints de Data Serves ou WebAPIs, por que retornam somente dados, e não mark-ups

Essas Web APIs não são somente para construir Mobile e Tablet Apps, outros sites podem consumir essas APIs e construir novas funcionalidades

E não são somente para GET data mas podem MODIFY data

**Frameworks**

ASP.NET Web API -> Segue os mesmo princípios de arquitetura do ASP.NET MVC

Tem Routing, Controllers, Actions, ActionResults....

Com algumas diferenças pequenas

**API:**

**GET** /api/customers -> Pegar lista de clientes

**GET** /api/customers/1 -> pegar informações de cliente específico

**POST** /api/customers -> Para adicionar um cliente, adiciona customer data no request body

**PUT** /api/customers/1 -> Update a customer

**DELETE** /api/customers/1 -> Para deletar um cliente

**RESTful Convention –** Representational State Transfer

**Building na API**

Dentro de controllers, criar pasta API

Dentro da pasta API, criar new controller, na versão Web API 2 Controller – empty

E depois seguir as instruções do Readme para adicionar os packages no Global.aspax

Depois construímos as ações de Create, Update, Delete e Get no Controller criado

ASP.NET web API tem Media Formatter, o que é retornado na ação é formatado baseado no que o cliente pede

Object -> Media Formatter -> Serialize -> Output (xml, json)

Maioria das vezes o melhor format de output é Json pois é nativo para o código Javascript.

XML é usado mais em grandes empresas, ou empresas governamentais, que estão desatualizadas quanto a tecnologia

**DTO : Data Transfer Object**

É uma estrutura básica de dados usada para transferir dados do cliente para o server

Criar DTOs reduz as chances de quebrar a API, enquanto refatoramos o modelo domínio

O que se aprende é

A API jamais deve receber ou enviar domain objects

Customer... Listo f Customers..., também abre brechas de segurança na aplicação

**Automapper:**

Package Manager

Install-package automapper –version:4.1

Criar nova classe em app\_start com o nome MappingProfile : Profile

Adicionar construtor com os métodos Mapper.CreateMap<ClasseOrigem, ClasseDestino>()

Depois ir em global.asax, adicionar no construtor

Mapping.Initialize(c => c.AddProfile<MappingProfile>())

Depois no controller da API: Alterar onde está customer para customerDto

E adicionar “.Select(Mapper.Map<Customer,CustomerDto>)” Para delegar o context para customerDto

Onde está o getCustomer por ID

No retorno adicionamos Mapper.Map<Customer,CustomerDto>( Aqui passamos o retorno “Customer)

**Camel Notation:**

Para configurar o Camel notation, adicionamos settings dentro do WebApiConfig

URI (Unified Resource Identifier)

**IHTTPActionResult**

Em WebAPIs preferencialmente utilize IHttpActionResult como o retorno das ações

**Para tratar a exception de Id no Update de Movies e Customers**

**Precisamos adicionar**

Mapper.CreateMap<MovieDto, Movie>().ForMember(m => m.Id, opt => opt.Ignore());

**ForMember... no mappingProfile, no mapping de Dto para o Objeto em todas as situações**

**Client Side Development:**

Jquery Ajax para consumir as APIs

Jquery Plugins: BootBox and DataTables (pagination, searching and sorting)

**Jquery**

jQuery é uma biblioteca de Javascprit que nos permite a trabalhar com objetos num documento HTML. O Html que usamos para construir uma página é eventualmente carregada dentro de uma árvore de objetos chamada de Document Object Model (DOM).

No desenvolvimento front-end nos precismoa olhar para certos elementos no DOM e mostrar/ocultar eles baseados em algumas condições. Ou outras tarefas mais complexas que precisamos atravessar e modificar elementos DOM.

jQuery então é uma biblioteca Javascript que nos dá uma API simples e unificada para trabalhar com DOM, independente do browser que o client-side app está rodando.

Jquery é somente uma biblioteca e não uma linguagem de programação.

Document ready handler:

Tudo que você codar em jQuery fica dentro de um desses, isso garante duas coisas, garante que o código não rode enquanto o DOM não está pronto, e confirma que cada elemento acessado realmente existe, para não retornar erros. E também garante que seu código é discreto, separado o conteúdo HTMl da apresentação CSS.

