

---

**UNIVERSITATEA „SAPIENTIA” DIN CLUJ-NAPOCA**  
**FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE,**  
**TÎRGU-MUREȘ**  
**SPECIALIZAREA CALCULATOARE**

**Prezență bazată pe telefon**  
**PROIECT DE DIPLOMĂ**

**Coordonator științific:**

**Ș.l. dr. ing Vajda Tamás**

**Absolvent:**

**Kalith Norbert**

**2021**

UNIVERSITATEA “SAPIENTIA” din CLUJ-NAPOCA  
Facultatea de Științe Tehnice și Umaniste din Târgu Mureș  
Specializarea: Calculatoare

Viza facultății:

**LUCRARE DE DIPLOMĂ**

Coordonator științific:  
**ș.l. dr. ing. Vajda Tamás**

Candidat: **Kalith Norbert**  
Anul absolvirii: **2021**

**a) Tema lucrării de licență:**

Prezență bazată pe telefon

**b) Problemele principale tratate:**

- Studiu bibliografic privind sistemele de PWA
- Realizarea unei aplicații pentru prezență bazată pe telefon

**c) Desene obligatorii:**

- Schema bloc al aplicației
- Diagrame UML privind software-ul realizat.

**d) Softuri obligatorii:**

-Aplicație de prezență bazată pe telefon

**e) Bibliografia recomandată:**

- [1] R. Sam és P. LePage, „What are Progressive Web Apps?,” 6 január 2020. [Online]. Available: <https://web.dev/what-are-pwas/>. [Hozzáférés dátuma: 16 08 2020].
- [2] A. Tal, „Chapter 1. Introducing Progressive,” in Building Progressive Web Apps, United States of America, O’Reilly Media, Inc., 2017, pp. 16-29.
- [3] A. Oluwatofunmi, A. Chigozirim , O. Nzechukwu , Olawale és J. O. Olawale , „Dawning of Progressive Web Applications (PWA): Edging Out the Pitfalls of Traditional Mobile Development,” American Scientific Research Journal for Engineering, Technology, and Sciences , %1. kötet68, %1. szám1, pp. 85-99, 2020.

**f) Termene obligatorii de consultații: săptămânal**

**g) Locul și durata practicii:** Universitatea Sapientia,  
Facultatea de Științe Tehnice și Umaniste din Târgu Mureș

Primit tema la data de: 15.05.2020

Termen de predare: 27.06.2021

Semnătura Director Departament

Semnătura coordonatorului

Semnătura responsabilului  
programului de studiu

Semnătura candidatului

---

## Declarație

Subsemnatul/a Kalith Norbert absolvent(ă) al/a specializării Calculatoare promoția 2020/2021 cunoscând prevederile Legii Educației Naționale 1/2011 și a Codului de etică și deontologie profesională a Universității Sapienția cu privire la furt intelectual declar pe propria răspundere că prezenta lucrare de licență/proiect de diplomă/disertație se bazează pe activitatea personală, cercetarea/proiectarea este efectuată de mine, informațiile și datele preluate din literatura de specialitate sunt citate în mod corespunzător.

Localitatea, Târgu Mureș

Data: 24. 06. 2021.

Absolvent

Semnătura



---

# Prezență bazată pe telefon

## Extras

Lucrarea mea de diplomă se referă la proiectarea și implementarea unei aplicații web progresive pentru a ajuta profesorii și elevii să gestioneze și să urmărească prezența.

Gestionarea prezenței este o activitate care se repetă frecvent pentru a permite instructorilor să țină evidența prezenței elevilor la ore. Majoritatea profesorilor folosesc metode tradiționale pentru a înregistra prezența. Principalul dezavantaj al acestor metode este că sunt procese care necesită mult timp. Prin utilizarea instrumentelor digitale, timpul petrecut cu această activitate poate fi redus la minimum, iar înregistrarea prezenței este mai transparentă atât pentru profesori, cât și pentru elevi.

Scopul acestei lucrări de diplomă a fost de a proiecta și de a construi o aplicație care să faciliteze și să accelereze procesul de gestionare a prezenței, profitând de avantajele tehnologiei. Pentru a produce software-ul, am testat și am folosit o nouă metodologie de dezvoltare a software-ului numită Progressive Web Application. Am creat o aplicație independentă de platformă care utilizează tehnologia Web NFC API pentru a gestiona prezențele rapid și fiabil și am creat, de asemenea, o aplicație nativă bazată pe Android care utilizează tehnologia Bluetooth în cazul în care dispozitivul mobil al utilizatorului nu este capabil să utilizeze tehnologia NFC.

Rezultatul a fost o aplicație web progresivă multi-platformă și o aplicație nativă. Ambele software-uri, ținând cont de situația epidemiologică, oferă o urmărire a prezenței fără contact și accelerează și facilitează gestionarea și urmărirea prezenței.

**Cuvinte cheie:** managementul prezenței, aplicație web progresivă, independență de platformă, transmisie de date NFC

**SAPIENTIA ERDÉLYI MAGYAR  
TUDOMÁNYEGYETEM  
MAROSVÁSÁRHELYI KAR  
SZÁMÍTÁSTECHNIKA SZAK**

**Telefon alapú jelenlét  
DIPLOMADOLGOZAT**

**Témavezető:**

**Dr. Vajda Tamás**

**egyetemi adjunktus**

**Végzős hallgató:**

**Kalith Norbert**

**2021**

---

# Kivonat

A dolgozatom témája egy progresszív webalkalmazás megtervezése és kivitelezése, amely a jelenlétek kezelésében és nyomon követésében segíti a tanárokat és a diákokat.

A jelenlétkelés egy gyakran ismétlődő tevékenység annak érdekében, hogy az oktatók nyomon tudják követni a diákok részvételét az órákon. A legtöbb tanár a jelenlétek nyilvántartására a hagyományos módszereket használja. Ezen módszerek legnagyobb hátránya, hogy időigényes folyamatok. A digitális eszközök használatával erre a tevékenységre szánt idő minimálisra csökkenthető, valamint a jelenlétek nyilvántartása könnyebben átlátható a tanárok és a diákok számára is.

A dolgozat célja egy olyan alkalmazás kivitelezése és elkészítése volt, amely a technológia nyújtotta előnyök kihasználása által megkönnyíti és felgyorsítja a jelenlétek kezelésének folyamatát. A szoftver előállításához egy új szoftverfejlesztési módszertant teszteltem és használtam, amelynek a neve Progresszív Webalkalmazás. Létrehoztam egy platformfüggetlen alkalmazást, ami a Web NFC API technológia segítségével képes a jelenlétek gyors és megbízható kezelésére, valamint készítettem egy Android alapú natív alkalmazást is, ami Bluetooth technológiát használ arra az esetre, ha a felhasználói mobil eszköz nem képes az NFC technológia használatára.

Eredményként egy multiplatformos progresszív webalkalmazást, valamint egy natív alkalmazást kaptunk. Mindkét szoftver, a járványhelyzetet figyelembe véve egy kontaktus nélküli jelenlét nyomon követést biztosít, valamint felgyorsítja és megkönnyíti a jelenlétek menedzselését és nyomon követését.

**Kulcsszavak:** jelenlét kezelés, progresszív webalkalmazás, platformfüggetlenség, NFC adatátvitel

---

# Abstract

The theme of my thesis is the design and implementation of a progressive web application to help teachers and students manage and track attendance.

Attendance management is a frequently recurring activity in order to allow instructors to keep track of students' attendance in class. Most teachers use traditional methods to record attendance. The main drawback of these methods is that they are time-consuming processes. By using digital tools, the time spent on this activity can be minimized and attendance recording is more transparent for both teachers and students.

The aim of this thesis was to design and build an application that would facilitate and speed up the process of attendance management by taking advantage of the benefits of technology. To produce the software, I tested and used a new software development methodology called Progressive Web Application. I created a platform-independent application that uses the Web NFC API technology to manage presences quickly and reliably, and I also created an Android-based native application that uses Bluetooth technology in case the user's mobile device is not capable of using NFC technology.

The result was a multi-platform progressive web application and a native application. Both software, taking into account the epidemiological situation, provide a contactless presence tracking and speed up and facilitate the management and tracking of the presence.

**Keywords:** presence management, progressive web application, platform independence, NFC data transmission

# Tartalomjegyzék

Ábrák jegyzéke.....	10
1. Bevezető.....	12
1.1. Célkitűzések.....	13
2. Elméleti háttér .....	15
2.1. Alkalmazás .....	15
2.2. Webhely.....	17
2.3. Reszponzív weboldalak .....	18
2.4. Progresszív Webalkalmazás .....	20
2.4.1. PWA használatának előnyei:.....	21
2.4.2. PWA használatának hátrányai:.....	22
2.5. Jelenleg elérhető alkalmazások .....	22
2.5.1. Hagományos módszerek digitális környezetben .....	22
2.5.2. Bluetooth használata a jelenlét kezeléshez.....	23
2.5.3. Raspberry Pi és NFC olvasó.....	25
2.5.4. Okos jelenlét WIFI technológiával .....	26
3. Szoftver követelmények.....	28
3.1. Felhasználói követelmények .....	28
3.2. Rendszer követelmények .....	30
3.2.1. Funkcionális követelmények.....	30
3.2.2. Nem-funkcionális követelmények.....	32
4. A rendszer részletes leírása .....	35
4.1. Firebase szolgáltatások által megvalósított funkcionalitások.....	35
4.1.1. Hosting .....	36
4.1.2. Szükséges kód a Firebase használatához.....	37
4.1.3. Firestore.....	37
4.1.4. Authentication .....	38
4.1.5. Regisztráció .....	40
4.1.6. Bejelentkezés.....	41
4.1.7. Kijelentkezés .....	41
4.2. Felhasználói felület testre szabása .....	41
4.3. Firestore biztonsági szabályok.....	42
4.4. PWA megvalósítás .....	43
4.4.1. Manifest fájl .....	43



4.4.2.	Az offline tartalék oldal.....	44
4.4.3.	Service Workers .....	45
4.4.4.	Service Workers Life-cycle.....	46
4.4.5.	Service Workers eseményei .....	48
4.4.6.	Gyorsítótár – Cache Storage .....	49
4.4.7.	IndexedDB .....	52
4.5.	Near Field Communications (NFC) megvalósítás.....	53
4.5.1.	Web NFC API használata .....	54
4.5.2.	NFC címke beolvasása .....	54
4.5.3.	NFC címke írása.....	55
4.5.4.	NFC biztonsági engedélyek ellenőrzése .....	55
4.6.	Natív alkalmazás főbb komponenseinek bemutatása .....	55
4.6.1.	Bluetooth inicializálás és szükséges engedélyek megszerzése .....	56
5.	Üzembe helyezés és kísérleti eredmények .....	58
5.1.	Üzembe helyezés .....	58
5.2.	Felmerült problémák és megoldásaik .....	61
5.3.	Kísérleti eredmények .....	62
6.	Következtetések .....	66
6.1.	Összefoglaló .....	67
6.2.	Továbbfejlesztési lehetőségek .....	68
	Hivatkozások .....	69
9.	Függelék .....	72

## Ábrák jegyzéke

1. ábra: Reszponzív weboldal kép forrása: [9] .....	19
2. ábra Top1000 Mobil applikáció vs. Top 1000 mobil weboldal tulajdonságaik Kép forrása: [13] .....	20
3. ábra Attendance Tracker funkcionalitásai .....	23
4. ábra Blicher Kép forrása: [24].....	24
5. ábra Blicher - Saját azonosító létrehozása Kép forrás: [24] .....	25
6. ábra NFC olvasó és Raspberry PI kompozíció kialakítása .....	26
7. ábra My Attendance Tracker működése Kép forrása: [26] .....	27
8. ábra WiFi Analyzer Pro alkalmazás funkcionalítások .....	27
9. ábra - A rendszer használati eset diagramja .....	28
10. ábra PWA böngésző támogatottság.....	33
11. ábra - A rendszer architektúrája .....	35
12. ábra Firebase Hosting konfigurálása .....	36
13. ábra Firebase Hosting által létrehozott fájlok .....	36
14. ábra - lokális változók a Firebase szolgáltatások használatához-.....	37
15. ábra - A Real Time Listener használata a webalkalmazásban .....	38
16. ábra Firebase Authentication Kép forrása: [33].....	39
17. ábra Regisztrációs modal mobil és asztali böngészőben.....	40
18. ábra Bejelentkezési modal mobil és asztali böngészőben .....	41
19. ábra - Felhasználó státusz változásának figyelése Forrása: [31].....	41
20. ábra Firestore biztonsági szabályok .....	42
21. ábra A Manifest fájl kötelező tartalma .....	43
22. ábra - iOS ikonok optimalizálása .....	44
23. ábra - Google Chrome által biztosított offline oldal .....	45
24. ábra Service Workers feladatai Kép forrása: [36] .....	45
25. ábra Service Workers erőforrás használata .....	46
26. ábra Service Workers életciklusa Kép forrása: [36].....	47
27. ábra Service Workers telepítése .....	48
28. ábra Service Workers "Fetch Event" Kép forrása: [36] .....	48
29. ábra Statikus gyorsítótár hozzáadása.....	49
30. ábra Dinamikus gyorsítótár megvalósítása .....	50
31. ábra Gyorsítótárak verzió függővé tétele .....	51

32. ábra Fallback oldal/ kép megjelenítése .....	51
33. ábra IndexedDB működése Kép forrása: [36] .....	52
34. ábra IndexedDB engedélyezése .....	52
35. ábra NDEF típusú kommunikáció Kép forrása: [37] .....	53
36. ábra NDEF támogatás ellenőrzése .....	54
37. ábra Saját Mac cím lekérése .....	56
38. ábra A Bluetooth eszközök szkennelése és ezek elmentése .....	57
39. ábra Mobil és asztali webböngésző által felkínált telepítési lehetőség .....	59
40. ábra A mobil és webböngésző általi megerősítés az alkalmazás telepítésére .....	59
41. ábra A telepített alkalmazás ikonja a kezdőképernyőn .....	59
42. ábra A telepített alkalmazás .....	60
43. ábra Felhasználó azonosítása a natív alkalmazásban .....	61
44. ábra Natív alkalmazás kezdőképernyője .....	61
45. ábra Progresszív webalkalmazás tesztelésének kiértékelése .....	64
46. ábra Lighthouse teszt által vizsgált kritériumok .....	65

## 1. Bevezető

A technológiai fejlettségnek köszönhetően az emberek nem tudnák elképzelni az életüket a digitális eszközök és használatuk nélkül. A technológia olyan ütemben fejlődik, hogy nap mint nap jelennek meg az újabbnál újabb és a jobbnál jobb digitális eszközök. Ezek az eszközök beépültek az oktatásba is, az integrálódás oka pedig, hogy a tanárok a diákoknak a legkézenfekvőbb módszerekkel tudják majd a tudásukat átadni, szemléltetni és jobban megértetni. Manapság szinte lehetetlen volna elképzelni egy jól működő interaktív órát a multimédiás eszközök használata nélkül (kvízek, egyedi feladatok), éppen ezért napjainkban nagyon fontos szerepet játszanak az oktatásban is ezek a számítástechnikai megoldások. A koronavírus világjárvány kirobbanása során a kialakult az online oktatás, amely elképzelhetetlen lett volna olyan okos készülékek nélkül (okostelefon, laptop, táblagép, személyi számítógép), amelyek megfelelnek a mai modern elvárásoknak [1].

“Az internet egyszerre globális kommunikációs és médiarendszer, az információ és a tudás terjesztésének eszköze, az információs társadalom tagjainak egymás közötti interakcióját biztosító csatornája” [2, p. 42]. Ezért fontos, hogy az alkalmazás ne csak lokálisan tárolja az adatokat, hanem online is annak érdekében, hogy egyformán érje el a tanár és a diák is. Ezért felmerül egy olyan probléma, hogy mi van akkor, ha offline állapotba kerülő a felhasználó nem lesz képes elérni az online adatokat, ennek érdekében egy olyan alkalmazást szeretnék kifejleszteni, ami online és offline egyaránt jól működik.

A technológiai fejlődéshez a felsőoktatási intézmények gyorsan és hatékonyan tudnak alkalmazkodni, ezáltal érik el azt, hogy a lehetőségeknek megfelelően próbálják a modernebb és újabb technológiákat használni az oktatás során annak érdekében, hogy a megfelelő tudást és modern környezetet biztosítani tudják a diákoknak. De sajnos nem minden területen van meg ez a fejlődés, hiába van meg a technológiai fejlettség, ez általában azért alakult így ki, mert még nem találhatók meg a piacon erre a célra legalkalmasabb alkalmazások.

Megfigyelhető, hogy az egyetemeken még mindig a hagyományos jelenléti rendszereket használják. A jelenlét kezelése egy gyakran ismétlődő tevékenység (minden óra elején vagy végén), ami a hagyományos módszerekkel sok időt elvesz a tanórából. A papír alapú jelenlét írásnak több formája is megfigyelhető, azonban számos hátránnyal rendelkeznek. Egyik bevett módszer, amikor az oktató által kibocsátott papírra minden jelenlévő hallgató felírja a saját nevét és miközben folyik az óra minden tanuló sorra kerül. Ennek a módszernek azonban két komoly hátránya is van az első az, hogy könnyen kijátszható, mert senki se figyeli azt, hogy egy diák ne csak a saját nevét írja fel, a második nehézség, hogy ez plusz munkával jár a

tanárnak mert az óra után valahogy rendszereznie kell a jelenléteket. Egy másik módszer a névsorolvasás, amikor a diákok neveit egyesével olvassa fel a tanár és a visszajelzés alapján eldönti, hogy jelen van a bizonyos hallgató vagy nincs. Előnye ennek a módszernek, hogy nem lehet könnyen kijátszani viszont sokkal időigényesebb, mint az előző módszer és szükséges, hogy a tanárnak rendelkeznie kell a diákok aktuális névsorával. A legidőigényesebb eljárás, amikor maga az oktató írja meg a jelenléti ívet, minden jelenlévő diák nevének a feljegyzésével.

A legnagyobb hátránya a hagyományos jelenlét kezelő rendszereknek, hogy a diákok nem mindig tudják nyomon követni a saját részvételüket az órákon, hacsak nem jegyzik fel maguknak külön, valamint azok a tanulók, akik valamilyen eset folytán később érkeznek az órára nem minden esetben kapják meg a jelenlétet.

### 1.1. Célkitűzések

A dolgozat célja egy olyan szoftver tervezése és kivitelezése Progresszív Webalkalmazás technológiát használva, amely diákoknak és tanároknak az órákon való jelenlét kezelését oldja meg. A dolgozat válaszokat is keres arra, hogy mennyire lehet hatékonyan, és hol vannak határai a Progresszív Webalkalmazás technológiának.

Céлом egy olyan alkalmazás kivitelezése, amely megkönnyíti a tanárnak a jelenlétkezelést annak érdekében, hogy minél gyorsabban zajljon le, így kevesebb idő teljen el a tanórából e a folyamat lebonyolításával. Mindezt a jelenleg érvényes digitális eszközök által nyújtott technológiával lehet a legjobban megvalósítani, annak érdekében, hogy hatékony, gyors és megbízható legyen. Ennek érdekében megvizsgálom a jelenlegi megoldásokat és technológiákat, valamint hagyományos módszerekkel ötvözve létrehozok egy olyan alkalmazást, ami megfelel a mai igényeknek, amely egy biztonságos és felhasználóbarát környezetben valósul meg.

A legfontosabb funkcionálisok, amelyekkel kell rendelkezzen egy jelenlétkezelő alkalmazás azok a következők: felhasználók regisztrálása, valamint felismerése, új osztályok létrehozása és törlése, diákok felvétele a meglévő osztályokba esetleg a tanár meglátása szerint a diák törlése az osztályból, jelenlétek hozzáadása esetleg törlése, jelenlétek nyomon követése mind a tanárok mind a diákok számára.

A projekt két nagy részből tevődik össze: egy korszerű, letisztult és modern felhasználói felületből, valamint egy back-end részből. A felhasználó felület által bármely felhasználó képes lesz használni a back-endben megvalósított funkcionálisokat mindezt úgy

megvalósítva, hogy a felhasználó különböző eszközein a kezelő felület kialakítása ne változzon. A back-end feladata lesz az, hogy a felhasználó által kiváltott különböző funkciókat megfelelően kezelje és esetlegesen felmerülő hibák esetén közölje azt a felhasználókkal a felhasználói felületen.

Céljaink közé tartozik az is, hogy a progresszív webalkalmazást mint egy viszonylag új szoftverfejlesztési módszertant teszteljük. Kíváncsiak vagyunk arra, hogy mik azok a plusz szolgáltatások, amiket nyújt egy sima weboldalhoz vagy natív alkalmazáshoz képest és mik azok a funkciók, amiket nem lehet általa megvalósítani. Mindezt annak érdekében fogjuk elvégezni, hogy rájövünk arra, hogy a progresszív webalkalmazások képesek lehetnek-e majd a jövőben leváltani a natív alkalmazásokat, valamint jelen pillanatban melyik technológiát érdemes használni egy felmerülő probléma megoldására.

