

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



**SAPIENTIA**  
ERDÉLYI MAGYAR  
TUDOMÁNYEGYETEM

Decentralizált üzenetküldés és adatmegosztás

**DIPLOMADOLGOZAT**

Témavezető:  
dr. Jánosi-Rancz Katalin-Tünde,  
Egyetemi adjunktus

Végzős hallgató:  
Pálffy Rolland

**2023**

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



UNIVERSITATEA  
SAPIENTIA

Mesagerie și partajarea datelor în mod descentralizat

**LUCRARE DE DIPLOMĂ**

Coordonator științific:  
dr. Jánosi-Rancz Katalin-Tünde,  
Lector universitar

Absolvent:  
Pálffy Rolland

**2023**

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



SAPIENTIA  
HUNGARIAN UNIVERSITY  
OF TRANSYLVANIA



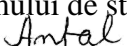

Decentralized messaging and data sharing

**BACHELOR THESIS**

Scientific advisor:  
dr. Jánosi-Rancz Katalin-Tünde,  
Lecturer

Student:  
Pálffy Rolland

**2023**

UNIVERSITATEA „SAPIENTIA” din CLUJ-NAPOCA Facultatea de Științe Tehnice și Umaniste din Târgu Mureș Programul de studii: Informatică		<b>Viza facultății:</b>
<b>LUCRARE DE DIPLOMĂ</b>		
Coordonator științific: <b>dr. János-Rancz Katalin-Tünde</b>	Candidat: <b>Pálffy Rolland</b> Anul absolvirii: 2023	
<p><b>a) Tema lucrării de licență:</b>          Proiectarea și implementarea unei aplicații pentru mesagerie și partajarea datelor în mod descentralizat</p> <p><b>b) Problemele principale tratate:</b>          Comunicare între clienți fără servere centrale, gestionarea datelor, criptarea și securitatea datelor</p> <p><b>c) Desene obligatorii:</b>          Diagramă de clasă, diagramă de stare, diagramă arhitecturală, comparații, interfață de utilizator</p> <p><b>d) Softuri obligatorii:</b>          Aplicație multiplatformă bazată pe tehnologiile React și React Native, scrisă în TypeScript, care conectează utilizatorii într-o rețea descentralizată. Aplicația permite utilizatorilor să trimită mesaje criptate și să partajeze date în mod privat, să comenteze liber pe fiecare site și să adauge conținut propriu prin extensii.</p> <p><b>e) Bibliografia recomandată:</b>          -Woko Ovunda, Asagba Prince Oghenekaro, Database Systems Model: Distributed, International Journal of Computer Science and Mathematical Theory E-ISSN 2545-5699 P-ISSN 2695-1924, Vol 7. No. 1 2021 <a href="http://www.iiardjournals.org">www.iiardjournals.org</a>          -Mark Nadal, Dr. Amber Cazzell, A Trustless Decentralized Bandwidth Incentive, <a href="http://web.stanford.edu/~nadal/A-Decentralized-Data-Synchronization-Protocol.pdf">http://web.stanford.edu/~nadal/A-Decentralized-Data-Synchronization-Protocol.pdf</a>          -Wyatt Lloyd, Michael J. Freedman, Michael Kaminsky, Intel Labs; David G. Andersen, Don't Settle for Eventual Consistency, Stronger properties for low-latency geo-replicated storage, <a href="https://queue.acm.org/detail.cfm?id=2610533">https://queue.acm.org/detail.cfm?id=2610533</a></p> <p><b>f) Termene obligatorii de consultații:</b> Studentul a început dezvoltarea aplicației în septembrie 2022 și ne-am întâlnit la fiecare 2-3 săptămâni</p> <p><b>g) Locul și durata practicii:</b> Universitatea „Sapientia” din Cluj-Napoca, Facultatea de Științe Tehnice și Umaniste din Târgu Mureș, sala / laboratorul 327A          Primit tema la data de: mai 2022          Termen de predare: 2 iulie 2023</p>		
Semnătura Director Departament 	Semnătura coordonatorului 	
Semnătura responsabilului programului de studiu 	Semnătura candidatului 	

### Declarație

Subsemnatul/a PÁLFFY ROLLAND, 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, TÂRGU MUREȘ  
Data: 16.06.2023

Absolvent

Semnătura.....Pálffy.....

