csarp-back-01-01-01-student-update

# Diák adatok frissítése a desktop alkalmazásban

## Refaktoráció

1.

A diák adatainak frissítését backend oldalon már megcsináltuk. Ezért

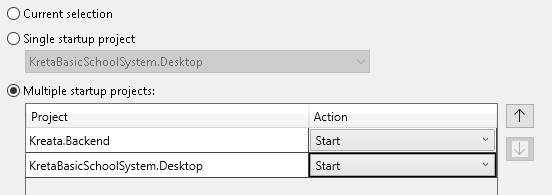
* vagy megírjuk a kódokat a backend oldalon a desktop projektben is
* vagy kicseréjük a backend kódját (a Kreta.Backend mappát kitöröljük és letöltjük az új backendet, és a kitörőlt helyébe bemásoljuk)

A letölthető backend diák frissítéssel [itt letölthető](https://github.com/csarp-backend/csarp-back-02-01-01-student-update/commits/main/) és elérhető.

A vizsgaremek fejlesztésekor az 1. pontra nem lesz szükség mert a vizsgaremekben együtt fejleszti a backendet és a desktopot (ajánlott a vizsgaremek fejlesztésekor a bakcend és forntend fejlesztést is külön-külön branchekben csinálni).

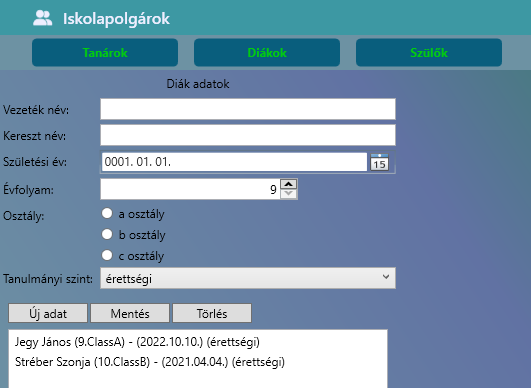
2.

Beállítjuk, hogy mind a két alkalmazás, a backend és desktop induljon el:



3.

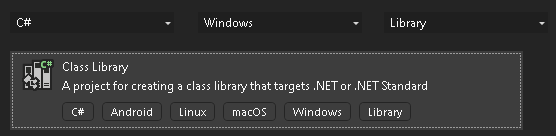
Teszt indításakor tapasztaljuk, hogy a diák menüpont alatt a diák adatok megjelennek



## Kódolás a desktop alkalmazásban

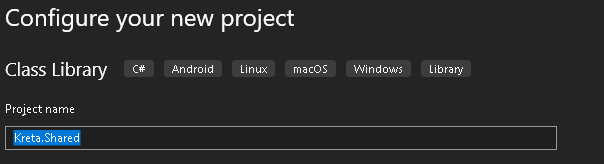
A backend oldalon adott egy Student osztály amelyben van Id, a desktop oldali Student osztályban nincs Id (lásd később DTO). Ugyan úgy az EducationLavel osztály mindkét oldalon megtalálható. De a ControllerResponse osztály csak backend oldalon van, a desktop program így nem ismeri ezt az osztályt, pedig szüksége lesz rá amikor pl. az update híváskor megkapja a választ.

Készítsük egy olyan projektet, melyben az osztályok a Desktop és Backend (Maui, Blazor) projektek számára is elérhető lesz!



A projekt egy Class Library templete projektből épül fel, ezt válasszuk.

A projekt neve legyen Kreta.Shared.



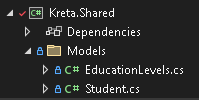
Az eredmény:



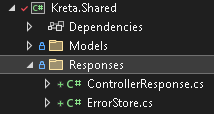
Először ezeket az anomáliákat javítsuk ki, és

A következő tevékenységeket kell elvégezni az osztott projektben:

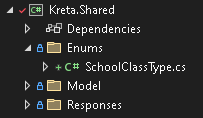
1. Töröljük ki a Class1 mappát
2. Hozzuk létre a Model mappát, abban a Student és EducationLevel osztályt.



