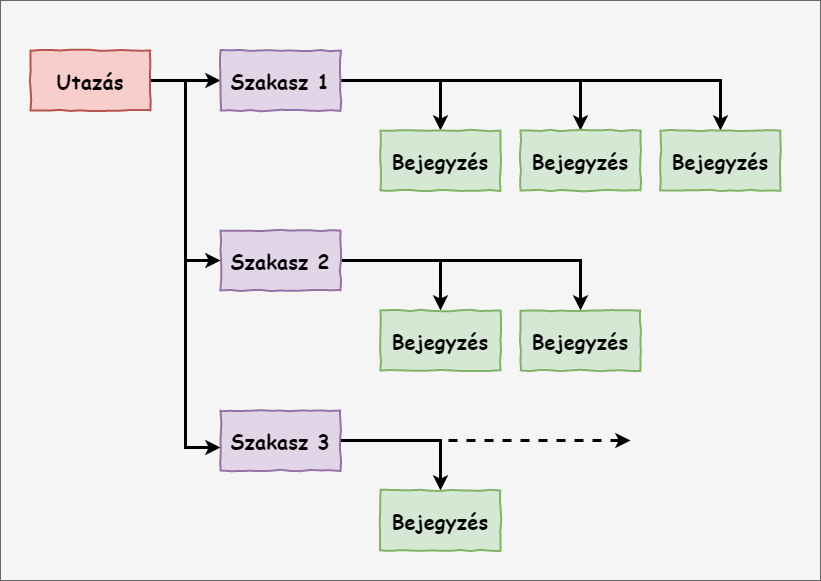
# 3. Fejlesztői kézikönyv

## A DiceTravel koncepciója

### Mi is a DiceTravel?

A DiceTravel alapvetően egy naplózó alkalmazás, mely utazásokkal kapcsolatos eseményeket tárol, rendszerez és jelenít meg a felhasználó számára. A DiceTravel továbbá egy platformot nyújt, hogy a felhasználók megoszthassák egymással az élményeiket.

Minden utazást kisebb részekre lehet osztani, ezeket szakaszoknak nevezzük. Egy esemény, melyről bejegyzés készül, mindig egy adott szakasz alatt történik. Ezért az adott bejegyzés az adott szakasz alatt kerül rögzítésre.

Egy szakasz befejezése egy új szakasz indítását vagy az utazás végét indukálja. Mivel ez egy logoló, és nem pedig tervező alkalmazás, nem lehetséges egyszerre több utazáson, szakaszban részt venni. Ettől függetlenül egy felhasználónak tetszőleges számú lezárt utazása lehet, egy utazásnak tetszőleges számú szakasza.

A bejegyzéseink, szakaszaink és utazásaink láthatóságának három szintje van: Privát, Barátok, Publikus.

Baráti kapcsolatokat jelölés-visszaigazolás módszerrel lehet létesíteni, ami azt jelenti, hogy tetszőleges felhasználónak küldhetünk barátjelölést, viszont a barátság csak abban az esetben kerül aktiválásra, ha ezt a célfelhasználó elfogadja.

A felület egyik legfontosabb része, amit szeretnék már itt kiemelni: a Folyam.  
A Folyamban fognak megjelenni a különböző keresésekkel kapcsolatos találatok, valamint összességében elmondható, hogy egy kontrol, az nem más mint egy adatbázis entitás. Például a felhasználó utazásai, más felhasználók utazásai, vagy egy felhasználó barátai. Ezek mind valamely Control osztály példányai lesznek.

### Kulcsszavak

* **Utazás (Journey):** A DiceTravel utazással kapcsolatos legnagyobb logikai egysége, mely alatt a Szakaszok találhatóak. Egy kezdőponttal rendelkezik, valamint egy elnevezéssel.
* **Szakasz (Trip):** Az utazás elemei. Minden szakasz egy végponttal rendelkezik. A kezdőpont az előző szakasz végpontja határozza meg, vagy ha még nincs teljesített szakasz, akkor az Utazás kezdőpontja lesz.
* **Bejegyzés (Entry):** A DiceTravel utazással kapcsolatos legkisebb logikai egysége. Minden Bejegyzés egy szöveges bejegyzésbő, ami maximum 1024 karakter hosszú lehet, és/vagy egy képből áll. Utóbbi maximum 16 MB méretű lehet.
* **Folyam (Flow):** A különböző utazással/barátokkal kapcsolatos szűrések nézetei. Egy felhasználó Utazás folyama például az összes általa készített Utazást listázza ki és jeleníti meg. A következő folyamok vannak például a DiceTravelben: Utazás, Történet, Aktuális, Barát.
* **Barát (Friend):** Olyan felhasználó, akivel láthatjuk egymás csak barátoknak szánt bejegyzésit is.

