

# Haladó Fejlesztési Technikák

## Féléves Feladat

2023/24/2 félév

A tantárgy féléves feladatában minden hallgatónak egy saját adatbázisra épülő rétegzett CRUD alkalmazást kell fejlesztenie verziókövető rendszerrel támogatva.

A féléves feladat alapvető elvárásai alább kerülnek ismertetésre. A kötelezően elvárt részeknek maradéktalanul meg kell felelni, különben a féléves feladat nem fogadható el.

### Az alapvető elvárások a projekttel kapcsolatban

- A féléves feladatot a **git** verziókövető rendszerrel kell elkészíteni az első lépésektől kezdve. Utólag nem adható a git-hez a projekt (hangsúlyozandó, ez nem egy feltöltő oldal!);
- A git verziókövető rendszer lokális verziókövetéséhez csatlakoztatni kell egy távoli kódtárat (repository-t) is, amelyet a **github.com** oldalon kell létrehozni;
- A **GitHub**-on létrehozandó repository neve: **ABC123\_HFT\_2023242** (ahol az ABC123 helyére a saját neptunkódját kell minden hallgatónak írnia, HFT a tárgy neve, 2023241 a jelenlegi félévet egyértelműen azonosító kód);
- A GitHub repository láthatósága **privát** kell, hogy legyen, melyhez szükséges collaborator-ként a **gyakorlatvezető tanár github accountját** és az **oenikprog** usert is meghívni a projektbe. (A meghívást követően nem jelenik meg azonnal collaborator, mivel az oktatóknak a meghívókat egyesével el kell fogadniuk, emiatt várhatóan néhány nap múlva fog megjelenni);
- A kód írásakor egy-egy egység elkészülte után **commit**-olni szükséges. A féléves feladatban **minimum 25 commit-ot várunk el**;
- A mérföldkövek előtt szükséges a **GitHub repository**-ba is felküldeni a kódot a **push** utasítással (utólagos közzététel késésnek számít);
- Hallgató és javító oktató között kód csak és kizárólag a **GitHub**-on keresztül közlekedhet. E-mailes és egyéb kódbeadásra nincs lehetőség.

### Az alapvető elvárások az üres solution felépítéssel kapcsolatban

- A féléves feladat kezdetekor el kell indítani a Visual Studio (továbbiakban VS)-t és létre kell hozni egy új **Console Appot .NET 5.0**-ben! A **solution** neve legyen: **ABC123\_HFT\_2023241** (ahol ABC123 helyére szintén a neptun kódot kell írni). A **project** neve legyen: **ABC123\_HFT\_2023241.Client**
- Létre kell hozni a **solution**-ben az alábbi **projekteket** (a projekt nevében szerepeljen a solution neve is prefixként, szintén saját neptun kóddal):
  - **ABC123\_HFT\_2023242.Models** (Class Library);
  - **ABC123\_HFT\_2023242.Logic** (Class Library);
  - **ABC123\_HFT\_2023242.Repository** (Class Library);
  - **ABC123\_HFT\_2023242.Endpoint** (ASP.NET Core Empty és nem kell https támogatás);
  - **ABC123\_HFT\_2023242.Test** (Class Library).

