|  |  |  |
| --- | --- | --- |
|  | **Eötvös Loránd Tudományegyetem**  Informatikai Kar  Programozáselmélet és Szoftvertechnológiai Tanszék |  |

**Rajt reakcióidő mérő alkalmazás**

|  |  |  |
| --- | --- | --- |
| *Témavezető:*  Nagy Barnabás  PhD hallgató  Gazdaságinformatikus MSC | *Külső témavezető:*  Horváth Ádám  IT projektvezető  Mérnökinformatikus MSC | *Szerző:*  Kemenes Ákos  programtervező informatikus BSc |

Budapest, 2018

Téma bejelentő: a szakdolgozat bekötve kell, hogy tartalmazza a kitöltött és jóváhagyott (az Informatikai Kar dékánja által aláírt) Szakdolgozat-téma bejelentőt. **témabejelentőt**. Ennek a lapnak a helyére kell bekötni.

\*\*\*

Tartalomjegyzék

[1. Az alkalmazás célja 4](#_Toc512941269)

[1.1 Motiváció 4](#_Toc512941270)

[1.2 Az alkalmazás funkciói 4](#_Toc512941271)

[1.3 A szoftver célközönsége 4](#_Toc512941272)

[2. Felhasználói dokumentáció 5](#_Toc512941273)

[2.1 A szoftver használata 5](#_Toc512941274)

[2.1.1 Platformok 5](#_Toc512941275)

[2.2 A szoftver felépítése 5](#_Toc512941276)

[2.2.1 Kezdőképernyő 5](#_Toc512941277)

[2.2.2 Regisztráció 6](#_Toc512941278)

[2.2.3 Bejelentkezés 7](#_Toc512941279)

[2.2.4 Funckiók 8](#_Toc512941280)

[2.2.5 Edzői oldal 8](#_Toc512941281)

[2.2.6 Sportolói oldal 9](#_Toc512941282)

[3. Fejlesztői dokumentáció 11](#_Toc512941283)

[3.1 A program szerkezete 11](#_Toc512941284)

[3.2 MultiPeerConnectivity 13](#_Toc512941285)

[3.2.1 Kommunikációs keretrendszer darabjai 13](#_Toc512941286)

[3.2.2 Kommunikáció – Csatlakozás 13](#_Toc512941287)

[3.2.3 Kommunikáció – Csatlakozás elfogadása 14](#_Toc512941288)

[3.3Időszinkronizáció 15](#_Toc512941289)

# Az alkalmazás célja

## Motiváció

Gyermekkorom óta sportoló vagyok. Sokféle sportot és csapatot megjártam ez idő alatt, de végül az atlétika mellett tettem le a voksomat. Ezen belül is a sprintszámok valamint a gátfutás lett a számomra meghatározó. Minden edzésen amikor bármiféle rajt vagy ahhoz kapcsolódó gyakorlatot végeztünk, egy apró dolog hiányzott amit még senki nem eszközölt ki. Ez a dolog pedig a versenyhelyzethez hasonló indítás szimulálása. Ezen alkalmazás ezt a kicsi hiányosságot hivatott kipótolni, méghozzá egy igencsak egyszerű módon, szükséges célhardver nélkül, több okostelefon összekapcsolódásával és folyamatos szinkronizációjával.

## Az alkalmazás funkciói

Az alkalmazás indulása után dönthetünk, hogy edzői vagy sportolói módban jelentkezünk be. Amennyiben edzőként tesszük, akkor tudunk csatlakozni a sportolói készülékekhez és adhatjuk ki nekik a rajtparancsot majd a rajt után listázva megkapjuk az adott rajt reakció idejeit. Ha sportolóként jelentkezünk be akkor nincs más dolgunk mint megvárni még az edzőnk csatlakozást indít felénk, nekitámasztani készülékünk a rajtgépnek, majd a rajtparancsra elindulni. Sportolóként, hisztorikusan listázva megkaphatjuk az eddigi mért reakcióinkat. Mindezeken felül, az alkalmazás lehetőséget biztosít egy félautomatizált időmérésre is. Erre edzői módban van lehetőségünk. A „lövéssel” egy időpontban indul egy stopper, amelyet már kézzel kell majd megállítani, így kiküszöbölve a kézi mérés két emberi tényezőjéből az egyiket.

