

ASP.NET Web APIs

SOLID

Single Responsibility Principle

עקרונות SOLID

Single Responsibility Principle

Open Closed Principle

Liskov's Substitution principle

Interface Segregation principle

Dependency Inversion principle

עקרונות מנחים בתכנות מונחה עצמים

אוסף הנחיות לקוד בר תחזוקה

מקל על ביצוע תוספות מבלי לכתוב מחדש את כל הקוד

עקרונות SOLID

Single Responsibility Principle עקרון האחריות הבודדת

לכל מחלקה או פעולה צריכה להיות אחת

אחריות - סיבה אחת ויחידה שבגללה נרצה לשנות את הקוד

עקרונות SOLID

Single Responsibility Principle עקרון האחריות הבודדת

לכל מחלקה או פעולה צריכה להיות אחריות אחת

אחריות - סיבה אחת ויחידה שבגללה נרצה לשנות את הקוד

מאפשר השקעה בלוגים

מאפשר לכתוב את הלוגים לקובץ או לשרת לוגים או מקומי

אפשר לפרמט את הלוגים, להוסיף תאריך שעה וכו'!

עקרונות SOLID

Open Closed Principle

קוד פתוח להרחבה = קוד סגור לשינויים

```
public class GalaxyS8 {  
    //props:  
    private int xResolution;  
    private int yResolution;  
  
    public void sendCommand(String command) {  
        //impl  
    }  
}
```

נרצה לעבוד עם דגם GalaxyS9

Device Manager
תלוי בצימוד בגלקסי 8
ולא פתוח לקבלת מכשירים חדשים

(לא פתוח להרחבה)

```
public class DeviceManager {  
    public void openLockScreen(List<GalaxyS8> devices) {  
        for (GalaxyS8 device : devices) {  
            device.sendCommand("swipe {x coordinates...}");  
        }  
    }  
}
```

עקרונות SOLID

Open Closed Principle

קוד פתוח להרחבה = קוד סגור לשינויים

```
public interface IDevice {  
    void sendCommand(String command);  
}
```

```
public class GalaxyS8 implements IDevice{  
    //props:  
    private int xResolution;  
    private int yResolution;  
  
    @Override  
    public void sendCommand(String command){  
        //impl1  
    }  
}
```

```
public class GalaxyS9 implements IDevice{  
    //props:  
    private int xResolution;  
    private int yResolution;  
  
    @Override  
    public void sendCommand(String command){  
        //impl2  
    }  
}
```

עקרונות SOLID

Open Closed Principle

קוד פתוח להרחבה = קוד סגור לשינויים

```
public interface IDevice {  
    void openLockScreen();  
}
```

```
public class GalaxyS8 implements IDevice{  
    //props:  
    private int xResolution;  
    private int yResolution;  
  
    @Override  
    public void openLockScreen() {  
        sendCommand("swipe to the left");  
    }  
  
    private void sendCommand(String command){  
        //impl  
    }  
}
```

```
public class DeviceManager {  
    public void openLockScreen(List<IDevice> devices) {  
        for (IDevice device : devices) {  
            device.openLockScreen();  
        }  
    }  
}
```

אם נוסיף עוד מכשיר iPhone -
המכשיר יממש IDevice

```
public class GalaxyS9 implements IDevice{  
    //props:  
    private int xResolution;  
    private int yResolution;  
  
    @Override  
    public void openLockScreen() {  
        sendCommand("swipe to the right");  
    }  
  
    private void sendCommand(String command){  
        //impl  
    }  
}
```

המחלקה DeviceManager
פתוחה להרחבה - יכולה לקבל כל Device

סגורה לשינויים - אין צורך לשנות את המחלקה
בהוספת מכשיר חדש

עקרון ההחלפה של Liskov's

האם כדאי לרשת/לממש?

האם נרצה שמחלקה A תירש ממחלקה B?

נבצע ירושה רק אם ניתן להחליף בין A ל B בלי לגרום לתקלות.

