Dokumentáció

**Rendszerterv**

*Jelölő, szavazó rendszer készítése*

*Szoftverarchitektúrák (BMEVIAUM105) tárgy házi feladat*

*Konzulens: Simon Gábor*

Készítették:

Barta Ágnes

Cseppentő Lajos

Tartalom

[A rendszer célja, funkciói és környezete 4](#_Toc405467772)

[Feladatkiírás 4](#_Toc405467773)

[A rendszer által biztosítandó tipikus funkciók 4](#_Toc405467774)

[A program környezete 5](#_Toc405467775)

[Megvalósítás 6](#_Toc405467776)

[Architektúra 6](#_Toc405467777)

[Adatbázis réteg 7](#_Toc405467778)

[Adatdefiníciók 8](#_Toc405467779)

[Adatelérési réteg 8](#_Toc405467780)

[Üzleti logikai réteg 8](#_Toc405467781)

[REST Service 8](#_Toc405467782)

[Modellek az adatkötéshez 9](#_Toc405467783)

[Film és sorozat adatok betöltése 10](#_Toc405467784)

[Grafikus felület (Kliens) 10](#_Toc405467785)

[Felhasználói grafikus felület 10](#_Toc405467786)

[Adminisztrátori grafikus felület 16](#_Toc405467787)

[Adat- és adatbázisterv 20](#_Toc405467788)

[A program objektum modellje 20](#_Toc405467789)

[Az alapobjektumok 20](#_Toc405467790)

[Objektumok közötti kapcsolatok 21](#_Toc405467791)

[POCO objektumok tárolása 21](#_Toc405467792)

[Az adatbázis 22](#_Toc405467793)

[Administrator 23](#_Toc405467794)

[User 23](#_Toc405467795)

[Poll 23](#_Toc405467796)

[Nomination 23](#_Toc405467797)

[Vote 24](#_Toc405467798)

[PollSubject 24](#_Toc405467799)

[News 24](#_Toc405467800)

[GUI terv 24](#_Toc405467801)

[Hitelesítés és biztonság 25](#_Toc405467802)

[Tesztelés 25](#_Toc405467803)

[A program készítése során felhasznált eszközök 27](#_Toc405467804)

[Összefoglalás 28](#_Toc405467805)

[Továbbfejlesztési lehetőségek 29](#_Toc405467806)

[Telepítési leírás 30](#_Toc405467807)

[Hivatkozások 31](#_Toc405467808)

# A rendszer célja, funkciói és környezete

## Feladatkiírás

Egy olyan weboldal készítése a feladat, amellyel a jelölős-szavazós-díjnyerős események kezelhetők, pl. a rajongói sorozatos díjak (pl. <http://www.sorozatjunkie.hu/tag/jaws/>). A jelölési fázisban nem egy listából kell választani (mivel az túl hosszú lenne), hanem szövegdobozos beírással, amit validáció illetve auto-complete segít. A szavazási fázisban már csak a legtöbb szavazatot kapott jelöltek közül lehet választani. Lehessen követni a szavazás állását és a rendszer lehetőleg védje ki az egyszerűbb csalási módszereket

## A rendszer által biztosítandó tipikus funkciók

A projekt során célunk egy olyan jelölő-szavazó alkalmazás készítése, amely segítségével a felhasználó képes filmeket és sorozatokat jelölni egy adott szavazásra, majd a szavazás megnyílása után képes ezekre szavazni.

A program fő funkciója a jelölési és szavazási folyamat megvalósítása. A folyamat a következő lépésekből áll:

* Egy adminisztrátor kiír egy jelölés-szavazást, melyet az összes látogató lát.
* A bejelentkezett felhasználók jelölhetnek a jelölési határidőig:
  + Meg kell nevezniük a jelöltet, valamint opcionálisan egy rövid indoklást is írhatnak.
  + Egy felhasználó tetszőleges számú filmet vagy sorozatot jelölhet egy szavazásra.
  + Jelölni csak létező filmet, ill. sorozatot lehet.
  + A jelölési időszak végéig a felhasználó módosíthatja a saját jelöléseit és akár törölheti is azokat.
* Jelölések összegzése, amely magában foglalja:
  + a beküldött jelölések felülvizsgálatát,
  + az irreleváns jelölések törlését és
  + az esetleges duplikátumok kiszűrését.
* Szavazás, ami magában foglalja:
  + a jelöltek nyilvánosságra hozását,
  + a szavazás aktiválását és
  + a szavazatok leadását, amely során minden bejelentkezett felhasználó pontosan egy jelöltre szavazhat.
* Szavazás lezárása:
  + Szavazni egy előre megadott időpontig lehet.
  + A határidő letelte után inaktívvá válik a szavazás.
* Eredményhirdetés:
  + Az eredményhirdetési időpont után a szavazás eredménye az összes látogató számára láthatóvá válik.

Az alkalmazásnak lehetővé kell tennie, hogy egyszerre párhuzamosan tetszőleges számú aktív jelölési-szavazási folyamat lehessen a rendszerben. Emellett a szoftverben egyéb funkciókat is meg kell valósítani, melyek a következők:

* Alapszintű védelem csalás ellen:
  + Facebook fiókos felhasználói beléptetés alkalmazása.
  + Gyanús felhasználói fiókok adminisztrátor általi kitiltásának a lehetősége.
* Hírek:
  + Lehessen híreket kiküldeni a kezdőoldalra, melyeket a látogatók szabadon olvashatnak.
* „Gyakran ismételt kérdések” (GYIK) és „Kapcsolat” címmel két statikus oldal.

## A program környezete

A definiált alkalmazás felhasználói felülete *HTML5* alapokon készült el, így bármely modern böngészőből és okostelefonról is használható, ezzel biztosítva, hogy minél hatékonyabban minél szélesebb célközönséget érjünk el. A backend egy *adattárolási rétegből* és egy *REST API*-t kínáló webes csatlakozási pontból áll, utóbbit *.NET* platformon implementáltuk. Az alkalmazás a *Microsoft Azure* felhőben képes futni, garantálva a szoftver gyors skálázhatóságát és az üzemeltetés költséghatékonyságát. Az adattároláshoz a *Microsoft Azure* által nyújtott szolgáltatásokat használtuk.

# Megvalósítás

Az alkalmazást a feladatkiírásnak megfelelően egy többrétegű alkalmazásként készítettük el. Az általunk elkészített programot „Nominate and Vote” névre kereszteltük, utalva a program funkcionalitására. Az elkészült program egy kliens-szerver architektúrát valósít meg.

A fejezetben áttekintést adunk a program architektúrájáról, bemutatjuk az egyes komponensek feladatait és felelősségeit, továbbá részletesen ismertetjük a használt adatmodellt és a grafikus felhasználói felület felépítését.

## Architektúra

A rendszer architektúrája az 1. ábrán látható. A felhasználó interneten keresztül tudja elérni az alkalmazást felhasználói felületét. Az alkalmazás a Microsoft Azure [[1]](#_Hivatkozások) felhőben fut. Az adatok tárolására a Table Storage [[2]](#_Hivatkozások) szolgáltatását használjuk a felhőnek, a webes API megvalósítására pedig az Azure Web Role-ját. Az előbbieken kívül még használunk egy klienst, ami a kliens oldali kódot futtatja. A rendszer előnye, hogy a felhő használatának köszönhetően könnyen skálázható rendszert kaptunk. A rendszer működése során a kommunikáció az ábrán látható piros vonalak mentén történik. A felhasználó a klienssel kommunikál, ami a Web Role web API-ját hívogatja, ami eléri a Table Storage-ban található adatokat.



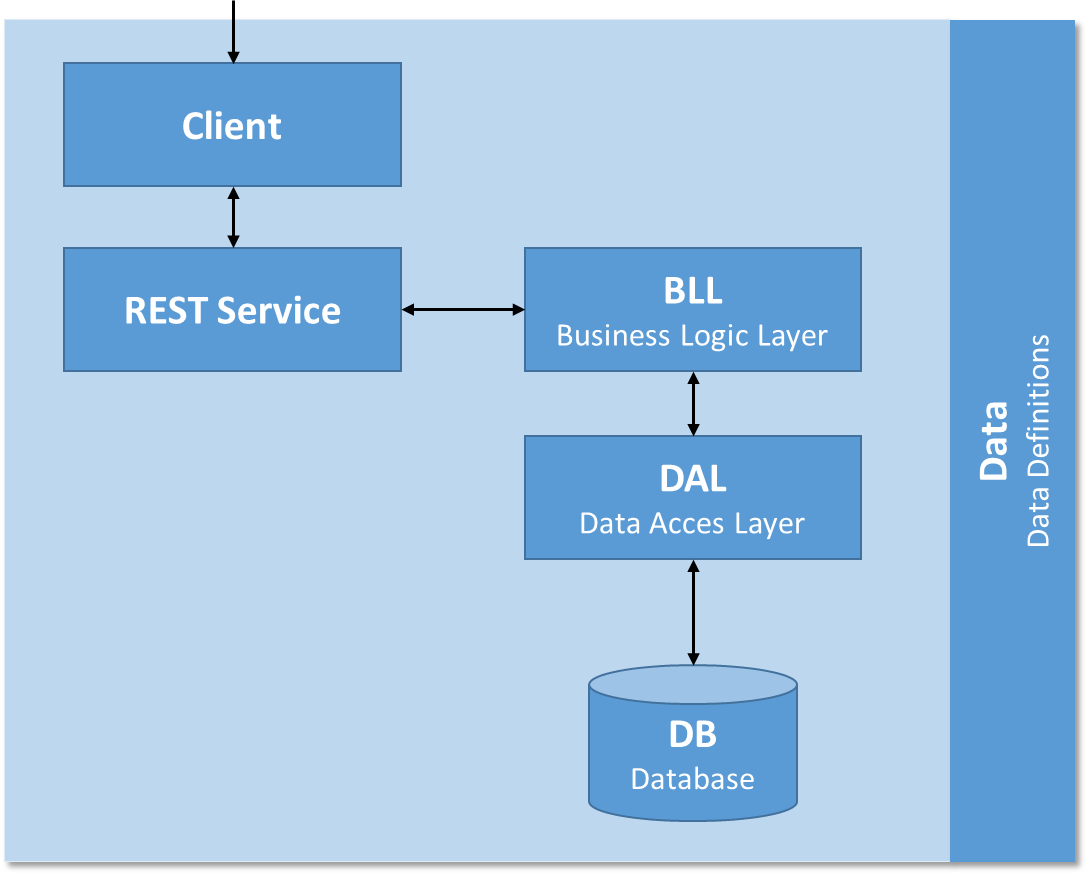
1. ábra Rendszer architektúra

A Nominate and Vote architektúrája 4 különálló modulra bontható, amik a klasszikus N-rétegű alkalmazás architektúrának felel meg:

* adatbázisréteg (Database Layer, DB)
* adatelérési réteg (Data Access Layer, DAL)
* üzleti logikai réteg (Business Logic Layer, BLL)
* kliens (Client)

A fentieken kívül még definiálnunk kellett a rendszerben használandó adat típusokat. A rendszer tervezése során alkalmaztuk az MVVM architektúra mintát.

