

# Summary Module

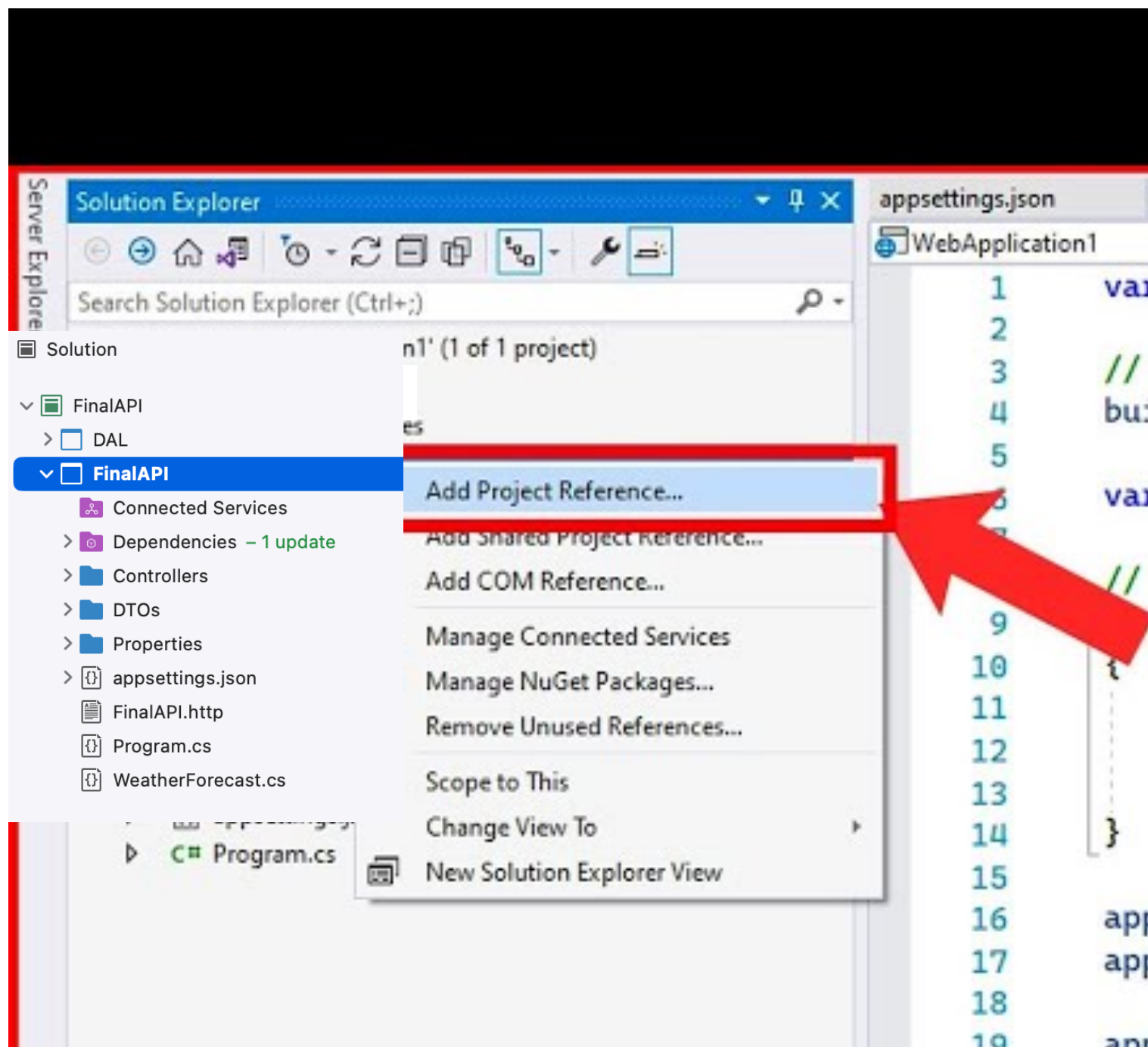
**Final Project and Cloud Deployment**

**דגשים לפרוייקט מסכם  
חזרה על נושאים נבחרים  
העלאה לענן**

TomerBu

# שילוב בין 2 הפרוייקטים - Final Api משתמש בפרוייקט DAL

FinalApi  
uses DAL



# Final Api שילוב בין 2 הפרוייקטים - DAL משתמש בפרוייקט

Dal->Data->ContextDal.cs

```
public class ContextDAL(DbContextOptions<ContextDAL> options) : DbContext(options)
{
    public DbSet<Category> Categories { get; set; } = default!;
    public DbSet<Product> Products { get; set; } = default!;

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);
        modelBuilder.Entity<Category>()
            .HasData([
                new Category(){Id = 1, Name = "Toys"},
                new Category(){Id = 2, Name = "Electronics"},
                new Category(){Id = 3, Name = "Games"}
            ]);
    }
}
```

