Gradus Vol 8, No 2 (2021) 137-151
ISSN 2064-8014

NATÍV ÉS CROSS-PLATFORM MOBIL FEJLESZTÉS BEMUTATÁSA

DEMONSTRATION OF GENUINE NATIVE AND CROSS-PLATFORM MOBLIE DEVELOPMENT

Agg Péter András [1], Bolla Kálmán Milán [1], Kovács Márk [1], Pásztor Attila [1],*

[1] Informatika Tanszék, GAMF Műszaki és Informatikai Kar, Neumann János Egyetem, Magyarország
https://doi.org/10.47833/2021.2.CSV.005

Kulcsszavak:
Android
iOS
Xamarin
Cross-platform
Mobil fejlesztés

Keywords:
Android
iOS
Xamarin
Cross-platform
Mobile app development

Cikktörténet:
Beérkezett 2021. augusztus 31.
Átdolgozva 2021. októbe 1
Elfogadva 2021. október 5

Összefoglalás
Írásunk célja a mobil alkalmazásfejlesztés lehetőségeinek bemutatása natív Android, iOS és keresztplatform Xamarin keretrendszer használatával egy előre meghatározott követelményrendszer alapján. Az eredmények alapján egy következő vizsgálatban meghatározzuk egy megvalósítandó csomagoló géphez használandó mobil alkalmazásfejlesztés optimális keretrendszerét.
Jelen kutatásunkat az "Ipar 4.0 moduláris felépítésű ipari csomagológép fejlesztése integrált adatelemzéssel és mesterséges intelligenciára épülő optimalizálással, hibaelemzéssel, 2020-1.1.2-PIACI-KFI-2020-00062 számú pályázat" céljaihoz igazítva végeztük.

Abstract
This thesis intends to demonstrate the possibilities of mobile application developments by -using native Android iOS and cross-platform Xamarin framework System of requirements has been pre-defined. We will define the optimal framework of mobile application development for the future packaging machine in a following examination based on results.
Our research was carried out with the development of the Industry 4.0 modular industrial packaging machine with integrated data analysis and optimization based on artificial intelligence, error analysis, adapted to the goals of the tender No. 2020-1.1.2-PIACI-KFI-2020-00062.

1. Bevezetés

A mobil alkalmazásfejlesztés egyik nagy kihívása a megfelelő programozási nyelv és keretrendszer kiválasztása a szoftver tényleges megvalósítását megelőzően. Tervezési szakaszban az elkészítendő szoftverrel kapcsolatban már – optimális esetben – kellő információval rendelkezünk, azonosítottuk a kockázatokat, meghatároztuk a funkcionális és nemfunkcionális követelményeket. Megvalósítás szempontjából két utat választhatunk: a két nagy mobil platform (Android és iOS) által hivatalosan is támogatott, karbantartott natív megoldásait, vagy abban bízva, hogy egy jól kiválasztott cross-platform megoldást használva időt és pénzt spórolhatunk.

* Kapcsolattartó szerző: pasztor.attila@gamf.uni-neumann.hu

Tételezzük fel, hogy a megvalósítandó mobil alkalmazásunkkal kapcsolatban az alábbi követelmények állnak fenn:

- Lista-részletes nézet megjelenítés
- Dashboard felület diagramok felhasználásával
- Szerveroldali alkalmazáshoz csatlakozás REST API-on keresztül
- Értesítések létrehozása

Ezek közül két legfontosabb a listamegjelenítés, amely szinte minden Android, iOS alkalmazásban megtalálható, valamint a szerveroldali komponenssel való kommunikáció, mivel az esetek nagy részében a mobil alkalmazások csak vékonykliensként működnek egy központi szerveroldali alkalmazással együtt. Cikkünkben a felsorolt követelmények alapján mutatjuk be a natív Android, iOS és Xamarin cross-platform fejlesztési lehetőségeket.

Az első három fejezetben bemutatjuk az adott keretrendszerre vonatkozó fejlesztőkörnyezetet, preferált programozási nyelvet, dashboard felület létrehozását, listás megjelenítést, értesítések létrehozását, valamint a REST API-hoz való csatlakozás és kommunikáció lehetőségeit. Első fejezetben natív Android, második fejezet a natív iOS, majd a harmadik a keresztplatform Xamarin fejlesztéssel foglalkozik.

2. Android (natív)

Az Android [7][38][39] egy Linux kernelen alapuló mobil operációs rendszer, melynek fejlesztését és javítását a Google végzi. Android rendszer alábbi négy fő rétegből áll (egymásra épülve):

- Linux kernel
- Hardware absztrakciós réteg (HAL)
- Natív osztálykönyvtárak és ART
- Java API

A Linux kernel feladata a memória és az eszköz energia menedzsmentjének megvalósítása, továbbá itt helyezkednek el a hardverelemek működtetéséhez szükséges meghajtóprogramok is. HAL (Hardware Abstraction Layer) rétegben szabványos interfészen keresztül érhetjük el az egyes hardverelemeket (pl. kamera) magasabb szintű rétegekből (pl. Java API). Natív osztálykönyvtárak további funkciókat biztosítanak az eszközben található hardware elemek eléréséhez nem csak C/C++ nyelven, hanem akár Java interfészen keresztül is. Ugyanezen a szinten található meg az ART (Android Runtime), amely az Android 5.0-s (21-es API szint) verziótól kezdődően a korábbi Dalvik Virtual Machine-t volt hivatott leváltani. Bevezetésével az alkalmazások kevesebb erőforrást használnak és gyorsabb futási sebességgel működteti őket az operációs rendszer. Negyedik réteg a Java API, ahol a fejlesztéshez használt főbb funkciók érhetők el, mint például a komponensek és rendszerszintű szolgáltatások. Legfelső réteget az operációs rendszerrel érkező alapértelmezett alkalmazások alkotják. Ilyen a böngésző, tárcsázó alkalmazás, névjegyzék, e-mail kliens, stb.

2.1. Fejlesztőkörnyezet

Az Android fejlesztés hivatalos fejlesztőkörnyezete az Android Studio [8], amely egy IntelliJ IDEA alapú integrált fejlesztőkörnyezet kifejezetten Android platformra való fejlesztéshez. Több operációs rendszerre is elérhető, 2013-as megjelenésétől kezdődően a Google ezt a fejlesztőkörnyezetet ajánlja (korábbi Eclipse fejlesztőkörnyezetet váltotta le). Telefon és tablet alkalmazásokon túl TV és viselhető eszközök Android alapú fejlesztéséhez is használhatjuk.

Az Android Studio modern fejlesztőkörnyezetéhez hozzátartozik a kódkiegészítés, navigálás, layout menedzsment, automatikus tesztek futtatása, verziókezelés. Továbbá innen valósíthatjuk meg a különböző Android SDK verziók (SDK Manager) és emulált eszközök (AVD Manager) kezelését.

Szolgáltatásai:

- Gradle-alapú fordító rendszer
- Emulátorok támogatása
- Minden Andoid eszköz támogatása
- Instant Run: változásokat azonnal meg tudjuk jeleníteni az eszközön anélkül, hogy új apk állományt kellene létrehozni

- Projekt template-ek
- GitHub integráció
- Automatikus tesztelés
- Eszközök az alkalmazások elemzéséhez (memóriahasználat, processzor használata, stb.)
- C++ és Android NDK támogatás
- Google különböző platformjai be vannak építve a fejlesztőkörnyezetbe (pl. Firebase)

2.2. Programozási nyelv

Android fejlesztés esetén két programozási nyelvet használhatunk (Android Studio mind a kettőt támogatja):

- Kotlin
- Java

A Google 2017-ben jelentette be, hogy hivatalosan is támogatja Android alkalmazások fejlesztését Kotlin nyelven, később pedig a Java nyelv helyett a Kotlin vált az Android fejlesztés elsődleges programozási nyelvévé.