$(document).ready(function() {

//All Jquery code goes here

} );

Selecionando elementos em jQuery

A biblioteca jQuery permite selecionar elementos do seu HTML envelopando eles dentro de $(“ ”) (pode-se usar aspas simples também)

$(“div”); //seleciona todos os HTML div elements

$(“myElement”); // seleciona um elemento HTML como ID “myElement”

$(“.myClass”); //seleciona os elementos HTML com a classe “myClass”

$(“p#myElement); //seleciona elementos de paragrafo com o id “myElement”

$(“ul li a.navigation”); //seleciona anchors com a classe “navigation”

jQuery suporta o uso de todos os CSS selectors, até os de CSS3.

$(“p>a”); //selecion anchors que são filhos diretos de paragraphs

$(“input[type=text]”); //seleciona inputs que tem tipos específicos

$(“a:first”); //seleciona a primeira anchor da pagina

$(“p:odd”); //seleciona todos os parágrafos impares

$(“li:first-child”); //todos os itens de lista que são primeiros filhos da lista

Selectores customizados:

$(“:animated”) //seleciona elementos animados

$(“:button”); //seleciona qualquer botão (inputs or buttons)

$(“:radio”) //seleciona radio buttons

$(“:checkbox”) //seleciona checkboxes

$(“:checked”) //seleciona checkboxes selecionadas ou radio buttons

$(“:header”) //seleciona elementos headers (h1,h2..)

jQuery permite a adicionar, remover e alterar CSS classes.

$(“div”).addClass(“content”) //adiciona classe contente para todos os elementos div   
$(“div”).removeClass(“content”) //remove a classe content de todos os elementos div

$(“div”).toggleClass(“content”) //toggles the class content, adiciona se não tiver, remove se tiver

Também é possível checar se um elemento selecionado tem alguma classe CSS específica, e rodar algum código caso tenha. Ou checar vários também, além de um único elemento, o resultado retorna verdadeiro se algum dos elementos conter a classe

If($(“#myElement”).hasClass(“content”)) {

//do something

}

Estilos CSS podem ser facilmente adicionados usando jQuery, e é feito em um cross-browser fashion.

**$("p").css("width", "400px"); // adds a width to all paragraphs**

**$("#myElement").css("color", "blue") // makes text color blue on element #myElement**

**$("ul").css("border", "solid 1px #ccc") // adds a border to all lists**

Tem várias formas de manipular grupos de elementos em jQuery, inclusive manipular o conteúdo desses elementos (texto, inline elements, etc)

Pegar o HTML de qualquer elemento:

**let myElementHTML = $("#myElement").html(); // variable contains all HTML (including text) inside #myElement**

Se não quiser o HTML, mas só o texto de um elemento:

**let myElementHTML = $("#myElement").text(); // variable contains all text (excluding HTML) inside #myElement**

Usando a mesma sintaxe, é possível mudar o HTML ou o texto do conteúdo de um elemento específico:

**$("#myElement").html("<p>This is the new content.</p>"); // content inside #myElement will be replaced with that specified**

**$("#myElement").text("This is the new content."); // text content will be replaced with that specified**

Para acrescentar conteúdo à um elemento:

**$("#myElement").append("<p>This is the new content.</p>"); // keeps content intact, and adds the new content to the end**

**$("p").append("<p>This is the new content.</p>"); // add the same content to all paragraphs**

jQuery também oferece os comandos appenTo(), prepend(), preprenTo(), before(), insertBefore(), after(), insertAfter(), que funcionam parecidos

Dealing with events in jQuery:

Temos event handlers específicos, que podem ser estabelecidos como o código abaixo:

**$("a").click(function() {**

**// do something here**

**// when any anchor is clicked**

**});**

O código dentro de function só rodará quando anchor for clicado.

Outros eventos que podemos usar jQuery:

Blur, focus, hover, keydown, load, mousemove, resize, scroll, submit, select.

