

# **ETDK-dolgozat**

Hosu Gábor

Kránics Ákos

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

**Informatika II.: innovatív számítástechnikai termékek, alkalmazások szekció**

**Kolozsvár, 2025. május 15–18.**

**B.o.B.**

## **Intelligens dokumentum-felhasználáson alapuló chatbot**

### **Szerzők:**

**Hosu Gábor**

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

**Kránics Ákos**

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

### **Témavezetők:**

**Bándi Nándor**, doktorandusz,

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar

**Farkas Eszter**, szoftverfejlesztő,

Codespring

**Petkes Kinga**, szoftverfejlesztő,

Codespring



# Tartalomjegyzék

|                                                           |           |
|-----------------------------------------------------------|-----------|
| <b>Bevezető</b>                                           | <b>1</b>  |
| <b>1. Projekt leírása</b>                                 | <b>3</b>  |
| 1.1. Funkcionalitások . . . . .                           | 3         |
| 1.2. Architektúra . . . . .                               | 4         |
| <b>2. A RAG módszer és annak alkalmazása</b>              | <b>6</b>  |
| 2.1. Retrieval Augmented Generation . . . . .             | 6         |
| 2.2. Nagy nyelvi modellek . . . . .                       | 7         |
| 2.2.1. Enkóder modell . . . . .                           | 8         |
| 2.2.2. Dekóder modell . . . . .                           | 9         |
| 2.3. Vektoradatbázis . . . . .                            | 10        |
| 2.4. Optical Character Recognition . . . . .              | 11        |
| 2.5. A RAG módszer megvalósítása . . . . .                | 11        |
| 2.5.1. Architektúra . . . . .                             | 11        |
| 2.5.2. Fájlfeldolgozás . . . . .                          | 13        |
| 2.5.3. Cím- és válaszgenerálás beszélgetésekben . . . . . | 15        |
| 2.5.4. Beszélgetések memóriája . . . . .                  | 16        |
| 2.6. Példa a RAG módszer alkalmazására . . . . .          | 17        |
| <b>3. Szerver</b>                                         | <b>19</b> |
| 3.1. Architektúra . . . . .                               | 19        |
| 3.2. Funkcionalitások . . . . .                           | 20        |
| 3.2.1. Adatbázis és adatmodell . . . . .                  | 20        |
| 3.2.2. Felhasználókezelés . . . . .                       | 21        |
| 3.2.3. Állománykezelés . . . . .                          | 22        |
| 3.2.4. Beszélgetések kezelése . . . . .                   | 23        |
| 3.3. Verifikáció . . . . .                                | 23        |
| 3.4. Felhasznált technológiák . . . . .                   | 24        |
| <b>4. Webalkalmazás</b>                                   | <b>25</b> |
| 4.1. Architektúra . . . . .                               | 25        |

|           |                                                         |           |
|-----------|---------------------------------------------------------|-----------|
| 4.2.      | Funkcionalitások . . . . .                              | 26        |
| 4.2.1.    | Bejelentkezés . . . . .                                 | 26        |
| 4.2.2.    | Kezdőoldal . . . . .                                    | 28        |
| 4.2.3.    | Állományok hozzárendelése beszélgetésekhez . . . . .    | 29        |
| 4.2.4.    | Globális állománykezelő rendszer . . . . .              | 31        |
| 4.3.      | Felhasznált technológiák . . . . .                      | 32        |
| <b>5.</b> | <b>Mobilalkalmazás</b>                                  | <b>33</b> |
| 5.1.      | Architektúra . . . . .                                  | 33        |
| 5.2.      | Funkcionalitások . . . . .                              | 34        |
| 5.2.1.    | Kamera opció . . . . .                                  | 35        |
| <b>6.</b> | <b>DevOps</b>                                           | <b>36</b> |
| 6.1.      | Folyamatos integráció . . . . .                         | 36        |
| 6.2.      | Kitelepítés . . . . .                                   | 37        |
|           | <b>Következtetések és továbbfejlesztési lehetőségek</b> | <b>38</b> |

# Bevezető

A B.o.B. - Bot of Brilliance projekt célja egy olyan felület létrehozása, amely lehetővé teszi a felhasználók számára, hogy hatékonyan és gyorsan nyerjenek ki információt dokumentumokból, ezáltal növelve produktivitásukat.

A nagy nyelvi modellek egyre inkább hozzájárulnak a produktivitás növeléséhez, különösen olyan esetekben, amikor rövid idő alatt kell releváns információt kinyerni valamilyen forrásból. Bár az ezen módszerek alkalmazása során elengedhetetlen a kritikus gondolkodás és a források ellenőrzése, kutatások szerint ezen eszközök tagadhatatlanul elősegítik a hatékonyabb munkavégzést is [1].

Gyakori probléma, hogy hosszú dokumentumokat kell átböngészni egy mindössze néhány soros válaszáért. Számos ingyenes és előfizetési eszköz létezik, amelyek ezt a problémát igyekeznek megoldani, például ChatGPT<sup>1</sup>, Deepseek<sup>2</sup> és Grok<sup>3</sup>. Ezen modellek képesek a felhasználók kérésére a tudásukat a feltöltött dokumentumok tartalmára korlátozni. Bár ezen eszközök sok esetben hasznosak, még mindig vannak olyan területek, ahol fejlesztésre szorulnak.

Az egyik legnagyobb hátrányuk például az, hogy a feltöltött fájlokat nem lehet áthelyezni egyik beszélgetésből a másikba anélkül, hogy azokat újra fel kellene tölteni. Ez különösen problémás, ha a felhasználónak egy dokumentumközpontú beszélgetésre van szüksége, ahol a fájlok könnyű kezelése kulcsfontosságú.

B.o.B. egy olyan felületet kínál, amely lehetővé teszi az információkinyerést dokumentumokból és képeken található szövegekből, miközben biztosítja az állományok perzisztens eltárolását. A feltöltött dokumentumokat a felhasználók bármely új vagy meglévő beszélgetéshez hozzárendelhetik, így elkerülhető az ismételt feltöltésekkel járó idővesztés. Ezzel a funkcionalitással B.o.B. kiemelkedik az általános chatbotok közül, és egy dokumentum-orientált beszélgetési rendszerré válik.

A dolgozat célja a B.o.B. alkalmazás különböző aspektusainak bemutatása, többek között: a felhasználói felület tervezési irányelvei, a megvalósított funkcionalitások, az architektúrális felépítés, az alkalmazott technológiák és fejlesztési eszközök.

Az 1. fejezetben a chatrendszer által nyújtott funkcionalitások kerülnek ismertetésre. A fejezet a rendszert képező alrendszerek és komponensek közti viszonyok és azok szerepköreinek

---

<sup>1</sup><https://chatgpt.com>

<sup>2</sup><https://www.deepseek.com>

<sup>3</sup><https://grok.com>

bemutatásával zárul.

A dolgozat 2. fejezetében a chatbot működését biztosító RAG módszer kerül részletesebb bemutatásra annak komponensein, a felhasznált technológiákon és annak megvalósításán keresztül. A fejezet végén a RAG rendszer működésének egy konkrét beszélgetés formájában történő bemutatása olvasható.

A 3. fejezet során a chatrendszer központi szerverének architektúrális felépítése, az általa nyújtott funkciók és a megvalósításához szükséges technológiák kerülnek ismertetésre.

A 4. fejezet a webalkalmazás architektúrális felépítésének bemutatásával indít, amelyet követve ismertetésre kerülnek a webfelület általa nyújtott funkciók és a hozzá felhasznált technológiák.

A dolgozat 5. fejezete a mobilalkalmazás működését mutatja be annak architektúráján és az általa nyújtott funkciókon át ismertetve, a mobilalkalmazás létfontosságát jelentő, kamerával történő adatbevitelt.

A 6. fejezetben az alrendszerek fejlesztését megkönnyítő folyamatos integrációhoz és az üzembe helyezéshez felhasznált technológiák ismertetése olvasható.

A dolgozat a következtetések és a továbbfejlesztési lehetőségek összegzésével zárul.

A projekt 2024-ben Kolozsváron indult a Codespring mentorprogram<sup>4</sup> keretein belül. A rendszer fejlesztése a nyári szakmai gyakorlat során kezdődött el, ahol a szerzők mentoraik irányítása mellett dolgoztak az alkalmazáson. Később, az egyetemi csoportos projekt tantárgy keretein belül a fejlesztéshez Brînda Krisztián, Kovács Álmos, Szász Csaba-Nándor és Tunya Lénárd Sándor csatlakozott.

A szerzők ezúton is köszönetüket fejezik ki mentoraiknak, Farkas Eszternek és Petkes Kingának, szakmai koordinátoraiknak, Bándi Nándornak és dr. Sulyok Csabának, a projektben részt vevő fejlesztőknek, valamint a Codespring-nek a lehetőségért és a támogatásért.

---

<sup>4</sup><https://edu.codespring.ro>

# 1. Projekt leírása

A B.o.B. projekt négy fő részből áll: a web- és mobilalkalmazás, amely kiszolgálja a felhasználó kéréseit; egy lokálisan futó, nagy nyelvi modellre épülő kérdésmegválaszoló mikroszolgáltatás, illetve a központi szerver, amely felhasználói felületekről érkező kéréseket szolgálja ki kapcsolatot létesítve a feltett kérdésekre választ adó mikroszolgáltatással.

Az alkalmazás fő célja a felhasználói kérdések minél pontosabban történő megválaszolása, a feltöltött állományok optimális kezelése, illetve a felhasználói élményt nyújtó funkcionalitások biztosítása. Az alábbi fejezet célja a fő funkcionalitások és az architektúrát alkotó alrendszerek közti kapcsolatok bemutatása, tárgyalása.

## 1.1. Funkcionalitások

Az alkalmazás számos funkcionalitással rendelkezik, amelyek elengedhetetlenek egy chatfelület szempontjából. Emellett néhány kiegészítő funkcionalitást is tartalmaz, amelyek egyszerűbbé teszik mind a beszélgetések folyamatát, mind az állományok kezelését.

Funkcionalitás szempontjából az alkalmazás két fő részre osztható. Az első csoportba a beszélgetések lebonyolítását segítő funkcionalitások tartoznak, melyek a következők:

- Új, üres beszélgetések létrehozása, amelyekben a felhasználó felteheti kérdéseit.
- Korábban létrehozott beszélgetések listázása időrendi sorrendben, amelyekhez törlési és visszalépési opció is tartozik.
- Mind az új, mind a már meglévő beszélgetésekhez állományok csatolhatók egy fájlfeltöltési opció segítségével.
- Minden beszélgetéshez tartozik egy állománykezelő ablak, ahol a felhasználó kiválaszthatja a felhasználni kívánt állományokat. Az ablak lehetőséget biztosít a már csatolt állományok szűrésére, így könnyebben áttekinthető, hogy a chatbot milyen fájlokból dolgozik.
- A chatbot emlékszik az adott beszélgetésben elhangzott kérdésekre és válaszokra, így azokat figyelembe veszi az új válaszok megfogalmazása során.
- A felhasználó értékelheti és kimásolhatja a bot válaszait.

- Minden válasz esetén a felhasznált kontextus megjelenítésre kerül.

Ezen túlmenően az alkalmazás rendelkezik egy globális állománykezelő rendszerrel, amely az állományok rendszerezését és kezelését szolgáltatja. Az ehhez kapcsolódó funkcionalitások:

- Mappák létrehozása az állományok kézenfekvő rendszerezésére.
- Keresési lehetőség a feltöltött állományok és mappák között egy keresősáv segítségével.
- Feltöltött fájlok és mappák törlése. Mappa törlése esetén annak teljes tartalma is eltávolításra kerül.
- Navigáció a mappák között, beleértve az azokba történő be- és kilépést.

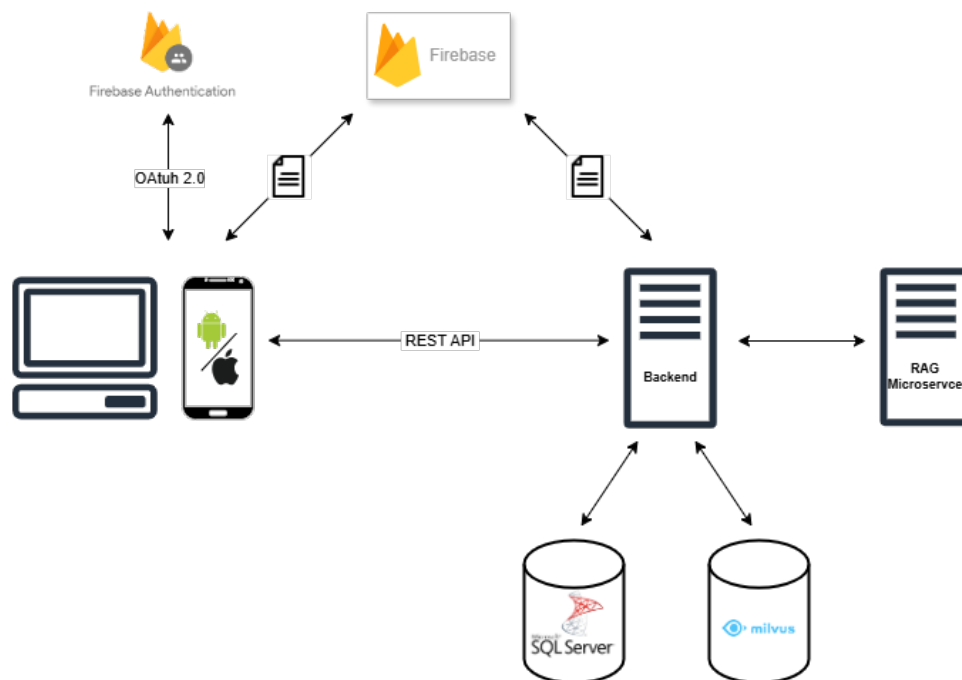
A regisztrációhoz és bejelentkezéshez az alkalmazás kétféle lehetőséget kínál:

- Klasszikus regisztráció, amely során a felhasználó egy felhasználónév, e-mail cím és jelszó megadásával hozhat létre egy fiókot, majd ezt követően bejelentkezhet a létrehozott fiókkal.
- Google-fiókkal történő belépés, amely lehetővé teszi, hogy a felhasználó egy már meglévő Google-fiókjával egyetlen lépésben regisztráljon és/vagy belépjen.

