TANSZÉKVEZETŐ

SZAKDOLGOZAT FELADAT

Kocka Dominik Csaba
Mérnökinformatikus hallgató részére

iOS alkalmazás fejlesztése swift alapú backenddel

Az Apple cég iPhone készülékeire fejlesztett iOS rendszerek mára már az egyik legelterjedtebb mobil operációs rendszereknek számítanak, ezért a fejlesztés erre a rendszerre komplex és érdekes feladat. A nyelv, amiben az alkalmazások készülnek, a Swift egy magas szintű programozásinyelv, amiben már az Apple készülkein kívül, harmadik féltől származó eszközökre is fejleszthetünk, például backendet alkalmazásainknak, a Vapor keretrendszer segítségével.

Az alkalmazás, amit meg szeretnék valósítani, egy közösségi munkavállalást elősegítő alkalmazás, mely segíthet az egyetemi közösségi élet segítőit összekötni a munkaadókkal, rendezőkkel. Ehhez egy backend is tartozni fog, amelyre később, akár más rendszereken futó alkalmazásokat is rá lehet majd kapcsolni.

A backend a Vapor környezet segítségével fog elkészülni Swift nyelven. Tartalmazni fog felhasználó kezelést, adatbázist és jogkörök meghatározását is. A frontend egy iOS alkalmazás lesz, melynek a felületének elkészítéséhez a UIKit környezetet fogom használni, VIPER architektúrával. A kettő réteg REST hívások segítségével fog kommunikálni egymással.

A hallgató feladatának a következőkre kell kiterjednie:

- Ismerje meg a Vapor környezetet és használatát
- Tervezze meg az alkalmazás belső logikáját
- Tervezze meg a backend logikáját, adatbázisát, jogköreit
- Tervezze meg az alkalmazás felhasználói felületét
- Készítse el a backendet a terveknek megfelelően
- Készítse el az alkalmazást a terveknek megfelelően
- Tesztelje le és javítsa a hibákat a rendszerben

Tanszéki konzulens: Dr. Ekler Péter, egyetemi docens

Budapest, 2021. október 6.

Dr. Charaf Hassan
egyetemi tanár
tanszékvezető

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék
H-1117 Budapest, Magyar tudósok krt 2. Q.B.207
Telefon: 463-2870. Fax: 463-2871
www.aut.bme.hu

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Budapest Műszaki és Gazdaságtudományi Egyetem - Villamosmérnöki és Informatikai Kar - Automatizálási és Alkalmazott Informatikai Tanszék

Kocka Dominik Csaba

IOS ALKALMAZÁS FEJLESZTÉSE SWIFT ALAPÚ BACKENDDEL

KONZULENS

DR. EKLER PÉTER, EGYETEMI DOCENS

BUDAPEST, 2021 Szakdolgozat

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Kocka Dominik Csaba** szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző, cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzé tegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2021.11.26.

…….
Kocka Dominik Csaba

Összefoglaló

A mai rohanó világban szeretjük kényelmi funkciók segítségével megkönnyíteni mindennapos feladatainkat. A legtöbb ilyen funkció eredete pedig az okostelefonunk. Ez a készülék mindig a kezünkben vagy zsebünkben lapul és könnyen elérhetővé teszi a közösségekkel való interakciókat. Egy ilyen interakció például a rendezvények megszervezése, megrendezése és ezekre a munkaerő keresése. A rendezvényszervezés folyamatának ezen aspektusai rendkívül sok belefektetett energiát és időt igényelnek a rendező személyek felől. Így nem csoda, ha többi feladatukkal szemben, ő maguk sem csinálják a folyamat ezen részeit teljes odaadással és maximális prioritással.

E dokumentumban ismertetett rendszer a rendezvényszervezés első lépéseit hivatott megkönnyíteni, segítve a rendezők munkáját. Egyszerű és gyors megoldást kínál többek között a felhasználóspecifikus információk kezelésére (ilyen, például az e-mail cím tárolása) és ezek elérhetőségére, valamint a rendezvények adatainak (például helyszín, időpont) tárolására és ezen események személyzetének jelentkeztetésére és kiválasztására. Az applikáció az iOS operációs rendszert futtató eszközökre készült el, de a teljes rendszer, igény szerint más platformokkal is, könnyedén kiegészíthető.

A rendszer használata segítségként szolgálhat például az egyetemeken megtalálható Mentori/Seniori köröknek (ezen körök a fiatalabb évfolyamok számára rendeznek különböző eseményeket, például gólyatábort, gólyabált), akik az alkalmazás segítségével könnyebben nyilvántarthatják rendezvényeiket és az azon dolgozó/dolgozott személyeket.

Abstract

In today's fast world, we love to have easy to use and easy to access solutions for our everyday tasks. The source to most of these solutions are, our smartphones. This device is always within reach in our pockets or in our hands and thus making it easy to stay connected with our community and interact with them. One of these interactions can be the planning, organizing of an event and searching of workforce to manage it. This aspect of event organizing is a time and energy consuming task, so it is not a surprise if the organizers aren't really motivated during this phase of the planning compared to others.

The system described in this document aspires to make the first steps in planning and organizing an event easier. It offers a quick and easy to use solution for managing and accessing data of the users (like storing e-mail addresses), storing the data of events (like location and date) and keeping track of, and accepting applications for opened jobs. The application is made for mobile devices running the iOS operating system, but the system as a whole, can be extended to other platforms with ease.

The usage of the system can be a big help to the Mentor/Senior groups (these groups help the younger generations and organize events like freshman camps and balls for them) at multiple universities, for whom the application can provide a great way to keep track of their events and the workers on them.

1 Bevezetés

Ez a fejezet a rendszer által ellátott feladatot és annak jelentőségét hivatott bemutatni, így megértetve az olvasóval annak létjogosultságát.

1.1 Témaválasztás indoklása

A rendezvények a társadalom minden szintjén megjelennek, elkerülhetetlen részei életünknek. Az ilyen események szervezése és rendezése rendkívül nagy erőfeszítésekkel és időigényes munkákkal járnak. Ilyen például a helyszín keresése, lefoglalása, a dolgozók jelentkezésének kiírása, elbírálása és elérhetőségeik begyűjtése. Míg a helyszínválasztás folyamata mindig változik (más típusú rendezvényhez, más helyszín az optimális, más helyszínnek más a foglalási eljárása), a dolgozók és jelentkezésük procedúrája általában fix. Ez egy rendkívül repetitív feladat, mely rendezői részről rengeteg űrlapkészítéssel, azok kitöltésével, határidők betartásával és betartatásával jár. Ezen feladatokon felül még számon kell tartani a jelentkezők adatait is és esetleg a rátermettség mérlegelésénél a jelölt által eddig rendezett rendezvényeket. Dolgozói szempontból pedig nem mindig egyértelmű, hogy hol, miként folyik a jelentkezés, valamint a rendezvény helyszíne és időpontja.

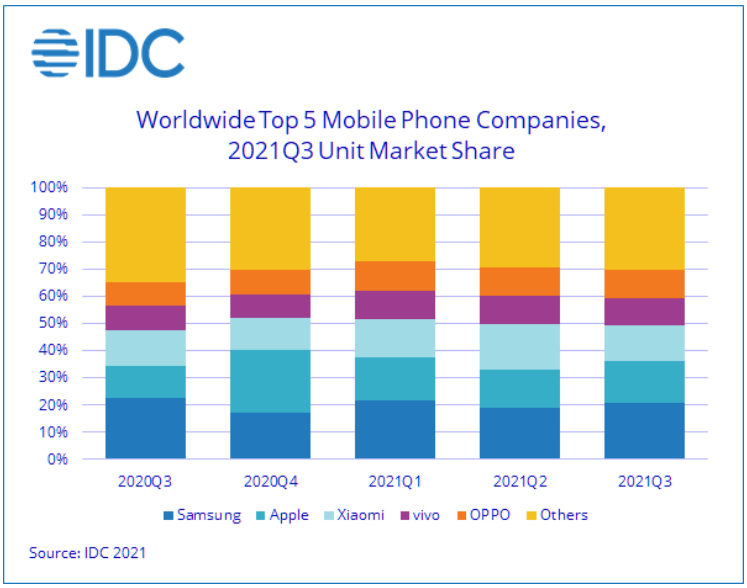
A feladatok repetitív jellegét csökkenteni tudjuk egyszerűen egy erre a célra specializált alkalmazás segítségével. Az alkalmazásoknak több formája és platformja van jelenleg a piacon, ezek egyike az iOS alkalmazás. Az iOS az Apple Inc. cég által a saját mobil készülékeikre (iPhone) fejlesztett korszerű operációs rendszer, amelyet a világon többmilliárdan használnak. Ezen alkalmazások nagy része közösségek számára készül és használati potenciáljuk maximumát több ember által használva érik el. Több készülék zökkenőmentes kommunikációjához viszont szükséges egy központi pont, azaz egy server (backend), ami számontartja és tárolja az információkat, valamint irányítja a kommunikációt, akár más rendszerrel működő eszközök között is.

Ha belegondolunk, a zsebünkben lapuló telefonunk segítségével meg tudjuk oldani az imént említett, rendezvényekkel kapcsolatos feladatokat, de itt is előjön a repetitívség és az időigényesség problémája. Milyen jó lenne, ha minden feladatot egyszerűen, pár kattintással meg tudnánk oldani és nem lenne szükség a különböző űrlapok ismételt létrehozására, megfogalmazására, egyértelműsítésére.

Sajnos minden problémára nem, de a szóban forgó rendezéssel járó feladatok problémájára megoldást kínál, az eme dokumentumban ismertetett alkalmazás, mely segít a felhasználónak az adatainak elérhetővé tételében, a dolgozóknak az események keresésében és az azokra való jelentkezésben, a rendezőknek az eseményeik kiírásában és az ezzel kapcsolatos adatok egy helyen tárolásában, valamint a jelentkezések elbírálásában.

1.2 Felhasznált technológia jelentősége/elterjedtsége

Az alkalmazás, az 1.1 előbb már említett iOS platformra készült, amely segítségével a rendszer elérhetővé válik az ilyen készülékkel rendelkező felhasználók számára. Az Apple iOS platform jelenleg is nagy részt birtokol az okostelefonok piacából.



1. ábra: Az okostelefon piac helyzete 2021 második negyedévében [1]

Mint az a diagramon is látható, az Apple terméket használó felhasználók száma minden évben nagy részét fedi le a piacnak, ezáltal elég nagy célközönséget tudnak megcélozni az erre a rendszerre készült applikációk és rendszereik.

A dokumentum a továbbiakban részletekbe menően bemutatja a rendszer minden aspektusát, felhasznált technológiáit. Tartalmazza a megvalósított feladat felhasználási eseteit, különböző folyamatait, a felhasznált technológiák képességeit és választásuknak okát, valamint pár összehasonlítást, más, hasonló funkciókat kínáló alkalmazásokkal továbbá a rendszer felépítését magasabb szinten és részletekbe menően is, vizuális szemléltető eszközöket (diagramok, modellek) és az alkalmazás használatának módját.

2 Feladatspecifikáció

Ebben a fejezetben a rendszer által megvalósított feladat részletekbe menő kifejtése olvasható.

2.1 Feladat részletes leírása

A feladat egy olyan rendszer megtervezése és elkészítése, mely képes felhasználókat azonosítani és hitelességüket ellenőrizni jelszó és tokenek (rövid egyedi karaktersorok, amelyek elősegítik az egyértelmű azonosítást, de csak a rendszernek) segítségével. Képes adataik tárolására (ilyen adat például az e-mail cím, a teljes név, a becenév, illetve a csoport, amihez tartozik a felhasználó). A rendszer továbbá tárolja és megosztja a különböző felhasználókkal egymás adatait. A felhasználókezelésen felül a rendezvények számontartása a feladat, amely a rendezvény helyszínének, időpontjának, rendezőjének és jelentkezéseinek információit tartja számon, valamint az egymáshoz hierarchikusan kapcsolódó rendezvényeket és azok adatait. Mindezt egy könnyen kezelhető rendszerben teszi, amelyben egyszerűen és érthetően megtalálható minden funkció. Ezen funkciók a következő követelményekben lettek megfogalmazva:

A regisztráció egyszerűen legyen végezhető, mindössze a személyes adatok megadásával tudjon egy új felhasználó csatlakozni a rendszerbe és ezt olyan ember tehesse csak meg, akinek e-mail címe még nincsen benne a rendszer által kezelt adatbázisban. A regisztrációhoz szükséges adatok: e-mail cím, jelszó, teljes név. Ezen adatok megadása nélkül a regisztráció ne legyen engedélyezett és ne is legyen lehetséges.

A bejelentkezéshez ne legyen másra szükség, mint a regisztrációnál megadott e-mail címre és jelszóra, ami segítségével azonosítása kerül a felhasználó és előállításra kerül számára egy token. A jelszó biztonságosan módon és visszafejthetetlen állapotban legyen tárolva. Az rendszer ezen túl már csak a token segítségével azonosítsa a felhasználót.

A kijelentkezés könnyű legyen és távolítsa el a felhasználó személyes adatait a készülékről, valamint törölje a tokent a rendszerből.

Az alkalmazáson belüli navigáció egyszerű és érthető legyen, hogy a felhasználó könnyen tudjon tájékozódni a rendszerben.

A profil adatokat könnyen meg lehessen tekinteni, ezeket lehessen szerkeszteni aktualizálás céljából. Lehessen profilképet feltölteni a könnyebb azonosítás céljából, illetve a már feltöltöttet módosítani.

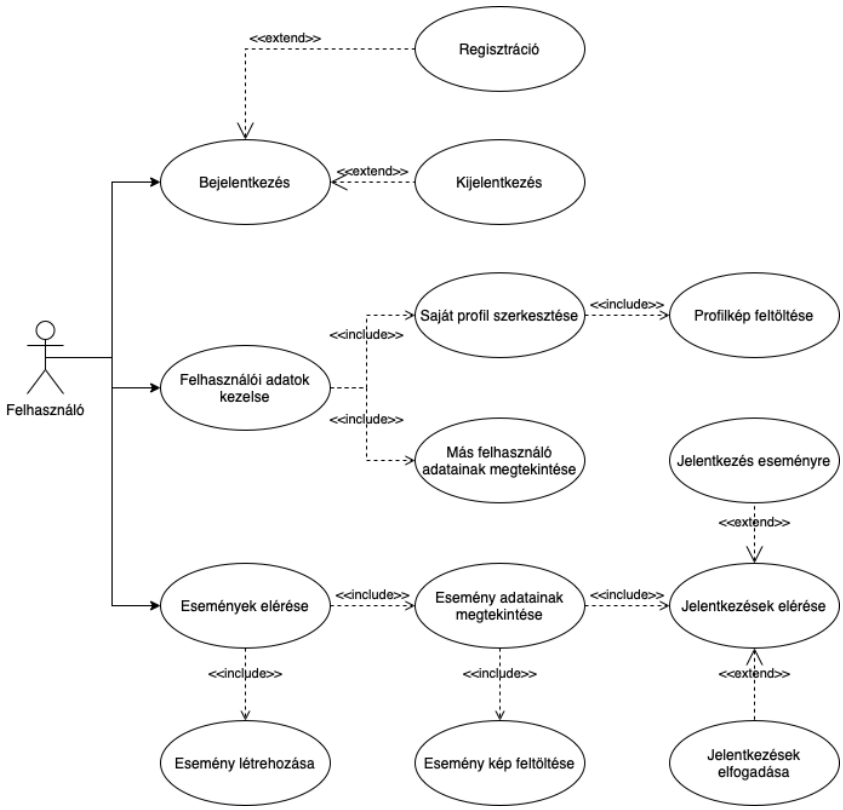
Főbb események listázása, mely elősegíti a dolgozók rendezvények közötti választását.

Új esemény létrehozása, amelyhez az esemény címe, az esemény helyszíne és kezdő, valamint záró időpontjának megadása kötelező. Ezen felül legyen lehetőség opcionális leírást megadni, hogy tisztázni lehessen az esetleges félreértéseket.

Meg lehessen nézni a már bevitt eseményeknek a felvitelnél megadott részleteit. Legyen mód a rendező felhasználói profilját, valamint az eddigi jelentkezéseket és azok aktuális állapotát megtekinteni. A helyszín térképen legyen szemléltetve.

Szervezőként meg lehessen nyitni az eseményekre való jelentkezést, lehessen elfogadni bizonyos felhasználók jelentkezését és meg lehessen tekinteni a jelentkezett felhasználók adatait is. Továbbá lehessen borítóképet feltölteni, illetve cserélni az adott felhasználó által rendezett eseményeknél.

2.1.1 Use case diagram: a program funkciói



2. ábra: A rendszer use case diagramja

A 2. ábra által szemléltetett diagram a rendszer funkcióit mutatja be. Látható, hogy a rendszernek 3 fő funkciója van, ezek pedig: a bejelentkezés kezelés, a Felhasználó kezelés és az esemény kezelés.

