Easy Learner

Készítette: Hajas András (CN98HE)

Bevezetés

Az alkalmazás lényege, hogy egy könnyen kezelhető felületet adjon a tanárok és diákok különóráik szervezésének céljából. Az alkalmakat gyorsan és egyszerűen meg tudjuk hirdetni tanárként, vagy akár diákként is kereshetünk magunknak tanárt. Egy listában böngészhetjük az időpontokat és akár felhalszánó értékelései alapján dönthetünk, hogy melyiket választjuk. Az összes óránkat szimplán át tudjuk nézni és le tudjuk mondani, vagy a már befejeződöttekről értékelést tudunk leadni.

A projekt nem csak az alkalmazásból áll, hogy a rendszer teljesen működőképes legyen készült hozzá egy adatbázis és egy webszolgáltatás is, amik tárolják és kezelik az alkalmazás által használt adatokat.

Az alábbi dokumentációnak két célja van. Először is, hogy egy egyértelmű leírást adjon a projekt felépítéséről és megvalósításáról, hogy ha más fejlesztőknek tovább kéne adni, ők a leírás alapján kiigazodjanak rajta és megértsék a működésének az elvét. Valamint, hogy a leírtakból meg lehessen állapítani a félév alatt a fejlesztésbe fektetett időmet és a munkám minőségét. Ebből adódik, hogy a dokumentumban vegyesen váltják egymást a szorosan a működéshez kapcsolódó részek, valamint a személyes tapasztalatok és megfontolások.

Felépítését nézve a dokumentáció eleje, a *Tervezés* cím alatt lévő részek a valós munka elejét tartalmazzák, ahol eldöntöttem a használni kívánt eszközöket, megterveztem a projekt architektúráját a nagyobb komponensek felelőssége alapján, majd megfogalmaztam a használati eseteket, majd egy ezekhez jól használható adatmodellt terveztem.  
Ezután komponensenként vizsgálom az alkalmazást az adatbázistól a webszolgáltatáson keresztül az alkalmazásig haladva. A szolgáltatásnak bemutatom a kódszervezését, az publikált API végpontjait, az adatbázis műveletek megvalósításának módjait és néhány érdekesebb problémát, amivel találkoztam. Majd végül az android alkalmazásban is végig tárgyalom az osztályok szervezését és felelősségeiket, valamint érdekesebb megfontolásokat a megvalósításukban, majd utána egy külön részben a kinézet tervezésével töltött munkámat és szempontokat, amiket figyelembe vettem a megvalósítás során.  
Végül egy a félévben végzett munkám összefoglalásával és a teljesítmény értékelésével, valamint a saját tapasztalataim megfogalmazásából álló rész zárja az dokumentációt.

Tervezés

Felhasznált technológiák

Egy rövid összefoglaló a feladat megvalósítása során alkalmazott programokról és fontosabb technológiákról, hogy

Az adatbázis egy **Oracle Database** példány (<https://www.oracle.com/database/>),  
amit a szintúgy Oracle által kiadott **SQL Developer**rel (<https://www.oracle.com/database/technologies/appdev/sql-developer.html>) szerkesztettem.

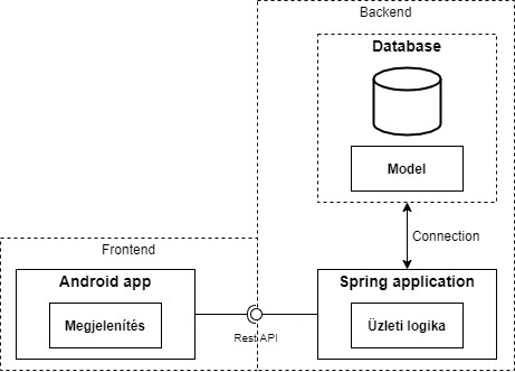
A webszolgáltatást **Spring Boot**tal (<https://spring.io/projects/spring-boot>) készítettem Java nyelven, a sok ismétlődő kódolást elkerüléséért a **Lombok** (<https://projectlombok.org/>) könyvtárat is használtam.

Az android alkalmazás **Kotlin** (<https://kotlinlang.org/>) nyelven készült. Fontosabb segéd könyvtárak közé tartozik a **Glide** (<https://bumptech.github.io/glide/>) képek betöltéséért, valamint a **Retrofit** (<https://square.github.io/retrofit/>) hálózati hívások kezeléséért felelős.

Teszteléshez és egyéb tervezéshez szükséges programokat nem, csak a projekthez közvetlen kapcsolódó technológiákat soroltam fel.

Architektúra

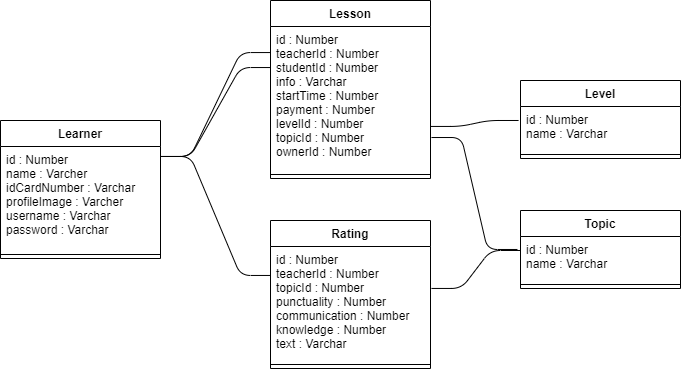
Magának az **applikáció** csak az adatok igény szerinti, rendezett megjelenítése és a felhasználói interakciók kezelése, továbbítása a szolgáltatás felé. Ez lesz az egyszerű **frontend**.

Szükség van egy **adatbázisra**, ami az adatok robosztus tárolásáért felelős. Erre épül majd egy **webszolgáltatás**, ez tartalmazza az üzleti logikát és összeköti felhasználói felületet az adatokkal. A rendszer e két komponense alkotja a **backendet**.  
A webszolgáltatás kapcsolódik az adatbázishoz és publikál egy REST API-t az alkalmazás felé. 

Amikor jön egy input a felhasználótól, például gombnyomás az alkalmazás egy kérést intéz az API megfelelő végpontja felé. A kérést a szolgáltatás feldolgozza, az adatbázishoz fordul az éppen érdekes adatok módosításáért vagy lekéréséért, ha szükség van rá a kérés szerint manipulálja az eredményhalmazokat, majd az alkalmazásnak megfelelő formában választ küld.