## A szoftver célközönsége

Az alkalmazás azon sportolóknak illetve edzőknek lehet hasznos, akik szeretnének a versenyhelyzetre a lehető legjobban felkészülni, és kihozni a legtöbbet a másodperc töredék része alatt is magukból.

# Felhasználói dokumentáció

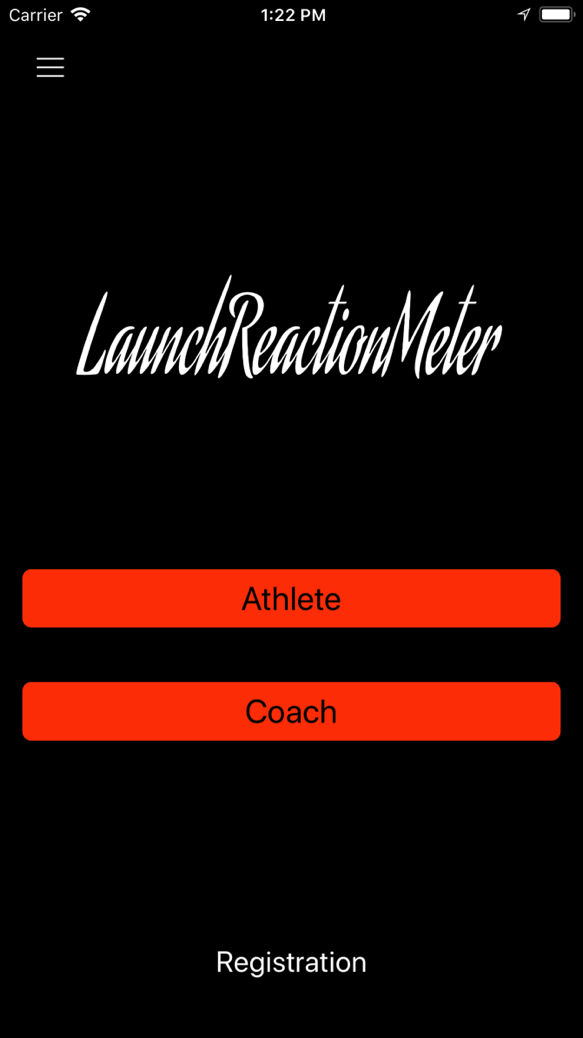
## 2.1 A szoftver használata

### 2.1.1 Platformok

Az alkalmazás iOS kompatibilitással rendelkezik. Ezen platform választást három fő érv hozta meg. Az egyik, hogy az atlétikának meghatározó része az Egyesült Államokban zajlik, ahol a legtöbb sportoló Apple termékeket, ezen belül is iPhone-t használ. Ezen felül saját tapasztalataim alapján a Magyar atlétikai életben is kedveltek ezek az eszközök. A harmadik érvről majd a dokumentáció fejlesztői részében lesz szó. A regisztrációhoz, bejelentkezéshez valamint az idő szinkronizációhoz elengedhetetlen az internet kapcsolat, így Wi-Fi vagy mobilinternet szükséges a szoftver használatához.

