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

Kvíz generálási lehetőségek vizsgálata egy fullstack megoldás megvalósítása keretében

Szakdolgozat

Készítette
Muzslai László

Konzulens
dr. Ekler Péter

2024. november 16.

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott *Muzslai László*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2024. november 16.

Muzslai László
hallgató

Kivonat

A közoktatásban és a felsőoktatásban is gyakori probléma mind a tanárok, mind a diákok számára az időhiány a rengeteg munka miatt. Ez az alkalmazás a tanárok munkáját hivatott segíteni az előző félévben elkészített REST API-t felhasználva, és a hozzá írt Android-eszközökre készült program továbbfejlesztése által. Az alkalmazás lehetőséget biztosít egy nagyméretű kérdésbank létrehozására és tárolására. A kérdések bármikor módosíthatók, törölhetők, vagy hozhatók létre újak. Természetesen nem csak egy ember dolgozhat ugyanazon a tárgyon; a kérdésbank és a számonkérések közösen szerkeszthetők.

Az alkalmazás lehetőséget biztosít a kérdések létrehozása mellett egyéni témakörök létrehozására, amivel a kérdéseket és feladatsorokat lehet csoportosítani. Továbbá el lehet készíteni a saját pontrendszert, akár több félét is, amelyet dinamikusan lehet változtatni a kérdéseknél. Fontos szempont volt az automatizált javítás segítése, így csak egyszerű kérdések vannak: igaz-hamis és feleletválasztós kérdések. Sajnos az AI még nem tart ott, hogy bármilyen kézírást pontosan felismerjen, és ebből a szövegből megállapítsa annak helyességét. Ennek ellenére a szövegfelismerő funkció így is támogatja a javítást, aminek az eredményét megkapja a javító.

Ezekből az elemekből áll össze a számonkérés. Ez a szoftver csak a kérdéssorok összeállításáért és kiértékeléséért felel. Ennek megfelelően elő kell állítani magát a feladatsort. Egy dolgozatot ki lehet exportálni PDF formátumban, erről egy előnézet is lesz, amin nagyjából látszik, hogyan fog kinézni, de a végső változat csak az exportálást követően fog látszani. Ezt követően szabadon nyomtathatóvá válik.

Egy modern szoftver esetén elvárt, hogy könnyen és kényelmesen lehessen kezelni, mindenki számára a neki tetsző módon. Ennek alapján úgy döntöttem, hogy felhasználom az Android fejlesztésben szerzett tapasztalataimat. 2021 augusztusában jelent meg a Compose Multiplatform technológia, amely kedvez az Android-fejlesztőknek, mivel a natív Android-megoldások könnyen átültethetők egy cross-platform alkalmazásba. Jelenleg stabilan működik Android-, asztali- és iOS-alkalmazások készítéséhez, eszköz hiányában az első kettőt készítettem el.

Abstract

In both public and higher education, time constraints are a common issue for both teachers and students due to the large workload. This application is intended to assist teachers by utilizing the REST API developed in the previous semester and enhancing the program created for Android devices. The application allows for the creation and storage of a large question bank. Questions can be modified, deleted, or new ones can be created at any time. Of course, more than one person can work on the same subject; the question bank and the tests can be edited collaboratively.

In addition to creating questions, the application also allows for the creation of custom topics, which can be used to organize questions and assignments. Furthermore, a custom scoring system can be created, even multiple types, which can be dynamically adjusted for different questions. An important aspect was to assist in automated grading, so only simple questions are included: true/false and multiple-choice questions. Unfortunately, AI is not yet at the level where it can accurately recognize any handwriting and determine its correctness from the text. Nonetheless, the text recognition function still supports grading, and the results are provided to the grader.

These elements come together to form the assessment. This software is responsible solely for compiling and evaluating the question sheets. Accordingly, the task sheet itself must be generated. A test can be exported in PDF format, with a preview available that roughly shows how it will look, though the final version will only be visible after exporting. After this, it can be freely printed.

For modern software, it is expected to be easy and convenient to use, allowing everyone to handle it in their preferred way. Based on this, I decided to leverage my experience in Android development. The Compose Multiplatform technology, released in August 2021, is favorable for Android developers, as native Android solutions can easily be adapted into a cross-platform application. It currently works stably for creating Android, desktop, and iOS applications; due to a lack of devices, I have implemented the first two.

1. fejezet

Bevezetés

Ebben a fejezetben kitérek arra, hogy miért ezt a témát választottam. Utána bemutatom a használt Compose Multiplatform jelentősségét és elterjedtségét. A legvégén összefoglalom a dolgozat ezt követő témáit.

1.1. Témaválasztás indoklása

2023/2024 őszi félévében ismerkedtem meg a mobil, azon belül is az Android fejlesztéssel. Az ezt követő félévben tovább mélyítettem a tudásomat ebben a témában, az Önálló laboratórium tárgy keretein belül elkezdtem fejleszteni a szakdolgozatom alapjaként szolgáló alkalmazást. Szintén ebben a félévben hallgattam az Android alapú szoftverfejlesztés és a Kotlin alapú szoftverfejlesztés tárgyakat, amik segítettek jobban megérteni ezt a területet. Az így elsajátított tudás és az önálló kutatás és tanulás során előállt egy Kotlin nyelven írt REST API, ami egy PostgreSQL adatbázissal biztosít kommunikációt, és természetesen egy Android alkalmazás, amelyet a szakdolgozat során tovább bővítettem és alakítottam át egy cross-platform szoftverré.

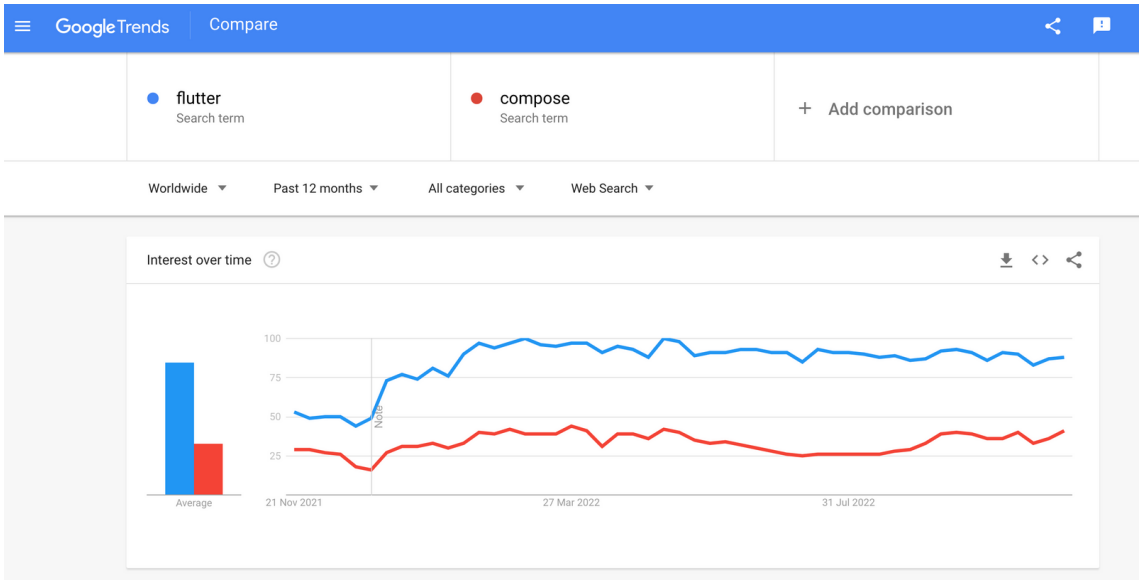
A kérdéssor összeállító alkalmazás ötletét a konzulensem vetette fel, hasznos lenne, ha létezne egy ilyen eszköz, amivel könnyen megoldható ez a feladat. Megtetszett nekem is az ötlet, mivel vannak ismerőseim és családtagjaim, akik szintén tudnának egy ilyen alkalmazást hasznosítani a munkájuk vagy egyéb elfoglaltságaik kapcsán. Egy nagyobb méretű fullstack alkalmazás előállítása túlmutat az Önálló laboratórium keretein, így rengeteg fejlesztési ötlet és lehetőség nem fért bele a félévbe. Továbbgondolva ezt a projektet, folytattam a munkát a szoftveren. Ezen kívül mindig szeretek új és érdekes dolgokat kipróbálni, és ha megtetszik, alaposan tanulmányozni és megtanulni. Pont ezért választottam a Google és a JetBrains legújabb megoldásait a szakdolgozathoz.

A most elkészített szoftvert jelentősen tovább lehet még fejleszteni, felhasználók, szervezetek regisztrálásával és elkülönítésével, több fajta kérdéstípus megvalósításával. Bevezetni a szervezeteken belül az oktató és diák csoportokat, és egy online kitöltési formát is megtervezni, létrehozni. Az Önálló laboratórium alatt is élveztem ezzel foglalkozni, és még mindig szívesen fejleszteném tovább, és tenném jobbá az alkalmazást.

1.2. Felhasznált technológia jelentősége/elterjedtsége

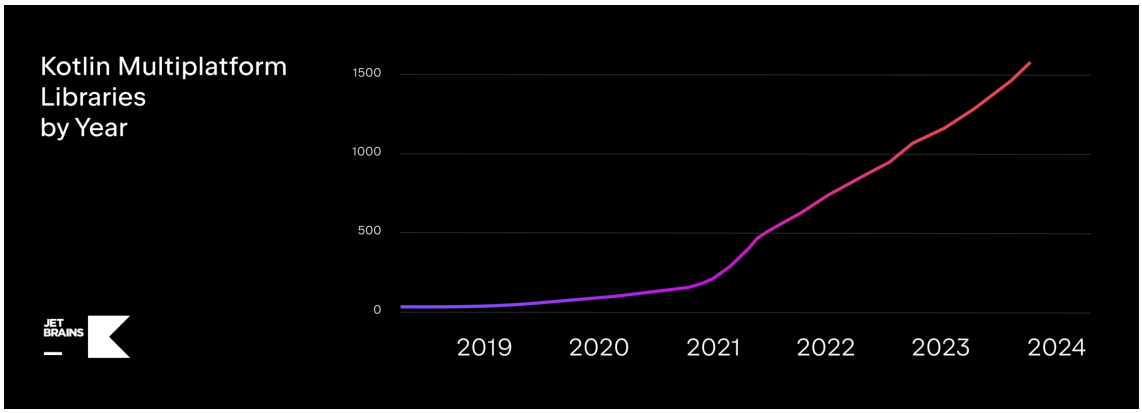
Megfigyelve a mai trendeket, láttam, hogy a multiplatform fejlesztés egyre népszerűbb, mivel gyorsabban és hatékonyabban lehet egyszerűbb alkalmazásokat elkészíteni több fajta felhasználói réteg számára. Úgy döntöttem, hogy kipróbálok egy új megoldást, ami a Compose Multiplatform; 2021 augusztusában jelent meg az 1.0 alpha verziója, és ezt a Kotlin Multiplatform egy évvel korábbi megjelenése tette lehetővé. Az Android fejlesztők körében kifejezetten népszerű lett, annak ellenére is, hogy még mindig van rengeteg funkció, amit nem támogat, de jelenleg is aktívan fejlesztik, és válik hónapról-hónapra egyre jobbá. A fejlesztések havonta / néhány havonta érkeznek mind a Kotlin nyelvhez és a Kotlin Multiplatformhoz[32], mind a Compose Multiplatformhoz[24]. Jelenleg vannak sokkal jobban elterjedt, használtabb cross-platform keretrendszerek, mint például a Flutter (1.1. ábra) vagy a React Native, ami iOS és Android fejlesztést tesz lehetővé JavaScript/TypeScript nyelven, így a webes fejlesztők gyakran használják natív mobilos appokhoz.

Szerencsére a fejlesztés folyamatos és gyors, amit segít az is, hogy sok nyílt forráskódú könyvtár is készül a felhasználók által.[28] A Compose Multiplatform ennek az ellenpontja lesz[12]; sajnos a webes támogatás még csak alpha verzióban van, így eléggé instabil, és sok olyan eszköz nem használható még, ami a többi területen már stabilan működik, így ezzel egyelőre a szakdolgozat keretein belül nem foglalkoztam részletesebben a webes megoldás megvalósításával.



1.1. ábra. A Flutter és a Compose keresési arányai [4]

A visszajelzések és a mostani szoftverfejlesztési irányokból arra lehet következtetni, hogy még hosszú jövő áll a technológiák előtt. Könnyen, gyorsan és hatékonyan lehet akár egyszerre az összes platformra alkalmazást fejleszteni, nagy méretű közös és kis méretű natív kódbázis írásával és karbantartásával, összesen két nyelv ismeretével. Webre, Androidra és asztali alkalmazáshoz elegendő lehet a Kotlin nyelv, esetleg egy kis HTML és JavaScript ismeret; iOS esetén minimális Swift és SwiftUI ismeret jól jöhet, de ez hasonlít a Kotlinra és a Compose-ra. Az a tény, hogy a Compose Multiplatform csupán Kotlin nyelven írt kódbázissal képes natívan megjeleníteni az elkészült alkalmazást minden platformon, nagyon erős eszközzé teszi. Például az Androidon futó alkalmazás az Androidos



1.2. ábra. A könyvtárak növekedésének gyors üteme [28]

gombokat, görgetést stb. használja, míg iOS-en az ott megszokott stílust és irányítást kapja a felhasználó.

1.3. Dolgozat felépítése

Az alábbiakban bemutatom a dolgozat felépítését.

1. *Feladatspecifikáció: (2. fejezet)*
   (a) *Feladat részletes leírása: (2.1. szakasz)*
   (b) *Diagramok: (2.2. szakasz)* Bemutatok néhány diagramot és ábrát, amelyek segítik az alkalmazás felépítésének megértését.
2. *Irodalomkutatás: (3. fejezet)*
   (a) *Felhasznált technológiák: (3.1. szakasz)* az 1.2. szakaszban leírtak és a használt könyvtárak részletes bemutatása
   (b) *Hasonló megoldások: (3.2. szakasz)* Rövid összevetés a többi hasonló cross-platform megoldással.
3. *Felsőszintű architektúra: (4. fejezet)*
   (a) *High level architektúra: (4.1. szakasz)*
   (b) *Rendszer felépítései, komponensei: (4.2. szakasz)*
4. *Részletes megvalósítás: (5. fejezet)*
   (a) *UML class diagramok*
   (b) *Enity-relation diagram*
   (c) *Szekvencia diagram*
   (d) *Kódrészek*
5. *Tesztelés:*
   (a) *Felhasználói leírás*
   (b) *A program bemutatása képekkel*
   (c) *Lehetséges tesztelési megoldások ismertetése*
6. *Összefoglalás, továbbfejlesztési lehetőségek*

2. fejezet

Feladatspecifikáció

Ebben a fejezetben bemutatom az alkalamzás céljait. Az első részben általános összefoglalom a felhasználói elvárásokat és legfőbb építőköveket a programzó szemszögéből. Ezt követően megmutatom a használati Usecaseeket.

Kitérek a főbb funkciókra, azok felépítésére és elvárt használati módjaira. Az utolsó fejezetben mindezt egy Usecase UML diagramban összefoglalom. Megtalálható benne az összes fent leírt használati eset, jól látható lesz, hogy a funkciók hasonlítanak egymásra így a felhasználó könnyen meg tudja tanulni az alkalmazás használatát.

2.1. Feladat részletes leírása

Ebben az alszakaszban részletesen kitérek az alkalamazás komponenseinek megtervezésére, és azok használatára. Bemutatom, hogy milyen egy feladatsor felépítése, és annak a részei milyen mezőket vagy értékeket tartalmaznak. Ebben a részben szövegesen végig éehet kísérni a 2.1. ábrán látható felépítést.

2.1.1. Általános célok bemutatása

A program célja az, hogy egy vagy több felhasználó létre tudjon hozni igaz-hamis illetve feleletválasztós (tetszőleges számú válaszopcióval) kérdésekből álló feladatsort. Az alaklamazás csak papír alapú kitöltést támogat, ezt olyan formában teszi, hogy az összeállító a kész kérdéssort ki tudja exportálni PDF formátumba.

A szoftver a fent leírtakon kívül rendelkezik egy automtizált javítási rendszerrel, okostelefont használva a Google ML Kit[20] segítségével be lehet scannelni egy megfelelően formázott választ, és az belekerül egy form-ba amit a szervernek elküldve visszaküldi az eredményt. A kézírásos szövegfelismerés sok esetben nem szolgáltatott megfelelően pontos eredményt, így ez a funkció csak alpha verzióban támogatott. Hibásan felismert, vagy felismerhetetlen váaszok esetén kézzel is szerkeszthető a válasz a kiértékelés előtt.

Az alkalmazás a főmenüből indul, innen lehet tovább navigálni az összes opcióhoz. A főmenü máshogy néz ki a használt eszköztől függően. A választásunk után egy listázó nézet tárul elénk, ahol látjuk az adott opcióhoz tartozó összes eddig felvett elemet. Itt tudunk új elemet is hozzáadni az adott kategórihoz, illetve a listán történő kattintással a részletes nézet tárul elénk. A részletes nézeten minden adatot egyszerre látunk, itt tudjuk törölni és módosítani is. Mind a létrehozás és a szerkesztés során *-gal vannak jelölve a kötelező értékek. Törlés során van egy figyelemfelhívó ablak is, mivel a törlés az végleges és

nem vonható vissza. Vannak egyediséget megkívánó mezők, így amennyiben már létezik a megadott értékkel egy felvett elem, jelzi a za alkalmazás, hogy ezt módosítani kell mentés előtt.

A szoftver müködési elve és kommunikációja röviden összefoglalva az alábbi. A felhasználó megnyitja az alkalamzást, majd interaktál akezelő felülettel. A kattintások során amik igénylik az adatbázis elérést (listázás, részletes nézet megjelenítése, új elem létrehozása, szerkesztés, törlés) az alaklamazás szabványos HTTP kéréseket intéz a REST API-hoz. Az adatok forgalma szabványos JSON formátummal zajlik mind az adatküldés, mind az adatok fogadása során mindkét irányban. A REST API és az adatbázis is egy virtuális gépen fut egy-egy Docker konténerben.