Adatmodell

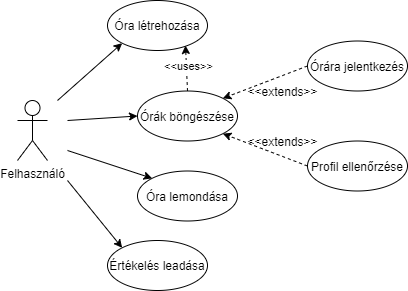
A szükséges adatstruktúra kialakításakor törekedtem a minél egyszerűbb megoldásra, hogy semmiképpen ne kerüljenek az adatbázisba redundáns adatok.



Az entitások és néhány fontosabb tulajdonságaik:

* **Learner**: a felhasználók adatait tárolja
  + ProfileImage: a felhasználó profilképének elérési útja a fájlrendszerben
  + Password: a felhasználó jelszava, titkosítva van tárolva
* **Lesson**: az órákról táról minden adatot
  + TeacherId, StudentId: a tanár és diák szerepeket betöltő felhasználó azonosítói
  + StartTime: az óra kezdési ideje, milliszekundumban
  + OwnerId: az órát létrehozó felhasználó azonosítója
  + LevelId, TopicId: a végzettségi szint és tantárgy azonosítója
* **Rating**:
  + TeacherId: az értékelt felhasználó azonosítója
  + Punctuality, Communication, Knowledge: az értékelések három szempontja egytől ötig terjedő szám
* **Level**: végzettségi szinteket felsorakoztató tábla
* **Topic**: a különböző tantárgyakat felsorakoztató tábla

Használati esetek

Az alkalmazás leírásából könnyedén megfogalmazhatjuk a használati eseteket. Egy falhasználó **böngészni tudja az órákat**, eközben a tevékenység közben az órákat hirdető felhasználóknak **megnézheti a profilját** és **jelentkezhet**, ha valami éppen neki megfelelő. **Létre tudja hozni a saját óráit**, ami elengedhetetlen a többi tevékenység végzéséhez. Ezen kívül, **le tud iratkozni** egy óráról, ha úgy dönt, mégsem felel meg, valamint a már befejeződött alkalmai után **értékelést tud adni** a másik felhasználóról, akivel közösen dolgozott.

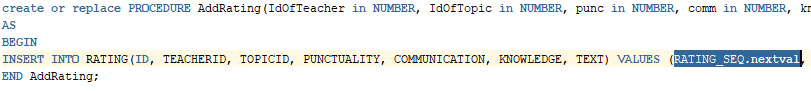
Backend

Adatbázis

Az adatbázis sémája a korábban bemutatott adatmodellel megegyezik. Ennek a szerepe az adatok megbízható és hosszútávú tárolása.

Annak érdekében, hogy a különböző táblákba az új elem felvételekor az azonosító mezők egyediek maradjanak és lehessen lekérdezésekhez használni, tárolt eljárásokat készítettem. Ez azért így lett megvalósítva, hogy szekvenciákat tudjak definiálni, ami sorban generál egymást követő értékeket és ezeket adjam azonosítóként az új rekordnak. Ezek az **Add** szóval kezdődő eljárások.





Ezen kívül tartalmaz további két eljárást, ezek a tanórák tábláját módosítják foglalás érkezésekor, ezek neve elején a **Book** szó áll.

Webszolgáltatás

Szervezés

A szolgáltatás részei csomagok szerinti lebontásban.

* **Entities**: mindegyik az adatbázisban lévő tábláknak megfelelő objektumok
* **Repository**: ezek az interfészek oldják meg az adattárolást és az adatok elérését az adatbázisból, minden entitásnak van hozzá kapcsolódó repositoryja
* **Controller**: a hálózati kéréseket alakítja át metódusokká, amik a műveleteket végzik, továbbá ezekben az osztályokban vannak definiálva az API végpontjai, minden entitáshoz tartozik saját controller
* **Dto**: el tudja fedni az entitások implementációját, ha ilyen típust adunk vissza entitás helyett, ha nem csak egy entitás tulajdonságait küldi vissza a szolgáltatás egy Dto-ban összefoglalja az abban a kérésben releváns tulajdonságokat, minden entitáshoz tartozik Dto, az érdekes osztályok:
  + LearnersProfileDto: egy felhasználó profiljának megjelenítéséhez szükséges adatok
  + LessonWithNamesDto: az óra entitások id tulajdonsága helyett a nevüket tartalmazza
  + RatingWithNamesDto: értékeléseket tartalmaz az id tulajdonság helyett elnevezésekkel
* **Mapper**: a mapper feladata lenne, hogy az entitásokat Dto-vá alakítsa. A szolgáltatás egyszerűsége miatt ezek jelenleg nincsenek bekötve a programba, ám a tervezési megfontolások miatt a projektben maradtak
* **Service**: ezek az osztályok tartalmazzák az üzleti logikát, itt intézi a program a konkrét lekérdezéseket az adatbázis felé, itt végzi el a szükséges módosításokat a kapott eredményhalmazokon és itt állítja össze a válasz tartalmát a kéréshez
* **Resources > Images**: az tárolt profilképek helye

API végpontok

Az végpontokig útközben több csomópont található az alapján, hogy éppen milyen témán belül folytatjuk a kérést.