# Kivonat

A decentralizált üzenetküldés és adatmegosztás ígéretes megközelítésként jelent meg a digitális korszakban a központosított rendszerek kihívásainak és korlátainak kezelésére. Ez a dolgozat a decentralizáció koncepcióját vizsgálja az üzenetküldés és az adatmegosztás összefüggésében, kiemelve annak előnyeit és lehetséges következményeit.

Az üzenetküldés és az adatmegosztás hagyományos megközelítése nagymértékben támaszkodik a központosított platformokra, ahol a felhasználók adatait egyetlen szervezet tárolja és ellenőrzi. Ez a központosított modell azonban korlátokkal jár, ideértve az adatvédelmi aggályokat, az adatszegéseket, a cenzúrát és az egyetlen hibaponttól való függőséget. Ezek a problémák egyre nagyobb érdeklődést váltanak ki a decentralizált alternatívák iránt.

A decentralizált üzenetküldő rendszerek egyenrangú hálózatokat és elosztott protokollokat használnak a felhasználók közötti közvetlen kommunikáció lehetővé tételére, kiküszöbölve a közvetítők szükségességét. Ez a megközelítés nagyobb adatvédelmet biztosít, mivel a felhasználók nagyobb ellenőrzést gyakorolhatnak adataik felett, és közvetlenül kommunikálhatnak anélkül, hogy központi szerverre támaszkodnának. Emellett a decentralizált üzenetküldő rendszerek ellenállnak a cenzúrázási kísérleteknek, mivel nincs központi hatóság, amely szabályozná vagy ellenőrizné az információáramlást.

Hasonlóképpen, a decentralizált adatmegosztás célja az adatokhoz való hozzáférés és azok ellenőrzésének demokratizálása. A blokklánc-technológia és az elosztott tárolórendszerek kihasználásával a felhasználók biztonságosan megoszthatják és tárolhatják az adatokat anélkül, hogy központi egységekre támaszkodnának. Ez nagyobb átláthatóságot, adatintegritást és a támadásokkal vagy adatvesztéssel szembeni ellenálló képességet tesz lehetővé. A decentralizált adatmegosztás továbbá ösztönzi az együttműködést és az innovációt, mivel a résztvevők bizalom- és engedély nélküli környezetben járulhatnak hozzá és férhetnek hozzá az adatokhoz.

Miközben a decentralizált üzenetküldés és adatmegosztás számos előnyt jelent, kihívásokkal is jár. A decentralizált rendszerek skálázhatósága, az adatok hitelességének ellenőrzése és a résztvevők ösztönzése a további kutatást és fejlesztést igénylő kulcsfontosságú területek közé tartozik. Emellett a jogi és szabályozási kereteket is hozzá kell igazítani e rendszerek decentralizált jellegéhez, biztosítva a magánélet védelmét és az elszámoltathatóságot.

A dolgozatom célja egy decentralizált rendszeren alapuló alkalmazás bemutatása és az elkészítéséhez szükséges technológiák megismertetése az olvasóval.

# Rezumat

Mesageria descentralizată și schimbul de date au apărut ca abordări promițătoare pentru a aborda provocările și limitările sistemelor centralizate în era digitală. Acest teză explorează conceptul de descentralizare în contextul mesageriei și al partajării datelor, evidențiind avantajele și implicațiile sale potențiale.

Abordarea tradițională a mesageriei și a partajării datelor se bazează în mare măsură pe platforme centralizate, în care informațiile utilizatorilor sunt stocate și controlate de o singură entitate. Cu toate acestea, acest model centralizat are limitări inerente, inclusiv preocupări legate de confidențialitate, încălcări ale datelor, cenzură și dependența de un singur punct de eșec. Aceste probleme au dus la creșterea interesului pentru alternativele descentralizate.