## A DiceTravel felépítése

### a GUI terve

Az alkalmazás főablaka 3 részre osztható:

* A menükbe kerülnek a fontosabb parancsok, de ezek többsége magán az ablakon is implementálásra kerülnek.
* Az adatáttekintő szekcióba Felhasználóval, Aktív utazássa, és a Következő Szakasszal kapcsolatos információk és legfontosabb funkciók kerülnek.
* A Folyamban jelennek meg a felhasználó által lekérdezett adatok. A jobb oldalán pedig az alapvető Folyam nézetek gombjai találhatóak.

### Osztálytípusok

Szeretném általánosságban ismertetni, hogy a Dice Travel hogyan épül fel, milyen osztálytípusokkal doglozik.

#### Entitások

Az entitások az adatbázissal való kommunikációra lesznek kifejlesztve, valamint az adott tábla egy adott sorának az programbeli leképzései lesznek. A hozzá tartozó metódusokkal könnyen képesek leszünk az alapvető CRUD műveletek végrehajtására, valamint a fontos lekérdezések indítására.

A koncepció alapján az alkalmazás az alábbi entitásokkal rendelkezik:

**Utazás, Szakasz, Bejegyzés, Felhasználó, Barátság.**

Ezek az entitások lesznek az alkalmazás adatbázisában tárolva, valamint ezek lesznek a programunk főbb osztályai, amik a megfelelő adatok manipulálására lesznek használva. Ezek mind az *Entity* absztrakt osztályból fognak származni.

Ezek egy közös *EntityClasses* mappába lesznek szervezve.

#### Formok

A felhasználó különböző űrlapokon keresztül tud majd létrehozni/módosítani/törölni bizonyos entitásokat az adatbázisból. Ezek szintén külön a *Forms* mappába lesznek szervezve, azon belül is entitásonkénti almappába. Valójában ezek az űrlapok fognak lehetőséget adni az adott entitás metódusainak felügyelt hívásaira.

Az alkalmazásnak a *MainForm* űrlap osztálya fog keretet adni, ez lesz a fő ablaka az alkalmazásnak. Ez szintén a Forms mappában fog szerepelni, közvetlenül az entitás almappák mellett egy szinten.

#### Kontrollok

A kontroll egy entitásnak a folyamban való megjelenése. Segítségével a felhasználó az adott entitással kapcsolatos műveleteket végezhet. Például törölheti az entitást, vagy kontextustól függően megváltoztathatja a folyamot. Az űrlapokhoz hasonlóan itt is a megfelelő entitások ellenőrzött hívásai fognak történni.

A kontrollok a Controls mappába lesznek szervezve.

#### Kiegészítő osztályok

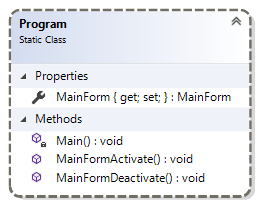
A Util mappán belül a különböző kiegészítő osztályok szerepelnek. Ebből három van:

* *ActiveUserStore*: Ebben a statikus osztályban van letárolva az éppen bejelentkezett felhasználó, valamint ez az osztály felel a ki és bejelentkeztetésért.
* *Encryptor*: Ez a statikus osztály tud egy string típusú adatot MD5 kódolási algoritmus segítségével titkosítani. Az *ActiveUserStore* igényli, hogy meg tudja csinálni a regisztrációt és bejelentkeztetést.
* *FlowElementProvider*: Ez az osztály felel a folyam összeállításáért/tárolásáért/frissítéséért.
* *ImageHandler*: Ez a statikus osztály képes egy képet bináris formára hozni, hogy azt fel lehessen tölteni az adatbázisba.

### Az osztályok részletezése

Terjedelemi okok miatt csak az alapvető osztályokat és metódusokat szeretném részletezni, valamint egy osztálytípusnál csak egy konkrét osztályt fogok bemutatni. A dokumentumban a Visual Studio 2019 által generált osztály-diagrammokat használom.

