



Eötvös Loránd Tudományegyetem

Informatikai Kar

Információs Rendszerek Tanszék

**Keretrendszer páciensek aktivitási adatainak
gyűjtésére és feldolgozására**

Dr. Laki Sándor

Egyetemi adjunktus

Németh Péter

Nappali tagozat, Programtervező

Informatikus BSc

Budapest, 2018

Tartalomjegyzék

1. Bevezetés	3
1.1. Az activity trackerek szerepéről	3
1.2. Rövid ismertető a projektről	4
2. Felhasználói dokumentáció	7
2.1. Bevezetés	7
2.2. Node.js követelményei	7
2.3. Előfeltételek telepítése és elindítása	8
2.4. Az alkalmazás használata	9
2.4.1. Bejelentkezési módok és kijelentkezés.....	9
2.4.2. Statisztikák.....	12
2.4.3. Feljegyzések készítése	16
2.4.4. Adatok CSV formátumban való letöltése	18
2.4.5. Profiladatok	19
3. Fejlesztői dokumentáció	21
3.1. Model-view-controller architektúra.....	21
3.1.1. Az MVC alapelvei	21
3.1.2. Gyakorlati megvalósítása az alkalmazásban	22
3.2. Az OAuth 2.0 működése	23
3.3. A helyi adatbázis	24
3.3.1. Felépítése	24
3.3.2. Kapcsolata a controller réteggel	25
3.4. A konfigurációs fájl szerepe	27
3.5. Tesztelés	27
3.6. Az alkalmazás bővítése	29
3.6.1. Továbbfejlesztési lehetőségek	29

3.6.2. A bővítés menete	30
4. Összegzés.....	33
5. Irodalomjegyzék	34
6. Mellékletek	37

1. Bevezetés

1.1. Az activity trackerek szerepéről

Napjainkban egyre népszerűbbek az ún. **activity trackerek**, amelyeket itthon leginkább *fitness-karóra*, valamint *lépésszámláló* néven forgalmazznak. Ezek az parányi eszközök segítséget nyújtanak a hétköznapi embernek sporttevékenységeik és ezek hatásainak hatékonyabb naplózására, ezzel tervezhetővé teszik pl. egy ember diétáját és pl. ehhez kapcsolt napi futás mennyiségét.

Nem véletlen, hogy az orvostudomány is felfigyelt eme apró szerkezetek hasznosságára, és napjainkban komoly kutatásokat folytatnak eme viszonylag egyszerű szerkezetek egészségügyi bevetése iránt, a páciensek követésével ugyanis az eddigieknél pontosabb diagnózist lehet felállítani.

A karórák ára széles spektrumon mozog, a legolcsóbb, legegyszerűbb szerkezetek általában a hagyományos karórákról már jól ismert gombelemekkel működnek, kijelzőjük pár szisztematikusan villogó led-et jelent a gyakorlatban, ami idő kijelzésére nem vagy csak pontatlanul (Misfit Flash esetén 5 perces pontossággal) alkalmas. Az ár növekedésével fokozatosan jelennek meg további funkciók, hagyományos monokróm vagy led-háttérvilágítású színes kijelző, pontosabb, több téren végzett mérések, elem helyett újratölthető akkumulátorok használata, továbbá egyes többcélú felsőkategóriás okosórák is képesek különböző fitness-tevékenységeket mérésére.

A karórák az adatokat általában a gyártó felhő-szerverére küldik el, ahonnan valamilyen ingyenes, hivatalos számítógépes vagy mobiltelefonos alkalmazással

Gyártói között is igencsak nagy változatosság figyelhető meg, vannak kifejezetten ezen eszközök gyártására szakosodott cégek (Fitbit, Misfit), de az okostelefonok gyártói is előszeretettel foglalkoznak fitness trackerekkel, illetve hasonló mérésekre képes okosórákkal (Samsung, Xiaomi, Garmin, Huawei).



1. ábra: Misfit Flash Button

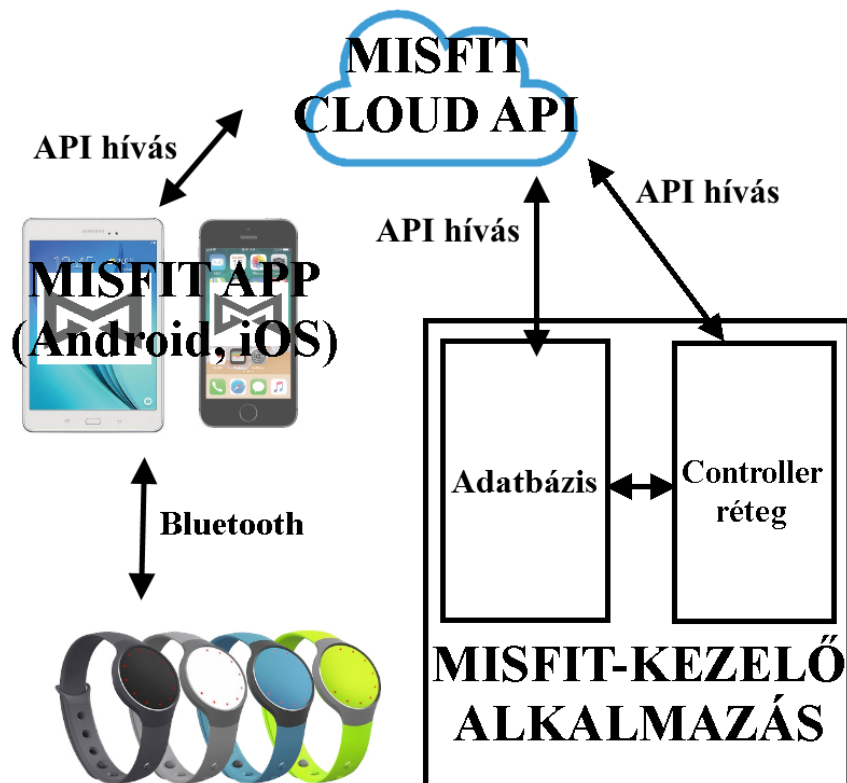
Szakdolgozatom elkészítéséhez 2 db Misfit márkájú Flash Button típusú activity trackert vettem igénybe. Ez egy egyszerű, alacsony árkategóriába tartozó karóra, amely óraelemmel működik, vízálló, 12 ledet tartalmaz ami az idő és a teljesített tevékenység kijelzésére szolgál.

A készüléket az Android [1] és iOS [2] operációs rendszereket használó mobiltelefonokat támogató Misfit nevű alkalmazás segítségével lehet szinkronizálni a telefon Bluetooth kapcsolatán keresztül. Az adatok a Misfit felhőjébe kerülnek, ahonnan felhasználóként a hivatalos Androidos alkalmazással lehet vizuálisan megtekinteni különböző diagramokon az adatokat. Fejlesztők számára azonban létezik egy ingyenes API [3] is, amellyel JSON formátumban le lehet kérdezni a tárolt adatokat. Az általam készített alkalmazás ez utóbbi felületet használja.

1.2. Rövid ismertető a projektről

Az alkalmazást Javascript nyelven fejlesztettem és a Node.js környezetben futtatható. Ez egy többféle platformon futtatható szofterkörnyezet webes alkalmazások fejlesztésére [4]. Előnye, hogy a webes hívások kezelésére praktikusán definiált eseménykezelőket használ, kihasználja a JavaScript aszinkron képességeit, valamint fájlkezelési funkciókkal egészíti ki. Előnye továbbá, hogy egy JSON formátumú fájlt is épít be az egyes projektekhez, ahol egyszerűen áttekinthető, lényegretörő formában megadhatók a telepítendő függőségek, azok pontos verziói, illetve a program fő adatai (szerző, program

verziója, license típusa). Ezen pozitívumok miatt számos nagy tech-cég támogatja fejlesztését és alkalmazza (IBM, Microsoft, Yahoo) [4].



2. ábra: A rendszer különböző elemeinek kapcsolati sémája

A Node.js nagyban támogatja a külső modulok fejlesztését, csomagkezelőjével az npm-mel (Node Package Manager [5]) kényelmesen, pár parancssori utasítással telepíthetők hasznos kiegészítők, melyekhez általában rövid, lényegretörő példákkal illusztrált webes útmutatók tartoznak. Ezekből két jelentős modult emelnék ki amelyek az objektum-orientált nyelveknél előszeretettel alkalmazott, ún. model-view-controller szerkezet kialakításában nyújtanak segítséget: az Express.js és a Handlebars.js. Ezek közül az Express.js az elérési út szerint szétválasztott controller-ek előállítását segíti (ez az ún. routing [6]), addig a Handlebars.js template-ekből állít elő weboldalakot, a controller-ek hívásai és paraméterei alapján [7]. A felmerülő hibák javítására szerver-, és kliensoldalon is alkalmazhatunk standard konzol kimenetet.

A template-ek is tartalmazzak ún. kliens-oldali JavaScript nyelven írt szkripteket, melyek főként a felület interaktivitását javítják, e tekintetben a JQuery [8] nevű külső modult veszem igénybe, amely a DOM elemeinek gyors, átlátható elérésére, valamint

hatékony manipulációjára alkalmas. A DOM (Document Object Model) a felhasználó által látott dokumentum objektumorientált nyelv- és platformfüggetlen fa-szerkezetű, szkriptekkel (JavaScript) tetszőlegesen alakítható, dinamikus modellje. Igyekszem továbbá a template-eket a HTML5 [9] szabványnak megfelelő formátumban kialakítani. Bizonyos oldalakon (pl. statisztika oldalon a feljegyzések) ún. AJAX [10] hívásokat végeznek a szerver felé, amely egy (elsősorban hálózatzbiztonsági megfontolások miatt) limitált tartományú, böngésző-átírányítást nem igénylő, népszerű aszinkron adatátviteli technika. A JavaScript kódok kialakításakor felhasználtam az új ECMAScript 6 szabvány újdonságaiból néhány hasznos nyelvi elemet:

- nyíl (arrow) operátor használata lambda kifejezéseknél
- halmaz típus (Set)

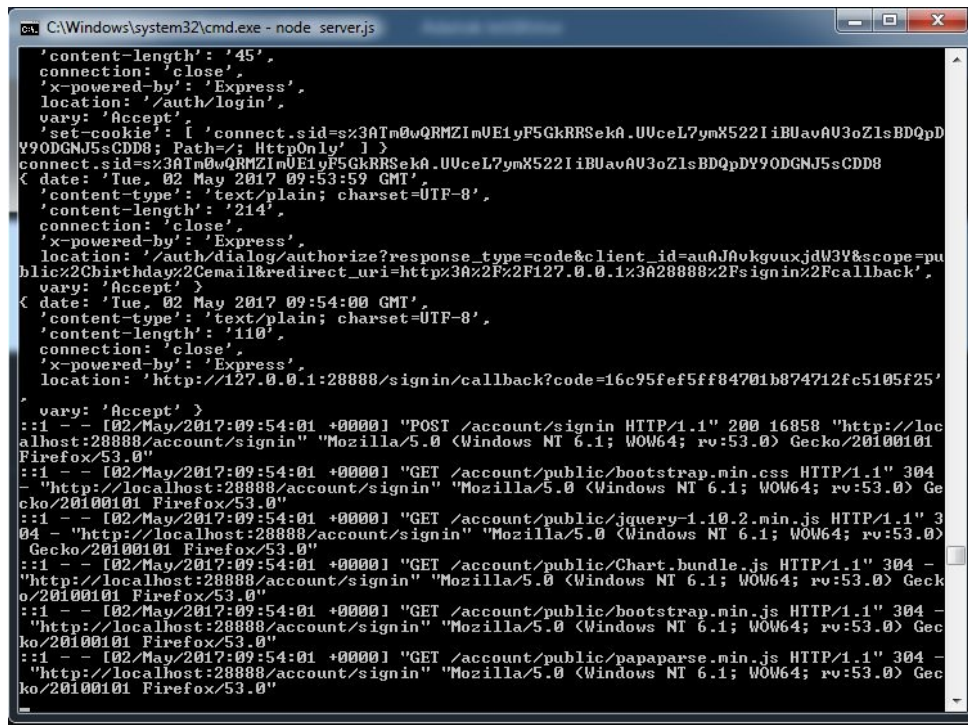
Az alkalmazás a népszerű Bootstrap témák egyikét, a Bootstrap Darkly-t [11] használom a webes megjelenítéshez, amely ingyenes, alkalmazkodik a különböző méretű kijelzőkhöz, ezáltal telefonon és tableten is könnyen kezelhető és olvasható marad. Az így dinamikusan létrejött weboldal az ún. business dashboard szemléletet követi, amely nagy mennyiségű adatból interaktív, valós időben kapcsolókkal állítható, finomítható interaktív diagramokat jelent a gyakorlatban. E szemlélet megvalósításához a Chart.js nevű modult használom. A korábban lekérdezett adatok egy része perzisztensen tárolódik a helyi szerveren egy adatbázisban, ezáltal bizonyos funkciók offline is elérhetőek.

