UNIVEM DIGITAL

.NET Platform

Especialização Full Stack Development

Prof. Dr. João Ricardo Favan

17 de Setembro de 2022

Agenda

- Cronogramas das Proximas Aulas
- Clean Architecture
- Dados em aplicativos
- Entity Framework Core
- DBContext

UNIVEM DIGITAL

10/09/2022

Desenvolvimento de aplicações com Clean Architecture

Aplicações MVC
Aplicações WebAPI

17/09/2022

Persistência de Dados
Testes de unidade e integração

Soluções em dotnet

24/09/2022

Machine Learning em ambiente dotnet

Treinamento e utilização de modelos de Machine Learning

Objetivo

- Conhecer o modelo de arquitetura Clean Architecture para .NET.
- Conhecer o Framework de controle de dados Microsoft Entity Framwork Core.
- Desenvolver Solução .Net baseada em Clean Architecture.

Entity Framework Core

O EF Core é um O/RM (mapeador relacional de objeto) que permite aos desenvolvedores do .NET persistir objetos bidirecionalmente em uma fonte de dados. Elimina a necessidade da maioria do código de acesso a dados

Entity Framework Core

Para usar o EF Core com um banco de dados do SQL Server, execute o seguinte comando da CLI do dotnet:

CLI do .NET — Copiar

```
dotnet add package Microsoft.EntityFrameworkCore.SqlServer
```

Para adicionar suporte para uma fonte de dados InMemory para teste:

CLI do .NET — Copiar

```
dotnet add package Microsoft.EntityFrameworkCore.InMemory
```

DbContext

Essa classe contém propriedades que representam coleções de entidades com as quais seu aplicativo trabalhará

UNIVEM DIGITAL

DbContext

```
public class CatalogContext : DbContext
{
  public CatalogContext(DbContextOptions<CatalogContext> options) : base(options)
  {

  }

  public DbSet<CatalogItem> CatalogItems { get; set; }
  public DbSet<CatalogBrand> CatalogBrands { get; set; }
  public DbSet<CatalogType> CatalogTypes { get; set; }
}
```

Configurando o DbContext

```
services.AddDbContext<CatalogContext>(options =>
      options.UseSqlServer
(Configuration.GetConnectionString("DefaultConnection")
));
```

```
services.AddDbContext<CatalogContext>(options =>
    options.UseInMemoryDatabase());
```

Buscando e Armazenando Dados

Para recuperar dados do EF Core, acesse a propriedade apropriada e use o LINQ para filtrar o resultado. Também use o LINQ para executar a projeção, transformando o resultado de um tipo para outro

Buscando e Armazenando Dados

```
var brandItems = await _context.CatalogBrands
    .Where(b => b.Enabled)
    .OrderBy(b => b.Name)
    .Select(b => new SelectListItem {
        Value = b.Id, Text = b.Name })
    .ToListAsync();
```

Buscando e Armazenando Dados

O EF Core controla as alterações nas entidades que ele busca da persistência. Para salvar as alterações em uma entidade controlada, basta chamar o método SaveChanges no DbContext, verificando se ela é a mesma instância de DbContext que foi usada para buscar a entidade

UNIVEM DIGITAL

Buscando e Armazenando Dados

```
// create
var newBrand = new CatalogBrand() { Brand = "Acme" };
_context.Add(newBrand);
await _context.SaveChangesAsync();

// read and update
var existingBrand = _context.CatalogBrands.Find(1);
existingBrand.Brand = "Updated Brand";
await _context.SaveChangesAsync();

// read and delete (alternate Find syntax)
var brandToDelete = _context.Find<CatalogBrand>(2);
_context.CatalogBrands.Remove(brandToDelete);
await _context.SaveChangesAsync();
```

Buscando dados relacionados

Quando o EF Core recupera entidades, ele popula todas as propriedades armazenadas diretamente nessa entidade no banco de dados. Propriedades de navegação, como listas de entidades relacionadas, não são populadas e podem ter seu valor definido como nulo. Esse processo garante que o EF Core não busca mais dados do que necessário, o que é especialmente importante para aplicativos Web, que devem processar solicitações e retornar respostas de uma maneira eficiente e com agilidade. Para incluir relações com uma entidade usando o carregamento adiantado, especifique a propriedade usando o método de extensão Include na consulta, conforme mostrado:

UNIVEM DIGITAL

Buscando dados relacionados

```
// .Include requires using Microsoft.EntityFrameworkCore
var brandsWithItems = await _context.CatalogBrands
    .Include(b => b.Items)
    .ToListAsync();
```

```
// Includes all expression-based includes
query = specification.Includes.Aggregate(query,
            (current, include) => current.Include(include));

// Include any string-based include statements
query = specification.IncludeStrings.Aggregate(query,
            (current, include) => current.Include(include));
```

Encapsulando Dados

O EF Core é compatível com vários recursos que permitem que o modelo encapsule seu próprio estado corretamente,e com isso, pode-se expor o acesso somente leitura às coleções relacionadas e fornecer métodos explicitamente definindo de que forma os clientes podem manipulá-las:

```
public class Basket : BaseEntity
{
  public string BuyerId { get; set; }
  private readonly List<BasketItem> _items = new List<BasketItem>();
  public IReadOnlyCollection<BasketItem> Items => _items.AsReadOnly();

  public void AddItem(int catalogItemId, decimal unitPrice, int quantity = 1)
  {
    if (!Items.Any(i => i.CatalogItemId == catalogItemId))
    {
      _items.Add(new BasketItem()
      {
        CatalogItemId = catalogItemId,
        Quantity = quantity,
        UnitPrice = unitPrice
      });
      return;
    }
    var existingItem = Items.FirstOrDefault(i => i.CatalogItemId == catalogItemId
    existingItem.Quantity += quantity;
  }
}
```

Conexões Resilientes

Recursos externos (bancos de dados SQL) podem não estar disponíveis, nesses os aplicativos podem usar a lógica de repetição para evitar gerar uma exceção

Para o BD SQL do Azure, o Entity Framework Core já fornece resiliência interna de conexão de banco de dados e lógica de repetição

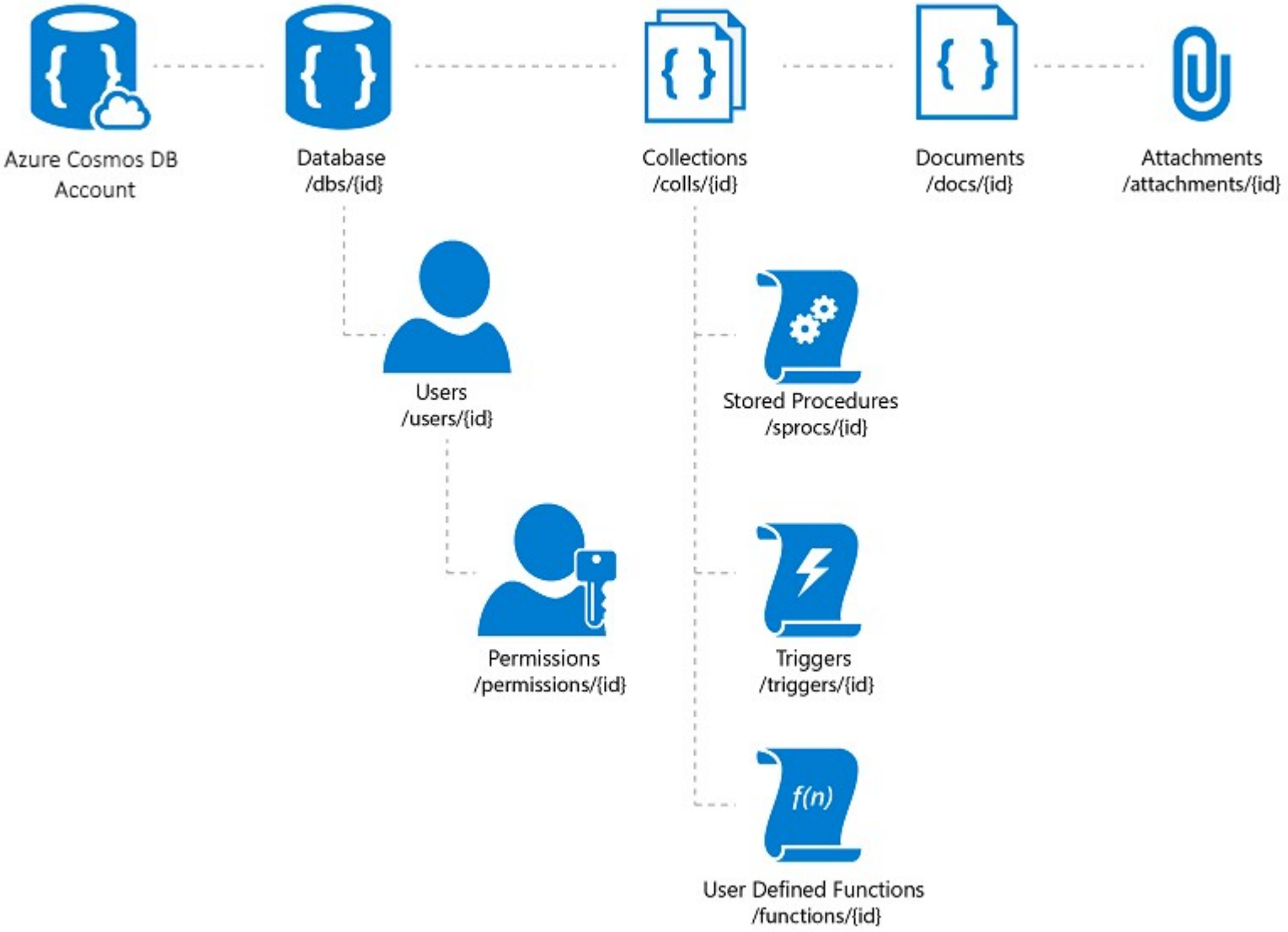
Conexões Resilientes

```
builder.Services.AddDbContext<OrderingContext>(options =>
{
    options.UseSqlServer(builder.Configuration["ConnectionString"],
        sqlServerOptionsAction: sqlOptions =>
        {
            sqlOptions.EnableRetryOnFailure(
            maxRetryCount: 5,
            maxRetryDelay: TimeSpan.FromSeconds(30),
            errorNumbersToAdd: null);
        }
    );
});
```

