
**Budapesti Műszaki és Gazdaságtudományi Egyetem**
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Simon Dorina Kíra

# ANDROID ALAPÚ SZOFTVERFEJLESZTÉS

Hallgatói Alkalmazás

KONZULENS

Dr. Ekler Péter

BUDAPEST, 2021

# Tartalomjegyzék

# Összefoglaló

Manapság az egyik legalapvetőbb eszköz, amelyre szinte minden embernek szüksége van, az a mobiltelefon. Fő funkciója még mindig az, hogy telefonáljunk vele, de egyre nagyobb teret hódítanak a mobilalkalmazások. Egy problémára rengeteg applikációt lehet már találni bármely App Store kínálatában, viszont ezek között általában alig van különbség – vegyük példának a hallgatóknak szánt appokat. Bőven lehet válogatni, hogy melyik alkalmazást szeretnénk, de az alap funkciók ugyanazok: tárgyak listázása, órarend megtekintése, házi feladatok számon tartása stb. Nem véletlenül ezek az alap funkciók, hiszen ezekre van a legtöbb hallgatónak szüksége, de kevés alkalmazás nyújt ennél sokkal többet. Sőt, kevés olyan alkalmazás van, amely külön a tanárok számára is nyújt hasonló megoldásokat.

Ezt a problémát alapul véve eldöntöttem, hogy elkészítek egy olyan alkalmazást, amely minden olyan funkciót tartalmaz, ami például egy egyetemi hallgatónak sokat segíthet a tanulás és a félévek teljesítése során, vagy akár egy előadónak az órák lebonyolításában. Az alap funkciók mellett több, általam ismert és használt alkalmazás tulajdonságait vettem bele a projektembe, mint megvalósítandó célt. Ezekről a funkciókról és a megvalósításukról ad egy részletesebb képet ez a dokumentum.

# 1 Bevezetés

## 1.1 A feladat értelmezése

A feladatom egy olyan hallgatói alkalmazás elkészítése volt, amellyel alapvetően a diák és a tanár számon tudja tartani a kurzusait, a diák a kurzushoz tartozó tanárait és teendőit. Mindezek tárolására az Amazon felhő alapú tárhelyét vettem igénybe.

### 1.1.1 A megvalósítandó funkciók

Mivel az alkalmazásnak kétféle felhasználója lehet (hallgató és előadó), a hozzájuk tartozó adatokat is meg kell tudni különböztetni a szerepek szerint. Ennek megoldására az alkalmazásomat úgy építettem fel, hogy regisztrációhoz és belépéshez legyen kötve, ezáltal csak hitelesített felhasználók férnek hozzá az adatbázisban tárolt adatokhoz, így a saját, felvett kurzusaikhoz is.

A másik alapvető funkció maguknak az adatoknak a tárolása: a felhasználók neve, szerepe, a hozzájuk tartozó kurzusok, diákok esetében a teendők is. Viszont nem csak a saját kurzusaikat tudják elérni az egyes felhasználók, hanem a tárgy felvételekor az összes, adatbázisban tárolt kurzus a rendelkezésükre áll, hogy abból tudjanak választani.

### 1.1.2 A fejlesztés folyamata

A dokumentum további részeiben bemutatom, hogy milyen alkalmazások inspirálták az ötletemet, milyen szolgáltatások állnak a háttérben, és részletesen leírom, hogy az egyes funkciókat hogyan sikerült megvalósítanom.

# 2 Irodalomkutatás, technológiák, hasonló alkotások bemutatása

## 2.1 Hasonló alkalmazások

A fejlesztés megkezdése előtt célszerű volt körülnéznem az elérhető, hasonló funkciókat megvalósító alkalmazások között. Az első ilyen applikáció, amit számba vettem, a hivatalos Neptun app [1]. Első ránézésre már rossz jel volt, hogy 1000 értékelés 1,6-os eredményt adott az 5-ből. A kommentekbe beleolvasva a fő hibája a megbízhatatlanság volt. Sokan panaszkodtak, hogy nem elérhető, nem működik, nem engedi belépni stb.

A következő alkalmazás, amit találtam, a Neptun Lite [2]. Ez egy hallgató által fejlesztett applikáció, röviden a hivatalos Neptun app javított változata. Erről már jobb véleménnyel voltak a használók, 183 értékelésből 4,4-es átlagot kapott az 5-ből. Az app bejelentkezés után lekéri Neptun-ból a felhasználó adatait, üzeneteit, órarendjét, kurzusait, vizsgáit, a félévekre lebontott eredményeit, és az időszakokat. Az órarend nézet több egész napos feladat esetén kicsit kaotikus, egyébként tökéletesen látszik minden. Tárgyaknál lehet böngészni az előző félévekben felvett tárgyakat is, ugyanez a helyzet a vizsgákkal is, színekkel szépen el van különítve, hogy melyik volt sikeres, sikertelen, vagy éppen melyik van még hátra. A féléves adatoknál meg lehet nézni félévekre lebontva diagramokon a felvett és teljesített krediteket, és ebből látható egy összesített diagram is. A féléves átlagokat is jól láthatóan lehet követni egy vonaldiagramon. Az időszakoknál lehet követni, hogy éppen milyen időszak van (szorgalmi, vizsgajelentkezési stb.). Nagyon tetszett ez az applikáció, sok inspirációt tudtam belőle szerezni.

