

TDK-dolgozat

Maric Viktor

2023

Coachy és Reality Tasks: Az Apple kiterjesztett valóság keretrendszerének
implementálása mobilalkalmazásokba

Coachy and Reality Tasks: The implementation of Apple's augmented reality
frameworks into mobile applications

Készítette: Maric Viktor

Budapesti Corvinus Egyetem, Gazdaságtudományi Kar

Gazdaságinformatika 4. évfolyam

Témavezető: Dr. Bagó Péter, egyetemi adjunktus

Kézirat lezárása: 2022. április 19.

Absztrakt

A kiterjesztett valóság még egy kevésbé kifejtett technológia, ami nagy növekedési potenciál előtt áll. Az okostelefonok processzorának és kameráinak folyamatos fejlődése teret ad arra, hogy egyre elérhetőbb legyen ennek a technológiának mindennapi használata. Jelenleg az Apple rendelkezik a világ legnagyobb kiterjesztett valóság platformjával és a legnagyobb piaci kapitalizációval, így most és a jövőben is meghatározó szerepet vállalhat ennek az elterjedésében. Az Apple kiterjesztett valóság üzleti modellje elsősorban B2B üzleti formát követ, de későbbiekben a B2C is erősödhet mellette egy esetleges új hardver kiadásával.

A dolgozatban arra keresi a szerző a választ két általa fejlesztett alkalmazáson keresztül, hogy az Apple kiterjesztett valóság fejlesztői keretrendszereit hogyan lehet implementálni mobilalkalmazásokba, ezek milyen lehetőségeket nyújtanak. A két alkalmazás elérhető az App Store-on keresztül, nevük: Reality Tasks és Coachy. A Reality Tasks egy projekt- és feladatkezelő alkalmazás, amely segítségével a felhasználó elhelyezheti a létrehozott teendőit cetli formájában egy falra. A Coachy pedig egy olyan személyi edző, amelynek a háromdimenziós modelljét a felhasználó életre keltheti a kiterjesztett valóság segítségével.

A kutatás először a kiterjesztett valóságot, a felhasználási területeit és a gazdasági helyzetét mutatja be. Ezt követően részletezi az Apple kiterjesztett valóság B2B értékajánlatát. Röviden ismerteti az Xcode fejlesztői környezetet és a Swift programnyelvet. Utána bemutatja a szerző az általa fejlesztett két alkalmazás sajátosságait és technológiai felépítését. Részletezi a kiterjesztett valóság programozói keretrendszerek – a RealityKitnek és az ARKitnek - implementálását és megjelenését ezekben a szoftverekben. Végül pedig összegzi a dolgozatot, amely során jól látszik, hogy az Apple ezt a számára stratégiaileg fontos technológiának implementálását viszonylag könnyűvé tette a fejlesztők számára. A RealityKit keretrendszer többek között lehetőséget nyújt háromdimenziós modellek generálására, importálására, animációkra és élethű textúrák betöltésére. Az ARKit segítségével pedig ezeket a modelleket az eszköz kameráján keresztül el lehet helyezni, szinte automatikusan kezeli az árnyékolást, a fényérzékelést, az embereknek meg a tárgyaknak a felismerését. A mobilalkalmazások pedig részben ezzel is differenciálni tudják magukat hosszú távon a versenytársakkal szemben.

Tartalomjegyzék

1. Bevezetés	1
2. Kiterjesztett valóság	2
2.1. Definiálás	2
2.2. Felhasználási területek.....	2
2.3. Gazdasági helyzete és jövője	3
3. Apple AR B2B értékajánlata.....	5
3.1. AR keretrendszerek.....	5
3.2. App Store.....	6
3.3. Megbízható hardver.....	7
3.4. Potenciális okoszeműveg.....	7
4. Technológiai háttér	9
4.1. Swift	9
4.2. Xcode.....	10
4.3. Keretrendszerek.....	10
5. AR keretrendszerek implementálása.....	12
5.1. Reality Tasks	12
5.1.1. Az alkalmazás funkciói	12
5.1.2. Struktúra modell	14
5.1.3. RealityKit implementálása	14
5.1.4. ARKit implementálása	16
5.2. Coachy	19
5.2.1. Az alkalmazás funkciói	19
5.2.2. RealityKit implementálása	20
5.2.3. ARKit implementálása.....	22
5.2.4. Megjelenítés virtuális háromdimenzióban	23
6. Összegzés	25
Irodalomjegyzék.....	27

Mellékletek	31
1. melléklet.....	31
2. melléklet.....	38
3. melléklet.....	44

Ábrajegyzék

1. ÁBRA - SAJÁT SZERKESZTÉS, FORRÁS: (PWC, 2019B) ALAPJÁN	3
2. ÁBRA - SAJÁT SZERKESZTÉS, FORRÁS: (APPLE, 2022A) ALAPJÁN	5
3. ÁBRA - SAJÁT SZERKESZTÉS, FORRÁS: (IQBAL, 2022) ALAPJÁN	6
4. ÁBRA - TECHNOLÓGIA ÉLETCIKLUS GÖRBE, FORRÁS: (BAUER ET AL., 2006, 347-363 OLD.) ALAPJÁN	8
5. ÁBRA - SAJÁT SZERKESZTÉS	12
6. ÁBRA - SAJÁT SZERKESZTÉS	13
7. ÁBRA - SAJÁT SZERKESZTÉS	13
8. ÁBRA - SAJÁT SZERKESZTÉS	14
9. ÁBRA - SAJÁT SZERKESZTÉS	15
10. ÁBRA - SAJÁT SZERKESZTÉS	15
11. ÁBRA - SAJÁT SZERKESZTÉS	16
12. ÁBRA - SAJÁT SZERKESZTÉS	16
13. ÁBRA - SAJÁT SZERKESZTÉS	17
14. ÁBRA - SAJÁT SZERKESZTÉS	17
15. ÁBRA - SAJÁT SZERKESZTÉS	18
16. ÁBRA - SAJÁT SZERKESZTÉS	18
17. ÁBRA - SAJÁT SZERKESZTÉS	19
18. ÁBRA - SAJÁT SZERKESZTÉS	20
19. ÁBRA - SAJÁT SZERKESZTÉS	21
20. ÁBRA - SAJÁT SZERKESZTÉS	22
21. ÁBRA - SAJÁT SZERKESZTÉS	22
22. ÁBRA - SAJÁT SZERKESZTÉS	23
23. ÁBRA - SAJÁT SZERKESZTÉS	24
24. ÁBRA - SAJÁT SZERKESZTÉS	24

1. Bevezetés

A kiterjesztett valóság még egy kevésbé kifejlett technológia, ami előtt jelentős növekedési potenciál áll (PWC, 2019a). Az okostelefonok processzorának és kameráinak folyamatos fejlődése teret ad arra, hogy egyre elérhetőbb legyen ennek a technológiának mindennapi használata (Apple, 2022a). Jelenleg főként az okostelefonokon keresztül lehet használni ezt a technológiát, azonban a jövőben egy okos szemüveg piacra dobása ugródeszkaként szolgálhat a további elterjedéshez (Seitz, 2021).

Mivel már évek óta foglalkozom 3D modellezéssel és mobilalkalmazások fejlesztésével, így kifejezetten érdekes számomra ez a technológia, hiszen ezt a két területet ötvözi. A kutatás során azt fogom vizsgálni két szoftverfejlesztési projekt által, hogy hogyan lehet implementálni az Apple kiterjesztett valóság keretrendszerait mobil- és tablet alkalmazásokba. A két szoftver, amelyeken keresztül ezt bemutatom: az egyik egy projekt- és feladatkezelő alkalmazás, a Reality Tasks; a másik pedig egy fitness alkalmazás, a Coachy. Mindkét alkalmazás elérhető az App Store-on.

A Reality Tasks fejlesztése mellett azért döntöttem, mivel kevés olyan alkalmazást találtam, amely modern kinézetű, megbízhatóan működik az Apple platformjain, és egyszerre biztosítja a kanban táblának és a teendő listának az elkészítését. Ebben a szoftverben a kiterjesztett valóság úgy jelenik meg, hogy a felhasználó elhelyezheti egy falra a teendőit cetli formájában, és ezáltal figyelheti meg a még el nem végzett feladatait.

A Coachy célja pedig egy olyan virtuális edző kialakítása, aki valóban egy személyes partnernek tűnik. Ezt úgy oldja meg az alkalmazás, hogy a személyre szabott edzéseken kívül a felhasználó életre keltheti Coachy háromdimenziós modelljét kiterjesztett valóságban.

Először a kiterjesztett valóságot, a felhasználási területeit és a gazdasági hatását foglalom össze röviden. Ezt követően részletezem az Apple értékajánlatát, ezzel megindokolva, hogy miért ennek a vállalatnak a platformját választottam a többihez képest. A technológiai háttérrel is írok, ami magába foglalja az alkalmazások fejlesztéséhez szükséges programozási nyelvet és környezetet. A továbbiakban pedig a két szoftveren keresztül mutatom be, hogy hogyan lehet az Apple kiterjesztett valóság fejlesztői keretrendszerait implementálni, illetve feltárom az általuk nyújtott lehetőségeket. Azért két alkalmazáson keresztül részletezem ezt, mivel az egyik a valóságban elhelyezett háromdimenziós objektumot csak kódból generálja, a másikon pedig egy külső szoftverből származik. Végül pedig összegzem a dolgozat eredményeit, és kitérek további lehetséges kutatási kérdésekre.

2. Kiterjesztett valóság

Az első kiterjesztett valóság eszköz Ivan Sutherland nevéhez fűződik, ami egy 1968-ban kifejlesztett fejre rögzíthető kijelző volt. Az első kiterjesztett valósághoz köthető fogyasztóknak szánt termék pedig 2008-ban jelent meg, amivel a BMW egyik autóját lehetett megtekinteni egy kijelzőn keresztül. Igazán akkor lett először népszerű a technológia, amikor a Pokemon GO alkalmazást publikálták a mobilalkalmazás piacán 2016-ban. Felhasználóinak száma hamar elérte a több tíz milliót. (Javornik, 2016)

2.1. Definiálás

A kiterjesztett valóság (vagy AR) digitális információkat, tárgyakat vagy médiát mutat be a valós világban mobileszközön vagy szemüvegen keresztül (PwC, 2019a). Különböző eszközökön – mint például a tableten és a telefonon – levő AR alkalmazások lehetővé teszik, hogy magával ragadó élményeket nyújtsanak, azzal, hogy zökkenőmentesen vegyítik a virtuális objektumokat a valós világgal, azt az illúziót keltve, hogy ezek az objektumok valóban léteznek (Apple, 2022a).

Könnyű összekeverni a kiterjesztett valóságot (AR) és a virtuális valóságot (VR). A fő különbség azonban az, hogy amíg a VR teljesen kizárja a valóságot virtuális világgal, addig az AR a valós világot bővíti virtuális elemekkel. (Tulane University, 2021)

2.2. Felhasználási területek

A kiterjesztett valóság széles felhasználási területet kínál, hiszen jelen van többek között az egészségügyben, a munkahelyi képzésekben, oktatásban, kereskedelemben, illetve a termék- és szolgáltatásfejlesztésben is. A technológia segítheti például a raktári munkatársakat abban, hogy hatékonyabban mozgósítsák a csomagokat, de abban is segíthet, hogy felpróbálhassunk otthon olyan ruhákat virtuálisan, amiket meg szeretnénk vásárolni. (PwC, 2019a, 9-13. old.)

Tim Cook elmondása szerint nagyon bizakodó a technológiával kapcsolatban, mivel természetében horizontális, azaz számos területen lehet alkalmazni. Külön kiemelte a játékipart, az egészségügyet és az oktatást. (Saïd Business School, University of Oxford, 2018)

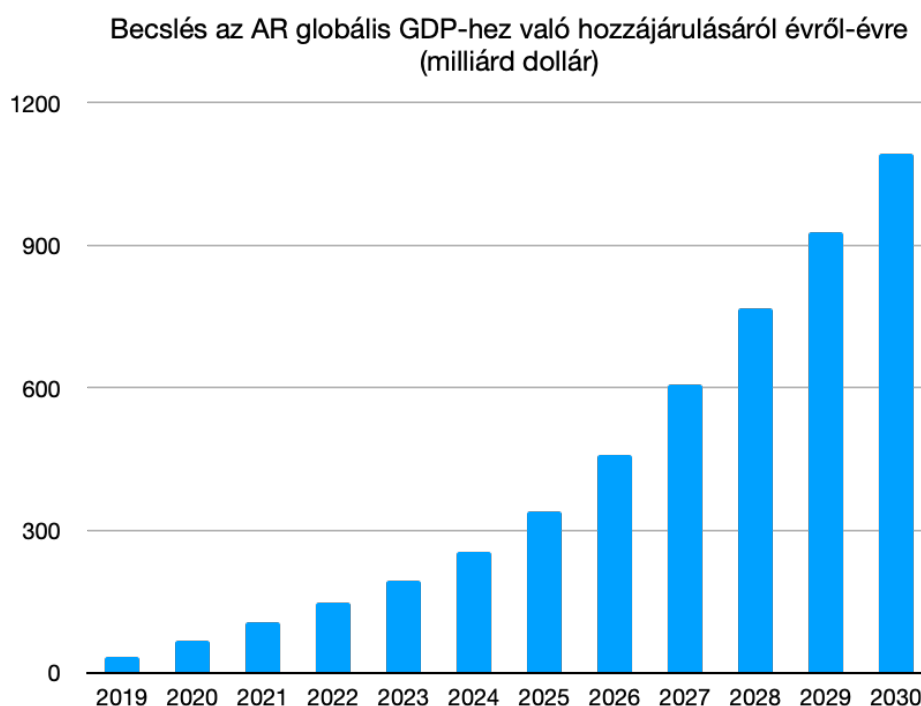
A technológia a hadiiparban is jelen van, amelyre példa az amerikai F-35-ös vadászgép 400 ezer dolláros AR sisakja (Cohen, 2021). A sisak kijelzője a pilóták számára repülési és taktikai információkat biztosít a helyzetfelismerés és biztonság érdekében (Airforce Technology, 2021).