A felhasználóknak, amennyiben az adatokat saját célra fel szeretnék hasznosítani, lehetőségük van azokat CSV [10], azaz vesszővel elválasztott szövegformátumban letölteni. A folyamat automatizálhatósága érdekében külön JSON formátumú lekérdezési paramétereket elfogadó mezőt is beépítettem az interaktív grafikus felület mellé.

2. Felhasználói dokumentáció

2.1. Bevezetés

Ebben a fejezetben az alkalmazás mindennapi használatát részletezem a telepítéstől az összes meglévő funkció részletezéséig, időnként betekintve a háttérben futó folyamatokba is.



```
C:\Windows\system32\cmd.exe - node server.js
'content-length': '45',
connection: 'close',
'x-powered-by': 'Express',
location: '/auth/login',
vary: 'Accept'
set-cookie: [ 'connect.sid=s%3A1m0wQRMZImUE1yF5GkRRSeKA.UUceL7ymX522IiBUavAU3oZ1sBDQpD
Y9ODGNJ5sCDD8; Path=/; HttpOnly' ]
connect.sid=s%3A1m0wQRMZImUE1yF5GkRRSeKA.UUceL7ymX522IiBUavAU3oZ1sBDQpDY9ODGNJ5sCDD8
date: 'Tue, 02 May 2017 09:53:59 GMT',
'content-type': 'text/plain; charset=UTF-8',
'content-length': '214',
connection: 'close',
'x-powered-by': 'Express',
location: '/auth/dialog/authorize?response_type=code&client_id=auAJAvkgvuxjdW3Y&scope=pu
blic%2Cbirthday%2Cemail&redirect_uri=http%3A%2F%2F127.0.0.1%3A28888%2Fsignin%2Fcallback',
vary: 'Accept'
date: 'Tue, 02 May 2017 09:54:00 GMT',
'content-type': 'text/plain; charset=UTF-8',
'content-length': '110',
connection: 'close',
'x-powered-by': 'Express',
location: 'http://127.0.0.1:28888/signin/callback?code=16c95fef5ff84701b874712fc5105f25'
vary: 'Accept'
::1 - - [02/May/2017:09:54:01 +0000] "POST /account/signin HTTP/1.1" 200 16858 "http://loc
alhost:28888/account/signin" "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:53.0) Gecko/20100101
Firefox/53.0"
::1 - - [02/May/2017:09:54:01 +0000] "GET /account/public/bootstrap.min.css HTTP/1.1" 304 -
"http://localhost:28888/account/signin" "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:53.0) Ge
cko/20100101 Firefox/53.0"
::1 - - [02/May/2017:09:54:01 +0000] "GET /account/public/jquery-1.10.2.min.js HTTP/1.1" 3
04 - "http://localhost:28888/account/signin" "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:53.0)
Gecko/20100101 Firefox/53.0"
::1 - - [02/May/2017:09:54:01 +0000] "GET /account/public/Chart.bundle.js HTTP/1.1" 304 -
"http://localhost:28888/account/signin" "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:53.0) Geck
o/20100101 Firefox/53.0"
::1 - - [02/May/2017:09:54:01 +0000] "GET /account/public/bootstrap.min.js HTTP/1.1" 304 -
"http://localhost:28888/account/signin" "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:53.0) Gec
ko/20100101 Firefox/53.0"
::1 - - [02/May/2017:09:54:01 +0000] "GET /account/public/papaparse.min.js HTTP/1.1" 304 -
"http://localhost:28888/account/signin" "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:53.0) Gec
ko/20100101 Firefox/53.0"
```

3. ábra: A Node.js szerveroldali kimenete futás közben

2.2. Node.js követelményei

A hivatalos weboldalon nem találtam asztali gépre való hivatalos követelményeket. A Node.js multiplatform, tehát többfajta operációs rendszeren és hardverarchitektúrán képes futni. Ezek a következők:

- ❖ Processzor-architektúrák:
 - Intel x86 (32 bites)
 - Intel x64 (64 bites)
 - ARMv6, ARMv7, ARMv8 (csak Linuxon)
 - SPARC (SunOS esetén)
- ❖ Operációs rendszerek:

- Linux disztribúciók (pl. Ubuntu, Fedora, Debian, stb.)
- Microsoft Windows 7, 8, 8.1, 10
- MacOS
- SunOS

A Node.js-nek szerveralkalmazás révén magas a memóriaigénye, a saját gépemen 4 GB RAM állt rendelkezésre, amellyel gond nélkül futott, a minimumkövetelmény kb. 1-2 GB RAM lehet. Processzor tekintetében érdemes minimum kettő vagy több magos modellt választani. A program futtatásához nem minden esetben, de szükséges élő internetkapcsolat. Tényleges többfelhasználós szerveren történő futtatás esetén érdemes a fentieknél erősebb hardvert alkalmazni.

A programok tesztelésére a node 7.7.1. számú 64-bites verzióját használtam

2.3. Előfeltételek telepítése és elindítása

A Node.js Windows rendszeren kényelmes, grafikus alkalmazással telepíthető, melyet a fejlesztő weboldaláról szerezhetünk be [4]. A telepítéssel egyidejűleg az npm csomagkezelő is felkerül a gépre. Linux-okon disztribúciófüggő a telepítés menete. Általában a rendszer elsődleges csomagtárában (ún. package repository) megtalálhatjuk a szoftvert. Ubuntu-nál például így néz ki [12]:

- ❖ `sudo apt-get update`
- ❖ `sudo apt-get install nodejs`
- ❖ `sudo apt-get install npm`

A node elindítása előtt szükségünk van még a csomag függőségeinek telepítésére. Ezt minden rendszeren a projektkönyvtárban elindított `"npm install <csomagnév> --save"` paranccsal tehetjük meg a `"--save"` paraméter a projektkönyvtárban levő **package.json** fájlba jegyzi be a telepített függőséget és annak verziószámát. Az alkalmazás függőségei a következők:

- express 4.15.0
 - body-parser 1.17.0 (volt *express.bodyParser*)
 - cookie-parser 1.4.3 (volt *express.cookieParser*)
 - errorhandler 1.5.0 (volt *express.errorHandler*)
 - morgan 1.8.1 (volt *express-logger*)
 - express.session 1.15.2 (volt *express.session*)

- hbs 4.0.1
- papaparse 4.1.4

A fenti öt alfüggőséget külön kiemelném, mivel ezek korábban az express beépített moduljai voltak, a 4-es verziótól viszont külön kell őket telepíteni [6].

A Node.js felületén futó programokat Microsoft Windows rendszereken a *"node <fájlnév>"*, Linux rendszereken (pl.: Debian) pedig a *"nodejs <fájlnév>"* paranccsal lehet futtatni, ahol a fájl nevéből a *.js* kiterjesztés elhagyható. Esetünkben (*Windows, server.js*) ez így néz ki:

❖ *node server*

Ezenkívül szükség van egy a JavaScript legújabb változatával kompatibilis böngészőre. A programot a Firefox 53.0 és a Google Chrome 57.0.2987.133 verziójával teszteltem.

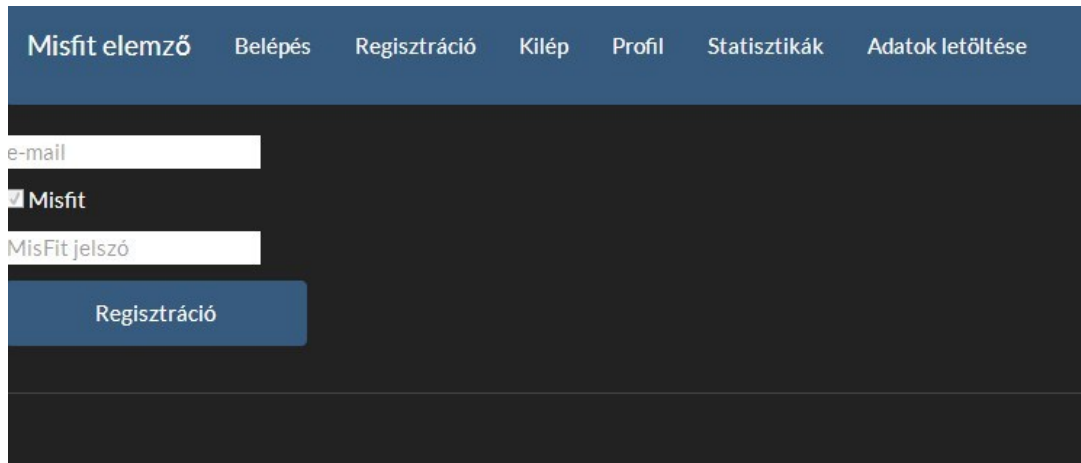
2.4. Az alkalmazás használata

2.4.1. Bejelentkezési módok és kijelentkezés

A felületet Node.js szerver elindítása után egy alkalmas böngészőben a *"localhost:28888"* címre navigálva érhetjük el. A címre lépve először a belépési oldal fogad minket. Itt két lehetőség közül választhatunk: a hagyományos belépési módot használjuk vagy JSON parancsot adunk meg.

4. ábra: Bejelentkezési oldal

A hagyományos bejelentkezési mód igénybevételekor fontos tudnivaló, hogy ehhez először regisztrálni kell. Erre az offline működés elősegítésének érdekében van szükség. Ezt az oldal alján található linkre kattintva tehetjük meg. A regisztrációhoz működő internetkapcsolatra van szükség, mivel ilyenkor a szerver megkísérli a felhőbe való belépést, majd siker esetén eltárolja adatainkat a helyi adatbázisban.



5. ábra: Regisztrációs oldal

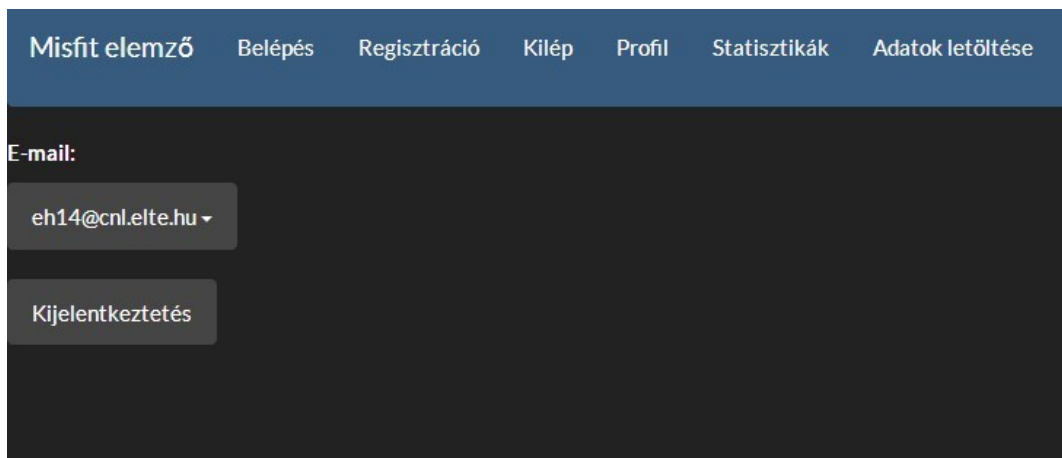
Ezután nincs más dolgunk, mint bejelentkezni. Ehhez csupán a felhasználónevet szükséges megadni. Sikeres bejelentkezés esetén a statisztikai oldal fogad bennünket, sikertelen esetben visszairányít minket a szerver a bejelentkezési oldalra piros színű *"A fiók nem létezik"* üzenettel.