## 2. Elméleti háttér

Mikor nekiálltam olyan szakirodalmi cikkek böngészésének, amelyek segítségével kaphatok egy összefüggő képet, hogy jelen pillanatban milyen megvalósítások és technológiák vannak jelen a piacon a keresési eredmények nagyon számottevőek voltak. Sokféle megoldás született már a mobil alapú jelenlétet illetően. Kutatásom során sok hasonló jellegű applikációt is kipróbáltam, de azok nagy többsége nem felelt meg a mai modern elvárásoknak. Viszont találtam olyan megoldások is, amik megfeleltek a mai modern elvárásoknak: gyors működés, hasznos funkcionalitások, átlátható felhasználói felület, könnyű kezelhetőség. A hasonló alkalmazások tesztelésénél próbáltam az alkalmazás mögött lévő technikai háttér megértését előnybe helyezni, persze ha lehetséges volt és kaptam hozzá megfelelő szakirodalmi háttérrel vagy dokumentációt.

Voltak olyan megoldások, amik kivitelezése igényelt bizonyos helyi infrastruktúrát, de kaptam olyanok is, amelyeket bárhol lehet használni a megfelelő digitális eszközök használatával.

Három szoftverfejlesztési irányt tanulmányoztam annak érdekében, hogy el tudjam dönteni, hogy az elvárásaimnak melyik felel meg a legjobban: natív alkalmazás, weboldal és a kettő hibridje a progresszív webalkalmazás (PWA). Ezeket alább részletezem, hogy melyiknek mi az előnye és mi a hátránya. Ennél fontosabb viszont az, hogy bármelyik mellett is döntök az multiplatformos (vagyis több operációs rendszeren működőképes) megoldás legyen. Hiszen fontos dolog manapság az, hogy mindenki el tudja érni az alkalmazást operációs rendszertől függetlenül vagy ha a felhasználó több eszközzel rendelkezik akkor ugyanazt a személyre szabott élményt nyújtja neki az alkalmazás az összes eszközén.

### 2.1. Alkalmazás

Mi is az az alkalmazás? Az alkalmazások olyan programok, amelyek megkönnyítik a felhasználók mindennapjait. Az alkalmazásokat bárki letöltheti a megfelelő piacterről (App Store, Google Play Áruház, Steam, Epic Games Store) ingyen vagy pénz ellenében. Ilyen alkalmazások lehetnek a játékprogramok vagy valamilyen szolgáltatást nyújtó más típusú alkalmazások, vagy célalkalmazások. A mobilos alkalmazások képesek kihasználni a mobiltelefon előre beépített lehetőségeit, mint például: kamera, GPS helymeghatározó rendszer, névjegyzék, WiFi hálózat, Bluetooth vagy NFC. Az alkalmazások legfőbb célja a felhasználók mindennapi életének könnyebbé tétele vagy szórakozás biztosítása [3].

Céges szempontból két okból érdemes mobil applikációt használni. Az egyik, hogy egy adott alkalmazás egy weboldalt hivatott helyettesíteni, annak akár egy leegyszerűsített változata vagy offline is használható reprezentálása. A másik az, hogy egy különálló célt képviseljen, ami lehet a cég belső érdeke vagy haszonszerzési lehetőség [3].

A legtöbb alkalmazás kizárólag egyetlen típusú operációs rendszere lett kifejlesztve ahhoz, hogy más operációs rendszereken is elinduljon szükséges a szoftver egy reprezentatív másolata. Ezért egy adott platformon vagy egy bizonyos eszközön lehet csak ezeket használni. Az Android, iOS, Windows telefon, Symbian vagy Black Berry rendszerekhez készített alkalmazások csak és kizárólag a saját platformján használhatók. Az Androidra írt alkalmazást nem lehet iOS operációs rendszerre telepíteni vagy azon futtatni [4].

2014-ben a 14 évnél idősebb Románia lakosoknak a 12 százaléka rendelkezett okostelefonnal és 5 százaléka táblagéppel is. Ennek a százaléknak a legjelentősebb részét középiskolások és egyetemisták teszik ki. Ők rendelkeznek a legtöbb okoseszközzel is ezen eszközöknek a 38%-a okostelefon [5].

2016-ban Romániában az emberek 95,1%-a okostelefonról használja az alkalmazásokat. A legelterjedtebb operációs rendszer az Android (76,1%). Viszont a középiskolások 78,4% nem lenne hajlandó fizetni egy applikációért, az egyetemi hallgatók 86,1 százaléka hajlandó fizetni egy applikációért akár 10-20 lejes díjat [5].

Android vagy iOS operációs rendszerre éri meg jobban alkalmazást fejleszteni? Ez a kérdés nagy vita tárgya fórumokon, cikkekben, blogokban vagy akár egy beszélgetés során is felmerülhet ez a kérdés. Erre mindenkinek más a válasza, ami főleg személyes tapasztalatokból vagy felhasználói szokások miatt van. Ha alkalmazást szeretnénk csinálni az első legfontosabb kérdés, hogy melyik platformra fejlesszük. A legnagyobb felhasználóbázist Android alatt tudjuk elérni, de az iOS felhasználókról sem szabad megfeledkezni [3].

Bármely applikáció lehet hasznos ettől függetlenül nem minden cégnek van rá szüksége hiszen költséges a megtervezése és a karbantartása is. Az okos eszköz felhasználók átlagosan 3-4 új applikációt töltenek le hetente, amelyek főként ismert cégek vagy vállalatok termékei. Az esetek túlnyomó többségében viszont csak néhány alkalmazás marad meg egy napnál tovább a mobiltelefonokon. Főként azok az alkalmazások vonzzák a felhasználókat, amelyek megkönnyítik a mindennapi életüket vagy megfelelő szórakozást nyújtanak [3].

A natív alkalmazások legfőbb előnye a nagy teljesítmény, valamint a megfelelő működés. Ezt úgy érik el a fejlesztők, hogy kihasználják a digitális eszköz nyújtotta technológiákat. Ezáltal képesek olyan platformra szabott alkalmazást létrehozni, ami a felhasználó igényeit kiszolgálja az eszköz által nyújtott technológiai lehetőségekkel [5].



Multiplatform alkalmazás készítése megoldást nyújt arra, hogy egyetlen kódrendszer segítségével mindkét platformra működőképes alkalmazást készítsünk

A natív alkalmazások legnagyobb hátránya, hogy magasabb költségekkel járnak. Mivel a natív alkalmazás egy platformra íródott ezért, ha azt szeretnénk, hogy más platformon is elérhető legyen akkor az alkalmazás másolatát kell létrehozni egy más fejlesztői környezet alatt. Minden platformon különböző támogatásokra és karbantartásra van szükség az alkalmazások számára [5].

A multiplatformos megoldás ugyan lehetőséget biztosít arra, hogy ezeket a költségeket csökkentjük, de ez felvet egy újabb problémát, hiszen nem lesz elérhető néhány funkció minden operációs rendszeren és teljesítménybeli eltérések is előfordulhatnak [3].

## 2.2. Webhely

A website, magyarul webhely vagy weboldal egy olyan kiterjesztésű dokumentum, amit képes egy webböngésző megjeleníteni. Minden webhely több weblapból épül fel, a weblapok képezik a webhelyek tartalmát. Például a „<https://www.google.hu>” egy webhely. Azonban a magyar nyelvben ez a kifejezés nincsen használatban, helyette inkább a weblap, weboldal vagy a honlap kifejezéseket szokták használni mindezt helytelenül. Annak köszönhető ez, hogy az internet Magyarországon történő elterjedésekor az angolból származó „website” szónak nem volt magyar fordítása, így a médiában tévesen kezdték használni, ami azt eredményezte, hogy a köznyelvben is így kezdték használni [6].

A webpage magyarul weboldal vagy weblap egy website aloldalait jelenti. Tehát mindenik website több weblapból épül fel. A magyar nyelvben a weboldal és weblap szót is szoktuk használni mindezek a web megjelenésekor fordítási problémákból erednek. Homepage (index oldal) gyakorlatilag minden webhely főoldala. Magyar nyelven ez a Kezdőoldal, Főoldal vagy Nyitóoldal. Innen érhetők el a webhely főbb részei (regisztráció, bejelentkezés és a fontosabb aloldalak is) [6].

Webhosting angolul: „Internet Hosting”, amit a magyarban hostingként szokás rövidíteni. Ez sokféle szolgáltatást jelenthet akár szerver szolgáltatás, weboldalak tárolása vagy akár fájlok tárolása. A tárhely (vagy osztott tárhely) egy olyan típusú szolgáltatás, ami a webhelyhez szükséges szerver oldali háttérrel biztosítja. A weboldal onnan fog betöltődni a böngészőbe. Tárhelyre mindig szükség van ahhoz, hogy a webhely elérhető legyen az interneten. Ha az ember ilyen szolgáltatást vesz igénybe célszerűbb nem a legolcsóbbat kiválasztani azért, mert itt is mint minden más helyen a minőség sokat számít. Mint hogy az ember egy számítógépet

sem úgy választ ki, hogy a legolcsóbb alkatrészek legyenek benne, mert senki se szeretne órákat várni míg elindul egy böngésző [7].

A domain egy internetes címet jelent, magyarul tartománynév. Minden weblaphoz szükséges egy domain név mert a weboldal látogatói ezt fogják beírni a böngésző címsorába (pl.: „www.facebook.hu”) annak érdekében, hogy azt a bizonyos weblapot töltsék be. A magyar domain nevek a .hu- ra végződnek, míg a romániaiak .ro-ra. De ezen kívül léteznek másfajta végzések is mint a nemzetközi: .net, .org, .com, .me vagy mint az EU saját .eu végződése. A romániai weboldalak esetében érdemes a .ro végződést választani mert ez kelti a romániai látogatókban a legnagyobb bizalmat viszont a magyarországi weboldalaknál a .hu végződés ajánlott [8].

A „World Wide Web” magyarul világhálót jelent. Gyakorlatilag az interneten fellelhető és jelenleg is elérhető összes weboldalt jelenti. De sokan tévesen a kifejezést magára az internetre is használják. Azért is szokás a weboldalak neve elé a „www” oda írni mert ezzel lehet azt jelölni, hogy egy weboldal és nem egy e-mail szerver vagy bármi más. De manapság kicsit lekezdett kopni ez az előtag és sok weboldal esetében csak másodszintű domain neveket használnak. Ezért érdemes csinálni egy átirányítást, hogy ha valaki kihagyja a „www” előtagot akkor is a megfelelő weboldalt töltsse be a böngésző [6].

Ami nagyon fontos léteznek reszponzív weboldalak, amik mobilbarátok vagyis bármilyen képernyő mérethez képesek igazodni.

### 2.3. Reszponzív weboldalak

Ahogy azt a neve is mutassa „válaszol” az aktuális képernyőfelbontásra. Ez azt jelenti, hogy monitorról, tabletről vagy okostelefonról nézve is a weboldal tökéletesen fog kinézni mert a felhasználói felület fogja követni az aktuális képernyő méretét és felbontást és ezeknek megfelelően fog felépülni. Ha a mobil optimalizálásról beszélünk akkor az egyik legfontosabb dolog a „reszponzív weboldal ” kifejezés [3].

A reszponzív weboldalak sajátossága, hogy felbontásfüggetlenek (1. ábra). Bevezetésük azért volt szükséges, mert egyre több méretű és felbontású képernyőt tartalmazó eszköz jelent meg. Nagy előnye, hogy egyetlen weboldallal ki lehet szolgálni minden képernyőméretet. Mert nem lehet előre tudni azt, hogy a jövőben milyen új képernyőméretek fognak megjelenni. A képernyőméret és a felbontás eszköztől eszközre, típusról típusra és gyártótól gyártóra változik. Nagyon erőforrás igényes lenne 100 különböző oldalt létrehozni annak érdekében, hogy minden eszközön jól jelenjen meg (és ha jön egy új százegyedik akkor annak is egy új

weboldalt létrehozni). Ezek mellett könnyen bele is lehet zavarodni a rengeteg oldalba, könnyen létrejöhetnek duplikált tartalmak és a frissítés is nagyon nehéz. A reszponzív weboldal jó látogatói élményt biztosít eszköztől függetlenül. Mindenhez elég egyetlen URL cím. Nemcsak a fejlesztés, hanem a keresőoptimalizálás is leegyszerűsödik, ha nincsen külön mobil oldala egy webhelynek (pl.: m.weboldal.hu), akkor a reszponzív oldallal le lehet tudni a mobilitást is. A mobilra optimalizált weboldalakat is külön le kell kereső optimalizálni, ami tovább tart és drágább is. Könnyű a bővítése is mert általában ezek úgy vannak létrehozva, hogy rugalmasak legyenek. A legfőbb előnye talán az, hogy gazdaságos mert csak egyszer kell belenyúlni a kódba vagy magába a tartalomba és nem kell külön a mobilos és asztali verziót módosítani tartalom bővítéskor vagy fejlesztéskor [9].



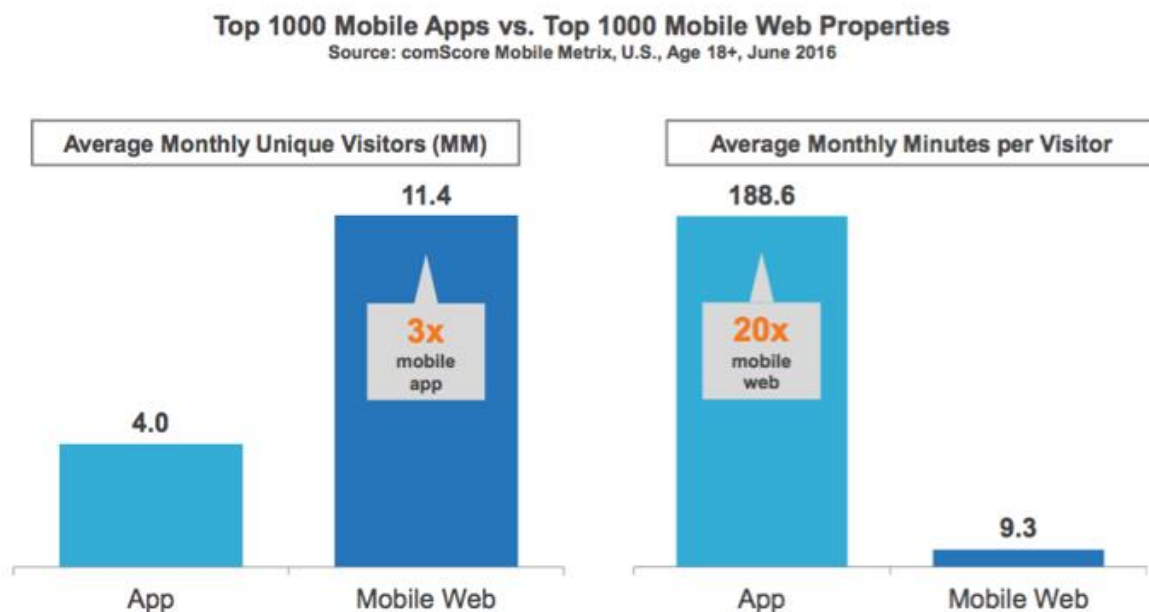
1. ábra: Reszponzív weboldal  
kép forrása: [9]

A Google is kifejezetten ajánlja a reszponzivitást, elismerik ők is azt, hogy a mobilra való optimalizálás legelterjedtebb formája ez a fajta design. A Google mobilbarát algoritmus az adaptív és reszponzív dizájnnal elkészült weboldalakat is mobilbarárnak rangsorolja [10].

De a mobilweben alacsony az elköteleződés ezért érdemes megnézni az app és a mobilweb használat számait is (2. ábra). Innen könnyen kiszűrhető az is, hogy a mobilweb közössége körülbelül háromszor akkora, mint az alkalmazásoké mindez mellett kétszer olyan gyorsan is nő a weboldalak száma. Hiába nagyobb a közösség viszont nem töltenek annyi időt egy mobilweben mint az alkalmazásokban [11].

Le lehet vonni azt a következtetést, hogy a mobilweb forgalmat nem mindig célszerű tartalommal kiszolgálni. A kihívást az jelenti, hogy a mobil weboldalak felhasználóit rá kell

vennünk arra, hogy letöltsék a megfelelő mobil applikációt. Ami nagy kihívást jelent mert az okostelefon használok fele havonta egy vagy nulla új alkalmazást tölt le [12].



2. ábra Top1000 Mobil applikáció vs. Top 1000 mobil weboldal tulajdonságaik  
Kép forrása: [13]

## 2.4. Progresszív Webalkalmazás

A progresszív webalkalmazások vagy progresszív webappok olyan típusú weboldalak, amelyek a mobilapplikációkhoz hasonló funkcionalitásokkal is rendelkeznek. Például képesek hozzáférni a Wi-Fi hálózathoz, kezelhetik a GPS helymeghatározó rendszert, kamerát, a telefon tárhelyét, Bluetooth vagy NFC rendszereket, de küldhetnek push üzeneteket is [14].

A progresszív webalkalmazásokat úgy fejlesztettek ki, hogy 3 fontos tulajdonságot betartásának: megbízhatóak és telepíthetőek legyenek és megfeleljenek az általuk hivatott célnak [15].

Manapság könnyen elkészíthető olyan hiper-helyi videókonferencia webalkalmazást, ami a WebRTC és a GPS helymeghatározó rendszer segítségével lett létrehozva. A WebGL és WebVR segítségével könnyen telepíthetővé lehet tenni egy ilyen típusú webalkalmazást. [15] A közelmúltig csak a natív alkalmazások támogatták ezeket a funkciókat. De még most is vannak olyan lehetőségek, amelyek jelen pillanatban még nem elérhetők, de a közelgő API-k majd megoldást fognak biztosítani ezekre a problémákra. Mindezt úgy próbálják megvalósítani, hogy biztonságérzetet kínáljanak a felhasználóknak, hogy egy webhelyre való belépés ne keltsen félelmet senkiben [16].

Mindemellett a megbízható progresszív webalkalmazás gyors és biztonságos kell legyen a hálózati kapcsolattól függetlenül is. Egy alkalmazásnak a sebessége rendkívül fontos, hogy egy megfelelő felhasználói élményt legyen képes nyújtani a felhasználók számára. Az oldallak átlagos betöltési sebessége 1-10 másodperc közötti időintervallumot fed le. Ráadásul a teljesítmény sem szabad csökkenjen a betöltési esemény után. Mivel a felhasználókat nem érdekli az, hogy egy gomb megnyomása után bekövetkezett-e az esemény. A görgetés és az animációk is simán akadozás mentesen kell lefussanak. Tehet a teljesítmény nagyon befolyásolja egy alkalmazás megítélését és a jó felhasználói élményt is. A felhasználónak a teljesség érzetét kell stimulálni az alkalmazáson belül [16].

Felhasználói elvárás az is, hogy az alkalmazások lassú, akadozó vagy offline kapcsolódási állapotban is elinduljanak. Azt remélik a felhasználók, hogy a legutóbb megtekintett adatok akkor is elérhetők legyenek amikor nehézkes a szerver elérése. Ha viszont mégse lehetséges egy kérést teljesíteni azt várják el, hogy az alkalmazás összeomlás vagy hallgatás helyett egy hibát adjon vissza. A felhasználók szeretik azokat az alkalmazásokat, amelyek, egy pillanat alatt reagálnak és megfelelő élményt nyújtanak számukra [15].

Telepítés után a progresszív webapplikáció egy új ablakban fog futni a böngésző lap helyett. Elvárás, hogy legyen elindítható a kezdőképernyőről vagy applikációs listából vagy a tálcáról is egyaránt pont úgy, mint egy egyszerű alkalmazás. Akár rájuk is kereshet a kereső felületen vagy az alkalmazás váltóval képes kell legyen ugrálni ezen alkalmazások között is azért, hogy azt az érzetet keltse, hogy az alkalmazás azon eszköz része amelyikre telepítve lett. Új lehetőségeket is nyújthat egy ilyen típusú alkalmazás. Akár alapértelmezett alkalmazássá is válhatnak más alkalmazások helyett [15] [17].

#### 2.4.1. PWA használatának előnyei:

A progresszív webalkalmazás mint szoftverfejlesztési módszertan ötvözi a weboldalak, valamint mobilalkalmazások pozitív tulajdonságait, ezáltal olyan funkcionalitásokat lehet megvalósítani, amelyek nem lennének elérhetőek egy átlagos weboldal számára [18].

Akár a weboldal kivitelezői is képesek megcsinálni az alkalmazást. Ebből adódik az is, hogy sokkal olcsóbb a megalkotásuk, mint egy vagy általában kettő natív alkalmazás esetében. A natív alkalmazásokkal szemben jóval kevesebb karbantartást igényelnek mert ha a jelenlegi operációs rendszernek új verziója jelenik meg akkor se kell kompilálási problémák miatt aggódni. Ha mind ezeket számokkal is szeretnénk szemléltetni akkor 20%-kal kevesebb egy progresszív webapplikáció fejlesztése és karbantartása, mint egy natív alkalmazásé [16].

#### 2.4.2. PWA használatának hátrányai:

Mint fent is említettem a sebesség nagyon fontos szempont egy alkalmazás kapcsán. De mégis a legnagyobb hátrányuk a sebességük. Persze egy egyszerűbb kisebb megoldások és alkalmazások esetében nem vevődik észre. De olyan alkalmazásoknál, ahol sok operációs műveletre lehet számítani a felhasználói oldalról ott inkább a natív applikációt éri meg választani [16].

A másik nagy hátránya a technológiából származik, mert számolni kell azzal is, hogy nem minden felhasználó használ kompatibilis böngészőt ezért náluk elé fordulhat, hogy nem fog megfelelően működni.

### 2.5. Jelenleg elérhető alkalmazások

A jelenleg elérhető alkalmazások többsége próbálja ötvözni a hagyományos módszereket a technológia által nyújtott lehetőségekkel. Megpróbáltam a jelenleg elérhető alkalmazásokat csoportosítani annak függvényében, hogy melyik miben tér el a hagyományos jelenlét kezelési módszerektől és miben tudnak újat nyújtani a digitális és technológiai fejlesztés által.

