Relembrando

- Linguagem C#
- Atividade avaliativa (UML Banco)

Conteúdo previsto

- Nas próximas aulas teremos uma introdução ao desenvolvimento web com **ASP. NET Core MVC** e o acesso ao banco de dados SQLServer com **Entity Framework**.
- Iremos fazer pequenos sistemas durante o semestre.
- Nesta aula departamentos, seus vendedores e as vendas.
- Vamos aprender os fundamentos de desenvolvimento MVC com template engine **Razor** com layout Bootstrap, vamos a fazer operações de CRUD e pesquisas usando **LINQ**.

Requisitos

- Visual Studio 2017
- ASP.NET Core 2.1
- Entity Framework Core
- SQL Server

Visão geral do ASP.NET Core MVC

- É um framework para criação de aplicações web
- Criado pela Microsoft e comunidade
  - Open source
- O framework trabalha com uma estrutura bem definida, incluindo:
  - Controllers
  - Views
  - Models
    - View Models

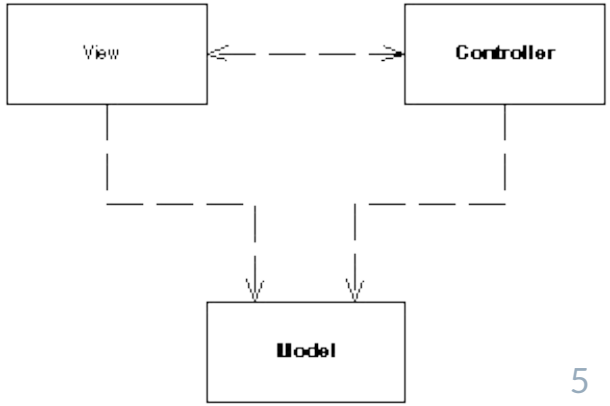
https://docs.microsoft.com/en-us/aspnet/core/mvc/overview

MVC

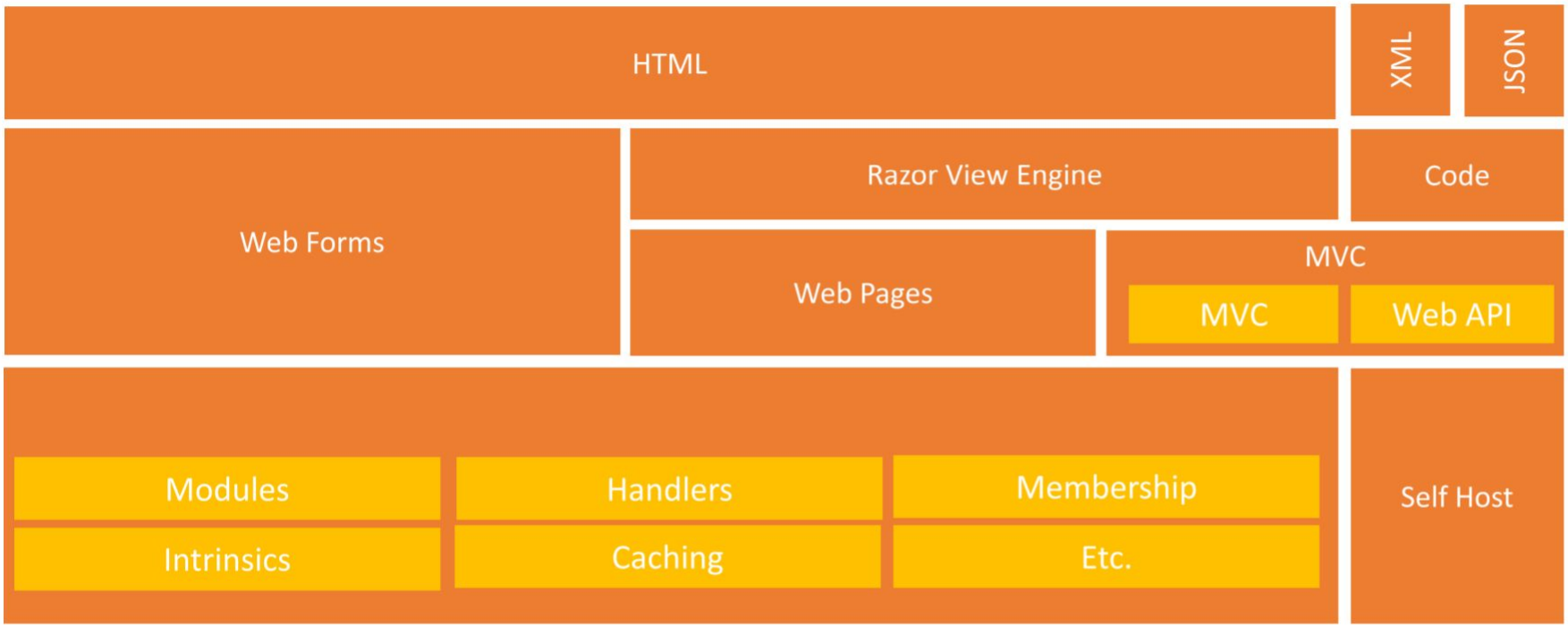
Model: objetos que representam o domínio da aplicação (Entidades e Serviços)