Több felhasználóval is beléphetünk egymás után, ha sikeres bejelentkezés után ismét a "Belépés" gombra kattintunk a menüben. Kijelentkezni is felhasználónként tudunk egy listából választva.

Ha egy email címet először veszünk igénybe a Misfit Cloud API szerveren, rendszerint egy űrlapot kell kitöltenünk arról hogy engedélyezzük a hozzáférést az alkalmazásnak a profiladatainkhoz. A program beléptetési logikája az ilyen esetekre tesztelési lehetőségek híján nincs felkészülve, ugyanis az űrlapot fiókonként mindössze egyszer kell elfogadni, utána a szerver nem kér több megerősítést. Emiatt ha hibát és a válasz headerekben html dokumentumot tapasztalunk ilyenkor a konzolon, üssük be az alábbi linket a `<CLIENT_ID>` és `<CALLBACK_URL>` pontokat a konfigurációs fájlban megadott alkalmazásadatok URL-kódolt megfelelőivel helyettesítve:

https://api.misfitwearables.com/auth/dialog/authorize?response_type=token&client_id=<CLIENT_ID>&scope=public,birthday,email&redirect_uri=<CALLBACK_URL>

Ha bejelentkeztünk és engedélyeztük a hozzáférést egy hibás linket kapunk, a konfigurációs fájlban megadott hagyományos OAuth 2.0 bejelentkezési végpont ugyanis már nem elérhető az alkalmazásban.



6. ábra: Kijelentkezési oldal

Lehetőségünk van hagyományos bejelentkezés helyett ún. JSON formátumú parancs megadására is. A JSON (JavaScript Object Notation) egy széles körben elterjedt, nyílt szabvány adatátvitelhez. Emberi szemmel is könnyedén áttekinthető, a JavaScript-ből ered, tulajdonképpen a nyelv objektumainak és tömbjeinek sztringgé alakítását takarja, de számos más programozási nyelv is támogatja [10].

A JSON parancsot a bejelentkezési oldal alsó mezőjében adhatjuk meg, majd a Parancs végrehajtása gombra való kattintással aktiválhatjuk. A kért adatok a következők:

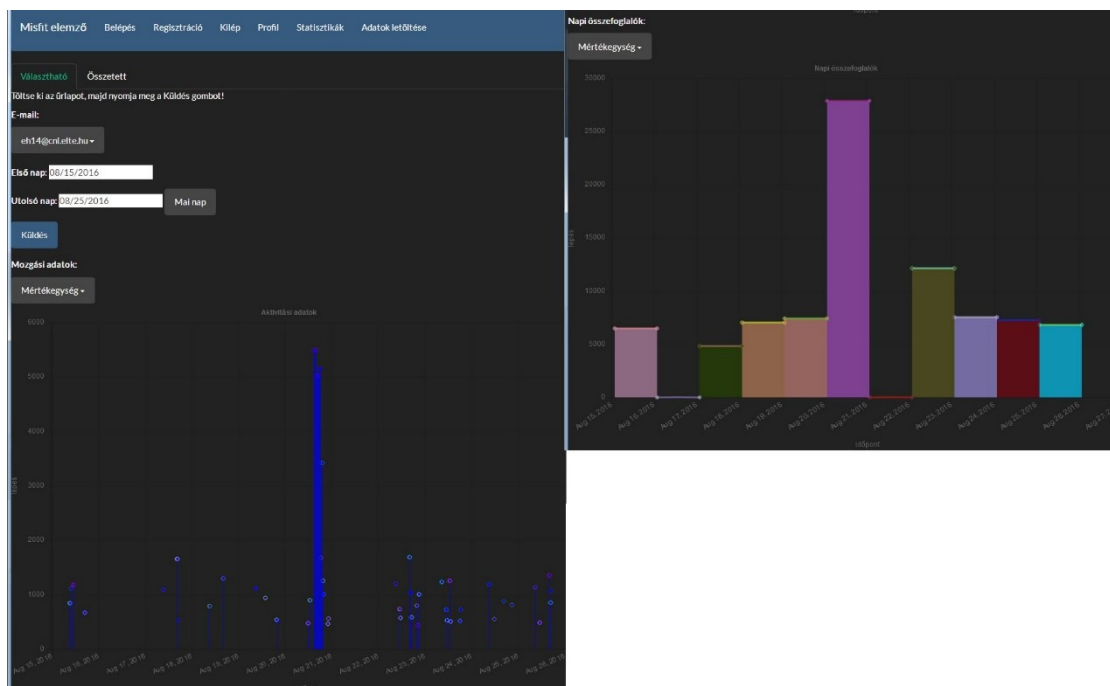
- **users:** felhasználó (email) és jelszó (password) kulcsú sztringeket tartalmazó objektumokból álló tömb
- **start_date, end_date:** ÉÉÉÉ-HH-NN sztring formátumú első és utolsó napja a lekérdezett időintervallumnak

Amennyiben nem megfelelő formátumban adtuk meg a parancsot, a szerver visszaküld minket "Hibás JSON parancs" üzenettel. Ha rossz felhasználónév-jelszó páros van a listában, akkor ezt külön jelzi. A JSON parancs használatához nem szükséges regisztrálni, ilyenkor az alkalmazás (amennyiben a felhasználó nincs a helyi adatbázisban) a Misfit szerveren kíséri meg a belépést. Sikeres belépés esetén egy CSV fájlt kapunk az összes felhasználó időintervallumbeli adataival. A JSON parancs tehát egyenértékű azzal, mint amikor hagyományosan bejelentkeztünk több felhasználóval, majd az "Adatok letöltése" oldalon az "Összetett" opciót választjuk.

The screenshot shows the Misfit app interface with a dark blue header containing navigation links: Misfit elemző, Belépés, Regisztráció, Kilép, Profil, Statisztikák, and Adatok letöltése. Below the header, a red error message reads "Hibás JSON parancs." (Invalid JSON command). Underneath, there is a form with an "e-mail" input field and a "Belépés" (Login) button. Below the form, a text input field contains the text "json parancs". At the bottom, there are two green buttons: "Parancs végrehajtása" (Execute command) and "Regisztráció" (Registration).

7. ábra: Hibaiüzenet hibás formátumú JSON bemenet esetén

2.4.2. Statisztikák



8. ábra: Statisztikai oldal grafikonokkal

A statisztika oldalon három fajta statisztikát tekinthetünk meg. Az egyik a "sessions", amely a Misfit karórák egy sajátossága. A felhasználó az órán levő gomb háromszori megnyomásával indíthat el egy "session"-t, ami méri az utána következő tevékenységeket és hat fajta szerint csoportosítja őket:

- biciklizés (cycling)
- úszás (swimming)
- sétálás (walking)
- tenisz (tennis)
- kosárlabda (basketball)
- soccer (foci)

Az úszást ezek közül külön megemlíteném, mivel elvileg az óra ismertetőjében vízállónak tüntetik fel, az interneten számos oldalon cáfolták ezt a tényt [13], kiemelve hogy az ún. ATM-minősítés 3-as szintjét éri el, ami úszáshoz nem megfelelőt jelent, kizárólag zuhanyzáshoz használható biztonságosan [14]. Emiatt ezt a funkciót nem teszteltem.

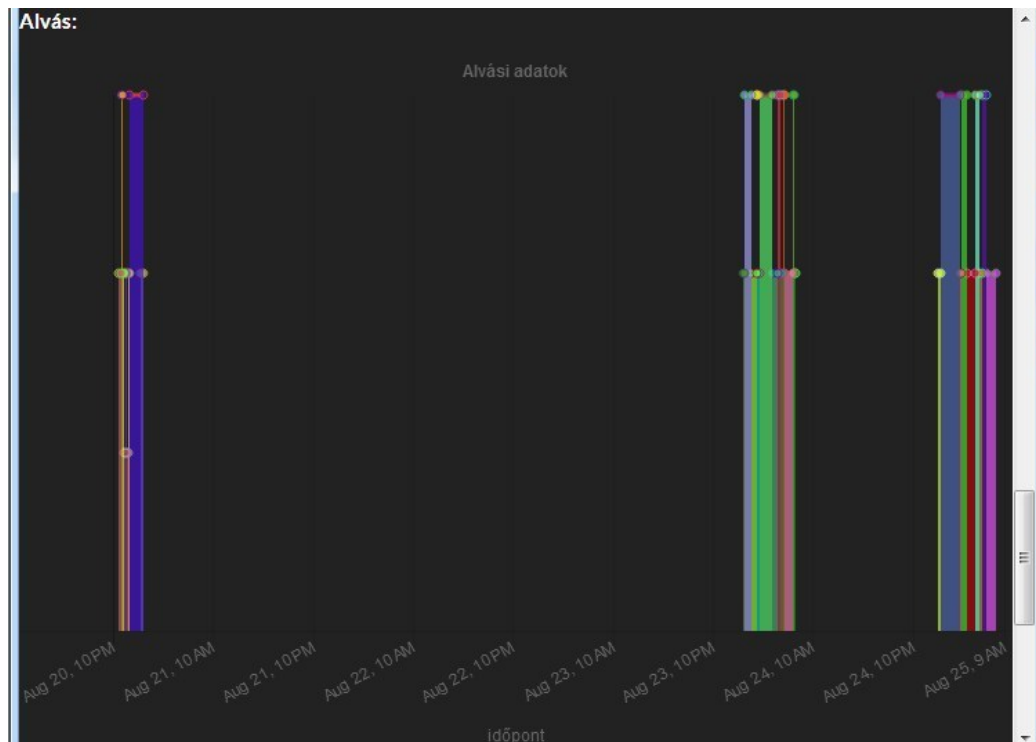
A másik statisztikai grafikon a *"summary"* mutatja az adott napon elvégzett tevékenységek összegét.

Végül, a harmadik grafikon a felhasználó alvási adatait ábrázolja interaktívan, megkülönböztetve az alábbi alvási fokozatokat:

- mély alvás időintervalluma (deep sleep, 3-as érték)
- átlagos alvás időintervalluma (average sleep, 2-es érték)
- felébredés pillanata (awake, 1-es érték)

Az alvási adatoknál kisebb pontatlanságok észlelhetők, ennek oka hogy a kapott adatszerkezetben [3] rejlik, ugyanis a kezdőidőpontból (startTime) és a globális alváshosszból (duration) kiszámolt végpont (a programban totalEnd néven található) olykor megelőzi az utolsó alvásfokozat-változást jelző pontokat (sleepDetails több timestamp-jei).

A grafikonon megjelenő időintervallumot és fiókokat megfelelő paraméterek megadása után a "Küldés" gombbal állíthatjuk át. Ilyenkor néha kicsit várnunk kell, mert amennyiben az adatok nincsenek meg a helyi adatbázisban, a szerver kérést küld a Misfit felhőnek, és siker esetén lementi az adatokat a helyi adatbázisba és megjeleníti a kliens gépén.