Az egyes komponensek kapcsolatait mutatja be a 2. ábra. A fejezetben a fenn említett komponenseket mutatjuk be részletesen.



2. ábra A szoftver architektúrája

## Adatbázis réteg

Az adatbázis réteg felel az adatok tárolásáért. Erre a célra mi a Microsoft Azure Table Storage rendszerét választottuk. A választást következők indokolják:

* Az alkalmazás felhőben való futtatása lehetővé teszi, hogy a rendszer skálázható legyen.
* Elterjedt a gyakorlatban is gyakran alkalmazott megoldás.
* Az Azure biztosít ingyenes kipróbálási lehetőséget.
* Ez tűnt a legjobb konstrukciónak, mind használat téren, mind számítási téren.

Az adatbázis táblákban szereplő adatok reprezentálásához a programban különböző entitásokat hoztunk létre. Az entitások egy DataManager komponensnek köszönhetően leképződnek a programban használt objektumokká.

## Adatdefiníciók

Annak érdekében, hogy a program minden komponensében elérhetők legyenek az egyes adatdefiníciók, ezt egy külön komponensként készítettük el.

## Adatelérési réteg

Célja az adathozzáférés biztosítása a felsőbb rétegek számára. Ennek megfelelően a réteg feladata:

* új entitások létrehozása,
* igény szerinti adathozzáférés biztosítása felsőbb rétegnek az adatokhoz.

## Üzleti logikai réteg

Célja az adatelérési rétegtől kapott adatok alapján kommunikálni a REST Service-szel. A mi esetünkben az üzleti logikai réteg kevés feladatot lát el, a programban található POCO objektumokon hajt végre néhány műveletet.

## REST Service

A REST [[3]](#_Hivatkozások) Service az üzleti logikai réteg és a kliens közötti hívásokat valósítja meg. Az MVVM mintát alkalmaztuk a tervezés során. A következőkben az általunk készített REST API-t mutatjuk be. A hívások során JSON objektumokkal illetve XML-ekkel dolgoztunk, a kéréseket http felett küldtük. Az objektumokat a programban használt objektumokból állítjuk elő, amire a különböző BindingModel-ek szolgálnak. A REST API megtervezése során alkalmaztuk a Facade mintát. Az üzenetek küldése során ügyeltünk arra, hogy ne csak a kliens oldalon történjen validáció, hanem a szerver oldalon is. A REST API segítségével küldött objektumokat mindig validáljuk, megvizsgáljuk, hogy minden kötelező adattagot meg van-e adva, érvényes adatok érkeztek-e. Emellett felkészültünk azokra az eshetőségekre is, ha null érkezik a kérésben.

A jelölés-szavazások kezeléséhez a következő API hívások megalkotására volt szükség:

* **GET api/Poll/GetPoll?pollId={pollId}:** Segítségével a jelölés-szavazások kérdezhetők le azonosító alapján.
* **GET api/Poll/ListNominationPolls:** A „NOMINATON” (jelölés) állapotban lévő jelölés-szavazások lekérdezése.
* **GET api/Poll/ListVotingPolls:** A „VOTING” (szavazás) állapotban lévő jelölés-szavazások lekérdezése.
* **GET api/Poll/ListClosedPolls:** A „CLOSED” (lezárult) állapotban lévő jelölés-szavazások lekérdezése.

A jelölés-szavazás adminisztrátori funkciókat külön kontrollerben valósítottuk meg.

* **POST api/PollAdmin/Save:** A jelölés-szavazás mentése. A funkció segítségével egy új jelölés-szavazást lehet létrehozni, vagy egy létezőt módosítani.

Az adminisztrátor képes a felhasználókat tiltani és újra engedélyezni, ami a következő REST hívásokkal valósítható meg:

* **POST api/Admin/BanUser?userId={userId}:** Az azonosítóval megadott felhasználó tiltása.
* **POST api/Admin/UnBanUser?userId={userId}:** A megadott azonosítóval rendelkező felhasználó újbóli engedélyezése.
* **GET api/Admin/SearchUser?term={term**}: A felhasználó név alapján történő keresése.

A jelölések kezeléséhez szükséges API hívások:

* **GET api/Nomination/GetForUser?userId={userId}:** A felhasználó által leadott jelölések lekérdezése.
* **POST api/Nomination/Save**: Jelölés elmentése.
* **POST api/Nomination/Delete?pollId={pollId}&nominationId={nominationId}:** Az azonosítóval és a jelöléshez tartozó jelölés-szavazással azonosított jelölés törlése.

A PollSubject-eken végezhető hívások:

* **GET api/Poll/SearchPollSubject?term={term}:** A film vagy sorozat név alapján kereshető.

A felhasználó be és ki tud jelentkezni a rendszerbe és a rendszerből, az ehhez szükséges API definíciók:

* **POST api/Auth/Login:** Bejelentkezés.
* **POST api/Auth/Logout:** Kijelentkezés.

A hírek manipulálásához, lekéréséhez szükséges API hívások:

* **GET api/News/ListNews:** Hírek listájának lekérdezése.
* **POST api/NewsAdmin/Save:** Hír mentése. Segítségével új hírt lehet létrehozni vagy egy régit módosítani.
* **POST api/NewsAdmin/Delete?newsId={newsId}:** A megadott azonosítóval rendelkező hír törlése.

### Modellek az adatkötéshez

Az adatok továbbításának megkönnyítéséhez létrehoztunk különböző BIndingModel-eket, amelyek segítségével a POCO objektumok könnyedén olyan formátumra alakíthatók, hogy JSON objektumként küldhetőek legyenek. A modellekhez készítettünk két függvényt, amelyek segítségével a BindingModel és a POCO objektum között tudunk konvertálni.

A modellek összeállítása során validáljuk az adattagokat, hogy a kötelezőek ki vannak-e töltve, illetve, hogy megfelelő típusú adattal lettek-e kitöltve.

**SaveNominationBindingModel:** jelölés elmentésének támogatásához hoztuk létre

* Id (Required, Text)
* Text (Required, MultilineText)
* PollId (Required, Text)
* UserId (Required, long)
* SubjectId (Required, long)

**SavePollBindingModel**: jelölés-szavazás mentésének támogatására

* Id (Required, Text)
* Title (Required, Text)
* Text (Required, MultilineText)
* PublicationtDate (Required, DateTime)
* NominationDeadline (Required, DateTime)
* VotingStartDate (Required, DateTime)
* VotingDeadline (Required, DateTime)
* AnnouncementDate (Required, DateTime)

**SaveNewsBindingModel:** hírek módosításának támogatására hoztuk létre

* Id (Required, Text)
* Title (Required, Text)
* Text (Required, MultilineText)

## Film és sorozat adatok betöltése

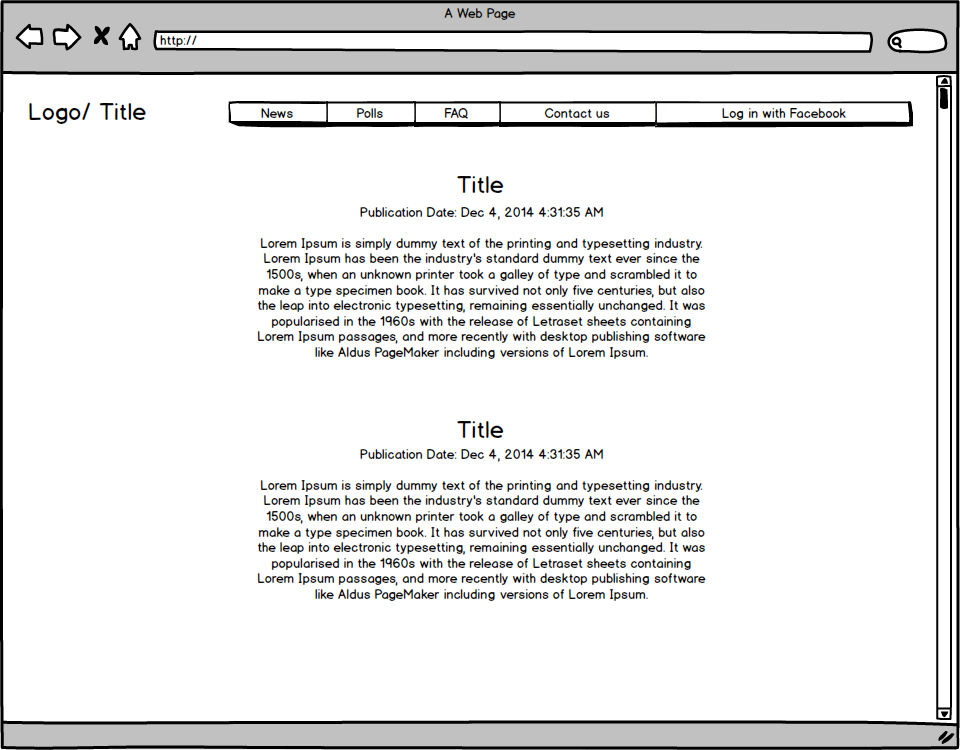
A program alapvető elvárása, hogy az adatbázis tartalmazzon filmeket és sorozatokat, amelyek közül a felhasználó a jelölés során tud válogatni. Az adatokat az IMDB által közzétett egyedi formátumú listából szereztük, amelyet egy Power Shell szkript segítségével csv formátumba parse-oljuk, majd ebből a fájlból betöltjük a Table Storage-ba.