**BaseUrl**

* **User**: a felhasználókhoz kapcsolódó kérések

**GET** - az összes rekordot visszaadja a Learners táblából

* + **Login**

**GET** - próbálkozás a bejelentkezéssel

* + **Register**

**POST** - regisztráció

* + **Pic/{id}**

**GET** - profilkép lekérése

* **Level**: a végzettségi szintekhez kapcsolódó kérések

**GET** - összes rekordot visszaadja a Levels táblából

* **Topic**: a tantárgyakhoz kapcsolódó kérések

**GET** - összes rekordot visszaadja a Topics táblából

* **Lesson**: a tanórákhoz kapcsolódó kérések

**GET** - össze rekordot visszaadja a Lessons táblából

* + **My**: a saját órákhoz kapcsolódó kérések
    - **Booked/{id}**

**GET** - saját lefoglalt órák

* + - **Finished/{id}**

**GET** - saját befejeződött órák

* + - **Free**: a szabad saját órák
      * **Teacher/{id}**

**GET -** saját tanárként hírdetett szabad órák

* + - * **Student/{id}**

**GET -** saját diákként hírdetett szabad órák

* + **Names**: órák lekérdezése azonosítók helyett nevekkel

**GET** - összes ilyen óra

* + - **Student**

**GET** - diákoknak elérhető órák

* + - **Teacher**

**GET** - tanároknak elérhető órák

* + **Add:** órák létrehozása
    - **Teacher**

**POST -** óra létrehozása tanárként

* + - **Student**

**POST -** óra létrehozása diákként

* + **Book**: óra lefoglalása
    - **Teacher**

**PUT -** óra lefoglalása tanárként

* + - **Student**

**PUT -** óra lefoglalása diákként

* + **Cancel/{lessonId}**

**PUT** - óra lemondása

* **Rating**: az értékelésekhez kapcsolódó kérések
  + **{id}**

**GET** - egy felhasználóhoz tartozó értékelések visszaadása

* + **Profile/{id}**

**GET –** egy felhasználó értékeléseit adja vissza a profil megjelenítéséhez szükséges adatokkal

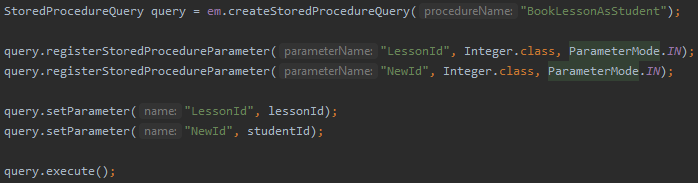
* + **Add**

**POST** - értékelés adása felhasználónak

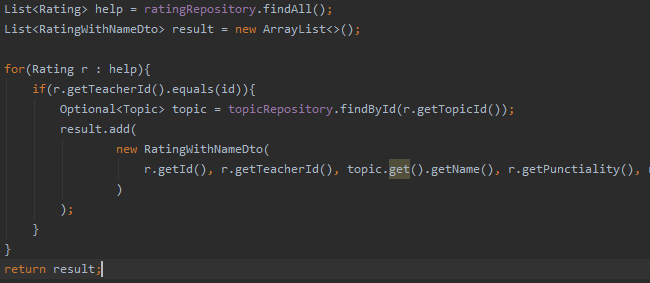
Adatbázis műveletek

Az alkalmazásban három különböző fő módot használok az adatbázis műveletek végrehajtásához.

1. Meghívok egy az adatbázisban lévő tárolt eljárást, ilyenkor csupán a felhasználóktól kapott paramétereket be kell regisztrálni a lekérdezéshez és futtatni kell a tárolt eljárást.



1. A repository által biztosított interfész findAll() metódusát használva először az adott típusú entitás minden előfordulását elkérem az adatbázistól, majd ezen az eredményhalmazon végzek átalakítást vagy szelektálok.



1. Egy teljes egész SQL parancsot építek a szolgáltatásban és ezt meghívom az adatbázison, a kérésben kapott paramétereket itt is be kell regisztrálni az utasításba.



Érdekesebb problémák

Fejlesztés közben egy érdekes hibába ütköztem, habár a mind az android alkalmazásban, mind a szolgáltatásban a java.util.Calender osztályt használtam idő kezelésére nagyon sokáig mégsem az elvárt viselkedés szerint működött a rendszer. Észrevettem, hogy az egyik verzió 1970-tól a másik 1900-tól számolja az eltelt időt milliszekundumban, azonban az androidon létrehozott időpont mindig 1900 évvel a valós idő elött járt a szolgáltatásban. Ahol szükséges idő összehasonlítás miatt, ott ez a különbség korrigálva van.

Regisztráláskor a felhasználó a paraméterben küldött adatai mellett elküldi a kérés testében egy byte tömbben a regisztrációkor készített képet. Ezt a tömböt a szolgáltatás beolvassa egy kép változóba, majd ezt kiírja ezt a fájlt egy mappába a többi kép mellé. Amennyiben ez nem sikerül, nem lesz felvéve az új felhasználó az adatbázisba. Ha sikeres, akkor az adatbázisba csak a képnek az elérési útvonalát tölti fel. Kép letöltésekor az útvonal alapján megkeresi a képet és ezt a válasz testében adja vissza, ezt tudja kezelni a Glide könyvtár képek letöltéséhez az appban.

Frontend

Android alkalmazás

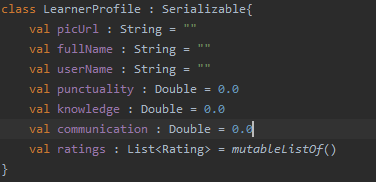
Szervezés és felelősségek

Az applikáció forráskódját szétosztottam, hogy az azonos felelősségű dolgok egy csomagban legyenek. A felhasználói felületen belül aszerint próbáltam meg rendszerezni, hogy az adott osztályok a nézetek közül melyiken jelennek meg, vagy vannak velük szoros kapcsolatban.

Tulajdonképpen ez a komponens az adatok megjelenítéséért felelős tehát az egyes részek nem tartalmaznak többet navigációnál, a megjelenítendő adatok nézetre illesztésénél és a felhasználói interakcióhoz szükséges eseménykezelők beállításával, hogy a megfelelő API végpontot hívják.

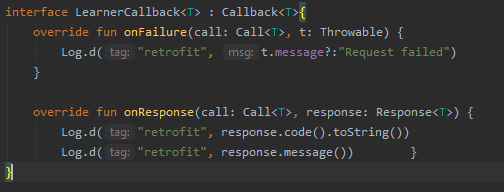
**Model**: az alkalmazásban megjelenő objektumok tulajdonságaikkal együtt, ez nagyrészt az adatbázisban lévő adatszerkezetet tükrözi, kis eltérésekkel. Azon osztályok, amelyeket nézetek között adok át implementálják a *Serializable* interfészt

* **Learner**  
  **Level**  
  **Rating**  
  **Topic**
  + Adatbázishoz hasonló
* **Lesson** - az adatbázishoz képest ki van egészítve *string* tulajdonságokkal a tanár és a hallgató nevének tárolásához
* **LearnerProfile** - adatbázisban nem szerepel, a felhasználó profiljának megjelenítéséhez szükséges adatokat foglalja össze egy objektumban



