ETDK-dolgozat

Mátis Krisztián

Szilágyi Krisztián-Attila

XXVI. reál- és humántudományi Erdélyi Tudományos Diákköri Konferencia (ETDK)

Informatika II.: innovatív számítástechnikai termékek, alkalmazások szekció

Kolozsvár, 2023. május 18–21.

Prophet

Hangfelismerés alapú automatikus teleprompter alkalmazás

Szerzők:

Mátis Krisztián

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatika szak, III. év

Szilágyi Krisztián-Attila

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatika szak, III. év

Témavezetők:

dr. Sulyok Csaba, egyetemi adjunktus

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar

Hegyi Boglárka, szoftverfejlesztő,

Codespring

Kecseti István, szoftverfejlesztő,

Codespring

Szakács Tas, szoftverfejlesztő,

Codespring

CODESPRING

OUR CODE YOUR SUCCESS

Kivonat

A Prophet projekt célja egy olyan alkalmazás létrehozása, mely segítséget nyújt az előadók számára a bemutatóik gyakorlásában, illetve megtartásában egy hangfelismerés alapú automatizált súgógép segítségével. A beszédszövegek papíron való karbantartása bonyodalmakhoz vezethet, a már létező teleprompteres megoldások állandó sebességgel rendelkeznek, ezért az előadó könnyen belekavarodhat, elfelejtheti, hogy mégis hol tart az előadásában. A projekt célja ezen problémák kiküszöbölése digitalizálás és automatizálás segítségével.

A projekt két fő részből áll: egy mobilalkalmazásból és egy dokumentumokat importáló szolgáltatásból. A mobilalkalmazás egyéni felhasználók igényei köré épül. A felhasználónak lehetősége van új beszédszöveg létrehozására, vagy már létező dokumentumok importálására, amely tovább szerkeszthető egy beépített rich-szövegszerkesztő segítségével. Az így kapott szövegre a felhasználó alkalmazhatja az automatizált teleprompter funkciót, mely hangfelismerés segítségével jelzi a beszéd előrehaladtát.

A dokumentumok importálása egy saját webes szolgáltatás alkalmazásával történik, amelynek szerepe a különböző formátumú szöveges dokumentumok tartalmának kinyerése, és egységes formába alakítása.

XXVI. ERDÉLYI TUDOMÁNYOS DIÁKKÖRI KONFERENCIA
CONFERINŢA ŞTIINŢIFICĂ STUDENŢEASCĂ DIN TRANSILVANIA (CŞST), EDIŢIA A XXVI-A
26TH TRANSYLVANIAN STUDENTS' SCIENTIFIC CONFERENCE (TSSC)
reál- és humántudományok • ştiinţe reale şi umane • real and human sciences
KOLOZSVÁR, 2023. MÁJUS 18–21.

Kolozsvári Magyar Diákszövetség • Uniunea Studenţească Maghiară din Cluj
400083 Kolozsvár, Petőfi Sándor/Avram Iancu utca 21 • tel./fax: 0755116251
e-mail: *etdk@kmdsz.ro, etdk.kolozsvar@gmail.com* • honlap/web: *https://etdk.kmdsz.ro*

Témavezetői igazolás

Alulírott *dr. Sulyok Csaba, Hegyi Boglárka, Kecseti István és Szakács Tas* igazoljuk, hogy *Mátis Krisztián és Szilágyi Krisztián-Attila* hallgatók a(z) *Prophet: Hangfelismerés alapú automatikus teleprompter alkalmazás* című dolgozatot az alulírottak szakmai irányításával készítették el, és javasoljuk az említett tudományos munka bemutatását a XXVI. Reál- és Humántudományi Erdélyi Tudományos Diákköri Konferencia (ETDK) *Informatika II.: innovatív számítástechnikai termékek, alkalmazások* szekciójában.

dr. Sulyok Csaba, egyetemi adjunktus
Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar

Hegyi Boglárka, szoftverfejlesztő,
Codespring

Kecseti István, szoftverfejlesztő,
Codespring

Szakács Tas, szoftverfejlesztő,
Codespring

Kolozsvár, 2023. március 31.

Tartalomjegyzék

Bevezető

Minden ember életében eljön egyszer az a pont, amikor valamilyen formában beszédet kell tartania. Legyen szó versmondásról, oktatási vagy üzleti bemutatóról vagy akár prédikációról, a tiszta, érthető beszéd kulcsfontosságú ahhoz, hogy megértessük gondolatainkat a közönséggel. Nagyon fontos egy előadás minőségének fenntartása, hiszen egy kevésbé professzionális előadás nem csak a mondanivalót teszi jelentéktelenné, hanem magát az előadót is jellemzi, kellemetlen helyzetbe hozhatja.

A beszédek esetében nem csak az előadás okozhat problémákat, hanem a szövegek karbantartása is. A legtöbb előadó papírra írt jegyzetekre, esetleg a teljes szöveg nyomtatott másolatára hagyatkozik előadás során, hogy elkerülje a kellemetlen fennakadásokat. Ez azonban egy elavult, környezetszennyező, bonyodalmas módszer. Bármilyen apró javítás vagy módosítás áthúzásokkal, firkálással jár, és mindezek a tényezők csökkentik a szöveg átláthatóságát. Ez hosszú szünetekhez, kétségbeesett dadogáshoz vezethet, ha gyors útmutatásra lenne szükségünk, de nem kapjuk meg azt. [33, 39]

A papír viszont nem minden. A beszéd egy komplex folyamat, a gondolkodás, nyelvi tervezés és a szavak kiejtése egyidejű folyamatok, ezért egy könnyen olvasható jegyzet használatakor is előfordulhatnak megakadások a beszédben, melyek kellemetlen csendszünetekhez vezetnek. [36]

Itt jön szóba a súgógép, vagy teleprompter használata. A jelenlegi súgógép megoldásoknak megvannak a saját hátrányaik. Az emberi beszéd nem állandó tempójú, ezért az állandó sebességgel gördülő teleprompter alkalmazások, mint például a SpeechWay [25], a Parrot Teleprompter [32], vagy a Teleprompter & Video Captions [35] nem tudnak kellő segítséget nyújtani, legtöbbször rossz sorokat helyeznek előtérbe, emiatt kizökkenthetik az előadót a beszédéből.[1, 24] A színházakban alkalmazott manuális teleprompterek egy külső személy beavatkozását igénylik, emiatt ez a módszer az egyéni előadók számára nem alkalmas, hisz több munkával, egyeztetéssel, közös gyakorlással jár.

A Prophet alkalmazás ezen problémákat orvosolja a beszédszövegek karbantartásának digitalizálásával, illetve egy automatizált teleprompter segítségével.

A dolgozat 5 különböző fő fejezetre bontható. Az 1. fejezet felvázolja az alkalmazásban létező szerepköröket, és az általuk elérhető funkcionalitásokat. A 2. fejezet bemutatja az alkalmazás legfőbb funkcionalitásait, és részletezi ezek elméleti háttereit. A 3. fejezet ismerteti az alkalmazott szerveroldali és mobil technológiákat, illetve a fejlesztéshez felhasznált

eszközöket. A 4. fejezetben részletes leírás található az alkalmazás és az átalakító szolgáltatás működéséről. Az 5. és egyben utolsó fejezet végső következtetéseket von le a dolgozatból, illetve bemutat pár továbbfejlesztési lehetőséget, amelyekkel kibővíthető az alkalmazás.

A projekt fejlesztése 2022 júliusában, a Codespring cégnél végzett nyári szakmai gyakorlat[1] alatt kezdődött el, ahol bár egy kezdetleges, de már használható állapotba került. Ezen túl a 2022-2023-as egyetemi tanév első félévében, a csoportos projekt tantárgy keretein belül[2] a projekt további fejlesztéseken ment át.

Köszönet illeti a témavezető tanárunkat, Dr. Sulyok Csaba egyetemi adjunktust, illetve Hegyi Boglárka, Kecseti István, valamint Szakács Tas szoftverfejlesztőket a mentorálásért a Codespring részéről. Köszönet illeti a Codespringet is az alkalmazás számára biztosított Kubernetes klaszterért. Továbbá köszönet illeti Szilágyi-Pap Dávidot, Tasnádi Róbertet, Topor Dánielt és Zediu Álmos-Ágostont, akik a csoportos projekt tantárgy által biztosított lehetőséget megragadva, ideiglenesen csatlakoztak a fejlesztői csapathoz, és így hozzájárultak az alkalmazás megvalósításához.

[1] A cég mentorprogramja
[2] A tantárgy adatlapja

1. A Prophet projekt

A Prophet alkalmazás fő célja a beszédek előadásának megkönnyítése egy automatizált súgógép segítségével, illetve a beszédszövegek tárolásának egységesítése, digitalizálása.

Jelen fejezet bemutatja az alkalmazásban létrehozható felhasználói típusokat, és a funkcionalitásokat, amelyeket ezek igénybe vehetnek.

1.1. Funkcionalitások

Az alkalmazás kétfajta felhasználói fiók létrehozására ad lehetőséget: vendégfelhasználó és regisztrált felhasználó. A regisztrált felhasználók a regisztrálási módszertől függően további két kategóriába sorolhatóak: hagyományos, e-mail címmel és jelszóval regisztrált felhasználók, és Google-lel regisztrált felhasználók.

A legtöbb funkcionalitás elérhető bármely típusú felhasználó számára. Az 1. ábra szemlélteti a különböző szerepkörű felhasználók számára elérhető funkcionalitásokat.

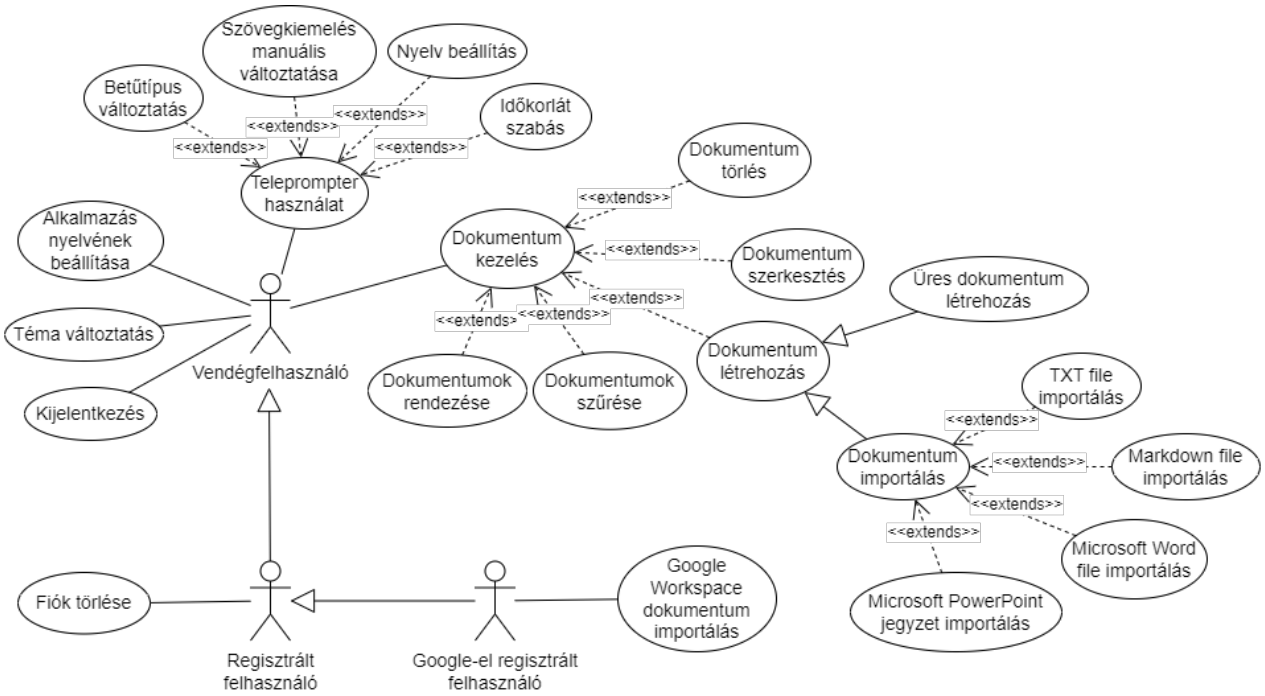
Az alkalmazás használatához a felhasználónak először be kell jelentkeznie felhasználói fiókjába, vagy szükség esetén létre kell hoznia egyet egy keresztnév, e-mail cím és jelszó megadásával, vagy pedig Google autentikációt használva. Ezen kívül, ha a felhasználó nem akar semmilyen jellegű adatot megadni magáról, bejelentkezhet vendégfelhasználóként is.