## Grafikus felület (Kliens)

A szoftver tartalmaz egy felhasználói és egy adminisztrációs felületet. A felhasználói felület célja, hogy egy egyszerű, könnyen átlátható felületet nyújtsunk, amely segítségével az összes funkció elérhetővé válik a felhasználók számára. Az adminisztrációs felület célja, hogy egyszerű, könnyen kezelhető legyen és segítségével az adminisztrátori teendők gyorsan és könnyedén elvégezhetőek legyenek. A munkánk során nagy hangsúlyt fektettünk a grafikus felületek megtervezésére, a felületek elkészítését mockup-ok tervezése és készítése előzte meg. A tervezés a specifikáció elkészítése és a követelmények meghatározása után történt, az implementáció elkezdése előtt. A grafikus felületeket HTML5 alapokon készítettük el, ami által lehetővé tettük, hogy a szolgáltatás akár mobiltelefonról is elérhető legyen. Az elkészült felületek statikus oldalak, amelyeket HTML5, javascript és Angular JS segítségével kezdtünk el megvalósítani. A felhasználói felület terveket az elkészült mockup-okkal mutatjuk be a következőkben. (A felhasználói felület a házi feladat leadási határidejére nem készült el, azonban a felület a tervek alapján fog elkészülni.)

### Felhasználói grafikus felület

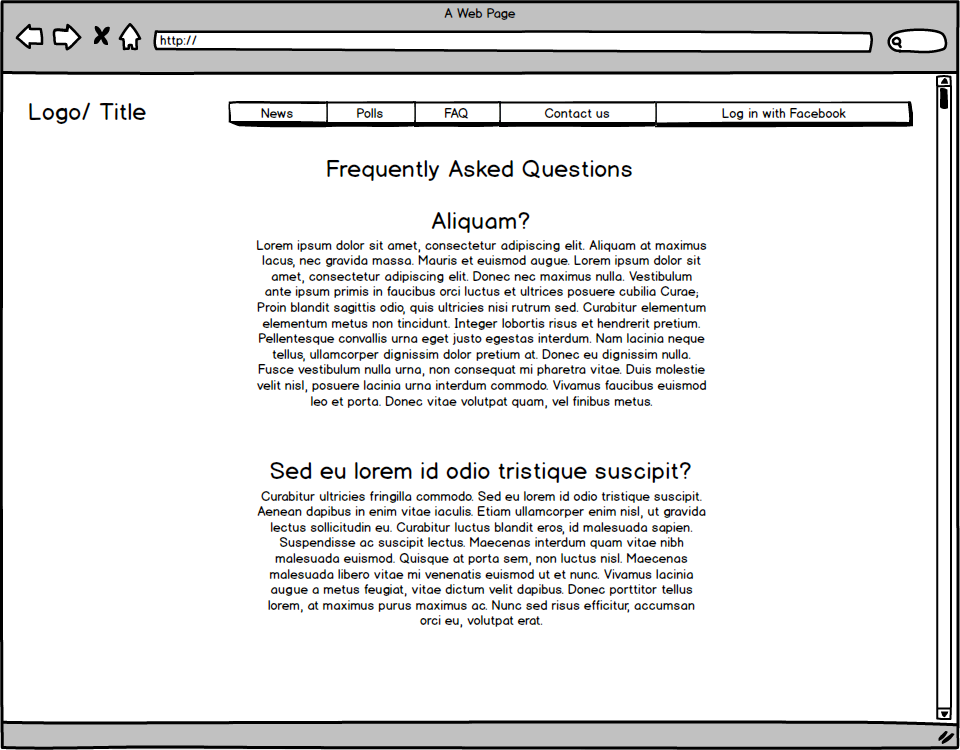
A program indulása után a felhasználót a böngészőben a kezdő képernyő várja, ami tartalmazza az eddig kiküldött híreket (2. ábra).



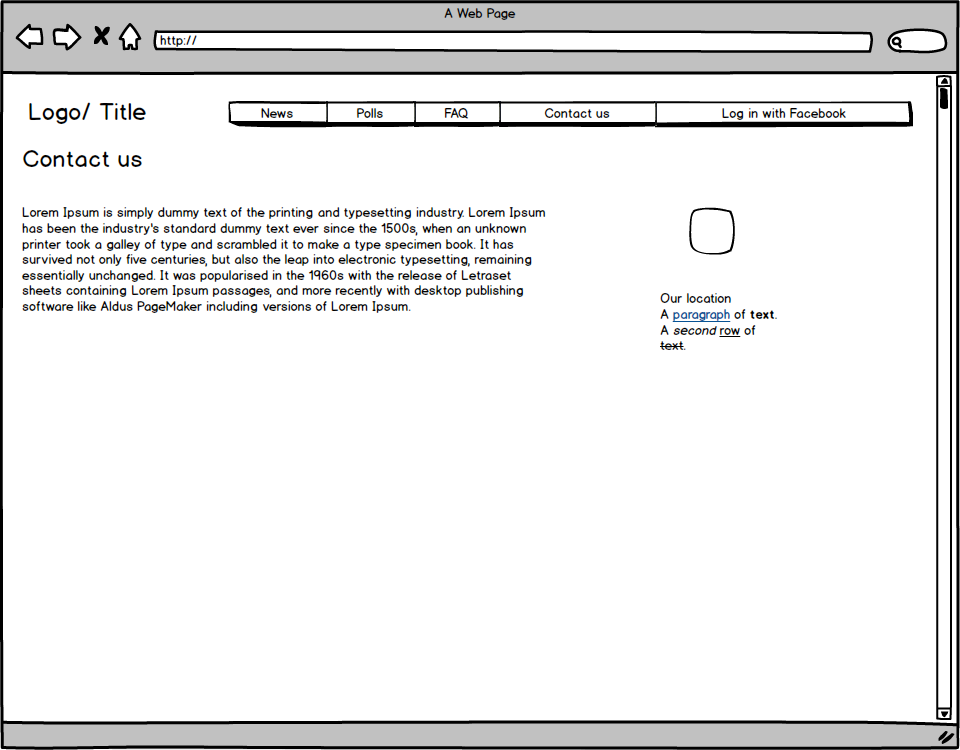
3. ábra Kezdőképernyő

A látogató számára a program biztosítja, hogy megtekinthesse a híreket, a lezárult szavazásokat, a statikus oldalainkat (gyakran ismételt kérdések, kapcsolat) és lehetőséget nyújt Facebook fiókkal történő belépésre.

A statikus oldalak egyszerűek, a legfontosabb információkat tartalmazzák szöveges formátumban. A gyakran ismételt kérdések a hírekhez hasonló formátumban jelennek meg. A kapcsolat oldal pedig tartalmazza az alkalmazással kapcsolatos legfontosabb információkat, a fejlesztők legfontosabb adatait, elérhetőségeit.

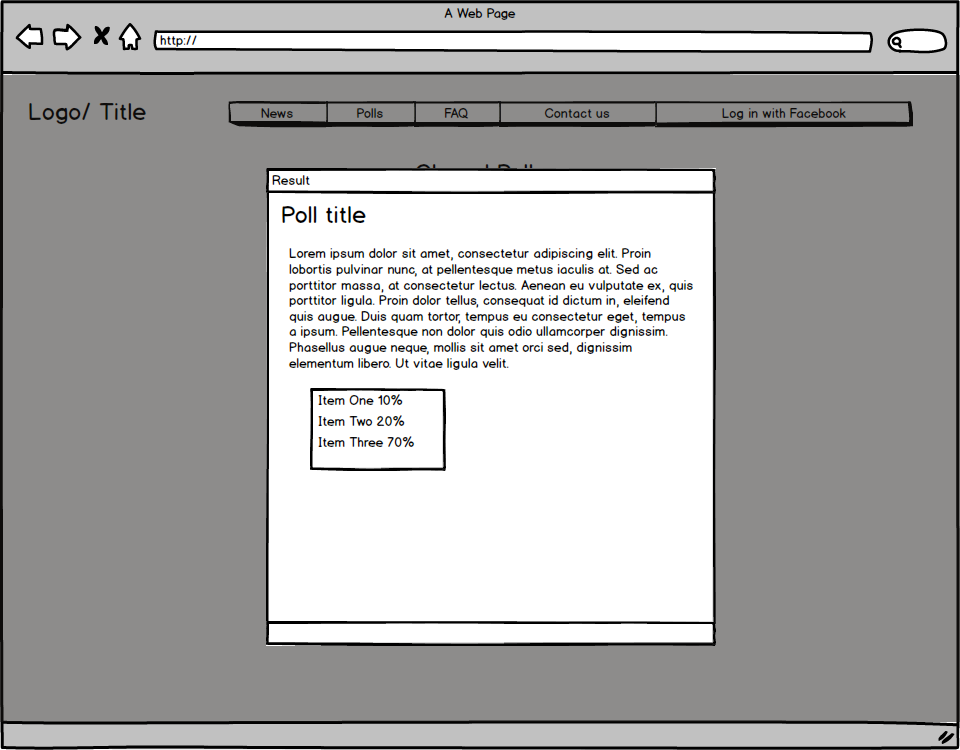


4. ábra Gyakran ismételt kérdések oldal

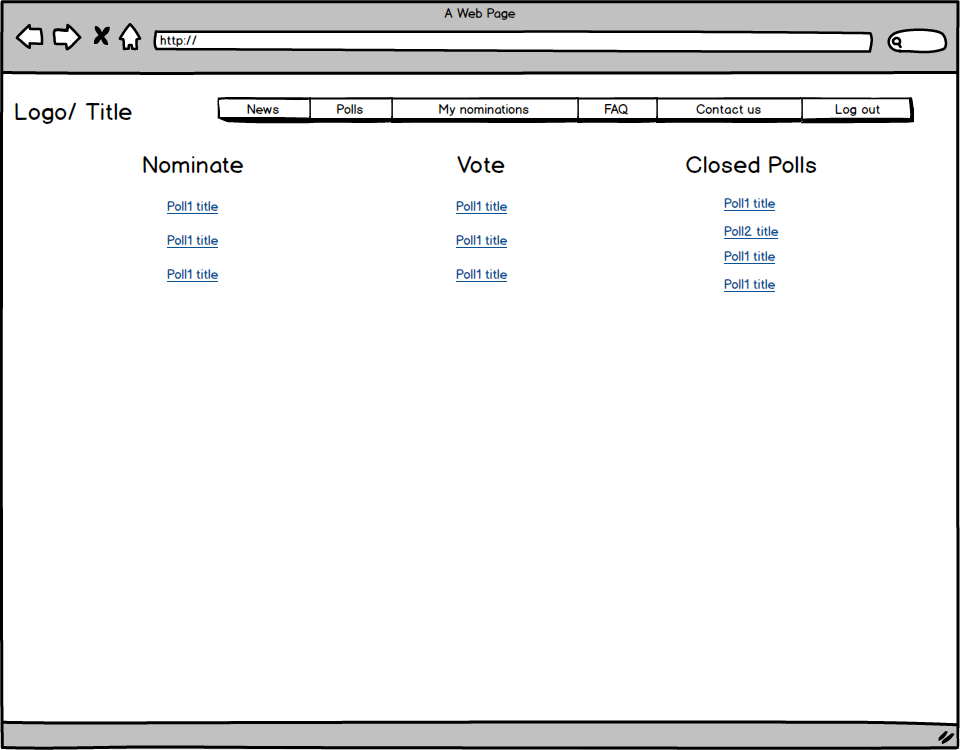


5. ábra Kapcsolat oldal

A látogatók meg tudják tekinteni a lezárt jelölés-szavazások eredményét. A jelölés-szavazások közül a szimpatikusat egy felsorolásból tudja kiválasztani, majd a megfelelőre rákattintva egy felugró ablakban tekintheti meg a szavazás végeredményét és a szavazatok százalékos megoszlását.