Az Apple példáján pedig keresztül jól látszik, hogy egy népszerű felhasználási terület a mobilalkalmazásokon keresztüli implementálás, melyekre néhány kedvelt példa az IKEA Place, a Hot Lava és a Museum Alive. Az IKEA Place alkalmazás segítségével el lehet helyezni a bútorokat az otthonunkban, és megtekinteni, hogy hogyan illenek be az adott szobába. A Hot Lava nevezetű játékkal a nappalinkat akadálypályává lehet varázsolni, a karakterrel pedig a valódi és virtuális tárgyak között ugrálva lehet teljesíteni a pályákat. A Museum Alive alkalmazással pedig ősi élőlényeket helyezhetünk el nappalinkban David Attenborough narrációjával. (Apple, 2022a)

2.3. Gazdasági helyzete és jövője

Tim Cook szerint a fejlett országok lakóinak a közeljövőben napi szinten lesz kiterjesztett valósághoz köthető élménye. A VR szerint nem lesz olyan elterjedt a mindennapokban, mint az AR, de az oktatás és a játékok területén kedvelt eszköz lesz. Véleménye alapján visszaneézve csodálkozni fogunk, hogy hogyan tudtunk AR nélkül élni, a telefonhoz hasonlóan. (Techlife Coach, 2016)

Maga a technológia 2019-ben 33 milliárd dollárt adott a globális GDP-hez, de 2025-ben ez már 338 milliárd, 2030-ban pedig 1092 milliárd dollár lehet. Jól látszik az 1. ábrán is a hozzájárulásnak rohamos növekedése. A 2019-es 33 milliárdhoz képest a 2030-as 1092 milliárd dollár nagyjából 33-szoros növekedést jelenthet. (PwC, 2019b)



1. ábra - saját szerkesztés, forrás: (PwC, 2019b) alapján

Az Apple évek óta nyújt be szabadalmakat az AR és a virtuális valóság (VR) technológiájára, felvásárolt több startupot is, és olyan AR-szakértőket alkalmaz, akik más vállalatok AR stratégiai kulcsemberei voltak. (Perry, 2021)

A Morgan Stanley csapata szerint az Apple belépése a szemüvegpiacra minden résztvevő számára változást jelent majd, ahogy a technológia népszerűsödik. Azt is kiemelték, hogy ha sikerül a technikai akadályokat legyőzni - mint az egész nap működő akkumulátor, 5G, vonzó kinézetű szemüveg kialakítása - akkor képes lesz első dobásra piacvezető lenni az okos szemüvegek piacán. (Seitz, 2021)

Tehát nagy potenciál van ebben a viszonylag új technológiában, amelynek ilyen nagymértékű elterjedését most még nehéz látni. A jövőbe tekintve pedig véleményem szerint izgalmas lesz, ha nem egy telefont magunk elé tartva kell megélni az AR élményt, hanem például egy okoszemüveg formájában lehetünk élvezői ennek a technológiának. Külön érdekes az is, hogy az élet szinte bármelyik területén el lehet képzelni kisebb vagy nagyobb interakciók formájában a felhasználását.

3. Apple AR B2B értékajánlata

Az Apple-t 1976-ban alapította Steve Jobs mellett Ronald Gerald Wayne és Stephen G. Wozniak. A fő termékei közé tartozik az iPhone, a Mac, az iPad, az AirPods, az Apple TV és az Apple Watch. Ezek mellett pedig még felhő- és streaming szolgáltatásokat is nyújt felhasználóinak. Nagyjából 150000 munkavállalót foglalkoztat és Tim Cook a CEO-ja. (Forbes, 2021)

Ahogy a 2. ábra is szemlélteti, az Apple hármass értékajánlatot biztosít a fejlesztők felé: modern fejlesztői eszközökkel tudnak alkalmazást fejleszteni csúcstechnológiás mobilkészülékekre, amiket publikálhatnak a világ legnagyobb AR platformjára, az App Store-ra. (Apple, 2022a)

Az Apple B2B AR értékajánlata



2. ábra - saját szerkesztés, forrás: (Apple, 2022a) alapján

3.1. AR keretrendszerek

Az Apple két keretrendszert – az ARKit-et és a RealityKit-et - biztosítja ingyen a vállalatoknak ahhoz, hogy egyszerűen tudjanak kiterjesztett valóság élményeket fejleszteni mobilalkalmazások formájában. Emellett, kifejlesztettek egy Reality Composer nevezetű szoftvert is, amely segítségével 3D modelleket és animációkat lehet létrehozni. (Developer Apple, 2022a)

Az ARKit egy 2017-ben publikált technológia, amely leegyszerűsíti az AR alkalmazások építését azzal, hogy nyomon követi az eszköz mozgását, feltérképezi a környezetet az eszköz kameráján keresztül, és elhelyezi a 3D modellt a feltérképezett környezetben. (Developer Apple, 2017)

A RealityKit pedig az ARKit-re épült, azonban első sorban a 3D modellek megjelenítésére fejlesztették ki, hiszen ezeket valósághűen jeleníti meg fotórealisztikus renderelés és animációk segítségével. A keretrendszert 2019-ben mutatták be először. (Developer Apple, 2019a)

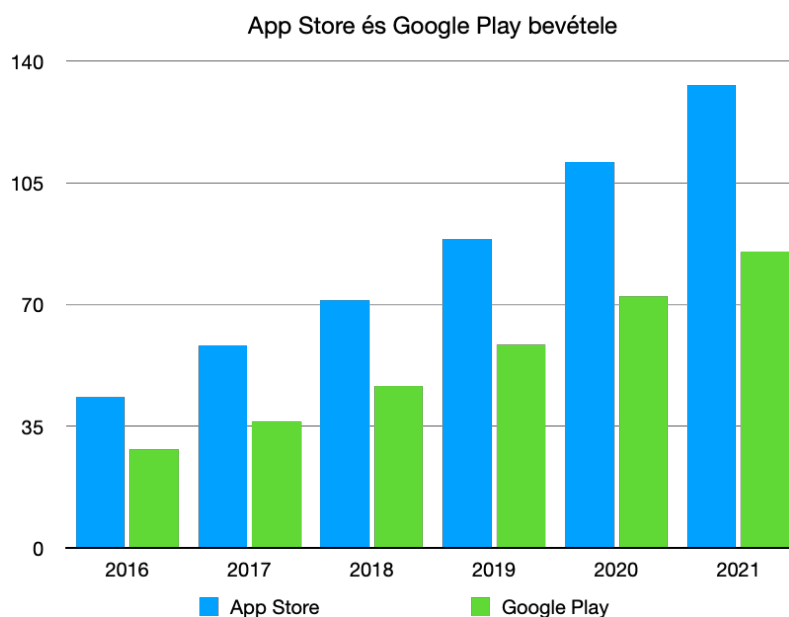
Emellett maga a vállalat fejleszti ki a szoftvert és a hardvert, így képes a kettőt jól integrálni és optimalizálni (Apple, 2022a). Erre példa az, hogy RealityKit és az ARKit közvetlen érik el a processzort, így nagyon jól kihasználják a hardver által nyújtott lehetőségeket (Developer Apple, 2022b).

3.2. App Store

A vállalat a világ legnagyobb AR platformjával rendelkezik az App Store által, amely segítségével több százmillió felhasználót lehet elérni (Apple, 2022a). Tehát már most jelentős platformot biztosít az áruházzal a vállalatoknak ahhoz, hogy ezt a viszonylag új technológiát rengeteg embernek elérhetővé tehesék.

Az App Store-on belül van egy dedikált rész csak arra, ahol kiemelt alkalmazásokat osztanak meg a felhasználókkal. Ez azt jelenti, hogy az Apple munkatársai folyamatosan keresik azokat az alkalmazásokat, amelyek innovatívok és jó élményt nyújtanak a felhasználóknak. Ezeket a minőségi alkalmazásokat úgy emelik ki az App Store-ban, hogy vállalatoknak nem kell fizetni érte. Ezzel segíthetik a kiterjesztett valóságot alkalmazó vállalatokat is abban, hogy az alkalmazásaik még több emberhez jussanak el. (Developer Apple, 2022c)

Az App Store mellett a másik nagy mobilalkalmazás platform a Google Play, azonban - ahogy a 3. ábra is szemlélteti - az App Store alkalmazásai 2021-ban nagyjából másfélszer annyi bevételre tettek szert, mint a Google Play alkalmazások. (Iqbal, 2022)



3. ábra - saját szerkesztés, forrás: (Iqbal, 2022) alapján

A vállalatok az Apple Developer Program segítségével publikálhatják alkalmazásaikat az App Store-ra. A tagság magában foglalja az alkalmazások fejlesztéséhez és terjesztéséhez szükséges összes eszközt, erőforrást és támogatást, beleértve a béta szoftverekhez való hozzáférést, a tesztelési eszközöket, az alkalmazáselemzést és még sok más. A tagság díja évenként 100 dollár. (Developer Apple, 2022d)

Az Apple az App Store-on letöltött alkalmazáson belüli vásárlásoknak a 15 vagy 30 százalékát kapja meg. Akkor 15%, ha az adott alkalmazás árbevétele nem haladja meg az egy millió dollárt évente, és akkor 30%, ha ezt az összeget meghaladja. (Apple, 2020a)

3.3. Megbízható hardver

A fejlett kamerák, kijelzők, mozgásérzékelők és az erős grafikus processzorok miatt pedig megbízhatóan működő telefonokra és táblagépekre készíthetik el az alkalmazásokat a külsős fejlesztők (Apple, 2022a). Tehát az AR programozóknak kevésbé kell aggódniuk a telefonok számítási kapacitásán és kameráik minőségén az alkalmazásfejlesztés során.

Az új telefonjai közül több olyan is van, amelyekben a kamerák mellett LiDAR szenzort is beépítettek. Ennek a szenzornak a lényege, hogy szinte azonnal feltérképezze a környezetet pontosan, és sötétben is jól működik. (Apple, 2020b)

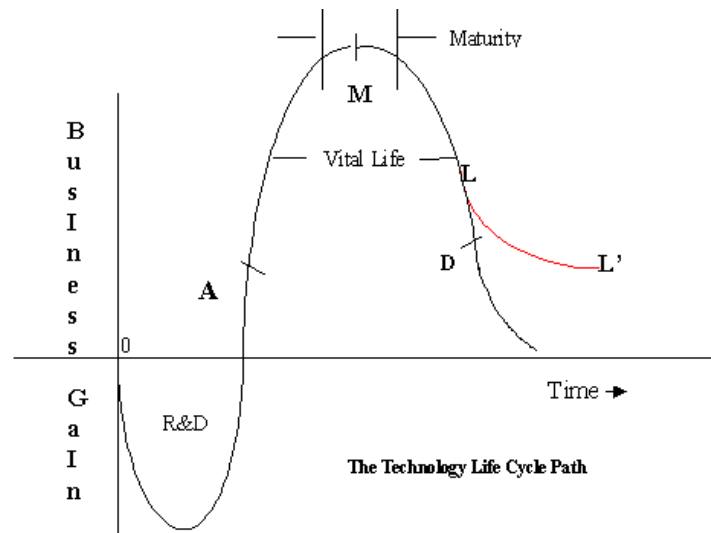
3.4. Potenciális okoszemüveg

Véleményem szerint érdemes egy potenciális új Apple hardverről, az okos szemüvegről is említést tenni, hiszen a vállalat több, mint 300 okos szemüveghez köthető szabadalommal rendelkezik (Patently Apple, 2022a). A Morgan Stanley csapata szerint a vállalat belépése az okoszemüveg piacra nagy változást jelenthet, hiszen hamar piacvezetővé válna (Seitz, 2021).

Az Apple múltjában sikeres új hardver bevezetésére példa az iPhone és az Apple Watch. Az iPhone megjelenését követő évben - azaz 2008-ban - már több, mint 11 millió darabot értékesített belőle (sec.gov, 2008, 41. old.). Az Apple Watch 2014-ben jelent meg, és az azt követő évben több, mint 12 millió darabot adtak el belőle, ezzel kétharmados részesedést szerezve az okosórák piacán rövid idő alatt (Canalys, 2016, 1. old.).

Szerintem érdemes egy távolabbi nézőpontból is megtekinteni a vállalat helyzetét. Ahhoz, hogy igazán jó kiterjesztett valóság élményben legyen részünk, kifejezetten fontos, hogy megfelelő kamerával, szenzorokkal és számítási kapacitással legyen felszerelve az eszköz (Apple, 2022a). Ezek a kamerák és szenzorok viszont megjelennek az Apple okoszemüveg (Patently Apple, 2022a), és az önvezető autó (Patently Apple, 2022b) szabadalmaiban is. Tehát

azzal, hogy a kamerákat, a szenzorokat és a processzorokat folyamatosan fejleszti a vállalat, nem csak jobb telefonokat és táblagépeket fog gyártani, hanem a potenciális okoszeműveg és önvezető autó piacra vezetését is elősegítheti.