# Final Api שילוב בין 2 הפרוייקטים - DAL משתמש בפרוייקט

זריעה של מוצרים בדטה-בייס:

```
modelBuilder.Entity<Product>()
    .HasData([
        new Product(){
            Id = 1,
            Name = "Macbook Pro",
            Description = "A Computer",
            CategoryId = 2,
            ImageUrl = "",
            Price = 999.9M
        },
        new Product(){
            Id = 2,
            Name = "Toy car",
            Description = "A Wooden Toy car",
            CategoryId = 1,
            ImageUrl = "",
            Price = 10.0M
        }
    ]);
```

# Final Api שילוב בין 2 הפרוייקטים - DAL משתמש בפרוייקט

זריעה של מוצרים בדטה-בייס:

בpackage manager console נריץ את השורות הבאות:

```
Add-Migration Seed  
update-database
```

# Final Api שילוב בין 2 הפרוייקטים - DAL משתמש בפרוייקט

בפרוייקט FinalApi: נוסף Connection String:

בקובץ appsettings.json

```
{ appsettings.json ×
FinalAPI > {} appsettings.json > ...
1 {
2   "Logging": {
3     "LogLevel": {
4       "Default": "Information",
5       "Microsoft.AspNetCore": "Warning"
6     }
7   },
8   "AllowedHosts": "*",
9   "ConnectionStrings": {
10    "ContextDAL": "Server=DESKTOP-GHIEC0F\\SQLEXPRESS;Database=FinalProjectDB;TrustServerCertificate=True;Trusted_Connection=True;MultipleActiveResultSets=true"
11  }
12 }
```

# Final Api שילוב בין 2 הפרוייקטים - DAL משתמש בפרוייקט

בפרוייקט FinalApi: נוסף את הservice של DbContext עם הגדרות חיבור לשרת SQL  
בקובץ Program.cs

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

```
// Add services to the container.  
builder.Services.AddDbContext<ContextDAL>(options =>  
    options.UseSqlServer(builder.Configuration.GetConnectionString("ContextDAL")  
        ?? throw new InvalidOperationException("Connection string 'ContextDAL' not found.")));
```

# Final Api שילוב בין 2 הפרוייקטים - DAL משתמש בפרוייקט

בפרוייקט FinalApi: ניצור קונטרולר שיגיש מוצרים מהדטה-בייס:

```
ProductsController.cs
FinalAPI > Controllers > ProductsController.cs > ...
1 using DAL.Data;
2 using Microsoft.AspNetCore.Mvc;
3
4 namespace FinalAPI.Controllers;
5
6 [Route("api/[controller]")]
7 [ApiController]
8
9 //TODO: USE Repo instead of DB CONTEXT
0 references
10 public class ProductsController(ContextDAL context) : ControllerBase
11 {
12     [HttpGet]
13     0 references
14     public ActionResult GetProducts()
15     {
16         return Ok(context.Products.ToList());
17     }
17 }
```

הרצת הפרוייקט:

נהפוך את FinalApi לפרוייקט הראשי ונריץ אותו:

בדיקה עם swagger:

<http://localhost:5059/swagger/index.html>

יופי - הכל עובד - יש לנו חיבור בין הפרוייקטים - זה עובד כך שהפרוייקט API

מגדיר את החיבור לדטה-בייס ומשתמש פשוט במחלקה של DbContext שנמצאת בפרוייקט DAL

אך למעשה יש לנו פרוייקט אחד שרץ ומשתמש במחלקה מ DAL (בקוד) כלומר ה API יוצר את האובייקט של DbContext



# תבנית העיצוב Repository

לפי התבנית - נכתוב את כל הקוד שעוסק בדטה-בייס במקום אחד מרכזי.  
נגדיר ממשק שמציין את כל הפעולות האפשריות על המידע שלנו ונממש אותו:

IRepository.cs

DAL > Data > IRepository.cs > ...

