SOLID Design principles Summery By Mahmoud Badawy

-----------------------------------------------------

ملخص SOLID Design principles

الملخص يحتوى على

- SOLID Design principles

المراجع المستخدمه فى الشرح   
<https://medium.com/@mirzafarrukh13/solid-design-principles-c-de157c500425>

**و الفيدوهات المهمه دى جميله**   
<https://www.youtube.com/playlist?list=PLZ-26vZNn3AsdC_eKuo2mXmXECExlPoI3>

MB

########################################################################

SOLID Meanning

-----------------------

كلمه SOLID دى عباره عن اختصار ل 5 حاجات و هى

1. ***S***RP The Single Responsibility Principle : و دى معناها ان كل كلاس مسؤول عن وظيفه واحده فقط يقوم بيها
2. ***O***CP The Open Closed Principle : و دى يا سيدى معناها انى اعمل كلاس مفتوح للزياده و مقفول للتعديل يعنى لو عاوز ازود عليه قشطه مفيش مشكله انما اعدل جواه لأ   
   بمعنى انى اعمل كلاس Abstract يكون ده parent لكل الكلاسيس الى فيها حاجات مشتركه معاه و اخليهم يورثوا منه فكده هما اخدوا نفس الخصائص بتعته و كل الى حصل انى جواهم عملت اوفر رايت و غيرت فى اللوجيكك بتعهم بس و بقوا كلهم ينفع اناديهم بنفس طريقه ال parent و بالتالى حبيت اضيف ابن جديد او اشيله مش هيخلينى اعدل فى الكود حاجه لانه فى كل الاحوال انا ببقى كاتب فى الكود للكل اوبيجكت واحد من نوع ال parent و مخزن جواه ال childs و وقت ما بحتجهم بستعملهم
3. ***L***SP The Liskov Substitution Principle : دى يا سيدى انى اقدر و انا بعمل اوبجيكت من ال base Class انى اعمل ريفرنس ل Child or Derived Class يعنى يبقى الكلاس الاب هو مثلا customer و الكلاس الابن هو Enquires , SilverCustomer, GoldCustomer فبدل اما انا و انا بانشى كلاس كاستومار كده

List<Customer> Customers = new List<Customer>();

Customers.Add(new Customers());

تبقى بتشاور على الابناؤ كده و تشتغل

List<Customer> Customers = new List<Customer>();

Customers.Add(new SilverCustomer());

Customers.Add(new goldCustomer());

Customers.Add(new Enquiry());

1. ***I***SP The Interface Segregation Principle : دى معناها ان المفروض انى و انا بعمل ال Interface اراعى ان اننا مجبرش حد على وراثه وظايف او سلوك هو مش هيحتاجه ذى مثلا انى ابقى عندى كذا لعبه الكذا لعبه دول فيهم صفات مشتركه ذى ان ليهم سعر و ليهم لون بينما بيختلفوا فى بعض الصفات ذى ان فى لعبه بتمشى و لعبه بتطير فكده انا عندى 4 صفات صفتين مشتركين و صفتين مختلفين يبقى انا المفروض مش اعمل انترفيس واحد احط فيه كل الصفات ال 4 لأ خالص الصح انى اعمل 3 interfaces واحد يبقى بيحتوى على ال 2 صفات المشتركه و 1 يحتوى على صفه المشى و واحد يحتوى على صفه الطير و لما احتاج حاجه اورثها و لما محتجش مورثش و شكرا على كده
2. ***D***IP The Dependency Inversion Principle : بيقولك هنا ان انت طالما عندك كلاسات ابناء و كلاسات اباء الى هما low-level و high-level بيقولك انت بقه ياباشا متحليش اعتماد الابناء على الاب ولا اعتماد الاب على الابناء لأ شفولك layer جديده بينهم تعزلهم عن بعض لان الاتنين عاملين ذى سلوك الكهرباء لو قربوا من بعض هنموت كلنا هههههههه طب ايه هى اللاير الى ممكن اخليهم الاتنين يعتمدوا عليها و تبقى حلقه الوصل بينهم ؟؟  
   abstract Class او الاحسن interface   
   يبقى انا دلوقتى هعمل interface و الانترفيس ده هيبقى حلقه الوصل بين الكلاس الاب و الابناء فكده اعزل بينهم تماما و يبقى الاب بيورث من الانترفيس و العيال بيورثوا من الانترفيس فكده هما مبقاش ليهم اى علاقه بتربيطهم ببعض غير ال Interface