Sistemele de mesagerie descentralizate utilizează rețele peer-to-peer și protocoale distribuite pentru a permite comunicarea directă între utilizatori, eliminând nevoia de intermediari. Această abordare asigură o mai mare confidențialitate, deoarece utilizatorii au un control mai mare asupra datelor lor și pot comunica direct, fără a se baza pe un server central. În plus, sistemele de mesagerie descentralizate pot rezista încercărilor de cenzură, deoarece nu există o autoritate centrală care să reglementeze sau să controleze fluxul de informații.

În mod similar, partajarea descentralizată a datelor urmărește să democratizeze accesul și controlul datelor. Prin utilizarea tehnologiei blockchain și a sistemelor de stocare distribuite, utilizatorii pot partaja și stoca date în siguranță, fără a se baza pe entități centralizate. Acest lucru permite o mai mare transparență, integritate a datelor și reziliență împotriva atacurilor sau a pierderilor de date. În plus, partajarea descentralizată a datelor stimulează colaborarea și inovarea, deoarece participanții pot contribui și accesa datele într-un mediu lipsit de încredere și fără permisiuni.

În timp ce mesageria descentralizată și partajarea datelor oferă numeroase beneficii, acestea prezintă, de asemenea, provocări. Scalabilitatea sistemelor descentralizate, verificarea autenticității datelor și stimularea participanților se numără printre domeniile cheie care necesită cercetări și dezvoltări suplimentare. În plus, cadrele juridice și de reglementare trebuie să se adapteze pentru a se potrivi naturii descentralizate a acestor sisteme, asigurând atât protecția vieții private, cât și responsabilitatea.

Scopul acestei teze este de a prezenta o aplicație bazată pe un sistem descentralizat și de a introduce cititorul în tehnologiile necesare pentru a construi această aplicație.

# Abstract

Decentralized messaging and data sharing have emerged as promising approaches to address the challenges and limitations of centralized systems in the digital era. This paper explores the concept of decentralization in the context of messaging and data sharing, highlighting its advantages and potential implications.

The traditional approach to messaging and data sharing relies heavily on centralized platforms, where users' information is stored and controlled by a single entity. However, this centralized model has inherent limitations, including privacy concerns, data breaches, censorship, and dependency on a single point of failure. These issues have led to growing interest in decentralized alternatives.

Decentralized messaging systems utilize peer-to-peer networks and distributed protocols to enable direct communication between users, eliminating the need for intermediaries. This approach ensures greater privacy, as users have more control over their data and can communicate directly without relying on a central server. Additionally, decentralized messaging systems can resist censorship attempts, as there is no central authority to regulate or control the flow of information.

Similarly, decentralized data sharing aims to democratize the access and control of data. By leveraging blockchain technology and distributed storage systems, users can securely share and store data without relying on centralized entities. This enables greater transparency, data integrity, and resilience against attacks or data loss. Moreover, decentralized data sharing incentivizes collaboration and innovation, as participants can contribute and access data in a trustless and permissionless environment.

While decentralized messaging and data sharing offer numerous benefits, they also present challenges. The scalability of decentralized systems, the verification of data authenticity, and the incentivization of participants are among the key areas that require further research and development. Additionally, legal and regulatory frameworks need to adapt to accommodate the decentralized nature of these systems, ensuring both privacy protection and accountability.

The aim of this thesis is to present an application based on a decentralised system and to introduce the reader to the technologies needed to build it.



# Tartalomjegyzék

<b>1. Bevezető</b>	<b>11</b>
<b>2. Célkitűzések</b>	<b>13</b>
<b>3. Elméleti megalapozás</b>	<b>14</b>
3.1. GUN	14
3.2. A GUN történelme	15
3.3. CAP	16
3.4. DAM	16
3.5. CRTD	16
3.6. Titkosítás és biztonság	17
3.7. Gráf adatstruktúra	18
3.8. Hitelesítés	18
<b>4. Programok, technológiák bemutatása</b>	<b>19</b>
4.1. React	19
4.2. Tailwind CSS	20
4.3. TypeScript	21
4.4. Vite	21
4.5. Prettier	22
4.6. ESLint	22
4.7. npm és pnpm	23
<b>5. Gyakorlati megvalósítás</b>	<b>24</b>
5.1. Projekt létrehozása és konfigurálása	24
5.2. GUN inicializálása	24
5.3. Felhasználói felület inicializálása	25
5.4. Bejelentkezési oldal	27
5.5. Főoldal	27
5.5.1. Oldalmenü	28
5.5.2. Chat	30
5.6. Felhasználói felület megjelenítése	31
<b>6. Felhasználói útmutatók</b>	<b>33</b>
6.1. Nem funkcionális követelmények	33
6.2. Az alkalmazás menete	33
6.2.1. Bejelentkezési folyamat	33