נבצע ירושה רק אם מתקיימת הזהות
A is a B

ISP - Interface segregation Principle

עקרון הפרדת ממשקים

נפריד את הממשקים שלנו לממשקים קצרים וענייניים

```
public interface OnProgressChanged {  
    void started();  
    void stopped();  
    void onProgressChanged(float position);  
}
```

עקרונות SOLID

Dependency Inversion (DI)

טיפוסים מסדר גבוה לא צריכים להיות תלויים בטיפוסים מסדר נמוך ישירות -

שניהם צריכים להיות תלויים באבסטרקציה.

מנהל תלוי ישירות במחלקה Worker
עדיף שיהיה תלוי באבסטרקציה!

```
class Manager {  
    Worker worker;  
  
    public void setWorker(Worker worker) {  
        this.worker = worker;  
    }  
  
    public void manage() {  
        if (worker == null) return;  
        worker.work();  
    }  
}
```

דרישה חדשה:

מנהל צריך לנהל גם עובדים רגילים

וגם עובדי Outsourcing

תלות באבסטרקציה:
מימוש ממשק / ירושה

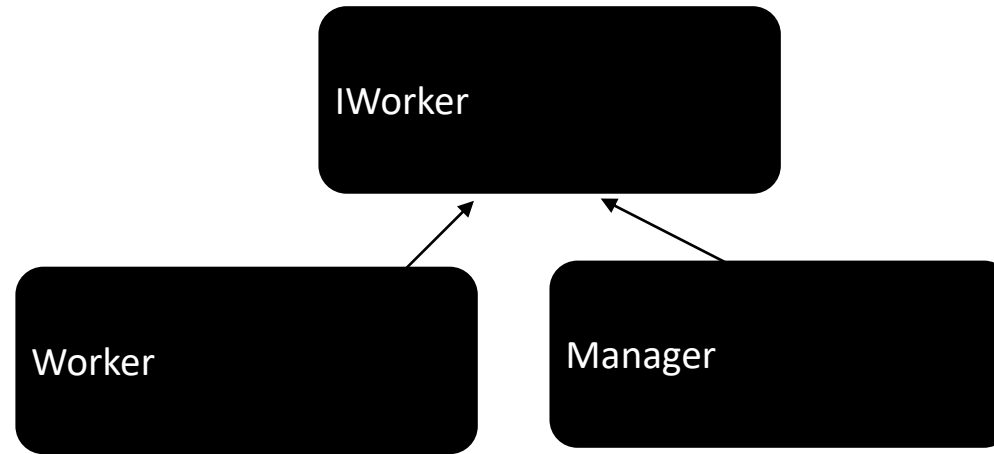
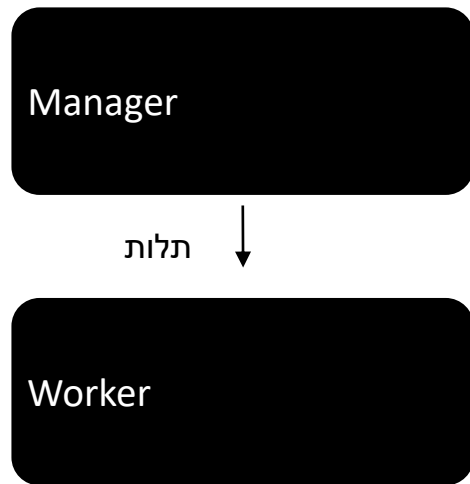
```
class Worker {  
    public void work() {  
        Console.WriteLine("Working...");  
    }  
}
```

תלות באבסטרקציה:
תלות בממשק (לא תלות במימוש קונקרטי)

Dependency Inversion Principle

התלות הפוכה - כל החיצים מופנים כלפי מעלה

כל החיצים מופנים כלפי מטה



Open Closed Principle

קוד פתוח להרחבה וסגור לשינויים

Liskov's Substitution Principle

האם כדאי לבצע ירושה/מימוש:

האם נרצה שמחלקה A תירש/תממש את מחלקה B

נבצע ירושה רק אם ניתן להחליף בין A לבין B בלי לגרום לתקלות

```
class Person{public string name;}
```

```
class Fish: Person{}
```