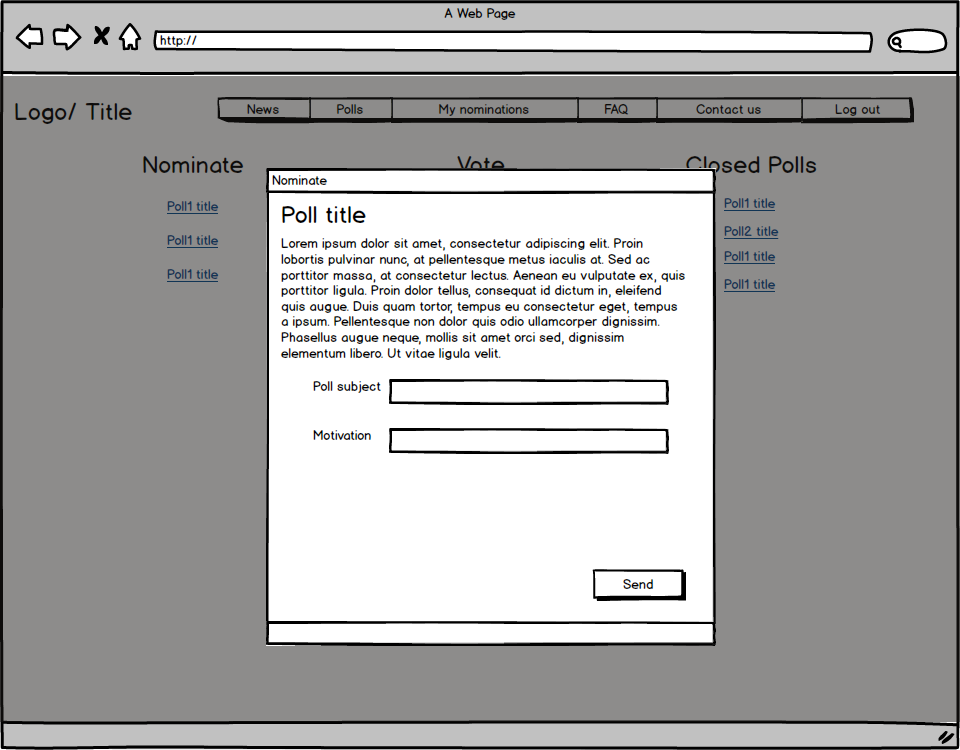
6. ábra Jelölés-szavazás eredménye



7. ábra Jelölés-szavazások listázása

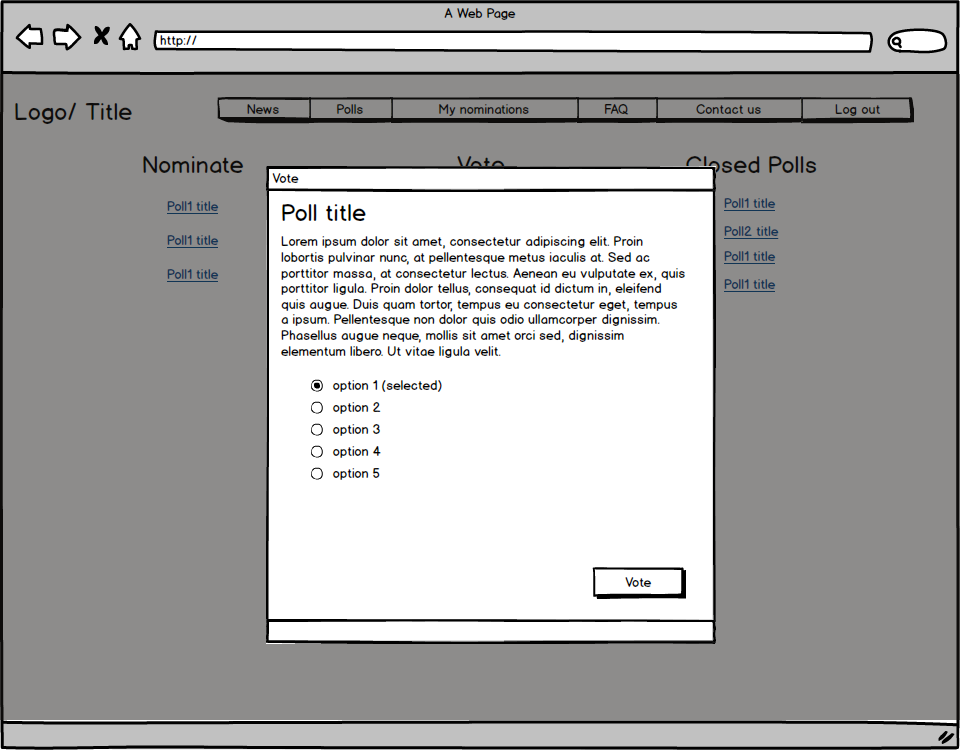
A Facebook fiókkal történő belépés után a felhasználónak lehetősége van böngészni az éppen folyamatban lévő jelölés-szavazásokat, filmeket, sorozatokat jelölni, illetve egy-egy jelölés-szavazáson tud szavazni, ha ezt korábban nem tette meg. A jelölés-szavazások listázására létrehoztunk egy külön oldalt, ahol a jelölés-szavazások a státuszuk szerint csoportosítva jelennek meg.

A felhasználó képes filmet jelölni a „Nominate” státuszú jelölés-szavazásokra. Ebben az esetben a szimpatikus jelölés-szavazásra kell kattintani, majd a felugró ablakban egy-egy szövegdobozban meg kell adni a jelöltet és egy rövid indoklást. A jelölt megadását segíti, hogy a szövegdoboz auto-complete funkcióval van felruházva. Miután kitöltöttük mindkét mezőt a Send gombbal el tudjuk küldeni a jelölésünket.



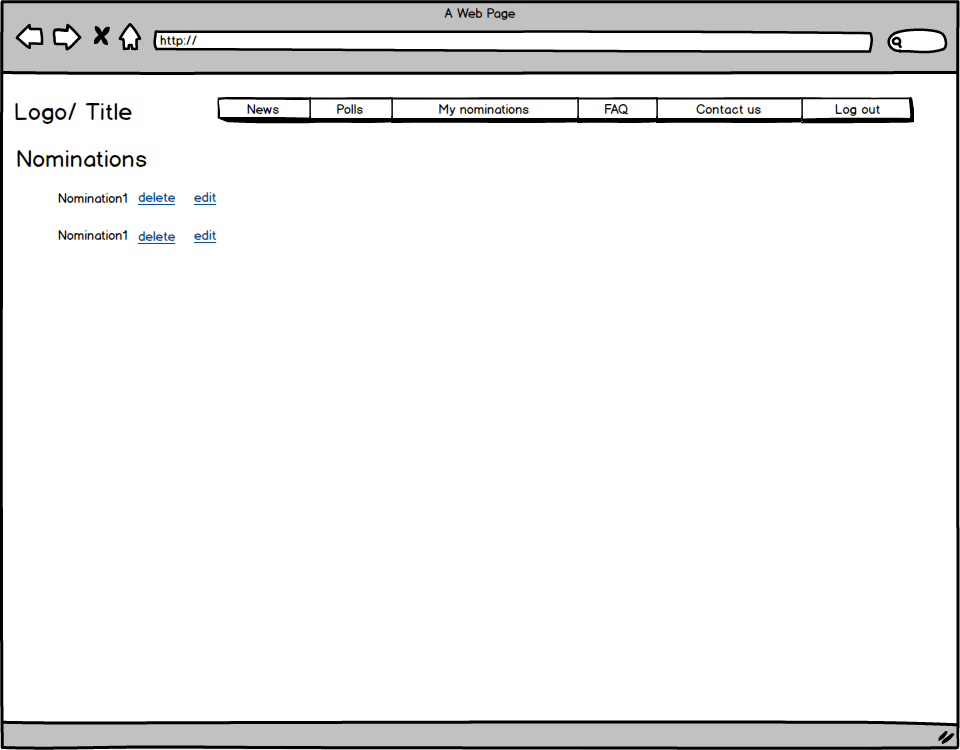
8. ábra Jelölés leadása

A felhasználónak lehetősége van szavazat leadására abban az esetben, ha korábban még az adott jelölés-szavazásra nem adott le voksot. Szavazni egy felugró ablakban tud, miután rákattintott a szimpatikus jelölés-szavazás nevére. A lehetőségek közül rádiós gombokkal tud választani, majd a Vote gombbal tudja véglegesíteni a szavazást.



