A szoftvertechnológia alapjai

Mobil alkalmazásfejlesztés alapjai

Beadandó dolgozat

Ekart Csaba [ZWPMKP]

2020. május 30.

Tartalomjegyzék

1. Bevezetés **3**

2. Az okostelefonok rövid története **4**

3. Android **7**

3.1. A fejlesztés nehézségei 8

3.2. Legfontosabb alkalmazás komponensekről röviden 9

3.2.1. Activity 9

3.2.2. Service 11

3.2.3. Content providers 11

3.2.4. Broadcast reciever 12

3.3. Package-ek 12

4. iOS **13**

5. Cross platform **15**

5.1. Progressive Web App 15

5.2. Xamarin 15

5.3. React Native 16

5.4. Flutter 16

6. A fejlesztés után **17**

7. Összefoglaló **18**

1. Bevezetés

Az utóbbi években az okostelefon piac robbanásszerű fejlődésen esett át. A felhasználók száma 2020-ra elérte a 3.5 milliárdot [1], a legsikeresebb gyártók pedig a világ legértékesebb cégei közé tartoznak: 2018-ban az mobil eszközök piacát meghatározó Apple Inc. történelmi rekordot ért el - az első vállalat vált, amely elérté az 1 billió dolláros piac kapitalizációs szintet. [2] Habár az utóbbi években lelassult a piac növekedése, így is átlagosan körülbelül 1.5 milliárd eszközt szállítanak le évente a legnagyobb gyártók összesítve. [3] [4]

Azonban nem csak a hihetetlen mennyiségű készülék miatt érdekes a témakör. Egy 2019-es kutatás szerint az átlag okostelefon felhasználó több mint 3 órát tölt naponta telefonja kijelzője előtt. [5] Ennek oka, hogy a kezdeti évekkel ellentétben - már nem csak telefonálásra, hanem számos egyéb tevékenységre használjuk őket, így a hagyományos média fogyasztásnál kezdce (zenehallgatás, filmek, játékok, stb.), a munka ügyek intézésén keresztül (email, naptár stb.), a közösségi életünk is nagy részét is ezeken bonyolítjuk le.

Ezen folyamatokat az eszközeinken található alkalmazások segítségével végezzük. Dolgozatom célja ezen applikációk működésének, a mobilalkalmazás fejlesztésének lehetőségeinek és különböző módszereinek rövid bemutatása.

2. Az okostelefonok rövid története

Jogosan merülhet fel a kérdés, hogy pontosan mi különbözteti meg a hagyományos mobiltelefonokat az okostelefonoktól. Általánosságban elmondhatjuk, az alapvető funkciókon túl ahhoz, hogy egy eszközt okosnak tekintsünk rendelkeznie kell egy kiterjedt és fejlett operációs rendszerrel (ezek a következő fejezetben részletesebb bemutatásra kerülnek), multimédiás képességekkel, internet eléréssel, és különböző szenzorokkal (pl. GPS, barométer, giroszkóp stb.). [6]

Az első olyan eszköz amely részben megfelel ezen kritériumoknak, és így az első okostelefonnak tekinthető, az 1994-ben bemutatott IBM Simon volt. [7] A Simon végül nem váltotta be a hozzáfűzött reményeket, főként magas ára, rossz üzemideje és nagy mérete miatt piaci bukás lett belőle, azonban az első mobil eszköz volt, amely modern LCD érintőképernyővel volt felszerelve és képes volt emailek és FAX üzenetek kezelésére. [6]

1. ábra. Kép az IBM Simonról [8]

Ezt követően az 1990-es években elkezdtek terjedni az ún. personal digital assistantek (továbbiakban PDA-k), melyek internet elérésre alkalmas mobileszközök voltak, jellemzően érintőkijelzővel és hozzátartozó érintőceruzával vagy műanyag QWERTY billentyűzettel rendelkeztek. Legfontosabb újításuk azonban a fejlett operációs rendszerek megjelenése (Palm

OS, Newton OS, Symbian, Windows), melyek utódjaként tekinthetünk a ma is népszerű Android és iOS rendszerekre. [6] A PDA piac habár réteg termékként került a köztudatba, folyamatosan növekedő piacot ért el, melyet végül a modern okos telefonok elterjedése állított meg. [9]

(a) HP iPAQ hw6515 Mobile Messenger [10]

(b) Apple Newton [11]

2. ábra. Két példa az 1990-es években hódító PDA eszközökről

Ekkoriban azokat az eszközöket tekintették okostelefonnak, melyek rendelkeztek egy PDA képességeivel, de hagyományos mobiltelefonként is használhatóak voltak.