Mostrando e escondendo elementos em jQuery:

Toggling, show or hiding:

**$("#myElement").hide("slow", function() {**

**// do something once the element is hidden**

**}**

**$("#myElement").show("fast", function() {**

**// do something once the element is shown**

**}**

**$("#myElement").toggle(1000, function() {**

**// do something once the element is shown/hidden**

**}**

S parâmetros são opcionais, o primeiro parâmetro indica a velocidade que irá sumir ou aparecer. Se não for indicado vai acontecer instantaneamente sem animação, um número como velocidade, é representada em milissegundos. O segundo parâmetro é uma função opcional que rodará quando o comando terminar de executar.

Também é possível fazer fade in ou fade out de um elemento. Sempre através de animação:

**$("#myElement").fadeOut("slow", function() {**

**// do something when fade out finished**

**}**

**$("#myElement").fadeIn("fast", function() {**

**// do something when fade in finished**

**}**

Para desaparecer um elemento parcialmente, in ou out:

**$("#myElement").fadeTo(2000, 0.4, function() {**

**// do something when fade is finished**

**}**

O segundo parâmetro 0.4 representa a opacidade do elemento, e é similar a opacidade setada de CSS. Dependendo da opacidade que o elemento surgir, ele irá para a opacidade de 0.4, na velocidade indicada. E a call-back function rodará quando a mudança de opacidade tiver completa.

Podemos arrastar elementos, animar elementos e assim como parar animações no meio delas.

**$("#myElement").slideDown("fast", function() {**

**// do something when slide down is finished**

**}**

**$("#myElement").slideUp("slow", function() {**

**// do something when slide up is finished**

**}**

**$("#myElement").slideToggle(1000, function() {**

**// do something when slide up/down is finished**

**}**

Para animar um elemento passamos ao jQuery o CSS style para qual o item deve mudar. jQuery irá setar o novo estilo, mas ao invés de setar instantaneamente, faz gradualmente, animando o efeito desejado.

**$("#myElement").animate(**

**{**

**opacity: .3,**

**width: "500px",**

**height: "700px"**

**}, 2000, function() {**

**// optional callback after animation completes**

**}**

**);**

**Quirks = peculiaridades.**

**Bootstrap dialog box**

**Install-package bootbox**

Incluir no Bundle Config, dentro do Include de Bootstrap

Adicionar “~/Scripts/bootbox.js”,

**Data Tables plugin**

**Install-package jquery.datatables**

Performance, Data Table pega a lista de todos os itens da API, e os armazena internamente para gerar a paginação, searching e sorting. Essa abordagem funciona bem quando temos algumas centenas ou milhares de registros, dependendo do tamanho dos objetos que retornamos da API.

Se estamos retornando um número muito alto de objetos da API, é melhor fazer a paginação, sorting e searching, no server-side, implementar explicitamente, e nos data tables habilitar o server-side processing (ESTUDAR)

**Single Page Applications (SPAs):**

Quando um usuário clica em um link, ao invés de enviar um request para o server, pega-se todo o mark-up da pagina, faz um Ajax call para a API e pega os dados necessários para gerar a view para no cliente.

Então substituímos a view na área principal do conteúdo com uma nova view. Mantemos os outros elementos da view intactos

É mais rápido e mais suave, pois não precisamos recarregar uma página completa quando um usuário navega de uma página para outra

Frameworks para construir SPAs: **Angular**, Backbone e Ember

Angular tem sido o melhor.

Podemos usar ASP.NET Web API e construir várias restfully APIs e podemos usar esses frameworks acima no cliente-side

**Auth and Authorization**

ASP.Net Identity Framework

Restringir todas as funcionalidades para usuários logados

Criar roles diferentes, Admin, limitar adicionar, atualizar e deletar filmes

**Authentication options:**

No Authentication – Aplicações que não precisa de serviços de autenticação e é acessível para usuários anônimos

Individual User Accounts – Opção padrão de autenticação para projetos ASP.Net MVC e é recomendável para Sites de internet, permite social Logins (FACEBOOK, GOOGLE, Twitter...)

