ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

OE-NIK Hallgató neve: **Németh Adrián**
2021/22 Hallgató törzskönyvi száma: **T/005937/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Szoftvertervezés és -fejlesztés Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve:	**Németh Adrián**
Törzskönyvi száma:	T/005937/FI12904/N
Neptun kódja:	[redacted]

A dolgozat címe:

Modern webalkalmazás fejlesztése SaaS alapokon
Development of a modern SaaS-based web application

Intézményi konzulens:	Kiss Dániel
Külső konzulens:	
Beadási határidő:	2022. május 15.
A záróvizsga tárgyai:	Számítógép architektúrák Vállalati információs rendszerek specializáció

A feladat

A "Software as a Service" (SaaS) alkalmazások napjaink divatos felhő alapú szolgáltatásai közé tartoznak. Ezek a hagyományos alkalmazásokkal szemben nem igényelnek telepítést a felhasználó számítógépén, azok a szolgáltató eszközein futnak. A terület szakértői szerint az ilyen alkalmazások a következő években még elterjedtebbé válnak a piacon.
A munka célja egy SaaS-alapokon nyugvó webalkalmazás létrehozása, amelyhez a szakdolgozónak meg kell ismernie a technológia legfontosabb jellemzőit, ezt követően meghatároznia az elkészítendő saját alkalmazás funkcióit és képességeit, megvalósítania, végül tesztelnie az alkalmazást.

A dolgozatnak tartalmaznia kell:

- a "Software as a Service" alkalmazások jellemzőinek ismertetését,
- az elkészítendő alkalmazás fontosabb jellemzőit,
- néhány SaaS-alapú webalkalmazás ismertetését és összehasonlítását, illetve a szoftver elkészítéséhez szükséges technológiai elemek rövid ismertetését,
- a saját webalkalmazás pontos specifikációját és rendszertervét,
- a fejlesztés részletes dokumentációját,
- az alkalmazás tesztelési tervét és a teszteléssel kapott eredményeket,
- a megvalósított SaaS-alapú webalkalmazás továbbfejlesztési lehetőségeit.

Vámossy

.....
Dr. Vámossy Zoltán
intézetigazgató

A szakdolgozat elévülésének határideje: **2024. május 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

Kiss Dániel
.....
intézményi konzulens

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat / diplomamunka saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban / diplomamunkámban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 2022. 05. 14.

.....

hallgató aláírása

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve:	Neptun Kód:	Tagozat:
Németh Adrián	[redacted]	**esti**

Telefon: Levelezési cím (pl.: lakcím):

[redacted]

Szakdolgozat címe magyarul:

Modern webalkalmazás fejlesztése SaaS alapokon

Szakdolgozat címe angolul:

Development of a modern SaaS-based web application

Intézményi konzulens: Külső konzulens:

Kiss Dániel

Alk.	Dátum	Tartalom	Aláírás
1.	2021. 02. 24.	Tématerv egyeztetése	Kiss Dániel
2.	2021. 03. 10.	Irodalmi források áttekintése	Kiss Dániel
3.	2021. 04. 19.	Alkalmazható technológiák	Kiss Dániel
4.	2021. 05. 03.	Specifikáció, rendszerterv elkészítése	Kiss Dániel

A hallgató a Szakdolgozat I. tantárgy követelményét teljesítette, beszámolóra bocsátható.

Kiss Dániel

Intézményi konzulens

Budapest, 2021. május 14.

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve:	Neptun Kód:	Tagozat:
Németh Adrián	[redacted]	**esti**

Telefon: Levelezési cím (pl.: lakcím):

[redacted]

Szakdolgozat címe magyarul:

Modern webalkalmazás fejlesztése SaaS alapokon

Szakdolgozat címe angolul:

Development of a modern SaaS-based web application

Intézményi konzulens: Külső konzulens:

Kiss Dániel

Alk.	Dátum	Tartalom	Aláírás
1.	2022. 02. 21.	Fejlesztői dokumentáció terve	Kiss Dániel
2.	2022. 03. 10.	Szerveroldali implementáció	Kiss Dániel
3.	2022. 04. 04.	Kliensoldali implementáció	Kiss Dániel
4.	2022. 05. 11.	Dolgozat szövegének véglegesítése	Kiss Dániel

A hallgató a Szakdolgozat II. tantárgy követelményét teljesítette, védésre bocsátható.

Kiss Dániel

.....

Intézményi konzulens

Budapest, 2022. május 11.

1. TARTALOM

2. BEVEZETÉS, CÉL MEGHATÁROZÁS

Napjainkban rengeteg féle technológia és architektúra elérhető különböző célú alkalmazás fejlesztésére. A témám célja elsősorban annak a bemutatása hogyan lehet a ma elérhető modern technológiákkal, egy felhőalapú Software as a Service („Szoftver, mint szolgáltatás”) webalkalmazást létrehozni. Mi is az a Software as a Service? A Software as a Service egy olyan üzleti szolgáltatás, amely a felhasználók számára azonnal, könnyedén – jellemzően vékony kliensen (böngészőn), de nem kötelezően – elérhető, felhasználók számától függően skálázódó megoldás. További jellemzője, hogy a felhasználóktól semmilyen infrastruktúrát nem igényel.

A mai modern alkalmazások általában web alapúak, amelynek előnye, hogy nem kell telepítéssel bajlódni, bármilyen készüléken elérhető, amin telepítve van valamely böngésző. Legyen az akár telefon vagy tablet, kijelző mérettől függetlenül használható rendszert kapunk.

A dolgozat során egy egyszerű privát képtároló alkalmazás segítségével fogom bemutatni az imént említett elvek mentén az alkalmazás fejlesztésének folyamatát.

3. LÉTEZŐ MEGOLDÁSOK, HASONLÓ RENDSZEREK

Ebben a fejezetben két közismert, nagy cég megoldását fogom bemutatni az elérhető funkciók szempontjából.

3.1 Google Photos [1]

3.1.1 Feltöltés

A Google megoldása a képek feltöltésére két lehetőséget ad: hagyományos módon, fájlkiválasztó segítségével, vagy úgynevezett „drag-and-drop” módszerrel, ami azt jelentim, hogy a böngészőre dobva a képeket feltöltésre kerülnek. A feltöltés folyamata megszakítható, illetve állapotáról egy folyamatjelző csík ad visszajelzést.

3.1.2 Böngészés

A már feltöltött képek a Fotók aloldalon ömlesztve, kép készítésének dátuma szerint csökkenő sorrendbe érhető el. Rendezési funkcióra nincs lehetőség. Egy kép megnyitása után a következő funkciók érhetők el:

- Megosztás
    - Más Google felhasználókkal,
    - Linkel,
    - Facebook, vagy Twitter közösségi platformokon
- Szerkesztés
- Kép információi
    - Dátum,
    - Név
    - Felbontás, méret
    - Hely adat
- Kedvencnek jelölés
- Albumhoz hozzáadás
- Törlés

A Felfedezés oldalon a következő tulajdonságok alapján böngészhetjük a képeket:

- Kedvencek,
- Nemrég feltöltött,
- Képernyőképek,
- Szelfik
- Videóklipek
- Mozgó fotók

3.1.3 Kedvencek

Kedvencként megjelölt képeinket böngészhetjük ezen az aloldalon.

3.1.4 Albumok

Az albumok aloldalon értelemszerűen albumokat hozhatunk létre, amivel csoportosítjuk képeinket. Itt már lehetőségünk van rendezésre album címe, legutóbbi módosítás, illetve legutóbbi fotó alapján.

3.1.5 Segédprogramok

A Google megoldása a felületen lehetőséget ad arra, hogy a képeinkből Filmet, Animációs filmet, kollázsokat hozzunk létre.

3.1.6 Archívum

Amennyiben nem szeretnénk bizonyos képeket böngészés során látni, de törölni sem szeretnénk őket az archiválás funkció erre nyújt megoldást.

3.1.7 Törlés, kuka

A törölt képek gyűjtőoldala. A rendszer nem rögtön véglegesen törli a képeket, 60 napig a felhasználók bármikor visszaállíthatják őket.

3.2 Keresés, Szűrés

A Google megoldása fejlett keresési, szűrési funkciókkal rendelkezik. A feltöltött képeket mesterséges intelligencia dolgozza fel, a képeken tartalma alapján kereshetünk például a következő szempontok alapján:

- Hely,
- Személyek,
- Dátum,
- Ételek, tárgyak, tájképek
- Domináns színek

3.3 iCloud Photos [2]

3.3.1 Feltöltés

Az Apple megoldása során a képfeltöltés egy szimpla fájl kiválasztó segítségével történik

3.3.2 Böngészés

A képeinket dátum szerint növekvő vagy pillanatok szerint böngészhetjük. A pillanatok funkció gyakorlatilag az egymással közel egyidőben készült képek halmazait szerint csoportosítva jeleníti meg képeinket. Lehetőség van arra is, hogy a képek méretét növeljük a megjelenített listában. Szintén megjelölhetjük képeinket kedvencként, ekkor megjelenik egy új menüpont, ahol elérhetőek a megjelölt elemek.

3.3.3 Elrejtés

A Google archívum funkciójával egyezik meg, elrejthetünk bizonyos képeket, amiket azonban még törölni nem szeretnénk.

3.3.4 Törlés

Szintén törölhetjük a feleslegessé vált képeinket, amik a törölt képek menüpont alatt jelennek meg. A végleges törlésre itt 40 nap után kerül sor.

3.3.5 Albumok

Két féle albumot különböztet meg a rendszer:

- Automatikusan létrehozott
    - Videók,
    - Portrék,
    - Panorámaképek,
    - Képernyőképek,
    - Animált képek
- Felhasználó által, manuálisan létrehozott

3.4 Összegzés

A fentebb említett funkciók alapján a két cég terméke teljesen más célt szolgál. A Google alkalmazása lényegesen funkció gazdagabb, lehetőséget ad közösségi megosztásra, mesterséges intelligencia alapú keresésre, az Apple megoldása szinte csak egy olyan alkalmazás, ami biztonsági mentésként tárolja képeinket.

4. FELHASZNÁLÓI FUNKCIÓK

Az előző fejezetben ismertetett, már létező szolgáltatások alapján állítottam össze a funkció listát. Ezen funkciók kiválasztásakor igyekeztem egy olyan halmazt választani, ami elégséges ahhoz, hogy egy használható alkalmazást kapjunk és olyan ésszerű mértékű fejlesztési idővel rendelkezik, ami belátható időn belül megvalósítható. Tartalmaz azonban olyan elemeket, ami nem a lehető legegyszerűbb architektúrát fogja eredményezni. A következő alpontokban, röviden ismertetem ezeket a funkciókat.

4.1 Felhasználók kezelése

Az alkalmazás mivoltából adódóan elkerülhetetlen a felhasználók kezelése. Ezen funkciók a regisztráció, belépés és elfelejtett jelszó.

4.2 Képek kezelése

4.2.1 Feltöltés

Az alkalmazás alap funkciója, hogy a felhasználók képeket tudjanak feltölteni. Egyszerre egy vagy több kép feltöltésére lesz lehetőségük, a hagyományos fájl kiválasztás, illetve „drag-and-drop” módszerrel. A felhasználói élmény érdekében a folyamat állásáról visszajelzést kapnak, beleértve az esetleges sikertelen feltöltést.

4.2.2 Böngészés

Szintén alap funkció. A Google megoldásához hasonlóan ez egy egyszerű oldal lesz, ahol a képek kilistázásra kerülnek. Dátum szerint rendezhetőek lesznek csökkenő, növekvő sorrendben.

4.2.3 Letöltés

A felhasználóknak szüksége lehet a képeikre az alkalmazás keretin kívül is, így a letöltés is szintén szükséges funkció.

4.2.4 Törlés

Előfordulhat, hogy egyes képekre már nincsen szükségük a felhasználóknak, erre szolgál a törlés funkció. Egyes képeink különös értékkel bírnak, ezért el kell kerülni a véletlen törlés lehetőségét: első alkalommal csak archiválásra kerülnek a képek, ami harminc napig még elérhető lesz a rendszerben, ezután kerülnek csak véglegesen törlésre. Ezen időn belül a képek bármikor visszaállíthatóak.

4.2.5 Albumokba rendezés

Csak úgy, mint a nyomtatott fotókat, a digitálisakat is érdemes rendszerezni a könnyebb átláthatóság érdekében, erre szolgál az albumokba rendezés funkció. A felhasználók manuálisan csoportosíthatják képeiket, albumaikat elnevezhetik.

4.2.6 Keresés / szűrés, mesterséges intelligencia

Ugyan az albumokba rendezés egy hasznos funkció, azonban időigényes és egy online kép tároló alkalmazás ennél sokkal több lehetőséget rejt kereshetőség és rendszerezés szempontjából. Minden feltöltött kép a Google megoldásához hasonlóan utólagosan feldolgozásra kerül majd mesterséges intelligenciával, ami többek között lehetővé teszi az arcok és érzelmek felismerését. A feltöltött képek továbbá tartalmazhatnak adatokat az elkészülés időpontjáról, ezt az alkalmazás szintén feldolgozza és menti az adatbázisba.

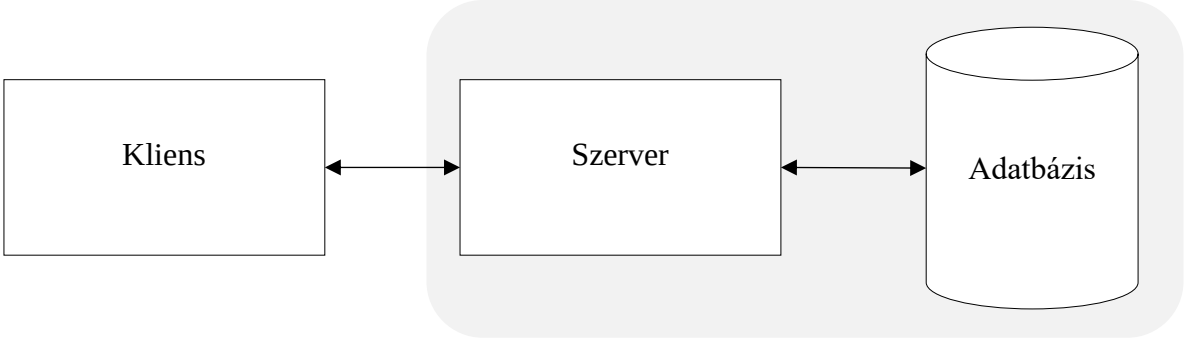
Ezen információk alapján lehetőség lesz különböző összetett szűrés megvalósítására a következő alapján:

- kép készültének ideje,
- felismert érzelmek az arcon
    - boldogság
    - szomorúság
    - mérgesség,
    - undor,
    - meglepődöttség,
    - nyugodtság,
    - félelem

Ezeket a feltételeket a felhasználók tetszőleges kombinációban alkalmazhatják.

5. FEJLESZTÉSHEZ SZÜKSÉGES TECHNOLÓGIÁK

A Felhasználói funkciók fejezetben említettem, hogy miért esett a választásom arra, hogy az alkalmazás egy weboldal legyen. Ehhez szükséges tisztázni, hogyan is néz ki egy webalkalmazás architektúrája egy magasabb absztrakciós szinten, amit a 5.1. ábra szemléltet.