9. ábra Szavazat leadása

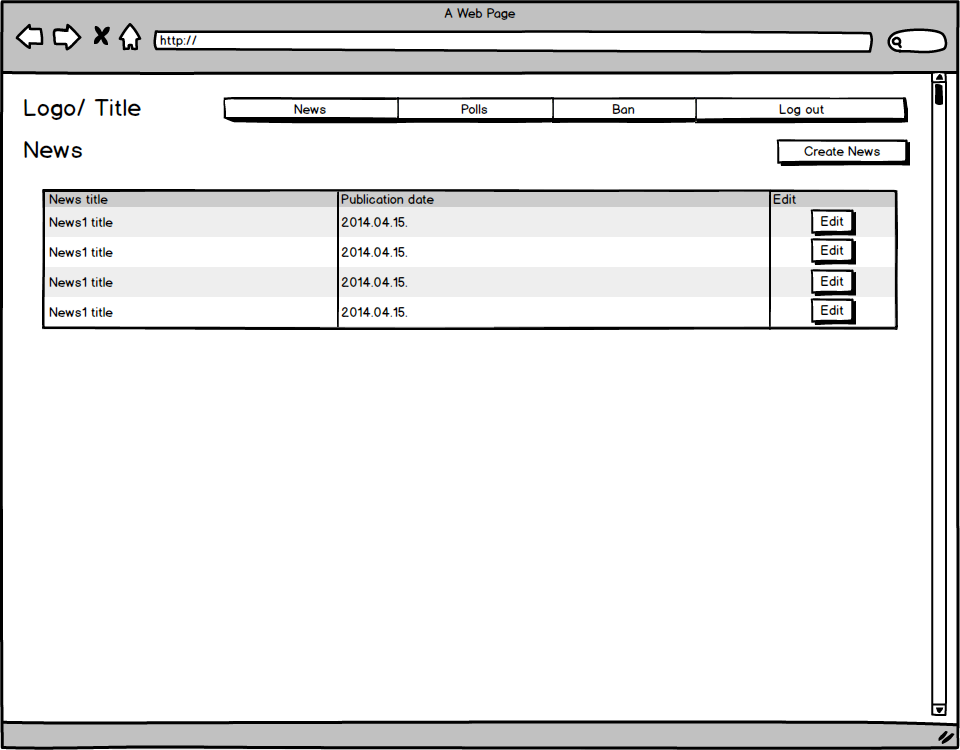
A felhasználónak lehetősége van a korábban leadott jelöléseit módosítani, törölni, amit a My nominations menüpont alatt a megfelelő jelölés kiválasztásával, szerkesztésével vagy törlésével tud elérni.



10. ábra Jelölések módosítása

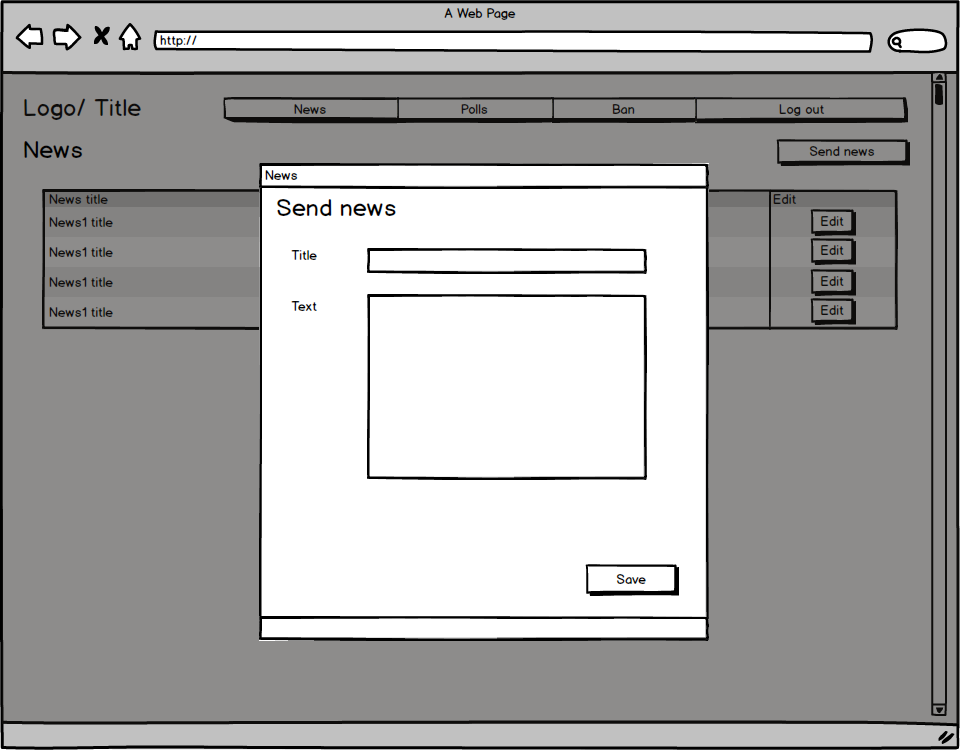
### Adminisztrátori grafikus felület

Az adminisztrátorok is Facebook fiók segítségével léphetnek be a rendszerbe, a belépő felhasználók szerepkörét az autentikációs folyamat során határozzuk meg. A belépés után az adminisztrátor külön oldalon tudja a jelölés-szavazásokhoz, a hírekhez és a felhasználókhoz kapcsolódó feladatokat elvégezni. A híreket egymás alatt felsorolva tudja megtekinteni, amiket az szerkesztés gomb segítségével tud szerkeszteni, illetve a „Create news” gomb segítségével tud új hírt létrehozni.



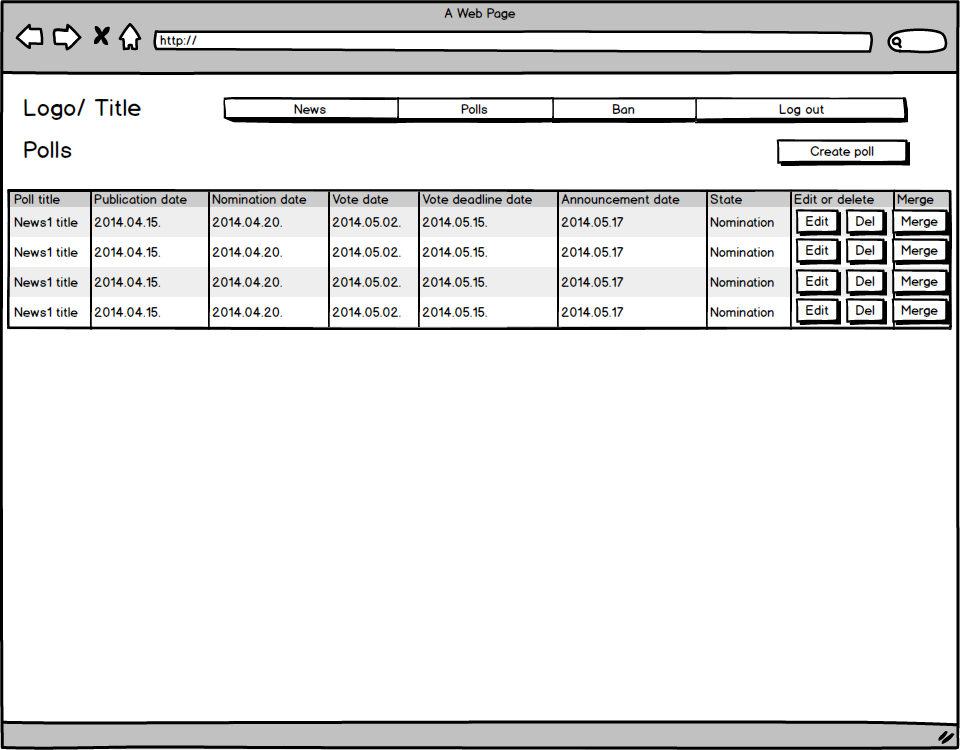
11. ábra Hírek karbantartása

Az új hír létrehozása és a hír szerkesztése teljesen hasonlóan egy-egy felugró ablakban történik. A hírnek meg lehet adni vagy módosítani lehet a címét és a szövegét.



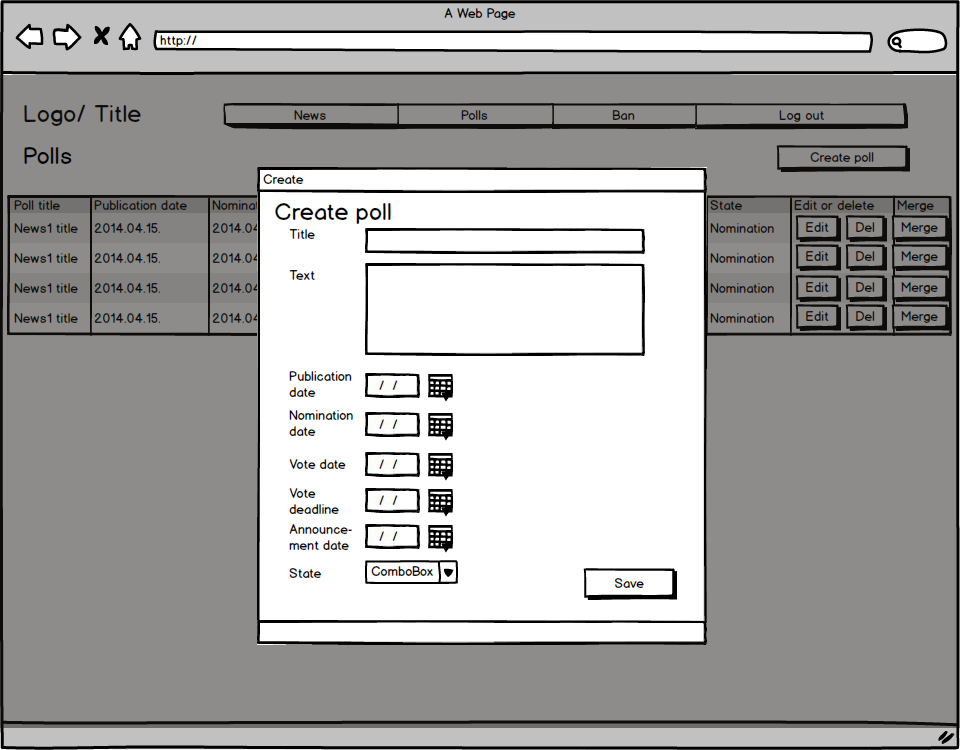
12. ábra Új hír kiküldése

A jelölés-szavazásokat egy táblázatban tekintheti meg az adminisztrátor. Az adminisztrátor képes új jelölés szavazást kiírni (Create Poll) és a meglévőket tudja szerkeszteni vagy törölni. Ezeken kívül lehetőség van még a leadott jelölések összefésülésére.



