

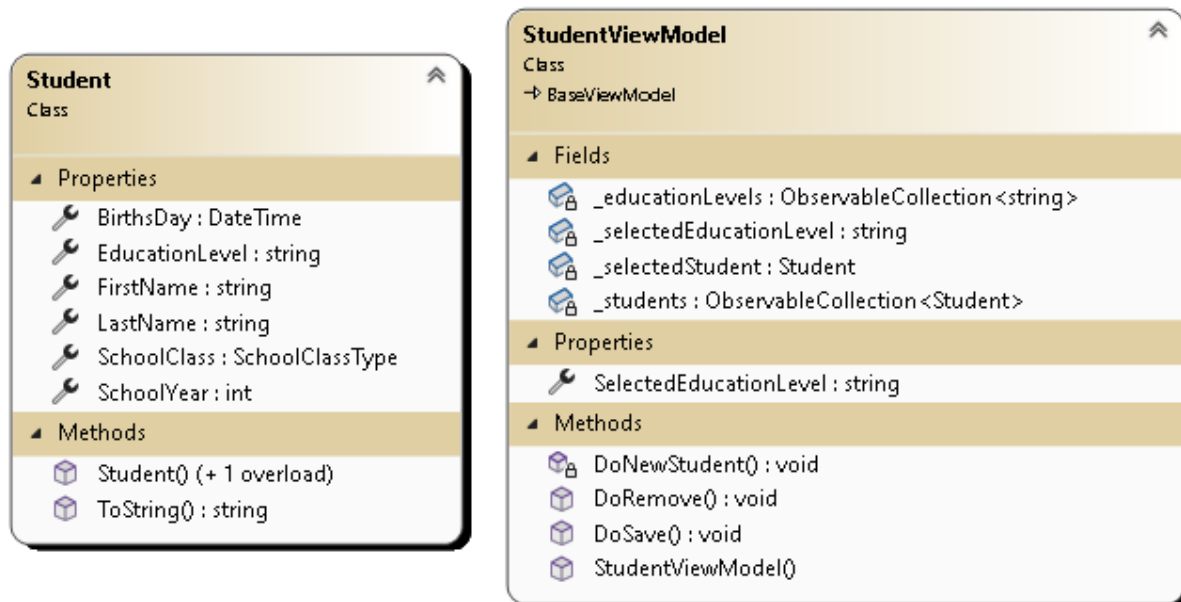
A tananyag elérhetősége: [05-00-00-wpf-mvvm-backend](#)

WPF MVVM menü és backend összekötése

A célunk, hogy a diák adatokat a backend oldalról jelenítsük meg az WPF MVVM menüs alkalmazásunkban!

Az alkalmazásunkban a Model rétegben található a Student osztály és az EducationLevels osztály.

A ViewModel rétegben a StudentViewModel réteg.



A cél, hogy a diák adatokat eljussanak a backendről a _studens változóba, amelynek Listához hasonló szerkesete van.

A backend projektet helyezük el a Desktop solution-ba

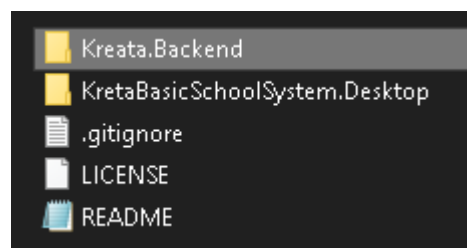
1.

A Backend kódját töltsük le és helyezük el a Desktop solution-ba!