**Retrofit**: a hálózati kommunikációhoz szükséges retrofit elemeket tartalmazza

* **RetrofitClient** -ez egy singleton osztály, hogy ne kelljen minden nézeten példányosítani hívások küldéséhez. Tartalmaz továbbá egy *LearnerCallback* interfészt, ami implementálja a *Retrofit Callback* függvényeit és feljegyzi a kérések eredményét, így a kéréseknél nem kell mindig újra megírni egy *Callback* objektumot
* **RetroLessons** – a korábban felsorolt API végpontok hívásához készült interfészeket tartalmazza





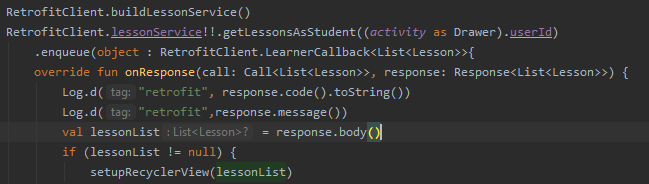
**Ui**: a megjelenítéshez szükségez nézeteket tartalmazza

* **Drawer** - *Activity*, az alkalmazás fő nézeten, itt tudjuk böngészni az órákat. A fix része az oldalt előugró *Navigation Drawer*, amúgy az egész képernyő egy keret, amiben cserélgeti a *Fragmenteket*. A drawerből való navigálás útjai:
  + Learn.Lessons -> LearnLessonFragmet
  + Learn.Lecture -> üres fragment
  + Teach.Lessons -> TeachLessonFragment
  + Teach.Lecture -> üres fragment
  + User.MyProfile -> MyProfileFragment dialógus felugrik
  + User.MyLessons -> MyLessonsFragment

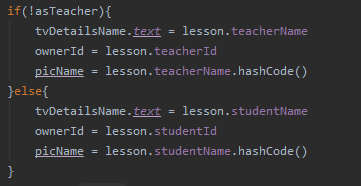
**>Lecture**: az előadásokhoz tartozó csomag, jelenleg nincs funkciója az alkalmazásba, a jövőbeli fejlesztések miatt maradt a forráskódok között

**>Lesson**: az órákhoz tartozó nézeteket tartalmazza

* **LearnLessonFragment** – az egyik *fragment* ami a fő *activityre* kerül, ez a diákoknak szóló órákat jeleníti meg egy *RecyclerViewban*. Az osztály továbbá implementálja a *LessonAdapter OnLessonClicked* interfészét a diákoknak szükséges viselkedésekkel megvalósítva. Az új óra felvétele gombbal diákként léphetünk az AddLessonActivityre
* **TeachLessonFragment** – a viselkedésének a mintája azonos az előző *LearnLessonFragmentével* annyi különbséggel, hogy itt mindent a tanár nezőpontjából látunk

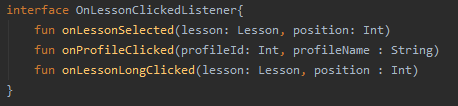


* **AddLessonActivity** – az órák felvételére szolgáló nézet. Vár egy *asTeacher* paramétert, ami alapján kiderül neki, hogy tanárként vagy diákként nyitottuk meg a nézetet, ez alapján dönti el, hogy az API melyik végpontját hívja a létrehozó gomb. Tartalmaz egy *DatePickerDialogot* és egy *TimePickerDialogoat*, amiknek implementálja az időválasztas interfészét. Egy *spinnerből* választhatunk szintet és tárgyat, aminek a tartalmát a szolgáltatástól kéri az alkalmazás, azonban ikonokat statikusan rendel hozzá
* **LessonDetailsActivity** – a főnézeten történt lista klikkelés után juthatunk ide, ez a nézet paraméterként megkapja a szóban forgó óra adatait és ez alapján tölti fel adatokkal a képernyőt. Hasonlóan kap egy *asTeacher* jelzőt, ami megszabja a melyik tulajdonságokat kell megjeleníteni, ami kérést intéz, az a profilkép betöltése a *glide* könyvtár segítségével. A profilképre vagy névre klikkelve előhozhatjuk a *MyProfileFragmentet* az adott felhasználó adataival



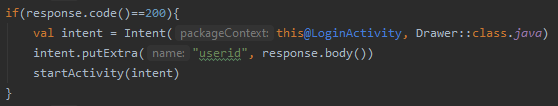
**>Adapter**: a *RecyclerView* megjelenítéséhez szükséges adaptereket tartalmazza

* **LessonAdapter** – az órák *adaptere* szintúgy tartalmazza a már megszokott *asTeacher* jelzőt, hogy az óráról a megfelelő adatokat jelenítse meg. Publikál egy interfészt a listaelemeken való hosszú és rövid érintésről, valamint külön kezeli, ha a profilképet vagy a nevet érintjük
* **RatingAdapter** – egy egyszerű *RecyclerView.Adapter* implementáció az értékelések megjelenítéséhez

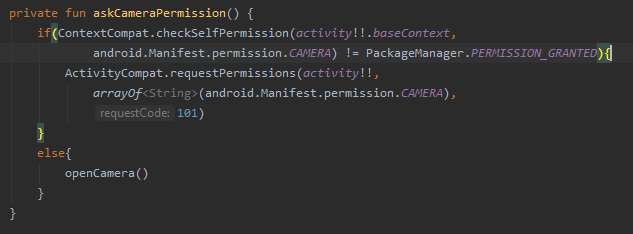


**>Login**: a bejelentkezéshez és a regisztrációhoz tartozó nézeteket tartalmazza

* **LoginActivity** – a bejelentkezéshez tartozó nézet, próbálkozhatunk a bejelentkezéssel, amiben kódolva küldöm a jelszót. Amennyiben sikeres a bejelentkezés a válaszban megkapjuk a bejelentkezett felhasználónak az azonosítóját, amit tovább adunk a fő nézetnek, mert szükség van rá a legtöbb hálózati híváskor. Innen átléphetünk a *RegisterActivityre*

