Initialisation :

> Pour initier un nouveau projet de type API web dans un répertoire ApiUniversity existant

```
git init
dotnet new webapi -n ApiUniversity
dotnet new gitignore
```

> Ajouter les packages relatifs à Entity framework

```
dotnet tool install --global dotnet-ef --version 6.0.0
dotnet add package Microsoft.EntityFrameworkCore.Design --version 6.0.0
dotnet add package Microsoft.EntityFrameworkCore.Sqlite --version 6.0.0
```

Créer contexte :

```
using Microsoft.EntityFrameworkCore;
using ApiUniversity.Models;

namespace ApiUniversity.Data;

public class UniversityContext : DbContext
{
    public DbSet<Student> Students { get; set; } = null!;
    public DbSet<Course> Courses { get; set; } = null!;
    public DbSet<Enrollment> Enrollments { get; set; } = null!;
    public DbSet<Department> Departments { get; set; } = null!;
    public DbSet<Instructor> Instructors { get; set; } = null!;

    public string DbPath { get; private set; }

    public UniversityContext()
    {
        // Path to SQLite database file
        DbPath = "ApiUniversity.db";
    }

     // The following configures EF to create a SQLite database file
locally
       protected override void OnConfiguring(DbContextOptionsBuilder
options)
    {
```

```
        // Use SQLite as database
        options.UseSqlite($"Data Source={DbPath}");
        // Optional: log SQL queries to console
                        //options.LogTo(Console.WriteLine,    new[]    {
DbLoggerCategory.Database.Command.Name }, LogLevel.Information);
    }
}
```

Faire 1ère migration car contexte :

> *Pour créer une migration*
> `dotnet ef migrations add InitialCreate`

> *Pour appliquer une migration (création de la db)*
> `dotnet ef database update`

Faire le seed data :

```
namespace ApiUniversity.Data;
public static SeedData{
      public static void Init()
      {
        using var context = new UniversityContext();

        // Look for existing content
        if (context.Students.Any())
        {
            return;   // DB already filled
        }

        // Add students
        Student carson = new()
        {
            FirstName = "Alexander",
            LastName = "Carson",
            Email = "alexander.carson@test.com",
            EnrollmentDate = DateTime.Parse("2016-09-01"),
        };
      context.Students.Add(carson);
      context.SaveChanges();
      }
}
```

A mettre dans program.cs :

```
using System.Text.Json.Serialization;
using ApiUniversity.Data;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers()
// Prevent circular references when serializing objects to JSON
    .AddJsonOptions(x =>
                            x.JsonSerializerOptions.ReferenceHandler    =
ReferenceHandler.IgnoreCycles
    );

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

var app = builder.Build();

// Seed data into DB
SeedData.Init();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();
```

Trucs utiles :

https://localhost:7044/swagger/index.html

Si y’a pb de certif :

> Si vous rencontrez des problèmes de certifications, essayer d’exécuter la commande suivante: `dotnet dev-certs https`

Les requêtes :

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using ApiUniversity.Data;
using ApiUniversity.Models;
[ApiController]
[Route("api/document")]
public class DocumentController : ControllerBase
{
    private readonly MediathequeContext _context;

    public DocumentController(MediathequeContext context)
    {
        _context = context;
    }

    // GET: api/document
    [HttpGet]
    public async Task<ActionResult<IEnumerable<GetDocumentDTO>>>
GetDocuments()
    {
        var documents = _context.Documents.Select(x => new
GetDocumentDTO(x));
        return await documents.ToListAsync();
    }

    // GET: api/document/id
    [HttpGet("{id}")]
    public async Task<ActionResult<GetDocumentDTO>> GetDocument(int id)
```

```
    {
        var document = await _context.Documents.SingleOrDefaultAsync(t
=> t.Id == id);
        if (document == null)
            return NotFound();
        return new GetDocumentDTO(document);
    }

    //modif
    // POST: api/document
    [HttpPost]
    public async Task<ActionResult<DocumentDTO>>
PostDocument(DocumentDTO documentDTO)
    {
        Document document = new(documentDTO);
        _context.Documents.Add(document);
        await _context.SaveChangesAsync();

        return CreatedAtAction(
            nameof(GetDocument),
            new { id = document.Id },
            new GetDocumentDTO(document)
        );
    }