5.1. ábra - Webalkalmazások architektúrája

Az oldal felkereséséhez szükségünk lesz először is egy böngészőre, ami a legtöbb operációs rendszeren alapból telepítve van. A böngészők a weboldalakat dolgozzák fel különböző információk lekérésével és megmutatják nekünk olyan módon, amit mi értelmezni tudunk, egy felület formájában.

5.1 Kliens oldali technológiák [3]

Kliens oldal lényegében azokat a technológiákat fogja össze, amivel a felhasználói felületet készítjük el. Ezeket a fájlokat a távoli kiszolgálótól éri el a felhasználó, amikor felkeres egy oldalt, majd az adott böngésző kezdi el feldolgozni és futtatni őket. Az évek során nem sokat változtak, főleg újabb változatok jelentek meg belőlük, amivel egyszerűbb a fejlesztés és kiszolgálja a modern igényeket.

Technológia	Leírás
HTML	A weboldalunk szerkezetét, struktúráját jelölő nyelv.
CSS	Az oldalunk stílusát írhatjuk le vele. Beállíthatunk vele színeket, betűtípusokat, animációkat, lényegében mindent, ami az oldalunk kinézetével kapcsolatos.
Javascript	A böngészők által használt alapértelmezett programnyelv. Az oldal interakciójához szükséges programsorokat ebben definiálhatjuk. Például: mi történjen, ha egy gombra kattintok.

5.1. táblázat – Kliens oldali technológiák

A leglényegesebb fejlődés a Javascripttel történt, amiről azt kell tudni, hogy egy gyengén típusos nyelv. Ez sok, futás idejű hibát hordozhat magában, erre fejlesztette ki a Microsoft a TypeScriptet [4] ami nevéből is adódóan erősen típusossá és egyéb funkciókkal ruházza fel a Javascriptet. A TypeScript Javascriptre fordul, így semmilyen extra függőségre nincs hozzá szükség a böngészőben való futtatáshoz, ezért használni fogom az alkalmazásban.

Elérhetőek olyan modern kliens oldali Javascript alapú keretrendszerek, amik a fentebb (5.1. táblázat) említett technológiákat foglalják egybe, illetve kibővítik az objektum-orientált programozás koncepcióival. A HTML alapvetően nem alkalmas arra, hogy az oldalunk egy-egy részét komponensekre bontsuk és újra felhasználjuk, a keretrendszerek ebbe is segítenek.

Rengeteg lehetőség közül választhatunk, viszont, ha figyelembe vesszük, hogy melyek a legkiforrottabbak és melyekhez érhető el a legtöbb segítség, **két közismert nagy cég jelöltjeire szűkül a kör: React – Facebook, Angular – Google.**

5.1.1 React és Angular összehasonlítása

A két keretrendszer összehasonlításánál [5] a következő szempontokat veszem figyelembe:

- tanulási görbe,
- külső, nyílt forráskódú könyvtárak,
- népszerűség,
- méret,

- licensz.

Tanulási ív

Mind a két keretrendszerrel volt már szerencsém dolgozni, ami alapján a következőket állapítottam meg: az Angular egy sokkal komplexebb, többet nyújtó keretrendszer, ami nagy cégeknek és nagyobb alkalmazások esetén hasznos lehet. A dokumentáció hozzá részletes, azonban elsajátítása a fejlesztés kezdeti fázisában több időt igényel.

Ezzel szemben a React egy kisebb, de flexibilisebb keretrendszer, használata véleményem szerint rövidebb idő alatt megtanulható. A dokumentáció szintén kielégítő hozzá.

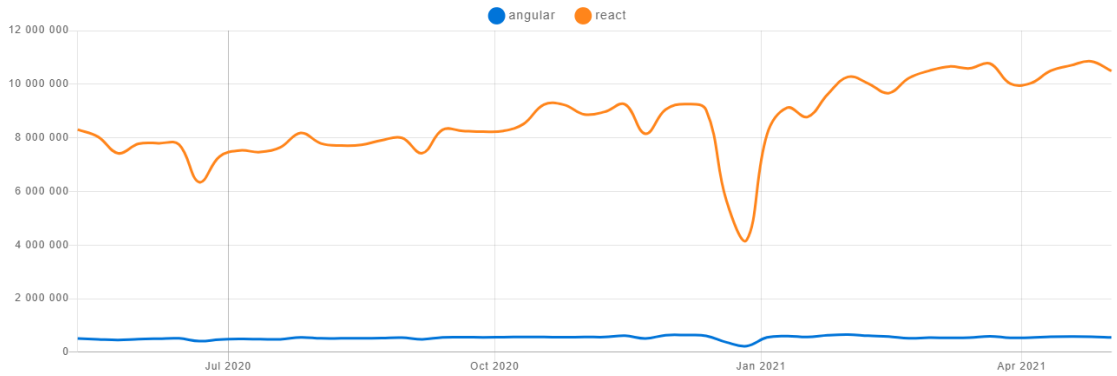
Külső, nyílt forráskódú könyvtárak

Külső, úgynevezett *3rd party* könyvtáraknak nevezzük, mások által elkészített segéd funkciókat, vagy akár vizuális elemeket, amiket felhasználhatunk az alkalmazásunkhoz. A fejlesztés során ezeket is célszerű figyelembe venni, hiszen lényegesen lerövidítheti az implementáció költségét, ha nem kell mindent saját magunknak elkészíteni.

A kliens oldali világban ezeket az NPM csomagkezelő segítségével telepíthetjük. Az oldalt felkeresve a tisztán látható, hogy az egyes keretrendszerekhez hány különböző csomag érhető el a megtekintés pillanatában. Angular esetén ez a szám [6]: **11720**, míg a React esetén [7]: **71754.**

Népszerűség

Népszerűség alatt két metrikát veszek figyelembe: letöltések száma, közösségi kedveltség. Ezek kinyerhetőek az előbb említett csomagkezelő, az NPM és a GitHub, a legnépszerűbb online forráskód kezelő adataira épülő oldal [8] alapján. A letöltések száma az elmúlt évben a következőképpen alakult (**Hiba! A hivatkozási forrás nem található.**):



5.2. ábra - React és Angular letöltési adatai az elmúlt évben (forrás: [8])

Tisztán látható, hogy a **React letöltési száma átlagosan négy-ötször több** mint az Angular-nak. A GitHub adatai alapján az Angular **~731000 db** csillagot, míg a React **~168000 db**-ot kapott. Ennek alapján a React kedveltebb a fejlesztők körében.

Méret

A keretrendszerek méretét azért fontos figyelembe venni, mert az általunk írt kódhoz hozzáadódik vagyis az oldal felkeresése során ezeket is le kell töltenünk. Kisebb méretű keretrendszer esetén több milliszekundumot, adott esetben például lassabb internet *(3G hálózat)* másodperceket is megspórolhatunk. A következő adatokat találtam az interneten [9]: **Angular ~ 143 Kilobyte; React ~ 31.8 Kilobyte.**

Licensz

Természetesen nem szeretném egyik keretrendszert sem jogtalanul felhasználni, illetve fizetni sem feltétlenül szeretnék érte. Mind a két keretrendszer nyílt forráskódú, szabadon elérhető a már említett „forráskód-tárhelyen" a GitHubon, ahol a licensz is szerepel. Mindkettő **MIT** licensszel érhető el, amiről azt az információt találtam [10], hogy mind üzleti, mind saját célokra ingyenesen és szabadon felhasználható.

5.1.2 CSS keretrendszer

A Felhasználói funkciók fejezetben említettem, hogy a webalkalmazások bármilyen készüléken, például okostelefonon is elérhető, azonban ez nem jelenti azt, hogy automatikusan ott is könnyedén használható. Ez alatt azt értem, hogy egyes kijelző mérettől függően változó lehet az oldal használhatósága. Ha például egy weboldalt csak asztali, nagy méretű kijelzőre terveztek az mobiltelefonon jelentősen lecsökkenti a felhasználói élményt, nehezen használható.

Szerencsére több megoldás is létezik erre a problémára. Hasonlóan az előző fejezetben említett két nagy cég Javascript keretrendszeréhez, úgy CSS keretrendszereket is előszeretettel fejlesztenek multinacionális vállalatok és publikálják mindenki számára felhasználhatóvá, ingyenesen. Leegyszerűsítve a következőképpen működnek: töréspontokat definiálnak, amik az egyes kijelző méreteket jelentik. Az oldal elrendezése így dinamikusan a különböző képméretekhez igazodik. Az ilyen oldalakat nevezzük **reszponzívnak**.

A különböző keretrendszerek ellen vagy mellett nehéz érvelni, ugyanis működési elvük szinte teljes mértékben megegyezik. Az egyes megvalósított dizájn elemek kinézete különbözik (gombok, táblázatok stb.) ami egyéni ízlés kérdése. Természetesen, amivel itt is számolni kell az a felhasználhatóság (licensz) és a népszerűség.

A két legnépszerű alternatíva a Twitter – Bootstrap [11] és a Google – „Material design" [12] elveit követő komponens könyvtárak. Mind a kettőből elérhető React-tal

kompatibilis verzió, szintén ingyenesen **MIT licensz** [10] alatt. Egyéni preferencia, azaz kinézet alapján én a Google változatát fogom felhasználni.

5.2 Szerveroldali technológiák

Szerver oldalon valósítjuk meg azokat a dolgokat, amit kliens oldalon nem szeretnénk publikussá tenni – ugyanis a böngészőben futó kód bárki által szemrevételezhető – azaz az üzleti logikát. Üzleti logika alatt értem például egy bejelentkezés folyamatát megvalósító függvényeket, eljárásokat továbbá mindent, ami az alkalmazás funkcióinak működéséhez szükséges. Ahogyan az 5.1. ábra - Webalkalmazások architektúrája mutatja, a szerver oldal felelős az adatmanipulációért is, kizárólag rajta keresztül érjük el az adatbázist.

A szerveroldali kódunk egy távoli kiszolgálón (szerveren) fut, jellemzően Linux/Unix alapú vagy Windows operációs rendszeren. A Linuxot futtató szerverek üzemeltetési költségei lényegesen olcsóbbak, ezért célszerű figyelembe venni a szerveroldali programozási nyelv, keretrendszer kiválasztásánál, hogy mely platformot támogatja. A legkonzekvensebb döntés egy cross-platform nyelv / keretrendszer választása, ami növeli a rugalmasságot a „hosting környezet” kiválasztásánál.

Megszámlálhatatlan lehetőség közül választhatunk, ezeknek részletes összehasonlítása, elemzése számtalan oldalt venne igénybe, ezért az egyetemi tanulmányaim során megismert két nagy hírű nyelvet, illetve keretrendszert fogom számba venni: a C# -.NET Core-t (Microsoft) és a Java-t (Oracle).

5.2.1 C# - .NET Core és Java összehasonlítása

A téma címéből is adódóan kritérium, hogy az alkalmazás cross-platformon működjön, ezért elsősorban ez a legfontosabb szempont. A Java nagy előnye, hogy kezdetektől fogva bármely rendszeren használható volt. A Microsoft 4-5 éve jelentette be az úgynevezett .NET Core keretrendszerét, ami napjainkban már stabillá forrotta ki magát és nem csak Windows környezeten fut, ezzel is leküzdve a hátrányát.

A két nyelv alapvetően szintaxisában nagyon hasonló, azonban a C# tartalmaz modernebb nyelvi elemeket, amelyekkel egyszerűbb a fejlesztés. Mind a két nyelv és keretrendszer tökéletesen megfelel az elvárásoknak, azonban a C# használatát mélyebben tanuljuk az egyetemen, illetve munkám során is napi szinten használom. Java választásával a fejlesztés nagy része a technológia és használatához szükséges infrastruktúra, eszközök kiismerésével telne, ami időben rossz hatással lenne a fejlesztésre. Ezért a C#, illetve .NET Core keretrendszert választom.

5.3 Adatbázis kezelés

Az alkalmazáshoz szükséges adatokat is tárolni kell, ehhez szükség lesz egy adatbázisra, amit a tárolás struktúrája szerint két típusra bonthatunk: relációs és nem relációs.

A relációs adatbázisban az adatokat táblaként tároljuk. A tábla oszlopai jelentik az egyes tulajdonságokat, rekordjai (sorai) pedig az adott egység elemeit. Minden sorhoz tartozik egy egyedi azonosító, ami alapján az egyes entitásainkat összelehet kapcsolni. Ezt az egyedi azonosítót más táblában idegen kulcsnak nevezzük.

Nem relációs adatbázist struktúra szerint négy típusra bonthatjuk [13]:

Kulcs-érték: A legegyszerűbb típusú a nem relációs adatbázisok között. A nevéből is adódóan minden (egyedi) kulcshoz egy adatot tudunk hozzárendelni. Nagyon gyorsan írhatunk és olvashatunk ki belőle,

Gráf struktúra: csomópontok és élek halmaza. Ezekhez hozzárendelhetünk tulajdonságokat, például kulcs-érték párokat. Akkor hasznos, ha bejárásokat szeretnénk végrehajtani, ez gráf szerkezetben rendkívül gyors.

Oszlop-tár: Relációs adatbázisokban egy sor lekérdezése gyors folyamat. Oszlop-tár struktúrájú adatbázisoknál egy adott oszlop adatait lehet gyorsan lekérdezni.

Dokumentum: az adatokat „dokumentumokban”, jellemzően XML vagy JSON formátumban tároljuk.

Látható, hogy a két modell lényegesen eltér egymástól, a leglényegesebb különbség azonban az, hogy a relációs adatbázis esetében egy adott entitás tulajdonságaihoz egy adatot tudunk tárolni soronként. Mi alapján lehet akkor dönteni? Az adatbázissal kizárólag a szerveren futó programunk fog kommunikálni, amihez elérhető olyan segédeszköz az úgynevezett **O**bject **R**elational **M**apping amivel az egyes tábláinkat az adott nyelv osztályaivá modellezhetjük le és azokat kezelhetjük a kódunkból. Ezzel kiküszöböljük, hogy az adatbázis lekérdező nyelvéhez is értenünk kelljen, használata rendkívül kényelmes. A fentebb választott szerveroldali keretrendszerhez ORM támogatás jelenleg csak néhány relációs-adatbázis motorhoz érhető el, így a kör leszűkül a következőkre:

- Microsoft SQL Server,
- PostgreSQL,
- MySQL.

C#/.NET Core alkalmazások tekintetében a legnépszerűbb ORM csomag az Entity Framework Core. [14] Az Entity Framework Core Microsoft fejlesztés révén a legjobb kompatibilitást az Microsoft SQL Serverhez nyújtja, így a választásom erre a technológiára esett.

5.4 Protokollok

A kliens és szerver közötti kommunikációhoz szükséges protokollt fogom jellemezni a következő két alpontban.

