**Softversko inzenjerstvo**

**Primer sa pripreme sa knjigama:**

**DATALAYER**

**DataLayer – sql.txt**

CREATE TABLE [dbo].Books

(

[Id] INT NOT NULL PRIMARY KEY identity(1,1),

Title nvarchar(50) NOT NULL,

Description nvarchar(50) NOT NULL,

NumberOfPages int NULL

)

**DataLayer – Model – Book**

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataLayer.Model

{

public class Book

{

[Key]

public int Id { get; set; }

[Required(ErrorMessage = "Naziv knjige je obavezan.")]

public string? Title { get; set; }

[Required(ErrorMessage = "Opis knjige je obavezan.")]

public string? Description { get; set; }

public int NumberOfPage { get; set; }

}

}

**DataLayer – IbookRepository**

using DataLayer.Model;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataLayer

{

public interface IBookRepository

{

bool InsertBook(Book book);

List<Book> GetAllBooks();

} }

**DataLayer – BookRepository**

using DataLayer.Model;

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataLayer

{

public class BookRepository : IBookRepository

{

private const string ConnectionString = "Data Source=(localdb)\\ProjectModels;Initial Catalog=\"LibraryDB \";";

[Obsolete]

public List<Book> GetAllBooks()

{

List<Book> list = new List<Book>();

using(SqlConnection sqlConnection = new SqlConnection(ConnectionString))

{

sqlConnection.Open();

SqlCommand sqlCommand = sqlConnection.CreateCommand();

sqlCommand.CommandText = "SELECT \* FROM Books";

SqlDataReader reader = sqlCommand.ExecuteReader();

while (reader.Read())

{

Book book = new Book();

book.Id = reader.GetInt32(0);

book.Title = reader.GetString(1);

book.Description = reader.GetString(2);

book.NumberOfPage = reader.GetInt32(3);

list.Add(book);

}

}

return list;

}

[Obsolete]

public bool InsertBook(Book book)

{

using (SqlConnection sqlConnection = new SqlConnection(ConnectionString))

{

sqlConnection.Open();

SqlCommand sqlCommand = sqlConnection.CreateCommand();

sqlCommand.CommandText = "INSERT INTO Books(Title,Description,NumberOfPages) VALUES(@Title,@Description,@NumberOfPages)";

sqlCommand.Parameters.AddWithValue("@Title", book.Title);

sqlCommand.Parameters.AddWithValue("@Description", book.Description);

sqlCommand.Parameters.AddWithValue("@NumberOfPages", book.NumberOfPage);

int result = sqlCommand.ExecuteNonQuery();

return result > 0;

}

} } }

**BUSINESS LAYER**

**BusinessLayer – IBusinessBook**

using DataLayer.Model;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BusinessLayer

{

public interface IBusinessBook

{

string InsertBook(Book book);

List<Book> GetBooksWith50();

}

}

**BusinessLayer - BusinessBokk**

using DataLayer;

using DataLayer.Model;

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BusinessLayer

{

public class BusinessBook : IBusinessBook

{

private readonly IBookRepository bookRepository;

public BusinessBook(IBookRepository bookRepository)

{

this.bookRepository = bookRepository;

}

public List<Book> GetBooksWith50()

{

return bookRepository.GetAllBooks()

.FindAll(item => item.NumberOfPage > 50);

}

public string InsertBook(Book book)

{

var validationResults = new List<ValidationResult>();

var validationContext = new ValidationContext(book, null, null);

if (!Validator.TryValidateObject(book, validationContext, validationResults, true))

{

return string.Join(", ", validationResults.Select(vr => vr.ErrorMessage));

}

if (book != null)

{

if (bookRepository.InsertBook(book))

{

return "Uspesno!";

}

return "Greska!";

}

return "Greska!";

} } }

**WEB:**

**Web-Counter:**@page "/books"

@rendermode InteractiveServer

@using BusinessLayer;

@using DataLayer.Model

@inject NavigationManager NV

@inject IBusinessBook bus

<**PageTitle**>Books</**PageTitle**>

<h1>Books</h1>

<button type="button" @onclick="Add" class="btn btn-primary" >Add Book</button>

<table class="table">

<thead>

<tr>

<th scope="col">ID</th>

<th scope="col">Title</th>

<th scope="col">Description</th>

<th scope="col">Number Of Page</th>

</tr>

</thead>

<tbody>

@if (Books != null)