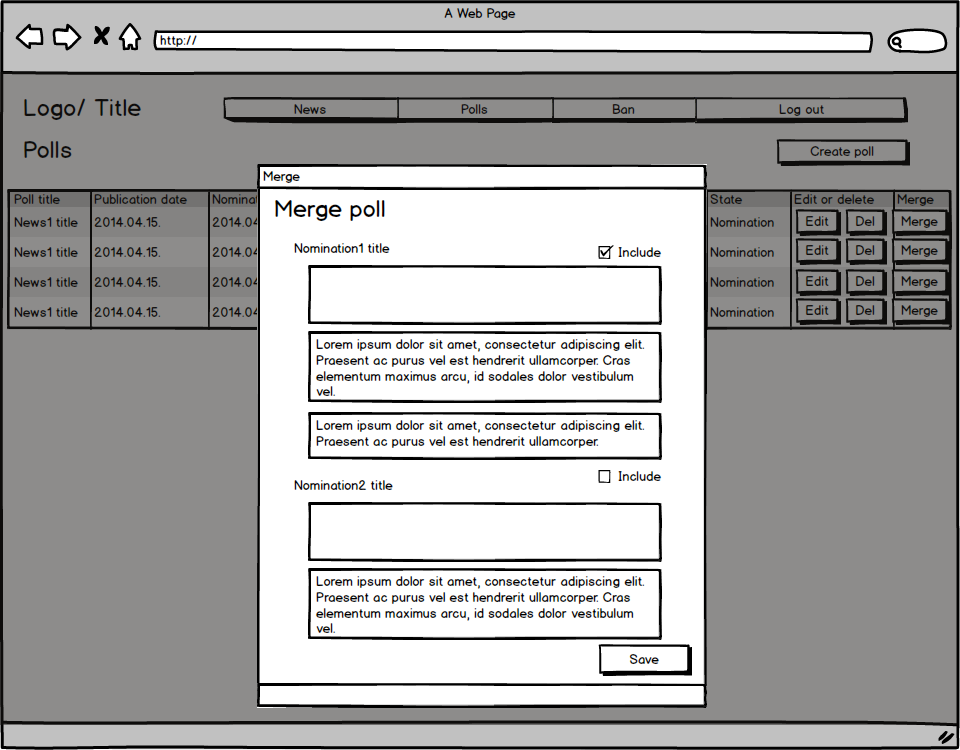
13. ábra Jelölés-szavazások admin oldal

Új jelölés-szavazást egy felugró ablak segítségével lehet létrehozni, ahol meg kell adni a címét és a hozzá tartozó leírást, illetve a különböző határidőket. A jelölés-szavazás szerkesztése egy hasonló felugró ablak segítségével történik.

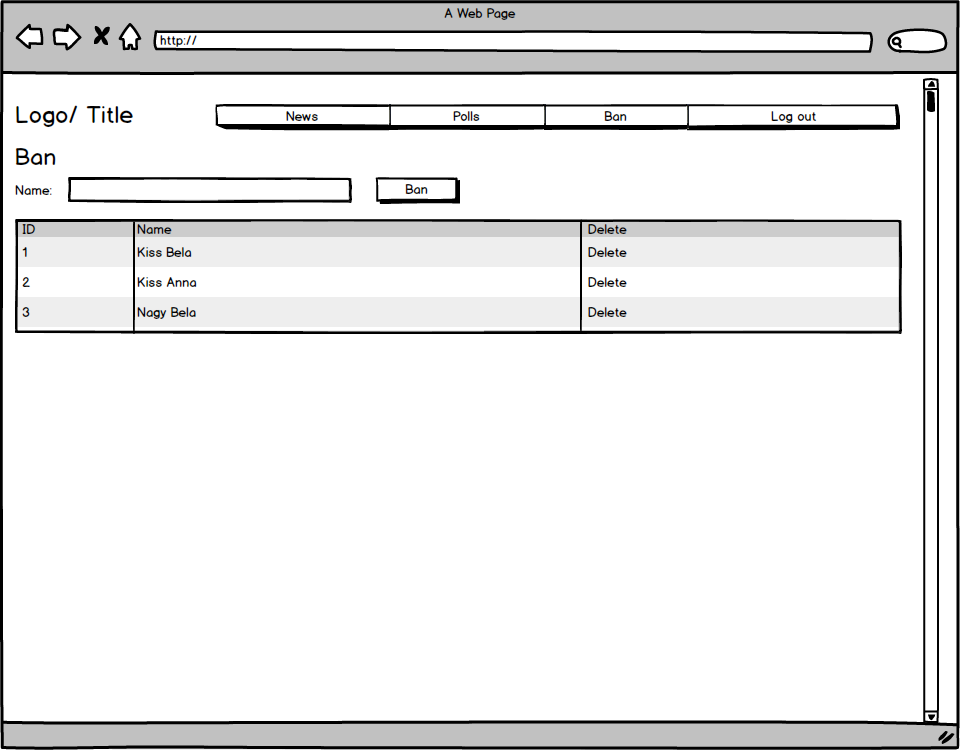


14. ábra Új jelölés-szavazás létrehozása

Az adminisztrátor feladata, hogy a jelölés-szavazásra leadott jelöléseket összefésülje, a nem releváns jelöléseket törölje, illetve ha ugyanazt a filmet többen is jelölték, akkor az indoklásokat egyesítse. Ezt a feladatot egy popup ablak segítségével tudja megvalósítani.



15. ábra Jelölések összefésülése



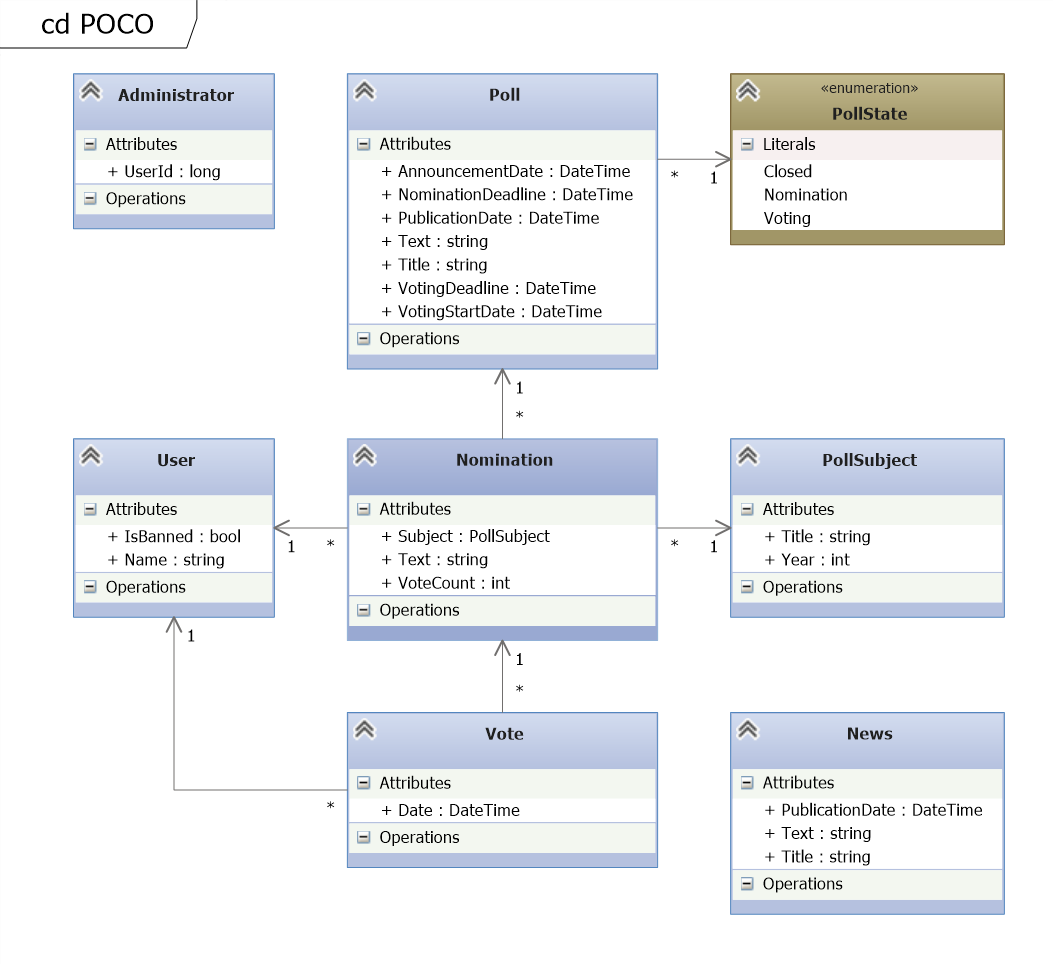
16. ábra Felhasználók letiltása

A szoftver lehetőséget biztosít, hogy a gyanús felhasználókat vagy azokat, akik valamilyen visszaélést követtek el kitiltsuk. A tiltást a felületen úgy tudja az adminisztrátor megtenni, hogy egy auto complete támogatással rendelkező szövegdobozba begépeli a felhasználó nevét, majd a Ban gombra kattint. A felületen listaként jelennek meg a letiltott felhasználók, akik tiltását a törlésre való kattintással tudjuk feloldani.

# Adat- és adatbázisterv

## A program objektum modellje

A következőkben az adatmodellt mutatjuk be részletesen, aminek az osztálydiagramja a 17. ábrán látható. Az objektum modell tartalmazza az Administrator, User, Poll, Nomination, Vote, PollSubject, News osztályokat és a PollState enumot.



17. ábra POCO objektumok

### Az alapobjektumok

Az alapobjektumok az adatmodellt reprezentálják, ezeket igyekeztünk POCO-kal megvalósítani (A Plain Old CLR Object). Az attribútumok definiálására jelen esetben Property-ket használtunk. A következőkben az 17. ábrán látható osztályokat és azok tulajdonságait mutatjuk be részletesen.

**Administrator:** Az adminisztrátor felhasználót reprezentálja, az adminisztrátor egy szerepkör, amihez a megfelelő felhasználót az azonosítója segítségével kötjük hozzá.

**User:** Felhasználó, aki rendelkezik névvel, illetve be van nála jegyezve, hogy ki van-e tiltva a rendszerből.

**Poll:** Szavazás-jelölés, ami rendelkezik címmel és szöveggel, van egy PollState típusú állapota, illetve tartoznak hozzá különböző határidők.

**Nomination:** Jelölés, ami tartalmazza a jelölt filmet vagy sorozatot (PollSubject), illetve egy a jelöléshez tartozó indoklást. Ezen kívül tároljuk, hogy melyik jelölés-szavazáshoz tartozik, illetve, hogy melyik felhasználó adta le a jelölést.

**Vote:** Szavazat, amit egy adott felhasználó ad le egy megadott jelölésre. Nyilvántartása azért fontos, mivel a szavazatok megvizsgálásával dönthető el, hogy egy felhasználó szavazott-e már az adott jelölés-szavazáson. A jelölés rendelkezik még egy dátummal, ami a leadás dátumát reprezentálja.

**PollSubject:** A jelölés-szavazásokra PollSubject-ek jelölhetők, amik a különböző filmeket és sorozatokat reprezentálják. Tartalmazza a film címét és a megjelenésének az évét.

**News:** A honlapra kikerülő hírek, amely tartalmazza a hír címét és a szövegét. A hírnek nyílván tartjuk a keletkezésének idejét.