5.4.1 A HTTP Protokoll [15]

A HTTP protokoll egy üzenet alapú modellt használ, ahol a kliens – ez esetünkben a böngésző – elküld egy üzenetet (5.5. ábr) a szerver felé, a szerver pedig válaszként szintén egy üzenetet küld vissza. Minden üzenet első sora szóközzel elválasztva a kérés típusa, végpontja és a protokoll verziója. Ezen kívül tartalmaz még fejléceket *(headers)*, amik a kérés különböző tulajdonságait írják le, illetve egy opcionális törzs *(body)* részt, amiben további adatokat juttathatunk el vagy válaszként kaphatunk a szervertől. A HTTP különböző metódusokat definiál az egyes kérésekhez, ilyen például két leggyakrabban használt GET és a POST. Ez a metódus jelenik meg a kérés első sorának első paramétereként. A GET a funkciója, hogy letöltsön egy adott forrást a szervertől. Ha adatokat szeretnénk továbbítani a szerver felé, azt a kérés URL-jében vagy a törzsben tehetjük meg a POST metódus használatával. A **U**niform **R**esource **L**ocator egy webforrás helyét írja le a következő formátumban: „protocol://hostname[:port]/[path/]file[?param=value]”.

A szervertől érkezett válasz hasonló módon épül fel, egyedül az első sorában különbözik.

```
GET http://uni-obuda.hu/ HTTP/1.1
Host: uni-obuda.hu
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:53.0) Gecko/20100101
Firefox/53.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: hu-HU,hu;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Cookie: SESS5271c37b9aa6f51d903b4e8f7292bcf0=191u5r80563rbebjnd829c5k84;
Connection: keep-alive
```

5.5. ábra - Egy példa HTTP kérés

A válasz esetében ugyanis a HTTP verzió, HTTP kód és a HTTP kód szöveges magyarázata kerül a sorba. Kulcsfontosságú szerepet töltenek be még a HTTP sütik. A sütiket a szerver a válaszban állíthatja be a *Set-Cookie* fejléc segítségével, amit a böngésző eltárol és minden egyes kéréshez automatikusan hozzáad a *Cookie* fejléchez, több süti esetén pontosvesszővel elválasztva.

A szerver a sütihez további tulajdonságokat rendelhet, amik a web alkalmazás biztonságára vannak hatással, ezáltal előírják, hogy a böngésző hogyan kezelje őket. Ilyen tulajdonságok lehetnek:

- *expires:* beállítja, hogy meddig érvényes az adott süti,
- *domain:* biztonsági okokból minden egyes sütihez csak az adott szerver férhet hozzá, ami ezzel állítható be,
- *path:* leírja, hogy milyen útvonalhoz érvényes a süti,
- *secure:* ha ez a mező be van állítva, a süti csak biztonságos csatornán keresztül továbbítható,
- *HttpOnly:* ha ez a mező be van állítva, a böngészőben futó kód (JavaScript) által nem érhető el a süti.

A protokoll sajátos tulajdonsága, hogy alapértelmezetten állapotmentes. Ez azt jelenti, hogyha egy felhasználó bejelentkezik az oldalon, majd bezárja és visszatér, újra be kell jelentkeznie, mert az állapot nem maradt meg. Lehetséges azonban az állapotot megtartani, mégpedig sütikkel. Ezt a meghatározott ideig fenntartott állapotot *session*-nek nevezzük. Ahogy az 5.5. ábrán látható, a *Cookie* fejléc egy egyedi azonosítót tartalmaz, ami segítségével a szerver azonosítja az adott állapothoz tartozó adatokat, ezáltal fenntartva azt.

Alapvető probléma a HTTP protokollal, hogy az üzenetek bárki számára olvasható formában utaznak a hálózaton. A hálózat különböző szoftverek segítségével viszonylag egyszerűen megfigyelhető, különösen akkor, ha valamilyen nyilvános hálózaton böngészünk. Ebből adódik, hogy egy nyilvános internet kapcsolaton keresztül böngésző felhasználó, ha bejelentkezik az alkalmazásba, akkor a név és jelszó páros a kérésben tisztán kivehető. Ennek a megoldására lett kifejlesztve a **S**ecure **S**ocket **L**ayer protokoll, amivel a HTTP protokoll kiterjeszthető, így alkalmas biztonságos kapcsolat létesítésére. Az így kiterjesztett protokollt HTTPS-nek nevezzük.

5.4.2 Az SSL Protokoll [16]

A Secure Socket Layer protokoll a nyilvános kulcsú titkosítást használja a hálózati forgalom védelmére. Kizárólag egyetlen csatornát véd és nem az egész hálózatot. Az SSL működéséhez egyaránt támogatásra van szükség kliens és szerveroldalon, szerencsére napjainkban ez már nem okoz problémát.

A modernebb böngészők például már figyelmeztetnek arra is, ha szenzitív adatokat szeretnénk továbbítani a szerver felé és ez nem HTTPS-en keresztül történik.

A kapcsolat létesítésekor szükséges meghatározni a protokoll verzióját, erre a kliens tesz javaslatot az első kérés során, egy titkosító algoritmussal együtt. A szerver válaszában visszaküld egy kulcsot a saját tanúsítványával együtt. Kéri továbbá a kliens tanúsítványát. A kliens visszaküldi a saját tanúsítványát, aminek hitelességét a szerver ellenőrzi egy megbízható, harmadik fél által, amit *Certification Authority*-nek nevezünk. Ha a szerver

mindent rendben talált, a kliens küld egy kulcsot a szerver felé ellenőrzési célokra, majd a szerver vissza jelez, hogy készen áll. Ezután a folyamat lezárása következik, a kapcsolat innentől kezdve biztonságos, az adatok titkosítva utaznak a hálózaton.

5.5 Verziókezelés

Napjainkban elengedhetetlen szoftverfejlesztés során verziókezelő rendszer használata, ami lehetővé tesz, hogy egyes pontokon *(megj.: úgynevezett „commitokon”)* az alkalmazásunkat elmentsük és később akár bármelyik pontra visszaálljunk. Az ilyen rendszerek teszik lehetővé továbbá a csapatok tagjai által fejlesztett rész-kódok egyszerűbb integrálását is. A szoftvert ugyan egyedül fogom fejleszteni, azonban ennek ellenére a későbbiekben mindenképp hasznos lehet a használata. Ingyenesen elérhetőek hozzá különböző tárhelyek is, ahol egy biztonsági mentést tárolhatunk az adott verziónkról. Erre a célra a legnépszerűbb rendszert fogom használni, a Gitet [17].

5.6 Felhőszolgáltatók

Modern webalkalmazásokat felhőszolgáltatók segítségével szokás üzemelni. A legnagyobb 3 felhőszolgáltatók a következők: Amazon Web Services, Microsoft Azure és a Google Cloud Platform. Az alkalmazás szemszögéből nézve az említett szolgáltatók közül mind tökéletesen alkalmas, így a következő szempontok alapján fogok választani: ár, elérhető funkciók száma, illetve az érzelem felismerő szolgáltatás hatékonysága.

A három szolgáltató közül az Amazon rendelkezik a legtöbb felhasználóval és funkcióval. Ár tekintetében az igénybe vett szolgáltatásoktól függően eltérő eredményeket kapunk, azonban mind a három szolgáltató ingyenes elérhetőséget ad a legtöbb funkcióra az új felhasználóknak 1 évig. Bizonyos szolgáltatásokra pedig használattól függően válnak fizetőssé.

A mesterséges intelligencia alapú kép felismerő funkciók szintén mind a három szolgáltatónál elérhetőek. Az interneten fellelhető információk, tesztek alapján azonban az Amazon megoldása a legmegbízhatóbb [18].

Ezek alapján az általam választott felhőszolgáltató az alkalmazás kiszolgálására az Amazon Web Services.

5.7 Összegzés

A kiválasztott technológiákat az 5.2. táblázat - Választott technológiák szemlélteti.

Technológia	Alternatíva	Választott
Javascript keretrendszer	AngularJS	ReactJS
CSS keretrendszer	Bootstrap	Material
Szerver oldali programozási nyelv / keretrendszer	Java	C# .NET Core
Adatbázis	PostgreSQL	MSSQL
Protokoll	HTTP	HTTPS
Verziókezelés	SVN	Git
Felhőszolgáltató	Microsoft Azure	Amazon Web Services

5.2. táblázat - Választott technológiák

6. RENDSZERTERV

Az előző fejezet alapján tehát az *Amazon Web Services* (továbbiakban AWS) felhőszolgáltatóra esett a választásom, ebben a fejezetben ismertetem az alkalmazás tervét komponens, illetve folyamatok szintjén. Ismertetem továbbá, hogy milyen felhőszolgáltatásokat veszek igénybe az egyes komponensekhez, illetve az adatbázis struktúráját is.

6.1 Komponensek

A legmodernebb alkalmazások már az úgynevezett „serverless” szolgáltatásokat alkalmaznak, amelynek előnye, hogy nem egy folyamatosan futó virtuális gépen fut az alkalmazás, hanem igény szerint a felhőszolgáltató, használat alapján (pl. oldal megnyitására) kezeli a hozzátartozó erőforrást. Ez mind ár és mind skálázhatóság szempontjából kedvező, hiszen egyes erőforrásokért csak akkor fizetünk amikor az alkalmazásunk ténylegesen használatban van. Ezen elvek mentén tervezem és mutatom be a következő pontokban az alkalmazás felépítését.

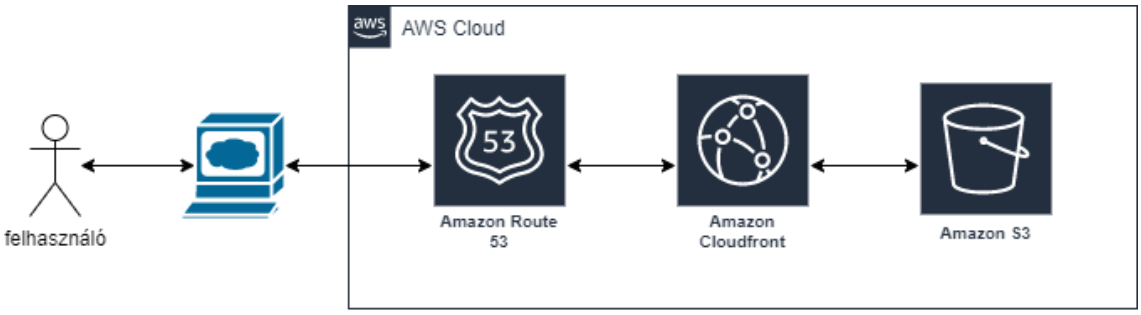
6.1.1 Kliens oldal

A kliens oldali fájljaink tárolására szükségünk lesz egy tárhelyre, ez az AWS-en belül a *„Simple Storage Service”* rövidebb nevén az S3. Az S3 szolgáltatáson belül úgynevezett „bucketeket” (tároló rekesz, vödör) hozhatunk létre a föld különböző régióiban. Ezek a „bucketek” lényegében kulcs-érték alapú tárhelyek [19].

A webalkalmazásunk a világ minden pontjáról elérhető, így, ha például Európában hozzuk létre a tárhelyünket, de egy felhasználó Amerikából szeretné elérni akkor az oldal betöltése lassú lesz. Ezért szükség lesz még egy úgynevezett „Content Delivery Network” (CDN) szolgáltatásra, ami a weboldalunk elérési idejét gyorsítja. Az AWS-en belül ez a „*CloudFront*” [20]. Ez a szolgáltatás lényegében egy gyorsítótárként fogható fel. Az AWS rendelkezik a világ minden pontján szerverekkel, amiket megfelelő konfiguráció után (esetünkben a tárhelyünkre kötve) automatikusan szétosztja köztük. A weboldal megnyitásakor a kérésünket ez a szolgáltatás mindig automatikusan a hozzánk legközelebb eső szerverhez fogja irányítani, így a betöltés gyors lesz.

Alap esetben a CloudFronton egy egyedi azonosító alapú webcímet kapunk, például: http://d111111abcdef8.cloudfront.net/. Ez a felhasználók számára nem túl barátságos, szükségünk lesz még egy DNS szolgáltatásra is, ahol megvásárolhatunk egy tetszőleges, még szabad, elérhető webcímet. Ez a szolgáltatás az AWS-en belül a „Route 53” [21].

Ezek alapján a kliens oldal architektúráját a 6.1. ábra - Kliens oldal architektúrájamutatja.



6.1. ábra - Kliens oldal architektúrája

6.1.2 Szerver oldal

A kliens oldalt adatokkal kell kiszolgálni, kell kiszolgálni ezért szükségünk lesz egy API-ra. Ahogy a fejezet elején említettem, „serverless” architektúrát fogok alkalmazni, az AWS megoldása erre az úgynevezett Lambda függvények. Működése röviden a következő: feltöltjük a futtatni kívánt kódunkat egy archívum fájlban, az Amazon igény esetén ezt az archívum fájlt kicsomagolja, majd futtatja a kódunkat. A Lambda függvények futását különböző eseményekkel lehet kiváltani az AWS ökoszisztémán belül. A webalkalmazásunk szemszögéből ez az esemény egy http kérés lesz, amire az AWS az úgynevezett „API Gateway” [22] szolgáltatásán keresztül nyújt lehetőséget. Ez lényegében egy interfész a Lambda függvényeink előtt, ami kezeli és tovább irányítja a http kéréseket.

Természetesen szükség lesz még egy adatbázisra, amiben az alkalmazáshoz szükséges entitások adatait tároljuk. Az AWS erre a „Relational Database Service” -t (RDS) nyújtja, ahol elérhető a korábbi fejezetben választott adatbázis motor is. [23]

A képek tárolására nem érdemes relációs adatbázist használni, költséghatékonysági okokból erre az előzőek során említett S3 kulcs-érték alapú tárhely a megfelelő, így a felhasználók által feltöltött képeket itt fogom elhelyezni.

A feltöltött képeken az arc és érzelem felismeréshez továbbá szükség lesz még mesterséges intelligencia alapú szolgáltatásra. Ilyen megoldást fejleszteni rengeteg időbe telik és az eredmény sem biztos, hogy a legmegbízhatóbb lenne, szerencsére az AWS erre is megoldást kínál. Ez a szolgáltatás a következő néven érhető el: Amazon Rekognition [24].