{

@foreach (var item in Books)

{

<tr>

<td>@item.Id</td>

<td>@item.Title</td>

<td>@item.Description</td>

<td>@item.NumberOfPage</td>

</tr>

}

}

</tbody>

</table>

@code {

private List<Book>? Books{ get; set; }

protected override void OnInitialized()

{

Books = bus.GetBooksWith50();

}

private void Add()

{

NV.NavigateTo("/addBook");

}

}

**WEB – AddBook**

@page "/addBook"

@rendermode InteractiveServer

@using BusinessLayer;

@using DataLayer.Model

@inject NavigationManager NV

@inject IBusinessBook bus

<h3>AddBook</h3>

@if (book != null)

{

<form>

<div class="mb-3">

<label for="exampleInput1" class="form-label">Title</label>

<input @bind="book.Title" type="text" class="form-control" id="exampleInput1">

</div>

<div class="mb-3">

<label for="exampleInput2" class="form-label">Title</label>

<input @bind="book.Description" type="text" class="form-control" id="exampleInput2">

</div>

<div class="mb-3">

<label for="exampleInput3" class="form-label">Title</label>

<input @bind="book.NumberOfPage" type="number" class="form-control" id="exampleInput3">

</div>

<button type="button" @onclick="Save" class="btn btn-primary">Submit</button>

<p class="text-danger">@message</p>

</form>

}

@code {

private Book? book;

private string? message;

protected override void OnInitialized()

{

book = new Book();

}

private void Save()

{

if (book != null)

{

string res = bus.InsertBook(book);

if (res == "Uspesno!")

{

NV.NavigateTo("/books");

}

else

message = res;

}

}

}

**Web - NavMenu:**

<div class="top-row ps-3 navbar navbar-dark">

<div class="container-fluid">

<a class="navbar-brand" href="">Web</a>

</div>

</div>

<input type="checkbox" title="Navigation menu" class="navbar-toggler" />

<div class="nav-scrollable" onclick="document.querySelector('.navbar-toggler').click()">

<nav class="flex-column">

<div class="nav-item px-3">

<**NavLink** class="nav-link" href="" **Match**="NavLinkMatch.All">

<span class="bi bi-house-door-fill-nav-menu" aria-hidden="true"></span> Home

</**NavLink**>

</div>

<div class="nav-item px-3">

<**NavLink** class="nav-link" href="books">

<span class="bi bi-plus-square-fill-nav-menu" aria-hidden="true"></span> Books

</**NavLink**>

</div>

<div class="nav-item px-3">

<**NavLink** class="nav-link" href="weather">

<span class="bi bi-list-nested-nav-menu" aria-hidden="true"></span> Weather

</**NavLink**>

</div>

</nav>

</div>

**Program.cs**

using BusinessLayer;

using DataLayer;

using Web.Components;

namespace Web

{

public class Program

{

public static void Main(string[] args)

{

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddRazorComponents()

.AddInteractiveServerComponents();

builder.Services.AddScoped<IBookRepository, BookRepository>();

builder.Services.AddScoped<IBusinessBook, BusinessBook>();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/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.UseAntiforgery();

app.MapRazorComponents<App>()

.AddInteractiveServerRenderMode();

app.Run();

}}}

**Primer sa pripreme sa automobilima:**

**DataLayer**

**DataLayer – sql.txt**

CREATE TABLE [dbo].Cars

(

[Id] INT NOT NULL PRIMARY KEY identity(1,1),

Title nvarchar(200) NOT NULL,

Year int NULL,

Price decimal(18,2) NOT NULL

)

**DataLayer – Model – Car**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataLayer.Model

{

public class Car

{

public int Id { get; set; }

public string? Title { get; set; }

public int Year { get; set; }

public decimal Price { get; set; } } }

**DataLayer – IcarRepository**

using DataLayer.Model;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataLayer

{

public interface ICarRepository

{

List<Car> GetCars();

bool Delete(Car car); }}

**DataLayer – CarRepository**

using DataLayer.Model;

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Linq;

using System.Runtime.ConstrainedExecution;

using System.Text;

using System.Threading.Tasks;

using System.Xml.Linq;

namespace DataLayer

{

public class CarRepository : ICarRepository

{

private const string ConnectionString = "Data Source=(localdb)\\ProjectModels;Initial Catalog=CarDB; ";

public bool Delete(Car car)

