C# programming language

Majeure big data & analytics (M1)

ECE Paris, January-March 2020

V 1.0 (30 of March 2020)

Document history

V 0.7	27 of January 2020
V 0.6	20 of January 2020
V 0.5	20 of January 2020
V 0.4	20 of January 2020
V 0.3	13 of January 2020
V 0.2	13 of January 2020
V 0.1	6 of January 2020

V 1.0	30 of March 2020
V 0.9	24 of February 2020
V 0.8	3 of February 2020

Personal background

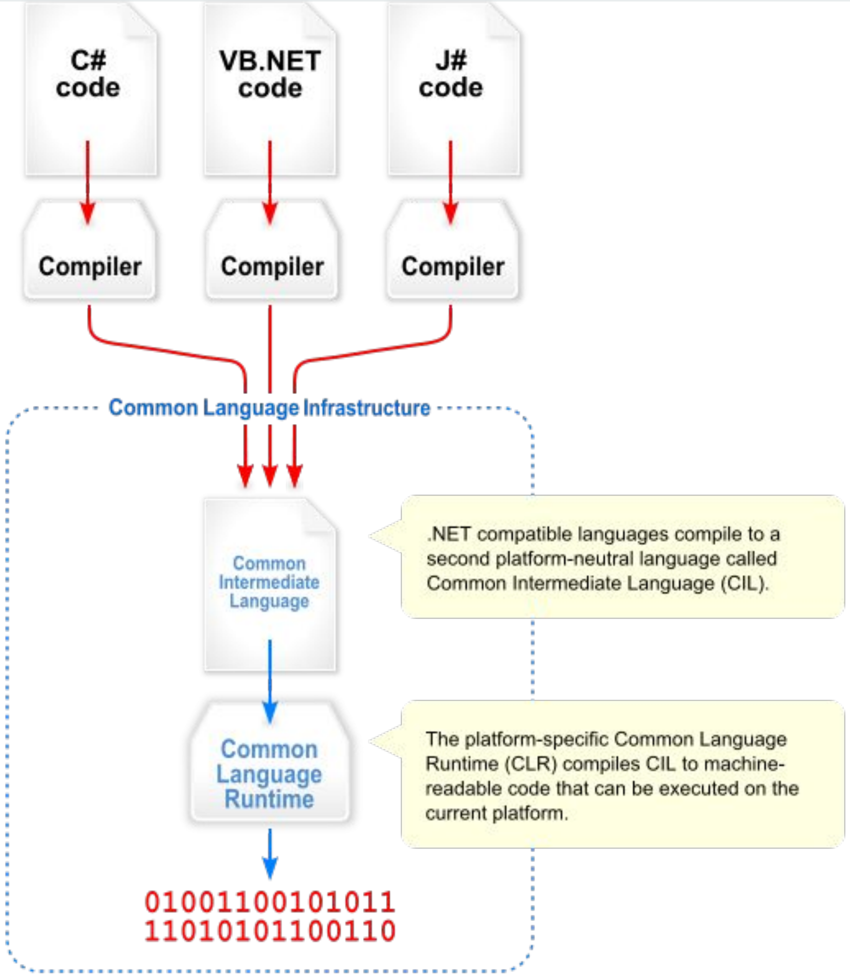
- Benoît Prieur, Soarthec (own company)
- (.Net) Freelance Software developer for years (MCP), C# & VB.NET
- 2 books in French about .Net:
  - *Programmation en C# - Préparation aux certifications MCSA - Examen 70-483* (2018)
  - *WPF - Développez des applications structurées (MVVM, XAML...)* (2017)
- Practical course on quantum computing gave at ECE Paris (2019)

Native *vs* Managed

- C/C++ building (compiling/linking) => Win32 application (binary)
- Virtual machine
  - JVM, Java
  - CLR (Common Language Runtime) .Net
    - C#/VB.NET => Common Intermediate Language (CIL) => Binary (Assembly, Executable)