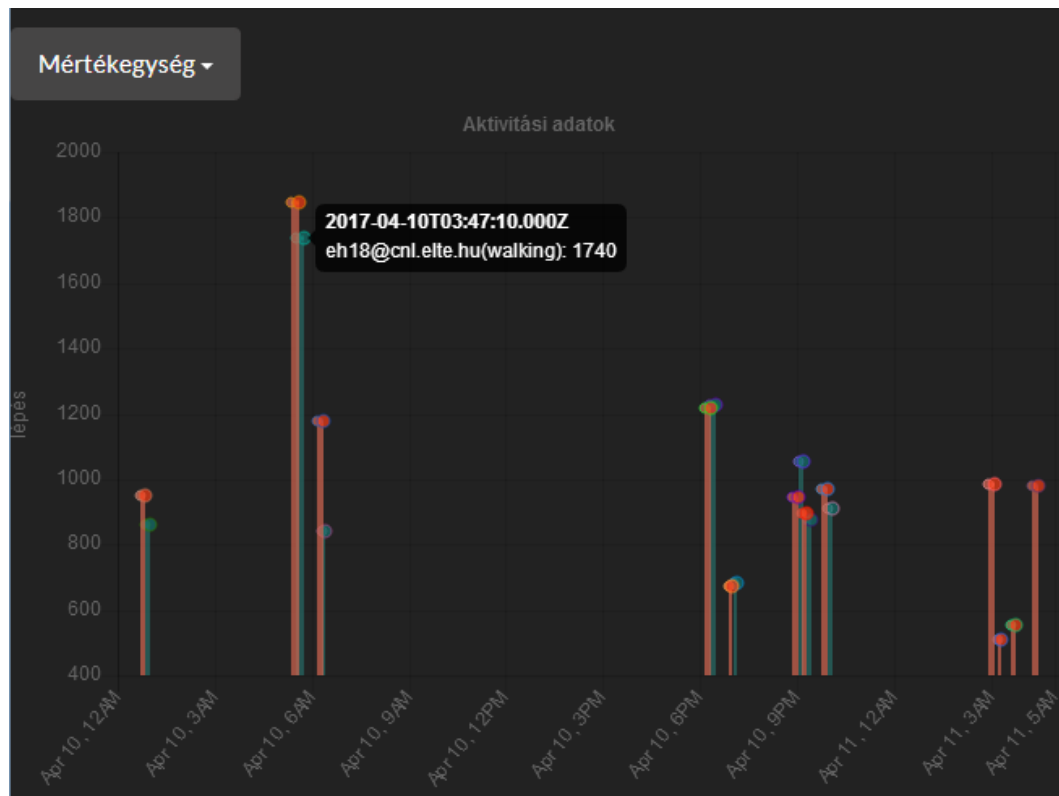


9. ábra: Alvási adatok

A grafikonoknál választhatunk mértékegységet:

- ❖ tevékenységekre számított pontok
- ❖ lépések száma
- ❖ megtett távolság kilométerben (mérőföldből átszámítva)
- ❖ felhasznált kalóriák
- ❖ kifejezetten tevékenységekhez felhasznált kalóriák (csak a "summary"-nél)

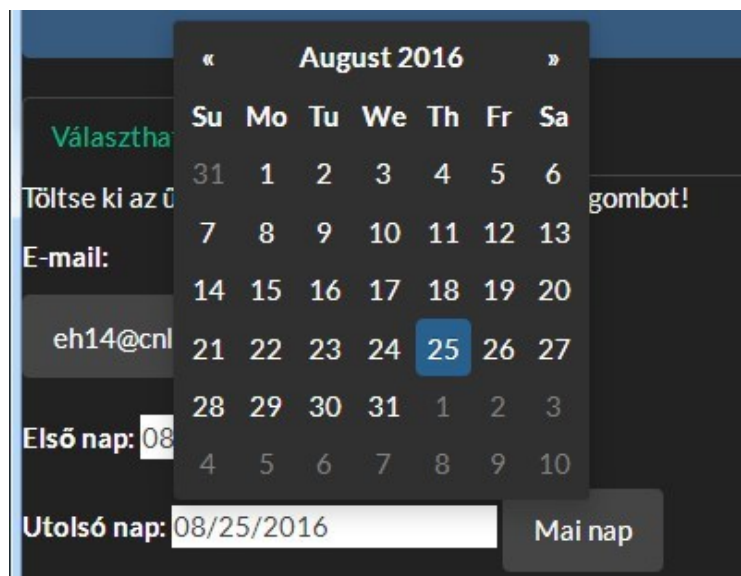
Lehetőségünk van kétféle mód közül választani az oldal tetején található fülön. "Választható" módban egy email címet tudunk kiválasztani megjelenítésre, az adatok a grafikonon véletlenszerű színárnyalatban jelennek meg. "Összetett" módban automatikusan az összes bejelentkezett email címre lekérdezi az adatokat és azokat felhasználónként különböző színnel jeleníti meg, amennyiben egynél több e-mail címmel vagyunk bejelentkezve.



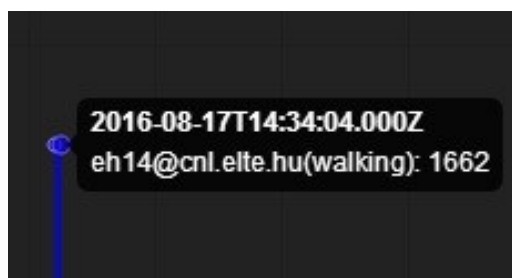
10. ábra: Összetett statisztikai oldal sessions grafikonja
(A két óra ezen a napon egyszerre volt a kezemen)

A lekérdezés első és utolsó napját a Bootstrap Datepicker nevű modul grafikus kalendáriumából [15] választhatjuk ki, az e-mail címet pedig egy JQuery szkript által vezérelt saját fejlesztésű Bootstrap felületi elemeket alkalmazó leugró menüből. A kényelem növelése érdekében beleraktam a felületbe egy "Mai nap" gombot is, amely az utolsó napot az aktuális napra állítja be.

A statisztikai modul az egyes adatok buborékaira helyezve az egérkurzort, információkat közöl a tevékenység típusáról, a felhasználóról, a kezdő- és végidőpontokról és az aktuális mértékegység pontos értékéről, az első két statisztikánál egy leugró menüben mértékegységet is válthatunk.



11. ábra: Bootstrap Datepicker dátumkiválasztásnál



12. ábra: Információk a grafikon elemekre történő rámutatásnál

2.4.3. Feljegyzések készítése

A statisztikai oldalon készíthetünk feljegyzéseket is. Ehhez navigáljunk az oldal aljára, majd a *Feljegyzések* címsor alatt kattintsunk az *Új* gombra. Ekkor egy dialógus ablak jelenik meg, amely a Bootstrap megjelenítési modul Modal nevű elemének lehetőségeit használja ki. Az ablak egy űrlapot tartalmaz, ahol ki kell választanunk a vonatkozó e-mail címet, meg kell adnunk a dátumot az előbb látott grafikus kalendárium segítségével. Az alapértelmezett dátum a statisztikai oldal aktuális lekérdezésének első napja. Ezután a feljegyzés tartalmát kell egy szövegdobozba beírni. A feljegyzés több sorból is állhat. Ezt követően nincs más tennivalónk, mint a *Mentés* gombra kattintani. Amennyiben az ellenőrző szkript hibát talál az űrlapon, az érintett sorokat vörösre színezve visszaküldi javításra. A műveletet vissza is vonhatjuk a *Bezárás* gomb választásával.

Feljegyzés létrehozása

E-mail:
eh14@cnl.elte.hu ▼

Dátum:
08/15/2016

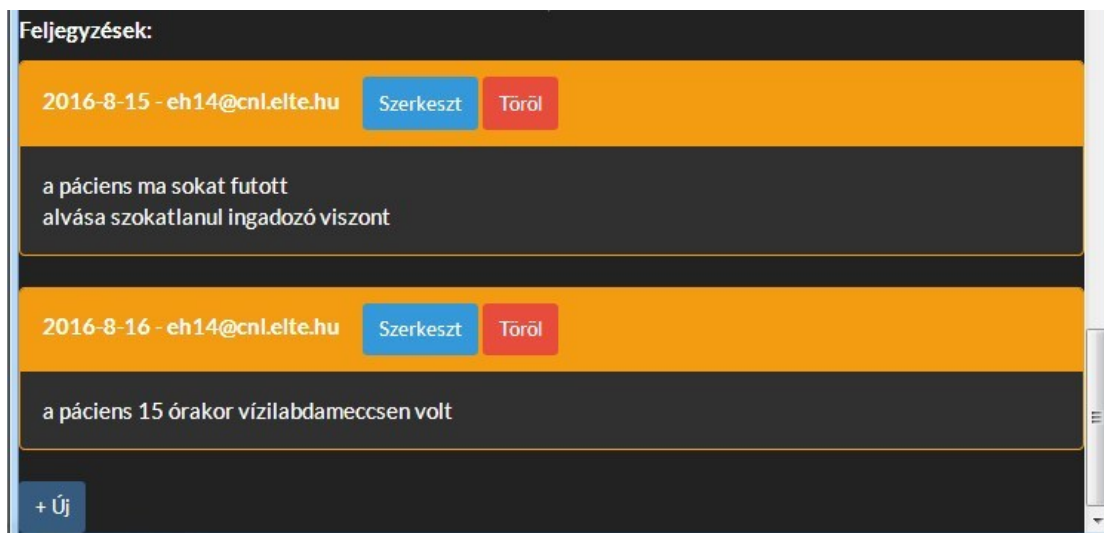
Feljegyzés tartalma:
rthrthrth
rtdrthrth

Mentés Bezárás

13. ábra: Új feljegyzés felvételére szolgáló dialógus ablak

Amennyiben az adatbázisba való rögzítést végző JQuery AJAX hívás sikerrel tér vissza, a feljegyzés azonnal megjelenik a statisztikai oldalon. Hiba esetén üzenet jelenik meg a szerver konzolján.

A feljegyzések dátum szerint növekvő sorrendben jelennek meg. Mindegyik egy sárga színű keretbe van foglalva amely tartalmazza az email-címet, dátumot, továbbá egy *Szerkeszt* és egy *Töröl* gombot. Szerkesztéskor a már említett felugró ablak jelenik, viszont ezúttal csak a feljegyzés tartalmát módosíthatjuk.



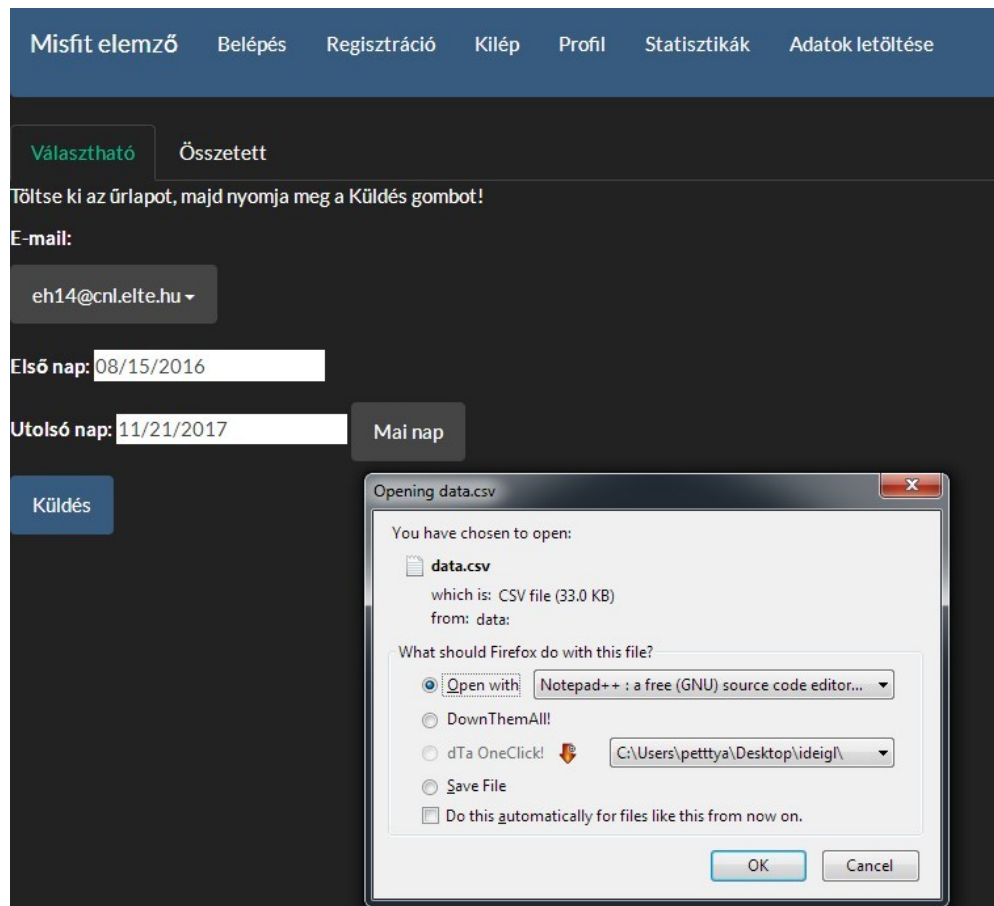
14. ábra Feljegyzések a statisztikai oldalon

2.4.4. Adatok CSV formátumban való letöltése

A letöltési oldal menüjének funkcióiban megegyezik a statisztikai oldalon látottakkal, viszont itt grafikon megjelenítése helyett a "Küldés" gombra kattintva egy CSV fájlt kapunk kézhez, melybe az adott időintervallum tevékenységei (misfit-sessions), napi összefoglalói (misfit-summary), alvási adatai (misfit-sleep) továbbá a kapcsolódó feljegyzések (comments) kerülnek, további adatfeldolgozás céljára.

