Első feladatod az EN-CO-nál

# Feladatkiírás

A feladatod egy (sportcélú) repülések GPS logfájljait feldolgozó és megjelenítő webalkalmazás készítése lesz. Az alkalmazás segítségével a pilóták feltölthetik a repüléseik logjait egy felületre, ahol azok automatikusan feldolgozásra kerülnek, majd egy adminisztrátor jóváhagyhatja őket.

A feladat megoldása során az alábbi témákkal fogsz találkozni: szöveges-, és Excel fájlok feldolgozása, adatbáziskezelés (Entity Framework, LINQ), felhasználókezelés (Identity), background processing, illetve a kliens oldalon lesz egy kis JS ☺ A site kinézete a feladat elfogadása szempontjából nem lényeges, a példa célja az éles fejlesztés során gyakran előforduló backend-es taszkok gyakorlása. Az alkalmazás készülhet .NET Framework, vagy .NET Core felé is, ezt rád bízzuk ☺

A rendszer pontos funkcióit menükre bontva az alábbiakban találod, illetve a dokumentum végén található egy kis segítség a fájlfeldolgozáshoz is.

# Szerepkörök

## Pilóta

Az adott repülés pilótája, aki azért lép be a rendszerbe, hogy feltöltse a repülését. Bejelentkezést igénylő szerepkör.

## Adminisztrátor

Ő tudja jóváhagyni azokat a repüléseket, amelyeket a pilóták feltöltöttek. Bejelentkezést igénylő szerepkör.

# Menük, és leírásuk

## Pilóta által elérhető menük

### Logfájl feltöltés

Az oldalon csak egy fájlfeltöltési lehetőség található. A felhasználó kitallózza az általa választott fájlt, majd a feltöltés gombra kattint. Ennek hatására a fájl feltöltődik a szerverre, erről pedig egy üzenetben értesül a felhasználó.

### Repüléseim

Egy táblázatban megjelennek a felhasználó által feltöltött repülések adatai. A táblázathoz a legegyszerűbb megoldást válaszd, nem szükséges rendezni, lapozni.

A táblázat egy sorában az alábbi mezők találhatóak:

* Repülés dátuma
* Időtartam (milyen hosszú volt a repülés)
* Felszállás helye
* Leszállás helye
* Státusz (elfogadásra vár, elfogadott, vagy elutasított)
* Gomb a részletes nézetre navigáláshoz

### Repülés részletes nézete

A részletes oldalon megjelennek a táblázatban is meglévő mezők, valamint egy térképen megjelenik a log. (a GPS fixeket összekötő törtvonalként) A térképszolgáltatót tetszőlegesen megválaszthatod (Google Maps, Bing Maps, OpenStreetMap)