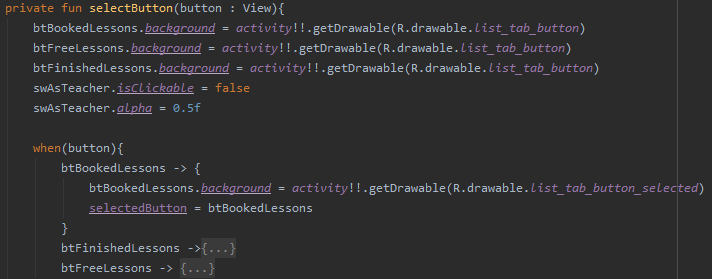


* **RegisterActivity** – ez a nézet vezérli a regisztrációt, két *fragment* között vált, valamint tartalmaz egy *newLearner* mezőt, hogy a *fragmentek* közösen férjenek hozzá az új felhasználó adataihoz, amivel majd regisztrálunk
* **RegInfosFragment** – egy egyszerű nézet, az első *fragment* a *RegisterActivityn* itt csupán megadhatjuk az alap adatokat és felszólítjuk az *activityt* a következő fragmentre navigálásra
* **RegPhotoFragment** – a második *fragment* a *RegisterActivityn*, egy profilképet készítünk, majd utána elindíthatjuk a regisztrációt. A fénykép készítéséhez a kamera használathoz futásidőben kell engedélyt kérni a felhasználótól



**>Profile**: a felhasználók profiljával és értékeléseivel foglalkozó nézeteket tartalmazza

* **AddRateFragment** – ez egy egyszerű *dialog*, amit a *MyLessonsFragmentről* érhetünk el, ha már befejezett óránkra hosszan kattintunk. Mindössze megválaszthatjuk az értékelésünket és ezt elküldhetjük a szolgáltatásnak
* **MyProfileFragment** – ez is egy egyszerű *dialog*, ami paraméterben várja a megjelnítendő adatokat, kizárólag a kép lekéréséhez intéz egy kérést. A profilhoz tartozó értékeléseket itt is egy *RecyclerViewban* látjuk
* **MyLessonsFragment** – ez egy összetetteb nézet, ahol a felhasználó a saját óráinak az állapotát tudja kezelni. Egy *RecyclerView* szerepel a központi részében, aminek a tartalmát a fölötte lévő kapcsolóval és gombokkal változtathatjuk. Egy *selectedButton* tulajdonság tartalmazza az éppen kiválasztott gombot, lenyomáskor frissül az érték és az aktuális gomb kiemelődik. A listán történő érintések által kiváltott események a kiválasztott gomb és a kapcsoló állapotától függ. Ez alapján lemondhatunk órát, vagy értékelést adhatunk le.