2.1.2. Egy feladatsor felépítése

Egy feladatsor a következő elemekből épül fel:

- *Témakör*
- *Kérdések:*
  - *Igaz-hamis:* A szokásos egyszerű igaz-hamis típusfeladat.
    - *Kérdés:* Meg kell adni magát az eldöntendő kérdést.
    - *Pontozási módszer:* Egyedi pontozási módszert lehet létrehozni. Beállítható az összpontszám és a helyes és helytelen válaszokra adott pontérték, amely lehet negatív is.
    - *Témakör:* Segít a kérdések kategorizálásában és szűrésében az összeállítás során.
    - *Helyes válasz:* Szükséges a javítás elvégzéséhez.
  - *Feleletválasztós:* A szokásos egyszerű feleletválasztós típusfeladat, testreszabható mennyiségű válaszopcióval.
    - *Kérdés:* Meg kell adni magát a kérdést, több helyes válasz is lehetséges.
    - *Pontozási módszer:* Egyedi pontozási módszert lehet létrehozni. Beállítható az összpontszám és a helyes és helytelen válaszokra adott pontérték, amely lehet negatív is.
    - *Témakör:* Segít a kérdések kategorizálásában és szűrésében az összeállítás során.
    - *Válaszok:* Meg kell adni a válaszopciókat.
    - *Helyes válasz(ok listája):* Szükséges a javítás elvégzéséhez.

2.1.3. Komponensek létrehozásának lépései

Az alábbiakban bemutatom az egyes alkotóelemek életútját az alkalmazáson belül.

2.1.3.1. Pont

A pontok listáját az ennek megfelelő menüpont kiválasztása után érjük el. Itt egy floating button segítségével adhatunk új pontot az alkalmazáshoz. A létrehozáshoz, amelyet szintén egy floating button segítségével érhetünk el a részletes nézetről, minden mező kitöltése kötelező, csak így biztosítható megfelelően az elvárt működése a javítási funkció miatt.

- *Típus/Név:* A pont típusa vagy neve. Ez egyedi mező is egyben, így lehet rá hivatkozni és megtalálni az alkalmazásban.
- *Pont:* A feladatra adható maximális pontszám.
- *Helyes válasz:* A helyes válaszokra adható pont, ajánlott úgy megvalósítani a pontozást, hogy ezeknek az összege a teljes pontszám legyen.
- *Helytelen válasz:* A helytelen válaszok során levont pontmennyiség. Amennyiben nincs levonás, az értéke 0; egyébként egy negatív szám.

A pont létrehozása után látható lesz a listás nézetben, ahol kiválasztva ellenőrizhetjük a megadott értékeket, szükség esetén módosíthatjuk is. A pontokat törölni is lehet, de csak akkor, ha egyetlen kérdéshez sem használjuk, különben inkonzisztens állapot alakulna ki és pont nélküli kérdések keletkeznének. Mindkét műveletet a részletes oldalon tehetjük meg.

2.1.3.2. Témakör

A témák listáját az ennek megfelelő menüpont kiválasztása után érjük el. Itt egy floating button segítségével adhatunk új témakört az alkalmazáshoz. A létrehozáshoz, amelyet szintén egy floating button segítségével érhetünk el a részletes nézetről, minden mező kitöltése kötelező, csak így biztosítható megfelelően az elvárt működés.

- *Témakör neve:* A témakör megnevezése, egyedi mező.
- *Témakör leírása:* Kötelező egy rövid leírást adni az egyértelműség érdekében.
- *Szülő témakör:* Megadható egy fölérendelt témakör is.

A témakör létrehozása után látható lesz a listás nézetben, ahol kiválasztva ellenőrizhetjük a megadott értékeket, szükség esetén módosíthatjuk is. A témaköröket törölni is lehet, de csak akkor, ha egyetlen kérdéshez és feladatsorhoz sem használjuk, különben inkonzisztens állapot alakulna ki, és témakör nélküli kérdések és feladatsorok keletkeznének. Mindkét műveletet a részletes oldalon tehetjük meg.

2.1.3.3. Igaz-hamis kérdés

Az igaz-hamis kérdések listáját az ennek megfelelő menüpont kiválasztása után érjük el. Itt egy floating button segítségével adhatunk új kérdést az alkalmazáshoz. A létrehozáshoz, amelyet szintén egy floating button segítségével érhetünk el a részletes nézetről, minden mező kitöltése kötelező, csak így biztosítható megfelelően az elvárt működés.

- *Témakör neve:* A kérdéshez tartozó témakör megnevezése, a meglévő elemek közül választható.
- *Pont típusa:* A kérdéshez tartozó pont megnevezése, a meglévő elemek közül választható.
- *Kérdés:* A kérdés szövege, egyedinek kell lennie.
- *Helyes válasz:* Meg kell adni a helyes válaszopciót, amely lehet igaz vagy hamis.

A kérdés létrehozása után látható lesz a listás nézetben, ahol kiválasztva ellenőrizhetjük a megadott értékeket, szükség esetén módosíthatjuk is. A kérdéseket törölni is lehet, de csak akkor, ha egyetlen feladatsorhoz sem használjuk, különben inkonzisztens állapot alakulna ki, és hiányoznának kérdések az összeállított feladatsorokból. Mindkét műveletet a részletes oldalon tehetjük meg.

2.1.3.4. Feleletválasztós kérdés

A feleletválasztós kérdések listáját az ennek megfelelő menüpont kiválasztása után érjük el. Itt egy floating button segítségével adhatunk új kérdést az alkalmazáshoz. A létrehozáshoz, amelyet szintén egy floating button segítségével érhetünk el a részletes nézetről, minden mező kitöltése kötelező, csak így biztosítható megfelelően az elvárt működés.

- *Témakör neve:* A kérdéshez tartozó témakör megnevezése, a meglévő elemek közül választható.
- *Pont típusa:* A kérdéshez tartozó pont megnevezése, a meglévő elemek közül választható.
- *Kérdés:* A kérdés szövege, egyedinek kell lennie.
- *Válaszok megadása:* Meg kell adni a válaszokat, és jelölni kell a helyes válaszokat.

A kérdés létrehozása után látható lesz a listás nézetben, ahol kiválasztva ellenőrizhetjük a megadott értékeket, szükség esetén módosíthatjuk is. A kérdéseket törölni is lehet, de csak akkor, ha egyetlen feladatsorhoz sem használjuk, különben inkonzisztens állapot alakulna ki, és hiányoznának kérdések az összeállított feladatsorokból. Mindkét műveletet a részletes oldalon tehetjük meg.

2.1.3.5. Feladatsor

A feladatsorok listáját az ennek megfelelő menüpont kiválasztása után érjük el. Itt egy floating button segítségével adhatunk új feladatsort az alkalmazáshoz. A létrehozáshoz, amelyet szintén egy floating button segítségével érhetünk el a részletes nézetről, minden mező kitöltése kötelező, csak így biztosítható megfelelően az elvárt működés.

- *Feladatsor neve:* Egyedi mező.
- *Témakör kiválasztása:* A kérdéshez tartozó témakör megnevezése, a meglévő elemek közül választható.

A feladatsor létrehozása után látható lesz a listás nézetben, ahol kiválasztva a részletes oldalt látjuk. Itt van lehetőségünk a kérdések hozzáadására és eltávolítására a feladatsorból. A témákra és kérdéstípusokra szűrve válogathatunk a kérdéseink között. Hozzáadás után a kérdések mozgathatóak a listában. A más típusuba tartozó kérdések más színnel vannak jelölve ezzel is segítve a felhasználót a kérdés típusának gyors felmérésében. A feladatsorok szabadon törölhetők. Innen léphetünk át a szerkesztés oldalra, ahol a nevet és a témakört tudjuk módosítani.

2.1.3.6. Válaszok ellenőrzése

Amennyiben ezt a menüpontot választjuk, akkor a 2.1.3.5. alalszakasz-ban is leírt feladatsor listát látjuk, innen is létre lehet hozni új feladatsort. Az itt kiválasztott elem viszont egy form beküldő oldalra navigál, ahol egymás alatt látszanak a kérdések. A más típusuba tartozó kérdések más színnel vannak jelölve ezzel is segítve a felhasználót a kérdés típusának gyors felmérésében.

Itt lehet megadni a válaszainkat, illetve mobil eszközön a szövegfelismerés funkciója gyorsíthatja meg a kitöltést. A lehetséges formátumra, amit a szoftver elvár található segítség, így nehezebb ezért rossz megoldást megadni. Ha mégis hibás formátumban küldi be a felhasználó a válaszokat kap róla figyelmeztetést. A beküldést követően hamarosan megjelenik az eredmény a képernyőn.

2.1.3.7. Feladatsorok exportálása

Amennyiben ezt a menüpontot választjuk, akkor a 2.1.3.5. alalszakasz-ban is leírt feladatsor listát látjuk, innen is létre lehet hozni új feladatsort. Az itt kiválasztott elem viszont egy előzetes feladatsor megjelenítő oldalra navigál.

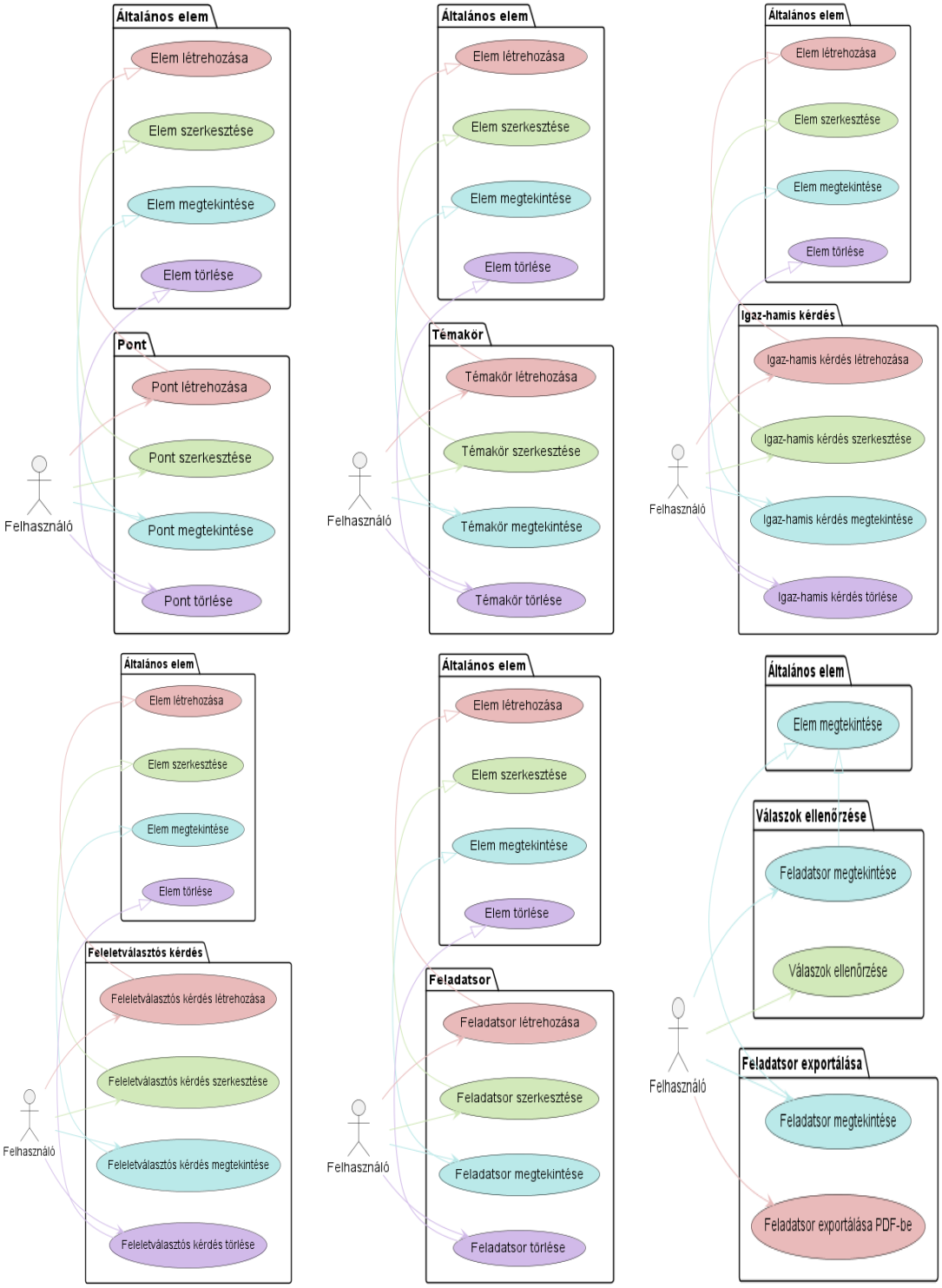
Amennyiben az itt látottakkal elégedettek vagyunk, rendben vannak a pontok és a kérdések, akkor exportálhatjuk is a munkát PDF formátumban. Az itt látottak csak egy vázlatos elrendezést adnak; az adatok ellenőrzésére szolgálnak. Előfordulhat, hogy a mobilos és az asztali verzió kis mértékben eltér egymástól a megjelenésben.

2.2. Diagramok

Ebben az alfejezetben az elkészült alkalamzás használati eseteit bemutató diagram található. A Usecase diagram segítségével egy tapasztalt fejlesztő gyorsan fel tudja mérni az alkalamzás fő felépítésétm funkcióit és megközelítő struktúráját.

A csak a magas szintű használati esetek jelennek meg a diagramon az átláthatóság miatt. A képernyők közötti navigáció az ábra átláthatóságán jelentősen rontana, így arról csak a szöveges leírásban esett szó.

A PlantUML segítségével elkészített diagramokon látható, hogy létezik öt egymáshoz hasonló működésű használati eset, így ezek származhatnak egy általános elemből vagy usecaseből. Az azonos színekkel jelölt esetek hasonló funkcionalitást jeleznek ezzel is segítve a gyors megértését a szoftvernek. A Felhasználóból induló nyilak a lehetséges cselekedeteket jelölik, míg az ezekből a usecasekből induló nyilak az általános cselekedetekből való leszármazást.



2.1. ábra. Usecase diagramja az alkalmazásnak.

3. fejezet

Irodalomkutatás

Ebben a fejezetben bemutatom az általam használt forrásokat és technológiákat. Esetenként ábrákkal és kódrészletekkel ilusztrálom az adott technológiát a könnyebb érthetőség kedvéért. Az alfejezet végén bemutatok néhány alternatív megoldást a multiplatform fejlesztésre.

3.1. Felhasznált technológiák

Ez az elfejezet a használt technológiák bemutatására fókuszál, a könnyebb érhetőség kedvéért ábrák, képek és forráshivatkozásokkal kiegészítve.

3.1.1. Jetpack Compose

A Jetpack Compose a korábbi Android fejlesztési módszer mellett hozott létre egy alternatív megoldást. Kezedtebn nem lehetett tudni, hogy a fejlesztők hogyan fognak reagálni az új irányra. Korábban a Java nyelv mellett megjelent a Kotlin nyelv is, ami később szinte teljesen leváltotta az elődjét. Ebből arra lehetett következtetni, hogy egy új és modernebb megoldás meg tudja állni a helyét az XML View megoldást ellenében. Jelenleg mind a két megoldás támogatott, de a fejlsztések iránya egyértelműen a Compose felé húz.

"A Jetpack Compose egy új, deklaratív UI toolkit, amit a Google hozott létre kifejezetten natív Android alkalmazások fejlesztéséhez."[26] A deklaratív nyelvekhez hasonlóan, azt kellmegadnunk, hogy mit szeretnénk látni és nem azt leírni, hogy ez hogyan történjen meg. Nekünk elegendő azt leírni, hogy a gomb hogyan nézzen ki és hol legyen és megadni egy lambda paraméternek, hogy a megnyomása soránmi történjen. Mivel ez egy UI toolkit, ezért az összes vezerlő és szerkezeti elem hasonlóan néz ki, hasonlóan lehet használni így a fejlesztői és a felhasználói élmény is egységes és megszokott minőségű lesz minden alkalommal. A Google ezt a Material design segítségével hozta létre, illetve annak újabb változataival. Erről részletesebben itt találhatók információk: `https://m3.material.io/`.

Az alábbiakban egy a Google által írt rövid kódrészleten (3.1. kódrészlet) bemutatom a legfontosabb részeit a Compose alapjainak.[19] Az első lépése az alaklamzás elkészítésének az a Composable függvény megírása. Minden a UI-t megjelenítő függvény a @Composable annotációt viseli. Innentől kezdve hagyományos Kotlin függvényként viselkedik, megadahatunk tetszőleges paramétereket (name, modifier) és alapértelmezett értékeket is. Egy Composable függvényből tetszőleges másik Composable függvény meghívható megfe-

lelőláthatóság esetén. Ilyen például a Text() is ami a Material könyvtárnak egyik tagja és egyszerű szöveget jelenít meg.

A UI megírása után ezt a megfelelő helyen meg is kell jelenítenünk, erre az alkalmazás belépési pontja után van lehetőségünk. Android esetén ez az activity onCreate függvénye. A setContent vár egy lambda függvényt, aminek viselnie kell a @Composable annotációt, használhatjuk hozzá a Kotlin trailing lambda megoldását, aholis, ha az utolsó paramétere a függvénynek egy lambda, akkor a többi paraméter megadása után között megadhatjuk afüggvény törzsét. A BasicsCodelabTheme is egy Composable függvény amiben az alap beállítások után meghívhatjuk a saját Greeting függvényünket. A UI felépítse innentől kezdve már egyszerű. A Composable függvények megírása után egymásból meghívva azokat előáll az alkalmazás.

```
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Text(
        text = "Hello $name!",
        modifier = modifier
    )
}

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            BasicsCodelabTheme {
                // A surface container using the 'background' color from the theme
                Surface(
                modifier = Modifier.fillMaxSize(),
                color = MaterialTheme.colorScheme.background
                ) {
                    Greeting("Android")
                }
            }
        }
    }
}

// Jetpack Compose forráskód
public fun ComponentActivity.setContent(
    parent: CompositionContext? = null,
    content: @Composable () -> Unit
)
```

3.1. kódrészlet. Példa a Compose használatára.

A következőkben bemutatom a UI toolkit fontosabb általam használt részeit. Kitérek arra, hogy mire jó, miért ezt használtam és hogyan lehet őket hatékonyan felhasználni a legújabb Compose Multiplatform verziókban.

3.1.1.1. State és StateFlow

A State és a StateFlow hasnló problémára ad megoldást. A State alapvetően Compose sepcifikus megoldás, ha változik az értéke akkor a lefut a recomposoition. Ezzel szemben a StateFlow a Kotlin nyelvben általánosan használt eszköz és sokkal bővebb felhasználással rendelkezik, mint az egyszerű State. Az egyszerű State-et általában egy Composable függvényen belül használják, míg a a StateFlowt ViewModelekben.