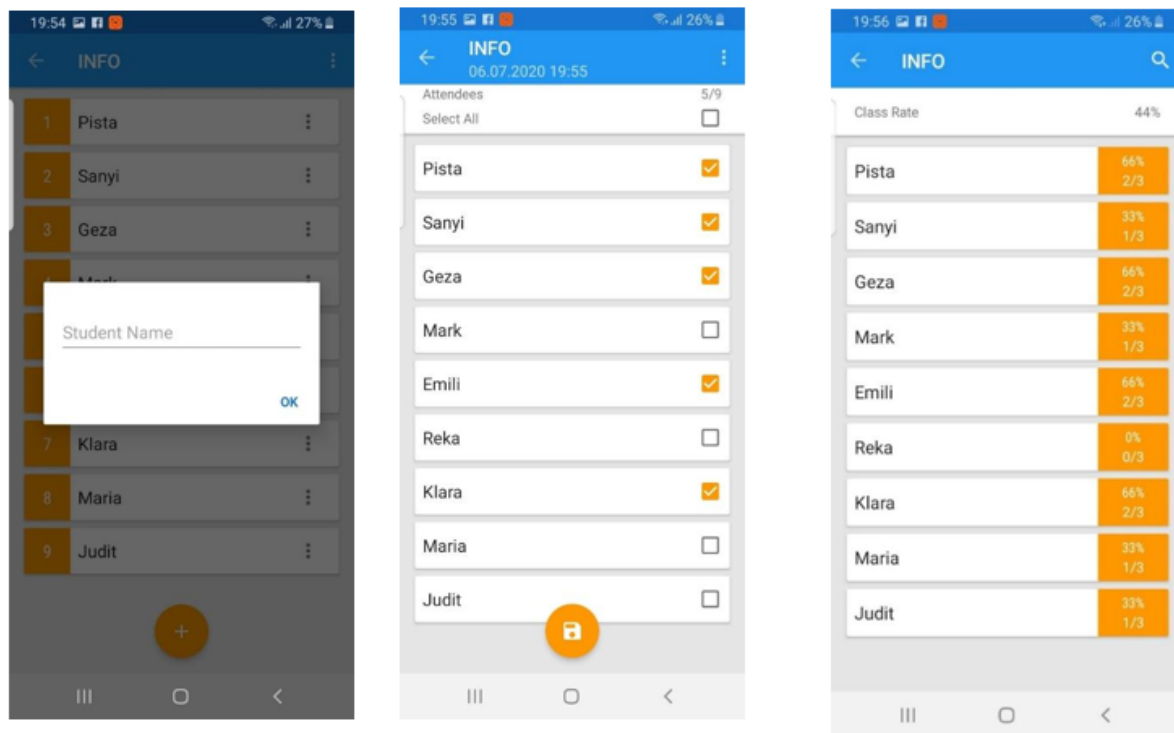
#### 2.5.1. Hagományos módszerek digitális környezetben

A legegyszerűbb megoldás egy a tanár digitális eszközén lévő alkalmazás jelenléte. Nem kellene így se internethez kapcsolódni se más digitális eszközök által nyújtott lehetőségeket használni. Egy lokális adatbázisban lehetne tárolni a szükséges adatokat (kurzusok, diákok, jelenlétek).

A tanár minden óra elején felolvas egy névsort és egyesével kijelöli, hogy ki van jelen és ki hiányzik. Ezután ezekből az adatokból képes az alkalmazás statisztikát csinálni, hogy ki hány órán volt jelen. A nagy előnye ennek az, hogy nagyon gyors és olcsón kivitelezhető. Ezáltal garantálni lehetne a megfelelő működést minden készüléken, így bármikor elérhető volna lassú, akadozó internet mellett is vagy akár offline állapotban.

Mivel kevés funkcionalitással rendelkeznek ezek az applikációk így biztosítják a gyorsaságot, könnyű kezelést és átláthatóságot. Viszont a hátránya az ilyen típusú applikációnak az, hogy a hagyományos módszereken alapul teljes mértékben. Ezért időigényes folyamat a diákok felvétele az adott osztályba, valamint a jelenlétek beírása. A jelenlétek nyomon követése csak a tanár számára lehetséges így a diák nem kap értesítést az esetleges hiányzásairól.

Az Attendance Taker nevű alkalmazás egyike azon tesztelt alkalmazásoknak, amelyek a hagyományos módszereket helyezik digitális környezetbe. Az alkalmazás elérhető a Google Play áruházból [19]. Az alkalmazás funkcionálisai: osztály hozzáadás, diák hozzáadása egy osztályhoz, jelenlét beírása, statisztika (3. ábra).



3. ábra Attendance Tracker funkcionálisai

Ezeknek a hátrányoknak a kiküszöbölésére több megoldást is találtam. Az időigényes osztály és diákok hozzáadása megtörténhet egy rendszergazda által, aki feltölti az osztályokat és a névsorokat egy adatbázisba és a tanár csak ki kellene válassza, hogy melyik osztályoknak tartja az óráit.

A legnagyobb probléma, ami rengeteg idővesztéssel jár az az, hogy a tanárnak minden óra elején fel kell olvasnia a névsort, így ezt a folyamatot automatizálni kellene valamilyen formában. A jelenlétek automatikus beírására több lehetőséget is találtam a jelenleg piacon lévő alkalmazások között ezek 23, 25, 26 oldalakon megtekinthetők.

## 2.5.2. Bluetooth használata a jelenlét kezeléshez

A Bluetooth technológia egyike a vezeték nélküli kommunikációs technológiáknak. Amely egyszerű, bárhol elérhető és biztonságos. Több milliárd eszközön elérhető technológia: mobiltelefonok, tabletek, laptopok, orvosi eszközök és otthoni szórakozásra alkalmas eszközök. Célja a kábelek helyettesítése miközben magas biztonságot képes nyújtani.

Egységes és lehetővé teszi a különböző eszközök kommunikációját, bármelyik ezt a technológiát alkalmazó eszköz képes más ugyanilyen technológiát használó eszközhöz való csatlakozásra [20].

Új áttörést a BLE (Bluetooth Low Energy) technológia hozta. Amely alkalmas az alacsony fogyasztásra így növelve az időtartamot. Akár egy érme méretű elemmel képes a BLE évekig működni. A Bluetooth-kompatibilis elektronikus eszközök közötti kapcsolatok lehetővé teszik ezen eszközök közötti kommunikációt vezeték nélkül kis hatótávolságú, ad-hoc hálózatokon keresztül, ami a Piconets névre hallgat. Piconetek létrejöhetnek dinamikusan és automatikusan, amikor a Bluetooth-kompatibilis eszközök belépnek a hálózatba. A Piconet hálózat minden egyes eszköze képes egyszerre kommunikálni. A technológia egyik alapvető erőssége az, hogy képes kezelni az egyidejű kéréseket. Ez a felhasználók számára rengeteg innovatív megoldást kínál [21] [22].

A Blicher (4. ábra) nevű alkalmazás Bluetooth technológia használatával oldja meg a jelenlétek automatikus beírását. Elérhető a gyártó hivatalos oldaláról [23] bárki számára. Az alkalmazás funkcióit: automatikus jelenlét beírás, óra közbeni interaktív kvízek.



4. ábra Blicher  
Kép forrása: [24]

Az alkalmazás működéséhez nem szükség manuálisan a két eszköz csatlakoztatására az alkalmazás használatához. Csak a Bluetooth-ot használja és láthatóvá teszi egy rövid időre. A tanári alkalmazás telepítése után a hallgatók letöltik a Student appot és bekapcsolják a



Bluetooth-ot és az alkalmazás által a hallgatók interakcióba kerülnek a tanári applikációval. A Blicher legjobb része az, hogy nincs szükség nehézkes hagyományos Bluetooth-párosításra [24].

Van egy Anonymus és egy azonosított módja is az alkalmazásnak. Valamint minden diák létrehozhat saját azonosított az alkalmazáson belül (5. ábra).



5. ábra Blicher - Saját azonosító létrehozása  
Kép forrás: [24]

### 2.5.3. Raspberry Pi és NFC olvasó

Ez a megoldás megköveteli NFC olvasó jelenlétét a mobil eszközökön annak érdekében, hogy lehetséges legyen az interakció a diák mobiltelefonja és a rendszer között.

Egyszerűen kihelyezzük a készüléket és a felhasználó el kell húzza az NFC címkéjét vagy okostelefonját amelyik rendelkezik NFC olvasóval. A kihelyezett szerkezet érzékeli az NFC kártyát és frissíti az NFC tulajdonosát az adatbázisban, hogy jelen van (időben érkezett vagy késéssel) vagy nincsen jelen. Ez az alternatíva nagyon jó a gazdaságos kivitelhez mert manapság az NFC technológia nagyon olcsó. AZ NFC várakozás nélkül is akár egyben képes észlelni a felhasználó NFC-jét. Hiába, hogy gyors és olcsó az NFC megoldás egy NFC kártya könnyen tönkre tehető, ha nem vigyázunk rá megfelelően, hiszen csak egy vékony kicsi tekercset tartalmaz [25].

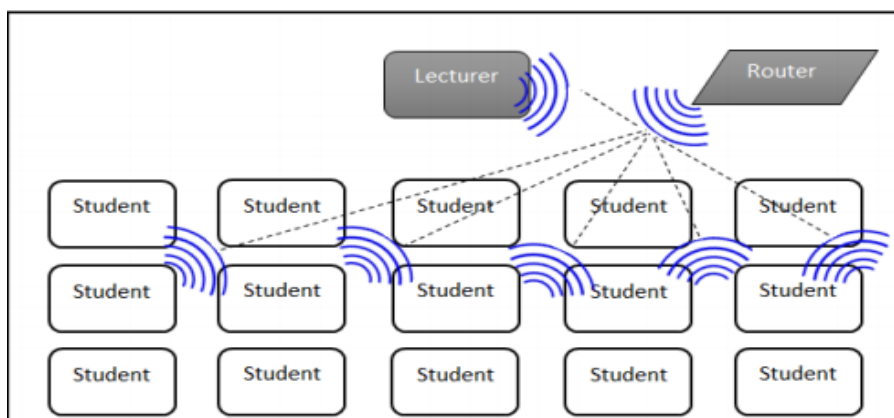
A Raspberry PI és NFC jelenlét beíró rendszer kinézetét a 6. ábra tartalmazza.



6. ábra NFC olvasó és Raspberry PI kompozíció kialakítása

#### 2.5.4. Okos jelenlét WIFI technológiával

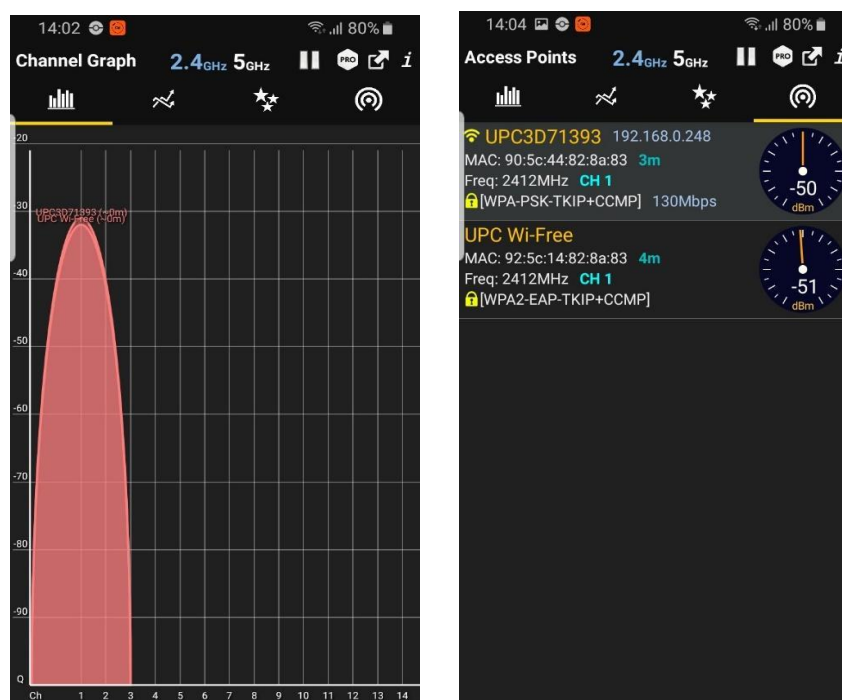
Az 7. ábra látható intelligens jelenléti rendszer egy olyan alkalmazás, amelyet a főiskolák, irodák és intézmények napi hallgatói részvételére fejlesztettek ki. Ez a rendszer egyetlen megkötése egy olyan eszköz, amely képes csatlakozni a lokális WiFi hálózathoz. A hallgatók csatlakoznak a WIFI-hez, majd bejelentkeznek a rendszerbe. Ezután a tanár kimentheti egy Excelbe a jelenléteket vagy akár statisztikát is készíthet belőle. A rendszer képes értesíteni a felhasználókat, ha a hiányzási arányuk nagyon megnő. Nem kell különálló nyilvántartást csinálni minden órán se statisztikát számolni kézzel a rendszer ezt megvalósítja magától. Előnye a biztonságosság. Az alkalmazás kihasználja pár specifikus elemét az operációs rendszernek. Egyik hátránya, hogy minden telefont kell csatlakoztatni a Wi-Fi rendszerhez, mivel egyes útválasztók egyszerre csak korlátozott felhasználót támogatnak így a hátránya a nagy létszámú osztályoknál lehet [26].



7. ábra My Attendance Tracker működése  
Kép forrása: [26]

Viszont van egy másik megoldási lehetőség is. Az elérhető Wi-Fi hálózatok jelerősség mérése. A rendszer érzékeli az elérhető Wi-Fi hálózattokat és azoknak a jelerősséget. Ha a tanár által érzékelt jelerősségek megegyeznek megadott százalékban a diák által érzékelt Wi-Fi hálózatok jelerősségével akkor az azt jelenti, hogy a diákat beírja a rendszer, hogy jelen van. Hátránya az, hogy sok intézményben nincsen megfelelő infrastruktúra kiépítve ennek megvalósításához. Mivel nagyon gyenge Access Pontok használata esetén a jelerősség komolyan eltérhet 10-15 méteren belül is.

A Wifi Analyzer Pro nevű alkalmazás (8. ábra) a közelben lévő WiFi hálózatokról gyűjt információkat és ezeket jeleníti meg a felhasználói felületen. Elérhető a Google Play áruházból [27]. Az alkalmazás funkcióit: elérhető hálózattok megjelenítése, elérhető hálózatok távolsága, elérhető hálózatok specifikációinak megjelenítése.

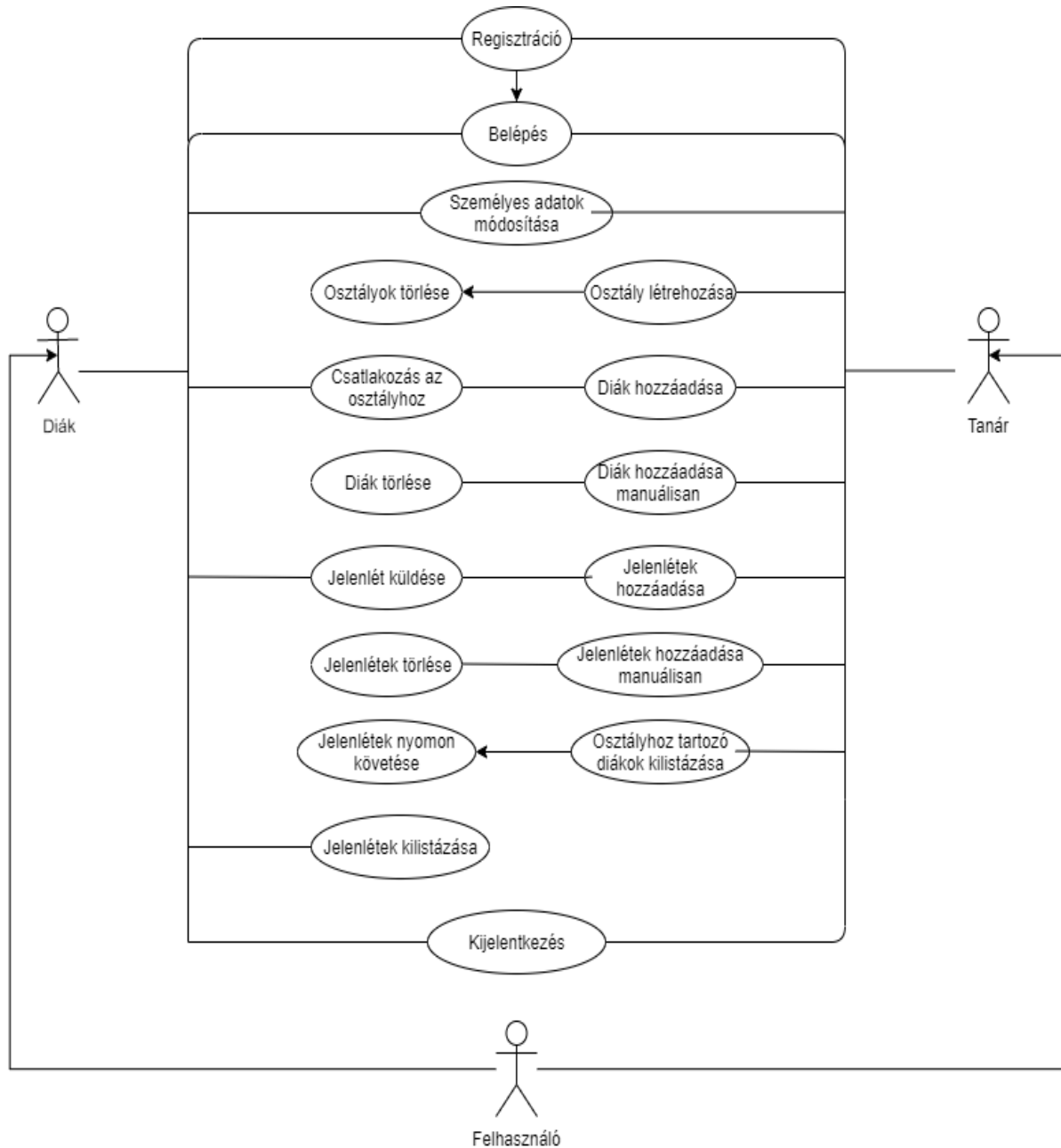


8. ábra WiFi Analyzer Pro alkalmazás funkcióit

### 3. Szoftver követelmények

#### 3.1. Felhasználói követelmények

A rendszer két típusú identifikált felhasználói móddal rendelkezik. Bármely felhasználónak lehetősége nyílik diákként vagy tanárként használni az alkalmazást. A felhasználói esetek bemutatása érdekében készítettem egy használati eset diagramot (use-case diagram), amely a 9. ábra tekinthető meg. A felhasználói követelmények tartalmazzák az összes elvárást a felhasználók irányába.



9. ábra - A rendszer használati eset diagramja

**Tanár:**

- Regisztráció lehetősége, annak érdekében, hogy a tanár a rendszer felhasználójává váljon.
- Belépés lehetősége, annak érdekében, hogy a tanár a különböző eszközein is képes legyen elérni a saját adatait.
- Új osztályterem létrehozása
- Diák hozzáadása a jelenléti listához kézi bevitellel
- Diák hozzáadása a jelenléti listához automatikusan NFC használatával
- Diák hozzáadása a natív alkalmazásból Bluetooth használatával
- Diák eltávolítása a jelenléti listáról
- Saját osztályok kilistázása a kezdőképernyőre
- Saját osztályok törlése
- Osztály adatok megtekintése
- Osztályhoz tartozó diákok jelenléteinek megtekintése
- Jelenlét hozzáadása a kiválasztott diáknak kézi bevitellel
- Jelenlét hozzáadása automatikusan a kiválasztott héthez
- E-mail küldése a diákoknak
- Saját profil adatok megjelenítése és módosítása

**Diák:**

- Regisztráció lehetősége, annak érdekében, hogy a diák a rendszer felhasználójává váljon.
- Belépés lehetősége, annak érdekében, hogy a diák a különböző eszközein is képes legyen elérni a saját adatait.
- Belépés különböző osztálytermekbe
- Jelenlét beírása
- Jelenlétek kilistázása
- Saját profil adatok megjelenítése és módosítása
- Felvett osztályok kilistázása

**Adminisztrátor:**

- Az oldal karbantartása jövőbeli funkciók tesztelése
- Admin jogadás más felhasználók számára

## 3.2. Rendszer követelmények

A rendszer megfelelően használatához a felhasználók be kell tartsanak néhány elvárás belli követelményt. Ha a felhasználó nem rendelkezik a megfelelő digitális eszközzel akkor nem lesz képes kihasználni a progresszív webalkalmazás vagy a natív alkalmazás nyújtotta lehetőségeket.