:פרויקט DAL

:תיקית Data

:קובץ חדש

IRepository.cs

```
1 using System.Linq.Expressions;
2
3 namespace DAL.Data;
  1 reference
4 public interface IRepository<T> where T : class
5 {
  1 reference
6     void Add(T entity);
  4 references
7     T? GetById(int id);
  3 references
8     IEnumerable<T> GetAll();
  2 references
9     IEnumerable<T> FindAll(Expression<Func<T, bool>> predicate);
  1 reference
10    T? FindOne(Expression<Func<T, bool>> predicate);
  1 reference
11    void Update(T entity);
  1 reference
12    void Delete(int id);
  2 references
13    void Delete(T entity);
  1 reference
14    void Delete(Expression<Func<T, bool>> predicate);
15 }
```

# תבנית העיצוב Repository

לפי התבנית - נכתוב את כל הקוד שעוסק בדטה-בייס במקום אחד מרכזי.  
נגדיר ממשק שמציין את כל הפעולות האפשריות על המידע שלנו ונממש אותו:

```
using Microsoft.EntityFrameworkCore;  
using System.Linq.Expressions;
```

דגשים: מתודה וירטואלית - אפשר לדרוס  
מחלקה אבסטרקטית - מחלקה שמיועדת לירושה

```
namespace DAL.Data;
```

```
public abstract class Repository<T>(ContextDAL context) : IRepository<T> where T : class  
{  
    DbSet<T> DbSet = context.Set<T>();  
    public virtual void Add(T entity)  
    {  
        DbSet.Add(entity);  
        context.SaveChanges();  
    }  
  
    public virtual void Delete(int id)  
    {  
        var item = GetById(id) ?? throw new ArgumentException("Item not found");  
        Delete(item);  
    }  
  
    public virtual void Delete(T entity)  
    {  
        DbSet.Remove(entity);  
        context.SaveChanges();  
    }  
}
```

המשך המחלקה בעמוד הבא:

פרויקט DAL:  
תיקית Data:  
קובץ חדש:  
Repository.cs

# תבנית העיצוב Repository

פרויקט DAL:  
תיקית Data:  
קובץ חדש:  
Repository.cs

המשך המחלקה מהעמוד הקודם:

```
public virtual void Delete(Expression<Func<T, bool>> predicate)
{
    DbSet.RemoveRange(FindAll(predicate));
    context.SaveChanges();
}
public virtual IEnumerable<T> FindAll(Expression<Func<T, bool>> predicate)
{
    return DbSet.Where(predicate);
}
public virtual T? FindOne(Expression<Func<T, bool>> predicate)
{
    return DbSet.SingleOrDefault(predicate);
}
public virtual IEnumerable<T> GetAll()
{
    //return context.Products.Include(p=>p.category);
    return DbSet.ToList();
}
public virtual T? GetById(int id)
{
    return DbSet.Find(id);
}
public virtual void Update(T entity)
{
    DbSet.Update(entity);
    context.SaveChanges();
}
}
```

הפרמטר של הפונקציה FindOne הוא ביטוי LINQ מייצג פונקציית חץ שמקבלת אובייקט ומחזירה בוליאני

זה כמו פונקציית filter בשפת Javascript ובאופן כללי - קוראים לפונקציות מהצורה הזאת Predicate

דגשים: מתודה וירטואלית - אפשר לדרוס  
מחלקה אבסטרקטית - מחלקה שמיועדת לירושה

# תבנית העיצוב Repository

פרויקט DAL:

תיקית Data:

קובץ חדש:

ProductRepository.cs

מחלקה קונקרטית לRepository  
אפשר לדרוס כאן מתודות

```
using DAL.Models;
using Microsoft.EntityFrameworkCore;

namespace DAL.Data;

public sealed class ProductsRepository(ContextDAL context) : Repository<Product>(context)
{
}
```

פרויקט DAL:

תיקית Data:

קובץ חדש:

CategoryRepository.cs

```
using DAL.Models;

namespace DAL.Data
{
    public sealed class CategoryRepository(ContextDAL context) : Repository<Category>(context)
    {
    }
}
```

# שימוש בRepository בפרויקט FinalApi

נוסיף את המחלקות שיצרנו למאגר של Dependency Injection:

DI Container

כך נוכל להזריק את האובייקטים הללו בקונטרולרים שלנו או במקומות אחרים שנרצה אותם:

Program.cs

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

// Add services to the container.
builder.Services.AddDbContext<ContextDAL>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("ContextDAL")
        ?? throw new InvalidOperationException("Connection string 'ContextDAL' not found"));

// builder.Services.AddScoped<IRepository<Product>, ProductsRepository>();
builder.Services.AddScoped<ProductsRepository>();
builder.Services.AddScoped<CategoryRepository>();
```

# Controller משתמש בRepository

נזריק את ProductsRepository לקונטרולר של המוצרים ונשתמש בו כדי להציג מוצרים:  
(במקום להשתמש ישירות בDbContext)