{

using(SqlConnection sqlConnection=new SqlConnection(ConnectionString))

{

sqlConnection.Open();

SqlCommand sqlCommand = sqlConnection.CreateCommand();

sqlCommand.CommandText = "DELETE FROM Cars WHERE Id=@Id";

sqlCommand.Parameters.AddWithValue("@Id", car.Id);

return sqlCommand.ExecuteNonQuery() > 0;

}

}

public List<Car> GetCars()

{

List<Car> list = new List<Car>();

using (SqlConnection sqlConnection = new SqlConnection(ConnectionString))

{

sqlConnection.Open();

SqlCommand sqlCommand = sqlConnection.CreateCommand();

sqlCommand.CommandText = "SELECT \* FROM Cars";

SqlDataReader reader = sqlCommand.ExecuteReader();

while (reader.Read())

{

Car car = new Car();

car.Id = reader.GetInt32(0);

car.Title = reader.GetString(1);

car.Year = reader.GetInt32(2);

car.Price = reader.GetDecimal(3);

list.Add(car);

} }

return list; } } }

**BUSINESSLAYER**

**BusinessLayer – IBusinessCar**

using DataLayer.Model;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.ConstrainedExecution;

using System.Text;

using System.Threading.Tasks;

namespace BusinessLayer

{

public interface IBusinessCar

{

List<Car> GetCars();

string Delete(Car car);

}

}

**BusinessLayer – BusinessCar**

using DataLayer;

using DataLayer.Model;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BusinessLayer

{

public class BusinessCar : IBusinessCar

{

private readonly ICarRepository carRepository;

public BusinessCar(ICarRepository carRepository)

{

this.carRepository = carRepository;

}

public string Delete(Car car)

{

if (carRepository.Delete(car))

{

return "Automobil obrisan";

}

return "Greska";

}

public List<Car> GetCars()

{

var x = from item in carRepository.GetCars()

orderby item.Price descending select item;

return x.ToList();

} }}

**WEB**

**Web – Counter**

@page "/"

@using BusinessLayer

@rendermode InteractiveServer

@using DataLayer.Model;

@inject IBusinessCar bus

<**PageTitle**>Counter</**PageTitle**>

<h1>Cars</h1>

<div class="container">

<div class="row">

@if (Lista != null)

{

@foreach(var item in Lista)

{

<div class="col-md-3">

<div class="card">

<img src="./car.jpg" class="card-img-top" alt="...">

<div class="card-body">

<h5 class="card-title">@item.Title</h5>

<p class="card-text">Godine: @item.Year</p>

<p class="card-text">Cena: @item.Price</p>

<button type="button" @onclick="()=>Delete(item.Id)" class="btn btn-primary">Delete</button>

</div>

</div>

</div>

}

}

</div>

</div>

@code {

public List<Car>?Lista{ get; set; }

protected override void OnInitialized()

{

Lista = bus.GetCars();

}

private void Delete(int id)

{

Car car = bus.GetCars().Find(x => x.Id == id)!;

if (bus.Delete(car) == "Automobil obrisan")

{

OnInitialized();

} }}

**WEB – NavMenu**

<div class="top-row ps-3 navbar navbar-dark">

<div class="container-fluid">

<a class="navbar-brand" href="">Web</a>

</div>

</div>

<input type="checkbox" title="Navigation menu" class="navbar-toggler" />

<div class="nav-scrollable" onclick="document.querySelector('.navbar-toggler').click()">

<nav class="flex-column">

<div class="nav-item px-3">

<**NavLink** class="nav-link" href="" **Match**="NavLinkMatch.All">

<span class="bi bi-house-door-fill-nav-menu" aria-hidden="true"></span> Home

</**NavLink**>

</div>

<div class="nav-item px-3">

<**NavLink** class="nav-link" href="counter">

<span class="bi bi-plus-square-fill-nav-menu" aria-hidden="true"></span> Counter

</**NavLink**>

</div>

<div class="nav-item px-3">

<**NavLink** class="nav-link" href="weather">

<span class="bi bi-list-nested-nav-menu" aria-hidden="true"></span> Weather

</**NavLink**>

</div>

</nav>

</div>

**Program.cs**using BusinessLayer;

using DataLayer;

using Web.Components;

namespace Web

{

public class Program

{

public static void Main(string[] args)