Bejelentkezés után elérhetővé válik az adott felhasználó összes dokumentuma. Ezeket rendezni lehet cím szerint ábécé sorrendbe, vagy az utolsó szerkesztés időpontja szerint csökkenő sorrendbe, továbbá szűrni lehet őket cím szerint, és természetesen törölni is lehet egyszerre többet is. Létre lehet hozni üres dokumentumot, vagy be lehet importálni bizonyos formátumú fájlokat a natív eszközböngészőn keresztül. Google-lel bejelentkezett felhasználók esetében lehetőség van a Google Drive [19] tárolójukon levő dokumentumok importálására is.

A felhasználó továbbá megnyithatja a beállítások menüjét, ahol meg lehet változtatni az alkalmazás témáját, a teleprompterben használt betűtípust, az alkalmazás nyelvét, illetve lehetőség van kijelentkezésre, és regisztrált felhasználó esetében a felhasználói fiók törlésére.

Egy dokumentum megnyitásakor újabb, dokumentum-specifikus beállítások válnak elérhetővé: törölni lehet az aktuális dokumentumot, oldalakat lehet beszúrni és törölni, illetve be lehet állítani a szöveg nyelvét, és a beszédre szánt időkorlátot. Ez utóbbi két beállítás a telepromptert érinti.

Megnyitva a telepromptert, a felhasználó elindíthatja a hangfelismerést, valamint szükség



1. ábra. Különböző felhasználók számára elérhető funkcionalitások

esetén gyorsan módosíthatja a szöveg méretét, a szövegkiemelés pozícióját, és a megjelenített oldalt. Ezen kívül visszaállíthatja a szövegkiemelést kezdetleges állapotba, és időkorlát használata esetén megváltoztathatja az időzítő megjelenését hátralevő idő mutatásáról az eltelt időre és fordítva.

Ezen funkcionalitások részletes leírása megtalálható az alkalmazás működéséről szóló, 4. fejezetben.

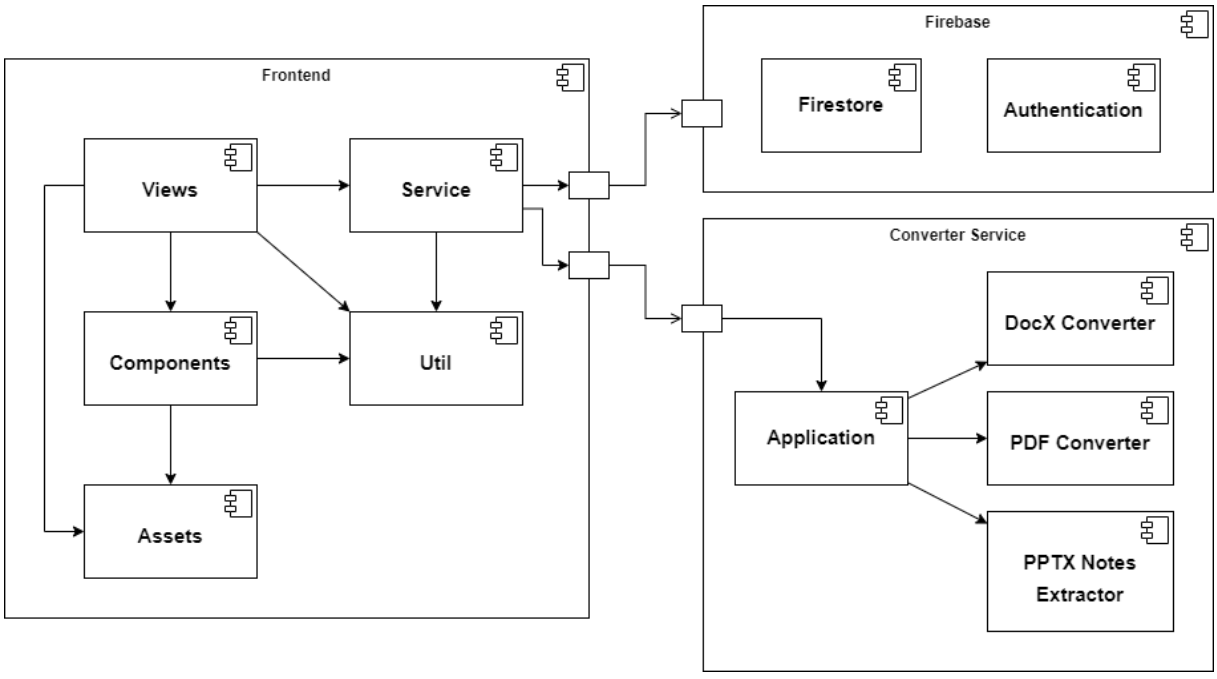
2. Az alkalmazás felépítése

A Prophet projekt két fő részből áll: magából a Prophet mobilalkalmazásból, és a dokumentumok importálását lehetővé tevő átalakító szolgáltatásból. Az alábbi alfejezetekben ezek részletes bemutatására kerül sor.

2.1. Az alkalmazás struktúrája

A mobilalkalmazás React Native [13] keretrendszerben van megírva, mivel egy gyors fejlesztési lehetőséget kínál cross-platform mobilalkalmazások fejlesztésére egyetlen kódbázissal. Emellett a React Native nagy fejlesztői közössége és a rengeteg közösségi csomag nagyban hozzájárultak a fejlesztési folyamathoz.

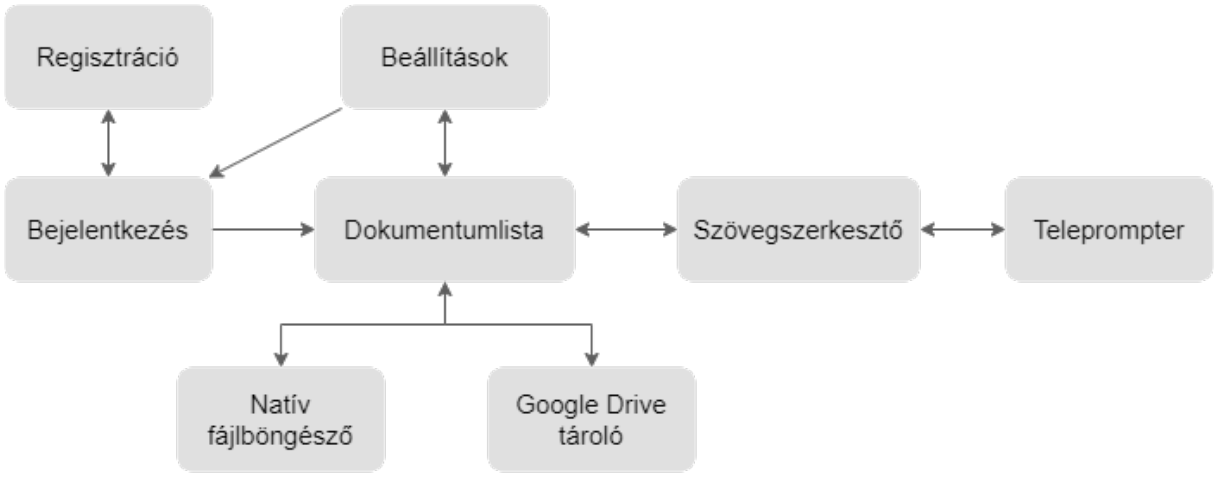
React Native-ben a Reacthez hasonlóan a felhasználói felület kisebb, egymástól független részekből tevődik össze. Az alkalmazás saját komponensekkel is rendelkezik, melyek az újrahasznosíthatóság és átláthatóság érdekében külön modulokra vannak választva. Öt modul különböztethető meg: a kisebb, többször felhasználható komponensek a *components* modulban helyezkednek el. Ezeket felhasználva épülnek fel az alkalmazás oldalai, melyek a *views* réteg elemeit képezik. Ez a réteg továbbá igénybe veszi a *util* csomagban deklarált stílus konstansokat, az *assets* csomagban tárolt képeket és ikonokat, az adatok feldolgozására pedig a *service* réteg függvényeit. A modulok közötti kapcsolatok a 2. ábrán vannak szemléltetve.



2. ábra. Az alkalmazás sturtkúrája, komponens diagram.

2.2. Navigáció

Az alkalmazáson belül a navigáció a React Navigation [30] közösségi csomag segítségével valósul meg. Az applikáció teljes kódja egy platformspecifikus integrációt megvalósító komponensbe van burkolva, amely a benne található oldalak kezeléséért felelős. Navigálás során az oldalak egy veremben vannak karbantartva, amely megkönnyíti a visszafele navigálást. Egy bizonyos oldal a `navigation.navigate('route')` függvény meghívásával érhető el, előző oldalra a `navigation.goBack()` függvény segítségével lehet visszatérni, a `navigation.popToTop()` függvény pedig kiüríti a vermet, és az alkalmazás kezdeti oldalára vezet. A 3. ábra szemlélteti az alkalmazáson belüli navigációs folyamot.



3. ábra. Alkalmazáson belüli navigáció.

2.3. Autentikáció

Ahogyan az 1.1 szekcióban is említve volt, a bejelentkezési felületen a felhasználó több féle módon tud bejelentkezni, azaz e-mail címmel és jelszóval, vendégfelhasználóként vagy Google fiókjával való autentikációval.

Az e-mail cím és jelszó mezők kitöltése és a bejelentkezés gomb megnyomása után a bejelentkezési adatokat az alkalmazás elküldi a 3.1. fejezetben tárgyalt autentikációs szolgáltatásának. A szolgáltatás leellenőrzi, hogy létezik-e a megadott adatokkal regisztrált fiók. Ha létezik ilyen fiók, és megfelelő a jelszó, válaszként visszaküldi a felhasználói fiókot azonosító adatokat. Ezután a kapott azonosító adatokkal az alkalmazás egy másik kérést intéz a 3.1.1. fejezetben tárgyalt adatbázisnak, melyben lekérdezi a fiókról eltárolt információkat, mint például az alkalmazásban megjelenítendő felhasználói nevet, eltárolja egy biztonságos

perzisztens tárolóba az eszközön az alkalmazásban való további használatra, majd átirányítja a felhasználót az alkalmazás főoldalára. Ellenkező esetben, ha a szolgáltatás nem talált a megadott adatoknak megfelelő regisztrált fiókot, egy hibát küld vissza, ami alapján az alkalmazás jelzi a felhasználó számára, hogy hibás adatokat adott meg, és így nem engedi tovább a következő oldalra.