Az igazi áttörést a piacon a 2007-ben Steve Jobs által bemutatott Apple iPhone hozta el, mely ezzel meg is teremtette a modern okostelefon piacot, és gyökeresen megváltoztatta az emberek hozzáállását a telefonokhoz. [12] Az iPhone legfontosabb újításai az egyetlen érintőképernyőn alapuló kezelő felület, a fizikai billentyűzet eltüntetése, az érintőpálca használatának elvetése, a multitouch technológia bemutatása, és egy új, fejlett Darwin alapú, kifejezetten ujjak általi használatra optimalizált operációs rendszer - az iOS bevezetése volt. [6] [13]

3. ábra. Az eredeti iPhone [14]

Az iPhone piaci sikerének következtében az okostelefon gyártók lassan átálltak az iOS-hez hasonló felületek fejlesztésére, és kezdetét vette a modern mobil operációs rendszerek piaci versenye.

A gyártók lassú reakciója az iPhone sikerére azt eredményezte, hogy a legtöbb akkoriban népszerű operációs rendszer mára teljesen eltűnt. Ezek közé tartozott a Windows Mobile platform, a Nokia Symbianja, de a Palm OS is. [6] A változásokra gyorsan reagáló Google azonban sikerrel alakította át az Android platformot, s gyorsan hatalmas népszerűségre tett szert. [15] Az elmúlt évtized próbálkozásai ellenére nem sikerült olyan operációs rendszert bemutatni, mely sikeresen elterjedt volna, sorban buktak el olyan cégek próbálkozásai, mint a Samsung, a Canonical, a Mozilla, a Nokia vagy a Microsoft. [16]

Manapság az okostelefonok piacán duopóliumról beszélhetünk, melyet az Apple iOS-e és a Google Android rendszere dominál. [17]

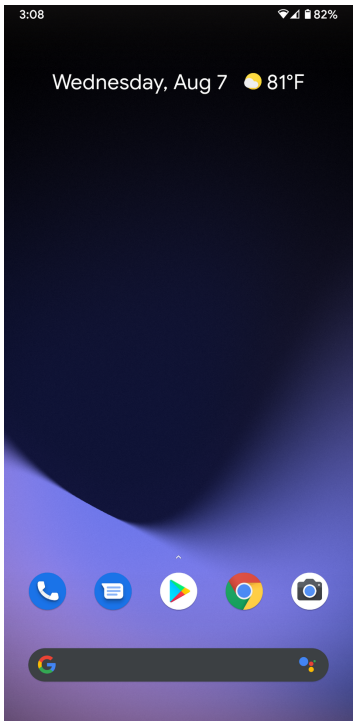
A továbbiakban ezen két platform fejlesztési lehetőségeit és egyszerű felépítését mutatom be.

3. Android

Az Android egy 2008-ban bemutatott kifejezetten mobil eszközökre készített, nyílt forráskodú Linux alapú operációs rendszer, melyet a Google vezetésével a mobiltelefon gyártókból és egyéb telekommunikációs és szoftverfejlesztő cégekből álló Open Handset Allience tagjai fejlesztenek. Legfrissebb verziója a 2019 szeptemberében bemutatott Android 10, mely a rendszer 17. fő kiadása. [15]

A natív Android alkalmazások fejlesztéséhez két hivatalosan támogatott programozási nyelvet használhatunk - a Javat, illetve az azzal teljes mértékben kompatibilis, szintén Java Virtual Machine-ban futó, újonnan bemutatott Kotlint. A Kotlin célja, hogy az Androidra történő fejlesztés gyorsabb és egyszerűbb legyen, illetve a Google és az Oracle közötti elmérgesedett viszony könnyítése, és egyéb költségek csökkentése. [18] [19]

Az Androidra írt alkalmazások (Lollipoptól kezdve) a Javaból / Kotlinból történő byte kóddá alakítás után egy a Google által módosított, kifejezetten mobil eszközökre optimalizált Android Runtime nevű virtuális gépen futnak. [20]



4. ábra. Képernyőkép az Android 10 alapértelmezett kezdőképernyőjéről [21]