## 1.2. Architektúra

A rendszer főbb részei közé sorolhatóak az alábbiak:

- *RAG mikroszolgáltatás*: A kérdések nagy nyelvi modellel történő megválaszolását és kontextust jelentő állományok feldolgozását biztosítja.
- *Szerver*: A rendszer központi eleme, ez a rész felelős a kliensek kiszolgálásáért és az adatbázisok illetve a RAG mikroszolgáltatással való kapcsolatfenntartásért.
- *Web- és mobilkliens*: A web- és az Android, IOS cross-platform mobilkliens egy felületet biztosít a felhasználók számára, amelyen keresztül interakcióba léphetnek a rendszerrel. Ezek mellett első sorban ez a komponens létesít kapcsolatot a Firebase Authentication és Storage rendszerével.



1. ábra. A rendszer fő alkotóelemei

- *Firestore*: Az állományok tárolását a Firestore Storage rendszer biztosítja, amely lehetővé teszi az állományok felhőben való tárolását és ezek zökkenőmentes elérését. Emellett a Firestore Authentication rendszer által valósul meg a Google fiókkal rendelkező felhasználók bejelentkeztetése is.
- *Adatbázisok*: A felhasználói adatok eltárolásáért egy Microsoft SQL Server adatbázis felel. A RAG módszerhez szükséges gyors kontextuskeresést egy Milvus vektoradatbázis támogatja.

## 2. A RAG módszer és annak alkalmazása

Ezen fejezet a Retrieval Augmented Generation (röviden RAG) módszer működésébe és annak a projektbeli megvalósításába ad betekintést. A fejezet a módszer felvezetésével indít, amely után a főbb architektúrális komponensek, felhasznált technológiák kerülnek bemutatásra. A fejezet végén a módszer megvalósításának összegzése és egy példabeszélgetés olvasható.

### 2.1. Retrieval Augmented Generation

A Retrieval Augmented Generation [4] egy természetes nyelvfeldolgozásra<sup>5</sup> alapuló módszer, mely megoldást próbál nyújtani a nagy nyelvi modellek (LLM<sup>6</sup>-ek) hiányosságaira.

Az általános feladatokra tanított nagy nyelvi modellek nem biztos, hogy egy konkrét feladatkörben kellően elvárt teljesítményt nyújtanak. Ezek újratanítása, jól megtervezett finomhangolás nélkül, kockázatos lehet a már meglévő tudásuk elvesztése miatt. Emellett ezen módszerek erőforrásigénye rendszerint magas. A RAG ezt a problémát külső források (szöveget tartalmazó állományok, a továbbiakban állományok) kontextusként történő felhasználásával oldja meg egy nagy nyelvi modellt használva a kérdések megválaszolásához.

Az LLM-ek bemenetének limitált mérete miatt szükséges a természetes nyelvfeldolgozásra alapuló módszerek használata egy felhasználói kérdéshez tartozó releváns kontextus megtalálásához. Emiatt szükségessé válik a forrásállományok szövegének vektorbeágyazásává<sup>7</sup> való átalakítása.

Egy vektorbeágyazás egy szövegrész numerikus reprezentációja, amely kódolja annak jelentését. Két adott vektorbeágyazás esetén a beágyazásoknak megfelelő vektorok távolságának mértéke jellemzi a nekik megfelelő szövegrészek hasonlóságát.

Az állományok vektorbeágyazássá történő átalakítását azok előfeldolgozása előzi meg. Az előfeldolgozás részének tekinthető például a képekből, táblázatokból való szövegrészek kinyerése. Illetve ide tartozik még a szövegek kisebb szövegrészekre való tagolása, ügyelve a tagolás által elvesztett információmennyiség minimalizálására.

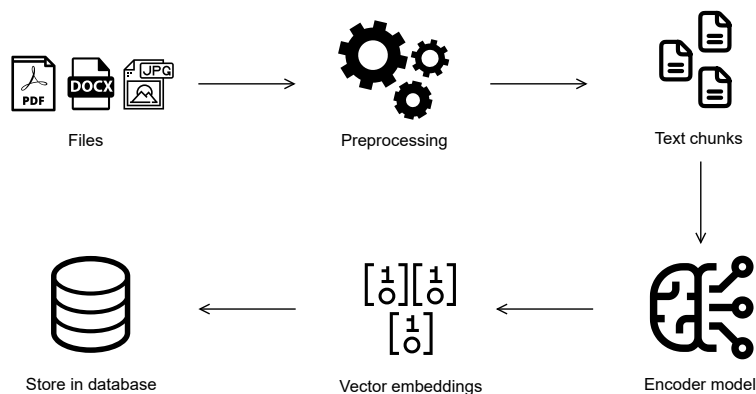
Az előfeldolgozást követve a szövegrészeket egy enkóder modell alakítja át vektorbeágyazásokká. Az enkóder modellek olyan nagy nyelvi modellek, melyek a transzformer architektúrából [9] csak az enkóder komponenst tartalmazzák.

---

<sup>5</sup>Natural Language Processing, röviden NLP

<sup>6</sup>Large Language Model

<sup>7</sup>vector embedding



2. ábra. Az állományok feldolgozásának folyamata a RAG módszer során [6]

A vektorbeágyazásokhoz tartozó vektorok perzisztens eltárolásához szükséges valamilyen vektoradatbázis használata. A vektoradatbázisok biztosítják a nagy dimenziós vektorok és a hozzájuk rendelt metaadatok optimális eltárolását, lehetőséget nyújtva az ezek közötti gyors keresésre. A keresés rendszerint valamilyen távolságfüggvény, metrika, alapján történik.

A felhasználó kérdésének megválaszolását a kérdés szövegének vektorbeágyazássá történő átalakítása előzi meg. Ezt a folyamatot a forrásállományok tartalmát vektorbeágyazássá alakító enkóder modell végzi.

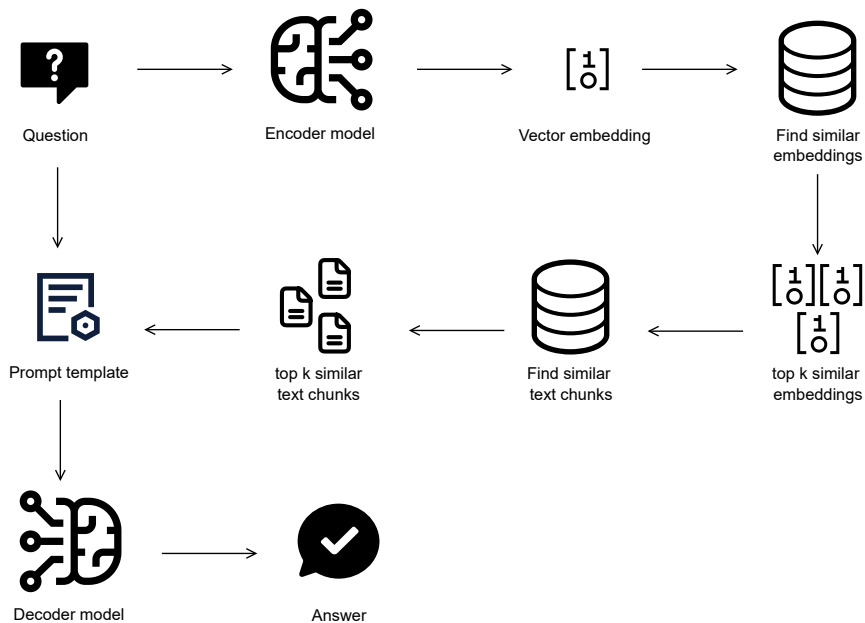
A felhasznált nagy nyelvi modellnek kontextusként szolgál a felhasználói kérdéshez legjobban hasonló  $k$  szövegrész. Ezen szövegrészekhez rendelt vektorok meghatározhatók a vektoradatbázisban történő kereséssel a generált beágyazás alapján. A talált vektorokhoz tartozó szövegrészek például a vektorokhoz rendelt metaadatokból, vagy valamilyen más adatbázisból a hozzájuk rendelt egyedi azonosítók alapján határozhatóak meg.

A felhasználói kérdéshez tartozó kontextus a kérdéssel együtt egy prompt-sablon formájában adódik át bemenetként a felhasznált nagy nyelvi modellnek, amely válaszol a feltett kérdésre.

A RAG módszer folyamatának szemléltetése állományok feldolgozása és kérdésmegválaszolás esetén rendre a 2. és 3. ábrákon látható.

## 2.2. Nagy nyelvi modellek

A nagy nyelvi modellek felépítése a transzformer-architektúrából [9] indul ki. Ezen architektúrát eredetileg az eltérő nyelvek közötti szövegfordításra tervezték. A forrásnyelvben megadott, lefordítandó, szövegből az enkóder komponens vektorbeágyazást generál, mely a



3. ábra. A felhasználói kérdés megválaszolásának folyamata a RAG módszer során [6]

forrásszöveg jelentését kódolja. A dekóder komponens a forrásszöveget fordítja le szöveget generálva a célnyelvben. A fordítás során dekódolva az enkóder komponens kimenetét javítja a generált, lefordított, szöveg minőségét.

Az enkóder- és dekóder nagy nyelvi modellek a transzformer architektúrából rendre csak az enkóder- és dekóder komponens tartalmazzák. A közhasználatban levő chatbotok dekóder modelleket használnak a válaszok generálásához.

A projektben felhasznált nagy nyelvi modellek futtatását és vezérlését az Ollama<sup>8</sup> eszköz végzi. Ez egy a nagy nyelvi modellek működését orkesztráló eszköz, amely megkönnyíti a nyilvánosan elérhető, előre betanított, modellek letöltését és izolált futtatását. Az eszköz emellett egy API-t nyújt segítve a modellek programkódból való futtatását különböző könyvtárak által<sup>9</sup>.

### 2.2.1. Enkóder modell

Enkóder modellként a RAG módszer megvalósításában az all-MiniLM-L6-v2 modell lett felhasználva, amely megközelítőleg 22,7 millió paraméterrel rendelkezik. Ezen modell a MiniLM-L6-v2 [10] modell továbbfejlesztett, finomhangolt változata, amely rövid, angol

<sup>8</sup><https://ollama.com/>

<sup>9</sup>Az Ollama eszközzel való kommunikációt biztosító könyvtár forrása: <https://www.langchain.com/langchain>

nyelvű szövegrészek jelentéshű kódolására alkalmas leginkább<sup>10</sup>.

A fenti enkóder modell kis, 384 dimenziós vektorokat állít elő vektorbeágyazásként. Ezáltal elősegíti a RAG rendszerben a gyors információ-előhívást, az eltárolt vektorbeágyazások közti hasonlóságon alapuló keresést.

### 2.2.2. Dekóder modell

Az alkalmazás fejlesztése közben az egyik legnagyobb kihívást a megfelelő minőségű válaszokat adó dekóder modell megtalálása jelentette. Több modell kipróbálásába telt a jelenlegi válaszminőség elérése.

A kezdeti prototipizálási fázisban a lokális futtathatóság céljából az OpenAI<sup>11</sup> cég által betanított GPT2 [5] modell volt használatban, konkrétan a 137 millió paraméteres verzió. Később, erősebb hardver elérhetőséggel, lehetőség nyílt a 380, 812 millió és 1,61 milliárd paraméteres változatok kipróbálására is. Viszont ezek közül egyik verzió sem hozta a kívánt válaszminőséget: a generált szöveg inkább hasonlított valamilyen külső forrásból történő szó szerinti idézésre, mintsem a feltett kérdéseknek válaszára.

A fejlesztés további szakaszában, a megfelelő hardverhez való hozzáféréssel, a Meta<sup>12</sup> cég által létrehozott Llama 2 [7], 7 milliárd paraméteres modell került használatba. A modell hozta az elvárt eredményeket: a feltett kérdésekre beszédszerű válaszokat adott beépítve a kontextusként kapott szövegrészeket azokba.

Ugyan a Llama 2 modell hozta az elvárt válaszminőséget a beszélgetések során, viszont a modell viselkedése nem volt annyira kontrollálható. Ez először a beszélgetések, azok tartalma alapján történő, címgenerálásánál okozott gondot: a generált szöveg hosszát nem lehetett annyira szabályozni, illetve néha túl kreatív címek születtek, melyek nem igazán kötődtek az adott beszélgetéshez.

Ezek alapján esett a választás a Mistral AI<sup>13</sup> cég által betanított 7,3 milliárd paraméterrel rendelkező Mistral [3] modellre, konkrétan annak az utasításkövetésre finomhangolt változatára. A modellt a készítői a Llama 2 modell vetélytársaként fejlesztették, javítva annak teljesítményén. Ezáltal a modell utasításkövető változata, különböző benchmark-okat tekintve, a Llama 2 13 milliárd paraméterű változatát múlja felül.

---

<sup>10</sup><https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2>

<sup>11</sup><https://openai.com/about/>

<sup>12</sup><https://about.meta.com/>

<sup>13</sup><https://mistral.ai/about>

A projekt jelenlegi állapotában a Mistral 7,3 milliárd paraméterrel rendelkező, utasításkövetésre finomhangolt, modell van használatban.

## 2.3. Vektoradatbázis

A RAG módszer kulcsfontosságú eleme a kérdéseket megválaszoló nagy nyelvi modell mellett egy optimalizált vektoradatbázis, amelyben a potenciális kontextust jelentő szövegrészeket numerikusan reprezentáló vektorbeágyazások vannak eltárolva.

A fejlesztési időszak első felében a Chroma<sup>14</sup> vektoradatbázis volt használatban, mely egy könnyen használható collection alapú perzisztens vektortárolót biztosított. Az adatbázis programkódból könnyen kezelhető volt, viszont nem támogatta a konkurens kérésfeldolgozást a vektorok lassú eltárolását eredményezve.

Erre a problémára jelentett megoldást a Milvus<sup>15</sup> vektoradatbázis használata. Ezen adatbázis a konkurens kérések támogatása mellett skálázhatóságot, több adat- és indextípust biztosít. Az adatbázis futtatásának részletesebb leírása a 6. alfejezetben olvasható.