Még két alkalmazás van hátra, melyeket már régebb óta ismertem, és nagyon tetszettek a funkciók, melyeket megvalósítottak. Az egyik ilyen a Kahoot! [3], ezzel több előadáson is találkoztam már. Össze lehet állítani vele egy kérdéssort, majd egy pin kódot megosztva akár böngészőből is lehet csatlakozni és válaszolni a kvízkérdésekre. Szerintem ez a tanulást nagyban segíti, mert legtöbbször jó helyezésért az előadók plusz pontokat adnak, és ez motiválja a hallgatókat arra, hogy jobban figyeljenek előadáson. Másik előnye, hogy össze lehet benne szedni az órai anyag fontosabb részeit, ezáltal a

hallgató, ha csak kitölti, már jobban megjegyzi a lényeges dolgokat. Egy ehhez hasonló funkciót szívesen integráltam volna az appomba.

Az utolsó alkalmazás a Quizlet [4], melyet egyszer vizsgára való készüléskor használtam, és sokat segített a vizsga sikeres teljesítésében. Ezzel az appal tanulókártyákat lehet magunknak készíteni, megosztani másokkal, és az elkészített kártyákat többször végig lehet pörgetni, miközben az alkalmazás figyeli, hogy mennyire sikeresen tanultad meg a kártyákon lévő tételeket, szavakat, igaz-hamis kérdéseket stb. Ehhez hasonló funkciót is terveztem megvalósítani az alkalmazásomban, mert szerintem ez is nagyon sokat tud segíteni a tanulásban.

A feladat megvalósítása előtt több, számomra új modern technológiával is mélyebben megismerkedtem, ezeket fogom a következőkben részletesebben bemutatni.

## 2.2 Amazon Amplify

Az egész alkalmazás hátterében az Amazon Amplify [5] backend áll, amely rengeteg funkciót kínál a fejlesztőknek. Ezen funkciók közül ugyan csak hármat vettem igénybe, a fejlesztést mégis sokszor nagyon megkönnyítette, hogy ezeket nem nekem kellett létrehoznom. A backend-del kellett szinte a legtöbbet foglalkoznom, hiszen ez volt az első alkalom, hogy igénybe vettem egy felhő alapú szolgáltatást, és mivel ez még viszonylag fiatal szolgáltatás, a legtöbbször nem is találtam egyből megoldást az interneten a felmerülő problémáimra.

### 2.2.1 Amazon Cognito

Az Amazon Cognito (később csak Cognito néven hivatkozok rá) alapvetően az hitelesítést, jogosultságkezelést és a felhasználók kezelését végzi. A Cognito része a User Pool, itt lehet beállítani pl. a regisztrációkor a jelszó követelményeit. A Cognito-val való kommunikáció könnyen ment, hiszen csak elküldtem a felhasználó nevét és jelszavát, a hitelesítést már intézte is helyettem. Ilyen egyszerűen ment a felhasználók regisztrálása is, mely során a megadott email-re a regisztráló a Cognito-n keresztül kap egy visszaigazoló kódot is, ezáltal lesz megerősített a regisztráció és hitelesített a felhasználó.

### 2.2.2 AWS AppSync, DynamoDB

A maradék két funkciót, melyet az Amazon-tól vettem igénybe, együtt fogom tárgyalni, hiszen a használatuk is egyszerre történt. Az AWS AppSync szolgáltatja és

kezeli az API-t, melyet a fejlesztés során használtam. Ez az API a DynamoDB nevű szolgáltatásra épül, mely az adatbázisomat kezeli. DynamoDB-n keresztül láthatom a felhőben tárolt tábláimat, és minden adatot, melyet tartalmaznak. Az API sémáját GraphQL nyelven írtam, ez egy egyszerűen használható lekérdező nyelv. Az API sémában lehet megadni, hogy egyes táblákhoz mely felhasználók férhetnek hozzá, itt kötöttem ki, hogy csak hitelesített felhasználók lássák az adatokat, valamely táblánál pedig kizárólag csak az adatot létrehozó felhasználó férhet hozzá a saját információihoz.