Azure Cosmos DB

O Azure Cosmos DB é um serviço de banco de dados NoSQL totalmente gerenciado que oferece armazenamento de dados sem esquemas baseado em nuvem

Apesar de ser um banco de dados NoSQL, os desenvolvedores podem usar funcionalidades avançadas e conhecidas de consultas SQL em dados JSON


Azure Cosmos DB Account
Database /dbs/{id}
Collections /colls/{id}
Documents /docs/{id}
Attachments /attachments/{id}
Users /users/{id}
Permissions /permissions/{id}
Stored Procedures /sprocs/{id}
Triggers /triggers/{id}
f(n)
User Defined Functions /functions/{id}

Obtendo dados para aplicativos BlazorWebAssembly

Para aplicativos que usam o Servidor Blazor, pode-se usar o Entity Framework e outras tecnologias de acesso direto a dados, no entanto, para aplivativos BlazorWebAssembly (SPA) será necessário acessar dados por meio de endpoints de Web API.

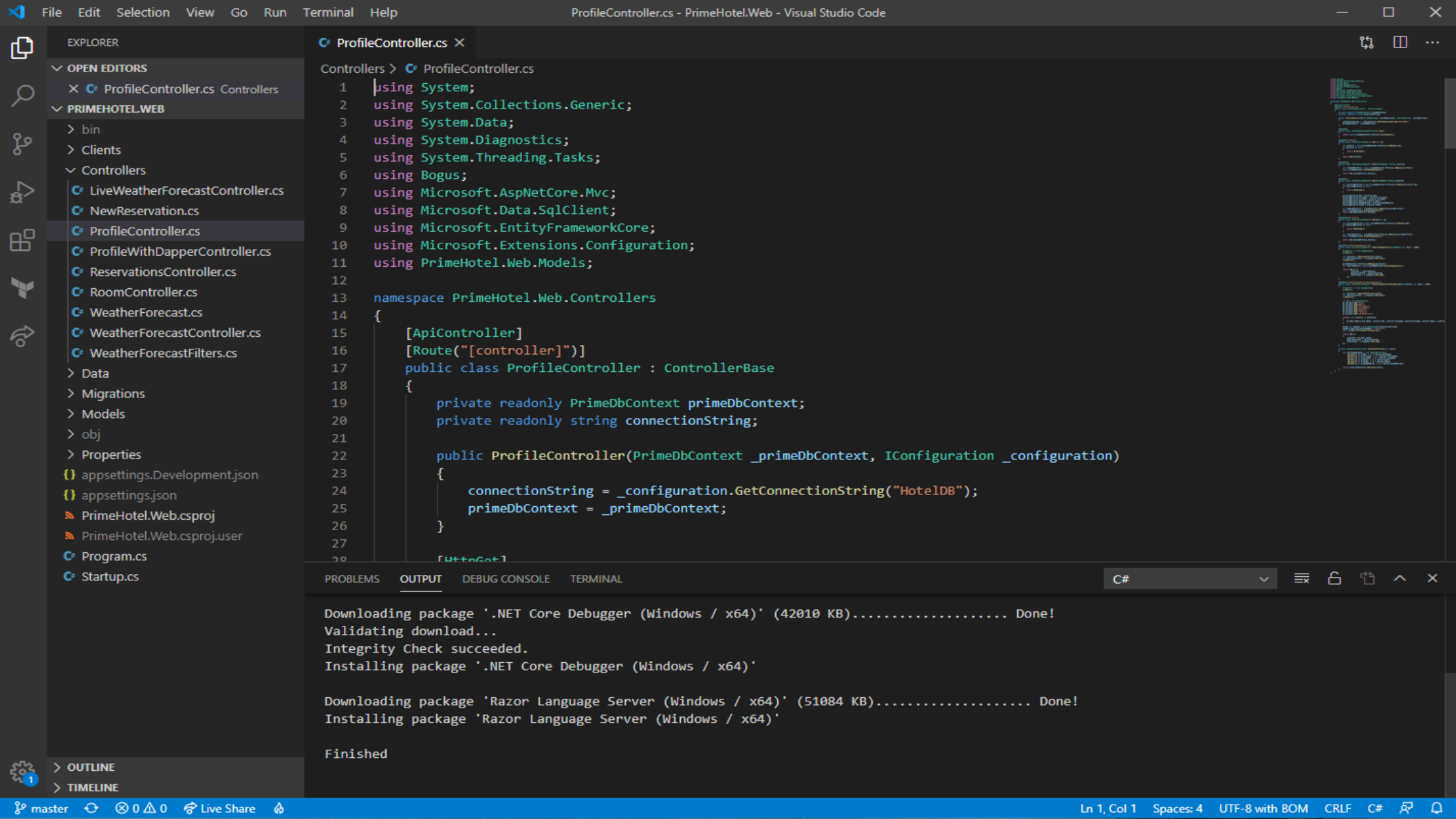
UNIVEM DIGITAL

Obtendo dados para APIs

```
builder.Services.AddScoped(sp => new HttpClient
  {
    BaseAddress = new Uri(builder.HostEnvironment.BaseAddress)
  });
```

```
_httpClient.DefaultRequestHeaders.Authorization =
  new AuthenticationHeaderValue("Bearer", token);
```

```
var item = await _httpClient.GetFromJsonAsync<CatalogItem>($"catalog-items/{id}");
```


File Edit Selection View Go Run Terminal Help
ProfileController.cs - PrimeHotel.Web - Visual Studio Code
EXPLORER
OPEN EDITORS
ProfileController.cs Controllers
PRIMEHOTEL.WEB
bin
Clients
Controllers
LiveWeatherForecastController.cs
NewReservation.cs
ProfileController.cs
ProfileWithDapperController.cs