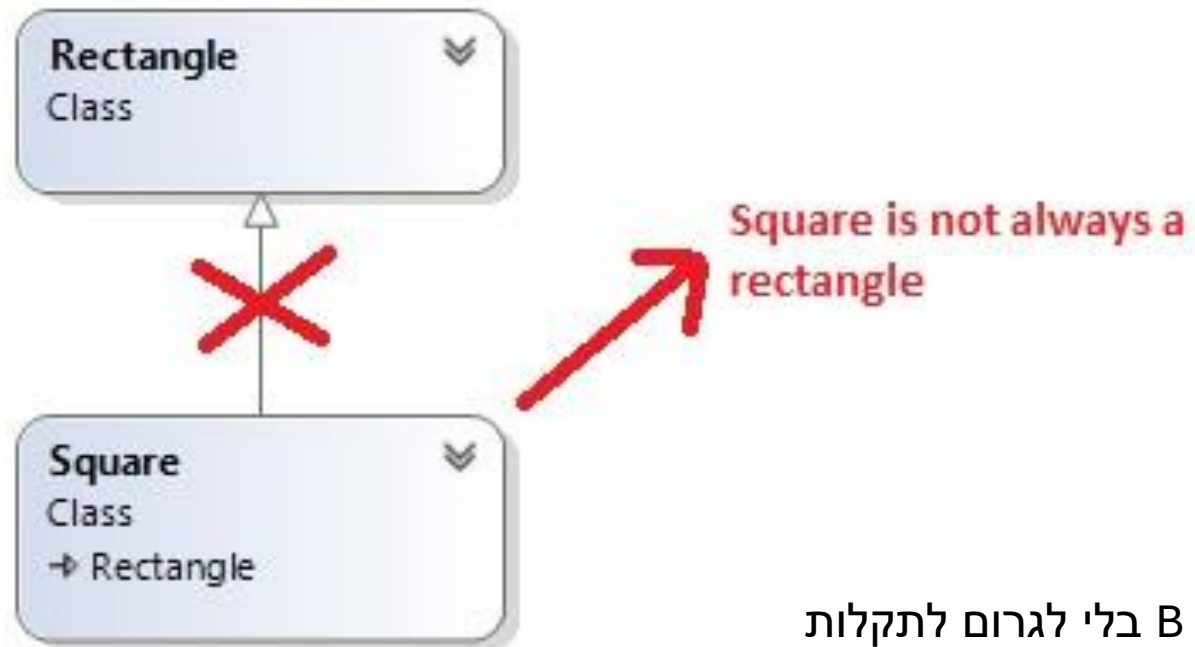


אם יש מתודה אחת או תכונה אחת לא מתאימה לא נירש
נחשב מסלול מחדש כדי ליצור הגיון וסדר.

נבצע ירושה רק אם מתקיים

$A \text{ IS a } B$

Liskov's Substitution Principle



נבצע ירושה רק אם ניתן להחליף בין A לבין B בלי לגרום לתקלות

נבצע ירושה רק אם מתקיים
A IS a B

SOLID
Interface Segregation Principle
עקרון הפרדת הממשקים:

נשתדל ליצור ממשקים קצרים וענייניים

```
interface OnClick{  
    void singleClick();  
    void doubleClick();  
    void longClick();  
}
```

היה עדיף להפריד ל3 ממשקים שונים
כי אחרת מי שיממש את זה – יכתוב מתודות ריקות שהוא לא צריך

SOLID

Dependency Inversion Principle

מחלקה מסדר גבוה – לא צריכה להיות תלויה במחלקה מסדר נמוך!

במקום זה 2 המחלקות יהיו תלויים באבסטרקציות

את כל העקרונות האלה נממש בפרוייקטים:

SRP

לכל מחלקה יש אחריות אחת – סיבה אחת לשינוי...

לכן לא נרצה לכתוב

Controller

שמבצע גם פעולות דטה-בייס וגם פעולות של ניתוב

מחלקה אחרת תטפל בפעולות דטה-בייס

MovieRepository – מימוש של המתודות
IMovieRepository הגדרה של המתודות

אחריות של קונטרולר:
לקבל בקשה ולהחזיר תגובה:

לחשוב על סטטוסים נכונים.
מבנה של התגובה.