MB

########################################################################

***S***RP The Single Responsibility Principle

----------------------------------------------

***S***RP The Single Responsibility Principle : و دى معناها ان كل كلاس مسؤول عن وظيفه واحده فقط يقوم بيها

هنا هوريك الاسلوب الغلط فى الكتابه و يتصلح اذاى   
فذى ما انت شايف الكلاس دى المفروض من اسمها انها ليها علاقه بال Data و الفانكشن الى فيها المفروض هتعمل insertData و خلاص و لكن انا لقيت ان الفانكشن بتعمل انسيرت للداتا فى الداتا بيز و كمان بتعمل اللوج فكده الفانكشن دى بتعمل كذا حاجه مش حاجه واحده

class DataAccess

{

public static void InsertData()

{

Console.WriteLine("Data inserted into database successfully");

Console.WriteLine("Logged Time:" + DateTime.Now.ToLongTimeString() +

" Log Data insertion completed successfully");

}

}

طيب المفروض الاحسن الكود دى يتعمل اذاى ؟؟

المفروض ان التعامل مع الداتا يبقى فى كلاس و التعامل مع اللوج يبقى فى كلاس تانى خالص و ده الى هعمله فى فعلا هعمل كلاس للداتا و كلاس للوج

class DataAccess

{

public static void InsertData()

{

Console.WriteLine("Data inserted into database successfully");

}

}

// Logger class is only responsible for logging related operations

class Logger

{

public static void WriteLog()

{

Console.WriteLine("Logged Time:" + DateTime.Now.ToLongTimeString() +

" Log Data insertion completed successfully");

}

}

كده الكود بقه كل كلاس مسؤول عن حاجع معينه يعملها و بالتالى الموضوع بقى جميل جدا و الحاجه منظمه جدا

MB

########################################################################

***O***CP The Open Closed Principle

----------------------------------------------

***O***CP The Open Closed Principle : و دى يا سيدى معناها انى اعمل كلاس مفتوح للزياده و مقفول للتعديل يعنى لو عاوز ازود عليه قشطه مفيش مشكله انما اعدل جواه لأ

هنا يا اسطى هيبان فايده الموضوع ده لما اعملك كود معفن و كود بالبرنسيبل دى فأولا انا بقولك انا عاوز احسب مساحه ال Rectangle و Circle و الاتنين حساب المساحه بتاعت شكلهم مختلف تماما   
فالكود المعفن انى اعمل كلاس Rectangle و كلاس لل circul و كلاس ال AreaCalculator و داخل كلاس ال AreaCalculator تبقى العمليه الحسابيه لكل واحد منهم مبنى على اساس ال if Condition لو الانتسانس الى جاى مستطيل احسبلى المستطيل و لو الى جاى دايره احسبلى الدايره

namespace Web.API

{

public class Rectangle

{

public double Width { get; set; }

public double Height { get; set; }

}

public class Circle

{

public double Radius { get; set; }

}

public class AreaCalculator

{

public double Area(object[] shapes)

{

double area = 0;

foreach (var shape in shapes)

{

if (shape is Rectangle)

{

Rectangle rectangle = (Rectangle)shape;

area += rectangle.Width \* rectangle.Height;

}

else if (shape is Circle)

{

Circle circle = (Circle)shape;

area += circle.Radius \* circle.Radius \* Math.PI;

}

}

return area;

}

}

}

طبعا انا كده كل اما هزود شكل مربع و ليكن جيت قولتلك زودلى معين عليهم ايه الى هيحصل ؟؟ الى هيحصل انى كده هروح اعمل كلاس جديد خاص بال مربع و اديه خصائصه و بعد كده الكارثه الكبرى انى هروح اغير فى الفانكشن بقه الى موجوده فى الكلاس AreaCalculator الى هى ِArea و هيبقى الكود كده بعد التعديل

namespace Web.API

{

public class Rectangle

{

public double Width { get; set; }

public double Height { get; set; }

}

public class square

{

public double Width { get; set; }

public double Height { get; set; }

}

public class Circle

{

public double Radius { get; set; }

}

public class AreaCalculator

{

public double Area(object[] shapes)