A Kotlin nyelv egy új programozási nyelv, amely Java platformon fordul le. Egy tömör, biztonságos, pragmatikus programozási nyelv, mely együtt tud működni Java kóddal. Majdnem mindenhol használható, ahol a Java is: szerveroldali fejlesztésekhez, Android alkalmazásokhoz és még sok máshoz. A Kotlin együttműködik a már meglévő Java könyvtárakkal és keretrendszerekkel, és ugyanolyan teljesítményt nyújt, mint a Java.

A fejlesztőkörnyezet mindkét programozási nyelvet támogatja, újonnan létrehozott projekt esetén kiválaszthatjuk melyiket részesítjük előnyben. Ettől függetlenül, ha Kotlin nyelvet választjuk a korábban létrehozott Java komponenseink szintén használhatók a projektekben, így akár vegyesen Kotlin és Java kódot is tartalmazhat egy alkalmazás.

2.3. Kotlin szolgáltatásai

A Kotlin használata mellett leggyakrabban a tömörség, olvashatóság és a biztonság érvek sorolhatók fel. Számos olyan szolgáltatást nyújt, amely elengedhetetlen egy modern programozási nyelvben. Egyik fontos tulajdonsága a nyelvek közötti átjárhatóság. Az nyelvi átjárhatóság lehetővé teszi, hogy C/C++ vagy iOS alkalmazásokban használjuk a Kotlin nyelvből fordított moduljainkat, valamint Kotlin nyelvű alkalmazásban is használhatunk natív C/C++ vagy iOS osztálykönyvtárakat.[34]

Szolgáltatási között szerepelnek a Java POJO osztályainak egyszerűsítése, Kotlin nyelvben bevezetett data class kulcsszavakkal egy sorban is megvalósíthatók privát adattagok, publikus getter-setter, equals, hashcode és toString metódusok és konstruktorok.

```
data class User(val name: String = ""  , val email: String? = "" )
```

Bővítő függvények használatával meglévő osztályok funkcionalitását egészíthetjük ki anélkül, hogy az adott osztálytól örökölni kellene. forEach extension method általános felépítése:

```
public inline fun <T> Iterable<T>.forEach(action: (T) -> Unit): Unit {
        for (element in this) action(element)
}
```

Kotlin összes szolgáltatása közül az egyik leghasznosabb és legnagyobb előnyt jelenti a Java nyelvvel szemben a Kotlin coroutines. Android alkalmazások fejlesztésekor fontos szerepe van a program többszálúsításának. Hálózati, adatbázis, háttértárral kapcsolatos műveleteket tipikusan háttérszálon kell végrehajtani, amennyiben ezt az alkalmazás főszálában próbálnánk meg futási időben kivételt kapunk. Ennek az az oka, hogy hosszantartó feladatok vagy erőforrásigényes folyamatok tartoznak az előbb említett műveletkehez. A Java-ban használatos szál létrehozás, callback megoldás vagy reaktív folyamatok helyett egyszerűbb és olvashatóbb megoldást kínál a Kotlin. A Kotlin coroutines úgy oldja meg az aszinkron műveletek végrehajtási problémáját, hogy a megírt aszinkron kódrészlet nem fog különbözni a szinkron programrésztől. Ezt úgy oldja meg, hogy nem blokkolja a végrehajtás idejére az adott szálat, hanem csak felfüggeszti annak végrehajtását. Később, amikor az eredmény rendelkezésre állt a program végrehajtása ugyanonnan fog folytatni.

2.4. Dashboard létrehozása

Aggreált adatokat diagramok segítségével tehetjük könnyen értelmezhetővé a felhasználók számára. Diagramok közül legtöbbször használtak az egyszerű vagy csoportosított oszlopdiagramok, vonaldiagram, kördiagram, pontfelhő. Android-ban ilyen jellegű vizuális megjelenítést lehetővé tevő vezérlőelemeket nem találunk az eszköztárban, ezért harmadik féltől származó csomagokra kell hagyatkoznunk. Több lehetséges megoldás közül az *MPAndroidChart* [16] csomag kerül bemutatásra.

Általános használata:

- Csomagfüggőség beállítása a modul gradle állományában
- Felületen (layout xml) helyezzük el a megjelenítéshez használt diagramtípust
- Activity, Fragment háttérkódjában elvégezzük a diagram megjelenítéséhez szükséges beállításokat, és átadjuk az adathalmaz referenciáját

Fontos megjegyezni, hogy a diagramok megjelenítéséhez szükséges paramétereket Java vagy Kotlin kódból állítjuk be. Ilyen beállítás a cím, adatsorokhoz rendelt szín, vízszintes és függőleges címkék, jelmagyarázatok, nagyítás engedélyezése-tiltása, animáció, stb.

2.5. Listák

A listák létrehozására két típus áll rendelkezésre. Egyszerű listák esetén használhatjuk a *ListView* osztályt, de a legtöbb esetben érdemes a *RecyclerView*-t használni. A *RecyclerView* a *ListView* osztály egy fejlettebb és flexibilisebb verziója, beépítetten támogatja a nézetek újrahasznosítását, ezáltal jobb teljesítményt biztosít nagy adathalmazok esetében.

Listák létrehozásának általános lépései Android platformon:

1. Modell osztály létrehozása egy listaelemhez (egyszerű listáknál elegendő a beépített típusok használata is, pl. String)
2. Egy listaelem felületének létrehozása (egyszerű esetben az Android-ban található, előre definiált felületeket is használhatjuk)
3. Listás vezérlő elhelyezése a felületen
4. Adapter létrehozása a megjelenítés vezérléséhez
5. Fragment vagy Activity komponensben listás vezérlő inicializálás, adapter beállítása

Minden listához szükséges egy Adapter-t definiálni, amely a kapcsolatot fogja biztosítani az adatok és a megjelenített nézetek között. Az adapterekből különböző implementációk léteznek attól függően, milyen formában áll rendelkezésre a megjelenítendő adat. Tömbök esetén használható az *ArrayAdapter* osztály, *Cursor* esetén pedig a *SimpleCursorAdapter* [1]. *RecyclerView* használatához viszont létre kell hoznunk egy saját adapter osztályt, ami a *RecyclerView.Adapter* osztálytól kell, hogy örököljön. Az adapter végzi a nézetek létrehozását, újrafelhasználását és az adatok frissítését. A működéshez szükséges még a listához tartozó *ViewHolder* osztály létrehozása, amely a lista egy elemét reprezentálja, és annak nézet elemeit (*View*) fogja tartalmazni. Az adapter a lista első betöltésekor szükség szerinti számban dinamikusan létrehozza a *ViewHolder* példányokat az *onCreateViewHolder* metódussal. Mielőtt egy *ViewHolder* megjelenítésre kerül, az *onBindViewHolder* metódus kerül meghívásra, amelyben megtörténik a megjelenítendő adatok beállítása az adott elemre. Ha egy *ViewHolder* példány lekerül a képernyőről és már nem lesz látható, akkor újra felhasználásra kerül és az *onBindViewHolder* metódusban megtörténik az adatok frissítése. A felület elrendezésétől függően (lista, rács stb..) meg kell határozni egy *LayoutManager* példányt, ami a listát alkotó elemek méretezéséért a pozícionálásáért felel. Egy oszlopos listákhoz használhatjuk például a *LinearLayoutManager* osztályt, több oszlopból álló rácsos felületekhez pedig a *GridLayoutManager*-t.