View: componentes que apresentam a interface de usuário

Controller: componentes que tratam a interação com o usuário, operam sobre modelos e selecionam a visão adequada



.NET web


HTML
XML
JSON
Web Forms
Razor View Engine
Code
Web Pages
MVC
MVC
Web API
Modules
Handlers
Membership
Intrinsics
Caching
Etc.
Self Host

ASP.NET Core MVC

- A maior parte da configuração se dá por convenções
  - Nomes de classes e pastas
  - Associações entre controladores e visões
  - Roteamento
- Todos os componentes podem ser customizados

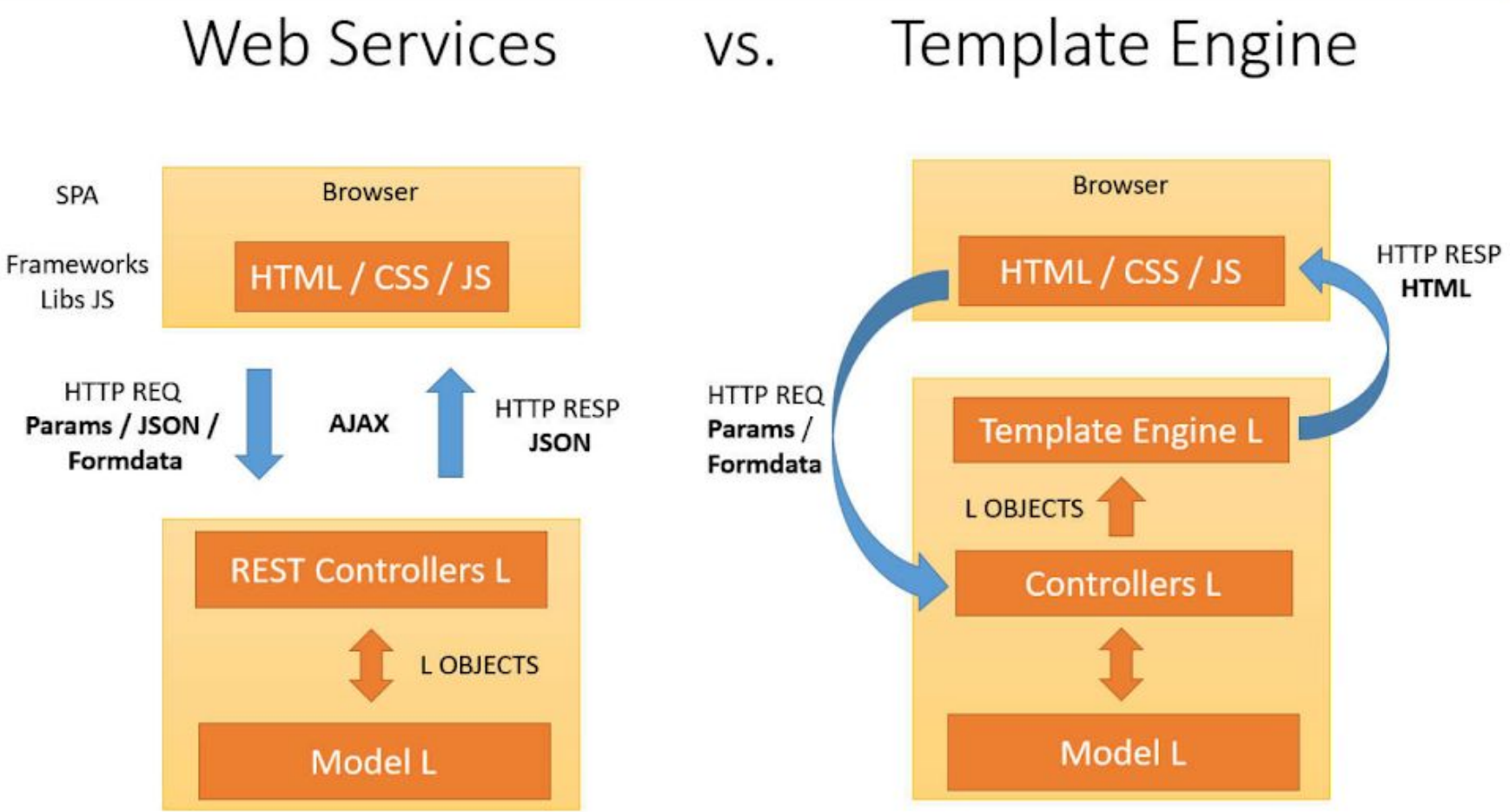
Internet Information Services (IIS)