#### Program osztály

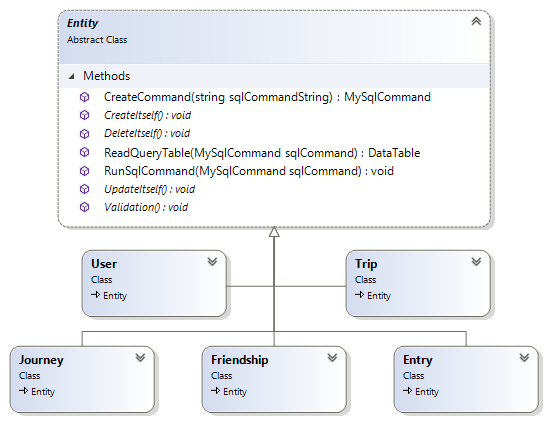
A statikus Program osztályunk tartalmazza a *Main* metódust. Ezen felül tartalmaz egy MainForm nevű statikus property-t, melyben a MainForm egy példánya lesz tárolva. Ennek csak a get ága lesz publikus, hogy a későbbiekben ne lehessen megváltoztatni. Rögtön a Program indulásakor megtörténik a MainForm példányosítása.

A Program osztály fog gondoskodni arról, hogy a MainFormunk aktiválásáról és deaktiválásáról. A Deaktiválásra akkor van szükség, ha egy párbeszédablak jelenik meg, természetesen aktiválni pedig akkor kell, ha bezáródik a párbeszédablak.

#### MainForm osztály

A MainForm osztály egy példánya lesz az alkalmazásunk főablaka. Egyetlen MainForm lesz az alkalmazás indulásakor, ami nem lesz módosítható a kód többi részében a private hatáskör miatt. \*\*\* ezt még jobban ki kell fejteni…

#### Entitások

Minden az adatbázisban szereplő sor egy-egy entitásnak felel meg. A különböző entitások hasonlóan viselkednek és ezért mind az Entity absztrakt osztályból származnak.

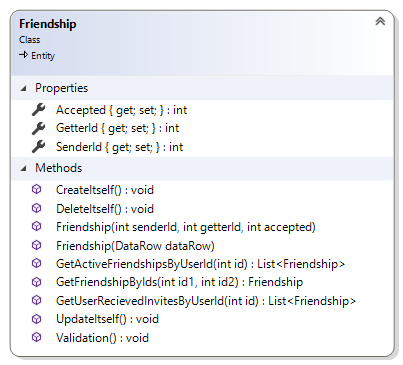
Az Entity osztály elvárja minden gyermek osztályától, hogy implementálja a *CreateItself, DeleteItself, UpdateItself, Validation* metódusokat. Ezek fogják megvalósítani az adott entitás alap adatbázis műveleteit. Ha manuálisan létrehozunk egy entitást, akkor a *CreateItself* segítségével tudjuk felvinni új sorként az adatbázisunkba. A *DeleteItself* metódus az adott entitást törli az adatbázisból. Ha egy entitáson módosításokat hajtunk végre, akkor az *UpdateItself* metódussal tudjuk az adatbázisban is módosítani az adott sort. A *Validation* metódussal meg tudjuk vizsgálni, hogy egy entitás az adatbázisnak megfelelő adatokat tartalmaz, ha bármi probléma adódik, akkor egy *ValidationException* fog dobásra kerülni.

Az Entity absztrakt osztály tartalmaz továbbá három statikus metódust is mely az adatbázishasználathoz szükséges gyakori lépéseket rendszerezi, hogy csökkentve legyen a kódban a duplikáció. A *CreateCommand* egy új *MySqlCommandot* hoz létre a megadott MySql parancs alapján, amit egy String paraméterként kap. A *RunSqlCommand* egy MySqlCommand futtatásáról gondoskodik. A *ReadQueryTable* pedig egy DataTable-t állít össze egy MySqlCommand alapján. Jellemzően akkor van használva, ha egy lekérdezést szeretnénk indítani.

##### Új entitás osztályok létrehozása

Minden entitásnak az Entity ősosztályból kell származnia, implementálnia kell annak absztrakt metódusati, valamint a következő adattagokkal, konstruktorokkal és metódusokkal kell őket kiegészíteni.

