



Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Távközlési és Médiainformaticai Tanszék

# Multi-platform sportkövető mobilalkalmazás tervezése és fejlesztése

SZAKDOLGOZAT

*Készítette*

Halász Antal István

*Konzulens*

Dr. Toka László

2017. május 7.

# Tartalomjegyzék

<b>Kivonat</b>	<b>4</b>
<b>Abstract</b>	<b>5</b>
<b>Bevezető</b>	<b>6</b>
<b>1. A SmartActive mobilalkalmazás koncepciója</b>	<b>8</b>
1.1. Kezdeti tervezek . . . . .	8
1.2. Követelménymódosulások . . . . .	11
<b>2. Multi-platform mobilsoftver-fejlesztő keretrendszerek</b>	<b>16</b>
2.1. A multi-platform mobilsoftver-fejlesztés . . . . .	16
2.2. Az elérhető multi-platform keretrendszerek . . . . .	17
2.2.1. Sencha . . . . .	17
2.2.2. Apache Cordova (PhoneGap) . . . . .	18
2.2.3. Appcelerator . . . . .	18
2.2.4. Corona . . . . .	18
2.2.5. Unity . . . . .	18
2.2.6. Kony . . . . .	19
2.2.7. Xamarin . . . . .	19
<b>3. A szoftverfejlesztő infrastruktúra</b>	<b>20</b>
3.1. A Microsoft Visual Studio 2015 Update 2 telepítése és beállítása . . . . .	20
3.1.1. Programfordítás iOS operációs rendszerre . . . . .	21
3.1.2. További beállítások a SmartActive mobilapplikáció fordításához . . . . .	22
3.2. Az okostelefon-emulátorok . . . . .	22
3.2.1. Az emulátorok használata . . . . .	22
3.2.2. Interakció az emulátorok fájlrendszerével . . . . .	23
<b>4. A mobilapplikáció fejlesztése és tesztelése</b>	<b>26</b>
4.1. A Xamarin multi-platform fejlesztési lehetőségei . . . . .	26
4.2. A fejlesztés során használt főbb programozási paradigmák . . . . .	28
4.2.1. Az MVVM . . . . .	28
4.2.2. C# grafikus elemek leírása XAML-ben, adatkötés . . . . .	28
4.2.3. Aszinkron mechanizmusok . . . . .	29

4.3.	Az alkalmazáslogika . . . . .	30
4.3.1.	Kommunikáció a webszerverrel . . . . .	30
4.3.2.	Az adatok perzisztálása . . . . .	31
4.3.3.	Az üzleti logika vezérlése . . . . .	31
4.4.	A felhasználói felület elkészítése Xamarin.Forms-ban . . . . .	32
4.4.1.	Alkalmazásoldalak . . . . .	32
4.4.2.	Elemek elhelyezése az oldalakon . . . . .	34
4.4.3.	Stílusozás . . . . .	35
4.5.	Az alkalmazás tesztelése . . . . .	36
4.5.1.	Unit tesztek . . . . .	37
4.5.2.	Manuális tesztek . . . . .	38
<b>5.</b>	<b>Az elkészült alkalmazás bemutatása</b>	<b>39</b>
5.1.	Bejelentkező-oldal . . . . .	39
5.2.	Információk, statisztikák az edzésekéről . . . . .	40
5.3.	Barátok kezelése . . . . .	42
5.4.	Bajnokságok szervezése, részvétel bajnokságokon . . . . .	44
<b>6.</b>	<b>Konklúzió, továbbviteli lehetőségek</b>	<b>48</b>
<b>Irodalomjegyzék</b>		<b>52</b>
<b>Mellékletek</b>		<b>53</b>
M.1.	Forrásfájlok . . . . .	53
M.2.	Telepíthető alkalmazáscsomagok . . . . .	53
<b>Függelék</b>		<b>54</b>
F.1.	A Visual Studio 2015 Update 2 komponenstelepítési beállításai . . . . .	54
F.2.	A Visual Studio emulátorokat kezelő szoftverek . . . . .	56
F.3.	Példakód az adatkötés (data binding) szemléltetésére . . . . .	57
F.4.	Példakód stílustulajdonság adatkötésére statikus változóhoz az x:Static XAML leíróbővítmény használatával . . . . .	58
F.5.	Példakód elem tulajdonságok adatkötésére statikus változókhöz az x:Static XAML leíróbővítmény használatával . . . . .	59
F.6.	Multi-platform képernyőképek az elkészült alkalmazásról . . . . .	60

## HALLGATÓI NYILATKOZAT

Feladatom során egy webes alkalmazást készítettem el, mely képes weboldalak szöveges tartalmának feldolgozására, elemzésére adat- és szövegbányászati módszerek segítségével. A feladat megvalósításának érdekében megismerkedtem a webalkalmazások működési elvénnyel, elsajátítottam a Python nyelv ismeretét, megismertem a Django webes keretrendszer használatát. Implementáltam a korábbi tanulmányaim alatt szerzett szöveg- és adatbányászati ismereteket, és az elért eredményemet felhasználva automatikus adatbányászati folyamatokat integráltam a webalkalmazásba. A webalkalmazás adat- és szövegbányászati eszközökkel felhasználva alkalmas általános információk kinyerésére, és weboldal kategóriák meghatározására. Keretrendszer nyújt specifikus weboldalakra történő, célzott adatbányászati elemzések elkészítéséhez. A webes felületen keresztül lehetősége van a felhasználónak az előre elkészített kutatás és általános célú elemzések futtatására is. A weboldal az alapvető webes funkcionálitásokkal rendelkezik (felhasználó azonosítás, internetes működés, reszponzív grafikus felületek), és a felhasználó által betaplált dokumentumokat és webcímeket analizál. A keretrendszerbe integrált adat- és szövegbányászati eljárások, web-crawler technológiák és megjelenítő eszközök segítségével, külső beavatkozás nélkül képes az adatok feldolgozására, eredmények közlésére. <TODO innen indul a számozás>

Alulírott *Halász Antal István*, 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 átvettettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), 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 autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervezet esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2017. május 7.

---

*Halász Antal István  
hallgató*

# Kivonat

Napjainkra a komplex operációs rendszereket futtató, számos szenzorral és funkcionalitással rendelkező okostelefonok az átlagember minden napjainak szerves részévé váltak. A sport, mint kikapcsolódási forma egyre többeket vonz, melynek az okostelefon-használat is részévé vált. Az emberek szeretnék teljesítményüket mérni, nyomon követni, valamint részletes statisztikák alapján sportolási tevékenységüket javítani, finomhangolni.

Amíg a jelenleg elérhető sportkövető mobilapplikációk kiválóan alkalmasak szabadtéri sportok analíziséhez, addig a beltéren végzett sportoknál kézzel kell felvinnie a felhasználónak edzése jellemzőit.

A tanszéki SmartActive projekt célja a squash sport lehetőségeit vizsgálva új elemeket vinni a játékba, illetve részletesebb statisztikákat nyújtani a játékosoknak teljesítményükről, kezdőtől profi szintig. A rendszer adatokat gyűjt a játék során, majd egy adatbázisban tárolja, amiket a felhasználók egy webportálon keresztül elérhetnek. A projekt egy másik célja a felhasználók közti interakció megvalósítása, például bajnokságok szervezésének lehetősége, amire a játékosok kihívhatják barátaikat. A feladatom egy olyan multi-platform mobilalkalmazás fejlesztése volt, ami támogatja ezeket a funkciókat.

Szakdolgozatomban először ismertettem a készítendő mobilalkalmazás koncepcióját. Ezután megvizsgáltam a jelenleg elérhető multi-platform mobilsoftverfejlesztőkeretrendszeret, és értékkeltem őket a megvalósítandó applikációhoz való alkalmasságuk szempontjából. Bemutattam a Xamarin multi-platform keretrendszerben történő fejlesztéshez szükséges eszközöket, azok beállítását és használatát. A keretrendszer fejlesztési lehetőségeit megvizsgáltam, ismertettem az alkalmazás elkészítése során felhasznált fontosabb elemeit, valamint a hibákat, amikkel a fejlesztés során találkoztam. Részleteztem az alkalmazás struktúráját, emellett kitértem az implementációhoz használt főbb programozási paradigmákra és a unit tesztelésre. Végül részletesen bemutattam az elkészült alkalmazás grafikus felületét és annak működését. Az applikáció funkcionalitása és struktúrája minden célplatformon megegyezik, ezzel azonos élményt nyújtva a felhasználóknak.

# Abstract

Smartphones, which run complex operating systems, are equipped with numerous sensors and provide a rich feature set, have become an integral part of our everyday lives. Exercising as a free time activity is more and more appealing and smartphones are used in this area as well. People would like to measure and track their progress, also improve and tune it based on detailed statistics about it.

While the currently available mobile applications are perfectly capable of tracking outdoor sports, indoor activity details have to be added manually to the app. The main goal of the department's SmartActive project is to examine the possibilities of squash to add new elements to it, also to provide detailed and useful statistics for the players about their performance, at any scale from beginner to professional levels. The system collects data during the gameplay, which is stored in a database that can be accessed through a web portal. Another goal of the project is to provide a community platform to the users, e.g. someone can organize a tournament, to which he/she can invite his/her friends. My objective was to design and develop a cross-platform mobile application, which can support these features.

In my bachelor thesis I first introduced the concept of the mobile application to be developed. After that I examined the currently available cross platform mobile software development frameworks and reviewed them by their compatibility to the SmartActive mobile application. Then I presented the needed tools for starting development in the Xamarin framework, also showed their correct settings and usage. I reviewed the options of development in the framework, presented its main elements used for the development, also their issues, which I encountered while implementing the program. I gave a detailed view of the mobile application's structure, also touched upon the main programming paradigms and unit testing concept I used for the implementation. Finally, I presented the completed application's user interface and its operation in detail. The structure and feature set of the program is the same on all target platforms, thus providing a consistent user experience.

# Bevezető

Napjainkra a komplex operációs rendszereket futtató, számos szenzorral és funkcionálitással rendelkező okostelefonok az átlagember minden napjainak szerves részévé váltak. 2016-ra több mint kétemilliárd okostelefon-használó van világszerte [41], ami az előrejelzések szerint jelentősen növekedni fog a közeljövőben [12], ezért alkalmazások fejlesztése ezekre az eszközökre egy fontos területnek tekinthető.

A sport, mint kikapcsolódási forma egyre többet vonz, melynek az okostelefon-használat is részévé vált. Az emberek szeretnék teljesítményüket mérni, nyomon követni, valamint részletes statisztikák alapján sportolási tevékenységüket javítani, finomhangolni.

A fő okostelefonoperációs-rendszerekre (Apple iOS, Google Android és Microsoft mobil rendszerek) számos sportkövető alkalmazás érhető el (pl. EndoMondo, Strava, Runkeeper, Sports Tracker, HeiaHeia). Ezekben közös, hogy helymeghatározás alapú követési módszert használnak, illetve a felhasználók közé tehetik tevékenységüket, kihívhatják barátaikat különböző versenyek teljesítésére (pl. egy hónap alatt 200 kilométer lefutása). Míg ez a sportkövetési módszer kiválóan alkalmas szabadtéri sportok analíziséhez, addig a beltéren végzett sportoknál kézzel kell felvinnie a felhasználónak edzése jellemzőit a rendszerbe.

A tanszéki SmartActive projekt célja a squash sport lehetőségeit vizsgálva új elemeket vinni a játékba, illetve részletesebb statisztikákat nyújtani a játékosoknak teljesítményükön, kezdőtől profi szintig. Ehhez elláttak egy squashpályát és squash ütőket különféle szenzorokkal, melyekkel többek között érzékelhetővé válik a labdamenet, a játékosok ütési jellemzői, valamint az általuk megtett lépésszám és távolság a meccsek során. Az adatokat akár valós időben is elemzi a rendszer, majd egy adatbázisban tárolja, amiket a felhasználók egy webportálon keresztül elérhetnek. A projekt egy másik célja a felhasználók közti interakció megvalósítása, például bajnokságok szervezésének lehetősége, amire a játékosok meghívhatják barátaikat.

Szakdolgozatomban először ismertetem egy, a SmartActive projekthez készítendő mobilalkalmazás koncepcióját, amivel a felhasználók bárholt könnyen és gyorsan hozzáférhetnek a sportolásuk szempontjából fontos információkhöz, valamint elérhetik a közösségi funkciókat is. Ezután megvizsgálom a jelenleg elérhető multi-platform mobilsoftverfejlesztőkeretrendszerket, és értékelem őket a megvalósítandó applikációhoz való alkalmasságuk szempontjából. Bemutatom a Xamarin multi-platform keretrendszerben történő fejlesztéshez szükséges eszközöket, azok beállítását és használatát. A keretrendszer fejlesztési lehetőségeit megvizsgálom, ismertetem az alkalmazás elkészítése során felhasznált elemeit, valamint a hibákat, amikkel a fejlesztés során találkoztam. Részletezem az alkalmazás

struktúráját, emellett kitérek az implementációhoz használt főbb programozási paradigmákra és a unit tesztelésre. Végül részletesen bemutatom az elkészült alkalmazás felhasználói felületét, annak működését.

## 1. fejezet

# A SmartActive mobilalkalmazás koncepciója

A következőkben bemutatom a mobilalkalmazás tervezésének kezdeti fázisát, a megvalósítandó funkciókat, valamint az applikáció tervezett kinézetét.

### 1.1. Kezdeti tervezet

Az alkalmazás tervezését azzal kezdtem, hogy kilenc, squash sportot különböző szinten ūzõ ismerősömmel interjút készítettem, hogy milyen funkciókat látnának szívesen egy, a sportról szóló mobilalkalmazásban.

Számos igényt gyűjtöttem, amik figyelembe vételével a szakdolgozat írásának kezdetén az alábbi funkcionális követelményeket támasztottuk az applikációval szemben:

- Az alkalmazás legyen multi-platform, ezzel a legelterjedtebb okostelefon-operációs-rendszereken (Apple iOS, Google Android és Microsoft mobil rendszerek) azonos funkcionalitást és felhasználói élményt nyújtva.
- Lehessen megtekinteni statisztikákat az edzésekről:
  - használt labda típusa (kezdõ, középhaladó, haladó, profi)
  - idõtartam
  - megtett lépések száma
  - megtett táv
  - átlagos pulzusszám
  - hány ütést érzékelt a rendszer, azokat milyen típusú ütésekbõl (tenyeres, fonák)
  - hány melléütés történt
  - mekkora volt az ütések átlagos ereje km/h-ban, azaz mekkora sebességre gyorsult fel a labda
- Lehessen listázni a barátokat.

- A barátokról legyenek elérhetők részletesebb információk, például e-mail cím, telefonszám.
- Tegye lehetővé az időpontra történő pályafoglalást.
- A lefoglalt pályára lehessen partnert hívni közös sportolásra.
- A felhasználók be tudjanak jelentkezni a SmartActive fiókjukba.
- Az alkalmazás legyen képes szinkronizálni az adatokat a SmartActive szerverrel, és internetkapcsolat hiányában is az eddig szinkronizált adatokat megjeleníteni.

Az app megjelenített nyelvénék az angolt választottam, mivel a szoftverfejlesztésben ez a legelterjedtebb, illetve ez biztosítja a legnagyobb kompatibilitást a grafikusalkalmazásfejlesztő keretrendszerrel.

Fontos kitérni arra, hogy a Windows Phone operációs rendszer 8.1-es és 10-es verziója között jelentős különbségek vannak mind funkcionalitásban, mind a megjelenített alkalmazáselemek kinézetében, ezért legtöbb esetben két külön célplatformként szokás rájuk tekinteni szoftverfejlesztési szemszögből nézve. A Microsoft ezt az álláspontot erősíti a felhasználók felé azzal is, hogy a legújabb verzió neve már Windows 10 Mobile, így a továbbiakban én is két külön célplatformként hivatkozok rá.

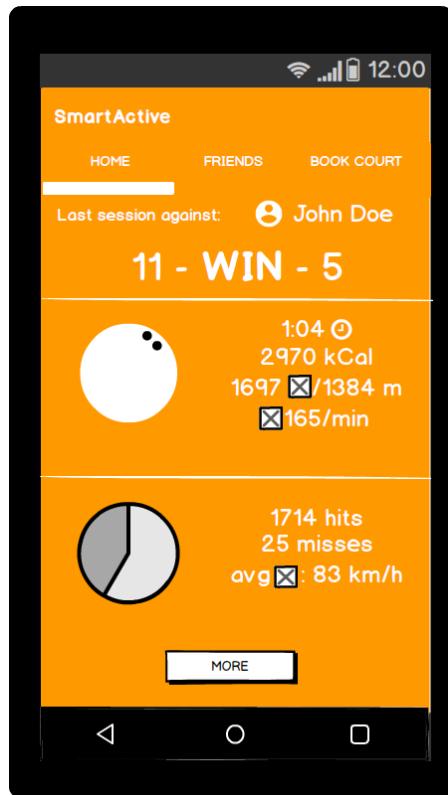
A fentiek alapján készítettem az alkalmazáshoz koncepciórajzokat, úgynevezett mockupokat<sup>1</sup>. Ehhez a Balsamiq Mockups 3 [11] nevű programot használtam, amihez a program online közössége (MockupsToGo community [2]) által készített sablonokat töltöttem le az alapértelmezettek mellé.

Az alábbiakban ismertetem az alkalmazás kezdeti kinézeti tervét:

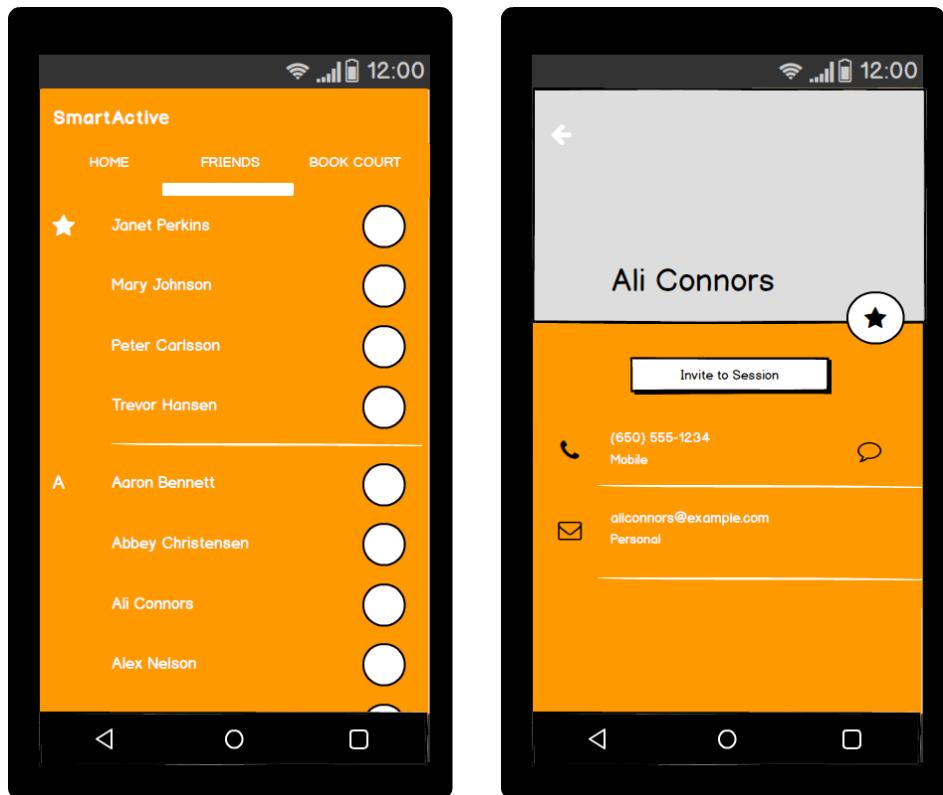
- *Nyitóképernyő*: az utolsó mérkőzés fent felsorolt statisztikái jelennek meg rajta, illetve az ellenfél neve (1.1. ábra). A *More* gombra kattintva egy statisztikatörténet érhető el a korábbi edzésekről.
- *Barátok lista*: a bejelentkezett felhasználó barátainak listája (1.2. ábra).
- *Barátrészletező oldal*: ez az oldal további részleteket jelenít meg adott ismerősről, továbbá az *Invite to Session* gomb megnyomásával meghívható egy már lefoglalt időpontra és helyszínre edzeni (1.2. ábra).
- *Squash termek lista*: azon létesítmények listája, ahol squash pálya foglalható (1.3. ábra).
- *Squashterem-részletező oldal*: megtekinthető a squash terem helye egy térképen, valamint dátum, ezen belül időintervallum szerint rendezett listában kiválasztható a szabad időpontok közül, mikorra foglal termet a felhasználó (1.3. ábra).

---

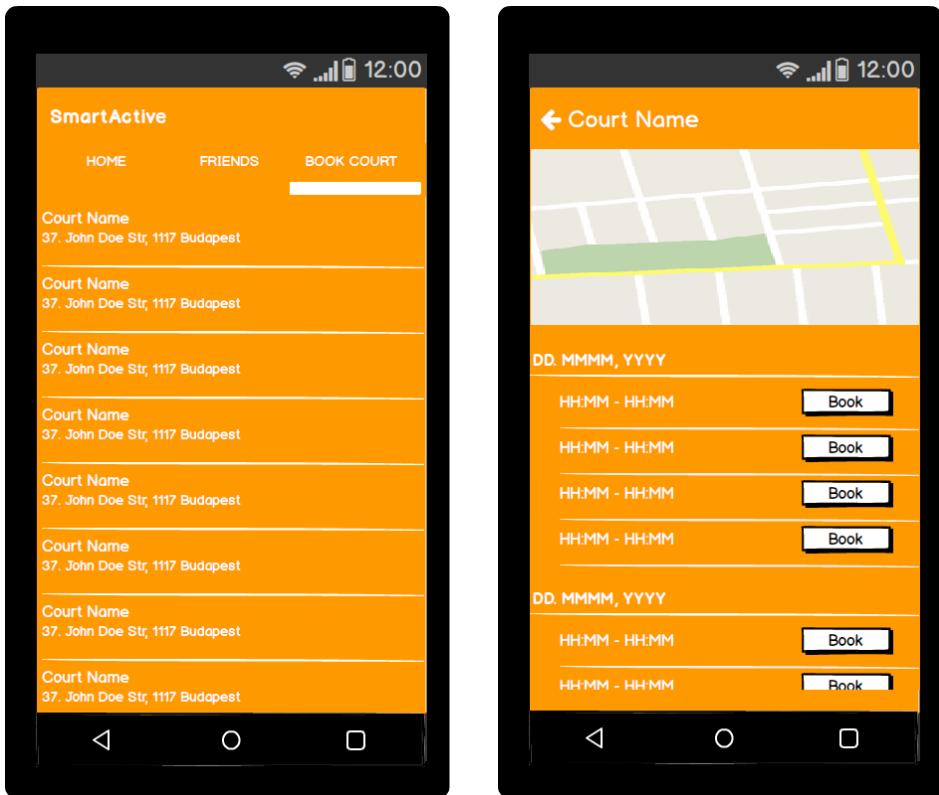
<sup>1</sup>A több helyen is látható fekete szélű, fekete átlókkal rendelkező fehér négyzet a képek és ikonok szokványos jelölése a mockupokon.



1.1. ábra. A kezdeti kezdőképernyő koncepciórajza



1.2. ábra. A kezdeti barátkezeléssel kapcsolatos koncepciórajzok



**1.3. ábra.** A kezdeti pályafoglalási koncepciórajzok

## 1.2. Követelménymódosulások

A SmartActive egy új és összetett projekt, ezért a megvalósítása során bekövetkezhetnek változások az eredeti tervekhez képest, amik oka technikai vagy a nyújtott szolgáltatások finomítása lehet tesztfelhasználói visszajelzések alapján. Ilyen okokból a mobilalkalmazással szembeni elvárások több ponton módosultak a munkám során.

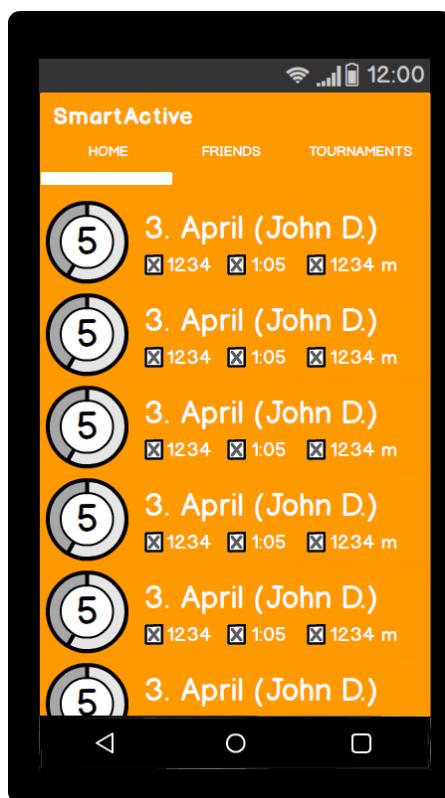
A változások listája:

- A kezdőképernyőn a korábbi edzések listája jelenjen meg, amiket az edzés dátuma és az ellenfél azonosít.
- Az edzésekkel lehessen megtekinteni az alatta játszott mérkőzéseket, azok eredményeit.
- A statisztikákban legyen grafikonon megtekinthető a szívritmus alakulása az edzés folyamán.
- A barátrészletező oldalon legyen látható a korábbi közös edzések listája.
- A barátok neve mellett az eddigi közös edzések száma is jelenjen meg.
- Legyen a barátlista kereshető.
- Legyen barátfelvételi lehetőség.
- A pályafoglalási funkció helyett az éppen futó bajnokságok listája jelenjen meg.

- Lehessen bajnokságot létrehozni annak idejének, helyének és a résztvevők maximális számának megadásával.
- Legyenek megtekinthetők a bajnokság részletei és résztvevőinek listája.
- A felhasználó tudjon csatlakozni bajnoksághoz.
- Csatlakozás után lehessen meghívni barátokat a bajnokságra.

A megváltozott kinézethez is készítettem koncepciórajzokat, amiket az alábbiakban mutatok be.

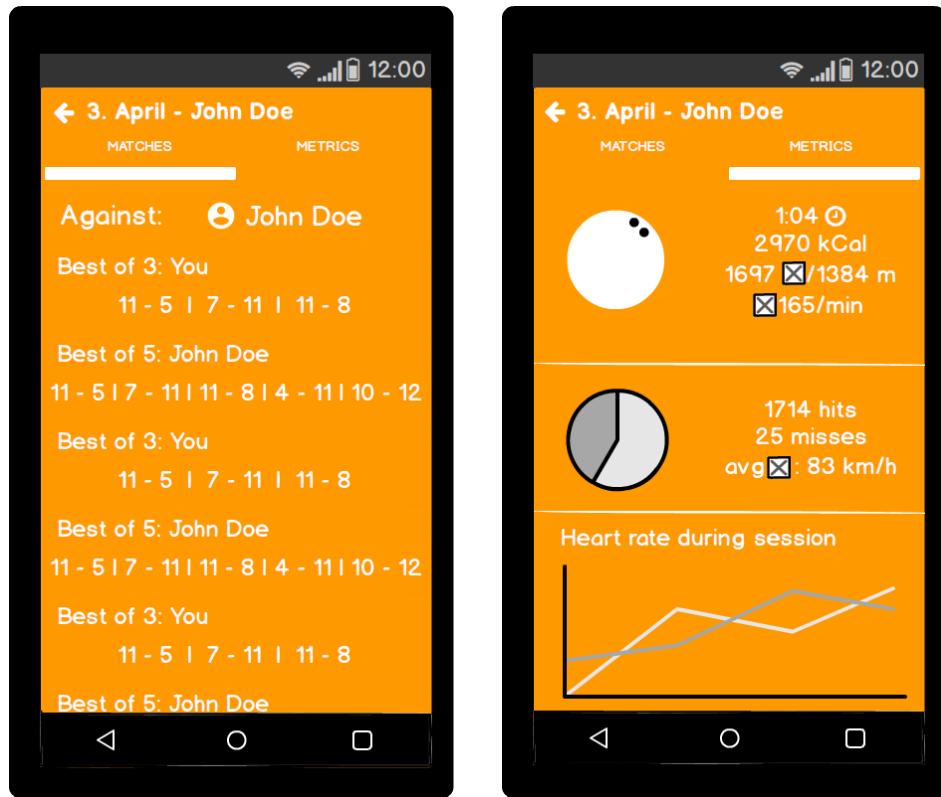
Az új kezdőképernyőn (1.4. ábra) a korábbi edzések listája tekinthető meg, valamint az edzés pár alapvető jellemzője: elégetett kalóriák, az edzés időtartama, és az edzés során megtett távolság. A listaelemek bal oldalán látható fánk diagramban lévő szám az edzés alatt játszott meccsek számát, míg a diagram a nyert és vesztes mérkőzések arányát mutatja.



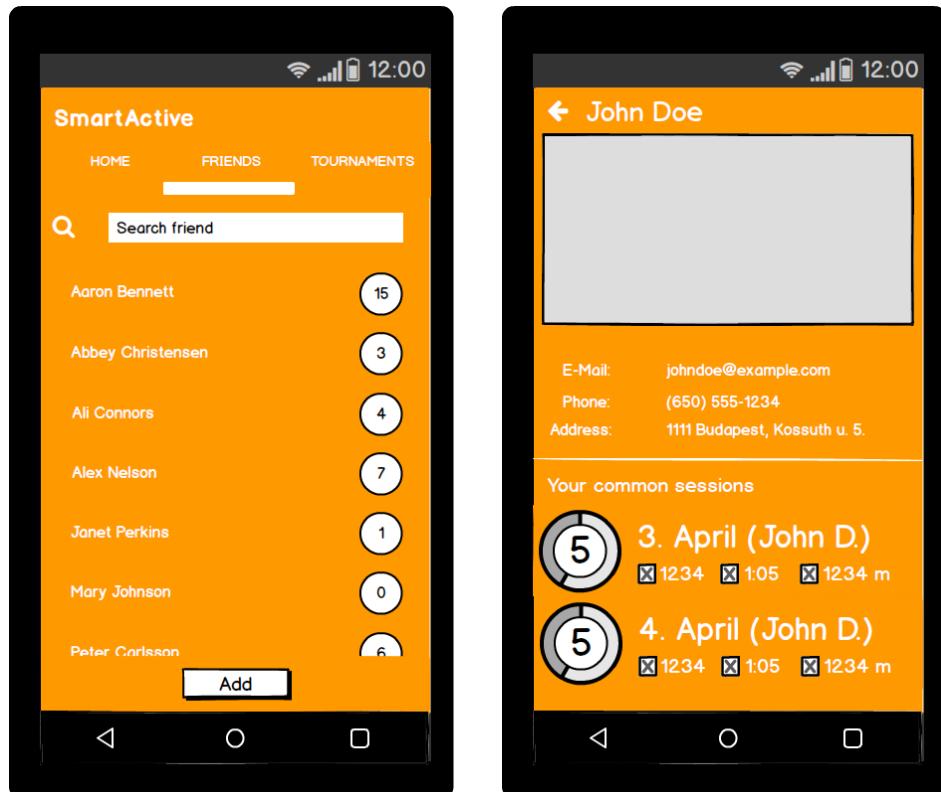
**1.4. ábra.** A módosult kezdőképernyő

Egy edzést kiválasztva arról részletesebb statisztikákat kaphatunk, ezt mutatja a 1.5. ábra. A *Matches* fülön láthatóak a lejátszott mérkőzések, azok típusa (legjobb a háromból vagy legjobb az ötből), a nyertes, és az egyes szettek eredményei. A *Metrics* fülön az edzés szintű statisztikák tekinthetők meg.

A barátok kezelése a kívánt elemekkel bővült (1.6. ábra), illetve lekerült a közös sportolásra meghívás lehetősége.



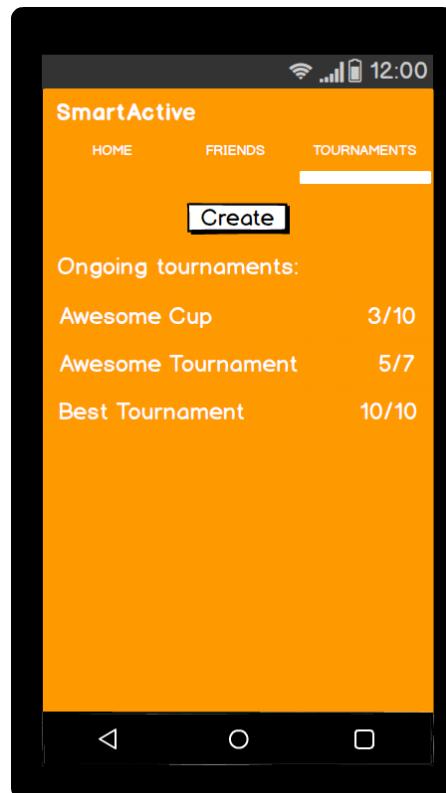
1.5. ábra. Az edzés részletes statisztikái



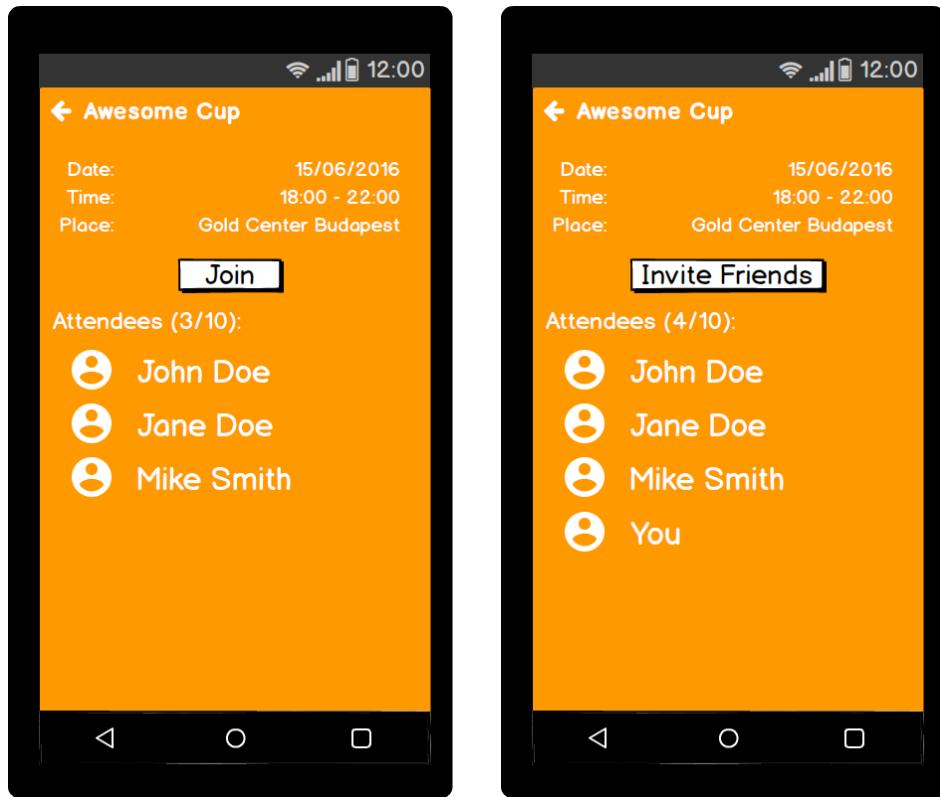
1.6. ábra. A bővített barátkezelés

A bajnokságokat a felhasználó a *Tournaments* fülön tekintheti meg, ugyanitt lehetősége van új bajnokság létrehozására is (1.7. ábra). A bajnokság részletezőoldalának (1.8. ábra) két állapota lehet: amíg a felhasználó nem csatlakozott, a *Join* gomb jelenik meg, utána az *Invite Friends*, amit választva a barátok meghívási oldalára juthat.