### 3.2.1. Funkcionális követelmények

#### **Tanár**

- Regisztráció és a kért adatok megadása: e-mail cím, ami megfelelő formátumban van, egy minimum 6 karakter hosszúságú és megfelelő biztonságú jelszó, amit nem használ korábban máshol, a tanár saját Neptun azonosítója, ami pontosan 6 karakter hosszúságú, teljes név, a digitális eszközének a Bluetooth Mac címe, majd ezen adatok elmentése a “Sign Up” gomb lenyomásával
- Bejelentkezés: azzal az e-mail cím és jelszóval párossal lehetséges, ami a regisztráció során elmentésre került az adatbázisba
- Jelszó visszaállítás: ha nem emlékszik az általa megadott jelszóra lehetősége van új jelszó beállítására, amit a regisztrált e-mail címre küldött link segítségével valósíthat meg
- Új osztály hozzáadása: a kezdőoldalon megjelenő „+” gomb lenyomásával, majd ezután meg kell adni a következő adatokat: osztály/csoport neve, tanár neve, tantárgy neve majd ezen adatok elmentése az adatbázisba az „ADD” gomb segítségével
- Osztályok megtekintése: az összes a tanár által létrehozott osztály a főoldalon fog megjeleni
- Osztály törlése: a megjelenő osztályok jobb oldalán levő törlés ikonnal lehetséges
- Osztály adatok és az osztályhoz tartozó jelenlétek megjelenítése: a kiválasztott osztályra kattintva megjelenik az osztály kezelőfelülete kicsit lefelé görgetve meg jelennek a jelenlétek
- Email küldése a diáknak: a kiválasztott diák jelenlétei után lévő „Email küldése ikon” lenyomásával lehetséges. Lenyomás után a digitális eszközön beállított E-mail küldő alkalmazás megnyílik.
- Diák hozzáadása/törlése: a „Student” gomb lenyomása után megjelenő menüből kilehet választani: automatikus/manuális hozzáadás és valamint a törlés lehetőségét

- Automatikus diák hozzáadása: az „Add Automatically” gomb lenyomásával lehetséges, majd a „Start Scan” gomb lenyomása után elkezdődik az olvasás, ezután a diák a saját megírt NFC-címkéjét csak oda kell érintse a tanár digitális eszközének azon részéhez, ahol található az NFC érzékelő
- Manuális diák hozzáadás: az „Add Manually” gomb lenyomásával lehetséges, majd ezután meg kell adni a diák saját 6 karakter hosszúságú Neptun azonosítóját
- Diák törlése: a „Delete Student” gomb lenyomásával lehetséges majd itt egy legördülő listából ki kell választani a törölni kívánt diákot
- Jelenlét hozzáadása/törlése: az „Attendance” gomb lenyomása után megjelenő menüből kilehet választani: automatikus/manuális hozzáadás és valamint a törlés lehetőségét
  - Automatikus jelenlét hozzáadása: az „Add Automatically” gomb lenyomásával lehetséges, itt a tanár egy legördülő listából ki kell válassza a megfelelő hetet, majd a „Start Scan” gomb lenyomása után elkezdődik az olvasás, ezután a diák a saját megírt NFC „tag” -jét csak oda kell érintse a tanár digitális eszközének azon részéhez, ahol található az NFC érzékelő
  - Manuális jelenlét hozzáadás: az „Add Manually” gomb lenyomásával lehetséges, ezután ki kell választani a hetet és a diákot név alapján két legördülő listából
  - Diák törlése: a „Delete Attendance” gomb lenyomásával lehetséges majd itt egy legördülő listából ki kell választani a hetet és a diák nevét majd a „Delete” gombot le kell nyomni
- Amennyiben a diák nem rendelkezik a megfelelő digitális eszközzel, vagy nincs lehetősége beszerezni NFC „tag” -et akkor lehetőség van egy Bluetooth alapú jelenlét megadására, amihez a natív applikációba kell bejelentkezni. Bejelentkezés után ki kell választani a hetet és a megfelelő osztályt egy legördülő listából. Kiválasztás után az „Add Attendance” gombot kell lenyomni.
- Profil adatok megtekintése: a menüből az „Account” elem megnyomásával történik
- Profil adatok szerkesztése: „Modify Profile” gomb lenyomása után ki kell tölteni a azokat a részeket amiket szeretne módosítani a felhasználó, ezek a következők lehetnek: teljes név, Bluetooth Mac Address, Neptun azonosító
- Kilépés: a menüben a „Logout” elem megnyomásával történik

## Diák

- Regisztráció és a kért adatok megadása: e-mail cím ami megfelelő formátumban van, egy minimum 6 karakter hosszúságú és megfelelő biztonságú jelszó jelszó amit nem használ korábban máshol, a diák saját Neptun azonosítója ami pontosan 6 karakter hosszúságú, teljes név, a digitális eszközének a Bluetooth Mac címe, majd ezen adatok mentése a “Sign Up ” gomb lenyomásával
- Bejelentkezés: azzal az e-mail cím és jelszóval párossal lehetséges, ami a regisztráció során elmentésre került az adatbázisba
- Osztályok megtekintése: a menüből a „Your Classes” elem megnyomásával történik, majd átkerül a diák a megfelelő oldalra
- Jelenlétek megtekintése: a kiválasztott osztály lenyomása által kilistázza az adott tantárgyhoz tartozó összes jelenlét
- Csatlakozás egy új osztályba: a „Join a New Class” gomb lenyomása által majd a telefont a személyes NFC „tag” -hez tartva lehet megírni a tag-et, ami a csatlakozáshoz szükséges információkat tartalmaz, majd ezt az NFC címkét kell a tanár telefonjához érinteni
- Jelenlét küldése: a „Send Attendance” gomb lenyomása által majd a telefont a személyes NFC „tag” -hez tartva lehet megírni a tag-et, ami a jelenlét beírásához szükséges információkat tartalmaz, majd ezt az NFC címkét kell a tanár telefonjához érinteni
- Jelenlét küldése, ha a diák nem rendelkezik megfelelő digitális eszközzel: a tanár kérésére be kell kapcsolni azon digitális eszköz Bluetooth-ját, aminek a Bluetooth Mac címét megadta regisztrációkor
- Profil adatok megtekintése: a menüből az „Account” elem megnyomásával történik
- Profil adatok szerkesztése: „Modify Profile” gomb lenyomása után ki kell tölteni a azokat a részeket amiket szeretne módosítani a felhasználó, ezek a következők lehetnek: teljes név, e-mail cím és jelszó
- Kilépés: a menüben a „Logout” elem megnyomásával történik

### 3.2.2. Nem-funkcionális követelmények

- A progresszív webalkalmazás minden olyan platformon működik, amely rendelkezik egy olyan böngészővel, amely megfelel a szabványoknak, beleértve a mobil és az asztali eszközöket is. A támogatott eszközök listáját a 10. ábra lehet megtekinteni.



- A natív alkalmazás futtatásához Android operációs rendszer szükséges, minimum Android 6.0 Marshmallow. A helyes működéshez szükséges, hogy a digitális eszköz rendelkezzen Bluetooth jeladóval.
- A regisztráláshoz, belépéshez és a felhasználó azonosításához szükséges a Firebase Authentication szolgáltatása
- Az osztályok, osztály tanulói, jelenlétek, profil adatok lekérdezéséhez, valamint módosításához a Firebase Firestore valós idejű adatbázis szolgáltatása szükséges
- Az progresszív webalkalmazás telepítéséhez a támogatott böngészők megléte szükséges (10. ábra)
- A diák rendelkezzen kell egy olyan digitális eszközzel, amelynek rendelkezik NFC támogatással és legalább egy NFC címkével, ha nem rendelkezik ilyen eszközzel akkor egy olyan eszközre van szüksége, ami rendelkezik Bluetooth jeladóval
- A tanár rendelkezzen kell egy olyan digitális eszközzel, amelynek van NFC támogatása, ha nem rendelkezik ilyen eszközzel akkor egy olyan eszközre van szüksége, ami rendelkezik Bluetooth jeladóval
- Az applikációba való regisztráláshoz a felhasználó kell rendelkezzen egy érvényes e-mail címmel, valamint érvényes Neptun azonosítóval
- Jelszó visszaállításához szükséges a felhasználó által megadott e-mail címhez való hozzáférés

Böngésző	Támogatás				
	Windows	Linux	Mac operációs rendszer	Android	iOS
Chromium-alapú	Igen	Igen	Igen	Igen	Nem
Firefox	Nem	Nem	Nem	Részleges	Nem
Safari	N/A	N/ A	Igen	N/ A	Igen (iOS 11.3+)

10. ábra PWA böngésző támogatottság

#### Chromium-alapú böngészők:

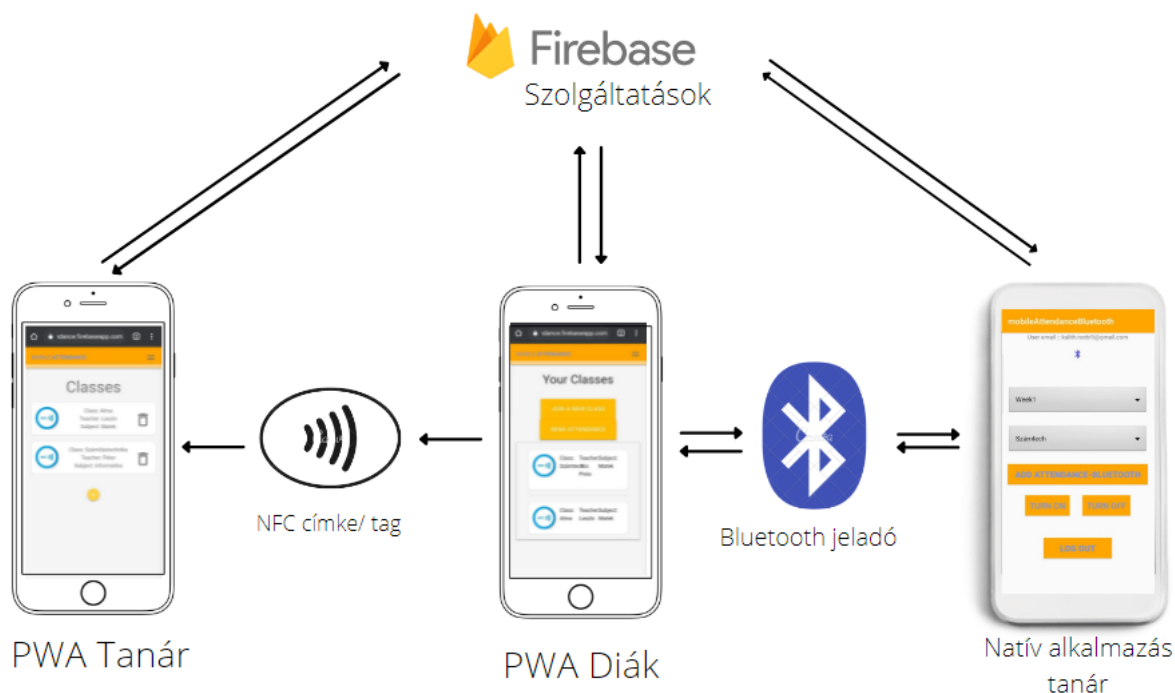
- Google Chrome
- Microsoft Edge
- Brave
- Opera
- Amazon Silk
- Samsung Internet
- Torch
- Epic BrowserA
- Avast Secure Browser

## 4. A rendszer részletes leírása

Fontos megemlíteni azt, hogy a progresszív webalkalmazás mint technológia fejlesztés alatt van a mai napig. Sok erőforrás nem elérhető vagy nagyon korlátozva van, amit a natív alkalmazások el tudnak érni. Ezért a kivitelezése sokszor körülményesebb, sőt nem is lehetséges bizonyos dolgoknak.

A progresszív webalkalmazás, valamint a natív alkalmazás a Firebase Authentication-t használja a felhasználó regisztrálására, valamint bejelentkezéskor történő azonosítására. Az adatok tárolására a Firebase Firestore valós idejű adatbázis szolgáltatást használtuk. A progresszív webalkalmazáshoz szükséges fájlokat a Firebase Hosting szolgáltatásán keresztül lehet elérni. Ezt szemlélteti a 11. ábra ahol a rendszer architektúrája látható.

A következő részben a létrehozott progresszív webapplikáció legfontosabb elemeit, valamint a felhasznált technológiákat fogom bemutatni.



11. ábra - A rendszer architektúrája

### 4.1. Firebase szolgáltatások által megvalósított funkcionalitások

A Firebase a Google olyan szolgáltatása, amely teljes platformot biztosít mobilalkalmazásunkhoz vagy weboldalunkhoz. A kisebb alkalmazásoknál ennek a segítségével már el lehet hagyni a szerver oldali implementációt, mivel ilyenkor maga a szolgáltatások csoportja lesz a szerverünk. A Firebase könnyen használható szolgáltatáscsomagokból épül fel, amelyek nagyban felgyorsíthatják az alkalmazások fejlesztését [28].

#### 4.1.1. Hosting

A Firebase Hosting gyors és biztonságos tárhelyet biztosít a webalkalmazások, statikus és dinamikus tartalmához, valamint a mikro szolgáltatásokhoz és azoknak a használatához. Hatalmas előnye az, hogy egyetlen paranccsal gyorsan és egyszerűen telepíthető a webalkalmazásunk számára, és egyaránt képes kiszolgálni a statikus és dinamikus tartalmakat [28].

A Firebase parancssori felület használatával a számítógépen található helyi könyvtárból a fájlokat áthelyezhessük a tárhely szerverre. Az összes tartalmat SSL-kapcsolaton keresztül, a globális CDN hálózat legközelebbi pontjából szolgáltatassa a felhasználók számára. [29]

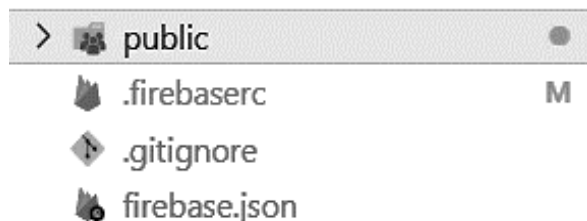
Első lépésben a Firebase eszközöket kell installáljuk a saját számítógépünkre globálisan. A következő paranccsal valósítható ez meg: „*npm install -g firebase-tools*” [29]. Ezt követően be kell jelentkezünk a Firebase felhasználónkba: „*firebase login*” [29]. Ezután inicializálunk egy új projektet a könyvtárunkban, ahol eddig dolgoztunk, vagyis ahol az alkalmazásunk található, amit fel szeretnénk tölteni a Hosting szolgáltatásra: „*firebase init*” [29].

A parancs futtatása után megjelenik egy lista (12. ábra), amiből ki kell válasszuk azokat az elemeket, amik nekünk kellenek. Mivel nekem első lépésben csak a Hosting szolgáltatásra volt szükségem így más opciót nem is választottam ki.

```
( ) Database: Configure Firebase Realtime Database and deploy rules
( ) Firestore: Deploy rules and create indexes for Firestore
( ) Functions: Configure and deploy Cloud Functions
>(*) Hosting: Configure and deploy Firebase Hosting sites
( ) Storage: Deploy Cloud Storage security rules
( ) Emulators: Set up local emulators for Firebase features
( ) Remote Config: Get, deploy, and rollback configurations for Remote Config
```

12. ábra Firebase Hosting konfigurálása

Az megfelelő opció/opciók kiválasztása után lehetőségünk van meglévő vagy akár új Firebase projectet is használni, ahonnan majd el szeretnénk érni a feltöltendő tartalmat. A parancsok futtatása után automatikus kigenerálja a szükséges állományokat (13. ábra) annak érdekében, hogy tudjuk használni a Hosting szolgáltatást.



13. ábra Firebase Hosting által létrehozott fájlok

A *public* mappába kell helyezzük az összes fájlt, amit szeretnénk, hogy feltöltsön az adatbázisba. Amit a következő paranccsal tehetünk meg: „*firebase deploy*” [29]. Sikeres futtatás esetén kapunk egy a Firebase által létrehozott linket, amin keresztül elérhessük a webalkalmazásunkat.

#### 4.1.2. Szükséges kód a Firebase használatához

Fontos kijelenteni, hogy ezeket a kódrészleteket a Firebase automatikusan kigenerálja, ezzel is megkönnyítve a fejlesztők dolgát. A kigenerált kódsorokat a webalkalmazásunk azon fájljaiba szükséges elhelyezni, ahol szükségünk van a Firebase szolgáltatásaira.

A Firebase kigenerál egy objektumot, ami tartalmazni fogja az összes szükséges információt a saját projektünk eléréséhez. A következő adatokat tartalmazza: API kulcsunkat, hogy azonosítsuk, melyik projektben vagyunk, a hitelesítő webhely domain címét, az adatbázis URL-jét, ami visszajelzést küld a csatlakozás állapotáról [30].

Szükségünk van lokális változókra (14. ábra), amelyeken keresztül képesek leszünk elérni a Firebase különböző szolgáltatásait.

```
const db = firebase.firestore();  
const auth = firebase.auth();  
const functions = firebase.functions();
```

14. ábra - lokális változók a Firebase szolgáltatások használatához-

#### 4.1.3. Firestore

A Firebase egyik szolgáltatás a Firestore vagyis a valós idejű adatbázis. Ezzel a funkcióval elfelejthetjük a háttér-kérelemre vonatkozó http-kéréseket, és helyette webaljakatokat használhatunk, amelyek sebessége csak az internetünk sebességétől függ. Ennek a szolgáltatásnak az egyetlen hátránya, hogy csak NoSQL adatbázist hozhatunk létre. A Cloud Firestore egy rugalmasan, méretezhető adatbázis a Firebase és a Google Cloud mobil-, web- és szerverfejlesztéséhez [31] [28].

A Firebase Realtime Database-hez hasonlóan, valós idejű hallgatókon keresztül szinkronban lehet tartani az adatokat az ügyfél alkalmazások között, és offline támogatást is kínál. Ezáltal az alkalmazásom a hálózati késéstől vagy az internetkapcsolattól függetlenül tud működni. A Firestore segítségével megvalósítható az összes szükséges adatbázis interakció (lekérdezések, tábla/elem hozzáadása, módosítás, törlés) [31].

Lehetőséget biztosít egy „Real Time Listener” létrehozásához (15. ábra), ami folyamatosan figyeli az adatbázis változásait. Ezáltal az összes módosítás nyomon követhető és így valós időben tudjuk módosítani a webalkalmazásunk által megjelenített tartalmakat. Ez azt jelenti,

hogy ha a felhasználó vagy egy külső esemény változást idéz elő az adatbázisban (dokumentumot add hozzá/ töröl) akkor a felhasználói felület is frissülni fog. Ezáltal szinkronban tudjuk tartani a felhasználó feleltünket az adatbázis gyűjteménnyel.

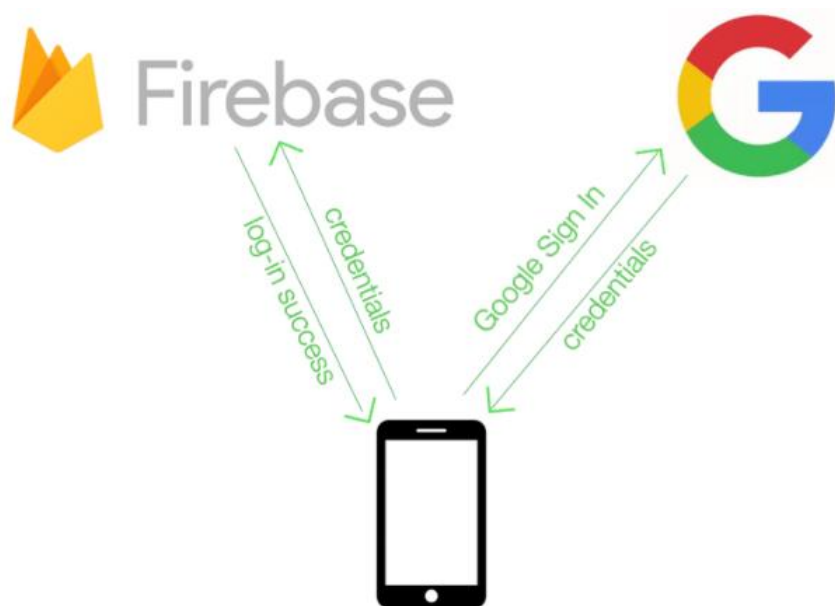
```
db.collection('Classes').where('userId', '==', user.uid).onSnapshot((snapshot) =>{
  snapshot.docChanges().forEach(change => {
    if(change.type === 'removed'){
      removeClass(change.doc.id);
    }
    if(change.type === 'added'){
      renderClasses(change.doc.data(), change.doc.id);
    }
    if(change.type === 'updated'){
      renderClasses(change.doc.data(), change.doc.id);
    }
  });
});
```

15. ábra - A Real Time Listener használata a webalkalmazásban

A „Classes” táblában történő összes módosítást figyeli az „onSnapshot” opció. Ha változás történik akkor küld egy pillanatképet az adatbázisról, amiben a változott dokumentumot és a változás típusát küldi el számunkra. Az „sanpshot” változó fogja tartalmazni ezt a pillanatkép. A „docChanges” metódus visszatéríti az összes változást, ami történt (ez történhet: oldal frissítéskor, betöltéskor). A „change” objektumon keresztül képesek vagyunk elérni ezeket a változásokat, a változás típusától (added, updated, removed) függően tudjuk őket lekezelni. 15. ábra megfigyelhető az, hogy a különböző változásokra más eseményeket váltok ki. Ha valamelyik osztály törlésre kerül az adatbázisban akkor meghívódik a „removeClass” metódus, ami a felhasználói felületről is eltávolítsa a törölt osztályt.

#### 4.1.4. Authentication

A Firebase külön szolgáltatással rendelkezik a felhasználók regisztrálására és beléptetésére. Ez az Authentication (16. ábra) nevű szolgáltatása. A Firebase Authentication lehetővé teszi a bejelentkezést egyszerű e-mail és jelszó használatával, de ezek mellett támogatassa a Facebook, Google vagy Twitter által történő bejelentkezéseket is [32]. A különféle közösségi média platformokkal történő bejelentkezés támogatásával növelhető a felhasználók száma, ez annak köszönhető, hogy a regisztrációs folyamatból kimarad a manuális kitöltés.