```
using DAL.Data;
using FinalAPI.DTOS;
using Microsoft.AspNetCore.Mvc;

namespace FinalAPI.Controllers;

[Route("api/[controller]")]
[ApiController]

public class ProductsController(ProductsRepository repository) : ControllerBase
{
    [HttpGet]
    public ActionResult GetProducts()
    {
        return Ok(repository.GetAll());
    }
}
```

הזרקה  
↓

שימוש  
↙

נריץ ונבדוק שהכל עובד.

יפה - אנחנו משתמשים בתבנית Repository  
והכל עובד היטב!

# Controller משתמש בRepository

צורך חדש - אנחנו רוצים להציג את פרטי הקטגוריה יחד עם פרטי המוצר.

פתרון (חלק ראשון) - נדרוס את המתודה בRepository שתחזיר תוצאה של Products JOIN Categories

```
using DAL.Models;
using Microsoft.EntityFrameworkCore;

namespace DAL.Data;

public sealed class ProductsRepository(ContextDAL context) : Repository<Product>(context)
{
    public override IEnumerable<Product> GetAll()
    {
        return context.Products.Include(p => p.Category);
    }
}
```

אם נריץ עכשיו - נגלה שיש לנו בעיה - מוצר מציג קטגוריה שמציגה מוצר שמציגה קטגוריה וכו'.

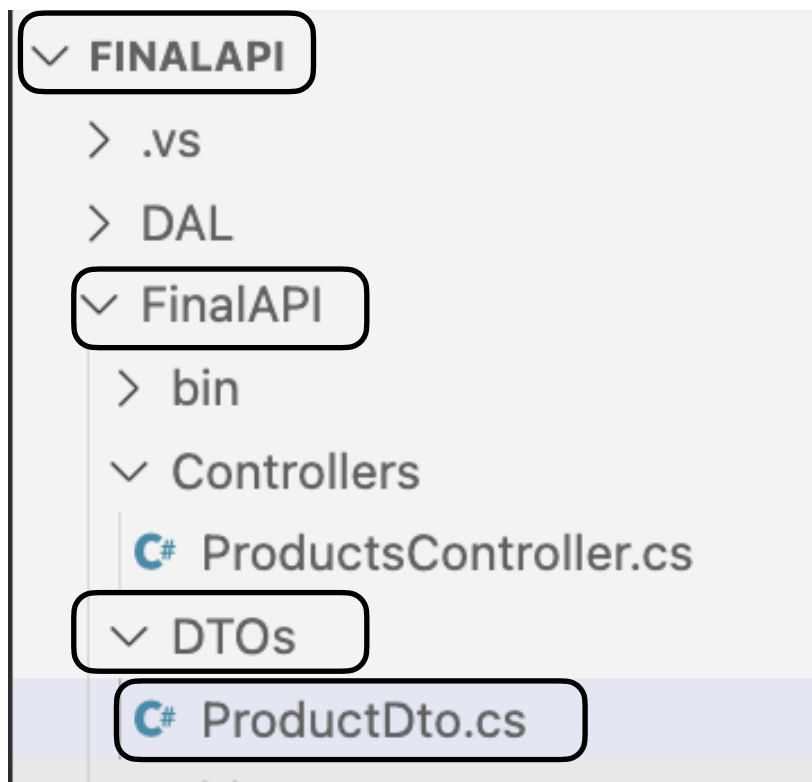
הJSON Serializer גורם כאן לרקורסיה כשהוא מנסה לתרגם את יחסי הגומלין לJSON  
הוא מוצא מוצר -> רואה קטגוריה -> רואה שיש בקטגוריה מוצרים וכל המוצרים הוא מוציא את הקטגוריה  
וכו'.....

פתרון (חלק שני) - ניצור DTO שיראה רק מוצר ושם קטגוריה וכך לא תהיה לנו רקורסיה בכלל.  
(DTO לבדו הוא לא פתרון קודם עושים JOIN ואז אפשר להציג את זה בDTO)  
(2 חלקים לפתרון).

# עבודה עם DTOs

צורך חדש - אנחנו רוצים להציג את פרטי הקטגוריה יחד עם פרטי המוצר.  
פתרון (חלק שני) - ניצור DTO שיראה רק מוצר ושם קטגוריה וכך לא תהיה לנו רקורסיה בכלל.  
(DTO לבדו הוא לא פתרון קודם עושים JOIN ואז אפשר להציג את זה ב-DTO)  
(2 חלקים לפתרון).

ניצור תיקיה DTOs בפרוייקט FinalApi  
ובתוכה קובץ חדש ProductDto.cs