{

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddRazorComponents()

.AddInteractiveServerComponents();

builder.Services.AddScoped<ICarRepository, CarRepository>();

builder.Services.AddScoped<IBusinessCar, BusinessCar>();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/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.UseAntiforgery();

app.MapRazorComponents<App>()

.AddInteractiveServerRenderMode();

app.Run();

}

}

}

**NAS PROJEKAT**

**KorisnikRepository- IKorisnikRepository**

using Core.Interface;

using Entities;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataAccessLayer

{

public interface IKorisnikRepository:IRepository<Korisnik>

{

Task<List<Korisnik>> GetAllAsync();

Task<bool> UpdateAsync(Korisnik item);

Task<bool> DeleteAsync(Korisnik item);

}

}

**KorisnikRepository- metoda Update ali izbaciti async**

public async Task<bool> UpdateAsync(Korisnik item)

{

using (SqlConnection sqlConnection = new SqlConnection())

{

sqlConnection.ConnectionString = ConnectionBase.ConnectionString;

await sqlConnection.OpenAsync();

SqlCommand sqlCommand = sqlConnection.CreateCommand();

sqlCommand.CommandText = "UPDATE KORISNIK SET jmbg=@jmbg, korisnickoIme=@korisnickoIme, lozinka=@lozinka, ime=@ime, prezime=@prezime, tipKorisnika=@tipKorisnika, kontakt\_telefon=@kontakt\_telefon WHERE idKorisnika=@idKorisnika";

sqlCommand.Parameters.AddWithValue("@jmbg", item.JMBG);

sqlCommand.Parameters.AddWithValue("@korisnickoIme", item.KorisnickoIme);

sqlCommand.Parameters.AddWithValue("@lozinka", item.Lozinka);

sqlCommand.Parameters.AddWithValue("@ime", item.Ime);

sqlCommand.Parameters.AddWithValue("@prezime", item.Prezime);

sqlCommand.Parameters.AddWithValue("@tipKorisnika", item.TipKorisnika);

sqlCommand.Parameters.AddWithValue("@kontakt\_telefon", item.KontaktTelefon);

sqlCommand.Parameters.AddWithValue("@idKorisnika", item.IdKorisnika);

int res = await sqlCommand.ExecuteNonQueryAsync();

return res > 0;

}

}

}

**IUserBusiness- metoda Update ali izbaciti async**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Core.result;

using Entities;

namespace BusinessLayer.Abstract

{

public interface IUserBusiness

{

ResultWrapper RegisterUser(Korisnik korisnik);

ResultWrapper LoginUser(string korisnickoIme, string lozinka, string tip);

List<Korisnik> GetAllUsers();

Korisnik GetUserById(int id);

Korisnik GetUserByUsername(string username);

Task<List<Korisnik>> GetAllUsersAsync();

Task<bool> UpdateUserAsync(Korisnik koirisnik);

Task<bool> DeleteAsync(Korisnik korisnik);

}

}

**UserBusiness – izbaciti await**

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Text;

using System.Text.RegularExpressions;

using System.Threading.Tasks;

using BusinessLayer.Abstract;

using Core.Interface;

using Core.result;

using DataAccessLayer;

using Entities;

namespace BusinessLayer.Implementacija

{

public class UserBusiness : IUserBusiness

{

private readonly IKorisnikRepository userRepository;

public UserBusiness(IKorisnikRepository userRepository)

{

this.userRepository = userRepository;

}

public ResultWrapper RegisterUser(Korisnik user)

{

if (user == null)

{

return new ResultWrapper { Success = false, Message = "Greška: Nema podataka o korisniku." };

}

var validationResults = new List<ValidationResult>();

var validationContext = new ValidationContext(user, null, null);

if (!Validator.TryValidateObject(user, validationContext, validationResults, true))

{

return new ResultWrapper

{

Message = string.Join(", ", validationResults.Select(vr => vr.ErrorMessage)),

Success = false

};

}

var emailRegex = new Regex(@"^[^@\s]+@[^@\s]+**\.**[^@\s]+$");

if (!emailRegex.IsMatch(user.KorisnickoIme))

{

return new ResultWrapper { Success = false, Message = "Neispravan format email adrese." };

}

var passwordRegex = new Regex(@"^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\d).{6,}$");

if (!passwordRegex.IsMatch(user.Lozinka))