Az Amazon Rekognition segítségével rengeteg információt kinyerhetünk képekről:

- tárgyakat,
- aktivitásokat,
- szövegeket,
- arcokat és hozzá tartozó tulajdonságokat (pl.: érzelem)

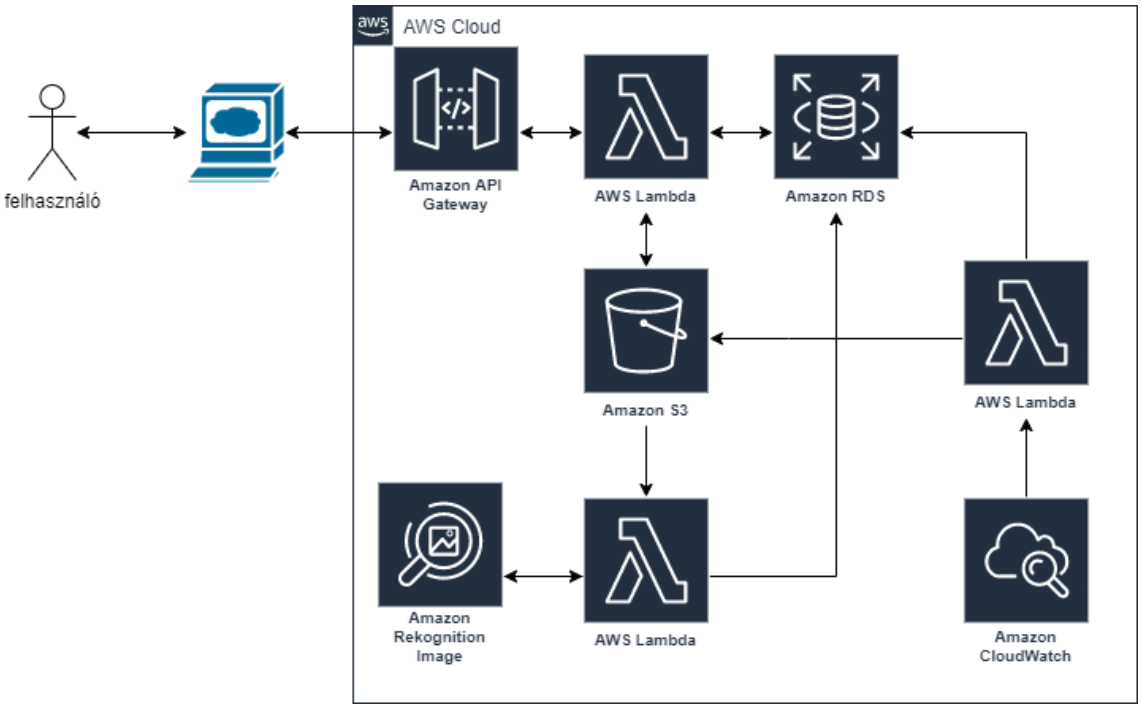
- híresség**e**ket.

A feltöltött képekből az arc és érzelem adatokat tehát ennek a szolgáltatásnak a segítségével fogom kinyerni a következő módon:

1. felhasználó feltölt egy képet, ami bekerül az S3 bucketba,
2. a feltöltésre egy esemény kivált egy másik, API-tól független Lambda függvény futását
3. ez a Lambda függvény meghívja az Amazon Rekognition szolgáltatást, illetve kinyeri a kép tulajdonságait. (dátum, hely)
4. a visszaadott adatokat (arc, érzelem, dátum, hely) lementi az adatbázisba a hozzátartozó képhez.

A kép törlése funkcióhoz szükség lesz még egy Lambda függvényre, ami minden nap egyszer ellenőrzi, hogy a kép hány napja lett törölve. Harminc napon túl véglegesen törli a képet S3-ból, illetve az adatbázisból.

A szerveroldali architektúrát a 6.2. ábra szemlélteti.



6.2. ábra - Szerveroldal architektúrája

6.2 Felhasználói funkciók

Ebben az alfejezetben az egyes felhasználói funkciókat fogom röviden technikai szempontból jellemezni és szekvencia diagramokkal szemléltetni.

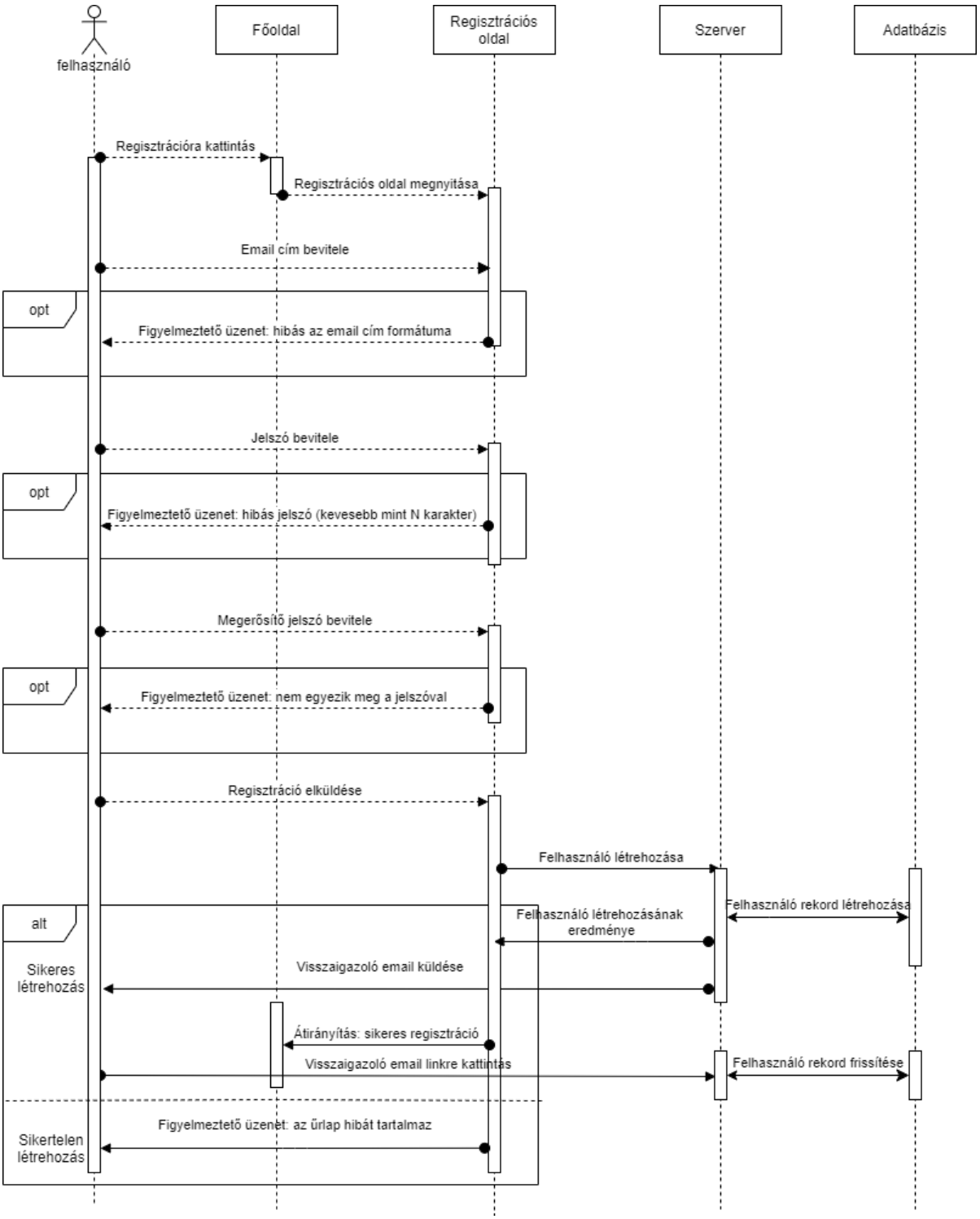
6.2.1 Regisztráció

A felhasználó a weboldal megnyitásakor egy bejelentkező képernyőt lát. Amennyiben még nem rendelkezik felhasználóval regisztrálnia kell. Regisztráció megnyomásakor átirányításra kerül a regisztrációs oldalra/űrlapra, ahol a következő adatokat kell kitöltenie:

- keresztnév,
- veztetéknév
- email cím,
- jelszó

Hibás email formátum, nem megfelelő jelszó erősség vagy ha a jelszó és megerősítő jelszó különbözik a felhasználó visszajelzést kap a felületen. Sikeres kitöltés esetén elküldheti az űrlapot. Az adatok szerver oldalon is ellenőrzésre kerülnek. Amennyiben nincs hiba, létrehozza a felhasználót az adatbázisban. A jelszó természetesen biztonságos formában, hashelve kerül eltárolásra. Sikeres regisztráció esetén továbbá egy megerősítő email is kiküldésre kerül. Ez azt a célt szolgálja, hogy illetéktelen felhasználók ne tudjanak más email címével regisztrálni. Az email egy linket tartalmaz egy egyedi azonosítóval, ami a felhasználóhoz van kötve és maximum két napig érvényes. Kattintásra a szerver lekérdezi az egyedi azonosítóhoz tartozó felhasználó rekordot az adatbázisból, majd elmenti egy mezőbe a megerősítés sikerességét.

Ezt a folyamatot a következő oldalon látható szekvencia diagram (6.3. ábra) ábrázolja.

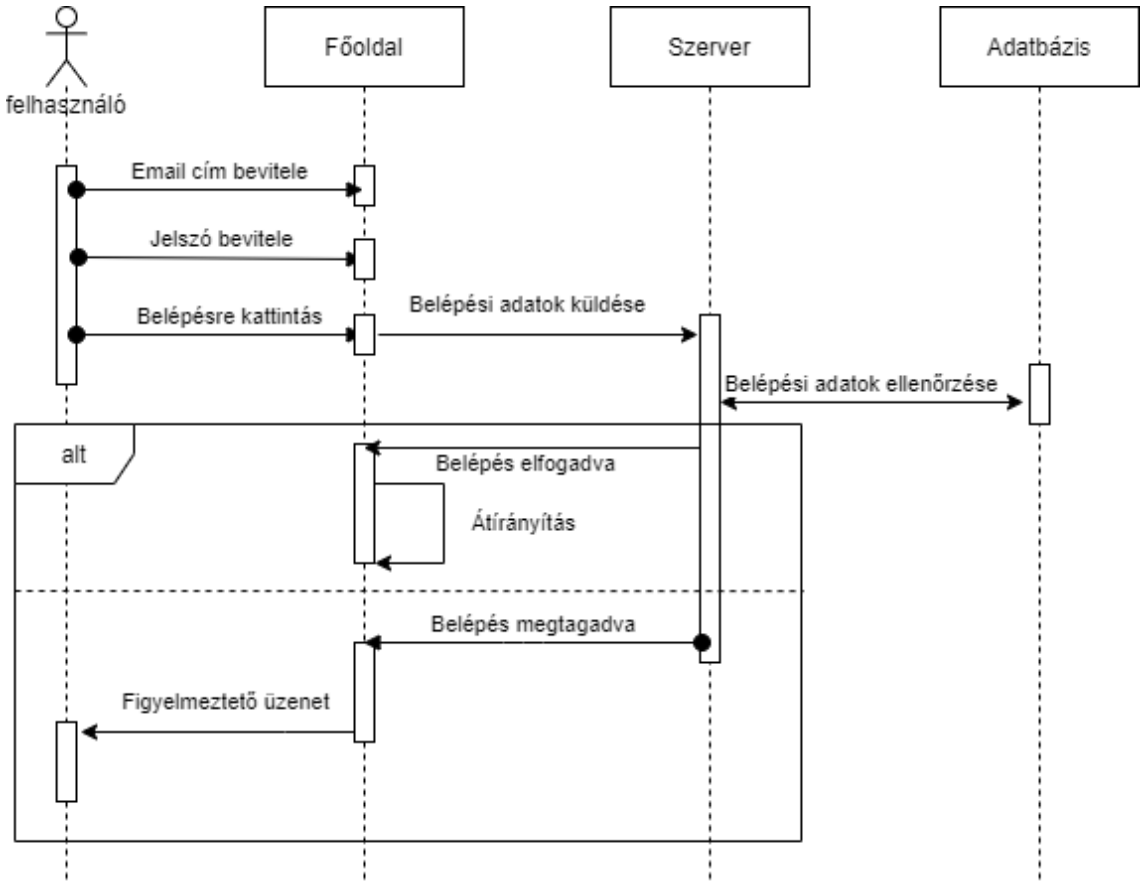


6.3. ábra - A regisztráció folyamata

6.2.2 Belépés

Amennyiben a felhasználó már regisztrált, az alkalmazás megnyitásakor beléphet. Az email és jelszó mezők kitöltése után a belépés gombra kattintva a szerver ellenőrzi az adatokat. A megadott jelszón hasító függvény alkalmazása után lekérdezzük az adatbázisból az email és hasított jelszó páros alapján létezik e felhasználó. Amennyiben igen, a felhasználó sikeresen belépett, a szerver egy Json Web Tokent (JWT) állít ki a felhasználó számára, amit visszaküld a kliens oldalra. Ez alapján a felhasználó egy adott ideig vagy kijelentkezésig belépve marad. Sikertelen bejelentkezés esetén a felhasználó a következő hibaüzenetet lát.

Ezt a folyamatot a 6.4. ábra szemlélteti.



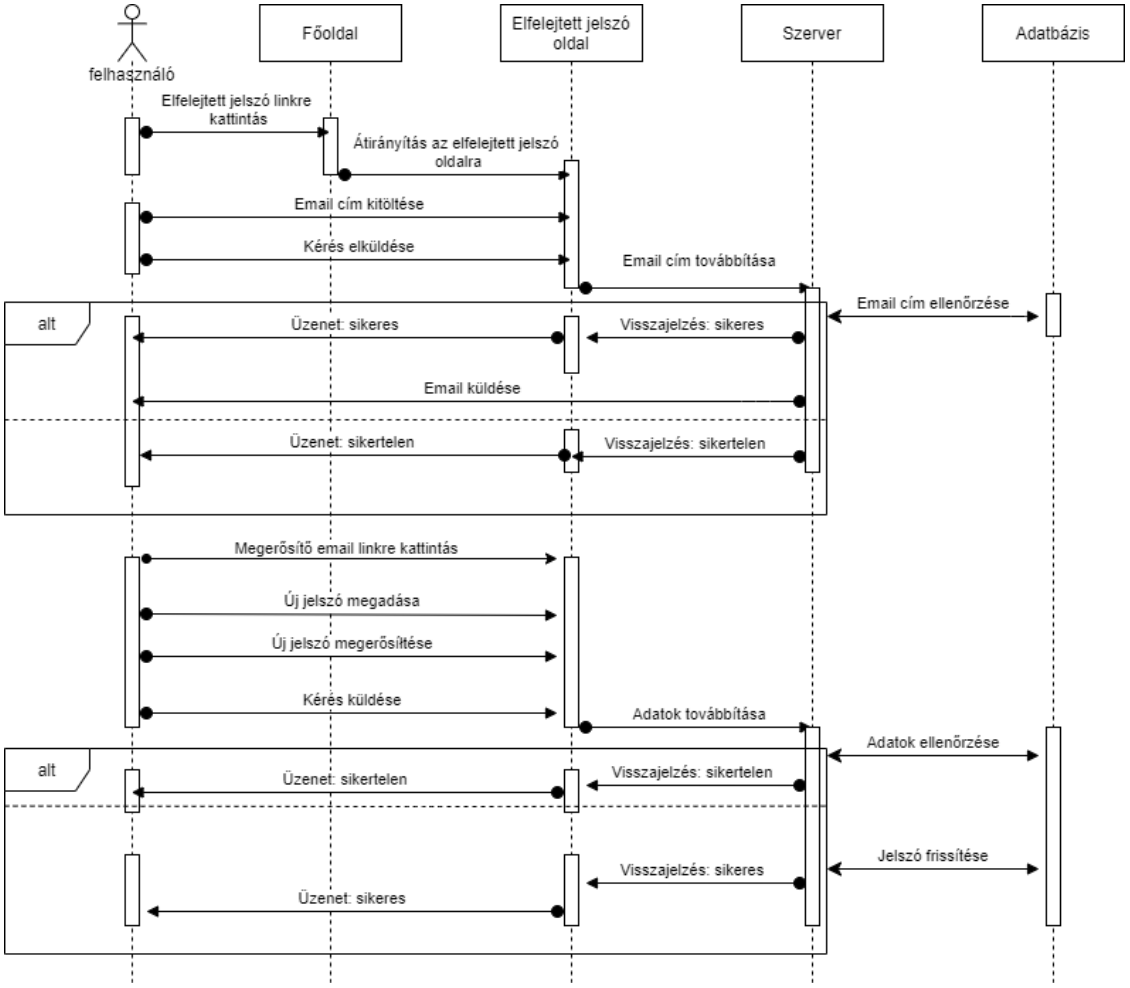
6.4. ábra - A belépés folyamata

6.2.3 Elfelejtett jelszó

Elfelejtett jelszó esetén a felhasználóknak ez a funkció segít abban, hogy visszaállítsák jelszavukat. A regisztráció megerősítés folyamatához hasonló módon fog történni:

1. Felhasználó az elfelejtett jelszó linkre kattint,
2. Kitölti az email címét,
3. Elküldi a kérést,
4. A szerver generál egy két napig érvényes, egyedi azonosítót, amit a felhasználóhoz rendel (email cím alapján),
5. Ezt az azonosítót a szerver kiküldi emailben egy link formájában,
6. A linkre kattintva betölt az oldal két beviteli mezővel: új jelszó, új jelszó megerősítése,
7. Kitöltés és gombnyomás után a szerver felé továbbítjuk az adatokat, az egyedi azonosítóval együtt
8. Amennyiben érvényes az egyedi azonosító a jelszóváltoztatás sikeres

A folyamatot a 6.5. ábra szemlélteti.