    // PUT: api/document/id
    [HttpPut("{id}")]
    public async Task<IActionResult> PutDocument(int id, DocumentDTO
documentDTO)
    {
        Document document = new Document(documentDTO, id);
        _context.Entry(document).State = EntityState.Modified;

        try
        {
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!_context.Documents.Any(x => x.Id == id))
                return NotFound();
            else
                throw;
```

```
        }

        return NoContent();
    }

    // DELETE: api/document/id
    [HttpDelete("{id}")]
    public async Task<IActionResult> DeleteDocumentItem(int id)
    {
        var document = await _context.Documents.FindAsync(id);

        if (document == null)
            {
             return NotFound();
            }
        _context.Documents.Remove(document);
        await _context.SaveChangesAsync();

        return NoContent();
    }
}
```

Si on a une table qui en lie 2 autres :

```
using ApiUniversity.Data;
using ApiUniversity.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace ApiUniversity.Controllers;

[ApiController]
[Route("api/enrollment")]
public class EnrollmentController : ControllerBase
{
    private readonly UniversityContext _context;

    public EnrollmentController(UniversityContext context)
    {
        _context = context;
    }
```

```
    // GET: api/enrollment/5
    [HttpGet("{id}")]
    public async Task<ActionResult<DetailedEnrollmentDTO>>
GetEnrollment(int id)
    {
        // Find student and related list
        // SingleAsync() throws an exception if no student is found
(which is possible, depending on id)
        // SingleOrDefaultAsync() is a safer choice here
        var enrollment = await _context.Enrollments.Include(x =>
x.Course).Include(x => x.Student).SingleOrDefaultAsync(t => t.Id ==
id);

        if (enrollment == null)
        {
            return NotFound();
        }

        return new DetailedEnrollmentDTO(enrollment);
    }

    // POST api/enrollment
    [HttpPost]
    public async Task<ActionResult<DetailedEnrollmentDTO>>
PostEnrollment(EnrollmentDTO enrollmentDTO)
    {
        Enrollment enrollment = new(enrollmentDTO);

        _context.Enrollments.Add(enrollment);
        await _context.SaveChangesAsync();

        // Complete the enrollment object by adding the related objects
        enrollment.Course = _context.Courses.First(c => c.Id ==
enrollment.CourseId);
        enrollment.Student = _context.Students.First(s => s.Id ==
enrollment.StudentId);

        return CreatedAtAction(nameof(GetEnrollment), new { id =
enrollment.Id }, new DetailedEnrollmentDTO(enrollment));
    }
}
```

Requetes Sans DTO :

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Entrainement.Data;
[ApiController]
[Route("api/todo")]
public class TodoController : ControllerBase
{
    private readonly TodoContext _context;


    public TodoController(TodoContext context)
    {
        _context = context;
    }


    // GET: api/Todo
    [HttpGet]
    public async Task<ActionResult<IEnumerable<Todo>>> GetTodo()
    {
        var todos = _context.Todos;
        return await todos.ToListAsync();
    }


    // GET: api/document/id
    [HttpGet("{id}")]
    public async Task<ActionResult<Todo>> GetTodo(int id)
    {
        var todo = await _context.Todos.SingleOrDefaultAsync(t => t.Id
== id);
        if (todo == null)
            return NotFound();
        return todo;
    }


    //modif
    // POST: api/document
    [HttpPost]
    public async Task<ActionResult<Todo>> PostTodo(Todo todo)
```

```
    {
        _context.Todos.Add(todo);
        await _context.SaveChangesAsync();


        return CreatedAtAction(
            nameof(GetTodo),
            new { id = todo.Id },
            todo
        );
    }


    // PUT: api/document/id
    [HttpPut("{id}")]
    public async Task<IActionResult> PutTodo(int id, Todo todo)
    {
        _context.Entry(todo).State = EntityState.Modified;
        try
        {
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!_context.Todos.Any(x => x.Id == id))
                return NotFound();
            else
                throw;
        }


        return NoContent();
    }


    // DELETE: api/document/id
    [HttpDelete("{id}")]
    public async Task<IActionResult> DeleteTodo(int id)
    {
        var document = await _context.Todos.FindAsync(id);


        if (document == null)
```