CIL & CLR architecture

Credit: Jarkko Piiroinen [Public domain], Wikimedia Commons


C# code
VB.NET code
J# code
Compiler
Compiler
Compiler
Common Language Infrastructure
Common Intermediate Language
.NET compatible languages compile to a second platform-neutral language called Common Intermediate Language (CIL).
Common Language Runtime
The platform-specific Common Language Runtime (CLR) compiles CIL to machine-readable code that can be executed on the current platform.
01001100101011
11010101100110

MSIL example

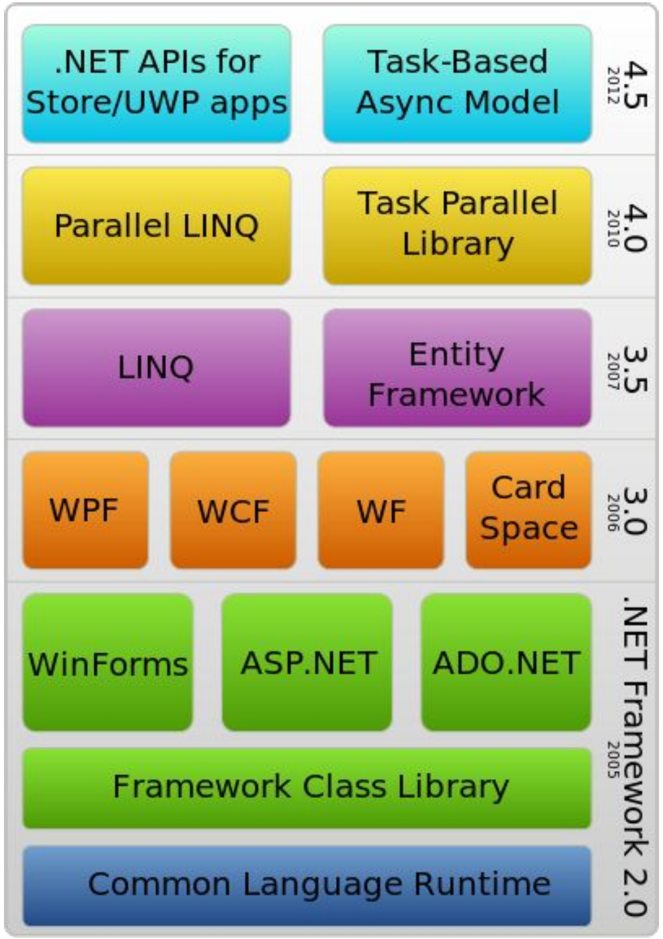
```
using System;

public class Hello
{
        public static void Main()
        {
                Console.WriteLine("Hello
World");
        }
}
```

=>

```
.assembly Hello {}
.assembly extern mscorlib {}
.method static void Main()
{
   .entrypoint
   .maxstack 1
   ldstr "Hello, world!"
   call void [mscorlib]System.Console::WriteLine(string)
   ret
}
```

.Net history

- Beta Version (2001)
- Version 1 (2002)
- Version 3.5 (2008), LINQ (*Language Integrated Query*)
- Version 4.5 (2012), asynchronism
- Version 4.6 (2015), Linux support



.Net Framework

- Composed by *namespaces* including *classes*.
- Namespaces and classes (similarities with Java language).
- About 10.000 classes included in specialized namespaces.
- Every class inherits from *Object* class (directly or not).

Beginning with C# language and VS Studio

```
using System;

public class Main
{
          public static void Main()
          {
             Console.WriteLine("Hello World");
          }
}
```

***Hello world* program:**

- Create a new project.
- Keyword ***using*** for referencing a namespace.
- *Console*, ***static*** class included in *System*.
- VS Studio: *Quick action and refactoring*.
- ***System.Diagnostics.Debug***
- Add a breakpoint.
- VS Studio: debug *vs* release.