4. ábra - Technológia életciklus görbe, forrás: (Bauer et al., 2006, 347-363 old.) alapján

Ahogy a 4. ábra is szemlélteti, egy technológiánál azok a vállalatok, amelyek még az életciklus feltörekvő fázisában csatlakoznak a piachoz, azoknak viszonylag hamar megtérül a befektetésük, stabil és gyors növekedés élvezői lehetnek (Bauer et al., 2006). A PwC (2019a) szerint még jelentős növekedés előtt áll a kiterjesztett valóság, a Morgan Stanley (Seitz, 2021) szerint pedig az Apple lehet a legmeghatározóbb szereplője a közeljövőben. Tehát ezért érdemes igazán most foglalkozni az Apple AR technológiáinak az implementálásával.

4. Technológiai háttér

Az AR keretrendszeren kívül a két bemutatott alkalmazáshoz más technológiák is szükségesek, amelyekről ebben a pontban írok röviden.

4.1. Swift

A Swift egy modern programozási nyelv, amellyel alkalmazásokat lehet fejleszteni az Apple platformjaira (Docs Swift, 2022a). A nyelvet már az alapjaitól kezdve úgy fejlesztették ki, hogy nagyon gyors és biztonságos legyen (Docs Swift, 2022a). 2014 augusztusában jelent meg az első verziója, azóta pedig az egyik leggyorsabban terjedő programozási nyelvnek számít (swift.org, 2022).

A Swift nagyon jó objektum-orientált programozásra, azonban a széleskörű felhasználású protokollokkal az első protokoll-orientált programozási nyelvnek számít. (Developer Apple, 2015)

A protokoll (*protocol*) többek között meghatározhat egy követelményt, tulajdonságot és metódusokat. A Swiftben három típus létezik: osztály (*class*), struktúra (*struct*) és enumeráció (*enum*). Ezek közül valamennyi megfeleltethet egy vagy több protokollnak. A már meglévő protokollokat pedig ki lehet egészíteni az *extension* kóddal. (Docs Swift, 2022b)

Az osztályok és az objektum-orientált programozás lehetővé teszi az absztrakciót, névteret, kibővíthetőséget, metódusok és tulajdonságok definiálását is. A protokollok struktúrákkal való kombinációja mindezt biztosítja és még ki is egészíti új funkciókkal. Pont ezért a protokollokat legtöbbször a struktúrák által származtatjuk. Az egyik nagy előnyük az, hogy amíg az objektum-orientált programozásban az osztály csak egy főosztályból származhat, addig a struktúrák bármennyi protokollból. Ez nagy rugalmasságot jelent az alkalmazások modelljének és az architektúrájának a kialakításában. Emellett a struktúrák az osztályokkal szemben nem referencia-, hanem érték típusúak. Ez azt jelenti, hogy a struktúrák nem hivatkoznak egy adott objektumra, hanem magát az adatot tárolják és a példányosítás során pedig másolják. Ezek miatt a struktúrák használata biztonságosabb és gyorsabb alkalmazást jelentenek a Swift nyelvben. Mindez nem azt jelenti, hogy az osztályok használata felesleges, hiszen az Apple több objektum-orientált keretrendszert is kialakított Objective-C-ben (a Swift elődjében), amiket még most is lehet használni Swift osztályokba beágyazva. Az újabb keretrendszerek viszont protokoll-orientált logikát követik, ami a fejlesztőket is arra ösztönzi, hogy osztályok helyett protokollal és struktúrákkal alakítsák ki az alkalmazásaikat. (Developer Apple, 2015)

4.2. Xcode

Az Xcode az Apple által fejlesztett programfejlesztői környezet, amely a macOS rendszeren érhető el ingyenesen. Ez segíti a programozókat abban, hogy fejlesszék, teszteljék és publikálják az alkalmazásaikat. (Developer Apple, 2022e)

A fejlesztői környezet többek között lefordítja az Apple által létrehozott Swift nyelvet és annak keretrendszereit. Ezeknek a segítségével lehet fejleszteni a vállalat operációs rendszereire, azaz az iOS-re, az iPadOS-re, a tvOS-re, a watchOS-re és a macOS-re. (Developer Apple, 2022e)

4.3. Keretrendszerek

Az Apple sok keretrendszert fejlesztett ki, amik közül azokról írok röviden, amelyeket a két alkalmazásban implementáltam.

A SwiftUI segítségével lehet a felhasználói felületet és annak viselkedését leprogramozni (Developer Apple, 2022f). 2019-ben publikálta a vállalat ezt a modern és deklaratív szintaxú keretrendszert (Developer Apple, 2019b).

A Combine kezeli az aszinkron történő eseményeket a SwiftUI-hoz hasonlóan deklaratív szintaxissal. (Developer Apple, 2022g)

A HealthKit lehetővé teszi, hogy az alkalmazások szinkronizáljanak adatokat az Apple Egészség alkalmazással. (Developer Apple, 2022h)

A már korábban említett RealityKit és ARKit felhasználásával lehet kiterjesztett valóság élményeket létrehozni.

A WatchKit segítségével lehet kezelni az Apple Watch okosórához tartozó háttérfeladatokat és kommunikációt az okostelefonnal. (Developer Apple, 2022i)

A Mac Catalyst lehetőséget nyújt arra, hogy az iPad alkalmazásokat könnyen a macOS-re szabjuk. (Developer Apple, 2022j)

Az EventKit keretrendszerrel lehet elérni a felhasználó által létrehozott emlékeztetőket és naptári eseményeket. (Developer Apple, 2022k)

A StoreKit segítségével lehet beépíteni alkalmazáson belüli vásárlásokat. (Developer Apple, 2022l)

A Core Data segít abban, hogy a felhasználó által létrehozott adatokat tárolja az eszközén, a CloudKit-tel integrálva pedig szinkronizálja az adatokat a felhasználó iCloud-jával (Developer Apple, 2022m). Az adatokat titkosítva tárolja az iCloud-on azért, hogy biztonságban legyenek a privát információk (Developer Apple, 2022n).

A Reality Tasks az adatok tárolására nem az iCloud-ot használja, hanem a Google Cloud Firestore-t. Ez egy NoSQL felhőalapú gyors és skálázható adatbázis, ami lehetővé teszi a valós idejű szinkronizációt az eszközök között (Firebase, 2022).

5. AR keretrendszerek implementálása

Az Apple két kiterjesztett valóság keretrendszere a RealityKit és az ARKit, amelyeknek implementálását fogom bemutatni két általam fejlesztett alkalmazáson keresztül. Egy rövid alkalmazás- és felhasználói felület bemutatását követően először a RealityKit, majd az ARKit felhasználásáról írok.

A két alkalmazás neve Reality Tasks és Coachy. A Reality Tasks egy olyan feladatkezelő alkalmazás, amely képes a létrehozott feladatokat megjeleníteni kiterjesztett valóságban. A Coachy pedig egy olyan fitness alkalmazás, amellyel a felhasználó a virtuális edző által elvégzett gyakorlatokat elhelyezheti az otthonában.

A Reality Tasks a háromdimenziós objektumokat teljes mértékben programkóddal generálja le és helyezi el. Gyakori azonban, hogy egy modellező szoftverben hozunk létre egy 3D-s objektumot vagy animációt, és azt helyezzük el egy kiterjesztett valóság alkalmazásban. Ezt az esetet pedig a Coachy-n keresztül mutatom be.

5.1. Reality Tasks

5.1.1. Az alkalmazás funkciói

A Reality Tasks egy modern projekt- és feladatkezelő alkalmazás, amely elérhető iOS, iPadOS és macOS rendszereken, ahogy az 5. ábrán is látható. Az egyik értékajánlata, hogy kombinálja a teendő listát és a kanban táblát.



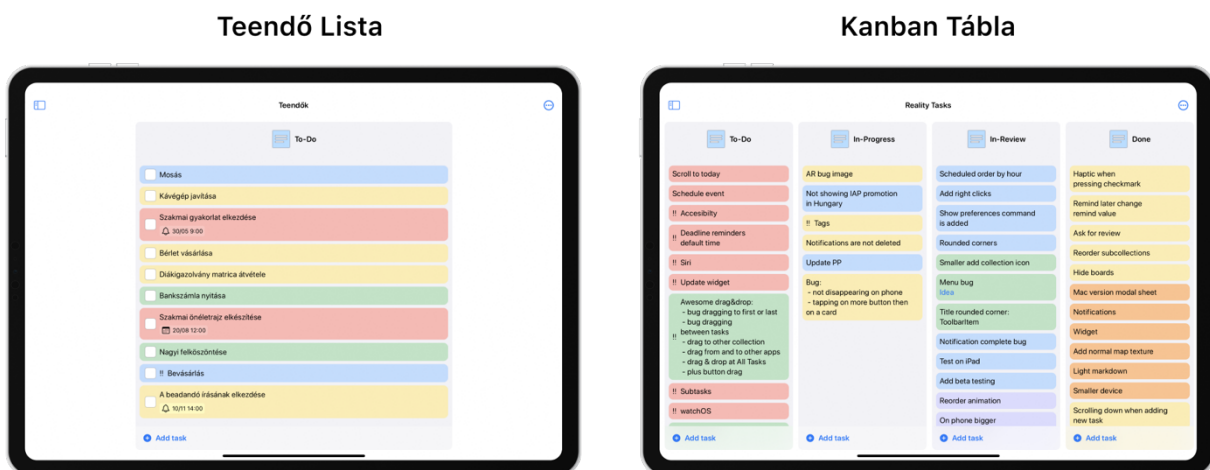
5. ábra - saját szerkesztés

A *Teendő Lista* (angolul To-do list) egy olyan eszköz, amely segít feladatainkat aszerint rendezni, hogy elvégeztük-e őket, vagy sem. (MindTools, N.A.)

A *Kanban Táblát* pedig az agilis projektmenedzsmentben arra használják, hogy a feladatokat hatékonyan oldják meg a csapatok. Ezt a táblát általában négy oszlopra osztják,

amelynek részei balról jobbra haladva a következők: teendő, folyamatban, tesztelés alatt és elvégzett. A feladatokat a csoporttagok a legelső oszlopban lévő teendőkhöz hozzák létre, és a feladat helyzetétől függően a következő oszlopba húzzák egészen addig, amíg az utolsó, elvégzett oszlopba nem kerül be. (Rehkopf, N.A.)

A Reality Tasks lehetőséget ad ilyen *Teendő Listának* és *Kanban Táblának* a létrehozására, ahogy a 6. ábrán is látható. A létrehozott feladatokhoz lehet leírást, szint, prioritást, emlékeztetőt, határidőt és kommenteket fűzni.



6. ábra - saját szerkesztés

Az alkalmazás a felhasználó által létrehozott *Teendő Listákat* és *Kanban Táblákat* le tudja képezni 3D modellé, és el tudja helyezni az eszközén lévő kamera segítségével egy falra. Az élethű digitális objektumok azt az illúziót keltik, hogy a létrehozott feladatok valóban egy a falon lévő táblára vannak elhelyezve cetlik formájában, ahogy a 7. ábra is szemlélteti.

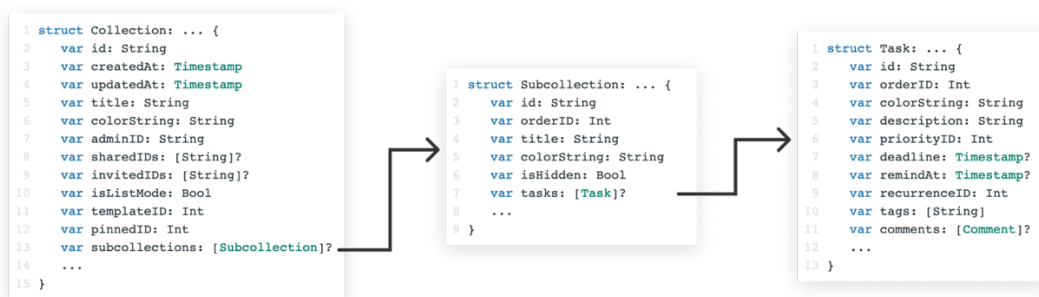


7. ábra - saját szerkesztés

5.1.2. Struktúra modell

A szoftver modelljének lényegi részét három struktúra adja: *Collection*, *Subcollection* és *Task*. A *Collection* egy gyűjteményt jelent, amelynek típusa lehet *Kanban Tábla* vagy *Teendő Lista*. A *Collection*-ön belül pedig vannak *Subcollection*-ök, azaz algyűjtemények. Egy *Kanban Tábla* gyűjtemény esetén négy algyűjtemény van (teendő, folyamatban, tesztelés alatt, kész), egy *Teendő Lista* esetén pedig kettő (teendő és kész). A *Subcollection*-ökön belül pedig vannak a *Task*-ok, azaz a feladatok.

Az alkalmazás modelljének struktúráját a 8. ábra szemlélteti. Bár ennek felépítése egy relációs adatbázishoz hasonló, azonban az adatok backendjét egy noSQL felhőalapú szolgáltatás, a Google Firestore biztosítja. Ennek köszönhetően, egy regisztrációs folyamatot követően a felhasználók adatait a szoftver a felhővel szinkronizálja, ami miatt elérhetik más eszközeiken is a létrehozott feladatokat a belépést követően.



8. ábra - saját szerkesztés

5.1.3. RealityKit implementálása

A cél az, hogy a felhasználó által létrehozott gyűjteményt, az azon belüli algyűjteményeket és a hozzá tartozó feladatokat a RealityKit keretrendszer leképezze 3D modellekké, majd az ARKit segítségével elhelyezze egy falra. Tehát az egész modellezés kódból történik, az alkalmazás nem használ külső 3D modellező szoftverből származó fájlt.