A CSV (Comma Separated Values, magyarul: vesszővel elválasztott értékek) egy népszerű adatcserére használt fájlformátum. Általában adatbázisoknál alkalmazzák, eredete még a Fortran idejébe nyúlik vissza a hetvenes évekbe. A JSON-höz hasonlóan könnyedén olvasható, mivel az egyes adatmezők vesszővel, a rekordok pedig újsorral vannak elválasztva egymástól [10].

A kapott adatok feldolgozásánál figyelembe kell venni azt hogy az alkalmazásom a különböző oszlopokkal rendelkező adattáblákat kever egy fájlba, kihasználva a JavaScript objektumokból álló tömb-definíciójának [16] rugalmasságát, a táblákat a kapott CSV fájlban egy *table* nevű oszlop különbözteti meg. Emiatt megjelennek olyan oszlopok is ahol egy adott tábla esetében csupa üres (*null*) értékkel találkozhatunk. Ez azért van, mert az érintett táblának nincs ilyen oszlopa (pl. *misfit-sessions* táblánál az *activityCalories* mező hiányzik), viszont a jelenlegi táblákban meglévő oszlopok nem vehetnek fel *null* értéket. Az adatok feldolgozása esetén a CSV fájl beolvasásánál erre a problémára ügyelni kell.



15. ábra: Letöltési oldal a "Küldés" gombra kattintás után

2.4.5. Profiladatok

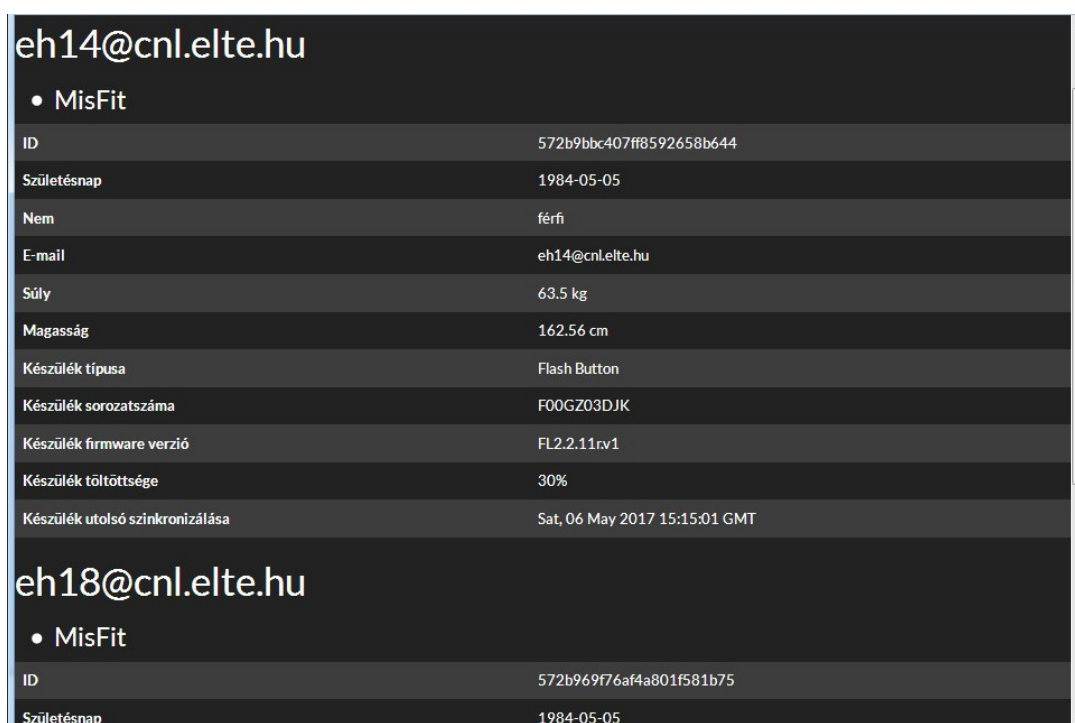
Létezik a fentiekén kívül egy profiladatokat tartalmazó oldal is, ahol alapvető információkat tudhatunk meg a fiók tulajdonosáról és a készülékről:

- ❖ Tulajdonos
 - e-mail címe:
 - neve
 - születésnapja
 - testmagassága centiméterben (hüvelykből átszámítva)
 - testsúlya kilogrammban (angolszász fontból átszámítva)
 - fiók sorszáma
- ❖ Készülék
 - típusa
 - sorozatszám
 - firmware verziója
 - töltöttsége

➤ utolsó szinkronizálás időpontja

Amennyiben több felhasználó is bejelentkezett, akkor a különböző fiókok adatait az e-mail címet fejlécként használva elválasztja egymástól. Az oldal megjelenítéséhez bizonyos adatok változékonysága (pl. utolsó szinkronizálás dátuma) miatt internetkapcsolat szükséges.

Egyes alapvető profiladatokat a Misfit internetes felhasználói felületén állíthatunk át más értékekre [3].



eh14@cnl.elte.hu	
• MisFit	
ID	572b9bbc407ff8592658b644
Születésnap	1984-05-05
Nem	férfi
E-mail	eh14@cnl.elte.hu
Súly	63.5 kg
Magasság	162.56 cm
Készülék típusa	Flash Button
Készülék sorozatszáma	F00GZ03DJK
Készülék firmware verzió	FL2.2.11rv1
Készülék töltöttsége	30%
Készülék utolsó szinkronizálása	Sat, 06 May 2017 15:15:01 GMT
eh18@cnl.elte.hu	
• MisFit	
ID	572b969f76af4a801f581b75
Születésnap	1984-05-05

16. ábra: Profiladatokat tartalmazó oldal

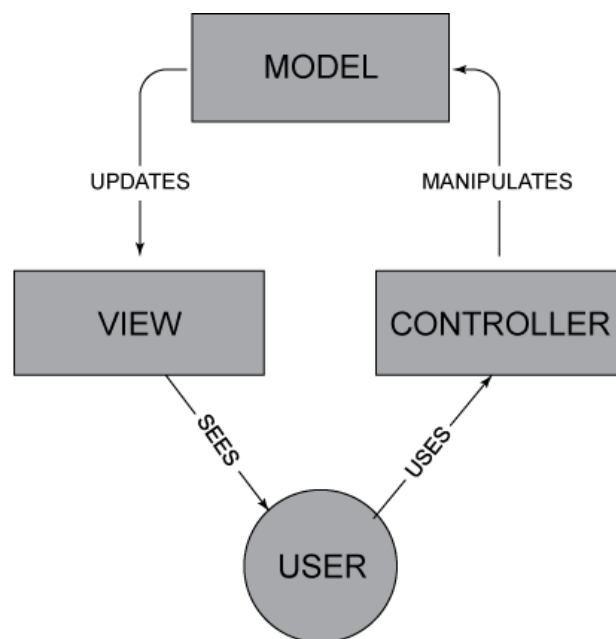
Mivel a profiladatok változékonyságuk miatt nem tárolódnak a helyi adatbázisban, az oldal megjelenítéséhez mindig szükségünk van működő internetkapcsolatra.

3. Fejlesztői dokumentáció

3.1. Model-view-controller architektúra

3.1.1. Az MVC alapelvei

A program fejlesztését egyetlen darab *server.js* nevű fájlban kezdtem meg. Hamarosan a megírt kód kezdett túlnőni egy fájl keretein, ekkor úgy döntöttem valamilyen séma alapján szétválasztom a program komponenseit. Rövid kutatás után az ún. *model-view-controller* sémát találtam a feladatra a legalkalmasabbnak.



17. ábra: A model-view-controller séma koncepciója [17]

A *model-view-controller* az egyik legismertebb tervezési sémája az interaktív, felhasználói felülettel rendelkező alkalmazásoknak. A forráskódot három szerkezeti részre választja, azon elv szerint, hogy mit lát a felhasználó, mit vezérel, és mi az ami e kettő mögött található. Támogatja a párhuzamos programozást, céges kultúrában is népszerű, mert nagyfokú munkamegosztást tesz lehetővé, továbbá elősegíti a rendezett kód komponenseinek újrafelhasználását más projekteknel (pl. megjelenítési rész). Mivel azonban a koncepció nincs pontosan definiálva, különböző implementációi léteznek a gyakorlatban és ez függ az adott programozási nyelv felépítésétől és szabályaitól. Mindezek okán szeretném először tisztázni az alapvető definícióját a három rétegnek, majd részletezni hogy ehhez képest az én megvalósításom hogyan épül fel.

A *modell réteg* az alkalmazás adatainak kezelését végzi helyi és/vagy távoli erőforrások segítségével. Ez az erőforrás általában egy adatbázis szokott lenni, vagy egy szerver REST végpontja [10]. Megvalósítja a perzisztencia elvét [10] amely az adatok tartós tárolását jelenti. Utasításokat, lehívásokat kap a kontroller felől az adatok részleges vagy teljes előhívására, majd azokat a megjelenítési rétegnek továbbítja közvetlenül vagy a kontrolleren keresztül.

A *kontroller réteg* bemeneti utasításokat kap a felhasználótól (pl.: egy billentyű lenyomása vagy egy gombra való kattintás grafikus felületen), majd lekéri a modelltől az adatokat, vagy éppen felkéri a modellt azok frissítésére a tartós adattáron. Mindezeket a felhasználó számára nem feltűnő, a megjelenítés folyamatát nem lassító módon, aszinkron [10] utasításokkal végzi, ezzel megvalósítva a párhuzamosság kritériumát. A legtöbb esetben a visszakapott adatokat is a kontroller kezeli, majd küldi el a megjelenítési rétegnek.

A *megjelenítési réteg* végzi a felhasználói felület kialakítását, az egyes felületi elemek kirajzolását, regisztrálja az egyes felületi elemekhez kötött eseményvezérlőket. A kontrollertől vagy a modelltől visszakapott adatok és utasítások alapján frissíti a felületet.

3.1.2. Gyakorlati megvalósítása az alkalmazásban

A Node.js esetében ennek a koncepciónak a webes átültetését alkalmaztam. A modell itt a *modules* könyvtárban található *dataHandler.js*, amely tartalmaz egy helyi adatbázisfájlt kezelő osztályt, és olyan rutinokat amelyek ebből az adatbázisból, illetve hiány esetén a Misfit Cloud API-ról kérdeznak le statisztikai adatokat, távoli szerverről való lekérdezés esetén a helyi adatbázist is frissítve.