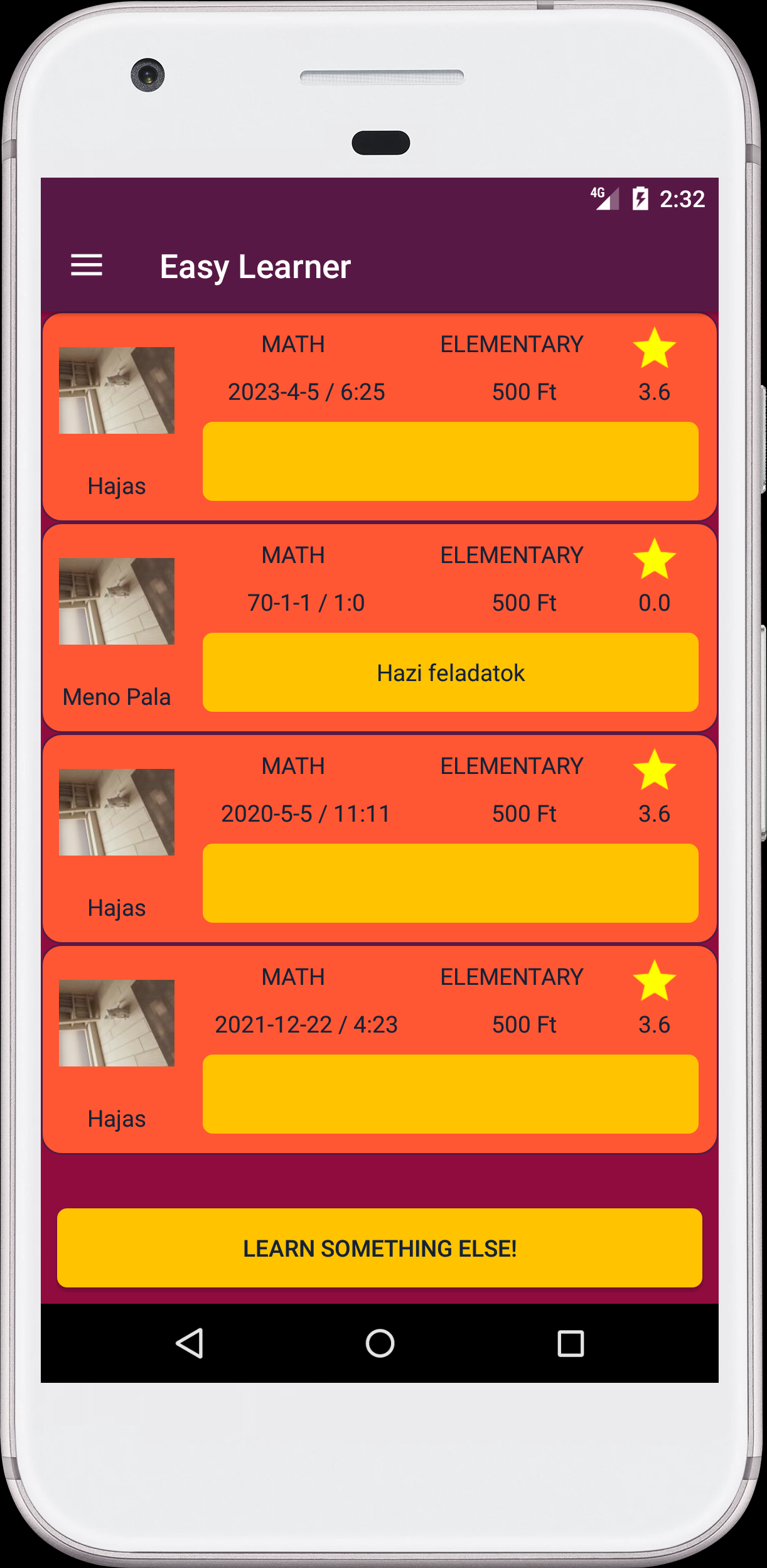
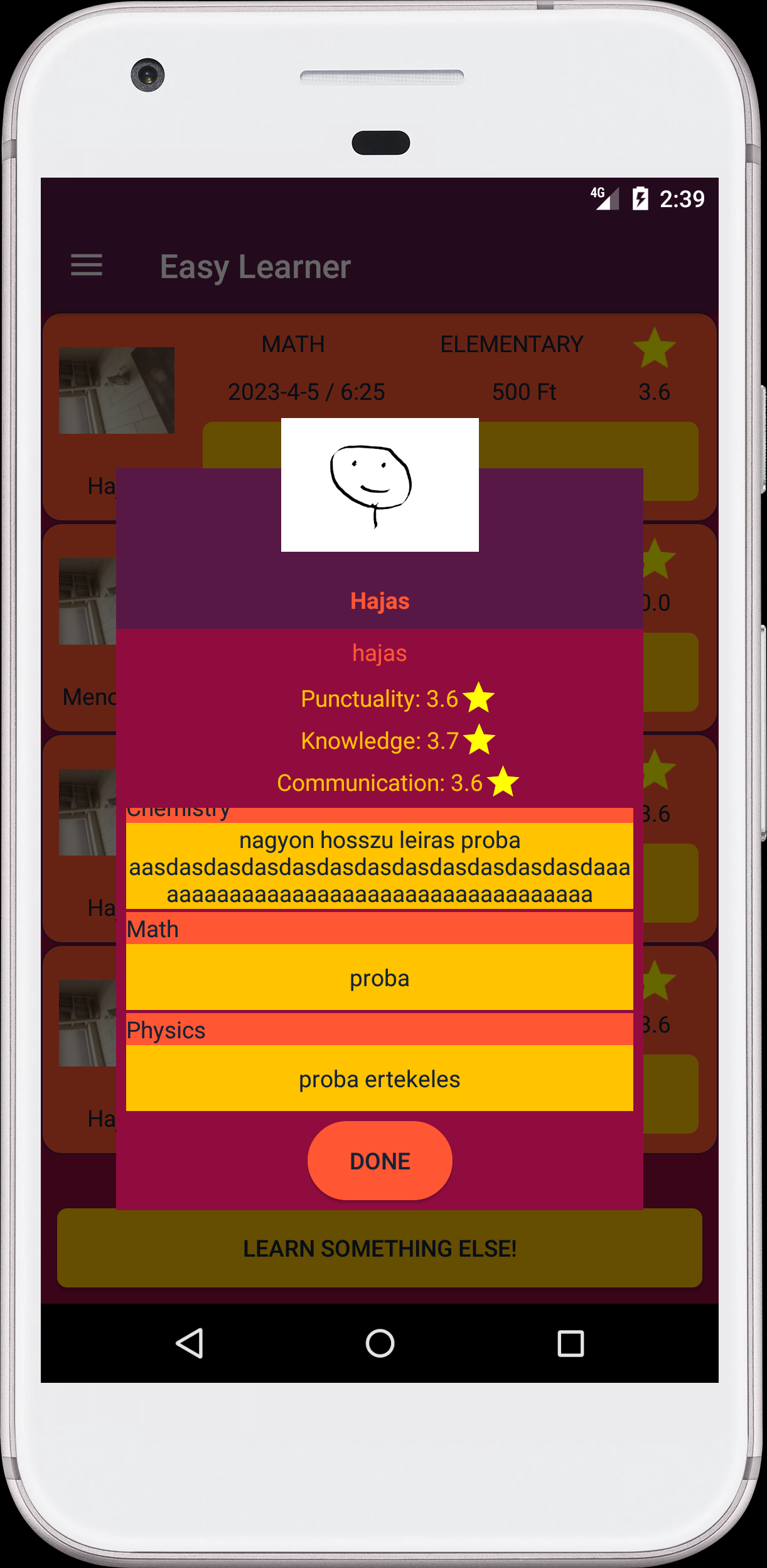
Megjelenés

A féléves munka során megpróbáltam nagyobb figyelmet fordítani arra, hogy az alkalmazás könnyen és egyszerűen kezelhető legyen, valamint stílusa és kinézete is igényes és letisztult legyen.