Google fiókkal való bejelentkezés esetén, az e-mail cím és jelszóval való bejelentkezéshez hasonlóan, a Google-lel való bejelentkezés gombjának megnyomása után az alkalmazás ez esetben is egy kérést küld az autentikációs szolgáltatásának, viszont itt már a Google által biztosított bejelentkezési ablak jelenik meg a felhasználó számára, amely egyben regisztrációként is működik még nem létező felhasználói fiók esetén. Ebben az ablakban a felhasználó kiválaszthat egyet az eszközén már bejelentkezett Google fiókok közül, vagy bejelentkezhet egy teljesen új Google fiókkal is. Ezután egy felhívás jelenik meg a bejelentkezési ablakban, mely felvilágosítja arról, hogy az alkalmazás hozzáférhet a Google Drive tárolóján lévő fájlokhoz a dokumentumok importálása miatt. Engedély adás, vagy megtagadás után a bejelentkezési ablak bezárul, majd az autentikációs szolgáltatás elküldi a fiókot azonosító adatokat. Ezeket, az e-mail és jelszóval való bejelentkezéshez hasonlóan, az alkalmazás egy biztonságos perzisztens tárolóba tárolja az eszközön, majd átirányítja a felhasználót az alkalmazás főoldalára. Ha a bejelentkezés folyamatában akárhol történik valamiféle hiba, a bejelentkezési ablak a hiba típusától függően vagy egyenesen a felhasználónak jelez, vagy az alkalmazásnak küld egy hibakódot, ami alapján pedig az egy megfelelő hibaüzenetet jelenít meg a felhasználó számára.

Vendégfelhasználói fiókkal való bejelentkezés esetén az autentikációs szolgáltatás létrehoz egy anonim vendég felhasználói fiókot, és visszaküldi annak azonosító adatait az alkalmazásnak. Ezután az alkalmazás, a többi bejelentkezési móddal való kompatibilitás érdekében, újabb kérések segítségével eltárolja ezt az újonnan létrehozott anonim fiókot az adatbázisba és az eszközön található, biztonságos perzisztens tárolóba, majd átirányítja a felhasználót az alkalmazás következő oldalára, korlátozott hozzáféréssel. Viszont a többi bejelentkezési móddal ellenkezőleg az adatok csak addig maradnak meg, amíg a felhasználó ki nem jelentkezik az alkalmazásból. Kijelentkezés esetén törlődik minden adat a fiókról, bele értve az összes létrehozott dokumentumot, ami az eszközön vagy az adatbázisban létezik.

2.4. Dokumentumok importálása

A felhasználónak lehetősége van teljesen üres dokumentum létrehozására, vagy importálhat különböző forrásból származó, különböző formátumban tárolt szöveges dokumentumokat.

Ha a felhasználó egy új szöveges dokumentumot szeretne létrehozni, akkor az alkalmazás először felépít egy szöveges dokumentum objektumot előre definiált kezdeti értékekből, hozzá rendeli a felhasználó azonosítóját, majd ezt a szöveges dokumentum objektumot feltölti az adatbázisba. Lekérdezi a létrejött dokumentum azonosítóját, majd ennek segítségével átirányítja a felhasználót az alkalmazás szövegszerkesztő oldalára, ahol a kapott dokumentum azonosító alapján megjelenik az újonnan létrehozott, üres szöveges dokumentum.

Már létező szöveges dokumentumok importálása esetén a felhasználónak el kell döntenie, hogy milyen forrásból, honnan szeretné azt importálni. Ebben az esetben az alkalmazás igénybe veszi a React Native által biztosított *DocumentPicker* komponenst, mely az adott eszköz beépített natív fájlkezelő rendszerének segítségével lehetőséget ad a felhasználó számára, hogy az eszközén található fájlok között böngészhessen, így kiválasztva, hogy melyiket szeretné importálni. Internetkapcsolat hiányában csak Plaintext és Markdown [11] típusú fájlok importálhatóak, minden más opció letiltásra kerül a fájlböngészőben. A fájlböngészőben kiválasztott fájl tartalma bekerül az alkalmazásba, ahol az üres dokumentum létrehozásához hasonlóan létrejön egy új dokumentum, viszont nem az előre definiált, hanem a kapott adatok segítségével.

Internetkapcsolat jelenlétében Plaintext és Markdown fájlok mellett az alkalmazás engedélyt ad *Portable Document Format (PDF)*, *Microsoft Word* vagy *Microsoft PowerPoint* típusú fájlok importálására is. Ez utóbbi három fájltípus esetén az átalakító szolgáltatás igénybevétele miatt van szükség internetkapcsolatra. A szolgáltatásra azért van szükség, mert a tartalom kinyeréséhez felhasznált csomagok nem működnek mobil eszközökön.

Az átalakító és az alkalmazás közötti kommunikáció HTTP [17] hívásokkal történik a Fetch API [15] segítségével. Az alkalmazás egy POST kérés testében elküldi az átalakítandó fájlt az átalakító címére[3], majd az a kapott fájl kiterjesztése és MIME [18] típusa alapján eldönti milyen módszerekkel nyeri ki a kívánt adatokat. Minden esetben egy harmadik féltől származó, közösségi npm csomagot alkalmaz. A kinyert adat végül vagy HTML, vagy JSON fájlként térül vissza az alkalmazás számára, a feltöltött fájl típusának függvényében.

PDF fájlok esetén egy *pdf2html*-nek [34] nevezett csomagot használ. Ez a csomag az

[3] Átalakító szolgáltatás címe: http://prophet-backend.k8s.edu.codespring.ro/convert

Apache Tika [2] eszköztár segítségével ki tudja nyerni a PDF fájlok tartalmát, majd ezt egy egyszerű szöveg vagy HTML kód formájába helyezni, ezáltal megtartva a tartalom szerkezetét és kinézetét. Viszont ahhoz, hogy a csomag működni tudjon, szükséges a JRE [9] jelenléte is.

Microsoft PowerPoint fájlok esetén a *pptx2json* [38] csomag segítségével az átalakító kinyeri a fájl teljes tartalmát JSON formátumba. Az alkalmazás számára viszont csak a diákon megjelenő jegyzet szükséges, ezért ki kell szűrni a felesleges tartalmat. Mivel a PPTX fájlok rendkívül sok információt tartalmaznak, szükség van egy olyan függvényre, amely a megfelelő kulcsok alapján kiválasztja a jegyzeteket, majd felépít belőlük egy listát olyan sorrendben, ahogyan azokat találta, végül pedig a listát vissza téríti az átalakító számára, ez látható az 1. kódrészletben is. Viszont ezekben a fájlokban nem feltétlenül lineárisan tárolódnak az adatok, sokkal inkább referenciák által, így megtörténhet, hogy átalakítás során a jegyzetek listájának a sorrendje nem követi a PPTX fájlban lévő diák sorrendjét.

A Microsoft Word dokumentumok tartalmának kinyerése a *Mammoth* [29] csomag segítségével történik. A csomag képes Microsoft Word, Google Docs vagy akár LibreOffice dokumentumokat is felismerni, és ezek tartalmát HTML fájlokba menteni. Mivel a Microsoft Word fájlok és a HTML fájlok struktúrája rendkívül különböző, ezért nem támogatott minden dokumentum komponens. Viszont ameddig a dokumentum nem túlságosan komplex, a csomag képes az átalakításra.

Az átalakító szolgáltatásban csak egyetlen egy POST típusú HTTP hívás lekezelése történik meg, a „`/convert`" címre elhelyezett routerrel. A szolgáltatás inicializálásakor létrejön a feltöltött fájlok ideiglenes tárolására szükséges mappa, illetve a naplózott üzeneteket tartalmazó fájl. Ha már létezik a napló fájl, vagy a feltöltési mappa, akkor ezek előbb törlésre kerülnek. A POST kérést kezelő metódus először leellenőrzi, hogy valóban érkezett fájl a kérésben. Amennyiben nem, egy 400-as státusz kódot vissza térítve jelzi az alkalmazás számára, hogy hibás a kérés és leáll a feldolgozással. Ellenőrzés után megnézi, hogy milyen formátumú a fájl, majd ez alapján meghívja a megfelelő átalakító csomagot. Sikeres átalakítás esetén a választ JSON formában téríti vissza az alkalmazás számára. Hiba esetén 500-as státusz kóddal jelzi az alkalmazásnak, hogy nem sikerült az átalakítás. Ha esetleg olyan fájl kerül feltöltésre, amelynek formátumát a szolgáltatás nem támogatja, azt egy 400-as hiba kóddal jelzi az alkalmazás számára. Legutolsó lépésként törli a feltöltött fájlt, legfeljebb a naplózott információ kerül tárolásra esetleges hibák javítása miatt.

A kapott válaszban lévő, átalakított fájl tartalma bekerül az alkalmazásba, akárcsak Plaintext

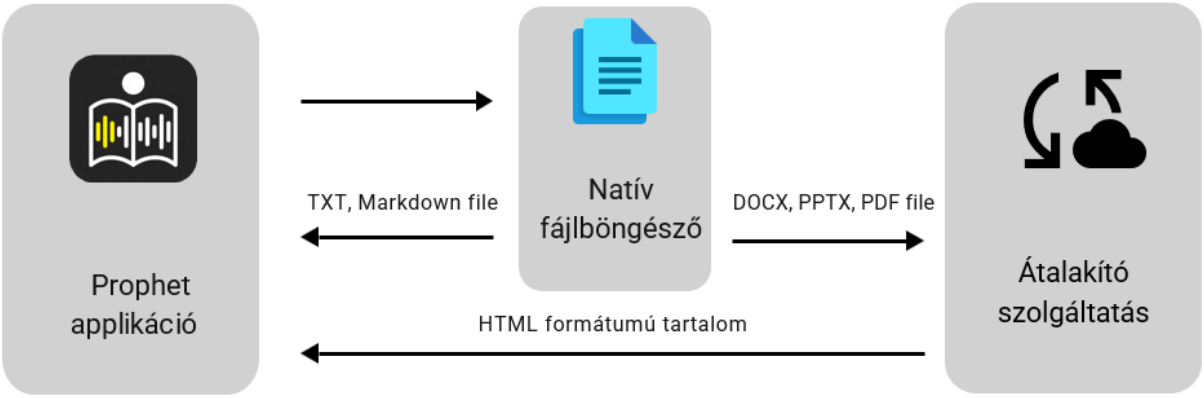
és Markdown fájlok esetében, ahol hasonló módon a kapok adatok alapján létrejön egy új dokumentum.

Ha a felhasználó Google fiókkal jelentkezett be, az adott fiók Google Drive tárolójában lévő dokumentumokat is képes importálni. Ebben az esetben egy új felugró oldal jelenik meg a főoldalon a felhasználó számára, amely tartalmaz minden olyan Google Docs fájlt, amelyet az alkalmazás talált az adott Google fiók Google Drive tárolóján. Miután a felhasználó kiválasztotta az importálandó fájlt, az alkalmazás a már említett módon a kapott adatok alapján létrehoz egy új dokumentumot, majd átirányítja a felhasználót a szövegszerkesztő oldalra,

```
export default function getPPTXNotes(document) {
  const keys = Object.keys(document);

  const filtered = keys.filter((key) =>
    key.includes('ppt/notesSlides/notesSlide'));
  const finalResult = [];

  filtered.forEach((key) => {
    let result = '';
    const texts = [];

    const body =
      document[key]['p:notes']['p:cSld'][0]['p:spTree'][0]['p:sp'][1]['p:txBody'];
    body.forEach((text) =>
      text['a:p']?.forEach((rw) =>
        rw['a:r']?.forEach((txt) => texts.push(txt['a:t'][0]))));
    result += texts[0];

    for (let index = 1; index < texts.length; index++) {
      const p1 = texts[index - 1];
      const p2 = texts[index];

      if (p1 === p1.trim() && p2 === p2.trim()) {
        result += ' ';
      }
      result += p2;
    }

    finalResult.push(result);
  });

  return finalResult;
}
```

1. kódrészlet. PPTX fájlok jegyzeteinek kinyeréséért felelős függvény

megjelenítve az importált fájl tartalmát.



4. ábra. Dokumentumok importálása.

2.5. Szövegszerkesztő

Az újonnan létrehozott, vagy külső forrásokból importált dokumentumokat a felhasználó egy beépített rich text szövegszerkesztő segítségével szerkesztheti. A rich text szerkesztő egy olyan eszköz, amely lehetővé teszi a szöveg félkövér, dőlt, aláhúzott és áthúzott karakterekkel, számozott és felsorolásjeles listákkal való gazdagítását, amely jobb átláthatóságot eredményez, és ezáltal egy fokozott felhasználói élményhez vezet.