- Conjunto integrado de serviços para um servidor Web
- Permite publicar conteúdo e disponibilizar arquivos e aplicações em um ambiente Internet/Intranet
- Dotado de uma interface administrativa gráfica
- Baseado no conceito de Diretório Virtual

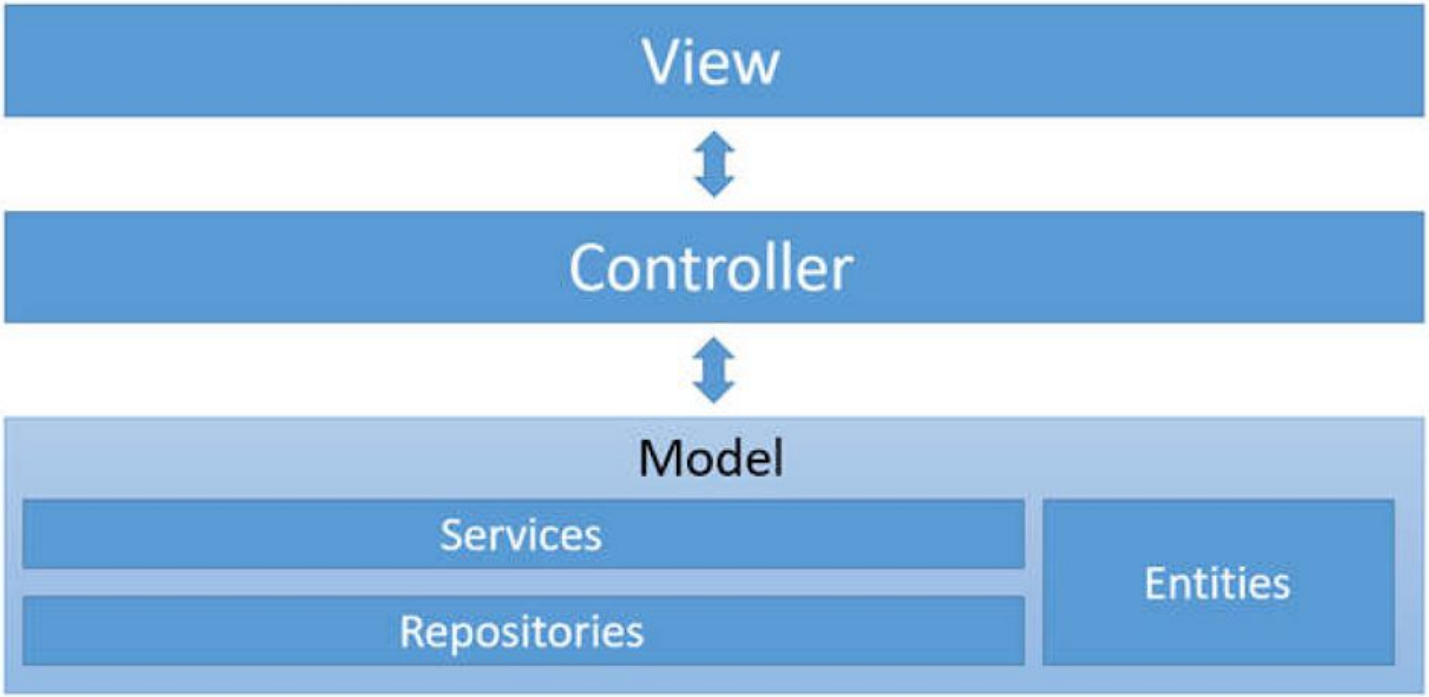
IIS Express

- Utilizado durante o desenvolvimento da aplicação
- Não necessita de configurações adicionais