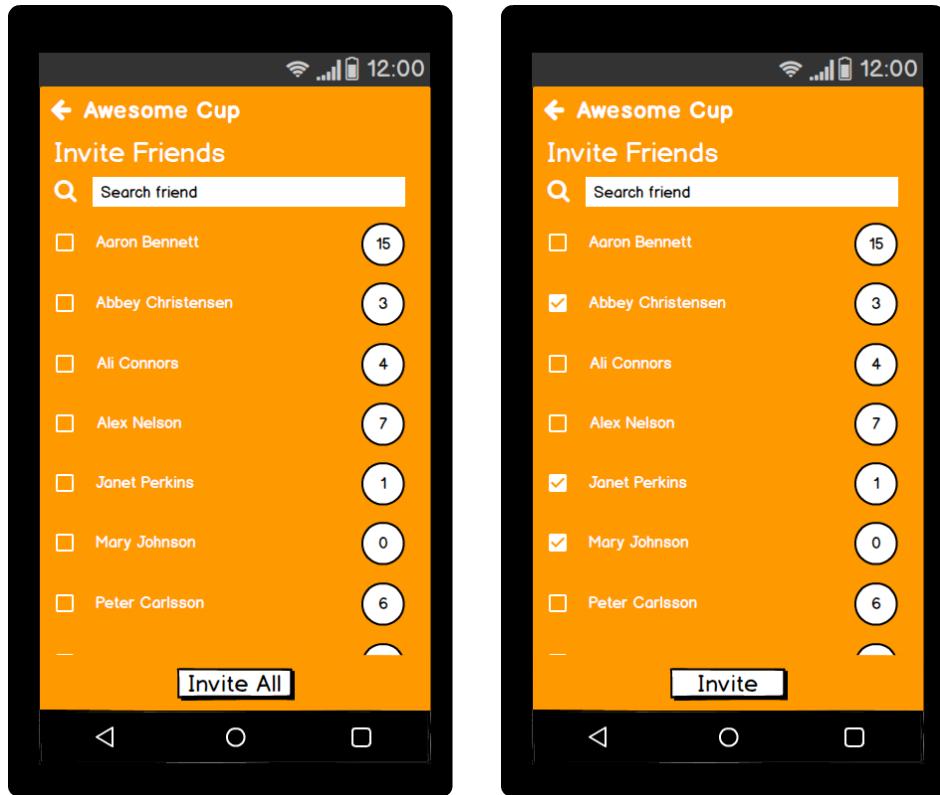
A bajnokságra meghívandó barátokat egy kereshető listából választhatja ki a felhasználó (1.9. ábra). A meghívást végrehajtó gombnak két állapota lehet: ha nincs kijelölve egy ismerős sem, akkor mindenkit meg lehet hívni egy gombnyomással (*Invite All*), ha a felhasználó kijelöl barátokat, akkor csak őket hívja meg (*Invite*).



1.7. ábra. A bajnokságok listája



1.8. ábra. Egy bajnokság részletei



1.9. ábra. Barátok meghívása bajnokságra

## 2. fejezet

# Multi-platform mobilszoftver-fejlesztő keretrendszerek

Ebben a fejezetben röviden összefoglalom a multi-platform szoftverfejlesztés témakörét, majd megvizsgálom a jelenleg elérhető multi-platform mobilszoftverfejlesztőkeretrendszereket, és értékelem őket a megvalósítandó applikációhoz való alkalmasságuk szempontjából.

### 2.1. A multi-platform mobilszoftver-fejlesztés

Az okostelefonok piacát jelenleg a Google Android operációs rendszerének különböző verzióival szállított készülékek uralják, második helyen stabilan az Apple iOS telefonok állnak, harmadik pedig a feltörekvőben lévő Microsoft Windows Phone platform [40]. A piac ezen osztottságán felül az operációs rendszerek különböző verziói és az azokat futtató készülékek képességei is jelentősen különbözhetnek egymástól. Jó példa erre az Android operációs rendszer szegmentáltsága, ahol egy kimutatás szerint 2016 májusában a piac egyharmadát a 4.4-es, "KitKat" kódnevű verzió teszi ki, második a közel húsz százalék-kal rendelkező 5.1-es "Lollipop", harmadik 16%-kal az 5.0-ás "Lollipop" [39]. Ennek a töredézettségnek az oka abban keresendő, hogy a telefonhardverek eszközillesztői jelentős újraoptimalizálást igényelnek az operációs rendszer minden frissítésekor. Ellenkező esetben a keletkező instabilitás a felhasználói élmény rovására mehet, például ha a telefon válaszideje jelentősen megnő. Extrém esetekben a készülék a hibás erőforrás-menedzsment miatt tönkre is mehet, például egy folyamat végtelen ciklusba kerül, amit a rendszer nem terminál, emiatt a processzor túlmelegszik, ezzel tönkre téve az akkumulátort és esetleg más áramköri egységeket. Mivel az eszközillesztők újraoptimalizálása és alapos tesztelése jelentős többletmunkát, ugyanakkor közvetlen bevételt nem jelent a cégek számára, ezért inkább ritkán, vagy egyáltalán nem adnak ki operációsrendszer-frissítést a forgalmazott modelljeik többségéhez.

Látható tehát, hogy minden egyes operációs rendszerre (és verzióra) optimalizált prog-

ramkódot - az adott platform programozási nyelvén - írni nem idő- és költséghatékony. Érdemes egy olyan multi-platform keretrendszer felhasználni a fejlesztéshez, amivel a kívánt operációs rendszereken minél nagyobb kód-újrahasznosítás érhető el, azaz elég egyszer implementálni egy funkcionálitást, mert a keretrendszer gondoskodik róla, hogy a megcélzott operációs rendszerekkel kompatibilis futtatható állományok generálódjanak.

## 2.2. Az elérhető multi-platform keretrendszerek

Jelenleg a piacon háromfajta multi-platform fejlesztési irányt különböztethetünk meg [1]. Lehetséges tisztán webes alkalmazást írni, ami a mobiltelefon webböngészőjét használja. Ez tulajdonképpen egy okostelefonokra optimalizált weblap, JavaScript, HTML5 és más webes technológiák felhasználásával készített tartalommal. Ezek a weblapok azonban sokszor olyan hibákba ütköznek, mint például a böngészőapplikáció szkriptfuttató-motorjának<sup>1</sup> hibái, illetve az operációs rendszer által a böngészőknek engedélyezett API-k<sup>2</sup> korlátozottságá.

Egy másik megközelítés a hibrid alkalmazás fejlesztése. Ekkor szintén webes technológiák használatával készül az app, de a keretrendszer biztosít egy, az operációs rendszerre natív alkalmazásfuttató-héjat. Ezzel lehetőség nyílik több hardverközelebbi funkcionálitás egyszerűbb elérésére a keretrendszer API-jain keresztül, valamint nincs feltétlenül szükség internetelérésre. A szkriptek futtatásáért a héj felel, aminek hibáit a keretrendszer-fejlesztők felé lehet jelezni, akik számára fontosabb azok javítása, mint egy böngészőalkalmazás esetében.

Harmadik lehetőség a natív applikáció készítése. A natív appok teljes egészében hozzáérhetnek az operációs rendszer által kínált funkcionálitásokhoz (termézesesen a megfelelő jogosultságok birtokában), valamint a három alkalmazásfajta közül ezek kódja fut optimálisan és leggyorsabban. A natív alkalmazásokat fordító keretrendszerek a multi-platform fejlesztés legújabb irányát képviselik, és dinamikusan fejlődnek.

A továbbiakban bemutatok - a teljesség igénye nélkül - jelenleg rendelkezésre álló multiplatform mobilsoftver-fejlesztő keretrendszereket, és megvizsgálom őket a SmartActive mobilalkalmazáshoz való alkalmasságuk szempontjából, a saját programozási ismereteim figyelembe vételével.

### 2.2.1. Sencha

A Sencha egy webalkalmazáskészítő-keretrendszer, azaz információközlésre, valamint szöveges adatbevitelre alkalmas applikáció készíthető vele. Használatához folyamatos internetelérés szükséges. Programozási nyelve a HTML5 JavaScripttel kiegészítve. Gazdag grafikus elemek megjelenítésére alkalmas a saját fejlesztésű Ext JS [17] és Sencha Touch [18] JavaScriptben írt könyvtárakkal.

Mivel a készítendő mobilapplikáció a keretrendszer által támogatott funkciókat kívánja

---

<sup>1</sup>A szkriptek futtatásáért felelős programkomponensek összessége.

<sup>2</sup>Application Programming Interface: olyan alkalmazás- vagy operációsrendszer-funkcionálitások gyűjteménye, melyek elérhetők más alkalmazások számára, pl. helymeghatározási szolgáltatás, internetelérés.

megvalósítani, és a Sencha platform az összes cél-operációsrendszer támogatja, alkalmas választás lehetne, azonban az appnak követelménye az internetkapcsolat nélküli működés is, így ez a keretrendszer nem megfelelő hozzá.

### 2.2.2. Apache Cordova (PhoneGap)

Az Apache Cordova egy hibrid alkalmazások készítésére szolgáló keretrendszer, ami nyílt forráskódú (az Apache 2.0 licenc kereteiben), de léteznek a fejlesztéshez szükséges alapvető könyvtárakon túl számos egyéb funkcionálitást (például tesztelői szolgáltatást) nyújtó, szolgáltatásként megvásárolható verziói is (pl. Adobe PhoneGap) [6]. Az applikációk készítéséhez három webprogramozásban használatos nyelvet kell igénybe venni: míg a kinézetért HTML+CSS kombinációja felel, addig a funkcionálitást JavaScriptben írhatjuk, amiben a keretrendszerben biztosított könyvtárak által elérhetjük a telefon hardveres erőforrásait is (pl. gyorsulásmérő, GPS, háttértár).

Hibrid alkalmazásokban megvalósítható a SmartActive mobilapp összes kívánt funkcionálitása, és az Apache Cordova támogatja az összes célplatformot. Az applikációt azonban mégsem ebben a keretrendszerben fogom elkészíteni, mert nem rendelkezem elég tapasztalattal a webtechnológiákban egy ilyen komplex alkalmazás elkészítéséhez.

### 2.2.3. Appcelerator

Az Appcelerator platform teljes ökoszisztémát kínál natív multi-platform alkalmazások JavaScriptben történő készítésére a saját fejlesztőszoftvertől a tesztautomatizáláson át az elkészült alkalmazás életciklus-menedzsmentjéig (lifecycle management) [10]. A natív funkciókat a Titanium SDK-val érhetjük el.

A keretrendszer támogatja az összes célplatformot, és natívan futó alkalmazásként minden funkcionálitás megvalósítható, de a JavaScript nyelv és a számomra ismeretlen fejlesztőszoftver miatt nem az Appceleratort tartom a legmegfelelőbb választásnak.

### 2.2.4. Corona

A Corona egy főként kétdimenziós játékok fejlesztésére optimalizált SDK [9]. Programozási nyelve a Lua, és az alkalmazások natívan futnak az adott operációs rendszeren. A Windows mobiltelefonos platformok támogatása nem a legújabb Windows Runtime API-kon keresztül, hanem a kifutó Silverlight technológiával történik.

Ennek a keretrendszernek nem erőssége a készítendő alkalmazástípus, valamint a Silverlight technológiás windowsplatform-támogatás és a számomra ismeretlen Lua nyelv miatt nem tartom megfelelőnek a Coronát az applikáció elkészítéséhez.

### 2.2.5. Unity

A Unity egy alapvetően játékalkalmazásokhoz készített keretrendszer, legyen az kettő- vagy háromdimenziós. Fejlesztéshez a C# vagy a JavaScript jelentősen Unityhez igazított változata, a UnityScript nyelvek használhatók [20]. A Unity az egyik legszélesebb platformlefedettséget kínáló rendszer: ugyanaz az alkalmazás megcélozhat okostelefonokat, asztali

számítógépeket, kézi- vagy otthoni játékkonzolokat, virtuális valóság headseteket, okostévéket, és mindenmellett lehet akár webböngészőben futtatható is.

Ez a keretrendszer egyértelműen nem információközlő alkalmazások számára készült. Bár a kívánt funkciók megvalósíthatóak benne, de nem tartalmaz előre elkészített vizuális elemeket (például görgethető lista), valamint mivel egy teljes videojáték-motort tölt be az alkalmazás indításakor, ezért 5-10 másodpercbe is telik, mielőtt a fejlesztő által írt kód futtatása megkezdődhet.

### 2.2.6. Kony

A Kony [16] szintén egy teljes ökoszisztémát kínáló rendszert nyújt saját fejlesztőeszközökkel, tesztautomatizálási és alkalmazáséletciklus-menedzselségi lehetőségekkel, valamint rendelésre is vállalnak alkalmazáskészítést (Kony Apps szolgáltatás). A Gartner IT piackutató és tanácsadó cég úgynevezett "Magic Quadrant" besorolásában 2015-ben sorozatban harmadjára kerültek a piacvezető kategóriába [15]. Fejlesztőszoftverükben, a Kony Visualizerben JavaScripttel támogatott HTML-ben fejleszthetünk, valamint előre elkészített grafikus elemeket "fogd és vidd" (drag and drop) módszerrel integrálhatunk az alkalmazásba. A keretrendszer képes natív, hibrid és tisztán webes alkalmazások fordítására akár ugyanabból a kódábrisból, ezzel fedve számos okostelefon- és egyéb operációs rendszert.

A Kony, bár támogatja a célplatformokat és a Visualizer is intuitívnak tűnik, annak ingyenesen elérhető Starter Edition verziója korlátozott funkcionálitást kínál csak, illetve ahogy a 2.2.2 pontban írtam, a webes technológiákban nem rendelkezem kellő ismeretekkel.

### 2.2.7. Xamarin

A Xamarin [29] szintén egy teljes szoftverfejlesztési folyamatot lefedő keretrendszer. Windowson a Microsoft Visual Studio fejlesztőszoftverbe integrálódik. Előfizethető a Xamarin TestCloud, aminek segítségével akár több ezer különböző telefonon is futtathatunk automatizált funkcionális és grafikus felhasználói felület teszteket, emellett életciklus-menedzsment eszközökkel is rendelkezik. A Xamarin University pedig rögzített előadásokkal egy újajta megközelítésben kínál bevezetést és haladóknak szánt útmutatást a Xamarin használatához. Programozási nyelve a C#, amit a Microsoft .NET platform könyvtárainak jelentős részének elérhetősége egészít ki hasznos funkciókkal. Natívan futó alkalmazások készíthetők vele Androidra, iOS-re, Windows Phone 8-ra, 8.1-re és Windows 10 Mobile-ra, valamint célozhatóak a legújabb asztali Windows és Apple OSX operációs rendszerek is. A kód-újrahasznosítás jelentősen növelte a Xamarin.Forms [30] felhasználói felület programozási könyvtárral, ha az egységes kinézet a fontosabb, de lehetőség van csak a működési logika közös kódban tartására, és mellette a cél-operációsrendszerek dizájnútmutatóinak megfelelő egy-egy külön felhasználói felület létrehozására is.

Ez a keretrendszer megfelel a fejlesztendő alkalmazás követelményeinek, a Xamarin.Forms könyvtár használatával pedig igen nagy mértékű közös kódárist látok megvalósíthatónak. A C# programozási nyelvben szerzett tapasztalataim és a Microsoft Visual Studio fejlesztőkörnyezet miatt a Xamarint választottam a SmartActive mobilapplikáció elkészítéséhez.

### 3. fejezet

## A szoftverfejlesztő infrastruktúra

Az alábbiakban ismertetem, hogy a Xamarin keretrendszerben történő munka megkezdéséhez milyen eszközök szükségesek, valamint részletezem azok helyes beállítását és használatát.

### 3.1. A Microsoft Visual Studio 2015 Update 2 telepítése és beállítása

A szakdolgozatom készítésekor a Microsoft szoftverfejlesztő programjának (IDE<sup>1</sup>) legfrissebb ingyenes (Community Edition) verziójában is megtalálható már a Xamarin keretrendszer, mint telepíthető komponens. A Community Edition regisztráció után ingyenesen letölthető a Microsofttól, én az alkalmazás Enterprise verzióját telepítettem a BME DreamSpark programjának keretében szerzett licenc-szel.

Mivel a fejlesztendő alkalmazás a Windows 10 Mobile operációs rendszert is célozza, ezért követelmény, hogy a fejlesztőkörnyezetet futtató számítógépen a Windows 10 operációs rendszer Education, Professional vagy Enterprise kiadása legyen telepítve (a kiadások megkötése a Hyper-V technológia támogatása miatt fontos, lásd 3.2.1 pont), lehetőleg az összes elérhető rendszerfrissítéssel. A Visual Studiohoz telepítendő komponensek helyes kiválasztásának legegyszerűbb módja, ha a telepítés típusának a *Custom* módot választjuk, majd az összes komponensből kivesszük a kijelölést. Ezután a *Cross Platform Mobile Development* pontban kiválasztjuk a *C#/.NET (Xamarin v4.0.3)* pontot, ami automatikusan újra kijelöli a számára szükséges egyéb komponenseket. A Windows Phone 8.1 és Windows 10 Mobile cél-operációsrendszerek miatt nyissuk le a *Windows and Web Development* pontot, majd a *Universal Windows App Development Tools* és *Windows 8.1 and Windows Phone 8.0/8.1 Tools* pontokban lévő teli négyzetet módosítsuk pipára, ezzel telepítve a szükséges SDK-kat<sup>2</sup> és emulátorokat (lásd 3.2 pont). A helyes beállításokról készült képernyőképek a függelék F.1 pontjában találhatók.

Nagyon fontos figyelembe venni, hogy hiába választunk az IDE telepítési helyének a rendszermeghajtótól különböző meghajtót, az így is jelentős helyet fog elfoglalni ott, mivel a

<sup>1</sup>Integrated Development Environment: olyan szoftverfejlesztő környezet, ami egy alkalmazásként nyújtja a fejlesztéshez szükséges komponenseket (pl. szövegszerkesztő, fordító, hibakereső), valamint többletszolgáltatásokat (pl. automatikus kiegészítés) is nyújthat.

<sup>2</sup>Software Development Kit: Adott platformot, keretrendszer vagy szolgáltatást célzó program fejlesztéséhez szükséges függvénykönyvtárak és eszközök.

különböző SDK-k, emulátorok, és más közös komponensek mindenkorábban az alapértelmezett helyre települnek, mint például a C:\Program Files vagy a C:\Users\<felhasználónév>\AppData mappa. Tapasztalataim alapján ajánlatos legalább 50 GB szabad helyet rendelkezni a rendszermeghajtón a telepítés megkezdése előtt, különben akár végzetes rendszerhibák is felléphetnek.

A programot települése után nyissuk meg, majd a <https://www.xamarin.com/student> oldalon létrehozott felhasználói fiókkal jelentkezzünk be a Tools/Xamarin Account menüben, ezzel aktiválva a keretrendszeret. Ezután a telepített kiegészítők naprakészségét kell biztosítani a Tools/Extensions and Updates/Updates menü használatával.

Fontos engedélyezni a számítógépen lévő Windows PowerShell parancssor beállításaiban a távoli aláírt szkriptek futtatását, mert a Xamarin keretrendszer használ PowerShellben írt szkripteket. Indítsunk egy PowerShell parancsot rendszergazdai jogosultságokkal. Ha a Get-ExecutionPolicy -list parancsot futtatva a CurrentUser vagy LocalMachine szkópokhoz nem RemoteSigned jogosultságot látunk beállítva, módosítsuk azt a Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope <adott\_szkóp\_neve> parancssal.

### 3.1.1. Programfordítás iOS operációs rendszerre

Az Apple cég híresen zárt rendszerekkel dolgozik. Fejlesztési szemszögből ez azt jelenti, hogy Apple terméket célzó programot csak az Apple OSX operációs rendszeren futó Xcode programcsomaggal lehet fordítani. A Xamarin fejlesztői ügyesen áthidalta ezt a problémát: lehetőség van ugyanis egy távoli, OSX operációs rendszert futtató számítógépet úgy felkonfigurálni, hogy a Visual Studioba integrált Xamarin eszközök csatlakozni tudjanak hozzá, és a saját IDE-nkben fejlesztett kódot ott lefordítsuk, illetve az ott futó emulátorban (lásd 3.2 pont) távolról futtassuk az appot hibakeresési céllal.

A fejlesztéshez konzulensem biztosított egy, a tanszéken felkonfigurált Mac Minit. A pontos konfigurációs lépésekkről a Xamarin keretrendszer fejlesztőknek szóló dokumentációjában található további információ [22]. Fontos, hogy a megfelelő beállítások elvégzéséhez mindenkorábban rendszergazdai jogosultságokkal rendelkező fiókra van szükség, azonban a beállításkor megadhatunk nem rendszergazda felhasználót is a kapcsolódáshoz, én egy ilyen felhasználón keresztül dolgoztam.

Mivel a fejlesztett alkalmazás emulátorbeli futtatása csak a távoli gépen oldható meg, ezért szükség van valamilyen grafikus elérést biztosító távoli kapcsolódást nyújtó program használatára is. Én kezdetben a TeamViewer<sup>3</sup> programot használtam, azonban stabilitási problémák miatt áttértem a RealVNC<sup>4</sup> csomagra.

---

<sup>3</sup>További információ a TeamViewer termékről: <http://www.teamviewer.com/>

<sup>4</sup>További információ a RealVNC termékről: <https://www.realvnc.com/>

### **3.1.2. További beállítások a SmartActive mobilapplikáció fordításához**

A fordításhoz szükséges külső csomagok jelentős része NuGet<sup>5</sup> csomagként van referálva, azonban bizonyos komponenseket külön kell telepíteni.

A Visual Studio *Tools/Extensions and Updates/Online/Visual Studio Gallery* menüpontból szerezhető be a tesztek létrehozását segítő "NUnit Templates for Visual Studio" kiegészítő, valamint az alkalmazás Windows Phone 8.1-re és Windows 10 Mobile-ra fordításához szükséges *SQLite for Windows Phone 8.1* és *SQLite for Universal Windows Platform* csomagok. Az alkalmazás jelenleg 3.12.2-es verziójú SQLite csomagokat használ, ha csak újabb csomag érhető el a galériában, akkor telepítése után az adott platformspecifikus projektben frissíteni kell a referenciát.

Az appban látható diagramokhoz szükség van a Syncfusion Essential Studio for Xamarin termék legalább 14.1.0.41-es verziójának telepítésére, amihez közösségi licenc keretein belül ingyenesen hozzájuthatunk [19].

A Visual Studio kódkiegészítő- és javaslattevő funkciójának bővítéséhez, valamint a unit tesztek (lásd 4.5.1 pont) futtatásához a ReSharper Ultimate programcsomag *ReSharper* és *dotCover* komponenseit kell telepíteni, ezeket a <https://www.jetbrains.com/dotnet/> oldalról lehet elérni.

## **3.2. Az okostelefon-emulátorok**

Az emulátor egy olyan program, aminek célja egy olyan virtuális hardverkörnyezet biztosítása, ami működésében és funkcionalitásában megegyezik egy valódi hardverkörnyezettel (emulálja azt), lehetővé téve akár egy teljes operációs rendszer futtatását benne. Mobilsoftver-fejlesztésben az emulátorok használata kiemelkedő jelentőséggel bír, mert így nem kell az összes célplatformhoz és hardvertulajdonsághoz valódi eszközökkel rendelkeznünk. Fontos tudni, hogy az emulátor a futása során legalább annyi rendszermemóriát (RAM) foglal le, mint az emulált eszköz memóriája, így több emulátor egyszerre futtatása esetére érdemes kiszámolni, mennyi memóriával kell rendelkezzen a gazdaszámítógép<sup>6</sup>. (Én 16 GB-ra bővítettem, hogy egyszerre futtathassak az összes célplatform emulátoraiból egyet-egyet.)

### **3.2.1. Az emulátorok használata**

A Visual Studio képes többfajta emulátorral együttműködni, azonban én az Android, Windows Phone 8.1 és Windows 10 Mobile rendszerekhez a Microsoft saját emulátorait használtam. Ezek támogatják a Hyper-V technológiát, amivel megfelelő processzorral és operációs rendszerrel rendelkező gazdaszámítógépen lényegesen egyenletesebb és nagyobb teljesítményt nyújtanak, mint a szintén kipróbált Google vagy Xamarin keretrendszerrel érkező emulátorok. Ezeket legegyszerűbben úgy indíthatjuk el, ha a már létrehozott Visual

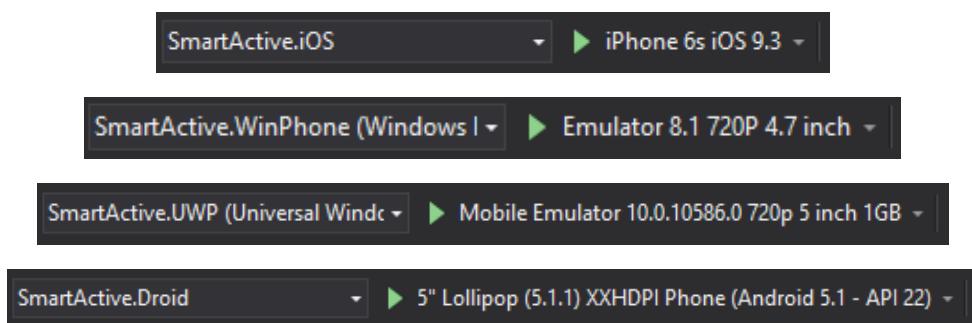
<sup>5</sup>A NuGet egy csomagkezelő szolgáltatás a Visual Studiohoz. Az így közzétett programkönyvtárakat ugyanúgy referálhatjuk, mint a keretrendszer sajátjait, azonban nem kell telepíteni őket, hanem fordításkor automatikusan letöltsük. Lehetőség van különböző projektekben ugyanannak a csomagnak más verzióját használni.

<sup>6</sup>Az emulátorokat futtató fizikai számítógép, angolul host machine.

Studio Solutionben a felső eszköztáron kiválasztjuk a célplatform-specifikus projektet (3.1. ábra), választunk hozzá egy emulátort, majd a mellette lévő zöld háromszögre kattintva a kód lefordul, bekapcsol az emulátor, települ az applikáció, ezután elindul a futtatás közbeni hibakeresés (debuggolás). Az emulátor a debuggolás végeztével is bekapcsolva marad, ezzel időt takarítva meg a későbbi hibakeresési folyamatok megkezdéséhez. Az alkalmazást hibakeresés indítása nélkül is telepíthetjük az emulátorokra. Miután kiválasztottuk az adott cél-operációsrendszerhez használandó emulátort, kattintsunk jobb egérgombbal a platformspecifikus projektre, majd válasszuk a *Deploy* lehetőséget. A művelet befejeztével az alkalmazás megjelenik az emulátor telepített programjai között, onnan indítható.

Egy másik lehetőség a Visual Studio Emulator for Android és Windows Phone Developer Power Tools 8.1 programok használata (ábrák a függelék F.2 pontjában). Ezekkel a programokkal a Visual Studiotól függetlenül indíthatjuk el az emulátorokat, telepíthetünk továbbiakat, illetve részletesebb információkat kaphatunk róluk.

Ahogy azt korábban említettem (3.1.1 pont), az iOS-es hibakeresést a távoli, OSX operációs rendszerű gépen kell végezni. Az ott futó emulátorprogram neve *Simulator*, amiben az emulált eszköz fizikai gombjai nem látszódnak, azok megnyomását billentyűkombinációkkal vagy a menüsor *Hardware* pontjában lehet kiváltani.



**3.1. ábra.** Példák emulátorok beállítására különböző platformokon történő futtatás közbeni hibakereséshez

### 3.2.2. Interakció az emulátorok fájlrendszerével

Mivel a SmartActive mobilalkalmazás képes a szerverről letöltött adatok alkalmazás-futások közötti tárolására a telefonon, ezért ismertetem, hogyan lehet a Visual Studio emulátorok és az Apple Simulator tárhelyére és tárhelyéről adatokat másolni. Az androidos és windows okostelefonos fájlműveletek megkönnyítésére és felgyorsítására írtam egy PowerShell szkriptet.

#### Android

A Visual Studio Android emulátorai ugyanazokkal a képességekkel rendelkeznek fájlerés tekintetében, mint a Google sajátjai. Az Android SDK-val érkező *adb.exe*<sup>7</sup> programot

<sup>7</sup>Alapértelmezett útvonala 64 bites operációs rendszeren: C:\Program Files (x86)\Android\android-sdk\platform-tools\adb.exe.

a *pull <forrásfájl az emulátoron> <célfájl a számítógépen>* parancssal hívva az emulátorról a számítógépre, a *push <forrásfájl a számítógépen> <célfájl az emulátoron>* parancssal pedig fordított irányba másolhatunk fájlokat.

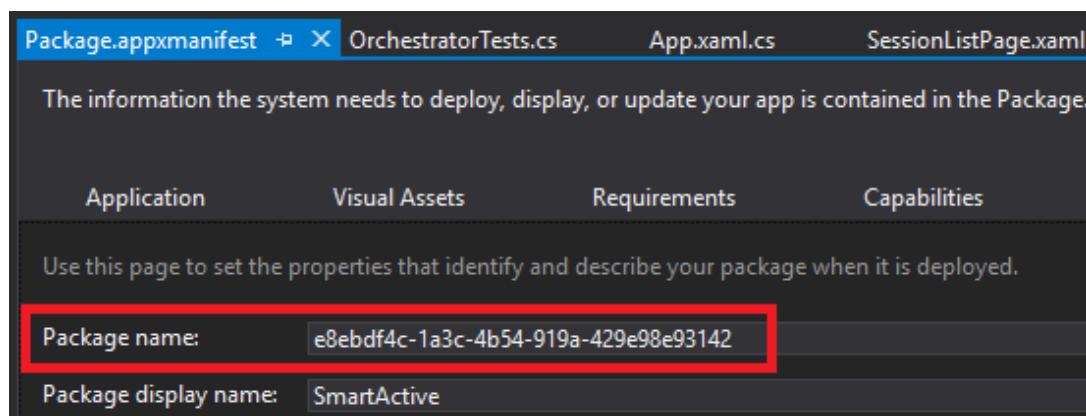
Lehetőség van a *shell* parancs kiadásával egy olyan, a linux shell parancsok részhalmazát, valamint egyéb, androidos fejlesztéshez használatos parancsokat elfogadó parancssor megnyitására, ami az emulátor fájlrendszerének gyökerében áll. Innen például a *cd /data/data/SmartActive.SmartActive/files* parancssal a telepített mobilalkalmazás saját fájljait tartalmazó mappába navigálhatunk, ahol - az app első futtatása után - listázható a szerverrel szinkronizált adatokat tároló *SmartActiveSQLite.db3* fájl az ismert *ls* parancs kiadásával.

Több Android emulátor egyszerre történő futtatása esetén először a *devices* parancssal listázzuk ki őket, majd a használni kívánt emulátor sorozatszámát a parancsokat megelőző *-s* kapcsolóval adjuk meg.

## Windows Phone 8.1 és Windows 10 Mobile

Ezeknek az emulátoroknak a tárhelyét a Windows 8.1 SDK-val települt Isolated Storage Explorer Tool<sup>8</sup> parancssori programmal lehet elérni. Az adb-vel szemben itt a teljes fájlrendszeren nem, csak az adott alkalmazás saját, izolált tárhelyén lehet műveleteket végezni. A teljes izolált mappa számítógépre másolása a *ts*, míg a mappa lecserélése egy, a számítógépen található mappára az *rs* parancssal történik. A két művelet paraméterei hasonlóak: elsőként a használni kívánt emulátor azonosítóját kell megadni, amit az *EnumerateDevices* parancssal kérdezhetünk le, majd az alkalmazás csomagazonosítója, végül parancstól függően a cél- vagy forrásmappa következik. A csomagazonosítót a platform-specifikus projekt *Package.appxmanifest* fájljában találjuk (3.2. ábra). Például az alábbi parancs a négyes számú emulátorra telepített SmartActive applikáció izolált mappáját a jelenlegi könyvtárban található *WinPhone* nevű mappára cseréli le:

```
ISETool.exe rs deviceindex:4 e8ebdf4c-1a3c-4b54-919a-429e98e93142 ".\WinPhone"
```



**3.2. ábra.** A mobilapp Windows Phone 8.1 platformspecifikus projektjének csomagazonosítója

<sup>8</sup>Alapértelmezett útvonala 64 bites operációs rendszeren: C:\Program Files (x86)\Microsoft SDKs\Windows Phone\v8.1\Tools\IsolatedStorageExplorerTool\ISETool.exe.

## iOS

A távoli Mac-en található Simulator fájlrendszerének eléréséhez indítsuk el legalább egyszer az appot a kívánt emulátoron, majd lépjünk a `/Users/<felhasználó_név>/Library/Developer/CoreSimulator/Devices` mappába (a Library mappa rejtett). Itt keressük rá a módosítani kívánt fájatra, mert amíg az emulátor egyedi azonosítóját (GUID) megnézhetjük a `device_set.plist` fájlban, addig a telepített applikáció GUID-ját már nincs mód megtudni. Az alkalmazás fájljainak mappája a `./<GUID1>/data/Containers/Data/Application/<GUID2>/Library` mintára illeszkedik.

## 4. fejezet

# A mobilapplikáció fejlesztése és tesztelése

A következőkben megvizsgálom a Xamarin keretrendszer fejlesztési lehetőségeit. Ezután kitérek a fejlesztés során használt főbb programozási paradigmákra, majd ismertetem az alkalmazáslogika struktúráját, és az annak implementálása során felmerült kihívásokat. Bemutatom a felhasználói felület készítésekor használt Xamarin.Forms könyvtár lényegesebb elemeit, valamint azon hibáikat, amikkel a fejlesztés során találkoztam. Végül kitérek arra, milyen módokon teszteltem az applikációt az elkészítése során.

### 4.1. A Xamarin multi-platform fejlesztési lehetőségei

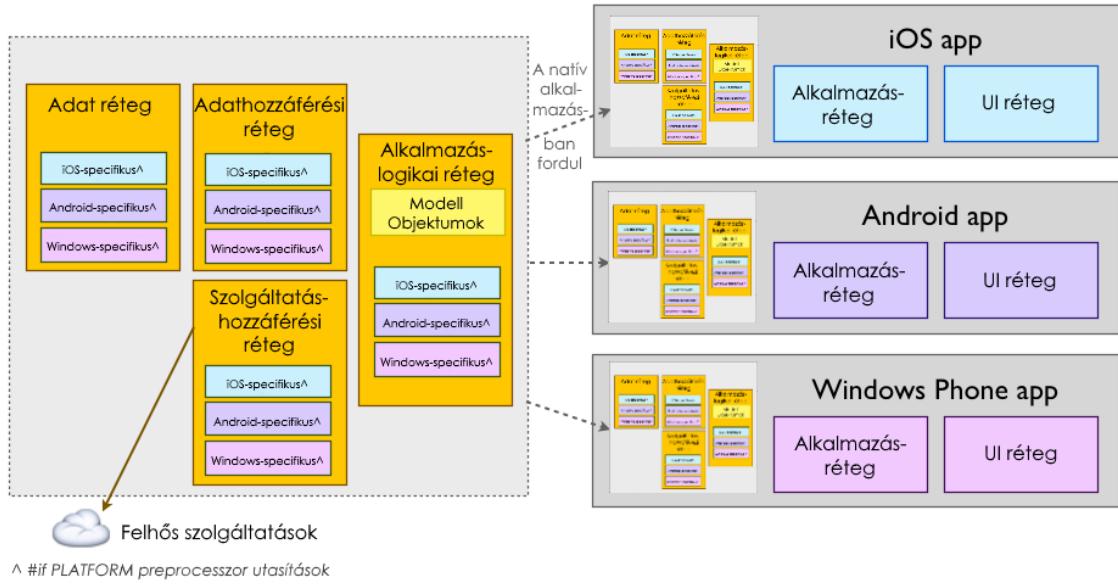
A multi-platform kód elkészítésére a keretrendszer két különböző megközelítést kínál: a megosztott projekteket (shared project) [24] és a hordozható osztálykönyvtárakat (portable class library, PCL) [23].

Megosztott projekt használatakor a közös funkciók mellett a platformspecifikus kód részletek is ugyanabban a forrásfájlban kapnak helyet. A C# fordító preprocesszora számára `#if PLATFORM ... #endif` utasításokkal jelezük, melyik cél-operációsrendszeren milyen kódot kell futtatnia. Az így körülvett kód részben használhatjuk az adott platform specifikus könyvtárait is. Adott platformra fordításkor csak az ahhoz tartozó kód kerül beépítésre a végleges alkalmazáscsomagba (4.1. ábra). Ennek a megközelítésnek hátránya, hogy a forrásfájl nehezen olvashatóvá válhat.