Organizational Accounts – É útil quando queremos habilitar o single sign on para internal e cloud apps usando Active Directory

Windows Authentication – Que é útil para Intranet Applications, quando o usuário logar no desktop na network ou rede, eles serão autenticados na aplicação automaticamente

As aplicações ASP.NET MVC já vem com as configurações de Login Log out Register  
Não é preciso recriar os forms para elas

Asp.Net Identidy framework, são as propriedades dentro de References no Solution Explorer que têm os Sufixos AspNet.Identity

**Architecture ASP.Net Identity**

**Domain classes:** IdentityUser, Role

**API/Service classes:** UserManager, RoleManager, SignInManager

Essas classes conversam com outro grupo de classes que representam a Persistent Store do ASP.Net Identity

**Persistence classes:** UserStore, RoleStore

ASP.Net identity nos dá uma implementação dessa persistente store usando entity framework e um banco de dados relacional, mas você também pode fazer sua própria implementação de Persistent Store, como um NoSql Data Store

Na primeira migração feita no projeto, criamos diferentes tabelas na migração inicial: aspNetRoles, AspNetUserRoles...

Essas tabelas são geradas baseadas no domain model do Asp.Net Identity Framework

Em models:

As classes IdentityUser e IdentityDbContext, são ambas parte ASP.NET Identity Framework

É possível adicionar o migration para registros criados no banco

Acessando os registros do banco em App Data, selecione os registros desejados, botão direito e Script, vai gerar os dados para inserção dos registros

Depois no migration criado, adicionatos Sql(@” SCRIPT AQUI “)

Para trabalhar com privilégios de roles, é melhor criar novas views para usuários com menos privilégios, para facilitar a manutenção.

**OAuth**

Facebooki, Google, Twitter, Microsoft... e outros external authentication providers usam um protocolo de autenticação chamado OAuth ou Open Authorisation.

Primeiro precisamos registrar nossa aplicação no facebook para criar uma espécie de parceria. Facebook nos fornece uma chave de API e um segredo (secret), como uma senha, e usamos isso para comunicar com o facebook por debaixo dos panos.

Entao quando um usário loga na nossa aplicação usando o Facebook, nos redirecionamos ele usando {key, secret} para que o Facebook saiba que o request esteja vindo ad nossa aplicação. Para nos prevenir de usuários maliciosos usamos Https ou Secure HTTP protocol para que os dados trocados entre essas partes sejam criptografados e ninguém consiga interceptar essa comunicação. Depois de logado no Facebook, o site é redirecionado para a nossa aplicação e Facebook nos envia um Authorization Token, que nos diz que o facebook conseguiu autenticar esse usuário com sucesso. Depois mandamos novamente para o Facebook, o {token, key, secret} para evitar que hackers nos mandem um token aleatório e verificar se realmente veio do Facebook e retornará o sucesso do request com um {access token}

Para usar Social Logins, precisamos seguir dois passos:

* Enable SSL (para que nossa comunicação com o facebook seja em um canal seguro)
* Registrar nossa aplicação no Facebook para Pegar nossa key e secret.

Passo a passo:

Selecionar o projeto no solutions explorer e apertar F4

Em properties, setar SSL Enabled para True

Copiar SSL URL

Voltar no solution Explorer

Projeto -> Properties -> Web -> Project URL -> Colar SSL URL

Registrando a aplicação no Facebook:

Developers.facebook.com

Criar o aplicativo

Tipo Web, colocar o caminho Https

Depois em Configurações -> Basico, teremos as informações de ID e Chave do App

Depois no projeto, em App\_start -> Startup.Auth, temos uma seção comentada de UseFacebook Authentication, devemos inserir as chaves nela

**Performance:**

Otimização prematura é a raiz de todos os males.

Não sacrifique a manutenibilidade do seu código por otimização de performance

**Three Tier Architecture**

**Tier:** Onde o código roda

Data Tier – Onde nosso banco de dados e queries ficam (SQL Server)

Application ou Middle Tier – webservice hosting our application (IIS) – servidor web criado pela microsoft

Client Tier – Computador do cliente (browser), front-end

Geralmente, a maioria dos gargalos de performance estão no Data Tier, depois em Application Tier e depois no cliente Tier, no Client Tier, temos menos resultados visíveis

**Regras:**

Não sacrifique a manutenibilidade do seu código com otimização prematura

Seja realista e pense como um engenheiro e não como um computador.