- Létre kell hozni a solution nevére kattintva (Add) **3 solution folder**-t, és ezekbe drag-and-drop módszerrel áthúzni a megfelelő projekteket:
  - **Backend** (Endpoint, Logic, Repository, Test);
  - **Frontend** (Client);
  - **Shared** (Models).
- Legyen beállítva, hogy a **Start** gombra kattintva mind a **backend** alkalmazás (a szerver), mind a **frontend** alkalmazás (a kliens) induljon el. A solution-ön jobb kattintás > Properties > Startup project > Multiple startup projects > Client és Endpoint megjelölése **start**-ként; (Ez a beállítás nem fog felkerülni a git-re, de ez nem probléma)
- Szükséges beállítani a függőségeket! Egy adott projekten jobb kattintás > Add > Project Reference. Az alábbi függőségeket szükséges beállítani. Más függőség nem szerepelhet a rétegek között, ám a Client-hez tesztelési célból felvehető bármelyik, azonban a féléves feladat beadási határidejekor már csak a Models-t ismerheti a Client;
  - **Endpoint**-hoz fel kell vennie: **Logic, Repository, Models** függőségeket;
  - **Test**-hez fel kell venni: **Logic, Repository, Models** függőségeket;
  - **Logic**-hoz fel kell venni: **Repository, Models** függőségeket;
  - **Repository**-hoz fel kell venni: **Models** függőségeket;
  - **Client**-hez fel kell venni: **Models** függőséget;
- Az eddig elvégzett projekt beállításokat követően érdemes a git repository inicializálása.
  - A VS solution-ben állva a Studio jobb alsó sarkában látható egy „**Add to source control**” gomb. Ez akkor látszik, hogyha a VS telepítésekor az **Individual components** fülön ki lett választva a **Git for Windows** és a **GitHub Extension for Visual Studio**. Amennyiben ez korábban nem lett elvégezve, akkor Vezérlőpult > Programok telepítése és törlése > Visual Studio Community 2019 > Change vagy a Visual Studio Installer indítása > Aktuális VS verzió mellett Modify. Az adatbázis kezeléshez szintén itt a telepítőben bekapcsolandó a **Data storage and processing Workload**;
  - Az „**Add to source control**” gombra kattintás után meg kell adni a saját GitHub account adatait, ellenőrizni a repository nevét (ugyanaz, mint a solution neve), ellenőrizni, hogy privát-e a repository, majd „**Create and Publish**” lehetőséget választani;
  - Hogyha VS-ből történik a repository létrehozása, akkor a **.gitignore** fájl létrejön és illeszkedik a projek típusához (tényleges C#-ra vonatkozó szabályokat fog tartalmazni);
  - Hozzon létre egy **Others** mappát a solution-ben, jobb kattintás > Add > Existing Item. Ezt követően legyen kiválasztva a **.gitignore** fájl a projekt gyökeréből (ez akkor jött létre, amikor „Add to source control”-t el lett végezve);
  - A kód írása során commit-oljon minél gyakrabban, értelmes módosítások után és tegye közzé a kódot (push segítségével) például minden nap végén egyben (1 push feltölthet akár több commit-ot is!);
  - Érdemes időnként a **GitHub**-ról letölteni a projektet a webes felületen található **code zöld gomb** > **download zip** lehetőséggel és kicsomagolni egy teljesen másik mappába. Az így

kicsomagolt solution-t ajánlott megnyitni és megnézni, hogy sikeresen lefordul a GitHub-on található kód. A javító tanár pont ugyanezt látja majd mikor ellenőrizni fogja (tehát amennyiben nem fordul le a letöltött kód, akkor meg kell nézni, hogy mi romolhatott el);

### Az alapvető elvárások a szoftverrel kapcsolatban

- A szoftvernek fordulnia kell a javító tanár számítógépén;
- **.NET 5.0** verzióban kell írni és **InMemoryDb** adatbázist kell használni **Entity Framework Core** felett. Tesztelési célból használható LocalDb, de a végső verzióban **InMemoryDb**-t kell használni. Más adatbázis és NET verzió nem fogadható el;
- A C# nyelvű program osztályait, metódusait, változóit **angol nyelven** nevezze el. A programban **kommentelni** lehet **magyar nyelven** is, de ajánlott **angolul**;
- A féléves feladatban legalább 3 adattáblát kell létrehozni, amelyek kapcsolódnak egymáshoz idegen kulcs kapcsolattal. Vagyis 3 db **Model** osztály készüljön a **Model Class Library**-ben. Például: egy márkához tartozik több autó, minden autóhoz tartozik több bérleti esemény. Hogyha több-a-többhöz kapcsolatot valósítunk meg kapcsolótáblával, akkor a kapcsolótábla nem számít bele a szükséges három tábla közé;
- A **Model** osztályokban legyenek letárolva az idegen kulcsok és használjon **Navigation Property**-ket **LazyLoader**-rel ahol lehet! A Linq lekérdezésekben akkor használjunk join-t, ha elkerülhetetlen;
- A **Data** rétegben a **DbContext** osztály **OnModelCreating** metódusában töltsse fel az adatbázist minden indításkor tesztadatokkal! A felhasználó tudjon indításkor már meglévő adatokból elindulni;
- A fejlesztés közben lehetőség van arra, hogy a **ConsoleApp** is megkapja **Project Reference**-ként a **Logic**, **Repository** és **Models** rétegeket. Ezzel tesztelje az alkalmazást működés közben, de a projekt végére a **ConsoleApp** majd csak **API hívásokkal** kommunikálhat az **Endpoint** réteggel és csak a Models library-t ismerheti!
- A **Logic** a **Repository**-t, mint **függőséget** csak **interfészen át**, **konstruktor** paraméterként kaphatja meg (Dependency Injection)! A **repository** a **DbContext** **függőséget** csak **konstruktor paraméterként** kaphatja meg, az **Endpoint controller**-jei a **logic** **függőséget** csak **interfészen át**, **konstruktor** paraméterként kaphatják meg! A függőségek beszúrását az **Endpoint** projekt végzi, **IoC konténer** segítségével! Tesztelési célból a konzolos alkalmazásban kézzel példányosíthatóak;
- Minden **Model** osztályhoz szükséges elkészíteni 1-1 **repository** osztályt, amely tartalmazza a **CRUD** metódusokat (Create, Read, ReadAll, Update, Delete). A **ReadAll** metódus **IQueryable<T>** interfészen át adja vissza a logic-nak a **DbSet**-eket;
- A **Logic** rétegnek szintén biztosítania kell ezeket a **CRUD** metódusokat, valamint szükséges legalább **5 db non-crud** metódust készíteni a **logic**-ban, amelyek több táblás lekérdezést használnak! A **CRUD** metódusok eredményeit **void**, **T** és **IEnumerable<T>**, a **non-crud** metódusok eredményeit **IEnumerable<T>** interfészen keresztül adja vissza a felsőbb rétegeknek! Non-