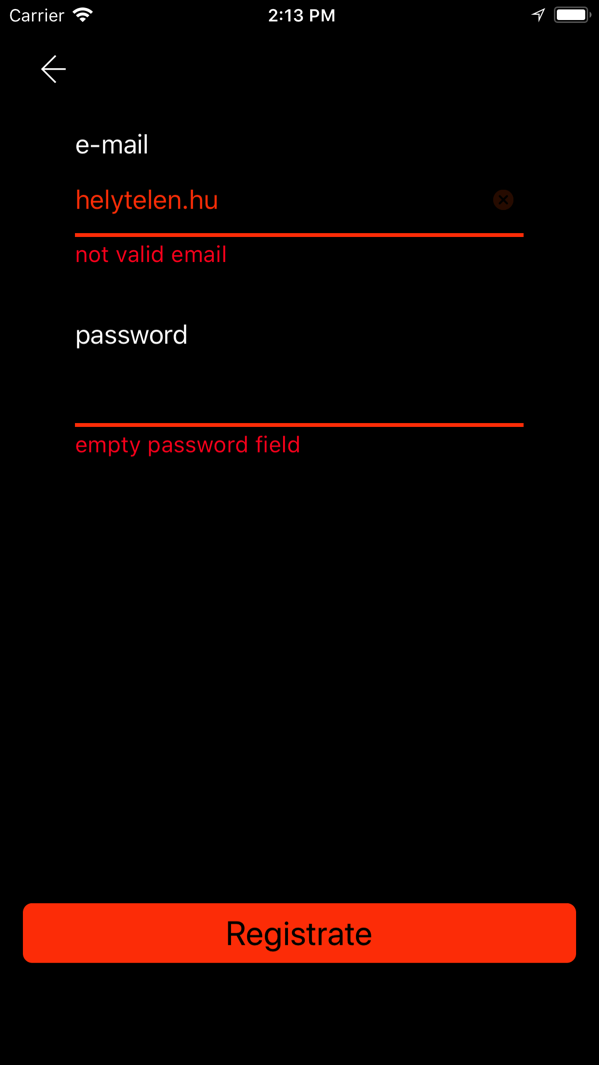
## 2.2 A szoftver felépítése

### 2.2.1 Kezdőképernyő

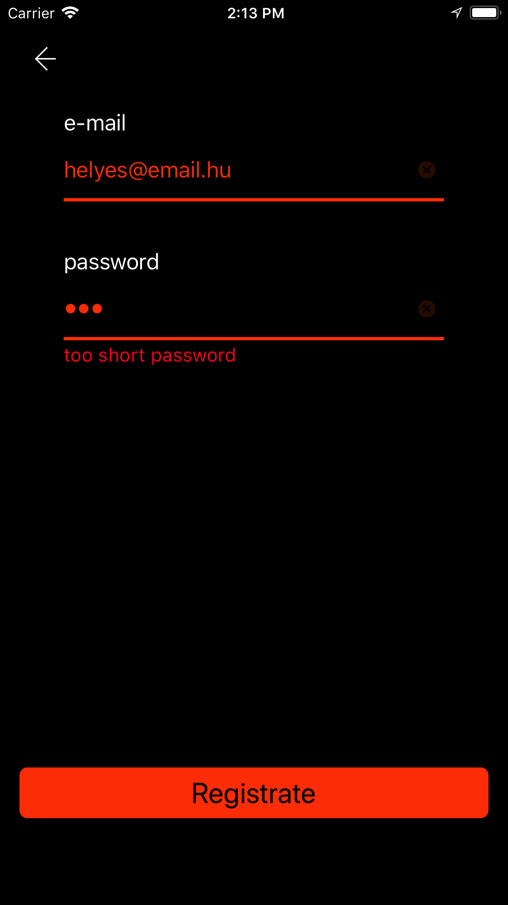


1. Az alkalmazás logója
2. Bejelentkezés sportolóként, előzetes regisztráció szükséges
3. Bejelentkezés edzőként, előzetes regisztráció szükséges
4. Regisztráció