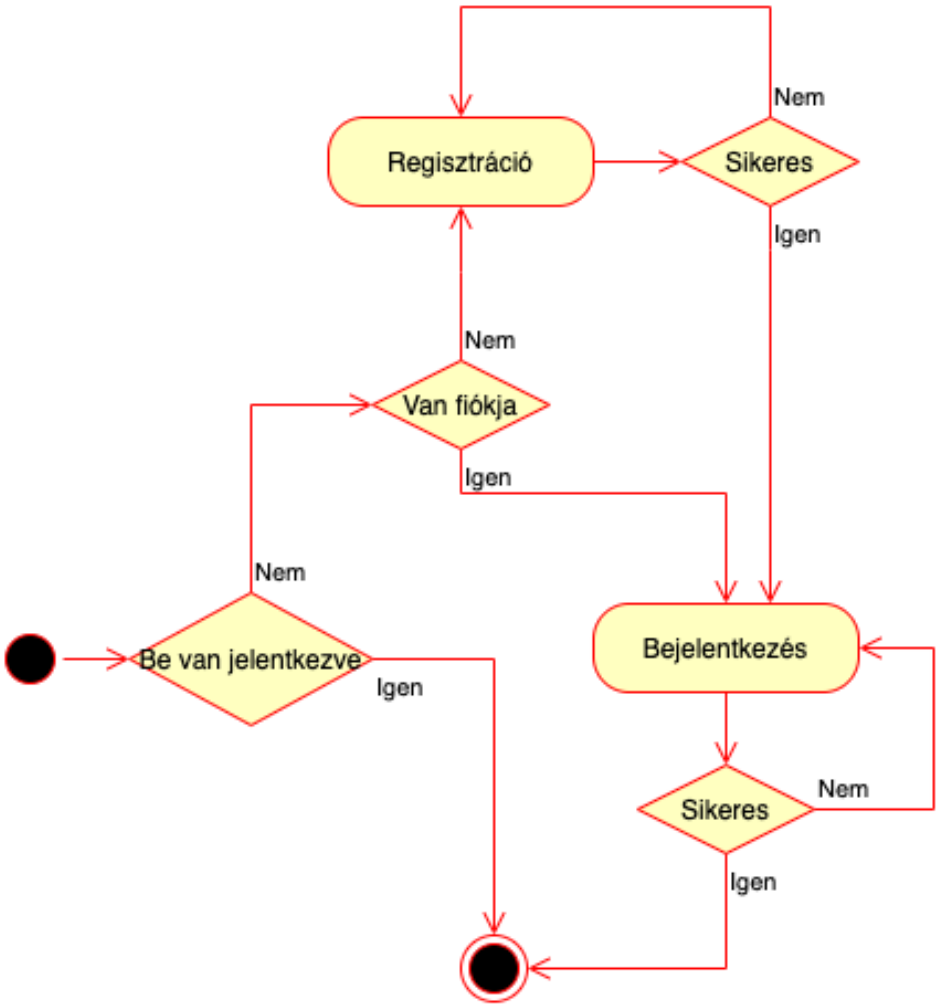
A bejelentkezési procedúrának kiegészítő folyamatai a regisztráció és a kijelentkezés.

A felhasználókezelés magába foglalja a saját profil szerkesztését és a más profiljának megtekintését. A saját profil szerkesztése tartalmazza a profilkép feltöltését.

Az események kezelésének része az új események létrehozása, valamint az események adatainak megtekintése. Ezen adatok megtekintése tartalmazza a hozzá tartozó kép lecserélését is, valamint a jelentkezések megtekintését is. A jelentkezések megtekintésének kiegészítő lehetőségei az eseményre való jelentkezés és saját eseményre való jelentkezések elfogadása.

2.1.2 Activity diagramok

2.1.2.1 Bejelentkezés

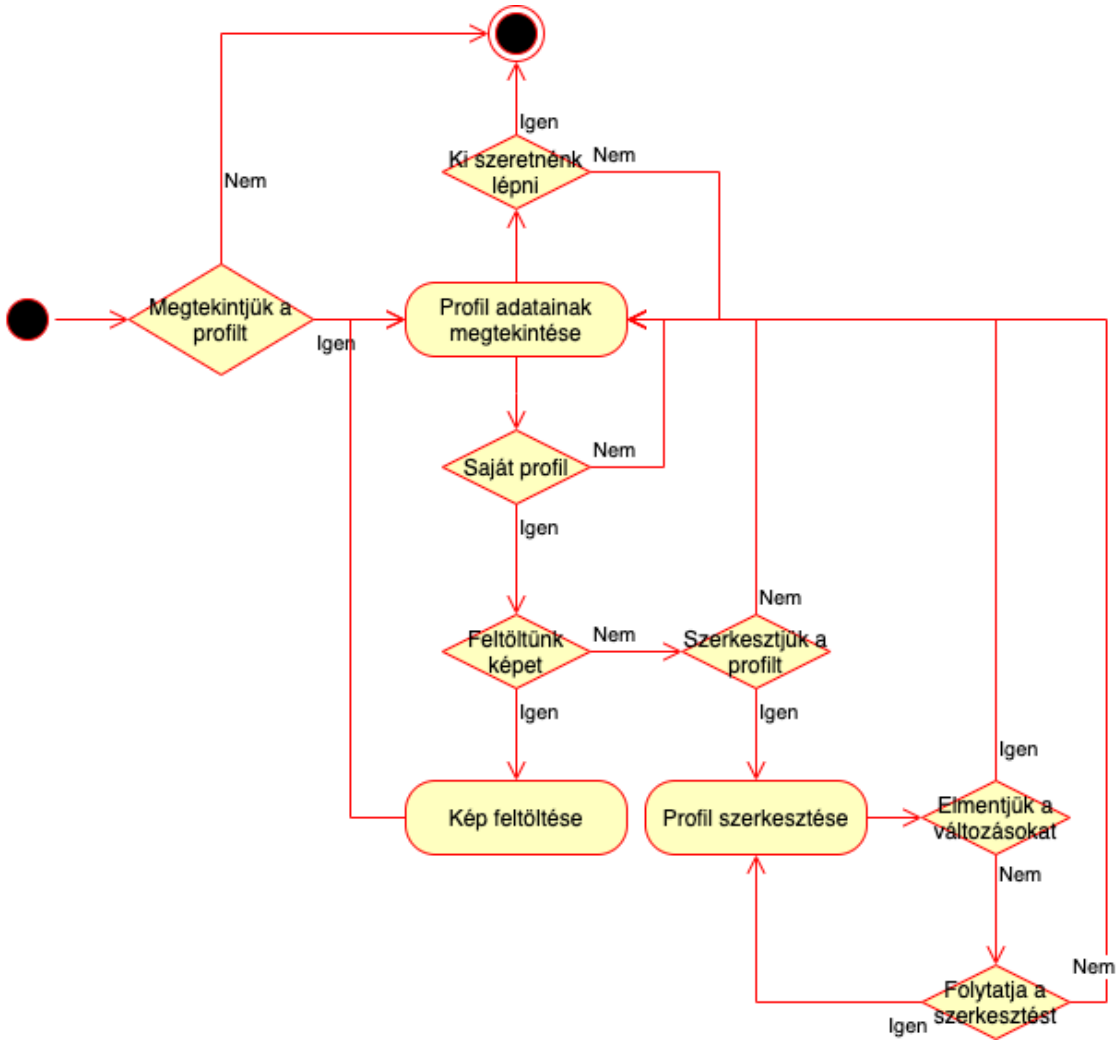


3. ábra: Activity diagram a bejelentkezés folyamatáról

A 3. ábra a bejelentkezés folyamatát mutatja be. Ezen folyamat része, hogy az alkalmazás megnyitásakor ellenőrizzük, hogy a felhasználó be van-e jelentkezve. Ha nincsen akkor a felhasználónak el kell döntenie, hogy van-e fiókja, amennyiben van, úgy a konkrét bejelentkezés veheti kezdetét, amennyiben nincs, akkor van lehetősége regisztrálni.

Ha a regisztráció nem sikertelen (például foglalt felhasználónév miatt), akkor a felhasználó bejelentkezhet újonnan létrehozott fiókjába. Amennyiben a bejelentkezés sikeres, a bejelentkezési folyamat véget is ér.

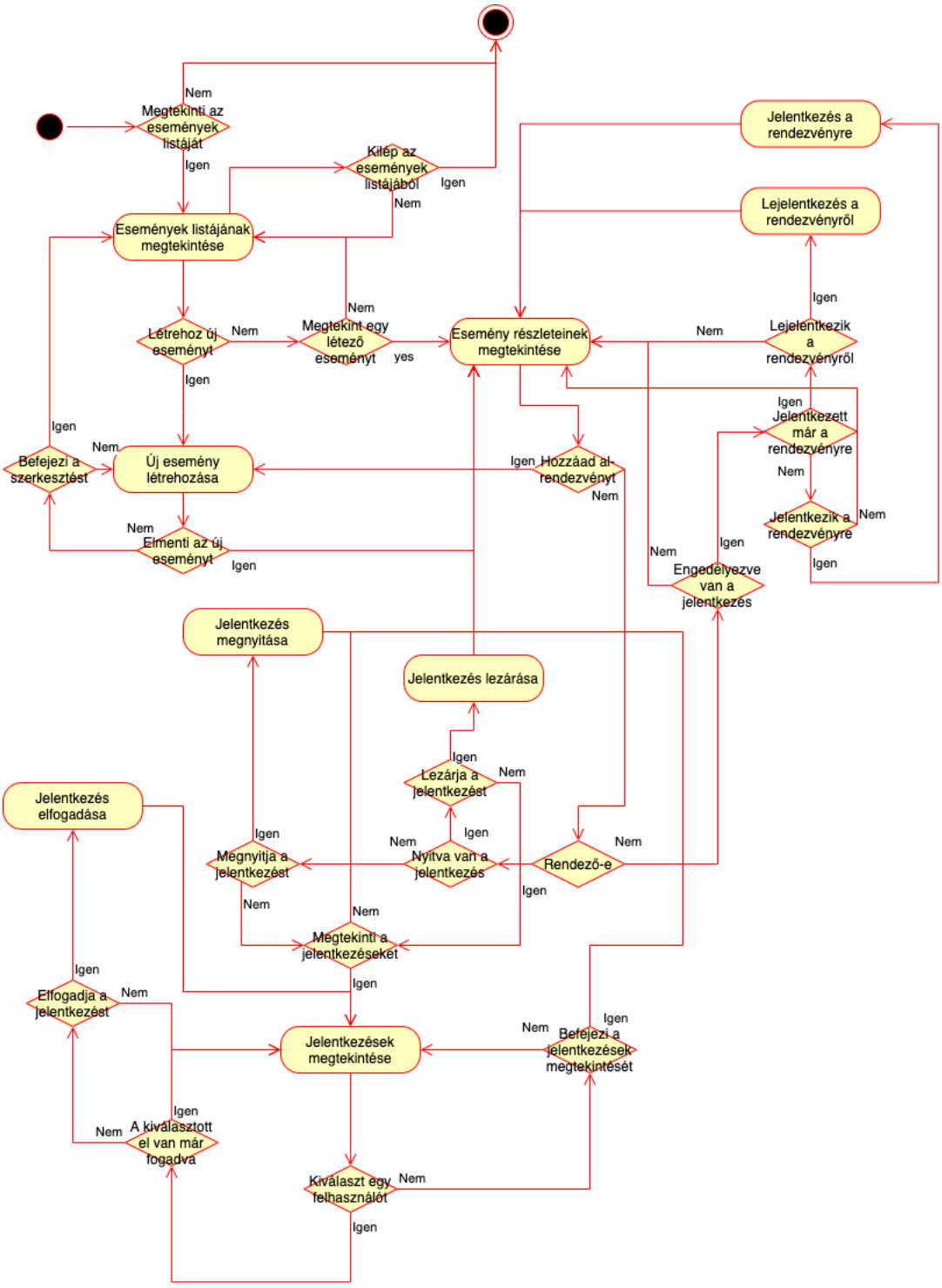
2.1.2.2 Profil kezelés



4. ábra: Activity diagram a profil kezelés folyamatáról

A 4. ábra a profil kezelés folyamatát szemlélteti. Ha úgy dönt a felhasználó, hogy megtekint egy profilt, abban az esetben a profil megtekintéséhez jut. Innen dönthet úgy, hogy kilép, ekkor véget is ér a folyamat, de, ha a megtekintett profil a sajátja, akkor határozhat úgy, hogy fel szeretne tölteni egy képet, ha feltölt, már az új képpel tekintheti meg a profilt. Amennyiben nem tölt fel, úgy dönthet, arról, hogy szeretné-e szerkeszteni az adatait, ha szeretné, akkor a szerkesztés folyamata kezdődik meg, ami véget sem ér, ameddig a változtatásokat nem menti, vagy abba nem hagyja a szerkesztést.

2.1.2.3 Rendezvény kezelés



5. ábra: Activity diagram a rendezvények kezeléséről

Az 5. ábra a rendezvények kezelésének menetét szemlélteti. A felhasználó, ha úgy dönt, hogy megtekinti a rendezvények listáját, akkor ezt meg is tudja tenni.

A rendezvények listájából dönthet úgy, hogy kilép és ezáltal véget is ér a folyamat. Amennyiben létre szeretne hozni egy rendezvényt, úgy az új esemény létrehozásának folyamata indul el. Azonban, ha nem szeretne létrehozni eseményt, akkor dönthet úgy, hogy megtekint egy már meglévőt, ebben az esetben a rendezvény megtekintésének folyamata indul el.

A felhasználó dönthet úgy, hogy hozzáad egy al-rendezvényt, ami által megkezdődik a rendezvény létrehozásának folyamata. Ha nem hoz létre új al-rendezvényt, akkor a program eldönti, hogy a felhasználó a megtekintett program rendezője-e vagy sem. Amennyiben nem rendező, úgy eldől, hogy a jelentkezés nyitva van-e. Ha nincs, akkor továbbiakban is megtekinti a rendezvény részleteit, de amennyiben nyitva van, úgy az alapján, hogy jelentkezett-e már, megteheti vagy lemondhatja azt.

Ha rendezőről van szó, úgy a jelentkezés nyitottságától függően lezárhatja vagy megnyithatja a jelentkezéseket. Amennyiben nem szeretne módosítani a jelentkezés nyitott/zárt állapotán, úgy megtekintheti a jelentkezéseket.

A jelentkezések megtekintésénél van módja kiválasztani egy jelentkezett felhasználót, majd, ha még nem fogadta el jelentkezését, akkor megteheti azt.

Az új esemény létrehozása folyamatban lehetősége van elmenteni az eseményt, így a részletek megtekintéséhez jut, de el is vetheti, ezzel pedig az események listája jelenik meg.

3 Irodalomkutatás

Ez a fejezet a felhasznált technológiákat mutatja be és összehasonlítja más alkalmazások hasonló megoldásaival a rendszert.

3.1 Felhasznált technológiák

A következőkben a rendszer elkészítéséhez felhasznált technológiákat ismerhetjük meg.

3.1.1 Swift

A Swift egy modern programozási nyelv, melynek készítése során ügyeltek, hogy elősegítse a biztonságos kód írását és a különböző tervezési minták megvalósítását, miközben gyorsan, teljesítményre optimalizáltan működik.

A Swift célja, hogy egy átfogó, könnyen érthető és tanulható, de mégis funkciókban bővelkedő programozási nyelvet biztosítson minden elérhető platformra, legyen az, rendszer programozás, mobil vagy számítógépes applikáció, de akár felhő alapú technológia is. A nyelv továbbá igyekszik a kód írását és karbantarthatóságát is segíteni a fejlesztő számára. Erre három főbb célt fogalmaztak meg a készítők.

Biztonság. Azaz a kód írása közben elképzelt funkcióknak futás közben is biztonságosan kell viselkednie. Ez azt jelenti, hogy a nem definiált viselkedéseket észre kell venni a környezetnek és a fejlesztői hibákat el kell kapni még a szoftver kiadása előtt. Ha a biztonság mellett döntünk, néha érezhetjük úgy, hogy nagyon szigorú a nyelv, de hosszú távon rengeteg idő takarítható meg evvel.

Gyorsaság. A Swift C (programozási nyelv) alapokra épül és ezt is tervezi helyettesíteni, tehát a többi hasonló alapú nyelvvel kell összehasonlítani a teljesítményét. Ennek a teljesítménynek kiszámíthatónak és konzisztensnek kell lenni, nem csak gyorsnak rövid időtartamokban, amik takarítást igényelnek később.

Kifejezőség. A nyelv a több évtizednyi számítástudománybeli fejlődésnek köszönhetően egy egyszerű szintaxissal rendelkezik, de ennek ellenére modern eszközökben sem szenved hiányt. Egy folyamatosan fejlődő nyelvről van szó, amely követi a trendeket és így folyamatosan fejlődik. [2]

3.1.2 UIKit

A UIKit Framework (keretrendszer) megteremti az infrastruktúrát iOS és tvOS (Apple TV készülékek operációs rendszere) appok fejlesztéséhez. Egy ablak és nézet architektúrát biztosít a különböző interfészek implementálásához. Segíti a Multi-Touch (egyszerre több érintés kezelése) és más bemeneti lehetőségek beépítését az applikációkba. A fő szálat használja a felhasználóval való interakció kezeléséhez.

Az egyéb, keretrendszer által biztosított funkciók közé tartozik az animációk támogatása, a dokumentumkezelés, rajzolás és nyomtatás támogatás, információk biztosítása a futtató eszközről, szövegek kezelése és megjelenítése, keresés támogatás, kisegítő lehetőségek könnyű implementálása, applikáció kiterjesztések támogatása, valamint az erőforráskezelés. [3]

3.1.3 Vapor

A Vapor egy HTTP (HyperText Transfer Protocol, egy hálózati kommunikációt elősegítő protokoll) web keretrendszer és a leggyakrabban használt környezet, amely támogatja a Swift alapú web fejlesztést. Egy olyan környezetet biztosít, mely kifejező és könnyen használható alapokat nyújt a weboldalak és API-k (application programming interface, azaz alkalmazásprogramozási felület) fejlesztéséhez. [4] [5]

Képességei közé tartozik: a nem blokkoló, esemény vezérelt architektúra, amely az Apple SwiftNIO (az Apple által készített esemény vezérelt hálózati környezet) keretrendszerére épült, Swiftben, az erőteljes, de könnyen tanulható programozási nyelvben íródott, Kifejező, protokoll orientált dizájn a típus biztonságra fordított figyelemmel.

A Vapor több mint csak egy webes keretrendszer. A projekt magában foglal több mint száz hivatalos és közösség által karbantartott szerver központú Swift csomagot. [6]

Egy alapvető API létrehozása nem több néhány sornál:

```
import Vapor

let app = try Application(.detect())
defer { app.shutdown() }

app.get("hello") { req in
    return "Hello, world."
}

try app.run()
```

3.1.4 Fluent

A Fluent egy Objektum-Relációs leképzésekkel dolgozó keretrendszer a Swift programozási nyelvhez. Tökéletesen használja ki a Swift erősen típusos nyelv voltát, hogy biztosítson egy könnyen használható interfészt az adatbázis felé. A Fluent használata a modell típusok készítése köré épít, amelyek az adatstruktúrákat reprezentálják az adatbázisban. Ezek a modellek, aztán fel vannak használva a CRUD (Create, Read, Update, Delete), azaz Létrehozás, Olvasás, Frissítés és Törlés műveletek megvalósításához, amely a nyers lekéréseket hivatott helyettesíteni. [7]

A Fluent egy adatbázis típustól független keretrendszer, így lehetővé teszi az adatbázisok könnyed cseréjét.