6.5. ábra - Az elfelejtett jelszó funkció folyamata

6.2.4 Képfeltöltés

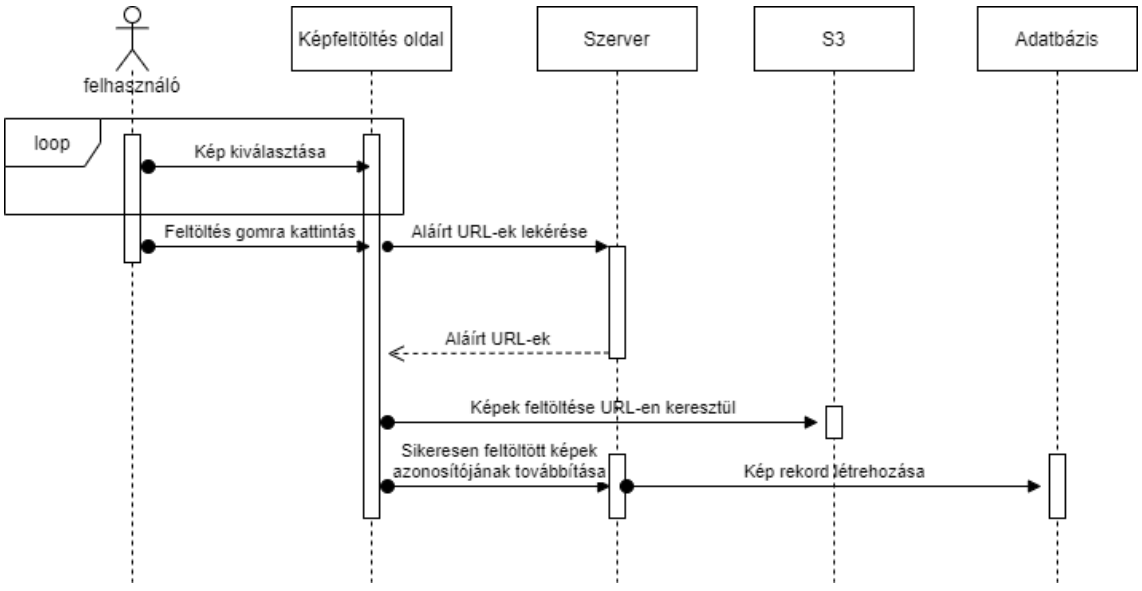
A képfeltöltés funkció technikailag a következő részekből tevődik össze:

- kép fájl eltárolása az S3 bucketben,
- kép rekord létrehozása az adatbázisban,
- kép hozzárendelése a felhasználóhoz.

A kép fájlt eljuttatni az S3 bucketba kétféleképpen lehetséges: az API-on keresztül a Lambda függvény feltölti vagy közvetlen a böngészőből töltjük fel a bucketba. Az utóbbi megoldás előnyösebb, hiszen nem használunk felesleges erőforrást, így ezt a módszert választottam. Az Amazon S3 erre olyan módon nyújt lehetőséget, hogy generálhatunk úgynevezett aláírt URL-eket, amin keresztül egy adott ideig közvetlen a tárhelyre tölthetünk fel a böngészőből. A folyamat tehát ezekből a lépésekből áll:

1. A felhasználó a felületen kiválasztja a feltöltésre szánt képet, képeket,
2. Az API-n keresztül URL-ek lekérése a képekhez,
3. Képek feltöltése közvetlen a tárhelyre
4. Sikeresen feltöltött képekhez az API-n keresztül létrehozzuk az adatbázisban a képet reprezentáló entitást, amit hozzárendelünk a felhasználóhoz.

A folyamat szekvencia diagramját 6.6. ábra mutatja.



6.6. ábra - A képfeltöltés folyamata

6.2.5 Kép letöltése

A felületre a képek már az elérési útvonalukkal érkeznek (adott ideig érvényes, aláírt S3 URL), így az egyes képek letöltése egyszerűen egy gomb és az URL segítségével fog történni.

6.2.6 Kép törlése

A felhasználónak lehetősége lesz a felületen egy vagy több kép törlésére egyidejűleg. Korábban említettem, hogy első sorban ezek a képek csak megjelölve lesznek törlésre, a tényleges 30 napon túli törlést egy külön, periodikusan futó Lambda függvény fogja végezni. A felhasználó kijelölés után megnyomja a törlés gombot, ezeknek a képeknek az azonosítója továbbításra kerül az API-hoz, ami megjelöli törlésre az adatbázisban az adott képeket. A Lambda függvény ezeket a képeket ellenőrzi naponta egyszer, amennyiben harminc nappal korábban lett törölve, véglegesen törli az adatbázisból és az S3 bucketból. Ezen megjelölt képek egy külön oldalon lesznek visszaállíthatóak, archívum menüpont alatt. A visszaállítás a törlés ellentétes folyamata.

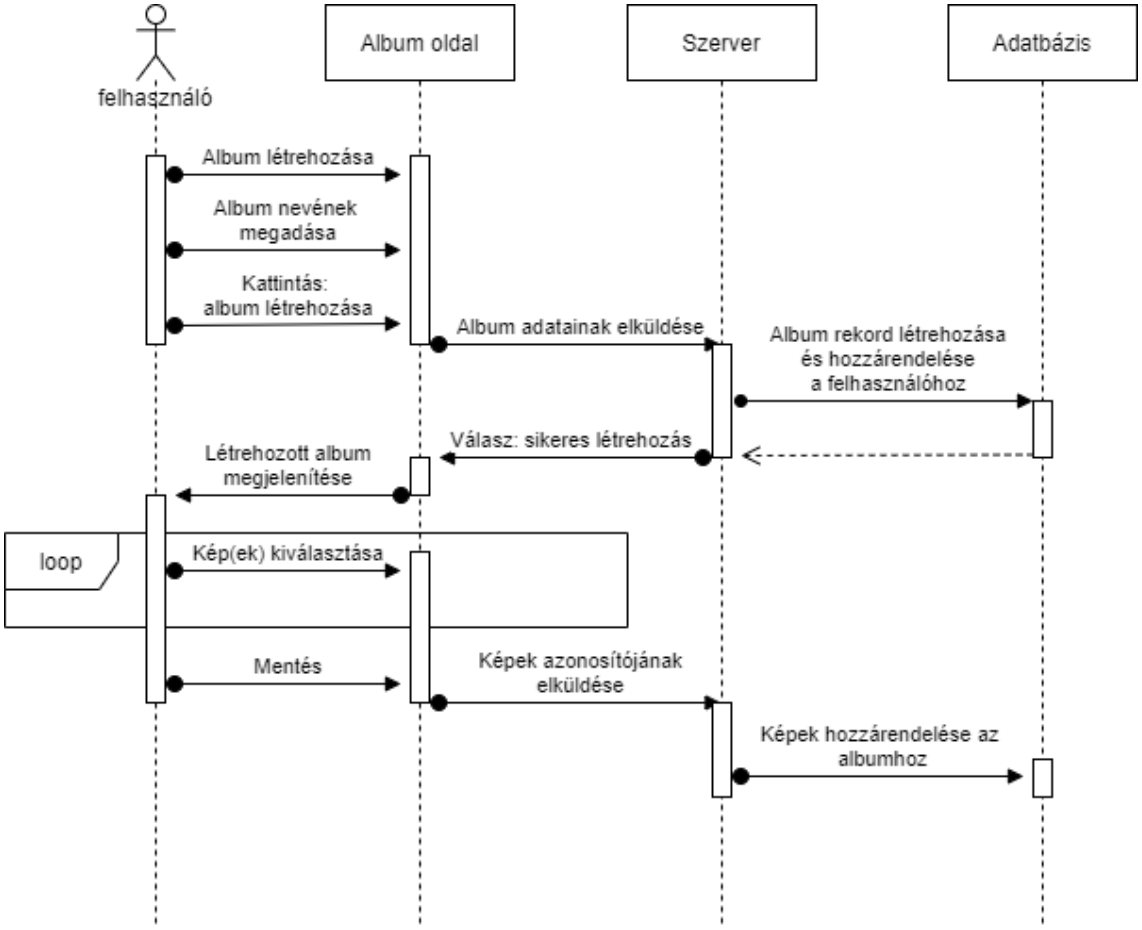
A folyamat szekvencia diagramját 6.7. ábra mutatja.

6.7. ábra - A kép törlésének folyamata

6.2.7 Albumokba rendezés

Az albumokba rendezés funkció két lépésből tevődik össze. A felhasználó létrehozza az albumot a felületen – amihez csak egy nevet kell megadnia - majd az albumra kattintva lehetősége nyílik kiválasztani a képeket és menteni.

A folyamat szekvencia diagramját 6.8. ábra mutatja.

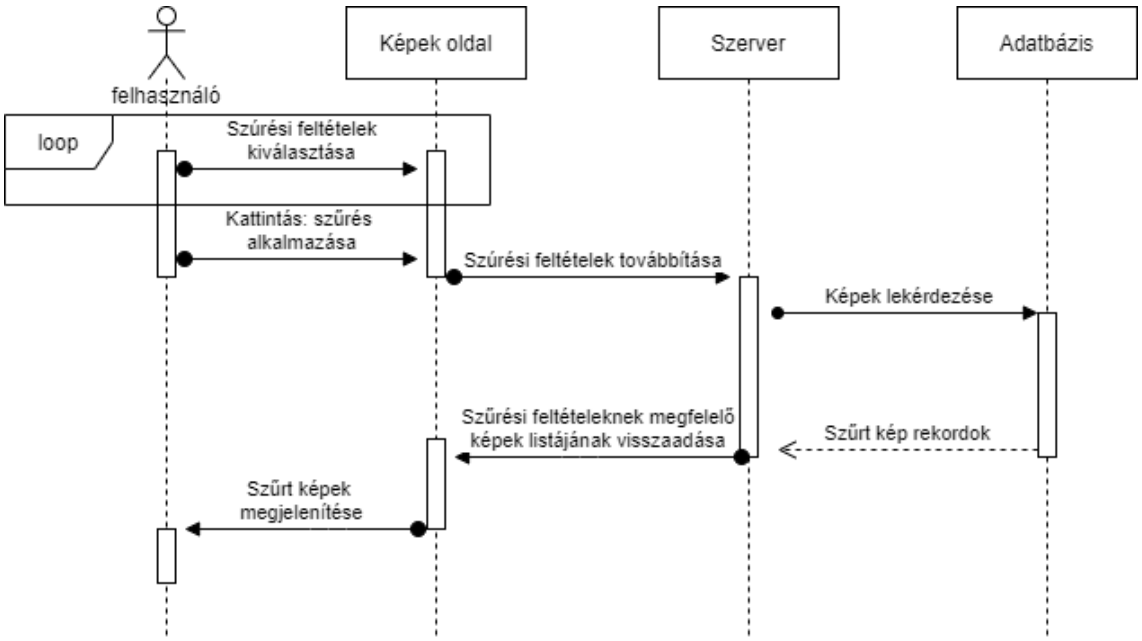


6.8. ábra - Az albumokba rendezés funkció folyamata

6.2.8 Szűrési funkciók

Egy korábban említett fejezetben a képekhez elérhető adatok alapján a felhasználók szűrni tudnak majd arc, dátum és különböző érzelmek alapján. A felületen ez dátum választó, illetve egy „checkbox” beviteli mező formájában fog megjelenni, aminek segítségével bepipálhatjuk, hogy mely érzelemre szűrünk. A képekhez, pontosabban a képekhez felismert arcokhoz tartozó érzelem egy számként érkezik a felhőszolgáltatótól, ami azt mutatja milyen bizonyossággal ismerte fel az érzelmet az arcon az algoritmus. Erre egy küszöböt fogok beállítani, ami minimum 90%-os bizonyosság esetén szűr csak a képre.

A folyamat szekvencia diagramját a 6.9. ábra mutatja.



6.9. ábra - Szűrési funkció folyamata

6.3 Adatbázis séma

Ebben a fejezetben az adatbázis séma tervét fogom bemutatni táblánként, majd a táblák kapcsolatát.