## Adminisztrátor által elérhető menük

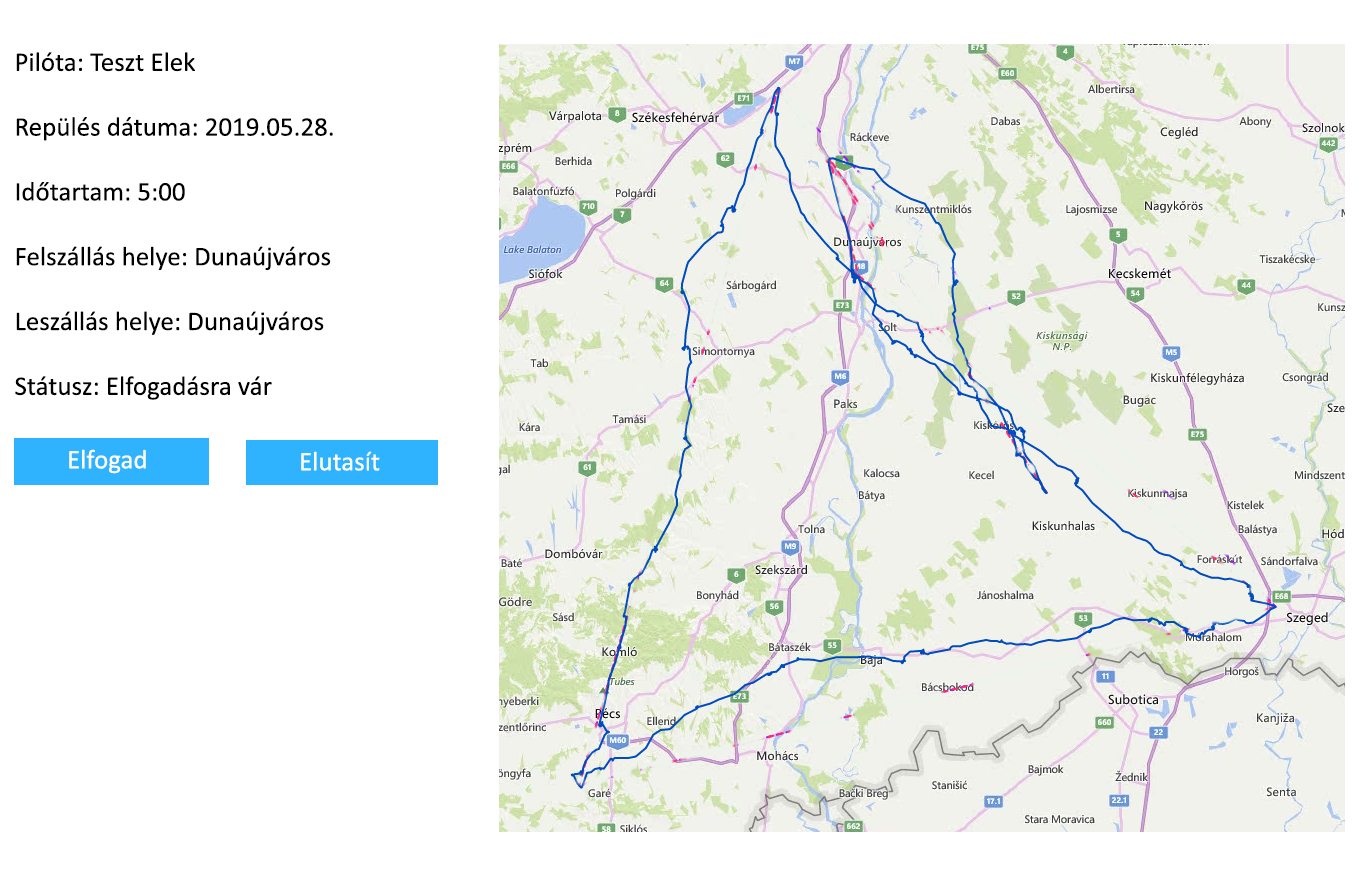
### Repülések

Az oldalon található egy dropdown, aminek elemei: „elfogadásra vár, elfogadott, elutasított). A dropdown kiválasztott értékének megváltozásával a dropdown alatt lévő táblázatban mindig az adott státuszú repülések jelennek meg. Alapértelmezetten az elfogadásra váró repülések jelennek meg. A táblázat elemei megegyeznek a fenti listában leírtakkal, de kiegészülnek a pilóta nevével.

### Repülés részletes nézete

A részletes oldalon megegyezik a pilótánál leírtakkal, azzal a különbséggel, hogy amennyiben „Elfogadásra vár” státuszú a repülés, akkor megjelenik egy „Elfogadom” és egy „Elutasítom” gomb is. Amennyiben ezek valamelyikére kattint az admin, akkor a repülés automatikusan a megfelelő állapotba kerül, az oldal pedig újratöltődik, és már az új státusz jelenik meg.