3.1.5 Alamofire

Az Alamofire egy iOS és macOS (Az Apple MAC számítógépeinek operációs rendszere) rendszerekre készült, Swift alapú HTTP hálózati könyvtár, mely egy elegáns interfészt biztosít az Apple Foundation (alapszintű swift adatstruktúrák keretrendszere) hálózati funkciójának tetejére építve, ezzel leegyszerűsítve a legtöbb általános hálózati feladatot.

Az Alamofire kiváló módon biztosítja az egymás után fűzhető kérés/válasz metódusokat, JSON (JavaScript Object Notation, egy ember által is értelmezhető szöveg alapú szabvány a gépek közötti kommunikációra) paramétereket, válasz szerializációt, autentikációt és még sok más funkciót.

A keretrendszer eleganciája abban rejlik, hogy teljesen Swiftben íródott és semmit nem örökölt az Objective-C programozási nyelvben található alternatívájától (AFNetworking). [8]

3.1.6 SwiftGen

A SwiftGen egy eszköz, amely segít Swift kódot generálni, olyan erőforrásokhoz, melyeket szöveges alapon, név szerint kellene azonosítani a kódban. Ilyen erőforrások lehetnek, a képek, képernyők nevei stb. Ezzel a megoldással, viszont típus biztonságos (type safe) lesz a használatuk.

Több előny is származhat ennek az eszköznek a használatából. Ilyen előny az elgépelt azonosítókból eredő hibák kivédése, az automatikus kiegészítés használhatósága,

a nem létező erőforrásnevek használatától való védelem, ez mind a fordítóprogram által védve, így a futás közbeni fagyásokat és összeomlásokat elkerülve.

Nem mellesleg teljesen testre szabható konfigurációs sablonok segítségével, tehát még tartalmaz előre definiált sablonokat. Ezen felül elkészíthetjük sajátunkat is, amely úgy generál kódot, ahogy mi szeretnénk. [9]

3.1.7 KeychainAccess

A KeychainAcces egy nagyon egyszerű becsomagoló keretrendszer a Keychain (az Apple készülékeken, a jelszavak és különböző érzékeny adatok biztonságos és titkosított tárolását végző keretrendszer) rendszerhez, amely működik iOS és macOS készülékeken. Segít egyszerűen és sokkal kellemesebben használni a Keychain API-t. [10]

Az említett keretrendszer az alkalmazáson belül, a tokenek tárolásánál került alkalmazásra.

3.1.8 IQKeyboardManager

Az iOS alkalmazások fejlesztése közben könnyű belefutni olyan kellemetlenségekbe, mint a felcsúszó virtuális billentyűzet, mely eltakarja a használni kívánt beviteli mezők egy részét. Ezt kikerülni csak figyelmesen és sok kód implementálásával lehet. Ennek kiküszöbölésére született az IQKeyboardManager, ami segít a billentyűzettel kapcsolatos problémák megoldásában.

Főbb funkciói tartalmazzák a kódolás mentes beüzemelést és az automatikus működést.

Az IQKeyboardManager minden orientációban működik és támogatja az eszköztárak használatát. E mellett opcionális kényelmi funkciókkal is rendelkezik, melyek egyszerűen konfigurálhatók. Ilyenek, a beviteli mezőtől való távolság, a következő/előző gombok működése, hangok lejátszása navigálás közben és még sok más. [11]

3.1.9 LocationPicker

A LocationPickerViewController egy UIViewController (UIKitben található, képernyő megjelenésért felelős osztály) alosztály, amely segít a helyszínek

kiválasztásában. Mindezt egy funkcionális keresővel és egy térképpel segíti elő, mely egyszerű helymeghatározást biztosít.

Segítségével, a felhasználó egyszerűen választhat a keresőmezőt használva vagy hosszan nyomva tartva a térkép kijelölendő területét. [12]

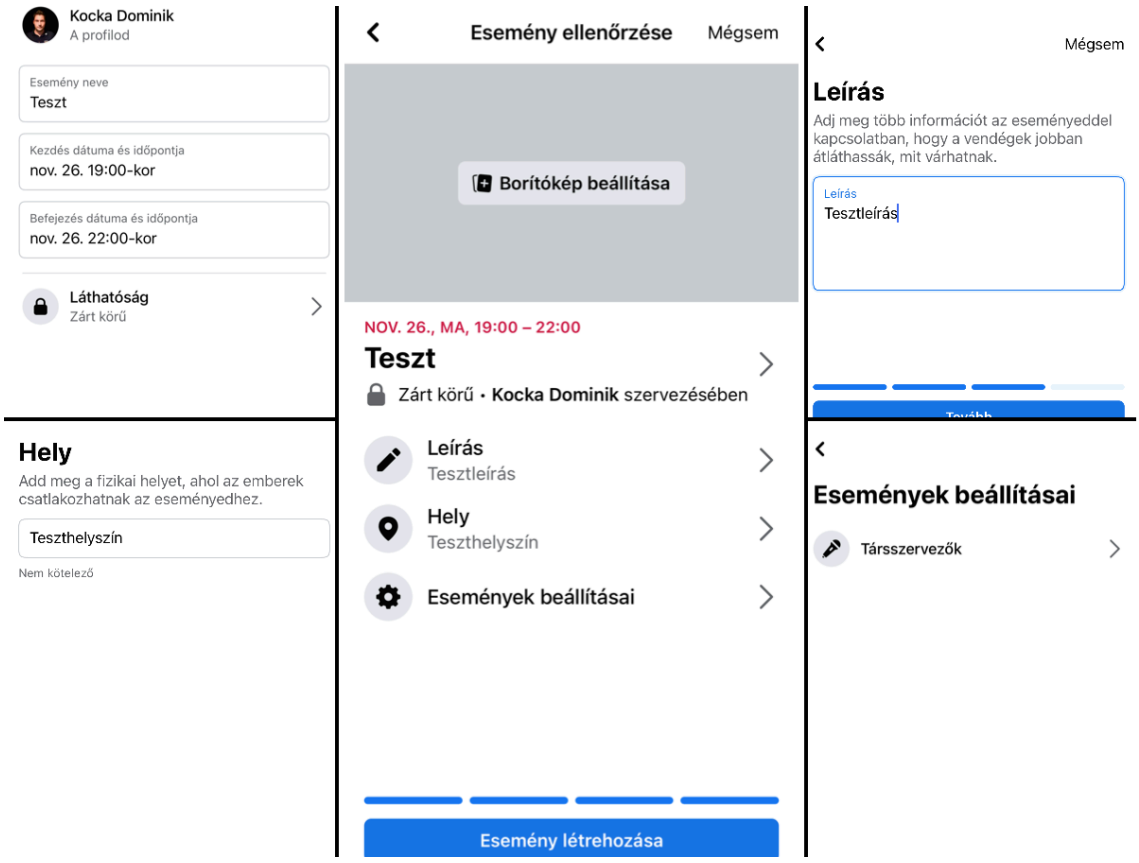
Az említett segédosztály az eseményhelyszínek kiválasztásánál került felhasználásra.

3.2 Hasonló megoldások

Ez a fejezet rávilágít más applikációk hasonló funkcióinak hiányosságaira.

3.2.1 Facebook események

A Facebook az amerikai Meta cég által üzemeltetett közösségi oldal, amelyet több millió ember használ napi szinten, akár események meghirdetésére is, de rendkívül sok funkcióval rendelkezik e-mellett.



6. ábra: Facebook [13], esemény létrehozása és beállításai

A Facebook is támogatja az események létrehozását és kezelését, de vannak hiányos funkciói. Ilyen a rendezők válogatására és kiválasztására való lehetőség, illetve a rendezőnek való jelentkezés funkciója.

A 6. ábra tartalma a Facebook rendszer iOS alkalmazásán egy esemény létrehozásának lépései.

A bal felső szekción látható, hogy az első képernyőn megadható adatok nagyjából fedik az ebben a dokumentuman részletezett rendszer által megvalósított eseménykezelés által tárolt adatokat.

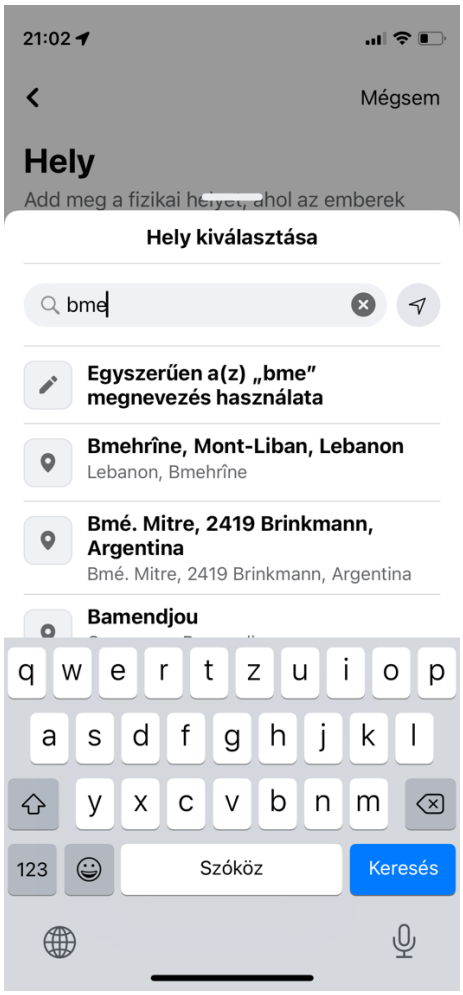
A bal alsó szekcióban található képen a helyszín meghatározása is megjelenik, hasonlóan a dokumentált alkalmazáshoz, itt mindkét említett alkalmazás rendelkezik térképes és szöveges alapú adatbeviteli lehetőségekkel.

A jobb felső szekció az esemény készítésének 3. lépését mutatja be, mely egy leírást takar, ami szintén megtalálható mindkét rendszerben.

A középső, nagy kép a létrehozás összefoglalását tartalmazza, ez lehetőséget biztosít az előző lépésekben megadott adatok módosítására, valamint egy kép feltöltésére is, amely megkönnyíti a felhasználók számára a rendezvény azonosítását. Ez a funkció (a kép feltöltése eseményhez) szintén megtalálható az eme dokumentumban részletezett rendszerben.

Az utolsó szekció, a jobb alsó, mely a középső képen található esemény beállításai menüre kattintva érhető el. Itt látható, hogy a társszervezők meghívásos alapon adhatók csak meg, tehát nincsen lehetőség toborzásra és jelentkezések bírálására.

A később található 7. ábra látható továbbá, hogy a Facebook applikációja nem ad lehetőséget a helyszínnek térképen való kiválasztására, erre csak szöveges keresőmezőt biztosít.



7. ábra: Facebook [13], helyszín keresése

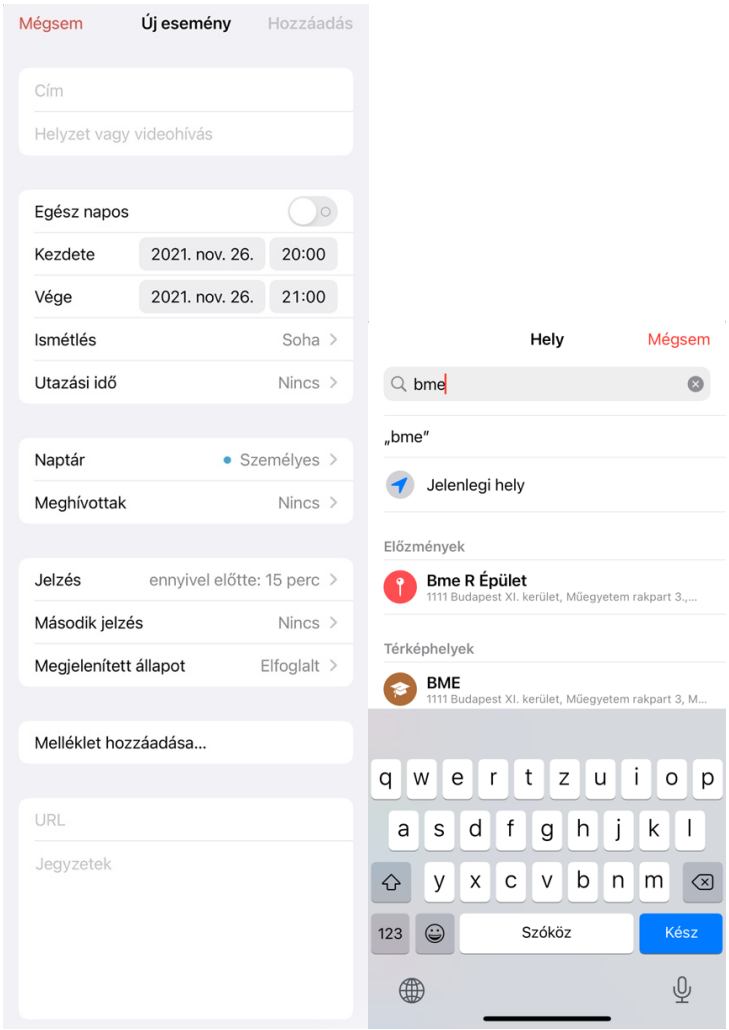
3.2.2 Apple naptár – események

Az Apple naptár alkalmazása főleg a cég készülékeit használó felhasználók között kedvelt módja az események rendszerezésének.

A később található 8. ábra bal oldalán látható, hogy az Apple készülékekbe integrált naptár alkalmazás sem fedi le a rendezvényszervezéshez szükséges összes funkciót.

Az Apple rendszere egy eseményhez rendelten képes kezelni: címet/nevet, helyszínt, időtartamot és meghívottakat is. Itt látható, hogy meghívottakat és nem jelentkezőket tud felvenni a rendező egy eseményhez.

Ezen a problémán felül még ott van, hogy nem lehet képet hozzárendelni az eseményekhez, erre nem ad lehetőséget a rendszer.



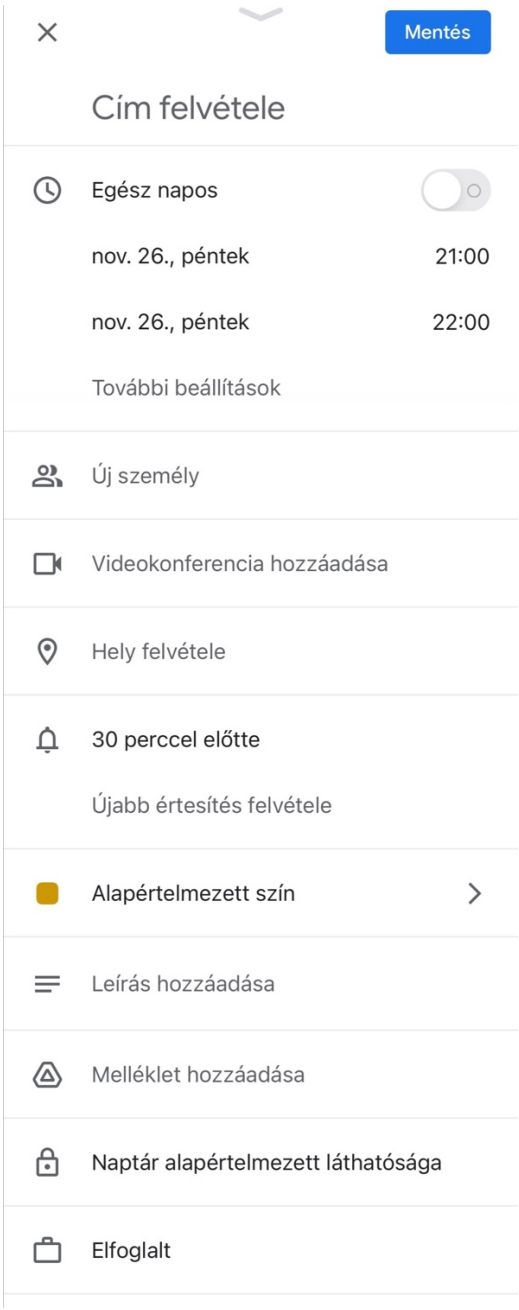
8. ábra: Apple naptár [14], új esemény hozzáadása

A 8. ábra jobb oldala mutatja, hogy a helyszín kiválasztásához itt is csak szöveges keresőmező segítségével van lehetőségünk a keresésre, itt is hiányzik a térkép alapú helymeghatározás.

3.2.3 Google Calendar – események

A Google Calendar a Google cég infrastruktúrájába tartozó naptár és esemény kezelő alkalmazás. Ezt az alkalmazást főleg a Google Android, mobilokon futó operációsrendszerének mindennapos használói preferálják, rendszerbe való integráltsága miatt.

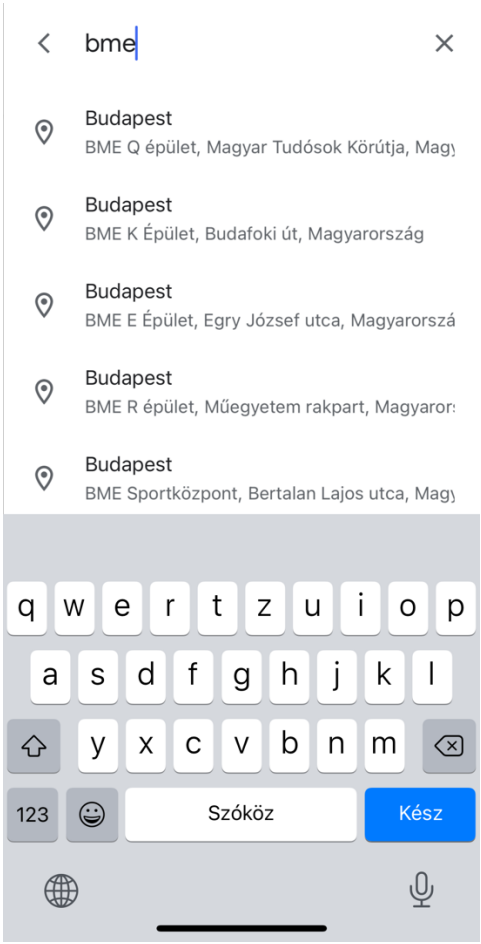
Míg ez egy inkább Android rendszert futtató készülékeken preferált módja a tervezésnek, létezik iOS-en futó verziója is, ezt fogom bemutatni.