6.3.1 User tábla

A „User” vagyis felhasználó tábla reprezentálja a felhasználót, a következő mezőkkel rendelkezik (6.1. táblázat):

mező neve	**típusa**	**leírás**
ID (PK)	int	a felhasználó egyedi azonosítója
Email (UNIQUE)	varchar	a felhasználó email címe
Password	varchar	a felhasználó jelszava, hashelve
ResetPasswordToken	GUID | null	elfelejtett jelszóhoz használt egyedi azonosító
ResetPasswordTokenValidUntil	date | null	elfelejtett jelszóhoz hasznát egyedi azonosító érvényességi ideje
FaceID (FK)	varchar | null	a felhasználó arcát jellemző egyedi azonosító
ConfirmationToken	GUID	a felhasználó megerősítéséhez használt egyedi azonosító
ConfirmationTokenValidUntil	date	a felhasználó megerősítéséhez használt egyedi azonosító érvényességi ideje
IsConfirmed	bit	megerősítette e a felhasználó a profilját, vagy sem

6.1. táblázat - A User tábla mezői

6.3.2 Image tábla

Az „Image” vagyis kép tábla reprezentálja a kép entitást és a következő mezőkkel rendelkezik (6.2. táblázat):

mező neve	**típusa**	**leírás**
ID (PK)	int	a kép egyedi azonosítója
S3Key	varchar	a kép elérési útvonala a képek tárolásához használt S3 bucketben
TakenAt	date | null	a kép készítésének dátuma
DeletedAt	date | null	a kép törlésének időpontja

6.2. táblázat – Az Image tábla mezői

6.3.3 User_Image tábla

Kapcsolótábla, ami tárolja az egyes felhasználókhoz tartozó képeket. (6.3. táblázat)

mező neve	típusa	leírás
UserID (FK)	int	felhasználó egyedi azonosítója
ImageID (FK)	int	a felhasználóhoz tartozó kép egyedi azonosítója

6.3. táblázat - A User_Image tábla mezői

6.3.4 Album tábla

Reprezentálja az album entitást, a következő mezőkkel rendelkezik (6.4. táblázat):

mező neve	típusa	leírás
ID	int	az album egyedi azonosítója
Name	varchar	az album neve

6.4. táblázat - Az album tábla mezői

6.3.5 Album_Image tábla

Kapcsolótábla, ami tárolja az egyes albumokhoz hozzárendelt képeket. (6.5. táblázat)

mező neve	típusa	leírás
AlbumID (PK)	int	az album egyedi azonosítója
ImageID (FK)	int	az albumhoz hozzárendelt kép egyedi azonosítója

6.5. táblázat - Az Album_Image tábla mezői

6.3.6 User_Album tábla

Kapcsolótábla, ami tárolja az egyes felhasználókhoz hozzárendelt albumokat. (6.6. táblázat)

mező neve	típusa	leírás
UserID(FK)	int	a felhasználók egyedi azonosítója
AlbumID (FK)	int	a felhasználó által létrehozott album egyedi azonosítója

6.6. táblázat - A User_Album tábla mezői

6.3.7 Face tábla

Az egyes képeken felismert arcokat reprezentáló entitás. (6.7. táblázat)

mező neve	típusa	leírás
ID (PK)	varchar	az archoz tartozó egyedi azonosító
Name	varchar	a felhasználó által megadott, archoz tartozó név
S3Key	varchar	az archoz tartozó profilkép S3 bucket útvonala

6.7. táblázat - A Face tábla mezői

6.3.8 Image_Face

Kapcsolótábla, ami tartalmazza az egyes képeken felismert arcokat és hozzá tartozó érzelmek bizonyosságát. (6.8. táblázat)

mező neve	típusa	leírás
FaceID (FK)	varchar	a képen felismert arc egyedi azonosítója
ImageID (FK)	int	a kép egyedi azonosítója
HappyConfidence	float	információ arra vonatkozóan, hogy az Amazon Rekognition szolgáltatás milyen bizonyossággal ismerte fel az adott érzelmet az arcon
SadConfidence	float	
AngryConfidence	float	
DisgustedConfidence	float	
SurprisedConfidence	float	
CalmConfidence	float	
FearConfidence	float	

6.8. táblázat - Az Image_Face tábla mezői

6.3.9 Táblák kapcsolata

Adatbázis séma tervezésekor célszerű az egyes táblák kapcsolatát is jellemezni. Az előző alfejezetekben ismertetett táblák egymáshoz való viszonyát fogom röviden leírni.

User – User_Image táblák kapcsolata

„Egy-több” kapcsolat van a két tábla között, mert a felhasználók természetesen több képet is feltölthetnek.

User_Image – Image táblák kapcsolata

A User_Image kapcsolótáblához természetesen hozzá kell rendelni a képet tartalmazó entitást is. Egy kapcsolótábla rekordhoz egy kép tartozhat. („egy-egy” kapcsolat)

Image – Image_Face táblák kapcsolata

Egy képen a mesterséges intelligencia alapú algoritmus több (vagy egyet sem) arcot is azonosíthat, ezért e között a két tábla között szintén „egy-több” kapcsolat van.

Image_Face – Face táblák kapcsolata

„Egy-egy” kapcsolat, mert egy képen felismert archoz egy darab arc rekord tartozik, ahol csupán az archoz tartozó adatok tárolódnak.

User – User_Album táblák kapcsolata

A felhasználók tetszőleges számú albumot hozhatnak létre az alkalmazásban, így ezen táblák kapcsolata szintén „egy-több”.

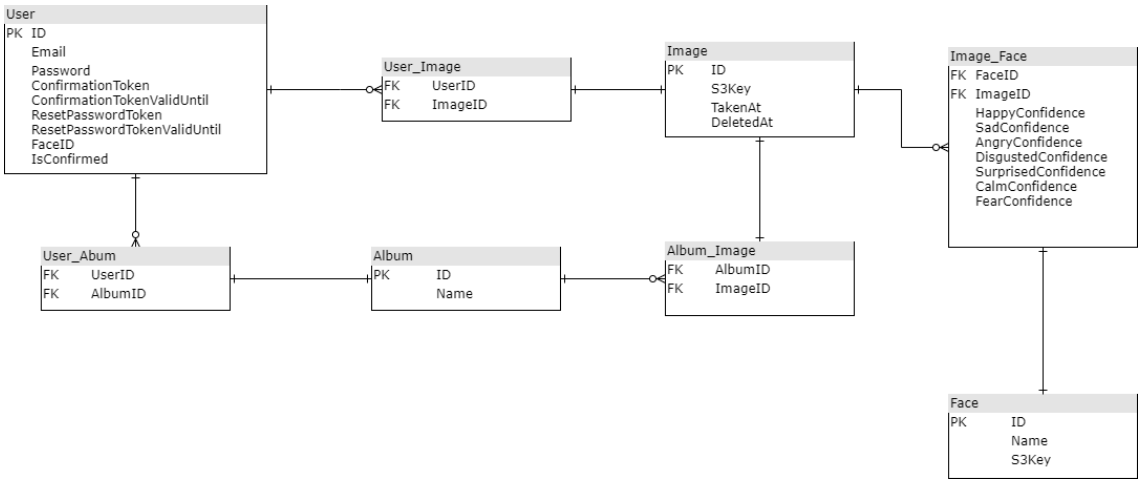
User_Album – Album táblák kapcsolata

A User_Album kapcsolótábla rekordjához egy darab Album rekord tartozik. („egy-egy” kapcsolat).

Album – Album_Image

Egy albumhoz több képet hozzárendelhetnek a felhasználók, így a két tábla kapcsolata „egy-több”.

A fentebb említett táblák viszonyai a 6.10. ábra mutatja.



6.10. ábra - A táblák kapcsolata

7. FEJLESZTÉS MENETE

7.1 Projekt inicializálása

Első lépésként létre kell hozni a projektet. A fejlesztésre én a JetBrains Rider eszközét választottam, amihez kérvényezhető egyetemi licensz. A projekt létrehozásához elérhetőek az eszközben sablonok, aminek segítségével nem kell manuálisan, lépésről lépésre beállítani a projektet. A sablon létrehozza mind a kliens mind a szerver oldalhoz szükséges elemeket.

A technológiai választásom alapján én egy React/C# WebAPI projektet hoztam létre. Sajnos nem volt elérhető olyan sablon, ami TypeScript-et állítana be a kliens oldali fejlesztéshez, így ezt manuálisan végeztem el. Kitöröltem a kliens oldali fájlokat, majd a „create-react-app" konzol eszközt használtam, amiben szintén választhatunk sablonokat. Kiválasztottam a TypeScript-es sablont, majd legeneráltam a kliens oldali projektet. Ezt utána belemásoltam a korábban Rider-rel létrehozott projektben a megfelelő helyre.

Következő lépésként inicializáltam a mappában a „git repository" -t amelynek segítségével folyamatosan nyomon tudom követni a változásokat a projektben.

7.2 Szerver oldal

7.2.1 Adatbázis

A rendszerterv (6. fejezet) alapján definiáltam az adatbázishoz szükséges modelleket, amik lényegében a táblákat, entitásokat jelentik a megfelelő típusú tulajdonságokkal (oszlopokkal).

A következő lépés az egyes entitások közötti kapcsolatok definiálása volt, ezt mindegyik entitáshoz megtettem.

Szükséges volt egy adatbázis kontextust létrehozni, amiben az egyes táblákat adtam meg, illetve itt kell beregisztrálni a táblák közötti kapcsolatokat leíró konfigurációkat.

A lokális fejlesztéshez teszteléshez létrehoztam egy adatbázist a gépemen, aminek elérési útvonalát beállítottam a megfelelő konfigurációs fájlban. Ezután beregisztráltam az adatbázis kontextust, amihez szükségem volt az elérési útvonalra, ehhez még telepítettem a Microsoft SQL Serverhez való kapcsolódás szükséges külső csomagot. Ettől a ponttól az adatbázis kontextus, mint függőség befecskendezhetővé válik bármelyik később kód számára.

Az adatbázisban a táblákat „code-first" megoldással hoztam létre, így nem kell külön SQL scripteket megírni és lefuttatni. Az adatbázis sémájának változásának kezelésére az Entity Framework Migrations nevű eszközét állítottam be.

7.2.2 Üzleti logika

Az authentikáció és authorizáció mint funkcionalitás az alkalmazás egyik központi és nélkülözhetetlen eleme, ezért ezzel kezdtem a fejlesztést. Kezdeti lépésként bekonfiguráltam a következő útmutató alapján [25] a JWT authentikációt.

Létrehoztam a „UserService” -t, ami a következő logikákért felel:

- Beléptetés
- Kiléptetés
- Regisztráció
- Elfelejtett jelszó kezelése
- Jelenleg belépett felhasználó információinak lekérése

Következő lépésként létrehoztam a kontrollert, amiben definiáltam a végpontokat, amiken keresztül a kliens eléri a felhasználóval kapcsolatos funkciókat:

- GET -> /user/current
- POST -> /user/sign-in
- GET -> /user/sign-out
- POST -> /user/register
- GET -> /user/forgot-password?email=
- GET -> /user/forgot-password/{id}/confirm

Ezután a fotók kezelésével kezdtem el foglalkozni, létrehoztam a „PhotoService” -t ami a képek kezeléséért felel:

- Feltöltés
- Képek listázása
- Kép törlése

A fotók kezeléséhez szükségem volt egy külső csomag telepítésére is, amivel elérhetővé válik az S3 hoz való kapcsolódás. Ehhez szükségem volt még egy AWS felhasználó létrehozására, majd egy kulcs generálására, amit környezeti változóként (AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY) beállítottam a gépemen. A fejlesztői eszköz újraindítása után már megtalálta az imént beállított kulcsokat, így már tudtam használni az S3 klienst, mint befecskendezhető függőséget.

Ezeket a funkciókat szintén kivezettem, az előbb említett kontrollerbe, hiszen a képek egy felhasználóhoz tartoznak, így megfelelőnek tartottam ide rakni ezeket a végpontokat is:

- GET -> /user/photos
- POST -> /user/photos
- DELETE -> /user/photos/{id}

A következő lépés az albumok kezelésével foglalkozó logika lefejlesztése volt, erre hoztam létre az „AlbumService” -t. A funkciókat látja el:

- Albumok listázása
- Album létrehozása
- Album törlése

Szintén kivezettem ezeket is végpontokba, amik a következően néznek ki:

- GET -> /user/albums
- POST -> /user/albums
- DELETE -> /user/albums/{id}

Minden lefejlesztett végponthoz létrehoztam úgynevezett adatátviteli objektumokat, amik arra szolgálnak, hogy ne egy az egyben szolgáltassuk az adatbázis modellt a kliens oldal felé, ezzel elkerülve az adatszivárgást.

```
[ApiController]
[Route("api/[controller]")]
public class UserController : ControllerBase
{
    private readonly UserService _userService;
    private readonly PhotoService _photoService;
    private readonly AlbumService _albumService;

    public UserController(UserService userService, PhotoService photoService,
AlbumService albumService)
    {
        _userService = userService;
        _photoService = photoService;
        _albumService = albumService;
    }

    [Authorize]
    [HttpGet("current")]
    public async Task<ActionResult<CurrentUserInfoDto>> GetCurrentUser()
    {
        return await _userService.GetCurrentUser();
    }

    [AllowAnonymous]
    [HttpPost("sign-in")]
    public async Task<ActionResult> SignIn([FromBody] SignInDto signInDto)
    {
        await _userService.SignIn(signInDto);

        return Ok();
    }
}
```

7.1. ábra példakód részlet a kontrollerről

7.2.3 Képfeldolgozó komponens

A képeken szereplő érzelemfelismeréshez szükséges funkciót egy külön alprojektként hivatkoztam a fő projekthez. Ebben egyetlen rövidebb kódot készítettem el, ami az S3 bucket kép feltöltés hatására érkező eseményére figyel, meghívja az Amazon Rekognition API-t és hozzáadja az adatbázishoz az adatokat.

A dokumentáció alapján kikerestem a megfelelő metódust *(„IndexFaces”)* az API-hoz, majd meghívtam a képre. Ebben a metódusban a kép S3 elérési útvonalát kell megadni, amit az eseményből kaptam meg. A válaszban (7.2. ábra) érkező adatokat így elmentettem az adatbázisba, az egyes arcokhoz.

Ezen komponens implementálásakor felismertem, hogy az adatbázist kontextust ebben az alprojektben is használni kell, ezért átmozgattam az adatbázishoz tartozó kódokat szintén egy külön alprojektbe, hogy az közösen felhasználható és könnyen hivatkozható legyen.

```
{
  "FaceRecords": [
     {
        "Face": {
           "BoundingBox": {
              "Height": number,
              "Left": number,
              "Top": number,
              "Width": number
           },
           "Confidence": number,
           "ExternalImageId": "string",
           "FaceId": "string",
           "ImageId": "string",
           "IndexFacesModelVersion": "string"
        },
        "FaceDetail": {
           "Emotions": [
              {
                 "Confidence": number,
                 "Type": "string"
              }
           ]
        }
     }
  ]
}
```

7.2. ábra IndexFaces válaszának sémája (részlet)

7.2.4 Képek periodikus törlését végző komponens

A képfeldolgozó komponens alapján hasonló módszerrel jártam el. Hozzáadtam egy külön alprojektet a főprojekthez, amiben egyetlen rövid kis funkciót implementáltam. Itt már könnyedén feltudtam használni az adatbázis kontextust, amivel lekértem a felhasználók által előzetesen törlésre megjelölt képeket, amik 30 napnál régebbeik. Ezeken a rekordokon végig mentem és kitöröltem mindet először az S3 bucketból, majd az adatbázisból.

7.3 Kliens oldal

7.3.1 CSS keretrendszer

A kliens oldali fejlesztést a CSS rendszer telepítésével kezdtem. Erre a *Material UI*-t választottam az 5.1.2-es fejezetben. Telepítettem a dokumentáció alapján a csomagot, ami rengeteg kész, újra felhasználható, megfelelő konfiguráció esetén reszponzív *React* komponensből áll a Material dizájn elveknek megfelelően.

7.3.2 Routing

A következő lépésként bekonfiguráltam a „routingot”. A „routing” lényegében az egyes elérési útvonalakhoz tartozó komponenseket kezeli, tehát egy adott útvonal esetén a megfelelő oldalra irányítja a felhasználót.

Az egyes útvonalakhoz különböző „védelmi szabály” tartozhat, például, ha egy felhasználó épp nincs belépve, akkor nem érheti el azokat az oldalakat, ami csak belépés után lenne látható. Ezeket a „szabályokat” is komponensek formájában kell definiálni, amiket megtettem.

7.3.3 Oldal váza

Ezután létrehoztam az alkalmazáshoz szükséges összes oldalt, egyelőre üresen. Az egyes oldalakat ezután hozzárendeltem a megfelelő útvonalakhoz. Létrehoztam továbbá az oldalnak a vázát, ami tartalmazza a fejléc szekciót a navigációs lehetőségekkel, illetve a tartalom részét, amibe az egyes, ezen a ponton még üres oldalak töltődnek be.