## 2.3 Room

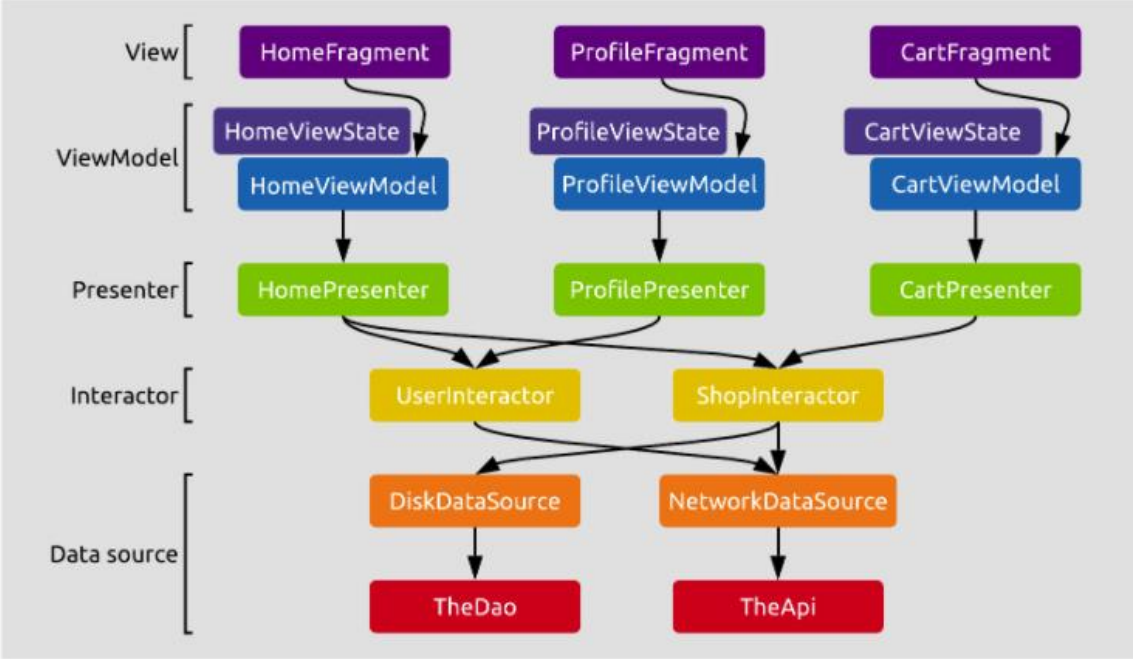
A következő szolgáltatás, mely az alkalmazásom alapját képezi, az Android Jetpack könyvtárhoz tartozó Room [6]. Ezt már használtam előző félévemben, így most nagyobb rutinnal tudtam beleépíteni az alkalmazásomba. A Room az adatok lokális tárolására, hozzáférésére szolgál, mindezt SQLite alapokra építve. A legfőbb oka annak, hogy felhő mellett lokálisan is tárolok néhány adatot az volt, hogy internetkapcsolat nélkül is működőképesek legyenek az alap funkciók.

# 3 Tervezés

## 3.1 Architektúra

A fejlesztés megkezdése előtti fontos lépés volt, hogy kiválasszam az architektúrát, melyre az alkalmazásomat tudom építeni. Miután kicsit jobban megismerkedtem vele, a választásom a RainbowCake [7] architektúrára esett. Ennek legfőbb oka az állapotkezelés volt, tetszett, hogy egy teljesen külön rétegben különül el ennek megoldása.

A RainbowCake architektúra összesen öt rétegből áll: maga a nézet (Fragment vagy Activity), a nézet állapotát kezelő réteg (ViewState és ViewModell segítségével), a megjelenítő réteg (ez felelős az alsóbb rétegekből érkező adatok átalakításáért a képernyőnek megfelelő formátumba), az üzleti logikát megvalósító réteg (ha kell, számításokat végez, adatokat aggregál stb.), és végül az adatkezelő réteg (ez szolgáltatja magukat az adatokat).



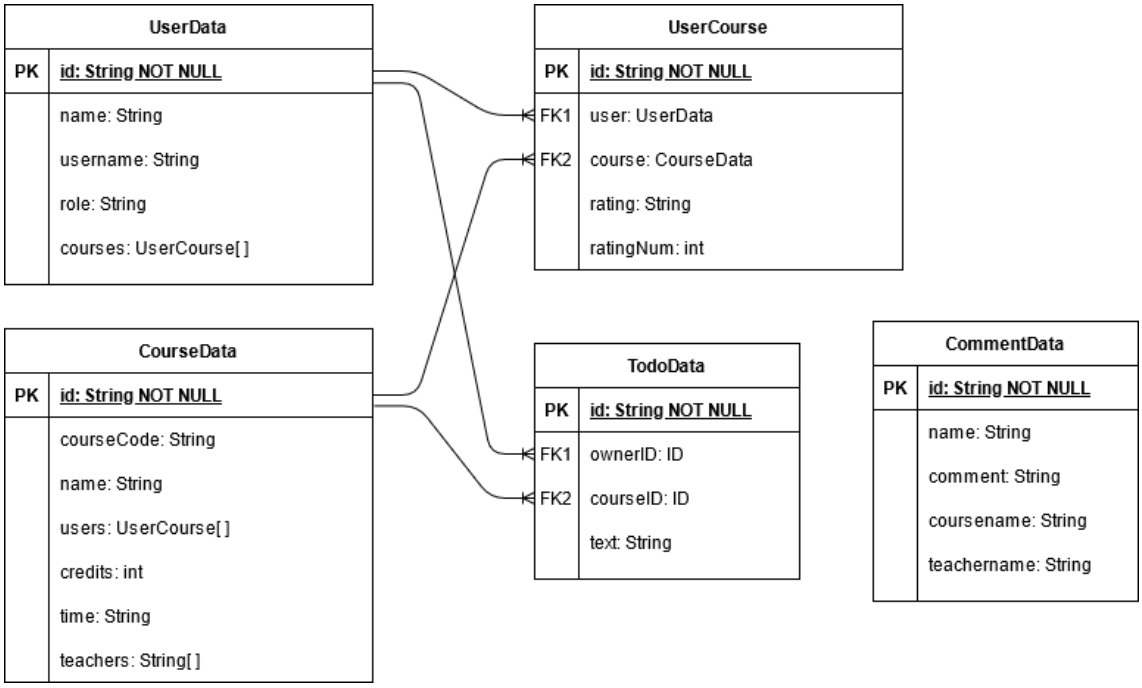

**1. ábra A RainbowCake architektúra felépítése**

Fontos még kiemelni, hogy az architektúra a ViewModel rétegben használ Coroutine-okat. A coroutine-ok aszinkron, nem blokkoló kód írását teszik lehetővé azzal, hogy a feladatokat külön szálon hajtja végre, így nem blokkolja a felhasználói felületet.