Ennek ellénére mind a két megoldás tökéletesen alkalmazható, jelenleg már Compose Multiplatform alkalmazásokban is. Mivel ViewModelben használva az adatok nem besznek el a képernyő elforgatása során ezért egyszerűbb műveletekre és adaokra nincs lényegi különbség a működésben.

"A StateFlow előnyei:"[5]

- *"Flow operátorok:* A StateFlow támogat olyan operátorokat, mint a map, filter, és combine, lehetővé téve az adatok rugalmas feldolgozását és összetett adatfolyamok létrehozását."
- *"Folyamat-megszakadás kezelése:* A SavedStateHandle-lel kombinálva biztosítja az UI állapot megőrzését, még a képernyő elforgatása vagy újraindítás esetén is."
- *"ViewModel újrafelhasználhatóság:* A StateFlow lehetővé teszi a ViewModel függetlenítését a UI-rétegtől, ami elősegíti a moduláris, tesztelhető és újrafelhasználható architektúrát."

A lenti kódban (3.2. kódrészlet) láthatunk példát mind az egyszerű State használatára, ebben az esetben a Screen állapotát tárolom el Statekben. A megjelnített adatok itt a StateFlow logikáját követik. Létezik egy privát MutableStateFlow amin történnek a változások például, ha adat érkezik. Van egy másik azonos nevű érték is ami ugyan annak a StateFlownak egy immutábilis változata, ehhez fér hozzá a UI, így onnan nem érkezhet változás közvetlenül a StateFlowba. Amennyiben erre szükség van, a viewmodel biztosíthat erre vonatkozóan függvényeket és beállíthatja a privát MutableStateFlow értékét.

```
class TopicListViewModel: ViewModel() {
    var topicListScreenUiState: TopicListScreenUiState by mutableStateOf(TopicListScreenUiState.
     Loading)
    private val _topicListUiState = MutableStateFlow(TopicListUiState())
    val topicListUiState: StateFlow<TopicListUiState> = _topicListUiState
...
    fun getAllTopicList(){
        topicListScreenUiState = TopicListScreenUiState.Loading // State Változás
        viewModelScope.launch {
            topicListScreenUiState = try{    // State Változás
                val result = ApiService.getAllTopicNames()
                _topicListUiState.value = TopicListUiState(     //StateFlow Változás
                    topicList = result.map { nameDto ->
                        TopicRowUiState(
                            topic = nameDto.name,
                            id = nameDto.uuid
                        )
                    }
                )
                TopicListScreenUiState.Success(result) // State Változás
            } catch (e: IOException) {
                TopicListScreenUiState.Error.errorMessage = e.toString()// "Network error"
                TopicListScreenUiState.Error // State Változás
            }
        }
    }
}
```

3.2. kódrészlet. Példa a State és StateFlow használatára.

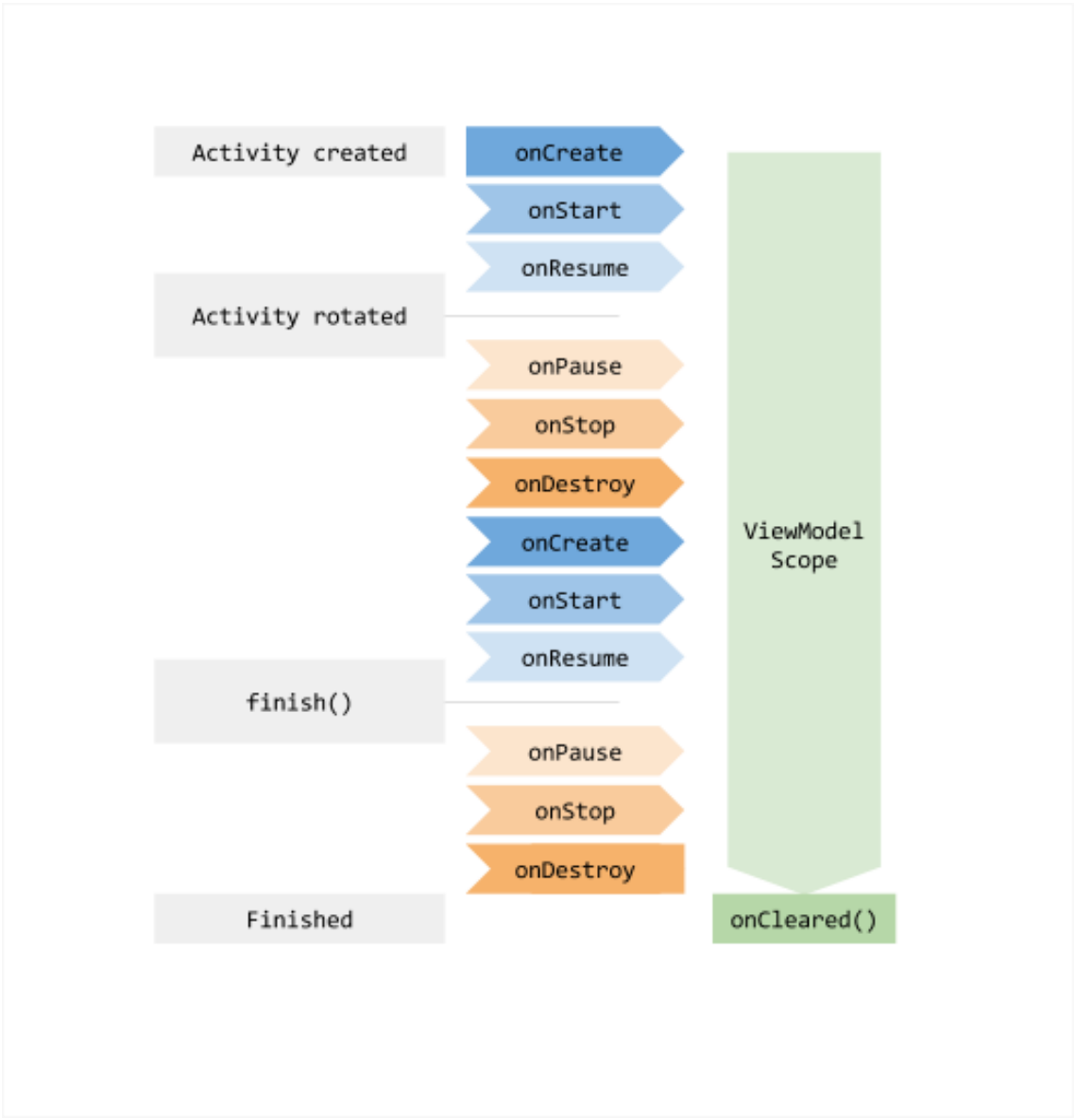
3.1.1.2. ViewModel

Az Android ViewModel már használható Kotlin- és Compose Multiplatform környezetekben is[8], így természetesen ezt a jól bevált megoldást választottam. Egy rövid összefoglalót szeretnék adni a hivatalos dokumentáció alapján a képességeiről:[21]

A ViewModel az Android Jetpack része, és az UI állapotának megőrzésére szolgál konfigurációs változások során, például képernyőforgatáskor. 3.1. ábra Fő előnyei közé tartozik az állapot tartósítása és az üzleti logika elérése a UI-rétegben. A ViewModel segít

elválasztani az adatkezelést a UI-rétegtől, ehhez a ViewModelben felhasználható a Kotlin coroutine ami egyfajta aszinkron működést tesz lehetővé, és kompatibilis olyan Jetpack könyvtárakkal, mint a Hilt, Compose, és a Navigation. Best practice szerint kerülni kell az életciklushoz kötött objektumok tárolását benne, hogy elkerüljük a memóriaszivárgást.

Mint minden MVM és MVVM architektúrában az adatok elérése, átalakítása a UI számára, tartós tárolása a ViewModel feladata. Több megközelítés is lehetséges: minden képernyő kapjon egy saját ViewModelt; egy ViewModel tudjon többet és legyen újra felhasználva több képernyőn. Én az első megoldást választottam, így könnyebb kezeleni a különböző képrenyők állaptait és egyszerűbb átlátni egy-egy egységet.



3.1. ábra. A ViewModel álltalánosságban véve hosszabb életű, mint egy View. Ez különsöen igaz Android környezetben, ahol számolni kell a képernyő elforgatással és az alaklmazás háttérbe kerülésével. A tartósan tárolnikívánt adatokat ezért mindig ViewModel kell tárolni. [21]

3.1.1.3. Navigation and routing

Az Androidban már jól működő navigációért felelős könyvtárak már használhatóak a Kotlin Multiplatform fejlesztéséhez [11]. Ennk mindösszesen pár egyszerű lépése van, így könnyű használni és rugalamsan működik minden környezettel. Mindezt kódvab is bemutatom. (3.3. kódrészlet) A következők a szükséges lépések: [11].

1. "Sorold fel a navigációs gráfban szereplő útvonalakat, mindegyik egyedi string határozza meg az útvonalat."
2. "Hozz létre egy NavHostController példányt a navigáció kezeléséhez"
3. "Adj hozzá egy NavHost komponenst az alkalmazásodhoz:"
    - "Válaszd ki a kezdő útvonalat a korábban definiált útvonalak közül."
    - "Hozd létre a navigációs gráfot közvetlenül a NavHost részeként vagy programozottan a NavController.createGraph() függvénnyel."

```
//Első lépés: útvonalak létrehozása
//Érdemes sealed classt használni és data objectként létrehozni a routeokat
sealed class ExamDestination(val route: String) {
    //Lehet egyszerű
    data object LoginScreenDestination : ExamDestination("LoginScreen")

    //Vagy paraméterekkel és adatokkal ellátott
    data object TopicDetailsDestination : ExamDestination("TopicDetails") {
        const val topicIdArg = "0"
        val routeWithArgs = "$route/{$topicIdArg}"
    }
}

//Második lépés: NavController létrehozása
fun NavigationComponent() {
    MaterialTheme {
        val navController = rememberNavController() //Itt történik
        Scaffold() { innerPadding ->
            ExamNavHost(    //Paraméternek egy Composable függvényt vár, ezen belül is egy NavHost fü
    ggvényt
                navController = navController,
                modifier = Modifier.padding(innerPadding)
            )
        }
    }
}

//Harmadik lépés: NavHost komponens hozzáadása
actual fun ExamNavHost(
    navController: NavHostController,
    modifier: Modifier
) {
    NavHost(
        navController = navController,  //NavController hozzárendelése
        startDestination = ExamDestination.MainScreenDestination.route, //Kezdő útvonal beállítása
        modifier = modifier
    ) {
        //Navigácós gráf egy elemének létrehozása
        composable(
            route = ExamDestination.TopicListDestination.route,
        ) {
            TopicListScreen(
                addNewTopic = { navController.navigate(ExamDestination.NewTopicDestination.route) },
                navigateToTopicDetails = { topicId ->
                    navController.navigate("${ExamDestination.TopicDetailsDestination.route}/${
    topicId}")
                },
```

```
                navigateBack = { navController.popBackStack() }
            )
        }
    }
}
```

3.3. kódrészlet. Példa a Navigation használatára.

3.1.2. Ktor

A Ktor egy Kotlin HTTP kommunikációt megvalósító könyvtár[9]. Alkalmas mind szerver oladli kód írására, a REST API-om is ezt használja és kliens oladli kód megvalósítsára is. A használata nagyon egyszerű, és testreszabható. Én egy Kotlin objectet használtam, ami magában foglalja az ApiServicet. Létre kellett hozni egy http klienst és beállítani a base url-t, illetve a content typenak a JSON üzenet formátumot. Ezt követően a végpont hívásokat kellett már csak létrehozni.

```
object ApiService {
    private var authToken: String? = null  // Mutable token that can be updated at runtime

    private val httpClient = HttpClient() {
        install(ContentNegotiation) {   // content type beállítása
            json(Json {
                ignoreUnknownKeys = true
                prettyPrint = true
            })
        }

        defaultRequest {    // Base url beálítása
            url("http://mlaci.sch.bme.hu:46258")  // Set the base URL   152.66.182.70:46258
     192.168.1.17:46258
            authToken?.let { token ->
                header(HttpHeaders.Authorization, "Bearer $token")  // Add the Bearer token if it's
     not null
            }
        }
    }
    suspend fun getAllPoints(): List<PointDto> = httpClient.get("/point").body()    //végpontok
}
```

3.4. kódrészlet. Példa a Ktor használatára.

3.1.3. KotlinX Serilizáció

A serilizáció a JSON formátum konvertálása miatt van szükség. Az alábbiakban a hivatalos dokumentációból olvasható egy részlet ami jól összefoglalja a használatát. Ez a technológia is használható Kotlin Multiplatform területen.

"A szerializáció során az alkalmazások adatait egy olyan formátumba alakítjuk, amely hálózaton átvihető vagy tárolható adatbázisban vagy fájlban. Az ellenkező folyamat, a deszerializáció, az adatokat külső forrásból olvassa be és konvertálja futásidejű objektummá. A Kotlinban a szerializációhoz elérhető a kotlinx.serialization eszköz, amely Gradle bővítményt, futásidejű könyvtárakat és fordítói bővítményeket tartalmaz, így segítve a különböző nyelvű rendszerek közötti adatcserét, mint a JSON és a protocol buffers formátumokkal."[13]

3.1.4. CameraX

A CameraX technológia kizárólag Android platformon használható. Itt azonban egy nagyon széles és gazdag APIt biztosít a fejlesztéshez. A legfontosabb felhasználható funkciói az Preview azaz előnézet, amikor kép készítése nélkül megkelenik a képernyőn a kamera képe. Az Image analysis, azaz képfeldolgozó funkcionalitás. Hozzáférhetünk a buffer tartalmához így fel tudjuk azt használni egyéb célokra különböző algoritmusok futtatásához és összekapcsolható a Google ML-Kit technológiákkal. 3.1.5. alszakasz A képeket menetni is tudjuk, hasonlóan a beépített kamera alkalamzáshoz és ugyan úgy videót is tudunk vele rögzíteni. Ezek a leírás a Google Android Developers dokumentációja alapján készült. Részletesebb információk itt találhatóak: [18]

3.1.5. ML-Kit

Az ML-Kit a Google által fejleszett mesterséges intelligiencia alapú API. Számtalan felhasználási területtel rendelkezik, ezekeből néhányat felsorolok: szöveg- és arcfelismerés, dokumentom szkennelés, kép feliraotozás, fordítás, nyelv detekció és még számos más lehetőség. Én ezek közül az Androidos alaklamazásban a képen történő szövegfelismerést próbáltam ki.[20] Sajnos ez a funkció is Android specifikus így egy iOS alaklamazáson ez a probléma más megközelítést igényelne. Messze nem tökéletes még ez a technológia, de kipróbálásra mindenféleképpen érdekes és hasznos lehet.

3.1.6. Accompanist-engedélykezelés

Az engedélykezelés nem egyszerű feladat az Android rendszerekben így célszerű erre kifejlesztett könyvtárakat használni. Egy ilyen könyvtár az Accompanist, amlyet a Google fejleszt. A használata sokkal egyszerűbbé teszi ezt a bonyolut folyamotot. Néhány egyszerű lépéssel egy kész megoldás kapunk.

Elsőként az szükséges engedélyeket be kell jegyezni a manifest fájlba. Következő lépésként ellenőrizni kell, hogy az alkalmazásunk rendelkezik a szükséges engedélyekkel vagy sem. Amenyiben nem akkor a használat előtt ezt kérnünk kell, de ezt csakúgy tehetjük meh, hogy a öbbi funkció elérető legyen. Az én esetemben, csak a szövegfelismerő funkcióra kattintás után kérem el az engedélyt, de maga a válaszokat elküldő képernyő használható az engedélyek nélkül is. Az elkért engedélyeket egy permissionStateben tároljuk el, így innen lehet ellenőrizni, hogy korábban már megadta-e a felhasználó, így legközelebb nem kell elkérni tőle. [25]

Egyedül a veszélyes engedélyeket kell ilyen müdon elkérni, mint például a kamera használata, tehát amik a felhasználót veszélyeztetik. Vannak nem veszélyes engedélyek is, ilyen az internet használat, ezt csak a manifest fájlban kell rögzíteni.

3.1.7. PdfDocument és PDFBox

A PdfDocument a Google tulajdonában lévő PDF szerkesztő eszköz. Ennek a felhasználsval egy PDF fájlba írhatunk tetszőlegesen elhelyezett szöveget és képeket. Létrehozhatók különböző Paint objektumok amik segítségével egyszerűen rajzolható táblázat és formázható a szöveg. Ez a megoldás csak Android eszközökkel kompatibilis.

"Az Apache PDFBox® könyvtár egy nyílt forráskódú Java eszköz PDF dokumentumok kezelésére. Lehetővé teszi új PDF dokumentumok létrehozását, meglévő dokumen-

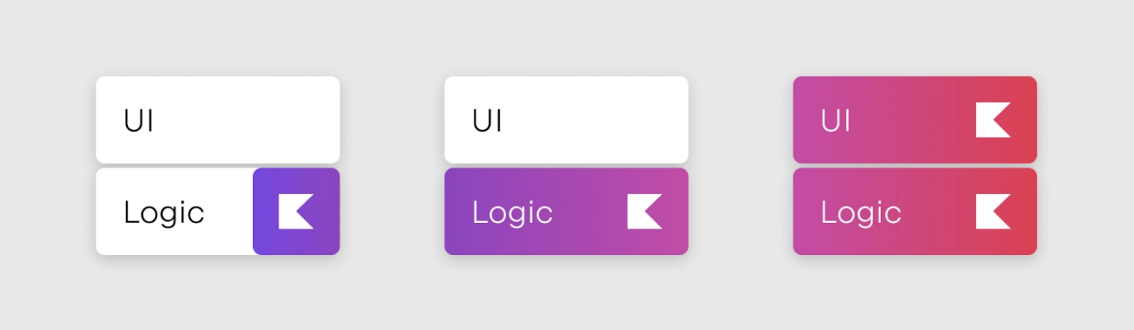
tumok módosítását és tartalom kinyerését a PDF fájlokból. Az Apache PDFBox több parancssori eszközt is tartalmaz, és az Apache License v2.0 alatt került kiadásra." [1] Hasonlóan használható, mint a PdfDocument, de valamelyest eltér a két könyvtát API készlete. Elsőként Androidra készült el az exportálás funkció, de mivel nem sikerül teljesen ugyan azt az eredményt létrehozni mind a két esetben, ezért egy valós projektben a multiplatform rendszerekben is használható PDFBox megoldást használnám minden platformon.