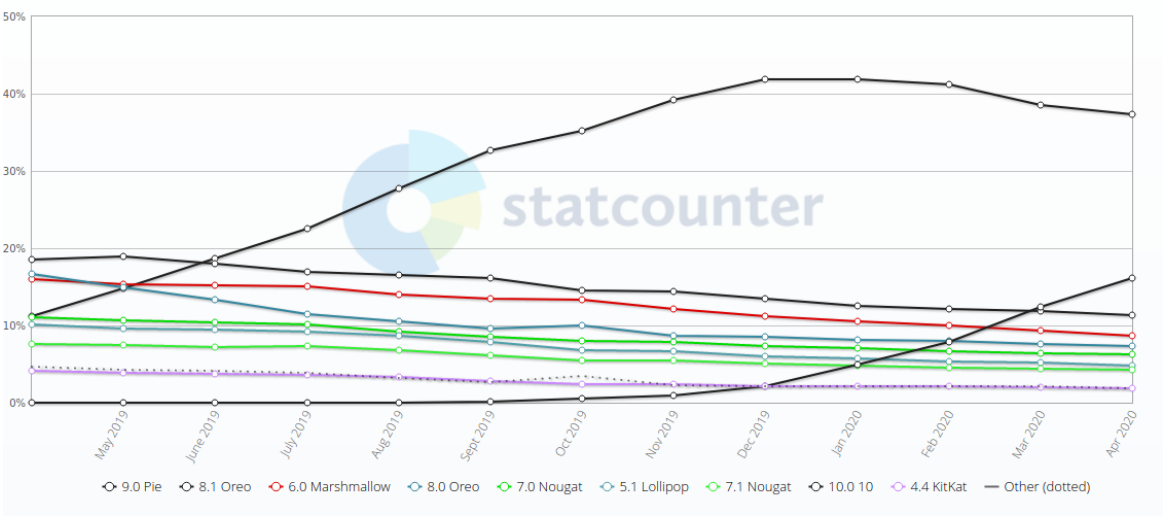
Az Androidra való fejlesztés a teljesen multiplatform Android Studio IDE-ben történik.

Teszteléshez használhatunk bármilyen Android eszközt kábellel csatlakoztatva, vagy emulátort használva konkrét készülékek pontos másolatát tudjuk futtatni a számítógépünkön. [18]

3.1. A fejlesztés nehézségei

A rendszerre való fejlesztésnél a legnagyobb nehézség, hogy számos különböző eszközt kell támogatnunk, melyek különböző képernyőmérettel, felbontással, hardware-es adottságokkal, és különböző szoftververziókkal rendelkeznek. Már 5 évvel ezelőtt is több mint 24 ezer különböző Androidot futtató típust becsültek, és ez a szám azóta is csak növekedett. [22] Fejlesztőként természetesen nincs lehetőségünk minden eszközre tesztelni az alkalmazásunkat így bármikor előfordulhat, hogy eszköz specifikus hibák lépnek fel.

További nehézségnek számít a különböző Android verziók lassú adaptálása, mely így egy rendkívül szegmentált piacot eredményez. A legfrissebb Android verzió jellemzően az eszközök csupán kis részén található, mivel a gyártók viszonylag rövid ideig támogatják a készülékeket. Ez azt jelenti, hogy a rendszer legfrissebb újdonságai és lehetőségei nem használhatóak a legtöbb céleszközön. Ennek megoldására készült az Android Support Library, mely lehetővé teszi, hogy régebbi verziókon is elérhetőek legyen bizonyos ilyen módszerek, és biztosítja, hogy az applikációnk az ezt futtató telefonokon is megfelelően fog működni. Éppen ezért a fejlesztés elkezdésekor megszabunk egy minimum SDK és egy target SDK verziót. A minimum SDK verzió fogja jelenteni a legkisebb API verziót, melyen az alkalmazásunk futni fog. Ezt célszerű úgy választani, hogy körülbelül a készülékek 90%-a támogatott legyen. A target SDK, pedig azt az iterációt jelöli, amin az alkalmazásunk számára biztosított feltételek a legoptimálisabbak. Ezt célszerű a legfrissebbre állítani. [23]



5. ábra. Az Android verziók feloszlása a Statcounter adatai alapján 2019 és 2020 áprilsa között. [24]

3.2. Legfontosabb alkalmazás komponensekről röviden

Röviden bemutatnám az előadáson is említett legfontosabb Android alkalmazás komponenseket: az Activityt, a Fragmentet, a Servicet, a ContentProvidert és a Broadcast Recievert.