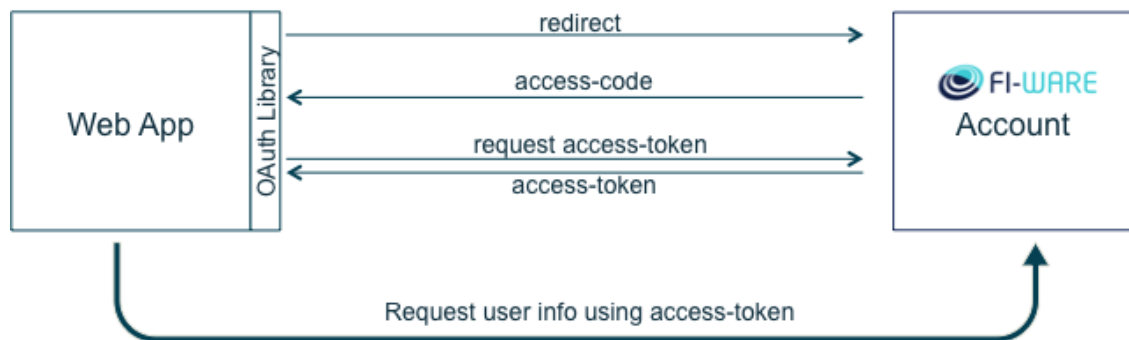
Kontrollerként a *controllers* könyvtár komponenseit használom. Itt a Node.js *express* [6] nevű csomagja segít. Ez ugyanis beépített útválasztási, ún. "*routing*" [6] megoldásokat kínál, ezáltal hatékonyabban tudom szétválasztani az egyes végpontok vezérlését külön fájlokba, ami nagyban javítja az áttekinthetőséget.

A megjelenítési réteg implementálásához a Node.js Handlebars.js [7] nevű modulját alkalmaztam, amely paraméterekkel ellátott sablonok, adatokkal való feltöltésével állít elő dinamikusan létrehozott weboldalakat, ezenkívül számos kiegészítő funkcióval is ellátja a fejlesztőt (iterálás tömböknél, HTML escape-elés, feltételes elágazás, saját

helperek írása). A template-eket a *views* könyvtárba helyeztem el, a kapcsolódó végpontoknak megfelelő elnevezésekkel.

3.2. Az OAuth 2.0 működése

Az OAuth egy nyílt szabványú protokoll, amely hozzáférést biztosít szervereknek külső, harmadik fél által fejlesztett alkalmazások számára felhasználói fiókokhoz való hozzáféréshez a felhasználó belépési adatainak (felhasználónév, jelszó) a harmadik fél számára történő kiadása nélkül. Legfrissebb verziója az OAuth 2.0, amely még könnyebbé teszi a protokoll gyakorlati implementálását kliensoldalon, és lehetőséget ad különböző eszköztípusok (pl. mobiltelefonok, szórakoztatóelektronikai eszközök) megkülönböztetett kezelésére. Az OAuth 2.0 szabványt előszeretettel használják nagy internetes cégek is (pl.: Google, Facebook, Microsoft, LinkedIn stb.).



18. ábra: Az OAuth 2.0 működése [15]

A külső szerverre való csatlakozás kliensoldalon, amely alkalmazásomban a Node.js szervert jelenti, a következőképpen történik: először a kliens kérést küld a szerver hitelesítési url-jére. Ezután a szervertől kap egy *access code*-ot (egyes helyeken *authorization code*-nak is hívják). Ez egy viszonylag rövid életidejű kód, ami *access token* és opcionálisan *refresh token* igénylésére használható.

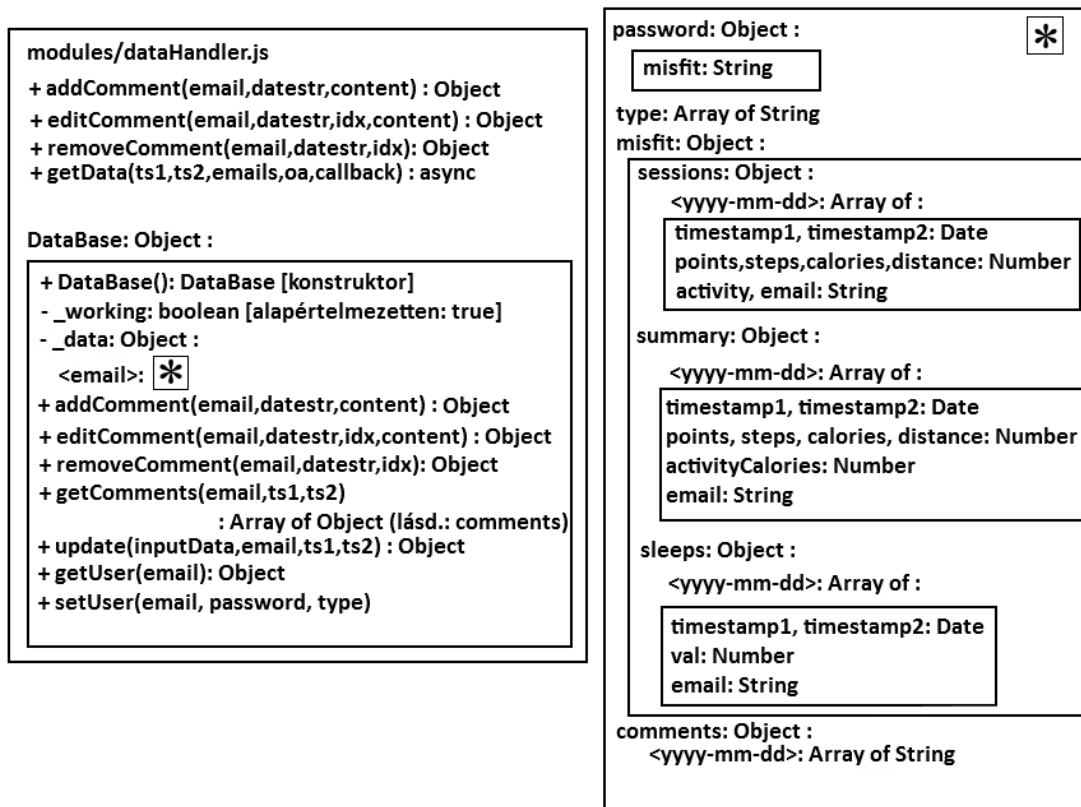
Ezzel szemben az access token hosszabb ideig igénybevehető, ez a gyakorlatban percek, órákat jelenthet nagyságrendileg. Az érvényességi idő lejártával új token-t kérnünk, egy *refresh token* segítségével lehetséges, amennyiben ezt a szerver támogatja [18].

Az bejelentkezés folyamatának meggyorsítása, és a helyi adatbázis adatainak integritása szempontjából témavezetőm segítségével az alkalmazáshoz egy saját OAuth 2.0 bejelentkezési modult fejlesztettem ki, egy a GitHub-on elérhető projekt alapján [15].

A felhasználó először köteles magát a helyi adatbázisban regisztrálni. A szerver a Misfit Cloud API-hoz csatlakozva az interaktív felületet megkerülve a Misfit szervertől kapott sütiket elmentve, szimulál POST hívásokat a bejelentkezési linkekre, ezáltal kiváltva egy *authorization code*-ot, ezt követően pedig egy *access token*-t.

3.3. A helyi adatbázis

3.3.1. Felépítése



19. ábra: A `dataHandler.js` szerkezeti felépítése

A modules könyvtárban található `dataHandler.js` nevű fájl egy helyi in-memory jellegű adatbáziskezelőt tartalmaz. Az adatbázis létrejöttkor beolvasásra kerül a `dbCloud.txt` nevű fájlból, amennyiben az létezik, a memóriában rövid ideig tárolódik, majd a módosítások kiírásra kerülnek a már említett fájlba, ezután az objektum élettartama véget ér.

Amennyiben a módosítási műveletek során, az objektum `_working` nevű *private* jellegű változója `false` értéket kapva zárolja a műveletet, és megakadályozza a kiírást, ezzel csökkentve az adatvesztés esélyét.

Az adatok lényegi része a *_data* nevű változóban tárolódik. Gyakorlatilag ez kerül JSON formában kiírásra. Típusa ún. Object, amely a JavaScript objektum-orientált típusainak létrehozását segíti elő ún. *prototype*-ként [16], de konstruktor nélkül a más nyelvekben (pl.: Java, Python, C#) általában *dictionary* [19] [20], illetve *map* [21] neveket viselő halmaz-elvű kulcs-érték párokból álló adattípus megvalósítására kiválóan alkalmas. Továbbá egy Object kulcs alá tetszőleges típusú értéket adhatunk, akár tömböt vagy egy újabb Object-et is, ezzel bonyolult többszintű adatszerkezetet létrehozva. A változón belül az adatok email címek szerint vannak *kulcs-érték* párként rögzítve.

Egy email címhez tartozó szintén *kulcs-érték* szerkezetű változón belül *password* kulcs tartalmazza az egy darab email címhez tartozó különböző márkájú activity trackerek webes profiljához tartozó jelszavakat, a *type* kulcs alatt az email címhez tartozó márkákat tömb formátumban. Jelen esetben a *password* kulcs alatt kizárólag *misfit* alkulcs található, illetve a *type* értéke mindig *['misfit']*, ahogy az a mellékelt ábrán is látható.

A felhőből származó adatok a márka nevével ellátott kulcs alá kerülnek. Ez a Misfit esetében a *sessions*, *summary*, illetve a *sleeps* alkulcsokat takarja. Ezeken belül az adatok az egyes napok szerint elnevezett kulcsok alá rendezett tömbökbe vannak rendezve.

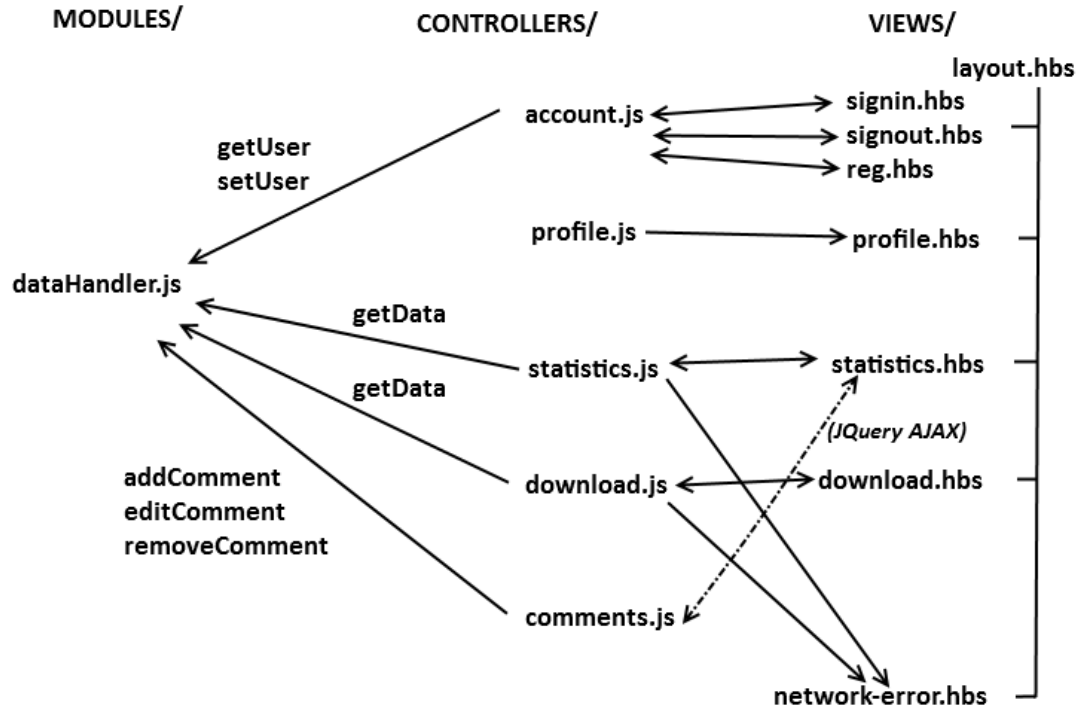
Ugyancsak az e-mail cím szintje alá kerülnek a feljegyzések is a *comments* kulcs tartalmazza őket, amelynek értéke egy dátum kulcsok alá rendezett sztring tömb értékekből álló adatszerkezet.

