Curso: C# COMPLETO - Programação Orientada a Objetos + Projetos

http://educandoweb.com.br

Prof. Dr. Nelio Alves

Projeto: GUI Web com ASP.NET Core

Objetivo geral:

- Introduzir o aluno ao desenvolvimento de aplicações web com ASP.NET Core MVC
- Permitir que o aluno conheça os fundamentos e a utilização do framework, de modo que ele possa depois prosseguir estudando as especificidades que desejar

PROJETO NO GITHUB:
https://github.com/acenelio/workshop-asp-net-core-mvc

Visão geral do ASP.NET Core MVC

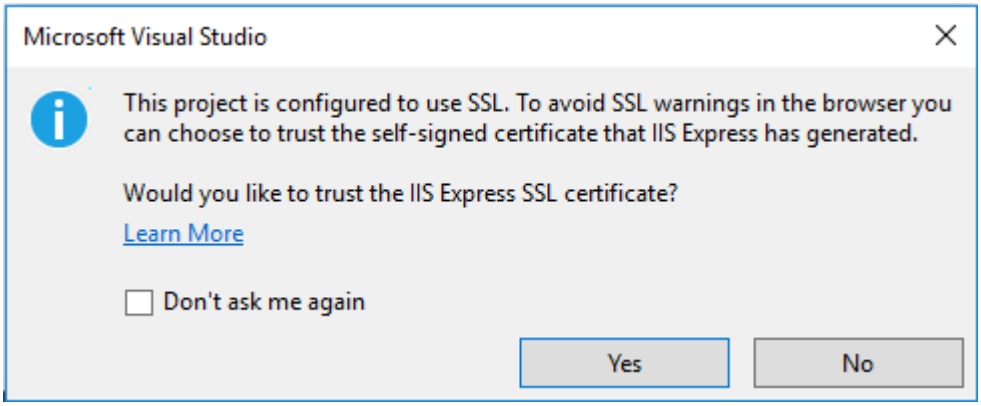
- É um framework para criação de aplicações web
- Criado pela Microsoft e comunidade
- Open source
- Roda tanto no .NET Framework quanto no .NET Core
- O framework trabalha com uma estrutura bem definida, incluindo:
  - Controllers
  - Views
  - Models
    - View Models
- https://docs.microsoft.com/en-us/aspnet/core/mvc/overview

Project creation

Checklist:

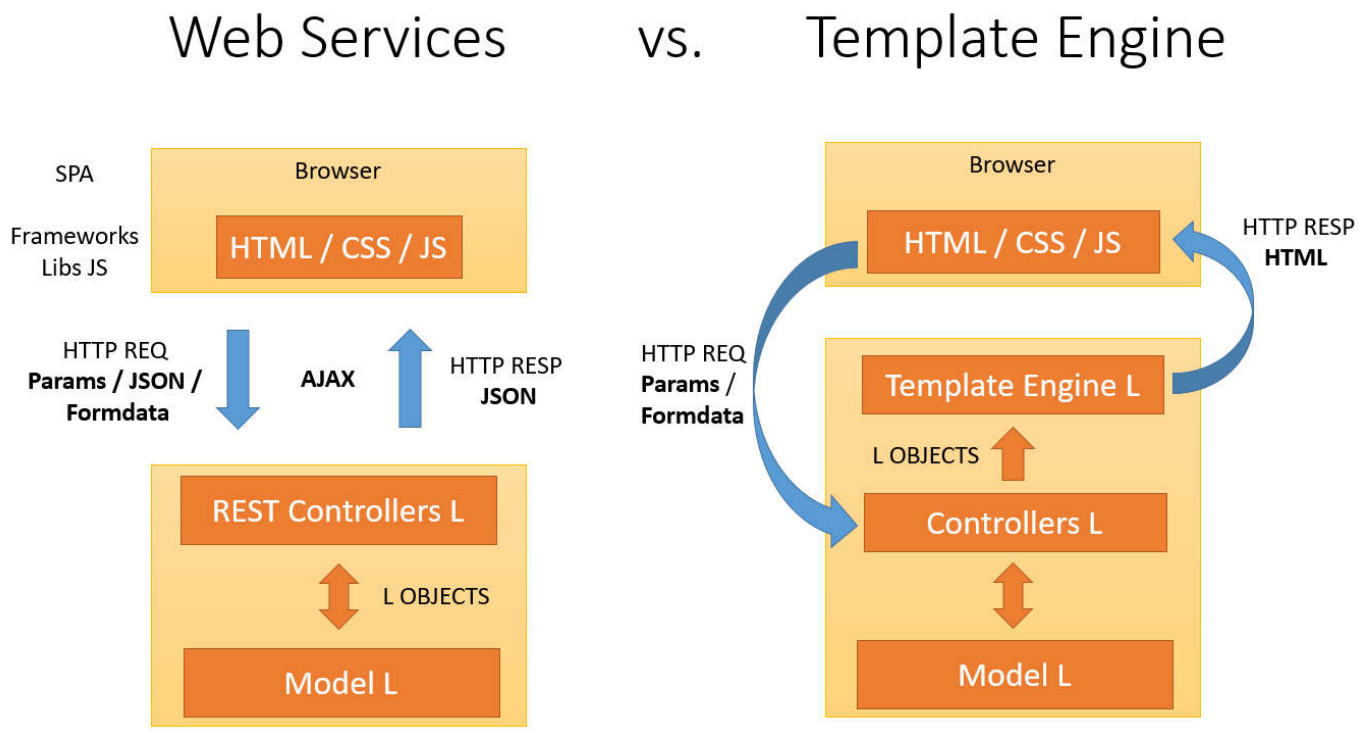
- File -> New -> Project -> Visual C# -> Web -> ASP.NET Core Web Application
  - Create directory for solution
  - Create new Git repository
  - Web Application (Model-View-Controller)
  - (NO) authentication
  - (NO) Enable Docker Support
  - Configure for HTTPS
- Observe project folder and commits
- Run project
  - With debug: F5
  - Without debug: CTRL+F5
    - Live reloading
    - It's possible to stop IIS manually
- Create remote Git repository and push project

If this dialog appears:



Yes -> Install Certificate -> Yes

Refresher: Web MVC applications with template engine



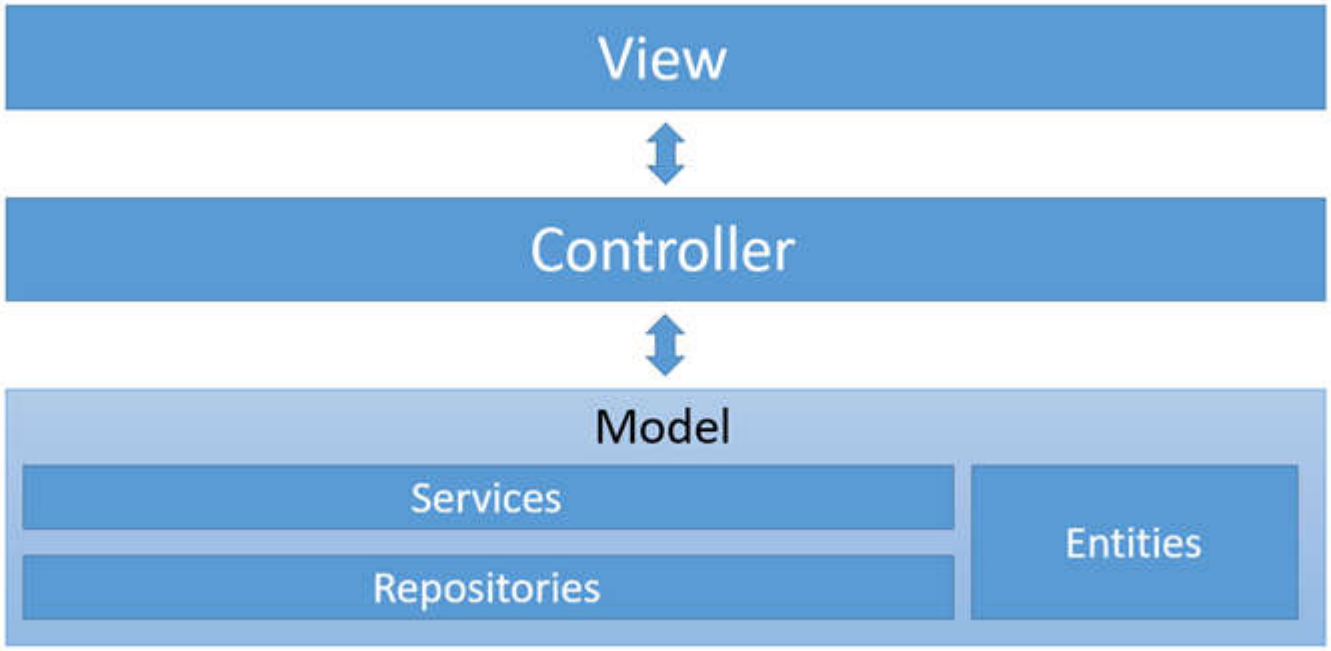
Responsibility of each MVC part:

- **Model**: domain entities structure and their transformations (domain model)
    - Entities
    - Services (related to entities)
        - Repositories (persistent data access)

- **Controllers**: receives user interactions and treat them

- **Views**: defines structure and behaviour of user interface

General architecture:



Project structure

Checklist:

- wwwroot: application resources (css, imagens, etc.)
- Controllers: application's MVC controllers
- Models: entities and "view models"
- Views: pages (notice naming conventions against controllers)
    - Shared: views used for more than one controller
- appsettings.json: external resources configuration (logging, connection strings, etc.)
- Program.cs: entry point
- Startup.cs: app configuration

First controller and Razor pages tests

Checklist:

- Route pattern: Controller / Ação / Id
  - Each controller method is mapped to an action
- Natural Templates
- C# block in Razor Page: @{ }
- ViewData dictionary
- Tag Helpers in Razor Pages. Examples: asp-controller and asp-action
- 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	-

First Model-Controller-View - Department

Checklist:

- Create new folder ViewModels e move ErrorViewModel (including namespace)
  - CTRL+SHIFT+B to fix references
- Create class Models/Department
- Create controller: right button Controllers -> Add -> Controller -> MVC Controller Empty
  - Name: DepartmentsController (**PLURAL**)
  - Instantiate a List<Department> and return it as View method parameter

- Create new folder Views/Departments (**PLURAL**)
- Create view: right button Views/Departments -> Add -> View
  - View name: Index
  - Template: List
  - Model class: Department
  - Change Title to Departments
  - Notice:
    - @model definition
    - intellisense for model
    - Helper methods
    - @foreach block

Deleting Department view and controller

Checklist:

- Delete controller
- Delete folder Views/Departments

CRUD Scaffolding

Checklist:

- Right button Controllers -> Add -> New Scaffolded Item
  - MVC controllers with views, using Entity Framework
  - Model class: Department
  - Data context class: + and accept the name
  - Views (options): all three
  - Controller name: DepartmentsController

MySQL adaptation and first migration

Note: we're using CODE-FIRST workflow

Checklist:

- Em appsettings.json, set connection string:
  - "server=localhost;userid=developer;password=1234567;database=saleswebmvcappdb"
- Em Startup.cs, fix DbContext definition for dependency injection system:

```
services.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

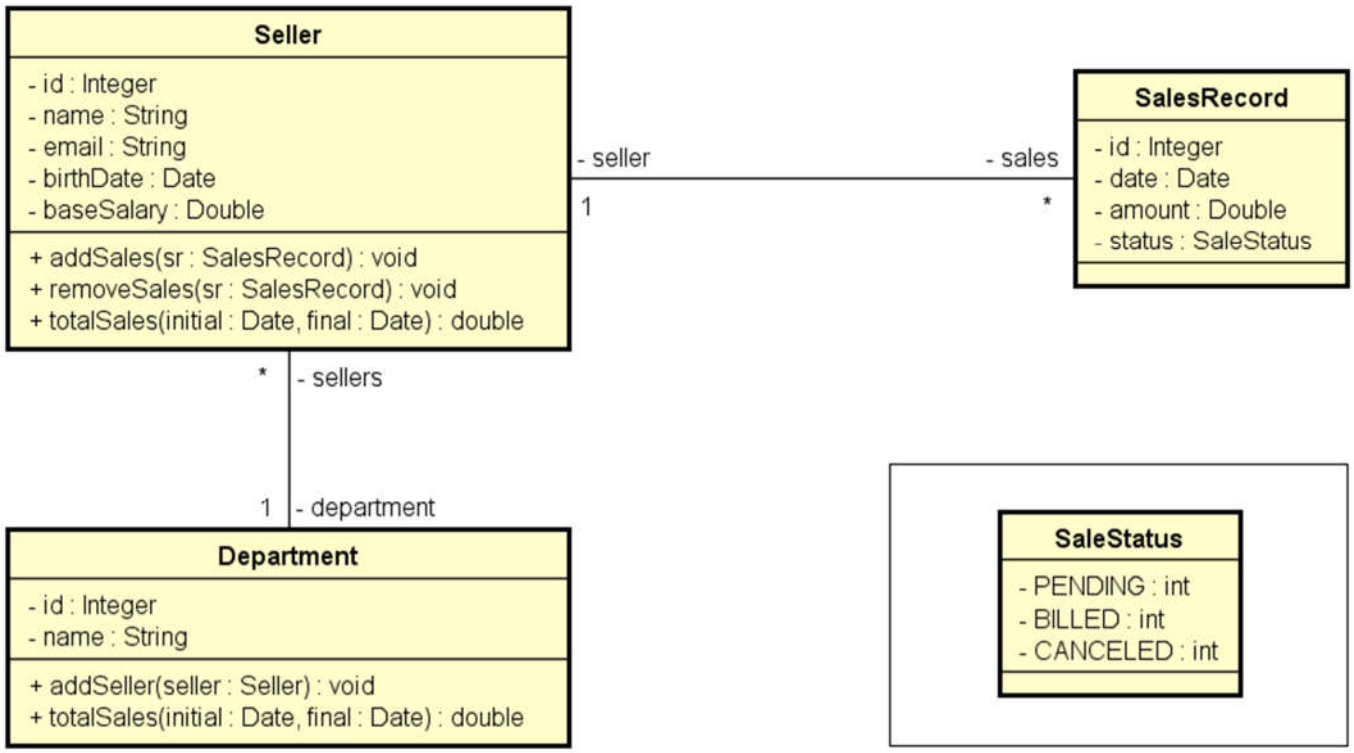
- Check database in MySQL Workbench
- Test app: CTRL+F5

Changing theme

Checklist:

- Go to: http://bootswatch.com/3 (check Bootstrap version)
- Choose a theme
- Download bootstrap.css
    - Suggestion: rename to bootstrap-name.css
    - Save file to wwwroot/lib/bootstrap/dist/css (paste it inside Visual Studio)
- Open _Layout.cshtml
    - Update bootstrap reference

Other entities and second migration



Checklist:

- Implement domain model
    - Basic attributes
    - Association (let's use **ICollection**, which matches List, HashSet, etc. - **INSTANTIATE!**)
    - Constructors (**default** and **with arguments**)
    - Custom methods
- Add DbSet's in DbContext
- Add-Migration OtherEntities
    - Update-Database

Seeding Service

Checklist:

- Stop IIS
- In Data, create SeedingService
- In Startup.cs, register SeedingService for dependency injection system
- In Startup.cs, add SeedingService as parameter of Configure method. Call Seed for development profile

SellersController

Checklist:

- Create Departments and Sellers links on navbar

- Controller -> Add -> Controller -> MVC Controller - Empty -> **SellersController**
- Create folder Views/Sellers
- Views/Sellers -> Add -> View
    - View name: Index
    - Change title

SellerService and basic FindAll

Checklist:

- Create folder Services
- Create SellerService
- In Startup.cs, register SellerService to dependency injection system
- In SellerService, implement FindAll, returning List<Seller>
- In SellersController, implement Index method, which should call SellerService.FindAll
- In Views/Sellers/Index, write template code to show Sellers

- Suggestion: user classes "table-striped table-hover" for table
- Note: we're going to apply formatting in later classes

Simple Create form

Checklist:

- In Views/Sellers/Index, create link to "Create"
- In controller, implement "Create" GET action
- In Views/Sellers, create "Create" view
- In Services/SellerService create Insert method
- In controller, implement "Create" POST action

Reference:

https://docs.microsoft.com/en-us/aspnet/core/security/anti-request-forgery

Foreign key not null (referential integrity)

Checklist:

- In Seller, add DepartmentId
- Drop database
- Create new migration, update database
- Update SellerService.Insert for now: `obj.Department = _context.Department.First();`

SellerFormViewModel and Department select component

Checklist:

- Create DepartmentService with FindAll method
- In Startup.cs, register DepartmentService to dependency injection system
- Create SellerFormViewModel
- In controller:
  - New dependency: DepartmentService
  - Update "Create" GET action
- In Views/Sellers/Create:
  - Update model type to SellerFormViewModel
  - Update form fields
  - Add select component for DepartmentId