16. ábra Firebase Authentification  
Kép forrása: [33]

21. ábra

A progresszív webalkalmazásomnak ismernie kell a felhasználó személyazonosságát annak érdekében, hogy biztonságosan tudja tárolni a felhőbe és ezáltal képes legyen ugyanazt a személyre szabott felhasználói élményt nyújtani a felhasználó összes eszközén.

A Firebase Authentication SDK lehetőséget biztosít olyan felhasználók létrehozására és kezelésére, akik az e-mail címüket és jelszavakat használják a bejelentkezéshez. A Firebase Authentication kezeli a jelszó-visszaállító e-mailek küldését is [32].

A progresszív webalkalmazásomban az Email és Password bejelentkezést fogom használni, mert regisztrációkor szükségem van több adatra is. A szükséges adatok, amik kellenek: e-mail cím, érvényes Neptun felhasználó, teljes név, valamint a Natív applikáció miatt szükségem van a felhasználó digitális eszközének Bluetooth MAC címére is. Azért is választottam ezt a módszert mert nem mindenki használja a teljes nevét vagy esetleg csak becenevet használ különböző szociális weboldalakon és ebből könnyen történhetnek komplikációk a diákok azonosításakor.

A bejelentkezési és regisztrációs oldalak helyett modalokat használok. A modal egy oldalon belüli felugró ablak, ami abban segít, hogy a felhasználó az adott objektumra koncentráljon. Ezáltal a felhasználó végre kell hajtja a modalban szereplő utasítást annak érdekében, hogy tovább tudja folytatni a webalkalmazás használatát. A megjelenítése történhet automatikusan vagy interakció által. Az automatikus modalok általában valami egyszerű üzenetet közölnek a

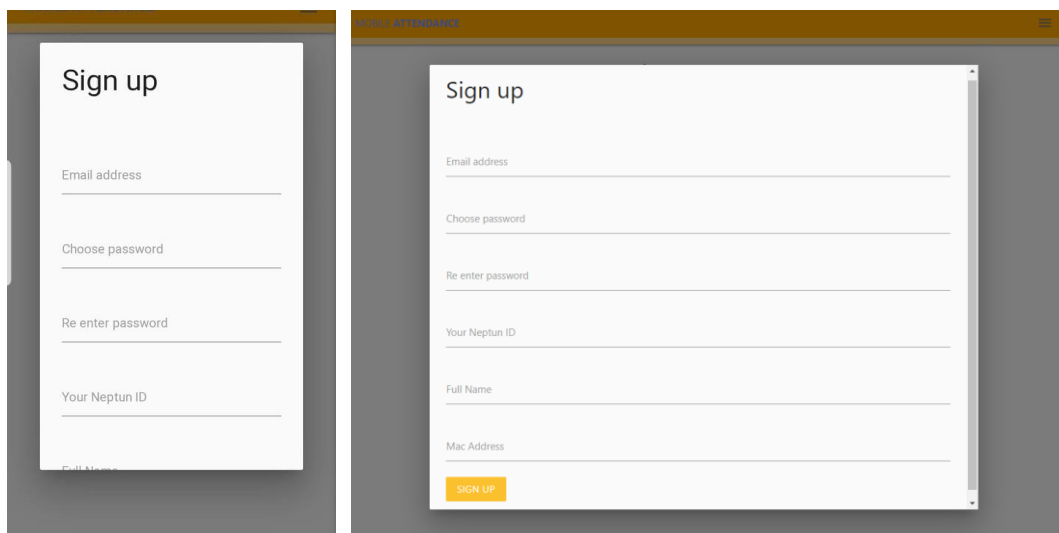
felhasználóval (figyelmeztetések, közlemények, hibázási lehetőségek). Az interakciós modalok általában egy tevékenység végrehajtására ösztönzik a felhasználót, így fókuszba helyezik a tevékenységet és összpontosítják a felhasználó figyelmét rá. A modalok megjelenítéséhez az ingyenes és bárki számára használható Materialize [33] metódusokat használtam. Ezek a Materialize hivatalos oldaláról töltöttem le.

#### 4.1.5. Regisztráció

A Firebase Authentication szolgáltatás használatához létre kell hozni egy referenciát, amelyen keresztül el tudjuk érni a szolgáltatás nyújtotta lehetőségeket (14. ábra).

A regisztrációnál az „*createUserWithEmailAndPassword* [32]” metódust használom itt át kell adjam a metódusnak a felhasználó által beírt e-mailt és jelszót. Sikeres lefutás esetén létrehozza az új felhasználói fiókot és egyből bejelentkezteti a felhasználót. A felhasználói modal a 17. ábra tekinthető meg.

A felhasználó regisztrálása után egy új felhasználói fiók jön létre, és összekapcsolódik a hitelesítő adatokkal - vagyis a felhasználó nevével és jelszójával, amelyre a felhasználó bejelentkezett. Amikor egy felhasználó regisztrál vagy bejelentkezik, akkor az a felhasználó lesz az „*Auth*” példány jelenlegi felhasználója. A példány megtartja a felhasználó állapotát, így az oldal frissítése (egy böngészőben) vagy az alkalmazás újraindítása után nem veszíti el a felhasználó adatait. Ez által oldottam meg azt, hogy ne kelljen minden oldalfrissítéskor újra bejelentkezni az alkalmazásba.

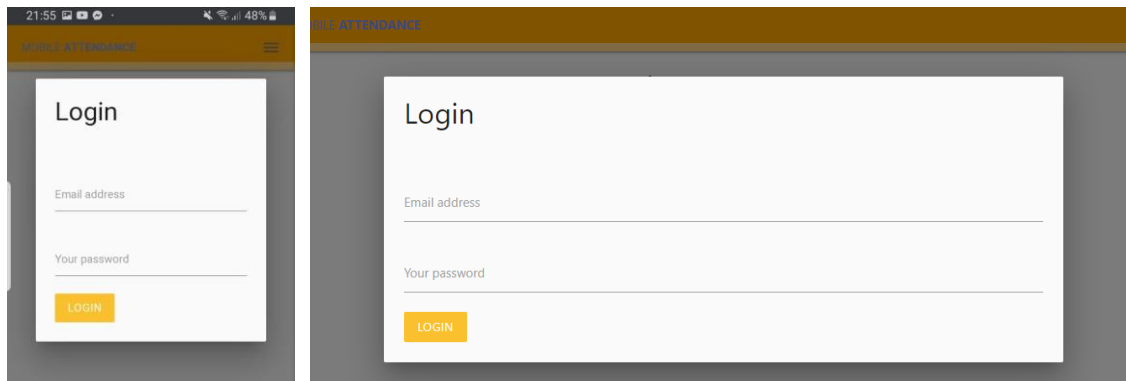
The image displays two versions of a 'Sign up' modal form. The left version is a mobile-optimized layout with a single column of input fields. The right version is a desktop layout with a similar single-column structure but with more spacing. Both forms include the following fields: 'Email address', 'Choose password', 'Re enter password', 'Your Neptun ID', and 'Full Name'. A yellow 'SIGN UP' button is positioned at the bottom of the form in both versions. The modals are set against a dark grey background with a brown header bar.

17. ábra Regisztrációs modal mobil és asztali böngészőben



#### 4.1.6. Bejelentkezés

A felhasználó jelszóval történő bejelentkezésének lépései hasonlóak az új fiók létrehozásának lépéseivel. A felhasználó bejelentkezéséhez a „*signInWithEmailAndPassword*” metódust használom, ennek a metódusnak is át kell adnom a felhasználó által beírt email címet és jelszót. Sikeres bejelentkezés esetén az „*Auth*” példány megkapja a felhasználó tokenét. A bejelentkezési modalok a 18. ábra tekinthetők meg.



18. ábra Bejelentkezési modal mobil és asztali böngészőben

#### 4.1.7. Kijelentkezés

A kijelentkezés nagyon könnyen megvalósítható a kijelentkezéshez is az „*Auth*” objektumra meghívom a „*signOut*” metódust. A metódusnak nem kell átadni az éppen bejelentkezve lévő felhasználó: felhasználónevét, jelszavát vagy e-mail címét. Amikor a felhasználó kijelentkezik, az *Auth* példány leállítja a hivatkozást a felhasználói objektumra, és már nem tartja fent az állapotát.

### 4.2. Felhasználói felület tesztelése

Az aktuális felhasználó megszerzésének módja egy megfigyelő beállítása az „*auth*” objektumra, ez a 19. ábrán figyelhető meg.

```
firebase.auth().onAuthStateChanged(function(user) {  
  if (user) {  
    // User is signed in.  
  } else {  
    // No user is signed in.  
  }  
});
```

19. ábra - Felhasználó státusz változásának figyelése  
Forrása: [31]

Azért van szükségem erre, hogy különböző felhasználó típusoknak különböző felhasználói felületet tudjak biztosítani. Ha nincsen bejelentkezve a felhasználó akkor ne férjen hozzá

olyan funkcionalitásokhoz, amelyek használatához csak a regisztrált felhasználóknak van jogosultságuk. Minden állapot változás esetén a webalkalmazás tartalma is változtatásra kerül. Például az admin felhasználók számára biztosított plusz funkcionalitásokat az oldal karbantartása érdekében ne érhék el az átlagos felhasználók. Így minden felhasználói típusnak biztosítható egy személyre szabott felhasználói felület kód duplikáció nélkül.

Használata által el tudom rejteni az adatokat az illetéktelen hozzáféréstől. Amíg a felhasználó nem jelentkezik be vagy regisztrál eltilthatóak számára az oldal bizonyos funkció így csak sikeresen azonosítás esetén férhet hozzá ezen funkciókhoz.

Az oldal betöltésekor minden olyan elem stílusa amelyik nem látható minden felhasználó számára „display: none” lesz. Miután ellenőrzöm a felhasználó típusát és állapotát beállítom a megfelelő elemeknek a láthatóságát.

### 4.3. Firestore biztonsági szabályok

A biztonsági szabályok használatával lehetőség van a felhasználói élmény kialakítására. A biztonsági szabályok elsődleges építő kővé a feltétel. A feltétel egy logikai kifejezés, ami meghatározza egy bizonyos adatbázis műveletet (írás, olvasás, módosítás) ki hajthat végre, vagyis engedélyezve van vagy nincs különböző felhasználói szinteken. Az összes Cloud Firestore biztonsági szabály „match” állítmányokból áll, amelyek azonosítják az adatbázisban lévő dokumentumokat és „allow” kifejezésekből, amelyek ellenőrzik a dokumentumokhoz való hozzáférést [31].

```
service cloud.firestore {
  match /databases/{database}/documents {
    match /Classes/{ClassesId} {
      allow read,write: if request.auth.uid != null
    }
    match /users/{userId} {
      allow read, update, delete: if request.auth != null && request.auth.uid == userId;
      allow create: if request.auth != null;
    }
  }
}
```

20. ábra Firestore biztonsági szabályok

A *Classes* tábla írási és olvasási művelete csak azoknak a felhasználóknak engedélyezett, akik be vannak jelentkezve a webalkalmazásomba. A felhasználók csak és kizárólag a saját adataikat olvashassák és módosíthassak a *Users* táblában annak érdekében, hogy ne történhessen adatlopás.

## 4.4. PWA megvalósítás

Ahogy fentebb már volt szó róla a progresszív webalkalmazások olyan típusú weboldalak, amelyek a mobilalkalmazásokhoz hasonló funkciókkal is rendelkeznek. Például képesek hozzáférni a WiFi hálózathoz, kezelhetik a GPS helymeghatározó rendszert, kamerát, a telefon tárhelyét, Bluetooth vagy NFC rendszereket, de küldhetnek push üzeneteket is.

### 4.4.1. Manifest fájl

A manifest fájl egy JSON-fájl, amely tartalmazza a szükséges leírásokat a böngésző számára, így a böngésző tudja, hogy sima weboldalként vagy progresszív webalkalmazásként kell kezelje az alkalmazásunkat. Tartalmaz minden szükséges információt arról, hogy hogyan kell viselkedjen böngészőben vagy a mobil eszközön telepítve. Egy tipikus jegyzékfájl, ami tartalmazza az alkalmazás nevét, az alkalmazás által használt ikonokat és a kezdő URL-címet, ami induláskor nyílik meg. Ezek mellett még sok más beállítást is meg lehet benne adni. A jegyzékfájlnak bármilyen neve lehet, de általában *manifest.json*-t használnak. Ezt a gyökérkönyvtárban (a webhely legfelső könyvtárában) szokás létrehozni. A specifikáció azt javasolja, hogy a kiterjesztés legyen *.webmanifest* de a böngészők a *.json* formátumot is támogatják [34].

```
"name": "Mobile Attendance",
"short_name": "M A",
"start_url": "/index.html",
"display": "standalone",
"background_color": "#FFF9D2",
"theme_color": "#FFF1C4",
"orientation": "portrait-primary",
"prefer_related_applications": false,
"icons": [
  {
    "src": "/img/icons/icon-72x72.png",
    "type": "image/png",
    "sizes": "72x72",
    "purpose": "any maskable"
  },

```

21. ábra A Manifest fájl kötelező tartalma

A „*name*” és „*short\_name*” tulajdonságok biztosítsák a telefonon és a telepítés után megjelenő nevet. Ha mindkettő adott akkor a „*short\_name*” a felhasználó kezdőképernyőjén az ikon alatt foglalt helyett, ahol korlátozott a megjelenítés. A „*name*” az alkalmazás telepítésekor használatos.

A „*start\_url*” kötelező tulajdonság. Általa tudtam beállítani azt, hogy az alkalmazás indításakor melyik oldalt kell először betöltsen a böngésző. A „*start\_url*” úgy kell

meghatározni, hogy a főoldalra navigálja a felhasználót induláskor és ne egy specifikusabb oldalra.

A „*background\_color*” tulajdonságot azért volt fontos számomra mert mikor az alkalmazás telepítve van és indul akkor ez ad egy háttérszínt a töltőképernyőnek.

A „*display*” tulajdonsággal meghatározom, hogy az alkalmazás indításakor milyen böngésző felhasználói felület jelenjen meg. Elehet rejteni a címsort és a böngésző sávját ezzel azt a hatást keltve mintha egy natív alkalmazást használna a felhasználó.

Az „*orientation*” azt határoztam meg, hogy az alkalmazás álló vagy fekvő módban jelenjen meg indításkor.

A „*prefer\_related\_applications*” tag egy logikai érték, amely segítségével meghatároztam, hogy webes alkalmazással szemben előnyben kell részesíteni a telepített alkalmazást.

Az „*icons*” tulajdonsággal határoztam meg a kép objektumok tömbjét. Amikor a felhasználó telepíti a PWA-t akkor a böngésző meghatározhat egy ikonkészletet számára a kezdőképernyőn, az alkalmazásindítóban, a feladatváltóban és a splash képernyőn. A Chrome esetében meg kell adnia legalább egy 192x192 pixel ikont és egy 512x512 pixeles ikont. Ha csak ezt a két ikon méretet van megadva, akkor Chrome automatikusan átméretezi az ikonokat az eszközhöz. Ha több ikon méretet adunk meg akkor telepítéskor kiválassza a legmegfelelőbbet. Minden kép objektumnak tartalmaznia kell az „*src*”, „*sizes*” és „*type*” tulajdonságot. A maszkolható ikonok használatához, amelyeket néha adaptív ikonoknak is neveznek az Android rendszeren, hozzá kell adnia „*purpose*”: „*any maskable*” tulajdonságot. Az iOS-en való tökéletes megjelenítés érdekében hozzáadtam egy „*apple-touch-icon*” tulajdonságot, ami az iPhone-n megjelenő ikon-t tartalmazza és az „*apple-mobile-web-app-status-bar*” tulajdonságot (22. ábra), ami a böngésző és a telepített applikáció felső sáv színét határozza meg.

```
<link rel="apple-touch-icon" href="/img/icons/icon-96x96.png">
<meta name="apple-mobile-web-app-status-bar" content="#AA7700">
```

22. ábra - iOS ikonok optimalizálása

#### 4.4.2. Az offline tartalék oldal

Mi történik, ha valaki internet elérés nélkül nyit meg egy applikációt? Általában van egy hibaüzenet, ami által tájékoztassák a felhasználót, hogy nem fér hozzá az eszköz az internethez és mit kellene csináljon esetleges hibák megoldása végett.

Ezzel szemben az interneten hagyományosan semmit sem kap offline állapotban. A Chrome biztosít egy offline oldalt, ami kiírja a lehetséges hibák okát sokan ezt a dinós vagy raptor játékként is ismerik (23. ábra).



## Nincs internet

Próbálja ki a következőket:

- A hálózati kábelek, a modem és a router ellenőrzése
- Újracsatlakozás Wi-Fi-hálózathoz
- [A Windows Hálózati diagnosztika futtatása](#)

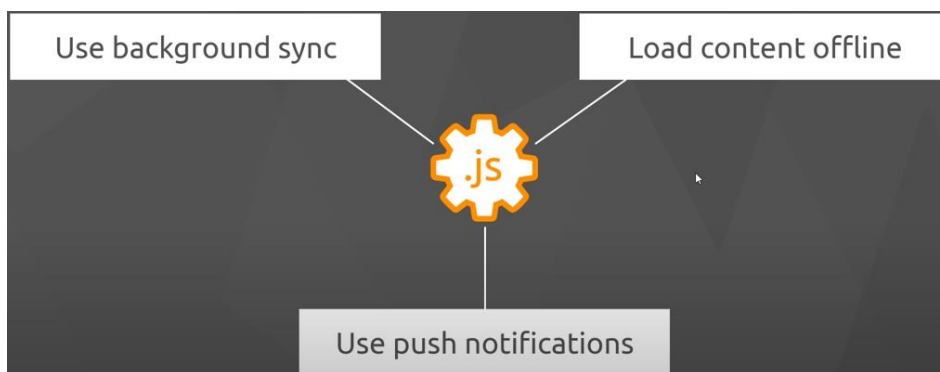
ERR\_INTERNET\_DISCONNECTED

23. ábra - Google Chrome által biztosított offline oldal

Ennek azonban nem kell így lennie. *Service Workers* és *Cache Storage* (gyorsítótár) alkalmazásával testreszabott offline élményt tudok nyújtani a felhasználóknak. Ezt lehet egy egyszerű oldal is, ami tartalmazza az információkat, hogy a felhasználó jelenleg offline állapotban van. De az is megvalósítható, hogy offline állapotban is elérhető legyen a webalkalmazásom amennyiben a szükséges fájlok a *Cache Storega*-ban vannak tárolva [35].

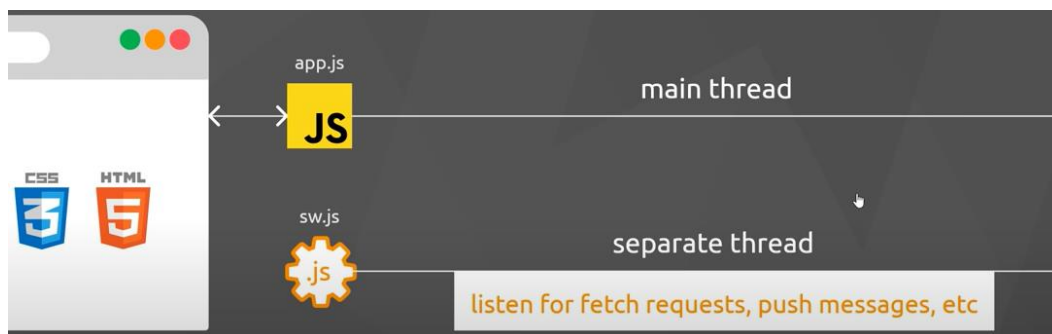
### 4.4.3. Service Workers

A *Service Workers* lehetővé teszi mindazt az alkalmazásom számára, amit egy mai modern alkalmazástól elvárnak. Lehetőséget nyújt azt adatok offline állapotban való megtekintéséhez, valamint gyorsabb hozzáférést biztosít a fájlokhoz azáltal, hogy nem kell minden alkalommal a fájlokat a kiszolgáló szerverről letöltsük. Ezáltal internet kapcsolat nélkül is lehetővé teszi számunkra a gyorsítótárazott adatok és a háttérszinkron használatát.



24. ábra *Service Workers* feladatai  
Kép forrása: [36]

Ha egy felhasználó megpróbál végrehajtani egy műveletet akkor általában internet kapcsolatra van szüksége (például egy adatbázis lekérést). De ha ez offline állapotban történik akkor a *Service Workers* a háttérben végrehajtsa a műveletet és amikor a kapcsolat helyreáll akkor történik meg az adatok szinkronizálása. A *Service Workers* a fájlokat a gyökérkönyvtárban kell elhelyezzük. Lehetőséget biztosít *push üzenetek* – használatára is, hogy az alkalmazásom valamilyen újdonságról értesítse az felhasználót az eszközén (például tartalmi emlékeztetőért).