```
using DAL.Models;

namespace FinalAPI.DTOs;

public class ProductDto
{
    public int Id { get; set; }
    public required string Name { get; set; }
    public required string Description { get; set; }
    public decimal Price { get; set; }
    public required string ImageUrl { get; set; }
    public required string Category { get; set; }
}
```

כל התכונות כמו בProduct אבל הקטגוריה שונה במקרה הזה -  
גם כדי למנוע רקורסיה וגם כי זה מבנה יותר נוח למשתמש



# עבודה עם DTOs

מתודת הרחבה -

מתודת עזר להמרה בקלות מProduct לProductDto

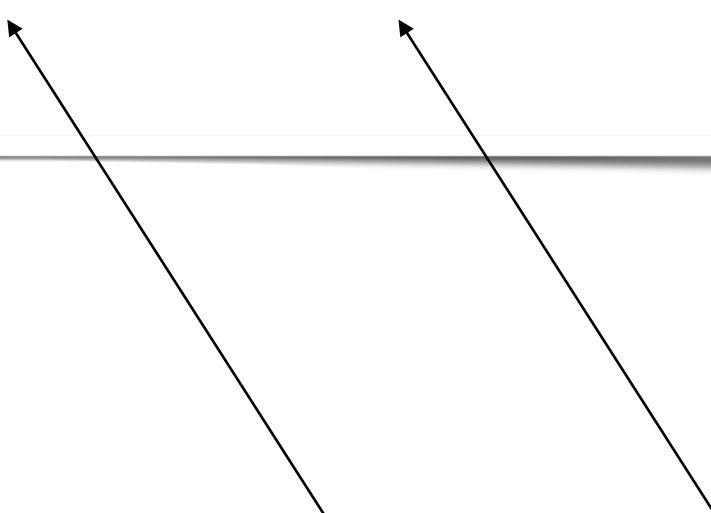
המתודה מקבלת Product ומחזירה ProductDto  
מתודה לנוחיות.

```
public static class ProductExtensions
{
    public static ProductDto ToDto(this Product p)
    {
        return new ProductDto() {
            Category = p.Category.Name,
            Id = p.Id,
            Name = p.Name,
            Description = p.Description,
            Price = p.Price,
            ImageUrl = p.ImageUrl,
        };
    }
}

//p.ToDto()
```

# שימוש ב DTO בקונטרולר:

```
public class ProductsController(ProductsRepository repository) : ControllerBase
{
    [HttpGet]
    public ActionResult GetProducts()
    {
        return Ok(repository.GetAll().Select(p => p.ToDto()));
    }
}
```



עבור כל מוצר נקרא למתודה שיצרנו בשלב הקודם - ToDto  
וכך אנחנו מבצעים את ההמרה  
מ IEnumerable של Product  
ל IEnumerable של ProductDto

## חזרה על התהליך - נוסף מתודה שמציגה מוצר לפי Id בקונטרולר:

```
[HttpGet("{id}")]
public ActionResult GetById(int id)
{
    var product = repository.GetById(id);

    if(product is null)
    {
        return NotFound();
    }
    return Ok(product);
}
```

שוב אותו רעיון - (אם רוצים להציג גם קטגוריה)

אנחנו צריכים לשלוח מהדטה-בייס גם את הקטגוריה ולהחזיר Dto שיציג יפה את המידע

(פתרון בעמוד הבא)

# חזרה על התהליך - נוסף מתודה שמציגה מוצר לפי Id בקונטרולר:

שליפה של מוצר כולל קטגוריה

```
public sealed class ProductsRepository(ContextDAL context) : Repository<Product>(context)
{
    public override IEnumerable<Product> GetAll()
    {
        return context.Products.Include(p => p.Category);
    }
    public override Product? GetById(int id)
    {
        return context.Products.Include(p => p.Category).SingleOrDefault(p => p.Id == id);
    }
}
```

```
[HttpGet("{id}")]
public ActionResult GetProductById(int id)
{
    var product = repository.GetById(id);

    if (product is null)
    {
        return NotFound();
    }
    return Ok(product.ToDto());
}
```

ה DTO

פותר 2 נושאים:

א) תמיד נחזיר Dtos כדי לשלוט בערך המוחזר.

ב) מונע רקורסיה במקרה הנוכחי

# שיעורי בית:

זה הזמן לחשוב על רעיון לפרוייקט.

מי שכבר מצא רעיון - מוזמנים לממש את הרעיון שלכם ולהתחיל לעבוד על הפרוייקט המסכם לפי ההנחיות שחולקו.