9. ábra: Google Calendar [15], új esemény hozzáadása

A 9. ábra tartalmán látható, hogy a Google alkalmazása is lehetőséget ad a már megszokott funkciók használatára. Ezek közé tartozik itt is a cím/név felvétele, az időpont kijelölése, valamint a helyszín felvétele.

Itt is hiányzik a jelentkezők kezelésére utaló funkció, csak meghívott személyeket tud a felhasználó hozzáadni és itt sincs lehetőség jelentkezők versenyeztetésére, előéletük megtekintésére vagy akár a jelentkezésük kezelésére.



10. ábra: Google Calendar [15], helyszín kiválasztása

A 10. ábra tartalma alapján látható, hogy a helyszín kiválasztására itt is csak a már jól megszokott szöveges keresőmező érhető el, a térképes keresésnek itt sincs nyoma.

4 Felső szintű architektúra

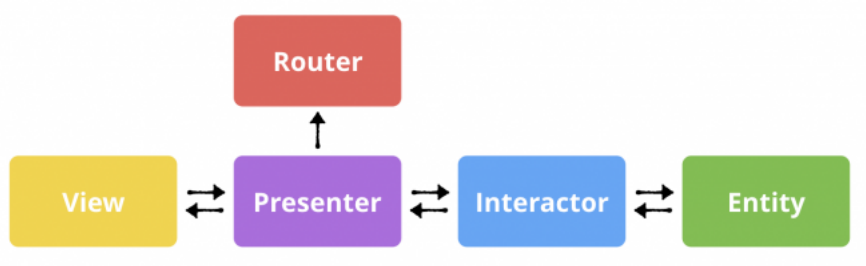
Ez a fejezet a rendszer, azaz az alkalmazás és a szerverprogram architektúráját hivatott bemutatni magas szinten.

4.1 High level architektúra

A teljes rendszer egy részben az iOS rendszerre szabott, módosított VIPER architektúrára épül.

4.1.1 A VIPER architektúra

A VIPER egy olyan tervezési és felépítési szemlélet, mint az MVC (Model-View-Controller) vagy az MVVM (Model-view-viewmodel), de még ezeknél is jobban szétválasztja a kódot úgy, hogy minden osztály csak kis részéért feleljen a kódnak. Az Apple stílusú MVC a fejlesztőket arra motiválja, hogy az alkalmazás logika nagy részét UIViewController leszármazott alosztályaiba helyezzék. A VIPER, mint elődje az MVVM is, ennek a problémának megoldására törekszik.



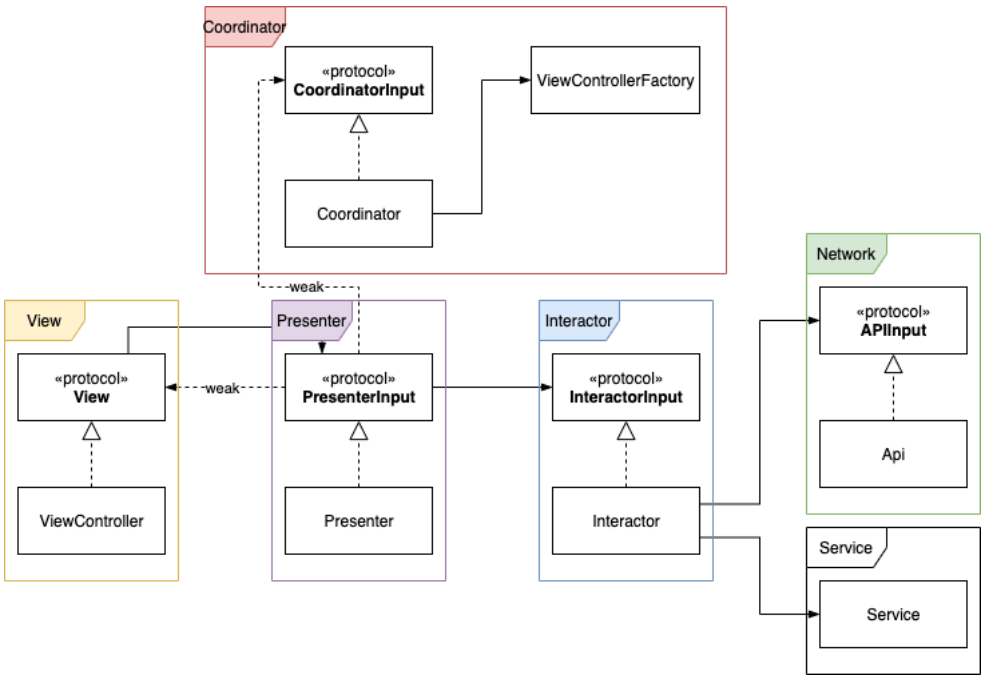
11. ábra: VIPER architektúra felépítése [16]

Mint az a 11. ábra is látható az architektúra komponenseinek nevéből épül fel a VIPER mozaikszó. (View-Interactor-Presenter-Entity-Router) Ha megnézzük a diagramot, láthatjuk, hogy egy teljes út vezeti az adatokat a nézet (View) és entitás (Entity) komponensek között.

A diagram alapján az is jól látható, hogy a VIPER a többi előbb említett architektúrával ellentétben elválasztja a nézetek (View) és adatmodellek (Entity) logikáját. Csak a Presenter kommunikál a nézettel és csak az Interactor beszél a modellel. A Presenter és Interactor egymással kommunikálva oldja meg a két végpont közötti utat. A Presenter csak a megjelenítéssel és a felhasználói interakcióval foglalkozik, míg az Interactor az adatok kezelésével van megbízva. [16]

4.1.2 Az alkalmazásban használt VIPER

Az alkalmazás a VIPER iOS specifikus verzióját alkalmazza, ami segít a könnyű bővíthetőségben és a komponensek könnyű lecserélésében, kihasználva a Swift nyújtotta protokollok képességeit.



12. ábra: Az alkalmazás által használt architektúra

A 12. ábra tartalmaz egy osztálydiagramot, ami az alkalmazás által használt architektúrát mutatja be.

4.1.2.1 View

A View feladata szigorúan a megjelenítéssel kapcsolatos feladatok ellátása, csak ez használhatja a UIKitben specifikált megjelenítési osztályokat, például a UIView és UIViewController leszármazottjait.

A UIViewControllerből leszármazott osztályoknak meg kell valósítani a hozzájuk tartozó View protokollt, mivel ezek definiálják a Presenter részéről elérhető interfészt. A Presenterre referenciát közvetlenül nem, csak protokollon keresztül tart.

Az adatok utaztatása a View és Presenter között egy PresentationModel struktúrába csomagolva történik. A PresentationModel feladata az összetartozó egyszerű adatok becsomagolása és ezáltal egy köztes adattovábbítási réteg létrehozása a View és Presenter között.

A felhasználói interakciók által kiváltott eseményeket a View továbbítja a Presenter felé.

```
protocol XView: class {
  var presenter: XPresenterInput? { get set }
  func foo(number: Int)
  func bar(presentationModel: XPresentationModel)
}
```

Példa egy view protokollra

4.1.2.2 Presenter

A Presenter implementálja a nézettel és megjelenítéssel kapcsolatos logikát. A protokolljában definiálva vannak a felhasználó által kiváltható eseményeknek megfelelő függvények. Az Interactorral való kommunikációja során általában modell típusú adatokat kap, ezeket átalakítja vagy becsomagolja egy PresentationModel struktúrába, ezáltal a View számára megjeleníthetővé téve azt. A hozzá tartozó View objektumokat csak a protokolljukon keresztül tartja számon egy gyenge referenciával.

A felhasználói események, amelyek adatmódosítással járnak, továbbításra kerülnek az Interactor felé, amelyek viszont navigációval járnak, azok a Coordinator felé, amit szintén csak a protokollján keresztül ismer a Presenter és gyenge referenciaként tárol.

```
protocol XPresenterInput: AnyObject {
  weak var view: XView? { get set }
  var interactor: XInput? { get set }
  func foo()
}
```

Példa egy Presenter protokollra

4.1.2.3 Coordinator

A Coordinator felelős a képernyők és nézetek közötti navigáció kezeléséért.

Ebben a rétegben jönnek létre az Interactor, a View és a Presenter objektumok és itt is történik meg a beállításuk, konfigurálásuk. Míg a különböző nézetek létrehozását a ViewControllerFactory intézi el, a konfigurációjuk és referenciáik beállítása a Coordinator feladata.

Az Alkalmazás elindításakor egy, a többi Coordinator felett álló AppCoordinator jön létre, ami kezeli a különböző funkciók Coordinatorai közötti váltásokat, illetve a navigációs stackben használt gyökér ViewControllert.

```
protocol XCoordinatorInput {
    func navigateToY()
}
```

Példa a Coordinator protokolljára

4.1.2.4 Interactor

Az Interactorok egy adott funkció üzleti logikával kapcsolatos műveleteiért és adatok módosításáért felelnek. Ez a réteg vagy a memóriában tárolt modellekhez nyúl vagy – mint jelen esetben is – hálózati hívásokkal kommunikál a háttérrendszerrel és kezeli a Modelleket.

Az adatokkal történő műveletek eredményeit továbbítja a Presenter felé.

```
protocol XInteractorInput: AnyObject {
    func foo(model: Model,
             completion: @escaping (Result<XInteractorSuccess,
                                    XInteractorError>) -> Void)
}
```

Példa egy Interactor protokollra

4.1.2.5 Network

A hálózati kommunikációs rétegen keresztül történnek a háttérrendszer felé a HTTP hívások. A különböző funkciók feladatait is különböző, hálózati hívást kezelő osztályokba csomagoljuk. A hálózati kommunikációban utazó adatok JSON formátumúak és úgynevezett DTO-ba (Data Transfer Object, egy olyan objektum, mely tulajdonságaiban, paramétereiben és azok elnevezésében megegyezik minden platformon, így biztosítva a konzisztens sorosítást) csomagoltan kezelendők. Ezek a DTO osztályok az Encodable és a Decodabel, sorosításhoz használt osztályok leszármazottai.

```
protocol XAPIInput {
    func editX(with model: Model,
               completion: @escaping (XDto) -> Void)
    func getX(completion: @escaping (XDto) -> Void)
}
```

Példa a Network réteg egy protokolljára

4.1.2.6 Service

A Service réteg komplexebb üzleti logikai feladatok (például Autentikáció) kiszervezésére használt, ezek előnye, hogy általában singleton (Egy programozási minta, mely szerint egy osztálynak csak egy példánya lehet) mintát alkalmaznak, így az alkalmazás bármely pontjáról elérhetőek. Általában ezek a feladatok több interactor

között megoszló hasonló működést vagy ugyanazon információ kezelését egyszerűsítik le. Itt is érdemes protokollokba csomagoltan kezelni a Serviceket, mert így könnyen cserélhetővé válnak.

4.1.3 A backend felépítése

A háttérrendszer bár csak az adatok tárolásáért és ezek kliensek felé való kiszolgálásáért felelős, mégis említésre méltó, hogy ez a rendszer is több komponensből áll.

Felépítése hasonlít egy MVC architektúrára a V, azaz View nélkül.

Tartalmaz Modelleket, amely az adatok strukturáltságát és kezelését könnyíti meg.

És vannak benne Controllerek, amik a REST (Representational State Transfer, azaz Reprezentációs Állapot Átvitel) hívásokat hivatottak kiszolgálni és kezelni.

Itt említésre méltó továbbá a Migration osztályok feladata, amely előkészíti az adatbázist a modellek tartalmának tárolására.

4.2 Rendszer felépítései, komponensei

Ebben a részben a rendszer három főbb komponensét ismerhetjük meg.

4.2.1 Autentikáció

Az autentikáció a felhasználó azonosítását jelenti. Ez a funkció biztosítja, hogy a rendszer tudja melyik felhasználóhoz, mely adatok tartoznak. Az autentikációs modul feladatai közé tartozik a regisztráció kezelése, a bejelentkezés és a kijelentkezés.

Az autentikáció működése egy egyszerű token kezelésen alapszik.

A felhasználó bejelentkezéshez a jelszavát és felhasználónevét szükséges, hogy megadja, ez egy DTO-ba csomagolva JSON formátumban érkezik a backendhez. Az adatok biztonságos kommunikációját a HTTPS (HTTP protokoll biztonságos változata, SSL vagy TLS alapú titkosítás segítségével) protokoll által nyújtott titkosítás végzi.

A bejelentkezés sikeressége esetén a kliens egy tokent kap, mely segíti a háttérrendszer általi kizárólagos azonosíthatóságot. Ez a token egy véletlenszerűen generált karakterlánc, melyet a backend eltárol az általa kezelt adatbázisban, a megfelelő felhasználói azonosításra szolgáló adatokkal (User ID) együtt. A kliens a bejelentkezésen

kívül, minden autentikációt kívánó kérésénél, már csak a token segítségével azonosíthatja és azonosítja is magát.

A regisztráció működéséhez nem szükséges semmilyen autentikációs metódus, mivel itt még nem ismeri a backend a regisztráló felhasználót. A regisztrációhoz szükséges személyes adatok szintén egy DTO-ba csomagolva utaznak.

4.2.2 Profil kezelés

A profil kezelés a saját és más profiladatok megtekintését, valamint a saját profil módosítását jelenti. Bármely profillal kapcsolatos művelet létrejöttéhez tokenes autentikáció szükséges.

A saját profil adatainak megtekintéséhez a rendszer a tokennel való azonosítás alapján biztosítja az adatokat, míg, ha idegen felhasználó adatait szeretnénk megtekinteni, szükségünk van a felhasználó azonosító (User ID) kódjára, mely egy UUID (universally unique ID, egy 4 bájtos, az univerzális egyediséget biztosító alfanumerikus karaktersorozat) típusú objektum. Ennek a típusnak köszönhetően nehezen lehet csak találomra profilokat nézegetni, mint azon alkalmazásoknál, melyek egy szimpla számot használnak azonosító értékként.

```
A85FA00D-F454-4065-B20E-06A855D7F45F
```

Példa egy ilyen azonosítóra.

A profil szerkesztéséhez is szükség van a tokenre, itt nyilvánvalóan, csak a saját adatainkat tudjuk szerkeszteni. Ezek az adatok szintén egy DTO-ba csomagoltan jutnak el a backendhez.

Ezen funkciókon felül van lehetőség fénykép csatolására vagy lecserélésére a profilunkon. Az új fénykép bármilyen felbontással/minőséggel rendelkezhet, a kliens, ezt tömöríti, majd mivel erre a feladatra nem célszerű JSON formátumot használni, Multipart (nagyobb terjedelmű adatok továbbítására leggyakrabban használt HTTP alapú kommunikációs formátum) formátumban továbbítja a backend felé.

4.2.3 Rendezvény kezelés

A rendezvények kezelése szintén a tokenes autentikációt kívánja meg a kliensektől. Ez minden lehetséges művelet elvégzéséhez szükséges, akár csak megtekintésről, akár módosításról van szó.

A rendezvények listázását, bármely autentikált felhasználó elérheti, ez a fő rendezvények listáját egy tömbbe csomagolva adja vissza, mely csak az olyan eseményeket tartalmazza, melynek a hierarchiában nincsen felettes eseménye.

Az események részleteinek megtekintéséhez itt is szükséges lehet az adott rendezvény azonosítójára, mely itt is UUID típusú. Itt a kliens megtudja az adott esemény rendezőjének az azonosítóját is, így már elérheti annak adatait, vagy eldöntheti, hogy önmaga-e a rendező, ezáltal engedélyezve további műveleteket. Ezek a műveletek a jelentkezés megnyitása, illetve lezárása, valamint a jelentkezők elfogadása.

A rendezvény alá hierarchikusan létre lehet hozni bármely, nem csak a rendező felhasználónak egy al-rendezvényt, mely funkcióiban teljesen megegyezik a fő rendezvénnyel, de a teljes listában nem fog megjelenni, ezt csak az al-rendezvények megtekintésénél tudjuk elérni.

A jelentkezést megkezdeni csak a megnyitott jelentkezési időszakban lehetséges, melyet a rendező tud megnyitni. Itt szintén csak az előbb említett időszak alatt van lehetőség a jelentkezés lemondására is.

5 Részletes megvalósítás

Ebben a fejezetben az olvasó az alkalmazás és a backend, azaz a teljes rendszer részletes megoldásaival ismerkedhet meg.