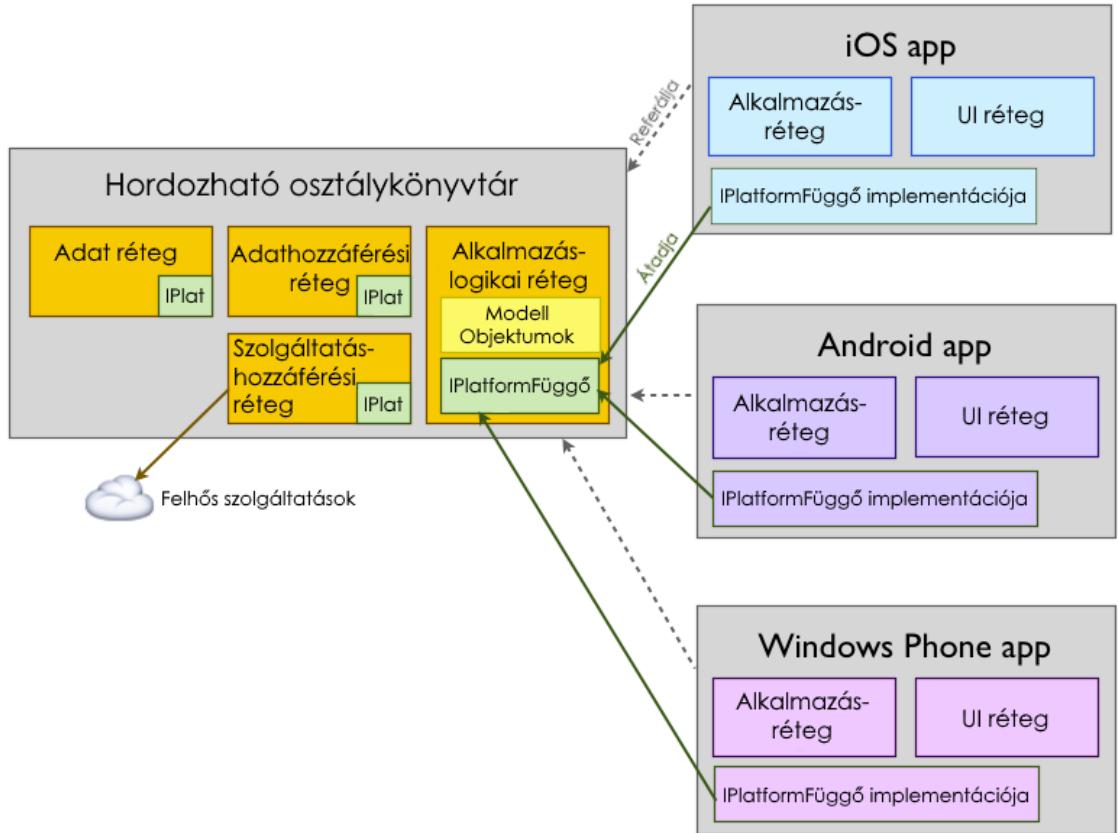
Hordozható osztálykönyvtár esetén a közös kód egy külön DLL-ben<sup>1</sup> kap helyet. Ekkor a megosztott kódban nem használhatunk egyedi platformkönyvtárakat, hanem interfésekkel definiálunk a platformspecifikus funkciók elérésére, amiket a cél-operációsrendszerek saját projektjeiben valósítunk meg (4.2. ábra).

---

<sup>1</sup>Dynamic Link Library, dinamikus csatolású könyvtár: olyan alkalmazás-segédfájl, ami többek között eljárásokat, kompatibilitást segítő eszközöket vagy úgynevezett "erőforrásokat" (pl. képeket, hangokat, lokalizált sztringeket) tárolhat.



**4.1. ábra.** Megosztott projektet használó alkalmazás struktúrája [21]



**4.2. ábra.** Hordozható osztálykönyvtárat használó alkalmazás struktúrája [21]

A közös kódban a Xamarin *DependencyService* osztályának használatával hívhatjuk meg az operációsrendszer-specifikus függvényeket. Az alábbi kódrészlet egy SQLite adatbázis megnyitását mutatja az *ISQLite* interfész használatával:

```
var SQLiteDb = DependencyService.Get<ISQLite>().GetConnection();
```

A *Device* osztály nyújt funkciókat a programot futtató eszközről információ szerzésére, mint például a platform (Android, iOS vagy Windows Phone), az idióma (asztali számítógép, telefon vagy tablet) vagy a nevesített betűtípusokhoz (micro, kicsi, közepes, nagy, alapértelmezett) tartozó számszerűsített betűtímet.

## 4.2. A fejlesztés során használt főbb programozási paradigmák

### 4.2.1. Az MVVM

Az MVVM egy szoftverarchitektúra-tervezési minta, feloldása "Model-View-View Model". Ez a tervezési minta az MVC (Model-View-Control) minta egy módosított változata, amit a Microsoft fejlesztői a .NET keretrendszer grafikus rendszerének (WPF<sup>2</sup>) figyelembe vételével készítettek az eseményalapú felhasználói felület fejlesztésének megkönnyítésére [34].

A *View* komponens határozza meg a grafikus felület struktúráját, kiosztását és megjelenését. Üzleti logikát nem tartalmaz, csak a felhasználói interakciók (például kattintás) értelmezése történik, majd az eseményt közli a *View Model*lel.

A *Model* feladata az adatok reprezentálása, valamint a validációs- és üzleti logika megvalósítása. Itt történik a kommunikáció az alkalmazáson kívüli komponensekkel is, például az adatok szinkronizálása egy távoli szerverrel.

A *View Model* az összekötő kapocs a *Model* és a *View* között. Feladata, hogy a *Model* által nyújtott adatokat a *View* számára alkalmas formára hozza, illetve a felhasználói interakciók hatását érvényesítse a *Modelben*. Ezek mellett a *View* állapotgépeként is szolgál, például hosszan tartó műveletek esetén utasíthatja a *View-t*, hogy egy töltőképernyőt jelenítsen meg.

### 4.2.2. C# grafikus elemek leírása XAML-ben, adatkötés

A C#-ban írt grafikus felületek struktúrájának átláthatóságára a Microsoft kifejlesztette a XAML-t<sup>3</sup>, ami egy olyan, XML alapú leíró nyelv, ami az objektumok inicializálásán és tagváltóinak beállításán kívül könnyen láthatóvá teszi az objektumok közötti hierarchiát [35]. Lehetőség van a beépített típusokon túl további típusokkal és bővítményekkel kiegészíteni az alkalmazásunk igényeinek megfelelően, akárcsak a szokványos C# kódot.

Egy XAML nézetleíró két részből áll: magából a leírásból és a hozzá tartozó mögöttes kódból (Code-Behind). A mögöttes kódból elérhetőek a XAML leírásban deklarált objektumok, és fordítva, ezzel meghagyva a fejlesztő szabadságát, hogy a különböző tulajdonságok beállítását melyik nyelv használatával végzi el. A mögöttes kód feladata az eseménykezelők megvalósítása, viszont minél kevesebb alkalmazáslogika kerüljön ide.

A *View Model* által megjelenítésre alkalmas (vagy ahhoz nagyon közel) formára hozott adatokat a grafikus elemekhez rendeljük, úgynevezett adatkötést (data binding) végzünk. A kötés típusa Xamarinban lehet egyirányú az adattól a megjelenített elem irányába, vagy

<sup>2</sup>A Windows Presentation Foundation egy egységesített programozási modell grafikus alkalmazások készítésére C#/.NET-ben.

<sup>3</sup>Extensible Application Markup Language, azaz bővíthető alkalmazásleíró nyelv.

vissza, illetve kétirányú, ekkor az adat frissülhet a View Model és a View irányából is. Ha a View-beli elem adatforrása nem a View-t megvalósító osztály tagváltozója, akkor Xamarinban az elem *BindingContext* (kötési kontextus) tulajdonságának értékül kell adni azt az objektumot, amiben a kötésre szánt adat található. A kötési kontextus automatikusan öröklődik, azaz ha a View-t implementáló osztály *BindingContextjének* adjuk értékül (tipikusan) a View Modelt, akkor az összes benne elhelyezett elem is ugyanazt a kötési kontextust kapja, kivéve, ha felüldefiniáljuk azt. Ezután az adatkötésnél már csak a kötésikontextus-osztály tagjaira hivatkozunk, amiknek publikus láthatóságúnak kell lenniük.

Ahhoz, hogy a kötött adat frissítéséről a View értesítést kapjon, és frissítse a megjelenített elemet, a View Modelnek implementálnia kell a *System.ComponentModel.INotifyPropertyChanged* interfész, valamint a kötött adat változásakor meg kell hívnia saját *OnPropertyChanged* implementációját. A függelék F.3 pontjának kód részletei a mobilapp barátkeresés funkcióján keresztül szemléltetik az adatkötést. Mivel a keresősáv egy beviteli mező, ezért kötése a *SearchFilter* taggal automatikusan kétirányú, így a megfelelő *set* eljárás használatával kényelmi funkcióként már a keresőszó megadása alatt, minden karakter bevitellel után lefut egy szűrés a barátlistára.

#### 4.2.3. Aszinkron mechanizmusok

A grafikus felülettel rendelkező alkalmazásoknál kiemelkedően fontos, hogy a fő programszalon, ami a felhasználói interfész (angolul user interface, UI) megjelenítéséért is felel, a lehető legkevesebb feladatot futtassuk. Ezzel biztosítjuk a felület reszponzivitását, azaz az animációk folyamatosságát és a felhasználói input feldolgozásának gyors megkezdését.

A .NET keretrendszer a 4-es verziótól kezdve rendelkezik egy feladat-párhuzamosítási könyvtárral (Task Parallel Library, TPL), ami a korábban is létező párhuzamosítási mechanizmusokat egy erőforrás-hatékonyabb, jobban skálázódó és nagyobb kontrollt nyújtó keretbe fogja össze. Ennek alapegysége a feladat (Task osztály), amin többek között a szokásos szálkezelési műveleteket is végrehajthatjuk (például indítás, várakozás, megszakítás, a visszatérési érték és kilépési kód lekérdezése).

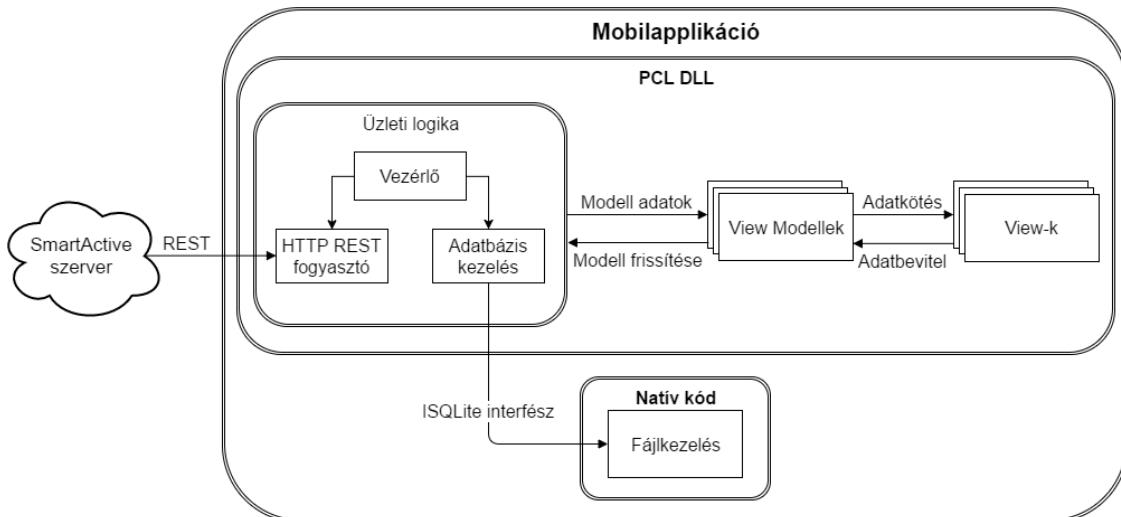
Az egyértelműen időigényes műveleteket - például fájl- és adatbázis-műveletek, kommunikáció más folyamatokkal és szolgáltatásokkal vagy internethozzáférések - mindenkorban érdemes háttérszálakon futtatni. Ha a kézi tesztelés során azt tapasztaljuk, hogy egy művelet érezhető ideig blokkolja a fő szálat, azt is futtathatjuk taszkaként. Kritikus azonban, hogy az adatkötött változók értékmódosítását mindenkorban a fő szálban végezzük, mert az adatkötés nem szálbiztos mechanizmus, és csak ezzel biztosítható az új adatok megjelenése a felületen (lásd a 4.4.2 pont *ObservableCollection* használatáról szóló részét). A fő szál hívását Xamarinban a *Device.BeginInvokeOnMainThread()* függvényteljesítővel tehetjük meg, azonban erre a taszkra nem lehet várakozni, ezért célszerű a műveleteket úgy ütemezni, hogy a kötések frissítése legyen az utolsó. Ajánlatos továbbá, hogy a fő szálban logikai műveletet már ne, hanem csak az értékkadást végezzük, ezzel is rövidítve annak blokkolási idejét.

### 4.3. Az alkalmazáslogika

Az alkalmazás elkészítéséhez a PCL megközelítést választottam, mert számonra az interfések használata tisztább és átláthatóbb, mint az egy forrásfájlban preprocesszor-utasításokkal tagolt részek.

Az alkalmazás felhasználófelület-rétegének fejlesztési lehetőségeit a 2.2.7 pontban ismertettem. A platformonként egyedi, ottani dizájn-irányelveknek teljesen megfelelő felhasználói felületek elkészítését a szakdolgozat keretein túlmutató feladatnak tekintettük konzulensemmel. A Xamarin.Forms multi-platform felületprogramozási könyvtár használata mellett döntöttünk, ami egységesebb kinézetet nyújt, és a hordozható osztálykönyvtár DLL-ben foglal helyet.

A 4.3. ábrán látható az alkalmazás struktúrája, amit az MVVM tervezési minta (4.2.1 pont) figyelembe vételével alkottam meg. A következőkben ismertetem az alkalmazáslogika egyes részeinek működését, a felhasznált könyvtárakat, és a felmerült kihívásokat a megvalósítás során.



4.3. ábra. A mobilalkalmazás struktúrája

#### 4.3.1. Kommunikáció a webszerverrel

A távoli szerverrel történő üzenetváltáshoz HTTP REST<sup>4</sup> üzeneteket használ az alkalmazás. A szerver a szinkronizálandó adatokat JSON<sup>5</sup> formátumban küldi át. Ennek feldolgozására a .NET keretrendszer nyújt egy könyvtárat, de én a több lehetőséget kínáló Newtonsoft.Json.NET-et [36] választottam. Ebben ugyanis ha rendelkezünk egy C# osztálytal, aminek a struktúrája és tagváltozónevei megegyeznek a JSON struktúrájával és

<sup>4</sup>A Representational State Transfer architektúrában a HTTP webprotokoll URI (Uniform Resource Identifier, egy rövid karakterszorozat, amelyet egy webes erőforrás azonosítására használunk, pl. egy honlapcím) elemét kihasználva lehet kommunikálni. Megfelelően felépített URI-k használatával például a GET üzenet lekérdezésre, a POST adatküldésre használható.

<sup>5</sup>A JavaScript Object Notation egy egyszerűen felépített adatcserére használható formátum. Alapegyése az objektum, ami egy kulcs-érték párból áll. A kulcs minden egy sztring. Az érték lehet objektum, értékeket tartalmazó tömb, sztring, szám, igaz-hamis érték vagy nullelem [37].

kulcssztringjeivel, akkor a rendszer képes azonnal deserializálni azt. A kulcssztring - tagváltozónév különbségek feloldhatók a tagváltozók megfelelő attribútumokkal történő ellátásával.

#### 4.3.2. Az adatok perzisztálása

A szinkronizált adatok helyi tárolását először a .NET keretrendszer szerializálási eljárásával akartam megvalósítani, azonban kiderült, hogy erre hordozható osztálykönyvtárban nincs lehetőség. Megoldásként egy SQLite adatbázis létrehozását választottam.

Az SQLite egy egyszerű felépítésű, szerver-kliens kialakítást nem igénylő, kevés külső függőséggel rendelkező, akár előzetes konfigurálás nélkül is használható tranzakcionális SQL adatbázismotor [8]. A teljes adatbázist egy fájl tárolja, ami tartalmazza a sémát, a kéyszereket és a rekordokat is. Ez kiválóan alkalmassá teszi beágyazott és mobileszközökben történő használatra. További érv a választás mellett, hogy az Android és iOS operációs rendszerekben integrálva van az SQLite adatbázismotor, így azt csak a Microsoft mobil operációs rendszereihez kell külön referálni (beszerzésüket lásd a 3.1.2 pontban).

Többféle NuGet csomag kínál programkönyvtárat SQLite adatbázisok .NET-ben történő multi-platform menedzseléshez. Először a kapcsolódó Xamarin útmutatóban [25] használt *SQLite-net PCL* [32] csomagot telepítettem. Ennek használatát körülmenyesnek találtam a kapcsolatok számosságának megadása szempontjából, mert megkövetelte, hogy az adatokat reprezentáló osztályaimat adatbázisbeli azonosítókat tároló tagváltozókkal egészítsem ki. Ezen kívül stabilitási problémákba ütköztem az olyan tagváltozók kezelésekor, amik típusa nem érték, hanem referenciatípus, például egy saját osztály.

Megoldásként áttértem a hasonló nevű *SQLite.Net-PCL* [31] csomag aszinkron verziójára. Ez az előző projekt elágaztatottja (angolul fork), és célja a kódminőség javítása, valamint annak a legújabb multi-platform technológiákhoz igazítása, mint például a hordozható osztálykönyvtárak. Ezt a kapcsolatszámosság kezelését megkönnyítő *SQLite-Net Extensions* [5] csomaggal egészítettem ki, aminél csak az objektum saját azonosítóját kell érték típusú tagváltozóban tárolni. A kapcsolatokat a fűződő osztály egy példánya (egy számosság) vagy egy azokból készített lista (több számosság) reprezentálja, csökkentve ezzel az adatbázis-lekérdezések számát. A csomag képes a körkörös referenciai kezelésére is.

Az adatbázisfájl kezeléséhez a 4.1 pontban említett DependencyService metodikát használtam. Az *ISQLite* interfész egy függvényből áll, ami egy platformspecifikus *SQLiteAsyncConnection* osztálypéldánnal tér vissza a natív kódból. A további műveletek már végezhetők ezen a példányon a hordozható osztálykönyvtárban.

#### 4.3.3. Az üzleti logika vezérlése

A vezérlő komponens feladata View Model felől érkező kérések kiszolgálása. A magasabb absztrakciójú műveletek itt kerültek megvalósításra az alkalmazáslogika egyéb komponenseinek használatával, mint például a felhasználó hitelesítése, vagy az adatok szinkronizálása a webszerverrel és ez alapján a helyi adatbázis frissítése.

A 4.2.3 pontban ismertetettek alapján a vezérlés kizárolag aszinkron műveletekkel dol-

gozik, ezzel megőrizve a felhasználói felület reszponzivitását.

#### 4.4. A felhasználói felület elkészítése Xamarin.Forms-ban

A Xamarin.Forms egy olyan osztálykönyvtár, ami az egyes platformok natív megjelenési elemei (pl. gombok, listák) köré egy úgynevezett csomagolót (angolul wrapper) tesz. Ezzel struktúrailag egységes felhasználói felületet hozhatunk létre úgy, hogy az egyes UI elemek a platform sajátjaiként néznek ki.

##### 4.4.1. Alkalmazásoldalak

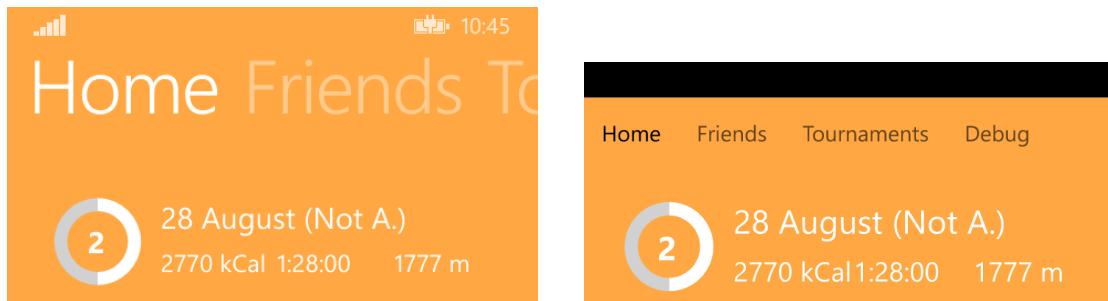
Xamarin.Formsban az alkalmazás megjelenési fő egysége az oldal, amik között kontextusmerő navigációval válthatunk. Utóbbi jelentése, hogy nem rendelkezik minden oldal saját navigációs objektummal, hanem az app főoldalának navigációs objektumát érjük el minden más oldalból is, és manipulálhatjuk annak navigációs vermét (angolul navigation stack), ezzel szabályozva például azt, hogy a vissza gomb megnyomásával milyen oldalra jutunk.

A könyvtár 6 előre elkészített alkalmazásoldal-típust nyújt [27]:

- *ContentPage*: Egy üres oldal, amibe egy darab Xamarin View elemet (lásd 4.4.2 pont) helyezhetünk. Ez tipikusan valamelyen elrendezési konténer.
- *TemplatedPage*: A legegyszerűbb oldaltípus. Tartalom közvetlenül nem helyezhető bele, hanem egy úgynevezett *ControlTemplate* sablonnal adhatjuk meg az oldal alapvető kiosztását. Ez az osztály a ContentPage ősosztálya is.
- *TabbedPage*: Füleket megjelenítő oldal, amiben a fülek oldal objektumokat tartalmaznak.
- *MasterDetailPage*: Az információt két panelen jeleníti meg. A fő panelen egy elemet kiválasztva láthatóvá válik - balról beúszik - az elemhez tartozó részleteket mutató panel.
- *CarouselPage*: Aloldalakat tartalmaz, amik között elhúzás gesztussal válthatunk, mint például egy fotógaléria alkalmazásban. A TabbedPage-dzsel ellentétben itt az aloldalak nevei nem látszódnak.
- *NavigationPage*: Ez az oldaltípus nem jelenít meg tartalmat, feladata a navigációs verem kezelése.

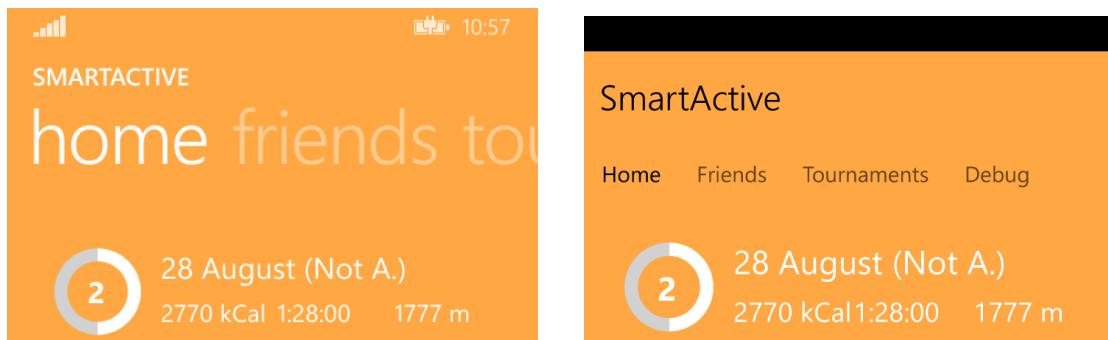
Az 1.2 pontban bemutatott koncepciórajzoknak megfelelően az applikáció elkészítése során a NavigationPage, ContentPage és TabbedPage oldaltípusokat használtam.

A felhasználói felület implementálása során a TabbedPage megjelenésével többször is hibákba ütköztem. A munka kezdetekori legfrissebb Xamarin NuGet verzió, a 2.1.0.6529-es rendelkezik egy hibával, ami miatt a Microsoft mobil operációs rendszereken a TabbedPage oldalcíme hiányzik, csak a fülek nevei jelennek meg (4.4. ábra).



**4.4. ábra.** Hiányzó TabbedPage oldalcím Windows Phone 8.1 és Windows 10 Mobile rendszereken

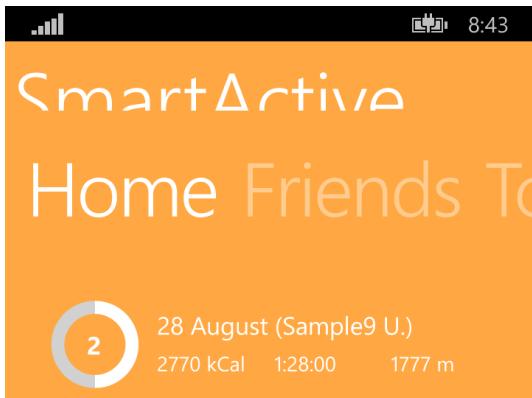
A probléma megoldására először átállítottam a platformspecifikus projektekben a Xamarin NuGet verziót 2.0.1.6505-re, ezzel a címprobléma megszűnt (4.5. ábra).



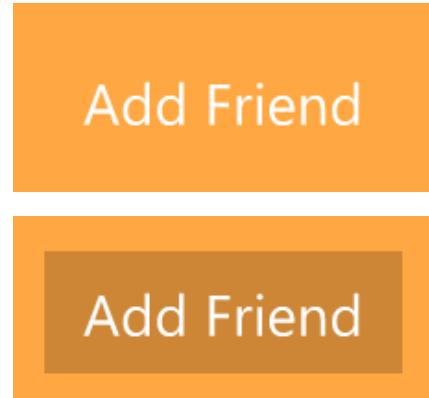
**4.5. ábra.** Helyes TabbedPage oldalcím Windows Phone 8.1 és Windows 10 Mobile rendszereken

Kézi teszteléskor azonban azt tapasztaltam, hogy Windows Phone 8.1-en ez a keretrendszer-verzió hibát tartalmaz a keresősáv elemben, a keresőszó helyett üres sztringet ad vissza lekérdezéskor. Az időközben kiadott 2.2.0.31-es Xamarin verzió változtatásai között szerepel a TabbedPage oldalcímének újbóli bevezetése a két platformon, így azokat frissítettem. Ekkor a keresősáv újra megfelelően működött Windows Phone 8.1-en, viszont az oldalcím betűtípusa túl nagy lett (4.6. ábra), de úgy döntöttem, a működő keresősáv fontosabb, ezért ennél a verziónál maradtam. Windows Mobile 10-en a frissítéstől eltűnt a gombok kerete (4.7. ábra). A gombkeret explicit meghatározásával sem sikerült helyre állítanom azokat, így végül visszaálltam a 2.0.1.6505-ös verzióra ezen a platformon.

Ezekből is látszik, hogy a Xamarin egy nagyon újszerű és dinamikusan fejlődő keretrendszer, illetve a multi-platform technológia számos kihívást nyújt még.



4.6. ábra. Túl nagy TabbedPage oldalcím



4.7. ábra. Hiányzó és meglévő gombkeret

#### 4.4.2. Elemek elhelyezése az oldalakon

A legtöbb oldaltípus rendelkezik egy *Content* tagváltozóval, aminek a típusa *Xamarin.Forms.View*. Ez az osztály az őse az összes megjeleníthető vizuális és vezérlőelemek. Mivel oldalanként csak egy Content tag van, ha több elemet kívánunk elhelyezni, egy elrendezési konténert (angolul layout) kell használni, amibe aztán a többi elem kerülhet. Mivel ezek az osztályok is Xamarin View-k, ezért tetszőleges mélységen és variációban ágyazhatóak egymásba komplex elrendezések megvalósítására. A keretrendszer 9 konténert kínál [26], ezek közül én a GridLayout, StackLayout és ScrollView osztályokat használtam.

A *StackLayout* a legegyszerűbb elrendezési forma. Beállítástól függően függőlegesen vagy vízszintesen helyezi egymás mellé a benne tárolt elemeket, ezen túl csak az elemek közötti térköz állítható.

*GridLayout* használatakor meghatározhatunk egy rácssonkerzetet annak sorainak és oszlopainak számával. Megadhatjuk a sorok és oszlopok méreteit, akár a teljes rendelkezésre álló terület százalékos eloszlásában is. Ez különösen hasznos a grafikusfelület-programozásban kezdőknek, akik számára a pixelekben történő pozicionálás igen nehézkes.

A *ScrollView* listák megjelenítésére használható. A cella kinézetét sablonnal határozzuk meg, adatait adatkötésből nyeri (lásd 4.2.2 pont), amihez tipikusan rendezett listát használnak. A listák kötésében segít a *System.Collections.ObjectModel.ObservableCollection<T>* sablonosztály, mert ennek műveletei az elemeken automatikusan frissítési mechanizmust váltanak ki.

Az *ObservableCollection*-ök használatakor két gyakran, de véletlenszerűen felbukkanó hibával találkoztam. Az egyik, hogy a ScrollView a várta ellentében nem frissül. Ennek megoldásaként az objektumot csak a futás kezdetekor inicializálom, majd még teljes frissítéskor sem cserélem le azt egy másik osztálypéldányra, hanem az objektum *Clear()*, majd *Add()* metódusait használom az új elemek beillesztésére. Ennek hátterében az áll, hogy az objektum cseréjével felbomlanak a meglévő adatkötések, amik megújítására nincs mód [38]. A másik probléma a megjelenített lista részleges frissülése volt. Ennek megoldásakor derült fény az adatkötés szálbizzességének hiányára (lásd 4.2.3 pont). Az adatok szűréséhez és ren-

dezéséhez LINQ<sup>6</sup> lekérdezéseket használ a program, amik a C# nyelv *yield return* funkcióját kihasználva előre nem, hanem csak akkor hozzák létre az eredményhalmaz-kollekciót, amikor azon iterálni kezdünk. Ez a mechanizmus olyan állapotot idézhet elő, amikor a grafikus felület ScrollView-ja még az adatkötött ObservableCollection frissítésének befejezése előtt végrehajtja a megjelenített lista frissítését. Ennek az állapotnak az elkerülésére az eredményhalmazt még a háttérszálban egy *System.Collections.Generic.List<T>* objektumba enumerálom, és felülvizsgálva az alkalmazás szálkezelését módosításokat eszközöldem, hogy az adatkötések frissítése mindenkor az utolsó lépés legyen az adott műveletsorban.

Az applikációban használt diagramokat kezdetben a nyílt forráskódú OxyPlot [4] cso-maggal kezdttem megvalósítani. Ez a rendszer stílusozhatósági képességeiben nem felelt meg a követelményeknek, valamint súlyos stabilitási problémákkal rendelkezik Windows 10 Mobile-on, ezért a Syncfusion Essential Studio for Xamarin [19] programcsomagra váltottam. A Syncfusion könyvtárai tisztán XAML-ben, adatkötéssel programozhatóak, így a diagramok integrálása és stílusozása gyors és problémamentes volt.

#### 4.4.3. Stílusozás

A Xamarin.Forms vizuális elemei külön beállítás nélküli az operációs rendszer alapértelmezett stílusával jelennek meg. Lehetőség van azonban az egyes grafikuselem-típusokhoz XAML stílusdefiníciók írására, amivel biztosítható az alkalmazás egységes kinézete [28].

A stílus lehet adott grafikus elemhez írott vagy globálisan elérhető, illetve explicit vagy implicit. Explicit stílusoknál az objektum *Style* tulajdonságát hozzá kell kötni a stílushoz, míg az implicit stílusdefiníció automatikusan érvényesítve lesz minden olyan objektumra, aminek típusát az implicit stílus célozza (hacsak felül nem bíráljuk azt az objektumon). Az implicit stílusokkal nem sikerült látható eredményt elérnem, ezért az alkalmazás explicit stílusokat használ.

A különböző platformokon használt stílusozási ajánlások különböző betűméretekkel dolgoznak, illetve az elemek alap kinézete is jelentősen különbözheto, az alkalmazás kinézetét ehhez igazítani igen körülményes. Ezen egyszerűsít a XAML *x:Static* leíróbővítménye, amivel lehetséges C#-ban deklarált publikus statikus változókat adatkötni. A függelék F.4 pontjában lévő kód részletek szemléltetik, ahogy a *Device.OnPlatform()* függvény a kívánt megjelenést biztosító értékre állítja a statikus változót, majd a XAML kód XML névtérként (*local*) hivatkozza a C# névteret, és a statikus változó értékét köti a stílus betűméret tulajdonságához.

Az *x:Static* használatával nem csak stílustulajdonságokhoz köthetünk statikus változókat. A gombok alapértelmezett kinézete iOS rendszeren nem megfelelő az alkalmazás színsémájához, mert nem különbözik az egyszerű szövegtől (4.8. ábra). Keret hozzáadásával azonban a szöveg túl közel helyezkedik el a kerethez (4.9. ábra). Megoldásul csak ennél a platformnál a szöveg elejére és végére is egy-egy szóközt raktam a megfelelő megjelenéshez (4.10. ábra). A kapcsolódó kód részletek a függelék F.5 pontjában találhatók.

---

<sup>6</sup>A Language-Integrated Query könyvtár hatékony kollekciómanipulálási funkciókat nyújt szinte bármilyen adatforráshoz [33].



**4.8. ábra.** iOS gomb alapértelmezett kinézete

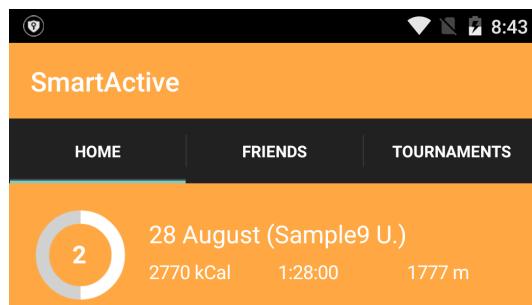


**4.9. ábra.** iOS gomb kerettel



**4.10. ábra.** iOS gomb platformspecifikus szöveggel és kerettel

A stílusozás folyamán is problémákba ütköztem a TabbedPage osztálytal. A 4.5. ábrán látható, hogy Windows 10 Mobile rendszeren az oldalcím és a fülek nevei fekete színűek. A 4.11. ábra pedig a fülsáv-stílusozás hiányát mutatja Android operációs rendszeren. Mindkét probléma arra vezethető vissza, hogy jelenleg a TabbedPage osztály nem rendelkezik tulajdonságokkal a fülsáv stílusozására. A 2.0-ás verzió előtt ez lehetséges volt a TabBar tagváltozó megfelelő tulajdonságainak beállításával, azonban ezt a funkciót eltávolították. A nyílt forráskódú Xamarin-Forms-Labs [7] projekt korábban egy kerülő megoldást nyújtott *ExtendedTabbedPage* osztályával, azonban 2015 folyamán a stílusozás funkció hatástarlánná vált, mára pedig a forráskód is törölve lett a projekt GitHub tárhelyéről. Ezek miatt a stílushibák a véleges alkalmazásban is jelen vannak.



**4.11. ábra.** A stílushoz nem illeszkedő fülsáv Android rendszeren

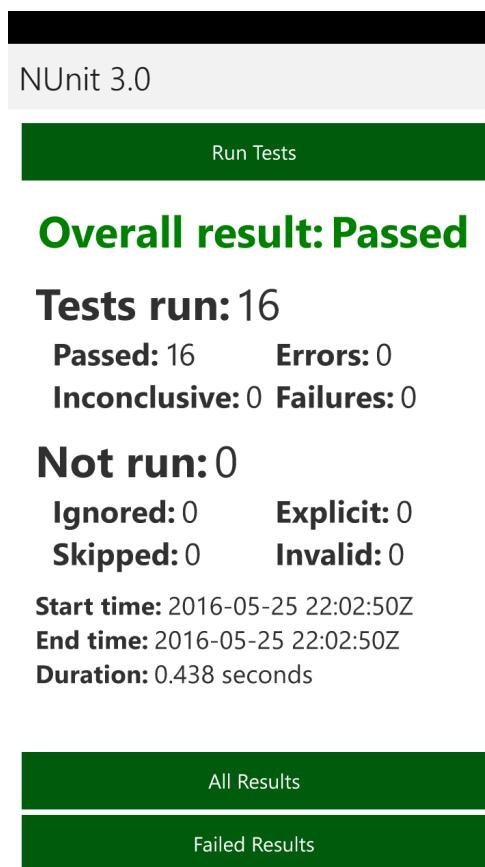
#### 4.5. Az alkalmazás tesztelése

A tesztelési feladatokon túl már az implementáció során szerettem volna csökkenteni a hibák számát, ezért a munka megkezdésekor telepítettem a JetBrains Resharpert [14]. Ez egy gazdag funkcionalitású automatikus kódkiegészítő és folyamatos futású statikushiba-kereső program. Nem csak szintaktikai, de egyes szemantikai hibákat is képes jelezni, valamint javaslatokat tesz kódrészletek optimalizálási lehetőségeire, ezzel javítva a kódminőséget.