{

return new ResultWrapper { Success = false, Message = "Lozinka mora imati najmanje 6 karaktera, jedno veliko i jedno malo slovo i broj." };

}

var existingUser = userRepository.GetAll().FirstOrDefault(u => u.KorisnickoIme == user.KorisnickoIme);

if (existingUser != null)

{

return new ResultWrapper { Success = false, Message = "Korisničko ime već postoji." };

}

userRepository.Add(user);

return new ResultWrapper { Success = true, Message = "Uspešna registracija!" };

}

public ResultWrapper LoginUser(string korisnickoIme, string lozinka, string tipKorisnika)

{

if (string.IsNullOrWhiteSpace(korisnickoIme) || string.IsNullOrWhiteSpace(lozinka))

{

return new ResultWrapper { Success = false, Message = "Korisničko ime i lozinka su obavezni." };

}

var user = userRepository.GetAll().FirstOrDefault(u =>

u.KorisnickoIme == korisnickoIme &&

u.Lozinka == lozinka &&

u.TipKorisnika == tipKorisnika);

if (user == null)

{

return new ResultWrapper { Success = false, Message = "Neispravni podaci za prijavu." };

}

string token = Jwt.GenerateToken(user.KorisnickoIme, user.TipKorisnika);

return new ResultWrapper { Success = true, Message = "Prijava uspešna.", Data = token };

}

public List<Korisnik> GetAllUsers()

{

return userRepository.GetAll();

}

public Korisnik GetUserById(int id)

{

return userRepository.GetAll().FirstOrDefault(u => u.IdKorisnika == id);

}

public Korisnik GetUserByUsername(string username)

{

return userRepository.GetAll().FirstOrDefault(u => u.KorisnickoIme == username);

}

public async Task<List<Korisnik>> GetAllUsersAsync()

{

return await userRepository.GetAllAsync();

}

public async Task<bool> UpdateUserAsync(Korisnik korisnik)

{

return await userRepository.UpdateAsync(korisnik);

}

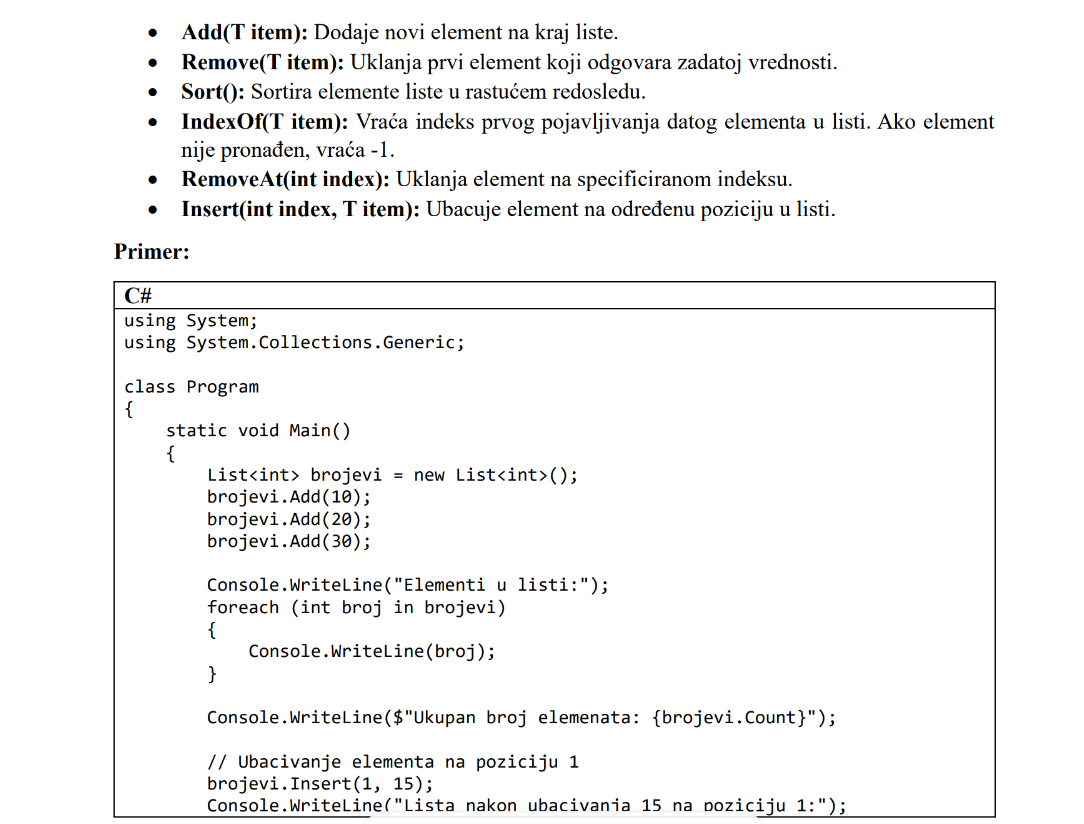
public async Task<bool> DeleteAsync(Korisnik korisnik)