5.1 UML osztálydiagramok

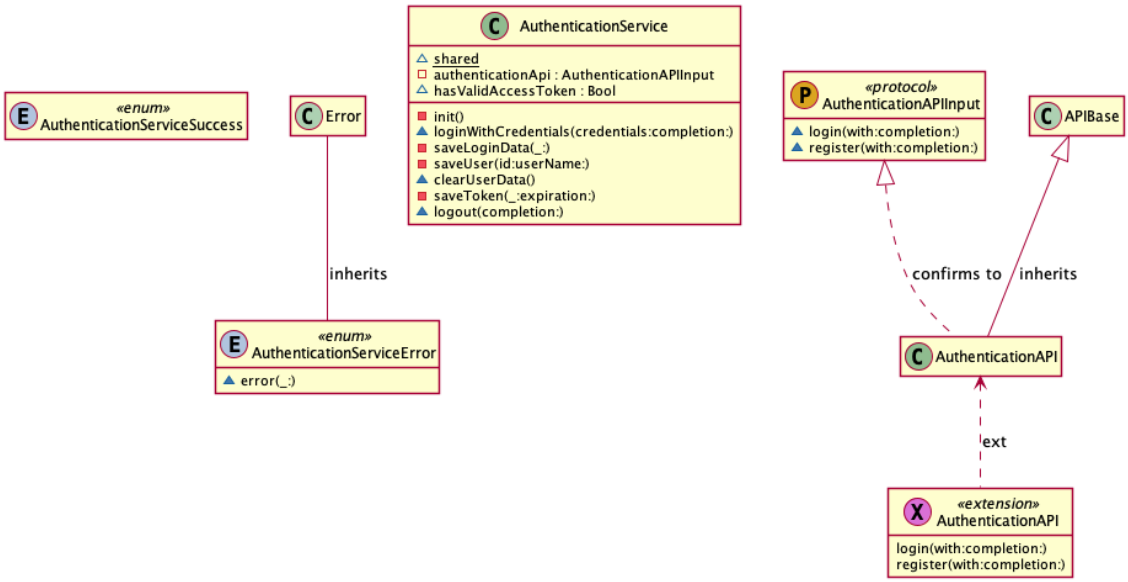
A dokumentum ezen részében a különböző komponensek egymással való kapcsolata lesz bemutatva UML osztálydiagramokon keresztül szemléltetve.

5.1.1 Autentikáció

Itt bemutatásra kerül az autentikációhoz kapcsolódó osztályok felépítése.

5.1.1.1 Frontend

Ebben a részben az iOS kliens felépítése olvasható.



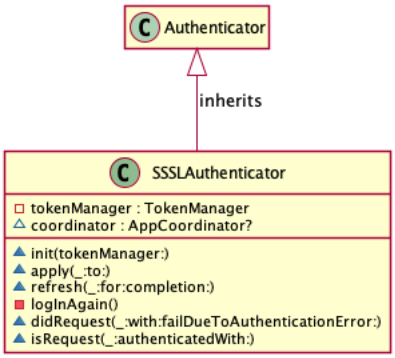
13. ábra: Autentikáció Api és service

A 13. ábra szemlélteti, hogy létezik egy APIBase, amely tartalmazza a minden API által megismételt feladatokat. Ebből származik le az AuthenticationAPI is, mely az autentikációs feladatokat oldja meg. Mint látható, az API kiterjesztésében (extension) vannak a protokolljában definiált követelmények megvalósítva, ezt a megoldást több helyen is felfedezhetjük a rendszerben. Az extension használatával való protokoll

megvalósítás egy bevett szokás a Swift alapú fejlesztésben, mivel így strukturálisan jobban átlátható lesz a kód.

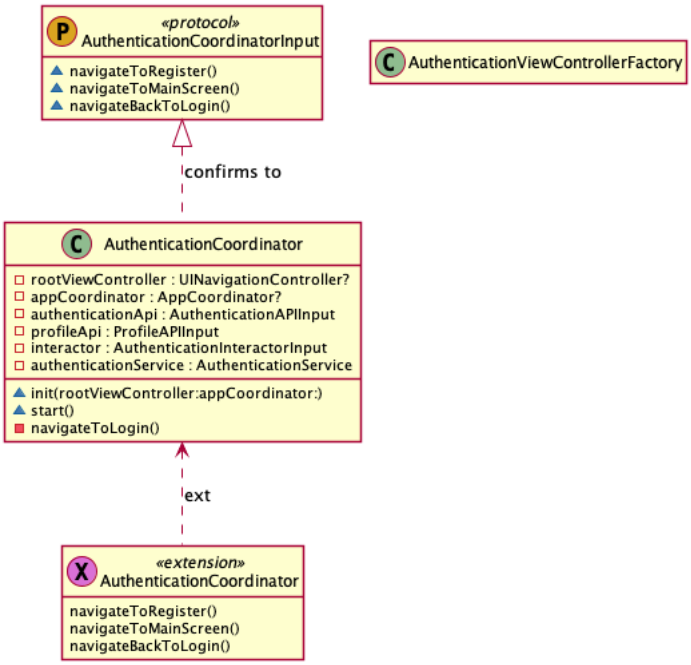
A képen látható Service az autentikációhoz köthető felhasználói adatok kezelését könnyíti meg, ilyen például a token elmentése a készüléken.

A Hiba és a Siker enumerációkat a Service használja fel, amikor completionként paraméternek kapott függvényekbe visszahív.



14. ábra: Az autentikátor

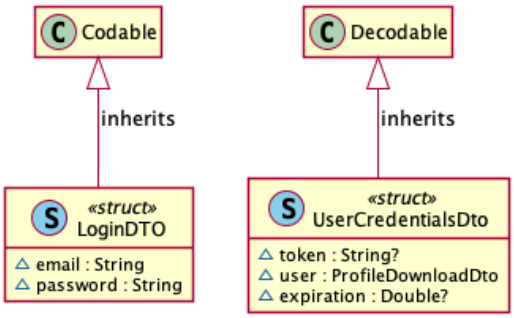
A 14. ábra által tartalmazott diagramban az Alamofire által használt Authenticator osztály leszármazott osztálya látható. Ez az osztályt az autentikációs folyamatok testre szabása végett lett létre hozva és ezeket a folyamatokat segíti.



15. ábra: Autentikáció Coordinator

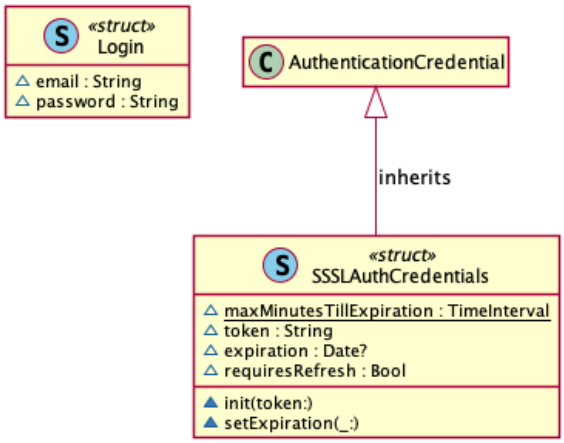
A 15. ábra tartalma alapján láthatjuk, hogy a Coordinator réteg az autentikációs komponensben miként épül fel. A ViewControllerFactory egy singleton osztály, melyet nem tárol el a Coordinator, csak a nézetek létrehozására használja.

A Coordinatornál is láthatjuk, hogy protokolljának megvalósítása egy kiterjesztésen keresztül történik.



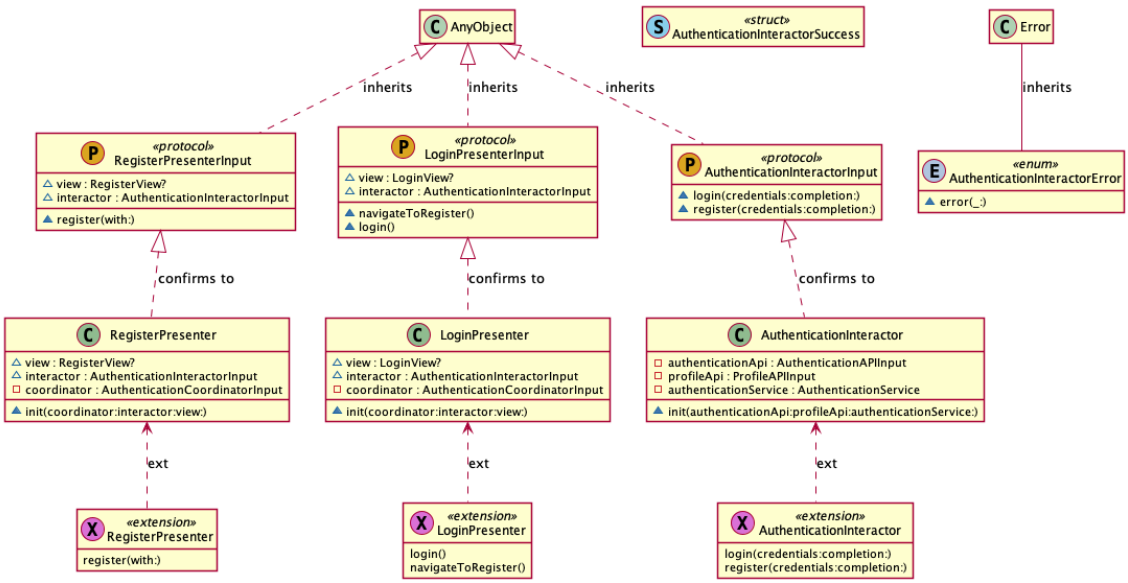
16. ábra: Autentikációs DTO

A 16. ábra láthatjuk, hogy a bejelentkezéshez használt DTO és a visszakapott válasz miként épül fel. Azért nem jelenik itt meg a regisztrációhoz használt DTO, mert az megegyezik a szerkesztéshez használt DTO-val, ezért ez a Profil kezelés komponensben található meg.



17. ábra: Autentikátor Modellek

A 17. ábra tartalmazza az autentikációért felelős komponens által használt modell objektumokat, a Login értelem szerűen a bejelentkezési adatokhoz szükséges, azonban az AuthCredentials struktúra az Alamofire által megkövetelt AuthenticationCredentials struktúrát egészíti ki.

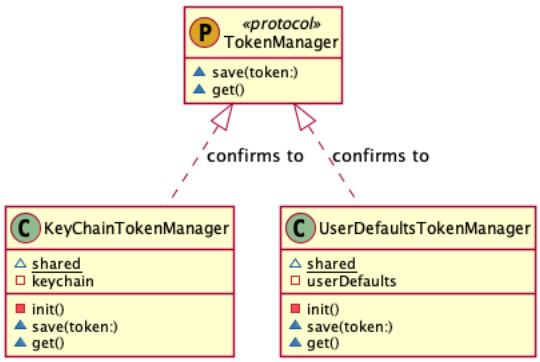


18. ábra: Autentikáció Presenter és Interactor

A 18. ábra tartalmazza a komponens Interactorának felépítését, amely láthatóan az AnyObjecttől leszármazott protokolljának megvalósítását kiterjesztésben implementálja.

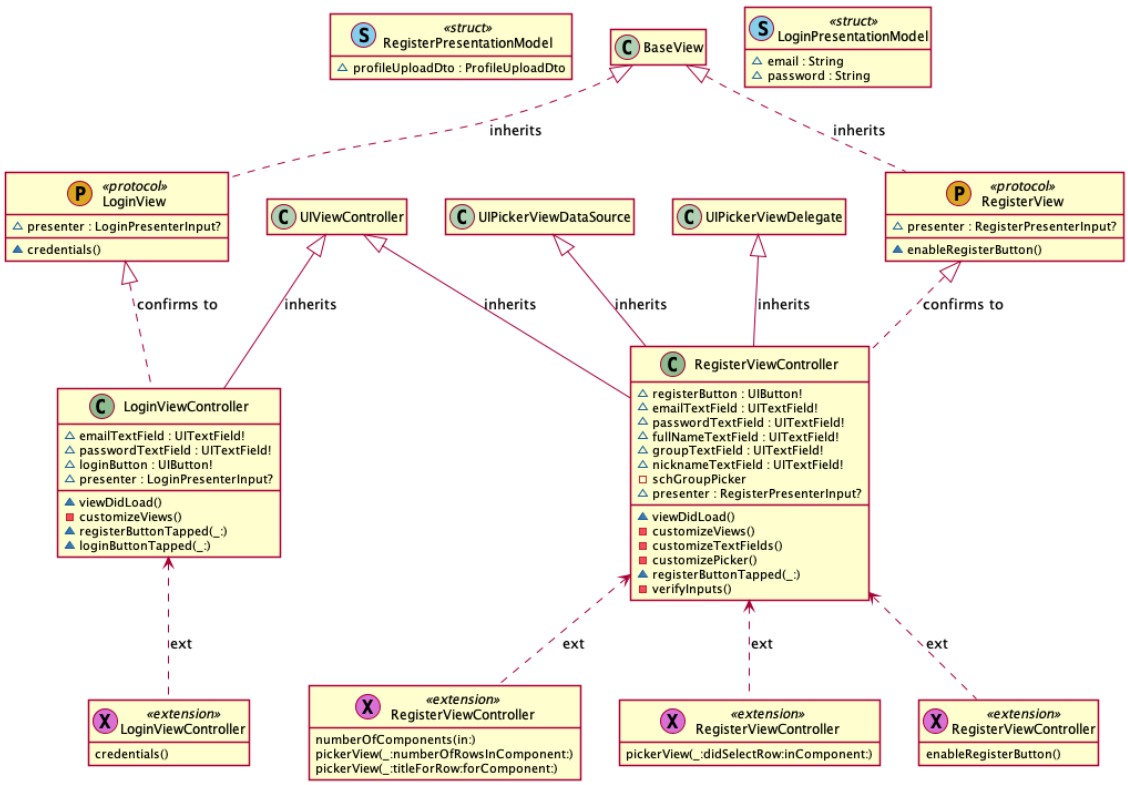
Az Interactorhoz tartozik továbbá egy Error és egy Success típus is, ez szintén a műveletek visszatérésének eredményénél használatos.

A Presenterek közül kettőt láthatunk, ez a kettő tartozik az autentikációs komponenshez. Ez a kettő a Login és Register, amelyek a nevükben található funkciókért felelősek.



19. ábra: Autentikációs token kezelők

A 19. ábra található TokenManager osztályok a token tárolásáért felelősek, ennek két módja is van. Vagy a keychainben tárolja biztonságosan vagy a UserDefaultsban, ami egy szimpla titkosítás mentes tároló. A tarolás módját az alkalmazási körülmények határozzák meg.



20. ábra: Autentikációs megjelenítők

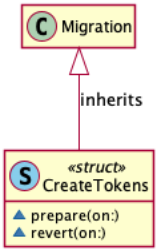
A 20. ábra tartalmazza az összes megjelenítéshez köthető osztály felépítését.

Láthatjuk, hogy a PresentationModelek is megjelennek itt.

A View protokollok láthatóan egy BaseView-ból származnak, mely definiálja az alapvető funkciókat. A ViewControllerek a 4.1.2.1 fejezetben említett UIViewControlleren kívül leszármaznak több UIKitből származó elemből is, amelyek az adatok különböző megjelenítési formájáért felelősek és ezekkel való interakciót segítik elő.

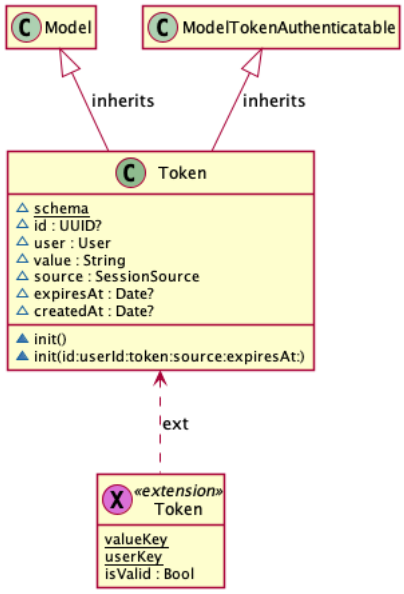
5.1.1.2 Backend

A háttérrendszer kevesebb komponensből áll, mint a kliens, de ezek is említésre méltók.



21. ábra: Autentikációs adatbázis migrációk

A 21. ábra tartalmazza a tokenek adatbázisban való tárolásához használt migrációkat, ezek a Fluent által biztosított Migration osztály leszármazottjai és szükségesek az adatbázis sémájának létrehozásához.



22. ábra: Autentikációhoz használt modellek

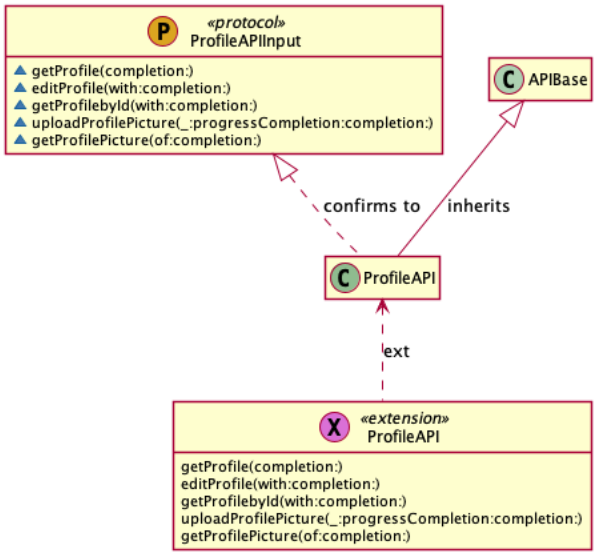
A 22. ábra tartalmán látjuk, hogy az autentikációhoz, ha az tokent használ, a Token osztály adja a keretrendszert. Ez az osztály a Fluent által biztosított Model osztályból származik, mely az adatok tárolását teszi lehetővé, valamint a ModelTokenAuthenticatable osztályból, melyet a Vapor tesz elérhetővé és a tokenes autentikációt könnyíti meg. A felhasználókhoz használt autentikáció itt azért nem jelenik meg, mert ez a felhasználókezeléshez szorosabban kapcsolódik a backend struktúrájában.

5.1.2 Profil kezelés

Itt bemutatásra kerülnek a profillal és annak kezelésével kapcsolatos osztályok.

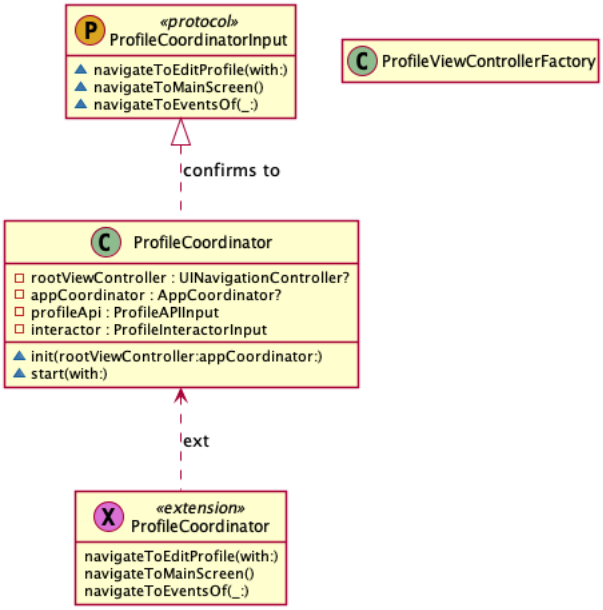
5.1.2.1 Frontend

Itt olvashatunk az iOS kliens profilkezeléséről.