7.3.4 Felhasználó kezeléssel kapcsolatos oldalak

Következő lépésként a felhasználókkal kezelésével kapcsolatos oldalakkal folytattam. Létrehoztam a belépés képernyőt, ami fogadja azokat a felhasználókat, akik jelenleg nincsenek belépve. A belépés képernyőről tovább lehet navigálni két különböző oldalra, ami a regisztráció és az elfelejtett jelszó. Ezeket az oldalakat hoztam létre következő lépésként és a bekötöttem a navigációt a belépés képernyőre. Amint végeztem ezekkel az oldalakkal, integráltam az alkalmazást a szerver oldallal.

Belépés

Az oldal betöltésekor meghívom a jelenleg belépett felhasználóhoz tartozó végpontot, amely kérés tartalmazza az autentikációhoz használt tokent. A felhasználó amennyiben érvényes tokennel rendelkezett a szerver oldal visszaadja a hozzá tartozó információkat, ami alapján eltároltam a felhasználót memóriában, és tovább navigáltam az oldal kezdőképernyőjére. Sikertelen kérés esetén hibaüzenetet írtam ki a belépés képernyőre.

Kilépés

A kilépés funkcióra egy felhasználó ikont helyeztem fel a fejlécre, amire kattintásra felnyílik egy menü egy kilépés opcióval. Kattintásra meghívtam a szervert, ami érvényteleníti a felhasználó tokenjét, majd kitöröltem a felhasználót kliens oldalon a memóriából.

Regisztráció

A regisztrációs oldalra egy űrlapot raktam, amiben a 4.1-es fejezetben leírt információkat kell kitölteni. A gomb megnyomásának hatására bekötöttem a szerver oldali hívást, amihez hozzáadtam ezeket az információkat. Sikeres kérés esetén egy információs üzenetet írtam ki, sikertelen hívás esetén egy hibaüzenetet. Hozzáadtam az oldal aljára egy navigációs linket is, amivel a belépés képernyőre tudunk visszatérni.

Elfelejtett jelszó

Az elfelejtett jelszó oldalt és hozzá tartozó űrlapot lényegében a regisztrációval megegyező logika alapján készítettem el. Egyetlen email cím mezőt raktam az űrlapba, illetve egy gombot, aminek hatására a szerver oldali integrációt kötöttem be. Hozzáadtam ide is egy navigációs linket, amivel visszalehet térni a belépés képernyőre.

7.3.5 Képekkel kapcsolatos oldalak

Képek oldal

A képek oldalt két alszekcióra bontottam, egy fejlécre és egy tartalom részre. A fejlécbe tettem a szűrési funkciót érzelmek alapján, a dátum szerinti rendezést és szűrést, illetve két gombot, amellyel albumot lehet létrehozni és képet feltölteni. A feltöltés az oldalra „dobva”, „drag and drop” módszerrel is lehetséges.

Az oldal megnyitásakor a szerver felé intézett kérés visszaadja a felhasználó összes képét, minden szűréshez szükséges adattal, így a szűrést kliens oldalon valósítottam meg. A visszaérkező adatokból kiszámítottam az elérhető érzelmek listáját, ami egy lista formájában jelenik meg kattintás után, több választható lehetőséggel.

Az érkezett képeket ezután a tartalom részben kilistáztam. Szűrés esetén automatikusan változik a tartalom. Ha az egyes képek fölé visszük az egeret, akkor kijelölhető lesz a kép, megjelenik egy törlés gomb. Kijelölés esetén az album létrehozása gomb aktiválódik, kattintható lesz. Kattintásra egy megnyílik egy dialógus, ami egy nevet vár az albumhoz, alatta mentés és mégse gombokkal.

Albumok oldal

Az albumok oldal megnyitásakor szintén elküldök egy kérést a szerver felé, ez alapján kilistázom az albumokat. Albumra való kattintáskor egy görgethető dialógusban kilistázásra kerülnek a képek.

8. TESZTELÉSI TERV, TESZT EREDMÉNYEK

8.1 Szerver oldal

A szerver oldali alkalmazás, pontosabban annak végpontjainak tesztelésére a „Postman” alkalmazást választottam. Ezzel az alkalmazással http kérések küldhetőek egy adott címre. A végpontokat még a kliens oldali fejlesztés megkezdése előtt teszteltem. Az elvégzett teszteseteket a 8.1. táblázat szemlélteti.

#	Teszteset	Eredmény
0	**GET /user/current** *A kérésben nem szerepel érvényes token. A várt eredmény http 401.*	✓
1	**POST /user/register** *Kérésben a regisztrációhoz szükséges adatok. A várt eredmény http 200.*	✓
1	**POST /user/sign-in** *Kérésben létező felhasználónév és jelszó. A várt eredmény a válaszban egy token.*	✓
2	**GET /user/current** *Tokennel. A várt eredmény a tokenből elérhető felhasználó adatai.*	✓
3	**POST /user/photos** *A kérésben tokennel és képpel. A várt eredmény a kép azonosítója.*	✓
4	**GET /user/photos** *A kérésben tokennel. A várt eredmény a felhasználó feltöltött képei.*	✓
5	**POST /user/albums** *A kérésben tokennel, a kérés törzsében album nevével és egy kép azonosítójával. A várt eredmény az album azonosítója.*	✓

6	**GET /user/albums** *A kérésben tokennel. A várt eredmény a felhasználó albumai.*	✓
7	**DELETE /user/albums/1** *A kérésben tokennel. A várt eredmény http 204.*	✓
8	**DELETE /user/photos/1** *A kérésben tokennel. A várt eredmény http 204.*	✓
9	**GET /user/forgot-password** *Token nélkül, query paraméterben létező felhasználó email címével. A várt eredmény http 200.*	✓
10	**GET /user/forgot-password/da0ca76b-4147-40f1-a8b8-5387c8d78186/confirm** *A kérésben érvényes egyedi megerősítő azonosítóval. A várt eredmény http 200.*	✓
11	**GET /user/sign-out** *A kérésben tokennel. A várt válasz visszavonja a felhasználó érvényes tokenjét.*	✓

8.1. táblázat - Szerver oldalon elvégzett tesztesetek

8.2 Kliens oldal

A kliens oldali alkalmazás tesztelésére a manuális tesztelést, mint módszert választottam. Ezt a fejlesztés során, a funkciók implementálása folyamatában elvégeztem. Amikor az összes funkció elkészült elvégeztem az egész alkalmazás regressziós tesztelését.

(A regressziós tesztelés a korábban már működő funkciók újra tesztelését jelenti és arra a célra szolgál, hogy az új funkciók fejlesztése során egy régi funkció működőképes maradt e. Az elrontott létező funkciókat regressziós hibának nevezzük.)

8.3 Alkalmazás telepítése és tesztelése felhőben

Az alkalmazás telepítéséhez szükséges volt létrehoznom az összes infrastruktúrában szereplő komponenst. Ezeknek a létrehozására az Amazon Web Services-nél elérhető CloudFormation szolgáltatást használtam. A CloudFormation egy olyan szolgáltatás, amiben egy sablonnal leírhatjuk az infrastruktúránk elemeit YAML vagy JSON formátumban, a szolgáltató saját leíró nyelvével. Ennek az előnye, hogy az egész infrastruktúra forráskód alá helyezhető, így jobban karbantartható és a későbbiek folyamán egy új környezet létrehozása sokkal egyszerűbb.

Létrehoztam tehát a sablont a dokumentáció alapján, majd kitelepítettem az alkalmazást. Az alkalmazáshoz nem vásároltam egyedi címet *(domain)*, így az az alapértelmezett úton volt elérhető, amelyben egy egyedi véletlenszerű azonosító szerepelt. Ezen az elérhető címen egy újabb regressziós teszt kört végeztem az alkalmazáson, amellyel ellenőriztem a funkciók működését.

8.4 Összegzés

Ebben a fejezetben leteszteltem és megvizsgáltam az alkalmazás funkcióinak működését lokális környezetben és felhőben. Az egyes regressziós tesztek alapján megbizonyosodtam az alkalmazás használhatóságáról és a funkciók helyes működéséről.

A tesztkörök során több olyan esetet is észrevettem, amikor a felhasználói élmény jelentősen javítható, továbbfejleszthető lett volna, mind szerver oldalról (sebesség) mind kliens oldalról.

Néhány a továbbfejlesztési lehetőségekről, amely vagy a felhasználói élményt vagy a tesztelés könnyedségét javítja a következő Továbbfejlesztési lehetőségek fejezetben számolok be.

9. TOVÁBBFEJLESZTÉSI LEHETŐSÉGEK

9.1 Parancs- és lekérdezési felelősség elkülönítése [26]

A parancs és lekérdezési felelősség elkülönítése egy tervezési minta, ami az adatbázis olvasási és írási műveleteit elválasztja egymástól. Ennek a mintának a bevezetésével növelhetjük az alkalmazás teljesítményét, tesztelhetőségét és biztonságát. Rugalmasabbá teszi továbbá a rendszert, és a későbbi funkciók fejlesztését is könnyebbé teszi, hiszen megakadályozza az olvasási és írási műveletek ütközését. Az alkalmazásom esetében ez lényegében a szerver oldali kód átszervezését jelentené.

9.2 Naplózás és monitorozás

Az alkalmazás karbantarthatósága és minőségbiztosítása szempontjából nagyon fontos a megfelelő naplózás és monitorozás bevezetése.

Naplózás

A naplózás lényegében az alkalmazásból küldött eseményekről való információk eltárolását jelenti. Hiba esetén így könnyedén megtekinthetjük a naplóbejegyzéseinket, amik fényt deríthetnek egy adott hibára. Célszerű kellő mennyiségű információval ellátott bejegyzéseket menteni, ellenkező esetben a haszontalan információkkal nem tudunk egy adott hibára megoldást találni. Mind kliens és szerver oldali eseményeket érdemes naplózni.

Monitorozás

A felhőben használt komponensekhez, szolgáltatásokhoz az Amazon alapból biztosít felületén monitorozásra alkalmas eszközöket, azonban az alkalmazás kliens és szerver oldali hibáira magunknak kell beállítani monitorozást. Amennyiben megfelelő naplóbejegyzéseket hozunk létre, ezekre létre tudunk hozni riasztásokat, ami például hibás esemény alapján egy értesítést küld.

9.3 Gyorsítótár

Az alkalmazás skálázhatóságán és felhasználói élményén jelentős mértékben tud javítani a gyorsítótárak *(angolul cache)* használata. A kliens oldali statikus fájloknál ez az architektúrából adódóan jelen van, hiszen amit a CDN kezel, azonban a szerver oldal felé irányuló kérések estében is belehet vezetni gyorsítótárat. Ez azt jelentené, hogy az adott hívás eredményét egy nagy sebességű tárhelyen – jellemzően memóriában – kulcs érték párok formájában eltároljuk egy adott lejárati ideig. Így a rákövetkező hívások már innen érkeznek a kliens felé, nem kell az egész üzleti logikának lefutnia és az adatbázisig elmenni.

A lejárati idő lehet abszolút, vagy „csúszó”. Az abszolút azt jelenti, hogy egy adott idő megadása, például 15 perc után mindenképp elévül a gyorsítótárban eltárolt rekord. A „csúszó” lejárat pedig azt jelenti, hogy 15 percen belüli hívás újból 15 percig érvényben tartja a tárban a rekordot.

A szerver oldali alkalmazás ahogyan a 5.6-fejezetben leírtam, nem egy állandóan futó szerveren van, így annak a memóriáját nem célszerű gyorsítótárként használni, hiszen bármikor indulhat vagy leállhat egy példány belőle, amit a felhőszolgáltató kezel. Erre az esetre célszerű egy dedikált elosztott szolgáltatást használni, erre az *Amazon ElastiCache* [27] -t találtam. Ez gyakorlatilag egy újabb komponenst jelent az infrastruktúrában, ami egy önálló állandóan futó szerver, ahol memóriában tudunk tárolni bejegyzéseket, így bármelyik szerver oldali példány fut, elérhető lesz adott esetben egy rekord a gyorsítótárban.

A gyorsítótárak használata rengeteg komplexitást ad hozzá az alkalmazáshoz, hiszen sok esetben ezeket az eltárolt bejegyzéseket érvénytelenítni kell. Egy konkrét egyszerű példa:

1. felhasználó megnyitja a képeim oldalt
2. bekerül a szerver oldal felé intézett hívás eredménye a gyorsítótárba
3. a felhasználó feltölt egy új képet
4. ráfrissít az oldalra

Ha nem érvénytelenítjük az eltárolt képeit, akkor az újonnan feltöltött kép nem fog megjelenni.

„There are only two hard things in Computer Science: cache invalidation and naming things.”

Phil Karlton

9.4 Analitika bevezetése

Amennyiben már a felhasználók számára is elérhető alkalmazásról beszélünk, érdemes lehet különböző analitikai eszközök, szolgáltatásokat is beépíteni. Erre egy nagyon népszerű megoldás a *Google Analytics* [28]. Ezekkel nyomon tudjuk követni a felhasználói aktivitást különböző mutatók alapján például:

- Összes felhasználó
- Új felhasználók
- Aktív felhasználók
- Oldalmegtekintés
- Egyedi oldalmegtekintés

Beállíthatunk még egyéb speciális nyomkövetést is, például, hogy hány felhasználó tölt fel képet.

10.ÖSSZEFOGLALÓ

A dolgozatomban egy modern webalkalmazás fejlesztéséről írtam. Modern alatt értem a jelenleg, dolgozat írásakor piacon is használt elérhető élvonalbeli technológiák használatát, fejlesztés szempontjából élen járó infrastruktúrát és architektúrát. Az applikáció, amin keresztül bemutattam a folyamatot egy képtároló alkalmazás, amelyekből jelenleg is elérhetőek népszerű változatok a piacon, azonban hozzáadtam egy olyan funkciót, amely ebben az időben nem volt elérhető. Ez a funkció a jelenlegi trendeknek megfelelően mesterséges intelligencia használatával érzelmek alapján kereshetővé teszi a feltöltött képeket és hozzáadott értéket ad a felhasználók számára az éppen elérhető népszerű alkalmazások mellett. Bemutattam a folyamatnak a lépéseit a kezdeti fázistól az irodalomkutatástól a tervezésen át egészen az alkalmazás tényleges telepítéséig, amellyel éles környezetbe helyezhető a rendszer. Megbizonyosodtam a funkciók helyes működéséről és további lehetőségeket mutattam be az esetleges továbbfejlesztési lehetőségekről.

11.SUMMARY

In my thesis topic I wrote about creating a modern webapplication. By modern I mean the usage of the currently available cutting-edge technologies both from development point of view and infrastructure and architecture, at the time that I wrote. The application which I shown the process is a photo storage application, that are already available ont he market, but I extended the functionalities with a feature which was missing at this time. This functionality also follows the current trends and uses artificial intelligence in the background which processes the images uploaded by the users, so they can browse and filter them by emotions. This functionality provides additional value for the potential users of this application. Throughout the thesis I showed the steps needed from the research phase up until the deployment to a live environment. I verified the functionalities and also wrote suggestions about potential, future developments which would make the application better.

12. IRODALOMJEGYZÉK

[1] [Online]. Available: https://www.google.com/photos/.

