

**SAPIENTIA ERDÉLYI MAGYAR TUDOMÁNYEGYETEM  
MAROSVÁSÁRHELYI KAR,  
INFORMATIKA SZAK**



**SAPIENTIA**  
ERDÉLYI MAGYAR  
TUDOMÁNYEGYETEM

Személyre szabott tartalomajánlás

**DIPLOMADOLGOZAT**

Témavezető:  
Dr. Márton Gyöngyvér,  
Egyetemi adjunktus

Végzős hallgató:  
Magyari Dóra

**2023**

UNIVERSITATEA SAPIENTIA DIN CLUJ-NAPOCA  
FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE,  
SPECIALIZAREA INFORMATICĂ



UNIVERSITATEA  
SAPIENTIA

Recomandări personalizate

LUCRARE DE DIPLOMĂ

Coordonator științific:  
Dr. Márton Gyöngyvér,  
Lector universitar

Absolvent:  
Magyari Dóra

2023

**SAPIENTIA HUNGARIAN UNIVERSITY OF  
TRANSYLVANIA  
FACULTY OF TECHNICAL AND HUMAN SCIENCES  
COMPUTER SCIENCE SPECIALIZATION**



**SAPIENTIA**  
HUNGARIAN UNIVERSITY  
OF TRANSYLVANIA

Personalized content recommandation

**BACHELOR THESIS**

Scientific advisor:  
Dr. Márton Gyöngyvér,  
Lecturer

Student:  
Magyari Dóra

**2023**


## Declarație

Subsemnatul/a MAGYARI LÓRA, absolvent(ă) al/a specializării  
INFORMATICA, promoția 2023, cunoscând  
prevederile Legii Educației Naționale 1/2011 și a Codului de etică și deontologie profesională a  
Universității Sapientia cu privire la furt intelectual declar pe propria răspundere că prezenta  
lucrare de licență/proiect de diplomă/disertație se bazează pe activitatea personală,  
cercetarea/proiectarea este efectuată de mine, informațiile și datele preluate din literatura de  
specialitate sunt citate în mod corespunzător.

Localitatea, CORUNCA

Data: 15/06/2023

Absolvent

Semnătura 

**LUCRARE DE DIPLOMĂ**

Coordonator științific:  
**dr. Márton Gyöngyvér**

Candidat: **Magyari Dóra**  
Anul absolvirii: 2023

**a) Tema lucrării de licență:**

Proiectarea și crearea unei website care să își ofere utilizatorilor conținuturi unice personalizate, gestionând diferite tipuri de imagini.

**b) Problemele principale tratate:**

- studiu bibliografic privind sistemele web care oferă conținuturi personalizate utilizatorilor
- studiul kitului de stocare a parolelor de tip JWT (json web token)
- studiul sistemului React și Java script
- proiectarea și realizarea unei aplicații software care gestionează imagini, gestionează datele utilizatorilor, și oferă conținuturi unice, personalizate
- determinarea recomandărilor bazat pe timpul petrecut pe imagini

**c) Desene obligatorii:**

- schema arhitecturală a sistemului
- diagrama use-case a sistemului
- schema de funcționare a JWT-ului

**d) Softuri obligatorii:**

- aplicație bazată pe:
  - Java script, și React care au fost folosite pentru realizarea aspectul site-ului, ambele responsabili pentru vizualizarea grafică a aplicației,
  - C# care a fost folosit pentru a rezolva înregistrarea, logarea utilizatorilor,
  - Azure care a fost folosit pentru gestionarea datelor.

**e) Bibliografia recomandată:**

- [1] <https://jwt.io/introduction>
- [2] <https://medium.com/@prashantramnyc>
- [3] Matthew A. Russell, Mikhail Klassen: Mining the Social Web, 3rd Edition, O'Reilly Media, Inc, 2019.

**f) Termene obligatorii de consultații:** lunar. preponderent online

**g) Locul și durata practicii:** Universitatea „Sapientia” din Cluj-Napoca,  
Facultatea de Științe Tehnice și Umaniste din Târgu Mureș, sala / laboratorul de informatică cu nr. 414

Primit tema la data de: 08.11.2022

Termen de predare: 02.07.2023

Semnătura Director Departament

Semnătura responsabilului  
programului de studiu

Semnătura coordonatorului

Semnătura candidatului

# Kivonat

Dolgozatom témájaként a személyre szabott tartalomajánló rendszert választottam weboldalon, mivel a felhasználói élmény és az egyedi tartalom elengedhetetlenek a sikeres weboldalakhoz. A személyre szabott tartalomajánló rendszer lehetővé teszi, hogy a felhasználók a számukra releváns tartalmakhoz jussanak, amelyek érdeklődési körükbe tartoznak. Ezáltal növelhetik a weboldalon eltöltött időt, javíthatják az oldalra való visszatérés arányát és mélyebb kapcsolatot alakíthatnak ki az oldallal.

A személyre szabott tartalomajánlás előnyei közé tartozik az egyedi felhasználói élmény, a tartalom fokozott relevanciája, valamint a felhasználók elégedettségének növelése. Az efajta rendszer figyelembe veszi a felhasználók korábbi böngészési adatait, szokásait, különböző algoritmusok használatával, hogy pontosabb ajánlásokat kínáljon számukra. Ezáltal a felhasználók könnyebben megtalálhatják az érdeklődéseiknek megfelelő tartalmakat, valamint felfedezhetnek új dolgokat.

Dolgozatom egy weboldalt mutat be, mely képeket tartalmaz kategóriánként. Azért választottam ezt, mivel rendkívül fontosnak találom a képek szerepét az ilyen típusú weboldalakon a figyelem felkeltés miatt. A képek vonzóak és gyorsan felkeltik a felhasználók a figyelmét. Ezért egy képekkel teli tartalomajánló rendszer kiváló lehetőséget nyújt arra, hogy a felhasználók szórakozásképpen könnyebben megragadják érdeklődésüket.

Az általam készített tartalomajánlás megvalósítása a képeken eltöltött idő mérése által történik. A képeken eltöltött idő mérése segíthet a tartalomajánlás hatékonyságának növelésében. Ha a felhasználók hosszabb ideig nézik vagy vizsgálják a képeket, az arra utalhat, hogy az adott tartalom érdekes vagy vonzó számukra. Ez az információ lehetővé teszi a rendszer számára, hogy következtetéseket vonjon le a felhasználók preferenciáiról és ízléséről. Például, ha valaki hosszabb ideig nézi a sportfotókat, az azt jelentheti, hogy az ilyen típusú tartalom érdekli. Ennek alapján a rendszer további hasonló tartalmakat ajánlhat neki, ami nagyobb valószínűséggel nyeri el az érdeklődését.

# Rezumat

Am ales ca subiect al tezei mele sistemul de recomandare de conținut personalizat pentru site-uri web, deoarece experiența utilizatorului și conținutului unic sunt esențiale pentru site-urile web de succes. Un sistem de recomandare de conținut personalizat permite utilizatorilor să acceseze conținut relevant pentru interesele lor. Acest lucru poate crește timpul petrecut pe site, poate îmbunătăți ratele de revenire și poate construi o relație mai profundă cu site-ul.

Beneficiile recomandării de conținut personalizat includ o experiență unică a utilizatorului, o relevanță sporită a conținutului și o satisfacție sporită a utilizatorului. Sistemul epaafă ia în considerare istoricul și obiceiurile de navigare ale utilizatorilor, folosind diferiți algoritmi pentru a le oferi recomandări mai precise. Astfel, utilizatorilor le este mai ușor să găsească conținut care să corespundă intereselor lor și să descopere lucruri noi.

Teza me prezintă un site web cu imagini pe categorii. Am ales să fac acest lucru deoarece consider că rolul imaginilor pe acest tip de site web este extrem de important în atragerea atenției. Imaginile sunt atractive și atrag rapid atenția utilizatorilor. Prin urmare, un sistem de recomandare de conținut plin de imagini este o oportunitate excelentă de a capta interesul utilizatorilor într-un mod disrtactiv.

Recomandarea mea de conținut este implementată prin măsurarea timpului petrecut pe imagini. Măsurarea timpului petrecut de imagini poate contibui la creșterea ecifienței recomandării de conținut. Dacă utilizatorii petrec mai mult timp uitându-se la imagins sau explorându-le, acest lucru poate indica faptul că conținutul este interesant sau atractiv pentru ei. Aceste informații permit sistemului să targă concluzii despre preferințele și gusturile utilizatorilor. De exemplu, dacă cineva se ută la fotografii sportive, pentru o perioadă mai lungă de timp, poate indică faptul că este interesat de acest tip de conținut. Pe această bază, sistemul poate recomanda conținut similar, care are mai multe șanse să le atragă interesul.

# Abstract

I chose the personalized content recommendation system of websites as the topic of my thesis, because user experience and unique content are essential for successful websites. A personalized content recommendation system allows users to access content that is relevant to their interests. This can increase time spent on this site, improve return rates and build a deeper relationship with the site.