#### 4.5.1. Unit tesztek

A unit tesztek készítése során az alkalmazás funkcióinak függvényszintű működését ellenőrzük pozitív és negatív tesztesetekkel. Másképp fogalmazva a függvényeknek szabályos vagy szabálytalan inputot adunk, és a tesztnél azt ellenőrzük, hogy a bemenetet a várt módon kezeli-e a függvény. A unit tesztek hasznosságát bizonyítja a tesztközpontú fejlesztés (Test Driven Development, TDD) megjelenése és térnyerése is. A TDD-ben az osztálystruktúra megtervezése és vázának elkészítése után először az átfogó unit tesztek készülnek el, csak ezután történik meg a funkciók implementációja.

A .NET keretrendszerben írt kódok unit tesztelésére mára kvázi szabvánnyá vált a nyílt forráskódú NUnit [3]. A rendszernek létezik Xamarinhoz optimalizált verziója, ekkor a tesztfuttató környezet egy, a telefonra települő program, ami elindításakor lefuttatja a teszteket, majd egy összefoglalót nyújt az eredményekről (4.12. ábra).

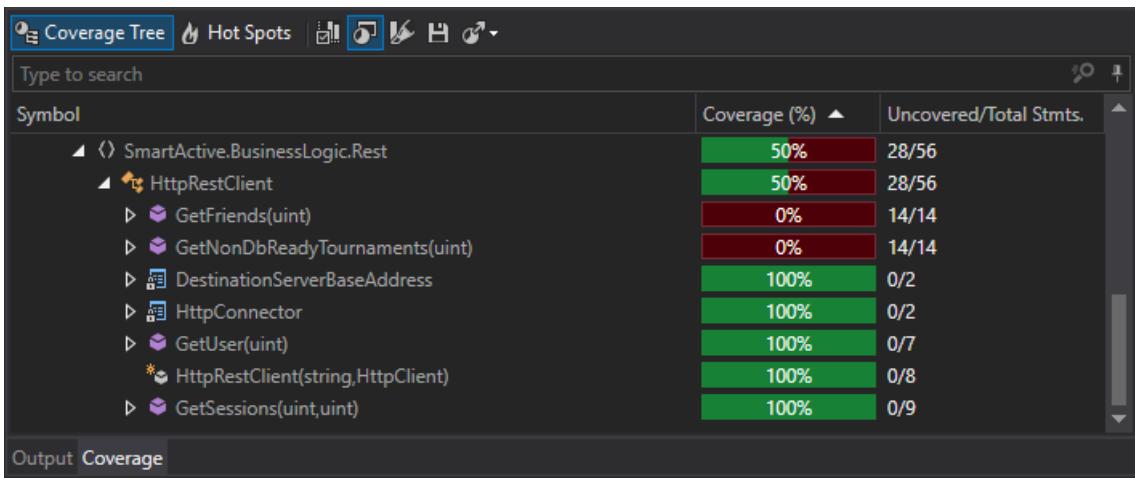


4.12. ábra. Egy NUnit tesztfutás eredménye Windows 10 Mobile-on

A hordozható osztálykönyvtárbeli kód Visual Studioban futtatható tesztelésére létrehoztam egy asztali .NET alkalmazást célzó projektet, a tesztkódot pedig egy megosztott projektbe helyeztem. Így már használható tetszőleges unitesztfuttató-környezet a tesztek Visual Studioban történő futtatására és kiértékelésére.

Erre a célról a JetBrains dotCover [13] futtatókörnyezetet választottam, ami a teszteredmények megjelenítésén túl képes kódlefedettség-vizsgálatra is, azaz megmutatja, hogy a tesztek során a termékkód mely sorai kerültek lefutásra. Egy ilyen vizsgálat eredmé-

nyét mutatja a 4.13. ábra. Minél több kódöt fedünk le minél sokrétűbb tesztekkel, annál biztosabbak lehetünk, hogy az applikáció az elvárt viselkedést produkálja szélsőséges körfülmények között is.



**4.13. ábra.** Egy dotCover kódlefedettség-vizsgálat eredménye a *HttpClient* osztályon

Az applikációhoz 16 unit tesztet készítettem, amik az üzleti logika viselkedését ellenőrzik. NUnit tesztekkel önmagában a grafikus felület működése nem ellenőrizhető. Erre a Xamarin a TestCloud termékét kínálja, amihez a Xamarin.UITest könyvtár használatával lehet írni teszteket. Ilyen teszteket nem állt módomban készíteni, mert a Xamarin Student Partner Programban<sup>7</sup> igényelt TestCloud hozzáférés-kérelmemet nem bírálták el kellő időben.

#### 4.5.2. Manuális tesztek

A grafikus alkalmazások fejlesztése megköveteli az írott kód folyamatos ellenőrzését. A Xamarin.Forms a Microsoft WPF-fel szemben még nem rendelkezik úgynevezett betekintőnézet funkcióval, azaz csak úgy látható a vizuális elemek implementációjának eredménye, ha az alkalmazást lefordítjuk, majd futtatjuk például egy emulátorban (lásd 3.2.1 pont). Mivel az applikáció multi-platform, ezért minden ellenőrzést négy emulátorban kellett elvégezniem, hogy a platformspecifikus hibákra is fény derüljön.

A fejlesztés során két ismerősömet kértem meg, hogy teszteljék az alkalmazást az emulátorokban. Ők funkcionális gondokat nem, csak platformspecifikus megjelenési hibákat jeleztek, mint például kilógó szövegek, vagy a TabbedPage osztály azon hibája, amit a 4.4.1 pontban részletesen ismertettem. Ezeket a hibákat - a lehetőségekhez mérten - javítottam. Az alkalmazás végleges androidos verzióját valódi telefonon is teszteltem. Egy Android 6.0 "Marshmallow" rendszert futtató LG G3 (D855) készülékre telepítettem az appot. Az applikáció a vártaknak megfelelően viselkedett, nem mutatott eltérést az emulátorban tapasztaltakhoz képest.

<sup>7</sup>A program honlapja: <https://www.xamarin.com/student>

## 5. fejezet

# Az elkészült alkalmazás bemutatása

Az alábbiakban bemutatom a féléves munkám során elkészített alkalmazás végleges verzióját, annak oldalait és a felhasználó számára elérhető funkciókat. Ebben a fejezetben képernyőképek csak az alkalmazás iOS verziójáról láthatók. Az iOS platform sajátja, hogy a visszafelé navigálás ikonja mellett megjeleníti annak az oldalnak a nevét, ahova a navigáció visz, emiatt az oldalak fejléce több ponton sűrűnek tűnhet. A többi platformon más elrendezésű fejlécek jelennek meg. Az applikációról készült multi-platform képernyőképek a függelék F.6 pontjában foglalnak helyet.

Az alkalmazás jelenlegi állapotában nem a SmartActive webszerverrel kommunikál, mert annak a komponensnek a felelőse nem készült el időben feladatával. Kerülőmegoldásként a Dropbox<sup>1</sup> ingyenes felhős tárhelyszolgáltatásban hoztam létre JSON fájlokat, és azok URI-jait<sup>2</sup> beégettem a kommunikációt megvalósító függvényekbe. Ahhoz, hogy a dropboxban lévő fájlokat módosítani tudjam az alkalmazásban, saját azonosító adataimat is be kellett volna égetnem, ezt pedig adatvédelmi okokból nem tettem meg. Ennek következményeképp amikor az alkalmazás szinkronizál a szerverrel, a helyi adatbázisban tárolt változások elvesznek (pl. hozzáadott barát, létrehozott torna).

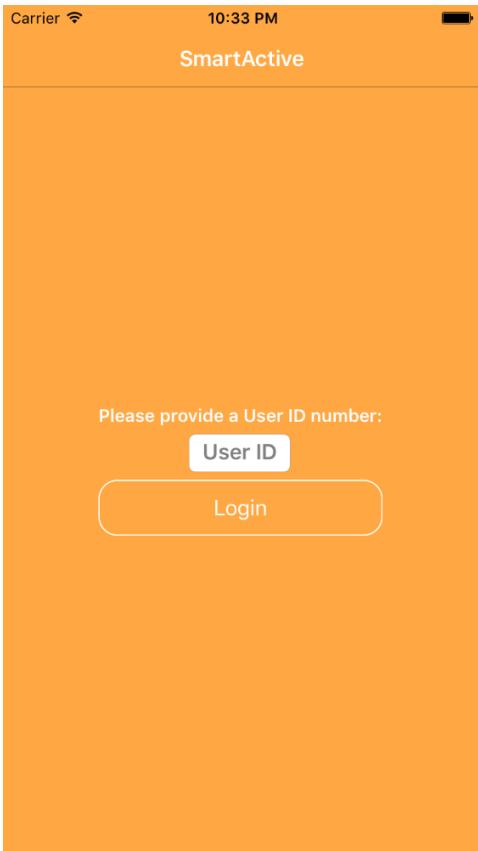
### 5.1. Bejelentkező-oldal

Az applikáció első indulásakor a bejelentkező-képernyő látható, ami a felhasználó egyedi azonosítóját kéri (5.1. ábra). A dropboxtárhely-használat miatt kér az app azonosítószámot a megszokott felhasználónév - jelszó páros helyett, mivel utóbbi funkciót felesleges lett volna megvalósítani beégetett fájlútvonalak használata mellett. A megadott azonosító sikeres bejelentkezés esetén eltárolódik, így a további alkalmazásindításokkor az app már nem kéri azt. Hibás azonosító megadása esetén hibaüzenet jelenik meg (5.2. ábra).

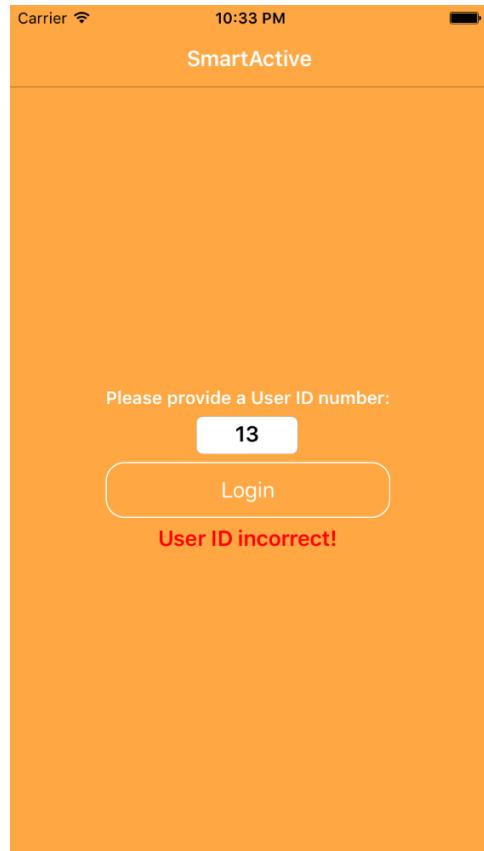
---

<sup>1</sup>További információ a Dropbox termékről: <https://www.dropbox.com/>

<sup>2</sup>Uniform Resource Identifier, egy rövid karaktersorozat, amelyet egy webes erőforrás azonosítására használunk, pl. egy honlapcím.



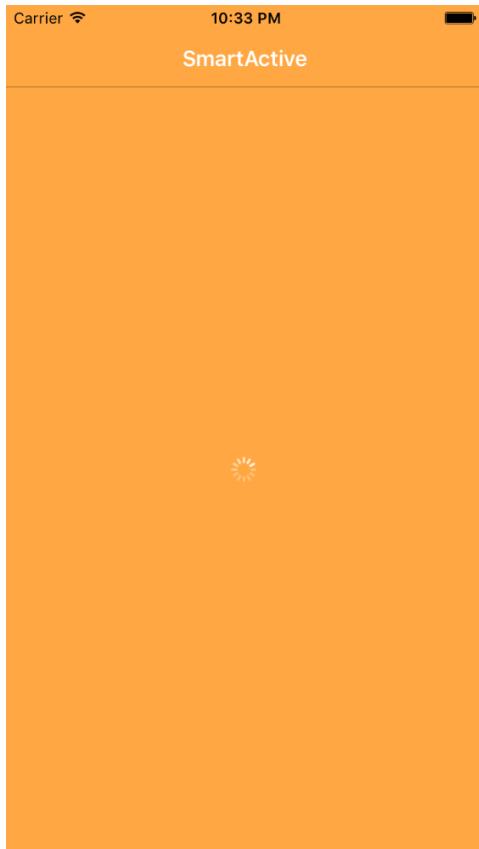
**5.1. ábra.** A bejelentkező-oldal



**5.2. ábra.** Hibaüzenet hibás felhasználóazonosítónál

## 5.2. Információk, statisztikák az edzésekről

A bejelentkezés utáni első adatszinkronizálás 5-10 másodperces ideje alatt töltőképernyő jelenik meg (5.3. ábra), majd az alkalmazás kezdőoldalaként a korábbi edzések listája látható, illetve itt lehet kijelentkezni is a *Logout* gombbal (5.4. ábra). A listaelémek egy-egy rövid összefoglalót mutatnak az egyes edzésekről. A cellák felső szövegsorában látható az edzés dátuma, zárójelben az ellenfél keresztneve és vezetéknévénél kezdőbetűje. Ezek alatt az edzés során elégetett kalóriák száma, az edzés hossza óra:perc:másodperc formátumban, illetve a megtett táv olvasható. A szöveges információktól balra egy fánkdiagram helyezkedik el, benne az edzés során játszott meccsek számával, a diagram eloszlása pedig a győztes és vesztes meccsek arányát mutatja sorrendben fehér és szürke színnel. A lista támogatja a lehúzással történő frissítést, erről képernyőkép a függelék F.6.7. ábráján látható. Ez a funkció a Xamarin jelenlegi verziójával csak Android és iOS operációs rendszereken érhető el.

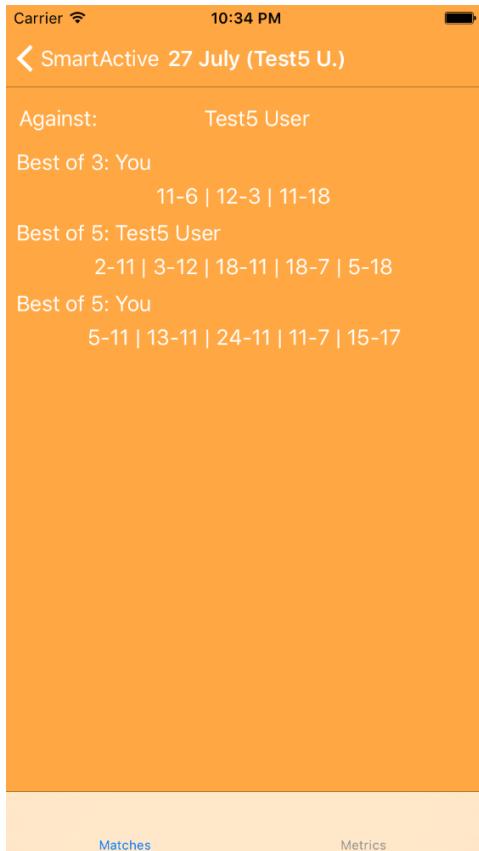


**5.3. ábra.** Töltési animáció a bejelentkezés alatt

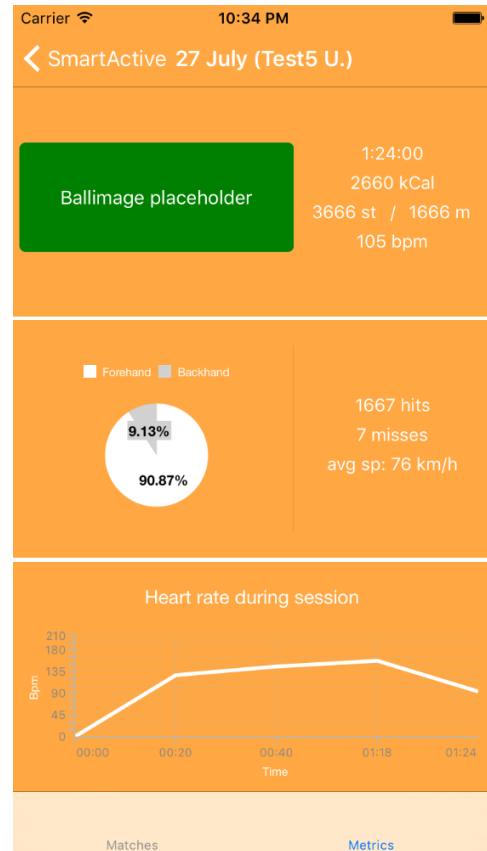
SmartActive			
2	28 August (Test6 U.) 2770 kCal 1:28:00 1777 m		
3	27 July (Test5 U.) 2660 kCal 1:24:00 1666 m		
2	26 June (Test6 U.) 2550 kCal 1:20:00 1555 m		
3	25 May (Test6 U.) 2440 kCal 1:16:00 1444 m		
2	24 April (Test5 U.) 2330 kCal 1:12:00 1333 m		
Logout			
<a href="#">Home</a>	<a href="#">Friends</a>	<a href="#">Tournaments</a>	

**5.4. ábra.** A korábbi edzések listája

Az edzéslista egy elemét kiválasztva arról részletes információkat tekinthetünk meg. Az első fülön (5.5. ábra) látható az ellenfél teljes neve, a játszott meccsek típusa ("legjobb a háromból" vagy "legjobb az ötből"), a meccsgyőztes neve, valamint az egyes szettek eredményei "felhasználó pontjai - ellenfél pontjai" formában. Az edzőpartner nevére kattintva részletesebb információkat kaphatunk róla, amiket az 5.8. ábra szemléltet. A második fül (5.6. ábra) az edzés során készült statisztikákat mutatja. Megtekinthetők biometrikai adatok, úgy mint az elégetett kalóriák száma, a megtett lépések száma és a megtett táv mértéke, valamint az átlagos szívritmus és annak változása az edzés során vonaldiagram formájában. Ezen túl a labdamenetről az ütések és melléütések számát, azok átlagos sebességét, valamint egy kördiagramon az ütések típusának (tenyeres vagy fonák) százalékos eloszlását mutatja az oldal. A vonaldiagram egy pontját, illetve a kördiagram egy szeletét kiválasztva megtekinthető annak számértéke is. A zöld téglalap egy úgynevezett helyőrző (angolul placeholder), ahova az edzés során használt labda típusáról (kezdő, középhaladó, haladó vagy profi) kerülhet kép.



**5.5. ábra.** Egy edzés meccseinek eredményei



**5.6. ábra.** Egy edzésről készült statisztikák

### 5.3. Barátok kezelése

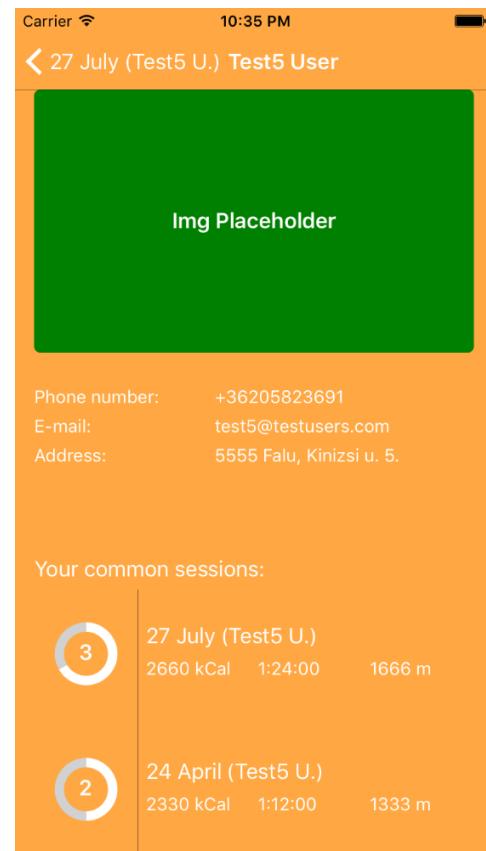
Az alkalmazás második fő fülén a barátok kereshető, ábécé sorrendben rendezett listája látható a barát teljes nevével, valamint az eddigi közös edzések számával (5.7. ábra). A barátlista is frissíthető lehúzással Android és iOS rendszereken (függelék F.6.8. ábra). A keresés funkciót a függelék F.6.13. ábrája szemlélteti.

A lista egy elemét kiválasztva az adott barátról részletesebb információkat kapunk (5.8. ábra). Itt a helyőrző a barát profilképét mutathatja, alatta a telefonszáma, e-mail címe és lakkíme tekinthető meg, végül pedig az eddigi közös edzések listája látható, aminek egy elemét választva annak részletezőoldalára jutunk (5.5. és 5.6. ábrák).

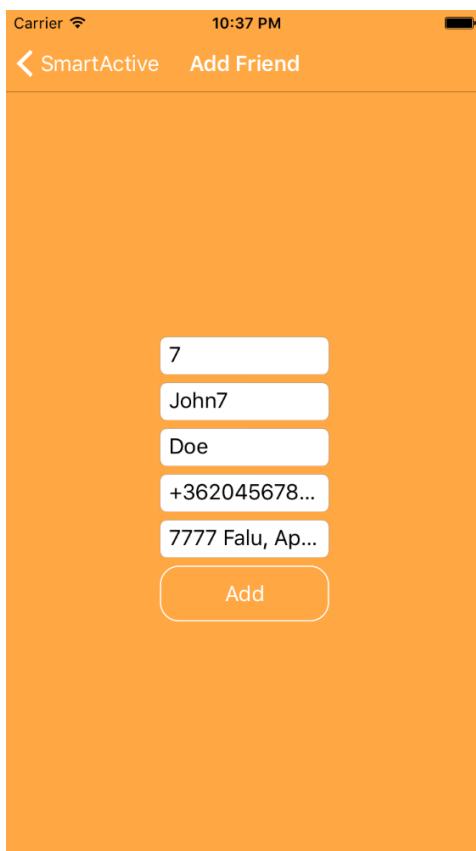
Lehetőség van barátok hozzáadására is az *Add Friend* gombbal. A webszerver komponens hiánya miatt a hozzáadási oldalon (5.9. ábra) lévő adatok megadásával egy új barátrekordot hozunk létre a helyi adatbázisban, nem pedig a rendszerbe regisztrált felhasználók listájából választunk. Az *Add* gombra kattintva a barátlista oldalra lép vissza az app, ahol már az új barát is látszik (5.10. ábra).



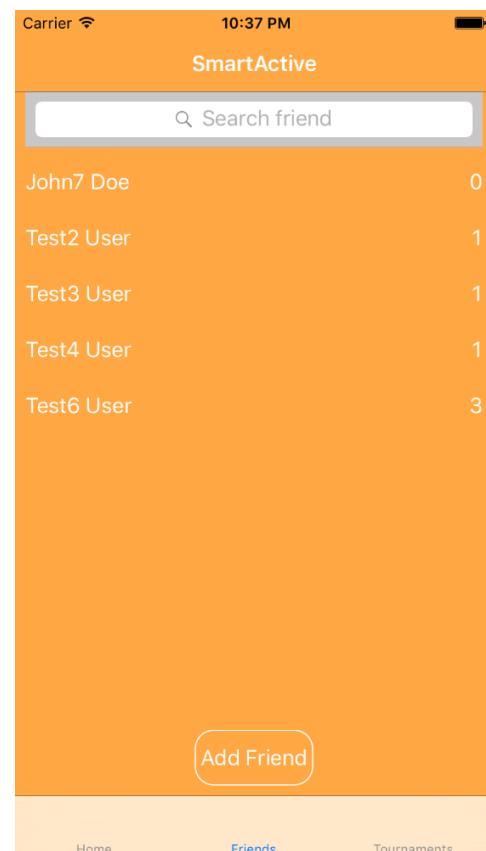
5.7. ábra. A barátok listája



5.8. ábra. Részletes információk egy barátról



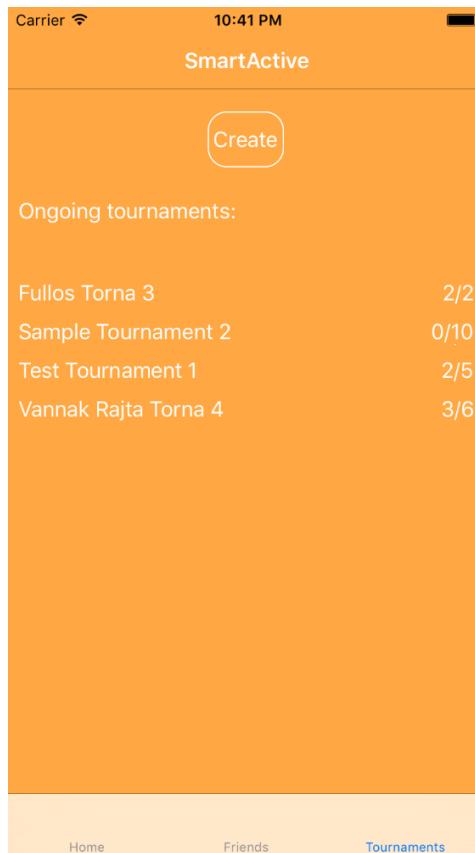
5.9. ábra. Új barát hozzáadása



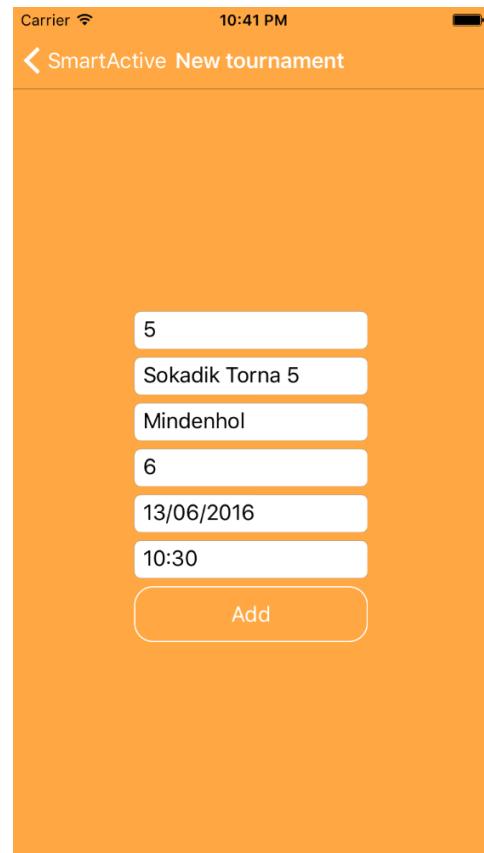
5.10. ábra. Hozzáadás után az új barát a listában van

## 5.4. Bajnokságok szervezése, részvétel bajnokságokon

A harmadik fő alkalmazásfül a bajnokságokról ad áttekintőt (5.11. ábra). Látható a futó bajnokságok listája ábécé rendben, ahol a bajnokság neve mellett a résztvevők aktuális és maximális létszáma is szerepel. A lista Android és iOS rendszereken lehúzással frissíthető (függelék F.6.26. ábra). Lehetőség van új bajnokság létrehozására a *Create* gombbal megjelenő hozzáadási oldalon (5.12. ábra). Itt az *ID* mező megadása azért szükséges, mert a barátok felvételéhez hasonló módon itt is egy rekordot hozunk létre a helyi adatbázisban, mivel a webszerver komponens jelenleg nem érhető el. A függelék F.6.16. és F.6.17. ábrái szemléltetik, hogy a dátum és idő megadásához a platform natív grafikus elemeit használja az app.

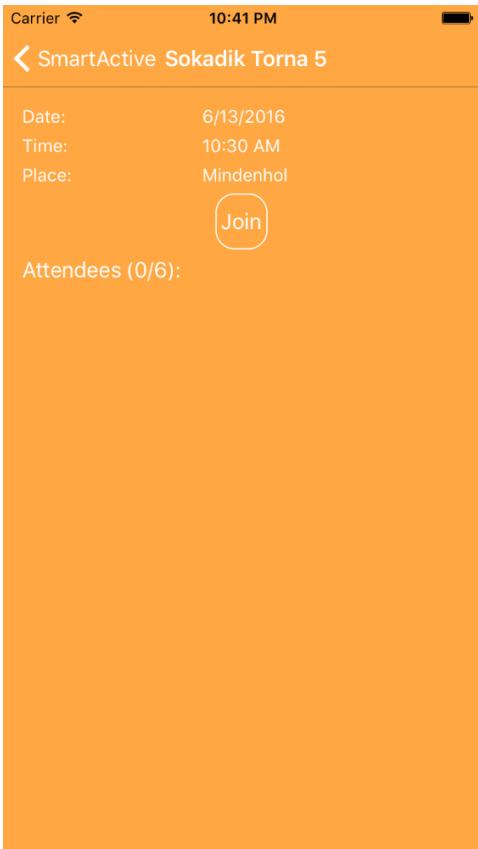


5.11. ábra. A bajnokságok oldala

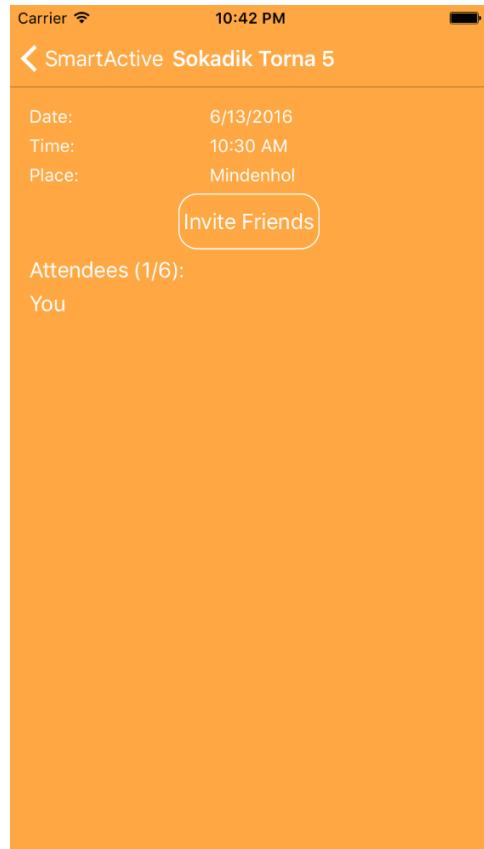


5.12. ábra. Új bajnokság létrehozása

Egy bajnokságot kiválasztva a listából annak részleteit tekinthetjük meg (5.13. ábra), a *Join* gombra kattintva csatlakozhatunk a bajnoksághoz. Ekkor a felhasználó "You" néven megjelenik a résztvevők listájában, ahol a későbbiekben is elsőként szerepel majd, illetve meghívhat barátokat a bajnokságra az *Invite Friends* gomb használatával (5.14. ábra).



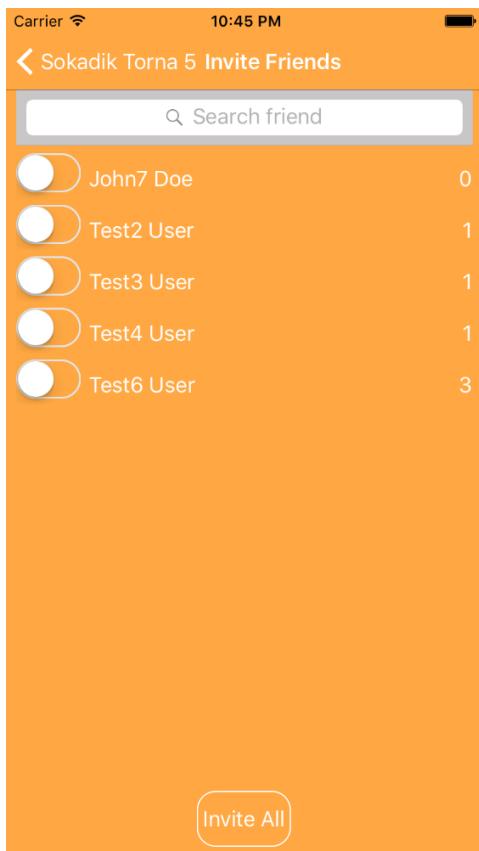
**5.13. ábra.** Az új bajnokság részletei



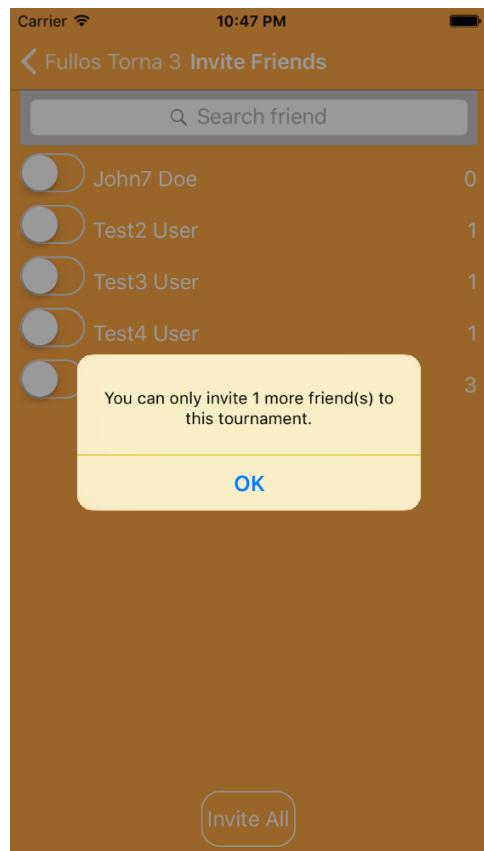
**5.14. ábra.** A bajnokságrészletező-oldal csatlakozás után

A barátok meghívására szolgáló oldalon (5.15. ábra) meghívhatjuk egy gombnyomással az összes ismerősünket, azonban ha ezek száma több, mint a bajnokság maximális létszáma, hibaüzenetet kapunk a még meghívható barátok számával (5.16. ábra). Az ismerőök listája itt is kereshető, a szűrt listában történő kijelölésekkel megjegyzi az alkalmazás a keresőszó törlése után is (5.17. és 5.18. ábrák). A meghívást az *Invite* gombbal véglegesítjük.

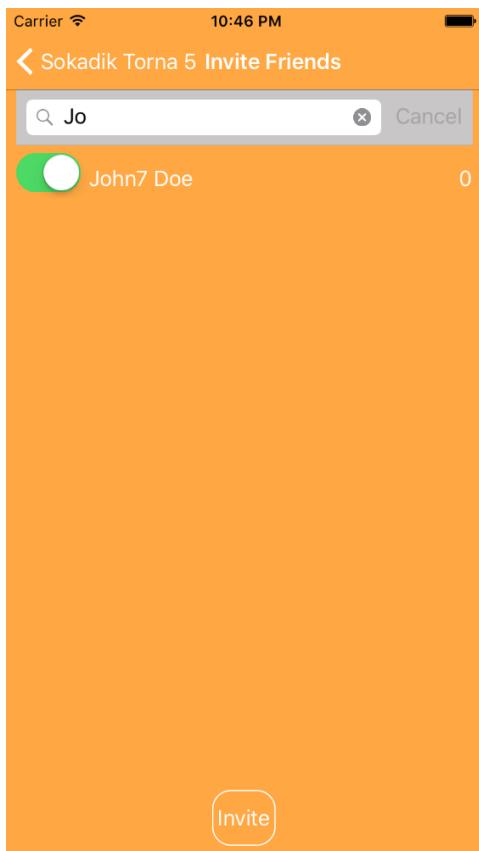
Meghívás után a bajnokság résztvevőinek listája azonnal frissül (5.19. ábra). A meghívás elfogadása lépés azért hiányzik, mert a dropbox tárhez való kapcsolat egyirányú, így jelenleg nem megoldható, hogy A felhasználó részvételi kérelmet küldjön B-nek. Ha a meghívottak hozzáadásával egy bajnokság megtelt, az *Invite Friends* gomb eltűnik, ezzel megszűnik a további résztvevők hozzáadásának lehetősége (5.20. ábra).