A Swift nyelv sajátosságait kihasználva a szoftver igyekszik mindezt modern módon megoldani. Az ismétlés elkerülése miatt az AR funkciók alapját egy protokoll biztosítja, aminek az alkalmazás modelljét képező struktúrái megfelelnek. A struktúrákat pedig egy osztályba beágyazva megjeleníti a felhasználó telefonján.

Ahogy a 9. ábrán látható, az említett protokoll neve *ARPresentable*, és csak egy függvényt tartalmaz, amelynek visszatérési értéke egy *Entity* osztály. Az *Entity* a RealityKit keretrendszer egy eleme, amely egy olyan 3D-s modell, ami elhelyezhető kiterjesztett valóságban. Ennek a protokollnak felel meg a *Collection*, a *Subcollection* és a *Task* struktúra.

```

1 protocol ARPresentable {
2     /// Generál egy Entity-t, ami egy megjeleníthető 3D modell.
3     func generateEntity() -> Entity
4 }

```

9. ábra - saját szerkesztés

A *Task* struktúra *ARPresentable* protokollnak való megfeleltetésének rövid verzióját, és ennek vizuális illusztrációját az 10. ábra szemlélteti. A *generateEntity()* függvény generál egy lapos és színes téglatestet, amely úgy néz ki mint egy kis papírlap vagy cetli. Ehhez még egy papír textúrát is hozzárendel, amely a létrehozott cetlin kis érdes felületet jelenít meg ahhoz, hogy még élethűbb legyen. Ezt követően pedig a cetli közepére helyezi a feladatnak a leírását. Ezzel tehát az alkalmazás legenerál 3D modellé egy feladatot.

Programkód

```

1 extension Task: ARPresentable {
2     func generateEntity() -> Entity {
3         // Egy lapos 10x10cm téglatest generálás, ami egy cetlit vizuál.
4         let cardMesh = MeshResource.generateBox(width: cardWidth, height: cardWidth, depth: 0.001,
5             cornerRadius: 0.01, splitFaces: false)
6         // A cetli színezése.
7         var cardMaterial = PhysicallyBasedMaterial()
8         cardMaterial.baseColor.tint = UIColor(named: self.colorString) ?? .yellow
9         // A cetli textúráját érdesse teszi, hogy élethűbb legyen.
10        if let normalResource = try? TextureResource.load(named: "paperNormalMap") {
11            let normalMap = MaterialParameters.Texture(normalResource)
12            cardMaterial.normal = PhysicallyBasedMaterial.Normal(texture:normalMap)
13        }
14        // A fenti kódrészeket összefűzése egy Entity-vé, ami egy színes 3D-s cetli.
15        let cardEntity: Entity = ModelEntity(mesh: cardMesh, materials: [cardMaterial])
16        // A feladathoz tartozó szöveg legenerálása modellé.
17        let textEntity = RUIText(with: self.description, width: CGFloat(cardWidth)-0.005, height:
18            CGFloat(cardWidth)-0.005, font: UIFont(name:"Helvetica", size: 0.01) ?? .systemFont(ofSize:
19            0.01), extrusion: 0.001, color: .label)
20        ...
21        // A szöveg elhelyezése a cetli elé.
22        textEntity.transform.translation = [0, 0, 0.001]
23        textEntity.transform.rotation = simd_quatf(angle: -Float.pi, axis: [0, 1, 0])
24        let c = cardEntity
25        c.addChild(textEntity)
26        // A visszatérési érték pedig egy feladat 3D modellje, ami a cetli a hozzá tartozó szöveggel.
27        return c
28    }
29 }

```

Programkód vizuális
lehetséges megjelenése
(illusztráció)



A beadandó
írásának
elkezdése

10. ábra - saját szerkesztés

A 11. ábrán az látszik, hogy a *Subcollection* struktúra hogyan felel meg a protokollnak, és ez hogyan jelenik meg vizuálisan. Itt a *generateEntity()* függvény annyit tesz, hogy a hozzá tartozó feladatokon iterál, egyenként leképezi és elhelyezi őket egymás mellé és alá. Végül pedig elhelyezi felettük az algyűjtemény címét.

Programkód

```
1 extension Subcollection: ARPresentable {
2   func generateEntity() -> Entity {
3     // Végigiterál az algyűjtemény feladatain, és azokat elhelyezi egymás mellé és alá.
4     let value = Entity()
5     ...
6     for card in cards {
7       ...
8       let cardEntity = card.generateEntity()
9       value.addChild(cardEntity)
10      cardEntity.transform.translation = [xValue, yValue, 0]
11      ...
12    }
13    // Hozzáadja az algyűjtemény címét.
14    let titleMesh = MeshResource.generateText(self.title, extrusionDepth: 0.001, font:
15      UIFont(name:"Helvetica", size: 0.021) ?? .systemFont(ofSize: 0.021), containerFrame: CGRect(),
16      alignment: .center, lineBreakMode: .byWordWrapping)
17    ...
18    value.addChild(titleEntity)
19    titleEntity.transform.translation = [titleXValue, titleYValue, 0]
20    // A visszatérési érték az algyűjtemény leggenerált modellje.
21    return value
22  }
23 }
```

Programkód vizuális lehetséges megjelenése (illusztráció)



11. ábra - saját szerkesztés

Ahogy a 12. ábra szemlélteti, a *Collection* struktúra is megfelel az *ARPresentable* protokollnak, amivel az egész gyűjteményt leképezi 3D modellé. A *generateEntity()* függvény generál egy lapos és fehér téglatestet, amely egy táblának az illúzióját kelti, illetve leképezi és elhelyezi egymás mellé a gyűjteményben lévő algyűjteményeket.

Programkód

```
1 extension Collection: ARPresentable {
2   func generateEntity() -> Entity {
3     ...
4     // Algyűjtemények legenerálása 3D modellés és elhelyezése egymás mellé.
5     for subcollection in wrappedSubcollections.filter({ $0.isHidden == false }) {
6       let subcollectionEntity = subcollection.generateEntity()
7       ...
8       subcollectionsEntity!.addChild(subcollectionEntity)
9       xValue += subcollectionEntity.width+spaceBetweenSubcollections
10      subcollectionEntity.transform.translation = [xValue, 0, 0]
11    }
12    ...
13    // Egy fehér lap létrehozása. Ez egy tábla illúzióját kelti.
14    let mesh = MeshResource.generatePlane(width: width, height: height, cornerRadius: width*0.01)
15    let material = SimpleMaterial(color: color, roughness: 0.95, isMetallic: false)
16    let entity = ModelEntity(mesh: mesh, materials: [material])
17    ...
18    // A legenerált algyűjtemények elhelyezése a táblán.
19    boardEntity.addChild(subcollectionsEntity!)
20    // A visszatérési érték a létrehozott tábla, amelyen el vannak helyezve az algyűjtemények.
21    return boardEntity
22  }
23 }
```

Programkód vizuális lehetséges megjelenése (illusztráció)



12. ábra - saját szerkesztés

A teljes kódrészlet az 1. mellékleten látható.

5.1.4. ARKit implementálása

A 3D-s modelleket a RealityKit segítségével leképezi az alkalmazás az előző alponban részletezve, azonban azoknak megjelenítéséhez szükséges a telefon kameráján keresztül feltérképezni a felhasználó környezetét és elhelyezni a modelleket. Erre fejlesztette ki az Apple az ARKit keretrendszert (Developer Apple, 2017). A keretrendszer segítségével az *ARView* osztályon keresztül a szoftver eléri a felhasználó kameráját. Az *ARView* ösosztálya egy

Objective-C kódrészletből származik, ami miatt nem lehet közvetlen struktúrán és protokollokon keresztül megjeleníteni, hanem szükséges hozzá egy *UIViewController*-ből származtatott osztály, amelynek a neve az alkalmazásban *ARTasksViewController*, ahogy az 13. ábra is mutatja. Ezen belül többek között van egy változó és egy konstans. A változó neve *anyARPresentable*, amely bármilyen típus lehet, ami megfelel az előző pontban részletezett *ARPresentable* protokollnak. Ezt a változót az osztály példányosításakor lehet inicializálni, és ezen keresztül lehet legenerálni a 3D modellt. A legenerált modellt pedig az *arView* konstans segítségével lehet megjeleníteni a telefonon.

```
1 class ARTasksViewController: UIViewController {
2     ...
3     // Ez bármilyen típus lehet (struct, class vagy enum), amelyik megfelel az ARPresentable
    protokollnak. Ezeknek a típusoknak van egy generateEntity() metódusuk, amivel le lehet generálni a
    3D modellt.
4     var anyARPresentable: ARPresentable
5     // Ezen keresztül lehet megjeleníteni a legenerált 3D modellt.
6     private let arView = ARView()
7     ...
8 }
```

13. ábra - saját szerkesztés

Az *ARView* példányának konfigurálását a 14. ábrán lévő *setupARView()* metódus részletezi. Beállítja, hogy az eszköz egy olyan vertikális felületet keressen, amelyen elhelyezhető az előző alponthan létrehozott 3D modell. A környezeti fényeket és az árnyékolást automatikusan kezeli. Azt is konfigurálja, hogy ha megjelenik egy ember a 3 dimenziós modell előtt, akkor azt kitakarja. Ennek a vizuális eredményét az 16. ábra jobb oldala szemlélteti. Az ARKit ezt úgy valósítja meg, hogy azonosítja azokat a régiókat, ahol emberek tartózkodnak, és megakadályozza, hogy a virtuális tartalom belerajzoljon az adott régió pixeleibe (Developer Apple, 2022o).

```
1 private func setupARView() {
2     ...
3     // Beállítja, hogy egy vertikális objektumra (például egy falra) helyezze el a modellt.
4     let arConfig = ARWorldTrackingConfiguration()
5     arConfig.planeDetection = .vertical
6     // A környezeti fényeket a kamera, processzor és AI együtt automatikusan értelmezik.
7     arConfig.environmentTexturing = .automatic
8     // Érzékeli, ha valaki a modell előtt jelenik meg.
9     if ARWorldTrackingConfiguration.supportsFrameSemantics(.personSegmentationWithDepth) {
10         arConfig.frameSemantics = .personSegmentationWithDepth
11     }
12     ...
13     // Elindítja az AR folyamatot.
14     arView.session.run(arConfig)
15     ...
16 }
```

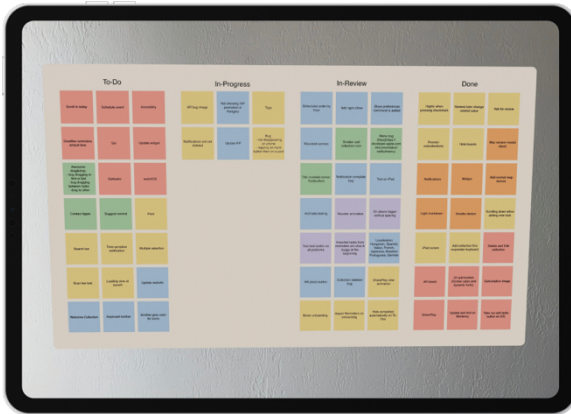
14. ábra - saját szerkesztés

Végül pedig egy *addCardBoard()* metódussal legenerálja és hozzáadja a modellt az ARView példányához. Ezt az 15. ábra mutatja, a végső megjelenést pedig az 16..

```
1 private func addCardBoard() {
2     ...
3     // Létrehoz egy láthatatlan "horgonyt", amihez csatolni fogja a 3D modellt.
4     let anchor = AnchorEntity(plane: .vertical, classification: .any)
5     arView.scene.addAnchor(anchor)
6     ...
7     // A modellt legenerálja, és hozzácsatolja a horgonyhoz.
8     let boardEntity = anyARRepresentable.generateEntity()
9     anchor.addChild(boardEntity)
10    ...
11 }
```

15. ábra - saját szerkesztés

AR megjelenés



Ember-elzárás (angolul: people occlusion)



16. ábra - saját szerkesztés

Tehát a szoftver tartalmaz egy olyan osztályt, ami feltérképezi a felhasználó környezetét, és elhelyezi a legenerált 3D-s modellt. Az osztálynak egyetlen paramétere van, az *anyARPresentable*, ami bármilyen típus lehet, ami megfelel az *ARPresentable* protokollnak. Ahogy az 4.1-es rész is írja, a protokollnak a feladaton szereplő struktúrán kívül akár egy osztály vagy egy enumeráció is megfelelhet. Ezzel az alkalmazás kihasználja a protokoll-orientált programozás egyik nagy előnyét, hogy generikus és skálázható kódrészleteket tartalmaz. Tehát például, ha a jövőben hozzá szeretnék adni egy naptár funkciót az alkalmazáshoz, és azt is meg szeretném jeleníteni kiterjesztett valóságban, akkor elég, ha a naptár típusát megfeleltetem az *ARPresentable* protokollnak. Ezt a típust pedig az *ARTasksViewController*nek az *anyARPresentable* paraméteréhez behelyezve meg is lehet jeleníteni a felhasználó eszközén keresztül.

Tehát jól látszik, hogy a csak kódból generált 3D modelleknél a keretrendszer nagy rugalmasságot ad a különböző alakítgatásokra, a jövőbeli skálázásra, illetve a szoftver méretét sem növeli meg. Limitáció azonban az, hogy komplex modelleket nehéz vele generálni.

Az osztály teljes változata a 2. mellékleten látható.

5.2. Coachy

Coachy magának a fitness alkalmazásnak és az alkalmazásban megjelenő 3D-s személyi edzőnek a neve. A Reality Tasks a 3D-s modellt csak kódból generálja, míg Coachy egy 3D modellező szoftverből importált animációkat implementál és szerkeszt kódból tovább.