The benefits of personalized content recommendation include a unique user experience, increased relevance of content and increased user satisfaction. The system takes into account users browsing history and habits, using different algorithms to provide them with more accurate recommendations. This makes it easier for users to find content that matches their interests and discover new things.

My thesis presents a website with images by category. I have chosen to do this because I find the role of images on this type of website extremely important in attracting attention. Images are attractive and quickly attract the attention of users. Therefore, a content recommendation system full of images is an excellent opportunity to capture the interest of users in an entertaining way.

My content recommendation is implemented by measuring the time spent on images. measuring time spent on images can help to increase the effectiveness of content recommendation. If users spend more time looking at or exploring images, it may indicate that the content is interesting or attractive to them. This information allows the system to draw conclusions about users preferences and tastes. For example, if someone looks at sports photos for a longer period of time, it may indicate that they are interested in this type of content. Based on this, the system can recommend further similar content, which is more likely to attract teh interest.



# Tartalomjegyzék

<b>1. Bevezető</b>	<b>10</b>
<b>2. Programok, technológiák bemutatása</b>	<b>12</b>
2.1. Frontend Fejlesztése . . . . .	12
2.1.1. Learn - React . . . . .	12
2.2. Backend Fejlesztése . . . . .	14
2.2.1. Swagger - Swagger UI . . . . .	14
2.2.2. Jason Web Token - JWT . . . . .	15
2.3. Összehasonlítás . . . . .	19
2.3.1. Megközelítések összehasonlítása . . . . .	19
<b>3. Követelmények</b>	<b>20</b>
3.1. Felhasználói Követelmények . . . . .	20
3.2. Rendszer Követelmények . . . . .	20
3.2.1. Funkcionális Követelmények . . . . .	21
3.2.2. Nem Funkcionális Követelmények . . . . .	21
<b>4. Megjelenés megvalósítása</b>	<b>22</b>
4.1. Node Package Manager - npm . . . . .	22
4.2. Képek elérése - Pexels API . . . . .	22
4.3. Összességében . . . . .	24
<b>5. Szoftver</b>	<b>25</b>
5.1. A szoftver bemutatása . . . . .	25
5.2. Ajánlás algoritmus . . . . .	27
5.3. Adatbázis . . . . .	28
5.4. Diagram . . . . .	30
5.4.1. Use Case diagram . . . . .	30
<b>6. Célkitűzés</b>	<b>31</b>
6.1. Fontos tartalomajánló rendszer napjainkban: - Netflix . . . . .	32
<b>Összefoglaló</b>	<b>33</b>
<b>Könyvészet</b>	<b>34</b>
<b>Ábrák jegyzéke</b>	<b>35</b>

# 1. fejezet

## Bevezető

Felmerülhet bennünk az a kérdés, hogy miért is van szükségünk ilyen weboldalakra? A válasz egyszerű, hiszen számos olyan felhasználó létezik aki szórakozásképpen használja a képekkel rendelkező weboldalakat.

A projektem célja egy olyan weboldal létrehozása, amely személyreszabott tartalomajánlást nyújt minden felhasználónak, szórakozás céljából.

A megvalósításhoz első sorban létrehoztam egy weboldalt, mely kategóriánként tartalmazza a képeket. Kezdetben a felhasználó kategóriánként böngészhet a képek közt. Jelenlegi projektemben hét kategória van jelen, melyek a következők: Animals, Car, bike, Retro, Movies, Food, Event, Science. A felhasználó ezen kategóriákon belül figyelheti a képeket, és tekintheti meg jobban azokat amelyek érdeklik.

Az ajánlás megvalósításához a felhasználónak regisztrálni kell, majd ha rendelkezik fiókkal, akkor be kell jelentkeznie. A regisztrációhoz szükséges megadnia a Vezetéknévét, a Keresztnevét, illetve az email címét, majd végül a választott jelszavát is. Ahhoz hogy a weboldal ajánlani tudjon a felhasználónak, a felhasználó bejelentkezve kell a képeket megtekintse. A bejelentkezéshez meg kell adni az email címét, majd a jelszavát. Ha bejelentkezés után böngészett a képek között, akkor az Ajánlás oldalra olyan képek kerülnek melyeket a legérdekesebbnek tart.

Egy személyre szabott tartalomajánlás weboldalon többféleképpen működhet, megvalósítható. Az általam kidolgozott algoritmus a megvalósításhoz a következő: Miután a felhasználó bejelentkezett és elkezd nézegetni a különböző kategóriájú képeket, minden rákattintott kép felnagyított formában jelenik meg, és a szoftver méri az adott képen töltött idejét a felhasználónak, illetve figyeli azt is, hogy melyik kategóriába tartozik az adott tartalom. Ha a képen töltött idő meghaladja az öt szekundumot, akkor a szoftver eltárolja ezt a kategóriát, amelyen lévő képen időzött a felhasználó, majd egy külön oldalra (Ajánlás) illeszti a nézett kategóriájú képeket, hogy ott könnyebben és nyugodtabban tudjon böngészni azok a tartalmak között, amelyek őt legjobban érdeklik.

Tehát, a felhasználók által a képekre kattintva megjelenő nagyobb méretű nézet lehetőséget nyújt részletesebb vizsgálatra. Ez a funkció teszi lehetővé, hogy a felhasználók felfedezhessék a weboldal tartalmait, és mélyebben megismerhessék azokat. Az eltöltött idő rögzítése lehetővé teszi, hogy egy pontosabb képet kapjunk a felhasználók érdeklődéséről egy adott tartalom iránt. A rendszer figyelemmel követi, hogy melyik tartalomra kattintott a felhasználó, valamint követi, hogy melyiket tekintette meg részletesebben. Ezen keresztül gyűjtött információk segítségével képes személyre szabni a weboldalon lévő

tartalmakat. Az idő mérése lehetővé teszi, hogy pontosabban meghatározzuk a felhasználók érdeklődését. Ha valaki hosszabb ideig tekint meg egy képet mint öt szekundum, az azt jelentheti, hogy nagyobb érdeklődést mutat az adott tartalom iránt.

A rendszer biztonsága érdekében a felhasználói bejelentkezéshez a JWT, Jason Web Token-t használok, amely egy token alapú hitelesítési mechanizmus, a bejelentkezés után generálódik és a felhasználó biztonságos azonosítását és védelmét szolgálja. Használatával a felhasználói azonosító és más privát adatok nem a kiszolgálón tárolódnak, hanem a tokenben kódolva vannak eltárolva. Ez a módszer, alkalmazás jelentősen csökkenti az adatvesztélyeket, mivel maga a token nem tartalmazza a felhasználó bizalmas adatait, hanem ehelyett a szükséges azonosítási információkat foglalja magába, amelyek a kliens és szerver közötti kommunikáció során használatosak.

A rendszer megvalósításához, pontosabban a frontend fejlesztéséhez a JavaScript(JS) és ReactJS keretrendszert használtam. A backend fejlesztéséhez és a rendszer üzemeltetéséhez pedig a C# programozási nyelvet és az Entity Framework Core-t használtam.

## 2. fejezet

# Programok, technológiák bemutatása

### 2.1. Frontend Fejlesztése

A React JS egy rendkívül népszerű keretrendszer az alkalmazások frontendjeinek fejlesztéséhez, ezért is választottam használatát. A Facebook fejlesztette ki a dinamikus és interaktív felhasználói felületek létrehozására. A JavaScript alapú könyvtár, komponens alapú megközelítést kínál, amely lehetővé teszi a fejlesztő számára az alkalmazás önálló egységeire való felépítését, amelyek újrahasználhatóak és könnyen kezelhetőek. Ezért választottam a React JS-t a frontend megvalósításához.

#### 2.1.1. Learn - [React](#)

A React-nek számos előnye van, amiért használni érdemes.

- **Komponens alapú** fejlesztés: Komponens alapú fejlesztése lehetővé teszi, hogy az alkalmazás önálló egységekre épüljön. Ezek a komponensek olyan egységek, amelyek saját állapotot kezelnek. Ezáltal az alkalmazást jól áttekinthető részekre lehet osztani. A komponensek újra felhasználhatóak, vagyis többször lehet használni ugyanazt a komponenst különböző részben vagy alkalmazásban. Ez időt és erőforrást takarít meg, mivel nem kell ugyanazt a funkcionalitást többször megírni.

```
function App() {  
  const [user, setUser] = useState(null);  
  
  return (  
    <div className="container p-4">  
      <React.Fragment>  
        <Navbar />  
        <div className="container p-4">  
          <Router>  
            <Routes>  
              <Route path="/home" element={<Home />} />  
              <Route path="/animals" element={<Animals />} />  
              <Route path="/car" element={<Car />} />  
            </Routes>  
          </div>  
        </div>  
      </React.Fragment>  
    </div>  
  );  
}
```

```

        <Route path="/science" element={<Science />} />
        <Route path="/retro" element={<Retro />} />
        <Route path="/movies" element={<Movies />} />
        <Route path="/food" element={<Food />} />
        <Route path="/event" element={<Event />} />
        <Route path="/login" element={<Login />} />
        <Route path="/signup" element={<Signup />} />
      </Routes>
    </Router>
  </div>
</React.Fragment>
</div>
);
}

export default App;

```

### 2.1. kódrészlet. React példakód.

A 2.1 kódrészlet az App komponenset mutatja be, amely az alkalmazás gyökéreleme, az itt meghívott további komponensek jelennek meg az alkalmazásban. Ebben a komponensben helyezem el a Navbar és a Router komponenseket. Az alkalmazás útválasztását a Router komponensben helyezem el a Routes és a Route komponensek segítségével. A Routes komponens az összes útvonalat tartalmazza, míg a Route komponens meghatározza az egyes útvonalakhoz rendelt komponenseket.