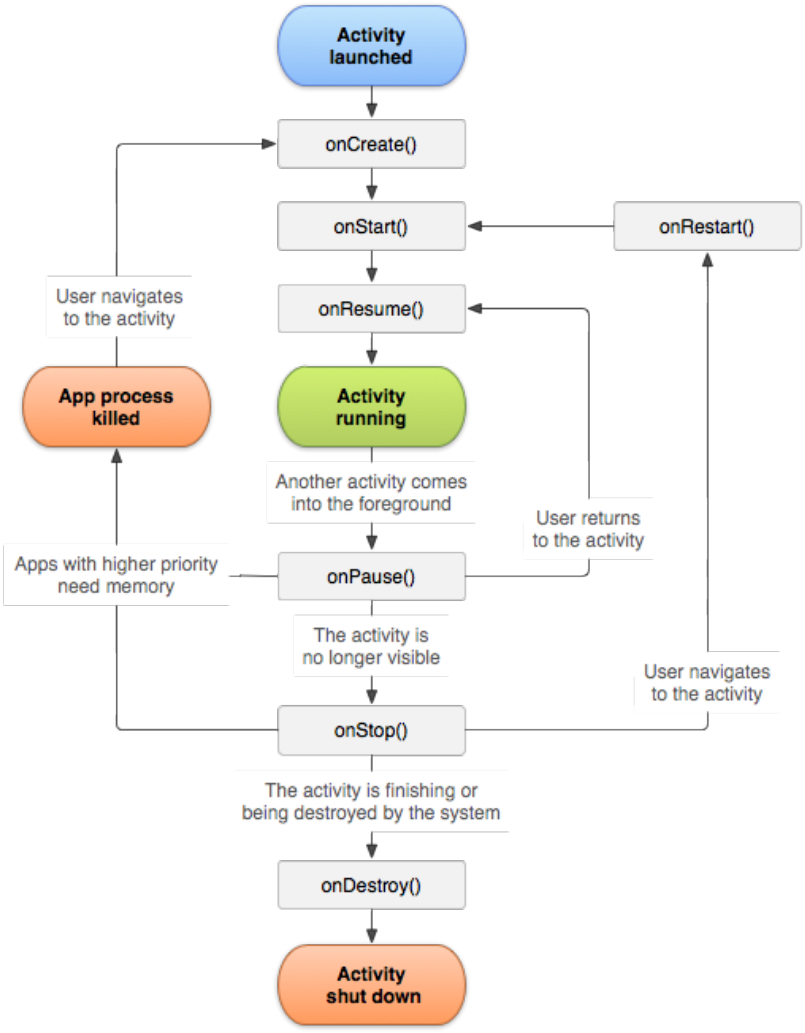
3.2.1. Activity

A mobil alkalamzások használata és az asztali programok használata közötti egyik alapvető különbség, hogy a mobilos alkalmazásoknál nem minden indításnál ugyanazt a felületet kívánjuk látni. Számtalanszor előfordul velünk, hogy egy értesítésre rányomva például konkrétan az ehhez tartozó felülettel találkozunk, nem pedig ugyanazzal, amivel az alkalmazás alapvető indításakor. Tehát az alkalmazásunknak ki kell elégítenie egy olyan paradigmát, hogy nem minden esetben ugyanaz a belépési pontja. Ennek megvalósítására hozták létre az Activityket. [25]

Az Android hivatalos dokumentációja szerint egy activity egyetlen fókuszban lévő dolog, amit a felhasználó csinálhat. [26] Egyszerűen, köznapi értelemben Activitynek hívjuk azokat az "ablakokat", amelyekkel a felhasználó többnyire kommunikál. Egy alkalmazás általában több Activityből épül fel. Egy Activity általában kitölti a teljes kijelzőt, de előfordulhat, hogy kisebb "lebegő" ablakról van szó.