6.2.2. Beszélgetés . . . . .	34
6.2.3. Nyilvános beszélgetések és felhasználók keresése . . . . .	34
6.2.4. Kiegészítők . . . . .	35
6.2.5. Kijelentkezés . . . . .	36
<b>7. Továbbfejlesztési lehetőségek</b>	<b>37</b>
<b>Összefoglaló</b>	<b>38</b>
<b>Ábrák jegyzéke</b>	<b>39</b>
<b>Irodalomjegyzék</b>	<b>40</b>

# 1. fejezet

## Bevezető

Az információs technológia robbanásszerű fejlődése jelentős változásokat hozott a kommunikáció és az adatmegosztás terén. A hagyományos központosított rendszerek hatékonysága és megbízhatósága ellenére számos korlátozást és kihívást rejtenek magukban. A központosított platformokra való túlzott függés, a magánszféra megsértése, az adatbiztonsági problémák és a cenzúra veszélye olyan problémákat vet fel, amelyekre alternatív megoldásokra van szükség. Ebben a kontextusban az utóbbi években egyre növekvő érdeklődés övezte a decentralizált üzenetküldés és adatmegosztás koncepcióját.

Ez a dolgozat célja, hogy részletesen bemutassa a decentralizált üzenetküldés és adatmegosztás elméleti és gyakorlati alapjait, valamint rávilágítson a decentralizáció előnyeire és lehetőségeire. A decentralizált rendszerek a felhasználók közvetlen kapcsolatán és egyenrangúságán alapulnak, lehetővé téve a közvetlen kommunikációt és az adatmegosztást anélkül, hogy közbeiktatnának egy központi entitást.

A decentralizált megközelítés megnöveli a felhasználók adatvédelmét, hiszen közvetlen irányítást kapnak adataik felett, és képesek közvetlenül kommunikálni egymással anélkül, hogy egy központi szerverre támaszkodnának. Emellett a decentralizált üzenetküldési rendszerek ellenállóbbak a cenzúrával szemben, hiszen nincs központi hatóság, amely szabályozná vagy ellenőrizné az információáramlást. [OO21]

A decentralizált adatmegosztás célja a hozzáférés és az adatok irányításának demokratizálása. Az elosztott tárolórendszerek felhasználásával a felhasználók biztonságosan oszthatják meg és tárolhatják az adatokat anélkül, hogy központi entitásokra támaszkodnának. Ez növeli az átláthatóságot, az adatintegritást és a támadásokkal vagy adatvesztéssel szembeni ellenálló képességet. Ezenkívül a decentralizált adatmegosztás ösztönzi a együttműködést és az innovációt, hiszen a résztvevők hozzájárulhatnak és hozzáférhetnek az adatokhoz egy bizalmatlan és engedélyezés nélküli környezetben.

A dolgozat továbbá részletesen bemutatja egy decentralizált rendszer működését. A rendszer lehetővé teszi az adatok küldését, tárolását és titkosítását a felhasználók között, kiküszöbölve a központi szerverre való támaszkodást. Az alkalmazott technológiák között szerepelnek a peer-to-peer hálózatok, elosztott tárolási rendszerek és kriptográfiai eljárások. Az applikáció felépítése és az implementált funkcionalitás részletes bemutatásával is foglalkozom, hogy megértsük, hogyan valósítható meg a decentralizált üzenetküldés és adatmegosztás a gyakorlatban.