- **Virtuális DOM:** A DOM egy egyszerű JavaScript objektum, amely hatékony frissítéseket tesz lehetővé. A React JS keretrendszer a virtuális DOM (Document Object Model) használatával növeli az alkalmazások teljesítményét, amely a valós DOM absztrakt reprezentációja. Ez az alkalmazás felhasználói felületének aktuális állapotát tükrözi. Amikor a felhasználó interakcióba lép a weboldallal, például adatokat ír be, ekkor a React változásokat hoz létre a virtuális DOM-ban. Ezután ezeket a változásokat összehasonlítjuk a valós DOM-mal. Az alkalmazásra csak a megváltozott különbségek kerülnek.
- **Nagy közösség és támogatás:** A React keretrendszer széleskörű támogatással rendelkezik. Folyamatosan bővül ez a keretrendszer, és rengeteg olyan online fórum és dokumentáció létezik, amelyek segítik tanulását.
- **Kompatibilitás más könyvtárakkal és keretrendszerekkel:** A React jól integrálható más könyvtárakkal és keretrendszerekkel, lehetővé téve a kód újrafelhasználását és a meglévő rendszerekbe való könnyű beillesztését.
- **Kiváló teljesítmény:** A frissítések lehetővé teszik a gyorsabb reakciókat, illetve a hatékonyabb rendszerfrissítéseket. Ezáltal nagyobb teljesítményt nyújtva, ami fontos szerepet játszik a túlterhelés kezelésében.

Ezek az előnyök együttesen teszik a React keretrendszert egy erőteljes és keresett választássá a frontend fejlesztés terén, ezért alkalmaztam én is ezt a keretrendszert projektem felépítéséért.

## 2.2. Backend Fejlesztése

Projektemben a backend fejlesztéséhez, mint programozási nyelvet, a C#-ot használtam az ASP.NET webalkalmazás-fejlesztési platformmal kombinálva. Használata előnyös több szempontból is. Mivel modern nyelv, valamint objektumorientált, egyszerűbb és hatékonyabb. Olvasható szintaxissal rendelkezik, lehetővé téve a gyorsabb kódolást. Jól optimalizált és magas teljesítményű, így lehetővé teszi a gyors adatfeldolgozást. Az alkalmazásokat nagy terhelés alatt is futtatja, amely az én esetemben használatos volt webalkalmazásom számára.

A dolgoztaomban használt Entity Framework Core kombinálásával könnyen kezeltem az adatbázist. Az Entity Framework Core egy ORM (objektum relációs leképzési keretrendszer), ami hozzájárul az adatbázisok könnyű kezeléséhez, illetve a hatékony adatműveletek végrehajtásához. Ennek a segítségével hoztam létre az adatbázisom, valamint a regisztrációs és bejelentkezési funkciók létrehozásának biztonságát a JWT token segítségével végeztem el.

### 2.2.1. Swagger - Swagger UI

[1] A Swagger lehetőséget nyújt a felhasználók számára, hogy RESTful webes szolgáltatásokat hozzanak létre, dokumentáljanak, teszteljenek. Biztosítja a felhasználók számára a felülről lefele és az alulról felfele irányuló API-fejlesztési megközelítések alkalmazását.

A felülről lefele irányuló megközelítés, amit design-first módszernek is nevezhetünk, azt jelenti, hogy a Swagget az API tervezése előtt használjuk. Ez teszi lehetővé, hogy definiálni lehessen az API struktúráját, paramétereit és válaszait a Swaggerben. Ezután a Swagger kigenerálja a számunkra szükséges kódot és tesztfájlokat, ami alapján fejleszthetjük az API-t.

Az alulról felfele irányuló módszer során, először az API kerül elkészítésre, majd a Swagger segítségével teszteljük azt. Az API kódjából a Swagger automatikusan generál, majd ez alapján létrejön a Swagger UI, így könnyebben megvalósítva a tesztelést. Lényegében tehát az általam is használt Swagger UI egy grafikus felhasználói felület, ami megkönnyíti az API-val való kommunikációt. A felület segítségével könnyebben létrejöhet a tesztelés, illetve könnyebben számon lehet tartani a különböző fontos információkat. Projektemben lévő bejelentkezés és regisztráció, illetve a kategóriák és a képek lekérésének működése egy adott kategória szerint a Swagger UI-ban lett ellenőrizve, tesztelve.

```
builder.Services.AddControllers();
builder.Services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new() { Title = "UserTrack.API", Version = "v1" });
});
```

#### 2.2. kódrészlet. Swagger példakód.

Weboldalam építésekor a Swagger beállítása a 2.2 kódrészlet használatával történt meg. Itt hajtódik végre az API dokumentációjának konfigurálása.

```
// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
```

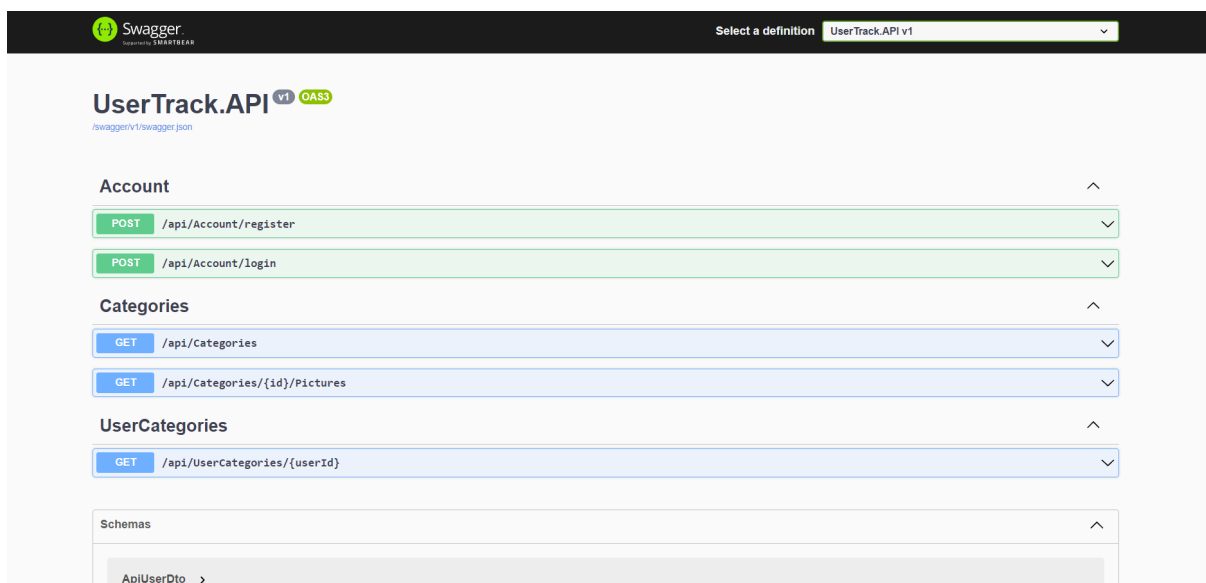
```

{
    //join
    app.UseDeveloperExceptionPage();
    //
    app.UseSwagger();
    app.UseSwaggerUI((c => c.SwaggerEndpoint("/swagger/v1/swagger.json",
        "UserTrack.API v1")));
}

```

### 2.3. kódrészlet. Swagger UI példakód.

A 2.3 kódrészlet segítségével elérhetővé válik az alkalmazás fejlesztési környezetében a Swagger UI. Beállítva azt az útvonalat, ahol az API Swagger JSON dokumentáció elérhető lesz, megkönnyítve a tesztelést, illetve böngészést.



2.1. ábra. Swagger UI felület megjelenése

A 2.1 ábrán látható Account rész, az AccountController által lett megvalósítva, ahol POST-tal küldöm a szerver fele az adatokat, majd le tudom azokat kérni.

A POST kéréshez megadom a szerver címét és az API végpontot. A szerver feldolgozza a kérést, majd válaszként visszaad egy status kódot és válaszdatot, amelyet a webalkalmazás megkap és feldolgoz.