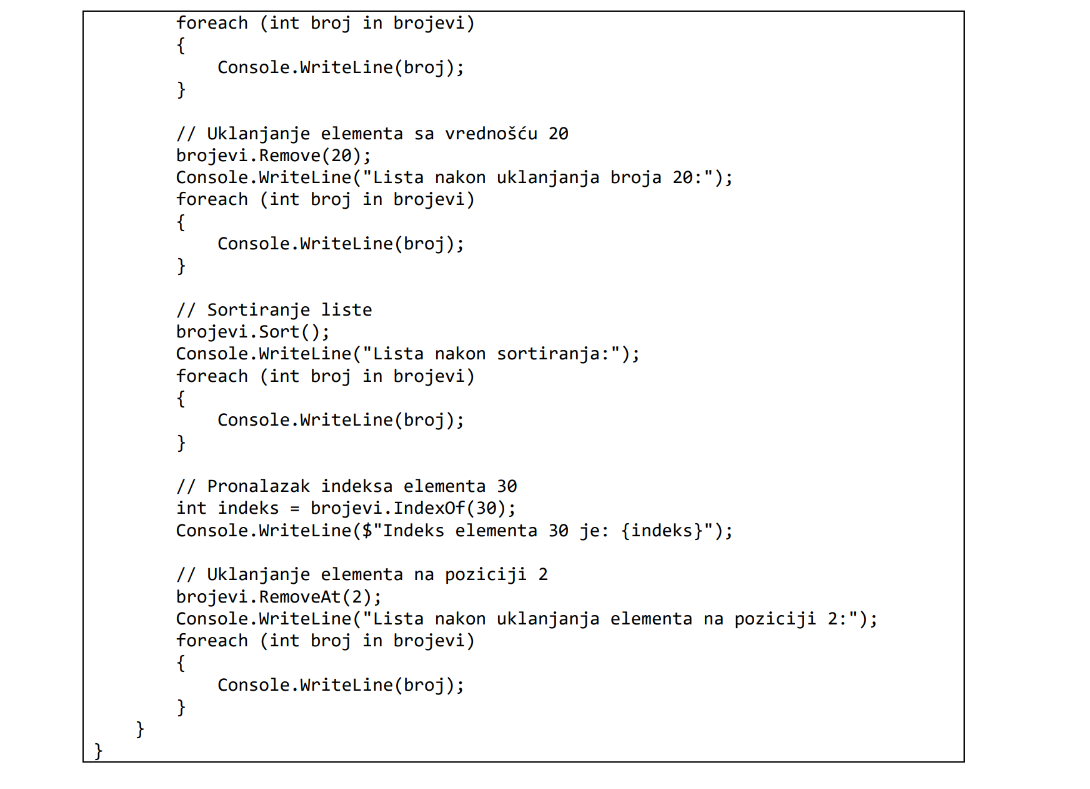
{

return await userRepository.DeleteAsync(korisnik);

}}}

**PRIMERI SA CASA**





**SORTIRANJE:**

Opadajuce:

1. Nacin

public List<Car> GetCars()

{

var x = from item in carRepository.GetCars()

orderby item.Price descending

select item;

return x.ToList();

}

1. Nacin

return carRepository.GetCars()

.OrderByDescending(c => c.Price)

.ToList();

Rastuce:

1. Nacin

public List<Car> GetCars()

{

var x = from item in carRepository.GetCars()

orderby item.Price // podrazumevano rastuće

select item;

return x.ToList();

}

1. Nacin

return carRepository.GetCars()

.OrderBy(c => c.Price) // rastuće

.ToList();