2.6. Értesítések

Értesítések (notification) [20] az alkalmazás felületén kívül megjelenő rövid, szöveges üzenetek, melyeket elsősorban a felhasználó figyelemfelkeltésére használjuk. Meghatározott esemény hatására vagy bizonyos időközönként állhat elő egy értesítés. A felhasználók interakcióba léphetnek a notification-nel, amivel az alkalmazásba való navigációt valósíthatjuk meg vagy akár az

értesítés felületére is ki tudunk vezetni alkalmazásfunkciókat. Értesítéseket az Android rendszer status bar elemén érhetjük el, illetve Android 5.0-tól kezdődően záróképernyőn is megjelenhetnek.

Értesítés létrehozásához szükséges típusok az *androidx.core.app* csomagban találhatók, ezért a modulhoz tartozó gradle állomány függőségei között szerepeltetni kell. *NotificationCompat.Builder* segítségével hozhatunk létre az értesítés (*Notification*) típust. Amennyiben új értesítést szeretnénk létrehozni és nem a meglévőt akarjuk felülírni, abban az esetben mindig új *NotificationCompat.Builder* típust kell példányosítanunk. *Notification* típuson keresztül tudjuk az értesítés tartalmát meghatározni.

Notification kötelező elemei:

- Ikon: status bar-on is megjelenő ikon, amely alapján könnyebben beazonosítható melyik alkalmazásból érkezett az értesítés
- Alkalmazás neve: rendszer biztosítja, nem kell beállítanunk
- Időbélyeg: szintén a rendszer által biztosított, az értesítés keletkezését jeleníti meg

Notification opcionális elemei:

- Nagy méretű ikon: értesítés jobb oldalán megjelenő kép
- Cím: tartalomhoz tartozó cím
- Tartalom: értesítés szöveges tartalma
- Prioritás: értesítés fontossága itt állítható

Végül a *NotificationManagerCompat* típust használjuk a felparaméterezett értesítés kiküldéséhez. [12]

2.7. REST API integráció

Habár az Android biztosít a Java API-on keresztül olyan típusokat, amellyel szerveren tárolt erőforrások érhetők el, mégis egy harmadik féltől származó csomaggal, a Retrofit-tel érdemes megvalósítani az alkalmazásunk REST API-hoz való csatlakozását.

A Retrofit [25] egy típusbiztos HTTP kliens Android és Java platformra REST szolgáltatások használatához. Előnye, hogy nagyban leegyszerűsíti a kommunikáció felépítésével és kezelésével kapcsolatos műveleteket, így több idő marad további funkciók fejlesztésére, tesztek írására. Tartalmazza a kapcsolatok létrehozásának kezelését, szálkezelést, sikertelen kérések kezelését, válasz feldolgozását és a hibakezelést, tehát rengeteg hasznos funkciót.

A Retrofit a kérések leírásához annotációkat és interfészeket használ. Előnye, hogy a végpontokat dinamikusan kezelhetjük, URL szegmensek, Query string-ek, fejlécadatok beállítása egyszerűen megvalósítható. HTTP metódusok közül többet is támogat, melyek teljesen lefedik a REST webszolgáltatásokkal való kommunikációt.

Retrofit használatbavételének általános lépései:

1. INTERNET engedély beállítása a manifest állományban
2. Gradle csomagfüggőségek beállítása
3. API végpontok definiálása
4. Konverter meghatározása
5. Retrofit kliens létrehozása
6. Kérések végrehajtása és a visszakapott adatok feldolgozása

Retrofit által a HTTP metódusok Java annotációk segítségével érhetőek el, ahol az URL-ben meghatározott erőforrásokat az annotációkon belül kell definiálni. A Retrofit 5 ilyen beépített annotációt tartalmaz: GET, POST, PUT, DELETE és a HEAD.

Erőforrások eléréséhez az URL szegmenseket és Query string-eket is szabályozhatjuk a Retrofit segítségével. URL szegmensek lehetnek statikusak, de akár változóból dinamikusan is állíthatóak. Ez a funkció nagyon hasznos lehet, pl. egy URL-ben tárolt felhasználói azonosító, hitelesítéshez használt API kulcs, stb. esetén. Query string-ek szintén statikusan vagy dinamikusan meghatók az annotációkkal és függvényparaméterekkel. URL paraméterezésén túl lehetőségünk van fejlécadatok (header paraméterek) statikus és dinamikus kezelésére is.

Példa Retrofit interfész egy függvényére, amely a HTTP GET kérést valósít meg:

```
@Headers({ "Accept: application/vnd.github.v3.full+json", "User-Agent: Retrofit-Sample-App" }) @GET("users/{username}") Call<User> getUser(@Path("username") String username);
```

Az előző példában látható, hogy a függvény visszatérési értéke egy *Call<T>* típusban kapjuk vissza. *T* ebben az esetben a választ (response) reprezentáló *User* DTO típus. *Call<T>* típusra azért van szükség, mert amikor ezt a függvényt meghívjuk valójában a kérést nem küldjük el, csak a kéréssel kapcsolatos metaadatokat kérdezzük le és tároljuk el egy változóban. A kérés (*Call<T>*) szinkron vagy aszinkron módon is végrehajthatjuk. Utóbbi esetben egy másik típusban a *Callback*-ben (interface típus) kapjuk meg a kérés eredményét. Két függvényt tartalmaz: *onResponse* és *onFailure*. Hálózati hiba esetén *onFailure* függvény kerül meghívásra, *onResponse* függvényhez pedig akkor kerül a végrehajtás, ha válaszolt a szerveroldal. Ebbe beletartozik az is, ha szerveroldali (500) hibát kapunk vagy a hitelesítéssel (401, 403) volt probléma a kérés során.

Kliens és szerver közötti kommunikáció során el kell dönteni milyen formátumban történik az adatok továbbítása, amely formátum mindkét fél számára ismert kell, hogy legyen. A Retrofit számos adatformátumot kezel az elérhető szöveges és bináris adatformátumok közül, amelyek közül mindegyikhez konverterek tartoznak (pl.: json formátumhoz a Gson konverter). Amennyiben szükséges a szerializáció, deszerializáció folyamatát is befolyásolhatjuk, pl. speciális dátumformátum kezelése, válasz struktúrája eltér az alkalmazásban definiált DTO-któl.

Retrofit által támogatott konverterek:

- Gson: com.squareup.retrofit2:converter-gson
- Jackson: com.squareup.retrofit2:converter-jackson
- Moshi: com.squareup.retrofit2:converter-moshi
- Protobuf: com.squareup.retrofit2:converter-protobuf
- Wire: com.squareup.retrofit2:converter-wire
- Simple XML: com.squareup.retrofit2:converter-simplexml

3. iOS (natív)

Az iOS [35][36] egy mobil operációs rendszer, amit az Apple fejleszt iPhone-okra, iPad-ekre. Az Android operációs rendszer után az iOS a legnépszerűbb rendszer. Hasonlóan az Android-hoz az iOS is négy egymásra épülő fő rétegből áll: [9][13]

- **Core OS**
- **Core Services**
- **Media Layer**
- **Cocoa Touch (vagy alkalmazói réteg)**

A *Core OS* rétegben találhatók az alacsony szintű szolgáltatások, mint:

Core Bluetooth Framework

- Külső kiegészítő keretrendszerek
- Accelerate Framework
- Biztonsági szolgáltatások keretrendszere
- Helyi engedélyezési keretrendszer

Mindhárom felsőbb szintű réteg ettől a rétegtől függ. Felelős a memóriamenedzsmentért, az alkalmazások életciklusainak nyomon követeséért, valamint az előbbiek szerinti leállításukért. Kommunikál a hardverrel közvetlenül, valamint a fájlrendszer helyes működéért is ez a réteg felel.