A dolgozat végén összefoglalom a decentralizált üzenetküldés és adatmegosztás potenciálját, és rámutatok a kihívásokra és további kutatási irányokra. Ahhoz, hogy a de-

centralizáció teljes potenciálját kiaknázzuk, további kutatásra, technológiai fejlesztésekre és egy olyan szabályozói környezetre van szükség, amely egyaránt biztosítja a magánszférát és a felelősségre vonhatóságot.

## 2. fejezet

# Célkitűzések

A dolgozat célja egy olyan multiplatform alkalmazás fejlesztése és bemutatása, amely lehetővé teszi az emberek számára a szabad véleménynyilvánítást egy független és decentralizált platformon keresztül. Az alkalmazás kifejlesztéséhez a GUN ökoszisztémát használok fel, amely egy nyílt forráskódú, decentralizált adatbázis és üzenetküldési rendszer.

Az alábbiakban felsorolom a dolgozat konkrét célkitűzéseit:

- A GUN ökoszisztéma megismerése: Áttekintem a GUN decentralizált adatbázis és üzenetküldési rendszerét, megértem a működését, és megismerem annak API-ját és funkcionalitását.
- A felhasznált technológiák felsorolása és bemutatása.
- Az alkalmazás részletes tervezése: Először is megtervezem az alkalmazás funkcionális és felhasználói követelményeit. Kifejlesztmem a felhasználói interfészt és meghatározom az alkalmazás felépítését és architektúráját.
- Az alkalmazás implementálása: Fejlesztmem az alkalmazást a különböző platformokra (web, böngésző kiegészítő, natív alkalmazások). Az alkalmazás lehetővé teszi a felhasználók számára, hogy üzeneteket küldjenek és adatokat osszanak meg a GUN rendszeren keresztül.
- Az alkalmazás éles környezetben való tesztelése: A kész alkalmazást tesztelni automatikus tesztrendszerek segítségével tesztelni fogom, hogy felfedezzem az esetleges hibákat.
- Továbbfejlesztési lehetőségek felsorolása.

A fent említett célkitűzések teljesítése segít abban, hogy egy olyan decentralizált üzenetküldésre és adatmegosztásra képes alkalmazást fejlesszek, amely lehetőséget ad az embereknek a szabad véleménynyilvánításra egy független platformon, és példaként szolgáljon hasonló alkalmazások fejlesztésére, illetve népszerűsítse a decentralizációt.

## 3. fejezet

# Elméleti megalapozás

### 3.1. GUN

A GUN decentralizált gráf alapú adatbázis egy rugalmas és skálázható rendszer, amely lehetővé teszi a felhasználók számára az adatok tárolását és megosztását a decentralizáció elvei alapján. Ez a technológia alapvető fontosságú az általam fejlesztett alkalmazás szempontjából, mivel lehetővé teszi a felhasználóknak, hogy közvetlenül kommunikáljanak egymással és megosszák az információkat a hagyományos központi szerverekre való támaszkodás nélkül.

A GUN adatbázis a gráf adatstruktúrát használja, amelyben a különböző entitások (pl. felhasználók, üzenetek stb.) és az azok közötti kapcsolatok úgy vannak tárolva, hogy a felhasználók könnyen navigálhassanak a kapcsolatok között és lekérdezéseket hajthassanak végre. Ez egy különösen hatékony megközelítés olyan alkalmazások esetében, ahol fontos a kapcsolati hálózatok, például közösségi média vagy csoportos üzenetküldő alkalmazások.