Aplicações web MVC com template engine



Arquitetura Geral

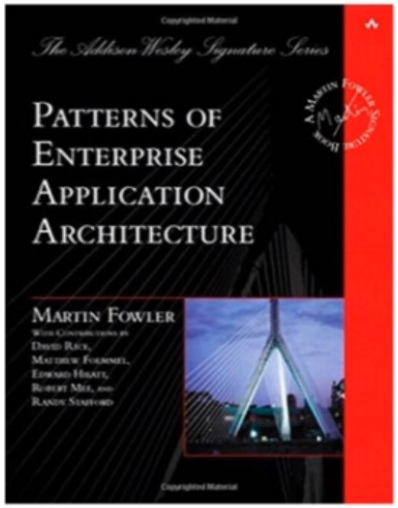


Frameworks/Libs JS (web services)

Frameworks e Templates Engines (MVC)

Problema

Por muitos anos, uma grande dificuldade de se criar sistemas orientados a objetos foi a comunicação com o banco de dados relacional.



Martin Fowler: ~30% do esforço de se fazer um sistema

Exemplo simples

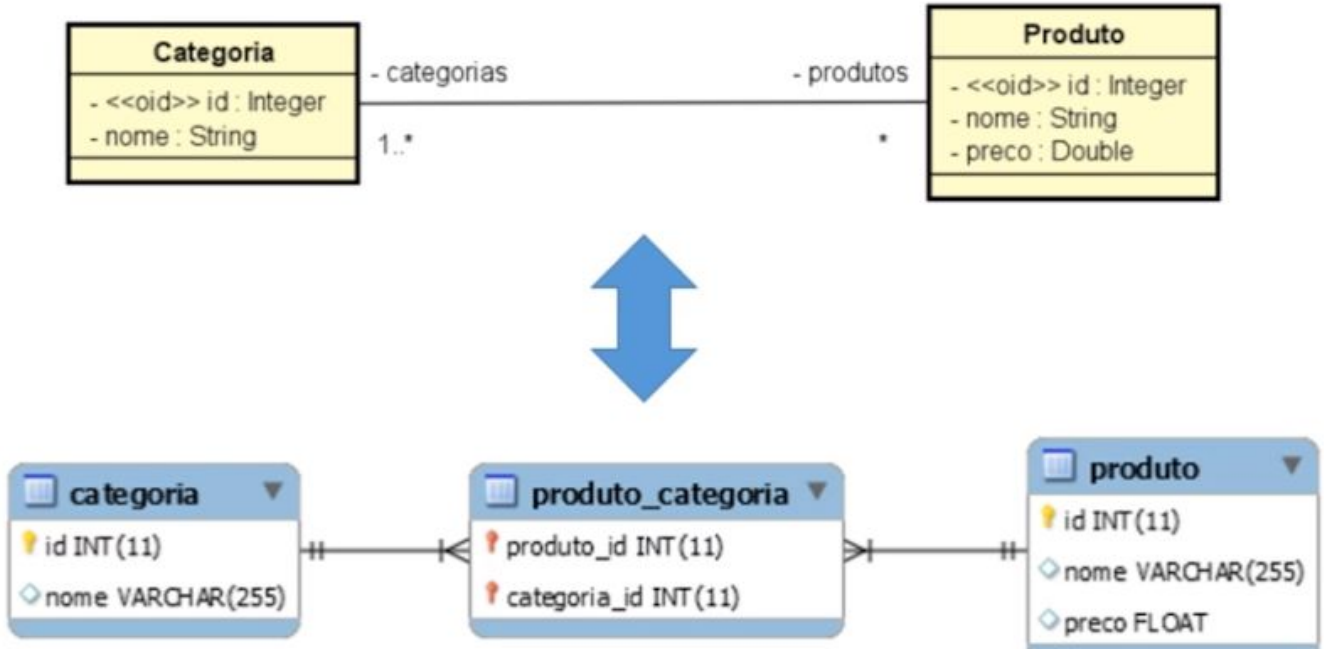
```
Client client = null;
using (connection)
{
    using (var command = new SqlCommand("SELECT * FROM Clients WHERE Id = @id;", connection))
    {
        command.Parameters.Add(new SqlParameter("@id", id));
        connection.Open();
        using (var reader = command.ExecuteReader())
        {
            if (reader.Read())
            {
                client = new Client();
                client.Id = reader.GetString(0);
                client.Name = reader.GetString(1);
                client.Email = reader.GetString(2);
                client.Phone = reader.GetString(3);
            }
        }
    }
}
return client;
```