A *Core Services* réteg absztrakciót nyújt a Core OS rétegben nyújtott szolgáltatások felett. Alapvető hozzáférést biztosít az iPhone operációs rendszer szolgáltatásaihoz. Az alapszolgáltatási réteg a következő összetevőkből áll:

- Collections, Core Data: az adatok tárolására és menedzselésére alkalmas összetevők
- Address Book: a kontaktok eltárolása
- CFNetwork: a hálózati kommunikációért felelős az iOS rendszerben
- Threading: szálkezelés
- File Access: hozzáférést biztosít az alacsonyabb szintű operációs rendszerek szolgáltatásaihoz.
- Core Location: az eszközök helyének meghatározására szolgál
- HomeKit: az okosotthon kialakításához tartozó eszközökkel képes kommunikálni

A *Media Layer* olyan multimédiás szolgáltatásokat nyújt, amelyeket az iPhone-on és más iOS-eszközökön használhatunk. A média réteg a következő összetevőkből áll:

- Core Media: kezeli az audió fájlok és stream-ek lejátszását és rögzítését, illetve hozzáférést biztosít az eszköz hangfeldolgozó egységeihez.
- OpenGL: 2D és 3D animációk létrehozására szolgál
- Core Graphics: képi erőforrások kezelésére alkalmas
- Core Animation: nézetek és egyéb tartalmak animálásához használja az iOS rendszer
- AVFoundation: az audiovizuális média rögzítésére, feldolgozására, ellenőrzésére, importálására szolgál

A *Cocoa-Touch* réteg absztrakciós réteget biztosít a különféle könyvtárak eléréséhez az iPhone és más IOS-eszközök programozásához a Multi-Touch segítségével:

- UIKit: a felhasználói grafikus interfészt jelenti az összes felületi vezérlőt ide sorolhatjuk
- MapKit: az alsóbb rétegek felhasználásával a térképnézet összes tulajdonságáért és megjelenítésért felel
- Message UI: új üzenet érkezésekor ez a speciális nézet jelenik meg, ami nem minden esetben takarja el teljesen az aktuálisan megnyitott alkalmazás, illetve bezárása nem szükséges
- PushKit: felugró értesítések a különböző szolgáltatásokról
- GameKit: lehetővé teszi, hogy a különböző játékokban kapcsolatot létesíthessenek egymással a játékosok, ezzel lehetővé téve a többjátékos módot.
- EventKitUI: a különböző események kezelésére szolgál

3.1. Fejlesztőkörnyezet

Az Apple az iOS, iPad, macOS, watchOS, tvOS alkalmazások fejlesztéséhez az Xcode fejlesztőkörnyezetet biztosítja. Mindenki számára ingyenes elérhető és használható. 2003 szeptemberében jelent meg az első verziója, ami akkor még csak macOS programok fejlesztésére volt alkalmas. [33]

Az első Xcode verzió, ami az iPhone applikációfejlesztéshez is megfelelő volt az Xcode 3.0. Jelenleg az Xcode 13-as verziójánál járunk, amely a kezdetek óta hatalmas fejlődésen esett át, mert az Apple összes platformjára tudunk alkalmazásokat vele fejleszteni.

A fejlesztőkörnyezet segítségével könnyedén kialakíthatjuk a felületünket, illetve a mögötte működő kódokat. Támogatja a fejlesztett alkalmazások tesztelését a beépített Xcode Simulator segítségével, ami hasonló körülmények között kipróbálható, mintha fizikai eszközön futtatnánk. Minden Xcode frissítéssel érkezik a legújabb iOS operációs rendszer is, amivel tesztelhető alkalmazásunk működése. [10]

Amennyiben szeretnénk fizikai eszközökön is kipróbálni, illetve AppStore-ba publikálni, abban az esetben be kell regisztrálnunk az Apple Developer Program-ba, amit éves szinten 100 dollár ellenében kérhetünk.

Lehetőségünk van kipróbálni a legújabb még nem kiadott Xcode béta verziókat is, amivel a készülő, de még nem megjelent iOS operációs rendszer verzióival is tesztelhetjük alkalmazásunk működését. Ezzel hamarabb megtalálhatjuk azokat a felmerülő hibákat, melyeket ki kell javítanunk az újabb iOS szoftver megjelenése előtt.

3.2. Programozási nyelv

Az Xcode korábban az Objective-C nyelvet használta alkalmazások fejlesztéséhez. Azonban az Objective-C az 1980-as évek eleje óta nagyrészt változatlan volt, és hiányzott belőle a modern nyelvi jellemzők.

Természetesen a főbb célok közé tartozott a korábban használt nyelv együtt tudjon működni, ezért 2014-ben az Apple kifejlesztette saját programozási nyelvét a Swift-et. A fejlesztés során, ami még nem volt megvalósítható Swift nyelven azt megírhattuk közvetlenül a Swift fájlban Objective-C nyelven is.

A Swift támogatja a protokoll kiterjeszthetőségének koncepcióját, egy kiterjesztési rendszert, amely típusokra, struktúrákra és osztályokra alkalmazható, melyet az Apple a programozási paradigmák valódi változásaként támogat, az általuk "protokoll-orientált programozásnak" neveznek.

Jelenleg a Swift nyelv 5.4-es verziójánál járunk, amely többek között Linuxon vagy Windows-on is lefordítható. A Swift nem csak alkalmazások fejlesztésére alkalmas, hanem hasonlóan más objektum-orientált nyelvekhez készíthetünk asztali macOS alkalmazásokat vagy akár kisebb grafikus felület nélkül rendelkező úgynevezett szkripteket.

3.3. Felület kialakítási technológiák

Két fő technológiát használhatunk a felület kialakításáért: [11]

- UIKit
- SwiftUI

A UIKit egy régebben bevezetett technológia az első iPhone-ok megjelenése óta. Egy egyszerűbb néhány nézetes felületet kitudunk könnyedén alakítani, azonban több pl. 5-nél több View esetében már alig átlátható a nézetek közötti kapcsolat.

UIKit esetében, ha egy vezérlő működését szeretnénk leírni, ahhoz szükséges egy speciálisan csak az Xcode-ban használt kapcsolat a vezérlő és a kód között, amit Outlet-nek nevezünk. UIKit esetében szinte teljesen független a kód és a felület az előbbiek megvalósítása után.

A SwiftUI első verziója 2019 őszén érkezett meg, amely a Swift nyelvet felhasználva, tovább fejlesztve ad lehetőséget a felület kialakítására a SwiftUI felületet leíró programnyelv segítségével. Ha nem szeretnénk a teljes projektet lefordítani csak a felület működését tesztelni, rendelkezésünkre áll egy Preview Layout. Az előbbi segítségével Live üzemmódban láthatjuk a felület legapróbb módosításának eredményét az alkalmazás futtatás nélkül.[28]

A SwiftUI az előbbi problémákat képes orvosolni. Ha több nézetből álló összetettebb felületet szeretnénk készíteni, ehhez jó megoldás lehet a SwiftUI. Ha szeretnénk akár az eddig használt Swift nyelv elemeit is használhatjuk egy SwiftUI felületet leíró fájlon belül.