{

double area = 0;

foreach (var shape in shapes)

{

if (shape is Rectangle)

{

Rectangle rectangle = (Rectangle)shape;

area += rectangle.Width \* rectangle.Height;

}

else if (shape is Circle)

{

Circle circle = (Circle)shape;

area += circle.Radius \* circle.Radius \* Math.PI;

}

else if (shape is square)

{

square square = (square)shape;

area += square.Width \* square.Height;

}

}

return area;

}

}

}

و من هنا هيظهر مشكله انك كل شويه هتحتاج تروح تغيير فى الفانكشن دى يا اسطى و دى حاجه مش حلوه يعنى طب ما انت تقدر تحلها بطريقه جميله جدا   
انك تعمل Abstract Class اسمه Shapes يكون BaseClass فيه الصفات المشتركه بين الاشكال التلاته و تكون الصفات دى abstract و عشان كده على المثال بتعنا الكلاس ده هيبقى فيه Abstract function اسمها Area الى هى هنستخدمها فى كله و بما ان دى حاجه ابستراكت فانا اقدر كده اعملها override و بالتالى الفانكشن دى هتبقى ثابته فى الكلاس ال Base بس اقدر اغيرها و ازود عليها فى الكلاسات الى هتورث و الى هى كلاس المستطيل و الدايره و المربع بينما فى الكلاس ال Base لا انا بغيرش فيها حاجه و لما باجى استخدم الكلاس ال Base فى الكود ببقاش مضطر انى اغير كل شويه برضوا فى الكود و ده هيبان من الكود الى هعمله دلوقتى

namespace Web.API

{

public abstract class Shapes

{

public abstract double Area();

}

public class Rectangle : Shapes

{

public double Width { get; set; }

public double Height { get; set; }

public Rectangle(double Width, double Height)

{

this.Width = Width;

this.Height = Height;

}

public override double Area()

{

return Width \* Height;

}

}

public class square : Shapes

{

public double Width { get; set; }

public double Height { get; set; }

public square(double Width, double Height)

{

this.Width = Width;

this.Height = Height;

}

public override double Area()

{

return Width \* Height;

}

}

public class Circle : Shapes

{

private double Radius { get; set; }

public Circle(int Radius)

{

this.Radius = Radius;

}

public override double Area()

{

return Radius \* Radius \* Math.PI;

}

}

public class MyProgram

{

public static void Mystart()

{

List<Shapes> shapes = new List<Shapes>();

shapes.Add(new Circle(5));

shapes.Add(new Rectangle(5, 7));

shapes.Add(new square(5, 7));

// shapes.Add(null);

AreaCalculator Ar = new AreaCalculator();

Ar.Area(shapes);

}

}

public class AreaCalculator

{

public double Area(List<Shapes> shapes)

{

double area = 0;

foreach (var shape in shapes)

{

area += shape.Area();

}

return area;

}

}

}

من الاصفر ده هتلاحظ انى الكلاس ال Base انا بعملها و بغيرش فيها حاجه و لو عاوز اضيف اى شكل جديد بروح اعمل كلاس جديده و اخليها تورث من ال Base عشان اعرف اضيفها ل list<shapes> براحتى لانى طالما ورثت من كلاس ما اقدر اضيف فى اللست بتعته من الابناء عادى و طلما ضيفت الحاجات دى فى لست من نوع shape يبقى اقدر اللف عليهم واحد واحد اجيب مسحته عادى فى ال foreach من غير ما اكتب ده انهى كلاس ولا انهى شكل و بالتالى اضافه اى شكل او كلاس او حذف مش هيأثر على الكود تماما بمعنى لو مسحت مربع و لا مسحت مستطيل و لا عملت اى ابن كلب او عدلت جوه الابناء فى الفانكشنز الى فيها override مش هيأثر خالص على الفانكشن الموجوده فى ال AreaCalculator و مش هاجى ناحيه الكود و كذلك مش هغير خالص فى ال BaseClass و يبقى انت كده برنس فى نفسك و الكود جامد اخر حاجه من الاخر الكلام ده معمول عشان لما اجى اكبر المشروع مجيش ناحيه الكود و يبقى مجرد بعمل كلاسات جديده او احذف كلاسات مش محتاجها و كده