* Minden adatbázisban szereplő oszlopnak létre kell hozni egy megfelelő típusú és elnevezésű publikus property-t get és set ággal egyaránt.
* Készíteni kell egy konstruktort, ami **adatbázis** **lekérdezés** segítségével fogja előállítani egy entitás példányát. A paraméter egy DataRow lesz, és ennek a megfelelő értéke kerül beállításra az entitás megfelelő propery-jébe.
* Készíteni kell egy másik konstruktort, ami a **felhasználó** **által az adatbázisban** **létrehozandó új** entitások előállítására lesz fenntartva. Paraméterként meg kell kapnia az összes adattagot, kivéve az Id azonosítókat, mert ezeket az értékeket közvetlenül az adatbázis fogja beállítani.
* Dinamikus Get metódusokat, amik az adott entitáshoz szorosan köthető más entitásokkal térnek vissza az adatbázisból. Például egy utazás alá egy vagy több szakasz tartozhat, szükséges létrehozni egy olyan metódust, ami az adott utazáshoz tartozó szakasz entitások listájával tér vissza. Ezt a lista lesz felhasználva egy utazáshoz tartozó szakasz-folyam kialakításakor.
* Statikus Get metódusokat kell létrehozni a különböző adatbázis olvasással kapcsolatos feladatokhoz. Ezek vagy adott entitással térnek vissza, vagy adott entitás típusú listákkal. Az applikáció továbbfejlesztése esetében csak a megfelelő adatbázis lekérdezésekkel kell az osztályt bővíteni.

**Friendship osztály bemutatása:**

A Friendship osztály reprezentálja az adatbázisban, hogy a különböző felhasználók milyen baráti kapcsolatban vannak. Három mező található erre az adatabázis ’friends’ kapcsolótáblájában: a *sender\_id*, a *getter\_id*, valamint az *accepted*. A *sender\_id* azonosítja a barátkérelem elküldőjét, a *getter\_id* a barátkérelem célpontját, az *accepted* pedig azt, hogy a barátkérelem elfogadásra került-e. (0: még nem került elfogadásra, 1: elfogadásra került) Abban az esetben, ha elutasításra került egy barátkérelem, akkor a megfelelő sor törlésre kerül a táblából.

A Friendship osztálynak ennek megfelelően rendelkeznie kell típushelyes SenderId, GetterId, és Accepted property-kkel. Kell lennie egy írásra és egy olvasásra alkalmas konstruktornak. Valamint szükséges lesz a következő három metódusra.

* **GetFriendshipByIds**(int id1, int id2):Friendship:ezzel lehet keresni egy barátságra, ahol tudjuk a két felhasználó azonosítóját.
* **GetRecievedFriendshipInvitesByUserId** (int id): List<Friendship>: Ez a metódus egy adott felhasználóhoz tartozó olyan barátságokkal tér vissza, melyeket még nem fogadott el a felhasználó. Paraméterként a felhasználó azonosítóját kell megadni.
* **GetActiveFriendshipsByUserId** (int id): List<Friendship>: Ez a metódus egy felhasználó aktív barátságlistájával tér vissza. Paraméterként a felhasználó azonosítóját kell megadni.

Ha szeretnénk adott esetben egy új funkciót bevezetni, például hogy legyen kilistázva a flowban egy felhasználónak az összes általa elküldött barátkérelme, akkor csak létre kell hozni egy új statikus metódust, aminek a neve lehet **GetSentFriendshipInvitesByUserId,** ami szintén egy List< Friendship >-val tér vissza.

#### Formok

Az entitásokat két féle képpen lehet kezelni ­- gondolok itt azok létrehozására, törlésére, módosítására, - formok, vagy pedig kontrollok segítségével. Most a Form osztályokról szeretnék pár szót szólni.

Minden form hozzárendelhető egy entitáshoz, és rendelkezik olyan mezőkkel, vagy más felületi elemmel, melyek a felhasználó által megadandó információk bekérését teszik lehetővé. Minden entitáshoz két fajta form tartozik, az egyik az entitás létrehozásáért a másik az adott entitás frissítéséért felel.

A projekten belül a Form osztályok a Forms mappába vannak szervezve, valamint a az egy Entitáshoz tartozó osztályok az EntitásNeveForms mappában találhatóak.

Minden Formnak validálnia kell, hogy a bevitt adat megfelelő, erre a Validation metódust kell használni, ami meghívja az adott entitás validációját, mert minden entitás tudja, hogy mi a jó neki. Csak ezután történhet adatabázis utasítás végrehajtás.

Fontos, hogy már a beviteli és más mezők esetében is biztosítani kell, hogy lehetőleg csak a megfelelő formátumú és típusú adatok kerülhessenek bevitelre.