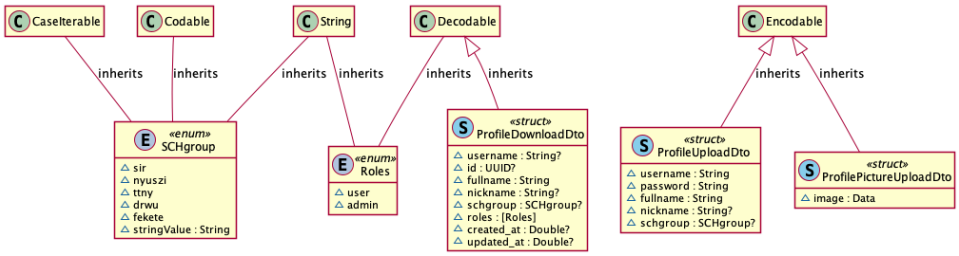
23. ábra: Profil API

A 23. ábra szerint az API hasonlóan az autentikációhoz, az APIBase osztályból származik le és kiterjesztésen implementálja protokollja követelményeit.



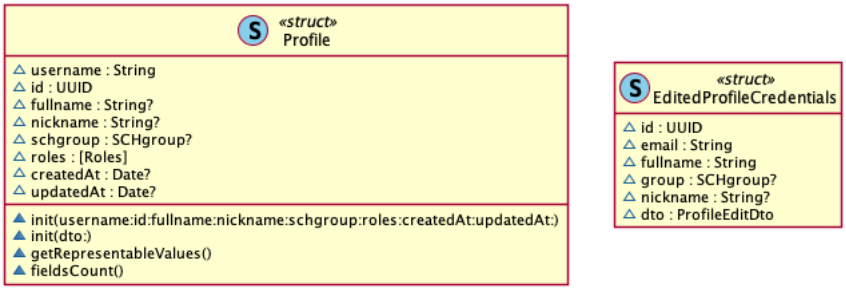
24. ábra: Profil Coordinator

A 24. ábra tartalmazza a Profil kezelés komponenshez tartozó Coordinator réteget.



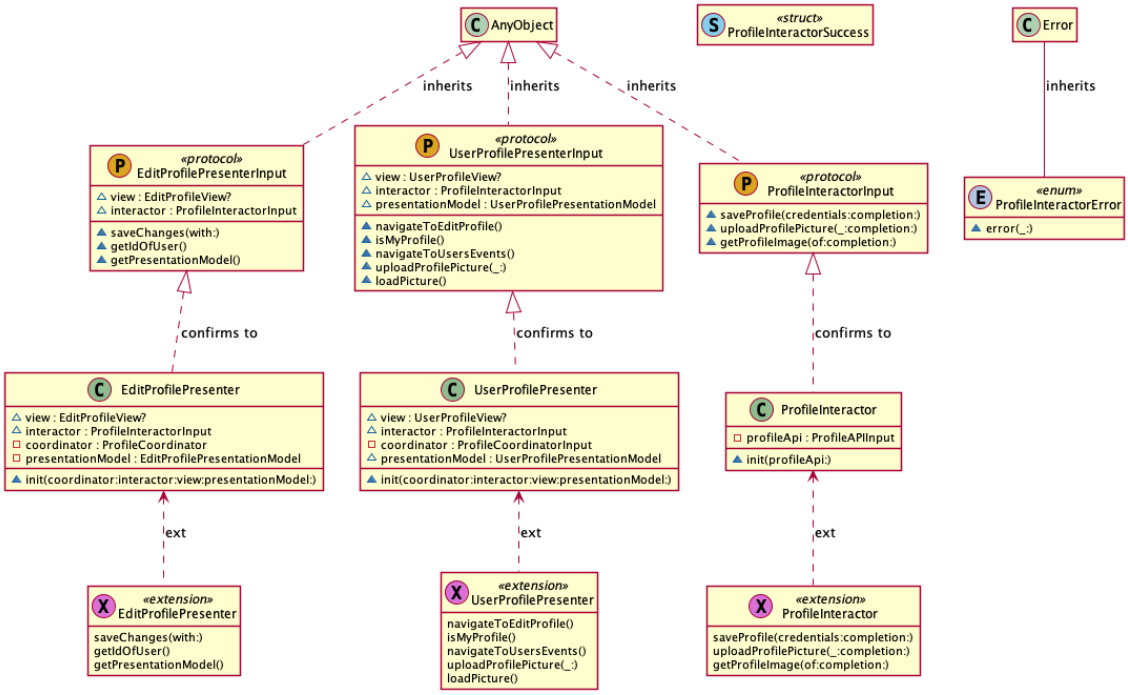
25. ábra: Profil DTO

A 25. ábra a Profillal kapcsolatos hálózati kommunikáció során használt komponenseket tartalmazza, illetve az alkalmazás egészében használt SCHgroup modellt, mely a csoportok azonosítását segíti. Itt láthatjuk a ProfileUploadDto-t, ami a regisztrációnál is használva van.



26. ábra: Profil Modellek

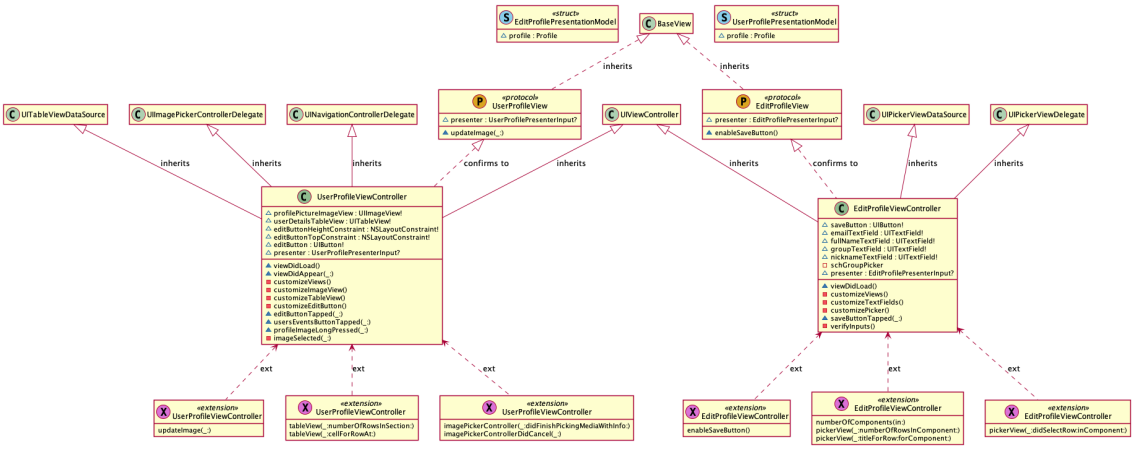
A 26. ábra a Profil komponens modelljeit láthatjuk. Az itt látható EditedProfileCredentials a profil szerkesztésénél kerül felhasználásra.



27. ábra: Profil Presenter és Interactor

A 27. ábra láthatjuk, hogy a Profil kezeléshez kettő presenter tartozik, ezek a profil szerkesztése és a profil megtekintése.

Az Interactor a tipikus interactor felépítést követi és a saját hiba és siker típusával rendelkezik.

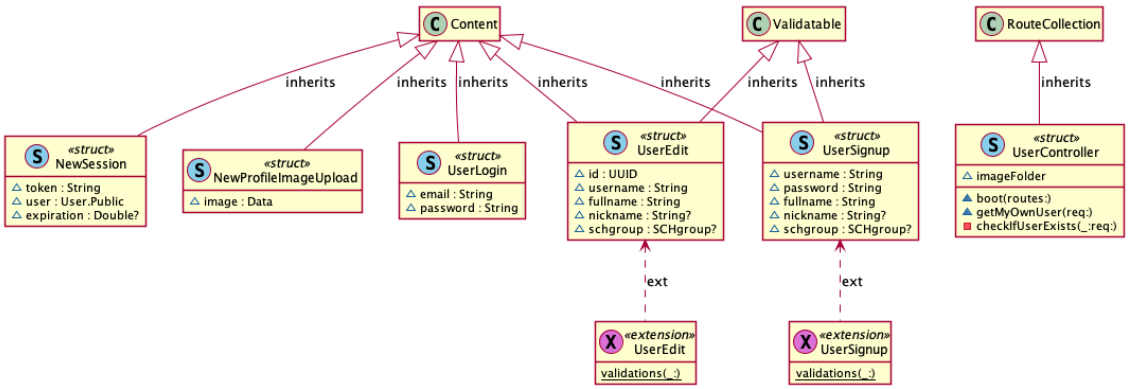


28. ábra: Profil megjelenítők

A 28. ábra az előbb felvázolt kettő presenterhez hasonlóan a kettő funkciónak megfelelő ViewController szokásos felépítéssel látható.

5.1.2.2 Backend

A backend profilkezelése sokszor összefonódik az autentikációval, ezért mindkét komponenshez tartozó részek itt lesznek ismertetve.

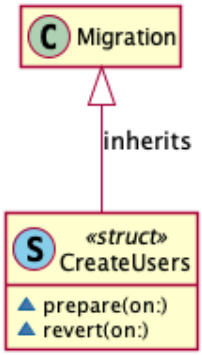


29. ábra: Profil üzleti logika

A 29. ábra tartalmazza a Profillal kapcsolatos üzleti logikáért felelős struktúrákat.

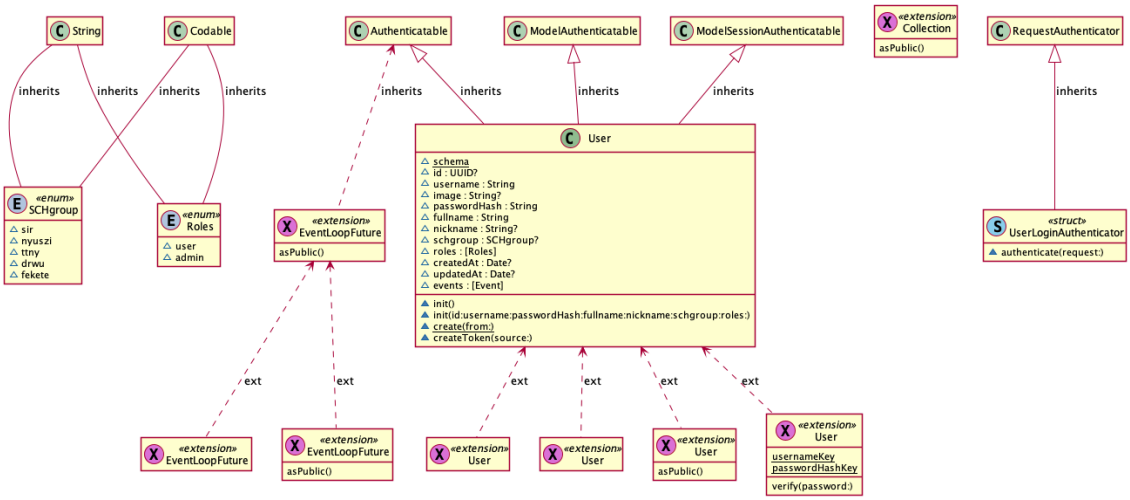
Itt a Vapor által biztosított Content, sorosításra használt osztályból leszármazó struktúrák a különböző kérések bejövő és kimenő tartalmát reprezentálják, míg a szintén a keretrendszerben megtalálható Validatable leszármazottjai képesek a bejövő adatok megadott rendszer szerinti validálását, ellenőrzését.

A környezet RouteCollectoin osztályából származó UserController, a különböző elérési utak (URL-ek) logikáját valósítja meg, és kezeli az adatbázisban történő változásokat.



30. ábra: Profil adatbázis migráció

A 30. ábra a profilhoz tartozó adatbázist előkészítő struktúrát tartalmazza.



31. ábra: Profilhoz tartozó modellek

A 31. ábra bemutatja a profilhoz tartozó modelleket.

Itt látható a két enumerációnk szövegként való kezelését biztosító String ősosztály és a sorosításért felelős Codable osztály. A két enumeráció a profil típusát és a csoportba való tartozását reprezentálja.

A User osztály a különböző Authenticatable osztályokból leszármazva biztosítja a többféle autentikáció lehetőségét.

Az extensionök a beépített és Vapor által biztosított adattárolók kiegészítését tartalmazzák, mely megkönnyíti a kintről is megtekinthető adatok kiszűrését.

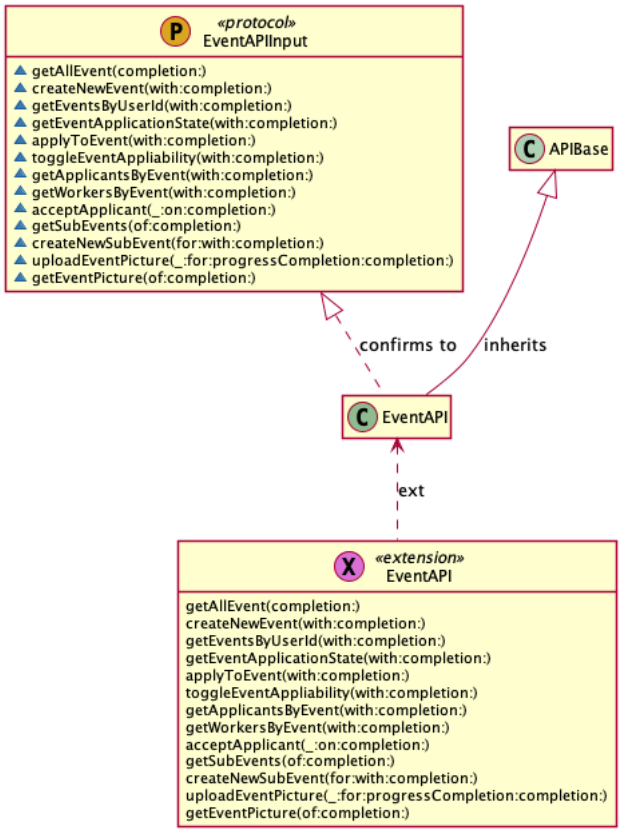
Továbbá itt láthatjuk az autentikációhoz is kapcsolódó UserLoginAuthenticatort, mely a bejelentkezést segíti elő.

5.1.3 Rendezvény kezelés

Ebben a részében a dokumentumnak, a rendezvények/események kezeléséért felelős objektumok kerülnek bemutatásra.

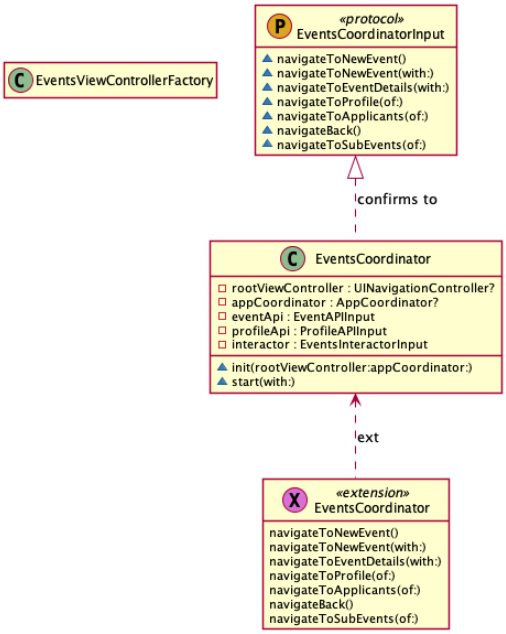
5.1.3.1 Frontend

Itt olvasható az iOS kliens események kezeléséért felelős részek felépítése.



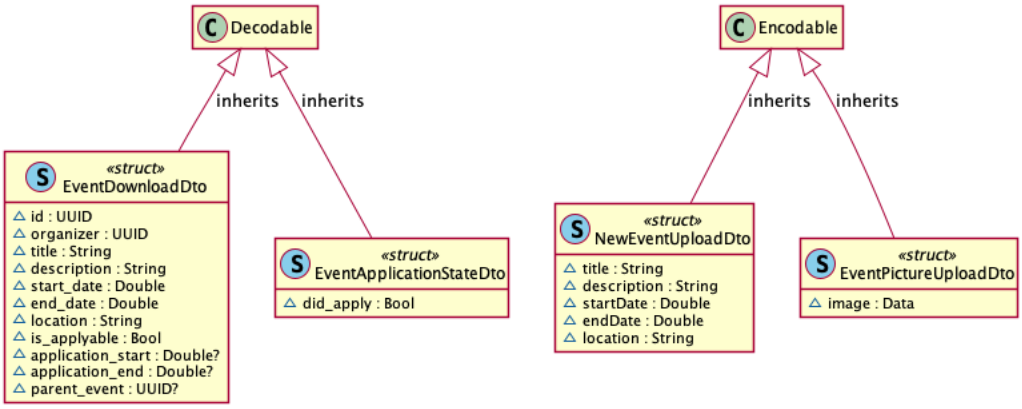
32. ábra: Esemény API

A 32. ábra tartalmazza az események kezeléséhez használt hálózati kommunikációs osztály felépítését a már ismertetett módon.