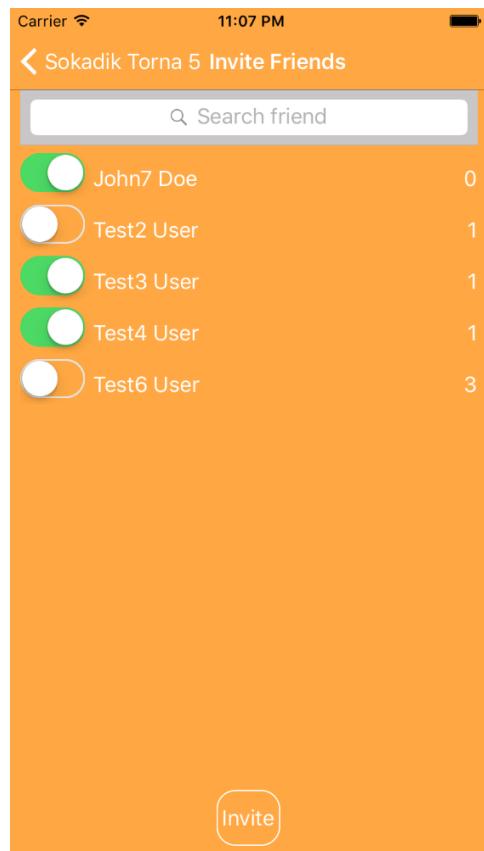
5.15. ábra. Barátok meghívása oldal



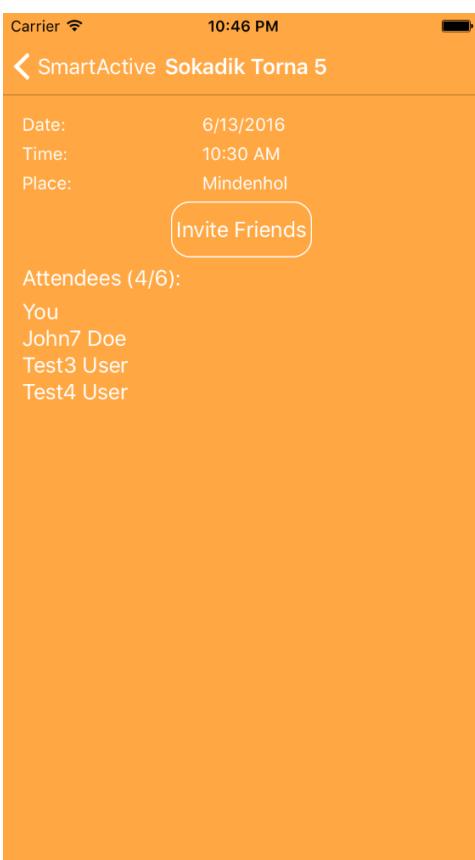
5.16. ábra. Hibaüzenet túl sok ismerősz meghívásakor



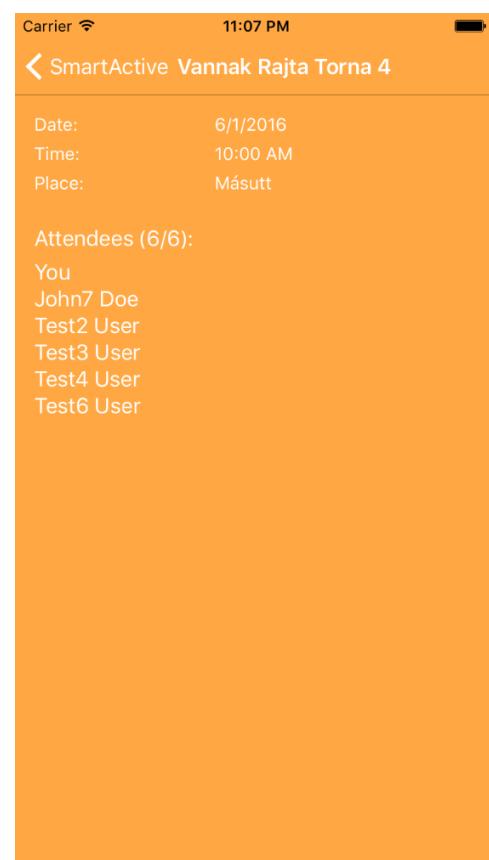
5.17. ábra. Meghívás kereséssel



5.18. ábra. Kiválasztott barátok meghívása



**5.19. ábra.** A bajnokság résztvevői meghívás után



**5.20. ábra.** Egy megtelt bajnokság

## 6. fejezet

# Konklúzió, továbbviteli lehetőségek

Munkám során megismerkedtem a Xamarin multi-platform szoftverfejlesztő keretrendszerrel, és tapasztalatot szereztem egy komplex grafikus alkalmazás elkészítésében a C# nyelv, Microsoft fejlesztőeszközök és korszerű programozási paradigmák használatával. Beteiktést nyertem a unit tesztek készítésébe az NUnit keretrendszerrel, és megtapasztam ezen tesztek hasznosságát a kódminőség szempontjából.

Elkészítettem a SmartActive projekt keretében egy mobilalkalmazást, ami négy különböző mobiloperációs-rendszert támogat. Az alkalmazás funkcionalitása és struktúrája minden célplatformon megegyezik, ezzel azonos élményt nyújtva. A felhasználó mobilkészülékén gyorsan és hatékonyan hozzáérhet a squash sporttevékenységéhez kapcsolódó statisztikákhoz, valamint a projekt nyújtotta közösségi funkciókat is igénybe veheti: megtekintheti barátait, és bajnokságokat szervezhet, amire meghívhatja ismerőseit.

Véleményem szerint a Xamarin a C#-ban járatos fejlesztők számára egy hasznos és funkcionalitásban gazdag keretrendszer, használatával nincs szükség a konkurens termékek nagy részénél megkövetelt webtechnológiák beható ismeretére. További előnye, hogy a .NET könyvtár jelentős része használható a közös osztálykönyvtárbeli kódban is. Ugyanakkor a Xamarin.Forms multi-platform grafikusfelület-könyvtár még fiatalabból adódóan problémákkal és hiányosságokkal küzd, ahogy ezekre munkámban rámutattam.

Az alkalmazás utólagos kiértékelésekor megállapítottam, hogy a barátok hozzáadása funkcionál (5.3 pont) implementált kerülőmegoldás - ami egy barátrekord létrehozása a helyi adatbázisban - helyett használhattam volna egy JSON fájlt a dropbox tárhelyen, ami a felhasználók listáját adja vissza, és a bejelentkezett felhasználó ebből választ. A webszerver komponens elkészültéig ez a módosítás eszközölhető, utána pedig az alkalmazás bővíthető kétirányú szinkronizálással nem csak a barátok hozzáadásánál, de a bajnokságok létrehozásánál és kezelésénél is. Még egy hibajavítási opción lehet a TabbedPage oldalcímének platformspecifikus eltüntetése Windows Phone 8.1-en az x:Static leíróbővítmény használatával (lásd 4.4.3 pont), amíg a Xamarin keretrendszerben ki nem javítják a 4.6. ábrán látható "túl nagy oldalcím-betűmérő" hibát.

Az applikáció dizájnja a későbbiekben finomítható, például a helyőrzők lecserélhetők valós képekre, illetve a statisztikai elemek megjelenése fiatalosabbá tehető piktogramokkal. Továbbá az applikáció a SmartActive projekt előrehaladtával új funkciókkal bővíthető,

például a kezdeti tervekben szereplő pályafoglalási lehetőséggel. Szintén egy bővítési opción, hogy az edzések információi megoszthatóak legyenek más szociális platformokon, mint például Facebook vagy Twitter, valamint importálhatóak legyenek barátok más portálok ról, illetve a készülék telefonkönyvéből. A platformtámogatás kiterjeszthető a viselhető eszközökre, például okosórákra, így az app akár a sportolás során is szolgálhat hasznos információkkal, és adatok nyerhetők ki az órába épített szenzorok segítségével. Ezzel a fejlesztéssel a mobilapplikáció már nem csak fogyasztójává, hanem szolgáltatójává is válna a sportolási statisztikáknak.

# Irodalomjegyzék

- [1] Provab blog. Hybrid vs cross-platform vs native application development (2016. május 18.). <http://www.blog.provab.co.in/hybrid-vs-cross-platform-vs-native-application-development/>.
- [2] Mockups To Go Community. Balsamiq mockups to go home page (2016. május 26.). <https://mockupstogo.mybalsamiq.com/projects>.
- [3] NUnit Developer Community. Nunit home page (2016. május 25.). <http://nunit.org/>.
- [4] OxyPlot Developer Community. Oxyplot documentation welcome page (2016. május 25.). <http://docs.oxyplot.org/en/latest/index.html>.
- [5] TwinCoders Community. Sqlite-net extensions project site (2016. május 24.). <https://bitbucket.org/twincoders/sqlite-net-extensions>.
- [6] Wikipedia Community. Apache cordova angol nyelvű szócikk (2016. május 20.). [https://en.wikipedia.org/wiki/Apache\\_Cordova](https://en.wikipedia.org/wiki/Apache_Cordova).
- [7] Xamarin.Forms Labs Developer Community. Xamarin.forms labs project github page (2016. május 26.). <https://github.com/XLabs/Xamarin-Forms-Labs/>.
- [8] SQLite Consortium. Sqlite project home page (2016. május 24.). <https://www.sqlite.org/>.
- [9] CoronaLabs. Corona product home page (2016. május 20.). <https://coronalabs.com/>.
- [10] Appcelerator Inc. Appcelerator development products page (2016. május 20.). <http://www.appcelerator.com/mobile-app-development-products/#AppsSect>.
- [11] Balsamiq Inc. Balsamiq mockups product page (2016. május 26.). <https://balsamiq.com/products/mockups/>.
- [12] Ericsson Inc. Ericsson mobility report: 70 percent of world's population using smartphones by 2020 (2016. május 8.). <http://www.ericsson.com/news/1925907>.
- [13] JetBrains Inc. Dotcover product page (2016. május 25.). <https://www.jetbrains.com/dotcover/>.

- [14] JetBrains Inc. Resharper product page (2016. május 25.). <https://www.jetbrains.com/resharper/>.
- [15] Kony Inc. 'a leader. again.' blogbejegyzés (2016. május 20.). [http://forms.kony.com/2015-Gartner-Magic-Quadrant-Mobile-Application-Development-Platforms\\_1GetMQReport.html](http://forms.kony.com/2015-Gartner-Magic-Quadrant-Mobile-Application-Development-Platforms_1GetMQReport.html).
- [16] Kony Inc. Kony home page (2016. május 20.). <http://www.kony.com/>.
- [17] Sencha Inc. Sencha ext js framework product page (2016. május 20.). <https://www.sencha.com/products/extjs/>.
- [18] Sencha Inc. Sencha touch framework product page (2016. május 20.). <https://www.sencha.com/products/touch/>.
- [19] Syncfusion Inc. Syncfusion essential studio for xamarin product page (2016. május 25.). <https://www.syncfusion.com/products/xamarin>.
- [20] Unity3D Inc. Unity3d tutorials - scripting (2016. május 20.). <http://unity3d.com/learn/tutorials/topics/scripting>.
- [21] Xamarin Inc. Xamarin guides - code sharing options (2016. május 26.). [https://developer.xamarin.com/guides/cross-platform/application\\_fundamentals/building\\_cross\\_platform\\_applications/sharing\\_code\\_options/](https://developer.xamarin.com/guides/cross-platform/application_fundamentals/building_cross_platform_applications/sharing_code_options/).
- [22] Xamarin Inc. Xamarin guides - connecting to the mac (2016. május 19.). [https://developer.xamarin.com/guides/ios/getting\\_started/installation/windows/connecting-to-mac/](https://developer.xamarin.com/guides/ios/getting_started/installation/windows/connecting-to-mac/).
- [23] Xamarin Inc. Xamarin guides - introduction to portable class libraries (2016. május 23.). [https://developer.xamarin.com/guides/cross-platform/application\\_fundamentals/pcl/introduction\\_to\\_portable\\_class\\_libraries/](https://developer.xamarin.com/guides/cross-platform/application_fundamentals/pcl/introduction_to_portable_class_libraries/).
- [24] Xamarin Inc. Xamarin guides - introduction to shared projects (2016. május 23.). [https://developer.xamarin.com/guides/cross-platform/application\\_fundamentals/shared\\_projects/](https://developer.xamarin.com/guides/cross-platform/application_fundamentals/shared_projects/).
- [25] Xamarin Inc. Xamarin guides - working with a local database (2016. május 24.). <https://developer.xamarin.com/guides/xamarin-forms/>.
- [26] Xamarin Inc. Xamarin guides - xamarin.forms layouts (2016. május 25.). <https://developer.xamarin.com/guides/xamarin-forms/controls/layouts/>.
- [27] Xamarin Inc. Xamarin guides - xamarin.forms pages (2016. május 25.). <https://developer.xamarin.com/guides/xamarin-forms/controls/pages/>.
- [28] Xamarin Inc. Xamarin guides - xamarin.forms styles (2016. május 25.). <https://developer.xamarin.com/guides/xamarin-forms/user-interface/styles/>.

- [29] Xamarin Inc. Xamarin home page (2016. május 20.). <https://www.xamarin.com/>.
- [30] Xamarin Inc. Xamarin.forms product page (2016. május 20.). <https://developer.xamarin.com/guides/xamarin-forms/>.
- [31] Oystein Krog. Sqlite.net-pcl project site (2016. május 24.). <https://github.com/oysteinkrog/SQLite.Net-PCL>.
- [32] Frank A. Krueger. Sqlite-net pcl project site (2016. május 24.). <https://github.com/praeclarum/sqlite-net>.
- [33] Microsoft Developer Network. Language-integrated query (linq) (c#) (2016. május 25.). <https://msdn.microsoft.com/en-us/library/mt693024.aspx>.
- [34] Microsoft Developer Network. The mvvm pattern (2016. május 23.). <https://msdn.microsoft.com/en-us/library/hh848246.aspx>.
- [35] Microsoft Developer Network. Xaml overview (2016. május 23.). <https://msdn.microsoft.com/en-us/windows/uwp/xaml-platform/xaml-overview>.
- [36] James Newton-King. Json.net documentation - introduction (2016. május 23.). <http://www.newtonsoft.com/json/help/html/Introduction.htm>.
- [37] ECMA International Standard Organization. Introducing json (2016. május 26.). <http://www.json.org/>.
- [38] Michael L. Perry. Common mistakes while using observablecollection (2016. május 24.). [http://updatecontrols.net/doc/tips/common\\_mistakes\\_observablecollection](http://updatecontrols.net/doc/tips/common_mistakes_observablecollection).
- [39] Statista Statistics Portal. Distribution of android operating systems used by android phone owners in may 2016, by platform version (2016. május 9.). <http://www.statista.com/statistics/271774/share-of-android-platforms-on-mobile-devices-with-android-os/>.
- [40] Statista Statistics Portal. Global market share held by the leading smartphone operating systems in sales to end users from 1st quarter 2009 to 4th quarter 2015 (2016. május 9.). <http://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/>.
- [41] Statista Statistics Portal. Number of smartphone users worldwide from 2014 to 2019 in millions (2016. május 8.). <http://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/>.

# Mellékletek

## M.1. Forrásfájlok

A Diplomaterv Portálon (<https://diplomaterv.vik.bme.hu/>) a szakdolgozatom mellékleteként feltöltött *melleklet\_portal.zip*, illetve a bekötött szakdolgozathoz mellékelt DVD lemezen lévő *melleklet\_dvd.zip* tömörített állományok *M\_1\_Forrasfajlok* mappájában találhatóak a mobilalkalmazás végleges verziójának git verziókezelő használatára alkalmas forrásfájljai.

A dolgozat 3. fejezetében ismertettem annak az infrastruktúrának a létrehozását, amelyben az alkalmazás fordítható. A *Visual Studio Solution* fájl az *M\_1\_Forrasfajlok\SmartActive\SmartActive.sln* útvonalon található. Fordítási nehézségek esetén az *M\_1\_Forrasfajlok\README.md* szöveges fájl tartalma ad útmutatást.

## M.2. Telepíthető alkalmazáscsomagok

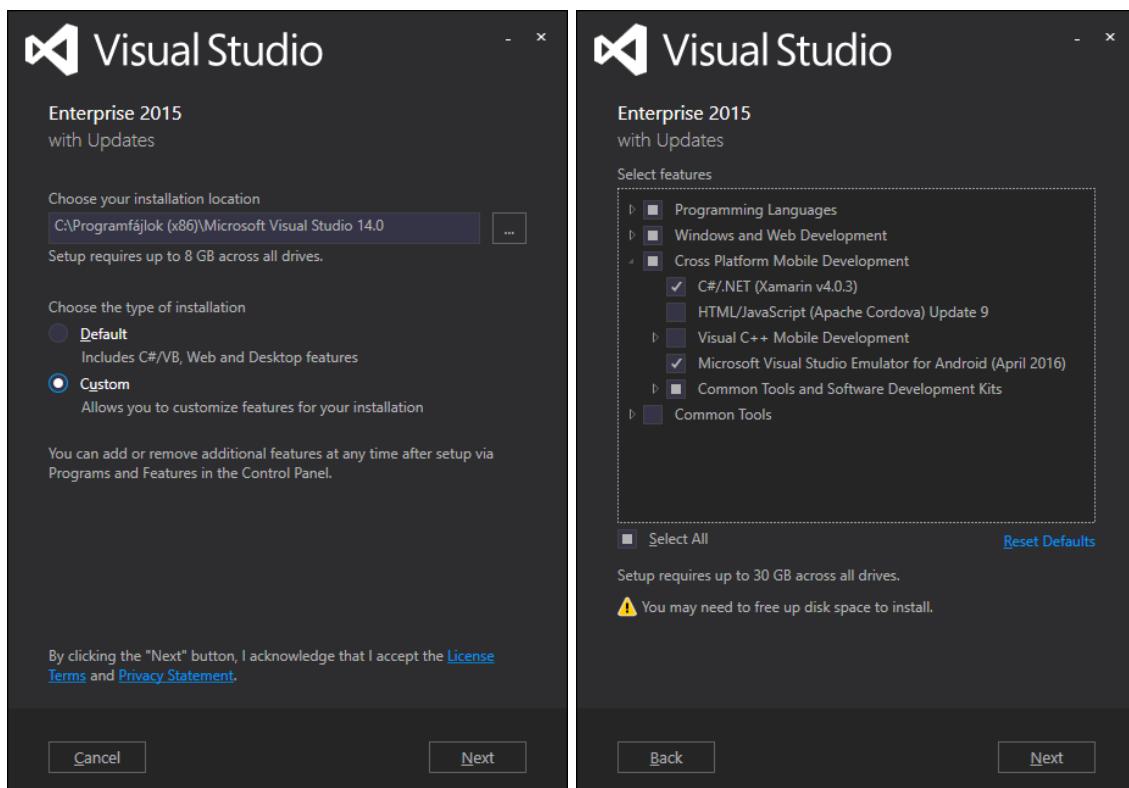
A bekötött szakdolgozathoz mellékelt DVD lemezen lévő *melleklet\_dvd.zip* tömörített állomány *M\_2\_Telepitheto\_alkalmazascsomagok* mappájában találhatóak azok a fájlok, amik telepíthetők egy megfelelően konfigurált Android, Windows Phone 8.1 vagy Windows 10 Mobile operációs rendszerű telefonra. iOS operációs rendszerhez nem készült ilyen alkalmazáscsomag, mert nem rendelkezem a lehetőséget nyújtó Apple Developer Enterprise fejlesztői licenc-szel. Ez a melléklet a Diplomaterv Portál (<https://diplomaterv.vik.bme.hu/>) maximális megengedett fájlméretét meghaladja, ezért oda nem került feltöltésre.

További információk a csomagok telepítéséhez az alábbi weboldalakon találhatók:

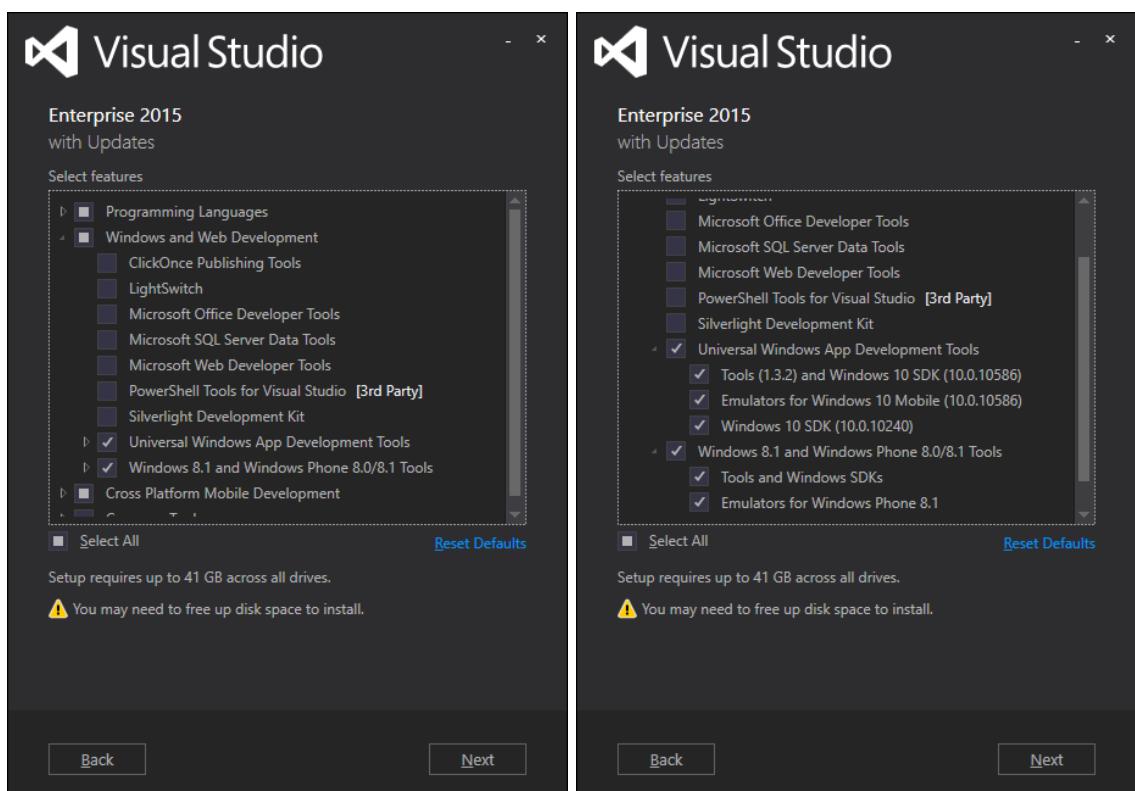
- Android: <http://www.cnet.com/how-to/how-to-install-apps-outside-of-google-play/>
- Windows Phone 8.1: <https://msdn.microsoft.com/en-us/library/windows/apps/xaml/dn832613.aspx>
- Windows 10 Mobile: <https://msdn.microsoft.com/windows/uwp/packaging/install-universal-windows-apps-with-the-winappdeploycmd-tool>

# Függelék

## F.1. A Visual Studio 2015 Update 2 komponenstelepítési beállításai

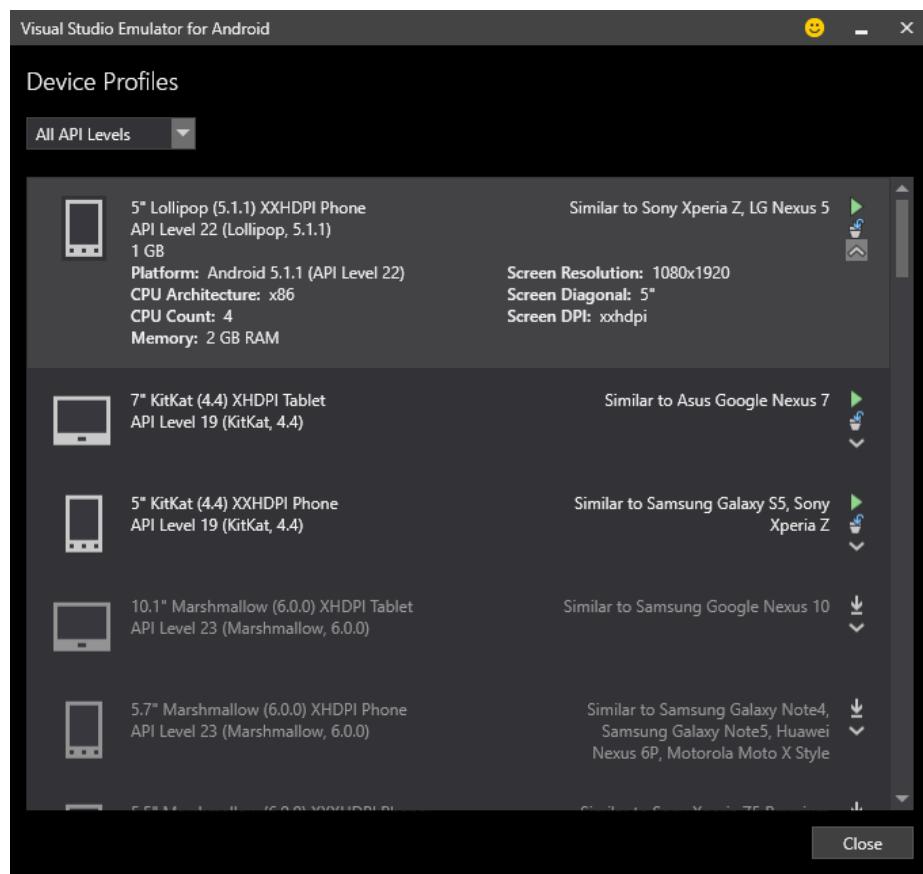


F.1.1. ábra. A telepítendő komponensek kiválasztása 1

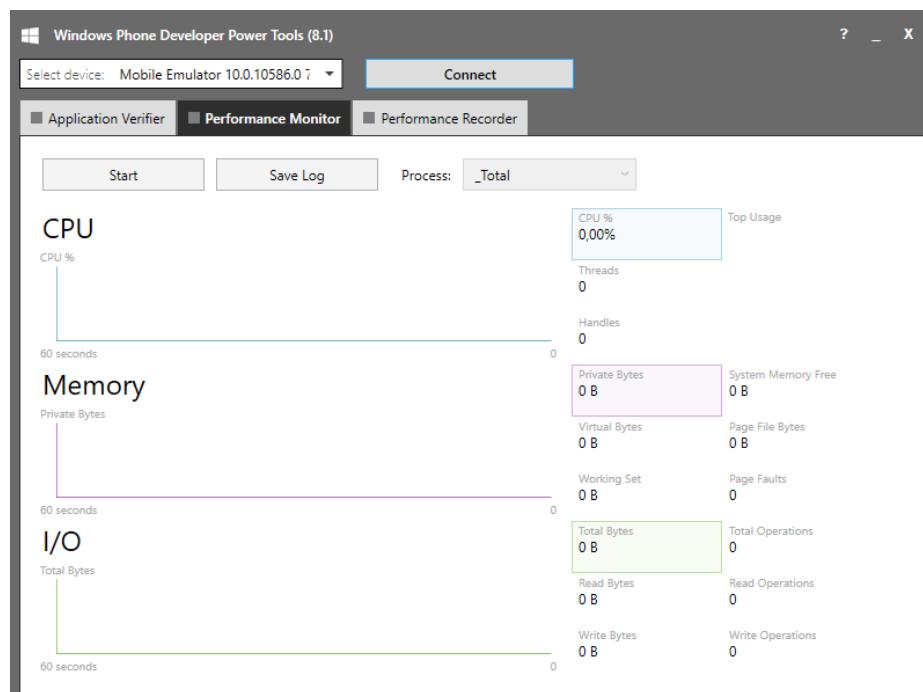


F.1.2. ábra. A telepítendő komponensek kiválasztása 2

## F.2. A Visual Studio emulátorokat kezelő szoftverek



F.2.1. ábra. A Visual Studio Emulator for Android program



F.2.2. ábra. A Windows Phone Developer Power Tools 8.1 program

### F.3. Példakód az adatkötés (data binding) szemléltetésére

**F.3.1. kódrészlet.** A barátlista keresőszávjának deklarációja adatkötéssel XAML-ben

```
<SearchBar x:Name="FriendSearchBar"
            Placeholder="Search friend"
            Text="{Binding SearchFilter}"/>
```

**F.3.2. kódrészlet.** A kötött adat a FriendListViewModel osztályban

```
public string SearchFilter
{
    get
    {
        return _searchFilter;
    }
    set
    {
        _searchFilter = value;
        OnPropertyChanged(nameof(SearchFilter));
        FilterContacts();
    }
}
```

**F.3.3. kódrészlet.** A keresés gomb megnyomásának kezelése a mögöttes kódban

```
public FriendListPage()
{
    // ...
    // kötési kontextus beállítása a View Modelre
    BindingContext = ViewModel = new FriendListViewModel();

    // feliratkozás a "keresés gomb megnyomása" eseményre
    FriendSearchBar.SearchButtonPressed += FriendSearchBar_SearchButtonPressed;
    // ...
}

// eseménykezelés
private void FriendSearchBar_SearchButtonPressed(object sender, EventArgs e)
{
    ViewModel.SearchFilter = (sender as SearchBar).Text;
    ViewModel.FilterContacts();
}
```

#### F.4. Példakód stílustulajdonság adatkötésére statikus változóhoz az x:Static XAML leíróbővítmény használatával

F.4.1. kódrészlet. A feliratok betűméretének platformfüggő deklarációja statikus változóval C#-ban

```
namespace SmartActive
{
    public partial class App : Application
    {
        //...
        public static readonly double SaMediumFontSize = Device.OnPlatform(
            Android: Device.GetNamedSize(NamedSize.Medium, typeof(Label)),
            WinPhone: Device.GetNamedSize(NamedSize.Small, typeof(Label)),
            iOS: Device.GetNamedSize(NamedSize.Medium, typeof(Label)));
        //...
    }
}
```

F.4.2. kódrészlet. Statikus változó adatkötése a feliratok egy explicit stílusához az x:Static leíróbővítmény használatával

```
<?xml version="1.0" encoding="utf-8" ?>
<Application xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:SmartActive;assembly=SmartActive"
    x:Class="SmartActive.App">
    //...
    <Style x:Key="SaLabelMediumNoWrapTextStyle" TargetType="Label">
        <Setter Property="FontSize" Value="{x:Static local:App.SaMediumFontSize}"/>
        //...
    </Style>
    //...
</Application>
```

## F.5. Példakód elemtulajdonságok adatkötésére statikus változókhöz az x:Static XAML leíróbővítmény használatával

### F.5.1. kódrészlet. Gomb tulajdonságok deklarációja statikus változókban

```
namespace SmartActive.Resources
{
    public static class ButtonResources
    {
        public static readonly double FontSize = Device.OnPlatform(
            Android: Device.GetNamedSize(NamedSize.Small, typeof(Button)),
            WinPhone: Device.GetNamedSize(NamedSize.Micro, typeof(Button)),
            iOS: Device.GetNamedSize(NamedSize.Medium, typeof(Button)));

        public static readonly Color BorderColor = Device.OnPlatform(
            Android: Color.Default,
            WinPhone: Color.Default,
            iOS: (Color)App.Current.Resources["ContentColor"]);

        public static readonly int BorderRadius = Device.OnPlatform(
            Android: 0,
            WinPhone: 1,
            iOS: 15);

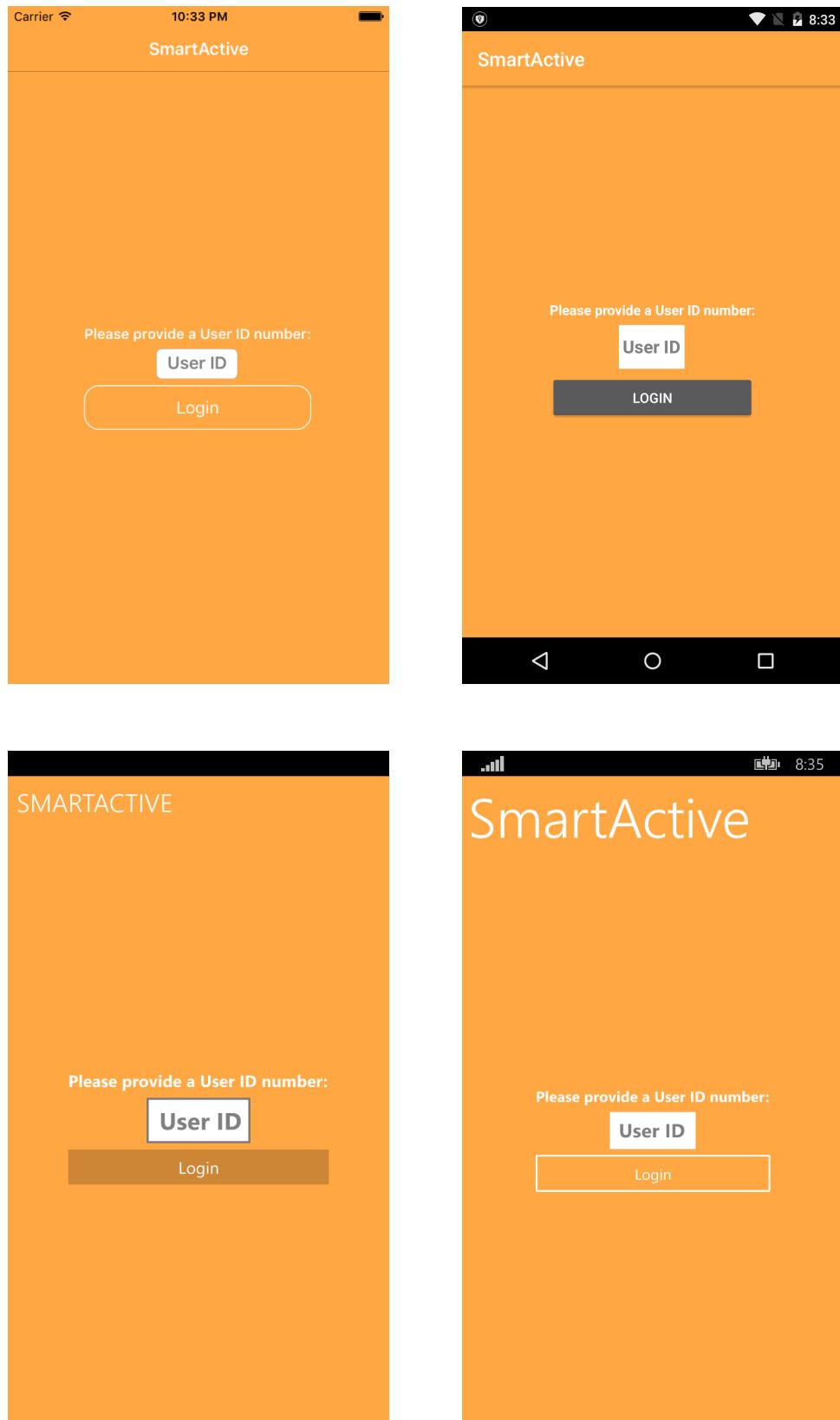
        public static readonly double BorderWidth = Device.OnPlatform(
            Android: 0.0,
            WinPhone: 2.0,
            iOS: 1.0);

        public static readonly string AddFriendButtonText = Device.OnPlatform(
            Android: "Add Friend",
            WinPhone: "Add Friend",
            iOS: " Add Friend ");
        //...
    }
}
```