C# syntax introduction

Deeply inspired from C/C++:

- Block definition with brackets: **{ … }**
- Statement separator (end of statement): **;**
- Flow control, conditions, loops: **if switch for while**
- Arithmetic operators: **+ - * / % ^**
- Logical operators: **& && | ||**

Before oriented object programming in C#

Visibility:

- *public*, no restriction
- *protected*, limited to class and derivatives
- *internal*, limited to the current assembly
- *private, limited to the current instance*

Instance construction:

- Default constructor
- Explicit constructor
- Copy constructor

Instance destruction:

- Garbage collector, automatically called at the end of scope.
- Usage of ***Dispose*** for non-managed resources.

C# and oriented object programming (1)

Encapsulation:

```
class Car
{
      protected string color;
      protected int numberDoors;

      public Car(string c, int n)
      {
         color = c;
         numberDoors = n;
      }
```

```
      public void start() {
         //implementation
      }

      public void stop() {
         //implementation
      }
}
```

```
static void Main(string[] args){
   Car mycar = new Car("red", 5);
}
```

C# and oriented object programming (2)

Inheritance:

- Possibility to declare a visibility.
- ***abstract*** class can be used (cannot be instantiated).
- Keyword ***sealed***: class cannot be derived.
- A method should be ***virtual*** (or *abstract*) to allow overriding (keyword ***override***)
- The base class behavior can be called with ***base*** keyword.

```
abstract class Vehicle
{
  public string Brand { get; set; }
}

class Car : Véhicule
{
 ...
}
```

C# and oriented object programming (3)

Polymorphism:

- Inheritance of more than class (can be from classes or interfaces).
- Precisions about interfaces.

```
interface INavigation
{
  void navigate();
}

class Car : Vehicle, INavigation
{
  ...

  public void navigate()
  {
    // implementation
  }
}
```

Value type *vs* reference type

- A value type is stored directly on the stack.
- A reference type is stored on the heap.
- **In C#, value types are:**
    - *struct* (structure)
    - *enum* (enumeration)
    - Numeric types: *int*, *float*, *decimal*, *bool* etc.
- **in C#, reference types are kind of pointers:**
    - *class*
    - *interface*
    - *delegate* (a delegate is an object which refers to a method).
    - Types like *string*, *dynamic*, *object.*

Define a C# enumeration

```
enum DAYS : int { MONDAY = 1,
    TUESDAY,
    WEDNESDAY,
    THURSDAY,
    FRIDAY,
    SATURDAY,
    SUNDAY }
```

Attribute [FLAGS]:

```
using System;

namespace Example
{
  [Flags]
  enum COLOR : int
  {
     RED = 1,
     GREEN = 2,
     BLUE = 4
   };
}
```

```
// Main
COLOR mycolor =
COLOR.RED | COLOR.BLUE;
string s = mycolor.ToString();
console.WriteLine("Current
color : " + s);
```

Structure in C#

- Public visibility by default.
- No empty constructor.
- No inheritance.

```
struct Coord
{
  public float latitude, longitude;
  public Coord(float lat, float lon)
  {
    latitude = lat;
    longitude = lon;
  }
}
```

C# class accessors

```
classe  Foo
{
    public int Data { get; set; }
}
```

<=>

```
classe  Foo
{
    private int data;
    public int Data
    {
            get { return data; }
            set { data = value; }
    }
}
```

Nullable type

- Value types cannot be ***null***.
- Usage of the operator **?** to declare nullable value types.

```
int? ii = 42;
double? dd = 42.42;
bool? bb = null;
char? cc = '42';
double?[] tt = double int?[10];
```

```
int? ii = 42
if (x.HasValue)
{
   System.Console.WriteLine(ii.Value);
}
else
{
   System.Console.WriteLine("No value");
}
```

Parameters modifiers in functions/methods

- Value types are passed by value (copy).
- ***ref*** modifier
  - Can be modified.
  - Must be initialized.
- ***out*** modifier
  - Can be modified.
  - Cannot be initialized.

Constant variables/attributes

- Two keywords:
  - ***const***, must be initialized.
  - ***readonly***, initialization is not mandatory.