A Milvus vektoradatbázis számos metrikát támogat a vektorok összehasonlításához. Ezek közül a koszinusz hasonlóság került használatba, melynek képlete:

$$\text{sim}(\mathbf{x}, \mathbf{y}) \stackrel{\text{def}}{=} \cos(\theta(\mathbf{x}, \mathbf{y})) = \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{x}\| \cdot \|\mathbf{y}\|}, \quad (1)$$

ahol  $\| - \| : \mathbb{R}^d \rightarrow \mathbb{R}$  a vektorok euklideszi normája és  $\langle -, - \rangle : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$  a vektorok skaláris szorzata.

A Milvus adatbázis a Chroma adatbázishoz hasonlóan collection-ökre épül. Az alkalmazás rendszerén belül minden feltöltött állománynak egyetlen vektoradatbázisbeli collection felel meg. Egy collection-ben a vektorbeágyazások a hozzájuk tartozó szövegrészekkel, a szövegrészek forrásállományának egyedi azonosítójával, illetve a szövegrész helyét leíró oldalszámokkal csoportosítva vannak eltárolva.

Az 1. képlet által leírt metrika egy adott vektorhoz a  $k$  legjobban hasonlító vektor keresésben használt. A leghasonlóbb "szomszédos" vektorok megtalálása túlságosan időigényes feladat sok vektor esetén. Erre a problémára nyújtanak megoldást a különféle adatbázisindexek, melyek a találatok pontosságát bizonyos mértékben feláldozva gyorsítanak nagymértékben a keresésen.

---

<sup>14</sup><https://www.trychroma.com/>

<sup>15</sup><https://milvus.io/>

A Milvus adatbázis indexei közül az IVF\_FLAT<sup>16</sup> adatbázisindex került alkalmazásra, mivel az tömörítés nélküli tárolás révén jobb pontosságot biztosít elfogadható memóriahasználat mellett.

Az IVF\_FLAT index az eltárolt vektorokat klaszterekbe sorolja kiszámolva azok centroid-ját. A centroid-ok számát az `nlist` paraméter írja le. Egy adott vektorhoz a `k` legközelebbi "szomszédos" vektor megtalálása először `nprobe` számú legközelebbi centroid megtalálásával kezdődik. Ezután a megtalált centroid-oknak megfelelő klaszterek vektorai közül kerül kiválasztásra az adott vektorhoz `k` legközelebbi vektor.

A projektbeli megvalósításban a `k = 5`, `nlist = 128` és `nprobe = 10` paraméterértékek vannak használatban.

## 2.4. Optical Character Recognition

A szöveges állományok mellett a megvalósított RAG módszer fotózott szöveg felismerésére is képes. Ennek megvalósítása az Optical Character Recognition (röviden OCR) módszerrel történik, mely a feltöltött képekből a rajtuk levő szöveg kinyerését biztosítja.

OCR modellként az előre betanított pehelysúlyú, konvolúciós neurális hálóra épülő, a JaideAI<sup>17</sup> cég által létrehozott nyílt forráskódú EasyOCR<sup>18</sup> modell került használatba, amely elérhető programkönyvtár formájában is<sup>19</sup>.

## 2.5. A RAG módszer megvalósítása

Az alábbi alfejezetben a saját megvalósítások kerülnek tárgyalásra a korábbi alfejezetekben bemutatott komponensekre, technológiákra alapozva.

### 2.5.1. Architektúra

A felépített RAG rendszer egy több endpoint-ból álló mikroszolgáltatásra és egy Milvus adatbázisszerverre épül. A RAG rendszer által nyújtott főbb szolgáltatások az ezen alrendszerrel kommunikáló serveroldali komponensek által kerültek megvalósításra. A serveroldali komponensek az alfejezet további részében kerülnek bemutatásra.

Az említett szolgáltatások közé sorolhatóak: a kérdésmegválaszolás; állományok kereshetőbb formában történő eltárolása és a releváns kontextus megtalálása ezekből a feltett

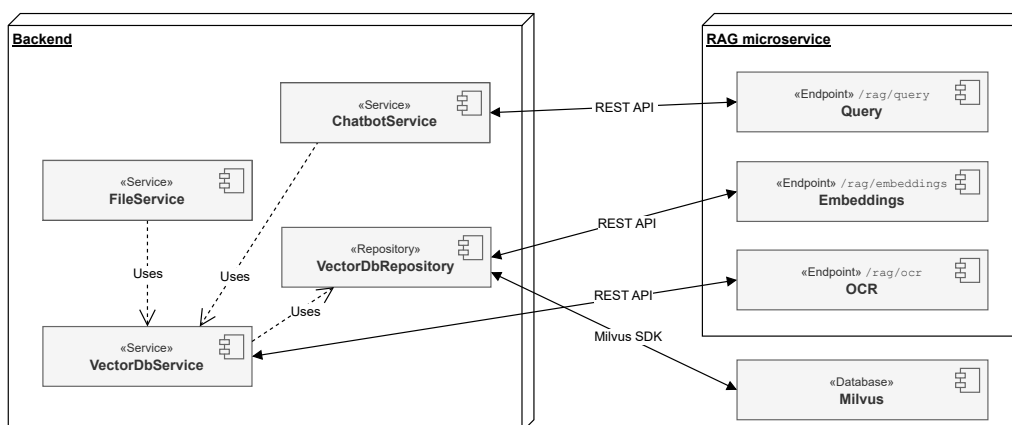
---

<sup>16</sup><https://milvus.io/docs/ivf-flat.md>

<sup>17</sup><https://www.jaide.ai>

<sup>18</sup><https://github.com/JaideAI/EasyOCR>

<sup>19</sup><https://pypi.org/project/easyocr/>



4. ábra. A RAG rendszer architektúráis felépítése

kérdés alapján, illetve a képekből történő szövegkinyerés.

A RAG mikroszolgáltatás az alábbi komponensekből épül fel, melyek viszonyát a 4. ábra illusztrálja:

- *Query* endpoint: A felhasználói kérdéseket válaszolja meg a dekóder modell által, az Ollama LLM orkesztráló eszközt használva. Működéséről részletesebb kifejtésre a 2.5.3. alfejezetben kerül sor.
- *Embeddings* endpoint: A vektorbeágyazások generálásáért felelős szintén az Ollama eszközt használva, az enkóder modellt futtatva. Ezen erőforrás az állományfeldolgozásnál lényeges. A felhasznált enkóder modellről részletesebb leírás a 2.2.1. alfejezetben olvasható.
- *OCR* endpoint: Egy feltöltött képben rejlő szöveges információ kinyerésére alkalmas. Működéséről részletesebb leírás a 2.4. alfejezetben olvasható.

A Milvus adatbázis a vektorbeágyazások és hozzájuk rendelt metaadatok perzisztens eltárolását és a köztük történő hatékony keresést szolgáltatja. Ezen komponenssel a backend szerver a Milvus SDK<sup>20</sup>-ra épülő Python programkönyvtár felhasználásával kommunikál. Részletesebb leírása a 2.3 alfejezetben található.

A mikroszolgáltatás endpoint-jai egy Hypercorn<sup>21</sup> aszinkron HTTP szerverre épülnek, melyek a FastAPI<sup>22</sup> programkönyvtár segítségével kerültek megvalósításra. A RAG módszer

<sup>20</sup><https://pypi.org/project/pymilvus/>

<sup>21</sup><https://pypi.org/project/Hypercorn/>

<sup>22</sup><https://fastapi.tiangolo.com/>

megvalósításában résztvevő backend szerver komponensei HTTP RESTful API-on keresztül érik el a mikroszolgáltatás endpoint-jait.

### 2.5.2. Fájlfeldolgozás

A B.o.B. alkalmazás számos típusú szöveges állományt (TXT-, DOCX-, PDF-fájlok), illetve képfájlt (PNG, JPG, JPEG-formátumok) támogat. A továbbiakban, az egyszerűség kedvéért, fájl alatt az említett kiterjesztésekkel rendelkező állományok értendők.

A fájlfeldolgozás célja a feltöltött fájlok tartalmának kinyerése és annak eltárolásra alkalmas, kereshető formára hozása. Ezen folyamat a felhasználói kérdéseket megválaszoló LLM korlátozott bemenetmérete miatt lényeges: a felhasználó fájljai nagy méretűek is lehetnek, viszont a nagy nyelvi modell bemeneti token-száma korlátozott.

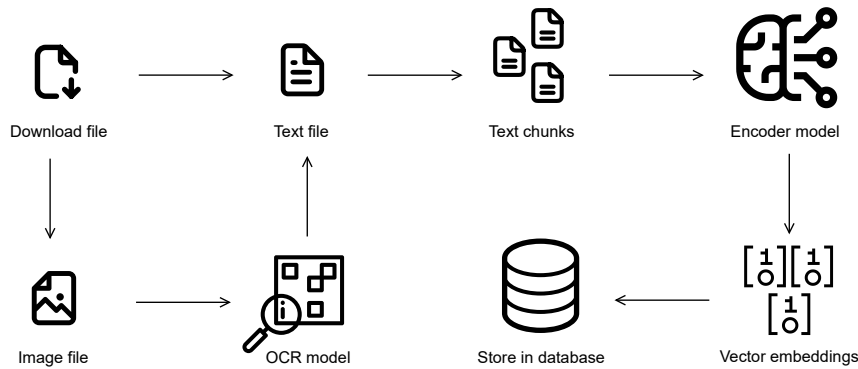
A fájlfeldolgozás folyamata előfeldolgozásból és azt követő feldolgozott szöveges tartalom eltárolásából áll. Az állományok szöveges tartalmának hatékonyan kereshető formában történő eltárolása teszi szükségessé egy vektoradatbázis használatát. A vektoradatbázis működésének részletesebb leírása a 2.3. alfejezetben olvasható.

Az alfejezet további részében az előfeldolgozási fázis kerül részletesebb bemutatásra. A fájlok feldolgozását a háttérben a backend szerver végzi. Ezen fázis a felhasználó által újonnan feltöltött fájl letöltését követően kezdődik el.

A letöltött fájl előfeldolgozása a fájlból történő szöveg kinyerésével kezdődik. Amíg egyszerű szöveges állományok esetén ez a folyamat egyenértékű az állományból történő szöveg beolvasásával, addig a képfájlokból való szöveges információ kinyerése egy OCR modell használatát igényli. A felhasznált OCR modell bemutatása a 2.4. alfejezetben olvasható.

Mivel a fájlok nagy méretűek lehetnek, ezért azok memóriába történő egyszeri beolvasás nem optimális. Így, szöveges állományok esetén, fájlformátumtól függően, a szöveg beolvasása iteratívan történik minimálisan beolvasható szövegrészek memóriába való betöltésével. Minimális szövegrész alatt értendők: PDF-fájlok esetén az oldalak, DOCX-fájlok esetén a paragrafusok, illetve TXT-fájlok esetén a sorok.

Az előbb említett minimálisan beolvasható szövegrészek, azonos méretű szövegrészek kialakításának céljából, hosszuk szempontjából egységesítésre szorulnak. Ez lényeges a felhasznált LLM bemenetméretének hatékony kihasználása érdekében. Emellett, elkerülve a szöveg részekre való bontásából származó nagymértékű tartalomvesztést, szükséges biztosítani rögzített méretű átfedést az azonos méretű szövegrészek között.



5. ábra. A fájlfeldolgozás folyamata egy újonnan feltöltött állomány esetén [6]

Előzetes, megfelelő külső könyvtárak és függőségek hiányában, melyek mind az optimális fájlbeolvasást és a szöveg megfelelő részekre történő bontását egyaránt biztosították volna, szükségessé vált a fájlokat iteratíván beolvasó és az előfeldolgozást elvégző fájliterátorok implementálása. Ezek fájltypusoknak megfelelően, iteratíván beolvasva a szöveget szöveges állományok esetén, a fentebb megszabott feltételeknek megfelelő szövegrészeket térítenek vissza. Az említett tulajdonsággal rendelkező szövegrészeket egy sliding window algoritmus biztosítja. Az algoritmus két "mozgó ablakot" nyújt a szövegbe, kétvégű várakozási sorok formájában. Ezáltal a szövegrészek kialakítását lineáris futási időben biztosítja.

Az említett iterátorok alól a képiterátor jelent kivételt, amely egy REST API híváson keresztül küldi el egy feltöltött kép tartalmát az OCR endpointnak, ami a képen levő szöveget téríti vissza. A visszatérített szöveg előfeldolgozása az előzőleg említett iterátorok által biztosított eljárással zárul.

A fájliterátorok az említett funkcionalitás mellett, a fájltypusnak megfelelően, ellátják a szövegrészeket két oldalszámmal, melyek leírják azok pontos helyét a forrásfájlban belül. Az oldalszám fogalma PDF-fájlok esetén értelmezett leginkább: DOCX-fájlok esetén, azok render-elése miatt, az oldalszámozás nem jóldefiniált; képek és TXT-fájlok esetén a tartalom egy oldalon található.

Az előfeldolgozás a `VectorDbService` komponensben került megvalósításra, ami a `VectorDbRepository` komponensre alapoz. A mélyebb rétegen rejlő komponens kommunikál a használatban levő vektoradatbázissal. Az említett szolgáltatáson belül, egy háttérzál felhasználásával, aszinkron kommunikáció útján kerülnek elküldésre a feldolgozott szövegrészek a vektoradatbázishoz eltárolásra. Ezen megoldás nagymértékben felgyorsítja a fájlok feldolgozásának idejét. Az említett komponensek és a RAG mikroszolgáltatás viszonyát

a 4. ábra illusztrálja.

A PDF-fájlok oldalankénti beolvasása a pdfminer<sup>23</sup> programkönyvtárral került megvalósításra.

### 2.5.3. Cím- és válaszgenerálás beszélgetésekben

A RAG rendszer egy kulcsfontosságú része a felhasználói kérdésekhez tartozó válaszok szövegének generálása. Emellé a folyamat mellé sorolható még a beszélgetéseket jellemző címek generálása is.

A lokálisan futó LLM által történő szöveggenerálás promptsablonokkal van szabályozva. Ez egy köztes, olcsó erőforrásigényű, megoldást nyújt a finomhangolással és az inferenciával<sup>24</sup> elért szövegminőség között. A prompt-sablonok a bemenetet jelentő szövegrészeknek egy közös kontextust adnak, ezáltal tudatva a LLM-mel az elvárt, generálandó, szöveg jellegét.