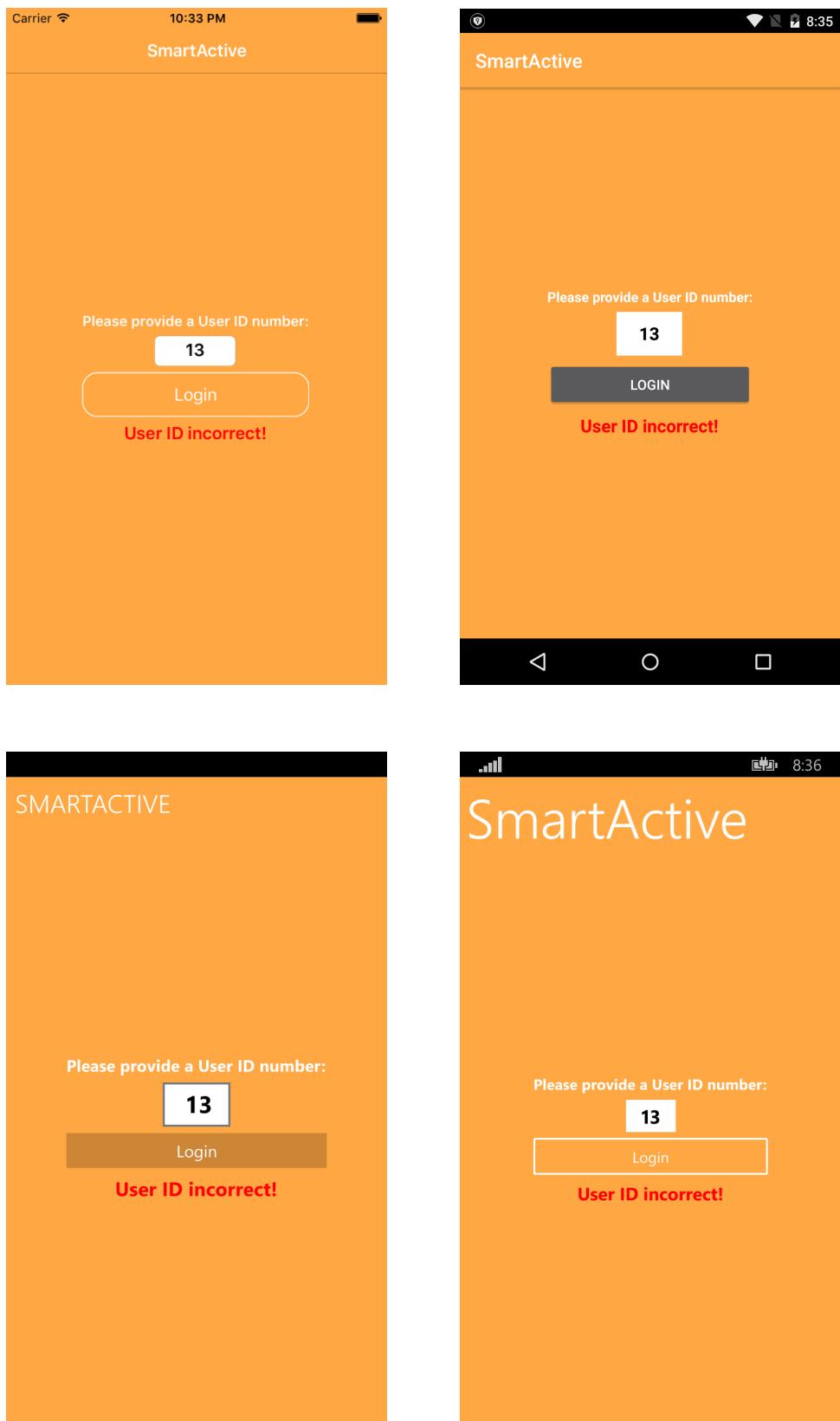
### F.5.2. kódrészlet. Gomb tulajdonságok adatkötése az x:Static leíróbővítmény használatával