Seja pragmático e garanta que seus esforços tenham resultados observáveis e gere valor

**Data Tier:**

Geralmente os problemas nesse tier são baseados nos Schemas ou Queries

Certifique que suas tabelas tenham suas chaves primárias e seus relacionamentos entre as tabelas

E também deve usar indexes nas colunas que usamos para filtrar registros nas queries

Problemas comuns de **Schema**:

* Não indexarem as colunas de foreign key, não criar os relationships e não ter primary key
* **EAB (Entity-Attribute-Value)** : Ao invés de termos tabelas concretas, como Movies, Customers, Rentals, nosso banco de dados tem um punhado de tabelas chamadas Entities, Attributes and Values e umas outras tabelas de suporte. As pessoas que costumam gostar deste tipo de schema, argumentam que não importa quantas classes temos na nossa aplicação domain model, não precisamos mudar o database schema, por que tudo pode ser modelado de uma maneira genérica, usando entidades, atributos e valores. Apesar dessa argumentação ser tecnicamente correta, tem um custo muito grande com essa abordagem
  + Não podemos usar O/RMs (Object Relational Mappers) como Entity Framework
  + Longas e gigantes queries: temos que escrever todas as queries à mão e essas queries podem facilmente se tornar gigantes
  + Extremely Slow: Isso torna nossa aplicação lenta, algumas queries podem levar cerca de meio minuto para rodar
  + FIQUE LONGE DESTE PADRÃO

Problemas nas **Queries:**

* Devemos ficar de olho nas queries que Entity Framework roda no nosso banco de dados, se as queries estiverem muito complexas, é melhore criar um store procedure e escrever uma query otimizada para ter o mesmo resultado.
* **Usar Execution Plan in SQL Server** para ver quais partes da query tem custos maiores então podemos otimizar a query reescrevendo ou então podemos criar indexes adicionais no banco de dados
* Se mesmo após otimizarmos as queries elas continuarem lentas, podemos considerar criar **um banco separado para as queries (Um Read Database)** tem algumas arquiteturas como CQRS (Command Query Responsability Segregation) que nos guia para essa direção. Faz sentido pois lemos os dados mais do que os modificamos. Nesse banco de dados nos já fazemos um pré-join algumas das tabelas para acelerar seu processamento, pois join afeta muito a performance da query (lembre-se que essa abordagem tem o custo de termos que fazer manutenção de dois bancos de dados sincronizados) – Não recomendável
* Usar **Caching – Com cache** nos rodamos uma query lenta e armazenamos ela na memória, os requests seguintes serão pegas desse cache da memória

**Glimpse:** Ferramenta para ter diagnostico real time e insights da aplicação

Package Manger -> install-package glimpse

Package Manger -> install-package glimpse.ef6 (entity framework 6)

Instalar os dois acima

Roda a aplicação depois acessa o link da homepage/glimpse.axd

(OBSOLETO)

**Application Tier:**

Output Caching Technique: [Cache para HTML page]

Podemos pegar uma página da nossa aplicação, sabendo que é uma página que não muda muito frequentemente, e que é uma página pesada e cachar o output dessa página que o HTML renderizado

Client -> Application -> Database (primeira vez)

Client -> Application -> Cache (Segunda vez) – evitando a execução da querie

Adicionar [OutputCache(Duration = X)] no topo das ações dos controllers

Podemos cachar os dados no cliente ou no server. Se a view é específica para um usuário, colocamos no cliente senão, colocamos no server.

Adicionamos a propriedade Location

[OutputCache ( Duration = X , Location = OutputCacheLotation.Server/Client)

Podemos usar o paramtetro VaryByParam, para cada tipo do parâmetro especificado teremos uma versão diferente da pagina cachada/armazenada

Só faça output cache depois de testar a performance da sua página e comprovar que ela é lenta e que há vantagem e cachar ela.

Para desabilitar Caching:

[OutputCache ( Duration = 0, VaryByParam = “\*”, NoStore = true)]

**Data Caching:**

MemoryCache.Default[“Nome do parâmetro”] = \_context.Classe.ToList() => Armazena os dados que queremos deixar no cache.

Só usar essa técnica se realmente precisar e depois de ter feito performance profiling.

E somente usada para ações que vão ler dados e não modifica-los

Deve ser usado para acessar dados frequentemente que não mudam muito.