**POSTUPAK ZA GIT – SA SOURCE TREE:**

* Prijava na git
* Name uneti
* Uci na profil / repositories
* New , name uneti, public, add readme on, add gitignore – kreiran projekat
* Code, main (ako treba jos neka grana kreirati)
* Kopirati deo iz CODE
* Na radnoj povrsini napraviti folder u kome ce se cuvati
* Uci u SourceTree – file – new , pa kopirati ono sa gita u code , potom izabrati folder koji je kreiran za cuvanje – CLONE
* PREKOPIRATI DOBAR gitignore iz projekta za apartmane
* Potom u vs create new project i birati da se cuva u istom folderu kao sto je u SourceTree izabran
* Na kraju otici u SourceTree – StageAll, komentar, Commit, PUSH

**POSTUPAK ZA GIT – BEZ SOURCE TREE:**

* Prijava na git
* Name uneti
* Uci na profil / repositories
* New , name uneti, public, add readme on, add gitignore – kreiran projekat
* Code, main (ako treba jos neka grana kreirati)
* Kopirati deo iz CODE
* Uci u vs – clone repozitorijum pa kopirati sa gita CODE
* Path folder na radnoj povrisni u kome se cuva, pa CLONE
* PROMENITI gitignore
* Dole u desnom uglu klik na main ima deo remotes pa izabrati na koju granu teba
* File, new, project, class library
* Sada ima gitChanges I solutionExplorer
* Kada se uradi kod ide se na git changes i stage all, pise se message i comit all i strelica ne gore (PUSH)

**KORACI ZA PROJEKAT:**

* Kreiranje projekta, class library, data layer, NET 8.0
* Bazu kreirati, pa tabelu
* Kreirati txt fajl u okviru Data Layer da se upise kako je kreirana tabela
* SVUDA INTERNAL MENJATI U PUBLIC
* Moze se dodati folder Model u DataLayer i u njega klasa koja ce predstavljati entitet, tu ukoliko je potrebno postaviti sta su obavezna polja
* Potom napraviti u okviru DataLayer-a Interfejs i Klasu koja implementira taj interfejs
* Tu u klasi se definise konekcioni string koji se kopira iz baze
* Ako nece paket da se instalira neki ici na Manage NuGet Packages
* Potom se ide na solution desni klik I dodaje se class library BusinessLayer
* Dodati tu isto interfejs I klasu, pogledati koje se metode traze i njihove potpies napisati u interfejs a u klasi implementirati
* U toj biznis klasi se mora uraditi private readonly povezivanje uz konstruktor
* Potom metode pisati sta treba
* Ako negde ne moze da se vidi nesto videte jesu li povezani slojevi
* U ovoj klasi ide dekoracija
* Sada ide web sloj, na solution desni klik pa se bira web, blazorwebapp, I paziti da se cekira Do not use top-level statement
* U Program.cs spojiti sta treba od interfejsa I klasa
* Onda se prave stranice
* Promeniti da se pokrece web aplikacija

**PAZITI**

* U web delu inject NavigationManager NV i inject IBusinessBook bus, ali pre toga mora biti veza ka biznis sloju
* Dugme ne sme submit nego button

**Primeri:**

* **Ako treba na po ceni koja je izmedju minimalne i maksimalne:**

public List<Stavka> GetByCena(decimal minCena, decimal maxCena)

{

return \_repo.GetAll()

.Where(x => x.Cena >= minCena && x.Cena <= maxCena)

.OrderBy(x => x.Cena)

.ThenBy(x => x.Naziv)

.ToList();

}}

* **Ako treba vratiti student sa prosecnom ocenom (koja moze biti null) vecom od neke**  
  Pravilo: za **nullable vrednosne tipove** (decimal?, int?, …) direktno poređenje (>, <, …) vraća bool?, a Where traži bool. Zato ili koristiš HasValue && Value, ili pretvoriš u nenull vrednost (??, GetValueOrDefault), ili pattern is >.

public List<Student> GetWithAverageAbove(decimal minAverage)

{

return \_repo.GetAll()

.Where(s => s.AverageMark.HasValue && s.AverageMark.Value > minAverage)

.OrderByDescending(s => s.AverageMark)

.ThenBy(s => s.Name)

.ToList();

}

Ili  
public List<Student> GetWithAverageAbove(decimal minAverage)

{

return \_repo.GetAll()

.Where(s => s.AverageMark != null && s.AverageMark > minAverage)

.OrderByDescending(s => s.AverageMark) // opciono

.ThenBy(s => s.Name) // opciono

.ToList();

}

Ipak primer sa HasValue && Value je **ekvivalentna**, ali **nije neophodna**.