Az architektúrát a fejlesztés során sikerült követni, és nagyban megkönnyítette a dolgom, hogy elkülönülve hoztam létre az egyes osztályokat. Egy ilyen nagy projektben már sokat tud számítani, hogy mennyire könnyű eligazodni a fájlok között. A Presenter réteg viszont nálam feleslegesnek bizonyult, így az architektúrám végül egy réteggel vékonyabb lett.

## 3.2 Az adatbázis felépítése

A tervezési fázis következő lépése az adatbázis megtervezése volt. Ezen a terven a fejlesztés során többször kellett változtatnom, ahogy egyre jobban belemélyedtem a feladat megoldásába. A végső állapot a 2. ábrán látható.




**2. ábra Az adatbázis felépítése**

### 3.2.1 A táblák részletezése

A UserData táblában tárolom a felhasználó adatait: a teljes nevét (name), a felhasználónevét (username), a szerepét (role – lehet diák vagy tanár), és a hozzá tartó kurzusokat (courses).

A CourseData tábla a kurzusok információit tartalmazza: a kurzus kódját (courseCode), nevét, a kurzushoz tartozó felhasználókat (users), a kreditet (credits), hogy milyen időpontban van (time – pl. Hétfő 8:15-10:00), és hogy mely tanárok tartják (teachers).

A UserData és a CourseData összekötésére szolgál a UserCourse tábla. Az egymáshoz tartozó kurzus és felhasználó mellett tárolja még, hogy tanár esetében az adott tanár az adott kurzusra milyen (rating) és mennyi (ratingNum) értékelést kapott.

A TodoData a feljegyzések tárolására szolgál, melyeket egy-egy tárgyhoz (courseID) lehet felvenni. Végül pedig a CommentData, amely a kommentek adatainak tárolására szolgál, mint a kommentelő neve (name), a komment szövege (comment), hogy melyik tárgyhoz (coursename) és tanárhoz (teachername) tartozik a komment.

# 4 A funkciók megvalósítása

## 4.1 Bejelentkezés, regisztráció

Amikor a felhasználó először nyitja meg az alkalmazást, a bejelentkező képernyő fogadja. Ezen a képernyőn tovább navigálva tud regisztrálni, ehhez meg kell adnia egy felhasználónevet, jelszót, email címet, majd az emailben kapott hitelesítő kódot. Mindez mögött az Amazon Cognito áll, mellyel az adatkezelő rétegen keresztül kommunikálok, ezen funkciók a `NetworkBackend` objektumban kerültek megvalósításra. Itt történik még az Amplify inicializálása, ennek során az alkalmazás indulásakor feliratkozok a hitelesítő eseményre, amely figyeli, hogy melyik felhasználó maradt utoljára bejelentkezve. Amennyiben van bejelentkezett felhasználó, a bejelentkezési képernyőt automatikusan átugrom.

A bejelentkezésnél, amint a felhasználó rákattint a `Sign In` feliratú gombra, elkezdem továbbítani a felhasználónevet és a jelszót a backend felé (3. ábra). A `SignInViewModel` coroutine-t indít, és átadja az adatokat a `UserInteractor` `signIn` függvényének. A `UserInteractor` továbbítja az adatokat a `NetworkDataSource` felé, itt már az üzleti logikai rétegben járunk, majd meghívva a `NetworkBackend` `signIn` függvényét el is érkezett a kérés a backend-hez. A backend visszaad egy üres sztringet, ha minden rendben történt, egyébként a hibaüzenettel tér vissza.

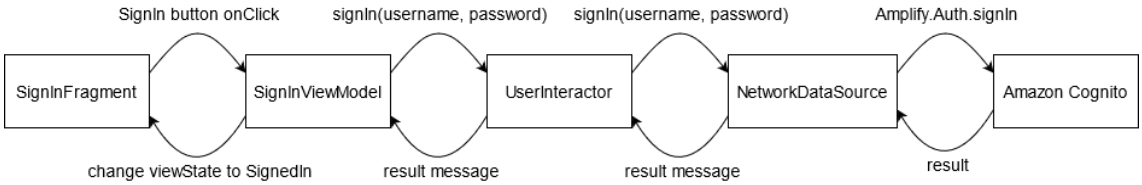
```
suspend fun signIn(username: String, password: String) : String {
    Log.i(TAG, "Initiate Signin Sequence")
    return suspendCoroutine { continuation ->
        Amplify.Auth.signIn(
            username,
            password,
            {result: AuthSignInResult ->
                Log.i(TAG, result.toString())
                continuation.resume("")
            },
            {error: AuthException ->
                Log.e(TAG, error.toString())
                continuation.resume(error.message.toString())
            }
        )
    }
}
```