Dev 1

Write a C# Sharp program to find the sum of first n natural numbers.
The user gives the number n.

Expected Output :

```
        The first 10 natural number is:
1 2 3 4 5 6 7 8 9 10
        The Sum is : 55
```

```
// Help
string str = Console.ReadLine();
int i = double.Parse(str);
```

- *overrid*
- *e*

Dev 2

Provide a scalar product calculation for 2D, 3D

- *Define an abstract class Vector including an abstract method Scalar.*
- *Define two classes Vector2D and Vector3D which inherit from Vector.*
  - *Each class includes a override method Scalar.*
  - *Dimension (2 or 3) can be defined in the base class*

Dev 2 (a code solution)

```
using System;

namespace ConsoleApp1
{
    public abstract class Vector
    {
        int dimension;
        public abstract double scalar(Vector V2);

        public Vector(int d)
        {
            dimension = d;
        }
    }

    partial class Program
    {
        static void Main(string[] args)
        {
            Vector2d v = new Vector2d(5.0, 3.0);
            Vector2d w = new Vector2d(-5.0, -3.0);
            Console.WriteLine(v.scalar(w).ToString());
        }
}
```

```
public class Vector2d : Vector
    {
        double x;
        double y;

        public double X { get => x; set => x = value; }
        public double Y { get => y; set => y = value; }

        public override double scalar(Vector v)
        {
            Vector2d v2 = (Vector2d)v;
            return this.x * v2.X + this.y * v2.Y;
        }

        public Vector2d(double xx, double yy) : base(2)
        {
            this.X = xx;
            this.Y = yy;
        }
    }
}
```

Type casting and conversion

- Numeric types:
  - ***TryParse***, includes a *try...catch* management.
  - **Parse**. no exception management.
- Type ***string*** (reference type):
  - ***ToString()*** when available.
- Casting:
  - *(ExampleType)obj* => can throws an exception.

String in C#

- Is a reference type (address + size).
- There exists a class ***String*** which provides methods:
    - *SubString, StartsWith, EndWith etc..*
- Type string is ***immutable***.
- Another type is ***mutable*** in C#: ***StringBuilder***.

Exception management in C#

```
try {
    // ...
}
catch (System.Exception e) {
     // ...
    throw new Exception();
}
finally {
    // ...
}
```

Interfaces in C#, syntax

```
Interface ICar {
	void Start();
	void Stop();
}

class Car : ICar {
	void Start() {}
	void Stop() {}
}
```

Containers and data structure (1): arrays

- Arrays are like in C++.
- It exists a class ***Array*** in C#: set of methods.

```
// Single dimension
int[] a = new int[] { 1, 2 };

// 2-dimensions
int[,] b = new int[,] { { 1, 2 }, { 3, 4 } };
```

```
Array.Reverse(a);

Array.LastIndexOf(a, 1);

Array.Sort(a);

// etc.
```

Containers and data structure (2): ICollection implementation

- Data structure implementing ICollection.
- For example: ***ArrayList***, ***Queue***, ***Stack***, ***HashTable***, ***SortedList***, ***Dictionnary***.
- Generic types: ***Dictionary<T>***.
- Notion of iterator: ***for each***.

Containers and data structure (3): example with Dictionary<T>

```
class Car
{
   public int ID { get; set; }
   public int NbDoors { get; set; }
   public int Year { get; set; }

   public Car(int id, int nb, int yyyy)
   {
      ID = id;
      NbDoors = nb;
      Year = yyyy;
   }
}
```

```
class Program
{
   static void Main(string[] args)
   {
      Dictionary<int, Car> dict = new Dictionary<int, Car>();

      Car car1 = new Car(12345, 5, 2006);
      dict.Add(car1.ID, car1);

      Car car2 = new Car(21345, 3, 2005);
      dict.Add(car2.ID, car2);

      foreach(KeyValuePair<int, Car> entry in dict)
      {
         Console.WriteLine(entry.Key.ToString() + ": " + entry.Value.Year.ToString();
      }
   }
}
```