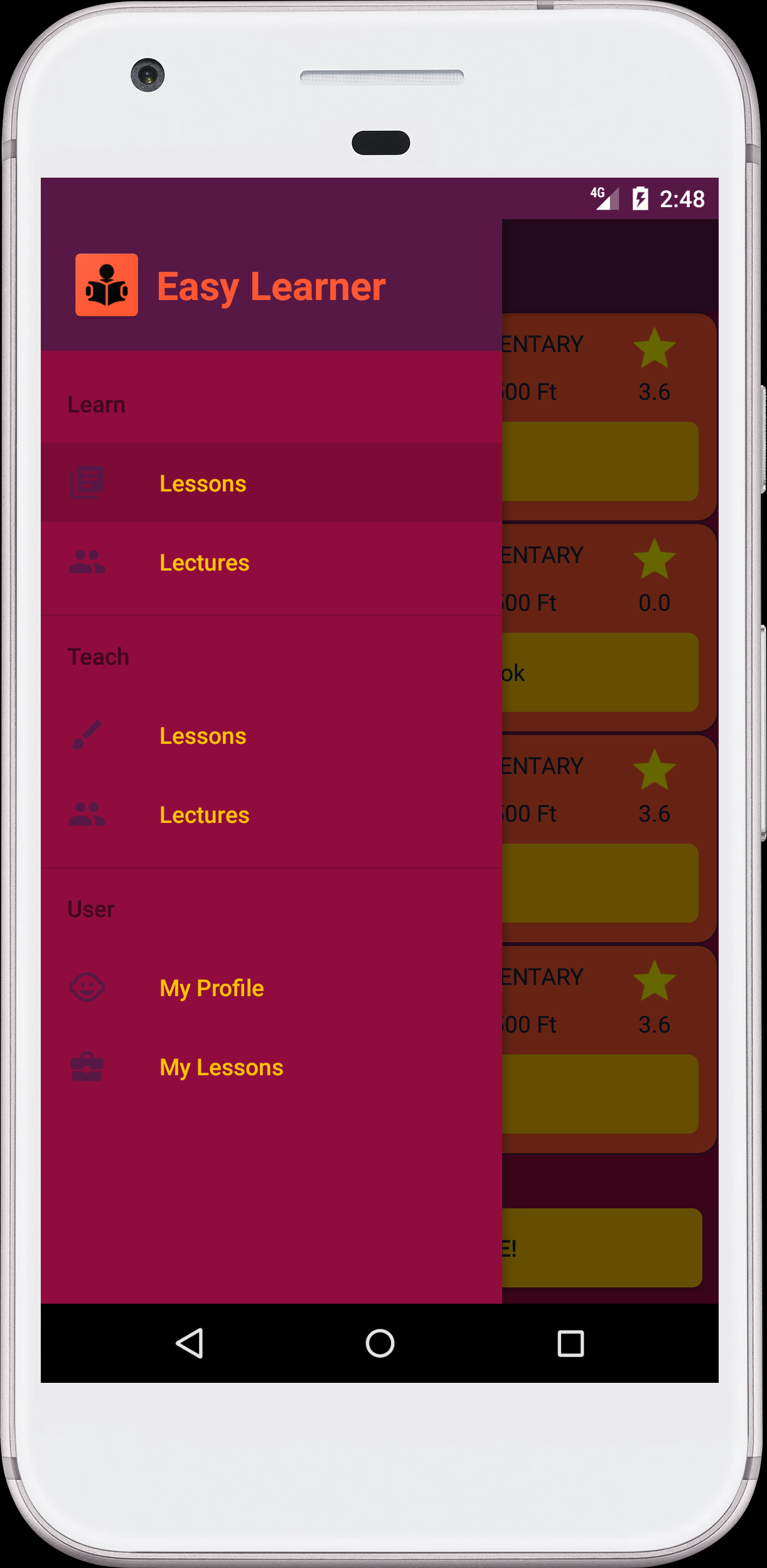
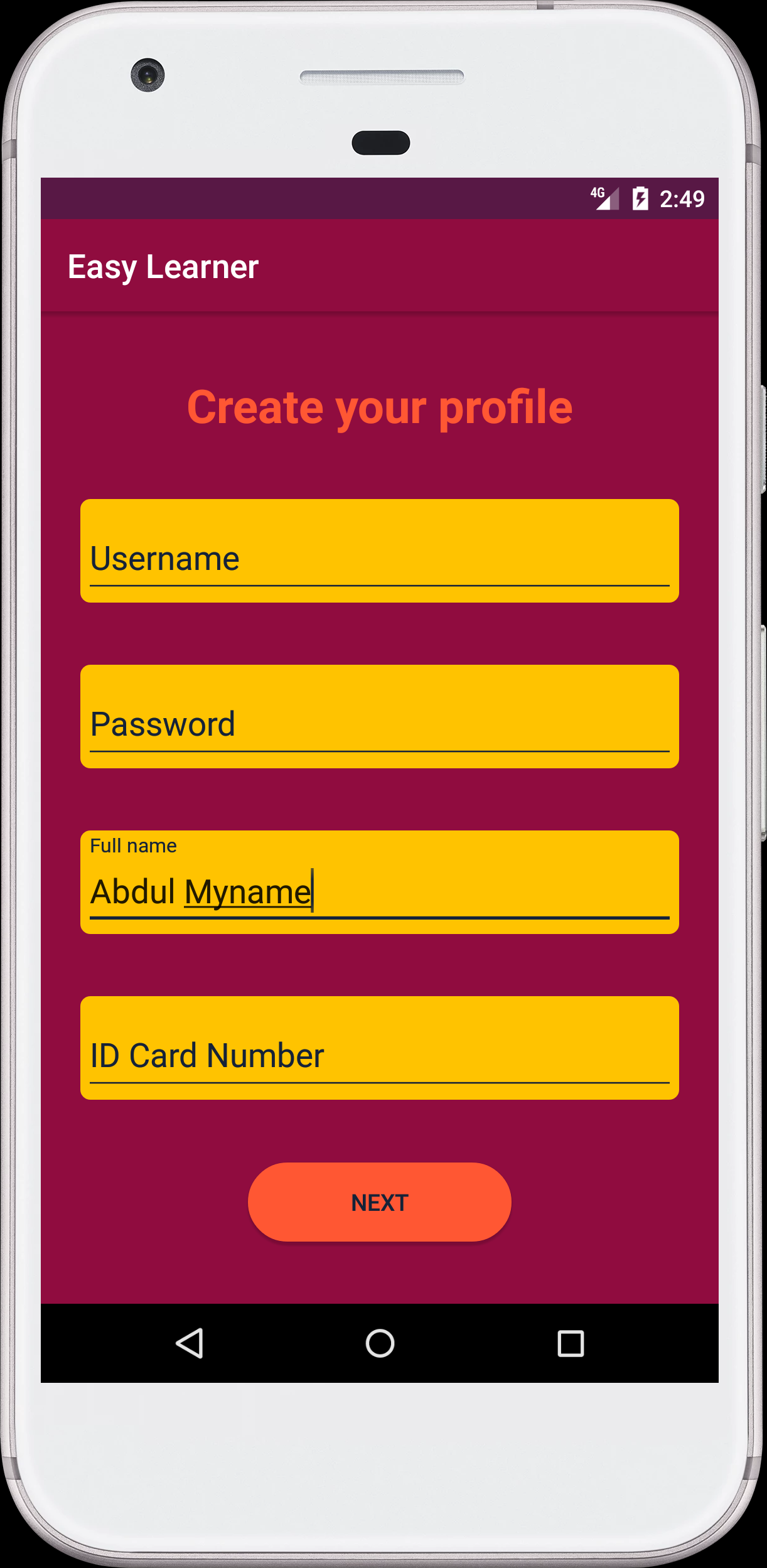
A dizájn tervezésé színválasztással kezdődött, megpróbáltam ajánlott egymás mellé passzoló színeket választani, amiknek a színpszichológiai jelentésük is pozitív dolgok.