Visszaérve a `UserInteractor` osztályhoz megnézem, hogy sikerült-e a kérés (tehát üres sztringet kaptam-e vissza), ha igen, a `UserDataSource`-on keresztül beállítom a

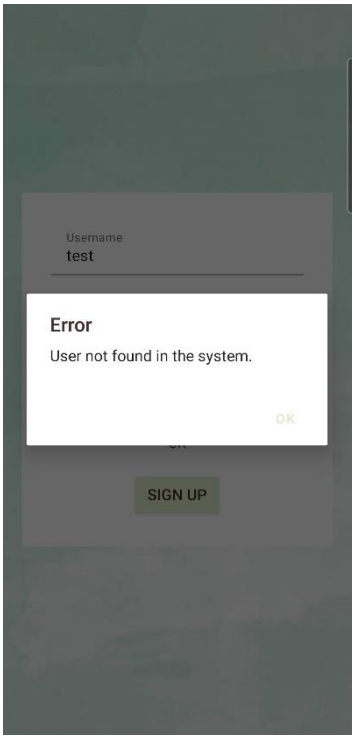
lokális felhasználót (`setLocalUser`), majd `CourseDataSource`-on keresztül kiürítem az előző felhasználó adatait a kurzusokat tároló lokális táblából (`deleteCourses`).

```
suspend fun signIn(username: String, password: String) : String {
    val success = networkDataSource.signIn(username, password)
    if(success == ""){
        userDatasource.setLocalUser(username)
        courseDataSource.deleteCourses()
    }
    return success
}
```

Az üzenet végül visszaér a `SignInViewModel`-hez, ha sikerült a bejelentkezés, akkor az állapot `SignedIn` lesz, majd elindítom a következő kérést – már a `UserDataBackend` felé -, hogy megkapjam a felhőben tárolt adatbázisból a felhasználó adatait. Amennyiben valami hibaüzenetet kaptam vissza, azt jelzem a felhasználó felé egy `AlertDialog` megjelenítésével, és a hibaüzenet kiírásával (4. ábra).




**3. ábra A bejelentkezés folyamata**




**4. ábra Hibaüzenet bejelentkezéskor**

### 4.1.1 A felhasználó adatainak lekérdezése

A felhasználó adatait a `UserDataBackend` `getUserData` függvényével valósítom meg, az alábbi módon:

```
suspend fun getUserData(username: String?) : User? {
    var user : User?
    return suspendCoroutine { continuation ->
        Amplify.API.query(
          ModelQuery.list(UserData::class.java,
          UserData.USERNAME.eq(username!!)),
            { response ->
                if(response.data != null){
                    for(userData in response.data){
                        user = User.from(userData)
                        currentUser = userData
                        continuation.resume(user)
                    }
                } else if(response.hasErrors()){
                    continuation.resume(null)
                }
            },
            { error ->
                continuation.resume(null)
            }
        )
    }
}
```

A függvény paraméterként kapja a keresendő felhasználónevet, ez az adatbázisban egyedi kulcsnak számít. A `ModelQuery.list` lekérdezés kikeresi a `UserData` táblából azon adatokat (ebben az esetben csak egyes számban az adatot), melyek `USERNAME` attribútuma megegyezik a paraméterben átadott `username` értékével. Ha van találat (tehát `response.data != null`), akkor visszaadom az átkonvertált adatot (`user`), egyébként `null`-lal térek vissza.

Minden olyan esetben, amikor az API-tól kérek le adatot, a függvény felépítése kisebb különbségekkel, de ugyanaz, mint a felhasználó adatainak lekérésénél.

## 4.2 Tárgy felvétele

Tárgy felvétele esetén a felhasználó választhat az adatbázisban már szereplő kurzusok közül, vagy létrehozhat egy újat, ha az még nem létezik.

### 4.2.1 Választás listából

Már létező kurzus felvétele esetén első lépésként kilistázom az adatbázisból a tárgyak adatait, ezt is API lekérdezéssel teszem, mint a felhasználó adatainak lekérésénél.

Az eredményeket megjelenítem egy kattintható, görgethető és kereshető listában. Ha a felhasználó kiválaszt egy kurzust, egy felugró ablakban figyelmeztetem, hogy ezzel hozzá fogja adni a tárgyat a sajátjaihoz, és amennyiben folytatni szeretné, a tárgyat hozzáadom a lokális adatbázishoz (5. ábra).

### 4.2.2 Új tárgy létrehozása

Új tárgy létrehozásakor a felhasználónak meg kell adni a tárgy nevét, idejét, kódját és kreditjét. Ezeket az adatokat elmentem egy `Course` típusú változóba, majd az adatkezelési rétegben hozzáadom mind a lokális, mind a felhőben tárolt adatbázishoz.