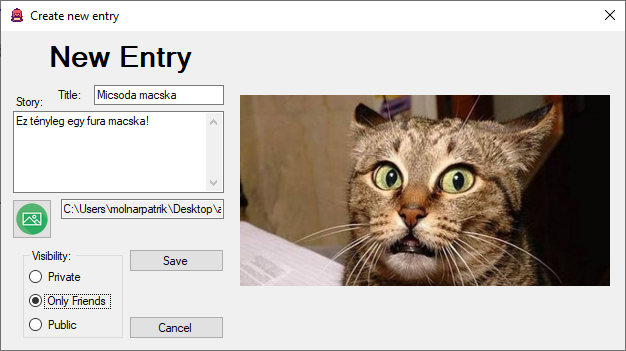
*Megjegyzem, hogy a MainForm, ami az alkalmazás fő ablaka, nem tartozik ehhez az osztálytípushoz.*

Természetesen minden Form a System.Windows.Forms.Form osztályt implementálja.

**Az Entry entitás Form osztályainak bemutatása:**

Az Entry entitásnak két Formja van az előzőekben tárgyaltak miatt.

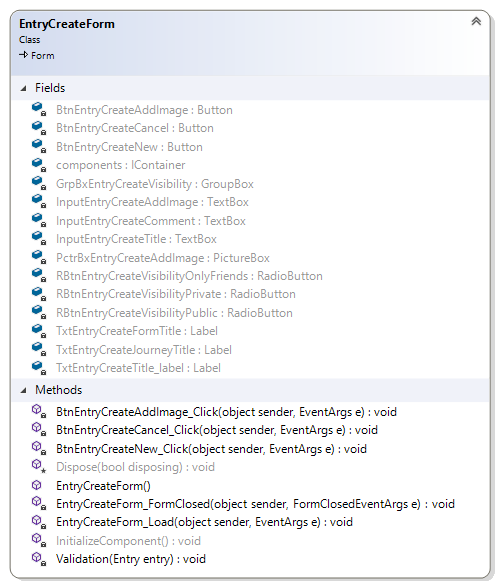
Az EntryCreateForm, valamint az EntryUpdateForm.

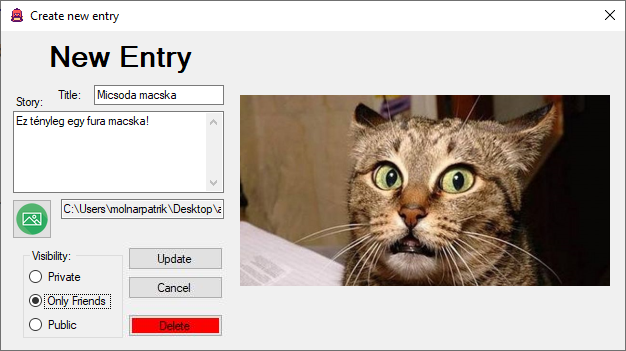
**Az EntryCreateForm:**

* **Title**: Ez az adott Entry címe, az adatbázisban az entries tábla title oszlopa fogja tárolni ezeket az értékeket.
* **Story**: Ez az adott Entry leírása, ide kerül az adott bejegyzés írásos része, az entries tábla comment oszlopa fogja tárolni.
* **Kép kiválasztása gomb:** Ennek a segítségével lehet egy megnyíló ablakban kiválasztani a feltölteni kívánt képet. Ez a kép fog az entries tábla megfelelő rekordjának picture oszlopában letárolódni.
* **Visibility:** Itt lehet beállítani az Entry láthatóságát, ez az entries tábla visibility oszlopába kerül.
* **Save gomb:** Ennek a segítségével validáció után történik meg az adatbázisba írás. Form bezárása.
* **Cancel gomb:** Az új bejegyzés megszakítása. Form bezárása.

**Az EntryCreateForm metódusai:**

Az EntryCreateFormnak csak paraméter nélküli konstruktora van. Ő maga fogja előállítani az Entry-t és a DB-be írni.