Hasonlóképpen készítettem el a Categories és Pictures adatokat is. Amelyek GET-el megkapják az adatbázisban lévő kategóriákat, illetve, ha megadom egy kategória ID-ját, akkor visszakapom abból a kategóriából származó képeket.

### 2.2.2. Jason Web Token - JWT

[2] A JSON Web Token(JWT) egy olyan nyílt szabvány, amely egy kompakt és önálló módszert határoz meg a felek közötti biztonságos információátvitelre JSON objektumnként. Ez az információ ellenőrizhető és megbízható, mivel digitálisan alá van írva. A JWT-t aláírhatók titkos, vagy nyilvános-magán kulcspárral. Bár a JWT-k titkosíthatók

a felek közötti titkosság biztosítása érdekében is, én az aláírt tokenet emelném ki. Ezek a tokenek képesek ellenőrizni a bennük foglalt állítások integritását, míg a titkosított tokenek elrejtik ezeket az állításokat más felek elől. Ha a tokenek aláírása nyilvános-magán kulcspárokkaal történik, az aláírás azt is igazolja, hogy csak a magánkulcsot birtokló fél írta alá a tokenet.

[2] Miért előnyös használni ezt a tokenet? Miután a felhasználó sikeresen bejelentkezett és megkapta a JWT-t, azután minden további kérés, amit a felhasználó küldhet a szervernek, tartalmazni fogja ezt a tokenet. A JWT lehetővé teszi a szerver számára, hogy azonosítsa a felhasználót, valamint meghatározza, hogy mikhez van jogosultsága. Az egyszeri bejelentkezés (Single-sign-on, SSO) egy olyan funkcionális, amelyben a felhasználó csak egyszer jelentkezik be a rendszerbe, és ezután automatikusan hozzáférhet más rendszerekhez és alkalmazásokhoz, anélkül, hogy újra be kéne jelentkezzen.

A JWT-t széles körben használják az SSO megvalósításához, mivel könnyen kezelhető, és erőforrásigénye kicsi. Egyetlen JWT Token elég ahhoz, hogy a felhasználó hozzáférhessen különböző tartományokban lévő szolgáltatásokhoz és erőforrásokhoz, ezáltal csökken a bejelentkezési folyamat ismétlése. JWT használatának módszere kényelmes a felhasználók számára, mivel csak egyszer kell bejelentkezniük, és azután automatikusan hozzáférhetnek az engedélyezett funkciókhoz a megfelelő token segítségével. A Jason Web Token szerkezete három elemből épül fel. Fejléc, tartalom és aláírás. Ezek az elemek pontokkal vannak elválasztva egymástól.

[2] A fejléc JSON formátumban tartalmazza az aláírás típusát és az alkalmazott algoritmust. Ezután a JSON-t a Base64Url kódolással a JWT első részévé alakítjuk. A fejléc információkat nyújt a token kezeléséhez és értelmezéséhez.

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

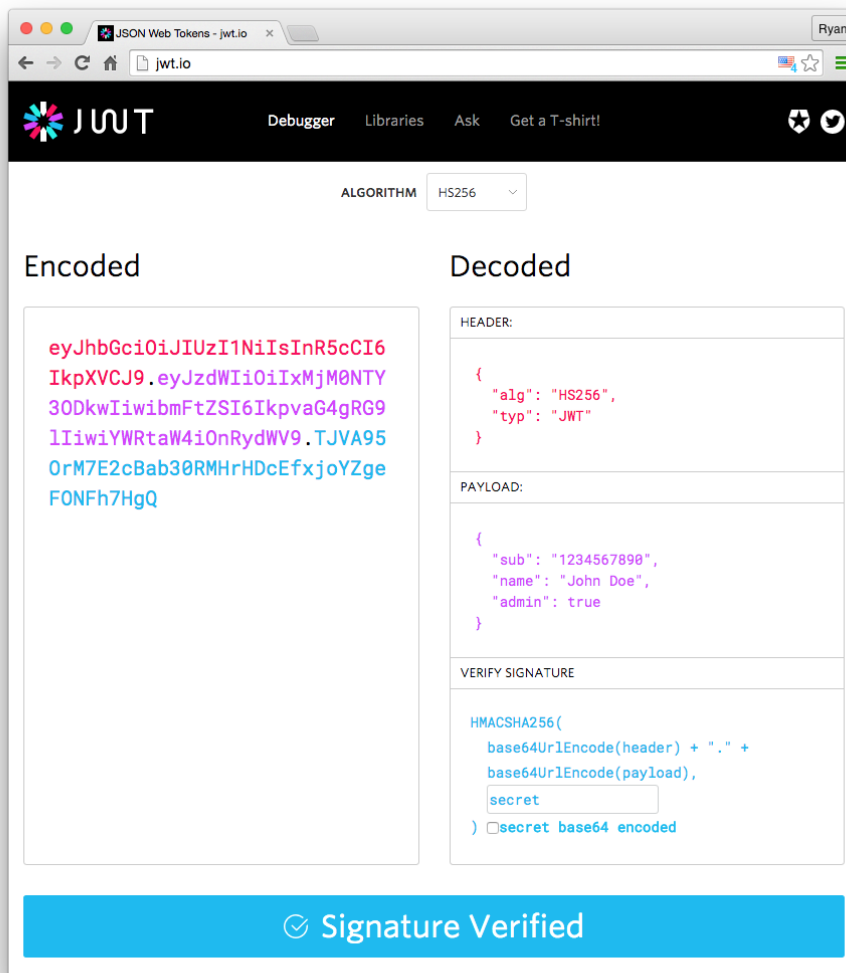
#### 2.4. kódrészlet. Példa Fejlécre.

A token második eleme a tartalom, amely különböző követeléseket tartalmaz. [2] Ez a rész állításokat tartalmazhat, például felhasználó azonosítója és más kiegészítő információk. Meghatározza a tokenelkapcsolatos jogosultságokat és a felhasználóhoz kapcsolódó adatokat. [2] Az állítások háromféleképpen lehetnek: regisztrált állítás, nyilvános állítás és privát állítás. A regisztrált állítások előre meghatározott állítások, nem kötelezőek, de ajánlott, mivel hasznos készletet biztosítanak. A nyilvános állításokat a JWT tetszőlegesen definiálja. A privát állítások olyan egyéni állítások, amelyeket az információ megosztására hoznak létre olyan felek között, akik egyetértenek használatukban és nem regisztráltak vagy nyilvános állítások.

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

#### 2.5. kódrészlet. Példa Tartalomra.





2.2. ábra. [2] Jason Web Token

A harmadik fontos elem az aláírás, ami a JWT biztonsági eleme. Kiszámításához a fejléc és a tartalom egy titkos kulcs segítségével kerül kódolásra. Ellenőrzésekor a szerver újból kiszámítja a kapott fejlécből és tartalomból, majd összehasonlításra kerül az eredeti aláírással, biztosítva így a JWT megbízhatóságát.

Ezek a fontos elemek járulnak hozzá a JWT felépítéséhez (lásd 2.2 ábra), amely a felhasználó hitelesítésének érdekében használatos. A fejléc és a tartalom olvasható információkat tartalmaz, míg az aláírás a token, védő, kódolt formában. A szerver a JWT ellenőrzése során dekódolja és ellenőrzi az aláírást titkos kulcs segítségével, hogy biztos legyen a token érvényessége. [2] Tehát önmagában a Jason Web Token kimenete egy három ponttal elválasztott Base64Url karakterlánc, amelyet könnyen lehet továbbítani HTML- és HTTP- környezetben. A Base64Url kódolás lehetővé teszi, hogy a JWT-t egyszerűen beágyazzuk URL-ekbe anélkül, hogy nagy helyet foglalna.

Projektemben a következőféleképpen állítottam be a Jason Web Tokent. A 2.6 kódrészlet a a JWT hitelesítési beállításait konfigurálja az ASP.NET Core alkalmazásomban.