A felhasználó által feltett kérdések megválaszolásához rendelt promptsablon a nagy nyelvi modellel tudatja annak feladatát a válaszolás során betartandó szabályokat rögzítve: interperszonális kommunikáció viszonzása, kérdésmegválaszolás csak a megadott kontextus és beszélgetés-előzmény alapján, udvarias hangvétel. Ezt a kontextus, az aktuális beszélgetés előzménye és a felhasználó kérdése követi.

A generált cím az első felhasználói üzenet és a hozzá rendelt kontextus összegzése. Ennek generálása a válaszgeneráláshoz hasonlóan történik külön promptsablont használva. Ezen sablon szintén a feladatspecifikus szabályok tisztázásával kezdődik: a felhasználói üzenet és a hozzárendelt kontextus összegzésére való felhívás. Ezt kontextuális, néhány megadott példa-interakcióra épülő, tanulás követi <sup>25</sup> [2], amely 5 példát tartalmaz címgenerálásra szabályozva ezáltal a generált szöveg jellegét. A promptsablon az első felhasználói üzenettel és annak kontextusával zárul.

Az Ollama orkesztrációs eszközön futtatott LLM-el történő interakciókat a LangChain könyvtár biztosítja. Ez lehetővé teszi a korábban említett promptsablonokba a szövegrészek beszúrásával a szöveggenerálást, illetve az LLM bizonyos hiperparamétereinek szabályozását.

A szöveggenerálásnál a hiperparaméterek közül a hőmérséklet és a generálandó token-ek maximális száma a mérvadó. Az 1-nél nagyobb hőmérsékletértékek esetén az LLM kreatívabb válaszokat ad. Ezért, a generált szöveg stabilitására való tekintettel, a LangChain könyvtár

---

<sup>23</sup><https://pypi.org/project/pdfminer/>

<sup>24</sup>A kérdés szövegének az LLM-nek bemenetkénti megadása

<sup>25</sup>Szakirodalomban few-shot learning-ként olvasható.

korlátozza ezt a  $[0, 1]$  intervallumra. A fentebb említett paraméterek esetén a cím- és válaszgenerálásnál rendre a 0, 9 és 256, illetve a 0, 9 és 20 értékek vannak használatban.

A tárgyalt funkciók architekturális szempontból a RAG mikroszolgáltatásában kerültek megvalósításra, amely REST API hívásokon keresztül kommunikál a backend szerver ChatbotService szolgáltatásával. A szolgáltatás és a mikroszolgáltatás viszonyát a 4. ábra illusztrálja.

#### **2.5.4. Beszélgetések memóriája**

A RAG módszer előző alfejezetekben bemutatott megvalósításnak egy legfőbb hiányossága az, hogy az inkább kérdés-válasz orientált figyelmen kívül hagyva a beszélgetés előző tartalmát, amelyben a felhasználó a kérdését tette fel. Így ezen megvalósítás potenciálisan kevésbé összefüggő beszélgetéseket eredményez.

Sok megoldás létezik az előbb említett memória-probléma kiküszöbölésére. Figyelembe véve a hardver-limitációt és a feltett kérdések megválaszolásának sebességét az alábbi megoldás lett implementálva a projekten belül: az állományokból létrehozott kontextus mellé a beszélgetés korábbi, legtöbb  $m$  rögzített számú üzenete kerül beszúrásra a prompt-sablonba. Ez alapján válaszolja meg a feltett kérdést a nagy nyelvi modell a kérdés kontextusa mellett. A projektbeli implementációban konkrétan az  $m = 6$  paraméterérték van használatban.

A beszélgetés-memória implementációját a ChatbotService szerveroldali szolgáltatás tartalmazza.

## 2.6. Példa a RAG módszer alkalmazására

A továbbiakban egy a chatbothoz folytatott beszélgetés látható. A beszélgetés kontextusát egy előre feltöltött PDF-állomány [8] nyújtja, amely egy, az egészséges táplálkozást támogató, szakácskönyv. Ebből egy, a beszélgetés során felhasznált, szövegrészlet a 6. ábrán olvasható.

A beszélgetés során a felhasználó, köszöntve a chatbotot tanácsot kér egy a hagyományosnak mondható, mogyoróvajastól eltérő, szendvics elkészítésében. A chatbot a kontextusként megadott szakácskönyv alapján válaszol a feltett kérdésre felsorolva néhány hozzávalót.

A felhasználó ezután megkéri a chatbotot, hogy az általa megadott hozzávalók alapján készítsen egy szendvicset. A chatbot válaszolva a felhasználó kérésére részletes leírást nyújt egy szendvicsről felsorolva annak összetevőit és elkészítésének lépéseit. Az üzenetváltás a felhasználó köszönetével és a chatbot udvarias viszonzó válaszával zárul. A leírt beszélgetés a 7. ábrán olvasható.

A chatbot valóban felhasználja kontextusként a 6. ábrán látható szövegrészt. Ezt a 8. ábra igazolja, amelyen megfigyelhetők egy válaszhoz tartozó források között az említett

### Sandwich Ideas

*Sandwiches are a great lunch staple and are very simple and easy to make. Getting bored of plain old peanut butter and jelly? Here are some ideas from USU dietetic students to try to spice up your everyday sandwiches.*

#### Breads:

Bread  
Tortilla  
Pita bread  
Naan bread (or other flat bread)  
Roll  
Bun  
French bread or other artisan breads  
English muffins  
Ciabatta

#### Spread:

Fancy mustard  
Hummus  
Pesto  
Cream cheese (plain or herbed)  
Mayonnaise (plain or flavored)  
Ranch or other salad dressings  
Salsa  
Guacamole  
Mashed beans

#### Protein:

Deli meat  
Grilled chicken  
Cooked egg  
Sausage  
Roasted turkey or other meat  
Tuna  
Cheese  
Beans  
Tofu  
Veggie Burgers  
Egg salad (or chicken salad)  
Lentils

#### Vegetables:

Lettuce  
Spinach



#### Tomatoes

Chopped carrots  
Pickles  
Coleslaw  
Peppers (green, red, yellow, orange, or hot)  
Onions  
Avocado  
Cucumber  
Mushroom  
Sprouts

### Cucumber Sandwiches (makes 4)

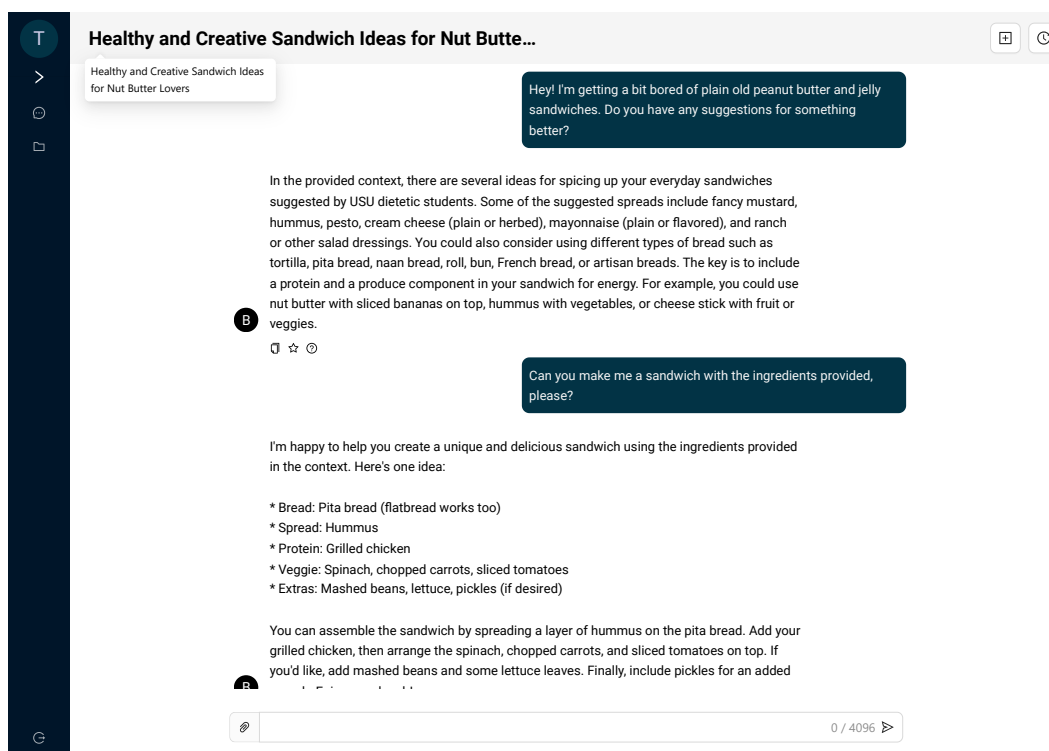
8 thick slices whole-wheat bread  
½ cup (4 oz.) cream cheese, softened (can be plain or flavored as desired)  
1 to 2 English cucumbers sliced into about 24 pieces  
½ cup alfalfa sprouts  
4 tsp. olive oil  
4 tsp. red wine vinegar  
2 tomatoes, sliced  
4 lettuce leaves  
1 cup sliced pepperoncini or banana peppers  
2 avocados, peeled, seeded, and mashed

1. Spread about 1 tablespoon of cream cheese on each bread slice. Arrange the cucumbers in a single layer over 4 bread slices, cover with sprouts, and sprinkle with oil and vinegar.  
2. Layer on the tomatoes, lettuce, and pepperoncini.  
3. Spread avocados over the cream cheese on the remaining 4 slices of bread. Place spread-side down on pepperoncini and serve right away.

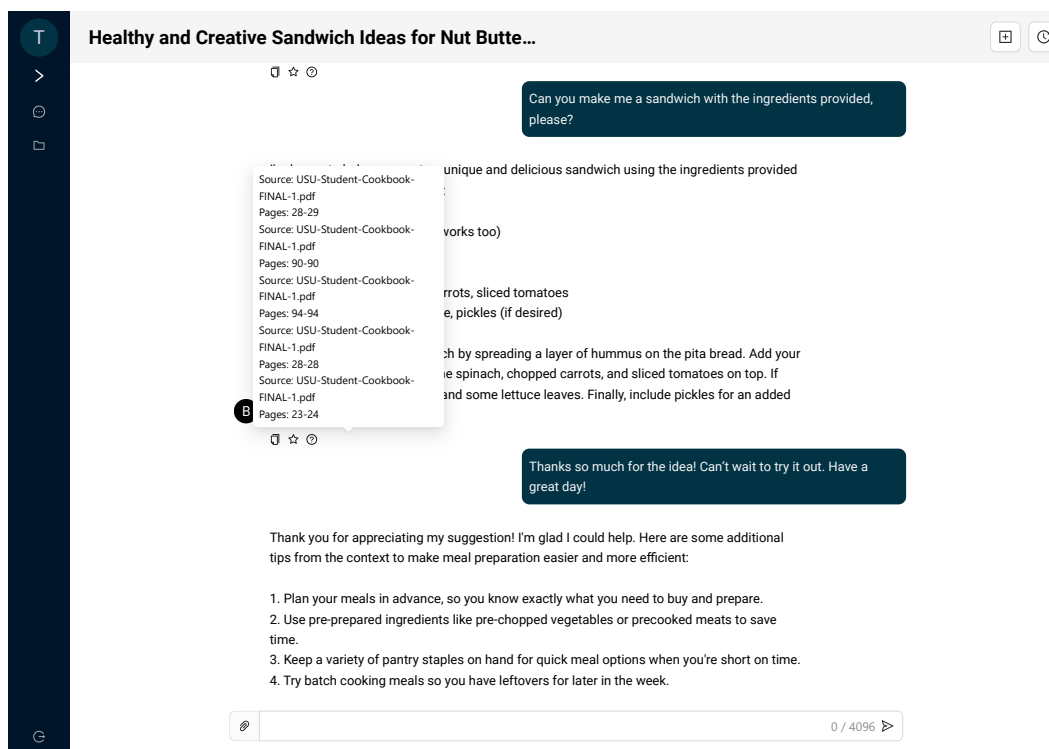
*Recipe adapted from allrecipes.com*

6. ábra. A megadott állományból [8] a beszélgetés során kontextusként felhasznált szövegrészek egyike (28-29. oldal)

szakácskönyvrészletet azonosító oldalszámok (28-29. oldal).



7. ábra. A megadott forrás-állományra [8] épülő beszélgetés, amely során a felhasználó szendvicskészítéshez kér tanácsot a chatbottól



8. ábra. A beszélgetésben történő forrásfelhasználást demonstráló ábra (forrásként megjelenik a 28-29. oldal)

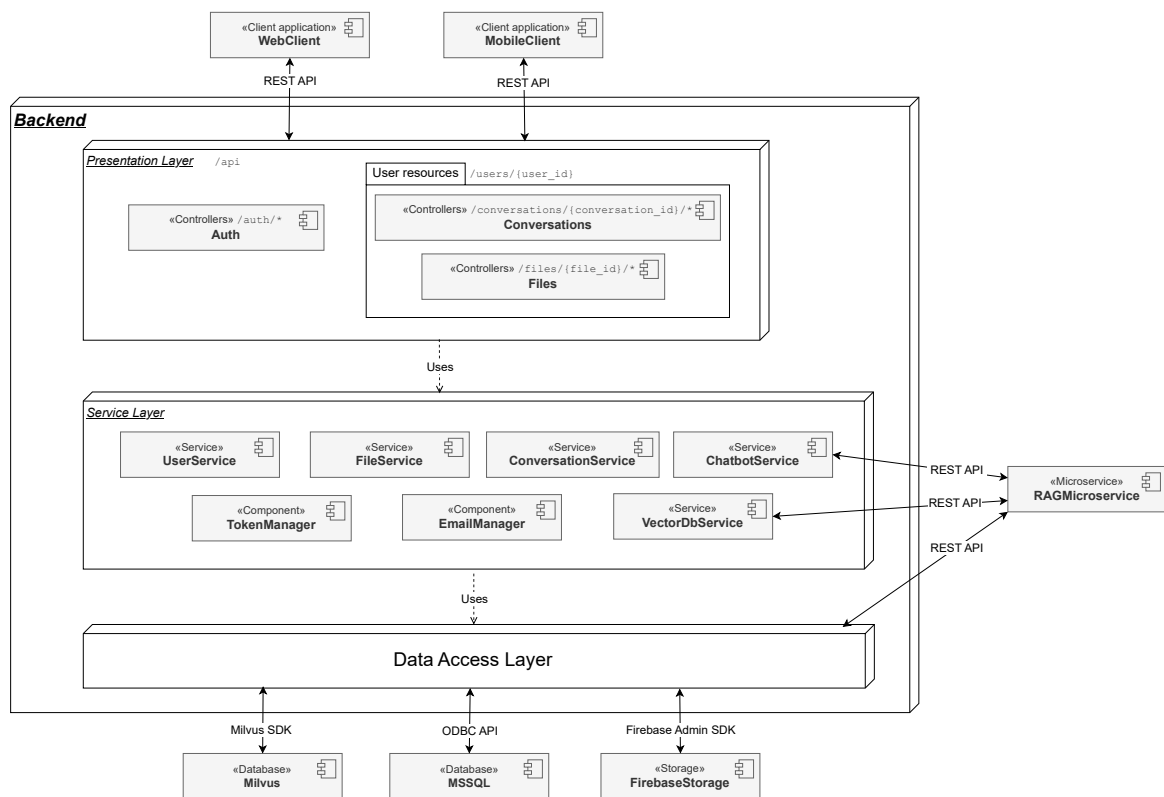
### 3. Szerver

A B.o.B. chatrendszer középpontjában a backend szerver áll. Ezen alrendszer felelős a felhasználói interakciók koordinációjáért összekötve az alkalmazást képező többi alrendszert. A továbbiakban a backend szerver kerül bemutatásra annak felépítésén és az általa nyújtott funkcionalitásokon keresztül.