### 2.2.2 Regisztráció

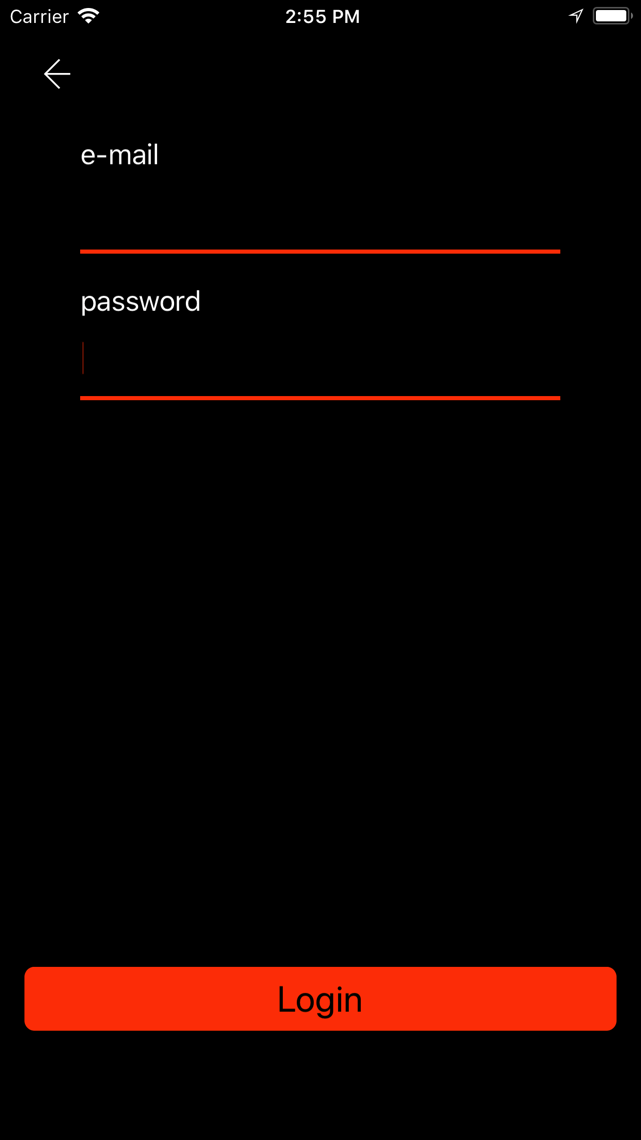
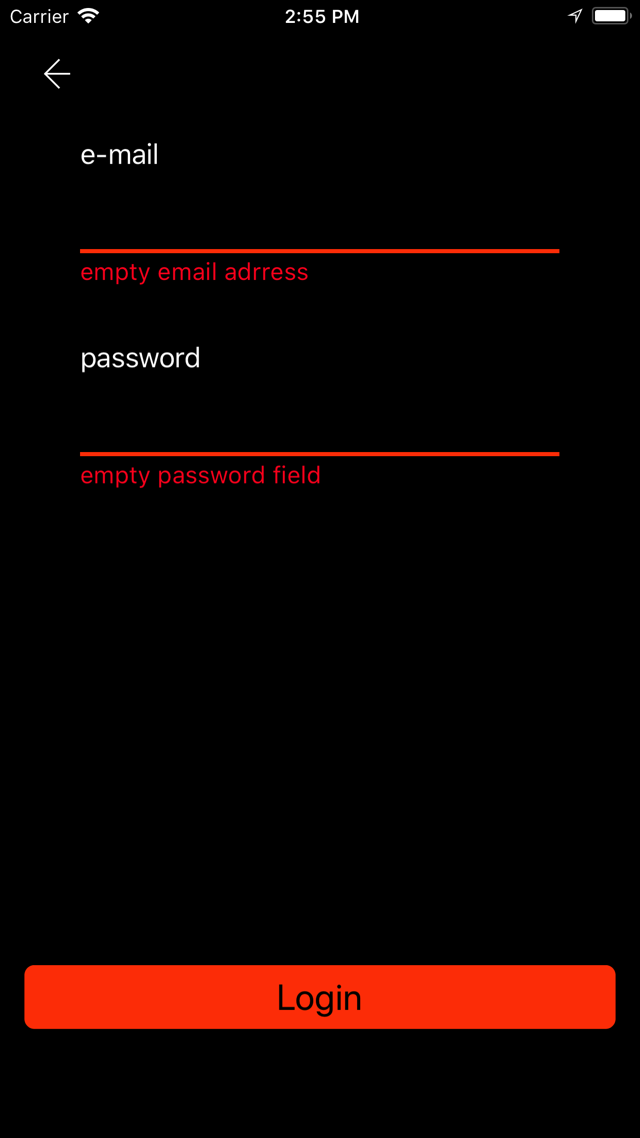






1. A leendő felhasználó email címe. Elvárt email formátum: [valami]@[domain].[tartomány]. Ez a formátum hibakezeléssel ki is van kényszerítve. Természetesen ugyanazzal az email címmel nem regisztrálhatunk kétszer.
2. A felhasználó által választott jelszó. A bejelentkezéshez lesz szükséges. Nem lehet rövidebb 5 karakternél.
3. Regisztráció gomb. Ez végzi az adatok validálását. Amennyiben az email, jelszó páros minden kritériumnak megfelel, sikeresen regisztrálhatunk.

### 2.2.3 Bejelentkezés

A bejelentkezéshez az előzőekben megadott email-jelszó párosra van szükségünk. Amennyiben helyesek a megadott adatok, a választott opció (edző, sportoló) bejelentkezett kezdőképernyőre jutunk.