* **BtnEntryCreateCancel\_Click(object sender, EventArgs e):** Ez történik a *Cancel* gombra kattintás esemény hatására. Az új bejegyzés megszakítása kerül a Form pedig bezáródik.
* **BtnEntryCreateNew\_Click(object sender, EventArgs e):** Ez történik a *Save* gombra kattintás esemény hatására. Az új Entry validálásra kerül, majd új rekordként hozzáadásra kerül az adatbázis entries táblájához.
* **Validation(Entry entry):** Meghívja a paraméterként átadott entry validációs metódusát, ami bármilyen kritikus hiba esetében *ValidationException*-t dob. Ez meggátolja, hogy az adatbázisba fel legyen véve az új rekord. Ez jelen esetben konkrétan azt jelenti, hogy csak akkor valid egy entry, ha a title és a comment nem üres, valamint kötve van hozzá ID alapján egy trip.
* **BtnEntryCreateAddImage\_Click(object sender, EventArgs e):** Ez a metódus a Kép hozzáadása gombra kattintáskor fut le, ez felel egy OpenFileDialog megnyitásáért, valamint az elérési útvonal beállításáért, ami majd az adatbázisba íráskor lesz fontos.
* **EntryCreateForm\_Load(object sender, EventArgs e):** Ez a metódus a form betöltésekor végrehajtandó fontos lépésekre való, mint például a MainForm deaktiválása, hogy amíg nyitva van a Form ne lehessen a főablakban műveleteket végrehajtani.
* **EntryCreateForm\_FormClosed(object sender, FormClosedEventArgs e):** Ez a metódus a form bezárásakor végrehajtandó fontos lépésekre való, mint például a MainForm aktiválása, hogy újra lehessen műveleteket végrehajtani a főablakban.

**Az EntryUpdateForm:**

Szinte teljesen megegyezik az *EntryCreateForm* felépítésével, azonban itt nem *Save* gomb található, hanem *Update*, valamint van még egy további gomb is a felületen, ami a *Delete*. A további elemek megegyeznek.

Az osztálynak egy paraméteres konstruktora van, ami egy *Entry*-t vár. Ez az *Entry* már egy, a DB-ben lévő entitás, azt szeretnénk majd a form segítségével UPDATE-elni. Ez egy *Entry* nevű property-ben lesz eltárolva. A form mezőiben már ennek az entitásnak az értékei szerepelnek, ha ezeket módosítjuk, majd az Update gombra kattintunk, akkor lefut az *Entry* validációja, és ha nincs probléma megtörténik a DB UPDATE.

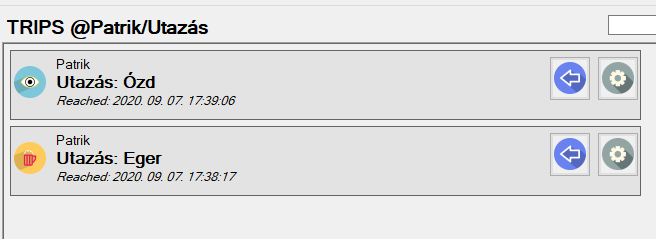
A konstruktornak ezen kívül biztosítania kell, hogy ha az *Entry* egy lezárt *Journey*-hez tartozik, akkor csak a *Visibility* értéke módosítható, a többi érték nem lehet változtatható. Ezt az adott mező lezárásával kell elérni.

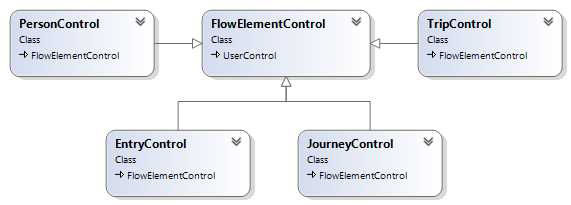
Az Update gombra kattintáskor a BtnEntryUpdateUpdate\_Click(object sender, EventArgs e) metódus fut le. Ez egyébként nagyon hasonlít a *Save* gombhoz, annyiban különbözik, hogy nem egy CREATE sql parancs fut le ha megtörtént a validáció, hanem egy UPDATE.

A Delete gombra kattintva lefut a BtnEntryUpdateDelete\_Click(object sender, EventArgs e) szignatúrájú metódus, ami az adott *Entry* entitás *DeleteItself()* metódusával törli az adott rekordot az adatbáziaból. Majd egy MainForm frissítés után bezáródik.

A további metódusok hasonló feladatokat valósítanak meg, mint az EntryCreateFormé**.**

#### Kontrollok

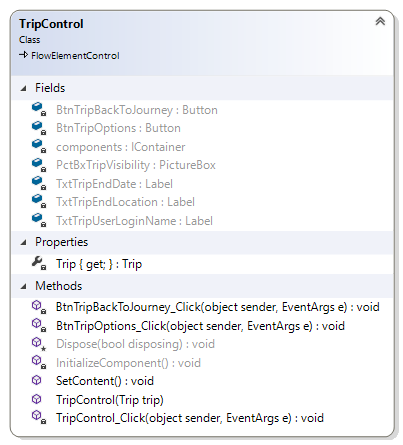
A DiceTravelben található kontrollok, azok a Flow elemeit jelentik. Hasonlóan a Formokhoz a Kontrollok szintén interaktív felületek, melyek megjelenítik az adott entitások legfontosabb információit, valamint lehetővé teszik bizonyos entitásokhoz köthető funkciók végrehajtását. *Ilyen például egy adott felhasználó barátnak jelölése, vagy, hogy megtekintsük, hogy egy adott Journey alatt milyen Tripek találhatóak.*

Minden kontroll a *FlowElementControl* osztályból származik, mely leginkább vizuális effekteket valósít meg.