Reflection in C#

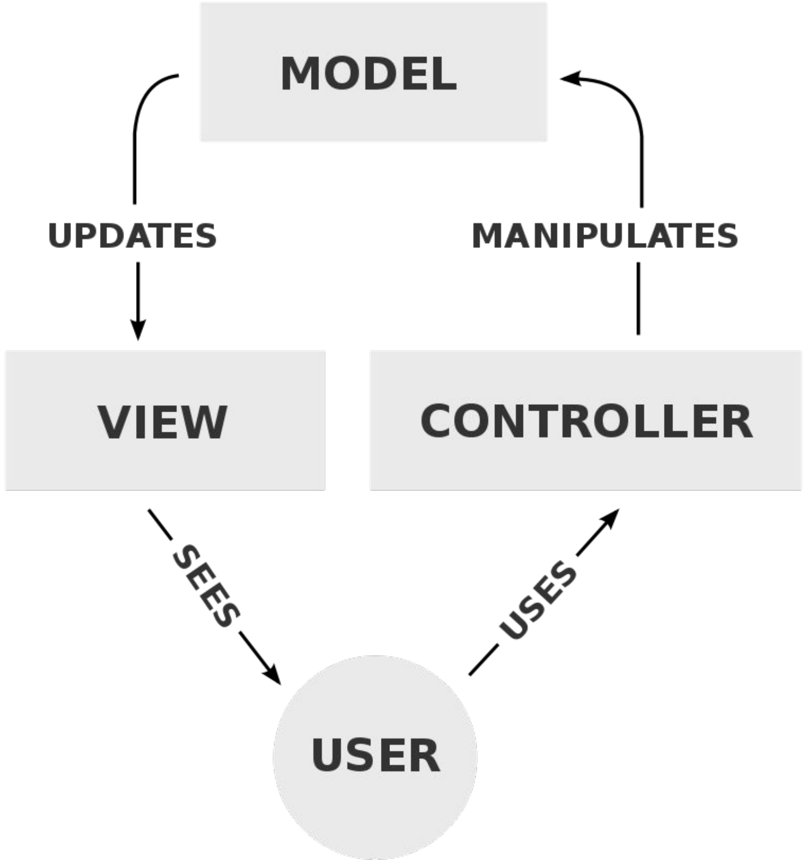
- Capability to describe modules, assemblies, types.
- Get metadata (classe) from an instance.

```
int i = 42;
Type type = i.GetType();
Console.WriteLine(type);
```

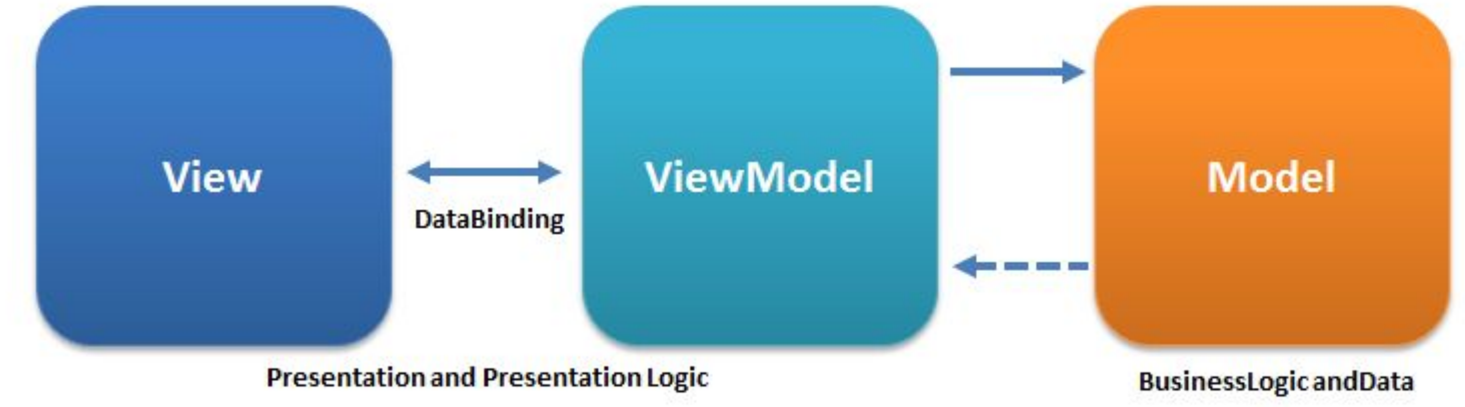
```
Type myType =(typeof(MyTypeClass));

 MethodInfo[] myArrayMethodInfo =
myType.GetMethods(BindingFlags.Public|BindingFlags.Instance|BindingF
lags.DeclaredOnly);
```