3.4. Dashboard létrehozása

Összesített adatok megjelenítésére a legszemléltetőbb módszer diagrammok használata. iOS-ben nincs erre beépített módszer, amivel könnyedén megtudjuk valósítani csak, abban az esetben, ha mi magunk megírjuk a teljes diagram kinézetét és működését, számolását.

Annak érdekében, hogy ne kelljen magunknak megvalósítani, használhatunk külső forrásból származó csomagokat. Ezek a külső csomagok olyan fejlesztőktől származnak, akik a globális programozási problémákra adnak általánosan felhasználható programblokkokat.

Az Xcode esetében az előbbi függőségek kezelésére a *CocoaPods* áll rendelkezésre. Ha leszeretnénk egyszerűsíteni, akár egy kisebb alkalmazásáruháznak is nevezhetjük a fejlesztők körében. Használata nagyon egyszerű, az alábbi paranccsal fel kell telepítenünk magát a CocoaPods dependency kezelőt a saját fejlesztői gépünkre egy terminál segítségével:

```
$ sudo gem install cocoapods
```

Telepítés után kereshetünk egy általunk tetszőleges projektet, amit felszeretnénk használni a saját projektünkhez. Ha ezzel megvagyunk a következő lépésben létre kell hoznunk egy Podfile-t a projektünk főkönyvtárában a következő tartalommal:

```
pod 'csomag_neve', '~> verzioszam'
```

A SwiftUI keretrendszerben a Piechart készítéséhez a PieChartSwiftUI csomagot használhatjuk. [22]

A Podfile létrehozása után a *pod install* paranccsal tudjuk betölteni a saját projektünkbe.

Ha minden rendben történt, akkor a kibővített projektünket már ne a . xcodeproj kiterjesztésű fájllal nyissunk meg, hanem a. xcworkspace fájllal, ami tartalmazni fogja a letöltött Pod projektet is. Ettől fogva használhatjuk fel a Pod projektet.

Jelen esetben SwiftUI-ban a következők szerint alakíthatjuk ki kördiagrammunkat, a PieChartSwiftUI Pod dokumentációja alapján: https://cocoapods.org/pods/PieChartSwiftUI

Egyes részarányok kiolvasása és eltárolása:

```
let firstItem = PieChartItemModel(value: Double(networkManager.factors[0].biomass), color: .orange)
let items = PCItems(items: [firstItem, secondItem, thirdItem])
```

Ezek után megjeníthetjük a diagrammot:

```
PieChartView(items: items, sliceSeparatorColor: .black)
        .frame(width: 175, height: 175, alignment: .center)
        .padding(36)
```

3.5. Listák

Abban az esetben, ha SwiftUI-ban szeretnénk megvalósítani egy listás megjelenítést érdemes különválasztani apróbb View elemekre, ami olvashatóbbá és újra hasznosíthatóvá teszi a fejlesztést.

Esetünkben egy kissé összetett lista egyes sorain belül található egy-egy kép, illetve több sorban szöveg. Ehhez egy külön SwiftUI View-ban érdemes létrehozni (jelen esetünkben ez a CarbonListItemView.swift állomány), ami tartalmazza egy általános sor adatait. Egy horizontális container, *HStack* segítségével tudjuk a képi erőforrást bal oldalra helyezni és mellé a szöveget.

```
HStack(alignment: .center, spacing: 16) {
Image(...)
            .resizable()
            .scaledToFill()
            .frame(width: 90, height: 90)
            .clipShape(RoundedRectangle(cornerRadius: 12))
VStack(alignment: .leading, spacing: 8) {
Text("Előrejelzés: \(forecast)")
... } }
```

Egy sor általános kialakítása után egy másik nézetben megjeleníthetjük az összes listaelemünket egy adathalmazból, ami lehet tömb vagy lista adatszerkezet:

```
NavigationView {
List(networkManager.datas) { item in
  NavigationLink(destination: CarbonDetailView(datas: item)) {
            CarbonListItemView(datas:item)
  } } .navigationBarHidden(true)
}
```

A NavigationView segítségével tudjuk megvalósítani, hogy egy lista elem kiválasztásakor megtudjuk jeleníteni a hozzá tartozó összes részletes adatot.

A részletes nézethez egy további View-t érdemes létrehozni (jelen esetünkben a CarbonDetailView.swift állomány). Ebben a nézetben helyezhetjük egy listaelem további adatait egyszerű *Text()* és *Image()* vezérlők segítségével.

3.6. REST API integráció

Ha nem csak statikus adatokat szeretnénk megjeleníteni az alkalmazásunkban, akkor szükséges valamilyen adatbázis végpontjához, REST API-hoz csatlakozni. Ebben az esetben egy JSON fájlt kell feldolgoznunk.

A Swift és SwiftUI nyelvek segítségével viszonylag könnyedén kialakíthatjuk valamelyik REST API-hoz történő csatlakozást külső csomag felhasználása nélkül. Első lépésben szükségünk van egy Modell osztály/struktúra létrehozására, ami tartalmazza a JSON válaszban érkező tömb összes típusának definícióját a pontos nevével. Segítségül szolgálhat számunkra valamelyik JSON konvertáló, ami átalakítja Swift modell struktúrává a válaszul kapott JSON tömböt. Használhatjuk erre a célra a https://app.quicktype.io/ oldalt. [23]

A következő lépésben a REST API-hoz való kapcsolódás következik. Egy külön Swift osztályt hozhatunk létre, ami a kimondottan ilyen célra használt *ObservableObject* protokolltól örököl. Meg kell határoznunk a végpont URL-jét (carbonintesity), valamint egy *URLSession* típust kell létrehoznunk a kapcsolódáshoz. Az *URLSession dataTask* függvényével tudjuk elindítani a kérést, illetve a választ fogadni. Természetesen az adatok kiolvasása előtt érdemes egy hibaellenőrzést végrehajthatunk, hogy minden adat jól megérkezett-e.

Az adatok kiolvasása *JSONDecoder* használatával történik a modell struktúra adattagjainak megfeleltetésével. A kapcsolat háttérszálon fut ezért a dekódolt adatokat vissza kell adnunk a főszálon belüli tömbnek. SwiftUI-ban *@Published* annotációval ellátott változókba érdemes eltárolni a feldolgozott elemeket, hogy megtudjuk őket jeleníteni a felületen az *ObservableObject* segítségével. [27]

```
func fetchList() {
if let url = URL(string: "https://api.carbonintensity.org.uk/intensity/date") {
            let session = URLSession(configuration: .default)
            let task = session.dataTask(with: url) { (data, response, error) in
```

```
            if error == nil {
                let decoder = JSONDecoder()
                if let data = data {
                    do {
                        let results = try decoder.decode(Response.self, from: data)
                        DispatchQueue.main.async { self.datas = results.data
                    } catch { print(error) } } } }
        task.resume()
    } }
```

3.7. Értesítések

Egy Push Notification létrehozása iOS-ben könnyedén megtehető szintén külső csomag használata nélkül. A következőkben egy egyszerű nyomógomb lenyomására elinduló Notification működése látható.