**PollState:** A jelölés-szavazás állapotait tartalmazza.

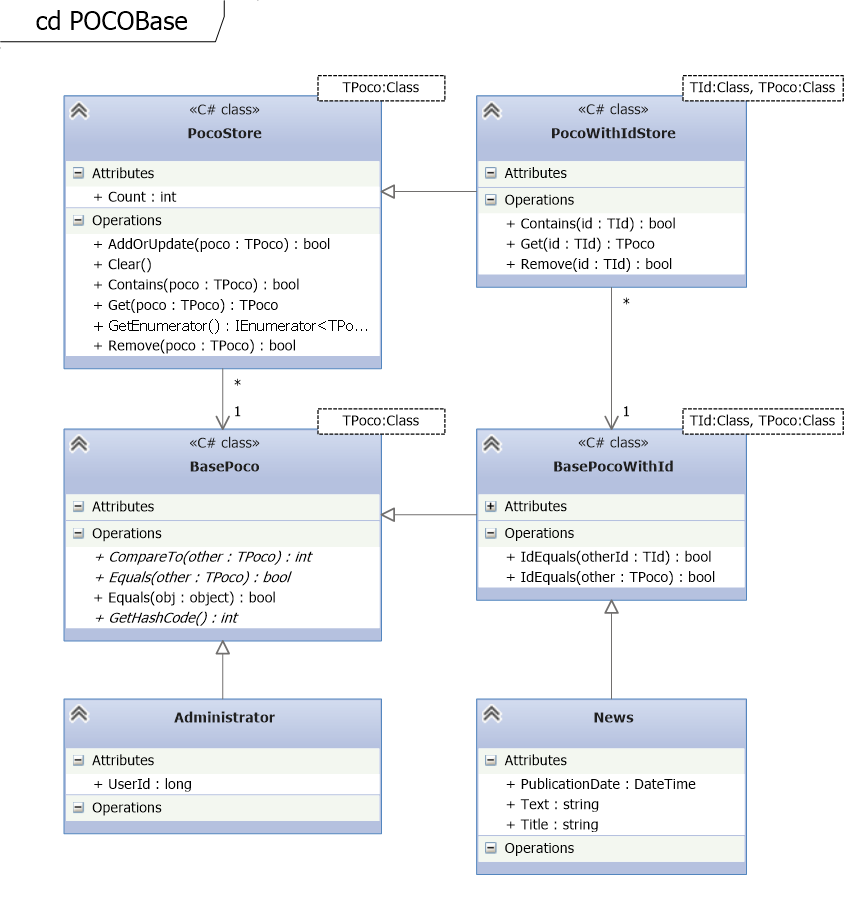
Minden objektumra definiáltuk az Equals/GetHashcode() metódusokat illetve implementáltuk a Comparable interfészt.

## Objektumok közötti kapcsolatok

Az objektum modellben a jelölés-szavazásokhoz tartozik egy állapot, ami azt írja le, hogy a jelölés-szavazás éppen melyik fázisban tart. A jelölés-szavazásokhoz tetszőleges számú jelölés tartozhat, azonban egy jelöléshez csupán egy jelölés-szavazás tartozik. A jelölésekben rögzítjük, hogy melyik felhasználó adta le, erre azért van szükség, hogy később a felhasználó tudja módosítani a saját jelöléseit. Egy felhasználó egy jelölés-szavazásra tetszőleges számú jelölést leadhat. A jelöléshez tartozik egy „PollSubject”, ami megát a jelölendő filmet vagy sorozatot reprezentálja. Egy jelöléshez pontosan egy ilyen objektumnak kell tartoznia, üres jelölést nem lehet leadni, azonban egy filmet több ember is jelölhet a különböző jelölés-szavazásokra. A jelölésekhez tartozhatnak szavazatok. Egy jelöléshez tetszőleges számú szavazat tartozhat a jelölés-szavazás szavazási fázisában. Egy szavazat pontosan egy jelöléshez tartozhat. A szavazatban nyilván tartjuk azt is, hogy ki adta le a voksot. Erre azért van szükség, mivel minden felhasználó egy jelölés-szavazásra pontosan egy szavazatot adhat le, azonban tetszőleges számú jelölés-szavazáson adhat le szavazatot.

## POCO objektumok tárolása

A 18. ábrán látható, hogy a POCO-k tárolására milyen struktúrát hoztunk létre, illetve az öröklési hierarchia is ezen az ábrán látható. A PocoStore valósítja meg a tároláshoz szükséges műveletet, megfelelő függvényeket definiál a POCO-k törlésére, módosítására, tartalmazás vizsgálatra. Tőle örököl a PocoWithIdStore osztály, ami azonosító alapján kezeli az objektumokat, a lekérés, törlés és tartalmazás vizsgálata ID alapján történik. A PocoStore tartalmazza a BasePoco objektumokat, ami megvalósítja a Equals/GetHashcode() metódusokat illetve a Comparable interfészt is megvalósítja. A PocoWithIdStore tartalmazza a BasePocoWithId objektumokat, ami a BasePoco osztálytól örököl. Ennél az objektumnál pluszként meg kellett valósítani az azonosító szerinti összehasonlítást végző metódust. Az abrán látható, hogy az Administrator egy BasePoco, a News pedig egy BasePocoWithId típusú objektumok, hiszen örökölnek az említett osztályoktól.



18. ábra Poco objetumok kapcsolata

## Az adatbázis

Az adatbázis tárolására a Microsoft Azure Table Storage megoldását választottuk. A módszer előnye, hogy jól skálázható, ezáltal alkalmassá tettük a rendszert arra, hogy akár valós környezetben is jól használható legyen. A Table Storage-ban egy-egy entitást egy PartitionKey és egy RowKey határoz meg, amelyek egyedi azonosítóként funkcionálnak. A PartitionKey szerepe, az, hogy az azonos partíciókban található adatokat ugyanazon a csomópontok kell tárolni, azonban a különböző kulccsal rendelkezők szétoszthatók a különböző csomópontok között. Ezen kívül még nyilvántart egy időbélyeget, ami az utolsó módosítás dátumát jelenti és a rendszer tart karban. Az adatbázisban az entitások a soroknak felelnek meg. Az entitások tulajdonságai típusos kulcs-érték párok, amik nevei megegyezik a táblában az oszlopok neveivel.

A következőkben a megtervezett adatbázis táblákat mutatjuk be, amelyek megtervezésének alapjául a program objektumai szolgáltak.

### Administrator

Az adminisztrátorok PartitionKey-e az adminisztrátor user ID-je.

|  |  |  |
| --- | --- | --- |
| Név | Típus | Megjegyzés |
| PartitionKey | string | User ID |

### User

A User tartalmazza a Name és az IsBanned tulajdonságokat, amiket már korábban definiáltuk. A tábla PartitionKey-e a felhasználókat azonosító ID.

|  |  |  |
| --- | --- | --- |
| Név | Típus | Megjegyzés |
| PartitionKey | string | ID |
| Name | string |  |
| IsBanned | bool |  |

### Poll

A Poll tartalmazza a korábban meghatározott határidőket, a jelölés-szavazás címét és leírását. A tábla PartitionKey-e a jelölés-szavazás állapota, a RowKey pedig a jelölés-szavazás azonosítója. A PartitionKey-nek azért a jelölés-szavazás állapotát választottuk, mivel a jelölés-szavazásokat a programunk jelölés-szavazás állapot alapján kérdezi le.

|  |  |  |
| --- | --- | --- |
| Név | Típus | Megjegyzés |
| PartitionKey | string | State |
| RowKey | string | ID |
| Text | string |  |
| PublicationDate | DateTime |  |
| NominationDeadLine | DateTime |  |
| VotingStartDate | DateTime |  |
| VotingDeadline | DateTime |  |
| AnnouncementDate | DateTime |  |

### Nomination

A Nomination tartalmazza a korábban meghatározott leírást, a szavazatok számát, illetve a jelölést leadó felhasználó ID-ját, a jelölést tartalmazó jelölés-szavazás azonosítóját és a hozzá tartozó jelölt azonosítóját. A táblában a PartitionKey a jelölés-szavazás azonosítója, ami azért praktikus választás, mert az adatbázisból általában csak az egy adott jelölés-szavazáshoz kapcsolódó jelöléseket kérdezzük le. A RowKey a jelölés ID-ja.

|  |  |  |
| --- | --- | --- |
| Név | Típus | Megjegyzés |
| PartitionKey | string | Poll ID |
| RowKey | string | ID |
| UserId | long |  |
| SubjectId | string |  |
| Text | string |  |
| VoteCount | int |  |

### Vote

A Vote tartalmazza a szavazatot leadó felhasználó azonosítóját, illetve annak a jelölésnek az azonosítóját, amelyre a szavazatot leadták. A szavazat tartalmazza még a leadásának dátumát. A táblában a PartitionKey a jelölés azonosítója, hiszen általában egy adott jelöléshez tartozó szavazatokat szeretnénk lekérdezni. A RowKey a szavazatot leadó felhasználó azonosítója.

|  |  |  |
| --- | --- | --- |
| Név | Típus | Megjegyzés |
| PartitionKey | string | Nomination ID |
| RowKey | string | User ID |
| Date | DateTime |  |

### PollSubject

A PollSubject tartalmazza az addot film vagy sorozat nevét és megjelenésének évét. A táblában a PartitionKey a PollSubject azonosítója.

|  |  |  |
| --- | --- | --- |
| Név | Típus | Megjegyzés |
| PartitionKey | string | ID |
| Title | string |  |
| Year | int |  |

### News

A News tartalmazza a hír címét, szövegét és a hír publikálásának dátumát. A tábla PartitionKey-e a hír azonosítója.

|  |  |  |
| --- | --- | --- |
| Név | Típus | Megjegyzés |
| PartitionKey | string | ID |
| Title | string |  |
| Text | string |  |
| PublicationDate | DateTime |  |

Az objektummodell és az adatbázis közötti konverziót egy általunk készített DataManager osztály végzi.

## GUI terv

A program tervezésekor az egyszerű felépítést tartottuk szem előtt. Ennek megfelelően a program érdemi működését egy egyszerű HTML alapú oldal látja el. A felhasználó egy egyszerű menüsor segítségével tud navigálni a funkciók között, a különböző szerkesztéshez, létrehozáshoz, módosításhoz tartozó funkciók használatához egy-egy popup ablakban található formot kell kitölteni. A grafikus felület részletes leírása a „Grafikus felületek” fejezetben található.