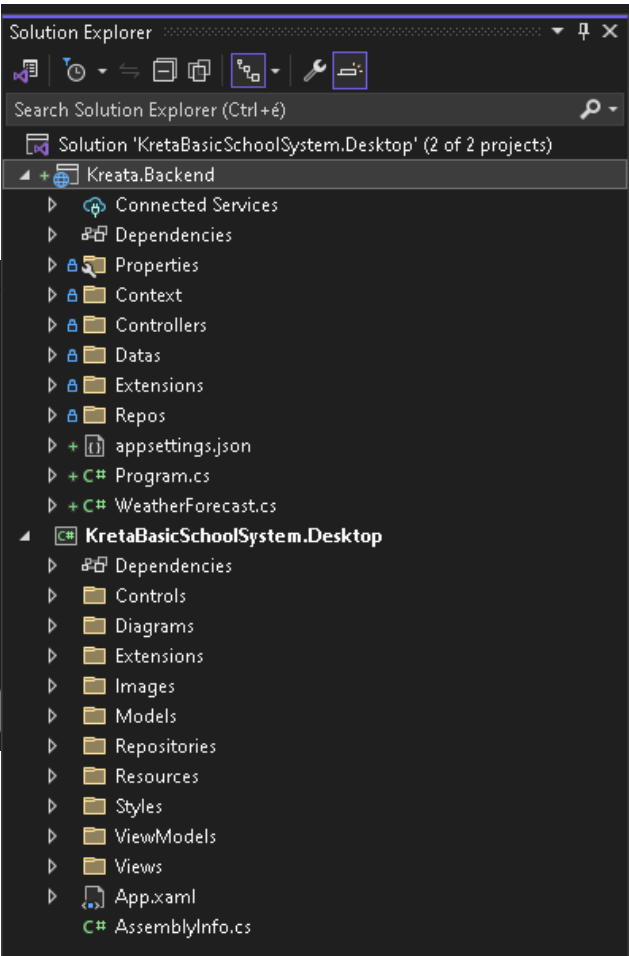
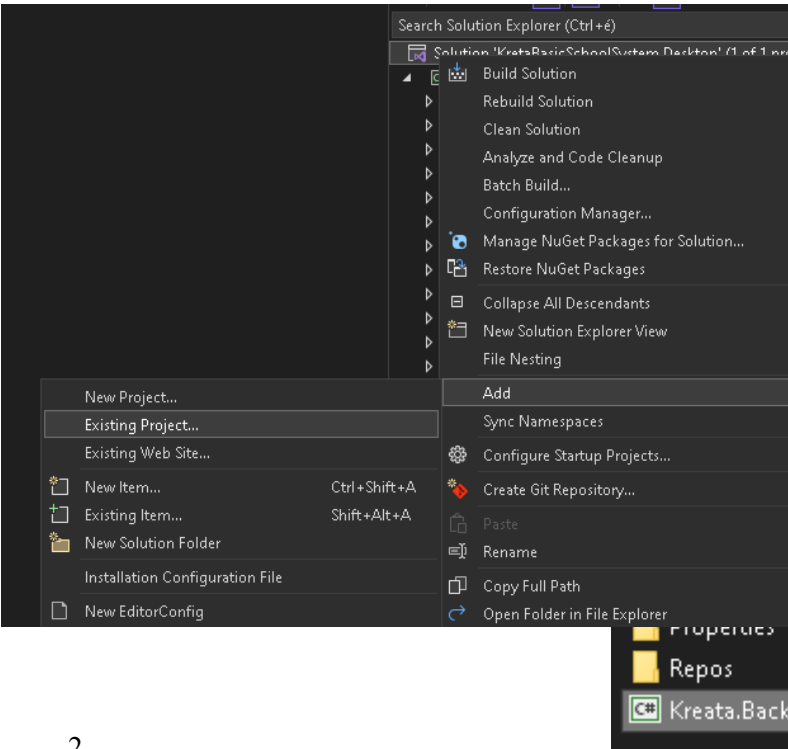
A backend kód:

- <https://github.com/csarp-backend/csarp-back-01-01-01-teacher-list-get-download>

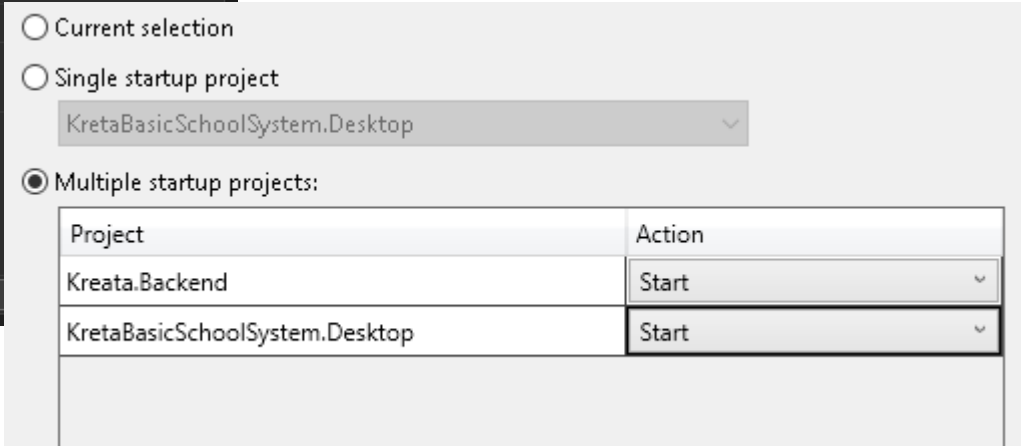
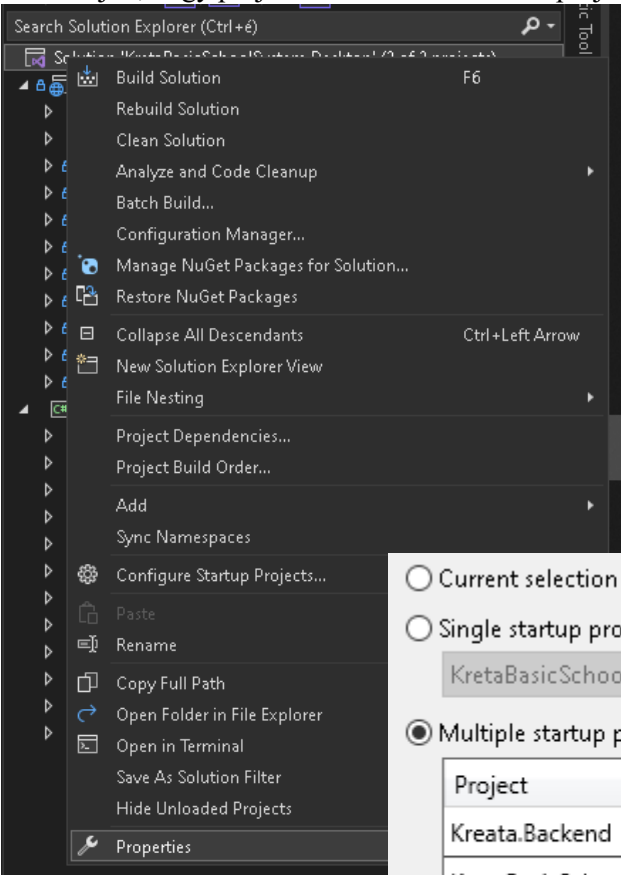
Töltsük le tömörítve és helyezük el a projektbe a Kreta.Backend mappát!