A nyomógomb kattintás után eseményére feliratkozunk egy UNUserNotificationCenter típussal. A gombnyomás után 5 másodperccel az alkalmazásból történő „kilépés” fog lefutni az értesítés: [26]

```
Button("Notification") {
UNUserNotificationCenter.current().requestAuthorization(options: [.alert, .badge, .sound])  {
            success, error in
                    if success {       print("authorization granted")
                    } else if let error = error {           print(error.localizedDescription)
                    } }
  let content = UNMutableNotificationContent()
            content.title = "Értesítés"
            content.subtitle = "PoC alkalmazás"
            content.body = "Ez az üzenet"
            content.sound = UNNotificationSound.default
  let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 5,repeats: false)
  let request = UNNotificationRequest(identifier: "notification.id.01", content: content, trigger: trigger)
  UNUserNotificationCenter.current().add(request)
}
```

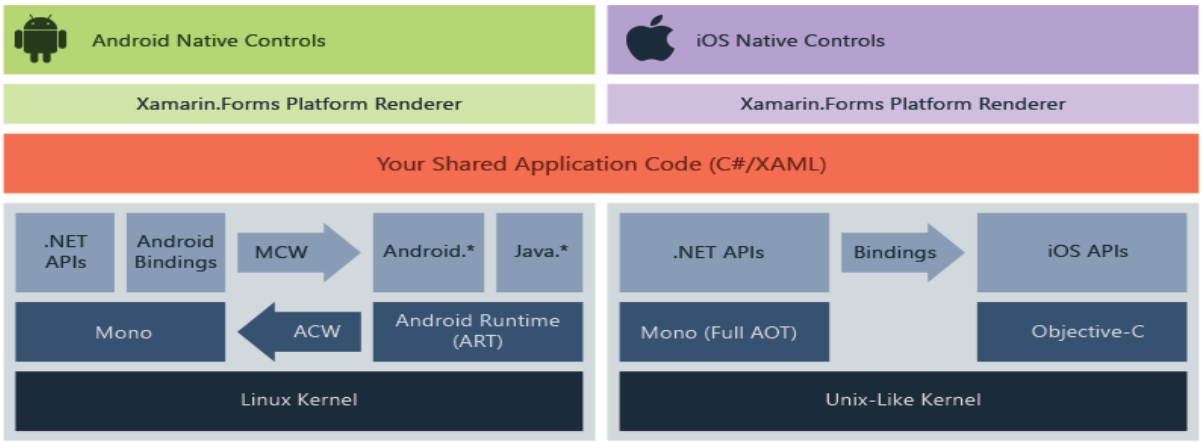
4. Xamarin (cross-platform)

Hagyományosan a két nagy platform (iOS, Android) teljesen különböző fejlesztői környezetek és programozási nyelvek használatát követelte meg, ami alaposan megnehezítette a fejlesztők az életét, ha több platformra szeretett volna egy-egy mobilalkalmazást írni. Például:
iOS: XCode (OS X-en elérhető csak), Objective-C, vagy Swift
Android: Eclipse, Android Studio, Java

4.1. Xamarin

A Xamarin [30][31][32][37] egy nyílt forráskódú platform alkalmazások fejlesztésére iOS, Android és Windows rendszerekhez .NET használatával. A Xamarin-alkalmazások írhatók PC-re vagy Mac-re, és natív alkalmazáscsomagokba állíthatók össze. A Xamarin 2016 óta a Microsoft tulajdona.

A Xamarin lehetővé teszi natív felhasználói felület létrehozását minden platformon, hozzá kapcsolva egy kódot C# formátumban, amely mindhárom platform számára elérhetővé válik. Természetesen néhány esetben szükség van bizonyos módosításikra pl. IOS esetében, de általában az mondható el, hogy az alkalmazás kódjának 90%-a megosztható a Xamarin használatával. A Xamarin a .NET keretrendszer tetejére épül, amely automatikusan kezeli az olyan feladatokat, mint például a memória kiosztása, vagy az átjárhatóság az alapul szolgáló platformok között.



1. *ábra: cross-platform Xamarin alkalmazás általános architektúráját [30]*

Funkciók - szolgáltatások:

- Mobile cross-platform support – Mobil platformok közötti támogatás
- Complete binding for the underlying SDKs - Teljes kapcsolat (adatkötés) a használni kívánt SDK-khoz (könnyű navigálás, gyorsabb működés, kevesebb hiba)
- Objective-C, Java, C, and C++ Interop használata
- Modern language constructs - Magasszintű, modern programozási megoldások – C# nyelv
- Robust Base Class Library (BCL) – osztályok - XML
- Modern Integrated Development Environment (IDE) – Modern integrált fejlesztői környezet – Visual Studio – előnyök

4.2. Xamarin főbb részei

Xamarin.Android

Xamarin. Android alkalmazásokat C#-ból fordítják az Intermediate Language (IL) [3] nyelvbe, amelyet aztán az Just-in-Time (JIT) [3] fordít egy natív összeállításba, amikor az alkalmazás elindul. Xamarin.Android alkalmazások a Mono végrehajtási környezetben futnak, egymás mellett az Android Runtime (ART) virtuális géppel. A Xamarin biztosítja a .Net összerendelését az Android.* és Java.* névterekhez.

Xamarin.iOS

Fordítás C#-ból ARM assembly kódba. Object C kezelése a C# számára, illetve az ellenkező irányban is.

Xamarin.Essentials

A Xamarin.Essentials egy olyan könyvtár, amely platformokon átívelő API-kat biztosít a natív eszköz funkcióihoz, és leegyszerűsíti a natív funkcionalitás elérésének folyamatát.

Xamarin.Forms

A Xamarin.Forms biztosítja a fejlesztők számára, hogy felhasználói felületeket hozzanak létre XAML-ben, C# kód mögött. A legfontosabb különbség a többi cross-platform megoldással szemben az, hogy a Xamarin.Forms-szal valóban natív alkalmazásokat készítünk, a natív UI elemeket használjuk, és a hivatalos SDK-k minden funkcióját elérjük!
A programcsomagok NuGet segítségével kezelhetők, frissíthetők, bővíthetők.

4.3. Fejlesztőkörnyezet

A Xamarin alkalmazás fejlesztése Visual Studio 2019-ben.
Előnyök:

- Fejlesztés különböző platformokon
- Könnyebb együttműködés
- Közreműködő mesterséges intelligencia

- Automatikus kódformázással
- Fejlett Debugging és Diagnosztika
- Tesztelési Eszközök
- Fejlesztési Platform Támogatás
- Integrált Fejlesztői környezet
- A beépített nyelvek közé tartozik a C, C ++, .NET, C #, F #, JavaScript, TypeScript, XML, Más nyelvek, például a Python támogatása plug-inek segítségével érhető el.

4.4. Programozási nyelv: C#

A C# programozási nyelv [4][5] a Microsoft által kifejlesztett programozási nyelv. Nyílt szabványú, bárki készíthet C# fordítóprogramot. Könnyen áttekinthető szintaktikája van, egyszerű, gyorsan tanulható. Alapja a C nyelv szintaktikája, emellett használja a C++, Java, Delphi nyelv adta lehetőségeket is. Objektumorientált (OOP) nyelv, vagyis minden függvényt és eljárást valamely osztályban kell elhelyezni. Ugyanez vonatkozik az adatokra is (mezők). Az osztály egy zárt egység, melybe mezők, és azokat kezelő metódusok tartoznak. Az létrehozott osztály, mint típus jelenik meg az alkalmazásban, amelyet példányosítani kell. Az osztályok adatait, metódusait védeni kell, melynek megvalósítására védelmi szintek használhatók. Public, illetve a private foglalt szavak használatával a mezők, metódusok hozzáférését módosíthatjuk, mellyel megakadályozható az illetéktelen használat, módosítás más alprogramok számára.