3.1.8. Kotlin- és Compose Multiplatform

Többször beszéltem már a Kotlin- és Compose Multiplatform fogalmakról. Legkönnyebben úgy lehet leírni a kapcsolatukat, mint a Compose Multiplatform részhalmaza a Kotlin Multiplatformnak. Rengeteg nagy cég használja a KMP technológiát köztük a Netflix, 9GAG, McDonalds' és Philips. [28] Az általam korábban felsorolt techonólógiák közül, ami leginkább ebbe a kategóriába esik az a Ktor (3.1.2. alszakasz) és a KotlinX szerilizáció (3.1.3. alszakasz). Mivel 2024 őszén elérhetővé vált az Android ViewModel (3.1.1.2. alalszakasz) és a navigáció (3.1.1.3. alalszakasz) is így ezeket is ide sorolhatjuk már a statekkel együtt (3.1.1.1. alalszakasz).

Az egyetlen fontosabb rész amit kihagytam, az maga a Compose deklaratív UI toolkit (3.1.1. alszakasz), ami a Compose Multiplatformot alkotja. 2021-ben vált lehetővé a Compose használata nem csak Android alapú rendszerekhez, a KMP viszont 2017-ben kezdte meg az úját. Nagy jelentősége van a CMP-nek mivel így kizárolóag Kotlin nyelven Android fejlesztők tudnak iOS és asztali alkalmazást fejleszteni minimális natív kóddal, de mégis natív élményt nyújtva. Jelenleg a webes irány még alpha verzióban van, de jelenleg is folyik a fejlesztés a Kotlin WASM-re (Web Assembly) valóhatékony fordításán. A Kotlint JavaScript kóddá is le lehet fordítani a Java mellett.

Három féle módon lehet Kotlin Multiplatform kódbázist fejleszteni (3.2. ábra). Az első, bal oldali ábra értelmezése, hogy a kódbázis egy kis része íródik KMP-ben, például csak az adatbázis vagy REST API elérés. A következő ábra azt mutatja, hogy a logika teljes egészben KMP-ben íródik, így például használjak a ViewModeleket (3.1.1.2. alalszakasz), de a UI natív módon készül, Androidra Composeban iOS-ben SwiftUIban. Az utolsó ábra már a Compose Multiplatform megjelnése, amikor minden platformra Composeban készül el a felhasználói felület. Balról jobbra haladva egyre nő a kód újrafelhasználhatósága, így kevesebb munka szükséges és könnyebb is a kód karbantartása, mivel előre láthatólag egyre kevesebb helyen kell megváltozatni azt.



3.2. ábra. A KMP fejlesztés változatai. [28]

Semmi sem teljesen tökéletes így előfordulaht, hogy az egyik platformna máshogy, nem lehet vagy nem érdemes valamit megvalósítani, mint például egy asztali alaklmazá-

son egy fotó elkészítését az én példámban. Ilyenkor jöhetnek szóba az expect és actual függvények. A közös kódan ilyenkor egy függvénytörzset definiálunk, csak itt lehet ebben az esetben alapértelmezett paramétereket beálítani, például egy Modifiert Composable függvény esetén. A megvalósítás ilyenkor az alaklamzás specifikus kódban történik (androidMain, desktopMain, iOSMain). (3.5. kódrészlet) Ehhez az actual functiont kell megvalósítani, és a platformtól függően ezek fognak meghívódni automatikusan, mivel a build során ezek lesznek behelyesítve az expect függvény helyére. Már létezik actual és expect osztály is, én alkalmaztam is, de ez még kisérleti verzióban van.

```
//commonMain-ben lévű expect függvénytörzs, alapértelmezett paraméterrel.
@Composable
expect fun MainCameraScreen(examId: String = "0", navigateBack: () -> Unit)

//androidMain-ben lévő valós megvalósítás
@Composable
actual fun MainCameraScreen(examId: String, navigateBack: () -> Unit) {

    val cameraPermissionState: PermissionState = rememberPermissionState(android.Manifest.permission.
     CAMERA)

    MainCameraContent(
        hasPermission = cameraPermissionState.status.isGranted,
        examId = examId,
        onRequestPermission = cameraPermissionState::launchPermissionRequest,
        navigateBack = navigateBack
    )
}

//desktopMain-ben lévő placeholder megvalósítás, értesíti a felhasználót, hogy ez a funkció az eszköz
     én nem támogatott
@Composable
actual fun MainCameraScreen(examId: String, navigateBack: () -> Unit) {
    Scaffold(
        topBar = {
            TopAppBarContent(stringResource(Res.string.camera), navigateBack)
                },
        ){innerPadding ->
            UnsupportedFeatureScreen(modifier = Modifier.padding(innerPadding))
    }
}
```

3.5. kódrészlet. Expect és actual használata

3.1.9. Gradle build rendszer

A Compose Multiplatform projektek is a Gradle build rendszert használják, első sorban a függőségek megszerzésere és az alkalmazás létrehozására. Egy átlagos fejlesztőnek mindösszesen annyi dolga van, hogy kigyűjti a használt függőségeket és a fejlesztőkörnyezet általában segít a megfelelő verziók megtalálásában. Újbban a Kotlin DSL használata terjedt el.

"A DSL (Domain-Specific Language) egy programozási nyelv, amely egy meghatározott problémakör megoldására összpontosít. Az általános célú nyelvektől eltérően a DSL-ek, például az SQL és a regexek, csak egy szűk területre fókuszálnak, ami lehetővé teszi a problémák deklaratív módon való megoldását."[27] Ennek a segítségével egyszerűbben tudjuk megadni a Gradle függőségeket.(3.6. kódrészlet)

```
kotlin {    //Részlet a build.gradle fájlból
    androidTarget {
        compilerOptions {
            jvmTarget.set(JvmTarget.JVM_11)
        }
    }
    sourceSets {
```

```
        androidMain.dependencies {
            implementation(libs.androidx.activity.compose)
        }
    }
}
```

3.6. kódrészlet. Kotlin DSL

Ezen kívül a verziók egyszerűbb karbantartására használhatunk version catalog fájlt, ez a libs.version.toml. A toml a "Tom's Obvious, Minimal Language" rövidítése, ezt első sorban egyszerűbb konfigurációs fájlok esetében használják, ha úgy tetszik egy "butább" yaml formátum. Az szükséges részei a [versions] és a [libraries], illeve szükség lehet [plugins]-re is. (3.7. kódrészlet) Az itt felvett értékekre lehet hivatkozni a build.gradle fájl(ok)ban.

```
#Részlet a libs.version.toml fájlból

[versions]
agp = "8.2.2"
android-compileSdk = "34"
android-minSdk = "24"
android-targetSdk = "34"
androidx-activityCompose = "1.9.2"
androidx-appcompat = "1.7.0"
androidx-constraintlayout = "2.1.4"
androidx-core-ktx = "1.13.1"

[libraries]
androidx-core = { module = "androidx.core:core", version.ref = "androidx-core-ktx" }
androidx-core-ktx-v1120 = { module = "androidx.core:core-ktx", version.ref = "coreKtx" }

[plugins]
androidApplication = { id = "com.android.application", version.ref = "agp" }
androidLibrary = { id = "com.android.library", version.ref = "agp" }
```

3.7. kódrészlet. Version catalog

3.1.10. Fejlesztőkörnyeztek

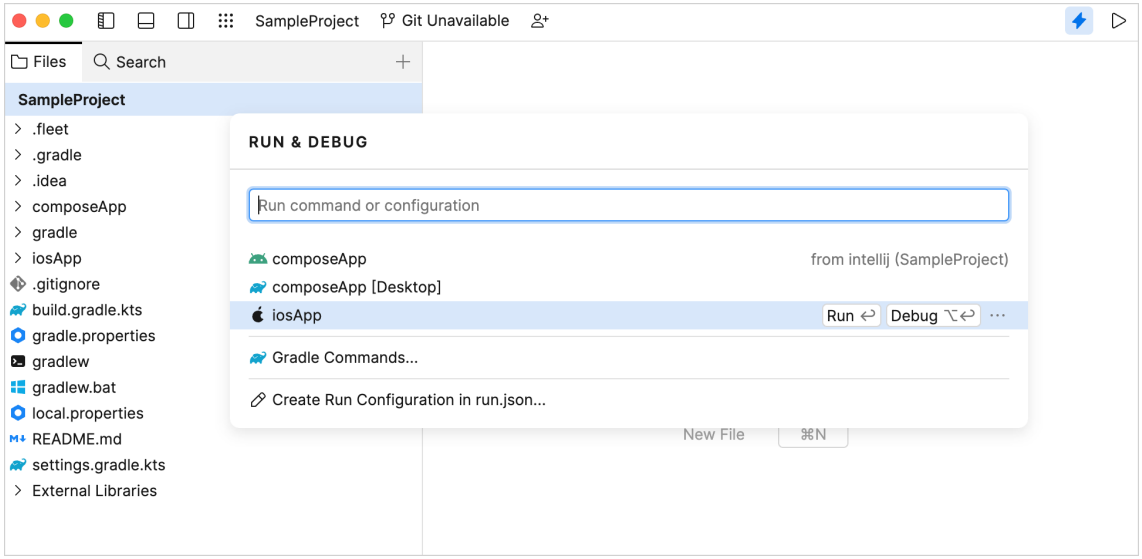
Fejlesztőkörnyezetnek a JetBrains egyik eszközét választottam, a Fleet-et. Kifejezetten Multiplatform fejlesztéshez készítettek, szinte az összes nyelvet támogata ami ebben a témában szóba jöhet, és minden funkcióval rendelkezik amivel a natív fejlesztésre készült IDE-ik is. Többek között kódkiegészítéssel és kód kiemeléssel is, így nem kell IDE-t váltani, ha más nyelven kell éppen dolgozni.

"Amikor az Smart Mode engedélyezve van, a Fleet nyelvspecifikus funkciókat kínál, mint például a kódkiegészítés, navigáció, hibakeresés és refaktorálás. Ha ez a mód le van tiltva, a Fleet egyszerű szövegszerkesztőként működik; gyorsan meg lehet nyitni fájlokat és módosításokat végezni, de a fejlettebb funkciók nélkül. A háttérben a Fleet kiválasztja a kód feldolgozásához szükséges háttérmotort. Intelligens módban például a Kotlin feldolgozási motorja az IntelliJ IDEA-hoz használt motor, így ismerős funkciók maradnak elérhetők." [14]

Egy másik hasznos funkció, hogy egy helyről lehet minden platformra buildelni az alkalamzást (3.3. ábra) így nem kell egy külön Android Studiot és egy Xcodeot is megynitni, ha szeretnéd tesztelni az alaklamzást az adott eszközön.

A fő fejlesztőkörnyezeten kívül amíg csak az Android applikációt fejleszettem az Android Studiot használtam. A REST API megírásához és karbantarásához szintén a JetBrains termékét az IntelliJ IDEA Ultimate-et használtam, ami kifejezetten Java és Kotlin projekezhez készült. Kisebb mértékben a fejlesztéshez és nagyobb mértékben a dokumen-

tációhoz és a szakdolgozat megírásához A Visual Studio Code-ot használtam, mivel sok hasznos bővítménnyel rendelkezik például LaTeXés PlantUML használatához.



3.3. ábra. A különböző eszközökre egy helyen lehet buildelni és futtani az alkalmazást. [14]

3.1.11. REST API, Postman és adatbázis

A kliens oladli alkalmazásokat egy saját REST API szolgálja ki. Ezt az előző félévben készítettem el. Teljesen egészében Kotlin nyelv felhasználásával. A HTTP kommunikáció megvalósításához a már korábban 3.1.2. alszakasz-ban bemutatott Ktor-t használtam. Itt a szerver oldali funkcionaliítása került előtérbe. A szerilizáció itt is a 3.1.3. alszakasz-ban leírtak szerint történt. Az adatbázis az úgynevezett code-first felfogás alapján készült. Ez azt jelenti, hogy kódban leírtam az adatázis felépítését és a kigenerálását rábíztam a Exposed-ra ami a JetBrains ltal fejlesztett adatbázis elérést lehetővé tevő könyvtár. A használt adatbázis javaslatok alapján a PostgreSQL lett. Ennek a folyamatnak egy részletesebb leírása a [10] forrásban olvasható, ez alapján készíttem el a saját szerveremet.

Mind az adatbázis, mind a REST API egy-egy Docker-konténerben fut egy viruális gépen, így biztosítva a folyamatos elérhetőséget. "Mi az a Docker? A Docker egy nyílt platform alkalmazások fejlesztésére, szállítására és futtatására. Lehetővé teszi, hogy az alkalmazásokat elválasszuk az infrastruktúrától, így gyorsabban tudunk szoftvereket szállítani. A Docker segítségével az infrastruktúrát ugyanúgy kezelhetjük, mint az alkalmazásokat. A Docker szállítási, tesztelési és kódtelepítési módszertanait kihasználva jelentősen csökkenthetjük az időt a kód megírása és a termelési környezetben való futtatása között." [30]

"A Docker platform: A Docker lehetőséget biztosít arra, hogy az alkalmazást egy lazán izolált környezetben, úgynevezett konténerben csomagoljuk és futtassuk. Az izoláció és a biztonság lehetővé teszi, hogy egy adott gépen egyszerre több konténert futtassunk. A konténerek könnyűsúlyúak, és mindent tartalmaznak, ami szükséges az alkalmazás futtatásához, így nem kell a gazdagépre telepített környezetre támaszkodni. A konténereket megoszthatjuk munka közben, biztosítva, hogy mindenki ugyanazt a konténert kapja, amely ugyanúgy működik." [30]

Rendelkezik saját DNS címmel is, így könnyen megjegyezhető és elérhető fejlesztés és tesztelés közben is. A teszteléshez a Postmant használtam. Ez egy olyan alaklamzás amiben HTTP kéréseket lehet létrehozni, és a mezőket tetszőlegesen testreszabni. Gyakran volt rá szükség ebben a félévben is, mert bizonyosadatformátumok/követelmények változtak és ez sokkal hatékonyabb hibakezelést tett lehetővé, mint egy böngészős HTTP kérés vagy az alaklamzásból történő debugolás.

3.1.12. Kipróbált, de végül nem használt egyéb érdekes megoldások

Az önálló laboratúrium során elkészített alkalmazás sok technológiát felhasznált, első sorban kisérlétezés miatt. Ezek egy részére találtam jól használahtó Multiplatform alternatívát, más részére nem, vagy csak nagyon sok munkával lehetet volna megvalósítani. Rendelkezett lokális Room adatbázissal is, erre egy alternatív megoldás KMP területen az SQLDelight technológia, de 2024 őszétől kezdve a Room is támogatott. Ezt userek tárolására használtam Firebase integrációval. A Firebase sajnos sokkal nehezebben használható ebben a környezetben és mivel az alkalamzás jelenlei állapotában a usereknek nincs jelentősség így ezek nem kerültek be a végső alkalamzásba.

Egy korábbi KMP ViewModel alternatíva a moko viewmodel (Mobile Kotlin Model-View-ViewModel architecture) volt, de ennél kényelmesebb az Androidos. Mivel nem volt szükséges így a user- és login servicekre és lokális adatbázisra így elvettem a dependency inkection használatát, mert így már nem okozott akorra problémát a viewmodelek létrehozása. Az Androidban használt Hilt itt nem használható még, és a Koin, ami egy hasonló KMP implementáció, nem könnyítené meg annyival ezt a folyamatot, mint a Hilt. Ezeknél a függőségeknél már a verziók összehangolása se volt egyértelmű feladat, minél több függőségre van szükség annál nagyobb a valószínűsége, hogy valami összeütközk vagy eltörik egy új frissítés miatt, főleg, ha az nem egy hivatalos függőség a Google vagy JetBrains által. Külön-külön rendesen működtek, de az előbb felsorolt pronlémák miatt úgy döntöttem, hogy egy egyszerűbb és letisztultabb megoldást használok a függőségek terén.

3.2. Hasonló multiplatform megoldások összehasonlítása

Négy különböző technológiát vizsgáltam meg és a korábbi tapasztalataim és információm alapján próbáltam választani közülök. Az első a Compose Multiplatform, a második a MAUI, a haramdik a React Native, a negyedik a Flutter. Az alábbiakban található egy rövid összefoglalás ezekről a technológiákról.

Kotlin- és Compose Multiplatform: A JetBrains és a Google által fejlesztett technológiák, amelyek lehetővé teszik a kód megosztását platformok között anélkül, hogy új nyelvet kellene bevezetni. Nemrégiben stabilizálták, és támogatja a felhasználói felület megosztását a Compose Multiplatform segítségével. [12]

.NET MAUI: A Microsoft C# és XAML alapú keretrendszere, amely cross-platform API-kat, hot reload-ot biztosít, és asztali, illetve mobil platformokra céloz. [12]

React Native: A Meta JavaScript alapú keretrendszere, amely a natív UI-t helyezi előtérbe, erős közösségi támogatással, a Fast Refresh-t használja, és a Flipper-t integrálja hibakereséshez. [12]

Flutter: A Google Dart alapú keretrendszere, amely a hot reload és az egyéni renderelés révén ismert, támogatja a Material Design-t, és széles körben használják cross-platform alkalmazásokhoz (pl. eBay, Alibaba). [12]

A kiválásztás során számos szempontot figyelembe vettem. Fontos volt a tanulási lehetőség a témából, lehetőleg olyan módon, hogy azt később is tudjam kamatoztatni. Egy másik szempont volt, hogy ne legyen teljesen ismeretlen a hasznát eszköz és környezet, ennek célja az volt, hogy az időm nagy részét ne tutorial videók nézésvel töltesem az alapoktól kezdve, hanem a dokumentációk alapján is el tudjak igazodni hatékonyan. Nem utolsó szempont volt, hogy szerettem volna folytatni az előző félévben elkezdett alkalmazásomat. Az összes opció megvizsgálása után alkalmasnak találtam a Compose Multiplatformot a szakdolgozat témájaként.