Nincs kontroll valamilyen entitás nélkül. Például a Trip kontroll minden eseteben tartalmaz egy Tripet.

A mezők többnyire szöveges információt jelenítenek meg, rendelkeznek egy vagy több funkciógombbal, valamint legtöbbjükre rá is lehet kattintani. Nincsenek beviteli mezőik.

**A TripControl osztály bemutatása:**

****A **TripControl** osztály egy a flowban megjelenő kontrollnak az implementációja, melynek példányai különböző tripeket fognak reprezentálni, valamint azok funkcióit elérhetővé tenni a felhasználó számára.

A trip kontrollon szerepel a trip birtokosának felhasználóneve, a Journey neve, a trip következő állomása, valamint egy jel, ami a láthatóságot szimbolizálja. Ezeken kívül egy vissza gomb, valamint egy beállítások gomb.

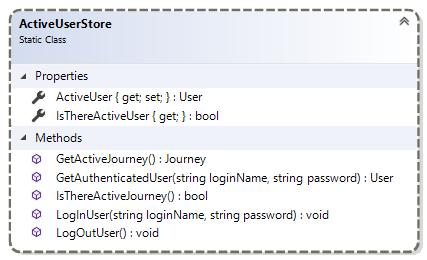
A Trip kontroll rendelkezik egy csak olvasható *Trip* típusú property-vel, melyet az egyetlen konstruktor fog beállítani a kapott *Trip* típusú paraméter alapján. A konstruktor ezen kívül beállítja még a kontrollon megjelenő adatokat a *SetContent()* metódus segítségével, valamint a kontroll stílusát is meghatározza a *Trip* alapján.

A kontrollra kattintva megnyílik az adott triphez tartozó *bejegyzés-folyam*. A vissza gombra kattintva visszaléphetünk az adott felhasználó *utazás-folyamjára*, a beállításokkal pedig az adott triphez tartozó *TripUpdateForm*-ja jelenik meg.

#### Kiegészítő osztályok

##### ActiveUserStore

Ez a statikus osztály gondoskodik a felhasználók be- és kijelentkeztetéséről, tárolásáról.

Az osztály az ActiveUser property-ben tárolja a bejelentkezett felhasználót. Ha ez az érték null, akkor nincs senki sem bejelentkezve a rendszerbe.

Annak eldöntésére, hogy valaki be van-e jelentkezve az IsThereActiveUser csak olvasható property adja vissza.

A LogOutUser() metódus segítségével tudjuk kijelentkeztetni a felhasználót, ilyenkor az ActiveUser property értéke null lesz, valamint lefut a FlowElementProvider ResetFlow() metódusa, ami alaphelyzetbe állítha a Flow-t.

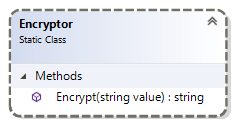
A LogInUser(string loginName, string password) metódus fogja a felhasználót beléptetni a rendszerbe, paraméterként a belépési nevet és jelszót kapja meg. Itt jelszó értéke már sózva és titkosítva van az Encryptor osztály segítségével.

A GetAuthenticatedUser(string loginName, string password) metódus visszatér egy User entitással abban az esetben, ha a felhasználónév, valamint a hozzá tartozó jelszó megfelelő. Ez egy private metódus, és egyedül a LogInUser metódus hívja.

A GetActiveJourney() metódus az épp bejelentkezett felhasználó aktív utazásával tér vissza.

Az IsThereActiveJourney() metódus igaz/hamis értékkel tér vissza, attól függően, hogy a bejelentkezett felhasználónak van-e aktív utazása. Azért készítettem ezt a metódust, hogy próbáljam növelni a kódom olvashatóságát.

##### Encryptor

Az Encryptor osztály felel a jelszavak titkosításáért. Egy metódust tartalmaz csak, ez az Encrypt(string value). Ennek a visszatérési értéke lesz a titkosított String típusú adat. Három helyen lesz használva, a regisztrációs felületen, a bejelentkezésnél, valamint a jelszó változtatásnál.