MB

########################################################################

***L***SP The Liskov Substitution Principle

----------------------------------------------

***L***SP The Liskov Substitution Principle : دى يا سيدى انى اقدر و انا بعمل اوبجيكت من ال base Class انى اعمل ريفرنس ل Child or Derived Class يعنى يبقى الكلاس الاب هو مثلا customer و الكلاس الابن هو Enquires , SilverCustomer, GoldCustomer فبدل اما انا و انا بعمل اوبجيكت من كلاس كاستومار كده

List<Customer> Customers = new List<Customer>();

Customers.Add(new Customers());

تبقى بتشاور على الابناء كده و تشتغل عادى و ذى الفل

List<Customer> Customers = new List<Customer>();

Customers.Add(new SilverCustomer());

Customers.Add(new goldCustomer());

Customers.Add(new Enquiry());

بمعنى انى اقدر استخدم اى كلاس ابن مكان الكلاس الاب فى اى مكان من غير ما الكود يتأثر يا باشا

طيب الكلام ده بيحصل بطريقه غلط ممكن توقعك فى الغلط و طريقه صح : ايه بقه الطريقه الغلط و ايه الطريقه الصح ؟

الطريقه الغلط انى اعمل الابناء بيورثوا من ال Customer بشكل مباشر لان كده هيورثوا كل حاجه من الكلاس كاستومار و بالتالى هيبقوا واخدين كل صفات ال Customer و ذى ما انت شايف عندك Enquiry ده المفروض عميل محتمل يعنى اكيد فيه صفات اقل من الكاستومار العادى فكده انا عشان مخليهوش كاستومار هبقى مضطر اعمله ايرور جوه الامبلمنتيشن بتاعه و لما هاجى اعمله اضافه فى العميل الكومبايلار مش هيعترض بس اول ما هيعدى على الكود هيطلعلك ايرور اد العجل فى وش الناس فى الابليكيشن ذى ما انت شايف هنا كده

namespace Web.API

{

class Customer

{

public virtual double getDiscount(double TotalSales)

{

return TotalSales - 15;

}

public virtual void insert()

{

// implementation code

}

}

class SilverCustomer : Customer

{

public override double getDiscount(double TotalSales)

{

return base.getDiscount(TotalSales) - 5;

}

public override void insert()

{

// implementation code

}

}

class goldCustomer : Customer

{

public override double getDiscount(double TotalSales)

{

return base.getDiscount(TotalSales) - 5;

}

public override void insert()

{

// implementation code

}

}

class **Enquiry** : Customer

{

public override double getDiscount(double TotalSales)

{

return base.getDiscount(TotalSales) - 5;

}

public override void insert()

{

throw new Exception("Not allowed");

}

}

class MyProgram

{

public void start()

{

List<Customer> Customers = new List<Customer>();

Customers.Add(new SilverCustomer());

Customers.Add(new goldCustomer());

Customers.Add(new Enquiry());

foreach (Customer o in Customers)

{

o.insert(); //throw exception for Enquiry

}

}

}

}

فى الكود ده فى ال enquires بالتحديد بما انها بتحتوى على فانكشن ال insert بسبب انها ورثته من ال customer فى حين ان دى صفه المفروض انها متبقاش فيها فانا عشان مشغلهاش اضطريت احطلها Throw Excption عشان تطلعلى ايرور لما استخدمها و لكن دى هتطلعلى run time Error مش Compiler Error و بالتالى دى خزوق ارض جو فى الكود   
  
طب حل المشكله دى اذاى ؟؟  
حل المشكله دى انى عمل لكل فانكشن هقوم بيها فى الكاستومار interface و احطه فيها لان انا اقدر اورث من كذا انترفيس انما مقدرش اورث من كذا كلاس و ده الى هيتيح ليا بقه انى اخد الوظيفه الى على مزاجى و مخدش الوظيفه الى على مزاجى عن طريق انى اورث انترفيس معين او مورسش انترفيس معين برضوا   
ففى حال ال customer نفسه هيورث من كل الانترفيسيس عشان يبقى متوفر فيه كل الوظائف الخاصه بالعملاء لانه اصلا ده ال Base Class