A bemásolt projektet már létező projektként adjuk hozzá a desktop projektünkhöz:

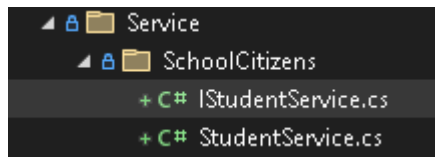


2. Beállítjuk, hogy projekt indulásakor mindkét projekt induljon el!



A backendel kommunikáló service réteg elkészítése

Elkészítjük a megfelelő mappába az interface-t és az osztályt is!



A service réteg a backendről az összes diák adatát kéri le:

```
interface IStudentService
{
    public Task<List<Student>> SelectAllStudent();
}
```

Implementáljuk az interface-t:

```
public class StudentService : IStudentService
{
    public Task<List<Student>> SelectAllStudent()
    {
    }
}
```

Telepítsük a **Microsoft.Extensions.Http** csomagot.

Konfiguráljuk a HTTP klienst és adjuk meg az alkalmazásunk URL-ét:

a) Extension/HttpCliensExtension.cs

```
public static class HttpCliensExtension
{
    public static void ConfigureHttpCliens(this IServiceCollection services)
    {
        services.AddHttpClient("KretaApi", options =>
        {
            options.BaseAddress = new Uri("https://localhost:7090/");
        });
    }
}
```

App.xaml.cs

```
public App()
{
    host = Host.CreateDefaultBuilder()
        .ConfigureServices(services =>
        {
            services.ConfigureViewViewModels();
            services.ConfigureHttpCliens();
        });
}
```

Injektáljuk a service-be a http clienst:

```
public class StudentService : IStudentService
{
    private readonly HttpClient _httpClient;

    public StudentService(IHttpClientFactory httpClientFactory)
    {
        _httpClient = httpClientFactory.CreateClient("KretaApi");
    }
}
```

Megírjuk a servic-t amely a backendről lekéri a diák adatokat:

```
public async Task<List<Student>> SelectAllStudent()
{
    if (_httpClient is object)
    {
        List<Student>? result = await _httpClient.GetFromJsonAsync<List<Student>>("api/Student");
        if (result is object)
            return result;
    }
    return new List<Student>();
}
```

b) A StudentService-t felvesszük a szükséges servicek közé (Extension/ ApiServiceExtensions.cs):

```
public static class ApiServiceExtensions
{
    public static void ConfigureApiServices(this IServiceCollection services)
    {
        services.AddScoped<IStudentService, StudentService>();
    }
}
```

App.xaml.cs

```
public App()
{
    host = Host.CreateDefaultBuilder()
        .ConfigureServices(services =>
        {
            services.ConfigureViewViewModels();
            services.ConfigureHttpCliens();
            services.ConfigureApiServices();
        })
        .Build();
}
```

Service beépítése a ViewModel rétegbe

1.
A StudentViewModel mielőtt alapértelmezetté válik és megjelenítené az adatokat a backendről fel kell tölteni adatokkal (inicializálás).