מבנה מעודכן ל-API

לקוח
משתמש ב
HTTP

Controller
קוד של בקשה/תגובה

Repository
קוד של פעולות דטה-בייס

אחסון נוח של Connection String

WeatherForecastController.cs appsettings.json X

Schema: <https://json.schemastore.org/appsettings.json>

```
1  {
2    "Logging": {
3      "LogLevel": {
4        "Default": "Information",
5        "Microsoft.AspNetCore": "Warning"
6      }
7    },
8    "ConnectionStrings": {
9      "MongoDBConnectionString": "mongodb://127.0.0.1:27017/iloved
10   },
11   "AllowedHosts": "*"
12 }
13
```

Service שיחזיק את החיבור לדטה-בייס פתוח

```
namespace WebAppMongo.Service
{
    4 references
    public class MongoService
    {
        private readonly IMongoDatabase _database;

        0 references
        public MongoService(IConfiguration config)
        {
            var connectionString = config.GetConnectionString("MongoDBConnectionString");
            var client = new MongoClient(connectionString);
            _database = client.GetDatabase(config["DatabaseName"]);
        }

        2 references
        public IMongoCollection<T> GetCollection<T>(string name)
        {
            return _database.GetCollection<T>(name);
        }
    }
}
```

הגדרת השירות שלנו כסינגלטון בקונטיינר של DI

```
public class Program
{
    0 references
    public static void Main(string[] args)
    {
        var builder = WebApplication.CreateBuilder(args)

        // Add services to the container.

        builder.Services.AddControllers();

        builder.Services.AddSingleton<MongoService>();
        // Learn more about configuring Swagger/OpenAPI
        builder.Services.AddEndpointsApiExplorer();
        builder.Services.AddSwaggerGen();
    }
}
```

מודל לעבודה עם Mongo לטובת סריליזציה אוטומטית לBsonDocument

▼ namespace ApisModuleLec3.Models

{

11 references | [Copy link to this type](#)

▼ public class Movie

{

[BsonId]

[BsonRepresentation(BsonType.ObjectId)]

2 references | [Copy link to this property](#)

public string? Id { get; set; }

0 references | [Copy link to this property](#)

public string Title { get; set; } = default!;

0 references | [Copy link to this property](#)

public string Description { get; set; } = default!;

}

}

עבודה עם Repository

```
namespace ApisModuleLec3.Repository
{
    3 references | Copy link to this type
    public interface IMovieRepository
    {
        2 references | Copy link to this method
        Task<IEnumerable<Movie>> GetAllMovies();

        2 references | Copy link to this method
        Task<Movie> AddMovieAsync(Movie movie);

        2 references | Copy link to this method
        Task<Movie?> GetByIdAsync(string id);

        0 references | Copy link to this method
        Task UpdateAsync(Movie movie);

        0 references | Copy link to this method
        Task DeleteAsync(string id);
    }
}
```

ממשק – רשימה של
פעולות למימוש

עבודה עם Repository

מימוש הממשק:

```
namespace ApisModuleLec3.Repository
{
    1 reference | Copy link to this type
    public class MovieRepository(IMongoService mongo) : IMovieRepository
    {
        //private props:
        private readonly IMongoCollection<Movie> _movies = mongo.GetCollection<Movie>("movies");

        2 references | Copy link to this method
        public async Task<Movie> AddMovieAsync(Movie movie)
        {
            await _movies.InsertOneAsync(movie);
            return movie;
        }
    }
}
```

הנגשה של Repository לכל האפליקציה - DI

Program.cs

```
using ApisModuleLec3.Repository;
using ApisModuleLec3.Service;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

//make our objects available to the rest of the app:
builder.Services.AddSingleton<IMongoService, MongoService>();
builder.Services.AddSingleton<IMovieRepository, MovieRepository>();

//SRP (1) how to connect to mongo
//      (2) repository: CRUD Mongo operations
//      (3) controller: routes and actions and responses
```