crudokra néhány példa: egy adott autó márkára ki az a megrendelő, aki a legnagyobb összegben adott le bérleti igényt. (Ehhez a lekérdezéshez pl. mindhárom entitásra szükség van);

- A Test projektben **NUnit** és **Moq** könyvtárakat kell használni. A **Logic** a **Moq** segítségével egy **áladatbázist** kap függőségként. A **unit tesztek** elsősorban a **non-crud** metódusokat tesztelik a **Logic**-ból! Valamint a **Logic**-ban lévő **Create** metódusok hibakezelését (pl.: névként üres string dobjon kivételt, stb.)! Egy **Logic**-beli **Create** abban különbözik egy **Repository**-beli **Create**-től, hogy hibakezelést is végez, **Exception**-öket dob. A **Repository**-beli **Create** ellenőrzés nélkül mentse el az adatbázisba a megkapott objektumot;
- A féléves feladatban minimum **10 db Unit tesztet** kell létrehozni! Pl: 5 db non-crud, 3 db create és 2 szabadon választott egyéb teszt;
- A féléves feladatban minden tesztnek sikeresnek kell lennie! Hogyha 20 tesztet készít, akkor mind a 20-nak sikeresnek kell lennie, **nem csak az elvárt 10 db-nak!**
- Minden **Model** osztályhoz tartozzon egy **Repository** osztály (pl: Car → CarRepository) és egy **Logic** osztály (Car → CarLogic). Egy **Logic** osztály ismerhet és felhasználhat több **Repository**-is (pl.: CarLogic ismerheti a CarRepository-t és a BrandRepository-t is, ha szükséges a lekérdezéshez több repository adata is);
- A projekt **Endpoint** rétege ismeri a **Logic** osztályokat, és a bennük lévő funkciókat publikálja a külvilág felé **API Endpointok** formájában! Minden **Logic** osztályhoz tartozhat egy vagy több **ApiController**. Az **ApiController**ek **Action**-jei feleltethetők meg a **Logic** rétegek metódusainak. Célszerűen:
  - HTTP GET → Read, ReadAll;
  - HTTP POST → Create;
  - HTTP PUT → Update;
  - HTTP DELETE → Delete.
- A **konzolos** alkalmazás **API kéréseket** küld az **Endpoint** felé **JSON üzenetek** formájában. A konzolos alkalmazásból elérhetőnek kell lennie az összes CRUD metódusnak és összes non-CRUD metódusnak! Erre használható a ConsoleMenu-Simple NuGet csomag opcionálisan.

## A projekt mérföldkövei

A féléves feladat során az alábbi mérföldköveket várjuk el, ám teljesítésüket nem fogjuk ellenőrizni.

**Március 27., 23:59:59**

- GitHub repository elkészült a megfelelő néven;
- oenikprog user és gyakorlatvezető meghívásra került collaborator-ként;
- Az üres projektek létre vannak hozva;
- Ez utóbbi művelet pusholva lett a GitHub-ra.

### **Április 3., 23:59:59**

- Model rétegben min. 3 osztály szerepel és ezekben tartalom van;
- Repository rétegben az XYZDbContext osztály szerepel (XYZ helyére nyilván saját tartalmat írunk);
- DbSeed-ben minta adatok kerülnek mindhárom táblába.

### **Április 17., 23:59:59**

- Repository rétegben min. 3 osztály szerepel;
- Ezekben az összes CRUD metódus meg van írva.

### **Május 3., 23:59:59**

- Logic rétegben az osztályok elkészültek;
- Ezekben meg van írva minden crud és non-crud metódus;
- 10/10 Unit teszt futtatható és sikeres.
- Endpoint app buildelhető, futtatható és API hívásokra reagál.

**Határidő:** Május 8., 23:59:59

**Javítási határidő:** Május 15., 23:59:59

A határidő azt jelenti, hogy a javító tanárok ekkor töltik le a GitHub repository-k tartalmát.

Sikeres munkát kívánunk!