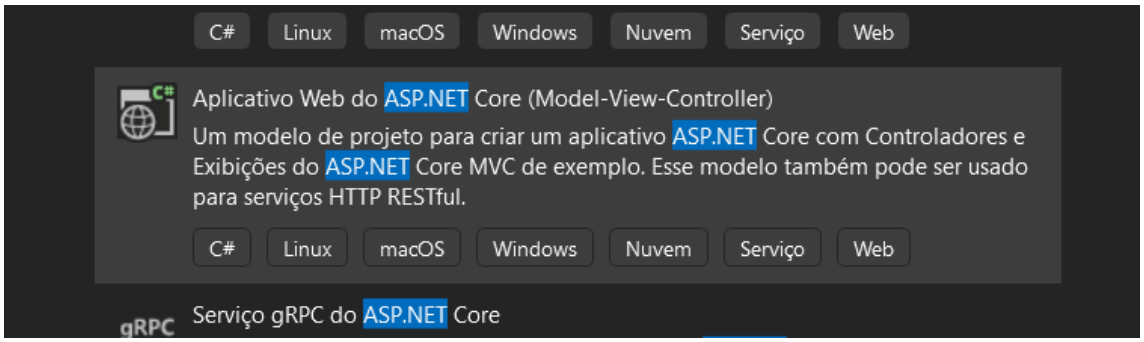
ASP.NET Core MVC

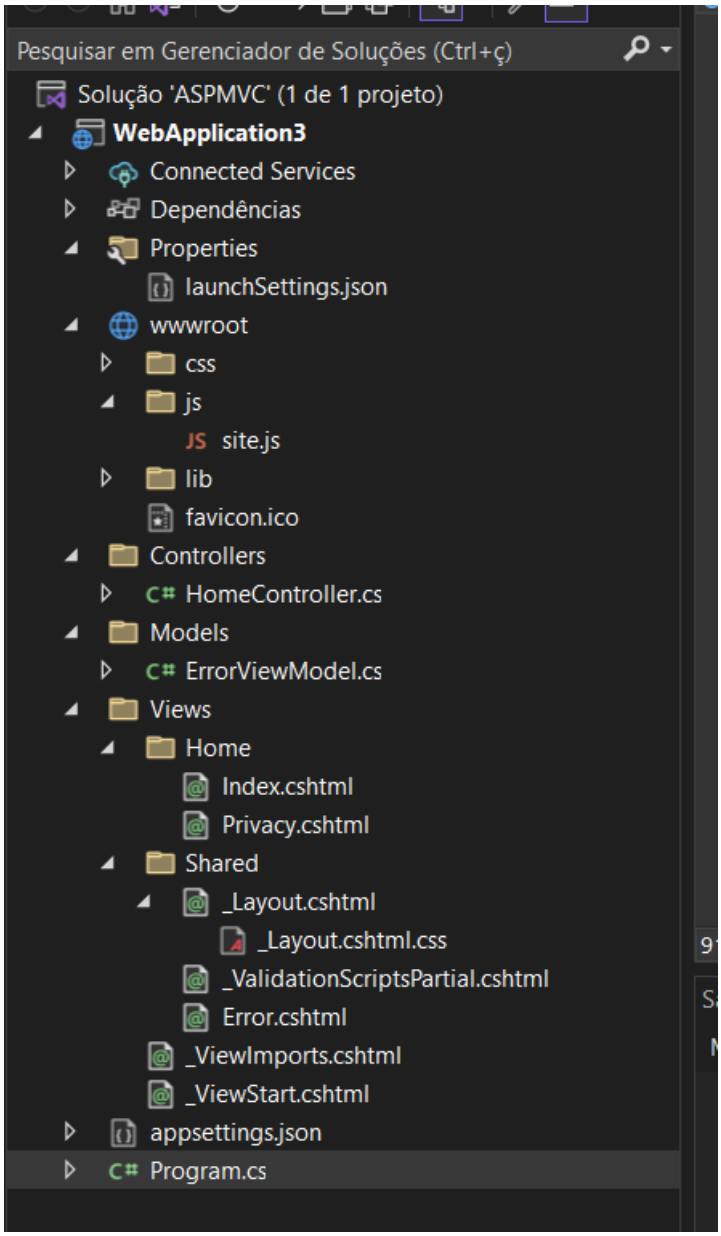
O ASP.NET Core é uma versão recente do ASP.NET, e o projeto ASP.NET Core MVC é um projeto ASP.NET básico que utiliza o padrão MVC.