Az alkalmazás beállításban különféle konfigurációk és szolgáltatások adódnak hozzá, melyek a következőképp néznek ki:

```
builder.Services.AddAuthentication(options => {
    options.DefaultAuthenticateScheme =
        JwtBearerDefaults.AuthenticationScheme; // "Bearer"
    options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
}).AddJwtBearer(options => {
    options.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuerSigningKey = true,
        ValidateIssuer = true,
        ValidateAudience = true,
        ValidateLifetime = true,
        ClockSkew = TimeSpan.Zero,
        ValidIssuer = builder.Configuration["JwtSettings:Issuer"],
        ValidAudience = builder.Configuration["JwtSettings:Audience"],
        IssuerSigningKey = new
            SymmetricSecurityKey(Encoding.UTF8.GetBytes(builder.Configuration["JwtSettings:Key"]));
    };
});
```

## 2.6. kódrészlet. JWT használata.

Az `AddAuthentication` metódus hozzáadja az azonosítás szolgáltatást az alkalmazáshoz és beállítja az alapértelmezett hitelesítési sémát a `JwtBearerDefaults.AuthenticationScheme`-re, amely a "Bearer" sémát jelenti, vagyis innentől a `JWTBearer` hitelesítést fogja használni. Az `AddJwtBearer` metódus további `JWTBearer` gitelesítési beállításokat konfigurál. Ezen beállítások a `TokenValidationParameters` objektumban vannak meghatározva, és a következő paramétereket tartalmazza:

- **ValidateIssuerSigningkey:** Ellenőrzi az érvényesítés során az aláírás kulcsát. Ha értéke `true` akkor ellenőrzésre kerül az aláírás a megadott kulccsal.
- **ValidateIssuer:** Ellenőrzi a kiadó azonosítóját. Ha értéke `true`, összehasonlításra kerül a token kiadója a konfigurációban megadott kiadóval.
- **ValidateAudience:** Ellenőrzi a célközönség azonosítóját. Ha értéke `true`, akkor összehasonlítja a token célközönségét
- **ValidateLifetime:** Meghatározza a token élettartalmának érvényességét. Ellenőrzi, hogy a token lejárt-e vagy sem.
- **ClockSkew:** Lejárat és érvényesség közötti időeltérést határozza meg. Lehetővé teszi az időeltérést a token érvényességi ideje és helyi ideje között.
- **ValidIssuer:** Azonosító, mellyel a token kiadója kerül ellenőrzésre.
- **ValidAudience:** Azonosító, mellyel a token célközönsége kerül ellenőrzésre.
- **IssuerSigninKey:** Titkos kulcs, mely az aláírás ellenőrzéséhez szükséges.

Ezek azok a paraméterek az alkalmazásban, amelyek által ellenőrizhetőek a beérkező JWT tokenek, illetve azok érvényessége és megbízhatósága.

## 2.3. Összehasonlítás

[3] Az alkalmazásokban általában szükség van a bejelentkezési "munkamenet" fenntartására, hogyha egy felhasználó egyszer bejelentkezett, akkor továbbra is hozzáférhesen az alkalmazás különböző részeihez anélkül, hogy újra és újra be kellene jelentkeznie minden egyes HTTP kérésnél. A webalkalmazásokban a felhasználók hitelesítése és állapotának nyomonkövetése nagyon fontos szerepet játszik. Ehhez két különböző megoldást alkalmazhatnak. Az egyik ilyen a munkameneti-sütik alkalmazása, míg a másik az általános tárgyalta Jason Web Token(JWT) alkalmazása.

A munkameneti-sütik alkalmazása a hagyományos módszert képviselik. A szerver küld egy sütit(adatdarabot) a böngészőnek, amelyben az azonosító található. Minden kérésnél ezt a böngésző automatikusan továbbítja a szervernek, majd a szerver ez alapján tudja azonosítani a felhasználót, illetve kezelni annak munkamenetét. Használatuk egyszerű, viszont erőforrásigényesek és nagy a biztonsági kockázatuk.

### 2.3.1. Megközelítések összehasonlítása

- **Munkameneti süti alapú megközelítés:** [3] A kiszolgáló létrehoz egy sessionId-t, melyet a titkos kulcs segítségével ír alá és elmenti ezt az Id-t egy sessionDB-be, majd a sessionId-t tartalmazó cookiet elküldi a böngészőnek. A böngésző miután megkapta ezt a cookiet, elmentia cookie tárolóba, majd minden további kiszolgálóhoz intézett kérésbe beépíti.
- **JWT JSON webes token megközelítés:** [3] A szerver létrehoz egy accessToken-t, amely a userId és más információkat titkosít az ACCESS\_TOKEN\_\_SECRET kóddal és elküldi ezt a token a böngészőnek. Miután a böngésző megkapta, elmenti az oldalon, és ez a token minden további kiszolgálóhoz intézett kérésbe bekerül.

Összességében mind a munkameneti sütik, mind a JWT-k hatékony megoldást képviselnek, viszont én a projektben a JWT-t biztonságosabbnak preferáltam az aláírás, érvényesség és rugalmasság szempontjából.

## 3. fejezet

# Követelmények

Az alkalmazás megfelelő és gördülékeny működéséhez számos követelménynek kell megfelelnie úgy a felhasználónak mint a rendszernek. Az alábbiakban ezeket a követelményeket fogom bemutatni.

### 3.1. Felhasználói Követelmények

A felhasználói követelmények meghatározása rendkívüli fontos szerepet játszik projektem felépítése, tervezése során. Legfontosabb felhasználói követelmények, melyek hozzájárulnak a projekt megfelelő tervezéséhez:

- **Személyre szabott tartalom:** A weboldalnak olyan tartalmakat kell ajánlania a felhasználók számára, amelyek felkeltik érdeklődését, valamint az ő preferenciáihoz igazodjon. Figyelemmel kell követni, melyek azok a tartalmak, amelyeken a felhasználó többen időzik, így meghatározva azt, melyek iránt nagyobb az érdeklődése.
- **Bejelentkezési lehetőség:** A felhasználó be tudjon jelentkezni a weboldalra, így a bejelentkezett felhasználót egyszerűbb követni, mely tartalmakat nézi szívesebben.
- **Egyszerű felhasználói felület:** A navigáció és a tartalomhoz való hozzáférésnek könnyen elérhetőnek kell lennie. A felhasználónak könnyen meg kell találnia az érdekes tartalmakat, emiatt vannak weboldalamon a képek kategorizálva.
- **Responsive Design:** A weboldalnak responsivenak kell lennie, ahhoz hogy a tartalmak megfelelően jelenjenek meg a különböző eszközökön, és alkalmazkodni tudjanak a különböző képernyőméretekhez. Ez az alkalmazás használatát biztosítja mindenféle eszközön.
- **Adatvédelem és biztonság:** A bejelentkezett felhasználó adatainak biztonságos tárolása.(JWT)

### 3.2. Rendszer Követelmények

A következő rendszer követelmények projektem működését, funkcionalitását, illetve elérhetőségeit határozzák meg. Az alábbiakban összefoglalom projektem rendszerkövetelményeit:

### 3.2.1. Funkcionális Követelmények

- **ASP.NET szerver** kiszolgálja a kéréseket és kezeli a számítást
- **React alkalmazás** megfelelően kezeli a felhasználók preferenciáit és előzményeit a tartalomajánlás során
- **Felhasználó** regisztrálása, illetve bejelentkezése(JWT Token generálása)
- **Ajánlott** tartalom megjelenítése

### 3.2.2. Nem Funkcionális Követelmények

- **Hardverkövetelmény:** megfelelő erőforrás a zavartalan futáshoz, illetve megfelelő hardveres erőforrás(CPU, RAM) a kérések megfelelő kiszolgálásáért
- **Operációs Követelmények:** React, ASP.NET Core futtatása különböző operációs rendszereken
- **Szoftverkövetelmények:** Node.js, React könyvtárak, Redux telepítése megfelelő verzióban, valamint a backendhez a .NET Core SDK és a szükséges NuGet csomagok telepítése(Entity Framework Core, Identity Framework)
- **Hálózatkövetelmények:** a frontendnek és backendnek egy hálózaton kell kommunikálnia a megfelelő működéshez, ha nem, akkor Cors használata, ahhoz hogy összekötve működjenek.
- **Biztonsági követelmények:** felhasználói adatok megfelelő titkosítása, adatbázis hozzáférés korlátozása, felhasználói jogosultságok kezelése
- **Teljesítménykövetelmények:** A React alkalmazásnak és az ASP.NET Core szervernek megfelelő teljesítménnyel kell rendelkeznie ahhoz, hogy a személyre szabott ajánlás a leghatékonyabban működjön és gyors válaszidőt biztosítson a felhasználónak.

## 4. fejezet

# Megjelenés megvalósítása

Az általam készített React projekt izgalmas és kihívást jelentő feladat volt. Számos fontos lépést kellett megvalósítanom, annak érdekében, hogy létrehozzam a kívánt megjelenést.

Először is megterveztem a projektem, valamint összefoglaltam a követelményeket a webalkalmazás megvalósításához. Ez segített abban, hogy tisztábban láthassam a végleges eredményt és megtervezzem a megfelelő struktúrákat, komponenseket. Az ehhez való megvalósításhoz a react keretrendszert választottam a projekt alapjául, mivel hatékony és népszerű a felhasználói felületek létrehozásának. A React előnyei között található a komponens alapú fejlesztési módszer, a virtuális DOM (Document Object Model) és az egyszerű újrafelhasználhatóság. Alkalmazásom megjelenítését a CSS (Cascading Style Sheets) segítségével valósítottam meg. Ezek a megfelelő stíluslapok valósították meg azt, hogy vonzó és felhasználóbarát felületet hozzak létre. A CSS keretrendszerek, mint például a Bootstrap használata szintén hasznomra vált az egyszerűbb fejlesztés megvalósításához.