1. A két osztály kódját a desktop proekből másoljuk át az osztott projektben lévő Student.cs és Education.cs fájlokba.
2. Az osztott mappába létrehozzuk a Responses mappát, abban az ErrorStore.cs és ControllerResponse.cs állományokt. Az állományok kódját a Backend projekt Datas\Responses almappájában lévő fájlok kódját áthelyezzük az osztott mappába.



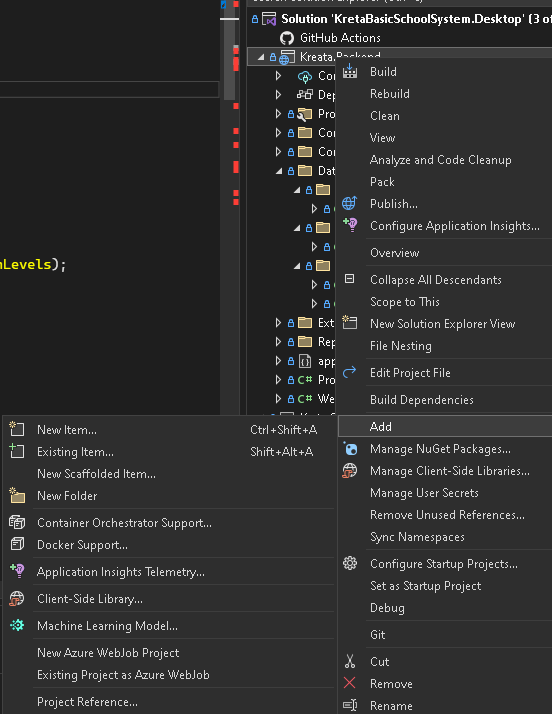
1. Az osztott mappába létrehozzuk az Enums mappát

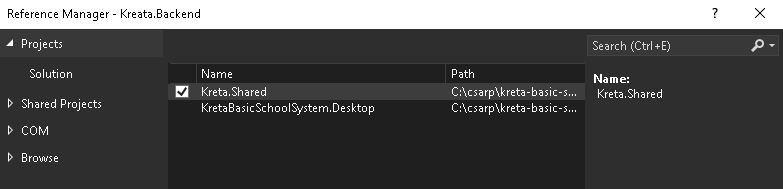


1. A SchoolClassType.cs állományba átmásoljuk a Backend oldalon lévő SchoolClassType.cs kódjából az enum típust.
2. A desktop alkalmazásban a Models mappában töröljük, a backend oldalon a felelslegesé vált Datas mappát
3. Töröljük a Studen osztályban lévő enum típust

Hogy a két projekt (Desktop, Backend) kapcsolatba legyen az osztott projekttel (Shared) úgynevezett project referenceket készítünk közöttük:

a) A Backend projekten kiválasztjuk a project referenc gyorsmenüt (bal alsó része a képnek)



b) Kiválasztjuk a „Kreta.Shared” projektet.

Ezt megismételjük a Desktop projekten is!

### DTO osztályok

Nem szerencsés, ha a backenden lévő adatok egy-az egyben érkeznek meg a desktopra vagy frontendre.

Pl.

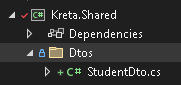
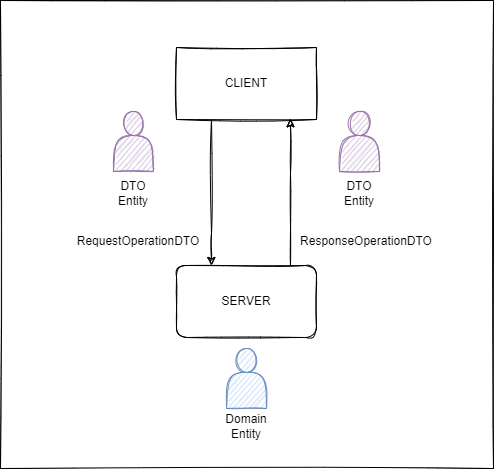
Ha van egy felhasználó, nem célszerű a felhasználói adatok lekérése során a jelszót is elküldeni a backendről, mert valaki megfejtheti.