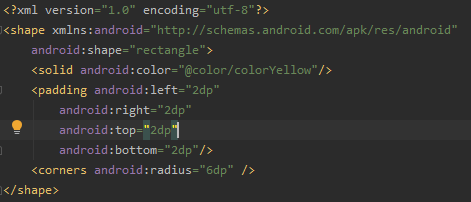
Ezután kezdődött a nézetek és tartalmuk megtervezése. Próbáltam úgy megvalósítani, hogy egy nézethez egy fő cselekmény tartozzon, amit ott végzünk. A nézeteket ezen kívül újrahasznosíthatóra és hasonlóra terveztem, hogy a felhasználónak ne kelljen minden új nézettel ismerkednie. Például ugyanaz a profil nézet nyílik meg, mindegy kiét nyitjuk meg és milyen helyről, vagy az órák böngészésekor a listák és a helyzetük mindig hasonló, csak különböző tartalmakat érhetünk el benne.

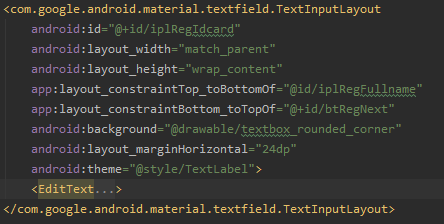
 

További szempontok voltak az egyértelműség és az egyszerűség. Ezek alapján próbáltam a felhasználónak mindig a tudatára adni, hogy éppen mit kell csinálni. Például általában azok a nézet elemek, amikkel interakcióba tudunk lépni, vagy ahol a számunkra legfontosabb információt találhatjuk vagy amikor valamit ki kell tölteni a felhasználó utasítva van mit hova írjon, vagy hol állítsa. Ezen kívül próbáltam a legnépszerűbb vagy leginkább ismert általános elemeket használni. Ezen megfontolás miatt használok NavigationDrawert, ami majdnem az összes alap android alkalmazásban is szerepel és alapból kezeli a csúsztatás mozdulatot, továbbá ezt még egy hamburger menü ikonnal is meg lehet nyitni, amiről minden felhasználó tudja, hogy egy navigációs panelt fog takarni. Ilyen elem például a fő képernyő alján a gomb is, ami sárga színnel ki van emelve, valamint méretében is nagyobb a többi gombnál, mert amikor szembe találja magát a felhasználó a nézettel valószínűleg a legszembetűnőbb elemet fogja megérinteni, aminek a hatására létre is hozhat saját órát.

A feladat ezen részéhez hozzátartozott, hogy a nézeten lévő kisebb elemeknek, mint például a gomb vagy a szövegdoboz egyedi stílusok készüljenek, mert általában sokkal esztétikusabban néznek ki a lekerekített sarkú elemek. Továbbá az is, hogy egyértelmű ikonok, akár piktogramok legyenek az egyes részekhez rendelve, amik konzisztensek a környezettel, amiben megjelennek.

 A felületek kialakításakor a legtöbb esetben ConstraintLayoutot használtam, mert ez a leginkább ajánlott összetett nézetek szerkesztésekor. Próbáltam sűrűség független nézeteket kialakítani, akár úgy, hogy konkrétan megadott méretek helyett az match\_parent vagy a warp\_content értékeket írtam, de úgy is, hogy az elemek közötti kényszerek szabják meg az egyes elemek nagyságát.



A layoutok elnevezése a hozzá tartozó osztály nevével majdnem megegyezik, az alábbi konvenció által történik az átalakítás. Ha van egy AddLessonActivity nevű osztályom a hozzá tartozó layout fájl neve activity\_add\_lesson lesz, hasonlóan a MyProfileFragment nevű osztályhoz tartozó layout fájl a fragment\_my\_profile lesz. Továbbá a lista elemek kinézetét tartalmazó elemek a row\_\* mintára illeszkedő névvel fájlok.

Összefoglalás

A féléves munkám során egy működőképes alkalmazást kellett megvalósítanom, teljes backend támogatással, tehát az elkészített anyagokkal az alkalmazásnak akár élesben való használatra is alkalmasnak kellett lennie. Ezalatt a munka alatt végeztem adatbázis kezelési feladatokat, erre építettem egy webszolgáltatást, valamint elkészült az ezen szolgáltatást használó android applikáció, aminek a kinézetét is jelenleg elfogadott és ajánlott megfontolások alapján próbáltam tervezni.

A legnehezebb részét a backend fejlesztése jelentette, ugyanis eddigi tanulmányaim és munkáim során nem fejlesztettem még webszolgáltatást és adatbázist sem szerkesztettem és használtam vezetett laborokon kívül máshol. Ebből adódóan bármilyen ebbe a kategóriába szánt fejlesztés viszonylag hosszú és nehézkes tanulás és információ gyűjtés előzte meg.  
A projekt ezen része egy elég egyszerű kis komponens lett, de mindenképpen hasznos volt ahhoz, hogy a hattér technológiákkal ismerkedhessek és a jövőben ezzel az alap tudással tudjam majd továbbfejleszteni ezt a projektet, vagy készíteni egyéb backendes szolgáltatást.

Az android applikációban elkészített részek és technológiák egy részével már találkoztam korábban, itt az ismert dolgok implementációjakor arra próbáltam meg figyelni, hogy az elkészített részek minél inkább megfeleljenek a szabványoknak és az ajánlott vagy legújabb konvencióknak. Tehát a már egyszer megvalósított elemeket próbáltam minél hatékonyabb és egyszerűbben újra írni.   
Továbbá a felhasználói nézet tervezésébe is mélyebben bele akartam menni, mert az ehhez kapcsolódó dolgokkal eddig nem nagyon foglalkoztam, általában az idő hiányában. Általános nézet tervezési elveket is tanultam, amiket mobil specifikus konvenciókkal is kiegészítettem, továbbá ehhez kapcsolódóan színpszichológiával és szoftver ergonómiával is foglalkoztam.

Összességében úgy gondolom hasznos munkát végeztem, ami valamennyire hasonlíthat egy éles alkalmazás teljes fejlesztési folyamatához. Sajnos az idő szűkössége miatt mégsem sikerült annyit kihozni a projektből, ami eredetileg tervben volt. Ehhez nagyban hozzájárult, hogy a tervezés fázisban alkotott elképzeléseimet nem mindig lehetett úgy használni, ahogy kitaláltam. Emiatt sokszor nagyobb változtatásokat kellett eszközölnöm a már megírt kódrészek között is, ami nagyban hátráltatta az egész féléves munkámat. Tanulság, hogy a következő hasonló kaliberű projektemnél biztos nagyobb hangsúlyt fogok fektetni a tervezésbe, mert azzal szebb eredményeket is el lehetett volna érni.