Ezek küzül az utolsót zártam ki a leghamrabb, a fenti négy közül ez az egyetlen ami nem natív UI eléményt nyújt a felhasználóknak és egy számomra teljesen ismeretlen nyelven a Google által létrehozott nyelvben a Dartsban kell programozni. Első sorban mobilos fejlesztésre használják, a többi része nem a legoptimálisabb és én ennél egy általánosabb megoldást kerestem. [31]

A következő technológia amit kizártam az a React Native volt, ez a React egyfajta változata ami natív élményt tud nyújtani weben, Androidon, iOSen és asztali alkalmazáson is. Ez egy JavaScript/TypeScript és HTML alapú megoldás és a korábbi tapasztalataim alapján ezek a nyelvek nem alkalmasak egy nagy mérteű projket fejlesztésre és karabantartására. JavaScriptben gyorsan lehet fejleszteni, de nagyon nehéz utána a kód továbbfejlesztése és karbantartása. [33]

A MAUI egy .NET alapú megoldás, így C# nyelven lehet programozni. A UI XAML alapokon nyugszik, hasonlóan, mint a WinUI. Így ez a megoldás az Windowsos asztali alkalmazásokhoz hasonlít a legjobban felfogásban és programozásban. Volt több elődje is, de jelenleg ez a legújabb változata a Windows alapokon nyugvó multiplatform fejlesztésnek. A legfőbb indok az volt, hogy én elsősorban Android fejlesztő vagyok és, ha van lehetőség akkor abból az irányból szerettem volna kiindulni. Egy másik érv ellene, hogy webes irányban nincs elmozdulás, míg az eleőző kettőnél igen, illetve a Compose Multiplatform is nyit ebeb az irányba. [22]

Mindent összevetve a Compose Multiplatform volt számora a legalkalmasabb és legérdekesebb technológia. Ebben volt a legtöbb tapasztalatom és a későbbiekben is szeretnék ezen a területen dolgozni, akár natív Android fejlesztés vagy a Compose és Kotlin Multiplatform világában. Továbbá mindig érdekes kihívás egy éppen aktívan fejlődő technológit megismerni és dolgozni benne. Így azt is ki tudtam próbálni, hogy milyen nehézségek járnak egy natív alkalmazás multiplatformba való átültetésével.

4. fejezet

Felsőszintű architektúra

Ebben a fejezetben bemutatom a szoftver felépítését. Ez a rész első sorban az architektúrális- és programszervezési megoldásokra fókuszál. Hasonlóan a 2. fejezet fejezethez ez is csak egy átfogó képet fog adni a szoftverről, a részletekbe nem merül el.

4.1. High level architektúra

Ez az alfejezet bemutatja a forráskód szerevzését és a legfontosabb követet tervezési mintákat.

4.1.1. A program struktúrális szervezése

Ez a rész a kód forrásfájlokba szervezését mutatja be. A hatékony munkához elengedhetetlen egy jól felépített és követett mintát konzisztensen használni a fejlesztés során.

4.1.1.1. A Kotlin Multiplatform alkalmazások felépítése

Minden Kotlin Multiplatform projekt tartalmaz legalább 3 Main könyvtárat az src mappán belül. Szükséges egy commonMain mappa, ami a közös kódrészleteket tartalmazza. Minél több tartalom található ebben megvalóstíva és nem külön kiszervezve annál jobb a kód újrafelhasználhatóságunk. A többi Main mappa a platform specifikus kódrészleteket tartalmazza. Az én alaklamazásomban van egy androidMain és egy desktopMain, ez a jvmMain-nek felelthető meg. Sajnos eszközhiány miatt iOS-re nem tudtam elkészíteni az alkalamzást, mivel annak lebuildeléséhez szükség van egy Mac számítógépre.

4.1. ábra. Fájl struktúrája az alkalmazásnak. Második kép: [7]

A build megfelelő működéséért a Run Configurations fájlok és a build.gradle.kts fájlok felelnek. A gradel fájlokban kell a függőségeket megadni, a verziók támogatására használható a libs.version.toml fájl.

A 4.1. ábrán is látható a build.gradle.kts fájlt (4.1. kódrészlet) is. Az ebben lévő Kotlin DSL-lel létrehozott Gradlenek is tükrözni kell ezt a felépítést.

```
import org.jetbrains.compose.desktop.application.dsl.TargetFormat //Importok
...
//Szükséges pluginok
plugins {  alias(libs.plugins.kotlinMultiplatform)
    ...
}

kotlin {    // Projekt struktúrájának létrehozása
    androidTarget {   // Android taget beállítása
        ...
    }
    jvm("desktop")  //Asztali alkalmazás beállítása

    sourceSets {    Itt kerülnek hozzáadásra a függőségek a különböző platformokhoz
        val desktopMain by getting

        //Android specifikus függőségek
        androidMain.dependencies {  implementation(libs.androidx.activity.compose)
            ...
        }
        //Minden támogatott alkalmazás által használt függőségek
        commonMain.dependencies { implementation(compose.foundation)
            ...
        }
        //Asztali alkalmazás specifikus függőségek
        desktopMain.dependencies { implementation(compose.desktop.currentOs)
            ...
        }
    }
}

//Egyéb Android beállítások
android {  namespace = "hu.bme.aut.android.examapp"
    ...
}
// Szükséges függőségek
dependencies {  implementation(libs.androidx.ui.android)
    ...
}
//Egyéb asztali beállítások
compose.desktop { application {mmainClass = "hu.bme.aut.android.examapp.MainKt"
        ...
    }
    ...
}
```

4.1. kódrészlet. build.gradle.kts fájl struktúrája

Az alkalamzást az összes platformra le kell fordítani, így szükség van egy belépési pontra minden eszköznél. Ez értelem szerűen nem lehet a közös kódbázisban, mivel minden platform más módon tud inicializálódni. Ellenben az a kód amit itt meg kell hívni, az már lehet egy közös Composable függvény. (4.5. kódrészlet) Egy egyszerűbb alkalamzásnál, ahol megfelelő közös nézetet tudunk létrehozni ennyi platform specifikus kód elég is. Az optimális a fenti lenne, de ez általban nem lehetséges, így szükség van a közös kódból való "kilépésre" és a platform specifikus kód meghívására. Ez egyszerűen megvalósítható az expect és actual függvények és osztályok segítségével (3.1.8. alszakasz).

A fő mappastruktúrán belül is fontos a tervezés. 2 fő részből tevődik össze a UI és a Service mappákból. A Service tartalmaz minden közvetlenül nem a megjelenített képernyőkkel foglalkozó részeket. Itt található a navigáció, a PDF rendrelése, és kivételesen

itt van az ahhoz tartozó képernyő is. Továbbá a szövegfelismerés és a HTTP kommunikáció is itt található. Az utóbb az api mappában és ezen belül találhatók a dto-k (data transfer object) amik az alkalmazás modell rétegét adják.

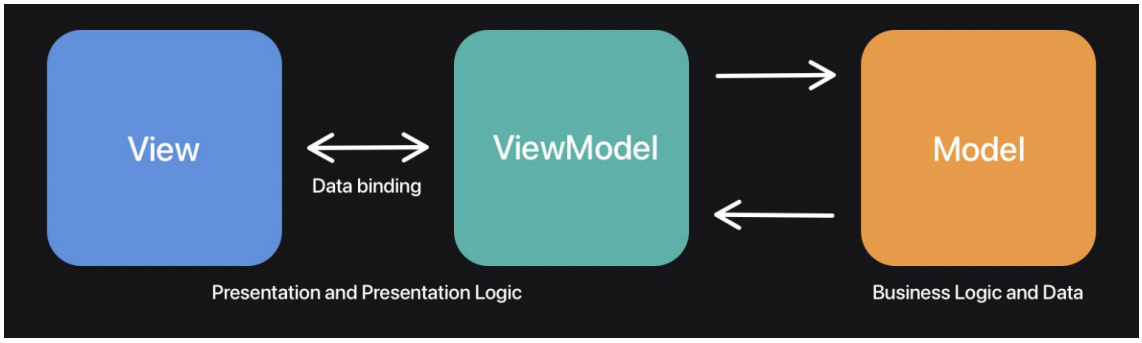
A UI mappában minden használati esetre van egy saját mappa ami a képernyőket tartalmazza, ezen kívül itt van a viewmodel mappa is ahol hasonló szervezési struktúrban találhatóak a képenyőkhöz tartozó ViewModelek. Erre is több lehetséges megoldás lenne, például egy közös ViewModel a közös tematikájú képernyőkhöz. Ebben a mappában van még a components menü is ami azokat a Composable függvényeket tartalmazza amit több képernyő is fel tud használni, például egy dropdown lista vagy a navigációs TopAppBar.

Egy másfajta tervezés lehet az is, hogy a képernyőt leíró fájlok mellett közvetlenül vannak a viewmodelek és ezek vannak közös mappákba szervezve. Ezt elvettem, mivel így is túl sok mappa volt és nehezebben láttam át ezzel a második megoldással a program szerkezetét.

A legtöbb fájl a commonMainben van, így a platform specifikus mappákban nem találhatóó meg minden mappa a fentiek közül, de a szükség actual függvények ennek megfelelő szerkezetben találhatóak meg itt is.

4.1.1.2. Követett tervezési minták

Az első és legfontosabb tervezési minta az **MVVM architektúra**. Ebben a mintában három fő komponenst lehet elkülöníteni egymástól: View, ViewModel és Model. Mindegyiknek megvan a szerepe és egy jól müködő és karabantartható szoftver esetén mindegyik létfontosságú. Ezek a rétegek egymást között tudnak kommunikálni, a leggyakoribb irány a View -> ViewModel -> Model -> ViewModel -> View. Tehát a View kér adatot valamilyen felhasználói esemény hatására, és a ViewModel ennek teljesítése érdekében a Modeltől kéri el a megjelenítendő adatokat. Ennek a láncnak azonban szinte bármely része megtörténhet akár tetszőleges irányban is, attól függően, hogy milyen funkcionalítással rendelkezik.



4.2. ábra. MVVM minta. [35]

Ezek közül a legegyszerűbb a **Model réteg**. (4.2. kódrészlet) Ennek egyetlen célja az adatok tárolása a memóriában. Célszerű ezt olyan formában megtenni ami már nem igányel sok átalakítást a köztes, ViewModel rétegben. Az adatok gyakran valamilyen REST APIból érkeznek, tehát egy másik szoftvertől. Ehhez úgynevezett DTO-kat (data transfer object) használunk, az én programomban ezek lényegében megegyeznek a modellekkel, mivel kifejzetten erre készült a REST API. Természetesen így is van szükség egyéb data classokra vagyis valós modellekre. A UInak szüksége lehet egyéb értékekre, például, hogy megfelelő módon van-e kitöltve létrehozás során az objektum vagy sem. Továbbá az összetett elemek esetén a hivatkozásokat a uuidkon keresztül fel kell oldani megjelenítés

előtt. Ezt a szerepet általában már a ViewModel végzi el. Amennyiben nem saját adatforrást használunk, vagy komplexebb adatokat kapunk érdemes külön modell osztályokat létrehozni csak a szükséges adatokkal tovább dolgozni. Ezt még a HTTP kommunikáciárt felelős komponensben érdemes megtenni.

```
import kotlinx.serialization.Serializable // Szerilizáció

@Serializable // Szerilizációért felelős kódrész
data class TopicDto(    // Kotlin data class
    val uuid: String = "",
    val topic: String,
    val description: String,
    val parentTopic: String = ""
)
```

4.2. kódrészlet. Egy dto struktúrája. Szerilizáció: 3.1.3. alszakasz

A leggyakoribb mód erre a HTTP alapú kommunikáció, ahol az adatok JSON formátumban érkeznek. Elképezelehtő erre más alternatíve, például az adatok XML formátumban érkeznek. Más módon működő rendszerek is lehetnek, például WebSocketek, ahol az első csatlakozás során kialakul egy kétirányú csatorna ahol, mind a szerver mind a kliens tud kommunikácót kezdeményezni. IoT környezetben az MQTT protocolt szokás alkalamazni. "Ez egy pub/sub protokoll, amely kis csomagmérettel és alacsony sávszélességgel rendelkezik, így ideális korlátozott hálózatokhoz és alacsony feldolgozási teljesítményű eszközökhöz. Az MQTT képes kezelni a szakaszos hálózati kapcsolódást, és támogatja a szolgáltatásminőség (QoS) szinteket a megbízható üzenettovábbítás biztosítása érdekében." [16] Egy további megoldás a gRPC ami valós alternatíva lehet a REST API kommunikációra ebben a környezetben is. "A gRPC egy nyílt forráskódú keretrendszer, amelyet a Google fejlesztett RPC API-k építésére. Lehetővé teszi a fejlesztők számára, hogy szolgáltatás-interfészeket definiáljanak, valamint kliens- és szerveroldali kódot generáljanak több programozási nyelven. A gRPC protokoll buffereket és nyelvfüggetlen adat-szerializációs formátumot használ a hatékony adatátvitel érdekében, ami ideálissá teszi magas teljesítményt igénylő alkalmazások számára. A gRPC nem feltétlenül a legjobb választás nagy mennyiségű adatmanipulációhoz vagy olyan alkalmazásokhoz, amelyek széleskörű kliens támogatást igényelnek. Ugyanakkor a gRPC magas teljesítményéről és alacsony erőforrásigényéről ismert, így jó választás azokhoz az alkalmazásokhoz, amelyek gyors és hatékony kommunikációt igényelnek a szolgáltatások között." [16]

A köztes szinten a **ViewModel** található. Megoldáshoz az Android ViewModel Kotlin Multiplatform implementációját használtam fel (3.1.1.2. alalszakasz), de bármi hasonló megoldás megfelel. Ennek a rétegnek számos feladata van. Az adatok lekérdezése itt valósul meg. Úgynevezett coroutine scopeokban kezdeményezhetünk hosszabb ideig tartó függvényhívásokat. "A Kotlin standard könyvtára csak minimális alacsony szintű API-kat biztosít, hogy más könyvtárak használhassák a koroutinekat. Ellentétben sok más hasonló képességű nyelvvel, az async és await nem kulcsszavak a Kotlinban, és még a standard könyvtár részei sem. Ezenkívül a Kotlin felfüggeszthető függvény (suspending function) koncepciója biztonságosabb és kevésbé hibára hajlamos absztrakciót kínál az aszinkron műveletekhez, mint a jövők (futures) és ígéretek (promises). A kotlinx.coroutines egy JetBrains által fejlesztett gazdag könyvtár koroutinekhoz. Számos magas szintű, koroutinekompatibilis primitívet tartalmaz, amelyeket ez az útmutató is tárgyal, beleértve a launch, async és más függvényeket." [6] Röviden összefoglalva aszinkron módon működnek így és nem UI szálon hajtódnak végre ezáltal nem blokkolják azt és reszponzív marad a teljes idő alatt. Mindebből következik, hogy a REST API service hívásai innen indulnak, majd kerülnek bele Statekbe vagy StateFlowkba (3.1.1.1. alalszakasz). Egy érdekesebb megközelítés a LiveData. Ez az eszköz az Observer mintát valósítja meg. Amennyiben például

egy WebSocket alapú megoldást választottam volna, akkor a szerver autmoatiksan tudna üzenetet üldeni amiben leküldi az új adatot a kliensnek és egyéb frissétés nélkül megjelne az új adat. Ez egy példa a Model -> ViewModel -> View irányra. A legtöbb chat applikáció (például Messenger) is hasonéó megközelítést alkalmazhat.

A ViewModelnek további feladatai közé tartozik az adatok transzformálása. Szükség lehet a hivatkozások feloldására például egy kérdés esetén nem egy uuid-t szeretne a felhasználó látni, a pont és a téma mezőkben, hanem azok neveit. Ezen kívül biztosíthat egyéb függvényeket az adatok módosítására, tovább navigálásra vagy bármilyen általános célokra. Ezzel elkerülehtő, hogy logikát leíró kódot a Viewn kelljen definálni, elég egy összefogó részben levő függvényt meghívni. (4.3. kódrészlet)

```
data class TopicDetails(
    val id: String = "",
    val topic: String = "",
    val parent: String = "",            //Ez itt még egy uuid
    val description: String = "",
    val parentTopicName : String = ""   //Itt már fel lett oldva, ez fog a UI-on megjelenni
)

fun TopicDetails.toTopic(): TopicDto = //Ez egy extension function ami visszaalakítja a UIon mgejelen
    ő adatot a REST API által kezelhrtő formára
    TopicDto(
        uuid = id,
        topic = topic,
        parentTopic = if (parent == "null") "" else parent,
        description = description,
    )

// A névfeloldott változat kiégszül egy valid mezővel, mivel ez egy szrekeszthető nézethez tartozó
    modell lesz
fun TopicDto.toTopicUiState(isEntryValid: Boolean = false, parentName: String): TopicUiState =
    TopicUiState(
    topicDetails = this.toTopicDetails(parentName),
    isEntryValid = isEntryValid
)

//Biztosított függvény az adatok módosítására
fun updateUiState(topicDetails: TopicDetails) {
    topicUiState =
        TopicUiState(topicDetails = topicDetails, isEntryValid = validateInput(topicDetails))
}
```

4.3. kódrészlet. Példa az dto átalakítására a valós UIon használt modellé és egy példa logikát leíró függvényre. Az eredeti dto: 4.2. kódrészlet

A **View** a megjelenítési réteg, a felhasználó ezzel a résszel tud közvetlenül interakcióba lépni. Általános működési elve a következő: amikor a felhasználó az adott képernyőre navigál az ViewModel betölti az adatokat. Ezt kövtően a felhasználó szabadon végezhet műveleteket rajta. Ahogy a 2.1. ábrán is látható megtekintheti az adott oldalt, esetenként törülheti vagy szerkesztheti az adatokat vagy felvehet újat. Ezeket a funkciókat a megfelelő ViewModel beli függvény meghívásával érhetjük el és valamely felhasználói interakció váltja ki például egy gomb megnyomása.

```
@Composable
private fun NewTopicScreenUiState(
    viewModel: TopicEntryViewModel, ...     // A használt ViewModel
) {
    ...
    Scaffold(...) { innerPadding ->
        TopicEntryBody(
            topicUiState = viewModel.topicUiState,          // A megjelenítendő UI adat
            onTopicValueChange = viewModel::updateUiState,  // ViewModelben definált függvény paramé
    terként való átadása
            onSaveClick = {
```

```
                coroutineScope.launch {                    // Aszinkron módon történő hívás, mentés
    funkció. Szintén a ViewModel függvényét hívja meg
                    if (viewModel.saveTopic()) {
                        navigateBack()
                    } else {
                        showNotify = true
                        notifyMessage = "Topic with this name already exists"
                    }
                }
            },
            modifier = Modifier                            // Fontos az igényes megjelenítés
                .padding(innerPadding)
                .verticalScroll(rememberScrollState())
                .fillMaxWidth()
        )
    }
}
```

4.4. kódrészlet. Egy a View rétegbe tartozó felület leírásának részlete

Összefoglalva az MVVM tervezési mintát azt lehet mondani, hogy nagyon hatékonyan használható és önmagában ezzel jelentősen növelhető a kód újrafelhasználhatósága. A mai lehetőségeket kihasználva a Compose Multiplatform területén ViewModeleket, dto-kat és Modeleket elegendő egyszer, a commonMain alatt létrehozni és megvalósítani. Az ezekhez tartozó esetleges függvényeket, amennyiben szükség van rá elegendő egy actual/expect párral lecserélni, nálam a kamera és a PDF exportális is így működik. (4.6. kódrészlet) Az **expect/actual tervezési elv** működik minden rétegben, de általában csak a UI és néhány specifikus funkció megvalósítására van szükség a logika megvalósításához. Ilyenkor is elég egy közös ViewModel és a build rendszer behelyesíti a megfelelő függényt a platformnak megfelelően.

```
// Tipikus Android compose alkalamzás. (androidMain)
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            App()
        }
    }
}

//Kotlin swing alkalmazás. (desktopMain)
fun main() = application {
    Window(
        onCloseRequest = ::exitApplication,
        title = "Exam App",
    ) {
        App()
    }
}

//A hívott közös kód. (commonMain)
@Composable
fun App() {
    MaterialTheme {
        NavigationComponent()
    }
}
```

4.5. kódrészlet. Alkalamzás elindítása

```
//Közös függvény fejléc, lehet alapértelmezett paramétere is.
@Composable
internal expect fun Notify(message: String)


//Android megvalósítás.
@Composable
```

```
internal actual fun Notify(message: String) {
    Toast.makeText(
        LocalContext.current, message, Toast.LENGTH_SHORT
    ).show()
}

//Asztali alkalmazás megvalósítása.
@Composable
internal actual fun Notify(message: String) {
    if (SystemTray.isSupported()) {
        val tray = SystemTray.getSystemTray()
        val image = Toolkit.getDefaultToolkit().createImage("logo.webp")
        val trayIcon = TrayIcon(image, "Desktop Notification")
        tray.add(trayIcon)
        trayIcon.displayMessage("Desktop Notification", message, TrayIcon.MessageType.INFO)
    } else {
        ...
    }
}
```

4.6. kódrészlet. Egyszerűbb példa az expect és actual függvények használatára. Debugolás során használt kódrészlet.