Valahogy így kellene kinéznie az oldalnak:



### Repülőterek

Az admin a repüléseken kívül egy repülőtér adatbázist is kezel. Ennek majd a repülés feldolgozása során lesz szerepe. Ebben a menüpontban töltheti fel ezt az adatbázist. Az adatok egy Excel fájlból származnak, mely minden repülőtérről az alábbiakat tartalmazza:

* Név
* Szélességi koordináta
* Hosszúsági koordináta

Az oldal ugyanúgy néz ki, mint a pilóta logfájl feltöltő oldala, csak egy fájlfeltöltési lehetőség található rajta. A feltöltés gomb megnyomására feltöltődik az Excel, az alkalmazás pedig feldolgozza a fájlt, és adatbázisba menti a repülőtereket. A feldolgozás után a képernyőn megjelenik a folyamat sikerességét jelző szöveg.

# GPS log feldolgozása

Most, hogy minden készen áll a repülés fogadására, jöhet az izgalmas rész ☺

A fájlfeltöltésnél említésre került, hogy a feltöltés gomb megnyomására a fájl csak felkerül a szerverre, de más nem történik fele. A tényleges feldolgozást ezután egy háttérfolyamat végzi, amit tetszőleges background process library felhasználásával valósíthatsz meg (pl. Hangfire). A folyamat adott időközönként lefut (érdemes a teszteléshez minél rövidebb időre állítani, hogy hamar meghívódjon) és megnézi, hogy van-e feltöltött fájl, amit fel kell dolgoznia. *(tipp: a fájl feltöltésekor keletkezzen egy feldolgozási sorként használt db táblába egy rekord, ahol nyilvántartod, hogy milyen repüléseket kell feldolgozni. A feldolgozó az itt lévő repülésekre néz rá, ha pedig egy feldolgozás kész, akkor az adott rekord törölhető. A process így minden futásánál feldolgozza a sorban lévő repüléseket, és kiüríti a táblát*.) Amíg nincs feldolgozva a repülés, addig se a pilóta, se az admin nem látja a listaoldalakon.

Mit kell csinálnia a feldolgozó függvénynek?

A GPS log egy .igc kiterjesztésű fájl, ami egy sima szövegfájl, nyisd meg Notepad-del, pörgess bele nyugodtan. Sokféle információt tartalmaz, de ezek közül neked csak a következő fejezetekben leírt 2 féle sor lesz érdekes a feladat megoldásához.

## Dátum rekord

A fájl második sora valahogy így fog kinézni, ez adja meg a repülés dátumát:

HFDTE250519

A sor betűi alapján találhatod meg a fájlban. A számok azonosítják a napot, amikor a log készült. A formátum DDMMYY, tehát ebben az esetben 2019.05.25.

## GPS fix rekord

Ez tartalmaz egy adott GPS fixet, tehát azt, hogy adott pillanatban hol volt a repülő. Ezekből fogod majd tudni rekonstruálni a repülést a térképen.

Egy ilyen „B” rekord így kéz ki:

**B1040334619875N01844550E**A006740071501609

**B**: Ez azonosítja, hogy egy olyan sorról beszélünk, ami egy GPS fixet tartalmaz. Gyakorlatilag a dátum rekordon kívül minden mást eldobhatsz a feldolgozás során, ami nem B betűvel kezdődik. Bár a „B” rekordok időrendben vannak a fájlban, de közéjük ékelődhetnek egyéb rekordok is, erre figyelj.

**104033:** Ez adja meg az UTC időt, amikor a fix készült. Feloldása: 10 óra 40 perc 33 másodperc. A „Repülés időtartama” mezőt az egyszerűség kedvéért számíthatod az első és utolsó „B” rekord időbélyege alapján.

**4619875N:** A szélességi koordináta. Feloldása: 46 fok 19.875 szögperc, észak.   
**NOTE:** Neked a „klasszikus”, tized fokos formátumra van szükséged, hogy tizedes törtként tudd tárolni a számot. Ha már rég lett volna a gimis geometria óra, akkor megsúgjuk, hogy 1 fok = 60 szögperc ☺ Ha szeretnéd ellenőrizni az átszámítást, akkor itt megteheted: <https://www.directionsmag.com/site/latlong-converter/>

**01844550E:** A hosszúsági koordináta. Feloldása 018 fok 44.550 szögperc, kelet. Mint látható, a fok érték mindig 3 karakteren van megadva, akkor is, ha 100-nál kisebb számról beszélünk. Feldolgozása a fentiekben leírtaknak megfelelően történik.