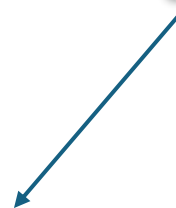
שימוש בController

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

0 references | [Copy link to this type](#)

```
public class MoviesController(IMovieRepository repo) : ControllerBase  
{
```

הזרקה בבנאי



הצגת כל הרשומות:

2 references | [Copy link to this method](#)

```
Task<IEnumerable<Movie>> GetAllMovies();
```

ממשק
IMovieRepository

2 references | [Copy link to this method](#)

```
public async Task<IEnumerable<Movie>> GetAllMovies()
{
    var cursor = _movies.Find(_ => true);

    //execute the find:
    var movies = await cursor.ToListAsync();

    return movies;
}
```

מימוש הממשק
MovieRepository

[HttpGet]

1 reference | [Copy link to this method](#)

```
public async Task<IActionResult> Get()
{
    var movies = await repo.GetAllMovies();
    return Ok(movies);
}
```

מתודה בController

הצגת רשומה לפי ID:

```
Task<Movie?> GetByIdAsync(string id);
```

ממשק
IMovieRepository

```
public async Task<Movie?> GetByIdAsync(string id)
{
    return await _movies.Find(m => m.Id == id).FirstOrDefaultAsync();
}
```

מימוש הממשק
MovieRepository

```
[HttpGet("{id}")]
```

1 reference | [Copy link to this method](#)

```
public async Task<IActionResult> Get([FromRoute] string id)
{
    var movie = await repo.GetByIdAsync(id);

    if (movie is null)
    {
        return NotFound();
    }

    return Ok(movie);
}
```

מתודה בController

מחיקת רשומה לפי ID:

```
Task DeleteAsync(string id);
```

ממשק
IMovieRepository

```
public async Task DeleteAsync(string id)
{
    await _movies.DeleteOneAsync(m => m.Id == id);
}
```

מימוש הממשק
MovieRepository

```
[HttpDelete("{id}")]
0 references | Copy link to this method
public async Task<IActionResult> Delete([FromRoute] string id)
{
    var existingMovie = await repo.GetByIdAsync(id);

    if (existingMovie is null)
    {
        return NotFound();
    }

    await repo.DeleteAsync(id);

    return NoContent();
}
```

מתודה בController

עדכון רשומה לפי ID:

```
Task UpdateAsync(Movie movie);
```

ממשק
IMovieRepository

```
public async Task UpdateAsync(Movie movie)
{
    await _movies.ReplaceOneAsync(m => m.Id == movie.Id, movie);
}
```

מימוש הממשק
MovieRepository

```
[HttpPut("{id}")]
```

0 references | [Copy link to this method](#)

```
public async Task<IActionResult> Put([FromRoute] string id, [FromBody] Movie movie)
{
    var existingMovie = await repo.GetByIdAsync(id);

    if (existingMovie is null)
    {
        return NotFound();
    }

    movie.Id = id;
    await repo.UpdateAsync(movie);

    return NoContent();
}
```

מתודה בController

הוספת רשומה – כולל Created At Action

```
Task<Movie> AddMovieAsync(Movie movie);
```

ממשק
IMovieRepository

```
public async Task<Movie> AddMovieAsync(Movie movie)
{
    await _movies.InsertOneAsync(movie);
    return movie;
}
```

מימוש הממשק
MovieRepository

[HttpPost]