A névtér (namespace) a csoportba foglalás egyik eszköze. Ha az osztályokat névtérben helyezzük el, úgy az osztály neve a névtér nevével bővül ki, mint előtag. Ez akkor hasznos, ha több osztálynak is ugyanaz a neve, akkor az egyértelmű azonosítás céljából használni kell a névtér nevét is.

C# projectek leggyakoribb típusai:

- Console Application: egyszerű ablakos parancssoros programot jelent, nem releváns a mobilalkalmazások készítésének terén.
- Windows Application: hagyományos grafikus felületű alkalmazásokat jelenti.
- Web Application: web szerveren futó, böngészőből használható programok. A megjelenítést a HTML nyelven írja le, és a böngésző végzi.
- Class Library: önállóan nem indítható osztálygyűjteményt jelöl, melyet más alkalmazásokhoz csatolva lehet használni.

4.5. Dashboard létrehozása

Az adatokat megjelenítésére sok esetben diagramokat használunk, mely segítségével a felhasználók könnyebben értelmezhetik a kapott információkat. Xamarin alatt bármilyen diagramot megjeleníthetünk, de ehhez szükségünk van valamilyen nem beépített csomag használatára. Több lehetséges megoldás is van, például a *Syncfusion.Xamarin.SfChart*[29] vagy a *Microcharts.Forms*[15] melyek közül az utóbbiról írunk, a könnyebb megvalósítás miatt.

A csomag hozzáadása:

`Tools - NuGet Package Manager – Browse - Microcharts.Forms`

A csomag használata:

- XAML felületen használjuk a telepített csomagot: `xmlns:forms="clr-namespace:Microcharts.Forms;assembly=Microcharts.Forms"`, illetve jelenítsük meg a megfelelő diagram típust. (pl: `<forms:ChartView x:Name="Chart2" HeightRequest="350" WidthRequest="500" HorizontalOptions="Center" VerticalOptions="Center"/>`)
- A háttérkódban adjuk meg a szükséges adatokat, illetve végezzük el a megjelenítéshez szükséges beállításokat. (pl:

```
List<Entry> entries = new List<Entry>
{
        new Entry((float)factor.Data[0].Biomass)
          {Color=SKColor.Parse("#00BFFF"),},
        new Entry((float)factor.Data[0].Coal)
```

```
            {Color=SKColor.Parse("#00CED1"),},
          new Entry((float)factor.Data[0].Oil)
            {Color=SKColor.Parse("#FF1493"),},
};
Chart2.Chart = new PieChart { Entries = entries };
```

A diagramok megjelenítéséhez szükséges minden paramétert C# kódból állítjuk be, mint például a cím, adatsorok megjelenési színe, feliratok, jelmagyarázatok stb.

4.6. Listák

A listák [2][14] létrehozására használhatjuk a *ListView* osztályt (*RecyclerView* osztály használata is tökéletes, itt most a ListView lesz bemutatva).
Listák létrehozásának általános lépései:

1. Modell osztály létrehozása listaelemekhez (pl: beépített típusok használata, pl. String)
2. Egy listaelem felületének létrehozása (egyszerű esetben az előre definiált felületeket is használhatjuk)
3. Listás vezérlő elhelyezése a felületen
4. Adapter létrehozása megjelenítés vezérléséhez
5. Listás vezérlő inicializálás, adapter beállítása

A listákhoz mindenképp szükséges egy Adapter-t definiálni, ami a biztosítja a kapcsolatot az adatok és a megjelenített nézetek között. Az adapterekből különböző típusok attól függően, milyen formában áll rendelkezésre az adatforrás. Tömbök esetén használható az *ArrayAdapter* osztály, *Cursor* esetén pedig a *CursorAdapter*, vagy használható a BaseAdapter - alaposztály a ListView és az adatforrás összekapcsolásához. Az adapter végzi a nézetek létrehozását, módosítását, törlést és az adatok frissítését. A működéshez szükséges még a listához tartozó osztály létrehozása, amely a lista egy elemeit tartalmazza. Az adapter a lista első betöltésekor szükség szerinti számban dinamikusan létrehozza a példányokat. Kezeléskor használható a SelectedItem(), az OnItemSelected() metódus is. Navigáláskor[6] jól használható a Guid.NewGuid() metódus, ami az indexeket tartalmazza.

4.7. Értesítések

Az értesítések (notification)[19][21] az alkalmazás felületén kívül megjelenő rövid, szöveges üzenetek, melyeket a felhasználó értesítésére használjuk. Meghatározott esemény hatására vagy bizonyos időközönként állhat elő egy értesítés. A felhasználók interakcióba léphetnek a notification-nel, amivel az alkalmazásba való navigációt valósíthatjuk meg, vagy akár az értesítés felületére is ki tudunk vezetni alkalmazásfunkciókat

Szükséges a *Plugin.LocalNotification* csomag telepítése. Az Android project-be külön is telepíteni kell.

```
Tools - NuGet Package Manager – Browse - Plugin.LocalNotification
```

NotificationRequest-nél a következő adatok megadása hasznos, amit pl a SendNotification metódusnál használhatunk pl. a gombkattintás eseményre:

```
BadgeNumber = 1,
Description = "PoC_V7",
Title = "Notification",
NotificationId = 1111,
NotifyTime = DateTime.Now.AddSeconds(5)
```

Az Android projektnél szükséges hozzáadni a *using Plugin.LocalNotification*;, illetve a *MainActivity.cs*-nél az *OnCreate* metódusnál a következőt:

```
NotificationCenter.CreateNotificationChannel();
```

4.8. REST API integráció

A REST API-hoz[24] való csatlakozás könnyedén megvalósítható Xamarin alatt (beépített HttpClient osztály segítségével)[3], csak az adatok konvertálásához van szükségünk kiegészítő csomagra. Ez a csomag NewtonSoft.Json[17][18]. A telepítéshez a következő lépések szükségesek:

```
Tools - NuGet Package Manager – Browse - NewtonSoft.Json
```

Az adatok kiolvasásnak érdekében szükséges osztályt, osztályokat a megfelelő mezőkkel el kell készíteni, illetve szükség van ezen adatok tárolásához egy megfelelő típusra. Ezek elkészítése az első lépések között kell, hogy legyenek.