### 4.1. Node Package Manager - npm

React projektben az npm nagyon fontos szerepet játszik. Az npm (Node Package Manager) egy Javascript könyvtár, amelyet a felhasználói felületek fejlesztésére használnak, illetve segítségével könnyedén kezelhetővé válnak a React-hez kapcsolódó csomagok. Ahhoz, hogy létrehozzuk a projektet, először a Node.js telepítésével kell kezdeni, ezután az npm automatikusan telepítve lesz. A weboldal fejlesztéséhez szükséges könyvtárakat, csomagokat az npm telepíti a node\_modules mappába és nyilvántartja ezeket a csomagokat a package.json fájlban is.

### 4.2. Képek elérése - [Pexels API](#)

A weboldalamon megjelenő képek a Pexels API által lekért képek, mely egy olyan fejlesztői felület, amely lehetővé teszi a hozzáférést a Pexels platform tartalmához és szolgáltatásaihoz. Az Api révén történik a képek lekérése, valamint tartalmak letöltése. A Pexels Api lehetővé tette számomra, hogy nyugodtan és tágabban böngézhessenek a képek között, ezáltal megvalósítva azt, hogy használhassam a projektben szereplő képek megjelenítéséért. Fontos tudni azt, hogy ez az Api ingyenes használható, viszont regisztrálni szükséges az Api kulcs eléréséhez.

Bemutatom, hogyan értem el azt, hogy ezen a platformon lévő képeket megkapja az én alkalmazásom is. Minden kategóriának külön komponenszt hoztam létre, amely magában meghívja a Card.js komponensemét, ami a kártyákat jeleníti meg a képek betöltésekor és ezekre a kártyákra töltődnek a Pexels Api-ból megszerzett képek.

```
export const Card = ({ description, imageUrl, onClick }) => {
  return (
    <a href="#" onClick={onClick}>
      <div className="w-full rounded overflow-hidden shadow-lg m-2">
        <div className="image-container">
          <img className="object-cover" src={imageUrl.sources.large}
            alt={description} />
        </div>
      </div>
    </a>
  );
};
```

#### 4.1. kódrészlet. Card komponens kártyák megjelenéséért.

A 4.1 kódrészlet, bemutatja a Card komponenszt, mely kattintható, vagyis rákattintva meghívódik egy olyan függvény, amely felnagyítja az adott kártyán lévő képet és a felhasználó időt tölthet el azon kép nézésével. Ez a komponens olyan parancsot vár, amely az Apiból való képet kéri le.

```
const fetchData = async () => {
  try {
    const response = await
      axios.get("https://localhost:5001/api/Categories/2/Pictures");
    setPictures(response.data);
    setLoading(false);
  } catch (error) {
    console.error(error);
    setLoading(false);
  }
};
```

#### 4.2. kódrészlet. Végpont elérése.

A 4.2 kódrészlet bemutatja azt, amikor a frontenden belül lévő kategória komponensekben az axios könyvtár segítségével aszinkron módon kérést indítok a megadott URL-en keresztül, melynek célja a backenden lévő végpont elérése, amely végül az adott kategóriához tartozó képeket adja vissza.

```
{
  "id": 1164778,
  "sources": {
    "large": "https://images.pexels.com/photos/1164778/pexels-photo-1164778.jpeg"
  }
}
```

#### 4.3. kódrészlet. Apiból való lekérés eredménye.

A 4.2 kódrészleten bemutatásra kerül az, hogy hogyan kapja meg az adott kategóriájú képet az alkalmazás az Apiból.

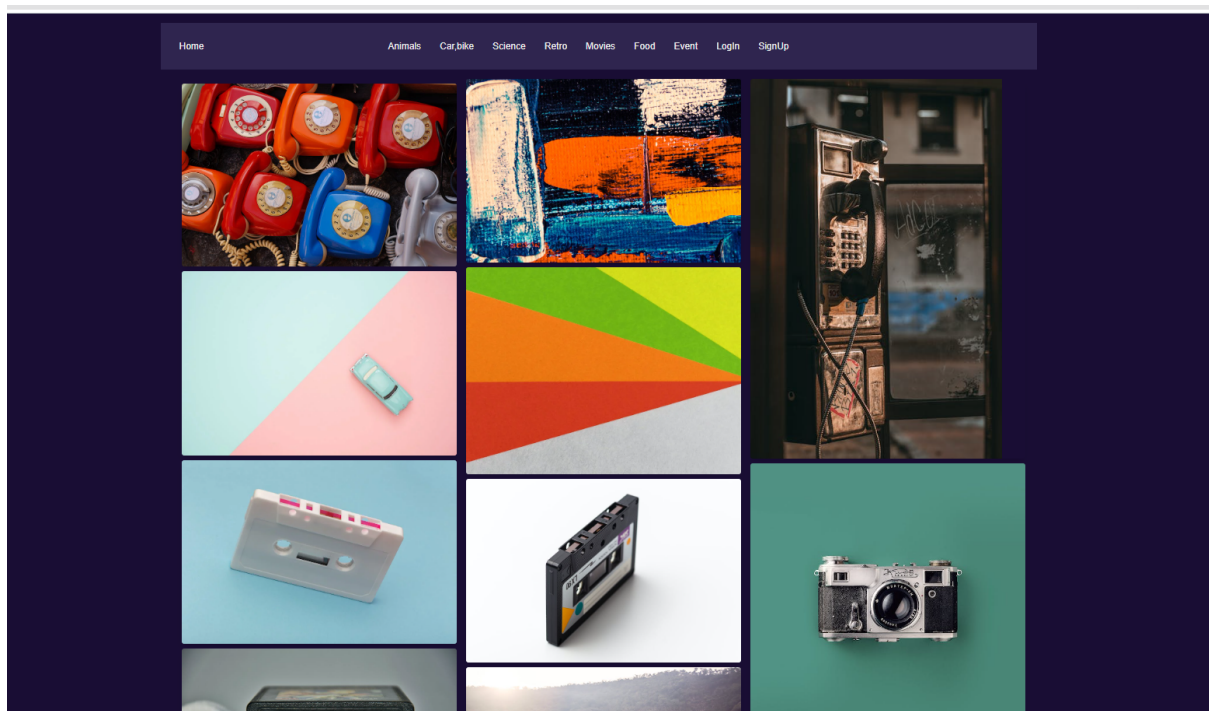
Ezek a kérésekhez hasonlóan történik a bejelentkezés és regisztráció is, mivel ugyanúgy egy aszinkron függvényt tartalmaznak, amely egy POST kérést indít a megadott URL-en, megcélózva azt a végpontot, amely a helyi szerveren (<https://localhost:5001>) fut.

### 4.3. Összességében

Összességében elmondhatom, hogy az általam létrehozott React webalkalmazás fejlesztéséhez kezdetben készítettem egy áttekintést a rendszerről, majd a követelmények meghatározása után megterveztem a felületet. Nagyszerű választásnak bizonyult a komponens alapú fejlesztési módszer és a virtuális DOM használata, mivel ez tette lehetővé azt, hogy hatékonyan és gyorsan létrehozzam a felhasználói felületet.

Az alkalmazásomban megjelennek a képek kategóriák szerint, amelyeket a pexels Api segítségével értem el. A szerverrel való kommunikáció aszinkron módon történik a Reactben lévő axios könyvtár segítségével.

A képek letöltéséhez, eléréséhez a GET kéréseket használtam kategóriánként, míg a POST kéréseket a regisztrációhoz, illetve bejelentkezéshez.



4.1. ábra. Weboldal Retro kategóriája alatt lévő képek az Apiból



## 5. fejezet

# Szoftver

### 5.1. A szoftver bemutatása

A projektben lévő rendszer egy személyre szabott tartalomajánló rendszert mutat be. A webalkalmazást Reactben fejlesztettem, míg a háttérben zajló műveletek, lekérések ASP.NET Core C nyelvben íródtak. A bejelentkezés és regisztráció elmentéséhez, és biztonságos megőrzéséhez a Jason Web Tokenet használtam(JWT), amely egy automatikusan választott idő elteltével megváltozik, így csakis a felhasználó használhatja fiókját, megelőzve azt, hogy bárki belépjen és megzavarja a felhasználót, ellentétben a hagyományos cookie módszerrel.

A bejelentkezéshez szükséges a felhasználónak egy email címmel és passworddal rendelkeznie, mely nem csupán egy akármilyen egyszerű password, hanem tartalmaznia kell 0-9ig számjegyet, nagybetűt, kisbetűt, illetve egy non alphabetic változót is.