**Kada ipak nešto drugo?**

Ako želiš da **NULL tretiraš kao 0**:

.Where(s => (s.AverageMark ?? 0m) > minAverage)

Ako želiš da **uključiš i NULL** (npr. “svi koji nemaju prosek ili im je > X”):

.Where(s => s.AverageMark == null || s.AverageMark > minAverage)

**Finalni preporučeni kod**

public List<Student> GetWithAverageAbove(decimal minAverage)

{

return \_repo.Query() // ili GetAll()

.Where(s => s.AverageMark > minAverage)

.OrderByDescending(s => s.AverageMark)

.ThenBy(s => s.Name)

.ToList();

}

* **Top N studenata po proseku**

int topN = 10;

var res = \_repo.GetAll()

.Where(s => s.AverageMark != null)

.OrderByDescending(s => s.AverageMark)

.Take(topN)

.ToList();

* **Provera postojanja**

bool imaOdlican = \_repo.GetAll().Any(s => s.AverageMark >= 9.5m);

bool imaNeaktivnih = \_repo.GetAll().Any(s => !s.Active);

bool sviAktivni = \_repo.GetAll().All(s => s.Active);

int brojAktivnih = \_repo.GetAll().Count(s => s.Active);

* **Broj, suma, prosek**

int brojAktivnih = \_repo.Query().Count(s => s.Active);

decimal? prosekSvih = \_repo.Query()

.Where(s => s.AverageMark != null) // moze I bez ovoga

.Average(s => s.AverageMark);

* **Provera postojanja**

“da li postoji bar jedan…”, koristi **Any** (efikasnije od Count() > 0):

bool postojiOdlican = \_repo.Query().Any(s => s.AverageMark >= 9.5m);

bool imaNeaktivnih = \_repo.Query().Any(s => !s.Active);

Još korisno:

bool sviAktivni = \_repo.Query().All(s => s.Active); // true samo ako su SVI aktivni

int brojAktivnih = \_repo.Query().Count(s => s.Active); // brojanje sa uslovom

* **Find, FindAll**

var list = \_repo.GetAll().ToList();

var aktivni = list.FindAll(s => s.Active);

var jedan = list.Find(s => s.Id == 123);

bool postojiOdlican = list.Exists(s => s.AverageMark > 9m);

* **Filtriraj po listi ID-jeva (Contains → IN)**

var allowIds = new[] { 1, 3, 5 };

var res = \_repo.GetAll()

.Where(s => allowIds.Contains(s.Id))

.ToList();

* **lista jedinstvenih godina studija (bez ponavljanja)**

var godine = \_repo.GetAll()

.Select(s => s.Year)

.Distinct()

.OrderBy(y => y) // opciono

.ToList();

* **imena koja sadrze ana**

var term = "ana";

var studenti = \_repo.GetAll() // vraća List<Student>

.Where(s => s.Name != null && s.Name

.Contains(term, StringComparison.OrdinalIgnoreCase))

.ToList();  
ili:  
string term = "ana";

var res = \_repo.GetAll()

.Where(s => s.Name != null &&

s.Name.IndexOf(term, StringComparison.OrdinalIgnoreCase) >= 0)

.ToList();

* **vraća sve entitete Kupovina za koje važi Cena \* Kolicina > prag (podrazumevano 1000m) i sortira ih opadajuće po tom iznosu:**

csharp

CopyEdit

public List<Kupovina> GetStavkePreko(decimal prag = 1000m)

{

return \_repo.GetAll()

.Where(k => k.Cena \* k.Kolicina > prag)

.OrderByDescending(k => k.Cena \* k.Kolicina)

.ToList();

}

* **Suma ukupnih iznosa (Cena \* Kolicina)**

public decimal GetUkupnoSvihKupovina()

{

return \_repo.GetAll().Sum(k => k.Cena \* k.Kolicina);

}

* Ako je lista prazna, Sum vrati **0** (pošto je izraz nenull decimal).
* **Suma sa filterom (npr. samo > 1000 RSD)**

public decimal GetUkupnoPrekoPraga(decimal prag = 1000m)

{

return \_repo.GetAll()

.Where(k => k.Cena \* k.Kolicina > prag)

.Sum(k => k.Cena \* k.Kolicina);

}