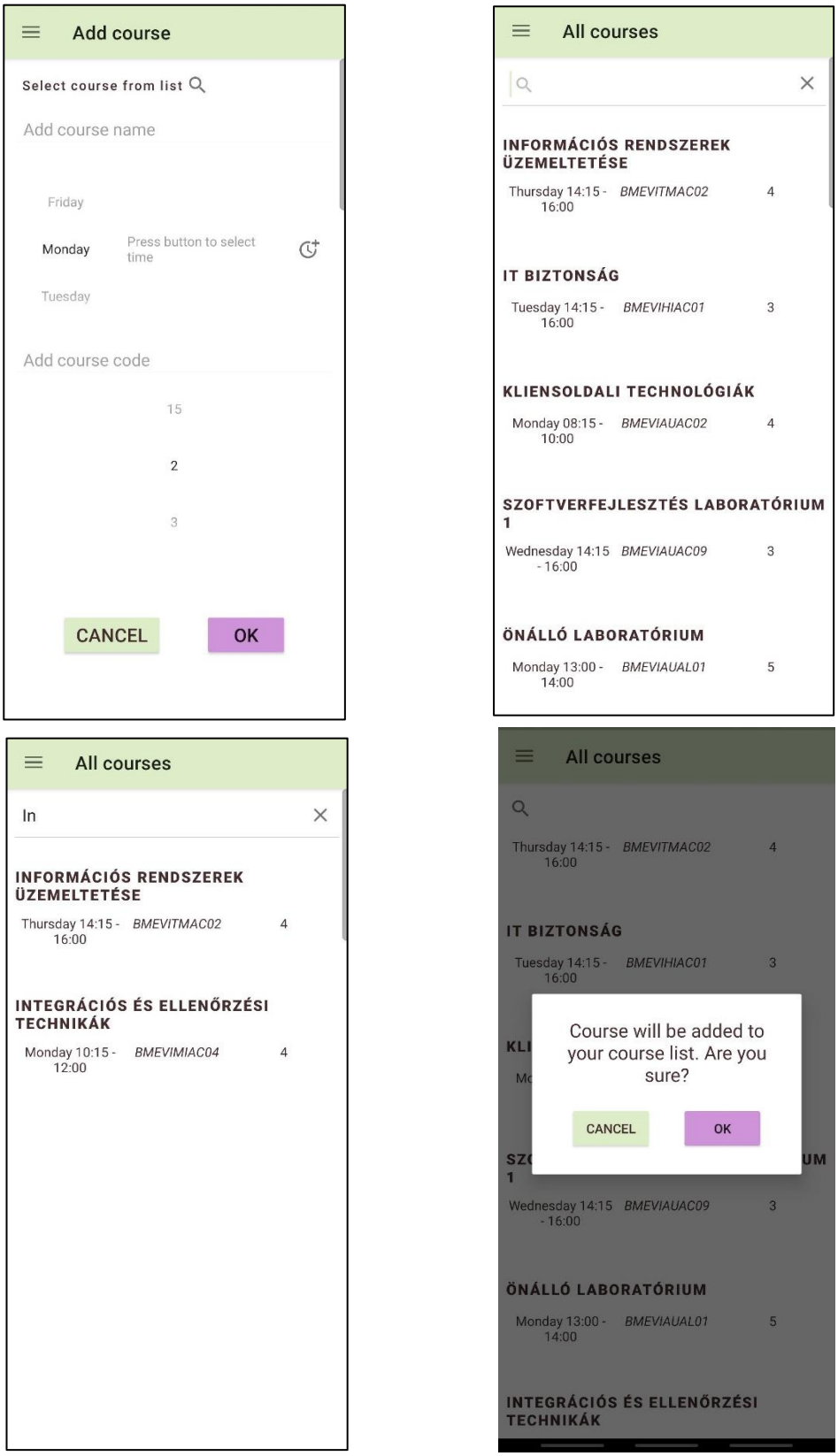
```
suspend fun createCourse(c: Course) : Boolean {
    Log.i(TAG, "Creating course")
    return suspendCoroutine { continuation ->
        Amplify.API.mutate(
            ModelMutation.create(c.data),
            { response ->
                Log.i(TAG, "Course created")
                if (response.hasErrors()) {
                    Log.e(TAG, response.errors.first().message)
                    continuation.resume(false)
                } else {
                    Log.i(TAG, "Created course with name: "+
                           response.data.name)
                    addCourse(Course.from(response.data))
                    addCourseToLocal(Course.from(response.data))
                    continuation.resume(true)
                }
            },
            { error ->
                Log.e(TAG, "Course create failed", error)
                continuation.resume(false)
            }
        )
    }
}
```

A függvény visszatérési értéke igaz vagy hamis, attól függően, hogy sikerült-e létrehozni az új tárgyat. Ha igen, el is mentem az összes tárgyhoz (`addCourse`), illetve a felhasználó tárgyaihoz (`addCourseToLocal`) egy-egy lokális változóba.

A lokális adatbázishoz való hozzáadást a `CourseDataSource` osztályban valósítom meg.

```
suspend fun addCourseToLocal(course: Course){
    UserDataBackend.addUserCourseData(course)
    courseDao.insert(LocalCourseEntity(course.id, course.courseCode,
          course.name, course.credits, course.time,
          UserDataBackend.currentUser.id,
          ListConverter().listToString(course.teachers)))
}
```