Ezért készítünk egy interface-t:

```
public interface IAsyncInitialization
{
    public Task InitializeAsync();
}
```

2.
A BaseViewModel mellé készítünk egy olyan modelt, amelyikbe muszáj implementálni az InitializeAsync metódust.

```
public class BaseViewModelWithAsyncInitialization : BaseViewModel,
IAsyncInitialization
{
    public virtual Task InitializeAsync()
    {
        return Task.CompletedTask;
    }
}
```

A metódus azért lesz virtual-is, hogy az örökölt osztályokba felülírható legyen (override).
A StudentViewModel az új osztálytól öröklődik:

```
public partial class StudentViewModel : BaseViewModelWithAsyncInitialization
{
    3.
```

A StudentViewModelben injektálunk egy IStudentService-t:

```
private readonly IStudentService? _studentService;
...
public StudentViewModel(IStudentService? studentService)
{
    ...
    _studentService = studentService;
}
```

és lekérjük a service segítségével a backendről a diákokat. A diákokat a listából átrakjuk az ObservableObject property-be, amely a diákok megjelenítését végzi.

4. Felülírjuk az InitializeAsync metódust, hogy letöltse a backenről az adatok és a letöltött adatokat átrakjuk a Students listába:

```
public override async Task InitializeAsync()
{
    if (_studentService is not null)
    {
        List<Student> students = await _studentService.SelectAllStudent();
        Students = new ObservableCollection<Student>(students);
    }
}
```

5.
A diákok menüpont választás a `SchoolCitizensViewModel` osztályban történik, ezért a ShowStudentView metódust átírjuk:

```
[RelayCommand]
public async Task ShowStudentView()
{
    await _studentViewModel.InitializeAsync();
    CurrentChildViewModel = _studentViewModel;
}
```

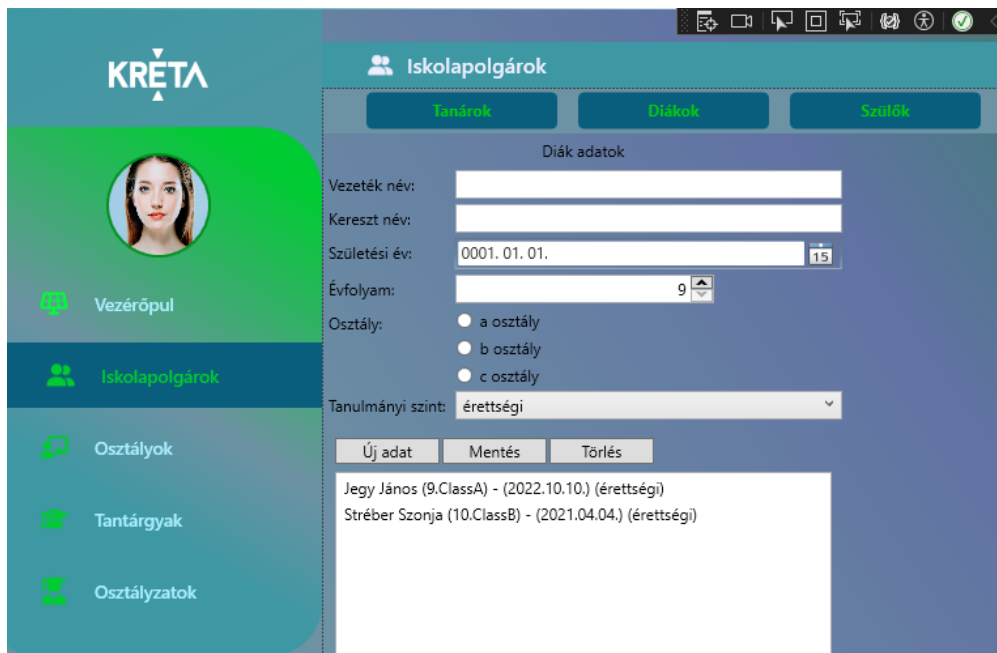
6.
Design módba alapértelmezett konstruktor fut le, ezt az esetet és működőképessé tesszük (lásd a commitot).

Kell legyen alapértelmezett konstruktor a StudentViewModel-ben (lásd commit).

Ha szükséges a StudentView DataContextusát pontosítjuk (lásd commit).

Ha valamelyik service hiányzik, vagy viewmodel akkor azt vegyük fel (lásd commit).

Ha minden jól csináltunk, az adatok megjönnek a backendről:



KRÉTA

Iskolapolgárok

Tanárok Diákok Szülők

Diák adatok

Vezeték név:

Kereszt név:

Születési év: 0001. 01. 01.

Évfolyam: 9

Osztály: ☐ a osztály ☐ b osztály ☐ c osztály

Tanulmányi szint: érettségi

Új adat Mentés Törlés

Jegy János (9.ClassA) - (2022.10.10.) (érettségi)
Stréber Szonja (10.ClassB) - (2021.04.04.) (érettségi)