فى حاله ال SilverCusomer و ال GoldCustomer دول فيهم صفات كل العملاء اصلا و يزيدوا فكده انا اقدر اريح دماغى و اورث من ال BaseClass الى هو ال customer و اعمل عليه override

اما فى حاله ال Enquires فى ده فيه صفات اقل من ال Cusomer اصلا و ده لانى ممكن اعمله ديسكونت بس هو لسه مش عميل فمينفعش اسجله فى الداتا بيز مثلا فده بقه اروح متجه معاه لحته انى استخدم فانكشن و فانكشن لأ طب مين الى بيوفر الحركه دى ؟؟ انى اخليه يورث من الانترفيس الانترفيس الى بيحتوى على الفانكشن الى هو عاوزها و ميورسش الى مش هيحتاجه و بالتالى ده خلاص هو مش من نوع customer اصلا و بالتالى لما اجى اضيفه على الكاستومار هيطلعلى compailer error

ده بقه مثال ياباشا يفهمك الوضع ايه

namespace Web.API

{

interface IDiscount

{

double getDiscount(double TotalSales);

}

interface IDataBase

{

void insert();

}

class **Customer : IDataBase , IDiscount**

{

public virtual double getDiscount(double TotalSales)

{

return TotalSales - 15;

}

public virtual void insert()

{

// Database code goes here

}

}

class **SilverCustomer : Customer**

{

public override double getDiscount(double TotalSales)

{

return getDiscount(TotalSales) - 5;

}

public override void insert()

{

// Database code goes here

}

}

class **goldCustomer : Customer**

{

public override double getDiscount(double TotalSales)

{

return getDiscount(TotalSales) - 5;

}

public override void insert()

{

// Database code goes here

}

}

class **Enquiry** : **IDiscount**

{

public double getDiscount(double TotalSales)

{

return getDiscount(TotalSales) - 5;

}

}

class MyProgram

{

public void start()

{

List<Customer> Customers = new List<Customer>();

Customers.Add(new SilverCustomer());

Customers.Add(new goldCustomer());

**Customers.Add(new Enquiry()); // Compailer Error**

**// customer can not refrence to it as it is not Inherit from**

**// customer but inherit from Interfaces**

foreach (Customer o in Customers)

{

o.insert();

}

}

}

}

MB

########################################################################

***I***SP The Interface Segregation Principle

----------------------------------------------

***I***SP The Interface Segregation Principle : دى معناها ان المفروض انى و انا بعمل ال Interface اراعى ان اننا مجبرش حد على وراثه وظايف او سلوك هو مش هيحتاجه ذى مثلا انى ابقى عندى كذا لعبه الكذا لعبه دول فيهم صفات مشتركه ذى ان ليهم سعر و ليهم لون بينما بيختلفوا فى بعض الصفات ذى ان فى لعبه بتمشى و لعبه بتطير فكده انا عندى 4 صفات صفتين مشتركين و صفتين مختلفين يبقى انا المفروض مش اعمل انترفيس واحد احط فيه كل الصفات ال 4 لأ خالص الصح انى اعمل 3 interfaces واحد يبقى بيحتوى على ال 2 صفات المشتركه و 1 يحتوى على صفه المشى و واحد يحتوى على صفه الطير و لما احتاج حاجه اورثها و لما محتجش مورثش و شكرا على كده

و ادى المثال الى يوضح المشكله و هو انى عامل انترفيس فيه كل حاجه فلما جيت اورث منه ورثت حتى الحاجات الى انا مش عاوزها

namespace Web.API

{

interface IToy

{

void setPrice(int price);

void setColor(String color);

void move();

void fly();

}

class ToyHouse : IToy

{

int price;

String color;

public void setPrice(int price)

{

this.price = price;

}

public void setColor(String color)

{

this.color = color;

}

public void move()

{

throw new Exception("Not allowed");

}

public void fly()

{

throw new Exception("Not allowed");

}

}

}

و كان حل المشكله انى اعمل للى انا مش عاوزهم دول interfaces لوحدهم عشان استخدمهم لما احب و مستخدمهمش وقت ما احب برضوا لما معوزهمش يعنى من الاخر