Készíthetünk külön Dto osztályt a létrehozásra és módosításra (StudentForCreation, StudentForUpdate) ha valami miatt (pl. adatbáziskapcsolatok) más propertyket kell kezelnünk.

Ezekben a DTO osztályokban általában csak tulajdonságok vannak:

* [Data transfer object - Wikipedia](https://en.wikipedia.org/wiki/Data_transfer_object)
* [model view controller - What is a Data Transfer Object (DTO)? - Stack Overflow](https://stackoverflow.com/questions/1051182/what-is-a-data-transfer-object-dto)
* [What is the Difference Between a DTO and a POCO? - Code Maze (code-maze.com)](https://code-maze.com/difference-dto-poco/)
* [.NET Core Web API Best Practices - Code Maze Blog (code-maze.com)](https://code-maze.com/aspnetcore-webapi-best-practices/)

Készítsünk az osztott mappában egy StudentDto-t, amely most nagyban hasonlítani fog a Student osztályra, de a jövőbe a Student osztályhoz adhatunk hozzá tulajdonságokat, a StudentDto osztályban csak az adatok lesznek.



A Student osztályba viszont vegyük fel az Id-t. A StudentDto osztályban is legyen Id. A StudentDto csak property-ket és konstruktort tartalmazzon!

## Refaktorálás

Refaktorálásnak hívjuk azt a folyamatot, amikor a kód adatait vagy annak algoritmusát újra végig gondoljuk és újratervezzük:

1. refaktorálás

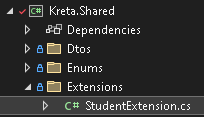
A backend (Cotnrollerek) DTO adatokat szolgáltatnak:

#### DM (DataModel) to DTO (DataTransfer Model)

Student->StudentDto

Készítünk metódusokat a DTO és DM osztályok közötti konvertálásra. Ehhez felhasználjuk a statikus osztályokat és statikus metósusokat.

### Statikus osztályok és metódusok

* [C# Intermediate - Static Members in C# - Code Maze (code-maze.com)](https://code-maze.com/csharp-static-members-constants-extension-methods/)

Az osztott (Shared) projektben az Extension mappába elkészítjük a StudentExtension statikus osztályt, abba a ToStudentDto metódust, amely a Student osztályt StudentDto osztállyá konvertálja. És elkészítjük ennek inverz műveletét is (itt az egyik metódust látjuk példaként).

public **static** StudentDto ToStudentDto(**this** Student student)

{

return new StudentDto

{

Id=student.Id,

FirstName = student.FirstName,

LastName = student.LastName,

BirthsDay = student.BirthsDay,

SchoolYear = student.SchoolYear,

SchoolClass = student.SchoolClass,

EducationLevel = student.EducationLevel

};

}

Ez egy kiterjesztett metódus, amely lehetővé teszi, hogy egy meglévő típust (this Student) további statikus metódusokkal bővítsünk. Az ilyen metódusokat egy statikus osztályon belül kell létrehoznunk, és az első paraméterük elé a this kulcsszó kerül.

De miért kell az első paraméter elé előtagot tennünk?

Mert ez a paraméter egy jelző, amely megmondja a fordítónak, hogy melyik típust bővítjük.

Tehát használhatjuk a következő képpen (ez csak példa, nem kell sehova begépelni):

Student student=new Student

{

Id=Guid.NewGuid(),

FirstName="János",

LastName="Jegy",

BirthsDay=new DateTime(2022,10,10),

SchoolYear=9,

SchoolClass = SchoolClassType.ClassA,

EducationLevel="érettségi"

}

StudentDto studentDto=student.ToStudentDto();

Lássuk mit kell tenni ezért a backenden:

2. Refaktorálás a backenden

a) A Backend Context mappájában javítjuk a megfelelő using-okat

b) Az IStudentRepo és StudentRepo mappába a using-ok javítása

c) A Controller rétegben a GetBy(Guid Id) StudentDto-t ad vissza

[HttpGet("{id}")]

public async Task<IActionResult> GetBy(Guid id)

{

Student? entity = new();

if (\_studentRepo is not null)

{

entity = await \_studentRepo.GetBy(id);

if (entity!=null)

return Ok(entity.**ToStudentDto()**);