Definiálnunk kell egy MainActivityt, amely az alapértelmezetten induló activity lesz az

alkalmazás megnyitása után. Különböző activityket az Intent osztály példányaival tudunk megnyitni. [25]



6. ábra. Android Activity Lifecycle [27]

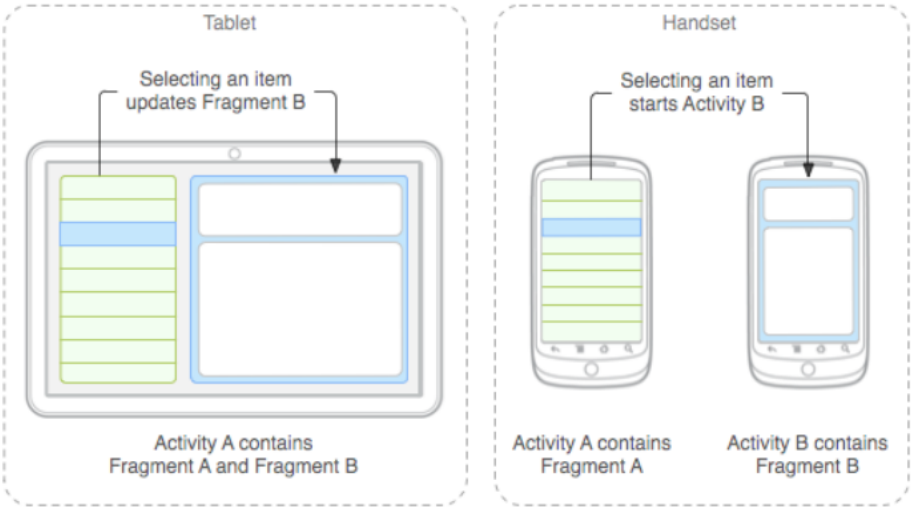
Egy Activity-nek több állapota lehet, amiket fejlesztőként célszerű külön kezelnünk egymástól. A legegyértelműbb eset amikor az Activity közvetlenül a felhasználó előtt van, azonban bármikor közbejöhet például egy hívás, vagy a felhasználó átválthat egy másik applikációra. Ilyenkor kiemelt figyelmet kell fordítanunk arra, hogy az alkalmazás ne omoljon össze ennek következtében, "ne felejtse el hol tartott", azaz őrizze meg a felhasználó által ott hagyott állapotot, illetve ne használjon több erőforrást a feltétlenül szükségesnél. Ezt kezelhetjük az Android rendszer által felhívott metódusok felülírásával.

A stagek és az ezek között felhívódó metódusok ábráját mutatja be az Android lifecycle,

melyben láthatjuk, hogy pontosan milyen sorrendben, mikor történnek ezek. [27]

3.2.1.1 Fragment

A Fragmentek célja, hogy a felhasználó felületnek csupán egy részét definiálják, ne pedig az egészét, mint az Activityk esetében. Erre azért mutatkozott szükség, hogy a nagyobb képernyőkön - például tabletek esetén - lehetőség legyen a UI bizonyos részeinek különböző elrendezésére. Míg a telefonok kisebb kijelzőjén lehet különálló activitykként gondolunk bizonyos részekre, ezeket megjeleníthetjük például egymás mellett táblagépek esetén. A Fragmantek tulajdonképpen az Activityk moduláris részei, és az életciklusok együtt kezelődik az őket tartalmazó Activityvel. [28]



7. ábra. Példa a Fragmentek felhasználására [28]

3.2.2. Service

A Service osztály célja, hogy lehetőséget adjon a fejlesztőknek a hosszú időn keresztül háttérben futó programkódok futtatására. Ennek célja, hogy amennyiben a felhasználó elhagyja az alkalmazásunkat tudjunk bizonyos szolgáltatásokat biztosítani. Ilyen lehet például a zenék lejátszása, vagy hosszabb hálózati szolgáltatások ellátása. [29]

3.2.3. Content providers

A Content providerek célja, hogy lehetővé tegye az alkalmazások által tárolt bizonyos adatok egymással való biztonságos megosztását. Akkor célszerű implementálni, ha szeretnénk más

alakalmazásoknak lehetőséget teremteni, hogy hozzáférjen vagy módosítson az általunk kezelt adatokon. [30]

A fejlesztés során használhatjuk a más alkalmazások által biztosított Content Providereket. Ilyen lehet például a telefon tárolt névjegy adatok lekérése vagy módosítása.