HIBAKEZELÉSRŐL KÉP-----------------

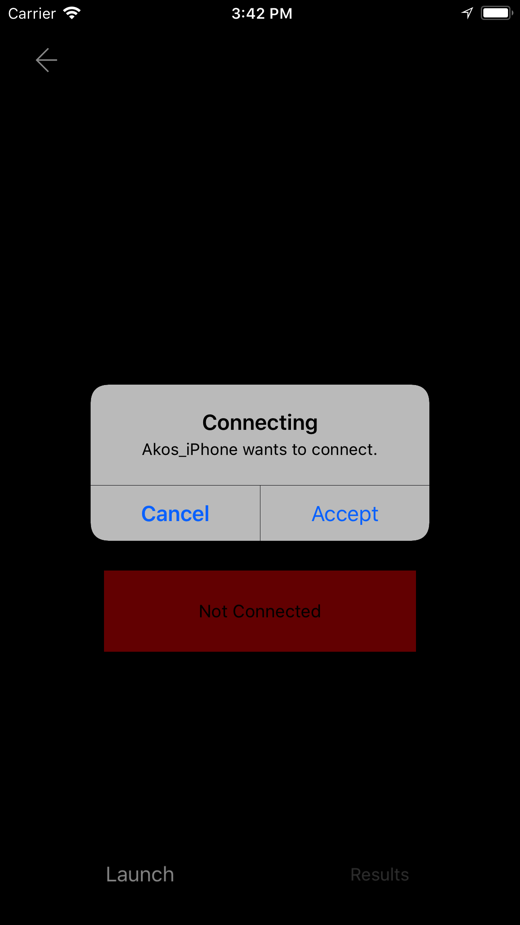
### 2.2.4 Funckiók

A szoftver – bejelentkezés után – 2 részből áll. Az egyik képernyőn vihetjük véghez az indítást, a másik fülön pedig megnézhetjük az eredményeinket, edzői esetben pedig az adott rajt eredményeit.

### 2.2.5 Edzői oldal

* Sikeres edzői bejelentkezés után, az itt látható képernyők közül az első kettő valamelyikére kerülünk, annak függvényében, hogy van-e már sportolói módba állított készülék.
* A listán a kívánt eszközt(sportolót) kiválasztva csatlakozhatunk hozzá.
* Amennyiben csatlakozásunk elfogadásra került, a narancssárga sáv zöldre, a GO! gomb pedig aktív állapotra vált.
* Ebben a helyzetben, már indításra is van lehetőségünk, hiszen akár egyedül is rajtolhat egy sportoló edzésen.

### Sportolói oldal



* Sikeres sportolói bejelentkezés után, a fent látható képernyők közül az elsőre érkezünk.
* Az alkalmazás használatához ebben a módban feltétlenül szükség van egy másik félre, aki edzői módban csatlakozik hozzánk.
* A csatlakozási kérést elfogadhatjuk, vagy elutasíthatjuk.
* Elfogadott csatlakozási kérés esetén csatlakozott állapotba kerülünk, és ezzel jelezzük, hogy készen állunk rajtolni.