}

return BadRequest("Az adatok elérhetetlenek!");

}

d) A SelectAllRecordToListAsync metódus StudentDto-t ad vissza

Lásd Select metódust: [LINQ Select (csharptutorial.net)](https://www.csharptutorial.net/csharp-linq/linq-select/)

[HttpGet]

public async Task<IActionResult> SelectAllRecordToListAsync()

{

List<Student>? users = new();

if (\_studentRepo != null)

{

users = await \_studentRepo.GetAll();

return Ok(**users.Select(student => student.ToStudentDto())**);

}

return BadRequest("Az adatok elérhetetlenek!");

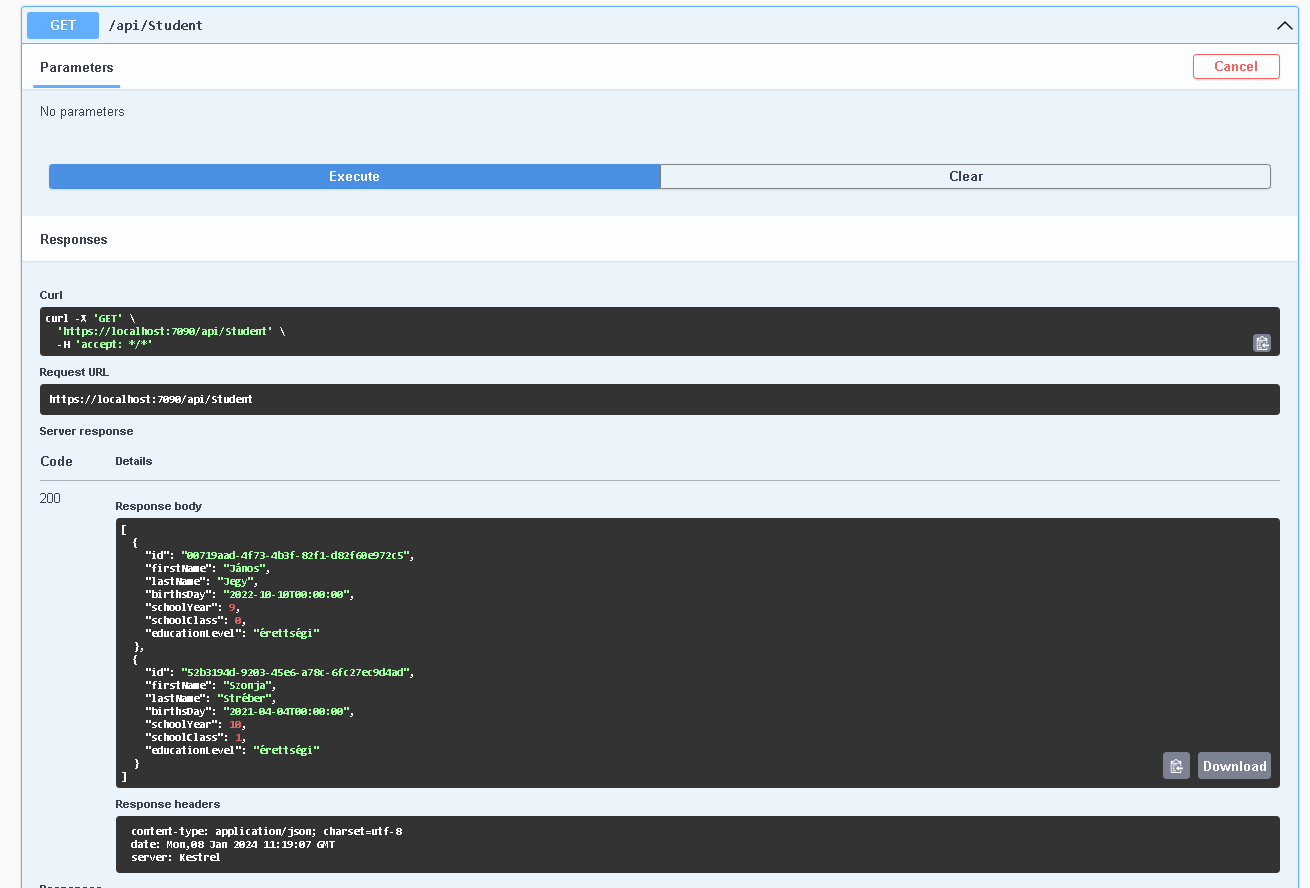
}

e) Az UpdateStudentAsync metódus StudentDto osztályt vár. Ennek megfelelően írjuk át a controller működését!