3.3.2. Kapcsolata a controller réteggel

A *dataHandler.js* modullal a controller réteg több komponense is interakcióba lép.

Az *account.js* amely a felhasználók regisztrációjáért és beléptetéséért felelős tartományt kezeli, a felhasználók jelszavait és fióktípusát (*password* és *type* kulcsok) kérdezi le és regisztráció esetén módosítja.

A *statistics.js* és a *download.js* a tevékenységi adatokat igényli a *getData* aszinkron függvényen keresztül, amennyiben a kért intervallum nem található meg a helyi adatbázisban, meghívásra kerül a *getDataFromMisfitCloud* segédfüggvénye. Az adatokat a *template* réteg kliens oldali szkriptjei alakítják át színekkel ellátott, grafikonon ábrázolható, illetve CSV fájlba letölthető formátumba.



20. ábra: A helyi adatbázis, a controller és a view réteg kapcsolatai

A *comments.js* a feljegyzések létrehozását, módosítását, törlését kéri az adatbázistól az *addComment*, *editComment*, *removeComment* függvények hívásával, ez a kontroller viszont nem generál HTML oldalakat, a végpontokon csupán a statisztikai oldal kliensoldali szkriptjei érintkeznek vele *jQuery AJAX* típusú POST hívásokkal, sikeres művelet esetén egy *true* értékű *success* boolean-t küld vissza, így a kliensoldalon azonnal megjelenhetnek a változtatások. A név ellenére a feljegyzések lehívása sem itt történik, a statisztikai és a letöltési oldal a *getData* függvényen keresztül kapja meg az adott időintervallumra és e-mail címekre vonatkozó feljegyzéseket. A *getData* függvény, amennyiben az adatbázisban a kért időintervallum nem, vagy csak részlegesen található meg, a hiányzó dátumokat e-mail szerint csoportosítva kigyűjti és az összefüggő intervallumokat csoportosítja. A Misfit Cloud API maximum 31 napos intervallumokat enged, ezért a program egy ennek megfelelő szétbontást is elvégez. Ezek után a dátumpárokra végigfutva, egy rekurzívan önmagát hívó aszinkron kérdezi le egymás után az időintervallumokat, a legvégén a kapott eredményt bejegyzik az adatbázisba és elküldi a kontrollernek egy paraméterként megadott, ún. *callback* függvény segítségével [22].

A controller rétegen kívül a `login_misfit.js` és az `oauth2_misfit.js` is érintkezik az adatbázissal a `getUser()` függvényen keresztül, az autentikáció folyamatának végrehajtása céljából.

3.4. A konfigurációs fájl szerepe

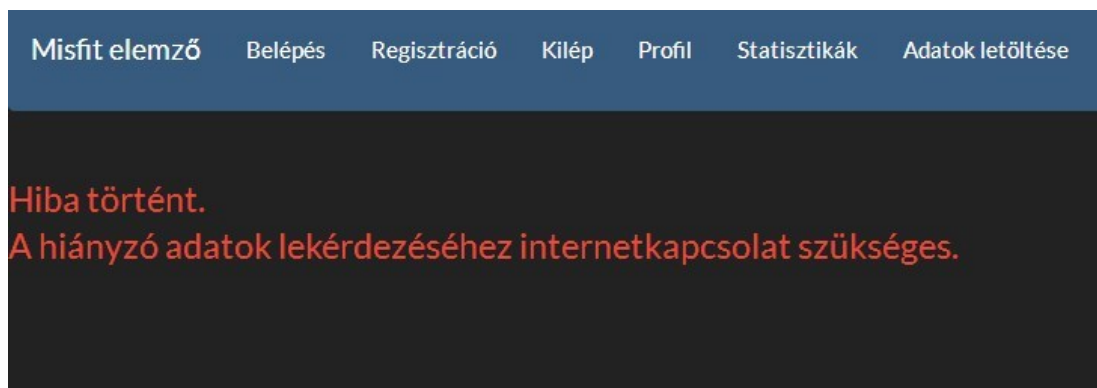
A projekt gyökérkönyvtárában található egy `cfg.js` nevű fájl, amely a felhőhöz kapcsolódó OAuth 2.0 autentikáció és az abból való adatlekérés céljára szolgáló URL-eket tartalmazza statikus sztring és esetenként paraméterekkel dinamikusan változtatható lambda-függvény formájában. Ezek az URL-ek a márka nevét viselő kulcs alatt találhatóak (pl.: `cfg.misfit.summaryURL`)

3.5. Tesztelés

A projekt tesztelését ebben a fejezetben mutatom be. A teszteseteket úgy válogattam össze, hogy azok a program lehető legtöbb használati esetét lefedjék, ugyanakkor a ténylegesen mindent lefedő hibamentességhez részletesebb tesztelések szükségesek. Az egyes eseteket ténylegesen ki is próbáltam éles környezetben a program legújabb verzióján. Az alábbiakban ismertetem az egyes teszteseteket és azok végeredményét, esetenként ábrával is illusztrálva.

- **Teszteset:** Regisztráció érvényes adatokkal
 - Zöld színű *Sikerés regisztráció* értesítés jelenik meg. A `dbCloud.txt` fájlban megjelennek a felhasználó adatai.
- **Teszteset:** Regisztráció még egyszer ugyanezekkel az adatokkal
 - Zöld színű *Nem történt változtatás* értesítés jelenik meg.
- **Teszteset:** Regisztráció érvénytelen adatokkal
 - Vörös színű *Rossz felhasználónév vagy jelszó* értesítés jelenik meg.
- **Teszteset:** Belépés regisztrált fiókkal
 - A `/statistics/selectable` oldalra irányít át a szerver
- **Teszteset:** Belépés nem regisztrált fiókkal
 - Vörös színű *A fiók nem létezik* értesítés jelenik meg
- **Teszteset:** A gyökérkönyvtárban elhelyezkedő `jsonproba.txt`-ben található JSON parancs végrehajtása
 - Egy `data.csv` nevű fájl kapunk.
- **Teszteset:** Hibásan megformázott JSON parancs végrehajtása (pl.: `'{'`)

- Vörös színű *Hibás JSON parancs* értesítést kapunk.
- **Teszt eset:** Hiányos JSON parancs végrehajtása (pl.: '{}')
 - Vörös színű *Hibás JSON parancs* értesítést kapunk.
- **Teszt eset:** Üres JSON parancs végrehajtása
 - Vörös színű *Hibás JSON parancs* értesítést kapunk.
- **Teszt eset:** Helytelen belépési adatok a JSON parancsban
 - Vörös színű *Rossz felhasználónév vagy jelszó* értesítés jelenik meg.
- **Teszt eset:** Statisztikai/letöltési űrlap elküldése érvényes adatokkal *Választható* módban
 - Megfelelő statisztikát kapunk.
- **Teszt eset:** Statisztikai/letöltési űrlap elküldése érvényes adatokkal *Összetett* módban
 - Megfelelő adatokat kapunk.
- **Teszt eset:** Statisztikai/letöltési űrlap elküldése üres *Első nap* vagy *Utolsó nap* mezővel
 - Elküldés helyett a dátum mező címkéje vörös színre vált.
- **Teszt eset:** Statisztikai/letöltési űrlap elküldése a megadott *Első napot* megelőző *Utolsó nappal*
 - Hibás eredmény, üres statisztikát vagy CSV fájlt kapunk
- **Teszt eset:** Statisztikai űrlap elküldése megszakadt internetkapcsolat esetén
 - Amennyiben az összes kért adat nem található a helyi adatbázisban, vörös színű *A kért adatok megjelenítéséhez internetkapcsolat szükséges* szövegű hibaüzenetet kapunk.



21. ábra: Hibaüzenet internetkapcsolat hiánya esetén

- **Teszt eset:** Feljegyzés felvétele érvényes adatokkal.

- Sikeres mentés, az új feljegyzés rögtön megjelenik az oldalon.
- **Teszt eset:** Feljegyzés felvétele érvénytelen dátummal
 - Sikertelen mentés, a *Dátum* mező címkéje vörösre vált.
- **Teszt eset:** Feljegyzés felvétele üres tartalommal
 - Sikertelen mentés, a *Feljegyzés tartalma* mező címkéje vörösre vált.
- **Teszt eset:** Feljegyzés módosítása érvényes tartalommal
 - Sikeres mentés, a módosított feljegyzés tartalma frissül.
- **Teszt eset:** Feljegyzés felvétele üres tartalommal
 - Sikertelen mentés, a *Feljegyzés tartalma* mező címkéje vörösre vált.
- **Teszt eset:** Feljegyzés törlése
 - Sikeres törlés esetén a feljegyzés azonnal eltűnik.
- **Teszt eset:** Profil oldal megjelenítése működő internetkapcsolat esetén
 - Az összes bejelentkezett felhasználó adatai megjelennek csoportosítva.
- **Teszt eset:** Profil oldal megjelenítése nem működő internetkapcsolat esetén
 - Vörös színű *A kért adatok megjelenítéséhez internetkapcsolat szükséges* szövegű hibaüzenetet kapunk.

3.6. Az alkalmazás bővítése

3.6.1. Továbbfejlesztési lehetőségek

A dolgozat készítése során számos lehetőség felmerült a projekt bővítésére, a Misfit-en kívül más, alternatív márkák támogatásának implementálásával történő bővítésre, illetve nagyobb léptékekben gondolkodva a program struktúrájának bizonyos mértékű áttervezésére valamilyen más koncepció alapján. Ezeket az opciókat szeretném itt röviden összegezni.

A Misfit termékeinél lényegesen több funkciót kínálnak a Fitbit [23] márkájú activity trackerek, melyek alapvetően egy jóval drágább árkatagóriát képviselnek a piacon. A Fitbit Web API-n Misfit Cloud API-nál is meglevő mérési típusok (tevékenységek, napi összefoglalók, alvási adatok) itt pontosabban, kiterjedtebben részletezve megtalálhatóak, ezenfelül a következő funkciókat is nyújtja [23]:

- ❖ kedvenc, gyakori ételek nyilvántartása, tápanyagok mennyiségével (*food logging*)

- ❖ vérnyomás mérése (*heart rate*)
- ❖ az activity tracker élettartama alatt elért rekordok, összesítések (*activity goals*)
- ❖ súly, túlsúly nyilvántartása (*weight log*)
- ❖ profiladatok szerkesztése Web API-n keresztül

A dolgozat elkészítése közben foglalkoztam adatbázis-orientált webalkalmazások fejlesztésével. Itt merült fel a lehetőség, hogy az alkalmazásomat nagyobb léptékű továbbfejlesztés esetén érdemes lenne átalakítani *front end* - *back end* [10] megközelítés szerint. A *front end*, amely a kliensoldalon futó részeit takarja az alkalmazásnak, ez esetben TypeScript nyelven Angular keretrendszerben készült sablonokból állna. A *back end*, a szervertoldali adatkiszolgáló rész egy Node.js szerver lenne egy komolyabb adatbáziskezelővel, erre a célra a MySQL vagy a MongoDB tökéletesen megfelelne.