# Hitelesítés és biztonság

A felhasználókat Facebook belépés segítségével hitelesítjük. A gyanúsnak tűnő vagy csaláson, a rendszer kijátszásán kapott felhasználókat az adminisztrátor ki tudja tiltani. A rendszer elkészítése során arra is ügyeltünk, hogy a kommunikáció is titkosított legyen. A teljes kommunikációt SSL-lel titkosítottuk.

# Tesztelés

A fejlesztés során a rendszert egységtesztekkel és integrációs tesztekkel ellenőriztük. A POCO objektumok helyes működését egységtesztekkel vizsgáltuk meg. A tesztelés során vizsgáltuk, hogy helyesen jönnek-e létre az objektumok, illetve hogy azok tárolása, módosítása, törlése helyesen zajlik-e.

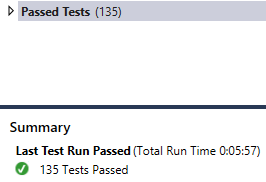
Emellett a REST API implementálása során a hívásokat megvalósító kontrollerek helyességét is teszteltük integrációs tesztekkel. A tesztelés során nem csak a helyes lefutást ellenőriztük, hanem hibás bemenetekkel is teszteltük a függvényeket. A tesztek elkészítését azért tartottuk fontosnak, mivel egy-egy apróbb módosítás után a tesztek lefuttatása után azonnal láttuk, hogy ha elrontottuk az adott függvényt, illetve a definiált teszteknek köszönhetően a megfelelő implementációk elkészítése után azonnal tudtuk ellenőrizni a helyességet. A teszteket először manuálisan létre hozott adatokon teszteltük amelyek a memóriában tároldónak, majd a teszteket implementáltuk arra az esetre is, amikor azok a DataStorage-ból kerülnek elérésre.

Mivel 40-60 másodperc egy Table Storage tábla törlése, és minden Table Storage-ot érintő teszteset esetén friss táblák kellenek, ezért minden tesztesethez egyedi névvel jönnek létre. A tesztesetek futtatását nem párhuzamosítottuk, mivel körülbelül 5 perc alatt sikeresen befejeződtek, valamint az MSTest és más népszerű keretrendszerek nem támogatják az integrációs tesztek párhuzamosítását.

A REST hívások helyességét manuálisan is teszteltük az Advanced REST Client [[4]](#_Hivatkozások) segítségével, ami egy Google Chrome-ba telepíthető szoftver. A felületen lehetőség van az API hívás megadására, illetve a JSON objektum definiálására is. Az elkészített API hívások elmenthetőek, így azokat a munkánk során akár többször is fel tudjuk használni.

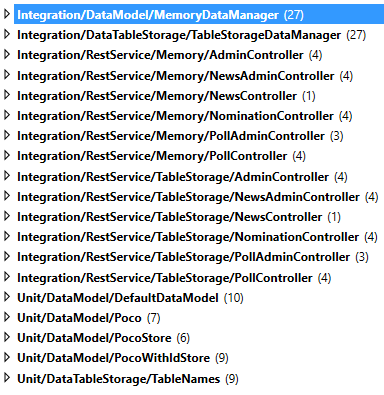
A korábban leírtakon túl külön teszteltük az elkészült lekérdezések helyes működését. Ezek vizsgálata is integrációs tesztek segítségével történt. A teszteket hibás bemeneti adatokra is elkészítettük.

A munkánk végére összesen 135 tesztesetet definiáltunk, amelyek segítségével a szerver oldali kódot ellenőrizzük. A grafikus felülethez tesztek definiálását nem tartottuk indokoltnak.



19. ábra A tesztek sikeres lefutása

A 20. ábrán láthatóak a definiált tesztesetek kategóriái, illetve hogy az egyes kategóriák mennyi tesztet tartalmaznak. A tesztek között vannak egységtesztek, amelyek a POCO objektumokon végezhető műveleteket tesztelik, illetve vannak integrációs tesztek, például a REST szolgáltatás kontrollereihez implementáltak. Azokban az esetekben ahol a programlogika egyszerű nem tartottuk indokoltan külön egységtesztek készítését.



20. ábraTeszteset kategóriák

# A program készítése során felhasznált eszközök

* Microsoft Visual Studio 2013 [[5]](#_Hivatkozások)
  + Felhasználás: fejlesztőkörnyezet
* Microsoft Azure Table Storage [[1]](#_Hivatkozások)
  + Felhasználás: adatbázis-kezelő rendszer
* Microsoft Azure Web Role [[7]](#_Hivatkozások)
  + Felhasználás: web API
* Advanced REST Client [[4]](#_Hivatkozások)
  + REST hívások tesztelése
* Microsoft PowerShell [[6]](#_Hivatkozások_1)
  + Imdb-ről filmek lekérése
* NuGet
  + Függőségek kezelése
* MSTest
  + Egységtesztek és integrációs tesztek készítése
* GitHub [[8]](#_Hivatkozások)
  + Felhasználás: verziókezelés, csapatmunka támogatása
* Microsoft Visio
  + Felhasználás: ábrák előállítása
* Microsoft Word
  + Felhasználás: dokumentáció elkészítése

# Összefoglalás

Munkánk során megterveztük, implementáltuk illetve dokumentáltuk a Nominate and Vote nevű szavazó-jelölő rendszert. Az elkészített alkalmazás segítségével szavazások írhatók ki, amikre a felhasználók tudnak filmeket és sorozatokat jelölni. A jelölési szakasz lezárulása után a felhasználók szavazni tudnak az adott jelölés-szavazás jelöltjeire, majd a szavazási időszak lezárása után megnézhetik az eredményeket. Emellett az oldalon különböző híreket olvashatnak, illetve megtekinthetik a készítőkkel kapcsolatos információkat.

A megvalósítás a következő rétegeket használja: adatbázis réteg, üzleti logikai réteg, REST Service és felhasználói felület. Az alkalmazás az adatokat a Microsoft Azure Table Storage-ában tárolja. A grafikus megjelenítést HTML 5 alapú webes felülettel oldottuk meg, amelyhez Angular JS-t és javascriptet használtunk fel. Az alkalmazás biztosítja, hogy a jelölés-szavazási procedúra végigvihető legyen a jelölés-szavazás kiírásától az eredmény kihirdetéséig. A szoftver biztosítja, hogy az adminisztrátori teendők egyszerűen és könnyedén elláthatóak legyenek. A program Facebook fiókos hitelesítést tesz lehetővé, a teljes komunikációt SSL-lel titkosítottuk. A rendszerben a függőségek kezelésére a NuGet-et használtuk. A rendszer belső működése különböző egységtesztekkel és integrációs tesztekkel ellenőrzött, amelyhez MSTest-et használtuk fel.

A feladat során megoldottuk, hogy a szerver oldal egység és integrációs tesztekkel ellenőrzött legyen, amit 135 tesztesettel valósítottunk meg. A feladat nagy kihívása volt a Microsoft Azure használata. Megterveztünk és megvalósítottunk egy Azure Table Storage-ban tárolható adatbázist, amely tervezése során figyelembe vettük, hogy az adatokat minél egyszerűbben, minél kevesebb lekérdezéssel és minél gyorsabban véghez tudjuk vinni. A programban megoldottuk, hogy a belső POCO objektumok és az adatbázisból érkező entitások automatikusan leképződjenek. Emellett törekedtünk arra, hogy tervezési minták használatával valósítsuk meg, a programban használtuk az MVVM architektúra mintát és a Facade tervezési mintákat. Mindemellett a felhasználói felület tervezésére is nagy hangsúlyt fektettünk, a specifikáció és a követelmények megfogalmazása után mockup-okkal megterveztük az oldalakat, majd ezek alapján készítettük el az implementációt. A tervezés során törekedtünk arra, hogy a felület praktikus és könnyen használható legyen.

Munkánk során részletes terveket készítettünk – részük jelen dokumentum tartalmát képezik – és jelentős mennyiségű implementációs munkát is végeztünk. Ennek eredményeképpen egy jól működő és megbízható alkalmazást készítettünk el, amely az elvárt alapvető igényeknek megfelel, feladatát képes ellátni.

# Továbbfejlesztési lehetőségek

Az alkalmazás továbbfejlesztésére számos lehetőséget látunk. Néhány fontosabb továbbfejlesztési lehetőség:

* A program funkcionalitása kiterjeszthető azzal, hogy lehetőséget biztosítunk arra, hogy a felhasználók filmeket és sorozatokat tudjanak értékelni 1-5-ig terjedő skálán.
* A másik továbbfejlesztési lehetőség, hogy készítünk, egy olyan felöletet a hozzá tartozó belső működést megvalósító komponensekkel, amik lehetővé teszik, hogy a sorozatrészek egyesével is értékelhetők legyen.
* Lehetőség biztosítása arra, hogy a felhasználók a jelölés-szavazások alá kommentet írhassanak.

A fejlesztés folyamán is odafigyeltünk ezekre a továbbfejlesztési irányokra és igyekeztünk olyan tervezői döntéseket hozni, amelyek segítik a program fejlesztésének folytatását.

# Telepítési leírás

A Nominate and Vote program készítésekor hangsúlyt fektettünk az egyszerű kezelhetőségre és üzembe helyezhetőségre. A program telepítése abból áll, hogy a Web szolgáltatást, REST szolgáltatást fel kell tenni egy ASP.NET-et tartalmazó szerverre, például egy IIS szerverre, a klienshez tartozó kódot pedig egy tetszőleges web szerverre. A tesztek futtatásához szükséges, hogy telepítsünk egy Microsoft Visual Studio 2013-at, amelybe telepíteni kell az MSTest-et.

# Hivatkozások

[1] Microsoft Azure

<http://azure.microsoft.com/hu-hu/>

[2] Microsoft Azure Table Storage

<http://azure.microsoft.com/en-us/documentation/articles/storage-dotnet-how-to-use->[tables/](file:///D:\Egytem\szar\hf\docs\df.docx)

[3] REST Reference

http://msdn.microsoft.com/en-us/library/dn631844.aspx

[4] Advanced REST Client

<https://chrome.google.com/webstore/detail/advanced-rest->

[client/hgmloofddffdnphfgcellkdfbfbjeloo](file:///D:\Egytem\szar\hf\docs\dg.docx)

[5] Visual Studio 2013

<http://www.visualstudio.com/>

[6] Microsoft PowerShell

<http://msdn.microsoft.com/en-us/library/ms714469%28v=vs.85%29.aspx>

[7] Microsoft Azure Web Role

<http://msdn.microsoft.com/en-us/library/azure/gg557553.aspx>

[8] GitHub

<https://github.com/>