#### 3.1. Architektúra

A backend szerver felépítése egy három rétegű architektúrán alapszik. Ezen típusú architektúrák célja a szoftver-komponensek szerepköreinek egyszerű módon történő elhatárolása a modularizált fejlesztést elősegítve. Az architektúrát felépítő rétegek a továbbiakban felülről lefele irányuló módon kerülnek bemutatásra. A backend szerver felépítését a 9. ábra illusztrálja.

A kliensalkalmazások a prezentációs réteget jelentő HTTP REST-alapú API-on keresztül kommunikálnak a backend szerverrel. A Model-view-controler (röviden MVC) tervezési



9. ábra. A backend szerver architektúráis felépítése

mintára alapozva, a bejövő HTTP kérések a FastAPI<sup>26</sup> keretrendszer által nyújtott endpoint-okhoz rendelt kontrollerekhez érkeznek meg.

Az említett kontrollerek három főbb csoportba sorolhatók: autentikációs és autorizációs kontrollerek, melyek a felhasználókezelést biztosítják, illetve a felhasználói erőforrásokat kezelő, beszélgetésekhez és fájlokhoz rendelt, kontrollerek.

A reprezentatív adatátvitelt elősegíti a Data Transfer Object (röviden DTO) strukturális tervezési minta használata. Ezen minta a beérkező adatok formátum-ellenőrzését illetve a kimenő, nem releváns adatok elfedését biztosítja.

A prezentációs réteg a szolgáltatási réteg által nyújtott szolgáltatás-interfészeket felhasználva válaszol a beérkező kérésekre. A szolgáltatási réteg ezen interfészekhez nyújt implementációt. A szolgáltatások többsége csak az adatelérési réteggel kommunikál néhány kivétellel: ChatbotService, VectorDbService. Ezek a RAG rendszer felépítésében játszanak fontos szerepet, amelynek bemutatása a 2. fejezetben olvasható.

A szolgáltatási réteg az adatelérési réteggel az által nyújtott interfészeken keresztül kommunikál. Az adatelérési réteg felelős az adatok perzisztens eltárolásáért és azok eléréséért. Ennek bemutatása a 3.2.1. alalfejezetben tekinthető meg.

## 3.2. Funkcionalitások

A backend szerver számos funkcionalitást nyújt, melyek elengedhetetlenek a chatrendszer esetén: a perzisztens adateltárolás és adatelérés, a felhasználó- és állománykezelés, illetve a chat-felületet képező beszélgetések kezelése. A felhasználó által feltöltött állományokra épülő kérdésmegválaszolásnak egy része is ezen alrendszerben került megvalósítására. Ennek részletes bemutatása a RAG rendszer leírásában olvasható külön, a 2. fejezetben.

A továbbiakban az említett funkcionalitások és azok megvalósítása kerül bemutatásra.

### 3.2.1. Adatbázis és adatmodell

Az adatelérési vagy repository réteg biztosítja az adatok perzisztens, redundancia-mentes eltárolását. Így ezen rétegen belül történik meg az alrendszerbeli külső adattárolókkal történő kommunikáció.

A perzisztens adattárolók közé sorolhatóak: a főbb felhasználói adatokat eltároló Microsoft

---

<sup>26</sup><https://fastapi.tiangolo.com/>

SQL Server<sup>27</sup> (röviden MSSQL) relációs adatbázis, a Milvus vektoradatbázis, illetve a Firebase Storage<sup>28</sup> fájl tároló.

A felhasznált vektoradatbázisról részletesebben a 2. fejezeten belül esett szó. A Firebase által nyújtott fájl tárolóval történő kommunikációt a Firebase Admin SDK<sup>29</sup> külső függőség biztosítja. A MSSQL-el történő kommunikáció az Open Database Connectivity (röviden ODBC) API-t implementáló SQLAlchemy<sup>30</sup> Object–relational mapping (röviden ORM) keretrendszer által lett megvalósítva.

Az alrendszer adatmodelljét az említett ORM keretrendszer szinkronizálja a relációs adatbázis sémájával. A backend szerver adatmodelljének a keretrendszertől való függetlenségének megőrzése érdekében, az ORM keretrendszer által nyújtott, a már előre létrehozott adatbázisséma leképezéséből kapott, adatmodell-objektumokat az alrendszer adatmodell-osztályai foglalják magukba rejtett attribútumként. Ez lehetőséget biztosít az adatmodell szerveroldali független tovább bővítésére.

Az említett ORM keretrendszer ugyan a háttérben elvégzi az adatbázisban eltárolt entitások életciklusának vezérlését, viszont nem nyújt alap-implementációt az entitásokkal történő CRUD műveletekre, ellentétben más programozási nyelvek által nyújtott keretrendszerekkel. Ezáltal merült fel a repository réteg saját kezű megvalósításának igénye a keretrendszer által biztosított lekérdezéseket felhasználva.

Az adatelérési réteggel a szolgáltatási réteg az Abstract DAO (Data Access Object) Factory tervezési mintát követve "kommunikál". Ezen minta segíti elő a magasabb absztrakciós szinten levő rétegek függetlenítését az adatelérési réteg műveleteinek implementációjától.

### 3.2.2. Felhasználókezelés

A backend szerver által nyújtott felhasználókezelés autentikációból és autorizációból áll. Az autentikáció egy adott felhasználó azonosítását, míg az autorizáció az azonosított felhasználó jogosultságainak meghatározását jelenti.

A szerver jelen pillanatban egyetlen szerepkört biztosít a felhasználók számára, ami az alkalmazás teljesértékű használatát biztosítja.

A felhasználói hitelesítés JSON Web Token (röviden JWT) alapú. Ezen token-ek kis méretű információ eltárolására és annak egy aláírással, illetve egy létrehozási dátummal való, titkosítás

---

<sup>27</sup><https://www.microsoft.com/en-us/sql-server/sql-server-2019>

<sup>28</sup><https://firebase.google.com/docs/storage>

<sup>29</sup><https://pypi.org/project/firebase-admin/>

<sup>30</sup><https://www.sqlalchemy.org/>

nélküli, felruházására alkalmasak. Ezáltal a felhasználó identitását meghatározó minimális információ relatív biztonságos kliensoldali-eltárolását és szerveroldali gyors feldolgozását segíti elő.

A szerveroldali hitelesítés access-refresh JWT token-párok használatával történik, a felhasználói munkamenetnek lejáratí időt adva. Ezáltal csak a felhasználóhoz rendelt refresh token-ek kerülnek szerveroldalon eltárolásra, ellentétben mindkét token kliensoldalon történő eltárolásával. A tokenek szerveroldali kezelését a `TokenManager` komponens biztosítja.

A felhasználó hitelesítését az adott felhasználó alkalmazásra történő regisztrációja előzi meg. Ezt követve a kliensalkalmazások új szerveroldali munkamenetet bejelentkezéssel kezdeményezhet, amely után minden kérése egy rövid élettartamú access token-nel lesz ellátva. Az access token-ek lejáratá után a kliensalkalmazások új access token-t kérhetnek a megfelelő API hívás segítségével, a refresh token lejáratáig. A refresh token lejáratá után az adott kliensalkalmazás automatikusan kijelentkeztetésre kerül a rendszerből.

A felhasználók regisztrálhatnak egy felhasználónév, egy e-mail cím és egy jelszó megadásával, vagy egy már létező Google fiók használatával, amit a Firebase Authentication<sup>31</sup> külső szolgáltatás biztosít. A szolgáltatás által nyújtott token-eket a kliensalkalmazások, bejelentkezéskor, továbbítják a szerver felé. Ezekből szerveroldalon új refresh token-ek épülnek fel, melyeket a kliensek megkapnak a bejelentkezési kérések válaszaiban a hagyományos, jelszó megadásával történő, bejelentkezéshez hasonlóan.

Az eddig bemutatott funkciók mellett a szerveroldal lehetőséget biztosít az elfelejtett jelszavak visszaállítására is. Egy jelszó-visszaállítás esetén a felhasználó e-mailben kap egy rövid lejáratú jelszó-visszaállítási token-t, amely segítségével kliensoldalon megtörténhet az új jelszó megadása. Az e-mailek elküldését az `EmailManager` komponens biztosítja egy SMTP kliens segítségével. Az említett komponens és az alrendszer viszonyát a 9. ábra szemlélteti.

### 3.2.3. Állománykezelés

A B.o.B. chatrendszer számos állománytípust támogat: PDF, TXT, DOCX szöveges állományok és PNG, JPG, JPEG képformátumok. A felsorolt támogatott kiterjesztésű állományok használhatóak fel a beszélgetések kontextusaként.

Egy állomány feldolgozása során az adott fájl feltöltésre kerül kliensoldalról a Firebase fájl tárolóba, ahonnan azután letöltésre kerül szerveroldalra a további fájl feldolgozás érdekében.

---

<sup>31</sup><https://firebase.google.com/docs/auth>

Az ezt követő fájlfeldolgozási folyamat részletei a 2. fejezetben olvashatóak. A fájlfeldolgozás végeztével szerveroldalról az ideiglenesen letöltött állomány törlésre kerül.

Egy fájlhoz tartozó osztott információ-eltárolás az alábbi módon került megvalósításra: az állomány tartalma a Firebase fájl tárolón van eltárolva, a RAG mikroszolgáltatás által nyújtott vektoradatbázis a feldolgozott állomány tartalmát tárolja el, illetve a feltöltött fájl eddig említett tárolókbeli elérhetőségei és a hozzá tartozó fontosabb metaadatok a relációs adatbázisban kerülnek eltárolásra.

Jelenleg az alkalmazás keretein belül a fájlok feltöltésére és azok törlésére van csak lehetőség. Az említett műveleteket a FileService szolgáltatás biztosítja, amelynek a szerveroldali architektúrában való elhelyezkedése a 9. ábrán látható.

### **3.2.4. Beszélgetések kezelése**

A B.o.B. alkalmazás keretén belül egy beszélgetés a felhasználó és a chatbot közti üzenetváltások összessége. A felhasználó számos, különböző tematikájú beszélgetést kezdeményezhet a chatbot felé a feltöltött állományai alapján.

A backend szerver beszélgetések esetén a létrehozás, törlés és átnevezés funkciókat biztosítja. Ezek közül a kliensalkalmazások jelenleg az első két funkciót biztosítják a felhasználók számára. Egy beszélgetésben levő üzenetek esetén a létrehozás és értékelés funkcionalitások kerültek megvalósításra. Ezek mellett egy beszélgetéshez kontextusként hozzárendelhetők a felhasználó által feltöltött állományok. Az említett funkcionalitásokat a ConversationService szolgáltatás biztosítja.

Az újonnan létrehozott beszélgetések egy a tartalmukat jellemző címmel lesznek ellátva létrehozásukat követően. A címgenerálást és a feltöltött állományok alapján történő kérdésmegválaszolást a ChatbotService komponens biztosítja kapcsolatot teremtve a RAG mikroszolgáltatással. Ennek részletesebb leírása a 2. fejezetben olvasható.

Az említett szolgáltatások és az alrendszer közti viszonyt a 9. ábra illusztrálja.

## **3.3. Verifikáció**

A szoftverfejlesztés során a verifikáció feladata az előre megszabott specifikáció, standardok ellenőrzése és azok betartatása.

A backend szerver esetén verifikációt biztosít a forráskód statikus ellenőrzése (lintelése),

illetve az egységtesztelés (unit testing). Ezeket a fejlesztési funkcionalitásokat rendre a pylint<sup>32</sup> és pytest<sup>33</sup> programkönyvtárak biztosítják.

A fejlesztés jelen pillanatában az adatelérési réteghez íródtak egységtesztek, szám szerint 97 tesztet. Ezek igényét az említett réteg saját kezű implementációja vetette fel. Az egységtesztek megvalósítása esetén az élesben használt relációs adatbázis helyett a kis méretű sqlite<sup>34</sup> memóriabeli adatbázis van használatban.

Az említett verifikációs lépések fejlesztés alatt történő folyamatos integrációja a 6. fejezetben kerül bemutatásra.

### 3.4. Felhasznált technológiák

A továbbiakban a fejezet előző alfejezeteiben még el nem hangzott lényeges technológiák kerülnek megemlítésre.

A szerveroldal, beleértve a RAG mikroszolgáltatást, a Python<sup>35</sup> programozási nyelv keretein került megvalósításra biztosítva ezáltal a gyors fejlesztést. A csomagok, programkönyvtárak, telepítését a pip<sup>36</sup> csomagtelepítő eszköz segítette. A továbbiakban említett könyvtárak ennek használatával kerültek telepítésre a Python Package Index tárolóból<sup>37</sup>.

A szerveroldali prezentációs rétegen az adatelfedést biztosító DTO-k a pydantic könyvtár segítségével kerültek megvalósításra, amit a FastAPI keretrendszer foglal magába. Az említett rétegen aszinkron HTTP szerverként a Hypercorn szerver van használatban.