[2] [Online]. Available: https://www.icloud.com/photos.

[3] C. FLYNN, „14 Technologies Every Web Developer Should Be Able to Explain," [Online]. Available: https://differential.com/insights/14-technologies-every-web-developer-should-be-able-to-explain/.

[4] „TypeScript," [Online]. Available: https://www.typescriptlang.org/.

[5] D. Tarnowski, „Angular vs React—the DEAL BREAKER," [Online]. Available: https://hackernoon.com/angular-vs-react-the-deal-breaker-7d76c04496bc.

[6] [Online]. Available: https://www.npmjs.com/package/@angular/core.

[7] [Online]. Available: https://www.npmjs.com/package/react.

[8] „NPM trends," [Online]. Available: http://www.npmtrends.com/@angular/core-vs-react.

[9] S. o. J. frameworks. [Online]. Available: https://gist.github.com/Restuta/cda69e50a853aa64912d.

[10] [Online]. Available: https://en.wikipedia.org/wiki/MIT_License.

[11] „Bootstrap," [Online]. Available: https://getbootstrap.com/.

[12] „Google - Material design," [Online]. Available: https://material.io/design/.

[13] J. Serra, „Types of NoSQL databases," [Online]. Available: http://www.jamesserra.com/archive/2015/04/types-of-nosql-databases/.

[14] „Entity Framework - Database Providers," [Online]. Available: https://docs.microsoft.com/en-us/ef/core/providers/.

[15] „An overview of HTTP," [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview.

[16] „Transport Layer Security," [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Security/Transport_Layer_Security.

[17] „Git," [Online]. Available: https://git-scm.com/.

[18] [Online]. Available: https://cloudacademy.com/blog/google-vision-vs-amazon-rekognition-a-vendor-neutral-comparison/.

[19] [Online]. Available: https://aws.amazon.com/s3/.

[20] [Online]. Available: https://aws.amazon.com/cloudfront/.

[21] [Online]. Available: https://aws.amazon.com/route53/.

[22] [Online]. Available: https://aws.amazon.com/api-gateway/.

[23] [Online]. Available: https://aws.amazon.com/rds/.

[24] [Online]. Available: https://aws.amazon.com/rekognition/.

[25] [Online]. Available: https://www.c-sharpcorner.com/article/jwt-json-web-token-authentication-in-asp-net-core/.

[26] [Online]. Available: https://docs.microsoft.com/hu-hu/azure/architecture/patterns/cqrs.

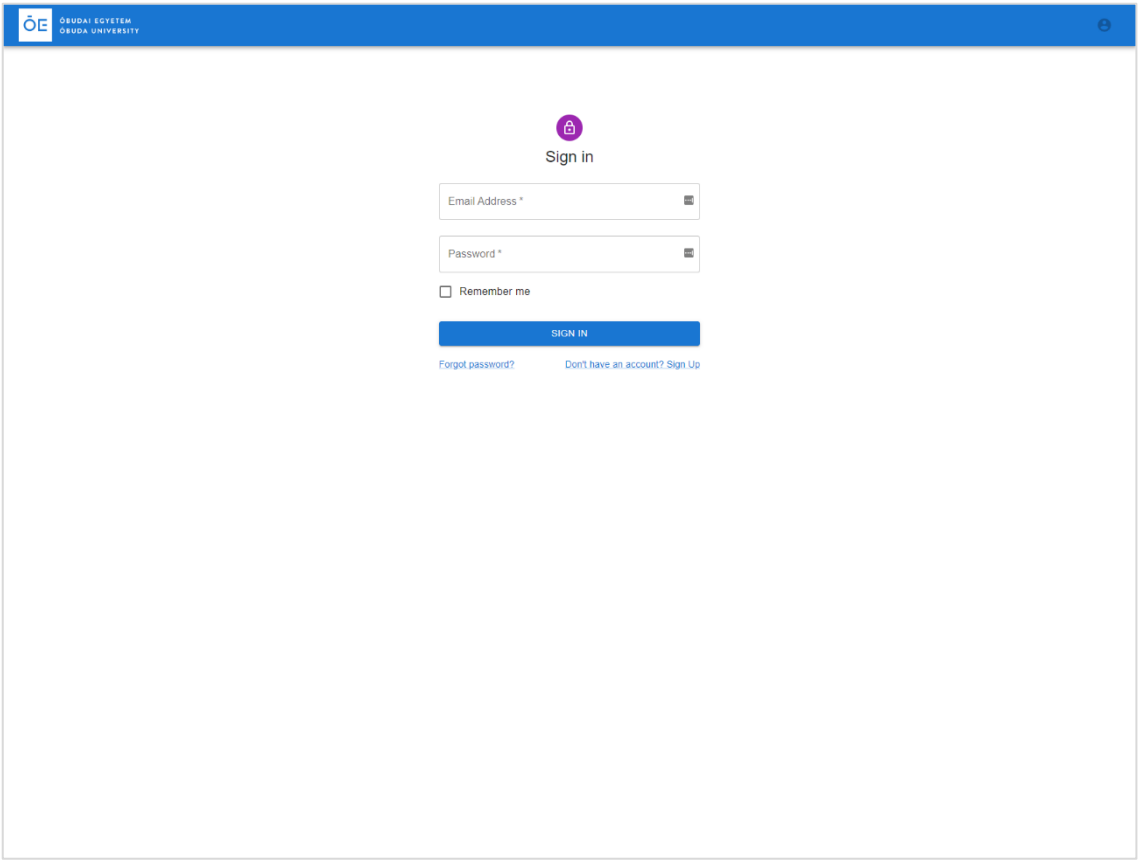
[27] [Online]. Available: https://aws.amazon.com/elasticache/.

[28] [Online]. Available: https://analytics.google.com/.

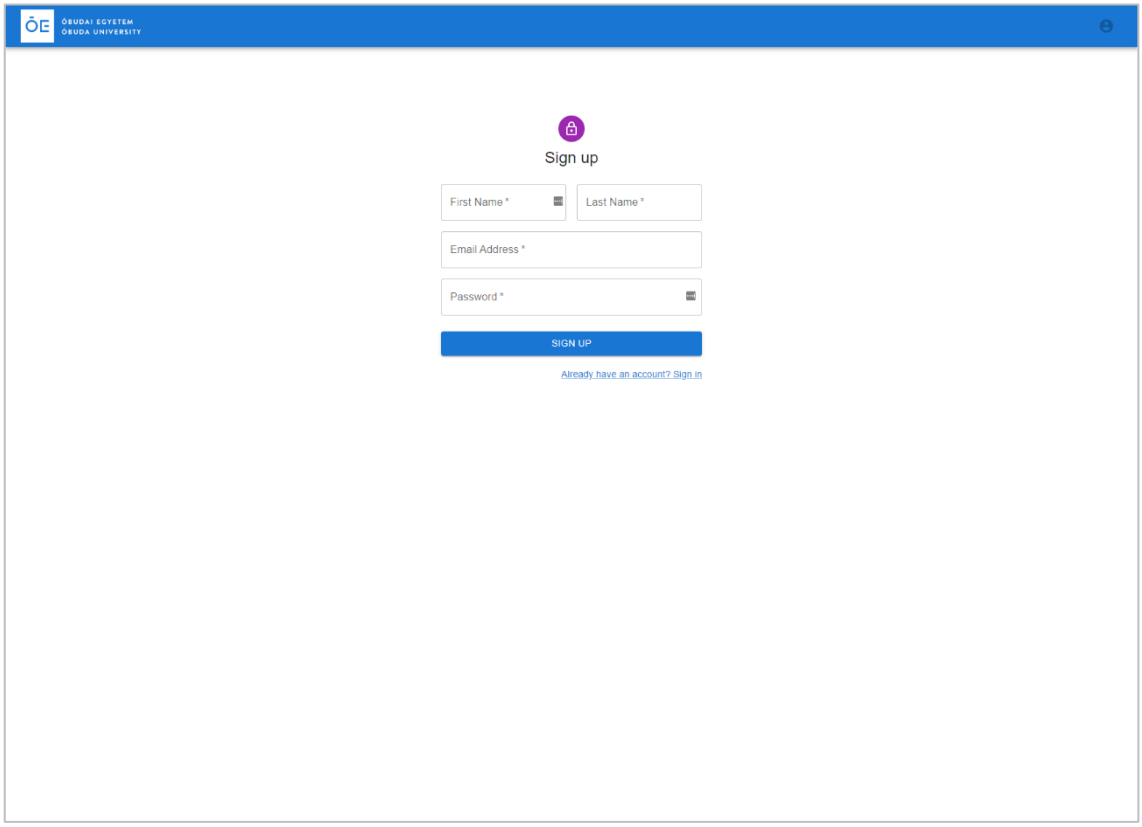
13. MELLÉKLETEK

13.1 Képernyőképek

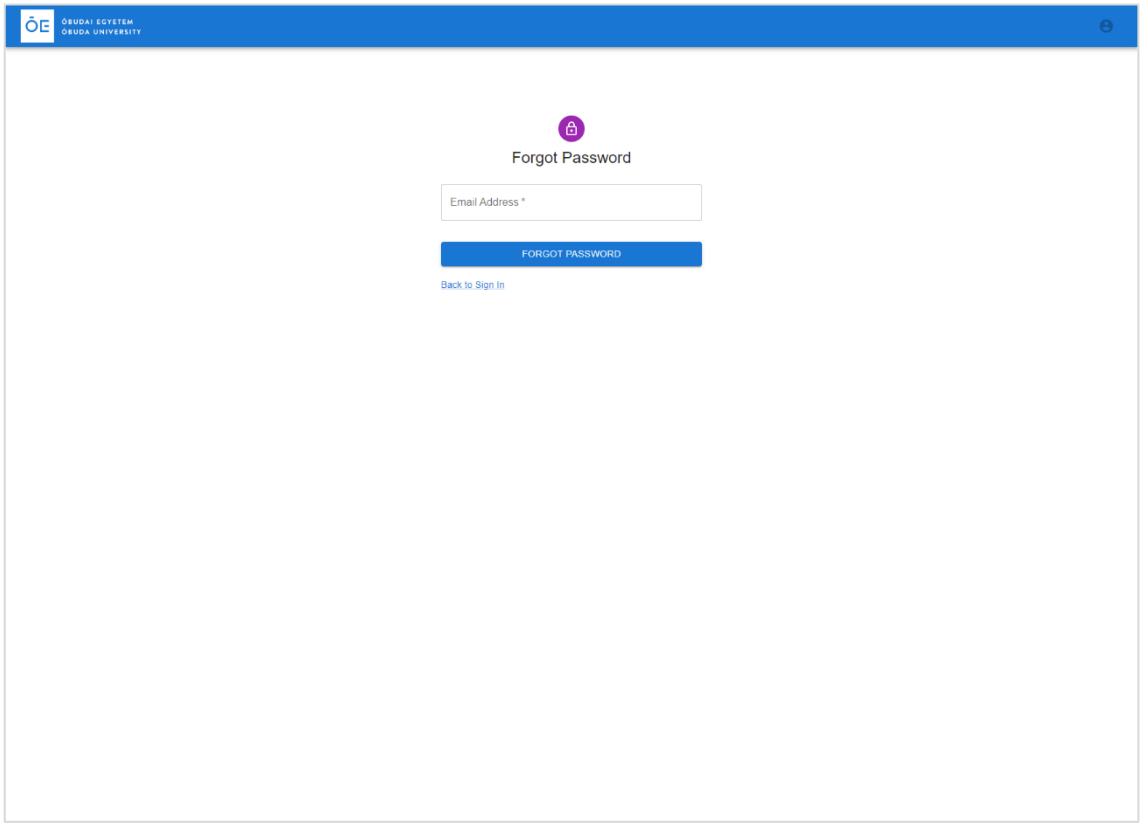
13.1.1 Belépés



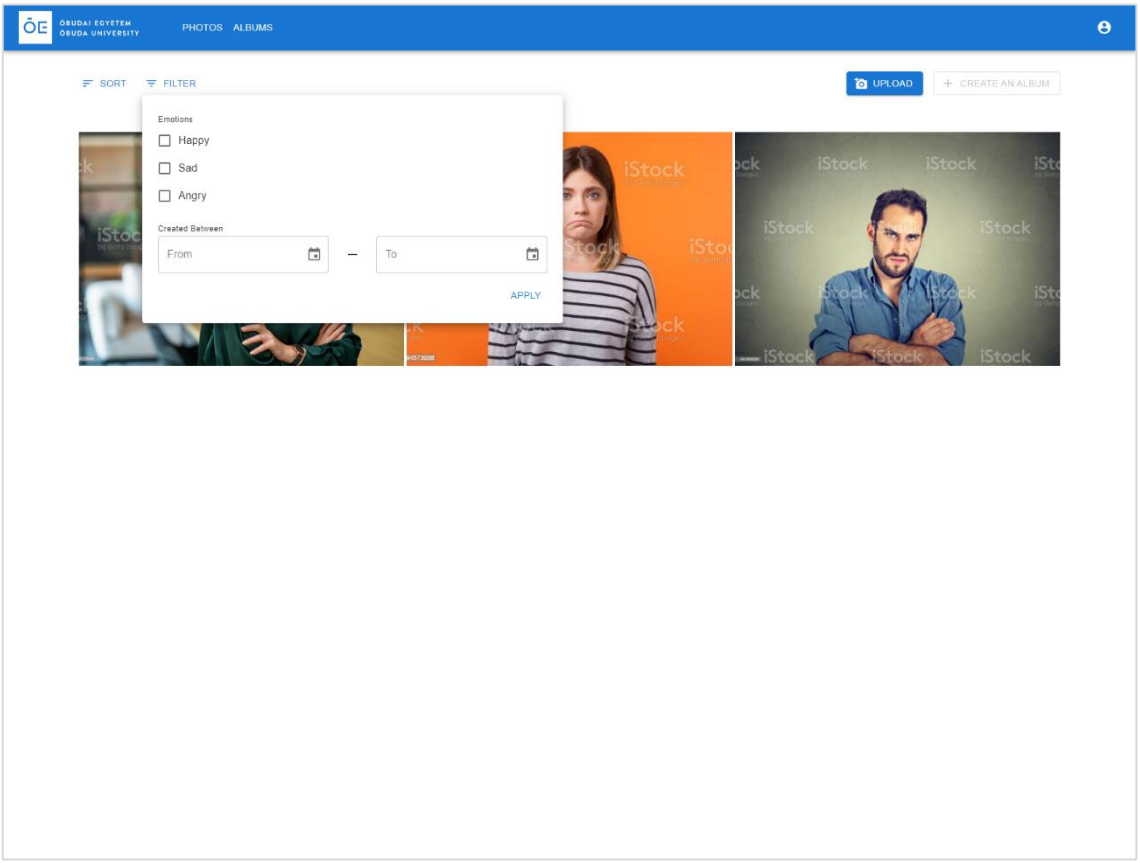
13.1.2 Regisztráció



13.1.3 Elfelejtett jelszó



13.1.4 Képek


ÓE
ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY
PHOTOS
ALBUMS
SORT
FILTER
UPLOAD
CREATE AN ALBUM
Emotions
Happy
Sad
Angry
Created Between
From
To
APPLY

13.1.5 Albumok