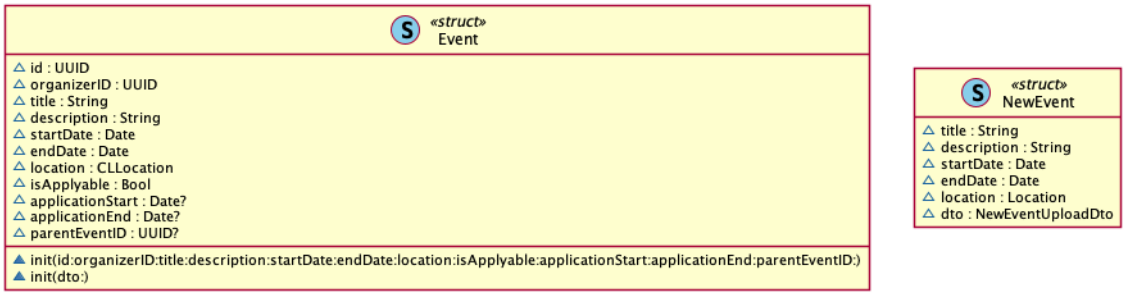
33. ábra: Esemény Coordinator

A 33. ábra látható az eseményekhez tartozó Coordinator réteg felépítése.



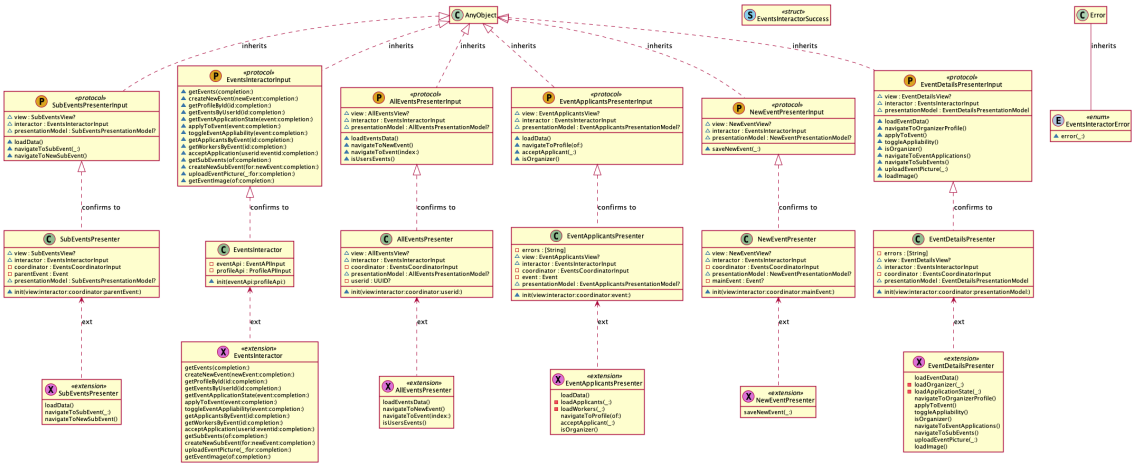
34. ábra: Esemény DTO

A 34. ábra tartalma az események kezeléséhez használatos hálózati hívások tartalmának becsomagoló struktúrái.



35. ábra: Események modelljei

A 35. ábra látható modellek a rendezvényt, illetve a rendezvény létrehozásához szükséges adatoknak adnak keretet.

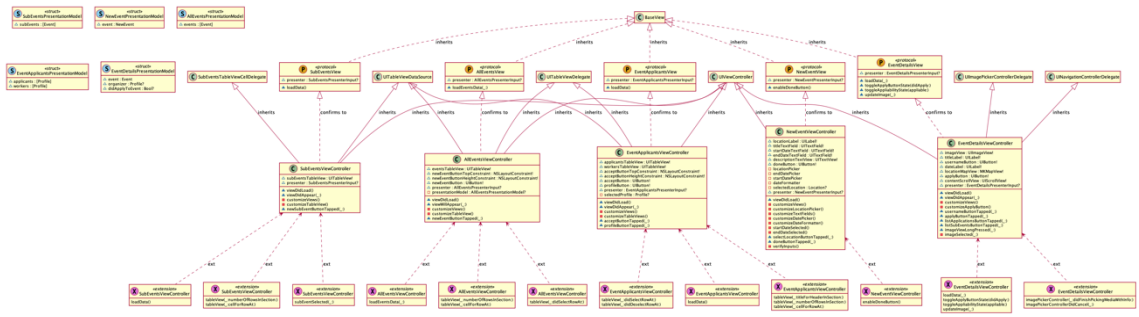


36. ábra: Esemény Presenter és Interactor

A 36. ábra bemutatja a már kifejtett architektúra szerinti felépítését a különböző funkcióknak.

Ezek sorrendben: al-rendezvény létrehozása, összes főrendezvény listája, jelentkezettek listája, új rendezvény létrehozása és a rendezvény adatainak megtekintése.

Megtalálható továbbá az ezen komponenshez tartozó Interactor is és a hozzá tartozó sikerességet és hibát reprezentáló objektumok.



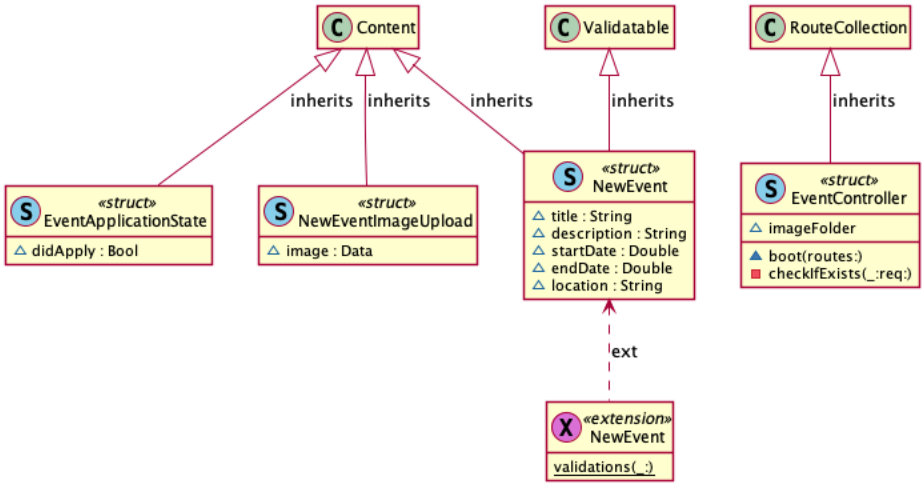
37. ábra: Események megjelenítői

A 37. ábra a rendezvények kezeléséhez tartozó ViewControllerek és protokolljaik láthatók. Ezeknek száma megegyezik a funkciók és a Presenterek számával.

Ezeken felül még a PresentationModelek olvashatók le a diagramról.

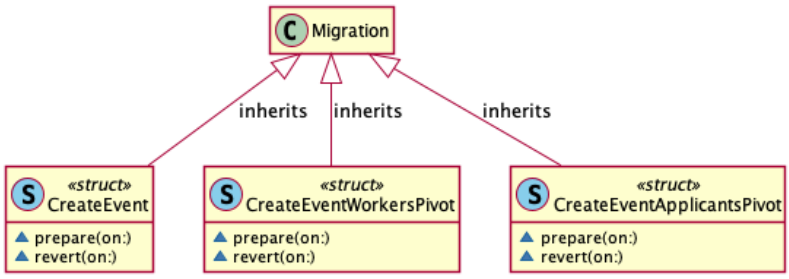
5.1.3.2 Backend

A Vapor alapú rendszer itt kissé összefonódik a Profil kezeléssel, ezért a közös részek itt kerülnek kifejtésre.



38. ábra: Eseménykezelés üzleti logikája

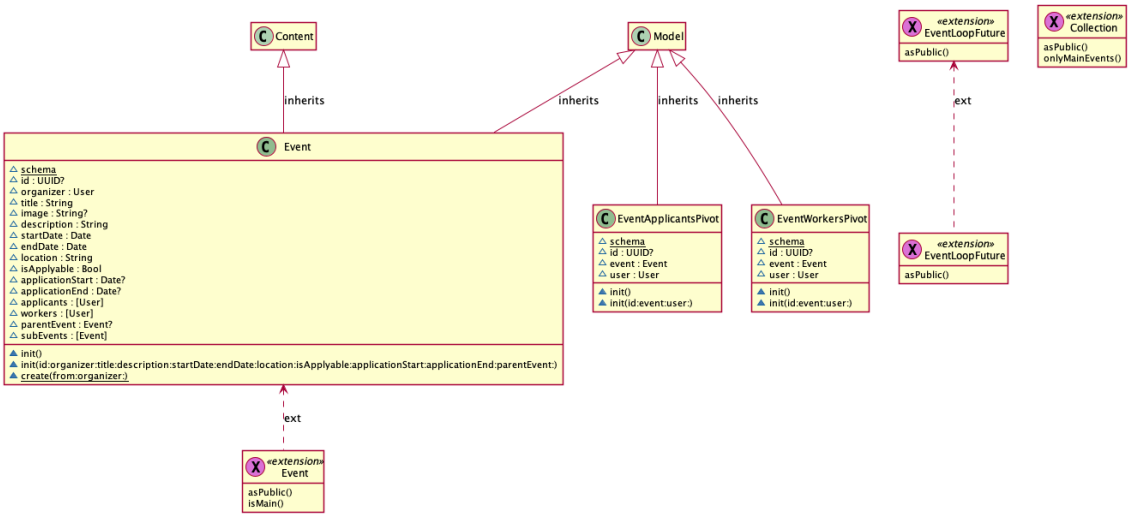
A 38. ábra tartalmazza a különböző eseménnyel kapcsolatos hálózati kommunikáció becsomagolásáért felelős, már megszokottan felépített struktúrákat.



39. ábra: Események migrációi

A 39. ábra tartalma a rendezvényekhez kapcsolódó, adatbázis előkészítésére szolgáló migrációkat.

Itt a CreateEvent, egyértelműen az eseményekért felel, de az CreateEventWorkersPivot és CreateEventApplicantPivot a jelentkezők, valamint a már elfogadott jelentkezőkhöz tartozó kapcsolótáblát készíti elő.

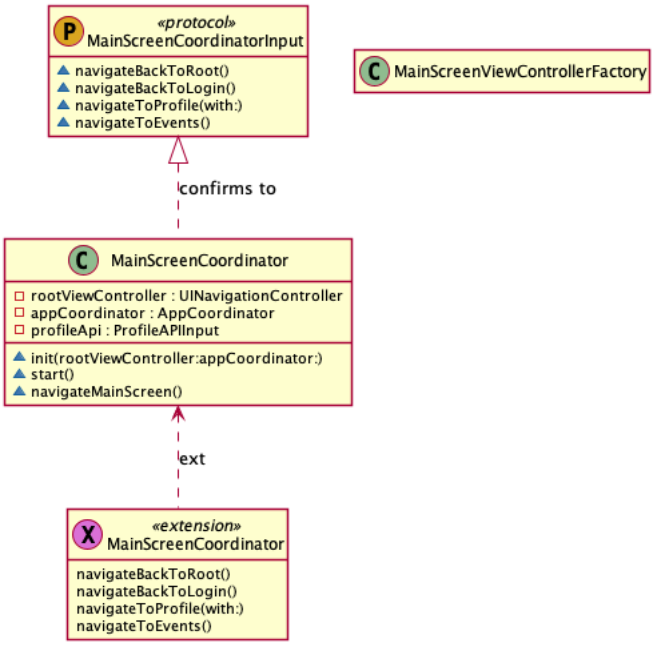


40. ábra: Eseményhez tartozó modellek

A 40. ábra bal oldalt látható a komponens, rendezvényekhez kapcsolódó adatok struktúrájának meghatározásáért, valamint azok tárolásáért, míg közép tájt a kapcsolatok felépítéséért felelős modellje.

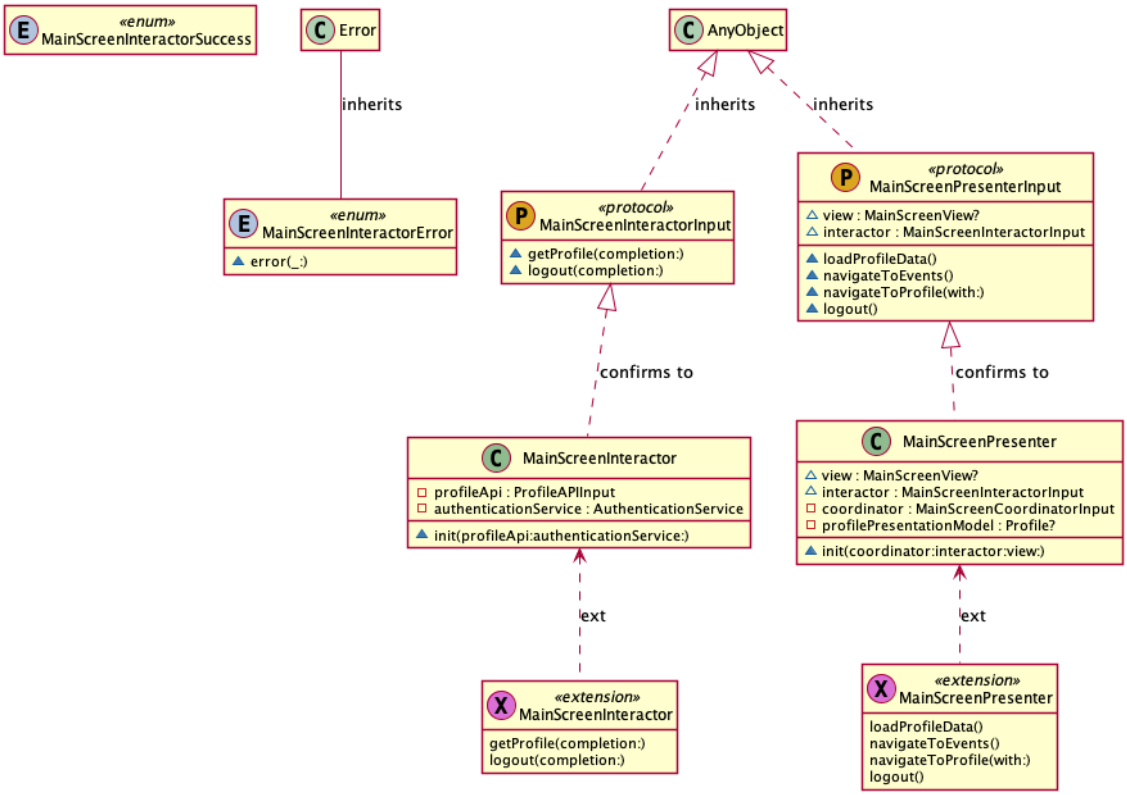
5.1.4 Navigációs képernyő

Az alkalmazáson belüli funkcióválasztást elősegítendő, készült egy navigációs komponens is MainScreen néven.



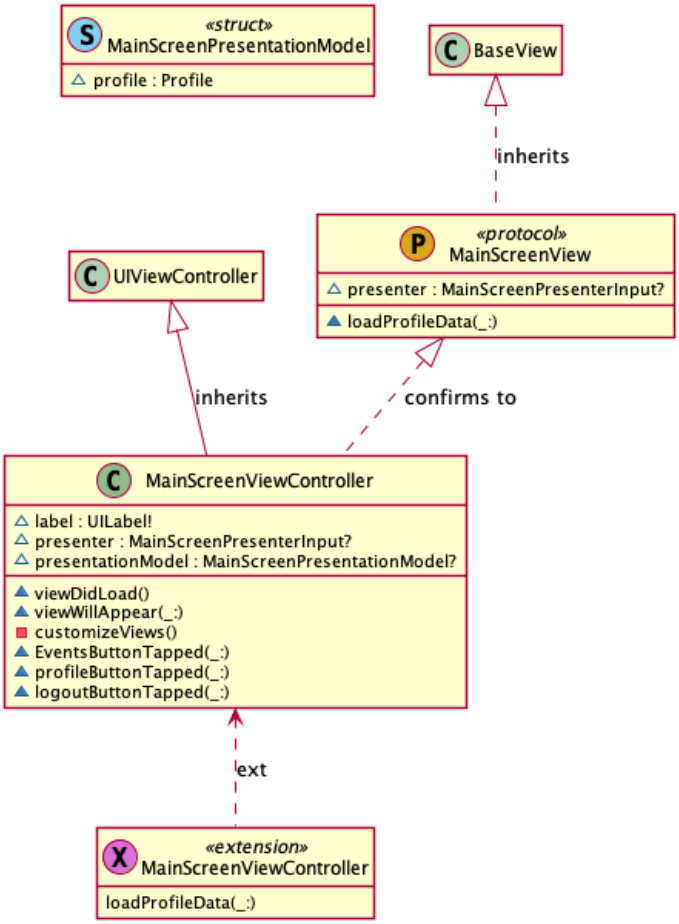
41. ábra: MainScreen Coordinator

A 41. ábra által prezentált Coordinator teszi lehetővé a funkciók indítását.



42. ábra: MainScreen Presenter és Interactor

A 42. ábra látható a komponenshez tartozó Interactor és Presenter. Az Interactor itt a profiladatok lekérésére képes, egy üdvözlőüzenet előállításához.

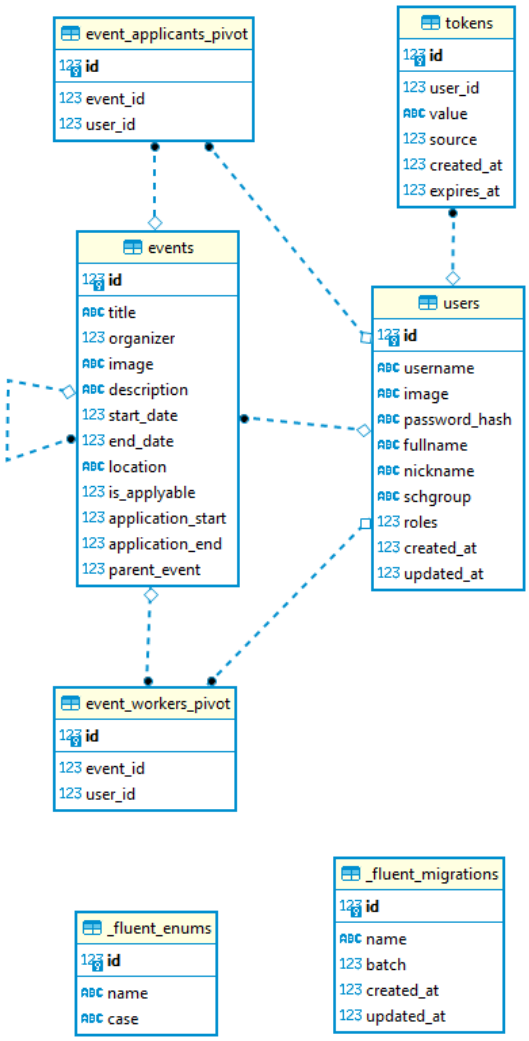


43. ábra: MainScreen nézet

A 43. ábra tartalma a navigációs nézetért felelős rész, a már ismertetettel megegyező felépítése.