Az alkalmazás a React Native Rich Text Editor [37] nevű csomagot felhasználva valósítja meg ezt a funkcionalitást, amely két fő komponenst nyújt: egy szövegbemenet komponenst, és egy eszközsáv komponenst. A szövegbemenet komponens valójában egy leegyszerűsített HTML megtekintő felület, ezért függőségként használja a React Native WebView [14] közösségi csomagot. Ez a komponens egy HTML formátumú szöveget tart fent a háttérben, és az imént felsorolt formázási opciókat HTML [16] tagek valós idejű beszúrásával valósítja meg, majd a kiegészített szöveget megjeleníti a webnézet komponensen keresztül. A másik fő komponens az eszközsáv, amely a szerkesztési opciókat tartalmazza. Egy ilyen opció kiválasztásakor a szövegszerkesztő kiegészíti a háttérben levő HTML formátumú szöveget a megfelelő HTML tagekkel. Ahhoz, hogy az eszközsáv kommunikálni tudjon a hozzá tartozó szerkesztővel, az eszközsávnak szüksége van egy, és csakis egy szövegszerkesztő komponens referenciájára.

Az alkalmazás azonban lehetőséget ad többoldalú dokumentumok létrehozására és szerkesztésére, és ezt több szövegszerkesztő egyidejű működtetésével teszi lehetővé. Ezek a szerkesztő komponensek egy vízszintesen görgethető nézetben foglalnak helyet egymás mellett,

míg az egyedüli eszközsáv az aktuálisan fókuszban levő szerkesztőtől függetlenül lebeg a kijelző alján. A szövegszerkesztő nézet megnyitásakor minden oldal számára létrejön egy külön szerkesztő komponens, amely az adott oldal szövegét tartalmazza. Ezzel egyidejűleg egy referencia tömb is inicializálódik, amely ezen szerkesztő komponensek refereciáit tartalmazza. Amikor a felhasználó egy adott oldalra görget, vagyis amikor a megfelelő szerkesztő komponens kerül fókuszba, az eszközsáv komponens szerkesztő referenciája lecserélődik, így egyetlen eszközsáv képes az összes szerkesztő komponenssel kommunikálni a szükséges időben. Egy új oldal beszúrásakor vagy egy létező oldal törlésekor megfelelően módosítjuk a referencia tömb tartalmát, és egy állapotváltozó módosításával újra rajzoljuk a kijelző tartalmát, amely az elvárt oldalak megjelenítését eredményezi.

2.6. Teleprompter

Az alkalmazás fő funkcionalitása a hangfelismerés alapú teleprompter, melynek lényege, hogy segítse az előadót a beszédszövege követésében. A funkcionalitás két fő folyamatra épül: a hangfelismerésre, és a felismert szavak követésére a beadott szövegben.

A hangfelismerést a React Native Voice [27] csomag teszi lehetővé. Ez a csomag elérést biztosít a mobil készülékeken levő natív hangfelismerési szolgáltatásokhoz. A csomag egyaránt támogatja az Android és iOS operációs rendszereket, azonban fejlesztés során a hangsúly az Android kompatibilitásra esett, így az alkalmazás a `googlequicksearchbox` [23] hangfelismerő motort használja hangfelismerésre.

Ez a megoldás számos előnnyel és néhány hátránnyal is jár. Egyik nagy előny, hogy a legtöbb Androidos készülék alapból ezzel a hangfelismerő motorral van bekonfigurálva gyárilag, így a felhasználónak nem kell külön telepítenie azt. A készülék tartalmazza az összes szükséges nyelvi modell fájlt, amely nem csak több tíz vagy akár száz gigabájtnyi tárhelyet spórol az alkalmazásnak, hanem pontos felismerést biztosít több mint 120 nyelv esetén. A funkcionalitás továbbá teljes mértékben működőképes offline üzemmódban is.

Hátrányok közé sorolható az, hogy a natív API több éve érintetlen hibákat tartalmaz, amelyek miatt bizonyos flagek és függvények teljesen működésképtelenek. Egyik ilyen flag a hangfelismerés működtetésének időtartamát hivatott meghosszabbítani. Ez a hangfelismerő motor rövidebb kérések fogadására lett létrehozva, így az előbb említett flag nélkül a felismerés az első felismert mondat, vagy 5 másodperc szünet után megáll. A teleprompter azonban folyamatos hangfelismerést igényel, így megoldáskepp az alkalmazás a felismerés ismételt

újraindításával éri el a folyamatos hangfelismerés hatását. Egy másik hátrány az Android limitációiból ered: a mikrofonhoz egyszerre csak egy folyamat férhet hozzá, így lehetetlenné vált a hangfelismerést hangrögzítéssel kombinálni későbbi visszahallgatás céljából.

A React Native Voice csomag különböző aszinkron metódusokat és eseménykezelő függvényeket biztosít a hangfelismerő használatára. A `Voice.isAvailable()` függvénnyel leellenőrizhető, hogy a készülék rendelkezik-e a szükséges hangfelismerő szolgáltatással. A `Voice.start(locale)` függvény létrehoz egy SpeechRecognizer példányt, amely kapcsolatot teremt a készülék hangfelismerő motorjával, és elindítja azt a paraméterként megadott nyelv felismerésére. A hangfelismerő megállítása csakis a `Voice.destroy()` függvénnyel volt lehetséges, amely törli az épp futó SpeechRecognizer példányt. Ezen függvények megfelelő meghívásával sikerült egy "folyamatos", szabadon szüneteltethető felismerést eredményezni.

Az eseménykezelő függvények közül a legfontosabbak a `Voice.onSpeechResults(event)` és `Voice.onSpeechPartialResults(event)` függvények. Az előbbi egy mondat vagy felismerési szakasz végén visszatéríti a teljes elhangzott mondatot 5 különböző változatban. Ezek a változatok tartalmazzák a lehetséges eredményeket a felismerési bizonyosság sorrendjében. Az utóbbi minden felismert szó után visszatéríti az addigi részlegesen felépített mondatot. A teleprompter ezeknek a kimeneteknek az utolsó szavait használja fel és keresi a beadott beszédszöveg azon részében, amely még nem hangzott el. Említésre méltó továbbá a `Voice.onSpeechError(event)` függvény, amely egy hibát térít vissza, ha nem sikerült egyetlen szót sem felismerni 5 másodperc alatt. Ezt a függvényt fel lehetne használni a későbbiekben hosszabb szünetek tartásának beazonosítására.

A szövegfelismerés javítására az alkalmazás a Damerau-Levenshtein [26] távolságkereső algoritmust használja, mely két string hasonlóságát vizsgálja. Az algoritmus figyelembe veszi a stringek egyenlővé tételéhez szükséges karakter beszúrások, törlések, helyettesítések és felcserélések számát, melyek különböző súlyokkal rendelkeznek, így megadva a két string közötti "távolságot".

Két fő megvalósítása létezik: Megszorított szerkesztési távolság és Egymás melletti transzpozíciókkal mért távolság. A két megvalósítás annyiban tér el, hogy míg az első a szerkesztési műveletek számát méri a két string egyenlővé tételéhez úgy, hogy minden al-string csak egyszer szerkeszthető, a második megvalósítás nem rendelkezik ilyen megkötésekkel. Ebből kifolyólag a második megvalósítással megnő az algoritmus komplexitása, de pontosabb és gyorsabb eredményeket tud adni. Az alkalmazás az első megvalósítást használja, viszont ez

egyszerűen lecserélhető az implementáció modularitása miatt.

Az algoritmus visszatérít egy számértéket, így ezzel meghatározható egy $(0, 1]$ intervallumon lévő százalékos érték, mely a két bemeneti string hasonlóságát reprezentálja. Ezzel a százalékos értékkel, illetve egy maximális távolsággal finomhangolható az algoritmus, így egyszerűen megváltoztatható a szöveg felismerésének pontossága.

Legelső implementációban egy 0.8-as százalékos határ és egy maximális 10 lépés távolsággal már lényegesebben jobb szöveg felismerést produkált az alkalmazás. Jelen dolgozat írásának idejében, az alkalmazás egy 0.5 százalékos és egy maximális 8 lépéses távolsági megszorítás figyelembevételével képes felismerni az akcentussal, inkonzisztens hanglejtéssel vagy sebességgel elmondott szavakat is. Egy későbbi implementációban a felhasználónak lehetősége lesz ezeket a finomhangolási értékeket személyre szabnia.

3. Felhasznált technológiák és eszközök

Jelen fejezet a fejlesztés során felhasznált technológiák részletes bemutatását tartalmazza.

3.1. Google Firebase platform

Az alkalmazás tulajdonképpeni backend részének megvalósítására a *Google Firebase* [20] szolgáltatásai vannak felhasználva.

A Google Firebase egy olyan felhő alapú platform, melynek szolgáltatásai megkönnyítik a mobilos alkalmazások tesztelését, fejlesztését és felügyeletét. A Google Cloudra [21] épül, elsődlegesen webes és mobil alkalmazásokra specializált platform. Számos funkcionalitással rendelkezik, mint például integráció Google Ads-al értékesítés céljából, a Google Play áruházzal android alkalmazások automatikus kitelepítése céljából. Továbbá saját autentikációs szolgáltatással is rendelkezik, megkönnyítve annak biztonságos implementálását. Ezen felül lehetőséget nyújt adatok tárolására a Google Firebase Firestore [22] segítségével, illetve számos bővítmény alkalmazható, melyekkel akár serverless függvényekkel is bővíthető egy adott projekt.

3.1.1. Adatok tárolása

Az adatok két helyen tárolódnak el: lokálisan az eszközön és a felhőben. Lokálisan tárolásra kerül a bejelentkezett fiók adatai, dokumentumai és beállításai. Ezek kijelentkezés után törlődnek az eszközről.

Különböző eszközök közötti szinkronizálás miatt ezek az adatok, a személyes beállításokon kívül, tárolásra kerülnek a felhőben is. Erre a Firebase Firestore NoSQL dokumentum alapú adatbázisát használtuk.

Egy dokumentum alapú adatbázisban, a relációs táblákkal ellentétben, nincsenek táblák. Helyettük az adatok különböző "dokumentumokban", kulcs-érték párosokban tárolódnak, amelyek különböző kollekciókat alkotnak. [3]

A 2.3 szekcióban említésre kerültek a különböző bejelentkezési módok. Noha a Firebase lehetőséget ad különböző bejelentkezési módra, ezeket az adatokat néhány esetben különbözőképpen is kezeli. Például Google fiókkal való bejelentkezés esetén minden fiókhoz tartozó információ a Google szerverein, azaz a fiók szolgáltatójánál tárolódnak. Ezzel ellentétben az e-maillel és jelszóval bejelentkezett fiókok esetében a szolgáltató

maga a Firebase, tehát itt kell eltárolni az adatokat. Viszont ez megnehezíti az adatok szinkronizálásának és karbantartásának feladatát, mivel a fiókok szolgáltatói különböző, saját ID generátorral rendelkeznek, így rendkívül komplex lenne a dokumentumok felhasználó fiókhoz való kötése.

Megoldásképpen az alkalmazás Firestore adatbázisa két külön kollekcióval rendelkezik: egy a felhasználói adatok számára, és egy a dokumentumok számára.

A felhasználói kollekcióba kerül tárolásra minden regisztrált felhasználó adata. Eltárolódik az alkalmazásban megjelenítendő felhasználó neve, a fiókhoz tartozó e-mail cím valamint a Firestore által generált dokumentum ID.

A dokumentum kollekcióban a létrehozott dokumentumok tárolódnak. Minden dokumentum esetében tárolásra kerül az utolsó módosítás dátuma, a dokumentum nyelve, egy előzetes megjelenítési szöveg, a dokumentum címe, a dokumentum alcímei, a dokumentum tartalma, a prezentációhoz beállított határidő, illetve azon felhasználói ID, amihez a dokumentum tartozik, illetve amit a Firestore generált a felhasználó számára.