Regisztrációnál a felhasználó meg kell adja a Vezetéknévét, Keresztnevét, amelyek eltárolódnak az adatbázisba. Miután a felhasználó regisztrált, ajánlatos bejelentkeznie, mivel csak bejelentkezés után mentődnek el azok a képek a profiljába, amelyek érdeklik, az általam megvalósított módszer szerint azok, amelyek kinagyított formában több mint öt másodpercig voltak megnyitva. Miután a regisztráció és bejelentkezés megtörtént, a Home pagen random jelennek meg a pexels Apiból lekért képek kategorizálva, majd bejelentkezés után a Profileban jelennek meg azok a képek, amelyek személyre szabottak.

```
const fetchData = async () => {
  try {
    const response = await
      axios.get("https://localhost:5001/api/Categories/4/Pictures");
    const data = response.data.map((image) => ({
      imageUrl: image.imageUrl,
      description: image.description,
    }));

    setPictures(response.data);
    setLoading(false);
  } catch (error) {
    console.error(error);
    setLoading(false);
  }
}
```

```
}  
};
```

### 5.1. kódrészlet. Képek lekérése a backendből.

Az 5.1 kódrészlet alapján a fetchData-t és axios csomagot használtam, ahhoz, hogy lekérjem a képeket az apiból, illetve arra is, hogy a lekért képeket, URL-eket megkapjam a backendtől. A fetchData egy aszinkron függvény, amely az axios.get metódust használja a képek lekérésére, jelen esetben a `https://localhost$5001/api/Categories/categoryId/Pictures` végponton keresztül. A választ feldolgozza és eltárolja a pictures változóba a képeket.

```
// GET: api/Categories/5/Pictures  
[HttpGet("{id}/Pictures")]  
public async Task<ActionResult<IEnumerable<PictureDto>>>  
    GetPicturesByCategoryId(string id)  
{  
    // Fetch the category from the repository  
    var category = await _categoriesRepository.GetAsync(id);  
  
    if (category == null)  
    {  
        return NotFound();  
    }  
  
    // Make an HTTP request to the Pexels API endpoint to retrieve  
    // pictures of the given category  
    HttpResponseMessage response = await  
        _httpClient.GetAsync($"https://api.pexels.com/v1/search?query={category.CategoryName}");  
  
    if (response.IsSuccessStatusCode)  
    {  
        var responseContent = await  
            response.Content.ReadAsStringAsync();  
        var apiResponse =  
            JsonConvert.DeserializeObject<PexelsApiResponse>(responseContent);  
        var pictures =  
            _mapper.Map<List<PexelsPhoto>>(apiResponse.Photos);  
        return Ok(pictures);  
    }  
    else  
    {  
        // Handle the API error response  
        return StatusCode((int)response.StatusCode,  
            response.ReasonPhrase);  
    }  
}
```

### 5.2. kódrészlet. Képek backendhez való átadása.

Az 5.2 kódrészlet alapján a backend megkapja a képeket. Az itt bemutatott kódrészlet egy GET HTTP kérést hajt végre az api-n keresztül a kategóriához tartozó képek lekéréséhez. A kód az ASP.NET Web Api alkalmazásom CategoriesController nevű fájlban található. Az alkalmazásomban lévő kontrollerek felelősek a HTTP kérések fogadásáért és feldolgozásáért. A repositoryk az adatok tárolásáért és kezeléséért, valamint az interfészek pedig biztosítják az absztrakciót az adatelérési réteg és az alkalmazás logikája között.

## 5.2. Ajánlás algoritmus

Webalkalmazásomban használt ajánlásom algoritmus a következőképpen valósítható meg. Miután a felhasználó bejelentkezett, elkezdi nézni a képeket, akár a Home page, akár kategorizálva, a rendszer méri a felnagyított képek megnyitási idejét, ezt az időt elküldi a backendnek és a backend kiválasztva ezeket a képeket, visszaküldi a frontendnek, az ajánlott képek közé. Ha egy képen több mint 5 másodpercet időzik a felhasználó, nagy valószínűséggel, hogy érdeklik őt az adott tartalom.

```
const sendToBackend = async () => {
  if (timeSpent > 5) {
    try {
      const response = await
        axios.get("https://localhost:5001/api/Categories/info");
      console.log(response.data);
    } catch (error) {
      console.error(error);
    }
  }
};

const onGrabData = (currentPage) => {
  return new Promise((resolve) => {
    setTimeout(() => {
      const data = pictures.slice(
        ((currentPage - 1) % TOTAL_PAGES) * NUM_PER_PAGE,
        NUM_PER_PAGE * (currentPage % TOTAL_PAGES)
      );
      resolve(data);
    }, 280);
  });
};
```

### 5.3. kódrészlet. Ajánlott képek átadása backendnek.

Az 5.4 kódrészlet bemutatja a következőket: Az onGrabData függvény felelős az adatok lekéréséért, illetve benne hajtódik végre az idő meghatározása is. Amikor ezt a függvényt meghívjuk, megadjuk neki a jelenlegi oldal számát és a függvény aszinkron módon visszaadja a kép adatát. A senToBackEnd függvény ellenőrzi a timeSpent változó által, ami azt jelzi, hogy a felhasználó mennyit időzött az adott képen, ha ez nagyobb mint 5 másodperc, akkor elküld egy GET kérést az adott végpontra a backend szerverhez.

```

[HttpGet("info")]
public async Task<ActionResult<IEnumerable<PexelsPhoto>>>
    ReceiveFromFrontend(string category)
{
    try
    {
        given category
        HttpResponseMessage response = await
            _httpClient.GetAsync($"https://api.pexels.com/v1/search?query={category}&p=1");

        if (response.IsSuccessStatusCode)
        {
            var responseContent = await
                response.Content.ReadAsStringAsync();
            var apiResponse =
                JsonConvert.DeserializeObject<PexelsApiResponse>(responseContent);

            var pictures =
                _mapper.Map<List<PexelsPhoto>>(apiResponse.Photos);

            var filteredPictures = pictures.Where(p => p.OpenDuration >
                3);

            return Ok(filteredPictures);
        }
        else
        {
            // Handle the API error response
            return StatusCode((int)response.StatusCode,
                response.ReasonPhrase);
        }
    }
    catch (Exception ex)
    {
    }
}

```

#### 5.4. kódrészlet. Ajánlott képek visszaküldése frontendnek.

Az 5.5 kódrészlet a `ReceiveFromFrontEnd` nevű HTTP GET végpontot prezentálja a backend alkalmazásban. Ez a végpont az alkalmazás részétől kap adatokat, jelen esetben az adott kategória nevét, mely érdekelheti a felhasználót. Ha megtalálta azokat a kategóriákat melyekből adott idő alatt nézett képet, akkor azokat megkeresi a Pexels Apiból és visszaküldi a frontendnek. Végül megjelennek a képernyőn a személyre szabott tartalmak.

### 5.3. Adatbázis

```

"ConnectionStrings": {
  "DefaultConnection":
    "Server=(localdb)\\MSSQLLocalDB;Database=UserTrack;Trusted_Connection=True;MultipleActiveResultSets=true",
},

```

### 5.5. kódrészlet. Connection.

A fenti kódrészlet egy kapcsolódási karakterláncot definiál, amely neve DefaultConnection. A karakterlánc értéke meghatározza, hogy az alkalmazás mely adatbázis szerverhez és adatbázishoz kapcsolódik. A server részben megtalálható a szerver neve, IP címe, amelyen az adatbázis fut. A localdb MSSQLLocalDB rész egy adatbázis szerver neve, amely a Microsoft SQL Server Express LocalDb-t jelenti, itt találhatóak az adatok.

```

using KnowItAll.API.Data.Configurations;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace UserTrack.API.Data
{
    public class UserTrackDbContext : IdentityDbContext<ApiUser>
    {
        public UserTrackDbContext(DbContextOptions options) : base(options)
        {
        }
        public DbSet<Category> Categories { get; set; }
        public DbSet<Picture> Pictures { get; set; }
        public DbSet<UserCategory> UserCategories { get; set; }
        public DbSet<UserPicture> UserPictures { get; set; }
        public DbSet<ApiUser> User { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);
        }
    }
}

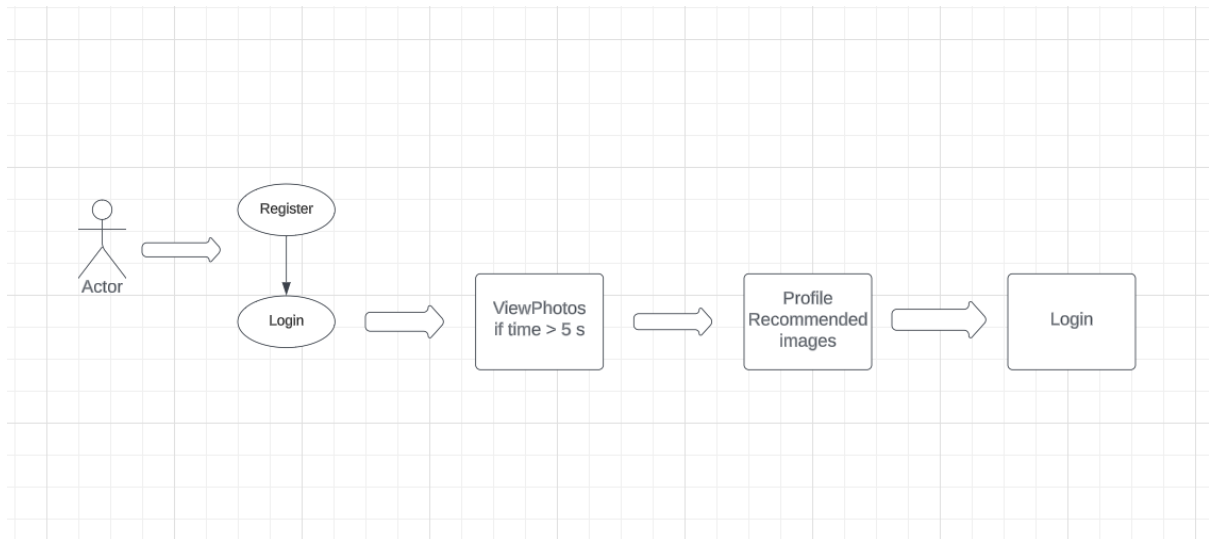
```