Agora, descrevendo um pouco mais tecnicamente, o ASP.NET Core MVC é um novo framework de código aberto para a construção de aplicações conectadas pela internet. Ele permite o desenvolvimento e a execução de aplicações em Windows, Mac e Linux, e estas podem ser executadas no .NET Core ou no .NET Framework (versão anterior do .NET).

Como criar o projeto:

Após termos criado a solução em branco criamos um projeto ASP.NET Core MVC:





Entendendo a estrutura do projeto:

Analisando as pastas percebemos logo de cara a pasta dependências onde terá as dependências do projeto, em seguida temos a pasta **Properties** essa pasta ira conter as propriedades do projeto, basicamente ira conter as configurações Relacionadas ao IIS Express, em seguida temos a pasta **wwwroot** essa pasta contém os arquivos secundários ao View, como imagens, css, js e as bibliotecas do FrontEnd, em seguida temos as pastas principais da aplicação os MVC, em seguida temos o arquivo **appsetting.json** onde ira conter as configurações relacionadas ao funcionamento do projeto como a configuração do Entity Framework por exemplo, em seguida temos o arquivo **Program.cs**:

```
pplication3
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();
```

A classe Program.cs possui as configurações para rodar a aplicação, inicialmente o importante para analisarmos é o método **MapControllerRoute** que é responsável por mapear as rotas dos nossos controllers.

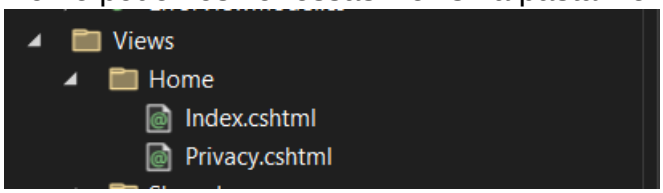
Basicamente como está agora ele está definindo que se não for especificado nenhum controller ele será o Home e caso não seja fornecido a action ela será Index.

Analisando os controllers:

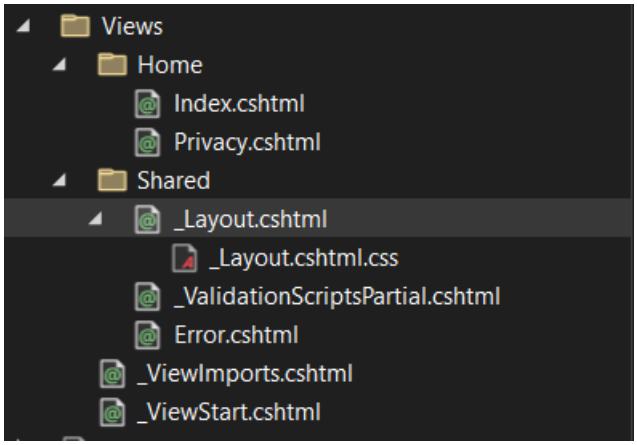
```
namespace WebApplication3.Controllers
{
    3 referências
    public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;

        0 referências
        public HomeController(ILogger<HomeController> logger)
        {
            _logger = logger;
        }

        0 referências
        public IActionResult Index()
        {
            return View();
        }

        0 referências
        public IActionResult Privacy()
        {
            return View();
        }

        [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
        0 referências
        public IActionResult Error()
```

Analisando a classe HomeController percebemos que todo Controller deve ter o seu nome seguido da palavra Controller e deve extender a classe Controller.

Analisando os métodos da classe percebemos que todos retornam um **IActionResult** e através disso retornam uma view, no caso dos métodos em Home podemos ver essas views na pasta view:



Entendendo a view:



Podemos analisar que todo controller terá a sua pasta em View com as suas páginas correspondentes, analisando percebemos que os arquivos com "_" serão compartilhados entre as Views, dessa forma temos o _Layout.cshtml que irá conter a estrutura html padrão, dessa forma não precisamos escrever toda a estrutura html em cada View, pois ela já é declarada em _Layout.cshtml e será compartilhada entre as Views.