Így elegendő ezt a két kollekciót vizsgálni, hogyha valamilyen adatra lenne szükség. Még el nem tárolt, új bejelentkezés esetén lekérdezésre kerülnek a fiók adatai a szolgáltatójától (amennyiben az nem a Firebase), lokálisan tárolódik egy biztonságos helyre, majd létrejön egy új dokumentum a felhasználók Firestore kollekcióban. Így kijelentkezés következtében, noha az adatok törlődnek a lokális tárhelyből, a felhőben megmaradnak, így elegendő újra bejelentkezni azok eléréséért. Anonim vendég felhasználói fiók esetében viszont a kijelentkezés következtében a felhőből is törlődik minden adat.

3.2. Szerver technológiák

A projekt több különböző szerver technológiát tartalmaz az átalakító szolgáltatás megvalósítása érdekében.

3.2.1. Kubernetes

A Kubernetes [10], vagy másképpen K8s, egy nyílt forráskódú rendszer, mely konténerizált alkalmazások automatikus kitelepítését, skálázását és kezelését hivatott megvalósítani. Szinte bárhol, bármilyen operációs rendszeren futtatható, rendkívül könnyű horizontálisan skálázni vele az alkalmazásokat, könnyen fel lehet osztani a tárhely erőforrásokat több alkalmazás között, illetve a terhelés elosztásával (Load Balancing) optimalizálni tudja a végpontok közötti

forgalmat.

A konténerizált alkalmazások hasonlóak a Virtuális Gépekhez. Saját fájlrendszerrel, processzor és memória erőforrással, illetve operációs rendszerrel rendelkeznek, illetve mindkettő teljesen elszigeteli az alkalmazásokat a gazda operációs rendszertől, ahol futnak. A különbség viszont, hogy míg a virtuális gépek egy teljes fizikai számítógépet szimulálnak és kezelik az erőforrásokat, a konténerek csak egy adott alkalmazást és az ahhoz szükséges könyvtárakat, erőforrásokat tartalmazzák és kezelik. Így egy konténer csak azt a feladatot végzi el, amiért létre lett hozva, semmi többet, nem pazarolja az erőforrásokat. Továbbá a konténerek kezelése sokkal egyszerűbb, könnyebben lehet őket létrehozni, skálázni, kezelni.

A konténerek könnyű kezelés érdekében úgynevezett Podokba kerülnek, melyek egy vagy több konténerből állnak. Ezekben a podokban a konténerek közös tárhely és hálózati erőforrásokkal rendelkeznek, így a konténerek karbantartásának felelőssége a podra hárul. Ha a konténerek a podokon kívüli világgal is szeretnének kommunikálni, ezt megtehetik a Service API segítségével, ami feltár egy adott alkalmazást egy adott podban, hogy elérhető legyen a külvilág számára. A podok különböző Node-okon futnak, amik vagy virtuális, vagy fizikai gépek, és tartalmazzák a podok futtatásához szükséges összes szolgáltatást. Egy vagy több node egy Clustert alkot, mely rendelkezik egy további vezérlő egységgel, amely a különböző node-okat hivatott kezelni. A vezérlő egység dönt arról például, hogy egy pod mikor és hogyan kellene újra induljon, vagy milyen szolgáltatásokat kell létrehozni.

A klasztereket különböző képekből (Image) lehet létrehozni. Ezek a képek leírják, hogy milyen környezetben, honnan, milyen további könyvtárakkal és milyen indító paranccsal induljon el a konténer. Majd ezeket a képeket egy konfigurációs állományban felhasználva létrehozhatóak a konténerek. Erre található egy példa a 2. kódrészletben található A Kubernetes ilyen konfigurációs állomány segítségével irányítható és kezelhető, különféle paraméterek beállításával.

Az alkalmazáshoz szükséges webes átalakító szolgáltatás is egy, a Codespring által biztosított Kubernetes klaszterben lévő konténerben fut. Mivel semmilyen más funkcionalitást nem lát el a szolgáltatást, csak HTTP kéréseket kiszolgálva alakít át fájlokat, ezért igénybe lehetett venni a konténerek által nyújtott előnyöket. A klaszterben létezik egy Load Balancer a terhelés elosztása végett, tehát az átalakító nagy mennyiségű kérést tud kezelni egyszerre anélkül, hogy összeomlana vagy késleltetve tudná feldolgozni az adatokat.

Valamint, ahogyan az a 2.4 szekcióban is említésre került, a PDF fájlok átalakítása miatt

szükség van a JRE jelenlétére. Ezért a konténer képe egy NodeJS konténer képet terjeszt ki a JRE telepítésével és beállításával, ahogyan az a 3. kódrészletben is látható. A konténerizálás továbbá lehetővé teszi az átalakító által létrehozott naplók alapján követni az átalakításokat, így megkönnyítve a hibák javítását.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: prophet-backend
spec:
  selector:
    matchLabels:
      app: prophet-backend
  template:
    metadata:
      labels:
        app: prophet-backend
    spec:
      containers:
      - name: prophet-backend
        image: r.edu.codespring.ro/mentor-prophet-2022/prophet-backend:latest
        resources:
          limits:
            memory: "256Mi"
            cpu: "500m"
        ports:
        - containerPort: 8080
```

2. kódrészlet. Átalakító szolgáltatás konténerének kitelepítéséhez szükséges konfigurációs állomány