A kapcsolat előzetes kialakításához érdemes egy osztály kialakítani (Pl. *RestApiClient*) és elvégezni a következő beállításokat. Használjuk a *client = new HttpClient();* megoldást. Fontos lehet egy várakozási idő megadása is, de ez természetesen nem szükséges.

```
client.Timeout = new TimeSpan(0, 0, 15);
```

Az URL megadása után

```
var url = type.GetDescription();
(Description(@"https://api.carbonintensity.org.uk/intensity/factors")] Factor,)
```

egy kérés küldése a feladatunk.

```
var response = client.GetAsync(url);
```

A kérésre érkezett választ például a következő módon kaphatjuk meg:

```
var responseString = response.Result.Content.ReadAsStringAsync();
```

Sikeres válasz után már használható a telepített csomag és az *JsonConvert.DeserializeObject* metódus segítségével átalakíthatjuk az adatokat a számunkra megfelelően.

```
switch (type)
 {case GetOperationEnum.Factor:
  result=JsonConvert.DeserializeObject<Model.Factor>(responseString.Result);break;
 }
```

A szükséges oldalon, az alábbi módon tudjuk elérni az API-nkat.

```
Client.RestApiClient apiClient = new Client.RestApiClient();
```

Alkalmazásunkban egy enum segítségével történik meg a két végpont megkülönböztetése.

```
var result = apiClient.GetOperation(Client.RestApiClient.GetOperationEnum.Factor);
```

A kapott Result eredmény alapján már könnyen feldolgozható a szükséges adatsor pl:

```
Model.Factor factor = (result as Model.Factor);
factor.Data[0].Biomass
```

Összegzés

Cikkünkben a három mobil fejlesztési lehetőséget mutattunk be az általunk meghatározott követelmények alapján. Bemutattuk az Android és iOS natív fejlesztést a jelenleg elérhető legmodernebb megoldásokkal, továbbá bemutatásra került egy keresztplatform megoldás is, a Xamarin. Az általunk meghatározott követelmények kielégítéséhez mind a három megoldást egyaránt alkalmasnak ítéltük meg, annyival kiegészítve, hogy szükséges volt harmadik féltől származó csomagok felhasználása is. Amennyiben mind a két mobil platformra (Android, iOS) tervezzük az alkalmazásunkat létrehozni, abban az esetben jó választás lehet (a felsorolt követelmények mellett) a Xamarin cross-platform fejlesztési irány is, egyéb esetben mindig az adott platform hivatalos megoldását érdemes választani. Fontos megjegyezni, hogy az alkalmazásboltok által megkövetelt további irányelvek, megszorítások nem lettek figyelembe véve, amely nagyban megnövelheti a cross-platform fejlesztési időt.

Köszönetnyilvánítás

Az írás szerzői köszönetet mondanak a projektben résztvevő intézmények - Nádor Rendszerház Kft., Controlsoft Automatika Szolgáltató Kft., Neumann János Egyetem GAMF Műszaki és Informatikai Kar - kollégáinak. Köszönettel tartozunk a kutatás támogatásáért, amely az “Ipar 4.0 moduláris felépítésű ipari csomagológép fejlesztése integrált adatelemzéssel és mesterséges intelligenciára épülő optimalizálással, hibaelemzéssel 2020-1.1.2-PIACI-KFI-2020-00062" pályázat keretében valósult meg. A projekt a Magyar Állam és az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával, a Széchenyi 2020 program keretében valósul meg.

Irodalomjegyzék

[1] Ekler Péter, Fehér Marcell, Forstner Bertalan, Kelényi Imre: Android-alapú szoftverfejlesztés. Szak kiadó. ISBN: 978-963-9863-27-9, 2012.
[2] Dan Hermes, Dr. Nima Mazloumi: Building Xamarin.Forms Mobile Apps Using XAML Mobile Cross-Platform XAML and Xamarin.Forms Fundamentals, Library of Congress Control Number: 2019930581, 2019, ISBN-13 (electronic): 978-1-4842-4030-4
[3] Daniel Hindrikes, Johan Karlsson: Xamarin.Forms Projects Second Edition, Packt Publishing Ltd., 2020.Június, ISBN 978-1-83921-005-1
[4] Ockert J. du Preez: Visual Studio 2019 In Depth: Discover and make use of the powerful features of the Visual Studio 2019 IDE to develop better and faster mobile, web, and desktop applications, BPB Publications, 2019, ISBN-13: 978-9389328325
[5] Reiter István: C# programozás lépésről lépésre, Jedlik Oktatási Stúdió Bt., 2018, ISBN: 9786155012174
[6] Ed Snider: Mastering Xamarin.Forms Third Edition, Packt Publishing Ltd, 2019.December, ISBN 978-1-83921-338-0
[7] Android hivatalos fejlesztői oldal: https://developer.android.com
[8] Android Studio fejlesztőkörnyezet: https://developer.android.com/studio
[9] Apple hivatalos fejlesztői oldal: https://developer.apple.com/
[10] Apple dokumentációk: https://developer.apple.com/documentation
[11] Apple design dokumentáció: https://developer.apple.com/design
[12] Create a Notification: https://developer.android.com/training/notify-user/build-notification
[13] Layers of the iOS architecture: https://subscription.packtpub.com/book/application_development/9781849691307/1/ch01lvl1sec12/layers-of-the-ios-architecture
[14] ListViews:https://docs.microsoft.com/en-us/xamarin/android/user-interface/layouts/list-view/
[15] Microcharts.Forms : https://www.c-sharpcorner.com/article/xamarin-forms-microcharts-app/
[16] MPAndroidChart: https://github.com/PhilJay/MPAndroidChart
[17] NewtonSoft.Json dokumentáció: https://www.newtonsoft.com/json/help/html/SerializingJSON.htm
[18] NewtonSoft.Json példa: https://github.com/JamesNK/Newtonsoft.Json
[19] Notification: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/local-notifications
[20] Notifications Overview: https://developer.android.com/guide/topics/ui/notifiers/notifications
[21] Notifacation példa: https://github.com/thudugala/Plugin.LocalNotification
[22] PieChartSwiftUI 1.0.1: https://cocoapods.org/pods/PieChartSwiftUI
[23] Quicktype: https://app.quicktype.io/
[24] Rest API https://docs.microsoft.com/en-us/learn/modules/consume-rest-services/3-consume-rest-service-with-httpclient
[25] Retrofit – A type-safe HTTP client for Android and Java: http://square.github.io/retrofit
[26] Scheduling local notifications with SwiftUI: hhttps://www.hackingwithswift.com/books/ios-swiftui/scheduling-local-notifications
[27] Sending and receiving Codable data with URLSession and SwiftUI: https://www.hackingwithswift.com/books/ios-swiftui/sending-and-receiving-codable-data-with-urlsession-and-swiftui
[28] SwiftUI dokumentáció: https://developer.apple.com/xcode/swiftui/
[29] Syncfusion.Xamarin.SfChart : https://help.syncfusion.com/xamarin/charts/getting-started
[30] Xamarin https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin
[31] Xamarin hivatalos oldal: https://dotnet.microsoft.com/apps/xamarin
[32] Xamarin telepítése https://docs.microsoft.com/en-us/xamarin/get-started/installation/windows
[33] Xcode fejlesztőkörnyezet: https://developer.apple.com/download/release/
[34] Kotlin Native Interoperability: https://kotlinlang.org/docs/native-overview.html#interoperability
[35] Wiertel, P., & Maria Skublewska-Paszkowska: Comparative analysis of UIKit and SwiftUI frameworks in iOS system. Journal of Computer Sciences Institute, 20, 170-174., 2021. DOI: 10.35784/jcsi.2662
[36] Cahill B.: Introducing SwiftUI. In: UI Design for iOS App Development. Apress, Berkeley, CA. 2021. DOI: 10.1007/978-1-4842-6449-2_1
[37] Kumar Vishal, Ajay Shriram Kushwaha: Mobile Application Development Research Based on Xamarin Platform, 2018 4th International Conference on Computing Sciences (ICCS), 14 January 2019. DOI: 10.1109/ICCS.2018.00027
[38] Jianye Liu, Jiankun Yu: Research on Development of Android Applications, 2011 4th International Conference on Intelligent Networks and Intelligent Systems, 2011. DOI: 10.1109/ICINIS.2011.40
[39] Ayu Bandu Retnomurti, Nurmala Hendrawaty, Leni Tiwiyanti: Development of android-based protadroid application in pronunciation practice learning for undergraduate students, Vol 7, No 2, 2019. DOI: 10.25134/erjee.v7i2.1721