3.2.4. Broadcast reciever

A broadcastok célja, hogy kommunikációt valósítson meg különböző alkalmazások között a hagyományos felhasználói szcenáriókon túl. Alkalmas lehet arra, hogy más alkalmazásokat értesítsünk, hogy valami folyamat befejeződött, vagy történt bármi olyan ami a működésük szempontjából releváns lehet. Maga az Android rendszer is küld ilyen üzeneteket, például a rendszer indításkor, vagy ha felcsatlakoztatjuk a készüléket töltőre. Ha ezen események hatására valamit szeretnénk végrehajtani fel kell iratkoznunk ezekre. [31]

3.3. Package-ek

Szerencsére, ahogy a legtöbb népszerű platformra való fejlesztésnél, nem kell mindent magunknak megalkotni az elejétől. Az Android közösség rengetek különböző csomagot állít rendelkezésünkre, melyek lehetővé teszik a gyors és egyszerű megoldását a gyakori problémáknak. A packageket a gradle fájlokba írva importálhatjuk a projektünkbe.

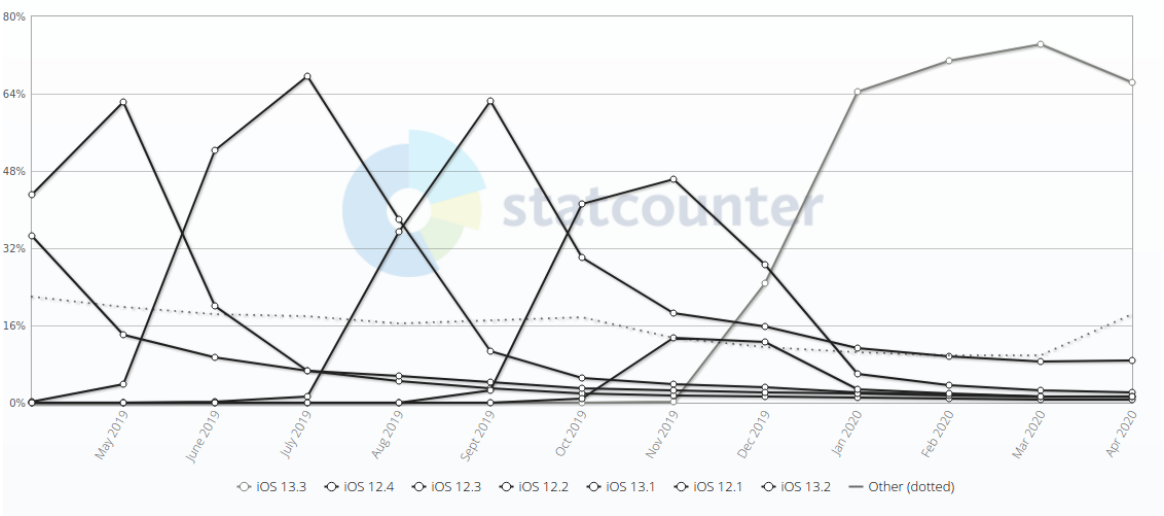
4. iOS

Az iOS egy Darwin alapú rendszer, melyet az Apple fejleszt a kizárólag általa forgalmazott iPhone és iPod készülékekre. [32]

Mivel az iOS számos közös alapot oszt a macOS rendszerrel, fejlesztésére csak az Apple saját platformjáról van lehetőség az XCode IDE segítségével. A natív alkalmazásokat a hősidőkben még NEXT által fejlesztett Smalltalk alapú Objective C nyelven fejleszthetjük, vagy az újonnan bemutatott, Apple által készített Swiftben. Mindkét esetben az általunk írt alkalmazás közvetlenül gépi kódra fordul. [32]

8. ábra. Képernyőkép az iOS 13-ról. [32]

Mivel a készülékek és azok szoftvere is házon belül az Apple-nél történik, a rendszerre való fejlesztés során az Androidnál ismertetett hátrányok legnagyobb része nem jelent problémát. Hála a relatíve kevés különböző készülék típusnak, a főbb frissítések jellemzően rendkívül gyorsan felkerülnek a készülékek nagy részére és tesztelés során is sokkal biztosabban ellenőrizhetjük a felhasználói élményt. [33]



9. ábra. Az iOS verziók elterjedségét mutató diagram [33]

5. Cross platform

A cross platform fejlesztési módszerek céja, hogy lehetővé tegyék a fejlesztők számára, hogy ne kelljen külön-külön natív alkalmazásokat létrehozni az elterjedt platformokra, hanem egy egyszer lefejlesztett alkalmazás használható legyen mind Androidon mind iOS-en. Bár az ilyen módszerrel készült alkalmazások minősége és sebessége jellemzően nem éri el a natív alkalmazásokét, de rendszerint olcsóbb és gyorsabb a fejlesztésük, valamint a későbbiekben is könnyebb a kód karbantartása, hiszen csak egy kódbázissal kell foglalkozni. [34]

Ebben a fejezetben néhány népszerű cross platform fejlesztési lehetőséget szeretnék bemutatni.