```
FROM node:18.0-slim
COPY . .
RUN npm install
# Install JRE
RUN mkdir -p /usr/share/man/man1 /usr/share/man/man2 && \
    apt-get update &&\
    apt-get install -y --no-install-recommends openjdk-17-jre && \
    apt-get install ca-certificates-java -y && \
    apt-get clean && \
    update-ca-certificates -f;
ENV JAVA_HOME /usr/lib/jvm/java-17-openjdk-amd64/
CMD ["node", "index.js"]
```

3. kódrészlet. Átalakító szolgáltatás konténerének képe[4]

3.2.2. NodeJS

A NodeJS [12] egy JavaScript értelmező motor, mely segítségével böngésző nélkül lehet JavaScript kódot futtatni állományokból, vagy akár interaktív konzolból. Első sorban hálózati alkalmazások készítésére lett létrehozva. Aszinkron, nem blokkoló műveleteket használ, egyetlen szálon futnak az események. A feladatokat az Eseménykezelő Ciklusba (Event Loop) helyezi el, és ez alapján végzi el azokat. Ha nincs több feladat, kilép a ciklusból, új feladat esetén viszont ismételten visszalép abba. Habár csak egyetlen szálat használ, a fejlesztőknek lehetőségük van gyerek folyamatok létrehozására.

Az átalakító szolgáltatás JavaScriptben van megírva, továbbá mivel az átalakító valójában egy hálózati alkalmazás, a NodeJS tökéletesnek bizonyult a megvalósításhoz. Segítségével létrehozható és tesztelhető volt az átalakító szolgáltatás még azelőtt, hogy az ki lett volna telepítve bárhová is. Ahogyan a 3.2 szekcióban is említve lett, az átalakító szolgáltatás harmadik felektől származó csomagok alkalmazásával működik, ehhez az npm került használatra, amiről a 3.4.1 szekcióban található részletesebb leírás.

3.2.3. Express

Az Express [6] egy web-keretrendszer NodeJS applikációk számára. Segítségével rendkívül egyszerű HTTP kéréseket írni és kiszolgálni, illetve más webes funkcionalitásokat is viszonylag egyszerűen lehet megírni. Teljesítmény szempontjából is kitűnő, mivel csak a legfontosabb alap, webes alkalmazások számára szükséges funkcionalitásokat biztosítja, így nem rontja el a NodeJS által nyújtott teljesítményt.

Middleware-eknek nevezett függvények segítségével rendkívül flexibilis módon kezelhetőek a kérések. Egy middleware feladata a különböző kérések köztes feldolgozása, nem térít vissza végleges választ. Elvégzi a számára kiszabott feladatot, majd a kérést tovább engedi a feldolgozási láncon. Több middleware is kapcsolható egymáshoz, melyek különböző módon viselkedhetnek a láncon előttük lévő hívó függvény eredménye alapján. Egy példa a middleware-ek működésére egy bejelentkezési folyam. A felhasználó beírja az adatait, a middleware ellenőrzi azokat, és az adatok hitelessége alapján a megfelelő oldalra irányítja át a felhasználót. Egy másik példa egy, a beérkező kéréseket naplózó middleware, melynek a kérések elmentésén kívül más feladata nincs.

Egy másik fő funkcionalitása a különféle végpontokra érkező kérések kezelése, ez egy

[4] A kód forrása: https://stackoverflow.com/a/73311333

Routingnak nevezett mechanizmussal valósul meg. Routernek nevezett objektumok képesek különbséget tenni a HTTP kérések típusa között. Az Express kiosztja a HTTP kérést a megfelelő routernek a kérésben szereplő URL alapján, majd a router a kérés fejlécében szereplő metódus által eldönti hogyan kezelje azt. Például GET metódus esetén a routernek vissza kell térítenie a kért erőforrást, POST metódus esetén viszont egy új erőforrást kell létrehoznia a kérés testében szereplő adatok alapján.

3.3. Mobil technológiák

A projekt központi eleme egy mobil alkalmazás, amit a felhasználók bárhol bármikor használhatnak a mobil eszközükről. Ennek megvalósítására különböző keretrendszerek lettek figyelembe véve, ezek közül három tűnik ki a leginkább: Kotlin Multiplatform Mobile, Flutter és React Native. Mindhárom keretrendszernek megvannak a saját előnyei és hátrányai, viszont végül a React Native-re esett a választás.

3.3.1. React Native

A React Native [13] a Meta Platforms, Inc. által kifejlesztett nyílt forráskódú, multiplatform UI szoftver keretrendszer, melynek elsődleges célja az olyan alkalmazások létrehozásának segítése, melyek igénylik az adott platform által nyújtott natív lehetőségeket, amelyre telepítve vannak, mely lehet például Android, iOS, macOS, Web, vagy akár Windows is. Ezt úgy éri el, hogy a felhasznált komponenseket az alkalmazás építésekor lecseréli egy adott platform által nyújtott megfelelőjére, például a React Native kódban lévő *View* komponenseket lecseréli Android esetén *ViewGroup*, iOS esetén *UIView* elemekre. Rendkívül sokoldalú, és ahogy a neve is sugallja, nagyon sok közös van közte és a React keretrendszer között, mint például, hogy mindkettő JavaScriptet használ az alkalmazások implementálására, vagy hogy mindkettő hasonló módon kezeli a komponenseket. Saját komponensek használata mellett megengedi a natív komponensek írását is, továbbá npm segítségével sok harmadik féltől származó csomag is felhasználható az alkalmazásban. Támogatja a hot-reload funkcionalitást a 3.3.2 szekcióban tárgyalt Expo segítségével, felgyorsítva a fejlesztés menetét.

Fontos megjegyezni, hogy habár sokban hasonlít a böngészők által támogatott Reactre, és sok Reactre szánt npm csomag is működni fog vele, mégsem garantált a teljes inter-kompatibilitás. Ennek megoldása érdekében sok npm csomag rendelkezik külön React Native-re szánt verzióval is.

Teljesítmény szempontjából megfelelt a projekt elvárásainak, elég gyors és stabil olyan szempontból, hogy nincs gondja a hardware által nyújtott lehetőségek kihasználásával, nem ütközik problémába. Valamint a 3.3 szekcióban is említett előzetes tesztek alapján megfelelő teljesítménnyel rendelkezett az alkalmazás megvalósításához. Így végül az alkalmazás fejlesztése a React Native keretrendszerrel történt meg.

3.3.2. Expo

Az Expo [5] egy nyílt forráskódú univerzális platform és keretrendszer, mely megkönnyíti a JavaScriptet/TypeScriptet használó és React/React Native-ben megírt alkalmazások kezelését. Segítségével lehetőség van új projektek létrehozására úgy, hogy előre létrehozza a fejlesztéshez szükséges konfigurációs fájlokat és mappákat, vagy akár már meglévő React/React Native projektekben is alkalmazható. Ha szükséges, Snackeknek nevezett kód környezetekkel példa kódokat is meg lehet osztani másokkal.

Lehetőséget ad kétféle futtatási módra. Az egyik ezek közült a menedzselt mód, ahol az Expo felel mindenért, beleértve az alkalmazás felépítését is, viszont emiatt csak egyetlen egy JavaScript/TypeScript kódbázist enged meg használni, nem ad lehetőséget natív komponensek megírására. A másik futtatási lehetőség a tiszta futtatási mód, ahol már a fejlesztő felel az alkalmazásért, így natív komponensek is használhatóak.

3.4. Eszközök

3.4.1. npm

A 3.2.2 szekcióban már említett NodeJS egyik nagy előnye, hogy npm [31] segítségével különféle funkcionalitásokkal bővíthetőek az alkalmazások. Az npm, vagyis Node Package Manager egy függőségkezelő rendszer a NodeJS-hez, több százezer csomag használatára ad lehetőséget, melyeket bárki letölthet, vagy melyek közé bárki feltöltheti saját csomagját, teljesen ingyen. Az npm konzol alkalmazás képes különböző, harmadik féltől származó csomagok telepítésére, frissítésére, egymás közti kompatibilitás ellenőrzésére és megoldására. NodeJS telepítésekor az npm is telepítődik.

3.4.2. Git & GitLab

A Git [7] egy nyílt forráskódú verzió követő rendszer, mely a kódbázisban történő változásokat követve lehetőséget ad annak visszaállítására egy előzetes verzióra, továbbá megkönnyíti a több fejlesztő által történő fejlesztést.