A másik legfontosabb architekturális elem a UiState használata. (4.7. kódrészlet) Ezzel a mintával egyszerűen kezelhető több fajta képernyő ugyan azon az oldalon. A megvalóstíása sealed interfészek segítségével történik, így a Kotlinban a switc-case szerkezetnek megfeleltethető when használatával egyszerűen válthatunk közöttük. Ebben a mintában gyakran három állapta van a képernyőnek: loading, succes és error.

A hosszú ideig tartó HTTP kérések ideje alatt így egy töltőképernyőt mutathatunk egy látszólag lefagyott vagy hiányos oldal helyett. Az adatok megérkezése után átadhatjuk azokat a Succes képernyőnek, hiba esetén pedig mutathatjuk a hiba képernyőt ahol található valamilyen magyarázat a felhasználó számára, például "Nem sikerült betölteni a kért oladlt mert a hálózat állapota nem megfelelő".

```
sealed interface TopicDetailsScreenUiState {
    data class Success(val point: TopicDto) : TopicDetailsScreenUiState
    data object Error : TopicDetailsScreenUiState{var errorMessage: String = ""}
    data object Loading : TopicDetailsScreenUiState
}
```

4.7. kódrészlet. UiState megvalóstása

A képernyő állaptát a ViewModelből lehet szabályozni, ahol felveszünk erre egy State-et. A képernyőre navigálás során az alapértelmezett érték a töltés. Az adatok megérkezése után a staten beéllítjuk a megfelelő értéket és a View erről kap egy értesítést és lefut a recomposoition. Ennek hatására megjelenik az új nézet. (4.8. kódrészlet)

```
var topicDetailsScreenUiState: TopicDetailsScreenUiState by mutableStateOf(TopicDetailsScreenUiState.
    Loading) // State alap loading értékkel
fun getTopic(topicId: String){
    topicDetailsScreenUiState = TopicDetailsScreenUiState.Loading   // Ha nem lenne beállítva beállí
    tjuk a loadingot
    viewModelScope.launch { // Aszinkron hívás környezete, így a loading animációt nemblokkoljuk
        topicDetailsScreenUiState = try{    // Az eredméyn megérkezése után vagy mutatjuk az adatot
    ...
            val result = ApiService.getTopic(topicId)
            ...
        TopicDetailsScreenUiState.Success(result)
        } catch (e: ApiException) { // ...vagy egy hiba képernyőt
            TopicDetailsScreenUiState.Error.errorMessage = e.message ?: "Unkown error"
            TopicDetailsScreenUiState.Error
        }
    }
}
```

4.8. kódrészlet. UiState beállítása

Összefoglalva ezt a mintát: a felhasználót aktívan tudjuk tájékoztatni az program állapotáról. A töltő képernyő miatt tudni fogja, hogy a háttérben tart az adatok betöltése és nem lesz olyan érzése mintha lefagyott volna az alkalmazás. Hiba esetén is tudunk tájékoztatást adni így lehetőséget adunk a felhasználónak, hogyha nála van a hiba (nincs internet) megoldja, vagy esetlegesen tudja jelezni a megadott elérhetőségen, hogy problémát tapasztal a működésben és a megjelenített információ miatt ezt vszonylag pontosan meg tudja tenni. Ezek hatására a felhasználói élmény jelentősen javulni tud.

4.2. Rendszer felépítései, komponensei

Ebben az alfejezetben komponens diagramokon keresztül mutatom be az alaklamzást. Elsőnek adok egy általános nézetet, majd a fontosabb, több tartalommal rendelkező egységeket a kisebb alegységein is bemutatom.

"Az UML komponensdiagramokat az objektumorientált rendszerek fizikai aspektusainak modellezésére használják, amelyek célja a komponensalapú rendszerek vizualizálása, specifikálása és dokumentálása, valamint végrehajtható rendszerek létrehozása előre- és visszafelé történő tervezéssel. A komponensdiagramok lényegében osztálydiagramok, amelyek a rendszer komponenseire összpontosítanak, és gyakran a rendszer statikus implementációs nézetének modellezésére használják." [34]

Ebből az idézetből kiderül, hogy ebben a diagram típusban a fizikai egységek jelennek meg. Ezt érthetjük szó szerint is, például más gépen futó backend és frontend, de akár önálló fordítási egységet is érthetünk alatta. Ilyen önálló egység lehet a bussines logic réteg, .NET környezetben ez egy teljesen általános megoldás, ahol valódi fordítási egységet alkot és önálló DLL jön belőle létre. Hasonló módon foghatjuk fel a Compose Multiplatformban szereplő közös és specifikus kódokat is.

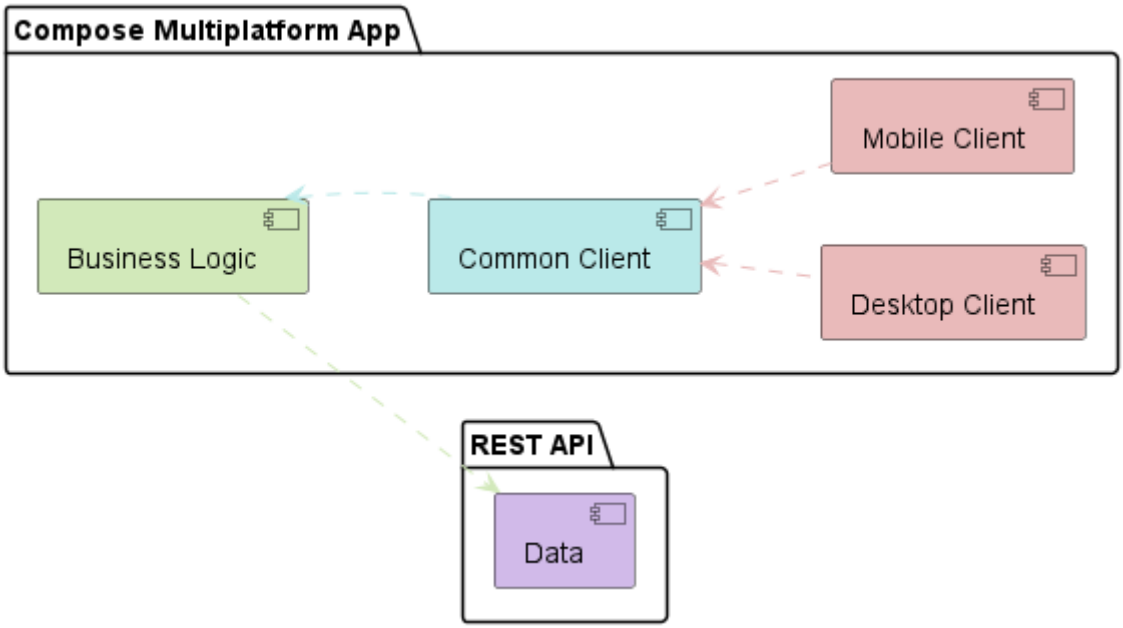
4.2.1. Általános komponensek felépítése

Két fő komponensből áll az alklamazás, a Compose Multiplatform applikációból és az adatokat kiszolgáló REST APIból. (4.3. ábra) Ebben a félévben a klines oldalon volt a hangsúly, ezért a backend részt nem bontom fel további részekre, kezelhetjük egyfajta *"black box"*-ként.

A kliens oldalon a bussines logic komponens kommunikál a backenddel, így függ tőle. Az abban történő változás vagy hiba kihat ennek a komponensnek a működésére. Többek között ebben az egységben találhatóak a ViewModelek és a Servicek közé sorolt funkciók.

A Common Client rész tartalmazza a View réteget. Itt találhatóak a Composable függényekkel leírt képernyők. Mivel ezeken az elemeken a ViewModelből származó adatok jelnnek meg a helyes működés szerint, ezért függ a Business Logictól. Ezáltal tranzitívan függ a a backend helyes működésétől is. Ennek a felépítéséről és komponenseirpl lesz egy részletesebb ábra.

Megjelnnek még az ábrán a Mobile Client és Desktop Client részek is. Ez a két komponens valósítja meg a platform specifikus kódrészletek megvalósításáért és a belépési pontok megvalósításáért felel. Itt is érdemes észrevenni, hogy ezek a komponensek függnek a közös kódtól. Ha megváltozik valami benne az minden platform specifikus kódra hatással lesz, ha létrehozunk egy expect függvényt azt minden más komponensben meg kell valósítani. Ezekről nem fogok ebben a fejezetben részletesebben foglalkozni.



4.3. ábra. Átfogó komponens diagram.

4.2.2. Business logic komponensek felépítése

A **bussines logic komponens** összetett és több alkotóelemből épül fel. (4.4. ábra) A ViewModelek önmagukban lehetnek komponensek, akár külön-külön is, de a diagramon egybe tüntetem fel. Ezek a viewModelek függnek a Servicektől.

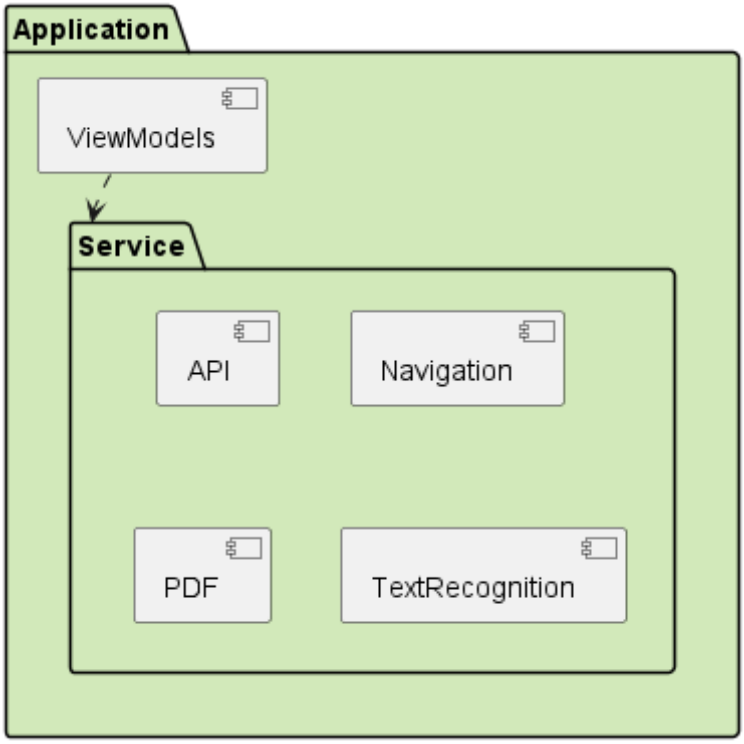
A Serviceek közé tartozik az API kommunikáció, a navigáciért felelős komponens is. A PDF elkészítéséért felelős kódrészek is egy komponensként kezelhető, hasonló módon, mint a szövegfelismerésért felelős funkció is. Az ebben az egységebn levő komponensek nagyrészt nem függnek szorosan más komponensektől, ez alól kivétel a HTTP kommunikáció ami a backendtől függ.

4.2.3. Common Client komponensek felépítése

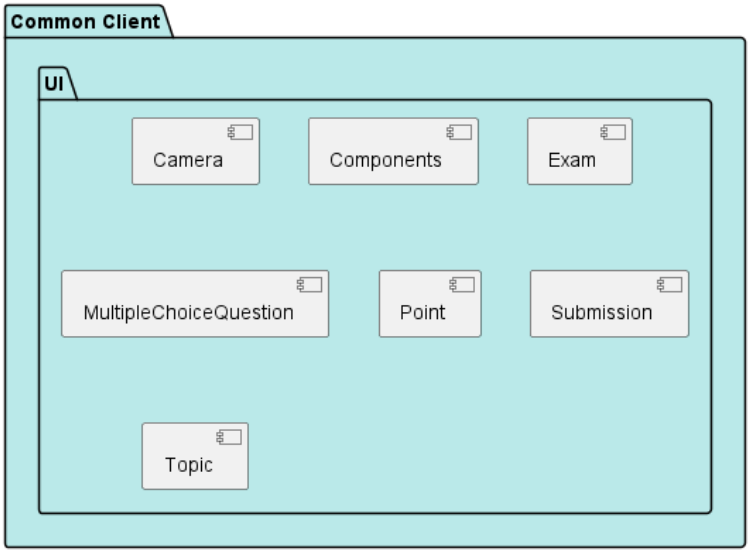
A **common Client komponens** is számos részből épül fel. (4.5. ábra) Ebben a részben találhatóak a képernyőn megjelenő elemek leírásásért felelős részek.

A kisebb komponenseket tartalmazó résztőll több másik is függ, ez az ábrán az átláthatósag miatt nem szerepel. Egy-egy ilyen UI komponens egy vagy több viewModeltől is függhet, de egymás között nincs egyéb függőség, így szabadan eltávilíthatóak vagy hozzáadhatóak új nézetek.

Ezektől a komponensektől a tényleges implementációk függnek, így a Mobil és asztali komponens is. Az itt történt bármilyen változás kihatással van a valós implementációkra. A közös elemek módosítása megváltoztatja minden platformon a képernyő kinézetét. Ha új expect függvényt veszünk fel, a többi kompenenst is külön el kell készítenünk.



4.4. ábra. Business logic komponens diagram.



4.5. ábra. Közös UI komponens diagram.

5. fejezet

Részletes megvalósítás

6. fejezet

A LaTeX-sablon használata

Ebben a fejezetben röviden, implicit módon bemutatjuk a sablon használatának módját, ami azt jelenti, hogy sablon használata ennek a dokumentumnak a forráskódját tanulmányozva válik teljesen világossá. Amennyiben a szoftver-keretrendszer telepítve van, a sablon alkalmazása és a dolgozat szerkesztése LaTeX-ben a sablon segítségével tapasztalataink szerint jóval hatékonyabb, mint egy WYSWYG (*What You See is What You Get*) típusú szövegszerkesztő esetén (pl. Microsoft Word, OpenOffice).

6.1. Címkék és hivatkozások

A LaTeX dokumentumban címkéket (`\label`) rendelhetünk ábrákhoz, táblázatokhoz, fejezetekhez, listákhoz, képletekhez stb. Ezekre a dokumentum bármely részében hivatkozhatunk, a hivatkozások automatikusan feloldásra kerülnek.

A sablonban makrókat definiáltunk a hivatkozások megkönnyítéséhez. Ennek megfelelően minden ábra (*figure*) címkéje `fig:` kulcsszóval kezdődik, míg minden táblázat (*table*), képlet (*equation*), fejezet (*section*) és lista (*listing*) rendre a `tab:`, `eq:`, `sec:` és `lst:` kulcsszóval kezdődik, és a kulcsszavak után tetszőlegesen választott címke használható. Ha ezt a konvenciót betartjuk, akkor az előbbi objektumok számára rendre a `\figref`, `\tabref`, `\eqref`, `\sectref` és `\listref` makrókkal hivatkozhatunk. A makrók paramétere a címke, amelyre hivatkozunk (a kulcsszó nélkül). Az összes említett hivatkozástípus, beleértve az `\url` kulcsszóval bevezetett web-hivatkozásokat is a `hyperref`[1] csomagnak köszönhetően aktívak a legtöbb PDF-nézegetőben, rájuk kattintva a dokumentum megfelelő oldalára ugrik a PDF-néző vagy a megfelelő linket megnyitja az alapértelmezett böngészővel. A `hyperref` csomag a kimeneti PDF-dokumentumba könyvjelzőket is készít a tartalomjegyzékből. Ez egy szintén aktív tartalomjegyzék, amelynek elemeire kattintva a nézegető behozza a kiválasztott fejezetet.

6.2. Ábrák és táblázatok

Használjunk vektorgrafikus ábrákat, ha van rá módunk. PDFLaTeX használata esetén PDF formátumú ábrákat lehet beilleszteni könnyen, az EPS (PostScript) vektorgrafikus képformátum beillesztését a PDFLaTeX közvetlenül nem támogatja (de lehet konvertálni, lásd később). Ha vektorgrafikus formában nem áll rendelkezésünkre az ábra, akkor a

[1] Segítségével a dokumentumban megjelenő hivatkozások nem csak dinamikussá válnak, de színezhetők is, bővebbet erről a csomag dokumentációjában találunk. Ez egyúttal egy példa lábjegyzet írására.

veszteségmentes PNG, valamint a veszteséges JPEG formátumban érdemes elmenteni. Figyeljünk arra, hogy ilyenkor a képek felbontása elég nagy legyen ahhoz, hogy nyomtatásban is megfelelő minőséget nyújtson (legalább 300 dpi javasolt). A dokumentumban felhasznált képfájlokat a dokumentum forrása mellett érdemes tartani, archiválni, mivel ezek hiányában a dokumentum nem fordul újra. Ha lehet, a vektorgrafikus képeket vektorgrafikus formátumban is érdemes elmenteni az újrafelhasználhatóság (az átszerkeszthetőség) érdekében.