و ادى الحل فى الكود اهه يا دهلكه و الى فيه عامل 3 انترفيسيس و الى هستخدمه بورث منه و الى مش هستخدمه بورثش منه حاجه عشان اريح دماغى و الكود ينضف ياعم مبقاش بكتب حاجه مش هستخدمها على الفاضى يعنى

namespace Web.API

{

interface IToy

{

void setPrice(int price);

void setColor(String color);

}

interface IMovable

{

void move();

}

interface IFlyable

{

void fly();

}

class ToyHouse : IToy

{

int price;

String color;

public void setPrice(int price)

{

this.price = price;

}

public void setColor(String color)

{

this.color = color;

}

}

class ToyPlane : IToy, IMovable, IFlyable

{

int price;

String color;

public void setPrice(int price)

{

this.price = price;

}

public void setColor(String color)

{

this.color = color;

}

public void move()

{

//code related to moving plane

}

public void fly()

{

// code related to flying plane

}

}

}

MB

########################################################################

***D***IP The Dependency Inversion Principle

----------------------------------------------

DIP The Dependency Inversion Principle : بيقولك هنا ان انت طالما عندك كلاسات ابناء و كلاسات اباء الى هما low-level و high-level بيقولك انت بقه ياباشا متحليش اعتماد الابناء على الاب ولا اعتماد الاب على الابناء لأ شفولك layer جديده بينهم تعزلهم عن بعض لان الاتنين عاملين ذى سلوك الكهرباء لو قربوا من بعض هنموت كلنا هههههههه طب ايه هى اللاير الى ممكن اخليهم الاتنين يعتمدوا عليها و تبقى حلقه الوصل بينهم ؟؟  
abstract Class او الاحسن interface   
يبقى انا دلوقتى هعمل interface و الانترفيس ده هيبقى حلقه الوصل بين الكلاس الاب و الابناء فكده اعزل بينهم تماما و يبقى الاب بيورث من الانترفيس و العيال بيورثوا من الانترفيس فكده هما مبقاش ليهم اى علاقه بتربيطهم ببعض غير ال Interface

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace Web.API

{

interface IServiceLocator { T GetService<T>(); }

interface IServiceA : IServiceLocator {}

interface IServiceB : IServiceLocator { }

interface IServiceC : IServiceLocator { }

class ServiceA : IServiceA

{

public T GetService<T>()

{

throw new NotImplementedException();

}

}

class ServiceB : IServiceB

{

public T GetService<T>()

{

throw new NotImplementedException();

}

}

class ServiceC : IServiceC

{

public T GetService<T>()

{

throw new NotImplementedException();

}

}

class ServiceLocator : IServiceLocator

{

// map that contains pairs of interfaces and

// references to concrete implementations

private IDictionary<object, object> services;

internal ServiceLocator()

{

services = new Dictionary<object, object>();

// fill the map

this.services.Add(typeof(IServiceA), new ServiceA());

this.services.Add(typeof(IServiceB), new ServiceB());

this.services.Add(typeof(IServiceC), new ServiceC());

}

public T GetService<T>()

{

try

{

return (T)services[typeof(T)];

}

catch (KeyNotFoundException)

{

throw new ApplicationException("The requested service is not registered");

}

}

}

}

طيب هو بيقولك بقه الموضوع ده هتلاحظه فين بقه انك بتستخدمه حلو كده ؟؟  
هتلاحظه فى ال Dependancy injection و انا بعمله هتلاحظ انى بعمل كلاس واحطه فيه كل اللوجيك و اسميه ServiceA و اروح اعمله انترفيس و اسميه IServiceA و يبقى ده اللاير الى هتبقى حلقه الوصل فمن هنا انا اقدر بال interface اروح اعمل ريجيستر للسيرفيس و اقدر اروح اعمل بيها Inject فى الكونستراكتور و اقدر برضوا اخلى السيرفيس تورث منها عشان تعمل امبليمينتيشن لنفسها و خد بالك طالما بتعمل سيرفيس و هتعمل ريجيستريشن للسيرفيس دى عن طريق الانترفيس يبقى بلاش تخلى اتنين كلاس يورثوا من الكلاس دى عشان ساعتها و هو بيسجل السيرفيس هيقولك هى الانترفيس دى اشغلك انتهى امبلمينتيشن بقه ؟؟ ساعتها هتبقى مضطر تستخدم الكلاس العادى مش الانترفيس بتاعه عشان الكود يعرف يفرق ده مين الى هيتعامل معاه دلوقتى