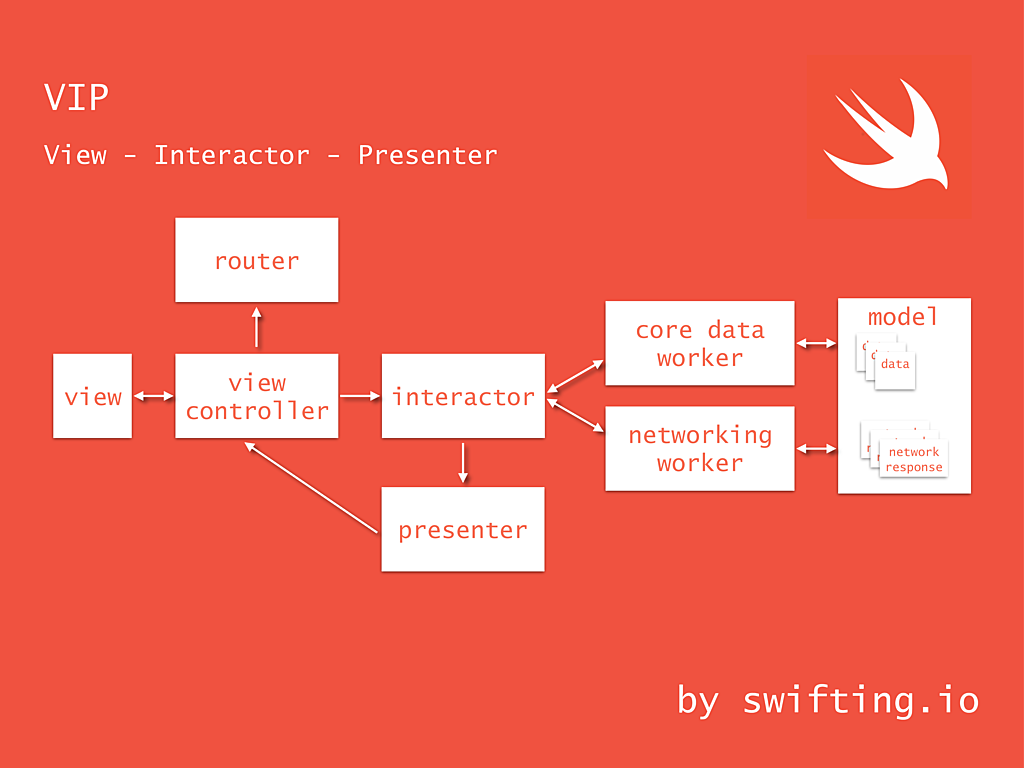
ss

LISTÁZÁS KÉPEKET MINDKÉT TÍPUSBÓL------------

# Fejlesztői dokumentáció

## A program szerkezete

A szoftver swift nyelven xcode fejlesztői környezetben készült clean swift arhitektúrában.



Forrás: hackernoon.com

Egy pár évvel ezelőtt az összes iOS alkalmazás kicsi, körülbelül 10 oldalt tartalmazó volt. A kódbázis ehhez mérten nem volt nagy, és a storyboardos fejlesztői megoldás tökéletesen működött. A kor előre haladtával egyre nagyobb 20-40 képernyős alkalmazásokra volt igény, amit már ez a módszer nem tudott kiszolgálni.

A Clean arhitektúrában minden modul protokollok segítségével kommunikál egymással, a fenti ábrán látható irányba.

Ez a módszer nagy mértékben hasonlít a sokunk számára ismert MVC (Model, View, Controller) programozási modellhez. Ez a szerkezeti minta mind fejlesztői mind felhasználó szempontból előnyös. Nagy mértékben csökkenti a szerkezeti bonyolultságot, növeli a rugalmasságot és az átláthatóságot. Az alkalmazás a bejelentkezéshez, valamint az eredmények tárolásához egy Firebase háttérrendszert alkalmaz. Ez fejlesztői szempontból egy igen egyszerű és könnyen átlátható megoldást biztosít.

* Az *Interactorok* esetünkben több dolgotot is végeznek:
  + Kommunikál a fentebb említett Firebase adatbázissal:
    - Regisztráció
    - Bejelentkezés
    - Eredmények lekérése
  + Validálja a formokat:
    - Hibás/Üres e-mail mező
    - Rövid/Üres jelszó mező
    - Nem regisztrált felhasználó
* A *ViewController*ek felelősek mindenért amit a felhasználó lát:
  + Animációk
  + Adatok, eredmények átlátható megjelenítése
* A *Presenter*ek végzik a kommunikációt az *Interactor* és *a ViewContoller*ek között:
  + Ha bármi logikai változás van a képernyőn akkor az *Interactor* szól *Presenter*nek, aki megformázva ezt a változást:
    - Létrehoz egy AlertControllert
    - Megformáz egy hibaüzenetet
  + Majd továbbadja a már kész objektumot a *ViewController*nek megjelenítésre.

Ezen szoftver fejlesztéséhez nincs szükségünk a clean arhitektúra minden elemére, így jelen működést a következő ábra szemlélteti:

## MultiPeerConnectivity

A telefonok közötti kommunikációt, egy a Swiftben megtalálható könyvtárral került megvalósításra. Ennek lényege, mint ahogy a nevéből is adódik, hogy több kommunikációs eszközt is használ. A felhasználói dokumentációban említett két érv mellett, ez a könyvtár, és a pratikussága a harmadik érv az iOS platformon való megvalósítás mellett. Ez a keretrendszer Wi-Fi, peer-to-peer Wi-Fi, és Bluetooth segítségével hozza létre a kapcsolatot az eszközök között, méghozzá pillanatok alatt.