A szerveroldali naplózást a structlog könyvtár biztosítja. A lokális környezeti változók fejlesztés alatt történő használatát a python-dotenv könyvtár segítette elő.

A jelszavak titkosításáért a bcrypt könyvtár felel biztosítva azok biztonságos eltárolását és összehasonlítását. A JWT tokenek előállítását és ellenőrzését a PyJWT könyvtár könnyítette meg.

---

<sup>32</sup><https://pypi.org/project/pylint/>

<sup>33</sup><https://pypi.org/project/pytest/>

<sup>34</sup><https://sqlite.org/>

<sup>35</sup><https://www.python.org/downloads/release/python-3110/>

<sup>36</sup><https://pypi.org/project/pip/>

<sup>37</sup><https://pypi.org/>

## 4. Webalkalmazás

A B.o.B. chatrendszer egyik szerves alkotóeleme a webkliens, amely biztosítja a felhasználónak a rendszer interaktív kezelését. Ez a fejezet részletesen bemutatja a weboldal által támogatott funkciókat, az architektúrát és a felhasznált technológiákat.

### 4.1. Architektúra

A B.o.B. projekt webkliense a Model–View–ViewModel (MVVM) architektúráis mintán alapul, amelynek célja a nézet (View) és az adatelérési réteg szétválasztása. Minden réteg egyértelmű, jól meghatározott szerepkörrel rendelkezik. A szétválasztás eredményeként az egyik rétegen végrehajtott módosítások minimális hatással vannak a többi rétegre, mivel nem hoznak létre szoros függőségeket, ezáltal elősegítve a rendszer karbantarthatóságát és bővíthetőségét.

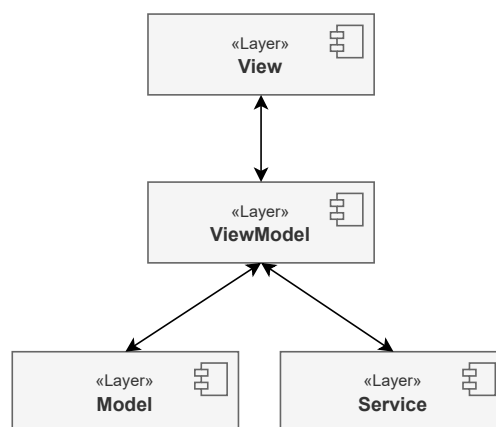
Az említett rétegek szerepköre a következő:

- *View (Nézet):*

Ez a réteg felelős az alkalmazás vizuális megjelenítéséért, valamint a felhasználói interakciók kezeléséért. Ide tartozik a vizuális komponensek állapotainak menedzsmentje és az adatok felhasználóbarát megjelenítése.

- *ViewModel (Nézetmodell):*

A ViewModel a nézet és a háttérlogika közötti közvetítő szerepet tölti be, funkcionálisát



10. ábra. A webszerver architektúráis rétegei

tekintve hasonló a kontrollerhez. A ViewModel biztosítja a nézet számára a szükséges adatokat, kezeli ezek állapotát, és a háttérben kapcsolatot tart a szerverrel egy Service rétegen keresztül.

- *Service réteg:*

A ViewModel által használt szolgáltatások a HTTP kérések kezeléséért felelősek: elküldik a megfelelő kéréseket, fogadják és feldolgozzák a válaszokat. Ez az absztrakció segít az adatkezelési logika tiszta elválasztásában és újrafelhasználhatóságában.

- *Model (Adatmodell):*

Ez a réteg az alkalmazás adatstruktúráit írja le: például a beszélgetések, felhasználók és állományok reprezentációját. Ezek a típusok tükrözik az adatokat, ahogy azok a rendszer különböző rétegeiben megjelennek.

A bemutatott architekturális rétegek viszonyát a 10. ábra illusztrálja.

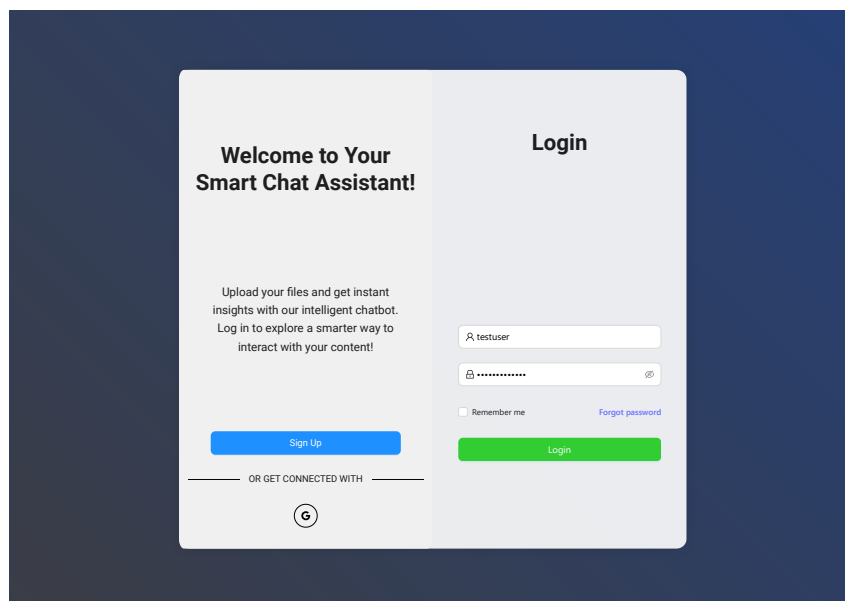
## 4.2. Funkcionalitások

A B.o.B. webkliens rendelkezik az összes általános funkcióval, amelyet a rendszer kínál, emellett számos kiegészítő funkciót is tartalmaz, amelyek a felhasználói élményt növelik. Ez a fejezetrész ezen funkciók részletes ismertetésével foglalkozik, ezáltal egy tisztább képet adva arról, hogy mire is képes a rendszer.

### 4.2.1. Bejelentkezés

Az alkalmazásba való belépéskor a felhasználó a bejelentkezési oldalon indít (ld. 11. ábra). A rendszer kétféle bejelentkezést támogat: egy saját rendszerre épülő bejelentkezést, valamint egy Firebase Authentication által biztosított, Google-fiókokon keresztül történő szociális bejelentkezést.

A saját bejelentkezési rendszer első lépése a regisztrációs folyamat. A felhasználónak meg kell adnia egy felhasználónevet, egy korábban még nem regisztrált e-mail címet és egy megfelelő jelszót. Ezeket a rendszer a *Sign Up* gomb lenyomása után elküldi a szervernek, ahol megtörténik az adatok validálása. Ha a szerver minden adatot megfelelőnek talál, visszaküld egy üzenetet, amely jelzi a felhasználónak, hogy a regisztráció sikeres volt. Ellenkező esetben a rendszer újrakéri az adatokat. A sikeres regisztráció után a felhasználónak be kell jelentkeznie az alkalmazás használatához.



11. ábra. A rendszer bejelentkezési felülete, amely biztosítja a regisztrációt, a hagyományos bejelentkezést, valamint a Google-alapú szociális hitelesítés lehetőségét

Bejelentkezés során a felhasználónak meg kell adnia a regisztráció során létrehozott felhasználónevét és jelszavát. A *Login* gombra kattintva a rendszer hitelesíti az adatokat, és ha azok helyesek, a kliensalkalmazás birtokába kerül egy access- és egy refresh token, amelyek meghatározott ideig érvényesek.

A bejelentkezési folyamat során a felhasználónak lehetősége van a refresh token érvényességi időtartamának meghosszabbítására a *Remember me* opció bejelölésével. A rendszer továbbá lehetőséget biztosít az elfelejtett jelszó megváltoztatására is. Ha a felhasználó a *Forgot password* opciót választja, egy felugró ablakban megadhatja az e-mail címét, amelyre a rendszer egy jelszó-visszaállító linket küld. A link előre meghatározott lejáratú idővel rendelkezik. Erre kattintva a felhasználó a jelszó módosító oldalra kerül, ahol megadhat egy új jelszót és annak megerősítését. A jelszó sikeres módosítása után újra be kell jelentkeznie az alkalmazás használatához.

A rendszer támogatja a Google-fiókokon keresztüli bejelentkezést, amelyet a Firebase Authentication tesz lehetővé. A Google szociális bejelentkezési ikonra kattintva egy felugró ablak jelenik meg, ahol a felhasználó kiválaszthatja a használni kívánt Google-profilt.

A Firebase Authentication az OAuth 2.0 protokoll segítségével hitelesíti a felhasználót, és egy ID token-t biztosít számára. Ezt az ID token-t a webkliens az alkalmazás szerverére továbbítja, ahol az validálásra kerül. Ha a validáció sikeres, a szerver a saját hitelesítési rendszerén belül generál egy access és egy refresh JWT token-t, amelyeket a felhasználó az

alkalmazáson belül használhat.

A sikeres bejelentkezés után a felhasználó egy megerősítő üzenetet kap, majd átirányításra kerül a kezdőlapra.

#### 4.2.2. Kezdőoldal

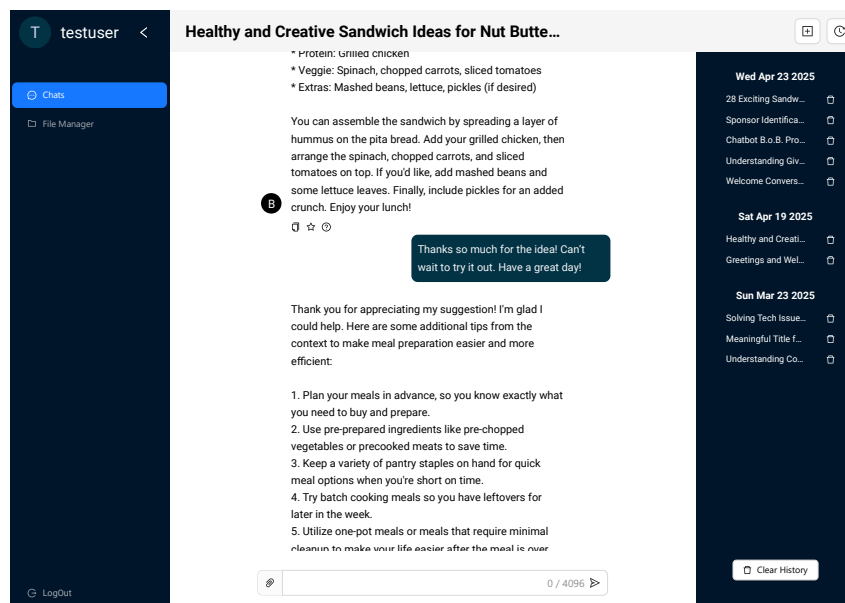
Az alkalmazás kezdőoldala egy új, üres beszélgetéssel fogadja a felhasználót, amely az ablak közepén helyezkedik el. Ebben a panelben zajlik az üzenetek küldése és fogadása, valamint lehetőség nyílik üzenetek másolására és értékelésére is.

A beszélgetési felület felett egy panel található, amely automatikusan generált címmel látja el az adott párbeszédet. Ezen a panelen két gomb is helyet kapott: az egyik egy új beszélgetés indítását teszi lehetővé, míg a másik a beszélgetési előzményeket tartalmazó, jobb oldalról előugró panel megjelenítésére vagy elrejtésére szolgál.

A beszélgetési előzmények megjelenítésekor a rendszer lekéri az összes korábbi párbeszédet a szervertől, és azokat időrend szerinti csökkenő sorrendben rendezi. Minden beszélgetés címe mellett egy törlés gomb is elérhető, amely lehetőséget biztosít az adott párbeszéd eltávolítására. Ha a felhasználó éppen egy aktív beszélgetést töröl, a rendszer automatikusan egy új, üres beszélgetést nyit meg. Emellett a panel alján egy *Delete All* gomb is megtalálható, amely az összes korábbi beszélgetést egyetlen művelettel törli. A felhasználó bármikor visszatérhet egy korábbi beszélgetéshez, ilyenkor az összes üzenet, értékelés és a válaszokhoz tartozó források betöltődnek, valamint az adott párbeszéd során használt állományok is automatikusan hozzárendelődnek, így a kommunikáció ott folytatható, ahol félbemaradt.

A bal oldali navigációs sáv egy avatárt és a felhasználó nevét tartalmazza, alatta pedig elhelyezkednek a különböző navigációs lehetőségek. Innen érhető el a beszélgetési felület és az állománykezelő rendszer is. A sáv alján egy kijelentkezés gomb található, amely törli a felhasználó hitelesítési adatait és jelzi a szerver számára a munkamenet lezárását. A navigációs sáv össze- és kihajtható; bezárt állapotban csak az ikonok és az avatár láthatók, míg nyitott állapotban minden gombhoz egy leíró szöveg is tartozik.

A beszélgetési ablak alsó részén egy beviteli sáv helyezkedik el, amely négy funkcionális elemet tartalmaz. A jobb oldalon található egy küldés gomb, amely továbbítja az üzenetet a chatbot-nak. Mellette egy karakter számláló kap helyet, amely visszajelzést ad a felhasználónak a maximális szövegméretéről. A középen egy szövegmező áll rendelkezésre az üzenetek beírására. A bal oldalon egy állomány csatoló gomb kapott helyet, amely két lehetőséget kínál:



12. ábra. A beszélgetéseket tartalmazó felület. A bal oldalon egy navigációs sáv és egy kijelentkezési gomb található. A fő panelen a felhasználó által feltett kérdés és a chatbot válasza jelenik meg, strukturált, felsorolásos formátumban. A jobb oldali panelen a korábbi beszélgetések jelennek meg időbélyeggel ellátva.