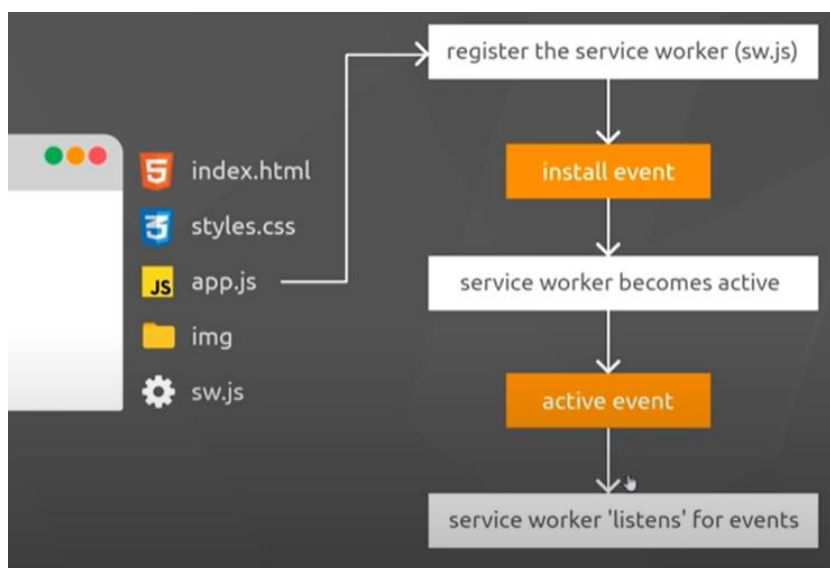
25. ábra *Service Workers* erőforrás használata

Ezeket *JavaScript* fájlokban valósítottam meg, de ezek nem a megszokott *JavaScript* fájlok. Amikor létrehozunk egy weboldalt akkor általában használunk *HTML*, *CSS* és *JavaScript* fájlokat. Az átlagos/megszokott *JavaScript* fájlok egyetlen szállón futnak, amely szorosan kapcsolódik a *HTML* oldalhoz és egyúttal hozzáfér a domhoz és manipulálhatja az oldal tartalmát. Ezek a kapcsolatok a *JavaScript* fájl és a dom között nagyon szorosak. A *Service Workers* *JavaScript* fájl nem ezen a fő szállón fog futni, amin a többi *JavaScript* fájl. Hanem elkülönítve egy másodlagos szállón. Ez miatt a másodlagos fájlnak nincs hozzáférése a domhoz így közvetlenül nem tudja olvasni vagy módosítani. Ez helyett a *Service Workers* fájlnek más szerepe van. A *Service Workers* fájlok a háttérben fognak futni. Ezek olyan háttérfolyamatok, amelyek futtatása nem szűnik meg ha a felhasználó bezárja a weboldalt vagy kilép a mobilos alkalmazásból. Az a feladatuk, hogy kezeljék az alkalmazás funkcióit a böngészőben előforduló események “lehallgatásával” és az eseményekre való reakcióval [35].

#### 4.4.4. *Service Workers* Life-cycle

Azért volt szükséges számomra a *Service Workers*, hogy a létrehozni kívánt webalkalmazásom úgy viselkedjen, mint egy natív alkalmazás. A *Service Worker*-nek van egy életciklusa, ami fontos az alkalmazásom számára mert a különböző életciklusok (26. ábra) alatt más-más funkciókat valósítok meg.

Ezeket a fájlokat a gyökérkönyvtárban hoztam létre annak érdekében, hogy hozzáférése legyen az összes fájlhoz, mert amikor egy alkönyvtárba helyeztem el akkor csak a könyvtáron belüli fájlokhoz kapott hozzáférést.



26. ábra Service Workers életciklusa  
Kép forrása: [36]

A *Service Workers* létrehozásakor az első lépésben, regisztrálnom kell a böngészőbe. A regisztrálását egy *app.js* fájlból oldottam meg, ami egy normális JavaScript fájl. Fontos, hogy *Service Workers*-ként kell regisztrálni a böngészőbe, mert mikor nem így oldottam meg akkor a böngésző nem ismerte fel. Ezután a böngésző elindítja a telepítési eseményt, amely egy másodlagos szálon fut. Miután a telepítése befejeződött aktívvá válik ekkor a böngésző indít egy aktív eseményt (*active event*). Ezen a ponton (miután aktív állapotba került) hozzáférhet a különböző oldalakhoz és fájlokhoz a hatókörében. Aktív állapotban „lehallgassa” a különböző lekérési eseményeket, HTTP-kéréseket és ha szükséges képes ezeket elfogni.

Oldal újbóli betöltésekor a böngésző ellenőrzi, hogy tapasztalható-e változás a fájlban. Ha nincs változás akkor nem történik meg az újratelepítés mert már ott van és nem törölődik oldalfrissítés esetén. Ha változás történik (13 ábra) akkor újra telepíti a *Service Workers* fájlt. Ez azt jelenti, hogy az új szolgáltatót aktiválja, majd a régit lecseréli.

#### 4.4.5. Service Workers eseményei

A telepítését az app.js fájlban valósítottam meg a telepítéshez szükséges kód a 27. ábra tekinthető meg.

```
public > js > JS app.js > ...
1   if('serviceWorker' in navigator){
2       navigator.serviceWorker.register('/sw.js')
3       .then((reg) => console.log('service worker ok',reg))
4       .catch((err) => console.log('some error',err))
5   }
```

27. ábra Service Workers telepítése

Első lépésben ellenőriztem, hogy a böngésző támogassa-e a *Service Workers* használatát. A „navigator” egy objektum JavaScriptben, amely a böngészővel kapcsolatos információkat tartalmazza. Ha támogatja akkor regisztrálom a *Service Workers*-t (27. ábra 2. sorában). Ezáltal a böngésző megkapja a sw.js fájlt. Aszinkron feladat így időbe telik amiig lefut.

Az sw.js fájlban használtam esemény figyelőket annak érdekében, hogy a különböző események (install, activate, fetch) után végezzen el hasznos feladatokat például gyorsítótárazza a különböző adatokat. A „self.addEventListener” paranccsal tudom a különböző eseményeket nyomon követni. A „self” az saját magára utal, az *addEventListener* figyel a különböző eseményeket (install, activate, fetch).

A *Fetch Event* (lehallgatási esemény) akkor jön létre amikor az webalkalmazásom kérést nyújt a webszervernek (kér egy képet, CSS fájlt). Viszont a *Service Workers* képes ezeket a kérési eseményeket lehallgatni. Ezen a ponton tudom „lehallgatni” ezeket a lekéréseket. A kérésekre különböző módokon tud reagálni a *Service Workers*: nem csinál semmit és tovább irányítsa a szerver felé, módosítsa a kérést végrehajtás előtt vagy le is állíthat bizonyos kéréseket és helyette egyedi kéréseket küld. Ennek a segítségével valósítottam meg a gyorsítótárazott adatok kezelését. A gyorsítótárazott adatok elérése sokkal gyorsabban történik, mint a szerverről való lekérdezés, valamint offline állapotban is elérhetőek.



28. ábra Service Workers "Fetch Event"  
Kép forrása: [36]



#### 4.4.6. Gyorsítótár – Cache Storage

A gyorsítótár használata azért volt fontos a webalkalmazásomban mert általa az alkalmazás gyorsabb lett, valamint eltudtam érni azt is, hogy gyenge, akadozó vagy teljesen offline állapotban is megfelelően működjön.

Tökéletes vezeték nélküli internet hozzáférésnél is a gyorsítótár megfelelő használatával jelentősen tudtam javítani a felhasználói élményt. Mert a gyakran használt fájlokat és adatokat a webalkalmazásom nem kell mindig újra betöltsse a kiszolgáló szerverről (például egy képet nem kell minden alkalommal újra letölteni a webhelyről, ha meg szeretnénk azt jeleníteni elég csak egyszer, valamint a gyakran használt és nélkülözhetetlen fájlokat is elég csak egyszer lekérjük a szerverről).

A *Cache Storage*-ba általában az alkalmazás fájl alapú tartalom betöltéséhez szükséges erőforrásokat szokás tárolni. Az *IndexedDB*-be pedig az egyéb adatokat. Az *IndexedDB* és a *Cache Storage* API minden modern böngészőben támogatott. Mindkettő aszinkron és nem blokkolják a főszálát [35].

A *Cache Storage* lehetőséget nyújt számomra, hogy csak azokat az adatokat tároljam el amelyek szükségesek a progresszív webalkalmazásom számára. Ezáltal tudom meghatározni a tartalmát és nem a böngésző tölti fel helyettem adatokkal, valamint bármikor vissza kereshetem az eltárolt adatokat amikor szüksége van rájuk a webalkalmazásunknak.

Két részre érdemes felosztani ezt a tárhelyet: egy statikus és egy dinamikus részre [35].

A Statikus gyorsítótárba azokat az erőforrásokat tárolom el, amelyek a webalkalmazás felhasználói felületét alkossák és elengedhetetlenek a megfelelő működés érdekében (CSS, JavaScript, HTML oldalak) ezek mind olyan fájlok, amiket nem kell gyakran módosítsak. Ezek teszik ki az webalkalmazásom alaptervét. Ezt a *Service Workers* „install” eseménye alatt valósítom meg mert ez minden alkalommal lefut amikor a *Service Workers* fájlt módosítom. Statikus gyorsítótár hozzáadása (29. ábra).

```
self.addEventListener('install', evt => {  
  evt.waitUntil(  
    caches.open(staticCacheName).then(cache => {  
      cache.addAll(assets);  
    })  
  );  
});
```

29. ábra Statikus gyorsítótár hozzáadása

A „*caches.open*” metódus által hozok létre egy új gyorsítótárat. Itt átadok egy karakterlánc a „*staticCacheName*” ami a gyorsítótár nevét tartalmazza. Ha nem létezik a statikus gyorsítótár,

akkor létrejön, ha viszont már létezik nem változtat rajta. Sikeres megnyitás esetén a „*cache.addAll*” metódussal hozzáadom az összes fájlt amelyet tartalmaz az „*assets*” változó. De viszont ez egy aszinkron feladat így időbe telhet míg ezeket a fájlokat elhelyezi a böngésző a gyorsítótárban viszont előfordul, hogy letiltotta a böngésző mert túl sok ideig futott. A probléma megoldásának érdekében az „*evt.waitUntil*” metódust használtam, ami addig vár amíg a „*cache.open*” ígéret igaz nem lesz, vagyis amíg be nem fejeződik a fájlok gyorsítótárba helyezése így mindaddig nem fejezi be a böngésző s telepítési eseményt amíg az ígéret nem teljesül.

Szembesültem azzal, hogy hiába van telepítve a gyorsítótár de a fájlokat a böngésző nem innen töltötte be. Ez azért volt mert a kéréseket nem szolgáltam ki a „*fetch*” esemény alatt.

Itt jön be a Dinamikus gyorsítótár fogalma. Ami azt jelenti, hogy változó tartalmú gyorsítótár lesz, ebbe mentsük el azokat az adatokat, amelyek nem szerepelnek a Statikus gyorsítótárban [35].

A dinamikus gyorsítótár megvalósítása a 30. ábra tekinthető meg.

```
cache.match(evt.request).then(cacheRes => {  
  return cacheRes || fetch(evt.request).then(fetchRes => {  
    return cache.open(dynamicCache).then(cache => {  
      cache.put(evt.request.url, fetchRes.clone());  
      maxCacheSize(dynamicCache, 99);  
      return fetchRes;  
    });  
  });  
});
```

30. ábra Dinamikus gyorsítótár megvalósítása

Első lépésben ellenőrzöm, hogy a kért fájlt tartalmazza-e valamelyik gyorsítótár. Ha tartalmazza akkor a gyorsítótárból fogom kiszolgálni a kérést, vagyis onnan fog betöltődni az adott fájl, ha viszont nem tartalmazza egyik gyorsítótár sem akkor elhelyezem a fájl másolatát a dinamikus gyorsítótárba. Azért a fájl másolatát helyezem el a gyorsítótárba mert az eredeti fájl fogja kiszolgálni az eredeti kérést. A „*cache.put*” metódussal adom hozzá a kért fájl másolatát a gyorsítótárhoz.

A dinamikus gyorsítótárnak a méretet korlátoztam egy maximum elemszámmra, annak érdekében, hogy ne használja fel a böngésző által biztosított maximális tárhelyet az progresszív webalkalmazásom. Ezt a „*maxCacheSize*” függvénnyel valósítottam meg. A függvény ellenőrzi, hogy hány elem található a dinamikus gyorsítótárban és ha az elemek száma meghalad egy korlátot akkor a legrégebb bekerült elemet eltávolítom onnan. A memória területének felszabadítására léteznek más algoritmusok is de nem mindig éri meg ezeket az algoritmusok választani.

Szükségem volt arra, hogy a gyorsítótárakat verzió függővé tegyem, mert, ha valamelyik gyorsítótárban változtattam egy fájlt akkor az nem került újra el tárolásra a gyorsítótárban. Ezért kellett a gyorsítótárakat verziókhoz rendelni. A régi verziót minden esetben törlöm emiatt az új verziójú gyorsítótárakat fogja használni a böngésző valahányszor változtatásokat végzek a gyorsítótárak tartalmán. Ezt a *Service Workers* „*activate*” vagyis aktiválási eseménye alatt oldottam meg (31. ábra).

```
    caches.keys().then(keys => {  
      return Promise.all(keys  
        .filter(key => key !== staticCacheName && key !== dynamicCache)  
        .map(key => caches.delete(key))  
      )  
    })
```

31. ábra Gyorsítótárak verzió függővé tétele

A „*keys*” egy aszinkron metódus hívást hajt végre, amely végigmegy az összes gyorsítótáron és megkeresi a kulcsokat (a gyorsítótár neveket). A „*filter*” metódussal szűröm a „*keys*” tömböt. Ha a „*key*” (jelenlegi gyorsítótárak neve) nem egyenlő a Statikus vagy Dinamikus gyorsítótár nevével akkor az állítás logikai igaz értékkel tér vissza és visszatéríti azt az elemet amelyik teljesítette az állítást. A „*map*” metódussal a kiszűrt elemeken végig megyek, majd a régi gyorsítótárakat törlöm a „*caches.delete*” metódussal.

A gyorsítótárak használatával megoldható az offline élmény. De mi történik akkor, ha valamelyik oldal vagy kép nem került be a gyorsítótárak egyikébe se? Akkor a böngésző megint a „Not connected to the Internet” oldalt tölti be a felhasználónak. Ennek elkerülése érdekében a *fetch* esemény alatt mikor ellenőrzöm a kéréseket és egy oldal vagy kép nem érhető el akkor kiszolgálom azt a kérést egy úgynevezett “fallback” vagyis visszaállítási oldallal vagy képpel (32. ábra) ezzel is növelve a felhasználói élményt.

```
    if(evt.request.url.indexOf('.html') > -1)  
    {  
      return caches.match('/pages/fallback.html');  
    }  
    if(evt.request.url.indexOf('.png') > -1)  
    {  
      return caches.match('/img/fallback.png');  
    }
```

32. ábra Fallback oldal/ kép megjelenítése

#### 4.4.7. IndexedDB

A Firestore (lásd Firebase) használja a böngésző beépített adatbázisát, vagyis az *IndexedDB*-t. Az *IndexedDB* egy nagyméretű *NoSQL* (Not Only SQL) tárolórendszer. Egy alacsony szintű API, amely nagy mennyiségű strukturált adat ügyféloldali tárolására alkalmas (33. ábra). Indexeket használ az adatok keresésére [36].



33. ábra IndexedDB működése  
Kép forrása: [36]

Mivel az progresszív webalkalmazásom offline is működőképes kell maradjon ezért az adatbázist offline állapotban is el kell tudjam érni. A probléma megoldásában segít az *IndexedDB* használata. Ha offline végez a felhasználó módosítást az adatbázison akkor az adatok először ide mentődnek el lokálisan és miután online állapotba kerül szinkronizálódik a webserverral. Ezért használom a Firebase Firestore nevű szolgáltatása mert ez elvégzi automatikus az adatok szinkronizálását (összes új adat feltöltése és az adatbázisból az új adatok letöltése) így nem kell manuálisan minden egyes adatot hozzáadjak az adatbázishoz miután újra online állapotba kerül a progresszív webalkalmazás.

```
db.enablePersistence()  
  .catch(err => {  
    if(err.code == 'failed-precondition'){  
      alert("Multiple tabs open!");  
    }else if(err.code == 'unimplemented'){  
      alert("Browser not supported");  
    }  
  })
```

34. ábra IndexedDB engedélyezése

A 34. ábra mutassa az IndexedDB funkció hozzáadását a webalkalmazásomhoz. Az adatbázis referencián keresztül engedélyezem a használatát az „*enablePersistence*” metódussal. Fontos, hogy lekezeljem a hibákat és ezeket tudassam a felhasználóval mert előfordulhat, hogy ha fennáll valamelyik hiba akkor nem lesz képes használni ezt a funkcionalitást. A hiba objektum rendelkezik egy kód tulajdonsággal, amit a Firebase küld a webalkalmazásnak. Az első eset, ha több oldal van nyitva mert ilyenkor szinkronizációs hiba léphet fel a nyitott

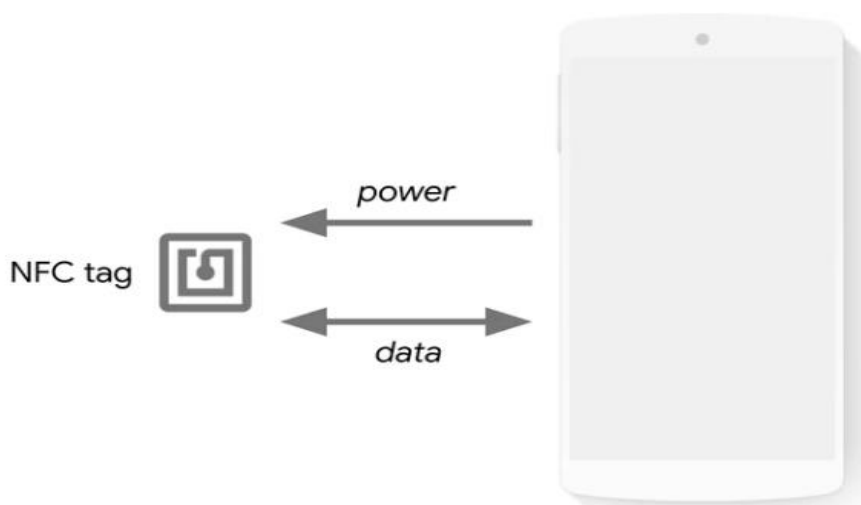
oldalak között. A második eset, ha a felhasználó webböngészője nem támogatja ezt a funkciót, de a mai modern webböngészőkben támogatva van.

A kivitelezés folyamata alatt találkoztam azzal a problémával, hogy az adatbázis lekérdezéseket a *Service Workers* elhelyezte a gyorsítótárban ezért az adatok szinkronizálása nem működött megfelelően. Ezt a problémát úgy oldottam meg, hogy ha a kérést a Firebase küldi akkor azokat figyelmen kívül hagyom.

#### 4.5. Near Field Communications (NFC) megvalósítás

Az NFC a Near Field Communications rövidítése, ami egy rövid hatótávolságú (1-10cm) vezeték nélküli technológia. A technológia 16,56 MHz-en működik és maximálisan 424 kbit/s-ig terjedő adatátviteli sebességet engedélyez. A Web API által engedélyezett NFC lehetővé teszi a webhelyek számára az NFC-címkék olvasását és írását, abban az esetben amikor a címkék a felhasználó digitális eszközének 1-10 cm-es közelében vannak [37].

**Fontos megemlíteni, a Web API nem támogatja még a készülékek közötti adatátvitelt NFC technológia használata mellett biztonsági okokból!** Jelenleg csak az NFC Data Exchange Format (NDEF) támogatott, amely egy egyszerű, bináris üzenetformátumra korlátozódik. Azért, mert az NDEF adatok írásának és olvasásának tulajdonságai könnyebben kezelhetők, vagyis nagyobb védelem garantálható az esetleges támadások ellen. Jelen pillanatban a Peer-to-Peer kommunikációs mód, az alacsony szintű I/O műveletek (NFC A/B, NFC-F) és a Host-based Card Emulation (HCE) nem támogatottak biztonsági okok miatt [38]. Az NDEF adatátvitel művelet diagramja 32. ábra látható.



35. ábra NDEF típusú kommunikáció  
Kép forrása: [37]

#### 4.5.1. Web NFC API használata

A hardver észlelésének folyamata eltér a szokásostól, mert itt nem a tényleges digitális eszköz hardverét fogom letesztelve hanem azt, hogy maga a böngésző támogassa-e az NFC használatát. Ez azt jelenti, hogy sok esetben a felhasználó saját kezűleg kell ellenőrizze a digitális készülékét, hogy támogassa-e az NFC technológia használatát.

A támogatás ellenőrzését a 36. ábra látható módon végzem. Az „*NDEFReader*” objektum azt jelzi, hogy a böngésző támogassa vagy esetleg nem támogassa a technológia használatát.

```
if ("NDEFReader" in window) {  
    const ndef = new NDEFReader();  
} else {  
    alert("Web NFC is not supported.");  
}
```

36. ábra NDEF támogatás ellenőrzése

Az „*NDEFReader*” objektum az NFC-címkékről érkező NDEF-üzenetek olvasására, valamint az NDEF-üzenetek hatótávolságon belüli NFC-címkékre történő olvasásáért felelős. Az „*NDEFReader*” objektum a Web NFC belépési pontja, ami előkészítő azokat az olvasási és írási műveleteket, amelyek akkor teljesülnek amikor egy NDEF címke a hatótávolságon belül kerül.

#### 4.5.2. NFC címke beolvasása

Először létrehoztam egy „*NDEFReader*” típusú objektumot. Erre az objektumra meg kell hívni a „*scan()*” metódust ami egy ígéretet térít vissza a sikeres olvasás befejeztével.