5.2.1. Az alkalmazás funkciói

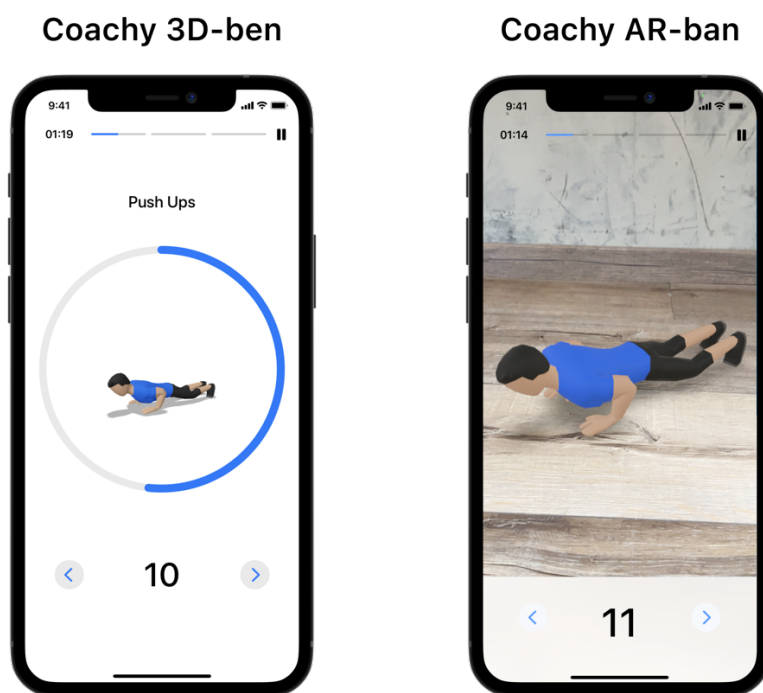
Az alkalmazás naponta generál személyre szabott - első sorban otthoni - edzéseket a felhasználónak az edzési múltja, kora, célja, az elérhető fitness eszközei és a korábbi edzéseinek visszajelzése alapján. Emellett böngészhet előre hozzáadott rutinok közül is, vagy elkészítheti sajátját. Minden edzéstervet tovább szerkeszthet a felhasználó az igényeinek megfelelően. Az alkalmazás felhasználói felületét a 17. ábra szemlélteti.



17. ábra - saját szerkesztés

Több, mint 200 gyakorlatot kínál Coachy, amelyek közül az összeset meg lehet tekinteni 3D animációként. Ezeket a gyakorlatokat el lehet helyezni a való világban is a kiterjesztett valóság segítségével. Ezzel azt az illúziót kelti az alkalmazás, hogy Coachy - a virtuális személyi edző - életre kel, és a felhasználóval együtt edz. Az elhelyezett gyakorlatot körbe lehet

járni, nagyítani, kicsinyíteni, forgatni és mozgatni. Ez a funkció akkor hasznos igazán, amikor a felhasználó épp két gyakorlat között pihen, és a soron lévő gyakorlatot a való világban elhelyezve megtekinti, körbejárja, és megérti, hogy hogyan kell helyesen elvégezni. Az AR funkciót a 18. ábra mutatja be.



18. ábra - saját szerkesztés

A felhasználó heti célokat tűzhet ki, illetve követheti az edzés múltját grafikonokon és egy naptáron keresztül. Az alkalmazás elérhető iOS, watchOS és macOS operációs rendszereken. Az eszközök között az edzések szinkronizáltak az iCloud felhőalapú tárhelyszolgáltatás segítségével.

Coachy integrált az Apple Egészség és Zene alkalmazásokkal. Tehát minden elvégzett edzés megjelenik az Egészség alkalmazásban is, a felhasználó pedig edzés közben kereshet és játszhat le zenéket anélkül, hogy a szoftvert elhagyná.

Coachy kinézetét a felhasználó ízlésének megfelelően szabhatja, az alkalmazás elérhető sötét módban is, az edzéseket ki lehet vetíteni a TV-re, illetve barátokkal és családtagokkal is együtt lehet edzeni a SharePlay technológia segítségével FaceTime-on keresztül.

5.2.2. RealityKit implementálása

Az alkalmazásban megjelenő animációkat Blenderben készítettem el és exportáltam ki glTF formátumba. Ezt a formátumot az Apple Reality Converter szoftverével átkonvertáltam USDZ-vé. Az USDZ fájlformátumot az Apple a Pixarral együtt fejlesztette ki, azzal a céllal,

hogy élethű háromdimenziós modelleket lehessen kiterjesztett valóságban megjeleníteni (Developer Apple, 2022p). Ezt a fájlt a RealityKit felismeri és könnyen tudja kezelni.

Ahogy a 19. ábra is részletezi, a *loadModelAsync()* metódus aszinkron betölti az USDZ fájlt, ami egy gyakorlat 3D modelljét és animációját tartalmazza, majd átalakítja Entity osztállyá. Ezen az Entity-n keresztül viszonylag sok tulajdonságát el lehet érni a modellnek attól függetlenül, hogy külső szoftverből lett importálva a fájl.

Az alkalmazás egyik funkciója, hogy a felhasználó szerkesztheti az edzőnek a kinézetét azzal, hogy átállítja például a hajának vagy a pólójának a színét. Ezt kódból úgy oldja meg a szoftver, hogy a betöltött Entity-nek a *materials* tulajdonságán keresztül eléri a 3D modellnek a különböző részeit, és átállítja azoknak a színét.

A felhasználó forgatni, kicsinyíteni, nagyítani és mozgatni tudja kiterjesztett valóságban a gyakorlatot az ujjával. Ezt az alkalmazás az *installGestures()* metódussal automatikusan hozzáadja. Végül pedig a *playAnimation()* függvénnyel elindítja a modellhez kapcsolódó animációt.

```
1 // USDZ fájl betöltése és Entity-vé alakítása.
2 ModelEntity.loadModelAsync(contentsOf: fileURL).sink(receiveCompletion: { loadCompletion in
3     ...
4 }, receiveValue: { modelEntity in
5     exerciseEntity = modelEntity
6     // Textúrák hozzáadása az Entity-hez.
7     loadMaterials()
8     // Gesztusok ráillesztése: ezzel lehet forgatni, méretezni és mozgatni.
9     exerciseEntity.generateCollisionShapes(recursive: true)
10    arView.installGestures(.all, for: exerciseEntity)
11    // Animáció elindítása.
12    exerciseEntity.playAnimation($0.repeat())
13    ...
14 })
```

19. ábra - saját szerkesztés

Tehát az alkalmazás először betölti az USDZ fájlt, Entity-vé alakítja, kiszínezi a 3D modell egyes részeit, interakciókat fűz hozzá, végül pedig elindítja a gyakorlat animációját. Ezzel jól látszik, hogy annak ellenére, hogy külső szoftverben készült a 3D-s modell, az USDZ formátum miatt a RealityKit segítségével tovább lehet szerkeszteni. Ennek egyik hátránya viszont, hogy az alkalmazás mérete jelentősen megnő, mivel egy-egy USDZ fájl nagyjából 3mb. Mivel 200 gyakorlatot kínál az alkalmazás, ez összesen 600 megabájt növekedést jelentene a szoftver méretében. Ennek a csökkenése érdekében, a gyakorlatokat külső szerverről tölti le az alkalmazás, és nem az alkalmazás mappáiban tárolja.

5.2.3. ARKit implementálása

A Reality Taskhoz hasonlóan itt is az ARKit-et használja az alkalmazás arra, hogy kamerán keresztül feltérképezze a környezetet és elhelyezze benne a betöltött gyakorlatot. Ahogy a 20. ábra is mutatja, a `setupARView()` metódus keres egy horizontális felületet (például egy padlót), és azon elhelyezi az előző pontban létrehozott animációt.

```
1 private func addEntityToARView() {  
2     // Egy "horgony" hozzáadása egy horizontális felülethez (például padlóhoz).  
3     let anchorEntity = AnchorEntity(plane: .horizontal)  
4     // A 3D modell "horgonyhoz" csatolása. Ezzel megjelenik a gyakorlat a valóságban.  
5     anchorEntity.addChild(exerciseEntity)  
6     arView.scene.addAnchor(anchorEntity)  
7 }
```

20. ábra - saját szerkesztés

A Reality Tasks beállítja, hogy az embereket felismerje a kamera, és kizárja a mögöttük megjelenő háromdimenziós modell pixeleit a kijelzőről. Coachy ennél egyvel tovább lép, és a 21. ábrán szemléltetve az embereken kívül az objektumokat (például falakat) is felismeri az eszköz, ezzel élethűbb vizuális megjelenést elérve. Véleményem szerint ez az emberi szemnek és agynak nagyon egyszerű folyamatnak tűnik, azonban az okoseszközöknek komoly számítási kapacitásába telik. Ezt a másik alkalmazás azért nem használja ki, mivel a falon elhelyezett objektum nagyon lapos, és a kamera sokszor nem ismeri fel, hogy a fal elé helyeztük el a cetliket, és nem mögé.

```
1 // A környezet tárgyait felismeri, és a mögötte megjelenő modell pixeleit nem jeleníti meg a kijelzőn.  
2 arView.environment.sceneUnderstanding.options = .occlusion
```



21. ábra - saját szerkesztés

5.2.4. Megjelenítés virtuális háromdimenzióban

A gyakorlatokat az alkalmazás nem csak kiterjesztett valóságban, hanem - ahogy a 18. ábrán lévő baloldali kép is mutatja - egy háromdimenziós virtuális környezetben is meg tudja jeleníteni.

Ahhoz hogy AR-ból ebben a virtuális környezetben jelenjen meg, elég a `arView.cameraMode = .nonAR` kódot használni. Ezzel automatikusan létrehoz egy virtuális környezetet, és kikapcsolja a telefon kamerájának a használatát. Ekkor azonban az eszköz nem tudja, hogy pontosan hova helyezze el a virtuális kamerát, illetve milyen fényeket és árnyékolást rendeljen a 3D modellhez. Ezekkel a problémákkal nem kell szembenézni akkor, amikor kiterjesztett valóságban helyezi el a gyakorlatot, hiszen az ARKit a telefon kamerájának a mozgását érzékeli, és felismeri a környezetet meg a megjelenő fényeket, ami miatt az árnyékolást is kezelni tudja.

Tehát ahhoz, hogy szépen jelenjen meg a 3D modell a kijelzőn, először egy kamerát helyez el az alkalmazás a virtuális környezetben, azt követően fényeket rendel az gyakorlat köré, végül pedig kezeli az árnyékolást.

A virtuális kamera elhelyezését a 22. ábrán megjelenő kóddal, az `addCameraToAnchor()` metódussal éri el. Ez létrehozza a kamerát, és beállítja, hogy a 3D modelltől horizontálisan és vertikálisan is a modell magasságának a másfélszeresével legyen elhelyezve.

```
1 private func addCameraToAnchor(anchor: AnchorEntity) {
2     // Virtuális kamera létrehozása.
3     let cameraEntity = PerspectiveCamera()
4     // A kamera elhelyezése a gyakorlathoz képest.
5     let height = exerciseEntity.visualBounds(relativeTo: nil).max.y -
6     exerciseEntity.visualBounds(relativeTo: nil).min.y
7     cameraEntity.look(at: [exerciseEntity.position.x, yValue, exerciseEntity.position.z], from:
8     [height*1.5, height*1.5, zValue], relativeTo: exerciseEntity)
9     anchor.addChild(cameraEntity)
10 }
```

22. ábra - saját szerkesztés

Ezt követően - ahogy a 23. ábra is részletezi - a gyakorlat felett 1 méterrel elhelyez egy virtuális fehér fényt az *addLight()* függvénnyel.

```
1 private func addLight() {
2     // Fény létrehozása.
3     let directionalLight = DirectionalLight()
4     directionalLight.light.color = .white
5     ...
6     // Fény eltolása felfelé.
7     directionalLight.orientation = simd_quatf(angle: -.pi/2, axis: [1,0,0])
8     ...
9     // Hozzáadás a virtuális világhoz.
10    let lightAnchor = AnchorEntity(world: [0, 0, 0])
11    lightAnchor.addChild(directionalLight)
12    ...
13 }
```

23. ábra - saját szerkesztés

Végül pedig az *addBackgroundPlane()* metódus elhelyez a modell alatt egy ötször öt méteres talajt, amin megjelenik az árnyék. Ennek a kódrészlete a 24. ábrán látható.

```
1 private func addBackgroundPlane(nonARAnchor: AnchorEntity) {
2     // Egy 5x5 méteres lap elhelyezése, amin megjelenik a gyakorlat árnyéka.
3     let mesh = MeshResource.generatePlane(width: 5, depth: 5)
4     let material = OcclusionMaterial(receivesDynamicLighting: true)
5     let catcher = ModelEntity(mesh: mesh, materials: [material])
6     nonARAnchor.addChild(catcher)
7 }
```

24. ábra - saját szerkesztés

Tehát a RealityKit segítségével nem csak AR-ban, hanem 3D-ben is meg lehet jeleníteni a modelleket. Ekkor azonban kezelni kell azokat a problémákat, amiket az ARKit szinte automatikusan megold kiterjesztett valóságban.

A teljes osztály programkódját, ami megjeleníti a Coachy-t kiterjesztett valóságban és virtuális háromdimenzióban a 3. melléklet tartalmazza.

6. Összegzés

A kiterjesztett valóság még egy viszonylag új technológia, ami jelentős növekedési potenciál előtt áll. Már most széles felhasználási területet kínál, hiszen jelen van a kereskedelemben, az oktatásban, a szolgáltatás-fejlesztésben, és rengeteg mobilalkalmazásban. (PwC, 2019a, 9-13. old.)