Kapcsolási rajzok legtöbbször kimásolhatók egy vektorgrafikus programba (pl. CorelDraw) és onnan nagyobb felbontással raszterizálva kimenthatők PNG formátumban. Ugyanakkor kiváló ábrák készíthetők Microsoft Visio vagy hasonló program használatával is: Visio-ból az ábrák közvetlenül PDF-be is menthetők.

Lehetőségeink Matlab ábrák esetén:

- Képernyőlopás (*screenshot*) is elfogadható minőségű lehet a dokumentumban, de általában jobb felbontást is el lehet érni más módszerrel.
- A Matlab ábrát a `File/Save As` opcióval lementhetjük PNG formátumban (ugyanaz itt is érvényes, mint korábban, ezért nem javasoljuk).
- A Matlab ábrát az `Edit/Copy figure` opcióval kimásolhatjuk egy vektorgrafikus programba is és onnan nagyobb felbontással raszterizálva kimenthatjük PNG formátumban (nem javasolt).
- Javasolt megoldás: az ábrát a `File/Save As` opcióval EPS *vektorgrafikus* formátumban elmentjük, PDF-be konvertálva beillesztjük a dolgozatba.

Az EPS kép az `epstopdf` programmal[2] konvertálható PDF formátumba. Célszerű egy batch-fájlt készíteni az összes EPS ábra lefordítására az alábbi módon (ez Windows alatt működik).

```
@echo off
for %%j in (*.eps) do (
  echo converting file "%%j"
  epstopdf "%%j"
)
echo done .
```

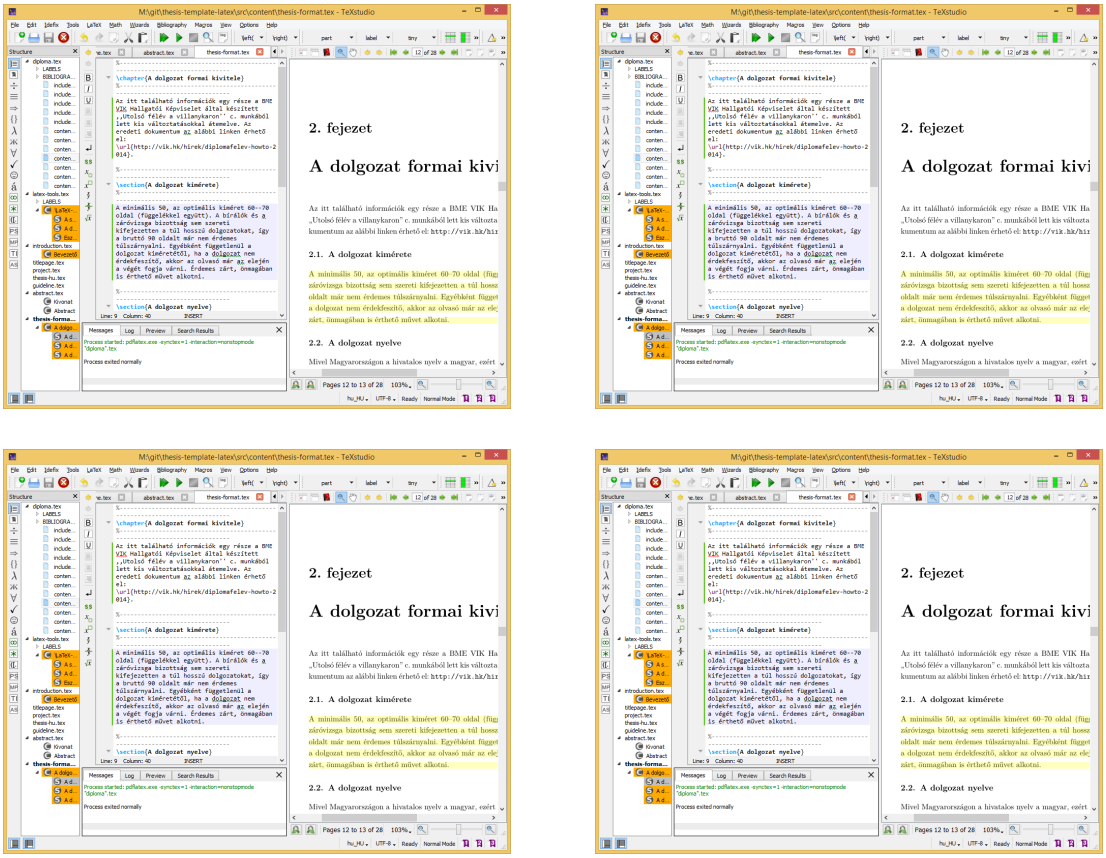
Egy ilyen parancsfájlt (`convert.cmd`) elhelyeztük a sablon `figures\eps` könyvtárába, így a felhasználónak csak annyi a dolga, hogy a `figures\eps` könyvtárba kimenti az EPS formátumú vektorgrafikus képet, majd lefuttatja a `convert.cmd` parancsfájlt, ami PDF-be konvertálja az EPS fájlt.

Ezek után a PDF-ábrát ugyanúgy lehet a dokumentumba beilleszteni, mint a PNG-t vagy a JPEG-et. A megoldás előnye, hogy a lefordított dokumentumban is vektorgrafikusan tárolódik az ábra, így a mérete jóval kisebb, mintha raszterizáltuk volna beillesztés előtt. Ez a módszer minden – az EPS formátumot ismerő – vektorgrafikus program (pl. CorelDraw) esetén is használható.

A képek beillesztésére a **??**. **??**+ben mutattunk be példát (**??**. **??**+). Az előző mondatban egyúttal az automatikusan feloldódó ábrahivatkozásra is láthatunk példát. Több képfájlt is beilleszthetünk egyetlen ábrába. Az egyes képek közötti horizontális és vertikális margót metrikusan szabályozhatjuk (6.1. ábra). Az ábrák elhelyezését számtalan tipográfiai szabály egyidejű teljesítésével a fordító maga végzi, a dokumentum írója csak

[2] a korábban említett LaTeX-disztribúciókban megtalálható

preferenciáit jelezheti a fordító felé (olykor ez bosszúságot is okozhat, ilyenkor pl. a kép méretével lehet játszani).



6.1. ábra. Több képfájl beillesztése esetén térközöket is érdemes használni.

A táblázatok használatára a 6.1 táblázat mutat példát. A táblázatok formázásához hasznos tanácsokat találunk a `booktabs` csomag dokumentációjában.

Órajel	Frekvencia	Cél pin
CLKA	100 MHz	FPGA CLK0
CLKB	48 MHz	FPGA CLK1
CLKC	20 MHz	Processzor
CLKD	25 MHz	Ethernet chip
CLKE	72 MHz	FPGA CLK2
XBUF	20 MHz	FPGA CLK3

6.1. táblázat. Az órajel-generátor chip órajel-kimenetei.

6.3. Felsorolások és listák

Számozatlan felsorolásra mutat példát a jelenlegi bekezdés:

- *első bajusz:* ide lehetne írni az első elem kifejését,
- *második bajusz:* ide lehetne írni a második elem kifejését,

- *ez meg egy szakáll:* ide lehetne írni a harmadik elem kifejését.

Számozott felsorolást is készíthetünk az alábbi módon:

1. *első bajusz:* ide lehetne írni az első elem kifejését, és ez a kifejtés így néz ki, ha több sorosra sikeredik,
2. *második bajusz:* ide lehetne írni a második elem kifejését,
3. *ez meg egy szakáll:* ide lehetne írni a harmadik elem kifejését.

A felsorolásokban sorok végén vessző, az utolsó sor végén pedig pont a szokásos írásjel. Ez alól kivételt képezhet, ha az egyes elemek több teljes mondatot tartalmaznak.

Listákban a dolgozat szövegétől elkülönítendő kódrészleteket, programsorokat, pszeudo-kódokat jeleníthetünk meg (6.1. kódrészlet).

```
\begin{enumerate}
  \item \emph{első bajusz:} ide lehetne írni az első elem kifejését,
  és ez a kifejtés így néz ki, ha több sorosra sikeredik,
  \item \emph{második bajusz:} ide lehetne írni a második elem kifejését,
  \item \emph{ez meg egy szakáll:} ide lehetne írni a harmadik elem kifejését.
\end{enumerate}
```

6.1. kódrészlet. A fenti számozott felsorolás LaTeX-forráskódja

A lista keretét, háttérszínét, egész stílusát megválaszthatjuk. Ráadásul különféle programnyelveket és a nyelveken belül kulcsszavakat is definiálhatunk, ha szükséges. Erről bővebbet a `listings` csomag hivatalos leírásában találhatunk.

6.4. Képletek

Ha egy formula nem túlságosan hosszú, és nem akarjuk hivatkozni a szövegből, mint például a $e^{i\pi} + 1 = 0$ képlet, *szövegközi képletként* szokás leírni. Csak, hogy másik példát is lássunk, az $U_i = -d\Phi/dt$ Faraday-törvény a $\operatorname{rot} E = -\frac{dB}{dt}$ differenciális alakban adott Maxwell-egyenlet felületre vett integráljából vezethető le. Látható, hogy a LaTeX-fordító a sorközöket betartja, így a szöveg szedése esztétikus marad szövegközi képletek használata esetén is.

Képletek esetén az általános konvenció, hogy a kisbetűk skalárt, a kis félkövér betűk ($\mathbf{v}$) oszlopvektort – és ennek megfelelően $\mathbf{v}^T$ sorvektort – a kapitális félkövér betűk ($\mathbf{V}$) mátrixot jelölnek. Ha ettől el szeretnénk térni, akkor az alkalmazni kívánt jelölésmódot célszerű külön alfejezetben definiálni. Ennek megfelelően, amennyiben $\mathbf{y}$ jelöli a mérések vektorát, ϑ a paraméterek vektorát és $\hat{\mathbf{y}} = \mathbf{X}\vartheta$ a paraméterekben lineáris modellt, akkor a *Least-Squares* értelemben optimális paraméterbecslő $\hat{\vartheta}_{LS} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$ lesz.

Emellett kiemelt, sorszámozott képleteket is megadhatunk, ennél az `equation` és a `eqnarray` környezetek helyett a korszerűbb `align` környezet alkalmazását javasoljuk (több okból, különféle problémák elkerülése végett, amelyekre most nem térünk ki). Tehát

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, \tag{6.1}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x}, \tag{6.2}$$

ahol $\mathbf{x}$ az állapotvektor, $\mathbf{y}$ a mérések vektora és $\mathbf{A}$, $\mathbf{B}$ és $\mathbf{C}$ a rendszert leíró paramétermátrixok. Figyeljük meg, hogy a két egyenletben az egyenlőségjelek egymáshoz igazítva

jelennek meg, mivel a mindkettőt az & karakter előzi meg a kódban. Lehetőség van számozatlan kiemelt képlet használatára is, például

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u},$$
$$\mathbf{y} = \mathbf{C}\mathbf{x}.$$

Mátrixok felírására az $\mathbf{A}\mathbf{x} = \mathbf{b}$ inhomogén lineáris egyenlet részletes kifejtésével mutatunk példát:

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}. \tag{6.3}$$

A `\frac` utasítás hatékonyságát egy általános másodfokú tag átviteli függvényén keresztül mutatjuk be, azaz

$$W(s) = \frac{A}{1 + 2T\xi s + s^2T^2}. \tag{6.4}$$

A matematikai mód minden szimbólumának és képességének a bemutatására természetesen itt nincs lehetőség, de gyors referenciaként hatékonyan használhatók a következő linkek:

`http://www.artofproblemsolving.com/LaTeX/AoPS_L_GuideSym.php`,

`http://www.ctan.org/tex-archive/info/symbols/comprehensive/symbols-a4.pdf`,

`ftp://ftp.ams.org/pub/tex/doc/amsmath/short-math-guide.pdf`.

Ez pedig itt egy magyarázat, hogy miért érdemes `align` környezetet használni:

`http://texblog.net/latex-archive/maths/eqnarray-align-environment/`.

6.5. Irodalmi hivatkozások

Egy LaTeX dokumentumban az irodalmi hivatkozások definíciójának két módja van. Az egyik a `\thebibliograhy` környezet használata a dokumentum végén, az `\end{document}` lezárás előtt.

```
\begin{thebibliography}{9}

\bibitem{Lamport94} Leslie Lamport, \emph{\LaTeX: A Document Preparation System}.
Addison Wesley, Massachusetts, 2nd Edition, 1994.

\end{thebibliography}
```

Ezek után a dokumentumban a `\cite{Lamport94}` utasítással hivatkozhatunk a forrásra. A fenti megadás viszonylag kötetlen, a szerző maga formázza az irodalomjegyzéket (ami gyakran inkonzisztens eredményhez vezet).

Egy sokkal professzionálisabb módszer a BiBTeX használata, ezért ez a sablon is ezt támogatja. Ebben az esetben egy külön szöveges adatbázisban definiáljuk a forrásmunkákat, és egy külön stílusfájl határozza meg az irodalomjegyzék kinézetét. Ez, összhangban azzal, hogy külön formátumkonvenció határozza meg a folyóirat-, a könyv-, a konferenciacikk- stb. hivatkozások kinézetét az irodalomjegyzékben (a sablon használata esetén ezzel nem is kell foglalkoznia a hallgatónak, de az eredményt célszerű ellenőrizni). felhasznált hivatkozások adatbázisa egy `.bib` kiterjesztésű szöveges fájl,

amelynek szerkezetét a A 6.2 kódrészlet demonstrálja. A forrásmunkák bevitelekor a sor végi vesszők külön figyelmet igényelnek, mert hiányuk a BiBTEX-fordító hibaüzenetét eredményezi. A forrásmunkákat típus szerinti kulcsszó vezeti be (`@book` könyv, `@inproceedings` konferenciakiadványban megjelent cikk, `@article` folyóiratban megjelent cikk, `@techreport` valamelyik egyetem gondozásában megjelent műszaki tanulmány, `@manual` műszaki dokumentáció esetén stb.). Nemcsak a megjelenés stílusa, de a kötelezően megadandó mezők is típusról-típusra változnak. Egy jól használható referencia a `http://en.wikipedia.org/wiki/BibTeX` oldalon található.

```
@book{Wettl04,
  author    = {Ferenc Wettl and Gyula Mayer and Péter Szabó},
  publisher = {Panem Könyvkiadó},
  title     = {\LaTeX~kézikönyv},
  year      = {2004},
}

@article{Candy86,
  author       = {James C. Candy},
  journaltitle = {{IEEE} Trans.\ on Communications},
  month        = {01},
  note         = {\doi{10.1109/TCOM.1986.1096432}},
  number       = {1},
  pages        = {72--76},
  title        = {Decimation for Sigma Delta Modulation},
  volume       = {34},
  year         = {1986},
}

@inproceedings{Lee87,
  author    = {Wai L. Lee and Charles G. Sodini},
  booktitle = {Proc.\ of the IEEE International Symposium on Circuits and Systems},
  location  = {Philadelphia, PA, USA},
  month     = {05~4--7},
  pages     = {459--462},
  title     = {A Topology for Higher Order Interpolative Coders},
  vol       = {2},
  year      = {1987},
}

@thesis{KissPhD,
  author      = {Peter Kiss},
  institution = {Technical University of Timi\c{s}oara, Romania},
  month       = {04},
  title       = {Adaptive Digital Compensation of Analog Circuit Imperfections for Cascaded Delta-
     Sigma Analog-to-Digital Converters},
  type        = {phdthesis},
  year        = {2000},
}

@manual{Schreier00,
  author       = {Richard Schreier},
  month        = {01},
  note         = {\url{http://www.mathworks.com/matlabcentral/fileexchange/}},
  organization = {Oregon State University},
  title        = {The Delta-Sigma Toolbox v5.2},
  year         = {2000},
}

@misc{DipPortal,
  author       = {{Budapesti Műszaki és Gazdaságtudományi Egyetem Villamosmérnöki és Informatikai Kar
     }},
  howpublished = {\url{http://diplomaterv.vik.bme.hu/}},
  title        = {Diplomaterv portál (2011. február 26.)},
}

@incollection{Mkrtychev:1997,
  author    = {Mkrtychev, Alexey},
  booktitle = {Logical Foundations of Computer Science},
  doi       = {10.1007/3-540-63045-7_27},
  editor    = {Adian, Sergei and Nerode, Anil},
```

```
  isbn      = {978-3-540-63045-6},
  pages     = {266-275},
  publisher = {Springer Berlin Heidelberg},
  series    = {Lecture Notes in Computer Science},
  title     = {Models for the logic of proofs},
  url       = {http://dx.doi.org/10.1007/3-540-63045-7_27},
  volume    = {1234},
  year      = {1997},
}
```

6.2. kódrészlet. Példa szöveges irodalomjegyzék-adatbázisra BibTEX használata esetén.

A stílusfájl egy `.sty` kiterjesztésű fájl, de ezzel lényegében nem kell foglalkozni, mert vannak beépített stílusok, amelyek jól használhatók. Ez a sablon a BiBTEX-et használja, a hozzá tartozó adatbázisfájl a `mybib.bib` fájl. Megfigyelhető, hogy az irodalomjegyzéket a dokumentum végére (a `\end{document}` utasítás elé) beillesztett `\bibliography{mybib}` utasítással hozhatjuk létre, a stílusát pedig ugyanitt a `\bibliographystyle{plain}` utasítással adhatjuk meg. Ebben az esetben a `plain` előre definiált stílust használjuk (a sablonban is ezt állítottuk be). A `plain` stíluson kívül természetesen számtalan más előre definiált stílus is létezik. Mivel a `.bib` adatbázisban ezeket megadtuk, a BiBTEX-fordító is meg tudja különböztetni a szerzőt a címtől és a kiadótól, és ez alapján automatikusan generálódik az irodalomjegyzék a stílusfájl által meghatározott stílusban.