Analisando a estrutura cshtml:

```
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - WebApplication3</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
    <link rel="stylesheet" href="~/WebApplication3.styles.css" asp-append-version="true" />
</head>
<body>
    <header>
        <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">
            <div class="container-fluid">
                <a class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">WebApplication3</a>
                <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target=".navbar-collapse" aria-controls="navbarSupported
                        aria-expanded="false" aria-label="Toggle navigation">
                    <span class="navbar-toggler-icon"></span>
                </button>
                <div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
                    <ul class="navbar-nav flex-grow-1">
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
                        </li>
                    </ul>
                </div>
            </div>
        </nav>
    </header>
    <div class="container">
```

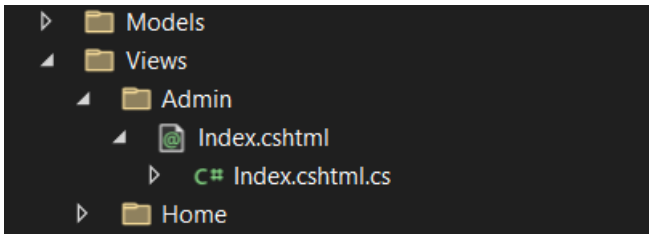
```
1   @{
2       ViewData["Title"] = "Privacy Policy";
3   }
4   <h1>@ViewData["Title"]</h1>
5
6   <p>Use this page to detail your site's privacy policy.</p>
```

Podemos perceber que a estrutura cshtml é semelhante ao que é o jsp para o java, temos código html com código c#, podemos analisar isso através da anotação “@” e alguns métodos como “asp-area”, “asp-controller” e “asp-action”.

Aplicando o contexto no controller:

Vamos aplicar a entidade que criamos em uma view e pegar os dados, para isso vamos primeiro criar o controller de admin:

```
using Microsoft.AspNetCore.Mvc;

namespace WebApplication3.Controllers
{
    0 referências
    public class AdminController : Controller
    {
        0 referências
        public IActionResult Index()
        {
            return View();
        }
    }
}
```



Agora vamos criar um layout de tabela para os dados:

```
@model IEnumerable<AdminModel>

<table class="table table-dark table-hover">
    <thead>
        <tr>
            <th scope="col">Id</th>
            <th scope="col">Nome</th>
            <th scope="col">Senha</th>
        </tr>
    </thead>
    <tbody>
        @if (Model != null)
        {
            @foreach(var admin in Model){
            <tr>
            <th scope="row">@admin.Id</th>
            <td>@admin.Nome</td>
            <td>@admin.Senha</td>
        </tr>
            }
        }
    </tbody>
</table>
```

Perceba que usamos um parâmetro IEnumerable<AdminModel> que deve ser passado pela classe controler:

```
namespace WebApplication3.Controllers
{
    1 referência
    public class AdminController : Controller
    {
        readonly private ApplicationDbContext _context;
        0 referências
        public AdminController(ApplicationDbContext context)
        {
            _context = context;
        }
        0 referências
        public IActionResult Index()
        {
            IEnumerable<AdminModel> admins = _context.Admins;
            return View(admins);
        }
    }
}
```

Perceba usamos o constructor para pegar o contexto e armazenar em uma variável, e através dessa variável pegamos os “Admins” e passamos para a view.

Entendendo a interação entre a view e o controller:

É importante entender que todos os métodos adicionados no controller serão rotas do sistema, sendo assim para mandarmos dados do view para o controller devemos apenas acessar essas rotas:

```
        public IActionResult Create(string nome, string senha)
{
    // código para criar um novo administrador
}
```

Pelo cshtml:

```
<form asp-controller="Admin" asp-action="Create">
    <div class="form-group">
        <label for="Nome">Nome</label>
```

```
        <input type="text" class="form-control" id="Nome"
name="Nome">
    </div>
    <div class="form-group">
        <label for="Senha">Senha</label>
        <input type="password" class="form-control" id="Senha"
name="Senha">
    </div>
    <button type="submit" class="btn btn-primary">Enviar</button>
</form>
```

Pelo JS:

Ajax:

```
  $(document).ready(function() {
        $("#sendButton").click(function() {
            $.ajax({
                url: "/Admin/Create",
                type: "POST",
                data: { nome: "João", senha: "123456" },
                success: function(result) {
                    // código para atualizar a página com os dados
retornados pelo servidor
                }
            });
        });
    });
```

Fetch:

```
fetch("/Admin/Create", {
            method: "POST",
            headers: {
                "Content-Type": "application/json"
            },
            body: JSON.stringify({ nome: "João", senha: "123456" })
        })
        .then(response => response.json())
        .then(data => {
            // código para atualizar a página com os dados
retornados pelo servidor
        });
```