A koordináta értékekben az „N” és „E” betűket veheted fixnek, csak magyarországi fájlokat kapsz próba input fájlként, a Föld másik féltekével nem kell foglalkozni.

A rekord további része olyan adatokat tartalmaz, amelyek nem szükségesek a feladat megoldásához.

# Repülőterek használata

Ahogy már említésre került, a repülőtereket az admin egy Excel importtal viheti fel a rendszerbe. Az Excel feldolgozásához ajánljuk az „EPPlus” nevű nuget csomagot, nagyon egyszerű a használata. A feldolgozás során mentsd le db-be az Excel-ben talált reptereket.

A GPS log feldolgozása során – mint azt már valószínűleg kitaláltad – azért lesz szükség a repülőterek koordinátáira, mert ebből fogod tudni megállapítani a felszállás és leszállás helyét. Ehhez a következő szabályrendszert kell alkalmazni:

* Felszállás helye: Az a repülőtér, amelynek a koordinátája 3km-nél közelebb van a log első fixéhez. (mindig pontosan 1 ilyen lesz)
* Leszállás helye: Ugyanez a szabály, ám néha előfordulhat, hogy egy repülő nem hivatalos repülőtéren, hanem „terepen”, pl. egy szántóföldön száll le. Ha nem találsz repteret a log utolsó fixéhez, akkor kerüljön a felületre egy „Terep” kiírás.

*Help: a távolságot az egyszerűség kedvéért gömb mentén, nem geoidon számoljuk. Ehhez az alábbi rövid függvényt tudod használni. (a bemenő paraméterek custom class-ok, tulajdonképpen csak szélesség-hosszúság párok)*

public double CalculateDistance(GlobalPoint start, GlobalPoint end)

{

double rlat1 = Math.PI \* start.Latitude / 180;

double rlat2 = Math.PI \* end.Latitude / 180;

double theta = start.Longitude - end.Longitude;

double rtheta = Math.PI \* theta / 180;

double dist =

Math.Sin(rlat1) \* Math.Sin(rlat2) + Math.Cos(rlat1) \*

Math.Cos(rlat2) \* Math.Cos(rtheta);

dist = Math.Acos(dist);

dist = dist \* 180 / Math.PI;

dist = dist \* 60 \* 1.1515;

return dist \* 1609.344; // Méterben adja vissza a távolságot

}

# Technikai elvárások

A végére még néhány dolog, amit szeretnénk viszontlátni a megoldásodban:

Szeretnénk, ha a projekt jól strukturált lenne. Ehhez az alábbi projekteket kell létrehoznod a solution-ben:

* Model: Adatbázis entitások (code first db-t kérünk szépen ☺)
* DAL: DB elérés (ApplicationContext, DbSet-ek)
* BLL: Minden üzleti logika, db-hez Manager osztályok, DTO-k
* Web: ViewModel-lek (hozzájuk AutoMapper használata ajánlott), Controllerek, UI, stb.

Egyebek:

* Dependency injection használata kötelező Autofac-kel, vagy .NET Core esetén a beépített IoC Containerrel.
* A routing tekintetében az alapértelmezett Controller/Action/Id megoldás helyett érdemes attribute routing-ot használni.
* A db-ben azonosítóként mindig Guid-ot használj, ne intet.
* Minden más, pl. unit of work, repository használata szabadon válaszható, ha az kényelmesebb, de nem elvárás.
* A kód legyen szépen kommentezett, ha egy osztály, vagy függvény célja nem triviális, akkor legyen rajta summary.

**A munkához sok sikert kívánunk! Ha kérdésed van, fordulj bátran a mentorodhoz! ☺**