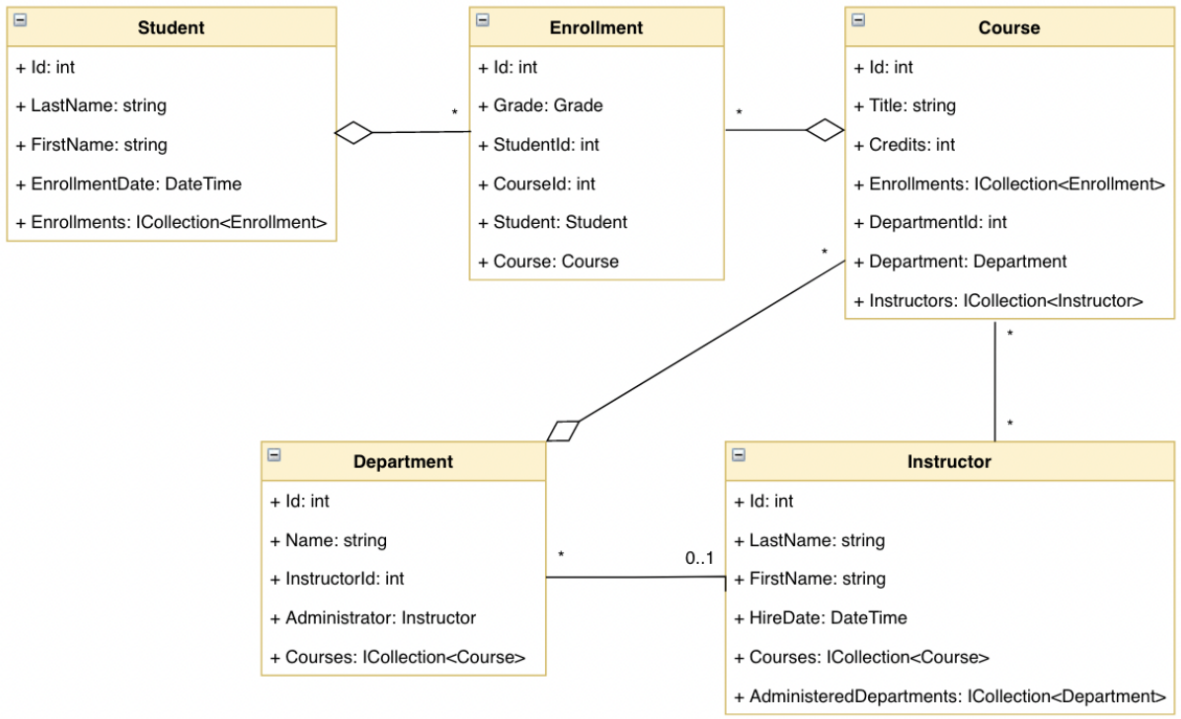
```
        {
            return NotFound();
        }
        _context.Todos.Remove(document);
        await _context.SaveChangesAsync();


        return NoContent();
    }
}
```

Question de cours

Qu’est ce qu’un diagramme de classe ?

Un diagramme de classes **décrit les types d’objets du système** et les différents types de relations statiques qui existent entre eux. Les diagrammes de classes montrent également *les propriétés et les opérations* d'une classe ainsi que les contraintes qui s’appliquent à la manière dont les objets sont connectés.



Qu’est ce qu’un diagramme de séquence ?