### 5.6. kódrészlet. EFC

A fenti kódrészlet egy adatbázis kontextus osztályt hoz létre, amelyet az Entity Framework Core használ a UserTrack adatbázis kezelésére. Az osztály az IdentityDbContext<ApiUser> osztályból származik, ami azt jelenti, hogy az alkalmazás az Identity keretrendszert használja a felhasználókezeléshez.

## 5.4. Diagram

### 5.4.1. Use Case diagram



**5.1. ábra.** Use case Diagramm

A felhasználó kezdetben be tud jelentkezni, majd a regisztráció után újból be kell jelentkezni az ajánlás érdekében. Bejelentkezés után elkezd böngészni a képek közt, majd amelyik kép megfelel az algoritmusnak, az a profile oldalba kerül, ahol a többi ajánlott kép található. Ugyanakkor ki is tud jelentkezni és azután a képek random lesznek megjelenve. Az ajánláshoz a bejelentkezés szükséges.

## 6. fejezet

### Célkitűzés

A személyre szabott tartalomajánló rendszerek fejlesztése során a projektem célja kerül a középpontba, amely meghatározza az irányt és a kívánt eredményeket. Az ilyen rendszer létrehozása olyan előnyöket biztosít, amelyek releváns és egyedi tartalmakhoz juttatják a felhasználókat. Ez jelentősen növeli az elégedettséget és használhatóságot, hiszen a felhasználóknak nem kell időt pazarolniuk a tartalmak keresésére vagy válogatására. Ehelyett az ajánlások alapján érdekes és őket érdeklő tartalmakhoz jutnak.

Az ilyen rendszerek további előnye, hogy lehetőséget adnak a felhasználóknak arra, hogy felfedezzenek új és izgalmas tartalmakat, amelyekre korábban nem is gondoltak volna. Ezen keresztül frissíthetik az online élményüket, és lehetőséget kapnak folyamatos tanulásra és felfedezésre.

Az ilyen rendszerek nemcsak a felhasználóknak jelentenek előnyt, hanem az üzleti szervezetek számára is. Az egyedi tartalom és az ajánlások segítségével hatékonyabban lehet célozni és elérni a célközönséget. Számos megközelítés és módszer létezik személyre szabott ajánlásra:

- **Tartalom alapú ajánlások:** ez a módszer a felhasználók korábbi tevékenységei alapján ajánlásokat készít, figyelembe véve a már korábbi fogyasztott tartalmakat és azok jellemzőit. Például, ha egy felhasználó szórakoztató képeket, cikkeket olvas, a rendszer hasonló témájú dolgokat ajánl.
- **Interakciós alapú ajánlások:** ez a módszer a felhasználók interakciói figyelembe véve ajánl. Figyeli a rendszer, hogy melyek azok a tevékenységek, interakciók, amelyeket a felhasználó végez és az alapján hasonló dolgokat oszt meg a felhasználóval, hogy még érdekesebb legyen a webböngészés.
- **Idő alapú ajánlások:** ez a módszer az időt használja fel az ajánláshoz. Figyelembe veszi a felhasználók legutóbbi tevékenységeit és a legfrissebb tartalmakat, és ennek alapján ajánlásokat készít.
- **Hely alapú ajánlások:** ez a módszer a felhasználók tartózkodási helyét használja fel az ajánlás megvalósítására. Vegyük például, ha egy felhasználó egy adott tartózkodási helyen tartózkodik a rendszer az ott lévő eseményeket, látnivalókat ajánlja.

## 6.1. Fontos tartalomajánló rendszer napjainkban: - [Netflix](#)

[5] Egy nagyon népszerű és fontos személyre szabott tartalomajánló rendszer, amit millióan használnak, az a jól ismert Netflix. A Netflix egy streaming szolgáltatás, ami személyre szabott filmeket, sorozatokat oszt meg a felhasználóval. A rendszer figyelembe veszi a felhasználó korábbi nézési szokásait, értékeléseiket és preferenciáit, majd ennek alapján ajánlja nekik az új tartalmakat.

[5] Rendszer sokféleképpen végzi el a személyreszabott tartalomajánlásokat. Az egyik megközelítés a tartalom alapú ajánlás, ahol a rendszer elemzi a felhasználó által korábban nézett tartalmak jellemzőit, melyek a stílusok, színészek, rendezők alapján ajánlja a tartalmakat.

[5] A másik tartalmi ajánlási módszer, az aszerint történik, hogy a felhasználók milyen stílusú filmeket néztek korábban. Milyen kategóriából való filmeket, sorozatokat.

[5] A Netflix egyre jobban figyeli a felhasználók viselkedési szokásait, és lehetővé teszi az egyre pontosabb és személyre szabottabb ajánlásokat.

Viszont ezek csak néhány példa a számos létező ajánlórendszer típusra. Az ajánlások generálása során gyakran alkalmazható több módszer kombinálása, hogy minél pontosabb és személyre szabottabb ajánlások szülessenek a felhasználók számára.

Ezek a rendszerek személyre szabott élményt nyújtanak, javítva az online élményt és növelve az elégedettséget. Ezeknek a rendszereknek köszönhetően a felhasználóknak nem kell időt pazarolniuk a keresésre vagy válogatásra. A rendszer automatikusan kiválasztja a számukra érdekes dolgokat.

Tehát összességében ez volt a célom, hogy a felhasználó szórakozva böngésszen az olyan képek között, amelyek a saját érdeklődései körébe tartozik.



# Összefoglaló

Végző sorban az általam írt dolgozat témája a személyre szabott tartalomajánlás weboldalon, amely szórakozás célját szolgálja. A weboldal különböző kategóriákba sorolja a képeket, és lehetővé teszi a felhasználóknak, hogy böngézhessenek és megtekinthessék ezeket a képeket.

A regisztráció és bejelentkezés után a felhasználók a képek megtekintése során eltöltött idő alapján kapnak személyre szabott ajánlásokat.

Az algoritmus szerint minden rákattintott kép felnagyított formában jelenik meg, és a rendszer figyeli az adott képen eltöltött időt, valamint a kép kategóriáját. Ha a felhasználó több mint öt másodpercet tölt el egy képen, akkor a rendszer rögzíti a kategóriát, és az ajánlás oldalon meg fognak jelenni a hasonló kategóriájú tartalmak. Ez lehetővé teszi a felhasználóknak, hogy könnyebben megtalálják és böngésszék azokat a tartalmakat, amelyeket leginkább tartanak érdekesnek.

Dolgozatom témájaként azért választottam ezt, mivel a személyre szabott tartalomajánlás továbbfejleszti a felhasználói élményt és lehetővé teszi a weboldal felhasználók jobb megismerését, megértését. Az eltöltött idő alapján a rendszer pontosabb képet kap a felhasználók érdeklődéseiről és ennek megfelelően tudja testre szabni a tartalmakat.

A felhasználók biztonsága érdekében a bejelentkezéshez a JWT hitelesítési mechanizmust használtam, míg a projekt frontend részéhez JavaScript és ReactJs keretrendszert. A backend fejlesztéshez és a rendszer üzemeltetéséhez a C# programozási nyelvet és az Entity Framework Core-t alkalmaztam.

# Könyvészet

1. <https://swagger.io/>
2. <https://jwt.io/>
3. <https://medium.com/@prashantramnyc>
4. <https://www.pexels.com/api/>
5. <https://help.netflix.com/hu/node/100639>
6. Matthew A. Russel, Mikhail Klassen: Mining the Social Web, 3rd Edition, 2019, O'Reilly Media

# Ábrák jegyzéke

2.1. Swagger UI felület megjelenése . . . . .	15
2.2. [2] Jason Web Token . . . . .	17
4.1. Weboldal Retro kategóriája alatt lévő képek az Apiból . . . . .	24
5.1. Use case Diagramm . . . . .	30