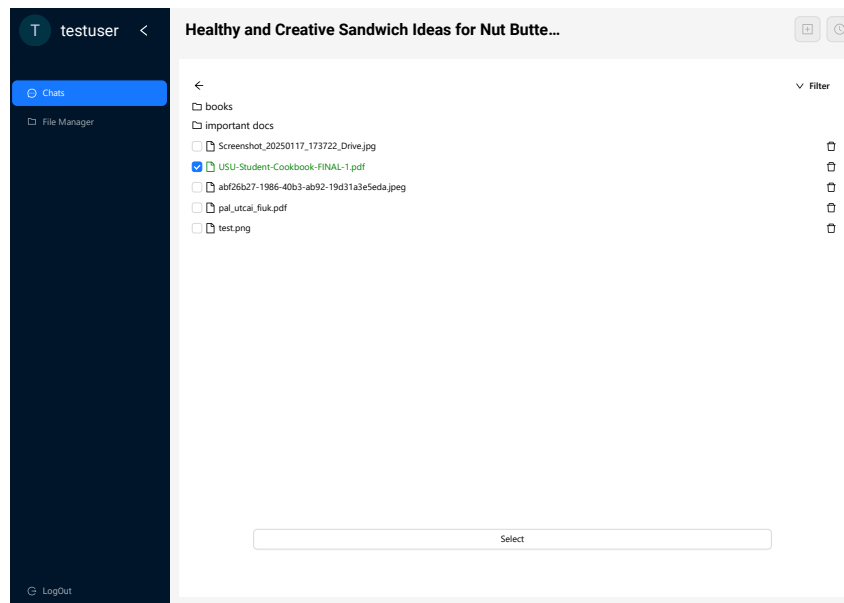
új fájlok feltöltését vagy korábban feltöltött állományok hozzácsatolását. Az új állományok feltöltése egy felugró ablakban történik, ahova a fájlok egyszerűen behúzhatók vagy manuálisan kiválaszthatók a fájlkezelőből. A feltöltési folyamat állapotát egy állapotsáv jelzi, a sikeresen feltöltött fájlok pedig automatikusan a Firebase Storage rendszerébe kerülnek. Ha a felhasználó a csatolás lehetőséget választja, a beszélgetési ablak felett egy új panel jelenik meg, amely a kiválasztott állományok hozzáadását teszi lehetővé.

A bemutatott beszélgetéseket tartalmazó felület a 12. ábrán látható.

#### 4.2.3. Állományok hozzárendelése beszélgetésekhez

Az állományok kiválasztására egy külön ablak áll rendelkezésre (ld. 13. ábra), amely a beszélgetési felület fölött jelenik meg. Az ablak fő tartalmát a feltöltött állományok és a létrehozott mappák alkotják, amelyek egymás alatt, függőleges elrendezésben helyezkednek el.

Az állományok kiválasztása interaktív módon történik: a felhasználó egy adott állományra kattintva annak címkéjét zöld színnel emeli ki, jelezve, hogy az kiválasztás alatt áll. Ezt a vizuális visszacsatolást egy kijelölő négyzet is kiegészíti, amely lehetővé teszi a pontosabb és egyszerűbb kijelölést. Az állományok mellett megjelenő kuka ikon segítségével a nem kívánt állományok közvetlenül eltávolíthatók.

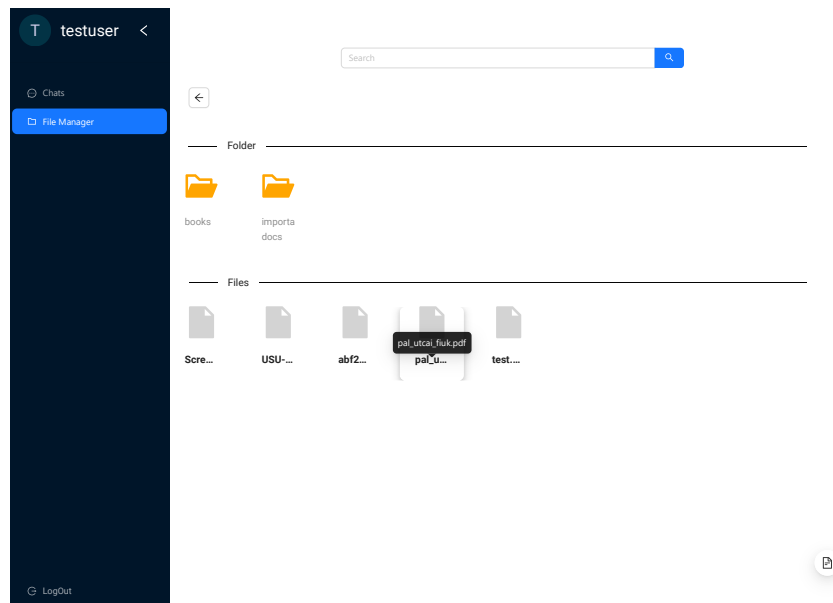


13. ábra. A beszélgetésekhez tartozó állománykiválasztó felület. A bal oldalon egy navigációs sáv és egy kijelentkezési gomb található. A fő panelen a kijelölhető állományok törlési opcióval vannak ellátva.

A mappák megjelenítése dinamikusan frissül: a felhasználó egy mappára kattintva annak tartalmát töltheti be az állománylistába. Az ablak felső részén egy funkcióssáv található, amely különböző segédfunkcionalitásokat biztosít. Jobb oldalon a szűrési opció helyezkedik el, amelynek aktiválásával kizárólag a kiválasztott állományok maradnak láthatók, amíg minden más eltűnik az ablakból. A szűrés állapotát egy *Show Selected* címkével ellátott tag jelzi, amely egy kis „x” gombbal kikapcsolható, így az állománylista visszatér az eredeti nézetéhez.

A felső sáv további fontos eleme a navigációs címke, amely a felhasználó aktuális helyzetét jeleníti meg a mappastruktúrában. Ez a címke interaktív, lehetővé téve a gyors visszalépést bármelyik szülőmappába egy egyszerű kattintással. A navigációt egy nyíl alakú gomb is támogatja, amely mindig egy szinttel feljebb lépteti a felhasználót a mappastruktúrában. Amennyiben a gyökérkönyvtárba ér, az újabb visszalépési próbálkozás automatikusan visszairányítja a beszélgetési felületre. Ha a felhasználó ezen a módon hagyja el az állományválasztó ablakot, a kijelölt állományok listája változatlan marad, biztosítva a korábban kiválasztott elemek megőrzését.

Az ablak alján egy megerősítő gomb található, amely a kiválasztott állományok listájának mentésére szolgál. Ennek aktiválásakor a rendszer visszairányítja a felhasználót a beszélgetési felületre, ahol egy visszajelző üzenet tájékoztatja a sikeres állománykiválasztásról.



14. ábra. A globális állománykezelő rendszer felülete a gyökérkönyvtárban levő mappákkal és állományokkal

#### 4.2.4. Globális állománykezelő rendszer

Amennyiben a felhasználó a bal oldali navigációs sávban kiválasztja az állománykezelő opciót, a rendszer átirányítja a globális állománykezelő oldalra (ld. 14. ábra). Ez az oldal a beszélgetési felület és a jobb oldalon található beszélgetés-előzmény panel helyén jelenik meg.

A képernyő középső részén a mappák és állományok kártya komponensekként jelennek meg. A átláthatóság érdekében először a mappák listázódnak, amit egy címke is megjelöl, majd ezek alatt elválasztva következnek az állományok. A könnyebb felismerhetőség érdekében minden mappa egy központi, nagyobb mappa ikonnal rendelkezik, alatta pedig a neve található. Az állományok megjelenési struktúrája megegyezik a mappákéval, megfelelő típusikonok használatával. Ha az állománykezelő üres, egy tájékoztató üzenet jelzi, hogy még nincsenek feltöltött állományok vagy létrehozott mappák.

A kártyák interaktívak, de eltérő funkcionalitással rendelkeznek. A jobb egérgombbal kattintva egy lebegő menü jelenik meg, amely törlési opciót kínál. Mappa törlése esetén annak teljes tartalma is eltávolításra kerül.

Az állományok mozgatása egyszerű fogd és vidd módszerrel történik: a felhasználó a bal egérgomb lenyomásával és húzásával áthelyezheti az állományt egy adott mappába. Mappák esetén a bal egérgombbal kattintva beléphetünk az adott mappába, amelynek tartalma az előzőleg ismertetett struktúra szerint frissül.

A képernyő jobb alsó sarkában egy lebegő gomb található, amely új mappa létrehozását teszi

lehetővé. A gomb megnyomásakor egy felugró ablak jelenik meg, amely bekéri a létrehozandó mappa nevét. Az adat megadása után a *Create* gomb lenyomásával a mappa létrejön és megjelenik a listában. Egy visszajelző üzenet megerősíti a sikeres létrehozást.

A képernyő felső részén egy keresősáv kap helyet, amely lehetővé teszi az állományok és mappák név szerinti keresését. A beírt szövegrészletet felhasználva automatikusan szűri a megjelenítendő állományok, mappák listáját.

A keresősáv és az állományokat, valamint mappákat tartalmazó terület között egy navigációs sáv és egy vissza gomb található, amelyek funkcionalitása megegyezik a már korábban 4.2.3. részben tárgyaltakkal.

A vissza gomb megnyomásával a rendszer visszairányítja a felhasználót a kezdőlapra.

### 4.3. Felhasznált technológiák

A webalkalmazás fejlesztése során számos modern és széles körben elterjedt technológia került felhasználásra. A felhasználói felület kialakítása a React<sup>38</sup> könyvtár segítségével történt, amely komponensalapú megközelítésével lehetővé tette újrafelhasználható elemek létrehozását, valamint a deklaratív UI fejlesztést. A JavaScript nyelv típusos kiterjesztéseként a TypeScript<sup>39</sup> alkalmazása elősegítette a kód minőségének javítását, valamint hozzájárult a hibák korai felismeréséhez. A vizuális komponensek egységes és strukturált megjelenésének biztosítására az Ant Design<sup>40</sup> komponenskönyvtár került felhasználásra, amely egy modern, letisztult felhasználói felület kialakítását teszi lehetővé.

A kód minőségének és egységességének fenntartása érdekében a projektbe integrálva lett az ESLint<sup>41</sup> statikus kódelemző eszköz, amely biztosította a konzisztens kódstílust, és segítette a potenciális hibák időben történő észlelését. A projekt függőségeinek kezelését a Yarn<sup>42</sup> csomagkezelő végezte, amely gyors és megbízható telepítési folyamatot biztosított. A fejlesztés a JetBrains WebStorm<sup>43</sup> integrált fejlesztőkörnyezetben zajlott, amely fejlett szerkesztési lehetőségeivel és kényelmi funkcionalitásaival támogatta a hatékony munkavégzést.

A felhasználói felület megtervezése mockupok létrehozása által valósult meg, ez a Moqups<sup>44</sup> nevű online tervezőeszköz felhasználásával készült el.

---

<sup>38</sup><https://react.dev>

<sup>39</sup><https://www.typescriptlang.org>

<sup>40</sup><https://ant.design/>

<sup>41</sup><https://eslint.org>

<sup>42</sup><https://yarnpkg.com>

<sup>43</sup><https://www.jetbrains.com/webstorm/>

<sup>44</sup><https://moqups.com>

## 5. Mobilalkalmazás

A B.o.B. projekt kliensfelületei közé tartozik az Android–iOS cross-platform mobilkliens, amely lehetővé teszi a rendszer elérését mobilalkalmazás formájában is. Ez a fejezet bemutatja a mobilkliens által támogatott funkciókat, az alkalmazott architektúrát és a felhasznált technológiákat.

### 5.1. Architektúra

A B.o.B. projekt mobilkliense a React Native és Expo technológiákra épül, amelyek lehetővé teszik a natív Android- és iOS-alkalmazások fejlesztését egy közös kódbázison belül. A képernyők közötti navigációt az Expo Router által biztosított file-based routing rendszer valósítja meg, amely elősegíti a moduláris, átlátható és könnyen bővíthető architektúra kialakítását.

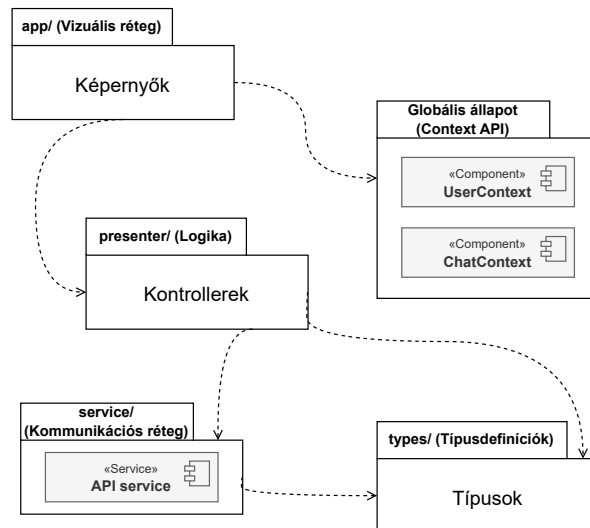
A file-based routing rendszer gyökere az `app/` mappában található. Ezen belül az almappák és fájlok automatikusan definiálják az alkalmazás útvonalait, így nincs szükség manuális útvonal-regisztrációra. Ez leegyszerűsíti a képernyők kezelését, és elősegíti a projekt strukturált felépítését.

Az alkalmazás architektúrája három fő rétegre tagolható: a vizuális réteg, a kommunikációs réteg, valamint az adatrepresentációs réteg. Ezek szétválasztása elősegíti a felelősségi körök tisztázását, a kód újrafelhasználhatóságát, valamint a karbantarthatóságot.

A vizuális réteg az `app/` mappán belül helyezkedik el, ahol a képernyők és komponensek kizárólag a felhasználói felület megjelenítéséért felelnek. Itt történik az adatok renderelése, felhasználói események kezelése, valamint a komponensek strukturálása.

A kommunikációs réteg két fő részre osztható. Az első a `presenter/` mappában elhelyezkedő kontrollerek, amelyek a komponensek logikájáért és az adatok frissítéséért felelősek. A második a `service/` réteg, amely HTTP kéréseken keresztül kapcsolatot teremt a backend szerver és a kliens között. A service réteg biztosítja az adatok lekérését, küldését, valamint a token alapú hitelesítés kezelését.

A `types/` mappa tartalmazza az alkalmazás számára fontos entitások típusdefinícióit. Ide tartoznak például a beszélgetések, állományok, mappák és a felhasználó típusai. A típusos modellalkotás TypeScript segítségével történik, ami növeli a rendszer robusztusságát és csökkenti a futásidejű hibák előfordulását.



15. ábra. A mobil kódrendszer architektúrási rétegei

Az alkalmazás működése szempontjából kulcsfontosságú a megfelelő állapotkezelés. A komponens szintű lokális állapotokon kívül két különálló globális állapot is kialakításra került, amelyek Context API segítségével valósulnak meg. Ezek az aktuálisan kiválasztott beszélgetést, illetve a bejelentkezett felhasználó adatait tárolják. A globális állapotok Context komponenseken keresztül egyszerűen elérhetők és kezelhetők az alkalmazás különböző részeiből.