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:res="clr-namespace:SmartActive.Resources;assembly=SmartActive"
    x:Class="SmartActive.Pages.LoginPage"
    Style="{StaticResource SaContentPageStyle}">
    //...
    <Button x:Name="LoginButton"
        Text="{x:Static res:ButtonResources.LoginButtonText}"
        TextColor="{StaticResource ContentColor}"
        FontSize="{x:Static res:ButtonResources.FontSize}"
        BorderColor="{x:Static res:ButtonResources.BorderColor}"
        BorderRadius="{x:Static res:ButtonResources.BorderRadius}"
        BorderWidth="{x:Static res:ButtonResources.BorderWidth}"/>
    //...
</ContentPage>
```

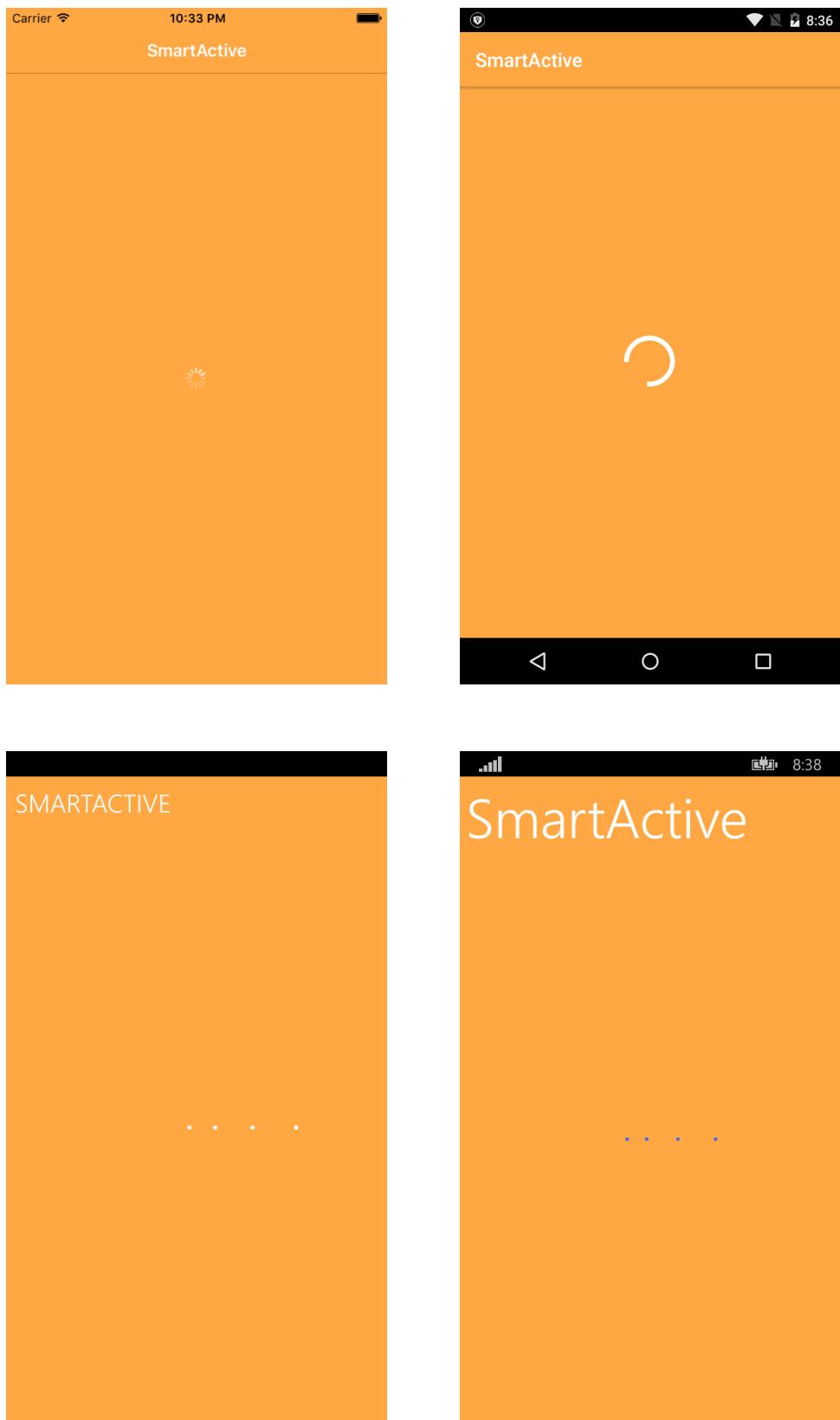
## F.6. Multi-platform képernyőképek az elkészült alkalmazásról



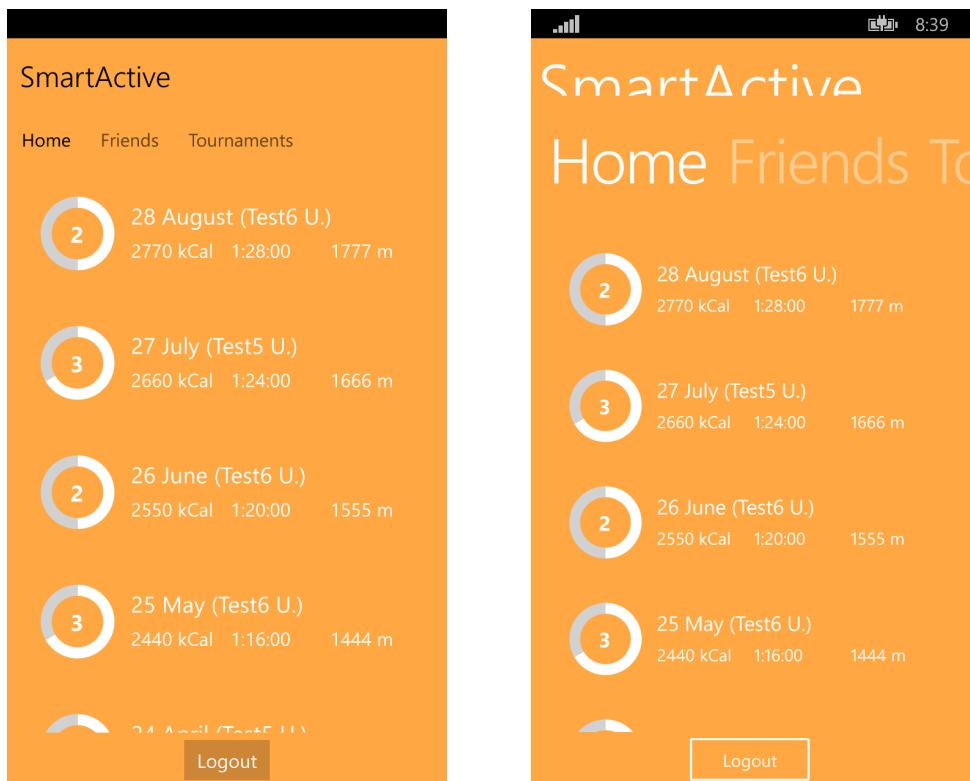
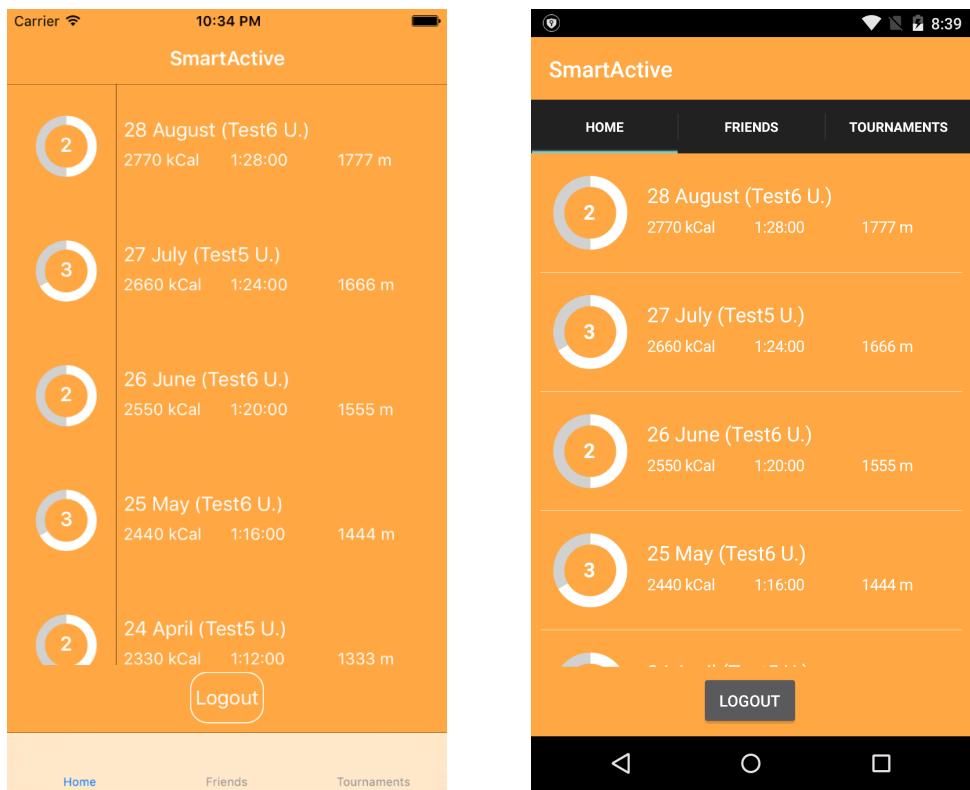
F.6.1. ábra. A bejelentkező-képernyő



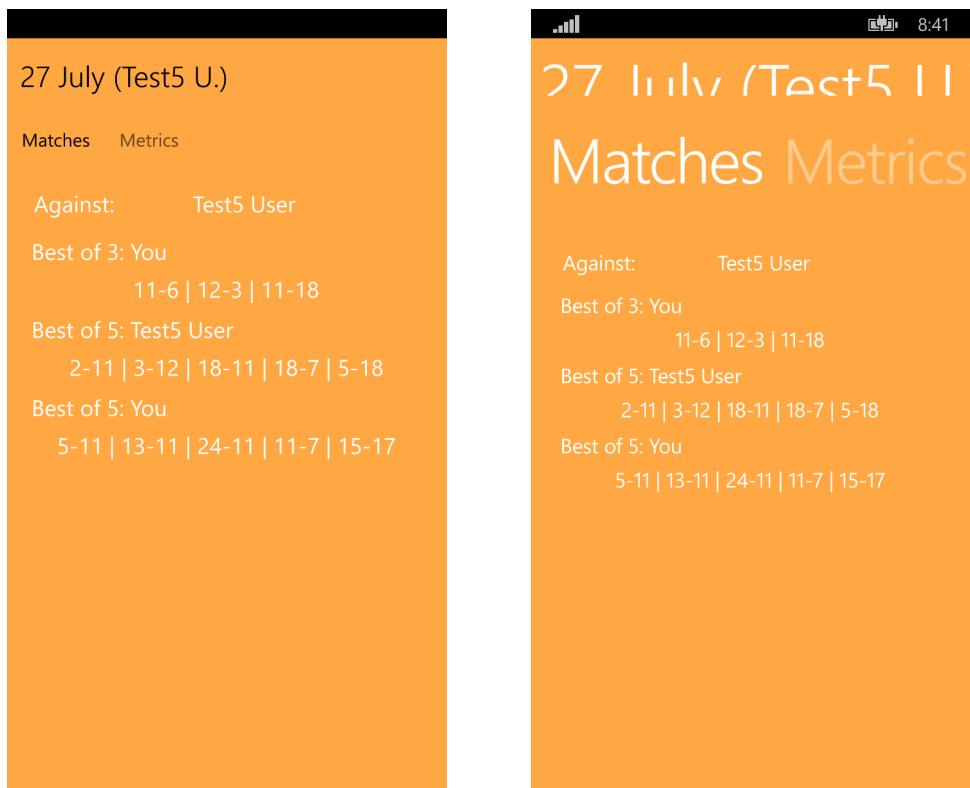
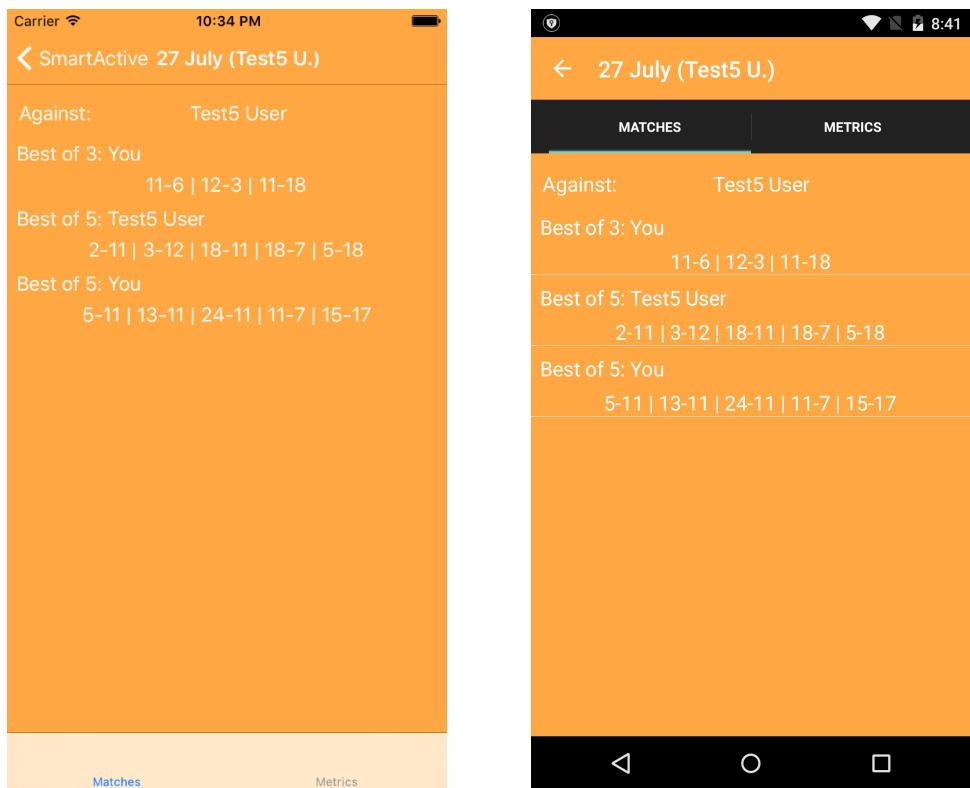
**F.6.2. ábra.** A bejelentkező-oldal hibaüzenete felhasználóazonosító megadásakor



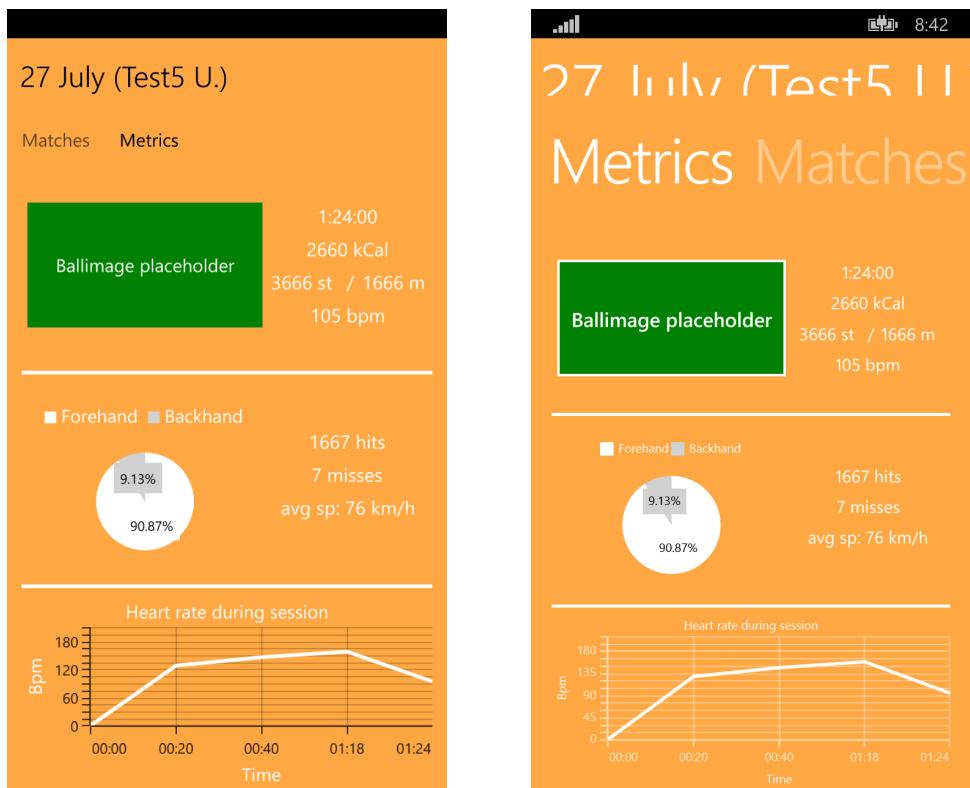
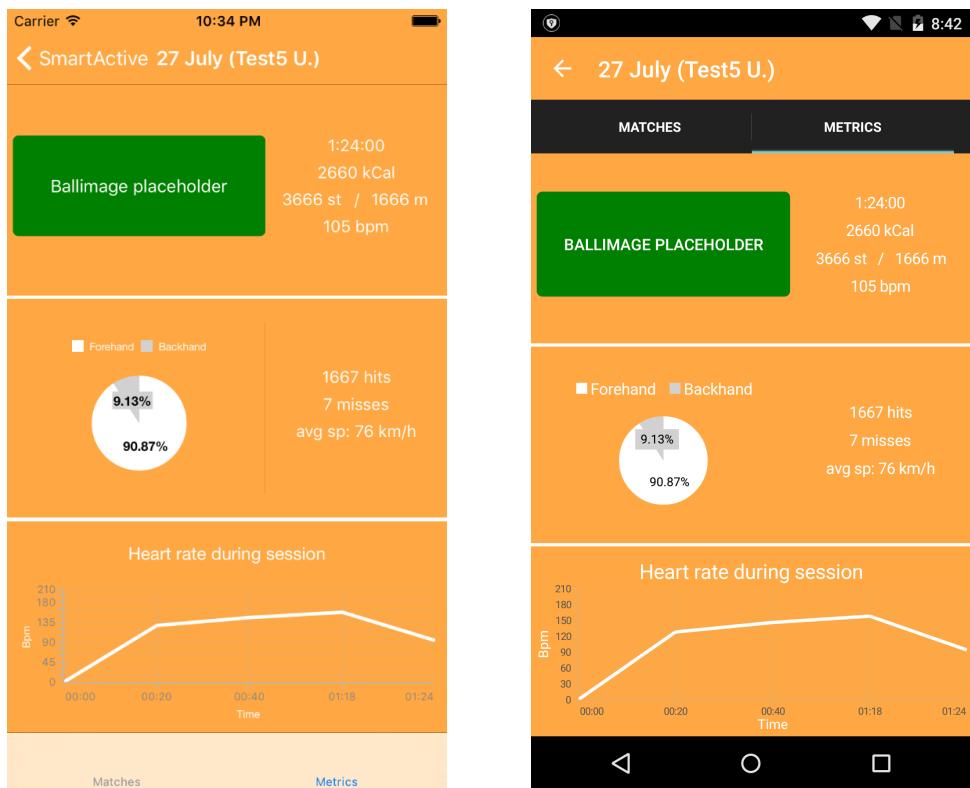
**F.6.3. ábra.** Töltési animáció a bejelentkezés alatt



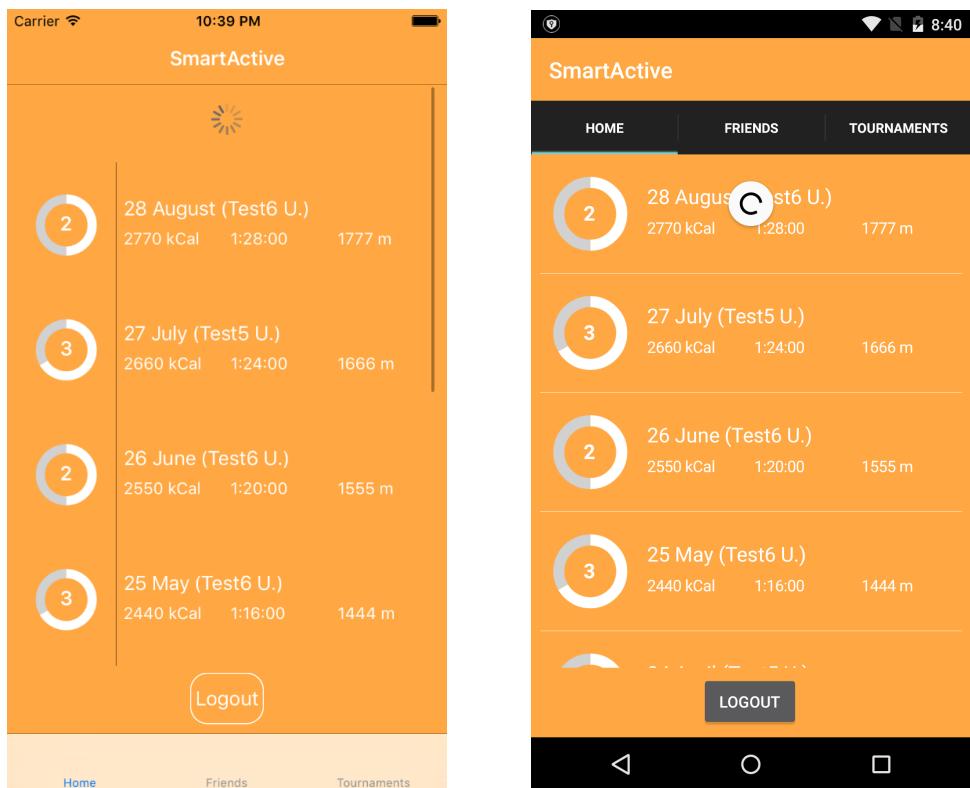
**F.6.4. ábra.** A korábbi edzések listája



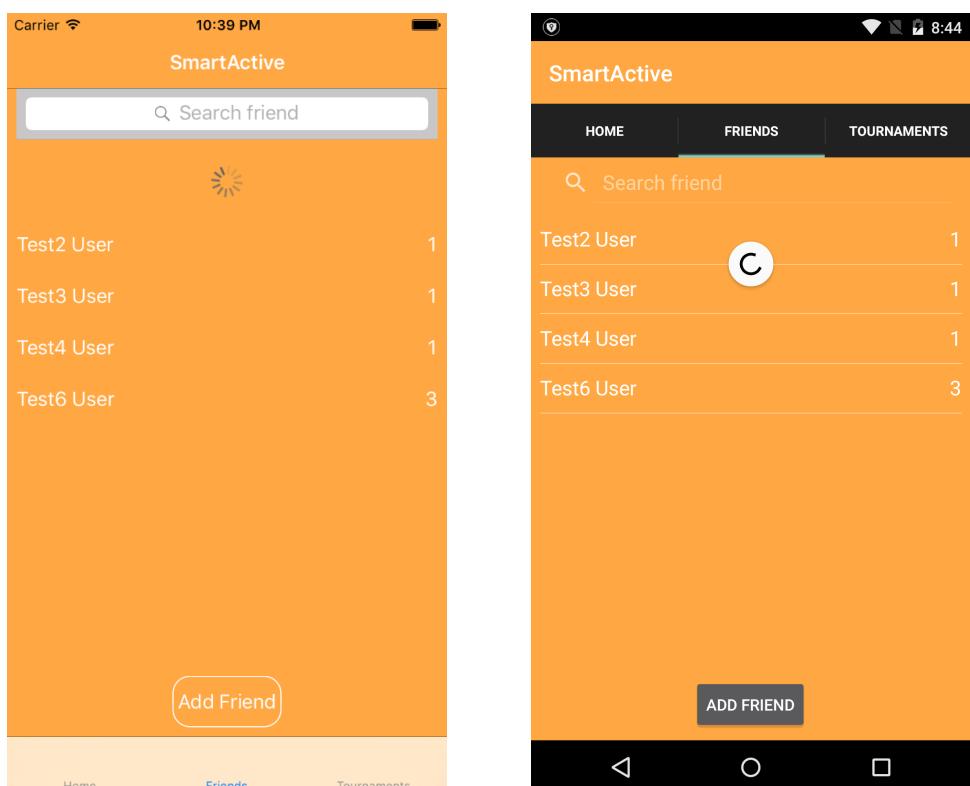
**F.6.5. ábra.** Egy edzés meccseinek eredményei



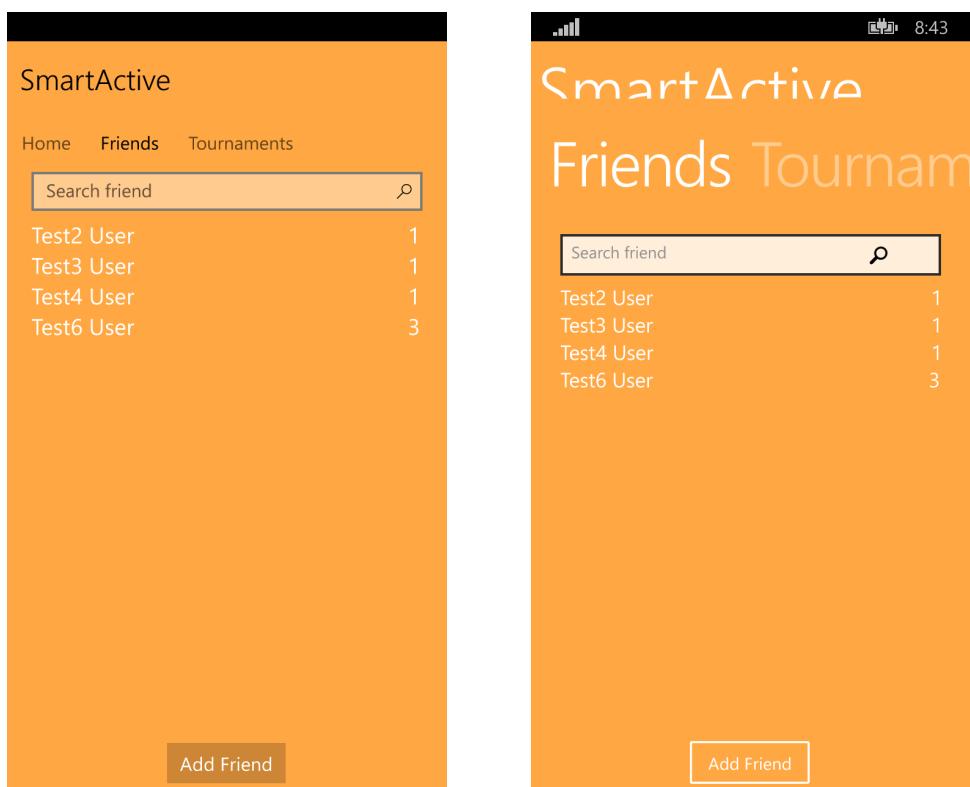
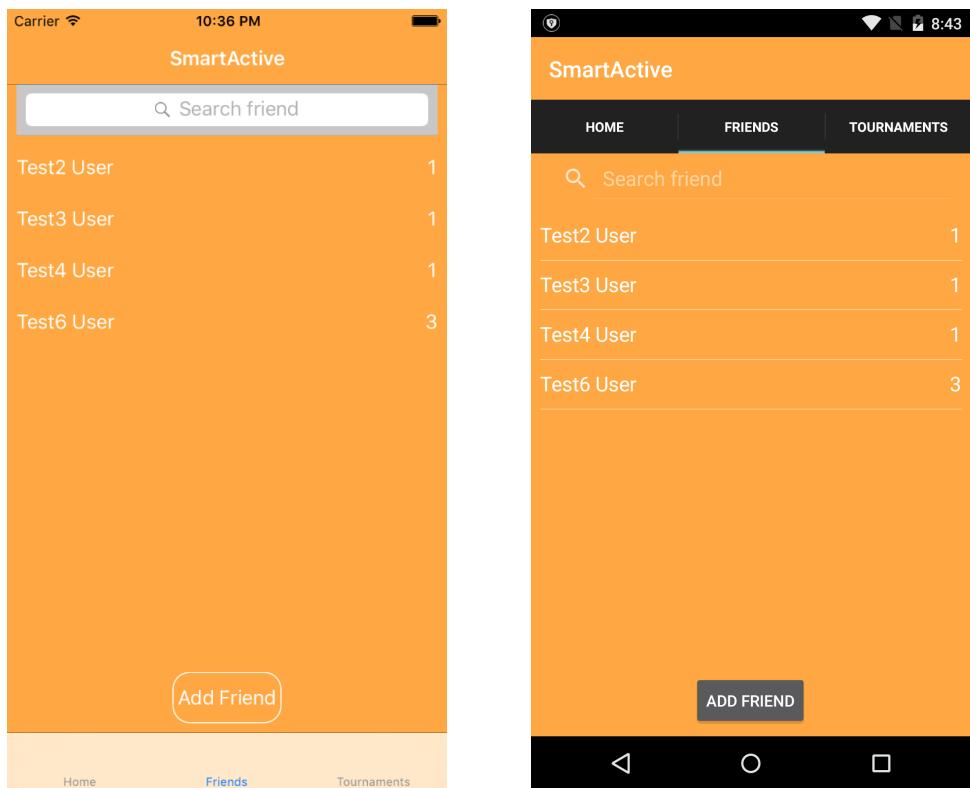
**F.6.6. ábra.** Egy edzésről készült statisztikák



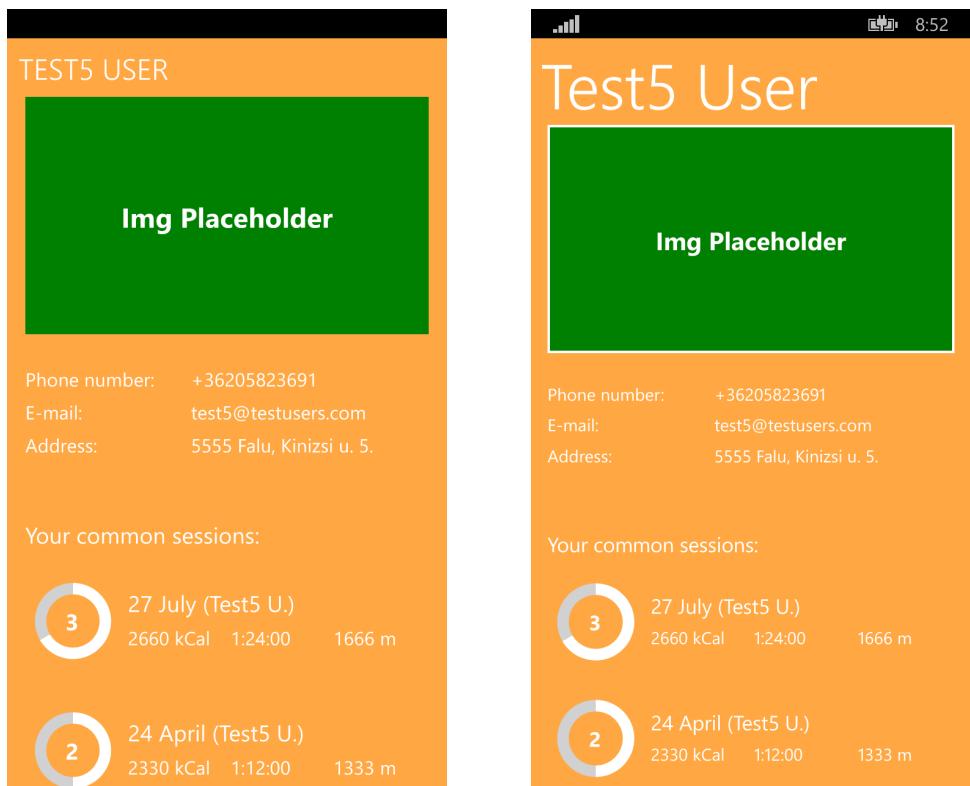
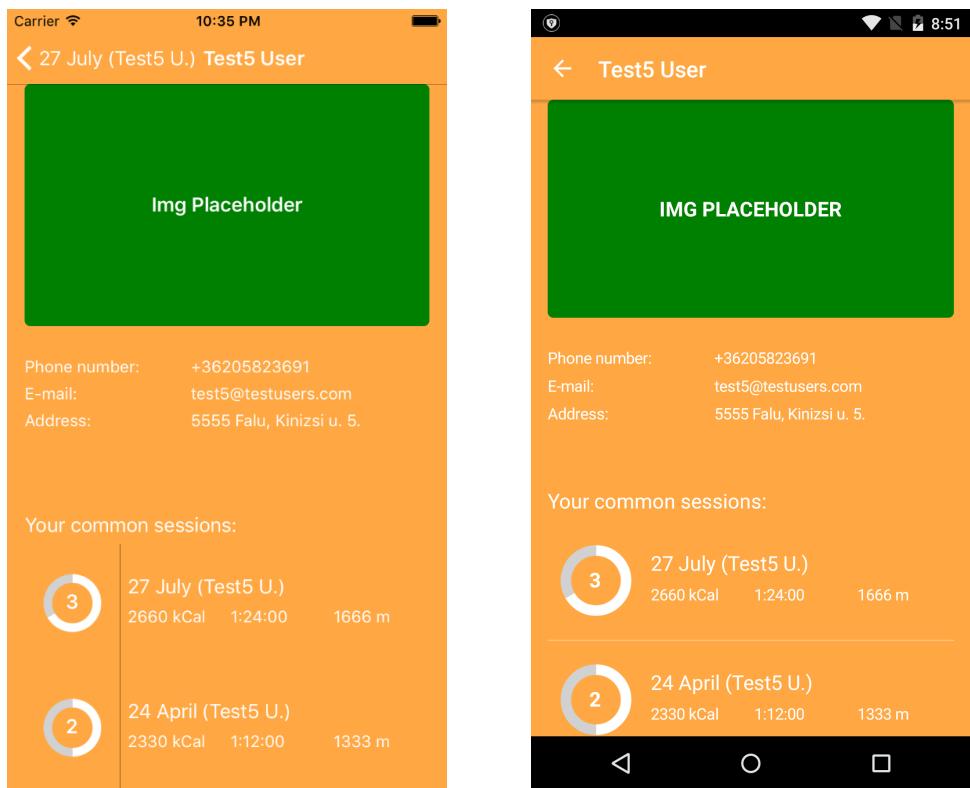
**F.6.7. ábra.** Az edzéslista szinkronizálása a szerverrel iOS és Android operációs rendszereken



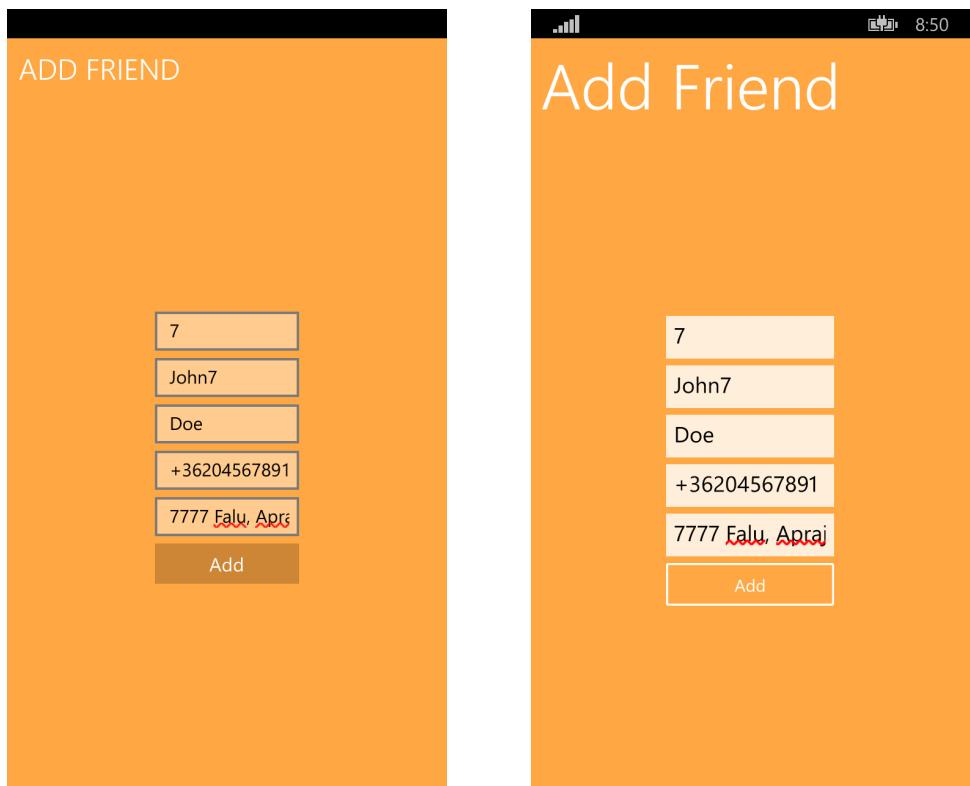
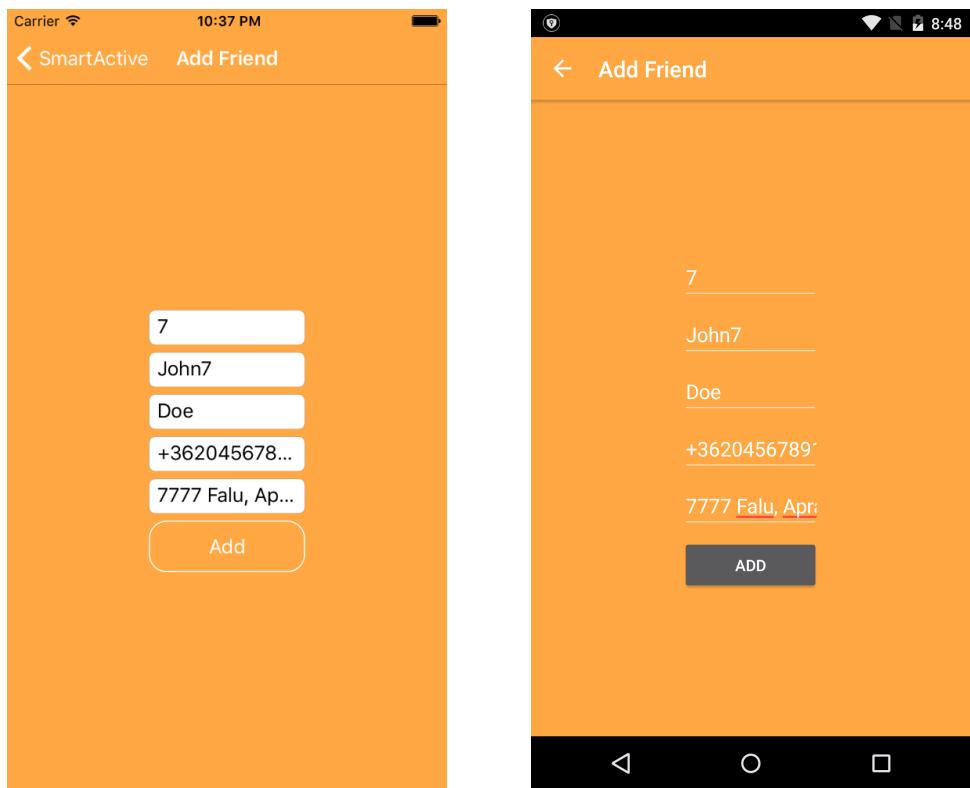
**F.6.8. ábra.** A barátlista szinkronizálása a szerverrel iOS és Android operációs rendszereken



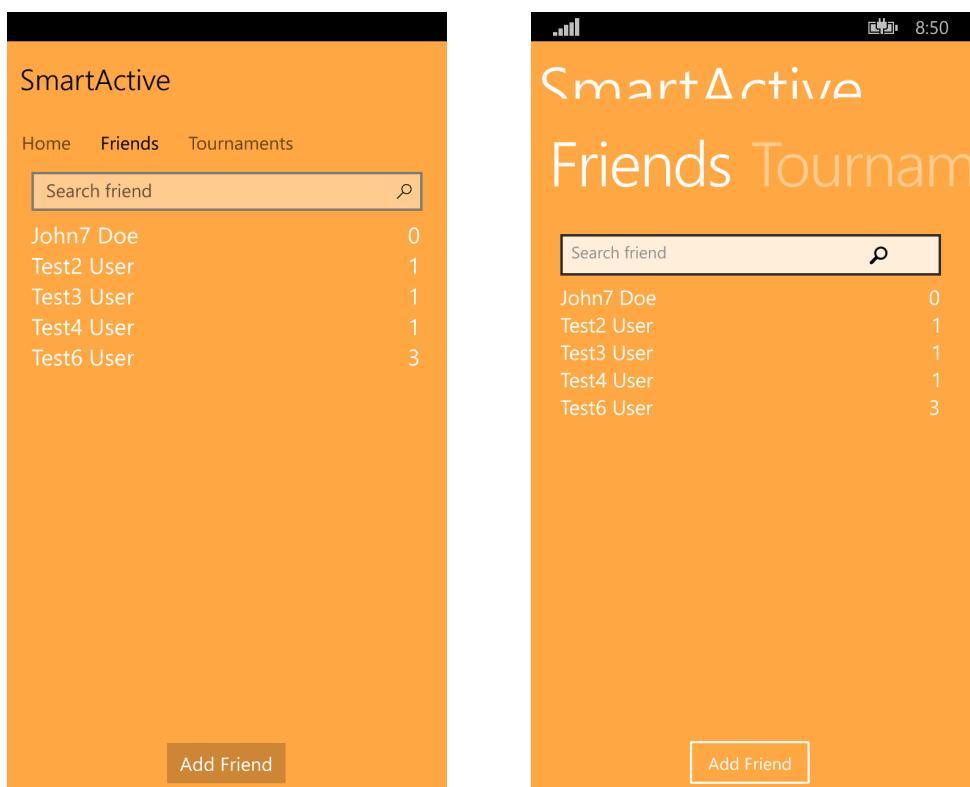
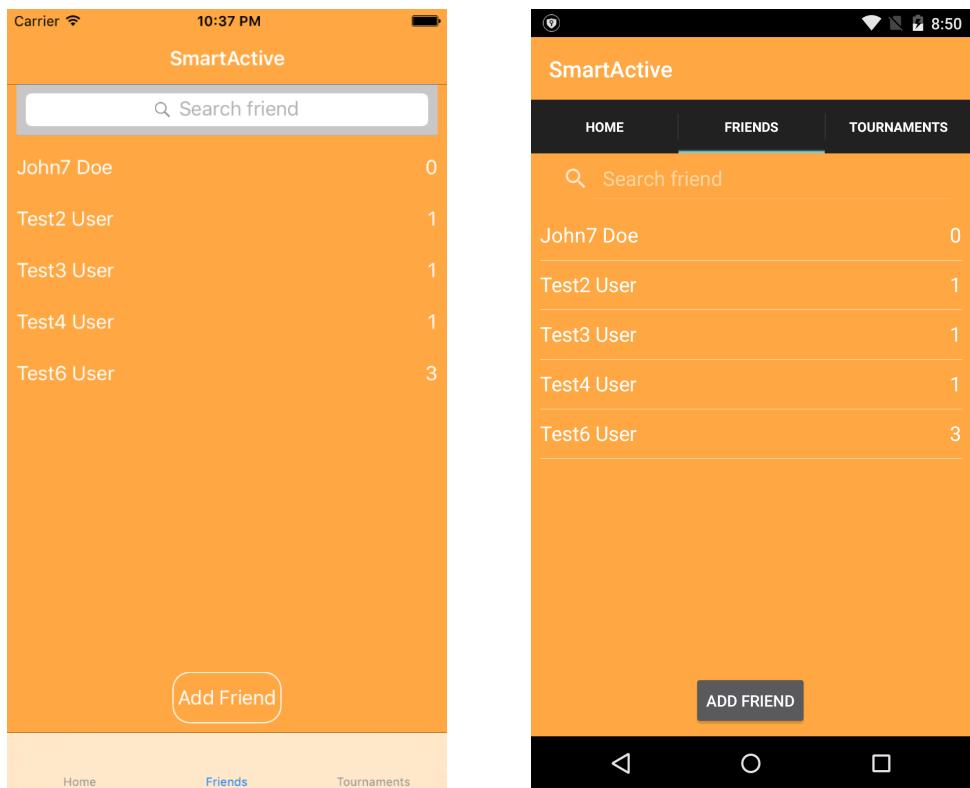
F.6.9. ábra. A barátok listája



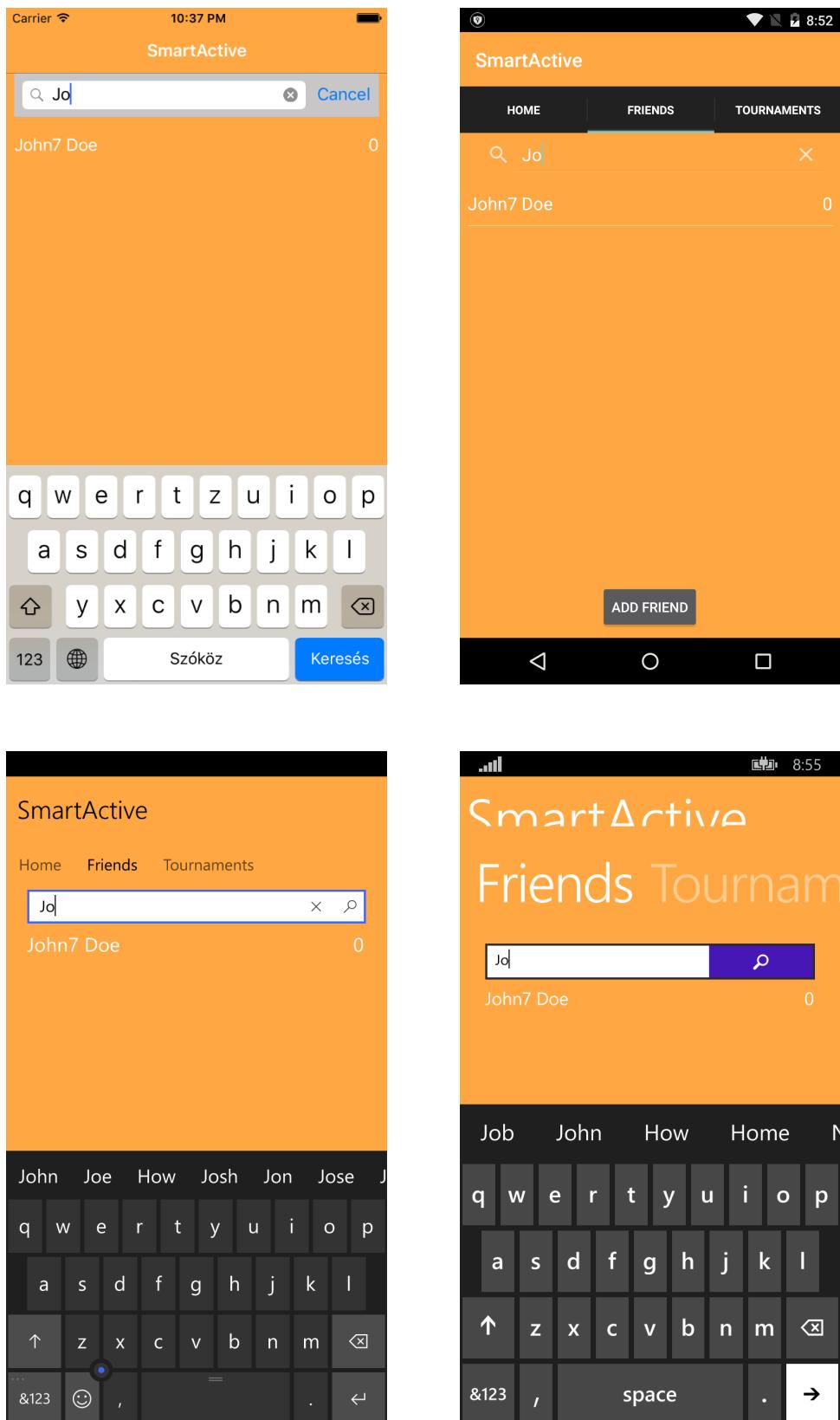
F.6.10. ábra. Részletes információk egy barátról