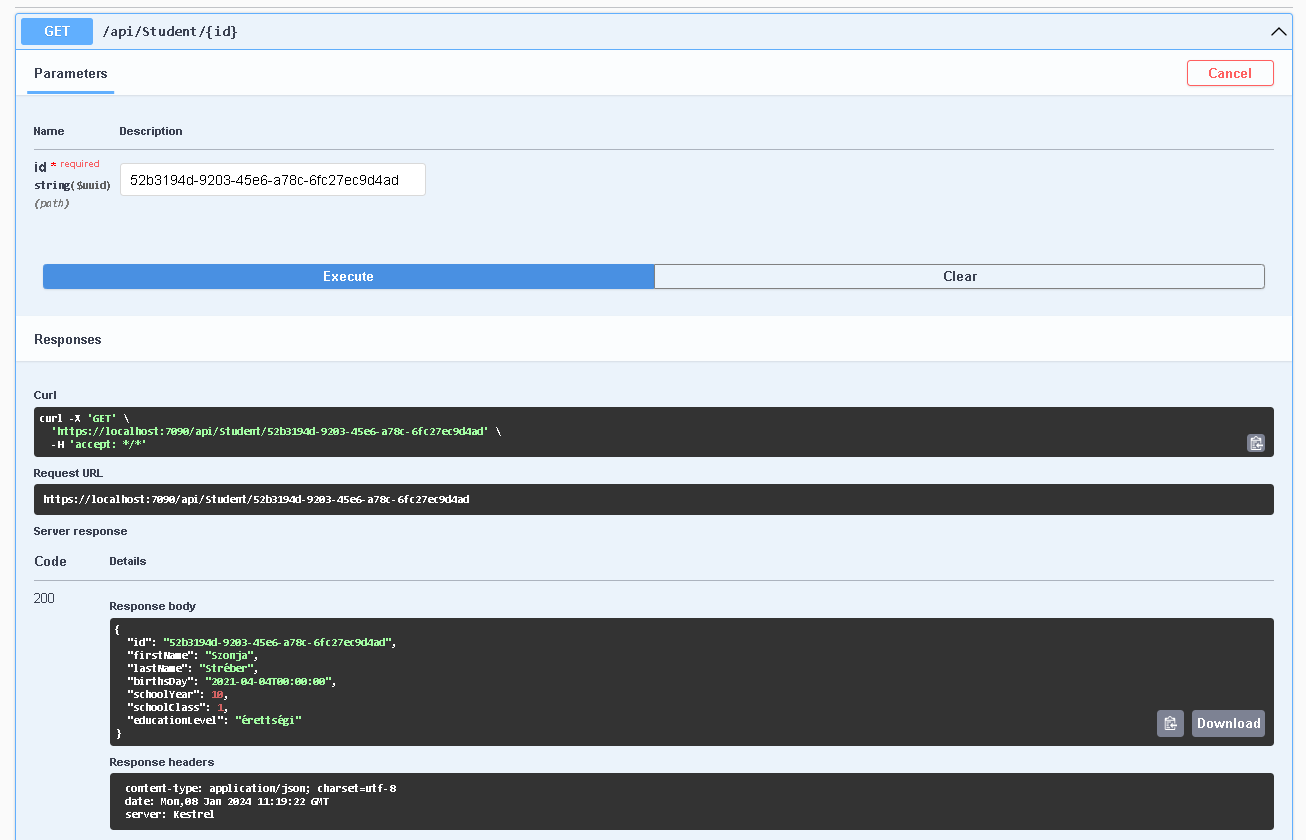
f) ModelBuilderExtension osztályban is módosítani kell a usingokat

Tesztelhetjük a backend-et!

a)



b)



c)