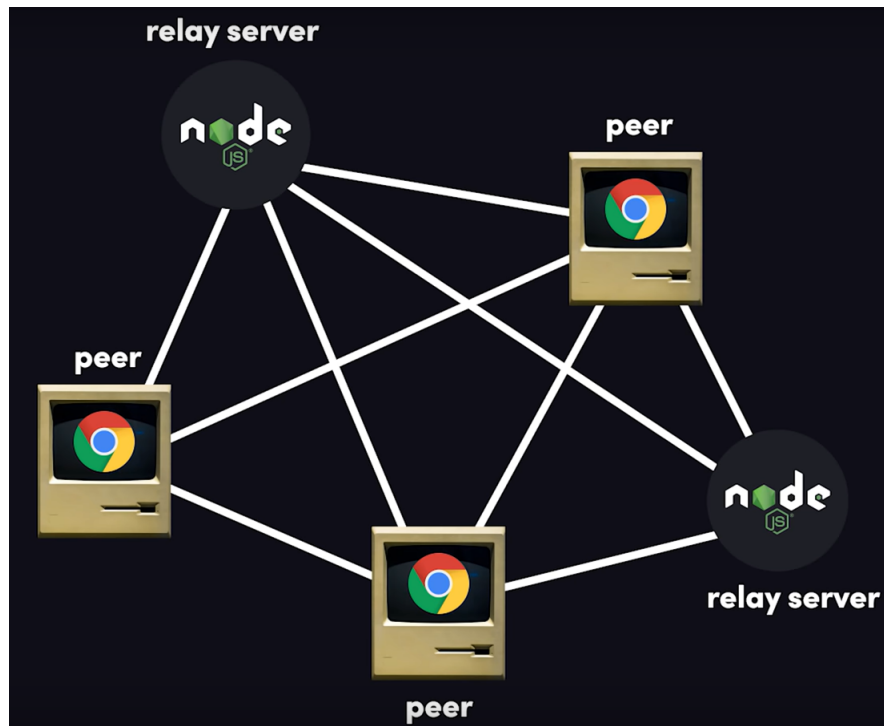
Az adatok tárolása a GUN-ban elosztott módon történik. A felhasználók rendelkeznek egyéni adatbázis példányokkal, amelyek egy peer-to-peer hálózaton keresztül kommunikálnak egymással. Ez azt jelenti, hogy minden felhasználó saját adatbázis-üzemeltetővé válik, és közvetlenül cserélhet információkat más felhasználókkal, anélkül, hogy központi szerverekre támaszkodna.

A GUN adatbázis számos előnyt kínál. Először is, a decentralizált természete miatt megnövekedett adatvédelmet biztosít. Mivel az adatok nem tárolódnak központi szervereken, a felhasználóknak több ellenőrzése és biztonsága van adataik fölött. Emellett a decentralizált adatbázis rendszer rugalmasan skálázható, mivel a felhasználók adatbázisai közvetlenül kommunikálnak egymással, így a terhelés megoszlik a hálózatban.

A GUN adatbázis támogatja a szinkron és aszinkron adatfrissítéseket is. Ez azt jelenti, hogy a felhasználók valós időben láthatják és frissíthetik az adatokat, lehetőséget biztosítva a gyors és interaktív kommunikációra. Emellett a GUN adatbázis API-ja egyszerű és jól dokumentált, ami megkönnyíti az alkalmazás fejlesztését és integrálását.

Az általam fejlesztett alkalmazásban a GUN adatbázist használom a felhasználók közötti üzenetek és adatok tárolására és megosztására. Az adatokat gráf struktúrában tárolom, ami lehetővé teszi a könnyű navigációt és lekérdezéseket a kapcsolatok között. A GUN decentralizált adatbázis segítségével biztosítom a felhasználók számára a ma-

gánszférát, az adatvédelmet és a közvetlen kommunikációt egymással a hagyományos központi szerverek nélkül. [NC] [WLA]



3.1. ábra. A GUN rendszere

### 3.2. A GUN történelme

A GUN-t Mark Nadal hozta létre 2014-ben, miután 4 éven át próbálta elérni, hogy kollaboratív webes alkalmazása hagyományos adatbázisok segítségével skálázható legyen.

Miután rájött, hogy a mester-szolga adatbázis-architektúra áthidalhatatlan akadályokkal rendelkezik, úgy döntött, hogy megkérdőjelezi az addig létező adatbázisok alapjait és saját architektúrát hoz létre.

A NoDB - nincs mester, nincsenek szerverek, nincs "egyetlen igazságforrás", nem valódi programozási nyelvvel vagy valódi hardverrel épül, nincs DevOps, nincs zárás, nem csak SQL vagy NoSQL, hanem mindkettő (minden - gráfok, dokumentumok, táblák, kulcs/érték).