## 5.2. Funkcionalitások

A B.o.B. projekt mobilkliens funkcionálisok szempontjából hasonló a webklienshez, viszont tartalmaz egy, csak ezen a felületen elérhető állományfeltöltési opciót, amely különbséget jelent a két kliens között. Örökölt funkcionálisok szempontjából a mobilkliens a következőket tartalmazza:

- felhasználói fiók létrehozása;
- beszélgetések teljes körű kezelése;
- üzenetek értékelése és másolása;
- állományok feltöltése, beszélgetésekhez hozzárendelése és törlése.

Az állománykezelő felület a felső sáv bal oldalán található mappá ikon segítségével érhető el. Ennek lenyomására megnyílik az állománykezelő ablak. A beszélgetés-előzmény

megtekintése a felső sáv jobb oldalán elhelyezett ikon segítségével megtekinthető, amely egy jobbról balra kicsúszó panelt jelenít meg.

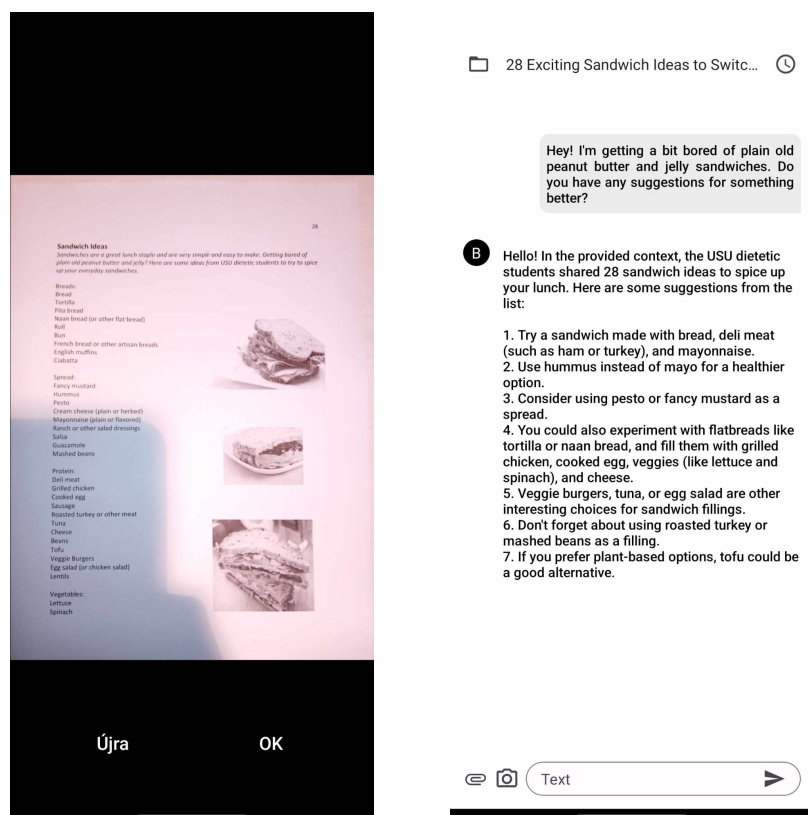
Fontos megemlíteni, hogy a mobilalkalmazás nem minden minden webes felületen elérhető funkcionalitást támogat. Ide tartoznak például a mappákkal kapcsolatos funkcionalitások, valamint a válaszok forrásának megjelenítése.

### 5.2.1. Kamera opció

Az állományok feltöltésén túl a mobilalkalmazás lehetőséget biztosít a felhasználó számára képek készítésére, és azok kontextusként való felhasználására a beszélgetésekben (ld. 16. ábra).

A beszélgetés-ablakhoz tartozó állományfeltöltő gomb mellett egy kamera ikon is található. Ezt megnyomva az alkalmazás engedélykérő ablakot jelenít meg, amely a kamera használatára kér engedélyt.

Amennyiben a felhasználó engedélyezi a hozzáférést, az alkalmazás lehetőséget biztosít fényképek készítésére a készülék kamerájával. Az elkészült képeket az alkalmazás automatikusan feltölti az aktuális beszélgetéshez, további lépések nélkül.



16. ábra. A mobilalkalmazás kamera opciójának szemléltetése egy szakácskönyvrészleten [8] alapuló beszélgetés által

## 6. DevOps

A folyamatos integráció és kitelepítés folyamata a szoftverfejlesztés során történő új változtatások automatizált verifikációját és üzembe helyezését foglalja magába.

Ezen fejezeten belül a fejlesztés során történő folyamatos integráció és kitelepítéshez használt eszközök kerülnek ismertetésre.

### 6.1. Folyamatos integráció

A B.o.B. chat-rendszer fejlesztése alatt a git<sup>45</sup> osztott verziókövető rendszer volt használatban egy-egy GitLab<sup>46</sup> által nyújtott tárolót használva távoli tárolókként alrendszerenként. Ezen platform által nyújtott folyamatos integráció és kitelepítési funkciók, GitLab CI/CD<sup>47</sup>, kerültek alkalmazásra munkafolyamatok, avagy pipeline-ok, formájában.

A webszerver fejlesztését elősegítő pipeline két job-ot tartalmaz, melyek a weboldal statikus kódellenőrzését és build-elését biztosítják eltávolítva a kitelepítendő webklienst egy artifact formájában. Ez kerül a későbbiekben kitelepítésre egy nginx<sup>48</sup> HTTP szerverre épülő Docker<sup>49</sup> image tartalmaként. Az alrendszerhez tartozó pipeline-t a 17. (a) ábra szemlélteti.

A mobilkliens fejlesztését megkönnyítő pipeline a webklienssel ellentétben a statikus kódellenőrzés mellett egy kompilálási fázist tartalmaz, melynek során a Typescript kód egy utólagos típusellenőrzésen esik át. A munkafolyamat illusztrációja a 17. (b) ábrán látható.

A központi szerverhez tartozó pipeline a statikus kódellenőrzés mellett automatizált egységtesztelést is végez biztosítva az alrendszer megbízhatóságát.

A RAG mikroszolgáltatáshoz tartozó pipeline automatizált statikus kódellenőrzést és kitelepítést biztosít az LLM futtatásához elegendő erőforrást nyújtó szerverre. Az alrendszer kitelepítése pipeline-ból egy Docker image létrehozásával és annak a GitLab által nyújtott konténertárolóba történő feltöltésével történik. Az image-k pipeline-ból való kitelepítését a kaniko<sup>50</sup> eszköz segíti. A szerverre a konténertárolóból kerül letöltésre az alrendszer módosított verziója. Az alrendszerhez tartozó munkafolyamatot a 17. (c) ábra illusztrálja.

---

<sup>45</sup><https://git-scm.com/>

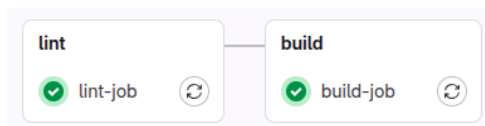
<sup>46</sup><https://about.gitlab.com/>

<sup>47</sup><https://about.gitlab.com/topics/ci-cd/>

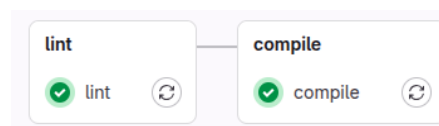
<sup>48</sup><https://nginx.org/>

<sup>49</sup><https://www.docker.com/>

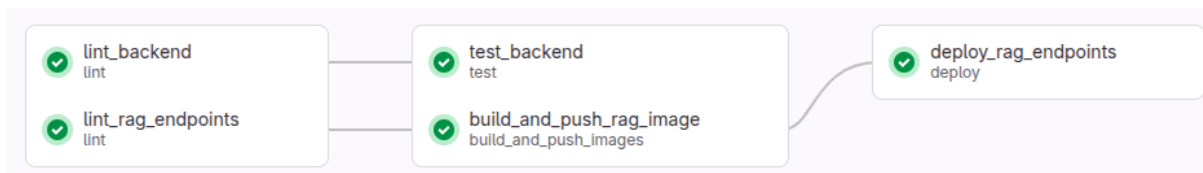
<sup>50</sup>[https://docs.gitlab.com/ci/docker/using\\_kaniko/](https://docs.gitlab.com/ci/docker/using_kaniko/)



(a) Webszerverhez tartozó pipeline a statikus kódellenőrzést és automatizált build-elést biztosító job-okkal



(b) Mobilklienshez tartozó pipeline a statikus kódellenőrzést és kompilálást biztosító job-okkal



(c) Backend szerver és RAG mikroszolgáltatás fejlesztését elősegítő pipeline mindkét alrendszer statikus kódellenőrzését, a szerver egységtesztelését és a mikroszolgáltatás build-elését és automatizált kitelepítését biztosító job-okkal

17. ábra. Az alrendszerek fejlesztését elősegítő munkafolyamatok

A szerveren futó alrendszer komponensei (Ollama, HTTP szerver, Milvus) izoláltan futnak Docker konténerek formájában. A konténerek automatizált, a szerveren történő, vezérlését a Docker compose<sup>51</sup> Docker által nyújtott bővítmény biztosítja.

## 6.2. Kitelepítés

A webszerver, backend szerver és MSSQL adatbázis futtatását az Azure<sup>52</sup> felhőszolgáltató biztosítja. A felsorolt architektúrális komponenseket egy, a szolgáltató által nyújtott, erőforráscsoport tartalmazza.

Az említett relációs adatbázis egy külön, az erőforráscsoporthoz tartozó, adatbázis szerveren fut. A backend- és webszerver egy szolgáltatáscsomagon belül egy-egy, Docker konténerekre alapuló, alkalmazás formájában fut.

A projekt jelenlegi állapotában a web és backend szerver kitelepítése csak manuálisan oldható meg, az Azure által nyújtott konténertárolóba történő előre felépített docker image-ek feltöltése útján.

<sup>51</sup><https://docs.docker.com/compose/>

<sup>52</sup><https://azure.microsoft.com/en-us/>

## Következtetések és továbbfejlesztési lehetőségek

A B.o.B. chatbot projekt keretében az összes előzetesen meghatározott funkcionalitás sikeresen megvalósításra került. A rendszer képes beszélgetések létrehozására és kezelésére, üzenetek értékelésére, valamint állományok feltöltésére és törlésére. Mindezek eredményeként egy olyan többplatformos alkalmazás jött létre, amely széles körben használható, és különféle felhasználói igényeket képes kiszolgálni.

Annak ellenére, hogy a projekt során a fő célkitűzések teljesültek, néhány funkció megvalósítására a rendelkezésre álló idő korlátozottsága miatt nem kerülhetett sor. A mobilalkalmazás esetében ilyen hiányosság például a mappák létrehozásának lehetősége vagy a külső szolgáltatókon keresztüli (például Google-alapú) hitelesítés. A webalkalmazás kapcsán pedig az állományok előnézeti funkciója maradt ki, amely a jövőben szintén hozzáadható.

A rendszer továbbfejlesztésére több irányban is lehetőség kínálkozik. Ilyen például a beszélgetések archiválásának és más felhasználókkal való megosztásának bevezetése, illetve az üzenetek értékelésének egy fejlettebb válaszolási mechanizmus által történő felhasználása, amelyek jobb felhasználói élményt nyújthatnak. Emellett olyan fejlesztések is indokoltak lehetnek, amelyek a háttérrendszer megbízhatóságát növelik, például a fájlműveletek tranzakciókezelésének bevezetése vagy a rendszer skálázhatóságának optimalizálása.

Összegzésként megállapítható, hogy a B.o.B. chatbot projekt egy stabil alapokon nyugvó, jól strukturált rendszerként valósult meg, amely megfelelő kiindulópontot jelent további bővítésekhez és valós környezetben történő alkalmazáshoz is.

## Hivatkozások

- [1] Humaid Al Naqbi, Zied Bahroun és Vian Ahmed. „Enhancing Work Productivity through Generative Artificial Intelligence: A Comprehensive Literature Review”. *Sustainability* 16.3 (2024). ISSN: 2071-1050. DOI: 10.3390/su16031166. URL: <https://www.mdpi.com/2071-1050/16/3/1166>.
- [2] Tom Brown és mások. „Language Models are Few-Shot Learners”. *Advances in Neural Information Processing Systems*. Szerkesztette H. Larochelle és mások. 33. kötet. Curran Associates, Inc., 2020, 1877–1901. oldal. URL: [https://proceedings.neurips.cc/paper\\_files/paper/2020/file/1457c0d6bfc4967418bfb8ac142f64a-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfc4967418bfb8ac142f64a-Paper.pdf).
- [3] Albert Q. Jiang és mások. *Mistral 7B*. 2023. arXiv: 2310.06825 [cs.CL]. URL: <https://arxiv.org/abs/2310.06825>.
- [4] Patrick Lewis és mások. „Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks”. *Advances in Neural Information Processing Systems*. Szerkesztette H. Larochelle és mások. 33. kötet. Curran Associates, Inc., 2020, 9459–9474. oldal. URL: [https://proceedings.neurips.cc/paper\\_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf).
- [5] Alec Radford és mások. „Language models are unsupervised multitask learners”. *OpenAI blog* 1.8 (2019), 9. oldal.
- [6] *SVG Repo - Free SVG Vectors and Icons*. <https://www.svgrepo.com/>. (elérés dátuma: 2025. ápr. 27.)
- [7] Hugo Touvron és mások. „Llama 2: Open foundation and fine-tuned chat models”. *arXiv preprint arXiv:2307.09288* (2023).
- [8] Utah State University. *USU-Student-Cookbook*. <https://www.usu.edu/aggiwellness/files/USU-Student-Cookbook-FINAL-1.pdf>. (elérés dátuma: 2025. ápr. 27.)
- [9] Ashish Vaswani és mások. „Attention is All you Need”. *Advances in Neural Information Processing Systems*. Szerkesztette I. Guyon és mások. 30. kötet. Curran Associates, Inc., 2017. URL: [https://proceedings.neurips.cc/paper\\_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf).
- [10] Wenhui Wang és mások. *MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers*. 2020. arXiv: 2002.10957 [cs.CL]. URL: <https://arxiv.org/abs/2002.10957>.