Az Apple AR értékajánlata a fejlesztők felé az, hogy modern programozói keretrendszerekkel fejleszthetnek megbízhatóan működő eszközökre alkalmazásokat, amiket publikálhatnak a világ legnagyobb AR platformjára, az App Store-ra (Apple, 2022a). Fontos azt is megemlíteni, hogy az okoszeműveg szabadalmainak száma jelentős, ami gyanúként szolgálhat egy új AR hardver kiadására a jövőbe tekintve (Seitz, 2021).

A kutatás célja bemutatni mélyen, hogy hogyan lehet az Apple kiterjesztett valóság keretrendszereit kihasználva modern mobil- és tabletalkalmazásokat fejleszteni. A Reality Tasks feladat- és projektkezelő szoftver az AR adta lehetőségeket arra használja ki, hogy a közkezdvelt cetlik használatát digitális formában, de mégis a valóságban elhelyezve tegye élménytelivé. A Coachy pedig egy olyan digitális személyi edzőt biztosít, amelyet a felhasználó életre kelthet a saját otthonában a kiterjesztett valósággal.

A Reality Tasks által jól látszik, hogy a Swift protokoll-orientált sajátosságát kihasználva egy generikus és skálázható programot lehet fejleszteni. A háromdimenziós objektumot kódból, egyszerű téglatestekből és szövegekből generálja a szoftver. Ez nagy rugalmasságot jelent, hiszen bármilyen teendő listát vagy kanban táblát meg tud jeleníteni anélkül, hogy jelentősen megnövelné az alkalmazás méretét.

Coachy a Reality Taskhoz képest egy sokkal komplexebb modellt jelenít meg animációval együtt. Egy ilyen modellt kódból előállítani egy nagyon nehéz folyamat, ami miatt egy külső 3D modellező szoftvert, a Blendert használtam. A Blenderből kiexportált fájlt az Apple Reality Converter segítségével könnyen USDZ-vé tudtam konvertálni, ami a RealityKit keretrendszerrel jól integrált. Az ARKit által helyezi el és árnyékolja az alkalmazásban ezt a modellt, illetve az emberek és más objektumok felismerését is kezeli. A valóság mellett, egy virtuális háromdimenziós térben is el lehet helyezni a gyakorlatokat, ekkor azonban a kamera elhelyezését, a fényeket és az árnyékolást manuálisan kell kezelni.

Véleményem szerint - annak ellenére, hogy csak pár évesek a használt AR keretrendszerek - már most viszonylag komplex és élethű kiterjesztett valóság élményben lehet részünk, aminek következő szintje egy jól működő és vonzó okoszeműveg lehetne. Ez a

hardver ugyanis jelentősen megnövelné a felhasználói élményt, és olyan felhasználási területeket tárna fel, amikre most még nem gondolunk.

A szoftverfejlesztési projekt célja az Apple AR technológiáinak mély implementálása és lehetőségeinek feltárása volt, azonban további kérdés lehet, hogy a Coachy-ban és a Reality Tasks-ban megjelenő kiterjesztett valóság funkciók mekkora értéket adnak hozzá jelenleg az alkalmazásokhoz.

Irodalomjegyzék

- Airforce Technology (2021): Kopin wins liquid crystal displays follow-on order for F-35 JSF programme. Letöltés helye: <https://www.airforce-technology.com/news/kopin-displays-f-35-jsf-programme/>. Letöltés ideje: 2022. Április 5.
- Apple (2020a): Apple announces App Store Small Business Program. Letöltés helye: <https://www.apple.com/newsroom/2020/11/apple-announces-app-store-small-business-program/>. Letöltés ideje: 2022. Április 8.
- Apple (2020b): Apple unveils new iPad Pro with LiDAR Scanner and trackpad support in iPadOS. Letöltés helye: <https://www.apple.com/hu/newsroom/2020/03/apple-unveils-new-ipad-pro-with-lidar-scanner-and-trackpad-support-in-ipados/>. Letöltés ideje: 2022. Április 3.
- Apple (2022a): Kiterjesztett valóság. Letöltés helye: <https://www.apple.com/hu/augmented-reality/>. Letöltés ideje: 2022. Április 10.
- Bauer, Rob – Otten, Rogér – Tourani-Rad, Alireza (2006): New Zealand Mutual Funds: Measuring Performance and Persistence in Performance. Accounting and Finance. 46. kötet, 3. szám, 347-363 old.
- Canalys (2016): Apple shipped two-thirds of all smart watches in 2015, 1. oldal. Letöltés helye: https://canalys-prod-public.s3.eu-west-1.amazonaws.com/static/press_release/2016/media-alert-05022016-apple-shipped-two-thirds-all-smart-watches-2015.pdf, 1. oldal. Letöltés ideje: 2022. Április 10.
- Cohen, Rachel (2021): F-35 helmets that fix ‘green glow’ are on their way — but not to the Air Force. Letöltés helye: <https://www.airforcetimes.com/news/your-air-force/2021/12/08/f-35-helmets-that-fix-green-glow-are-on-their-way-but-not-to-the-air-force/>. Letöltés ideje: 2022. Április 5.
- Developer Apple (2015): Protocol-Oriented Programming in Swift. Letöltés helye: <https://developer.apple.com/videos/play/wwdc2015/408/>. Letöltés ideje: 2022. Április 5.
- Developer Apple (2017): Introducing ARKit: Augmented Reality for iOS. Letöltés helye: <https://developer.apple.com/videos/play/wwdc2017/602/>. Letöltés ideje: 2022. Április 5.
- Developer Apple (2019a): Introducing RealityKit and Reality Composer. Letöltés helye: <https://developer.apple.com/videos/play/wwdc2019/603/>. Letöltés ideje: 2022. Április 5.
- Developer Apple (2019b): Introducing SwiftUI: Building Your First App. Letöltés helye: <https://developer.apple.com/videos/play/wwdc2019/204/>. Letöltés ideje: 2022. Április 5.

- Developer Apple (2022a): Augmented Reality. Letöltés helye: <https://developer.apple.com/augmented-reality/>. Letöltés ideje: 2022. Április 5.
- Developer Apple (2022b): Improving the Performance of a RealityKit App. Letöltés helye: https://developer.apple.com/documentation/realitykit/arview/improving_the_performance_of_a_realitykit_app. Letöltés ideje: 2022. Április 5.
- Developer Apple (2022c): Promoting your apps. Letöltés helye: <https://developer.apple.com/app-store/promote/>. Letöltés ideje: 2022. Április 5.
- Developer Apple (2022d): Apple Developer Program. Letöltés helye: <https://developer.apple.com/programs/>. Letöltés ideje: 2022. Április 5.
- Developer Apple (2022e): Xcode. Letöltés helye: <https://developer.apple.com/documentation/xcode>. Letöltés ideje: 2022. Április 4.
- Developer Apple (2022f): SwiftUI. Letöltés helye: <https://developer.apple.com/documentation/SwiftUI>. Letöltés ideje: 2022. Április 2.
- Developer Apple (2022g): Combine. Letöltés helye: <https://developer.apple.com/documentation/Combine>. Letöltés ideje: 2022. Április 4.
- Developer Apple (2022h): HealthKit. Letöltés helye: <https://developer.apple.com/documentation/healthkit>. Letöltés ideje: 2022. Április 2.
- Developer Apple (2022i): WatchKit. Letöltés helye: <https://developer.apple.com/documentation/watchkit>. Letöltés ideje: 2022. Április 4.
- Developer Apple (2022j): Mac Catalyst. Letöltés helye: https://developer.apple.com/documentation/uikit/mac_catalyst. Letöltés ideje: 2022. Április 2.
- Developer Apple (2022k): EventKit. Letöltés helye: <https://developer.apple.com/documentation/eventkit>. Letöltés ideje: 2022. Április 2.
- Developer Apple (2022l): StoreKit. Letöltés helye: <https://developer.apple.com/documentation/storekit>. Letöltés ideje: 2022. Április 2.
- Developer Apple (2022m): CoreData. Letöltés helye: <https://developer.apple.com/documentation/coredata>. Letöltés ideje: 2022. Április 2.
- Developer Apple (2022n): Encrypting User Data. Letöltés helye: https://developer.apple.com/documentation/cloudkit/encrypting_user_data. Letöltés ideje: 2022. Április 2.

- Developer Apple (2022o): Occluding Virtual Content with People. Letöltés helye: https://developer.apple.com/documentation/arkit/camera_lighting_and_effects/occluding_virtual_content_with_people. Letöltés ideje: 2022. Április 2.
- Developer Apple (2022p): USDZ Schemas for AR. Letöltés helye: https://developer.apple.com/documentation/arkit/usdz_schemas_for_ar. Letöltés ideje: 2022. Április 2.
- Docs Swift (2022a): About Swift. Letöltés helye: <https://docs.swift.org/swift-book/>. Letöltés ideje: 2022. Április 10.
- Docs Swift (2022b): Protocols. Letöltés helye: <https://docs.swift.org/swift-book/LanguageGuide/Protocols.html>. Letöltés ideje: 2022. Április 10.
- Firebase (2022): Cloud Firestore. Letöltés helye: <https://firebase.google.com/docs/firestore>. Letöltés ideje: 2022. Április 10.
- Forbes (2021): Apple (AAPL). Letöltés helye: <https://www.forbes.com/companies/apple/?sh=16193c8b5355>. Letöltés ideje: 2022.
- Iqbal, Mansoor (2022): App Revenue Data (2022). Letöltés helye: <https://www.businessofapps.com/data/app-revenues/>. Letöltés ideje: 2022. Április 5.
- Javornik, Ana (2016): The Mainstreaming of Augmented Reality: A Brief History. Letöltés helye: <https://hbr.org/2016/10/the-mainstreaming-of-augmented-reality-a-brief-history>. Letöltés ideje: 2022. Április 2.
- MindTools (N.A.): To-Do Lists. Letöltés helye: https://www.mindtools.com/pages/article/newHTE_05.htm. Letöltés ideje: 2022. Április 5.
- Patently Apple (2022a): HMDs, Smartglasses. Letöltés helye: <https://www.patentlyapple.com/patently-apple/face-object-recognition/>. Letöltés ideje: 2022. Április 10.
- Patently Apple (2022b): Project Titan, Vehicle Technology. Letöltés helye: <https://www.patentlyapple.com/patently-apple/face-object-recognition/>. Letöltés ideje: 2022. Április 2.
- Perry, Tekla S. (2021): This Is the Year for Apple's AR Glasses—Maybe. Letöltés helye: <https://spectrum.ieee.org/this-is-the-year-for-apples-ar-glassesmaybe>. Letöltés ideje: 2022. Április 5.
- PwC (2019a): Seeing is believing. Letöltés helye: <https://www.pwc.com/gx/en/technology/publications/assets/how-virtual-reality-and-augmented-reality.pdf>. Letöltés ideje: 2022. Április 10.

- PwC (2019b): Seeing is believing. Letöltés helye: <https://www.pwc.com/SeeingIsBelieving>. Letöltés ideje: 2022. Április 10.
- Rehkopf, Max (N.A.): What is a Kanban Board?. Letöltés helye: <https://www.atlassian.com/agile/kanban/boards>. Letöltés ideje: 2022. Április 2.
- Saïd Business School, University of Oxford (2018): In Conversation with Apple CEO Tim Cook - The Oxford Foundry Launch. Letöltés helye: <https://www.youtube.com/watch?v=43UzfpcqiEU>. Letöltés ideje: 2022. Április 5.
- sec.gov (2008): Form 10-K. Letöltés helye: <https://www.sec.gov/Archives/edgar/data/320193/000119312508224958/d10k.htm>, 41. oldal. Letöltés ideje: 2022. Április 10.
- Seitz, Patrick (2021): Augmented Reality Won't Take Off Until Apple Enters Market, Says Morgan Stanley. Letöltés helye: <https://www.investors.com/news/technology/augmented-reality-wont-take-off-until-apple-enters-market-says-morgan-stanley/>. Letöltés ideje: 2022. Április 12.
- swift.org (2022): Welcome to Swift.org. Letöltés helye: <https://www.swift.org>. Letöltés ideje: 2022. Április 10.
- Techlife Coach (2016): Tim Cook at Utah Tech Tour. Letöltés helye: <https://www.youtube.com/watch?v=ZkV78RZ73Mw>. Letöltés ideje: 2022. Április 5.
- Tulane University (2021): What's the Difference Between AR and VR? Letöltés helye: <https://sopa.tulane.edu/blog/whats-difference-between-ar-and-vr>. Letöltés ideje: 2022. Április 2.