Model–view–controller (ASP.NET MVC)

Credit: RegisFrey [Public domain]



Model–view–viewmodel (WPF)



Credit: Ugaya40 [CC BY-SA (https://creativecommons.org/licenses/by-sa/3.0)]

XAML (Extensible Application Markup Language)

- XML extension.
- C# code-behind (in View itself).

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

  <TextBlock>Hello, world!</TextBlock>

</Canvas>
```

MVVM and WPF: DataContext & Binding

- Windows Presentation Foundation.
- Declaring a DataContext on the View side.
- Binding between View (V) and View-Model (VM).

WPF & XAML: DataContext

```
public partial class MainWindow : Window
{
   public MainWindow()
   {
      InitializeComponent();
      this.DataContext = new VM();
   }
}
```

WPF & XAML: *INotifyPropertyChanged*

```
class VM : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;
    protected virtual void OnPropertyChanged(string propertyName)
    {
        if (this.PropertyChanged != null)
        {
            this.PropertyChanged(this, new
PropertyChangedEventArgs(propertyName));
        }
    }
```

```
private int myValue;
public int MyValue
{
    get { return this. myValue;  }
    set
    {
        this. myValue = value;
        OnPropertyChanged("MyValue");
    }
}
```

WPF & XAML: Binding, first example

- *<TextBlock Text="{Binding MyValue}" Width="500" Height="100" />* (XAML code)
- Obtaining updated value:
  - In VM from V.
  - In V from VM.
  - Both (Bidirectional).

WPF & XAML: binding collections, *INotifyCollectionChanged*

- With collection, Binding must monitor every update of every value but also monitors Add/Remove inside the collection itself.
- *INotifyCollectionChanged*
- Objects ever implementing *INotifyCollectionChanged*
    - *ObservableCollection*
    - *DataView* (easily defining from a DataTable).

A word about *DataSet, DataTable, DataRow*

- ADO .NET

```
static DataTable GetTable()
{
   DataTable table = new DataTable();
   table.Columns.Add("ID", typeof(int));
   table.Columns.Add("Name", typeof(string));

   table.Rows.Add(1, "John Doe");
   table.Rows.Add(2, "Benoît Prieur");

  DataView dv = new DataView(table);

  return dv;