Outras questões que devem ser tratadas

- Contexto de persistência (monitorar alterações nos objetos que estão atrelados a uma conexão em um dado momento)
  - Alterações
  - Transação
  - Concorrência
- Mapa de identidade (cache de objetos já carregados)
- Carregamento tardio (lazy loading)
- Etc.

Solução: Mapeamento Objeto-Relacional

ORM (Object-Relational Mapping): Permite programar em nível de objetos e comunicar de forma transparente com um banco de dados relacional



Entity Framework

https://docs.microsoft.com/en-us/ef/

Entity Framework Core

EF Core is a lightweight, extensible, and cross-platform version of Entity Framework.

Entity Framework 6

EF 6 is a tried and tested data access technology with many years of features and stabilization.

https://docs.microsoft.com/en-us/ef/core/providers/index

Principais Classes

- **DbContext:** um objeto DbContext encapsula uma sessão com o banco de dados para um determinado modelo de dados (representado por DbSet's).
  - É usado para consultar e salvar entidades no banco de dados
  - Define quais entidades farão parte do modelo de dados do sistema
  - Pode definir várias configurações
  - É uma combinação dos padrões Unity of Work e Repository
    - **Unity of work:** "mantém uma lista de objetos afetados por uma transação e coordena a escrita de mudanças e trata possíveis problemas de concorrência" - Martin Fowler.
    - **Repository:** define um objeto capaz de realizar operações de acesso a dados (consultar, salvar, atualizar, deletar) para uma entidade.
- **DbSet<TEntity>**: representa a coleção de entidades de um dado tipo em um contexto. Tipicamente corresponde a uma tabela do banco de dados.

Processo geral para se executar operações

Criação do projeto

- File -> New -> Project -> Visual C# -> Web -> ASP.NET Core Web Application
- Defina o diretório para criação do projeto
- **Já pode criar um repositório GIT**
- ASP.NET Core Web Application (Model-View-Controller)
- (Desabilitar) authentication
- (Desabilitar) Enable Docker Support
- (Desabilitar) Configure for HTTPS

- Run project
  - Com debug: F5
  - Sem debug: CTRL+F5
    - Live reloading

GIT! GIT! GIT!

- Abram o gitbash
- *git status* (para verificar se há algo para commitar)
- *git log --online* (mostra a lista de commits)

- criar repositório no github pi3-asp-net-core-mvc
- Não marquem commit inicial, pois ja existe.
- vamos copiar o comando git remote add origin para associar esse repositório com o diretório do meu computador.
- git remote add origin *https://github.com/nome_usuario/nome_diretorio.git*
- Agora vamos usar o comando:
- *git push -u origin master*

Estrutura do projeto

Connected Services ->Pode estar conectado com serviços online, como Azure
Dependencias são as dependencias que o projeto vai usar
Propriedades do projeto
wwwroot -> recursos do frontend - os arquivos css, imagens, js, outras bibliotecas como bootstrap, jquery
Controles
Modelos
Views -> se eu tiver Cadastro de produtos, terei uma pasta chamada “produtos” com os documentos html

- Por padrão já tem a pasta Home, que é controlado pelo HomeController
- Note o nome do método do controller, você pode acessar usando o nome do método na URL
- Páginas Razor são .cshtml, que permitem tanto html quanto C#
  _Layout -> define layout geral e faz importações js, bootstrat
  _ViewStart -> define o layout base
  _ViewImports -> Define outras importações
- A pasta shared são páginas compartilhadas por várias páginas.
- O _Layout é a base do layout geral da aplicação e é nela que são importadas as bibliotecas de html/js
- O viewimport e viewstart (laout vai ser com base no arquivo _layout.cshtml) tem a definição do que a aplicação vai usar