5.2 Enity-relation diagram

Ebben a részben megismerésre kerül az adatbázis sémája, melyben a rendszer az adatokat tárolja.



44. ábra: Az adatbázis ER modellje

A 44. ábra látható _fluent... kezdetű táblákat a Fluent keretrendszer használja a már futtatott migrációk és a rendszerben definiált enumerációk nyilvántartására.

A felhasználók és tokenek között több-egy kapcsolat van, mert több platformról is hozzáférhet a felhasználó a rendszerhez, ehhez különböző tokeneket használva.

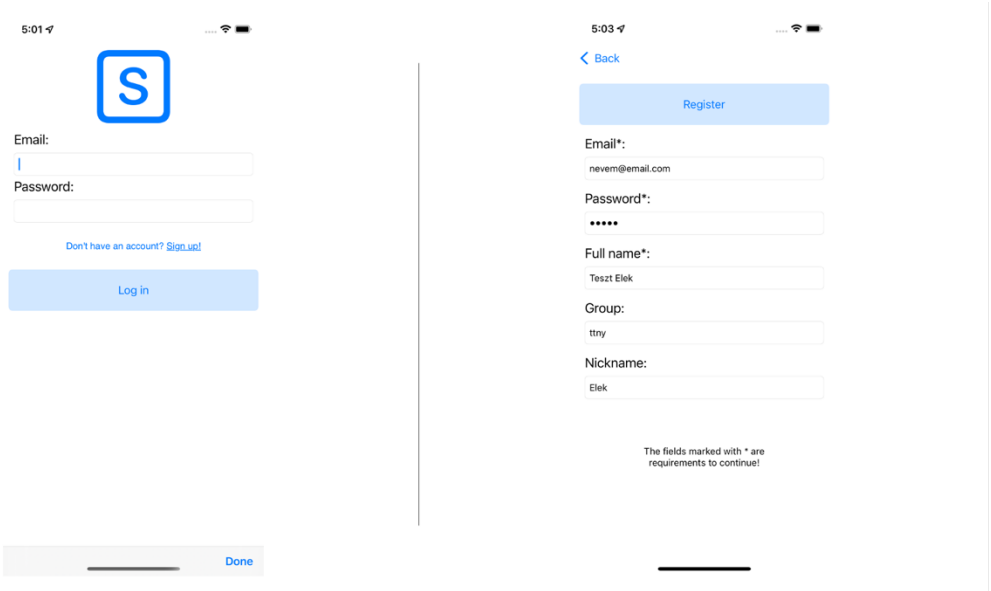
A felhasználók és események között is több-egy kapcsolat van, mert egy eseménynek csak egy főrendezője lehet, viszont egy felhasználó akárhány eseményt fő rendezhet.

Ez a kapcsolat az események és al-események között is fennáll, mivel fő esemény csak egy lehet, de al-esemény akármennyi.

A további két kapcsolótábla (..._pivot) a több-több kapcsolat elérésére hivatott. Ezek a jelentkezők és az elfogadott dolgozók kapcsolótáblái, melyek az esemény és felhasználó táblákat kötik össze.

6 Felhasználói leírás

Ez a fejezet néhány munkafolyamaton keresztül bemutatja az alkalmazás használatát.

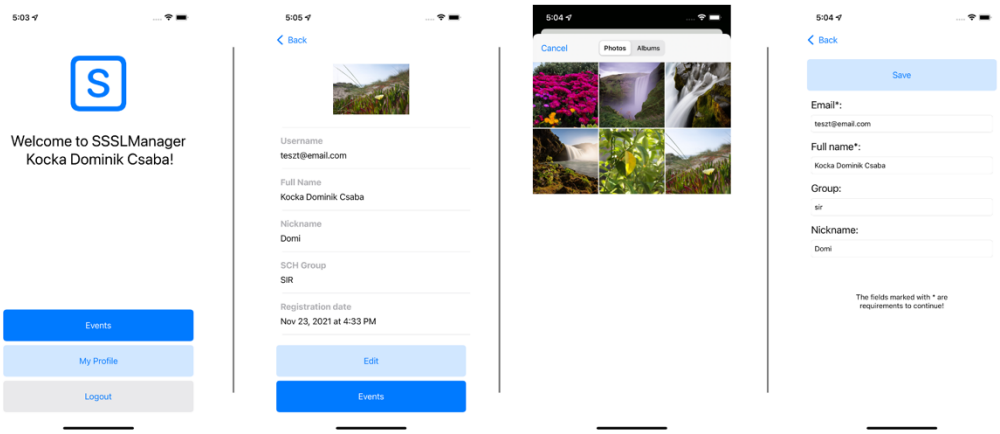


45. ábra: Regisztráció folyamata

A 45. ábra bemutatja a regisztráció folyamatát.

A felhasználónak a bejelentkezés képernyőn van lehetősége a „Sign up!” gombra nyomnia, ezáltal megjelenítve a regisztrációhoz szükséges űrlapot.

Itt az kötelező adatok kitöltése után lehetősége van a regisztrációra a „Register” gomb megnyomásával.



46. ábra: Profil szerkesztésének folyamata

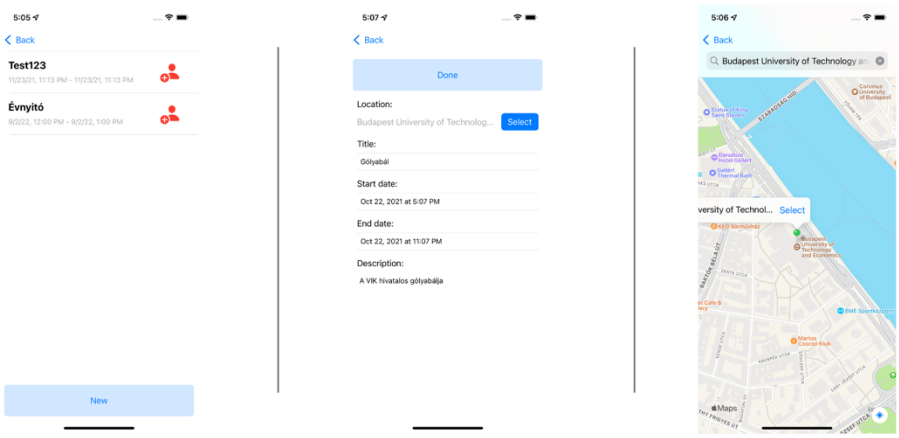
A 46. ábra a profil szerkesztésének folyamata látható.

Ezt a folyamatot bejelentkezés után a navigációs képernyőn a „My Profile” gombra kattintva kezdhetjük meg.

Megjelenik a saját profilunk, ahol a képünket nyomva tartva feljön egy képválasztó menü, ahol az új képünket választhatjuk ki az eszköz fotótárából.

Az profilunk adatainak képernyőjén az „Edit” gombra kattintva megjelenik a szerkesztő nézet. Itt tudjuk adatainkat módosítani, ennek végeztével a „Save” gombra kell nyomnunk a változtatások mentéséhez.

Ha a profilon az „Events” gombot nyomjuk meg, visszakapunk egy listát a csak a profil által rendezett eseményekről. (Ezek között már lehet al-esemény is)



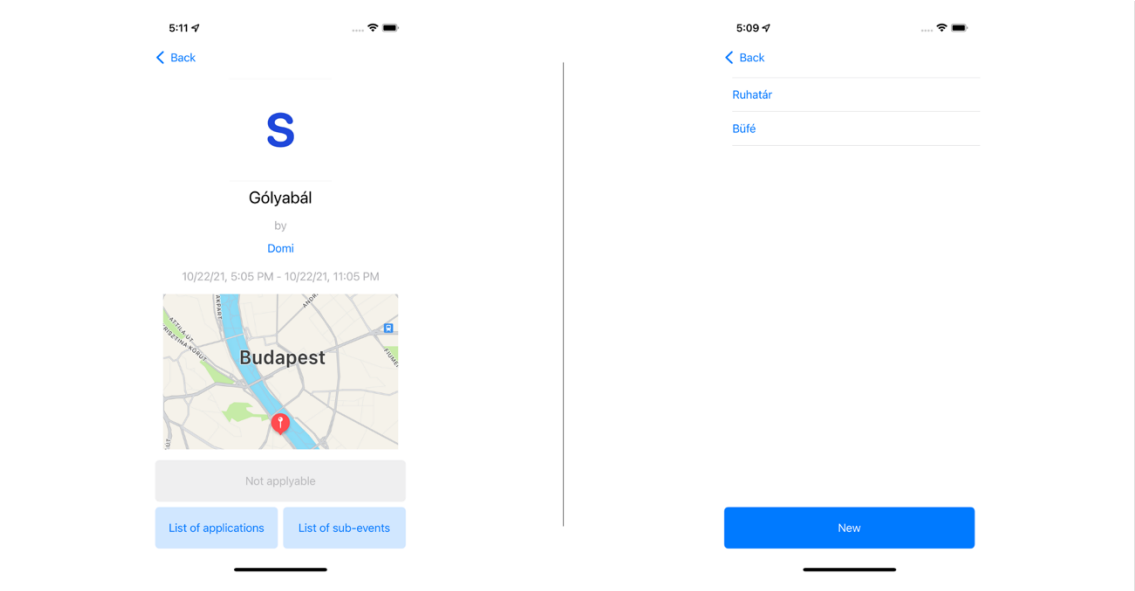
47. ábra: Új esemény felvitelének folyamata

A 47. ábra bemutatja az új esemény rendszerbe való felvitelének folyamatát.

Első lépés az összes esemény nézet megnyitása után, hogy megnyomjuk a „New” gombot, ennek hatására megjelenik az adatok megadására kérő nézet.

Itt a „Select” gombra nyomva megjelenik egy helyszín kereső, ahol keresőmező, valamint térkép segítségével tudunk helyszínt választani.

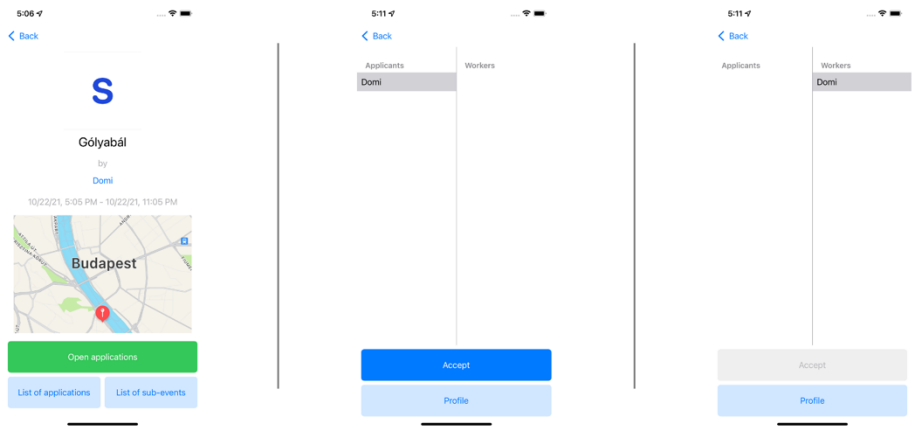
Ha a leíráson kívüli összes mezőt kitöltöttük, lehetőségünk van a „Done” gomb megnyomásával létrehozni az eseményt a rendszerben. (Ekkor megjelenik az új esemény részletes nézete.)



48. ábra: Esemény részletei és al események folyamata

A 48. ábra tartalmán bal oldalt látjuk az események részletes nézetét egy meg nem nyitott esemény esetében (ha meg lenne nyitva, a jelenleg szürke gomb zöldre/pirosra váltana, attól függően, hogy jelentkeztünk-e már), melynek nem a felhasználó a fő rendezője.

A jobb oldalon található a lista, melyben az al-események és részletes nézetükre történő navigáció érhető el.



49. ábra: Jelentkezés elfogadásának folyamata

A 49. ábra reprezentálja az eseményre való jelentkezés elfogadásának folyamatát.

A folyamat a részletes nézeten kezdődik, ahol rendezőként látható a zöld gomb, mellyel megnyithatjuk a jelentkezést, illetve, ha már meg lenne nyitva, egy piros gomb jelenne meg, mellyel lezárhatnánk azt.

Az utolsó két képernyőn láthatjuk, hogy kiválasztást, majd „Accept” gombot nyomva áttehető a jelentkezők oszlopából a dolgozókéba egy jelentkezett felhasználó. Ugyanezen képernyőkön a kiválasztást követően a „Profile” gomb megnyomása előhívja a jelentkező profilját. (A második két kép egy másik felhasználóval bejelentkezve egy másik esemény alatt készült)

7 Összefoglalás, továbbfejlesztési lehetőségek

A rendszer tehát kiküszöböli a hasonló szoftverekben meg nem található funkciókat és egy egyszerű, de hatékony megoldást biztosít az eseményrendezés első pár lépésének megkönnyítésére. A saját profilrendszernek köszönhetően a személyes adatokat harmadik személy nem láthatja, csak a felhasználók és a rendszer.

A rendszer jelenlegi állapotát tovább lehet fejleszteni a kliensprogramok felhozatalának kibővítésével, akár egy Android operációs rendszeren futtatható applikáció segítségével vagy egy weblap segítségével, ami akár mobil telefonokra is optimalizáltan, letöltés nélkül tudhatja megjeleníteni a rendszer által kínált tartalmakat.

Funkciók terén is kibővíthető a rendszer, akár az adatbázisban már létező jelentkezési időszakok lehetőségének kiépítésével, akár az elfelejtett jelszavak automatikus kezelésével, de még a UI design átdolgozásával is.

Megjegyzések

- A diagramok készítésére https://app.diagrams.net volt segítségemre
- A generált UML diagramok a PlantUML (https://plantuml.com) és a SwiftPlantUML (https://github.com/MarcoEidinger/SwiftPlantUML) segítségével készültek.
- Az adatbázis ER diagram a DBeaver (https://dbeaver.io) segítségével készült.
- A frontend (iOS) alkalmazás forráskódja elérhető a következő linken: https://github.com/domkoc/SSSLManager_iOS
- A backend (Vapor) háttérrendszer forráskódja elérhető a következő linken: https://github.com/domkoc/SSSLManager_backend
- A backend fejlesztése során továbbá felhasználásra került: [17]

8 Irodalomjegyzék

[1] K. Nagamine, "Supply Chain Constraints Finally Catch Up to the Global Smartphone Market, Contributing to a 6.7% Decline in Third Quarter Shipments, According to IDC," 28 Október 2021. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=prUS48342021. [Accessed 25 November 2021].

[2] Apple Inc., „About Swift," Apple Inc., 2021. [Online]. Available: https://www.swift.org/about/. [Hozzáférés dátuma: 26 11 2021].

[3] Apple Inc., „UIKit," Apple Inc., 2021. [Online]. Available: https://developer.apple.com/documentation/uikit. [Hozzáférés dátuma: 26 11 2021].

[4] L. W. a. t. h. o. m. o. t. V. c. Tanner Nelson, „Vapor Docs Welcome," Vapor Community, 2021. [Online]. Available: https://docs.vapor.codes/4.0/. [Hozzáférés dátuma: 26 11 2021].

[5] Vapor Community, „vapor / vapor," Vapor Community, 2021. [Online]. Available: https://github.com/vapor/vapor. [Hozzáférés dátuma: 26 11 2021].

[6] Qutheory, LLC, „Vapor - The future of web development.," Qutheory, LLC, 2020. [Online]. Available: https://vapor.codes. [Hozzáférés dátuma: 26 11 2021].

[7] L. W. a. t. h. o. m. o. t. V. c. Tanner Nelson, „Vapor Docs - Fluent," Vapor community, 2021. [Online]. Available: https://docs.vapor.codes/4.0/fluent/overview/. [Hozzáférés dátuma: 26 11 2021].

[8] R. Kliffer, „Alamofire Tutorial: Getting Started," 25 04 2018. [Online]. Available: https://www.raywenderlich.com/35-alamofire-tutorial-getting-started. [Hozzáférés dátuma: 26 11 2021].

[9] D. Jennes, „Github / SwiftGen," SwiftGen, 08 2021. [Online]. Available: https://github.com/SwiftGen/SwiftGen. [Hozzáférés dátuma: 26 11 2021].

[10] K. Katsumi, „Github / KeychainAccess,” 11 2021. [Online]. Available: https://github.com/kishikawakatsumi/KeychainAccess. [Hozzáférés dátuma: 26 11 2021].

[11] I. Qurashi, „Github / IQKeyboardManager,” 2019. [Online]. Available: https://github.com/hackiftekhar/IQKeyboardManager. [Hozzáférés dátuma: 26 11 2021].

[12] A. Sapargali, „Github / LocationPicker,” 2020. [Online]. Available: https://github.com/almassapargali/LocationPicker. [Hozzáférés dátuma: 26 11 2021].

[13] Meta, „Facebook,” Meta, [Online]. Available: https://facebook.com. [Hozzáférés dátuma: 26 11 2021].

[14] Apple Inc., „Apple calendar,” Apple Inc., [Online]. Available: https://www.icloud.com/calendar/. [Hozzáférés dátuma: 26 11 2021].

[15] Google LLC, „Google Calendar,” Google LLC, [Online]. Available: https://calendar.google.com/. [Hozzáférés dátuma: 26 11 2021].

[16] M. Katz, „Getting Started with the VIPER Architecture Pattern,” 20 04 2020. [Online]. Available: https://www.raywenderlich.com/8440907-getting-started-with-the-viper-architecture-pattern. [Hozzáférés dátuma: 26 11 2021].

[17] T. C. T. N. Logan Wright, Server-Side Swift with Vapor: third edition, Razeware LLC, 2021.