3.6.2. A bővítés menete

A projektet a más gyártók webes API-jával való kiegészítés céljából könnyen átalakítható szerkezetűre terveztem. A gyakorlatban kísérletet is tettem a fentebb részletezett Fitbit Web API beépítésére, de a hátralévő kevés idő miatt végül felhagytam ezzel a próbálkozással. Az alábbiakban pár pontban részletezem egy akár eltérő, sajátos protokollal rendelkező OAuth 2.0 autentikációt alkalmazó webszerverhez kapcsolódó rutin kialakítását, és az egyes kontrollerek átírását.

Először a gyökérkönyvtárban található `cfg.js` nevű konfigurációs fájlban készíteni kell egy új kulcsot az adott márka nevével, bejegyezni ez alá kulcs-érték párokként az autentikációhoz és az adatlekéréshez szükséges url-eket. Amennyiben dinamikusan felparaméterezhető URL-ekre van szükség (pl.: dátum, user id mint paraméter), a ES6 nyíl függvényeit érdemes használni, ezek ugyanis könnyen áttekinthetővé teszik a kódot.

Ezt követően az autentikációs folyamat kidolgozása következik. Az OAuth 2.0 szabványt használó szerverek által megkövetelt kötelező átirányítás helyett célravezetőbb egy böngésző munkamenetet szimulálni, a kapott süti lementésével (*Set-Cookie* [22] paraméter a *HTTP Response Header*-ben [22]). Válaszként kapott HTML dokumentum esetén, szerkezeti áttanulmányozást követően, az abban található űrlapot megfelelően ki kell tölteni, a felhasználónévről és a jelszóról sem megfeledkezve, majd egy DOM-ot szimulálva (amihez Node.js-nél a *jsdom* [5] nevű modul kiválóan alkalmas) a HTML form taggel ellátott űrlapját kell elküldeni az attribútumokban meghatározott eljárással (GET vagy POST) és címre. A sikeres módszer kifejlesztéséhez hozzájárulhat az űrlap

első kitöltésének tényleges böngészőben történő kipróbálása, a szerver irányába történő hívásokat és válaszokat a böngésző konzolján megfigyelve. Az elküldéshez a Node.JS-re is telepíthető *jQuery* [8] modul *AJAX* [10] hívását ajánlott használni. Érdemes arra is figyelni, hogy időnként a szerver engedélykérő űrlap kitöltését és küldését is igényelheti. Ez főleg első használatkor fordulhat elő.

Az adatbáziskezelőben (*modules/dataHandler.js*) a felhasználók adatait lekérdező és módosító függvényeket (*getUser* és *setUser* függvények) szükséges átalakítani a *type* tömbhöz adjuk hozzá az új márkanevet is, amennyiben kipipálták a bejövő adatoknál az adott márkát és be kell jegyezni a jelszót a *password* kulcs márkanevvel ellátott alkulcsa alá, ahogy az adatbázis felépítését taglaló fejezetnél látható. A *getData* függvénybe építjük bele a márka saját lekérdezési eljárását. Érdemes a felhőből való lekérdezésre szolgáló programrészt egy belső aszinkron függvénnyel különválasztani. Amennyiben több egymást követő, csak paraméterekben különböző *aszinkron* hívást kell elvégezni, érdemes rekurzív függvényszerkezetet kialakítani, úgy hogy a függvény az egyes iterációk során önmagát hívja meg, a végén pedig bejegyzí az összegyűjtött eredményül kapott adatokat a helyi adatbázisba, majd az induló paraméterként megkapott külső *callback* függvény meghívásával továbbítja azokat a *getData* függvénynek. Ezután a *getData* szintén *callback* függvénnyel továbbítja az adatokat a kontrollernek. Amennyiben ténylegesen kaptunk adatot egy adott márkáról, a márka nevét viselő *true* értékű boolean kulcsot a válaszként küldött Object-be kell beépíteni. Ennek a template szempontjából van szerepe, ugyanis csak akkor tölti be az adott márkához tartozó grafikonokat is a kapott HTML kódba, ha teljesül ez a feltétel. Ezenkívül ki kell bővíteni a *getData* bemeneti paramétereit az adott márka webes API-hoz használt OAuth moduljával (pl.: *'oa_fitbit'*).

Ezután a *controller* és a *view* réteg átalakítása van hátra. A letöltési és a statisztikai oldal az adatokat a *getData* függvényről szerzi, a kontroller fájlban a már említett plusz OAuth paraméteren kívül, egyéb tennivaló nincs. A template-ben viszont ki kell alakítani a kliens oldali szkripteket a márkához tartozó adatok grafikonon való megjelenítésére, illetve CSV fájlba konvertálására. Grafikon esetén létre kell hozni a sablonban az új adattípusok megjelenítésére szolgáló vázont, ki kell alakítani a mértékegységváltás szkriptjeit, és az erre szolgáló legördülő listát.

Utolsó lépésként a profiloldal átalakítása van hátra. Itt figyelembe kell venni, hogy ez a programrész az adatbázissal nem érintkezik, a profiladatok lekérdezéséhez internetkapcsolat szükséges. A kontrollerben az új OAuth hívásokat a régiek után köthetőek callback-ként, majd ki kell alakítani a template-nek átadandó háromszintű kulcs-érték párosokból álló adatszerkezetet, ahol a legfelső szinten szintet az e-mail címek, az alatta levőt az egyes márkák nevei képviselik, a legalsó szinten a sablonban előbbiek alatt táblázatban megjelenített adatok vannak.

4. Összegzés

Szakdolgozatomban részletesen bemutattam az általam fejlesztett webalkalmazást, amely Misfit márkájú activity trackerek adatainak feldolgozására és interaktív megjelenítésére szolgál, a témával kapcsolatos kutatásokat hatékonyan elősegítve. Az alkalmazás platformfüggetlen, a Node.JS szoftvercsomag, illetve a további segédkönyvtárak telepítésével bármilyen modern számítógépen üzembe helyezhető, kezelése kényelmes és könnyedén elsajátítható.

A program elkészítése során igyekeztem figyelembe venni a webes világ újdonságait, gondosan megválasztottam a felhasznált modulokat, hogy azok a kezelhetőséget és a külalakot a forráskód áttekinthetőségének károsítása nélkül javíthassák. Az alkalmazás kódját a *model-view-controller* tervezési séma mentén építettem fel, amely elválasztja a nézeti sablonokat, az események vezérlését és az ezek mögött álló üzleti logikát. Számos esetben külön kliensoldali kódot elhelyeztem a helyben kiszámítható, illetve rövid, dinamikus kéréseket igénylő feladatok futásidejének, valamint a szerveroldalt érintő többletterhelésének csökkentésére. Az adatok lekérdezése a Misfit felhő típusú webes kiszolgálójáról történik, de a korábban kért adatok egy általam kifejlesztett egyszerű, helyi adatbázisban tárolódnak, jelentősen rövidítve ezzel a várakozási időt, amelyet a távoli szerver elérése generál.

A tervezés során számításba vettem lehetséges továbbfejlesztési eseteket is. Az alkalmazást a megvalósításnál könnyen átépíthetővé alakítottam, hogy más márkájú activity trackerek kezelői egyszerűen beilleszthetők legyenek a meglévő kódba. Ezenkívül lehetséges a meglévő adatok saját célra való letöltése is adattáblák szerint csoportosított szöveges, CSV formátumban, akár gyorsabb, automatizálható módon is JSON formátumú lekérdezések segítségével.

5. Irodalomjegyzék

- [1] Google Play:
<https://play.google.com/store/apps/details?id=com.misfitwearables.prometheus>
(2017. 11. 28.)
- [2] Apple iTunes App Store:
<https://itunes.apple.com/us/app/misfit/id564157241?mt=8>
(2017. 11. 28.)
- [3] Misfit:
<https://build.misfit.com/docs/cloudapi/overview>
https://build.misfit.com/docs/cloudapi/api_references#sleep
<https://misfit.com/customer/account/login/>
(2017. 11. 28.)
- [4] Node.JS:
<https://nodejs.org/en/about/>
<https://nodejs.org/en/foundation/members/>
<https://nodejs.org/en/download/>
(2017. 11. 28.)
- [5] Node Package Manager:
<https://www.npmjs.com/>
<https://www.npmjs.com/package/jsdom>
(2017. 11. 28.)
- [6] Express.JS:
<https://expressjs.com/>
<https://expressjs.com/en/guide/routing.html>
<https://expressjs.com/en/guide/migrating-4.html>
(2017. 11. 28.)
- [7] Handlebars.JS:
<http://handlebarsjs.com/>
(2017. 11. 28.)

- [8] JQuery:
<https://jquery.com/>
(2017. 11. 28.)
- [9] World Wide Web Consortium:
<https://www.w3.org/TR/html5/>
(2017 11. 28.)
- [10] Wikipedia:
[https://en.wikipedia.org/wiki/Ajax_\(programming\)](https://en.wikipedia.org/wiki/Ajax_(programming))
[https://en.wikipedia.org/wiki/Asynchrony_\(computer_programming\)](https://en.wikipedia.org/wiki/Asynchrony_(computer_programming))
https://en.wikipedia.org/wiki/Comma-separated_values
https://en.wikipedia.org/wiki/Front_and_back_ends
<https://en.wikipedia.org/wiki/JSON>
[https://en.wikipedia.org/wiki/Persistence_\(computer_science\)](https://en.wikipedia.org/wiki/Persistence_(computer_science))
https://en.wikipedia.org/wiki/Representational_state_transfer
(2017. 11. 28.)
- [11] Bootswatch:
<https://bootswatch.com/3/darkly/>
(2017. 11. 28.)
- [12] DigitalOcean:
<https://www.digitalocean.com/community/tutorials/how-to-install-node-js-on-an-ubuntu-14-04-server>
(2017. 11. 28.)
- [13] HowToGeek:
<https://www.howtogeek.com/218747/water-resistant-gadgets-arent-waterproof-what-you-need-to-know/>
(2017. 11. 28.)
- [14] CNET:
<https://www.cnet.com/how-to/water-dust-resistance-ratings-in-gadgets-explained/>
(2017. 11. 28.)
- [15] GitHub:
<https://github.com/uxsolutions/bootstrap-datepicker>
<https://github.com/ging/oauth2-example-client>

- https://github.com/ging/oauth2-example-client/blob/master/docs/oauth2_tutorial.pdf
(2017. 11. 28.)
- [16] W3Schools:
https://www.w3schools.com/js/js_arrays.asp
https://www.w3schools.com/js/js_object_prototypes.asp
(2017. 11. 28.)
- [17] IBM:
<https://www.ibm.com/developerworks/library/mo-prototype-watson/>
(2017. 11. 28.)
- [18] Salesforce:
https://developer.salesforce.com/page/Digging_Deeper_into_OAuth_2.0_on_Force.com
(2017. 11. 28.)
- [19] Microsoft Developer Network (MSDN):
[https://msdn.microsoft.com/en-us/library/xfhwa508\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/xfhwa508(v=vs.110).aspx)
(2017. 11. 28.)
- [20] Python:
<https://docs.python.org/3/tutorial/datastructures.html#dictionaries>
(2017. 11. 28.)
- [21] Oracle:
<https://docs.oracle.com/javase/8/docs/api/java/util/Map.html>
(2017. 11. 28.)
- [22] Mozilla Developer Network:
https://developer.mozilla.org/en-US/docs/Glossary/Callback_function
<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie>
https://developer.mozilla.org/en-US/docs/Glossary/Response_header
(2017. 11. 28.)
- [23] Fitbit:
<https://www.fitbit.com/>
<https://dev.fitbit.com/reference/web-api>
(2017. 11. 28.)

6. Mellékletek

A mellékelt DVD a forráskódot a gyökérkönyvtárban egy *misfit-elemzo.zip* névvel ellátott ZIP típusú tömörített fájlban tartalmazza.

A dokumentáció elektronikus változata a *dokumentacio* könyvtárban helyezkedik el Microsoft Word 2013, valamint PDF/A formátumban.