Cuidado, pois usando essa técnica sem pensar, estaremos aumentando o consumo de memória da nossa aplicação e levando para todos os tipos de complexidades desnecessárias tanto para arquitetura e code level

Evitar o uso de magic strings “ “

**Async** não melhora a performance da aplicação e sim escalabilidade, porém o banco de dados sempre será um gargalo.

Usando SQL Cluster, NoSQL Database (Mongo, Raven) –modelados para escalabilidade ou SQL Azure, podemos melhorar o gargalo e a escalabilidade do banco de dados.

**Release Builds:** Mudar a forma de buildar a aplicação de Debug para Release quando lançar a aplicação melhora a performance geral dela, pois no Debug Mode criam-se alguns dados para debugar o código.

**Disabling Session:**

Session é um pedaço de memória no web server alocada para cada usuário. Podemos usar isso para armazenar dados temporários durante a sessão do usuário.

Qual o problema de usar Session, quanto mais usuários tivermos mais memória do web server nossa aplicação irá usar, mata a escalabilidade da aplicação.

Devemos deixar nossa aplicação Stateless, que significa, um request vem, é processado e fim, não fazemos manutenção do estado, não usamos Session.

Para desabilitar:

Vamos em Web.Config => em <system.web>

adicionamos : <sessionState mode = “Off”></sessionState>

**Client Tier:**

Client -> Server (como essas máquinas estão localizadas em diferentes áreas geográficas, devemos reduzir a quantidade de requests do cliente para o server e também o tamanho das respostas)

DTO - Ao construir API, manter os DTOs leves (exceto se for uma API pública, que será utilizada por várias aplicações)

JS – Reduzir requests combinando-os e para reduzir o tamanho do request, comprimimos os resultado

CSS – Reduzir requests combinando-os e para reduzir o tamanho do request, comprimimos os resultado

@Scripts e @RenderSection nas views, devem estar sempre perto do fim do Body, para carregar os mark-ups primeiro e depois carregar os scripts, renderizar o conteúdo primeiro

Ir no Web.Config <system.web> e remover o debug property do compilation reduz a quantidade de requests do JavaScript

Image -

Durante o desenvolvimento é recomendável deixamos o Disable Cache no Chrome Development Tools ativado, para não deixarmos nada da nossa aplicação em Cache e prol de debugar

**Building Feature End to end**

Understand the problem

Get the use case

Define the Inputs and Outputs

**Action**:

To return Markup use MVC Controller

To return Data e gerar mark-up no client use API Controller

@Ajax.BeginForm é uma ação destinada a MVC controllers e não API controllers, então usamos RAW Form

**Add Autocomplete no projeto**

Package manager -> Install-package Twitter.Typeahead

Depois em Bundle config adicionar

~/scripts/typeahead.bundle.js

Ir na pagina de examples do typeahead.js e copiar o código:

.typeahead {

background-color: #fff;

}

.typeahead:focus {

border: 2px solid #0097cf;

}

.tt-query {

-webkit-box-shadow: inset 0 1px 1px rgba(0, 0, 0, 0.075);

-moz-box-shadow: inset 0 1px 1px rgba(0, 0, 0, 0.075);

box-shadow: inset 0 1px 1px rgba(0, 0, 0, 0.075);

}

.tt-hint {

color: #999

}

.tt-menu {

width: 422px;

margin: 12px 0;

padding: 8px 0;

background-color: #fff;

border: 1px solid #ccc;

border: 1px solid rgba(0, 0, 0, 0.2);

-webkit-border-radius: 8px;

-moz-border-radius: 8px;

border-radius: 8px;

-webkit-box-shadow: 0 5px 10px rgba(0,0,0,.2);

-moz-box-shadow: 0 5px 10px rgba(0,0,0,.2);

box-shadow: 0 5px 10px rgba(0,0,0,.2);

}

.tt-suggestion {

padding: 3px 20px;

font-size: 18px;

line-height: 24px;

}

.tt-suggestion:hover {

cursor: pointer;

color: #fff;

background-color: #0097cf;

}

.tt-suggestion.tt-cursor {

color: #fff;

background-color: #0097cf;

}

.tt-suggestion p {

margin: 0;

}

Criar uma nova Style Sheet na pasta de contente com o nome typeahead.css e colar o código acima.