Mellékletek

1. melléklet

```
struct Collection: Identifiable, Hashable {

    var id: String
    var createdAt: Timestamp
    var updatedAt: Timestamp
    var title: String
    var colorString: String
    var adminID: String
    var sharedIDs: [String]?
    var invitedIDs: [String]?
    var isListMode: Bool
    var templateID: Int
    /// 0 if it is not pinned. Its value is the order of the pinning.
    var pinnedID: Int
    var subcollections: [Subcollection]?

    var color: Color {
        return colorString.cardColor.color
    }

    var colors: [Color] {
        return [colorString.cardColor.color, colorString.cardColor.color,
colorString.cardColor.color]
    }

    var template: Template {
        var value = emptyTemplate

        for template in templates {
            if templateID == template.id {
```

```

        value = template
    }
}

return value
}

}

extension Collection: ARPresentable {

    func generateEntity() -> Entity {

        let spaceBetweenSubcollections: Float = 0.05

        var xValue: Float = 0
        var subcollectionsEntity: Entity?

        if let wrappedSubcollections = subcollections {
            for subcollection in wrappedSubcollections.filter({ $0.isHidden == false }) {
                let subcollectionEntity = subcollection.generateEntity()
                if subcollectionsEntity == nil {
                    subcollectionsEntity = subcollectionEntity
                } else {
                    subcollectionsEntity!.addChild(subcollectionEntity)
                    xValue += subcollectionEntity.width+spaceBetweenSubcollections
                    subcollectionEntity.transform.translation = [xValue, 0, 0]
                }
            }
        }

        subcollectionsEntity!.transform.translation = [-
subcollectionsEntity!.width/2+spaceBetweenSubcollections, subcollectionsEntity!.height/2-
0.08, 0]

```

```

    let boardEntity = generateBoard(width: subcollectionsEntity!.width+0.1, height:
subcollectionsEntity!.height+0.1, color: UIColor.systemBackground)
    boardEntity.transform.rotation = simd_quatf(angle: -Float.pi/2, axis: [1, 0, 0])
    boardEntity.generateCollisionShapes(recursive: true)
    boardEntity.addChild(subcollectionsEntity!)

    return boardEntity

}

private func generateBoard(width: Float, height: Float, color: UIColor) -> Entity {

    let mesh = MeshResource.generatePlane(width: width, height: height, cornerRadius:
width*0.01)
    let material = SimpleMaterial(color: color, roughness: 0.95, isMetallic: false)
    let entity = ModelEntity(mesh: mesh, materials: [material])

    return entity

}

}

struct Subcollection: Identifiable, Hashable {

    var id: String
    var orderID: Int
    var title: String
    var colorString: String
    var isHidden: Bool
    var tasks: [Task]?

    var color: Color {
        return colorString.cardColor.color
    }
}

```

```

    }

    var colors: [Color] {
        return [colorString.cardColor.color, colorString.cardColor.color,
colorString.cardColor.color]
    }
}

extension Subcollection: ARPresentable {

    func generateEntity() -> Entity {

        let value = Entity()
        let numberOfColumns = 3

        var cardIndex = -1
        var xValue: Float = 0
        var yValue: Float = cardWidth + 0.01

        if let cards = self.tasks {
            for card in cards {
                cardIndex += 1
                if cardIndex <= 20 { // maximum shown cards is 20
                    if cardIndex % numberOfColumns == 0 {
                        xValue = 0
                        yValue -= cardWidth + 0.01
                    } else {
                        xValue += cardWidth + 0.01
                    }
                }
                let cardEntity = card.generateEntity()
                value.addChild(cardEntity)
                cardEntity.transform.translation = [xValue, yValue, 0]
            }
        }
    }
}

```

```

    }
}

    let titleMesh = MeshResource.generateText(self.title, extrusionDepth: 0.001, font:
UIFont(name:"Helvetica", size: 0.021) ?? .systemFont(ofSize: 0.021), containerFrame:
CGRect(), alignment: .center, lineBreakMode: .byWordWrapping)
    let titleMaterial = SimpleMaterial(color: UIColor.label, roughness: 0.95, isMetallic:
false)
    let titleEntity = ModelEntity(mesh: titleMesh, materials: [titleMaterial])
    let titleXValue = value.visualBounds(relativeTo: nil).center.x -
(titleEntity.model?.mesh.bounds.center.x ?? 0.1)
    let titleYValue = value.visualBounds(relativeTo: nil).max.y +
(titleEntity.model?.mesh.bounds.min.y ?? 0.003) + 0.01
    value.addChild(titleEntity)
    titleEntity.transform.translation = [titleXValue, titleYValue, 0]

    return value

}

}

struct Task: Identifiable, Hashable {

    var id: String
    var orderID: Int
    var colorString: String
    var description: String
    var priorityID: Int
    var deadline: Timestamp?
    var remindAt: Timestamp?
    var recurrenceID: Int
    var tags: [String]

```

```

var comments: [Comment]?

var color: Color {
    return colorString.cardColor.color
}

var data: [String : Any] {
    return [
        FBKeys.orderID.rawValue : orderID,
        FBKeys.colorString.rawValue : colorString,
        FBKeys.description.rawValue : description,
        FBKeys.priorityID.rawValue : priorityID,
        FBKeys.deadline.rawValue : deadline ?? NSNull(),
        FBKeys.remindAt.rawValue : remindAt ?? NSNull(),
        FBKeys.recurrenceID.rawValue : recurrenceID,
        FBKeys.tags.rawValue : tags
    ]
}

}

extension Task: ARPresentable {

    func generateEntity() -> Entity {

        let cardMesh = MeshResource.generateBox(width: cardWidth, height: cardWidth, depth:
0.001, cornerRadius: 0.01, splitFaces: false)

        var cardEntity: Entity {
            if #available(iOS 15.0, *) {
                var cardMaterial = PhysicallyBasedMaterial()
                cardMaterial.baseColor.tint = UIColor(named: self.colorString) ?? .yellow
                if let normalResource = try? TextureResource.load(named: "paperNormalMap") {
                    let normalMap = MaterialParameters.Texture(normalResource)

```

```

        cardMaterial.normal = PhysicallyBasedMaterial.Normal(texture:normalMap)
    }
    return ModelEntity(mesh: cardMesh, materials: [cardMaterial])
} else {
    let cardMaterial = SimpleMaterial(color: UIColor(named: self.colorString) ??
.yellow, roughness: 0.95, isMetallic: false)
    return ModelEntity(mesh: cardMesh, materials: [cardMaterial])
}
}

let textEntity = RUIText(with: self.description, width: CGFloat(cardWidth)-0.005,
height: CGFloat(cardWidth)-0.005, font: UIFont(name: "Helvetica", size: 0.01) ??
.systemFont(ofSize: 0.01), extrusion: 0.001, color: .label)
if #available(iOS 15.0, *) {
    var textMaterial = PhysicallyBasedMaterial()
    textMaterial.baseColor.tint = UIColor.label
    if let normalResource = try? TextureResource.load(named: "paperNormalMap") {
        let normalMap = MaterialParameters.Texture(normalResource)
        textMaterial.normal = PhysicallyBasedMaterial.Normal(texture:normalMap)
    }
    textEntity.textModel!.materials = [textMaterial]
} else {
    let textMaterial = SimpleMaterial(color: UIColor.label, roughness: 0.95, isMetallic:
false)
    textEntity.textModel!.materials = [textMaterial]
}
textEntity.transform.translation = [0, 0, 0.001]
textEntity.transform.rotation = simd_quatf(angle: -Float.pi, axis: [0, 1, 0])

let c = cardEntity
c.addChild(textEntity)

return c

```

```

    }
}

```

2. melléklet

```

class ARTasksViewController: UIViewController {

    private let arView = ARView()
    private let snapshotButton = UIButton()
    private let snapshotButtonOutline = UIButton()
    private let spinner = UIActivityIndicatorView(style: .medium)

    var collection: ARPresentable!
    var cameraMode: ARView.CameraMode!

    private var showingCoachingOverlay = true
    private var isCoachingOverlayAdded = false
    private var isBoardAdded = false

    override func viewDidLoad() {
        super.viewDidLoad()

        setupARView()
    }

    private func setupARView() {
        print("setupARView()")

        arView.translatesAutoresizingMaskIntoConstraints = false
        view.addSubview(arView)
        NSLayoutConstraint.activate([
            arView.bottomAnchor.constraint(equalTo: view.bottomAnchor),

```

```

arView.topAnchor.constraint(equalTo: view.topAnchor),
arView.leadingAnchor.constraint(equalTo: view.leadingAnchor),
arView.trailingAnchor.constraint(equalTo: view.trailingAnchor)
])

switch cameraMode {
case .nonAR:
    arView.cameraMode = .nonAR
    arView.environment.background = .color(.systemBackground)
    self.addCardBoard()
default:
    arView.cameraMode = .ar
    let arConfig = ARWorldTrackingConfiguration()
    arConfig.planeDetection = .vertical
    arConfig.environmentTexturing = .automatic
    if
ARWorldTrackingConfiguration.supportsFrameSemantics(.personSegmentationWithDepth) {
        arConfig.frameSemantics = .personSegmentationWithDepth
    }
    if ARWorldTrackingConfiguration.supportsSceneReconstruction(.mesh) {
        arConfig.sceneReconstruction = .mesh
    }
    arView.session.run(arConfig)

    addCoaching()

DispatchQueue.main.asyncAfter(deadline: .now()+4.0) {
    if self.isCoachingOverlayAdded == false {
        self.showingCoachingOverlay = false
        self.isCoachingOverlayAdded = true
        self.addCardBoard()
    }
}
}
}

```

```

}

private func addCardBoard() {
    print("addCardBoard()")

    DispatchQueue.main.async {
        self.setupSpinner()
    }

    let anchor = AnchorEntity(plane: .vertical, classification: .any)

    arView.scene.anchors.removeAll()
    arView.scene.addAnchor(anchor)

    let boardEntity = collection.generateEntity()
    arView.installGestures(for: (boardEntity as? Entity & HasCollision)!)

    anchor.addChild(boardEntity)

    DispatchQueue.main.asyncAfter(deadline: .now()+4.0) {
        self.spinner.stopAnimating()
        self.setUpButton()
    }
}

private func setupSpinner() {
    print("setupSpinner()")
    spinner.translatesAutoresizingMaskIntoConstraints = false
    spinner.startAnimating()
    view.addSubview(spinner)

    spinner.centerXAnchor.constraint(equalTo: view.centerXAnchor).isActive = true

```

```

        spinner.centerYAnchor.constraint(equalTo: view.centerYAnchor).isActive = true
    }

    private func setUpButton() {
        print("setUpButton()")

        snapshotButtonOutline.backgroundColor = .clear
        snapshotButtonOutline.layer.borderWidth = 3
        snapshotButtonOutline.layer.borderColor = UIColor.white.cgColor
        snapshotButtonOutline.translatesAutoresizingMaskIntoConstraints = false
        view.addSubview(snapshotButtonOutline)
        NSLayoutConstraint.activate([
            snapshotButtonOutline.bottomAnchor.constraint(equalTo:
view.safeAreaLayoutGuide.bottomAnchor, constant: -20),
            snapshotButtonOutline.centerXAnchor.constraint(equalTo: view.centerXAnchor),
            snapshotButtonOutline.widthAnchor.constraint(equalToConstant: 70),
            snapshotButtonOutline.heightAnchor.constraint(equalToConstant: 70)
        ])
        snapshotButtonOutline.layer.cornerRadius = 35

        snapshotButton.backgroundColor = .white
        snapshotButton.translatesAutoresizingMaskIntoConstraints = false
        snapshotButtonOutline.addSubview(snapshotButton)
        NSLayoutConstraint.activate([
            snapshotButton.centerYAnchor.constraint(equalTo:
snapshotButtonOutline.centerYAnchor),
            snapshotButton.centerXAnchor.constraint(equalTo:
snapshotButtonOutline.centerXAnchor),
            snapshotButton.widthAnchor.constraint(equalToConstant: 60),
            snapshotButton.heightAnchor.constraint(equalToConstant: 60)
        ])
        snapshotButton.layer.cornerRadius = 30
    }

```

```

        snapshotButton.addTarget(self, action:#selector(self.saveScreenshot), for:
        .touchUpInside)

    }

}

extension ARTasksViewController: ARCoachingOverlayViewDelegate {

    private func addCoaching() {
        print("addCoaching()")

        let coachingOverlay = ARCoachingOverlayView()
        coachingOverlay.delegate = self
        coachingOverlay.session = arView.session
        coachingOverlay.autoresizingMask = [.flexibleWidth, .flexibleHeight]
        coachingOverlay.goal = .verticalPlane
        arView.addSubview(coachingOverlay)

    }

    func coachingOverlayViewDidRequestSessionReset(_ coachingOverlayView:
    ARCoachingOverlayView) {
        print("coachingOverlayViewDidRequestSessionReset()")
    }

    func coachingOverlayViewDidDeactivate(_ coachingOverlayView:
    ARCoachingOverlayView) {
        print("coachingOverlayViewDidDeactivate()")
        if isBoardAdded == false {
            addCardBoard()
            isBoardAdded = true
        }
        showingCoachingOverlay = false
    }
}

```

```

        isCoachingOverlayAdded = true
    }

    func coachingOverlayViewWillActivate(_ coachingOverlayView:
ARCoachingOverlayView) {
        print("coachingOverlayViewWillActivate()")
        isCoachingOverlayAdded = true
        showingCoachingOverlay = true
    }
}

extension ARTasksViewController {

    @objc func saveScreenshot() {
        let feedbackGenerator = UINotificationFeedbackGenerator()
        feedbackGenerator.notificationOccurred(.success)
        self.arView.snapshot(saveToHDR: false, completion: { image in
            if let wrappedImage = image {
                UIImageWriteToSavedPhotosAlbum(wrappedImage, self,
#selector(self.image(_:didFinishSavingWithError:contextInfo:)), nil)
            }
        })

        let overlay: UIView = UIView(frame: CGRect(x: 0, y: 0, width:
arView.frame.size.width, height: arView.frame.size.height))
        overlay.backgroundColor = UIColor(red: 0/255, green: 0/255, blue: 0/255, alpha: 1.0)
        withAnimation(.easeInOut(duration: 0.15)) {
            arView.addSubview(overlay)
        }

        DispatchQueue.main.asyncAfter(deadline: .now()+0.15) {
            withAnimation(.easeInOut(duration: 0.15)) {
                overlay.removeFromSuperview()
            }
        }
    }
}