5.1. Progressive Web App

A Progressive Web App technológia lényege, hogy webes technológiákkal (HTML, CSS, JavaScript) létrehozott weboldalakat készíthessünk, melyek látszólag natív alkalmazásként jelennek meg a telefonon. Így működőképesek offline, tudnak értesítéseket küldeni, vagy épp képesek hozzáférni a fájlrendszerhez. Ezeket az alkalmazásokat nem a hagyományos módon a Google Play Store-ból vagy az App Store-ból telepíthetjük, hanem a böngészőnk segítségével az ezt támogató weboldalakon. A böngésző motorja lesz felelős az alkalmazás futtatásáért. [35]

Ehhez hasonló, de nem kimondottan ajánlott megoldás az, hogy az alkalmazásunk csupán egy WebView-ból áll, amely megjeleníti a weboldalunkat. Ennek előnye, hogy így már a népszerű alkalmazásboltokból telepíthetővé válik, viszont nem egy elegáns és effektív megoldás.

5.2. Xamarin

A Xamarin egy jelenleg Microsoft tulajdonában lévő szoftvercég, melynek eszköztára lehetővé teszi, hogy C# nyelvben fejlesszünk alkalmazásokat iOS, Android és Windows platformokra, úgy hogy csak egy kódot kell karbantartanunk és minden rendszeren az annak megfelelő natív UI elemeket fogja használni. [36]

5.3. React Native

A React Native egy Facebook által fejlesztett nyílt forráskódú multiplatform mobilalkalmazás fejlesztésére alkalmas Framework. Lehetővé teszi, hogy JavaScriptben a React könyvtár felhasználásával készítsünk felhasználói felületeket, melyeket aztán natív elemekként renderelünk. Ez lehetővé teszi, hogy a natív alkalmazásokhoz hasonló teljesítményt biztosítsunk, közben élvezve a multiplatform előnyeit. [37] A Facebookon kívül használja a Walmart, az Airbnb vagy a Discord is a technológiát. [38]

5.4. Flutter

Az egyik legújabb cross platform fejlesztési lehetőség a Google által fejlesztett nyílt forráskódú UI kit - a Flutter. Saját UI elemeket (widgeteket) használ, és ezeket a Google általt fejlesztett Skia grafikus motor segítségével rendereli. Ennek következtében a Flutter kimagasló teljesítményt ér el, amely némely esetben képes elérni a natív alkalmazások szintjét is. [39] [40]

Flutter esetén a fejlesztés a szintén a Google által készített Dart programozási nyelven történik.

6. A fejlesztés után

Az alkalmazásunkat még fejlesztésünk folyamatában célszerű bizonyos tesztautomatizációs metódusoknak alávetni. Jellemzően Unit teszteket készítünk, melyek az alkalmazás bizonyos részeinek logikáját és megfelelő funkcionalitását tesztelik, illetve UI teszteket, melyekben a felhasználó felület konkrét működését, az alkalmazásunk UI elemeit teszteljük (a megfelelő gombok azt csinálják e, amit kell stb.).

Az automatizált módszereken túl szokás tesztelőket is alkalmazni, akik az alkalmazás legújabb verzióját folyamatos használattal ellenőrzik, és hibát keresnek benne. Mind a Google Play Store, mind az App Store lehetőséget ad a fejlesztőknek, hogy béta tesztelésre publikálják az alkalmazásaikat, mely így csak a programban jelenlévő felhasználók eszközeire települ. A masszív tesztelés segíthet az alkalmazásban lévő hibák, gyors és egyszerű felfedezésére, mielőtt még az átlag felhasználók kezébe jutna a szoftver.