Az egyes forrásmunkákra a szövegből továbbra is a `\cite` paranccsal tudunk hivatkozni, így a 6.2. kódrészlet esetén a hivatkozások rendre `\cite{Wettl04}`, `\cite{Candy86}`, `\cite{Lee87}`, `\cite{KissPhD}`, `\cite{Schreirer00}`, `\cite{Mkrtychev:1997}` és `\cite{DipPortal}`. Az egyes forrásmunkák sorszáma az irodalomjegyzék bővítésekor változhat. Amennyiben az aktuális számhoz illeszkedő névelőt szeretnénk használni, használjuk az `\acite{}` parancsot.

Az irodalomjegyzékben alapértelmezésben csak azok a forrásmunkák jelennek meg, amelyekre található hivatkozás a szövegben, és ez így alapvetően helyes is, hiszen olyan forrásmunkákat nem illik az irodalomjegyzékbe írni, amelyekre nincs hivatkozás.

Mivel a fordítási folyamat során több lépésben oldódnak fel a szimbólumok, ezért gyakran többször is le kell fordítani a dokumentumot. Ilyenkor ez első 1-2 fordítás esetleg szimbólum-feloldásra vonatkozó figyelmeztető üzenettel zárul. Ha hibaüzenettel zárul bármelyik fordítás, akkor nincs értelme megismételni, hanem a hibát kell megkeresni. A `.bib` fájl megváltoztatáskor sokszor nincs hatása a változtatásnak azonnal, mivel nem mindig fut újra a BibTeX fordító. Ezért célszerű a változtatás után azt manuálisan is lefuttatni (TeXstudio esetén `Tools/Bibliography`).

Hogy a szövegbe ágyazott hivatkozások kinézetét demonstráljuk, itt most sorban meghivatkozzuk a [36], [3], [17], [15], [29] és a [23][3] forrásmunkát, valamint a [2] weboldalt.

Megjegyzendő, hogy az ékezetes magyar betűket is tartalmazó `.bib` fájl az `inputenc` csomaggal betöltött `latin2` betűkészlet miatt fordítható. Ugyanez a `.bib` fájl hibaüzenettel fordul egy olyan dokumentumban, ami nem tartalmazza a `\usepackage[latin2]{inputenc}` sort. Speciális igény esetén az irodalmi adatbázis általánosabb érvényűvé tehető, ha az ékezetes betűket speciális latex karakterekkel helyettesítjük a `.bib` fájlban, pl. á helyett `\'{a}`-t vagy ő helyett `\H{o}`-t írunk.

[3] Informatikai témában gyakran hivatkozunk cikkeket a Springer LNCS valamely kötetéből, ez a hivatkozás erre mutat egy helyes példát.

Irodalomhivatkozásokat célszerű először olyan szolgáltatásokban keresni, ahol jó minőségű bejegyzések találhatók (pl. ACM Digital Library,[4] DBLP,[5] IEEE Xplore,[6] SpringerLink[7]) és csak ezek után használni kevésbé válogatott forrásokat (pl. Google Scholar[8]). A jó minőségű bejegyzéseket is érdemes megfelelően tisztítani.[9] A sablon angol nyelvű változatában használt `plainnat` beállítás egyik sajátossága, hogy a cikkhez generált hivatkozás a cikk DOI-ját és URL-jét is tartalmazza, ami gyakran duplikátumhoz vezet – érdemes tehát a DOI-kat tartalmazó URL mezőket törölni.

6.6. A dolgozat szerkezete és a forrásfájlok

A diplomatervsablonban a TeX fájlok két alkönyvtárban helyezkednek el. Az `include` könyvtárban azok szerepelnek, amiket tipikusan nem kell szerkesztenünk, ezek a sablon részei (pl. címoldal). A `content` alkönyvtárban pedig a saját munkánkat helyezhetjük el. Itt érdemes az egyes fejezeteket külön TEX állományokba rakni.

A diplomatervsablon (a kari irányelvek szerint) az alábbi fő fejezetekből áll:

1. 1 oldalas *tájékoztató* a szakdolgozat/diplomaterv szerkezetéről (`include/guideline.tex`), ami a végső dolgozatból törlendő,
2. *feladatkiírás* (`include/project.tex`), a dolgozat nyomtatott verzójában ennek a helyére kerül a tanszék által kiadott, a tanszékvezető által aláírt feladatkiírás, a dolgozat elektronikus verziójába pedig a feladatkiírás egyáltalán ne kerüljön bele, azt külön tölti fel a tanszék a diplomaterv-honlapra,
3. *címoldal* (`include/titlepage.tex`),
4. *tartalomjegyzék* (`thesis.tex`),
5. a diplomatervező *nyilatkozat*a az önálló munkáról (`include/declaration.tex`),
6. 1-2 oldalas tartalmi *összefoglaló* magyarul és angolul, illetve elkészíthető még további nyelveken is (`content/abstract.tex`),
7. *bevezetés*: a feladat értelmezése, a tervezés célja, a feladat indokoltsága, a diplomaterv felépítésének rövid összefoglalása (`content/introduction.tex`),
8. sorszámmal ellátott *fejezetek*: a feladatkiírás pontosítása és részletes elemzése, előzmények (irodalomkutatás, hasonló alkotások), az ezekből levonható következtetések, a tervezés részletes leírása, a döntési lehetőségek értékelése és a választott megoldások indoklása, a megtervezett műszaki alkotás értékelése, kritikai elemzése, továbbfejlesztési lehetőségek,
9. esetleges *köszönetnyilvánítás*ok (`content/acknowledgement.tex`),
10. részletes és pontos *irodalomjegyzék* (ez a sablon esetében automatikusan generálódik a `thesis.tex` fájlban elhelyezett `\bibliography` utasítás hatására, a 6.5. szakaszban leírtak szerint),

[4] `https://dl.acm.org/`
[5] `http://dblp.uni-trier.de/`
[6] `http://ieeexplore.ieee.org/`
[7] `https://link.springer.com/`
[8] `http://scholar.google.com/`
[9] `https://github.com/FTSRG/cheat-sheets/wiki/BibTeX-Fixing-entries-from-common-sources`

11. *függelékek* (`content/appendices.tex`).

A sablonban a fejezetek a `thesis.tex` fájlba vannak beillesztve `\include` utasítások segítségével. Lehetőség van arra, hogy csak az éppen szerkesztés alatt álló `.tex` fájlt fordítsuk le, ezzel lerövidítve a fordítási folyamatot. Ezt a lehetőséget az alábbi kódrészlet biztosítja a `thesis.tex` fájlban.

```
\includeonly{
  guideline,%
  project,%
  titlepage,%
  declaration,%
  abstract,%
  introduction,%
  chapter1,%
  chapter2,%
  chapter3,%
  acknowledgement,%
  appendices,%
}
```

Ha az alábbi kódrészletben az egyes sorokat a % szimbólummal kikommentezzük, akkor a megfelelő `.tex` fájl nem fordul le. Az oldalszámok és a tartalomjegyék természetesen csak akkor billennek helyre, ha a teljes dokumentumot lefordítjuk.

6.7. Alapadatok megadása

A diplomaterv alapadatait (cím, szerző, konzulens, konzulens titulusa) a `thesis.tex` fájlban lehet megadni.

6.8. Új fejezet írása

A főfejezetek külön `content` könyvtárban foglalnak helyet. A sablonhoz 3 fejezet készült. További főfejezeteket úgy hozhatunk létre, ha új TEX fájlt készítünk a fejezet számára, és a `thesis.tex` fájlban, a `\include` és `\includeonly` utasítások argumentumába felvesszük az új `.tex` fájl nevét.

6.9. Definíciók, tételek, példák

Definíció 1 (Fluxuskondenzátor térerőssége). Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. ▪

Példa 1. *Példa egy példára. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.*

Tétel 1 (Kovács tétele). Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. ▪

Köszönetnyilvánítás

Ez nem kötelező, akár törölhető is. Ha a szerző szükségét érzi, itt lehet köszönetet nyilvánítani azoknak, akik hozzájárultak munkájukkal ahhoz, hogy a hallgató a szakdolgozatban vagy diplomamunkában leírt feladatokat sikeresen elvégezze. A konzulensnek való köszönetnyilvánítás sem kötelező, a konzulensnek hivatalosan is dolga, hogy a hallgatót konzultálja.

Irodalomjegyzék

[1] Apache: Apache pdfbox® - a java pdf library. *Apache PDFBox®*, 2024. 07. https://pdfbox.apache.org/.

[2] Budapesti Műszaki és Gazdaságtudományi Egyetem Villamosmérnöki és Informatikai Kar: Diplomaterv portál (2011. február 26.). http://diplomaterv.vik.bme.hu/.

[3] James C. Candy: Decimation for sigma delta modulation. *IEEE Trans. on Communications*, 34. évf. (1986. 01) 1. sz., 72–76. p. DOI: 10.1109/TCOM.1986.1096432.

[4] Droid Chef: Flutter vs jetpack compose: The battle of the century. *Ishan Khanna*, 2022. 11. https://blog.droidchef.dev/flutter-vs-jetpack-compose-the-battle-of-the-century/.

[5] Husayn Fakher: Compose state vs stateflow: State management in jetpack compose. *Medium*, 2024. 04. https://medium.com/@husayn.fakher/compose-state-vs-stateflow-state-management-in-jetpack-compose-c99740732023.

[6] JetBrains: Coroutines guide. *Official libraries*, 2022. 02. https://kotlinlang.org/docs/coroutines-guide.html.

[7] JetBrains: The basics of kotlin multiplatform project structure. *Multiplatform Development*, 2024. 09. https://kotlinlang.org/docs/multiplatform-discover-project.html#compilation-to-a-specific-target.

[8] JetBrains: Common viewmodel. *Kotlin Multiplatform Development*, 2024. 10. https://www.jetbrains.com/help/kotlin-multiplatform-dev/compose-viewmodel.html.

[9] JetBrains: Creating a cross-platform mobile application. *Ktor Documentation*, 2024. 04. https://ktor.io/docs/client-create-multiplatform-application.html.

[10] JetBrains: Integrate a database with kotlin, ktor, and exposed. *Ktor Documentation*, 2024. 09. https://ktor.io/docs/server-integrate-database.html.

[11] JetBrains: Navigation and routing. *Kotlin Multiplatform Development*, 2024. 10. https://www.jetbrains.com/help/kotlin-multiplatform-dev/compose-navigation-routing.html.

[12] JetBrains: Popular cross-platform app development frameworks. *Kotlin Multiplatform Documentation*, 2024. 09. https://www.jetbrains.com/help/kotlin-multiplatform-dev/cross-platform-frameworks.html#popular-cross-platform-app-development-frameworks.

[13] JetBrains: Serialization. *Official libraries*, 2024. 09. https://kotlinlang.org/docs/serialization.html#0.

[14] JetBrains: Use fleet for multiplatform development — tutorial. *Kotlin Multiplatform Development*, 2024. 10. https://www.jetbrains.com/help/kotlin-multiplatform-dev/fleet.html#prepare-your-development-environment.

[15] Peter Kiss: Adaptive digital compensation of analog circuit imperfections for cascaded delta-sigma analog-to-digital converters, 2000. 04.

[16] Markus Kohler: 7 alternatives to rest apis. *PubNub*, 2024. 01. https://www.pubnub.com/blog/7-alternatives-to-rest-apis/.

[17] Wai L. Lee – Charles G. Sodini: A topology for higher order interpolative coders. In *Proc. of the IEEE International Symposium on Circuits and Systems* (konferenciaanyag). 1987. 4-7 05., 459–462. p.

[18] Google LLC: Camerax overview. *Android Developers Guide*, 2024. 01. https://developer.android.com/media/camera/camerax.

[19] Google LLC: Jetpack compose basics. *Android codelabs*, 2024. 01. https://developer.android.com/codelabs/jetpack-compose-basics#0.

[20] Google LLC: Recognize text in images with ml kit on android. *ML-Kit Guides*, 2024. 10. https://developers.google.com/ml-kit/vision/text-recognition/v2/android.

[21] Google LLC: Viewmodel overview. *Android Developers Guide*, 2024. 07. https://developer.android.com/topic/libraries/architecture/viewmodel.

[22] Microsoft: .net multi-platform app ui. *.NET MAUI*, 2024. 11. https://dotnet.microsoft.com/en-us/apps/maui.

[23] Alexey Mkrtychev: Models for the logic of proofs. In Sergei Adian – Anil Nerode (szerk.): *Logical Foundations of Computer Science*. Lecture Notes in Computer Science sorozat, 1234. köt. 1997, Springer Berlin Heidelberg, 266–275. p. ISBN 978-3-540-63045-6. URL http://dx.doi.org/10.1007/3-540-63045-7_27.

[24] Elvira Mustafina: Compose multiplatform 1.7.0 released. *JetBrains Blog*, 2024. 10. https://blog.jetbrains.com/kotlin/2024/10/compose-multiplatform-1-7-0-released/.

[25] Lukoh Nam: Jetpack compose permissions using accompanist library, modalbottomsheet. *Medium*, 2023. 05. https://medium.com/@lukohnam/jetpack-compose-permissions-using-accompanist-library-b1c0fbbe8831.

[26] Lazar Nikolov: Getting started with jetpack compose. *Sentry Blog*, 2023. 02. https://blog.sentry.io/getting-started-with-jetpack-compose/.

[27] Ivan Osipov: Kotlin dsl: from theory to practice. *JMIX*, 2017. 10. https://www.jmix.io/cuba-blog/kotlin-dsl-from-theory-to-practice.

[28] Ekaterina Petrova: Kotlin multiplatform goes stable. *Kotlin Blog*, 2023. 11. https://blog.jetbrains.com/kotlin/2023/11/kotlin-multiplatform-stable/.

[29] Richard Schreier: *The Delta-Sigma Toolbox v5.2*. Oregon State University, 2000. 01. http://www.mathworks.com/matlabcentral/fileexchange/.

[30] Docker Team: What is docker? *Dockerdocs*, 2024. 11. https://docs.docker.com/get-started/docker-overview/.

[31] Flutter Team: Flutter. *Flutter*, 2024. 11. https://flutter.dev/.

[32] Kotlin Documentation Team: Update to the new release. *Kotlin Multiplatform Plugin Releases*, 2024. 09. https://kotlinlang.org/docs/multiplatform-plugin-releases.html#update-to-the-new-release.

[33] React Native Team: React native. *React Native*, 2024. 11. https://reactnative.dev/.

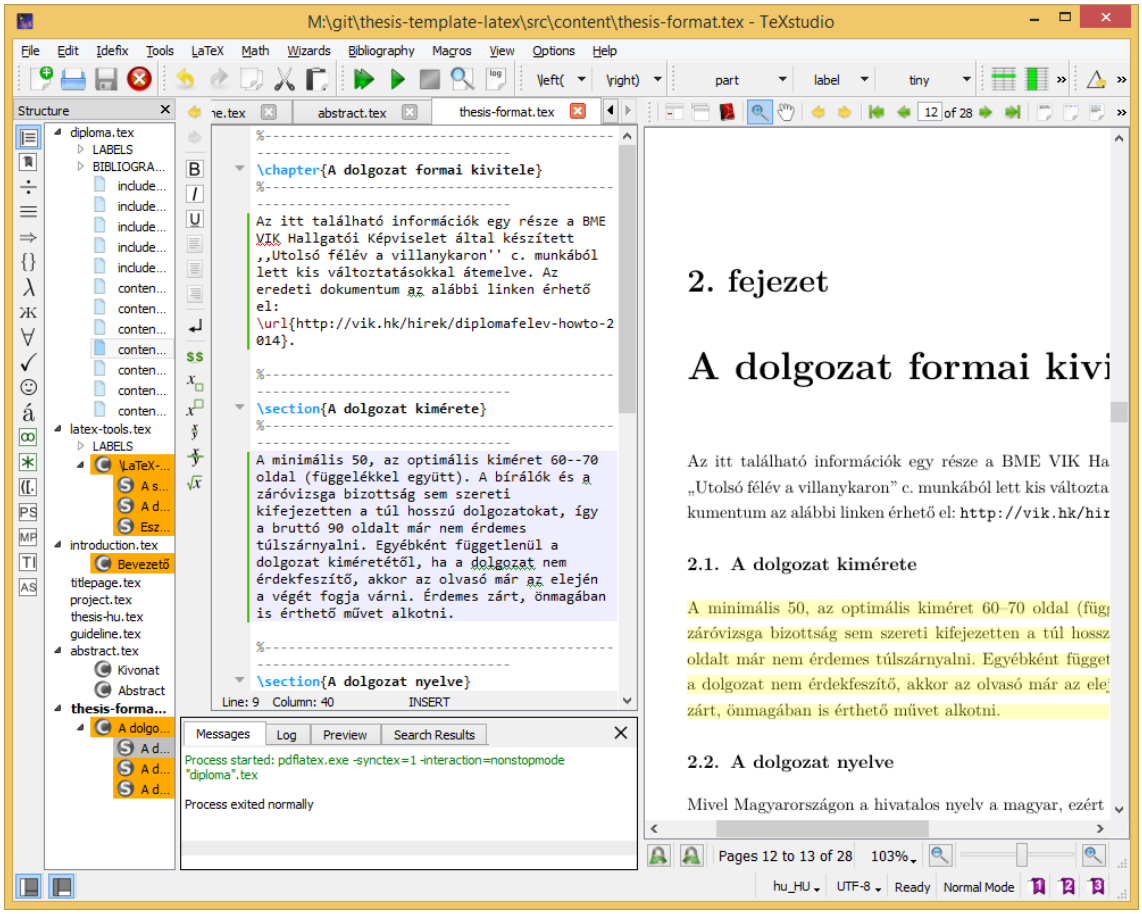
[34] Unknown: What is component diagram? *Visual Paradigm*, 2024. 11. https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/.

[35] Antoine van der Lee: Mvvm: An architectural coding pattern to structure swiftui views. *SwiftLee*, 2024. 05. https://www.avanderlee.com/swiftui/mvvm-architectural-coding-pattern-to-structure-views/.

[36] Ferenc Wettl – Gyula Mayer – Péter Szabó: *LaTeX kézikönyv*. 2004, Panem Könyvkiadó.

Függelék

F.1. A TeXstudio felülete



F.1.1. ábra. A TeXstudio LaTeX-szerkesztő.

F.2. Válasz az „Élet, a világmindenség, meg minden” kérdésére

A Pitagorasz-tételből levezetve

$$c^2 = a^2 + b^2 = 42. \tag{F.2.1}$$

A Faraday-indukciós törvényből levezetve

$$\operatorname{rot} E = -\frac{dB}{dt} \qquad \longrightarrow \qquad U_i = \oint_{\mathbf{L}} \mathbf{Edl} = -\frac{d}{dt} \int_A \mathbf{Bda} = 42. \tag{F.2.2}$$