```

```

    }
}

}

@objc func image(_ image: UIImage, didFinishSavingWithError error: Error?,
contextInfo: UnsafeRawPointer) {
    if let error = error {
        print("Error Saving ARKit Scene \(error)")
    } else {
        print("ARKit Scene Successfully Saved")
    }
}
}
}

```

3. melléklet

```

class ExerciseViewController: UIViewController, ARCoachingOverlayViewDelegate {

    // MARK: - properties
    var exercise: ExerciseStruct!
    var mode: ExerciseMode = .nonAR
    var showingExercise: Bool = true
    var isExerciseSheetPresented: Bool = false

    // non-ar specific
    private var backgroundColor: UIColor {
#if targetEnvironment(macCatalyst)
        return .systemBackground
#else
        return .secondarySystemGroupedBackground
#endif
    }
}

```

```

private var beginPointX = CGFloat(0)

// ar specific
private var showingCoachingOverlay = true
private var isCoachingOverlayAdded = false
private var oldAnchor: AnchorEntity?
private var oldEntity: Entity?

// shared
private var oldModelName = ""
private let networkingManager = NetworkingManager()
private let arView = ARView()
private let spinner = UIActivityIndicatorView(style: .medium)
private let errorLabel = UILabel()
private let defaults = UserDefaults.standard
private var exerciseEntity = ModelEntity()
private var cancellable: AnyCancellable? = nil

// MARK: - viewDidLoad()
override func viewDidLoad() {
    super.viewDidLoad()
    setupARView()
    if mode == .nonAR {
        updateEntity()
    }
}

// MARK: - entity methods
/// Updates the exercise entity: adds it to the scene or changes it.
func updateEntity() {
    print("updateEntity()")

    switch mode {
    case .nonAR:

```

```

        if showingExercise {
            arView.cameraMode = .nonAR
            arView.environment.background = .color(backgroundColor)
            if oldModelName != exercise.imageName {
                arView.scene.anchors.removeAll()
                setupSpinner()
                setupErrorLabel()
                DispatchQueue.main.asyncAfter(deadline: .now()+0.2, execute: {
                    self.loadEntity()
                })
                oldModelName = exercise.imageName
            }
        }
        case .ar:
            if isExerciseSheetPresented {
                arView.cameraMode = .nonAR
            } else {
                arView.cameraMode = .ar
            }
            if oldModelName != exercise.imageName && showingExercise &&
showingCoachingOverlay == false {
                oldModelName = exercise.imageName
                setupSpinner()
                setupErrorLabel()
                loadEntity()
            }
        }

    }

    /// Loads and shows the entity.
    private func loadEntity() {
        print("loadEntity()")
    }

```

```

networkingManager.loadExercise(exercise: exercise) { (path, error) in
    if let err = error {
        print("error occurred loading the exercise: \(err.localizedDescription)")
        DispatchQueue.main.async {
            self.errorLabel.isHidden = false
        }
    } else {
        if let unwrappedPath = path {
            switch self.mode {
            case .ar:
                self.loadAREntity(fileURL: URL(fileURLWithPath: unwrappedPath))
            default:
                self.loadNonAREntity(url: URL(fileURLWithPath: unwrappedPath))
            }
        } else {
            print("error occurred loading the animation because the path is invalid")
            DispatchQueue.main.async {
                self.errorLabel.isHidden = false
            }
        }
    }
}

/// Loads the non-ar entity.
private func loadNonAREntity(url: URL) {
    print("loadNonAREntity(): URL: \(url)")

    cancellable = ModelEntity.loadModelAsync(contentsOf: url)
        .sink(receiveCompletion: { loadCompletion in
        }, receiveValue: { modelEntity in
            self.exerciseEntity = modelEntity
        })
}

```

```

self.loadMaterials()

let nonARAnchor = AnchorEntity(world: [0, 0, 0])
self.exerciseEntity.generateCollisionShapes(recursive: true)
nonARAnchor.addChild(self.exerciseEntity, preservingWorldTransform: false)

self.addLight()
self.addBackgroundPlane(nonARAnchor: nonARAnchor)
self.addCameraToAnchor(anchor: nonARAnchor)

self.arView.scene.anchors.append(nonARAnchor)

self.exerciseEntity.availableAnimations.forEach {
    self.exerciseEntity.playAnimation($0.repeat())
}

let panGesture = UIPanGestureRecognizer(target: self, action:
#selector(self.panGesture))
self.arView.addGestureRecognizer(panGesture)

self.spinner.stopAnimating()
})
}

private func addLight() {

let directionalLight = DirectionalLight()
directionalLight.light.intensity = 1
directionalLight.light.isRealWorldProxy = true
directionalLight.light.color = .white

directionalLight.shadow = DirectionalLightComponent.Shadow()

```

```

directionalLight.orientation = simd_quatf(angle: -.pi/2, axis: [1,0,0])

let lightAnchor = AnchorEntity(world: [0, 0, 0])
lightAnchor.addChild(directionalLight)
arView.scene.addAnchor(lightAnchor)
}

private func addBackgroundPlane(nonARAnchor: AnchorEntity) {

    let mesh = MeshResource.generatePlane(width: 5, depth: 5)
    let material = OcclusionMaterial(receivesDynamicLighting: true)
    let catcher = ModelEntity(mesh: mesh, materials: [material])
    catcher.setPosition([0, 0, 0], relativeTo: self.exerciseEntity)
    nonARAnchor.addChild(catcher)

}

private func addCameraToAnchor(anchor: AnchorEntity) {
    // Virtuális kamera létrehozása.
    let cameraEntity = PerspectiveCamera()
    // A kamera elhelyezése a gyakorlathoz képest.
    let height = exerciseEntity.visualBounds(relativeTo: nil).max.y-
exerciseEntity.visualBounds(relativeTo: nil).min.y
    cameraEntity.look(at: [exerciseEntity.position.x, yValue, exerciseEntity.position.z],
from: [height*1.5, height*1.5, zValue], relativeTo: exerciseEntity)
    anchor.addChild(cameraEntity)
}

/// Loads the AR entity.
private func loadAREntity(fileURL: URL) {
    print("loadAREntity()")
    // Betöltés.
    cancellable = ModelEntity.loadModelAsync(contentsOf:
fileURL).sink(receiveCompletion: { loadCompletion in

```

```

switch loadCompletion {
case .failure(let error):
    print("error occurred loading the file: \(error.localizedDescription)")
case .finished:
    print ("finished loading the file ")
}
}, receiveValue: { modelEntity in
    self.exerciseEntity = modelEntity

    // Textúrák betöltése.
    self.loadMaterials()

    // Gesztusok ráillesztése.
    self.exerciseEntity.generateCollisionShapes(recursive: true)
    self.arView.installGestures(.all, for: self.exerciseEntity)

    if let entity = self.oldEntity {
        self.exerciseEntity.transform.scale = entity.transform.scale
        self.exerciseEntity.transform.translation = entity.transform.translation
    } else {
        self.exerciseEntity.scale = [0.7, 0.7, 0.7]
    }
    print("modelEntity.transform: \(self.exerciseEntity.transform)")
    self.oldEntity = self.exerciseEntity

    // Model megjelenítése.
    if self.oldAnchor != nil {
        self.oldAnchor!.children.removeAll([self.exerciseEntity])
    } else {
        let anchorEntity = AnchorEntity(plane: .horizontal)
        anchorEntity.addChild(self.exerciseEntity)
        self.arView.scene.addAnchor(anchorEntity)
        self.oldAnchor = anchorEntity
    }
}

```

```

        self.exerciseEntity.availableAnimations.forEach {
            self.exerciseEntity.playAnimation($0.repeat())
        }

        self.oldModelName = self.exercise.imageName
        DispatchQueue.main.async {
            self.spinner.stopAnimating()
        }
    })
}

func loadMaterials() {

    let blackColor = UIColor(red: 30/255, green: 30/255, blue: 30/255, alpha: 1)
    let grayColor = UIColor(red: 100/255, green: 100/255, blue: 100/255, alpha: 1)
    let redColor = UIColor(.red)

    var materialColors: [UIColor] = [UIColor]()
    if let exerciseModel = self.exerciseEntity.model {
        print("materials.count: \(exerciseModel.materials.count)")

        var materialColors = [getMaterialColor(materialIndex: 0),
getMaterialColor(materialIndex: 1), getMaterialColor(materialIndex: 2),
getMaterialColor(materialIndex: 3), getMaterialColor(materialIndex: 3),
solePantsColor(entityPartShouldBe: .pants), getMaterialColor(materialIndex: 6)]

        var materials = [PhysicallyBasedMaterial]()
        for color in materialColors {

            var material = PhysicallyBasedMaterial()
            material.metallic = 0.0
            material.roughness = 0.8

```

```

        material.baseColor = PhysicallyBasedMaterial.BaseColor(tint: color, texture: nil)

        materials.append(material)
    }
    self.exerciseEntity.model!.materials = materials

}
}

private func getMaterialColor(materialIndex: Int) -> UIColor {

    var defaultsColor: UIColor {
        switch materialIndex {
        case 0:
            return UIColor(red: 230/255, green: 185/255, blue: 152/255, alpha: 1)
        case 6:
            return UIColor(red: 240/255, green: 240/255, blue: 240/255, alpha: 1)
        default:
            return UIColor(red: 50/255, green: 50/255, blue: 50/255, alpha: 1)
        }
    }

    if let hexValue = defaults.string(forKey: "materialColor\\(materialIndex)") {
        return UIColor(hex: hexValue) ?? defaultsColor
    } else {
        return defaultsColor
    }
}

// MARK: - shared UI methods
private func setupSpinner() {
    print("setupSpinner()")
    spinner.translatesAutore sizingMaskIntoConstraints = false
    spinner.startAnimating()
}

```

```

view.addSubview(spinner)

spinner.centerXAnchor.constraint(equalTo: view.centerXAnchor).isActive = true
spinner.centerYAnchor.constraint(equalTo: view.centerYAnchor).isActive = true
}

private func setupErrorLabel() {
    print("setupErrorLabel()")

    errorLabel.isHidden = true
    errorLabel.translatesAutoresizingMaskIntoConstraints = false
    view.addSubview(errorLabel)

    errorLabel.centerXAnchor.constraint(equalTo: view.centerXAnchor).isActive = true
    errorLabel.topAnchor.constraint(equalTo: spinner.bottomAnchor, constant: 8).isActive =
true
    errorLabel.text = "check your internet connection"
}

private func setupARView() {
    print("setupARView()")

    switch mode {
    case .nonAR:
        arView.cameraMode = .nonAR
        arView.environment.background = .color(.clear)
    case .ar:
        if isExerciseSheetPresented {
            arView.cameraMode = .nonAR
        } else {
            arView.cameraMode = .ar
        }
    }
}

```

```

arView.translatesAutoresizingMaskIntoConstraints = false
view.addSubview(arView)
NSLayoutConstraint.activate([
    arView.bottomAnchor.constraint(equalTo: view.bottomAnchor),
    arView.topAnchor.constraint(equalTo: view.topAnchor),
    arView.leadingAnchor.constraint(equalTo: view.leadingAnchor),
    arView.trailingAnchor.constraint(equalTo: view.trailingAnchor)
])

if mode == .ar {
    let arConfig = ARWorldTrackingConfiguration()
    arConfig.planeDetection = [.horizontal]
    arConfig.environmentTexturing = .automatic
    if
ARWorldTrackingConfiguration.supportsFrameSemantics(.personSegmentationWithDepth) {
        arConfig.frameSemantics =
ARConfiguration.FrameSemantics.personSegmentationWithDepth
    }
    arView.environment.sceneUnderstanding.options = .occlusion
    if ARWorldTrackingConfiguration.supportsSceneReconstruction(.mesh) {
        arConfig.sceneReconstruction = .mesh
    }

    arView.session.run(arConfig)

    addCoaching()

    DispatchQueue.main.asyncAfter(deadline: .now()+2.0) {
        if self.isCoachingOverlayAdded == false {
            self.showingCoachingOverlay = false
            self.isCoachingOverlayAdded = true
            self.updateEntity()
        }
    }
}

```

```

    }
}

}

// MARK: - AR specific
private func addCoaching() {
    print("addCoaching()")

    let coachingOverlay = ARCoachingOverlayView()
    coachingOverlay.delegate = self
    coachingOverlay.session = arView.session
    coachingOverlay.autoresizingMask = [.flexibleWidth, .flexibleHeight]

    coachingOverlay.goal = .horizontalPlane
    arView.addSubview(coachingOverlay)
}

func coachingOverlayViewDidRequestSessionReset(_ coachingOverlayView:
ARCoachingOverlayView) {
    print("coachingOverlayViewDidRequestSessionReset()")
}

func coachingOverlayViewDidDeactivate(_ coachingOverlayView:
ARCoachingOverlayView) {
    print("coachingOverlayViewDidDeactivate()")
    showingCoachingOverlay = false
    isCoachingOverlayAdded = true
    updateEntity()
}

func coachingOverlayViewWillActivate(_ coachingOverlayView:
ARCoachingOverlayView) {

```

```

    print("coachingOverlayViewWillActivate()")
    isCoachingOverlayAdded = true
    showingCoachingOverlay = true
}

// MARK: - non-ar specific
@objc func panGesture(sender: UIPanGestureRecognizer) {

    let senderView = sender.view
    let translation = sender.translation(in: senderView)

    let changeX = Float(translation.x - beginPointX)

    switch sender.state {
    case .began:
        break
    case .changed:
        exerciseEntity.orientation = simd_quatf(angle: Float(beginPointX)+changeX/100,
axis: [0, 1, 0])
    case .ended:
        beginPointX = CGFloat(Float(beginPointX)+changeX/100)
    default:
        break
    }
}
}

```