}
```

Dev 3: a first WPF client

- *Teams for the final project.*
- *Countries and capital cities stored in a CSV file.*
- *Model in charge to read this file and provides data.*
- *Obligation to respect MVVM.*
- *On the view:*
    - *A ComboBox (or a autocomplete TextBox) in charge to search for a country.*
    - *Displaying the associated capital city.*

A solution here => https://github.com/benprieur/CSharp-WPF-20200113

WPF Controls (1)

- Web site in French: https://www.wpf-tutorial.com/
  - From https://www.wpf-tutorial.com/fr/14/les-controles-de-base/le-controle-textblock/

WPF Controls (2)

- TextBlock
- Label
- TextBox (autocomplete)
- Button
- CheckBox
- Image (very important for the final project)
- ComboBox: https://www.wpf-tutorial.com/list-controls/combobox-control/

WPF Controls (3) - Layout controls

- WrapPanel, DockPanel, StackPanel: https://www.wpf-tutorial.com/fr/25/panels/le-controle-wrappanel/
- Grid: https://www.wpf-tutorial.com/fr/28/panels/la-grid/

WPF Controls (4) - ListView

- https://www.wpf-tutorial.com/listview-control/simple-listview/
- https://www.wpf-tutorial.com/listview-control/listview-data-binding-item-template/

WPF Controls (5) - Styles

- https://www.wpf-tutorial.com/styles/using-styles/

Final Dev - Option 1 - OpenFoodFacts

- An example of category in French:
  - https://fr.openfoodfacts.org/categorie/pains.json
- List of categories:
  - https://fr.openfoodfacts.org/categories.json
- *Display a list with results including for each product: image, ingredients etc. Pagination is a plus ("/23.json" for the page number 23)*
- *Textual search:* https://fr.openfoodfacts.org/cgi/search.pl?search_terms=nutella&search_simple=1&action=process&json=1

Final Dev - Option 2 - Wikidata & Postal code

- SPARQL Request:
    - https://w.wiki/Foq (communes)
    - https://w.wiki/Fqa (NBA players)
- Display a list of communes related to this postal code. for each commune display image, area, population+date, maximum of available data in tuning SPARQL request.
- Teams
- Use action=wbgetentities and format=json; example

Asynchronous call in C#: await & async

```
static async void ExampleAsync()
{
   int t = await Task.Run(() => FunctionAsyncCall());
   Console.WriteLine("Compute: " + t);
}

static int FunctionAsyncCall()
{
   // Long treatment
   return size;
}
```

Http request in C#

```
static HttpClient client = new HttpClient();
 static async Task<int> GetProductAsync(string path)
 {
    Result res = null;
    HttpResponseMessage response = await client.GetAsync(path);
    //if (response.IsSuccessStatusCode)
    return 0;
 }
```

JSON Serialization/Deserialization

- Use NewtonSoft :
  - https://www.newtonsoft.com/json
  - Install with NuGet (Visual Studio)

Unit tests in C#

- Create a new Test project into the solution.
- A word about TDD.
- A word about Mocks.
- Define a set of unit tests able to be automatized.
- The notion of test coverage.
- *Final project: at least one unit test.*

Overview of ASP MVC .Net

- Web solution by Microsoft.
- Implements MVC, Model - View - Controller.
  - Derivation from *Controller* class.
  - *ActionResult* for routing.
- *Razor* language used in View.
- IIS but Apache too, etc.
- Extension Ajax MVC.

Tutorial in French

Overview of ASP MVC .Net (2)

```
public class Person
{
   public string Surname { get; set; }
   public string FirstName { get; set; }
   public DateTime BirthDate { get; set; }
}
```

Model

```
public class HomeController : Controller
{
   public ActionResult Index()
   {
      Person person = new Person
      {
         Surname = "Doe",
         FirstName = "John",
         BirthDate = new DateTime(1970, 1, 2)
      };

      return View(person);
   }
}
```

Controller

View (Razor)

```
@* Commentaire *@
@model Projet.Models.Person

<h2>Salut @Model.FirstName !</h2>

<div>
   <h5>Details:</h5>
   <div>
      @Html.LabelFor(m => m.Surname)
      @Html.DisplayFor(m => m.Surname)
   </div>
   <div>
      @Html.LabelFor(m => m.FirstName)
      @Html.DisplayFor(m => m.FirstName)
   </div>
   <div>
      @Html.LabelFor(m => m.BirthDate)
      @Html.DisplayFor(m => m.BirthDate)
   </div>
</div>
```

Overview of Entity Framework

- What is an ORM?
- Doctrine (PHP), SQLAlchemy (Python).
- Object *context* which Implements *DbContext*.

Tutorial in French

```
var team = new FootballTeam{ Name = "Ain Sud Foot", Country = "France" };
context.Add<FootballTeam>(team);
context.SaveChanges();
```