A GitLab [8] egy olyan rendszer, mely lehetőséget ad különböző kódbázisok tárolására, és azok kezelésére. Képes kezelni a kódbázisokban megjelenő verziók közötti különbségeket, meg tudja jeleníteni azokat a fejlesztők számára, illetve bizonyos esetekben automatikusan össze tud vonni két kódbázist vagy verziót. Továbbá különböző eszközökkel képes elősegíteni a fejlesztést, például képes a feladatok felvezetésére és kiosztására is a fejlesztők között. Egy másik hasznos funkcionalitása a Gitlab CI/CD[5] szoftverfejlesztési eszköz, mely segítségével automatizált feladatok végezhetőek el a fejlesztés különböző fázisaiban. Erre egy példa, hogy bizonyos, kódbázison belüli változások esetén futtasson le valamilyen teszteket, melyekkel ellenőrizni tudja a különböző funkcionalitások és verziók közötti kompatibilitást. Egy másik példa, hogy amikor a kódbázis a fejlesztők által egy stabil állapotba kerül, az automatikusan felépüljön és kitelepítődjön egy előre megadott helyre.

3.4.3. ESLint

Az ESLint [4] egy statikus kódelemző, melynek célja, hogy a kód konzisztens, valamint formailag és logikailag helyes maradjon a fejlesztés során. A projektben ezek a tesztek a 3.4.2 szekcióban már említett CI/CD mechanizmus segítségével minden alkalommal lefutnak, amikor valamiféle változás történik a kódbázisban, így sok hiba lett elkerülve a fejlesztés során.

3.4.4. VSCode

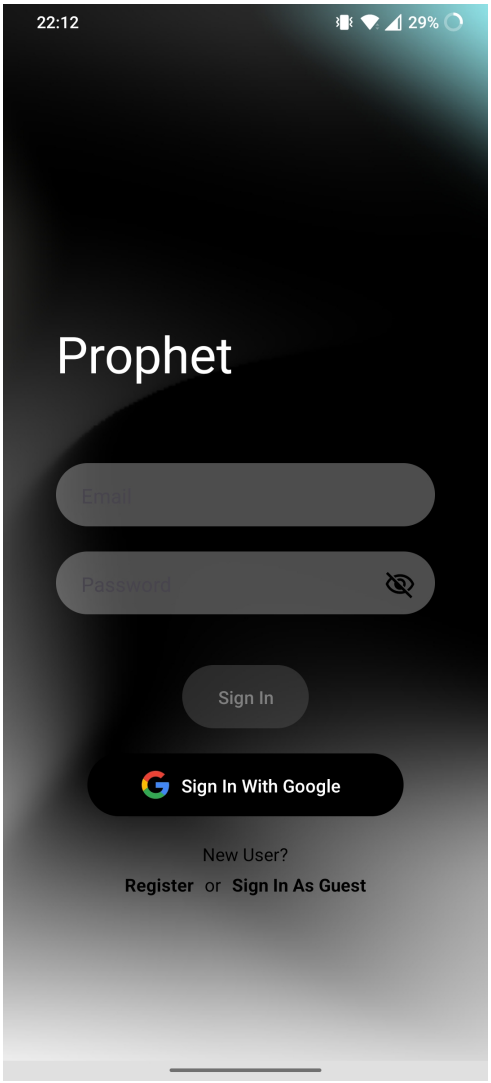
A Visual Studio Code [28] (röviden VSC) egy Microsoft által fejlesztett forráskód-szerkesztő alkalmazás. Számos programozási nyelvet támogat, intuitív kezelő felületének köszönhetően segít a fejlesztés folyamatában. Egy nagy erőssége, hogy rendkívül sok, akár harmadik féltől származó kiegészítő csomag áll rendelkezésre, melyekkel tovább fokozható a használati élmény. Például létezik olyan kiegészítő csomag, mely egy felhasználó barát kezelő felületet biztosít a 3.4.2 szekcióban már említett Git kezelésére, vagy olyan kiegészítő csomag, mely támogatja a statikus kódelemzők, mint például a 3.4.3 szekcióban említett ESLint automatikus futtatását is.

[5] CI/CD - Continuous Integration / Continuous Development

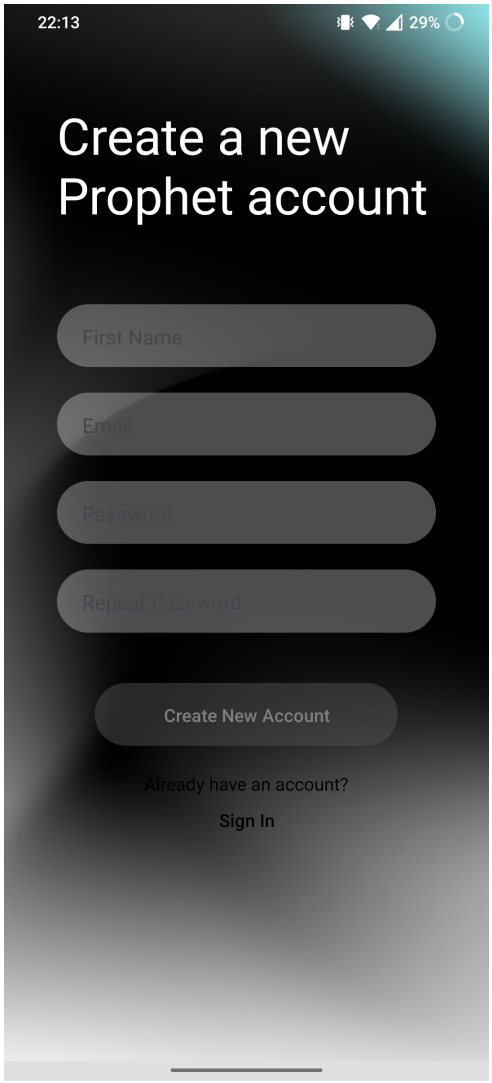
4. Az alkalmazás működése

Az alábbi fejezet bemutatja az alkalmazás működését, használati folyamatát. A különböző személyre szabhatósági lehetőségek szemléltetése érdekében egyes helyeken több képernyőkép kíséri a funkcionalitás leírását.

Az 1.1. fejezetben bemutatott funkcionalitások eléréséhez a felhasználónak először be kell jelentkeznie az alkalmazásba a bejelentkezési felületen keresztül (5. ábra), vagy ha még nem rendelkezik felhasználói fiókkal, létrehozhat egyet a regisztrációs oldalra navigálva (6. ábra), vagy egyetlen gombnyomással Google autentikációt használva. Hagyományos regisztráció esetén a felhasználónak meg kell adnia egy keresztnevet, e-mail címet, és egy jelszót. A bejelentkezési oldal „Sign In" gombja, és a regisztrációs oldal „Create New Account" gombja inaktív marad mindaddig, amíg az összes mező ki nem lesz töltve a szükséges adatokkal. A



5. ábra. Bejelentkezési oldal.

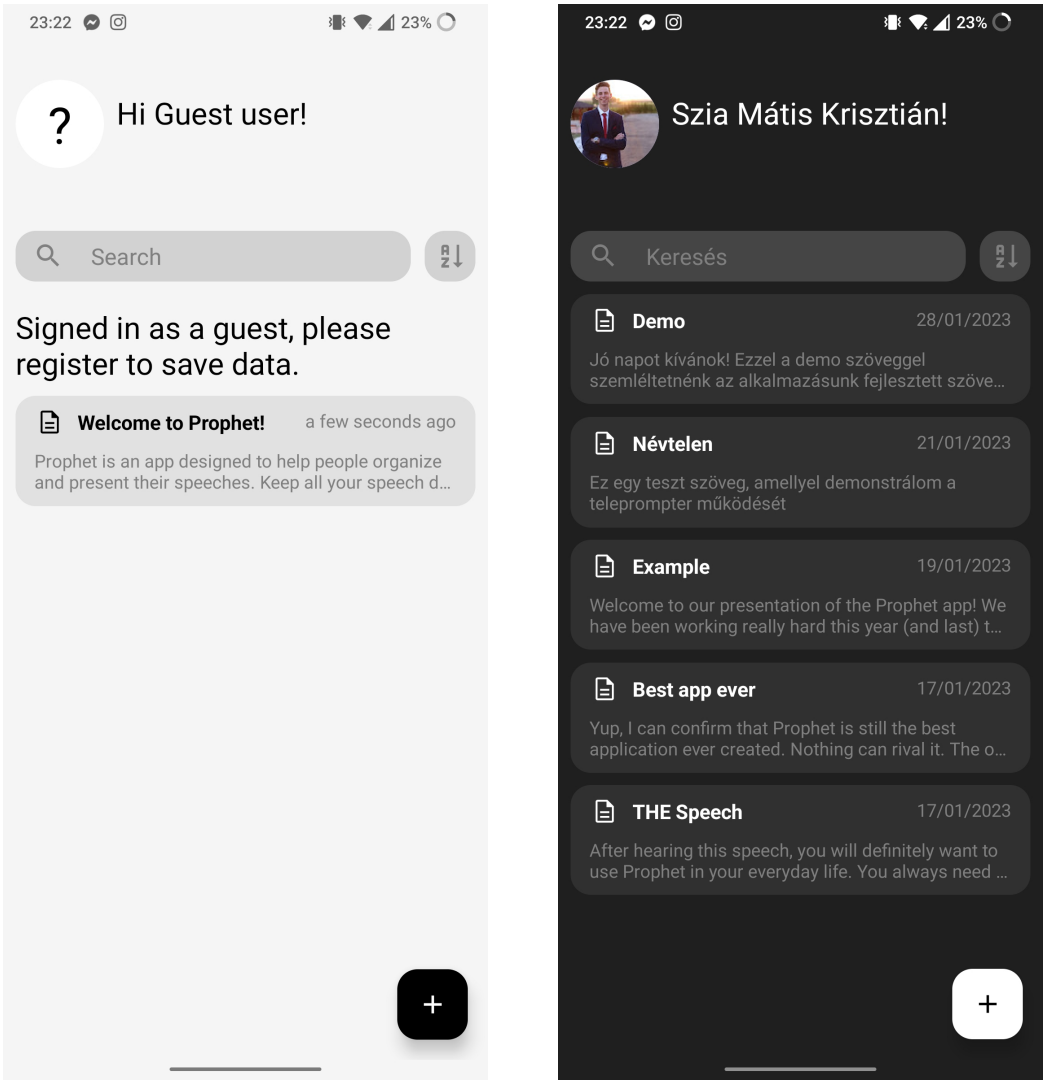


6. ábra. Regisztrációs oldal.

regisztrációs oldalon továbbá a bevitt adatok igazolva is vannak, és rossz formátum esetén egy figyelmeztető üzenet jelenik meg a kijelző tetején.

Sikeres bejelentkezés után elérhetővé válik az alkalmazás főoldala, a dokumentumlista (7. ábra), amely, ahogyan a név is sugallja, a felhasználó dokumentumait listázza. A lista tetején levő keresősáv segítségével szűrni lehet a dokumentumokat címeik szerint, a keresősáv jobb oldalán levő gombbal pedig megváltoztatható a listázás sorrendje az alapértelmezett, szerkesztési dátum szerinti rendezésről a címek szerinti ábécé sorrendbe. Egy lista elem nyomva tartásával a felhasználó kijelölheti az adott dokumentumot vagy dokumentumokat, és törölheti azokat a képernyő jobb alsó sarkában levő törlés gomb segítségével (8. ábra).

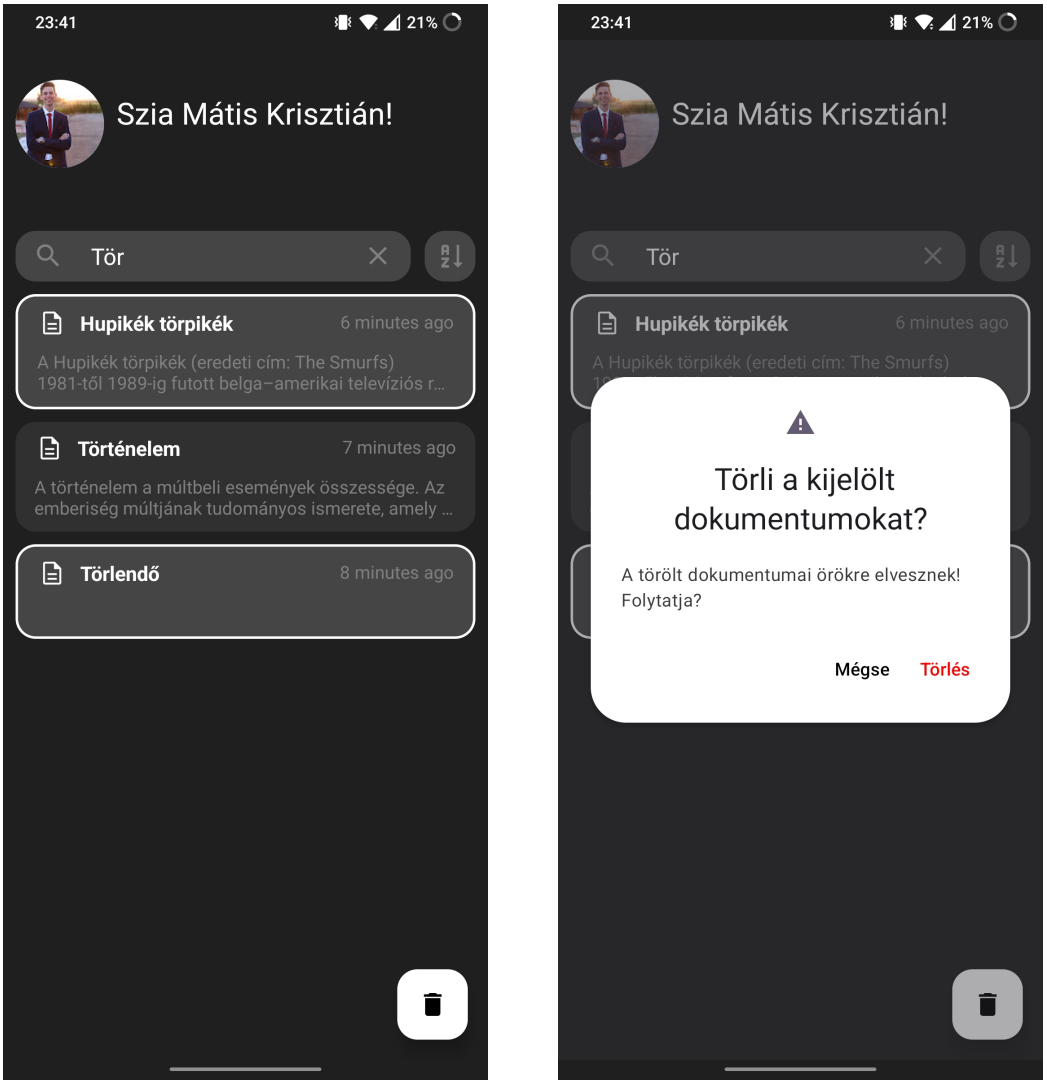
Ha egyetlen dokumentum sincs kijelölve, az imént említett törlés gomb egy másik szerepet tölt be, mégpedig a dokumentum létrehozás szerepét. Ezt megnyomva a bejelentkezett



7. ábra. Az alkalmazás főoldala.

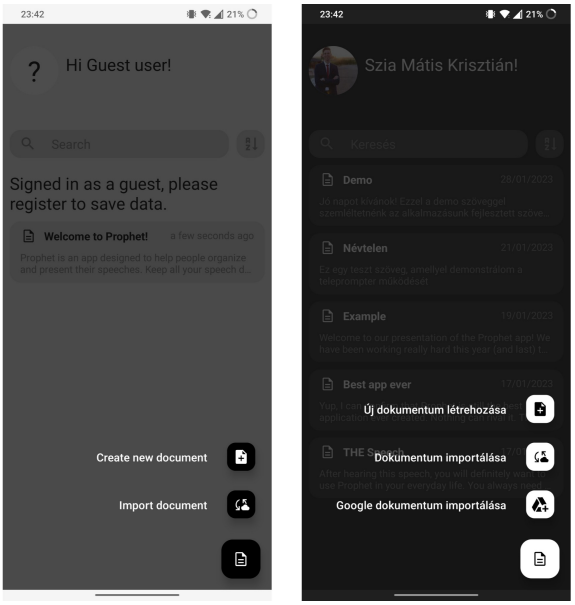
felhasználó szerepkörétől függően két vagy három opció jelenik meg, melyek a 9. ábrákon láthatóak: Új, üres dokumentum létrehozása, külső dokumentum importálása, és ha az illető felhasználó Google fiókjával jelentkezett be, Google Workspace dokumentum importálása is lehetővé válik. Az első opció megnyomására egy üres szövegszerkesztő felület nyílik meg, míg a második vagy harmadik opció választásakor a készülék natív eszközböngészője, vagy a kompatibilis Google Workspace dokumentumok listája jelenik meg (10. ábra). Ezeken kiválasztva egy dokumentumot, annak tartalma bekerül az applikációba és elmentődik mint egy új dokumentum, és ezzel egyidőben megjelenik a szövegszerkesztő felület az említett tartalommal.

Még mielőtt rátérnénk a szövegszerkesztő felületre, fontos megemlíteni még a dokumentumlista tetején látható üdvözlő üzenetet, és a mellette látható avatar ikont. Ha a

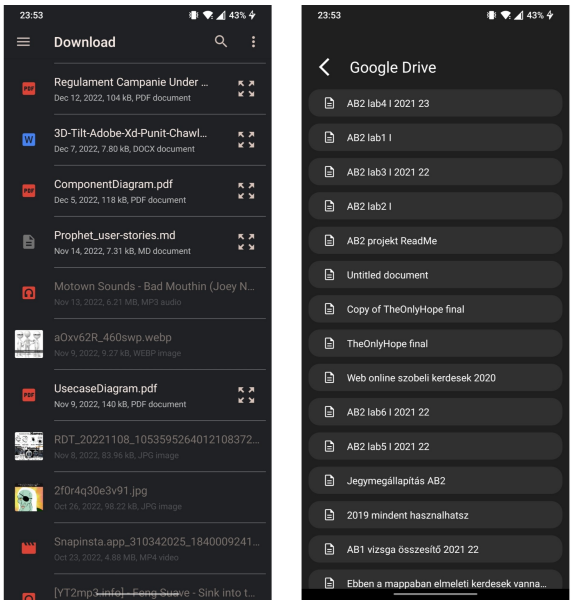


8. ábra. Dokumentumok törlése.

felhasználó Google fiókkal jelentkezett be, az avatar a Google fiók profilképét jeleníti meg, az üdvözlő üzenet pedig szintén a Google fiókhoz tartozó névvel köszönti a felhasználót. Hagyományosan regisztrált fiók esetén az üzenet a megadott keresztnevet, az avatar pedig a keresztév kezdőbetűjét tartalmazza, vendégfelhasználó esetében pedig az avatar egy anonimitást jelző kérdőjelet mutat, és az üdvözlő szöveg is a Vendégfelhasználónak lesz címezve.



9. ábra. Dokumentum létrehozás opciói.

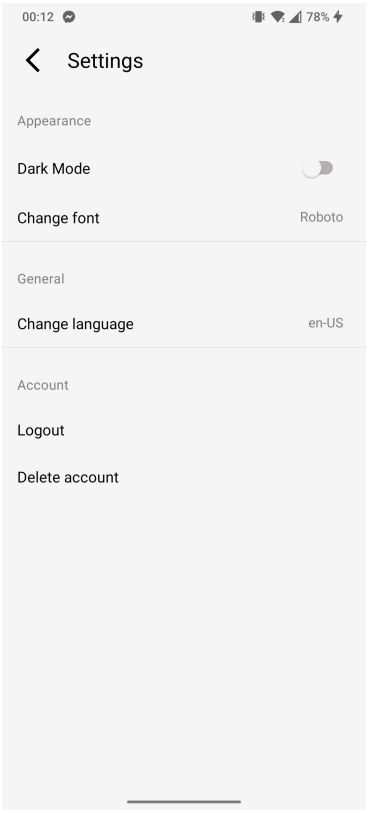


10. ábra. Importálandó dokumentum kiválasztása.

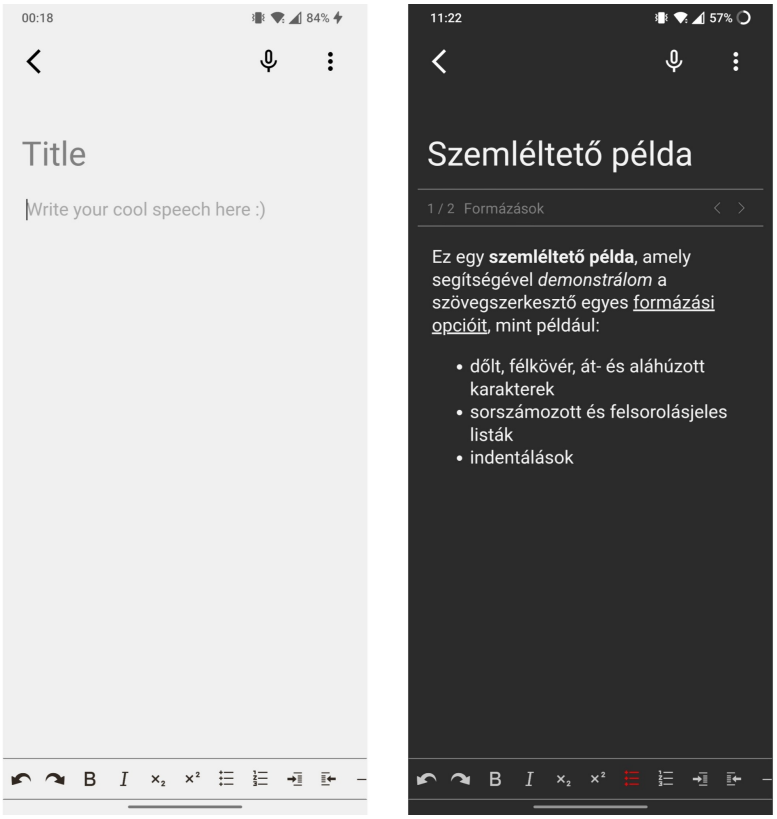
Az avatar nem pusztán vizuális elem, annak megnyomásával érhető el a beállítások menüje, amelyet a 11. ábra szemlélteti. Innen megváltoztatható az alkalmazás témája, a teleprompter által használt betűtípus, és az alkalmazás nyelve. Jelenleg angol és magyar nyelvek támogatottak. Továbbá itt van lehetőség kijelentkezésre, vagy a felhasználói fiók teljes törlésére.

Visszatérve a főoldalra, egy dokumentum megnyitása a szövegszerkesztő felületre vezet (12. ábrák). Itt módosítható a cím, többoldalú dokumentumok esetén minden oldal alcíme, és a tartalom. A tartalmat a kijelző alján levő eszközsáv opcióival lehet díszíteni az átláthatóság fokozása érdekében. A menügombra nyomva további opciók ugranak fel: új oldal beszúrása, oldal törlése, dokumentum törlése, illetve dokumentum nyelvének specifikálása, és opcionális időkorlát beállítása. Ez utóbbi két beállítás a teleprompter működését érinti, amely az oldal tetején levő mikrofon gomb segítségével érhető el.

A teleprompter oldal a 13. ábrákon látható leegyszerűsített felületen keresztül biztosítja a zökkenőmentes használatot. Csak a szöveg tartalma látható, semmilyen formázási opció nélkül, illetve a kijelző alján lebegő gyorsgombok, melyek szerepei: a betűméret módosítása, oldalak



11. ábra. Beállítások menüje.



12. ábra. Szövegszerkesztő felület.

közti váltakozás, és a hangfelismerés elindítása, leállítása. Az oldal tetején látható továbbá a kétfunkciós időzítő, mely piros hátterűre vált a hangelismerés elindításakor, és jobboldalt egy újrakezdési opció is elérhető. A hangfelismerés elindításakor a készülék folyamatosan figyeli a felhasználó hangját, és a felismert szavakat sárga háttérrel látja el, jelezve ezáltal a beszéd előrehaladtát. Ezen kívül a szöveg automatikusan gördül a legutóbbi felismert szóhoz, oldal végére érve az oldalváltás is magától megtörténik, így szükség esetén a felhasználónak elég csak rápillantania a kijelzőre, hogy megtalálja a beszédszöveg soron következő szavait. Ha esetleg mégis gyors korrekcióra van szükség, a megjelenített szöveg bármely szavát meg lehet érinteni, és a szöveg követése az érintett szótól fog folytatódni.