Az ígéret csak és kizárólag akkor oldódik fel, ha a következő feltételek teljesülnek:

- A felhasználó készüléke támogassa az NFC technológia használatát
- A felhasználó készülékén engedélyezve van az NFC használata
- A felhasználó engedélyezte azt, hogy a webhely interakcióba lépjen az NFC eszközökkel

A feltételek teljesülése után tesztelem, hogy olvasás közben történt-e valamilyen hiba. Ezt az „*onreadingerror*” metódussal végzem. Hiba akkor történhet olvasás közben, ha nem NDEF típusú eszközről vagy címkéről szeretnénk az olvasást elvégezni. Az olvasási művelet nem áll le ilyen esetben, de a felhasználót kell tájékoztatni, hogy nem megfelelő NFC eszközről szeretne adatokat olvasni.

A bejövő NDEF üzeneteket az „*onreading*” metódussal lehet megkapni. Ahhoz, hogy a leolvasott üzenetet megkapjuk végig kell mennünk az „*event.message.records*”

objektumokon. A *record.data* változóban van tárolva a bejövő üzenet, de, hogy ezt a pontos üzenetet megkapjuk szükséges, hogy a binárisan kódot dekódoljuk egy „*TextDecoder*” segítségével.

#### 4.5.3. NFC címke írása

Először létre kell hozni egy „*NDEFReader*” típusú objektumot. Erre az objektumra meg kell hívni a „*write()*” metódust ami egy ígéretet térít vissza.

Az ígéret csak és kizárólag akkor oldódik fel, ha a következő feltételek teljesülnek:

- A felhasználó készüléke támogassa az NFC technológia használatát
- A felhasználó készülékén engedélyezve van az NFC használata
- A felhasználó engedélyezte azt, hogy a webhely interakcióba lépjen az NFC eszközökkel
- A felhasználó a telefonját egy NFC címke 1-10 centiméteres közelébe tartsa és az NDEF üzenet sikeresen megírásra került

A szöveg írásához az „*ndef.write()*” metódusban át kell adni egy karakterláncot.

#### 4.5.4. NFC biztonsági engedélyek ellenőrzése

Az NFC használata által megnő az adatlopás vagy a felhasználó félrevezetésének az esélye, ezért az NFC elérhetőségét az Android operációs rendszer korlátozza. A Web NFC csak a legfelső szintű biztonsági protokoll használata mellett engedélyezett (HTTPS protokollt használó webhelyek esetében) [37].

Az „*NDEFReader*” „*scan()*” és „*write()*” metódusának a használatához a felhasználó felszólításra kerül annak érdekében, hogy engedélyezze az NFC hozzáférést a webhely számára abban az esetben ha a hozzáférés nem volt már korábban engedélyezve.

A beolvasás vagy írási művelet megkezdéséhez a weboldalnak láthatónak kell lennie amikor a felhasználó megérinti az eszközzel az NFC címkét. A böngésző jelzi, hogy NFC címkét észlelt és azt is, hogy amíg nem fejeződik be a művelet tartsa közel az NFC címkéhez a telefont. Ha a weboldal nem látható akkor az NFC általi tartalom fogadás és küldés le van tiltva mindaddig amíg újra nem lesz látható a weboldal.

### 4.6. Natív alkalmazás főbb komponenseinek bemutatása

A natív alkalmazás azért lett kifejlesztve mert nem minden mobiltelefon rendelkezik NFC technológiával viszont a mai modern mobiltelefonokban a Bluetooth megléte egy kötelező dolog. Bluetooth jeladó nélkül a felhasználók sokkal kevesebb mobil tartozékot tudnának

használni. Ezáltal sokkal több felhasználó tudja majd használni a jelenlétkelő alkalmazásomat.

#### 4.6.1. Bluetooth inicializálás és szükséges engedélyek megszerzése

Regisztrációkor azért szükséges a felhasználónak megadnia a digitális eszközének a MAC címét, mert biztonsági okokból ez a funkció nem érhető el azokon az Android eszközökön, amelyek Android 6.0 vagy régebbi verziót használnak. Ennek az az oka, hogy illetéktelen alkalmazások ne szerezzenek információt a telefon hardver címeiről adatlopás és más biztonsági okok megelőzése miatt [39].

Ezért a 37. ábra látható kód a telefon saját Bluetooth Mac címe helyett mindig a 02:00:00:00:00:00 értéket téríti vissza.

```
BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();  
String macAddress = mBluetoothAdapter.getAddress();
```

37. ábra Saját Mac cím lekérése

Az alkalmazásomnak szüksége van arra, hogy a Bluetooth elérhető legyen az mobiltelefonon. Bluetooth használatát kell engedélyezni az alkalmazás számára. A felhasználó be kell kapcsolja a Bluetooth-ot és az alkalmazás számára engedélyezve kell legyen a közelben található Bluetooth eszközök észlelése és felderítése. A manifest fájlba elhelyeztem ezen jogosultságokhoz való hozzáférési kérelmeket.

Első lépésben egy „BluetoothAdapter” típusú változót hoztam létre. Ezután az értékét „BluetoothAdapter.getDefaultAdapter()” értékre állítottam. Ezáltal érem el a telefon alapértelmezett Bluetooth adapterét.

A `getDefaultAdapter()` segítségével tudjuk elérni a mobil alapértelmezett Bluetooth adapterét. Ezután a felhasználó képes az alkalmazásból bekapcsolni a Bluetooth-ot a mobiltelefonján. Bekapcsolás után a Bluetooth felfedezhetőséget kell engedélyezni az alkalmazás számára. Az alkalmazásom miután minden szükséges engedélyt megkapott a felhasználótól elindul a közelben lévő eszközök szkennelése.

Létrehoztam egy listát, amiben a közelben lévő Bluetooth eszközök adatait tárolom el (38. ábra), annak érdekében, hogy a szkennelés után végig tudjak menni ezen a listán és a megkapott MAC címek ellenőrzése után a jelen lévő diákoknak beírja a jelenlétet.



```

private val bluetoothDiscoveryResult = object : BroadcastReceiver() {
    override fun onReceive(context: Context?, intent: Intent?) {
        if (intent?.action == BluetoothDevice.ACTION_FOUND) {
            val device: BluetoothDevice = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE)!!
            addDevice(device)
        }
    }
}

```

38. ábra A Bluetooth eszközök szkennelése és ezek elmentése

Ha a „*BluetoothDevice.ACTION\_FOUND*” esemény bekövetkezik, ami azt jelenti, hogy a Bluetooth jel adónk hatókörében érzékelhető más Bluetooth eszköz, akkor a „*device*” változóba lekérem a megtalált eszköz adatait, majd ezt hozzáadom egy listához, ahol tárolom a többi Bluetooth eszközt és azok adatait.

Ezt követően egy *for* ciklussal végig tudunk menni az összes Bluetooth eszközön. Első lépésben ellenőrzöm, hogy a megkapott Bluetooth Mac cím hozzátartozik-e valamelyik az osztályban lévő diákhoz. Ha nem tartozik egy diákhoz sem akkor vesszem a következő eszközt, ha valamelyik diákhoz hozzátartozik a Mac cím akkor a jelenléte sikeresen hozzáadódik a kiválasztott héthez.

## 5. Üzembe helyezés és kísérleti eredmények

A PWA mint technológia irányelv először 2015-ben jelent meg. A Google Chrome két mérnöke (Frances Berriman és Alex Russell) alkotta meg magát a progresszív webalkalmazás kifejezést annak az érdekében, hogy leírják azokat a webalkalmazásokat, amelyek képesek kihasználni a mai modern böngészők által támogatott összes új funkciót. A Google a terjesztésnek hála elérte, hogy 2016-ra a Firefox bevezesse a „Service Workers” támogatását a böngészőjébe. Ezt a támogatást 2018-ban az Apple Safari és a Microsoft Edge böngésző követte. 2019-re a PWA-k elérhetőek voltak a Google Chrome, Microsoft Edge, Safari és Mozilla Firefox böngészőkben. 2020 decemberében azonban a Firefox asztali böngészőjének a PWA támogatása megszűnt.

A PWA-k megalkotásánál az elsődleges szempont az volt, hogy minden böngészőben működjön, amely megfelel a jelenlegi webes szabványoknak. Ez azt jelenti, hogy a PWA-k képesek kihasználni a mai modern webböngészők összes pozitív tulajdonságát és ezeket képesek kiegészíteni pár hasznos funkcióval.

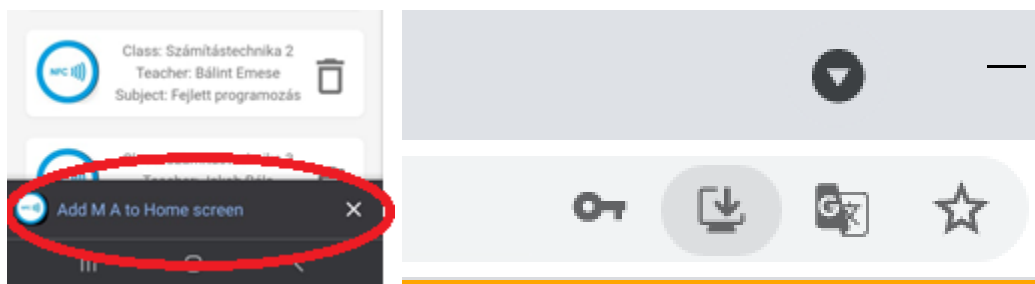
### 5.1. Üzembe helyezés

A legtöbb felhasználó ismeri az alkalmazások telepítését, valamint a telepítés nyújtotta előnyöket egy web böngészővel szemben. A telepített alkalmazások az operációs rendszer indító felületén jelennek meg. A legtöbb böngésző jelzi a felhasználó számára, hogy a PWA telepíthető, ha megfelel a szükséges kritériumoknak/ elvárásoknak valamint a böngésző támogatja a technológia használatát (10. ábra).

Szükséges kritériumok Google Chrome általi telepítéshez:

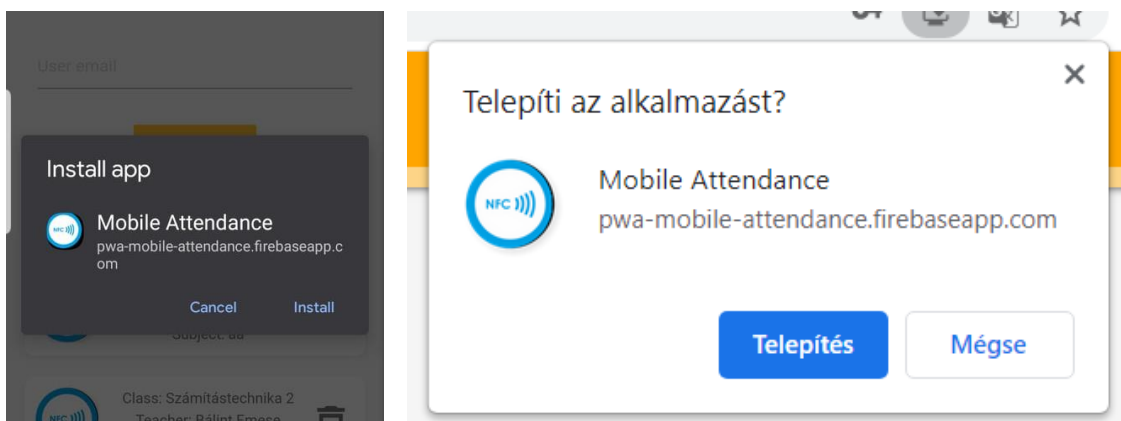
- még nem lett a webalkalmazás telepítve
- megfelel a GDPR feltételeknek
- HTTPS biztosítása
- tartalmaz egy Manifest fájlt
- tartalmaz egy Service Workers

Ha a progresszív webalkalmazás megfelel az összes Google Chrome által előírt kritériumnak a telepítéshez akkora webböngésző felajánlja a felhasználónak a telepítés lehetőségét. A 39. ábra. első felén egy mobil böngésző által biztosított telepítési lehetőség látható, míg a második felén egy asztali böngésző által biztosított telepítési lehetőség látható.



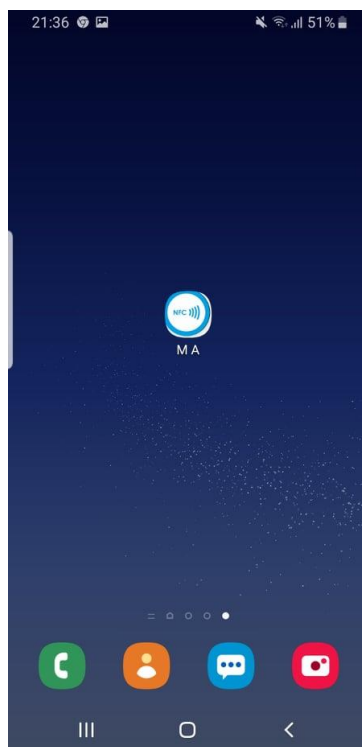
39. ábra Mobil és asztali webböngésző által felkinált telepítési lehetőség

Miután a felhasználó rányom a felajánlott lehetőségre a böngésző egy felugró ablak általi megerősítést kér a felhasználótól az alkalmazás telepítésére (40. ábra).



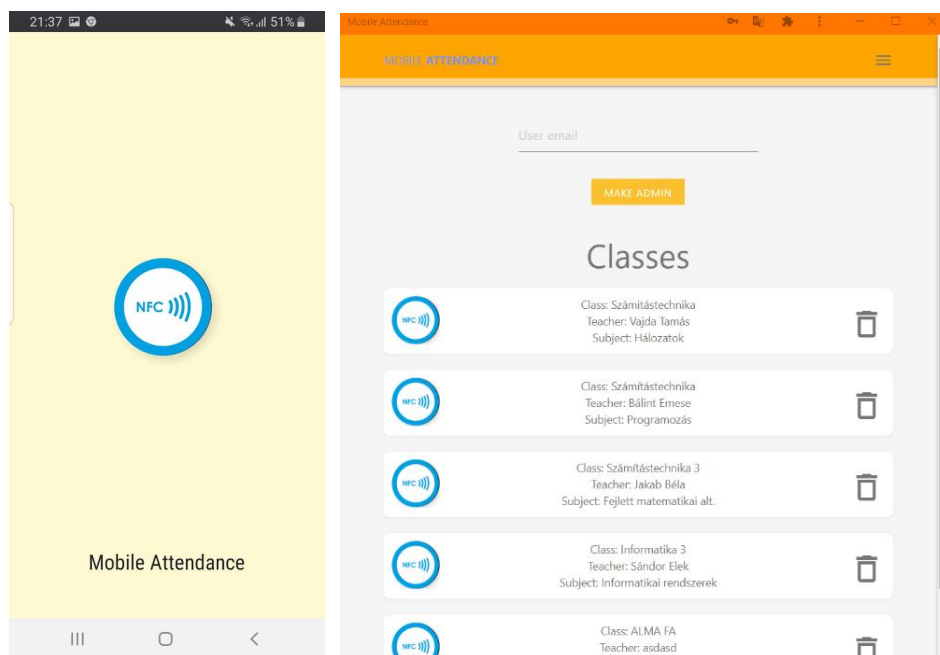
40. ábra A mobil és webböngésző általi megerősítés az alkalmazás telepítésére

Ha a felhasználó elfogadja ezt a lehetőséget a webböngésző telepíti az alkalmazást a kezdőképernyőre (41. ábra).



41. ábra A telepített alkalmazás ikonja a kezdőképernyőn

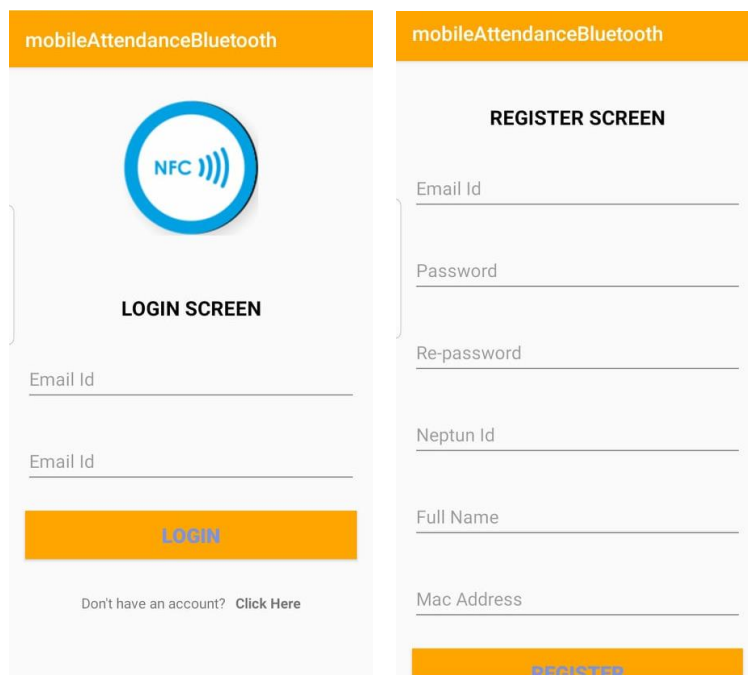
Ezután a progresszív webalkalmazás egy új böngésző ablakban fog futni a webböngésző helyett (42. ábra). Valamint mobiltelefonon kap egy betöltési képernyőt annak érdekében.



42. ábra A telepített alkalmazás

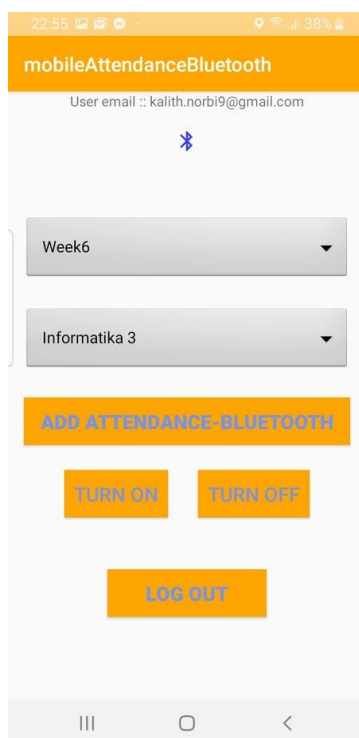
A telepítés után minden felhasználó kell azonosítsa magát belépés vagy regisztráció által. Sikeres azonosítás után a felhasználó a kezdő képernyőre kerül. Az alkalmazáson belüli a funkcionális követelmények betartása által a felhasználó képes kihasználni a progresszív webalkalmazás nyújtotta összes funkcionalitást.

Ha a felhasználók nem rendelkeznek a szükséges NFC technológiával és NFC-címkével akkor szükséges a natív alkalmazás telepítése, ami Bluetooth technológia által írja be a jelenléteket. Az alkalmazás telepítése után a felhasználó kell azonosítsa magát belépés vagy regisztráció által (43. ábra).



43. ábra Felhasználó azonosítása a natív alkalmazásban

Sikeres azonosítás után a felhasználó a kezdőképernyőre kerül, ahol a szükséges jogosultságok megadása után elindíthatja a jelenlétek hozzáadását a kiválasztott osztályhoz (44. ábra).



44. ábra Natív alkalmazás kezdőképernyője

## 5.2. Felmerült problémák és megoldásaik

A progresszív webalkalmazások jelen pillanatban nem támogatnak minden funkcionalitást, amit a natív alkalmazások támogatnak. A natív alkalmazások teljes mértékben képesek a

mobiltelefon akár összes erőforrását igénybe venni míg a progresszív webalkalmazások csak a Web API által támogatott és elérhető funkcionalitásokat képesek igénybe venni.

Ez miatt sok funkció, amit szerettem volna bevezetni vagy megvalósítani nem úgy működött, mint egy natív alkalmazás esetében. Olyan is előfordult, hogy egy funkció bevezetése nem sikerült a megfelelő támogatás hiányában.

Mivel sok cég használ NFC alapú beléptetési és kiléptetési rendszert az alkalmazottak nyilvántartására ebből adódott az ötletem, hogy szeretnék én is valami hasonló megoldást csinálni a jelenlétek nyomon követésére. A legtöbb cégnél egy NFC címke olvasó van kihelyezve és az alkalmazottnak ahhoz kell a saját kártyáját hozza érintse. De mi van akkor, ha otthon marad a kártya? Nem tudnak bemenni.

Mivel a mai modern közép és felsőkategóriás mobiltelefonok mind rendelkeznek NFC olvasóval innen jött az ötletem, hogy ezt az egészet meg lehetne valósítani csak mobiltelefonok használatával. Mindezt egy modern cross-platformos megoldás használatával. A kutatásom első lépésében megvizsgáltam, hogy a mai modern web böngészők támogatják-e az NFC használatát. Támogassak de csak NDEF formátumú NFC címkéket képesek írni és olvasni biztonsági okok miatt. Az alkalmazásom számára a Peer-to-Peer típusú NFC-s kommunikációra lett volna szükségem ahhoz, hogy egyik telefonról adatot tudjak továbbítani a másik telefonra, de ez még nem kivitelezhető a támogatás hiánya miatt.

Ennek a kiküszöbölésére próbáltam egy WiFi alapú rendszert kivitelezni, ami automatikusan érzékelt volna a hatókörben lévő WiFi eszközök jel erősségét és ennek folyamán írta volna be a jelenléteket automatikusan. De ennek is van egy hátránya, hogy a webböngésző kizárólag arról WiFi hálózatról tud adatokat megkapni, amelyekre a telefon éppen rá van csatlakozva.