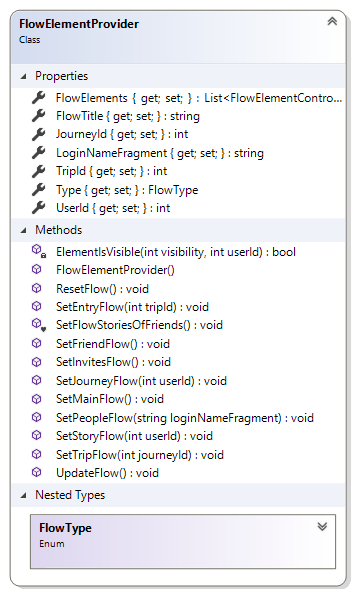
##### FlowElementProvider

A FlowElementProvider felelős a folyamban megjelenő elemek létrehozásáért, a flow összeállításáért és megjelenítéséért.

A MainForm rendelkezik egy példánnyal belőle, ami rögtön a MainForm példányosításakor jön létre.

A FlowElementProvider rendelkezik egy Enummal, ami az összes folyamtípust azonosítja, ez a FlowType.

A következő propertiket kell megvalósítani az osztálynak:

* Type: A Flow típusát tárolja, hogy a flow frissítése esetén emlékezzen, hogy mit kelll előállítania.
* UserId: A Flowban megjelenítendő elemekhez tartozó *UserId*-t tárolja. Abban az esetben, ha olyan flow kerül előállításra, ahol ez nem értelmezhető -1 értéket kap.
* JourneyId: A Flowban megjelenítendő elemekhez tartozó *JourneyId*-t tárolja. Abban az esetben, ha olyan flow kerül előállításra, ahol ez nem értelmezhető -1 értéket kap.
* TripId: A Flowban megjelenítendő elemekhez tartozó *TripId*-t tárolja. Abban az esetben, ha olyan flow kerül előállításra, ahol ez nem értelmezhető -1 értéket kap.
* FlowTitle: A folyam megjelenítendő címe kerül tárolásra ebben a property-ben.
* LoginNameFragment: A felhasználó keresésekor felhasznált szótöredék kerül eltárolásra ebben a property-ben.
* FlowElements: A Flowban megjelenítendő kontrolok kerülnek itt eltárolásra. Ennek beállítására csak a FlowElementProvidernek van jogosultsága.

A konstruktor egyedül a ResetFlow() metódust tartalmazza, ami a flow alaphelyzetbe állításáért felel.

Az UpdateFlow() metódus felel a folyam frissítéséért. Ha felhasználó a frissítés gombra kattint, vagy bármikor, amikor frissíteni kell a folyamot, ez a metódus fut le. Az eltárolt értékek alapján újra generálja a flowt.

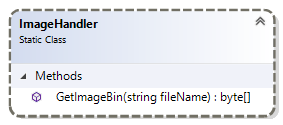
Egy nagyobb metódus fajta a folyambeállító metódusok.

A következő Flow beállító metódusokra van szükség:

* SetMainFlow(): A folyamnak a főfolyamot állítja be, ahol a bejelentekezett felhasználó ismerőseinek a 3 napnál korábbi bejegyzései jelennek meg.
* SetPeopleFlow(string loginNameFragment): A folyamnak Felhasználó-folyamot állítja be, ahol a kereső mezőben megadott feltételnek megfelelő felhasználók jelennek meg.
* SetFriendFlow(): Az bejelentkezett felhasználó Barát-folyama jelenik meg.
* SetJourneyFlow(int userId): Az adott felhasználó Utazás-folyama jelenik meg.
* SetTripFlow(int journeyId): Az adott utazás Szakasz-folyama jelenik meg.
* SetEntryFlow(int tripId): Az adott Szakasz Bejegyzés-folyama jelenik meg.
* SetStoryFlow(int userId): Az adott felhasználó Történet-folyama jelenik meg.
* SetInvitesFlow(): A bejelentkezett felhasználó Jelölés-folyama jelenik meg.

A kiegészítő ElementIsVisible(int visibility, int userId) metódus egy igaz/hamis értékkel tér vissza attól függően, hogy az adott felhasználó láthatja-e az adott láthatósági szintet. Ennek a segítségével lehet kiszűrni azokat a kontrollokat, amiket nem láthat a felhasználó.

##### ImageHandler

Ez a statikus osztály képes egy képet bináris formára hozni, hogy azt fel lehessen tölteni az adatbázisba.

A GetImageBin(string fileName) metódus egy elérési út alapján egy képet bináris többé alakít. Ez a bináris formátum lesz majd az adatbázisban letárolva.