13. ábra. Automatizált teleprompter.

Következtetések és továbbfejlesztési lehetőségek

A Prophet alkalmazás fejlesztése során egy olyan alkalmazás született, amely megkönnyíti a beszédek megtartását és a szövegek karbantartását digitalizáció és automatizált súgógép segítségével.

Az alkalmazás egy könnyen kezelhető, egységesített felületet biztosít a különböző forrásokból származó beszédszövegek tárolására, megtekintésére és szerkesztésére, valamint hozzájárul a beszédek előadásának megkönnyítéséhez a hangfelismerés alapú teleprompter által. A különböző személyre szabhatósági lehetőségek, és internetkapcsolat nélküli funkcionalitások egy akadálymentes felhasználói élményt eredményeznek.

Az átalakító szolgáltatás lehetővé teszi a különböző formátumú dokumentumok tartalmának kinyerését, és a formátum egységesítését.

A fejlesztés során több olyan funkcionalitás is megfogalmazódott, mely tovább bővíthetné az alkalmazás kínálatát és hasznosságát.

Egyik ilyen funkcionalitás a beszéd automatikus kiértékelése. Módosítani lehetne a teleprompter kódját úgy, hogy megjegyezze a beszéd közben elhangzott töltelékszavakat, hosszabb szüneteket, átugrott szavakat, és ezeknek előfordulási helyeit a szövegben, hogy az előadás vagy gyakorlás után a felhasználó visszanézhesse, hol vétett hibát.

A gyakorlásban sokat segítene az is, ha a felhasználónak lehetősége lenne hallani a saját hanglejtését és tempóját, amelyhez hozzájárulna egy hangrögzítési lehetőség implementációja. A felhasználónak lehetősége lenne visszahallgatni mindegyik előzetes próbálkozását beszédenként csoportosítva. Az Android azonban egyszerre csak egy folyamatnak enged mikrofonhozzáférést, ezért a jelenlegi hangfelismerő megoldás miatt ez a funkcionalitás nem megvalósítható.

Fejlesztési eszköz hiányában az alkalmazás legtöbb funkcionalitása csak Androidos készüléken lett tesztelve, ezért az iOS-re való teljes adaptálás is a továbbfejlesztési lehetőségek közé sorolható.

Hivatkozások

[1] Magyar Anikó. *A gyors beszéd*. 2010. URL: https://mnytud.arts.unideb.hu/szakdolgozat/1658/magyar%5C_a%5C_1658.pdf (elérés dátuma: 2023. ápr. 27.)

[2] Apache. *Apache Tika - a content analysis toolkit*. URL: https://tika.apache.org/ (elérés dátuma: 2023. ápr. 8.)

[3] Couchbase authors. *Comparing Document Databases and Relational Databases*. URL: https://developer.couchbase.com/comparing-document-vs-relational/ (elérés dátuma: 2023. ápr. 10.)

[4] ESLint authors. *ESLint documentation*. URL: https://eslint.org/docs/latest/ (elérés dátuma: 2023. ápr. 8.)

[5] Expo authors. *What is Expo?* 2022. URL: https://docs.expo.dev/home/core-concepts/ (elérés dátuma: 2023. ápr. 8.)

[6] Express authors. *Express*. URL: https://expressjs.com/ (elérés dátuma: 2023. ápr. 8.)

[7] Git authors. *About Git*. URL: https://git-scm.com/about (elérés dátuma: 2023. ápr. 8.)

[8] GitLab authors. *GitLab documentation*. URL: https://docs.gitlab.com/ (elérés dátuma: 2023. ápr. 8.)

[9] Java authors. *Java Downloads*. URL: https://www.java.com/en/download/manual.jsp (elérés dátuma: 2023. ápr. 8.)

[10] Kubernetes authors. *What is Kubernetes?* 2022. URL: https://kubernetes.io/docs/home/ (elérés dátuma: 2023. ápr. 8.)

[11] Markdownguide authors. *What is Markdown?* URL: https://www.markdownguide.org/getting-started/ (elérés dátuma: 2023. ápr. 25.)

[12] NodeJS authors. *About NodeJS*. URL: https://nodejs.org/en/about (elérés dátuma: 2023. ápr. 8.)

[13] React Native authors. *React Native introduction*. URL: https://reactnative.dev/docs/getting-started (elérés dátuma: 2023. ápr. 8.)

[14] React Native Community. *React Native WebView*. URL: https://github.com/react-native-webview/react-native-webview (elérés dátuma: 2023. ápr. 8.)

[15] MDN contributors. *Fetch API*. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch%5C_API (elérés dátuma: 2023. ápr. 25.)

[16] MDN contributors. *HTML: HyperText Markup Language*. URL: https://developer.mozilla.org/en-US/docs/Web/HTML (elérés dátuma: 2023. ápr. 25.)

[17] MDN contributors. *HTTP*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP (elérés dátuma: 2023. ápr. 25.)

[18] MDN contributors. *MIME types (IANA media types)*. 2023. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics%5C_of%5C_HTTP/MIME%5C_types (elérés dátuma: 2023. ápr. 8.)

[19] Google. *Easy and secure access to your content*. URL: https://www.google.com/drive/ (elérés dátuma: 2023. ápr. 25.)

[20] Google. *Firebase*. URL: https://firebase.google.com/ (elérés dátuma: 2023. ápr. 8.)

[21] Google. *Firebase & Google Cloud*. URL: https://firebase.google.com/firebase-and-gcp (elérés dátuma: 2023. ápr. 10.)

[22] Google. *Firestore*. URL: https://firebase.google.com/docs/firestore (elérés dátuma: 2023. ápr. 25.)

[23] Google. *Google Android applikáció*. URL: https://play.google.com/store/apps/details?id=com.google.android.googlequicksearchbox (elérés dátuma: 2023. ápr. 25.)

[24] Bóna Judit. „A felgyorsult beszéd produkciós és percepciós sajátosságai”. Disszertáció. ELTE, 2007, 5–6. oldal. URL: http://doktori.btk.elte.hu/lingv/bona/Phd%5C_dolgozat%5C_BonaJudit.pdf (elérés dátuma: 2023. ápr. 27.)

[25] Yaroslav Kulinich. *SpeechWay - 3 in 1 Teleprompte*. URL: https://play.google.com/store/apps/details?id=ua.kulya.speechway (elérés dátuma: 2023. ápr. 26.)

[26] madnoreason. *Damerau–Levenshtein distance*. URL: https://www.geeksforgeeks.org/damerau-levenshtein-distance/ (elérés dátuma: 2023. ápr. 8.)

[27] Ganesh Mani. *Build a React Native speech-to-text dictation app*. https://blog.logrocket.com/build-react-native-speech-to-text-dictation-app/. 2022. dec.

[28] Microsoft. *Visual Studio Code documentation*. URL: https://code.visualstudio.com/docs (elérés dátuma: 2023. ápr. 8.)

[29] mwilliamson. *Mammoth .docx to HTML converter*. 2013. URL: https://www.npmjs.com/package/mammoth (elérés dátuma: 2023. ápr. 19.)

[30] React Navigation. *React Navigation*. URL: https://reactnavigation.org/docs/getting-started/ (elérés dátuma: 2023. ápr. 20.)

[31] *npm Docs*. URL: http://docs.npmjs.com (elérés dátuma: 2023. ápr. 8.)

[32] ParrotTeleprompter. *Parrot Teleprompter*. URL: https://play.google.com/store/search?q=parrot+teleprompter%5C&c=apps (elérés dátuma: 2023. ápr. 26.)

[33] Lorelei A. Lingard PhD Richard J. Haber MD. *Learning oral presentation skills*. 2001. URL: https://link.springer.com/article/10.1046/j.1525-1497.2001.00233.x (elérés dátuma: 2023. ápr. 27.)

[34] shebin. *pdf2html*. 2018. URL: https://www.npmjs.com/package/pdf2html (elérés dátuma: 2023. ápr. 19.)

[35] BIGVU Video Studio. *Teleprompter & Video Captions*. URL: https://play.google.com/store/apps/details?id=bigvu.com.reporter (elérés dátuma: 2023. ápr. 26.)

[36] Krepsz Valéria. *Megakadásjelenségek előfordulása a beszédtempó függvényében*. 2015. URL: http://real.mtak.hu/25225/1/MegakiKotetEGYBEN%5C_vegl.pdf (elérés dátuma: 2023. ápr. 27.)

[37] wxik. *React Native Rich Text Editor*. URL: https://github.com/wxik/react-native-rich-editor (elérés dátuma: 2023. ápr. 8.)

[38] x1-. *pptx2json*. 2020. URL: https://www.npmjs.com/package/pptx2json (elérés dátuma: 2023. ápr. 19.)

[39] Mayada Mohamed Tawfik Zaki. *The Development of Fluency, Syntactic Complexity, and Grammatical Accuracy in Oral Presentations: A case of Egyptian EFL Learners*. 2010. URL: https://fount.aucegypt.edu/cgi/viewcontent.cgi?article=2157%5C&context=etds (elérés dátuma: 2023. ápr. 27.)