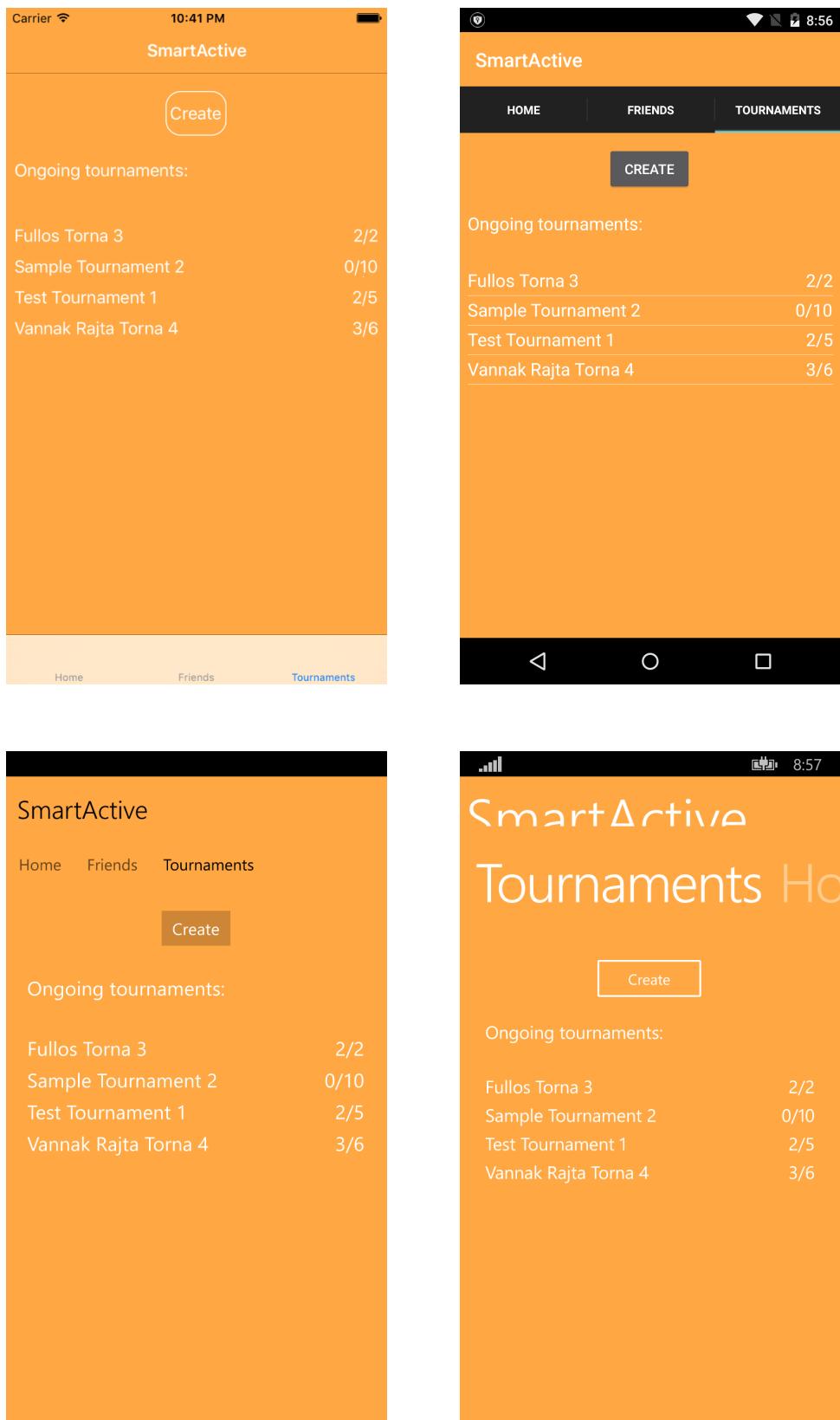
F.6.11. ábra. Új barát hozzáadása



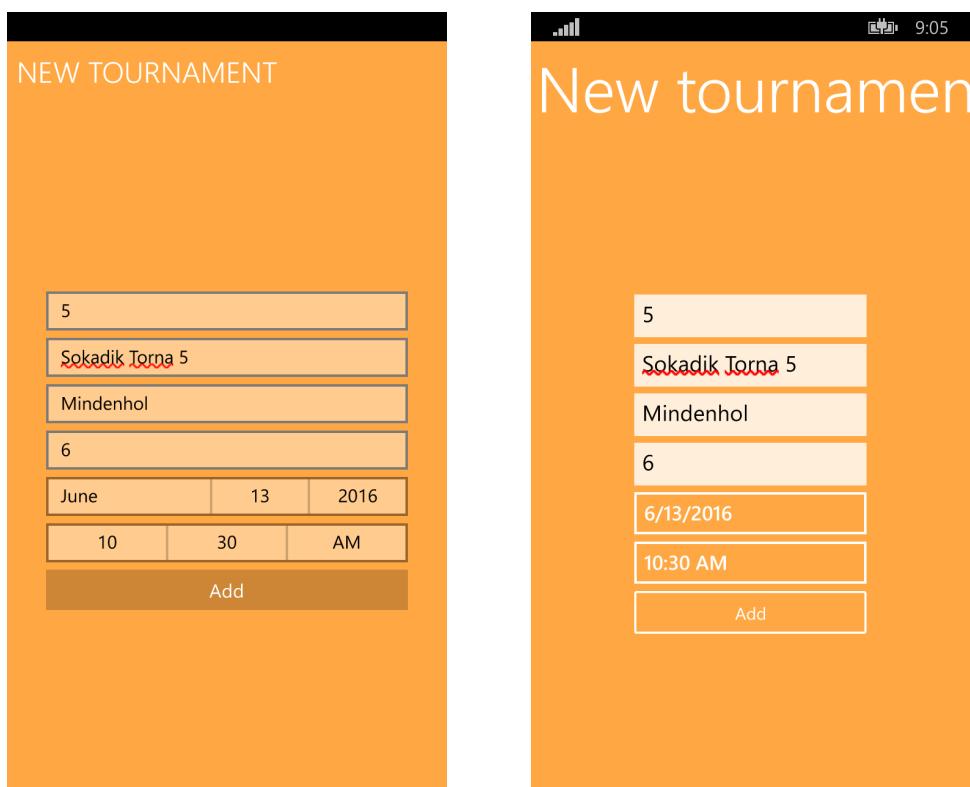
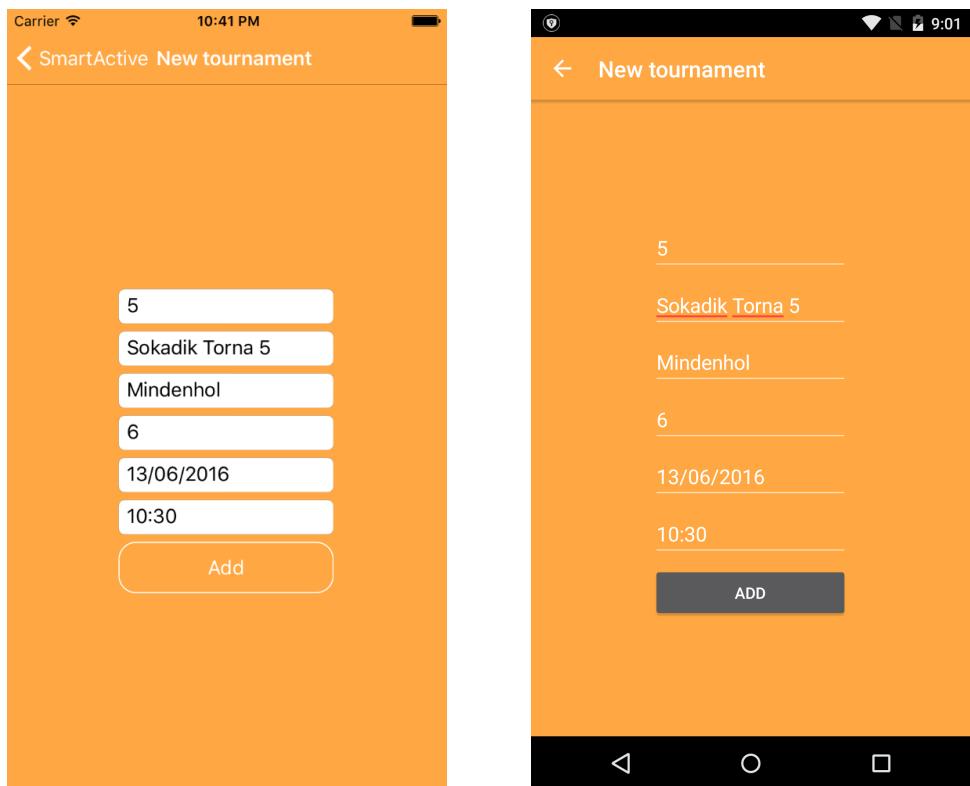
**F.6.12. ábra.** Hozzáadás után az új barát megjelenik a barátlistában



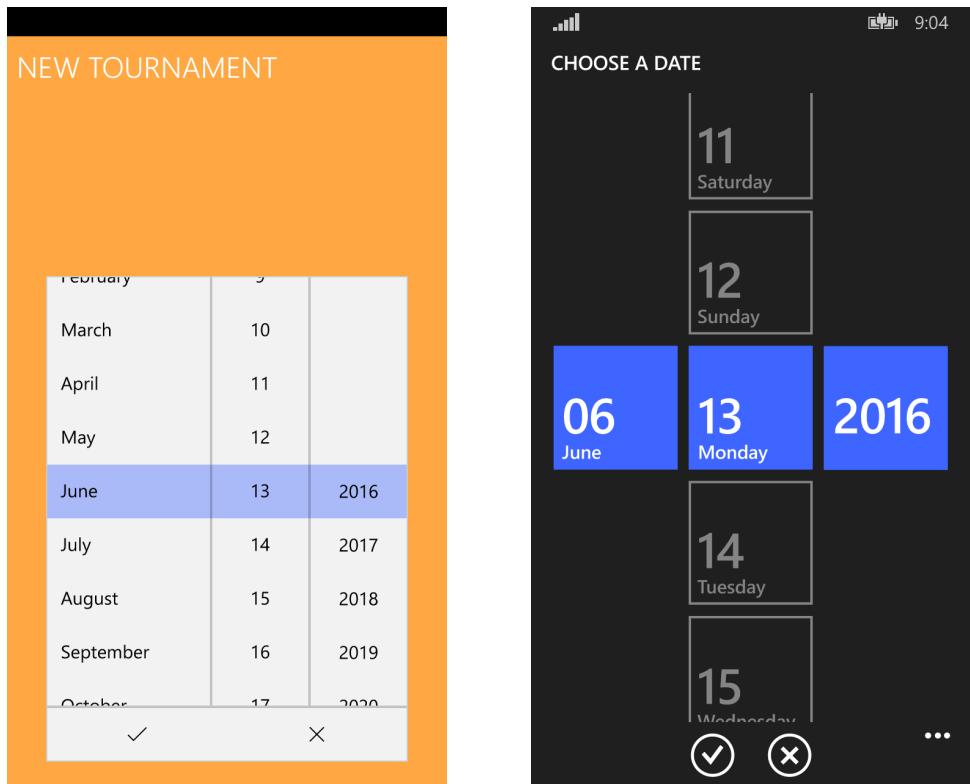
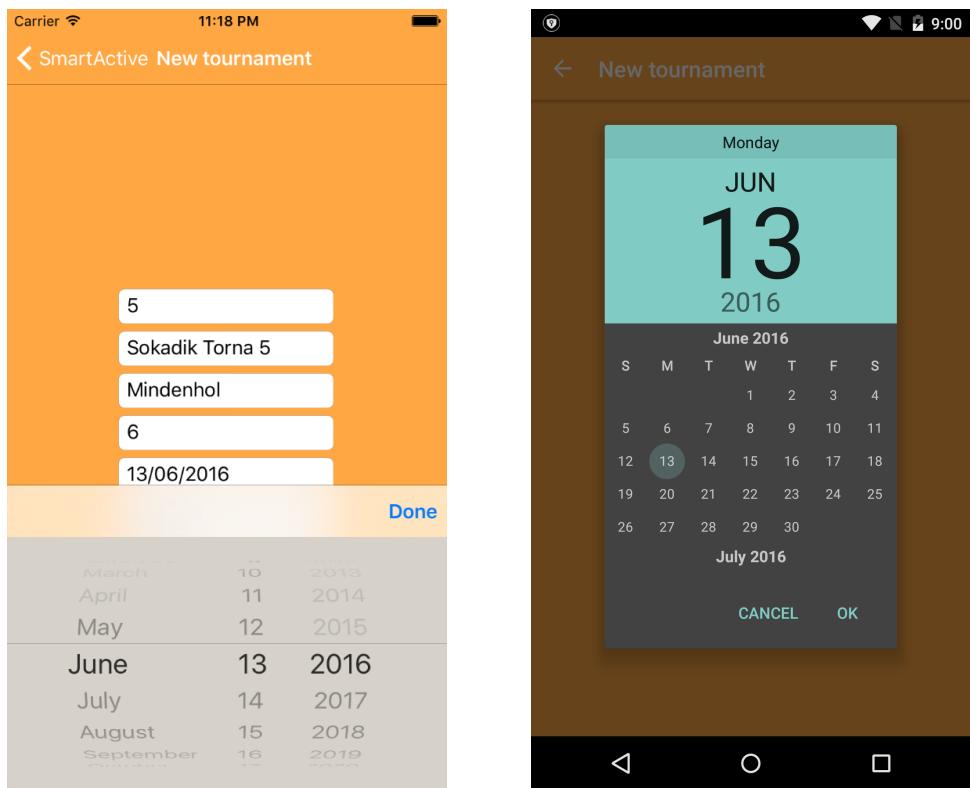
F.6.13. ábra. Keresés a barátok között



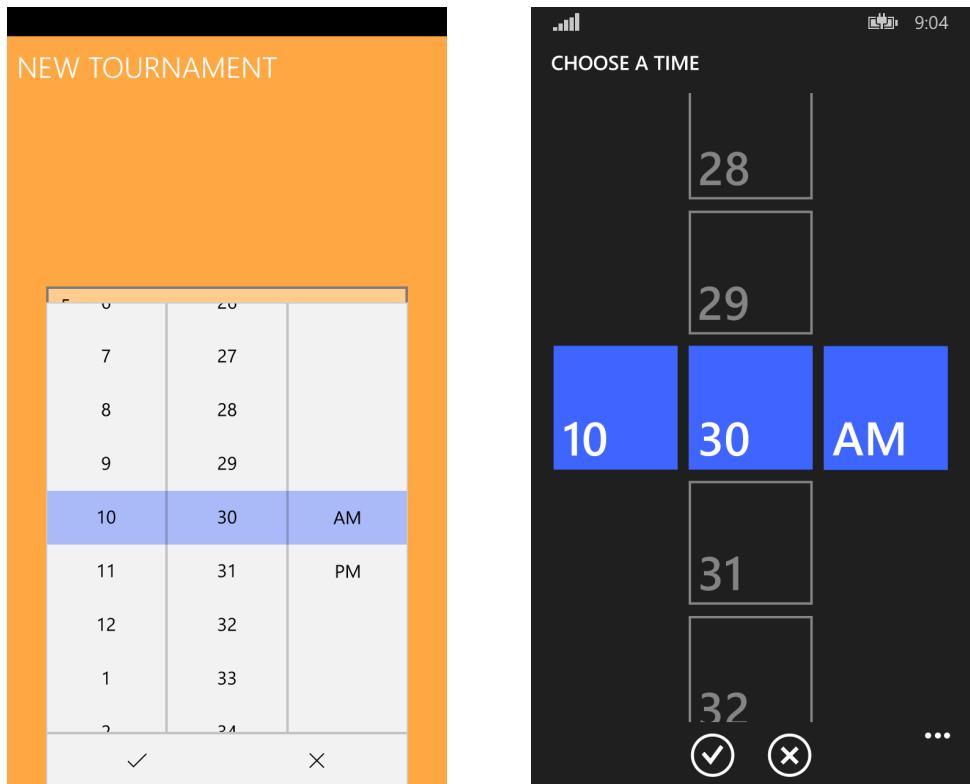
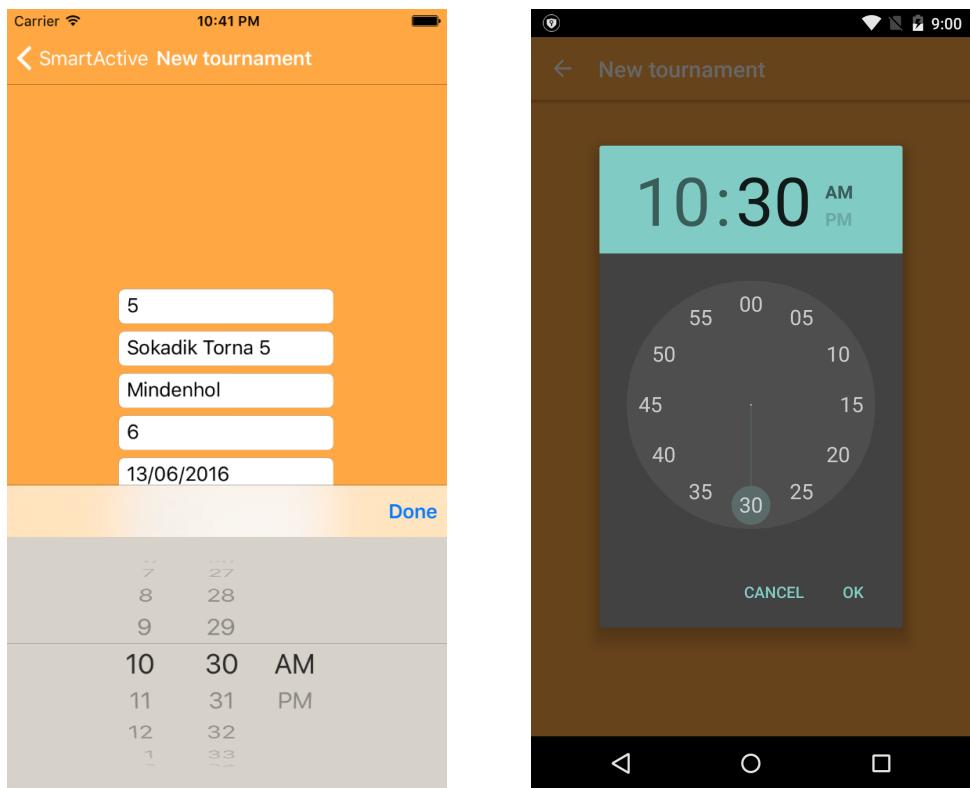
F.6.14. ábra. A bajnokságok oldala



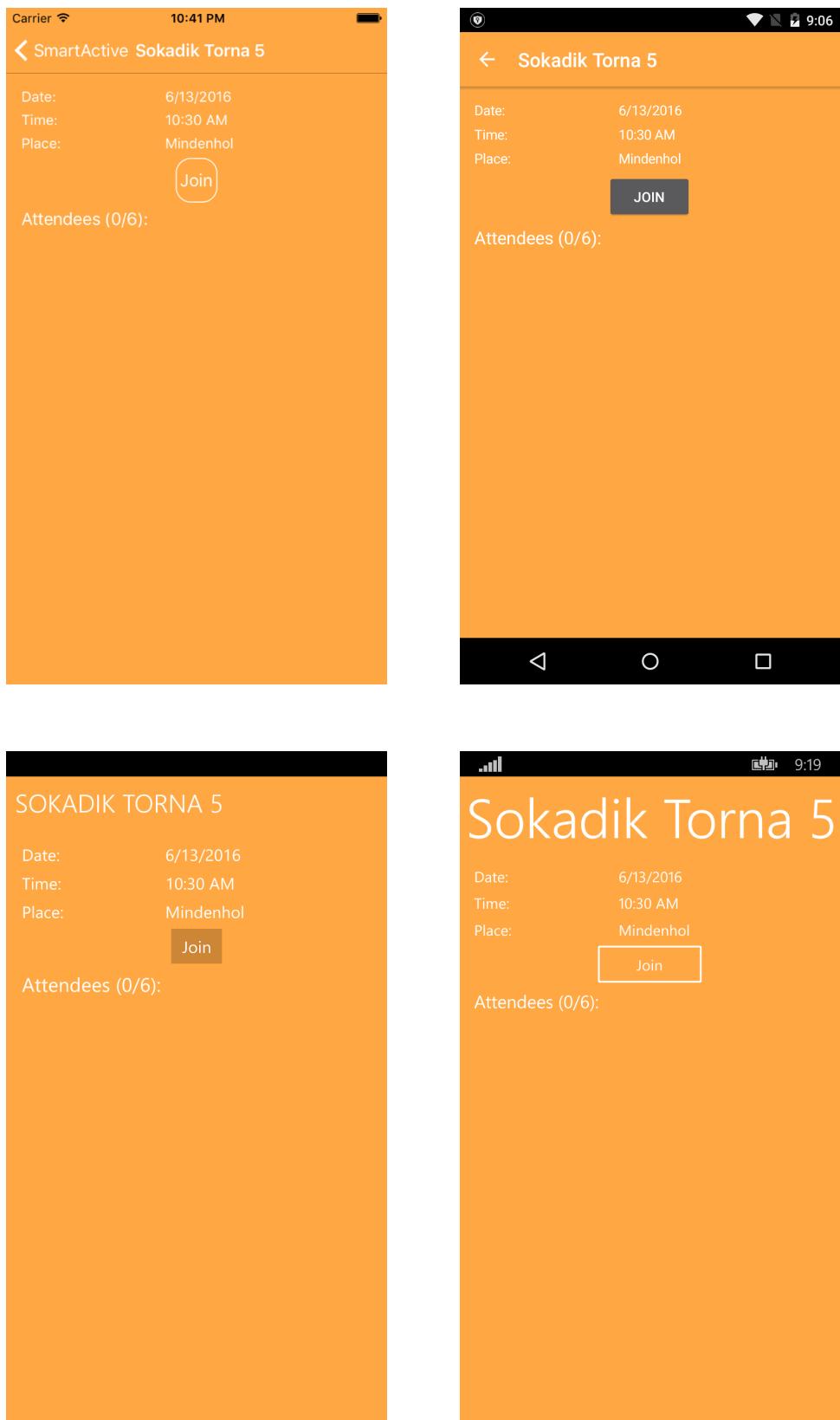
F.6.15. ábra. Új bajnokság létrehozása



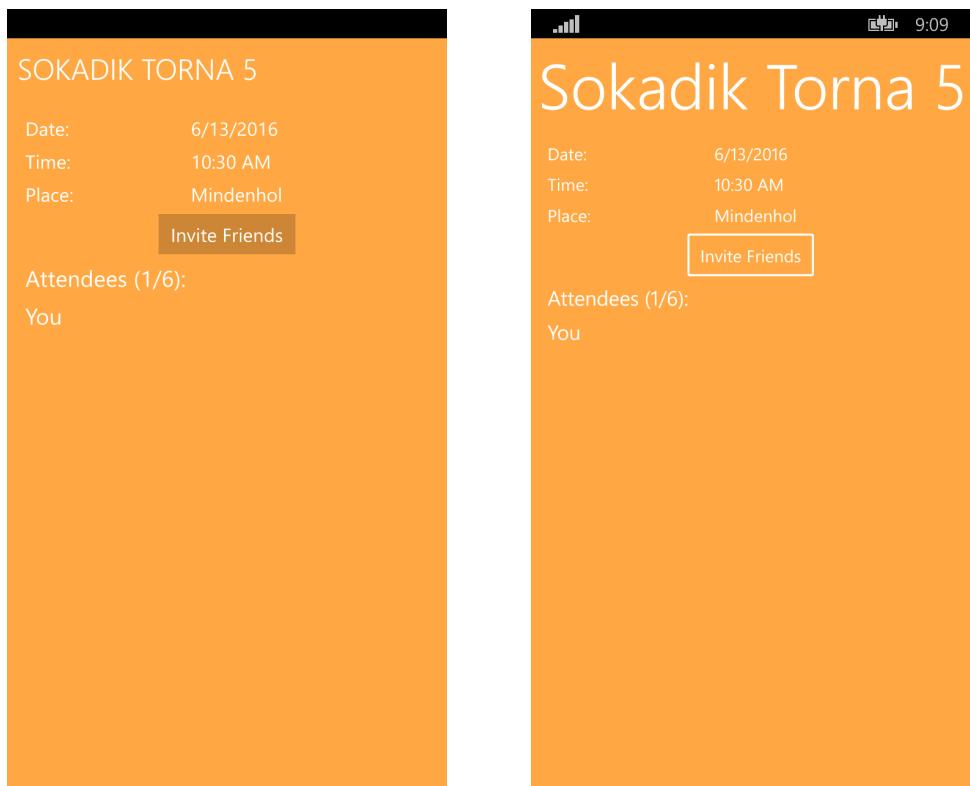
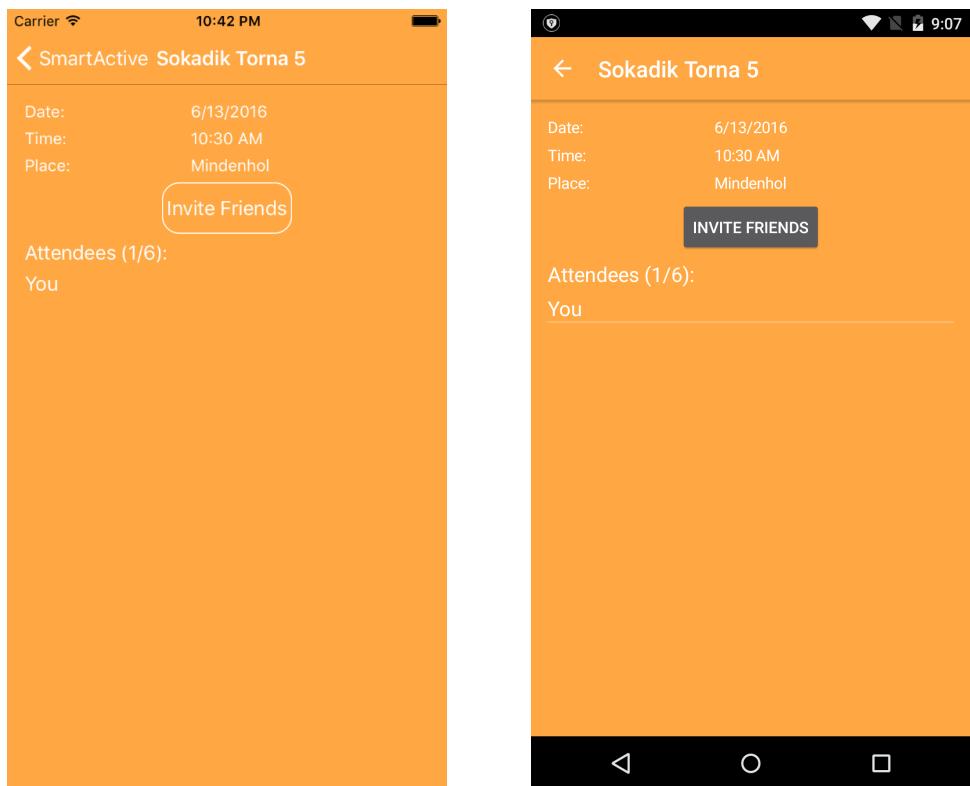
**F.6.16. ábra.** A dátum megadása a platform natív grafikus elemeivel történik



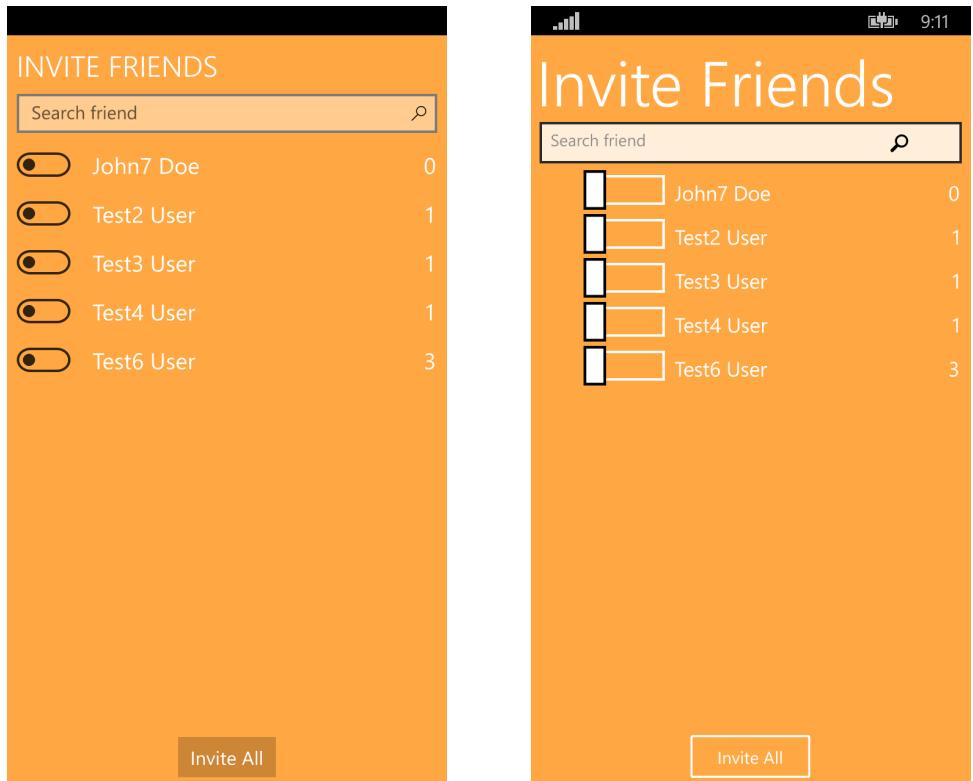
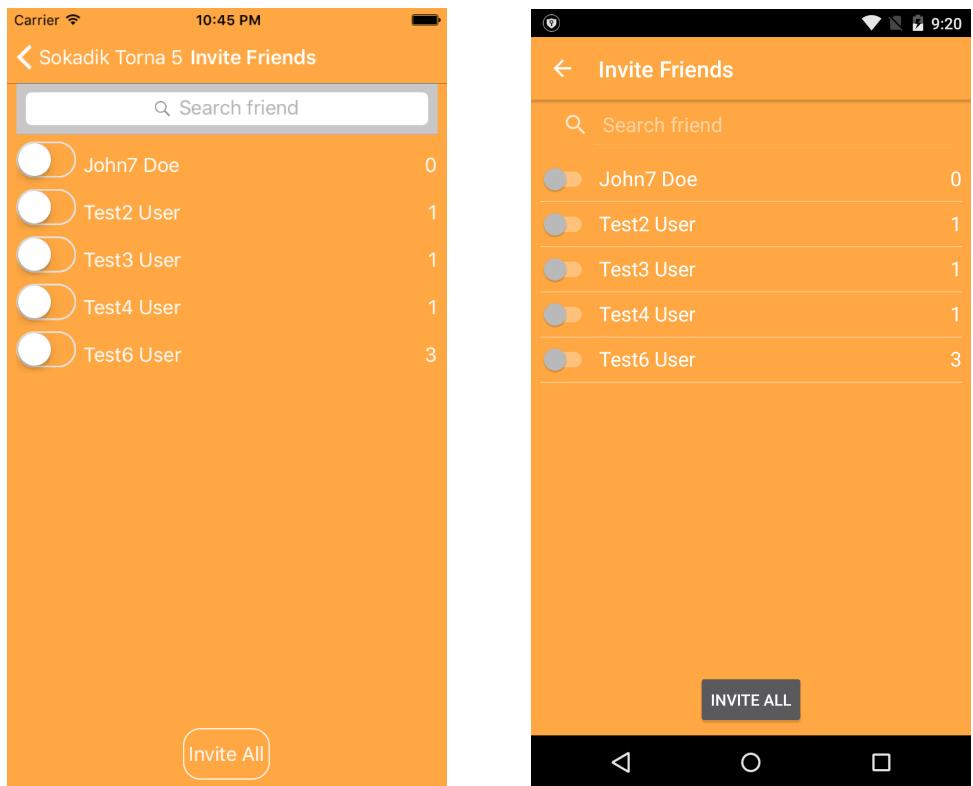
F.6.17. ábra. Az idő megadása a platform natív grafikus elemeivel történik



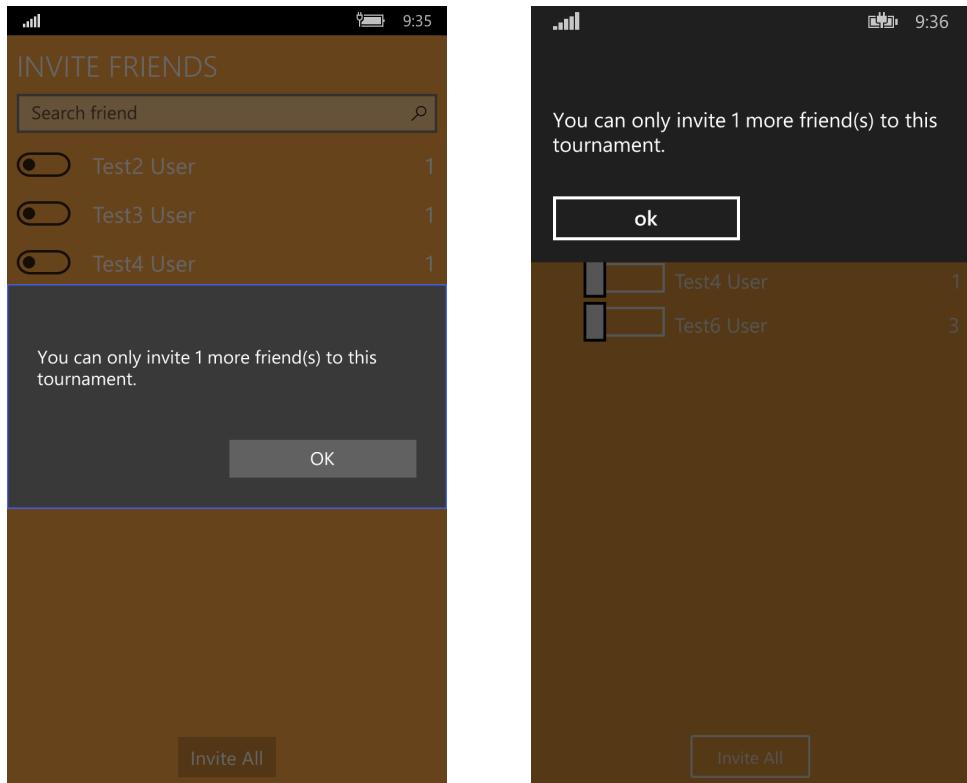
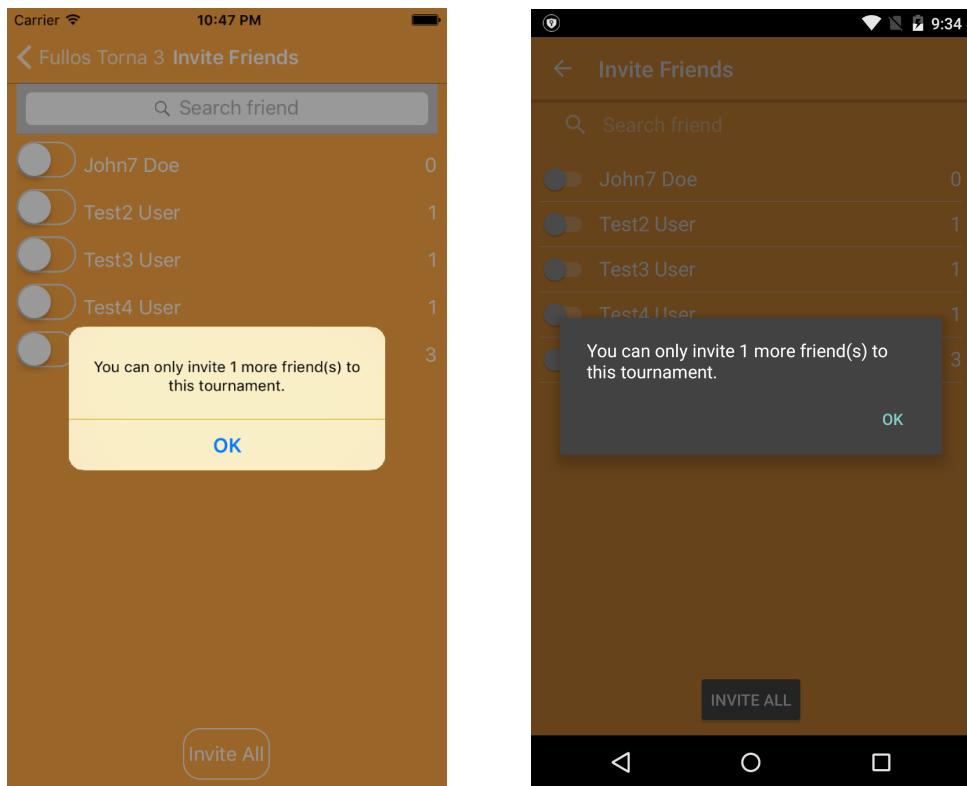
**F.6.18. ábra.** Az új bajnokság részletei



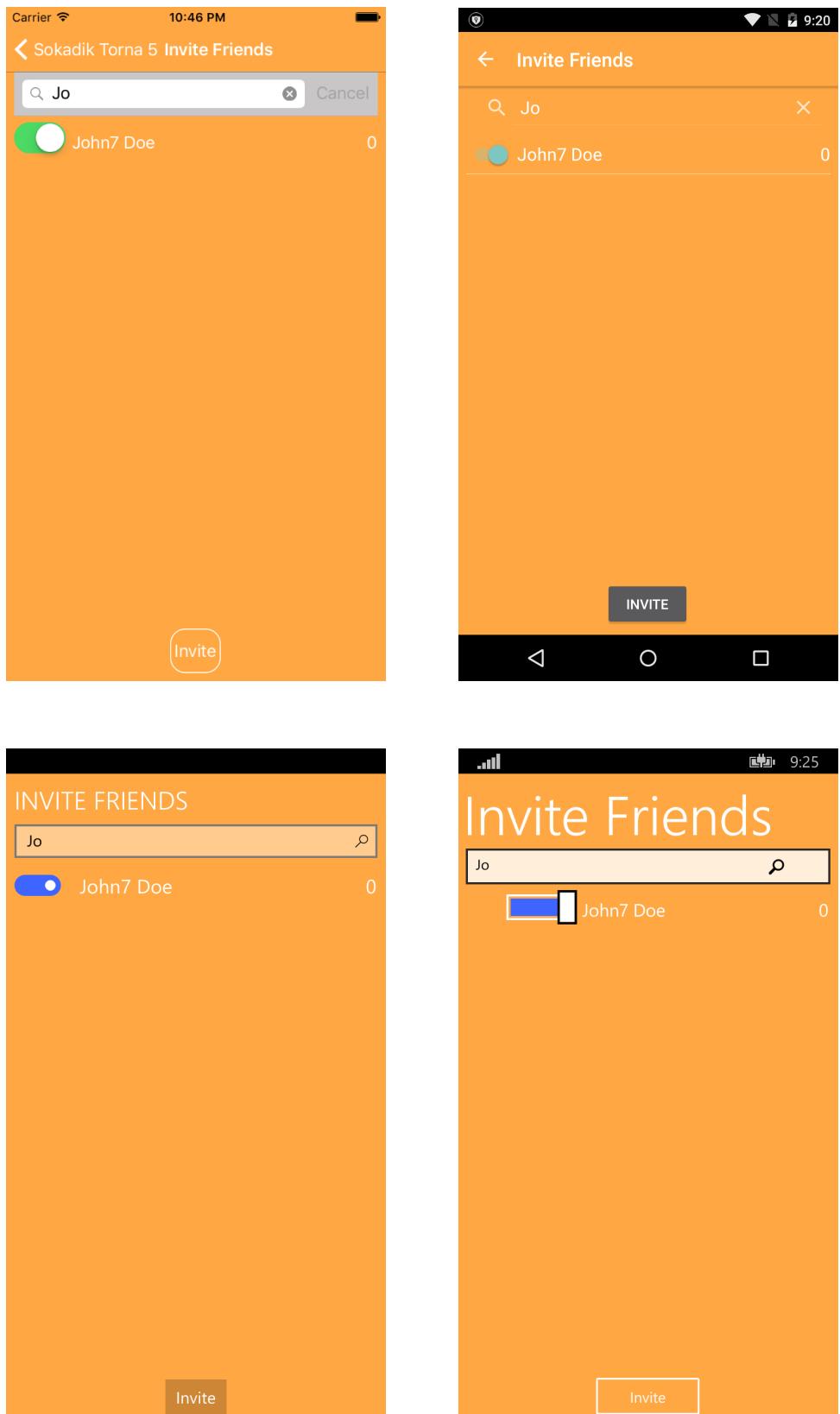
**F.6.19. ábra.** A bajnokságrészletező-oldal a csatlakozás után



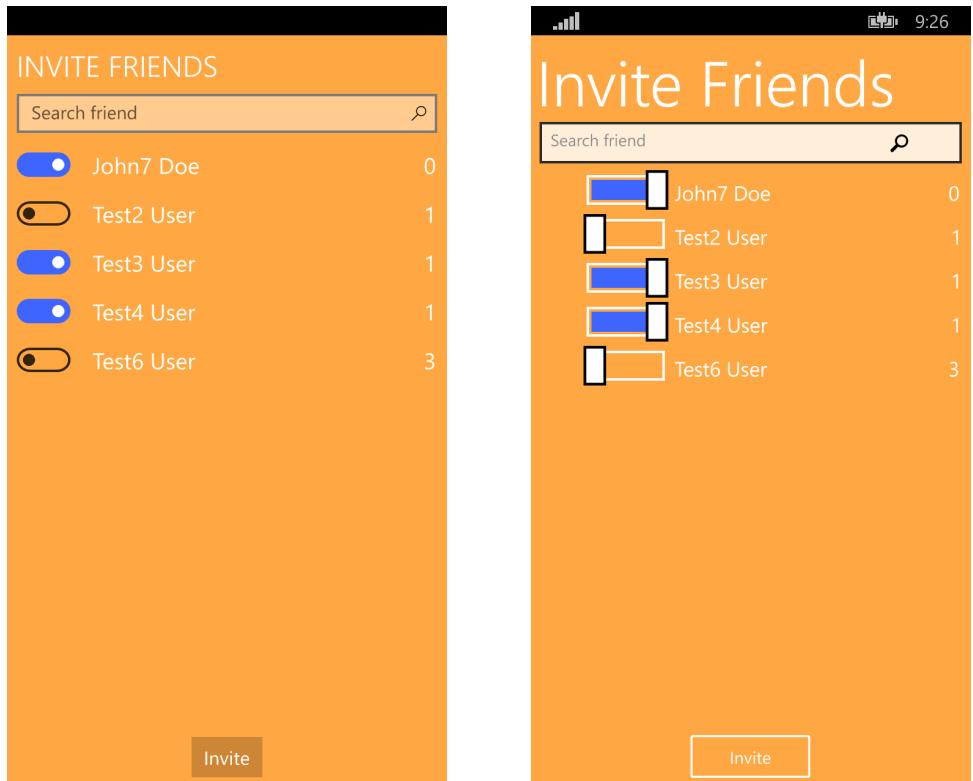
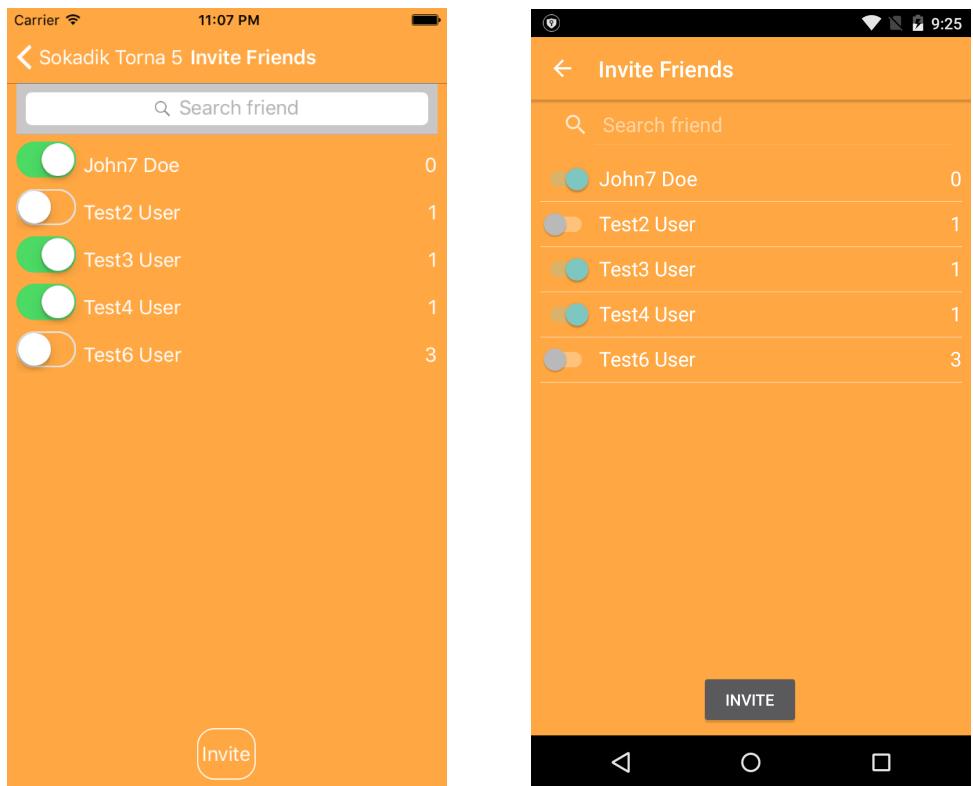
**F.6.20. ábra.** A barátok bajnokságra meghívására szolgáló oldal. Az *Invite All* gombbal az összes ismerős egyszerre meghívható.



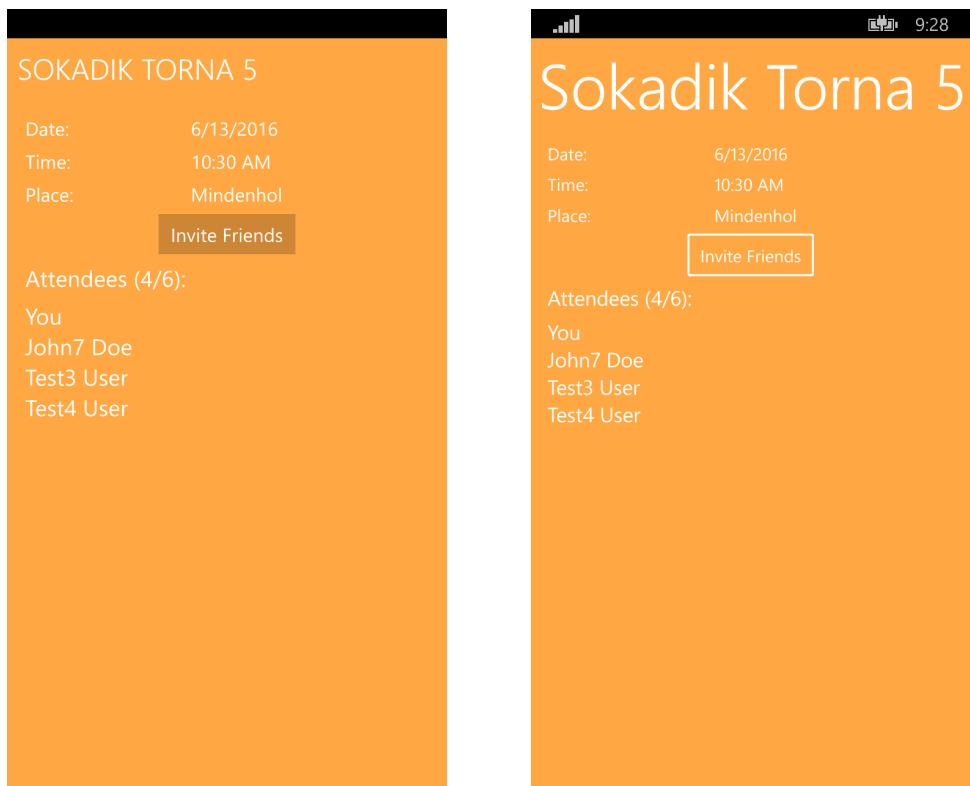
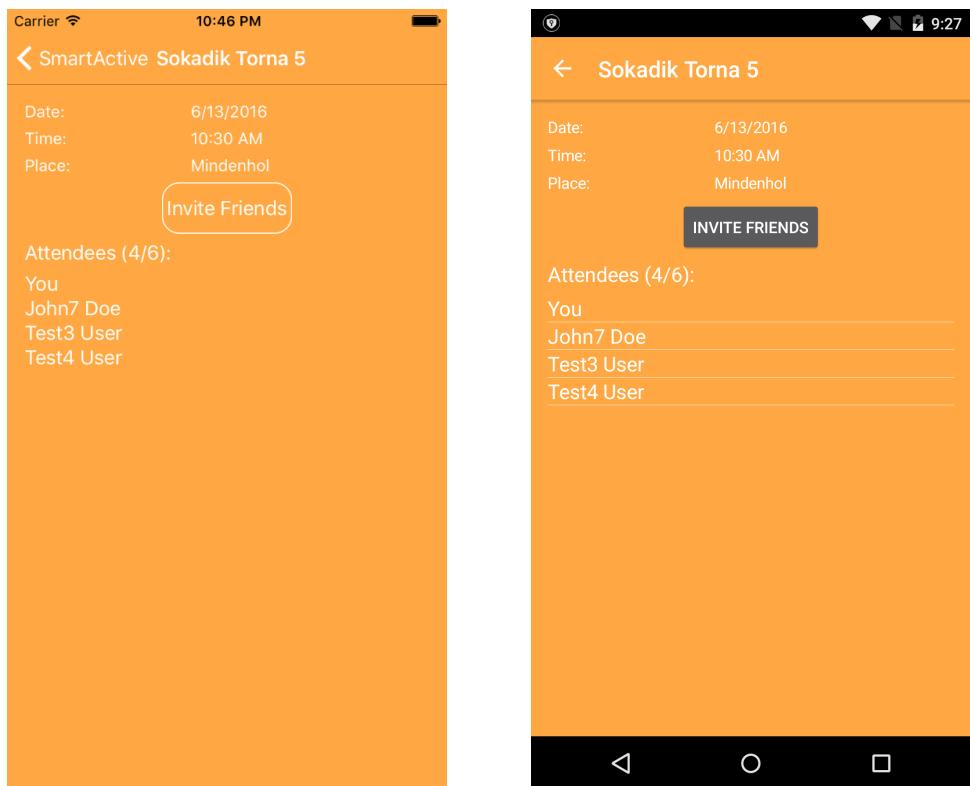
**F.6.21. ábra.** Ha a bajnokság maximális létszámát túllépné a meghívottak száma, hibaüzenet jelenik meg



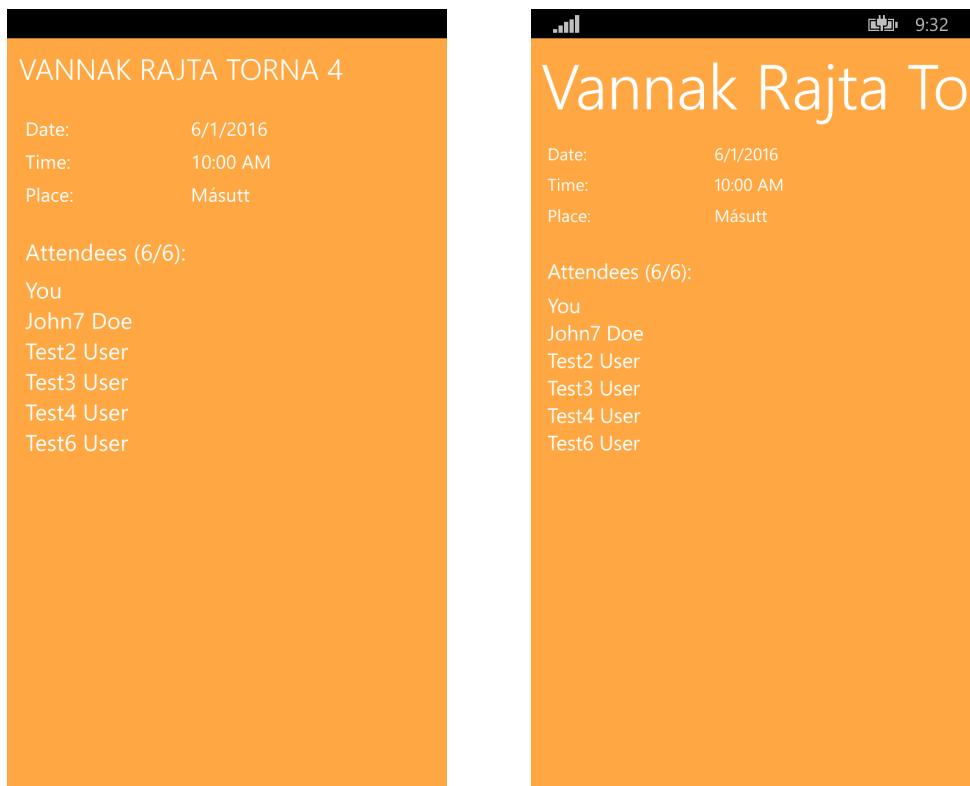
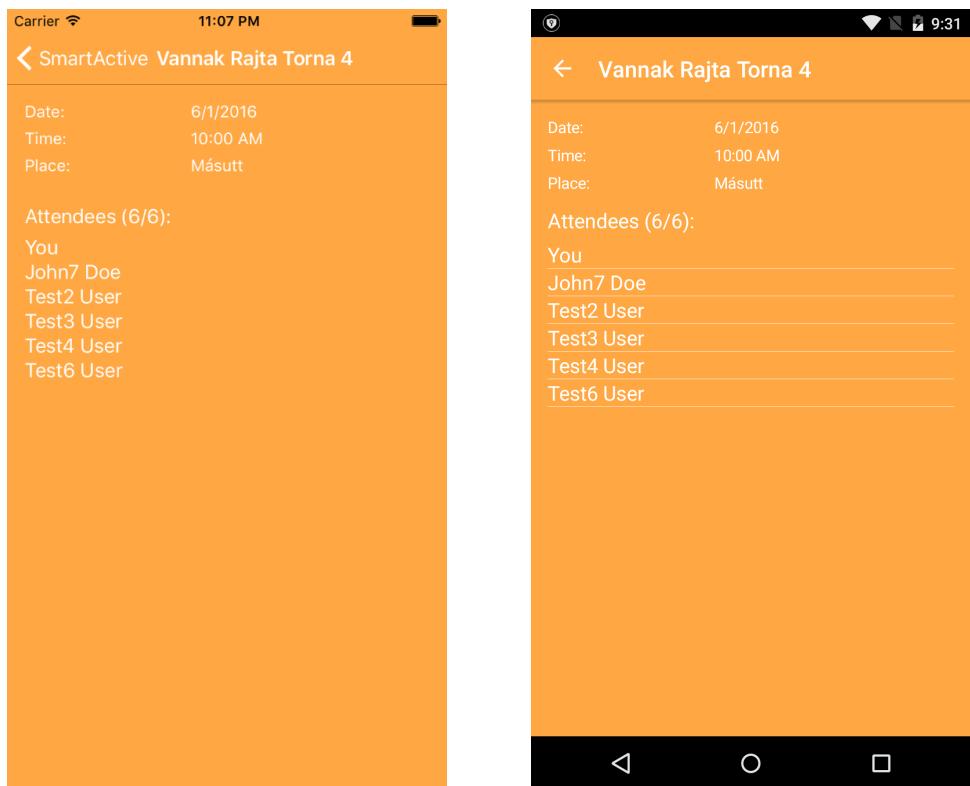
**F.6.22. ábra.** A keresés funkció használata ismerősök meghívásakor a bajnokságra



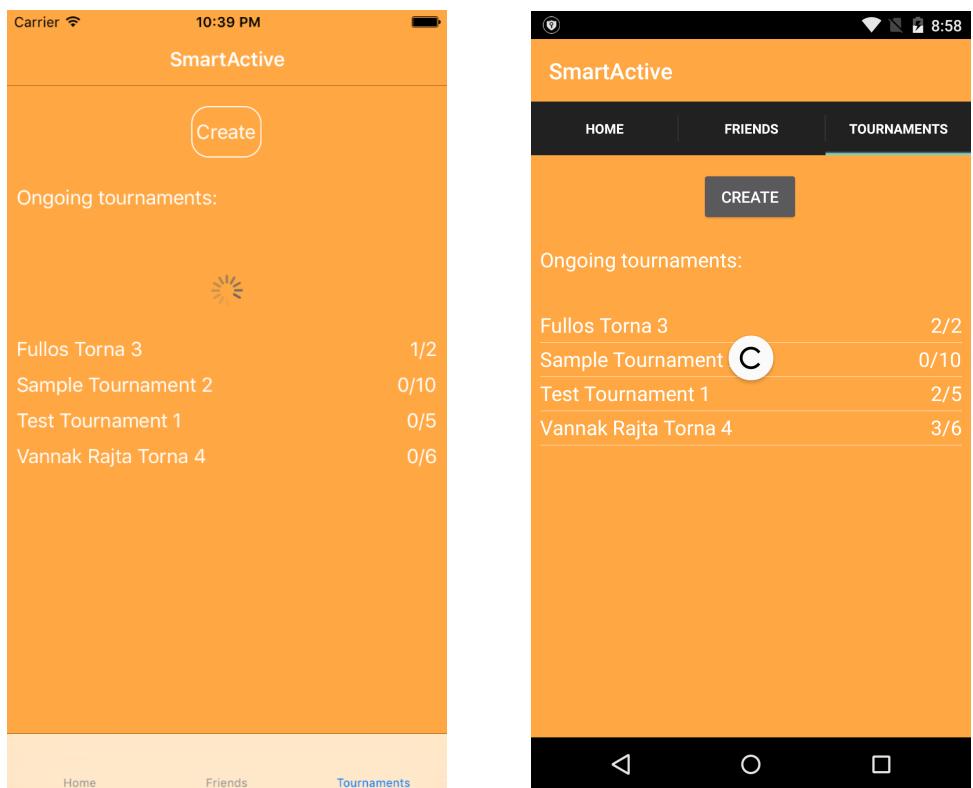
**F.6.23. ábra.** A kiválasztott barátok meghívása a bajnokságra az *Invite* gombbal tehető meg



**F.6.24. ábra.** A meghívott barátok megjelennek a bajnokság résztvevői között



**F.6.25. ábra.** Egy megtelt bajnokság részletezőoldala



**F.6.26. ábra.** A bajnokságok szinkronizálása a szerverrel iOS és Android operációs rendszereken