### Kommunikációs keretrendszer darabjai

A könyvtár megvalósítása során 4 dolog működik együtt:

* MCSession: Ez az objektum felel az adott sessionért, tárolja a csatlakozott illetve a látható eszközöket. Ezen felül ez valósítja meg a kommunikációt is a csatlakozott felek között.
* MCNearbyServiceAdvertiser: Ahogy a neve is árulkodik róla, ő valósítja meg az adott eszköz „hirdetésést”, azaz teszi lehetővé, hogy a következőekben felsorolt objektum majd láthassa azt aki hirdeti magát.
* MCNearbyServiceBrowser: Lehetővé teszi, hogy az alkalmazásunk a felsorolás előző elem segítségével látható legyen számára.
* MCPeerID: Egyértelműen azonosít egy futó alkalmazást. Ezzel tudjuk azonosítani a látható eszközöket.

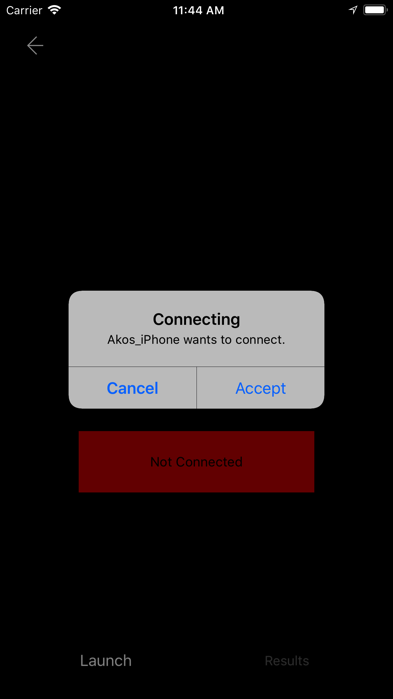
### Kommunikáció – Csatlakozás

A feljebb említett keretrendszer segítségével csatlakoznak az eszközök egymáshoz. Ehhez elég csupán annyi, hogy a három kommunikációs formából egy elérhető legyen.

A csatlakozás megkezdéséhez a feljebb említett MCNearbyServiceBrowserre lesz szükség, egész pontosan ennek egy tagfüggvényére, mégpedig az invitePeer(\_ peerID: MCPeerID, to session: MCSession, withContex: Data?, timout: TimeInterval)-re. Ennek paraméterei rendre a következők:

* peerID: Az az eszköz, akihez csatlakozni szeretnénk.
* session: Az az MCSession, aminek segítségével, amibe csatlakozni tervezünk.
* withContex: Ez egy nilable paraméter, előzetesen küldhetünk adatot, még a csatlakozás elfogadása előtt. Ezen alkalmazás ezt a paramétert nem használja.
* timeOut: Ahogyan a nevéből is adódik, a csatlakozásra megengedett időt várja.

### Kommunikáció – Csatlakozás elfogadása

A csatlakozás ezen fázisát meg kell előzze a csatlakozási kérelem. Ha ez megtörtént, az alkalmazás egy figyelmeztetést dob fel. Ezen hibaüzenet megjelenítése MCNearbyServiceAdvertiserDelegate segítségével lehetséges. Ez a protocol tartalmaz egy függvényt amely csatlakozási kéréskor meghívódik, a következő módon:

Advertiser(MCNearbyerviceAdvertiser, didReceiveInvitationFromPeer: MCPeerID,

withContext: Data?, invitationHandler: (Bool, MCSession?) -> Void)

Ezen hívás paraméterei rendre:

* didReceiveInvitationFromPeer: A csatlakozni kívánó fél
* withContext: Ahogy ezen függvény párjánál feljebb nem nilként hagytuk ezt a mezőt, úgy ennek megfelelően, itt is üres lesz.
* invatitionHandler: Egy olyan blokk amelynek mindig le kell futnia. Ez társítja az érkező eszközt a fogadó készülék MCSessionjéhez.

### Kommunikáció – Csatlakozva

Amennyiben a fent említett folyamat maradéktalanul végbe ment, két összekapcsolt eszközhöz jutunk. Ezen két eszköz között az MCSession send függvényével tudunk adatot küldeni amely Data típusú. Fogadni pedig a csatlakozáshoz hasonlóan az MCSessionDelegatettel tudjuk.