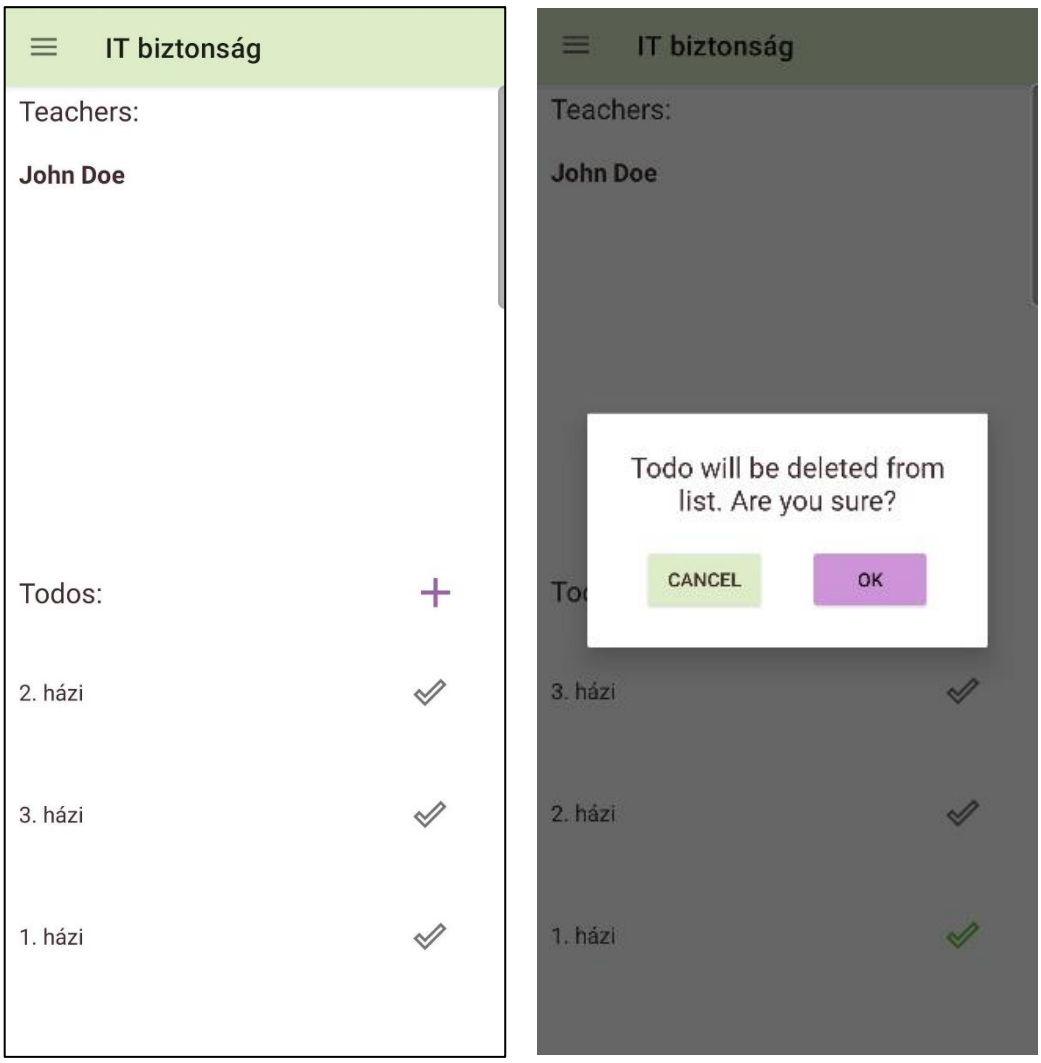
A `CourseDao` segítségével tudok adatokat lekérni és hozzáadni a lokális adatbázishoz, amely `LocalCourseEntity` példányokat tárol.




**5. ábra Tárgy felvétele**

## 4.3 Tárgy részletező nézete

Amikor a felhasználó rákattint egyre a felvett tárgyai közül, akkor a kurzus részletező képernyője jelenik meg. Itt listázom ki a tárgyhoz tartozó tanárokat és feljegyzéseket. Innen a felhasználó tovább navigálhat a tanár értékeléseit megjelenítő képernyőre, vagy hozzáadhat, illetve kipipálhat egy teendőt. Kipipálás esetén a teendő törlődni fog az adatbázisból, ezt jelzem is a felhasználó felé egy felugró dialógusablakban.




