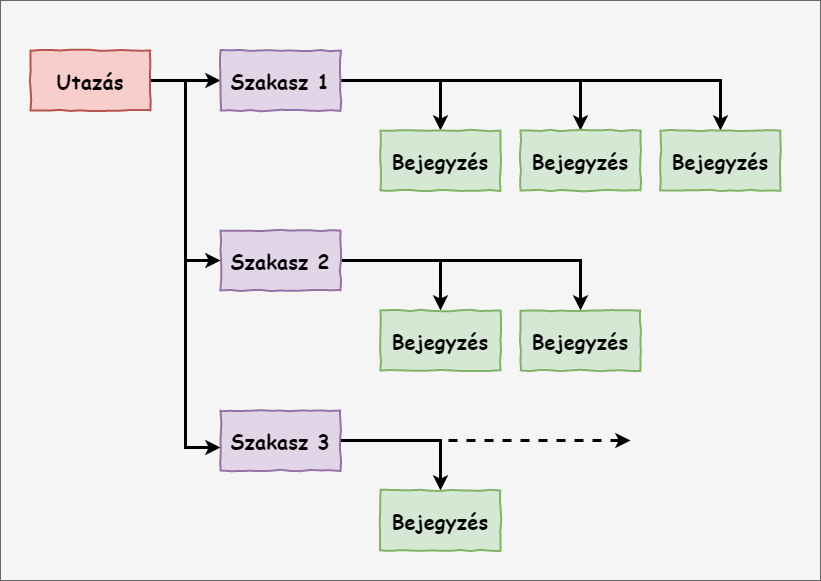
# 3. Fejlesztői kézikönyv

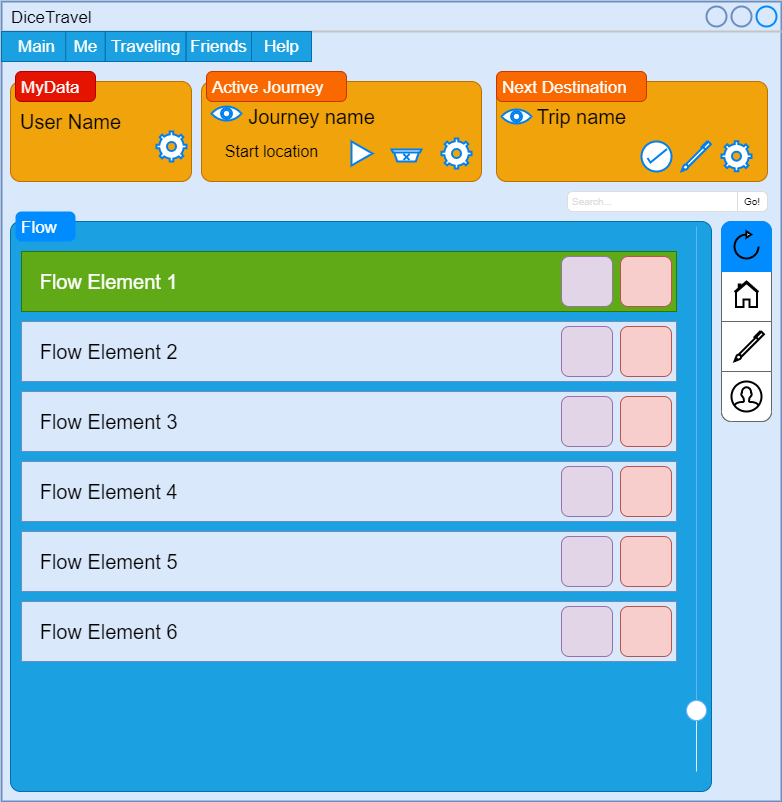
## A DiceTravel koncepciója

### 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.

Az alábbi képen az alkalmazás főablakának a terve látható:

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. Például a felhasználó utazásai, más felhasználók utazásai, vagy egy felhasználó barátai.

### 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

### 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

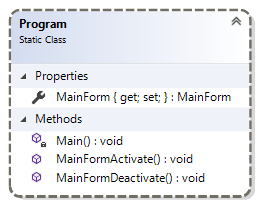
A Util mappán belül a különböző kiegészítő osztályok szerepelnek. Ebből három van:

* *ActiveUserStore*: Ebben a statikus osztályban van letárolva az éppen bejelentkezett felhasználó, valamint ez az osztály felel a ki és bejelentkeztetésért.
* *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

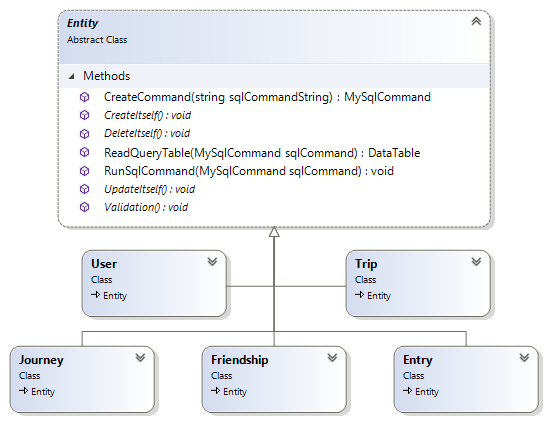
A statikus Program osztályunk tartalmazza a *Main* metódust. Ezen felül tartalmaz egy MainForm nevű statikus property-t, melyben a MainForm egy példánya lesz tárolva. Ennek csak a get ága lesz publikus, hogy a későbbiekben ne lehessen megváltoztatni. Rögtön a Program indulásakor megtörténik a MainForm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

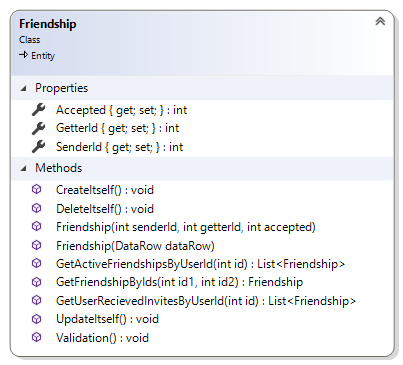
Az Entity osztály elvárja minden gyermek osztályától, hogy implementálja a *CreateItself, DeleteItself, UpdateItself, Validation* metódusokat. Ezek fogják megvalósítani az adott entitás alap adatbázis műveleteit. Ha manuálisan létrehozunk egy entitást, akkor a *CreateItself* segítségével tudjuk felvinni 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 >-pel tér vissza.

Tartalom

[3. Fejlesztői kézikönyv 1](#_Toc47612770)

[A DiceTravel koncepciója 1](#_Toc47612771)

[Mi is a DiceTravel? 1](#_Toc47612772)

[Kulcsszavak 2](#_Toc47612773)

[A DiceTravel felépítése 2](#_Toc47612774)

[Osztálytípusok 2](#_Toc47612775)

[Az osztályok részletezése 3](#_Toc47612776)