0 references | [Copy link to this method](#)

```
public async Task<IActionResult> Post([FromBody] Movie movie)
{
    var result = await repo.AddMovieAsync(movie);

    //status 201 = Created!
    //TODO: GET BY ID => created!

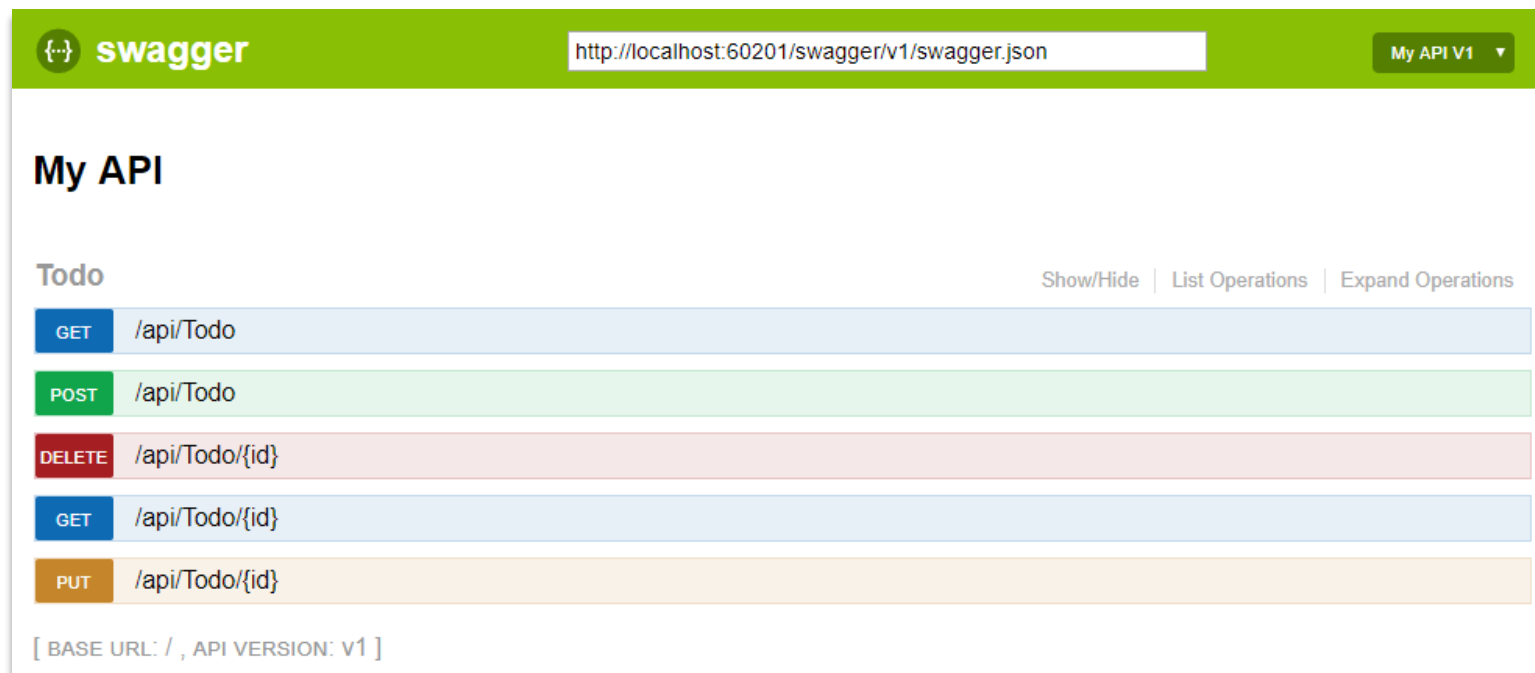
    //Created => goto Get/{id}
    return CreatedAtAction(nameof(Get), new { Id = result.Id! }, result);
}
```

מתודה בController

שיעורי בית:

(1) מומלץ לקרוא ולצפות בסרטונים בנושא
Solid Principles OOP

(2) צרו את ה API הבא על בסיס ההפרדה לController/Repository:



The image shows the Swagger UI interface for an API named "My API". The top bar is green and contains the Swagger logo, the URL "http://localhost:60201/swagger/v1/swagger.json", and a dropdown menu for "My API V1". Below the top bar, the title "My API" is displayed. Underneath, the section "Todo" is shown with a "Show/Hide" button, a "List Operations" button, and an "Expand Operations" button. The operations are listed in a table with columns for the HTTP method and the endpoint path.

Method	Path
GET	/api/ToDo
POST	/api/ToDo
DELETE	/api/ToDo/{id}
GET	/api/ToDo/{id}
PUT	/api/ToDo/{id}

[BASE URL: / , API VERSION: V1]