ReservationsController.cs
RoomController.cs
WeatherForecast.cs
WeatherForecastController.cs
WeatherForecastFilters.cs
Data
Migrations
Models
obj
Properties
appsettings.Development.json
appsettings.json
PrimeHotel.Web.csproj
PrimeHotel.Web.csproj.user
Program.cs
Startup.cs
OUTLINE
TIMELINE
ProfileController.cs
Controllers > ProfileController.cs
1 using System;
2 using System.Collections.Generic;
3 using System.Data;
4 using System.Diagnostics;
5 using System.Threading.Tasks;
6 using Bogus;
7 using Microsoft.AspNetCore.Mvc;
8 using Microsoft.Data.SqlClient;
9 using Microsoft.EntityFrameworkCore;
10 using Microsoft.Extensions.Configuration;
11 using PrimeHotel.Web.Models;
12
13 namespace PrimeHotel.Web.Controllers
14 {
15 [ApiController]
16 [Route("[controller]")]
17 public class ProfileController : ControllerBase
18 {
19 private readonly PrimeDbContext primeDbContext;
20 private readonly string connectionString;
21
22 public ProfileController(PrimeDbContext _primeDbContext, IConfiguration _configuration)
23 {
24 connectionString = _configuration.GetConnectionString("HotelDB");
25 primeDbContext = _primeDbContext;
26 }
27
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
C#
Downloading package '.NET Core Debugger (Windows / x64)' (42010 KB)..... Done!
Validating download...
Integrity Check succeeded.
Installing package '.NET Core Debugger (Windows / x64)'
Downloading package 'Razor Language Server (Windows / x64)' (51084 KB)..... Done!
Installing package 'Razor Language Server (Windows / x64)'
Finished
master
0 0
Live Share
Ln 1, Col 1
Spaces: 4
UTF-8 with BOM
CRLF
C#