### MultiPeerConnectivityManager

Természetesen az Apple által biztosított könyvtár számunkra szükséges funckióit egy osztályba összecsoportosítani, hogy az enkapszulációt biztosítsuk.

Ez az osztály a már feljebb említett típusokkal dolgozik, a következő módon:

* Mindkét bejelentkezési típushoz szükséges egy MCSession, ezért ez minden esetben a konstruktorban létrejön.
* Edzői esetben ezen kívül létrehozunk egy MCNearbyServiceBrowser-t majd a *startBrowsingForPeers()* tagfüggvényével aktiváljuk a keresést.
* Sportolói esetben a Browser párját az MCNearbyServiceAdvertiser-t hozzuk létre, majd a *startAdvertisingPeers()* függvényével elkezdjük hirdetni az eszközt.

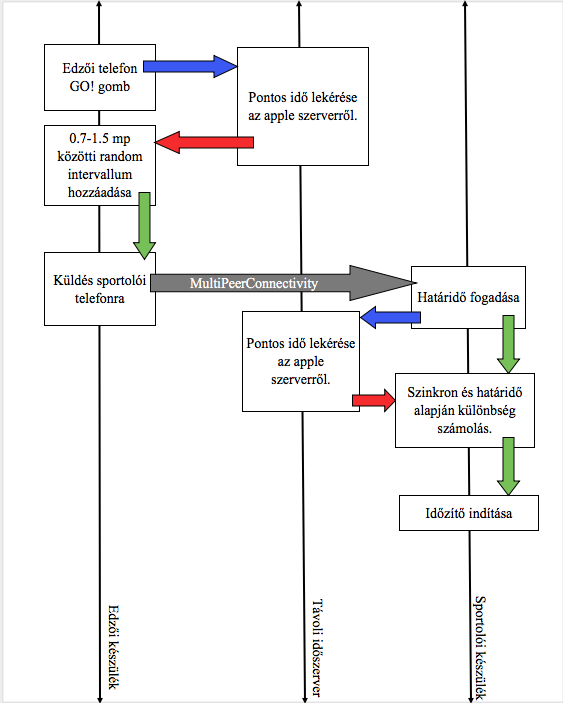
Ezen osztályszintű változókhoz tartozó kiegészítésekben valósítjuk meg a feljebb említett függvényeket, amelyekben a csatlakozás, bontás és kommunikáció lényegi része történik.

|  |
| --- |
| MultiPeerConnectivityManager : NSObject |
|  |
|  |

## 3.3Időszinkronizáció

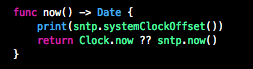
### 3.3.1 Folyamat ábra

Az alkalmazás működésének szíve egy idő szinkronizációs algoritmus. Az alábbi ábra szemlélteti ennek működését:



### Implementáció

Az időszinkronizáció implementálása a TimeManager osztályban került megvalósításra. Ezen belül meghívásra kerül két különböző valósidő harmadik féltől származó könyvtár. Erre azért van szükség, hogy az időpont lekérés redundáns lehessen. Egy igen egyszerű függvényen belül kéri le az osztály az időt, amely a következő:



Ez a Date visszatérési értékkel rendelkező függvény a két különböző Pod *now* függvényét hívja, és adja vissza eredményüket. Amennyiben az első hívásból kapunk eredményt, úgy a második (?? operátor utáni) már nem értékelődik ki.

*Az előző bekezdésben említett Pod kifejezés, egy olyan szoftver, amellyel xcodeban fejleszthető projektekhez könnyen adhatunk hozzá harmadik féltől származó könyvtárakat.*

*Bővebb információ: https://cocoapods.org*

Ezen könyvtárakról részletesebben a Github tárhelyén található információ:

* <https://github.com/lyft/Kronos>
* <https://github.com/DarkDust/MHSNTP>

Az osztály mindezeken felül még két lényeges függvényt tartalmaz, amelyek az időzítő indításának pontos idejét segítenek meghatározni:



Ezek segítségével állítjuk elő az indítás idejét az edzői készüléken, majd a második függvénnyel a sportolói eszközökön számoljuk, azt az idő intervallumot amennyi múlva az indítás esedékes.