Ezután a Bluetooth elérést próbáltam meg kivitelezni. De a Web API nem támogatja egyedi azonosító adatok beszerzését a Bluetooth eszközökről. Az eszköz nevét lehet csak elérni, ha rendelkezik azzal. Ennek megoldására készítettem egy natív alkalmazást, amely a Bluetooth jeladó hatósugarán belül lévő Bluetooth eszközök MAC címe alapján beírja a jelenléteket a diákoknak.

### 5.3. Kísérleti eredmények

Mivel a progresszív webalkalmazások fejlődés alatt állnak így sok funkcionalitás nem kínált elégséges megoldást számomra.

A legfontosabb megemlíteni azt, hogy webböngészőben futak ezáltal minden olyan eszközön elérhetők, ami rendelkezik egy megfelelő támogatottságú webböngészővel. Kísérleteim alatt

az alkalmazásomat kipróbáltam olyan webböngészőkön is, amelyek nem tartalmazznak PWA támogatottságot. A webalkalmazás elindul azokban a böngészőkben is, de csak egyszerű weboldalként működik és nem képes kihasználni a progresszív webalkalmazások nyújtotta előnyöket.

A következő funkciók nem érhetőek el a támogatás nélküli böngészőkben:

- Telepíthetőség (nem lehet telepíteni az alkalmazást, vagyis hozzáadni a kezdőképernyőhöz)
- Service Workers
- NFC technológia
- IndexedDB
- Bluetooth támogatás megszűnése
- Real Time Listener

Az NFC tesztelése során próbáltam megoldást keresni arra, hogy létre tudjak hozni két telefon között Peer-to-Peer adatátvitelt. A próbálkozásom nem járt sikerrel. Jelen pillanatban nem lehet még megvalósítani semmilyen eszközzel Web API használata mellett. Így az NFC-vel való adatátvitelt egyik eszközről a másik eszközre jelen pillanatban csak NDEF címkék segítségével lehet megvalósítani.

A WiFi elérés is nagyon korlátozott. A Network API csak információk szolgáltat a jelenlegi hálózatról. Az API fő feladata az, hogy lehetővé teszi a webalkalmazások számára azt, hogy a hálózat minősége alapján adaptáljuk a különböző funkcionalitásokat.

A közölt információk a következők:

- Állapot: online/offline
- Hálózat típusa: wifi, mobil, Ethernet, 3G, 4G
- Letöltési sebesség: ez a hatékony sávszélesség mbps-ben
- RTT: rendszer válaszideje

A webes Bluetooth API lehetővé teszi az alkalmazás számára, hogy Bluetooth Low Energy (BLE) technológiát használó eszközt csatlakoztassunk a webalkalmazásunkhoz. Lehetővé teszi értékek leolvasására vagy írására az eszköz és a webalkalmazás között. Viszont nem képes az eszközről adatokat gyűjteni, ami alapján képes lehetne a felhasználók azonosítására.

A progresszív webalkalmazás tesztelését a Google Chrome által biztosított *Lighthouse* szolgáltatással végeztem. A *Lighthouse* tesztelés során minden kritériumot ellenőriz, amelynek egy progresszív webalkalmazás eleget kell tennie.

A tesztelés menete:

- A tesztelni kívánt weboldalon F12 gomb megnyomása
- Az megjelenő menüben ki kell választani a *Lighthouse* fület
- Ki kell választani, hogy a weboldalt milyen kritériumok szerint akarjuk tesztelni
- Teszt futtatása – *Generate report* gombra kattintva
- A kész teszt kiértékelése

A teszt lefuttatása után megjelenik számunkra minden kritérium és a kritériumoknak a kiértékelése. A sikeres teszteket egy zöld pipával jelzi míg a sikerteleneket egy piros felkiáltójellel. Ha az alkalmazásunk megfelel az összes kritériumnak akkor a 45. ábra látható kép fogja nekünk ezt mutatni.



45. ábra Progresszív webalkalmazás tesztelésének kiértékelése

A teszt két fő dolgot vizsgál (46. ábra). A teszt kiértékelését a saját progresszív webalkalmazásom végeztem, ami megfelelt az összes kritériumnak:

- Telepíthető
  - A Manifest fájl és a Service Workers megfelel-e a telepítési elvárásoknak
- PWA optimalizált-e
  - Regisztrál egy Service Workerst, ami vezérli az oldalt és a start\_url-t
  - Átirányítja a HTTP forgalmat HTTPS-re
  - Egyedi splash képernyőhöz való konfigurálás
  - A téma színét beállítja a címsorban
  - A tartalom megfelelően igazodik a nézet ablakhoz
  - Van egy „<meta name = \"viewport\">” címke szélességgel vagy kezdeti léptékkal
  - Tartalmaz érvényes „apple-touch” ikont
  - A manifest fájl rendelkezik maszkolható ikonnal
- További elemek kézi ellenőrzése. Ezeket az ellenőrzéseket az alapszintű PWA ellenőrzőlista írja elő, de a Lighthouse nem ellenőrzi automatikusan.
  - A webhely működik több böngészőben



- Az oldal átmenetek nem blokkolják a hálózatot
- Minden oldal rendelkezik URL-címmel

A kézzel előírt elemek sikeres tesztelése után kijelenthetem, hogy a progresszív webalkalmazásom eleget tesz az összes PWA által előírt kritériumoknak.

+

Installable

●

Web app manifest and service worker meet the installability requirements

★

PWA Optimized

●

Registers a service worker that controls page and `start_url`

●

Redirects HTTP traffic to HTTPS

●

Configured for a custom splash screen

●

Sets a theme color for the address bar.

●

Content is sized correctly for the viewport

●

Has a `<meta name="viewport">` tag with `width` or `initial-scale`

●

Provides a valid `apple-touch-icon`

●

Manifest has a maskable icon

**Additional items to manually check (3)** — These checks are required by the baseline [PWA Checklist](#) but are not automatically checked by Lighthouse. They do not affect your score but it's important that you verify them manually.

46. ábra Lighthouse teszt által vizsgált kritériumok

## 6. Következtetések

A szakirodalmak tanulmányozása után rájöttem, hogy egy ilyen alkalmazás létrehozásához a legcélszerűbb a PWA technológiát használni. Azért döntöttem a PWA technológia mellett mert rendelkezik minden számomra szükséges elvárással amire egy jelenlét kezelő alkalmazásnak szüksége van. Összességében elmondható, hogy egy jó jelenlét követő alkalmazás a következő kritériumoknak felel meg:

1. Azonosítás: szükségünk van egy egyedi megbízható azonosítóra, ami minden diák esetében eltérő (a legjobb erre a Neptun- kód azonosító) és nehezen kijátszható legyen.
2. Adattárolás: valós időben tudjuk elérni az adatbázist, mert a rendszer alapját az adatok tárolása képezi.
3. Gyorsaság: a rendszer fő funkcionalitását az képezi, hogy képes legyen felismerni és valós időben rögzíteni a diák jelenlétét.
4. Eszköz: az igénybe vett eszköz esetünkben az okos mobiltelefon, legyen elérhető minden hallgató számára. Ha olyan megoldást választunk, amihez szükséges egy lokálisan előre telepített eszköz, azt minél költséghatékonyabban tudjuk előállítani.
5. Titkosítás: fontos a titkosított kommunikáció, mivel az alkalmazásunk meg kell feleljen a GDPR előírásainak a személyes adatok tárolása és védelme érdekében.

Azért választottam a PWA-t mert nem szükséges túl sok erőforrás felhasználása, ezért nem lesz lassú vagy ki van zárva az akadozás lehetősége az telepített applikáción belül, mindez mellett a tökéletes dizájn és felhasználói élmény megvalósítása is könnyebb, hiszen nem kell két natív applikációt létrehozni, hanem csak egyet. A karbantartás is fontos dolog, két natív applikáció kétszer annyi karbantartást igényel, mint egy PWA. Ezáltal elérhető az, hogy a rendszerünk gyors és valós idejű legyen úgy, hogy biztonságérzetet is nyújt a felhasználónak. Ezeknek a kritériumoknak a betartása, valamint a szükséges alap követelményeknek megfelelően létre tudtam hozni egy telefon alapú jelenlét kezelő rendszert. Bármilyen alkalmazás vagy rendszer megalkotásakor meg kell találjuk a megfelelő egyensúlyt a biztonság és használhatóság között, annak érdekében, hogy egy hatékony és működőképes alkalmazást tudjunk létrehozni, amely egyaránt növeli a felhasználói élményt is.

## 6.1. Összefoglaló

A projekt célkitűzéseinek nagyobb akadályait leküzdve sikerült egy olyan progresszív webalkalmazás létrehozása, ami nagy segítség lehet a tanároknak az órákon való részvétel nyomon követéséhez, mindezt egy megbízható, hatékony, felhasználóbarát és könnyen kezelhető környezetben. A felmerült technikai korlátozások miatt létrehoztam egy natív alkalmazást is annak érdekében, hogy bárki képes legyen használni az általam megalkotott alkalmazást.

A progresszív webalkalmazás használatával létrehoztam egy olyan szoftvert, amely:

- Telepíthető
- Offline is működőképes
- Multiplatformos

A progresszív webalkalmazásban megvalósított funkcionálisok:

- Regisztráció lehetősége
- Belépés lehetősége, annak érdekében, hogy a felhasználó különböző eszközein is képes legyen elérni a saját adatait.
- Új osztályterem létrehozása
- Diák hozzáadása a jelenléti listához kézi bevitellel
- Diák hozzáadása a jelenléti listához automatikusan NFC használatával
- Diák eltávolítása a jelenléti listáról
- Saját osztályok kilistázása a kezdőképernyőre
- Saját osztályok törlése
- Osztály adatok megtekintése
- Osztályhoz tartozó diákok jelenléteinek megtekintése
- Jelenlét hozzáadása a kiválasztott diáknak kézi bevitellel
- Jelenlét hozzáadása automatikusan a kiválasztott héthez
- E-mail küldése a diákoknak
- Saját profil adatok megjelenítése
- Belépés különböző osztálytermekbe
- Jelenlét beírása
- Felvett osztályok kilistázása
- A felvett osztályokhoz tartozó jelenlétek kilistázása

A natív alkalmazásban megvalósított funkcionálisok:

- Regisztráció lehetősége

- Belépés lehetősége
- Bluetooth technológia használatához szükséges engedélyek megadása az alkalmazásból
- Diák hozzáadása a jelenléti listához automatikusan Bluetooth használatával

Fontos megemlíteni, hogy érdemes kihasználni a progresszív webalkalmazások által nyújtott lehetőségeket ezáltal nem csak egy platformfüggetlen alkalmazást kapunk, hanem az anyagi kiadásokat is csökkentjük.

Próbáltam az egyszerűsége és könnyen kezelhető felhasználói felület létrehozására koncentrálni, ugyanis a szoftver fő feladata a jelenlétek kezelése és az egész procedura könnyebbé, valamint gyorsabbá tétele.

Összegezve sikerült egy olyan működőképes progresszív webalkalmazás megalkotása, amely megfelel a mai korszerű jelenlét kezelő alkalmazások elvárásainak, ezáltal megkönnyíti a felhasználók életét.

## 6.2. Továbbfejlesztési lehetőségek

A továbbfejlesztési lehetőségek között a legfontosabb az, hogy megalkossanak egy NFC Peer-to-Peer általi adatátvitelt. Ennek az implementálása csak azután valósítható meg, miután a Web API NFC támogatása kibővül ezzel a funkcionalitással.

Továbbá, ha a Web Bluetooth API lehetővé teszi a közeli eszközök „felfedezésének” lehetőségét, akkor a rendszer teljes mértékű automatizálása is további kérdéseket vet fel.

A jelenlegi rendszerben az Admin szerepkör nagyon kevés plusz lehetőséget biztosít, éppen ezért az Admin szerepkörébe tartozó lehetőségeket tovább lehetne bővíteni. Az oktató számára biztosítani egy olyan lehetőséget, hogy képes legyen egy osztályhoz tartozó jelenlétek exportálására.

A diákok számára biztosítani egy olyan lehetőséget, hogy az alkalmazás az óra kezdete előtt értesítést küldjön a következő óráról.

A natív alkalmazás használatának kiküszöbölése, az egyszerűbb használat érdekében.

A felhasználói felület még jobb optimalizálásának biztosítása mobilon és webes felületen egyaránt, valamint törekedni a még egyszerűbb használatot biztosító felhasználói felület létrehozására.

## Hivatkozások

- [1] H. Misley és S. Tóth-Mózer, „Digitális eszközök integrálása az oktatásba,” in *Jó gyakorlatokkal, tantárgyi példákkal, modern eszközlistával*, Budapest, Pest: Eötvös Loránd Tudományegyetem, 2019, pp. 117-145.
- [2] Z. Szűts, „A digitális pedagógia egységes elméleti kerete és alkalmazása a tanítás és tanulás folyamatában.,” Eszterházy Károly Egyetem, Eger, 2020.
- [3] H. Edwin, „Coderweb,” 2020.02.06 02 2020. [Online]. Available: <https://www.coderweb.hu/blog/reszponziv-weboldal-vagy-mobil-applikacio>.
- [4] D. Rosul, „What are the popular types and categories of apps,” 5 December 2016. [Online]. Available: <https://thinkmobiles.com/blog/popular-types-of-apps/>. [Hozzáférés dátuma: 17 08 2020].
- [5] A. Eneida, S. Hégen és S. László, „Mobilalkalmazások elfogadását befolyásoló tényezők romániai középiskolások és egyetemi hallgatók körében,” in *Forum on Economics & Business / Közgazdász Fórum*, Kolozsvár, UBBCLUJ, 2016, pp. 54-80.
- [6] S. Gábor, „Weblap, weboldal, honlap - webes alapfogalmak tisztázása,” 10 március 2019. [Online]. Available: <https://webfaktor.hu/weblap-weboldal-honlap-webes-alapfogalmak-tisztazasa>. [Hozzáférés dátuma: 19 08 2020].
- [7] L. Tomas, „Website Setup - A Detailed Guide to Different Types of Web Hosting,” 26 08 2019. [Online]. Available: <https://websitesetup.org/different-types-of-web-hosting/>. [Hozzáférés dátuma: 15 08 2020].
- [8] K. Edit, „Seomax - Domain név jelentése, mire való a domain?,” 01 november 2017. [Online]. Available: <https://seomax.hu/domain-nev-jelentese/>. [Hozzáférés dátuma: 25 08 2020].
- [9] T. István, „Reszponzív weboldal készítés legfontosabb tudnivalók,” 4 július 2020. [Online]. Available: <https://www.usernet.hu/blog/reszponziv-weboldal-keszites-legfontosabb-tudnivalok>. [Hozzáférés dátuma: 14 08 2020].
- [10] B. Kim, „Responsive Web Design, Discoverability, and Mobile Challenge,” *ALA TechSource*, %1. kötet4, %1. szám6, pp. 29-39, 2013.
- [11] M. Maxson, „Is a Mobile Website or a Mobile App Best for Your Business?,” 7 február 2018. [Online]. Available: <https://themanifest.com/mobile-apps/mobile-website-or-mobile-app-best-your-business>. [Hozzáférés dátuma: 26 08 2020].

- [12] N. Krisztián, „Mobil alkalmazás: miért lehet szükséged rá? Mire figyelj a kialakításánál?,” 19 május 2020. [Online]. Available: <http://blog.webshark.hu/2020/05/19/mobilalkalmazas/>. [Hozzáférés dátuma: 13 08 2020].
- [13] M. McKenzie, „Is a Mobile Website or a Mobile App Best for Your Business?,” 7 február 2018. [Online]. Available: <https://themanifest.com/mobile-apps/mobile-website-or-mobile-app-best-your-business>. [Hozzáférés dátuma: 10 08 2020].
- [14] M. Ivano , P. Giuseppe, N. Paul és V. Petar, „Assessing the Impact of Service Workers on the Energy Efficiency of Progressive Web Apps,” *Mobilesoft*, %1. kötet3, %1. szám15, pp. 35-45, 2017.
- [15] R. Sam és P. LePage, „What are Progressive Web Apps?,” 6 január 2020. [Online]. Available: <https://web.dev/what-are-pwas/>. [Hozzáférés dátuma: 16 08 2020].
- [16] A. Tal, „Chapter 1. Introducing Progressive,” in *Building Progressive Web Apps*, United States of America, O'Reilly Media, Inc., 2017, pp. 16-29.
- [17] A. Oluwatofunmi, A. Chigozirim , O. Nzechukwu , Olawale és J. O. Olawale , „Dawning of Progressive Web Applications (PWA): Edging Out the Pitfalls of Traditional Mobile Development,” *American Scientific Research Journal for Engineering, Technology, and Sciences* , %1. kötet68, %1. szám1, pp. 85-99, 2020.
- [18] S. N. Deepu, „4 important points to know about Progressive Web Apps (PWA),” 24 március 2017. [Online]. Available: <https://medium.com/@deepusnath/4-points-to-keep-in-mind-before-introducing-progressive-web-apps-pwa-to-your-team-8dc66bcf6011>. [Hozzáférés dátuma: 21 10 2020].
- [19] „Attendance Traker,” [Online]. Available: <https://play.google.com/store/apps/details?id=com.ferid.app.classroom&hl=ro>.
- [20] P. Baghwat, „Bluetooth: technology for short-range wireless apps,” *IEEE Internet Computing*, %1. kötet5, %1. szám3, pp. 96-103, június 2001.
- [21] W. Welbes, „What is Bluetooth Low Energy (BLE) and how does it work?,” 7 március 2019. [Online]. Available: <https://www.centare.com/insights/what-is-bluetooth-low-energy>. [Hozzáférés dátuma: 16 08 2020].
- [22] N. Gupta, „Introduction - Background of Bluetooth,” in *Inside Bluetooth Low Energy*, Boston, Artech House, 2016, pp. 1-34.
- [23] „Blicker,” The Interactive Studio, [Online]. Available:

- <http://www.theinteractivestudio.com/blicker/>.
- [24] B. Team, „Blicker for education,” The Interactive Studio, [Online]. Available: <http://www.theinteractivestudio.com/blicker/index.html>. [Hozzáférés dátuma: 29 08 2020].
- [25] Y. Chiu, „Attendance System Using Raspberry Pi and NFC Tag Reader,” 16 02 2014. [Online]. Available: <https://www.instructables.com/Attendance-system-using-Raspberry-Pi-and-NFC-Tag-r/>. [Hozzáférés dátuma: 24 08 2020].
- [26] M. Team, „MyAttendanceTracker,” MyAttendanceTracker, 2016. [Online]. Available: <https://www.myattendancetracker.com/>. [Hozzáférés dátuma: 25 szeptember 2020].
- [27] Olgor, [Online]. Available: <https://play.google.com/store/apps/details?id=abdelrahman.wifianalyzerpremium&hl=h u&gl=US>.
- [28] K. Chunnu és S. Pritam, „Application of Firebase in Android App Development-A Study,” *International Journal of Computer Applications*, %1. kötet179, %1. szám46, pp. 49-53, június 2018.
- [29] „Firebase,” [Online]. Available: <https://firebase.google.com/docs/hosting>. [Hozzáférés dátuma: 10 12 2020].
- [30] „Firebase Console,” Google, [Online]. Available: <https://console.firebase.google.com/u/0/project/projectName/overview>. [Hozzáférés dátuma: 14 12 2020].
- [31] „Firebase - Firestore,” Google, [Online]. Available: <https://firebase.google.com/docs/firestore>. [Hozzáférés dátuma: 10 12 2020].
- [32] „Firebase Authentication,” [Online]. Available: <https://firebase.google.com/docs/auth>. [Hozzáférés dátuma: 16 12 2020].
- [33] „Materilaize,” Materialize, 12 03 2020. [Online]. Available: <https://materializecss.com/>. [Hozzáférés dátuma: 15 09 2020].
- [34] L. Pete, S. Thomas és B. François, „Add a web app manifest,” 05 november 2018. [Online]. Available: <https://web.dev/add-manifest/>. [Hozzáférés dátuma: 15 01 2021].
- [35] S. Thomas és L. Pete, „Create an offline fallback page,” 24 szeptember 2020. [Online]. Available: <https://web.dev/offline-fallback-page/>. [Hozzáférés dátuma: 26 november 2020].

- [36] S. Maximilian, „Progressive Web Apps (PWA) - The Complete Guide,” 6 augusztus 2020. [Online]. Available: <https://www.udemy.com/course/progressive-web-app-pwa-the-complete-guide/>. [Hozzáférés dátuma: 16 11 2020].
- [37] B. François , „Interact with NFC devices on Chrome for Android,” 12 02 2020. [Online]. Available: <https://web.dev/nfc/>. [Hozzáférés dátuma: 16 03 2021].
- [38] B. François , „Web Dev,” 12 02 2020. [Online]. Available: <https://web.dev/nfc/>. [Hozzáférés dátuma: 16 03 2021].
- [39] „Android for Developers,” 11 03 2021. [Online]. Available: <https://developer.android.com/about/versions/marshmallow/android-6.0-changes#behavior-hardware-id>. [Hozzáférés dátuma: 15 04 2021].
- [40] M. Penny és K. Mustafa, „Patterns for promoting PWA installation,” 4 június 2019. [Online]. Available: <https://web.dev/promote-install/>. [Hozzáférés dátuma: 13 09 2020].

## 9. Függelék



UNIVERSITATEA SAPIENTIA DIN CLUJ-NAPOCA  
FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE, TÎRGU-MUREȘ  
SPECIALIZAREA CALCULATOARE

Vizat decan  
Conf. dr. ing. Domokos József

Vizat director departament  
ș.l. ing. dr. Szabó László Zsolt