A tesztek lefolytatása után az alkalmazást publikálhatjuk a nagy alkalmazás áruházakban, azonban ehhez a rendszerükben regisztrálni kell, a tagsági díjat ki kell fizetni, és az alkalmazásunknak a Google és az Apple követelményeinek meg kell felelnie. Az elkészült programot különböző teszteknek alávetik, biztosítva ezzel, hogy a storeba csak megfelelő minőségű, a szabályokat kielégítő alkalmazás kerül be. [41]

7. Összefoglaló

Kutatómunkám során megismerkedtem az utóbbi években óriásivá nőtt okostelefon piac részleteivel, a népszerű platformok (iOS és Android) alapvető fejlesztési lehetőségeivel, valamint a fejlesztett alkalmazások tesztelési, és publikálási módszereivel.

Láthatjuk, hogy habár továbbra is a natív alkalmazások fejlesztése a legnépszerűbb, és ez eredményezi a legjobb minőségű végeredményt, már vannak más lehetőségeink is. Az utóbbi években megjelent cross platform fejlesztési módszerek mind tudásban, mind teljesítményben képesek egyre inkább megközelíteni ezeket. Ezen modern technológiák lehetővé teszik, hogy bárki, rendkívül gyorsan és könnyen belekezdhessen saját alkalmazásának fejlesztésébe.

A legnépszerűbb platformokon pedig alkalmazásainkkal egy hatalmas, folyamatosan növekvő piachoz férünk hozzá, mely kellő motivációt jelenthet a bemutatott módszerek elsajátításához, és saját alkalmazások publikálásához.

Hivatkozások

[1] https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[2] https://index.hu/techtud/2018/08/02/tortenelmi_rekod_1_billio_dollart_er_az_apple/

[3] https://www.hwsw.hu/hirek/60636/okostelefon-piac-idc-huawei-samsung-apple-2019-q2.html

[4] https://infoter.hu/cikk/lassul-az-okostelefon-vilagpiaci-novekedese

[5] https://blog.rescuetime.com/screen-time-stats-2018/

[6] https://en.wikipedia.org/wiki/Smartphone

[7] https://www.weforum.org/agenda/2018/03/remembering-first-smartphone-simon-ibm/

[8] https://commons.wikimedia.org/wiki/File:IBM_SImon_in_charging_station.png

[9] https://www.pcmag.com/news/the-golden-age-of-pdas

[10] https://www.arukereso.hu/pda-c3576/hp/ipaq-hw6515-mobile-messenger-p611933/

[11] http://oldcomputers.net/apple-newton.html

[12] https://www.cnbc.com/2019/12/16/apples-iphone-created-industries-and-changed-the-world-this-decade.html

[13] https://en.wikipedia.org/wiki/IPhone_(1st_generation)

[14] https://hasznaltalma.hu/tech-hirek/2014/06/29/het-eves-az-iphone

[15] https://en.wikipedia.org/wiki/Android_(operating_system)

[16] https://www.youtube.com/watch?v=-N80hzTpglQ

[17] https://gs.statcounter.com/os-market-share/mobile/worldwide

[18] https://en.wikipedia.org/wiki/Android_software_development

[19] https://www.techyourchance.com/why-google-adopted-kotlin-for-android/

[20] https://en.wikipedia.org/wiki/Android_Runtime

[21] https://en.wikipedia.org/wiki/Android_10

[22] https://qz.com/472767/there-are-now-more-than-24000-different-android-devices/

[23] https://developer.android.com/training/basics/supporting-devices/platforms

[24] https://gs.statcounter.com/android-version-market-share/mobile-tablet/worldwide

[25] https://developer.android.com/guide/components/activities/intro-activities

[26] https://developer.android.com/reference/android/app/Activity

[27] https://developer.android.com/guide/components/activities/activity-lifecycle

[28] https://developer.android.com/guide/components/fragments

[29] https://developer.android.com/guide/components/services

[30] https://developer.android.com/guide/topics/providers/content-providers

[31] https://developer.android.com/guide/components/broadcasts

[32] https://en.wikipedia.org/wiki/IOS

[33] https://gs.statcounter.com/ios-version-market-share/mobile-tablet/worldwide

[34] https://ionicframework.com/resources/articles/what-is-cross-platform-app-development

[35] https://web.dev/what-are-pwas/

[36] https://en.wikipedia.org/wiki/Xamarin

[37] https://en.wikipedia.org/wiki/React_Native

[38] https://reactnative.dev/

[39] https://en.wikipedia.org/wiki/Flutter_(software)

[40] https://medium.com/swlh/flutter-vs-native-vs-react-native-examining-performance-31338f081980

[41] https://www.goodbarber.com/blog/how-to-publish-your-app-on-google-play-and-the-app-store-a107/