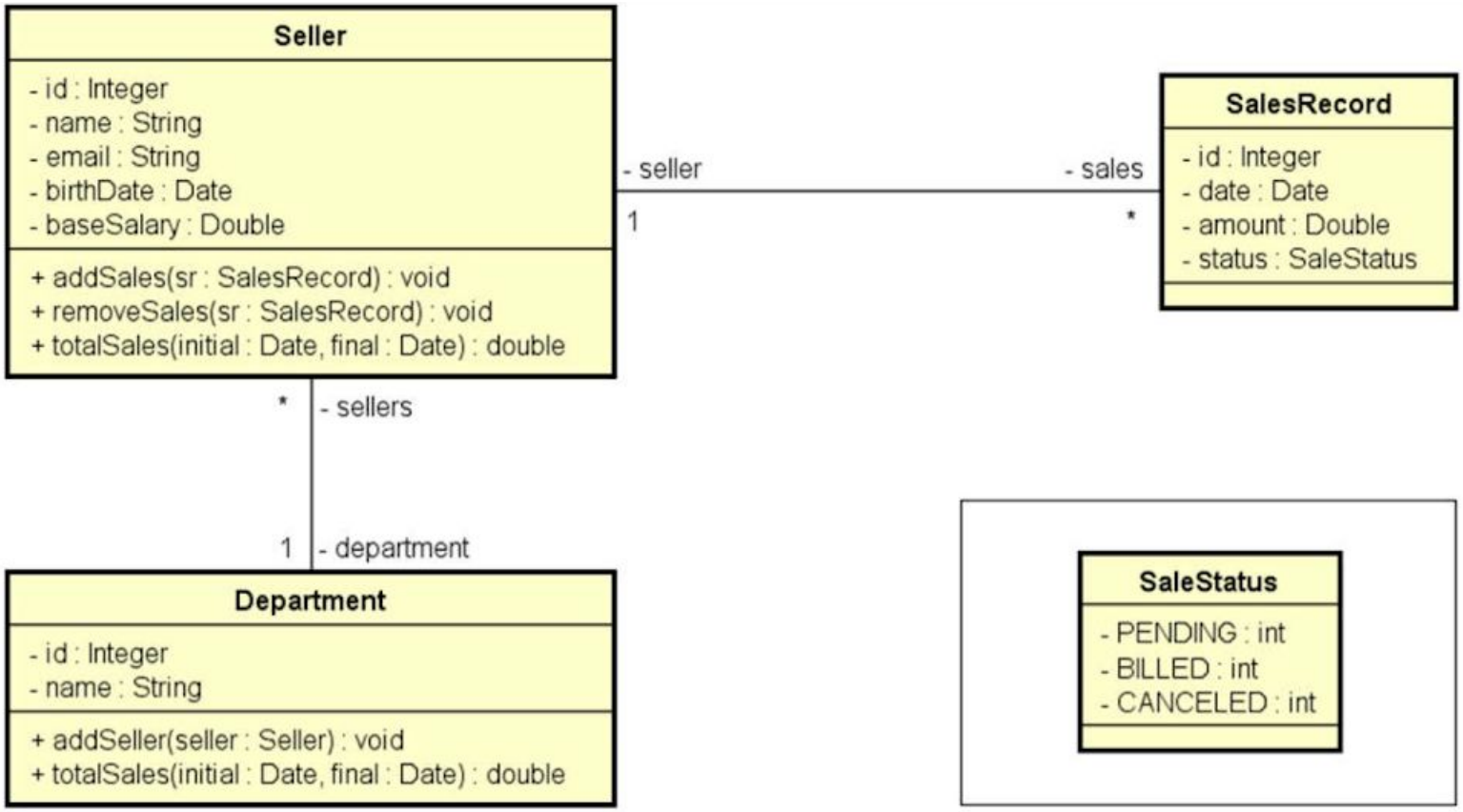
Appsettings.json vai conter configuração de recursos externos, como log, e banco de dados.
Program é o ponto de entrada -> contém o método que inicia a classe startup
Startup.cs Configuração do meu projeto. Tem dependência com configuration… e tem um construtor que recebe ele
Duas operações principais: *ConfigureServices, Configure (questões relacionadas ao comportamento das requisições - pipeline http - pode ser usado para add middleware...tem rotas)*