A cél egy olyan P2P (olyan informatikai hálózat, amelynek végpontjai közvetlenül egymással kommunikálnak) adatbázis létrehozása volt, amely bármilyen böngészőben futhat, és bármilyen offline tevékenység után helyesen szinkronizálja az adatokat bármely eszköz között.

A továbbiakban bemutatom a GUN adatbázis létrehozásához felhasznált protokollokat

### 3.3. CAP

A CAP rövidítés a "Consistency, Availability, Partition tolerance" kifejezésre utal, azaz konzisztencia, rendelkezésre állás és partíciótürés.

A GUN alapértelmezetten egy AP (Availability, Partition tolerance) rendszer, ami azt jelenti, hogy képes adatot írni és olvasni, még akkor is, ha más csomópontok nem érhetőek el. Ez azonban a szigorú konzisztencia rovására történik, mivel az adatok idővel konvergálnak, és csak időbeli koordinációval érhetőek el minden csomóponthoz.

A GUN egy eseményi konzisztenciát valósít meg, amely azt jelenti, hogy két felhasználó módosíthatja ugyanazt az adatot a világ két különböző pontján ugyanabban az időben, de nem tudnak egymás konfliktusos frissítéseiről, amíg az információ lassan terjed az hálózaton.

Az erősen konzisztens működéshez a GUN egy üzenetküldő rendszert használ, amely általában eseményi konzisztencia alapú. Az adatot módosítani kívánó csomópontok "zár" kérést küldenek a többi csomóponthoz, amelyek elfogadják vagy elutasítják a zárat. Az elfogadó csomópontok nem engedik a helyi felhasználóknak az adat olvasását vagy írását, ami magas késleltetést eredményez. Ha az eredeti csomópont minden elfogadást megkapott, akkor írja az adatot, majd várja meg a siker visszajelzést a többi csomóponttól.

### 3.4. DAM

DAM rövidítése a "Daisy-chain Ad-hoc Mesh-network" kifejezésre utal, ami a GUN alapértelmezett szállítási rétegét és P2P hálózati algoritmusát jelenti. A decentralizált kommunikáció gyakran intenzív, ezért a DAM segít mérsékelni az üzenetek számát.

Az API segítségével a DAM lehetővé teszi különböző szállítási rétegek (websocket, WebRTC stb.) cseréjét, hasonlóan ahogyan a RAD lehetővé teszi a tárolási rétegek (lemez, S3, IPFS stb.) cseréjét.

Az API nagy része a szomszédos csomópontokkal történő egyedi üzenetek kommunikálásával foglalkozik annak érdekében, hogy koordinálja az azonnali hatékonyságot. Ha az egyenesen csatlakoztatott csomópontok optimalizálva vannak, akkor az eredmény az egész hálózat hatékonyabbá válik.

Ha megtörtént a kérés a csatlakozásra, még nem jelenti azt, hogy a csomópont kötelezően kapcsolódik. Az elérés visszaélésének megakadályozása érdekében a kért csomópont csak elsőbbségi alapon csatlakozhat legfeljebb hat másik csomóponthoz, és gyorsan lecsatlakozhat a többi csomóponttól, ha hosszabb ideig offline maradnak, és nem próbál újra kapcsolódni, amíg újra nem kéri.

### 3.5. CRTD

A CRDT (Conflict-free Replicated Data Type) olyan algoritmus, amely a GUN rendszerének középpontjában áll. Ez az algoritmus biztosítja, hogy a csomópontok végül elérjék ugyanazt az állapotot, és kezeli az offline szerkesztéseket. A HAM (History-Aware Merkle) eljárásra épül, amely a konfliktusok feloldásáért felelős.

A HAM működésének követelményei a következők:



- Offline elsőlegesség: A rendszer magas rendelkezésre állást biztosít a szigorú konzisztencia helyett, lehetővé téve a felhasználóknak a szerkesztéseket még akkor is, ha az eszközük offline állapotban van.
- Sorrend: Az állapotot az érkező frissítések sorrendjétől függetlenül ugyanannak kell elérnie (kommutativitás).
- Konfliktuskezelés: Konfliktus esetén minden csomópontnak függetlenül kell ugyanazt az értéket választania (Erős Esetleges Konzisztencia).