**6. ábra Kurzus részletező nézet és teendő kipipálása**

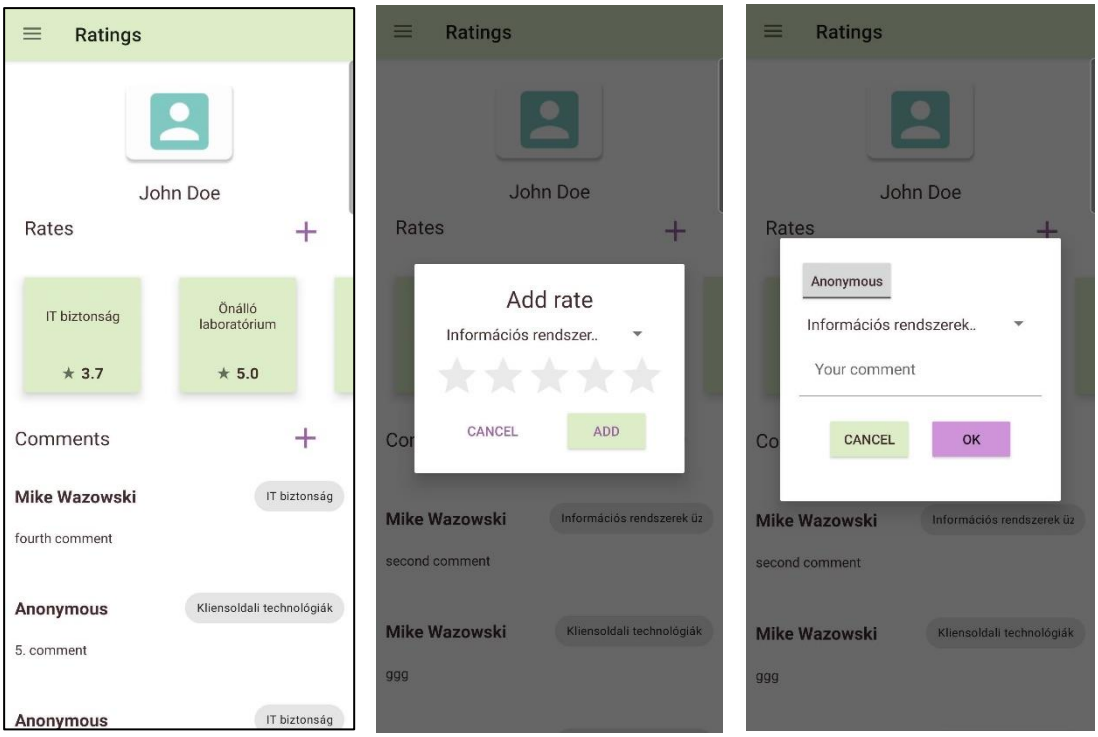
A teendő törlését szintén a `UserDataBackend`-ben valósítom meg:

```
fun deleteTodo(todo: Int) : List<Todo>? {
    val todos = _todos.value
    if(todos != null){
        val td = todos.elementAt(todo)
        todos.removeAt(todo)
        _todos.notifyObserver()
        Amplify.API.mutate(
            ModelMutation.delete(td.data),
            { response ->
                if (response.hasErrors()) {
                    Log.e(TAG, response.errors.first().message)
                } else {
                    Log.i(TAG, "Deleted Todo data: ${response.data}")
                }
            },
            { error ->
                Log.e(TAG, "Todo delete failed: ", error)
            }
        )
        return todos
    }
    return todos
}
```

A teendőket egy `RecyclerView` alapú listában jelenítem meg, és kattintáskor a teendő indexét kapom meg. A visszatérési érték az így megmaradt teendők listája, vagy ha a törlés sikertelen volt, akkor a változatlan lista.

## 4.4 Tanár véleményezése

A tanár értékeléseit megjelenítő képernyő csak a hallgató szerepkörű felhasználók számára elérhető. A felhasználó tudja az adott tanárt az adott kurzuson egytől-ötig terjedő skálán csillagozni, vagy tud megjegyzést írni a tárgyhoz névvel vagy anonim módon.




**7. ábra Tanár értékelései, értékelés hozzáadása, kommentelés**

# 5 Eredmények

A lelkesedésemnek hála elég sok funkciót találtam ki, amelyet meg akartam valósítani, de az időm nagy részét elvitte a backend-del való ismerkedés, így a terveimnek csak egy töredékét tudtam megvalósítani. A fejlesztés során voltak hullámvölgyek, de összességében élveztem a szoftver megvalósítását, és továbbra is a kedvenceim közé fog tartozni az Android operációs rendszer és a Kotlin nyelv.

## 5.1 Továbbfejlesztési lehetőségek

Az első fejlesztési lehetőség, amelyet ki szeretnék emelni, az a tanárhoz tartozó funkciók megvalósítása. A hallgatói szerephez sikerült elkészítenem plusz funkcióként a véleményezést és a kurzus részletes nézetet, a tanár szerepre viszont már nem maradt időm, hogy azt is ki tudjam dolgozni. Ott is szeretném egyszer megvalósítani, hogy legyen a kurzusokhoz részletező nézet, amely annyiban különbözne a hallgatóétól, hogy a tanár a kurzushoz tartozó órákat látná, ahhoz tudna megjegyzéseket írni, teendőket felvenni, és ott lennének listázva a kvízek, amiket annak az órának az anyagához készített.

A hallgatói szerephez még szeretném egyszer megvalósítani, hogy a Neptun Lite-hoz hasonlóan, itt is lehessen követni a féléves haladást, legyenek statisztikák, hogy pl. hány kreditet teljesített. Ezek mellett szeretném még, hogy a Quizlet-hez hasonlóan tanulókártyákat lehessen létrehozni és megosztani. A tanári szerephez tartozna még a kvízek létrehozására egy külön képernyő.

Mindkét szerephez szeretném még, hogy meg lehessen nézni az adott napra, hogy milyen órák, teendők várnak a felhasználóra.

# 6 Hivatkozások

[1] Play Store: Neptun app

https://play.google.com/store/apps/details?id=hu.SDA.NeptunMobile.Android

[2] Play Store: Neptun Lite

https://play.google.com/store/apps/details?id=com.infinitx.neptunlite

[3] Play Store: Kahoot!

https://play.google.com/store/apps/details?id=no.mobitroll.kahoot.android

[4] Play Store: Quizlet

https://play.google.com/store/apps/details?id=com.quizlet.quizletandroi

[5] Amazon Amplify

https://aws.amazon.com/amplify/

[6] Room

https://developer.android.com/training/data-storage/room

[7] RainbowCake

https://rainbowcake.dev/