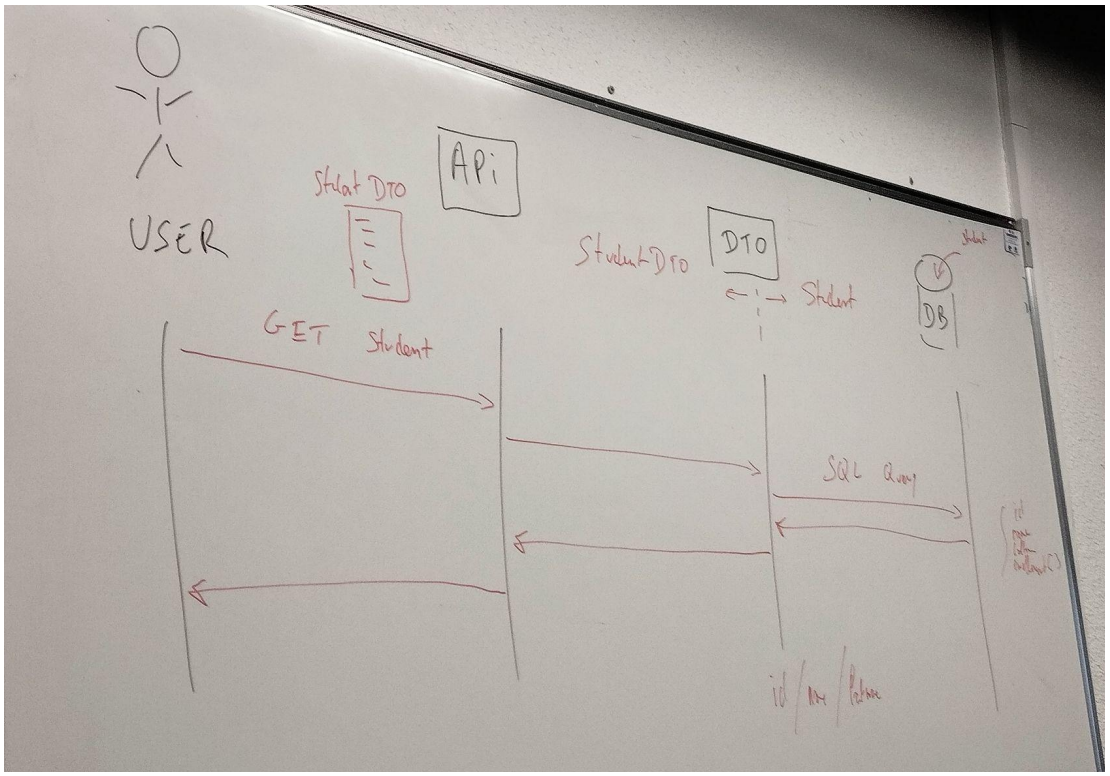
> Un diagramme de séquence montre la **collaboration dynamique** entre plusieurs objets en décrivant *l’ordre temporel* dans lequel les messages sont envoyés entre eux.

```
@startuml

actor Utilisateur

Utilisateur -> Système: Demande de l'historique
Utilisateur <-- Système: Liste des fichiers
Utilisateur -> Système: Sélectionner un ou plusieurs \
        nfichier(s) et vérifier les URLs
Utilisateur <-- Système: Liste des URLs invalides\ndu/
        des fichier(s) sélectionné(s)

@enduml
```



Qu’est ce qu’un diagramme d’activité ?

> Les diagrammes d’activités sont une technique pour décrire la **logique procédurale**, les **processus métier** et les **flux de travail**.

```
@startuml
start
while (data available ?)
  :read data;
  :generate diagrams;
endwhile
stop
@enduml
```

SOLID

- Principe de responsabilité unique
- Principe ouvert/fermé
- Principe de substitution de Liskov
- Principe de ségrégation des interfaces
- Principe d'inversion des dépendances

- Single Responsibility Principle
- Open/closed principle
- Liskov substitution principle
- Interface segregation principle
- Dependency inversion principle

Principe de responsabilité unique

Une classe doit avoir une et une seule raison de changer !

- La classe est plus facile à comprendre ;
- La classe est plus facile à maintenir ;
- La classe est plus réutilisable.

Le SRP équivaut à peu près à avoir une **« cohésion élevée »**. On dit qu'une classe a une forte cohésion si ses comportements sont fortement liés et fortement concentrés. Le SRP stipule qu'une classe doit être cohérente au point d'avoir une seule responsabilité, où une responsabilité est définie comme **« une raison de changement »**.


```
class XMLExporter
{
   URL runSaveDialog() { ...}
   void showSuccessDialog() { ...}
   String exportDocToXML(Document doc) { ...}
   void exportDocument(Document doc)
   {
```

```
      String xmlFileContent = exportDocToXML(doc); URL fileURL = runSaveDialog();
xmlFileContent.writeToURL(fileURL); showSuccessDialog();
   }
}

class XMLExporter
{
   URL runSaveDialog() { ...}
   void showSuccessDialog() { ...}
   String exportDocToXML(Document doc) { ...}
   void exportDocument(Document doc)
   {
      String xmlFileContent = exportDocToXML(doc); URL fileURL = runSaveDialog();
xmlFileContent.writeToURL(fileURL); showSuccessDialog();
   }
}
class XMLConverter
{
   String exportDocToXML(Document doc) { ...}
}
class XMLExporter
{
   URL runSaveDialog() { ...}
   void showSuccessDialog() { ...}
   void exportDocument(Document doc)
   {
      XMLConverter converter;
      String xmlFileContent = converter.exportDocToXML(doc); URL fileURL = runSaveDialog();
xmlFileContent.writeToURL(fileURL); showSuccessDialog();
   }
}
```

Principe ouvert/fermé

Vous devriez pouvoir étendre le comportement d'une classe, sans le modifier.
Une classe doit être ouverte à l'extension, mais fermée à la modification.