Depois adicionar no Bundle Config o nome do arquivo no CSS bundle

Acessar a pagina

<https://twitter.github.io/typeahead.js/examples/#remote>

é um typeahead de exemplo com um datasource remote, que é o nosso caso, web api

Copiamos o código javascript e colamos no arquivo do formulário embaixo como @section scripts

Trocamos a propriedade do whitespace para o nome da propriedade que desejamos auto preencher no caso ‘value’ para ‘name’

Trocar a url para a nossa

**Toaster:**

**Package-manager:** install-package toaster

Adicionar no bundle config CSS e Lib

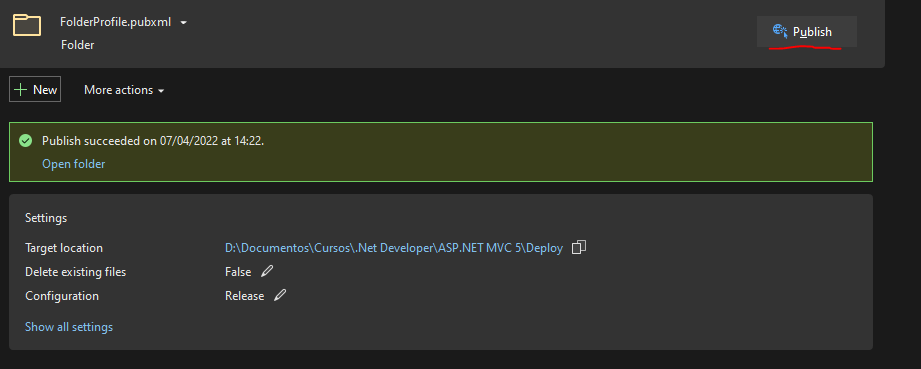
**Deploying:**

**Deploying app**

Botao direito no nome do projeto e publish

Folder -> Seleciona a pasta desejada -> Finish

Depois de checar se esta tudo certo, Apertar Publish na janela que aparece



**Deploying Database:**

Ao menos que vc esteja usando WebDeploy e usando Code-First migration, você deve deployar seu banco de dados manualmente.

Package Manager -> Update-database –script (isso irá gerar um SQL com todas as migrações, só devemos usar quando vamos migrar o banco de dados a primeira vez)

-> Update-database –script –SourceMigration: NomeDaMigração (isso gerara o script a partir da migração que deseja)

Para ver as migrações que rodamos, podemos ver dentro das tabelas do bando a tabela \_MigrationHistory

A vantagem de usar code-first migration é que sempre que fizermos alterações em reference data, tudo esta nas nossas migrações, schema e data changes, em caso de lançarmos uma nova versão, precisarmos modificar dados de referência, tudo estará incluso na nossa migração.

Alterar isso manualmente, gasta muito esforço.

**Bulding Configurations:**

Criar configuração de compilação para testes ou ambiente de testes.

Selecionar Configuration Manager clicando no dropdown do Debug do Visual Studio

Em Active Solution Configuration, selecionar NEW, dar um nome

Para Configurações de teste, é recomendável usar os settings de release, usar Debug somente quando estiver em desenvolvimento.

Clikar com botão direito em Web.Config e Add Config Transform, irá aparecer uma nova config.

**Application Settings:**

Geralmente as aplicações, temos mais de uma configuração que impacta como a aplicação opera. Pode usar um mail service para enviar e-mails, então deve saber o endereço, usuário e senha.

Entao como boa pratica, não armazenamos essas configs no código e sim em um arquivo de configuração.

Web.config

<appSettings>

Para ver as alterações, podemos clicar com botão direito na configuração criado e clicar em preview Transform

Encriptar dados:

aspnet\_regiis -pef "appSettings" "D:\Documentos\Cursos\.Net Developer\ASP.NET MVC 5\Deploy" -prov "DataProtectionConfigurationProvider"

Decriptar dados:

aspnet\_regiis -pdf "appSettings" "D:\Documentos\Cursos\.Net Developer\ASP.NET MVC 5\Deploy"

Elmah:

Gerenciador de erros

Install-package elmah

Pagina /elmah.axd para verificar

É possível configurar email, permitir somente certos usuários a acessarem a pagina, etc