Haladó Fejlesztési Technikák

Féléves Feladat

2021/22/1 félév

A tantárgy féléves feladatában minden hallgatónak egy saját adatbázisra épülő rétegzett CRUD alkalmazást kell fejlesztenie verziókövető rendszerrel támogatva. A CRUD alkalmazás az alábbi funkcionalitásokat jelenti:

- **C**reate – létrehozás;
- **R**ead – egy/több elem olvasása;
- **U**pdate – módosítás;
- **D**elete – törlés.

Témáját tekintve a jó ízlés határain belül tetszőleges lehet, kivéve a laborokon bemutatott téma/témák (pl.: CarShop). Javaslat hasonló szerkezetű és bonyolultságú feladatot önállóan kitalálni, de lehet több táblából álló projekt is, azonban ez többletmunkát fog jelenteni.

A féléves feladat alapvető elvárásai alább kerülnek ismertetésre. A kötelezően elvárt részeknek maradéktalanul meg kell felelni, különben a féléves feladat nem fogadható el. A dokumentumban később jelennek meg azok az elvárások is, amelyeknek nem kötelező megfelelni, ám ezek büntetőpontokat vonnak maguk után.

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_2021221** (ahol az ABC123 helyére a saját neptunkódját kell minden hallgatónak írnia, HFT a tárgy neve, 2021221 a jelenlegi félévet egyértelműen azonosító kód);
- A GitHub repository láthatósága **privát** kell, hogy legyen, melyhez szükséges collaborator-ként az **oenikprog** felhasználót meghívni a projektbe Admin szerepkörrel. (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_2021221** (ahol ABC123 helyére szintén a neptun kódot kell írni). A **project** neve legyen: **ABC123_HFT_2021221.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_2021221.Models** (Class Library);
    - **ABC123_HFT_2021221.Data** (Class Library);
    - **ABC123_HFT_2021221.Logic** (Class Library);
    - **ABC123_HFT_2021221.Repository** (Class Library);
    - **ABC123_HFT_2021221**.**Endpoint** (ASP.NET Core Empty és nem kell https támogatás);
    - **ABC123_HFT_2021221**.**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** (Data, Endpoint, Logic, Repository, Test);
    - **Frontend** (Client);
    - **Shared** (Models).

- Legyen beállítva, hogy a **Start** gomba 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;

- 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: **Data, Logic, Repository, Models** függőségeket;
    - **Test**-hez fel kell venni: **Logic, Repository, Models** függőségeket;
    - **Data**-hoz fel kell venni**: Models** függőséget;
    - **Logic**-hoz fel kell venni**: Repository, Models** függőségeket;
    - **Repository**-hoz fel kell venni**: Models, Data** 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), azonban szükséges az **mdf** és **ldf** fájlok verziókövetésének engedélyezése (lásd következő lépések);
- Hozzon létre egy **Others** mappát a solution-ben. Erre a mappára 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 fájl 265. és 266. sorában található *.mdf és *.ldf sorokat törlése vagy kikommentelése szükséges;
- Mivel a projektben történt módosítás (két sortörlése, kommentelése), készüljön egy commit a változásról az alábbiak szerint;
- A VS alsó sorában most már az „Add to source control” gomb helyett számos másik jelent meg. Keresse ki a ceruza ikont, ami mellett egy szám szerepel (hány fájl módosult az előző commit óta). Erre kattintva megjelenik a Git Changes ablak, melyben látható, hogy az sln és a gitignore fájlok módosultak. Írja be a szövegbeviteli mezőbe a commit üzenetét (message-t): „mdf and ldf removed from gitignore”. Kattintson alatta a **Commit All** gombra;
- Ekkor lokálisan létrejött a **commit**, szükséges lenne **push-**olni a **GitHub**-ra is. A panelen a felfele nyíl gomb jelenti a push-t. A gomb megnyomását követően a GitHub oldalon, a távoli repository-ban is megjelenik a módosítás;
- 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 **MS SQL** adatbázist kell használni **LocalDB-vel, Entity Framework Core** felett. 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 bérlési 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;

- Az **mdf** és **ldf** fájlokat a **Data** rétegben kell létrehozni, ahol be kell állítani a **Build Action-t Content** értékre, a **Copy to Output Directory** lehetőséget pedig **Copy always** értékre;

- Állítson be olyan **Connection String** értéket, amely a mindenkori munkakönyvtárban lévő lokális adatbázis fájlhoz kapcsolódik;

- A **Data** rétegben a **DbContext** osztály **OnModelCreating** metódusában töltse 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, Data** 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** és **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 bérlési igényt. (Ehhez a lekérdezéshez pl. mindhárom entitásra szükség van);

- A Test projektben **NUnit** és **Moq** könyvtárakat kell használni. A **Logic** a **Moq** segítségével egy **ál-adatbázist** 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;
- 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 **ApiControllerek Action**-jei feleltethetőek 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 ellenőrzése úgy történik, hogy az alábbi mérföldkövekkor lefut egy automatizmus (gitstat), ami minden hallgatóra ellenőrzi, hogy a mérföldkő követelményei teljesültek-e. Erről a hallgatók visszajelzést kapnak. Ez egy automatikus folyamat. Minden mérföldkő esetében a lefuttatott ellenőrzési folyamat megvizsgálja az előző mérföldköveket is. A 5. mérföldkő után a folyamat minden nap lefut.

Október 10, 23:59:59

- GitHub repository elkészült a megfelelő néven;
- oenikprog user meghívásra került collaborator-ként;
- Az üres projektek létre vannak hozva;
- .gitignore fájl módosítva lett;
- MDF és LDF fájl létre van hozva a Data rétegben;
- Ez utóbbi 3 művelet pusholva lett a GitHub-ra.

Október 17, 23:59:59

- Model rétegben min. 3 osztály szerepel és ezekben tartalom van;
- Data rétegben az XYZDbContext osztály szerepel;
- DbSeed-ben minta adatok kerülnek mindhárom táblába.

November 7, 23:59:59

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

November 21, 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.

November 28, 23:59:59

- Endpoint app buildelhető, futtatható és API hívásokra reagál.

Határidő: December 2, 23:59:59

Javítási határidő: December 9, 23:59:59

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

A projekt büntetőpontjai

A féléves feladatra 20 pont gyűjthető, alapvetően mindenki 20 pontról indul. 10 büntetőpont gyűjthető, utána a féléves feladat nem fogadható el. Amennyiben a december 2.-i határidőre beérkezett projektet a javító tanár ellát büntetőpontokkal, akkor még javítható december 9.-re. Mérföldkő nem teljesítése -1 pontot ér. A javító tanárok az alábbi hibákra adhatnak belátásuk szerint mínusz pontot, melyet a hallgatók felé indokolniuk kell!

- Rétegsértés;
- Rejtett függőségek;
- Értelmetlen unit tesztek;
- Futásidejű hibák;
- Értelmetlen osztály, változó, metódusnevek.

A dokumentumban a büntetőpontok fejezet előtt leírt minimum elvárások közül mindent be kell tartani. Ezek nem teljesülése nem mínusz pontot ér, hanem a teljes feladat elfogadhatatlan.

A féléves feladat és az oktatókkal való kommunikáció szabályai

- A heti előadások után érdemes kollektív kérdéseket az előadónak szóban feltenni;
- Egyéni problémákkal a laborvezetőt szükséges keresni a gyakorlatok végén;
- Végső esetben e-mailben a laborvezetőt lehet keresni.

Sikeres munkát kívánunk!