- La classe devient robuste, flexible et réutilisable ;
- Moins de modification implique moins de bogue.

Principe de substitution de Liskov

Les fonctions qui utilisent une classe de base doivent pouvoir utiliser des objets de classes dérivées sans le savoir.

Lorsque les sous-classes n'adhèrent pas correctement à l'interface de la classe de base, il faut parcourir le code existant et tenir compte des cas particuliers impliquant les sous-classes délinquantes. Il s'agit d'une violation flagrante du principe ouvert/fermé.

Lors de l'apprentissage de la programmation orientée objet, l'héritage est généralement décrit comme une relation « est un », mais elle entraîne **parfois une mauvaise utilisation de l'héritage**.

Principe de ségrégation des interfaces

Les clients ne devraient pas être obligés de dépendre d'interfaces qu'ils n'utilisent pas.

Ce principe permet d'éviter le couplage involontaire entre classes.

Principe d'inversion des dépendances

Les modules de haut niveau ne doivent pas dépendre de modules de bas niveau. Les deux devraient dépendre d'abstractions.

Ce principe permet de réduire le couplage entre composants.

Le principe d'inversion de dépendance est essentiellement un moyen d'ajouter des fiches et des prises à votre code. Il vous permet de créer des modules de haut niveau indépendants des modules de bas niveau. Les modules de bas niveau peuvent être créés ultérieurement et sont facilement remplaçables.

JSON

Pq on utilise json en fin de service ???????

Bases de donnée relationnelles.

Les bases de données NoSQL (« pas seulement SQL ») sont des bases de données non tabulaires et stockent les données différemment des tables relationnelles.

- Les bases de données de documents stockent les données dans des documents similaires aux objets JSON.
- Les bases de données clé-valeur sont un type de base de données plus simple dans lequel chaque élément contient des clés et des valeurs.
- Les base de donnée orientée large colonnes larges stockent les données en se focalisant sur chaque attribut et en les distribuant.
- Les bases de données graphes stockent les données dans des nœuds et des arêtes.

====> format simple, léger, normalisé, utilisé par tout le monde (langage machine-machine, API etc…)
====> GPT : Le concept clé ici concerne la prévention des références circulaires lors de la sérialisation d'objets vers JSON. Une référence circulaire se produit lorsqu'il y a une boucle dans la structure d'objets, où un objet fait référence à un autre objet, et cet autre objet fait à son tour référence au premier, créant ainsi une boucle infinie. Lors de la sérialisation JSON, cela peut entraîner des problèmes, tels que des boucles infinies et une taille de sortie JSON excessivement grande.

pq on parle de références circulaires ? c quoi fin de service ?

En gros si t'as une class qui utilise d'autres class ca fuck up tes tables de données et quand t'essayes de renvoyer une information avec une requête c'est pris obligatoirement dans une boucle infini. Le but du coup de ce truc *json* est d'empêcher que ça se fasse en gros. On parle de ca parceque quand t'as une table qui lie 2 autres tables ca fou la merde.
Fin de service c'est parce que tu le mets à la fin de ton programme.

=> on le met à la fin pour éviter de sérialiser du json à l'infini

y'a un truc EntityFrameWork qui coupe les boucles infinies dans program.cs ?

Pourquoi utiliser débogage et pas dotnetrun?

Pour exploiter le pouvoir de l'IDE ?

CODE UML

```
@startuml NomDuUml
skinparam Style strictuml
class ClassA{
    - attributeA1 : TypeA
    - attributeA2 : TypeB
    + ConstructeurA()
}
class ClassB{
    - attributeA1 : TypeC
    - attributeA2 : TypeD
    + ConstructeurB()
    }
class ClassC{
    - attributeA1 : TypeE
    - attributeA2 : TypeF
    + ConstructeurC()
    }
ClassB "0" -left-o "1" ClassC : association1
ClassB "0" -right-o "1" ClassA : association2
@enduml
```