```
<div class="form-group">
    <label asp-for="Seller.DepartmentId" class="control-label"></label>
    <select asp-for="Seller.DepartmentId" asp-items="@(new SelectList(Model.Departments,"Id",
"Name"))" class="form-control"></select>
</div>
```

- In controller, update "Create" POST action **-> NOT NECESSARY! :)**
- In SellerService.Insert, delete "First" call

Reference: https://stackoverflow.com/questions/34624034/select-tag-helper-in-asp-net-core-mvc

Delete seller

Checklist:

- In SellerService, create FindById and Remove operations
- In controller, create "Delete" GET action
- In View/Sellers/Index, check link to "Delete" action
- Create delete confirmation view: View/Sellers/Delete
- Test App
- In controller, create "Delete" POST action
- Test App

Seller details and eager loading

Checklist:

- https://docs.microsoft.com/en-us/ef/core/querying/related-data

Checklist:

- In View/Sellers/Index, check link to "Details" action
- In controller, create "Details" GET action
- Create view: View/Sellers/Details
- Include in FindAll: Include(obj => obj.Department) (namespace: Microsoft.EntityFrameworkCore)

Update seller and custom service exception

Checklist:

- Create Services/Exceptions folder
- Create NotFoundException and DbConcurrencyException
- In SellerService, create Update method
- In View/Sellers/Index, check link to "Edit" action
- In controller, create "Edit" GET action
- Create view: View/Sellers/Edit (similar do Create, plus hidden id)
- Test app
- In controller, create "Edit" POST action
- Test app

- **Notice:** ASP.NET Core selects option based on DepartmentId

Returning custom error page

Checklist:

- Update ErrorViewModel
- Update Error.cshtml
- In SellerController:
    - Create Error action with message parameter
    - Update method calls

App locale, number and date formatting

Checklist:

- In Startup.cs, define localization options
- In Seller:
    - Define custom labels [Display]
    - Define semantics for date [DataType]
    - Define display formats [DisplayFormat]

Validation

Checklist:

- In Seller, add validation annotations

```
[Required(ErrorMessage = "{0} required")]

[EmailAddress(ErrorMessage = "Enter a valid email")]

[Range(100.0, 50000.0, ErrorMessage = "{0} must be from {1} to {2}")]
```

- Update HTML for Create and Edit view

Summary:

```
<div asp-validation-summary="All" class="text-danger"></div>
```

Field:

```
<span asp-validation-for="Name" class="text-danger"></span>
```

Client-side validation:

```
@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

- Update SellersController

Asynchronous operations using Tasks (async, await)

Checklist:

- Update DepartmentService
- Update SellerService
- Update SellersController

Exception handling for delete (referential integrity)

Checklist:

- Create custom exception IntegrityException
- In SellerService.RemoveAsync, catch DbUpdateException and throw IntegrityException
- In SellersController, update Delete POST action

Preparing sales search navigation views

Checklist:

- Create SalesRecordsController with Index, SimpleSearch and GroupingSearch action
- Create folder Views/SalesRecords
- Create Index view with search forms
- Create "Sales" link on main navbar
- Create SimpleSearch and GroupingSearch views

Implementing simple search

Checklist:

- Create SalesRecordService with FindByDate operation
- In Startup.cs, register SalesRecordService to dependency injection system
- In SalesRecordsController, update SimpleSearch action
- Update SimpleSearch view
- Optional: format SalesRecord date and number

Implementing grouping search

Checklist:

- In SalesRecordService create FindByDateGrouping operation
- In SalesRecordsController, update GroupingSearch action
- Update GroupingSearch view