Primeiro controller e páginas Razor

- Padrão de rota Controller / Action / Id
  - Cada método do controlador é mapeado para uma ação
- Templates Natuais (nomeclatura)
- C# block in Razor Page: @{ }
- ViewData dictionary
- Tag Helpers in Razor Pages. Examples: asp-controller and asp-action
- IActionResult

Interface IActionResult

Type	Method builder
ViewResult	View
PartialViewResult	PartialView
ContentResult	Content
RedirectResult	Redirect
RedirectToRouteResult	RedirectToAction Ex: RedirectToAction("Index", "Home", new { page = 1, sortBy = price}))
JsonResult	Json
FileResult	File
HttpNotFoundResult	HttpNotFound
EmptyResult	-


Seller
- id : Integer
- name : String
- email : String
- birthDate : Date
- baseSalary : Double
+ addSales(sr : SalesRecord) : void
+ removeSales(sr : SalesRecord) : void
+ totalSales(initial : Date, final : Date) : double
- seller
1
- sales
*
SalesRecord
- id : Integer
- date : Date
- amount : Double
- status : SaleStatus
* - sellers
1 - department
Department
- id : Integer
- name : String
+ addSeller(seller : Seller) : void
+ totalSales(initial : Date, final : Date) : double
SaleStatus
- PENDING : int
- BILLED : int
- CANCELED : int

First Model-Controller-View - Department

- Criar pasta **ViewModels** e mover ErrorViewModel (incluindo namespace: models.viewModels) para essa pasta
  - CTRL+SHIFT+B *arrumar referências*
- Criar classe **Models/Department**
- Criar controller: botão direito em Controllers -> Add -> Controller -> MVC Controller Empty
  - Name: Department**s**Controller (PLURAL)
  - Cria uma lista List<Department>
  - return View(list)
- Create new folder **Views/Departments** (PLURAL)
- Create view: right button Views/Departments -> Add -> View
  - View name: Index
  - Template: List
  - Model class: Department
  - Modificar título da página para Departments
  - Note:
    - @model definition
    - intellisense for model
    - Helper methods
    - @foreach block

Deletando View e Controller de Department

Delete DepartamentsController

Delete pasta Views/Departments

CRUD Scaffolding

- Botão direito em Controllers -> Add -> New Scaffolded Item
  - MVC controllers with views, using Entity Framework
  - Model class: Department
  - Data context class: + aceita o nome sujerido
  - Views (options): marcar as 3
  - Controller name:
    - DepartmentsController

Banco de dados e migrações

- Em appsettings.json, set connection string:
  - "server=localhost;userid=developer;password=1234567;database=saleswebmvcappdb"
- Em Startup.cs, fix DbContext definition for dependency injection system:
  - AddDbContext<SalesWebMvcAppContext>(options => options.UseMySql(Configuration.GetConnectionString("SalesWebMvcContext"), builder => builder.MigrationsAssembly("SalesWebMvc")));
- Install MySQL provider:
  - Open NuGet Package Manager Console
  - *Install-Package Pomelo.EntityFrameworkCore.MySql*
- Stop IIS
- CTRL+SHIFT+B
- Start MySQL server:
  - Control Panel -> Administrative Tools -> Services
- Start MySQL Workbench
- Package Manager Console -> create first Migration:
  - *Add-Migration Initial*
  - *Update-Database*
- Check database in MySQL Workbench
- Test app: CTRL+F5

Git time!

Git add .

Git commit -m “criando crud dep”

Git push

Estilizando com bootstrap

- Escolha um tema em *bootswtch.com/3*
- Salva como bootstrap-nometema.css
- Copia o arquivo e cola pelo visual studio
  - www/lib/bootstrap/dist/css
- shared/_Layout -> atualiza ref do bootstrap
  - Environment development

Contato e repositório

rrdoliveira@senacrs.com.br

https://github.com/romuloreis/PI3

Ordem das etapas

Criar classes
Tipos básicos
Associações (ICollection)
Construtores
Métodos customizados
Adicionar DB Set’s no DB Context ()
Nova Migração