طبعا ال Dependency Injection دى حاجه عظيمه لانها خلتنى استخدم instance فى الكلاس براحتى و مضطرش كل شويه اغير فى الاوبجيكت و انا بناديه حتى لو كل شويه بزود بارميتارز فى الكلاس هو ولا كانك عدلت فيه حاجه انما لو مكنتش بستخدم الدبندينسى انجيكشن كان زمانك بتتفشخ فى البروجيكت كله تعديلات كل اما يزود حاجه فى الكونستراكتور او تبقى مضطر تعمل مليون كونستراكتور بقه عشان يظبط معاك فى اى حته و ده طبعا هبل يعنى

ادى انترفيس السيرفيس

namespace Web.API.ControllersServices

{

public interface IWeatherForecastService

{

Cities AddCity([FromBody] Cities VM);

IEnumerable<Cities> GetAllCities();

}

}

ادى كلاس السيرفيس ذى ما انت شايف الامبلمينتيشن هنا و خد بالك ان السيرفيس الاساسيه ورثت من الانترفيس عشان يتحقق العزل بين الكلاس و الكود

namespace Web.API.ControllersServices

{

public class **WeatherForecastService** : IWeatherForecastService

{

private IRepository<Cities> \_Citiesepository;

public WeatherForecastService(IRepository<Cities> Citiesepository)

{

\_Citiesepository = Citiesepository;

}

public IEnumerable<Cities> GetAllCities() => \_Citiesepository.GetAll();

public Cities AddCity([FromBody] Cities VM)

{

Cities item = new Cities();

item.ID = Guid.NewGuid();

item.CityName = VM.CityName;

item.CityCountry = VM.CityCountry;

item.CityDescription = VM.CityDescription;

item.Population = VM.Population;

\_Citiesepository.Insert(item);

return item;

}

}

}

ادى تسجيل السيرفيس فى المشروع و ذى ما انت شايف بما ان السيرفيس وارثه من الانترفيس اذا انا اقدر اعرف السيرفيس و اسجلها فى المشروع باسم ال interface عشان يبقى مش بستخدم الكلاس بشكل مباشر

public void ConfigureServices(IServiceCollection services)

{

var options = new DbContextOptionsBuilder<SPSDBContext>().

UseInMemoryDatabase(databaseName: "SPSDB").Options;

using (var context = new SPSDBContext(options))

{

// add service here

services.AddScoped<SPSDBContext>();

services.AddTransient<IWeatherForecastService, WeatherForecastService>();

services.AddScoped(typeof(IRepository<>), typeof(Repository<>));

services.AddAutoMapper(AppDomain.CurrentDomain.GetAssemblies());

services.AddControllers();

}

}

كده انا بقوله اول ما تشوف الانترفيس ده يبقى كانه شاف السيرفيس تماما

و فى الكنترولار بقه دى طريقه كتابه ال DI الاساسيه ياكبير الى هو بعرفها فى الكونسرتاكتور و اظبطها كده

public class WeatherForecastController : ApiBaseController

{

private readonly IWeatherForecastService \_CTRL\_Service;

private readonly IMapper \_mapper;

public WeatherForecastController(ILogger<WeatherForecastController> logger , IWeatherForecastService CTRL\_Service , IMapper mapper)

:base(logger) {

\_CTRL\_Service = CTRL\_Service;

\_mapper = mapper;

}

[HttpGet]

public IActionResult Get()

{

Cities Data = \_CTRL\_Service.GetAllCities().FirstOrDefault();

CitiesVM Item = \_mapper.Map<CitiesVM>(Data);

return Ok(Item);

}

[HttpPost]

public IActionResult Post(Cities VM)

{

Cities item = \_CTRL\_Service.AddCity(VM);

return Ok(item);

}

}

طبعا استخدام الانترفيس مكان الكلاس حاجه جميله جدا جدا لانها سيكيورد جدا و بتحميك من ان الداتا يتشاف منها حاجه لان كل حاجه انت بتكلم انترفيس يعنى انسى ان حد يعرف يجيب داتا بالدهلكه كده ولا غصبا عنك