Az implementáció szempontjából a frissítések gráf formátumban érkeznek, mivel a gun adatstruktúrája gráf-orientált. A kulcs-érték párok egyesítése a HAM szempontjából atomi folyamat, ahol a kulcs-érték párok egyszerűen össze vannak fésülve a megfelelő szabályok alapján. A frissítések sorrendjét a "state" metadat segítségével határozzák meg.

A konfliktusok kezelésekor az alábbi szabályokat alkalmazzák, amelyek biztosítják a konvergenciát:

- Ha két érték lexikálisan egyezik, akkor nincs konfliktus, és bármelyiket választhatjuk.
- Ha két érték lexikálisan különbözik, akkor a string értékeiket összehasonlítjuk a `JSON.stringify` segítségével, és a nagyobb értéket választjuk.

A "state" (állapot) értékek fontosak a rendszerben, és a megfelelő kezelésük nélkül súlyos sérülékenységet eredményezhetnek. A HAM egy relatív vektor segítségével kezeli ezeket az állapotokat, amelyeket a gépekhez viszonyítva határoznak meg. Az új frissítések csak akkor kerülnek feldolgozásra, ha a gépek elérnek egy adott állapotot, és a korábbi frissítések megfelelő sorrendben történnek.

A frissítések összeolvasztása gráfok esetén az egyes mezők iteratív összehasonlításával történik. Ha egy frissítésben egy mező létezik, amely nem szerepel az eredeti objektumban, akkor az eredeti mező állapotát  $-\infty$ -nek tekintik, vagyis mindig hozzáadják az új mezőt.

Fontos megjegyezni, hogy a HAM nem garantálja a többfolyamatú linearizálhatóságot, mivel a rendelkezésre álló rendszerekben nem lehet tudni, hogy mikor fejeződnek be az összes frissítés hálózati terjedése. Ehelyett az Erős Esetleges Konzisztenciát (SEC) garantálja.

### 3.6. Titkosítás és biztonság

A kriptográfia és a biztonság alapjai nagyon fontosak az alkalmazások védelme szempontjából. Az adatbiztonságot két fő módon lehet megvalósítani: hamis és valódi biztonsággal. A legtöbb weboldal hamis biztonságot használ, ahol a felhasználói jelszavak a szolgáltató szerverére kerülnek elküldésre, amely fennáll a kockázata annak, hogy illetéktelenek hozzáférjenek az adatokhoz. Valódi biztonságot jelent a jelszavak helyben történő kezelése és a bizonyíték alapú munkafolyamat alkalmazása.

A kriptográfiával és a titkosítással lehetőség van felhasználói fiókok létrehozására, privát adatok tárolására, tartalmak közzétételére, privát üzenetek küldésére és csoportos beszélgetésekre. A köz-/privát kulcs titkosítás és a bizonyíték alapú munkafolyamat

(PBKDF2) használata a biztonság és a felhasználói fiókokhoz való bejelentkezés biztosítása érdekében kulcsfontosságú.

Az adatok védelméhez a GUN az Advanced Encryption Standard (AES) titkosítást alkalmazza, amely ma az egyik legelterjedtebb és legbiztonságosabb módszernek számít biztonságos adatátvitelhez és tároláshoz. Az AES egy szimmetrikus kulcsú titkosítás, ami azt jelenti, hogy az adatok titkosításához és visszafejtéséhez ugyanazt a kulcsot használják. A publikus és privát kulcsok lehetővé teszik a privát kommunikációt és a csoportos beszélgetéseket, a digitális aláírások használata pedig lehetővé teszi az üzenetek eredetiségének ellenőrzését és az adott személy azonosítását.

### 3.7. Gráf adatstruktúra

A gráf adatszerkezet kiemelkedik a különböző adatbázis-szerkezetek közül, mint például dokumentumok, fa, táblázat, relációs adatbázis stb. A gráfban az objektumok nem foglalnak magukba más objektumokat, hanem csak hivatkozásokat tartalmaznak azokra. A GUN rendszerben ezeket a hivatkozásokat "souls" (lelkek) néven említik. A souls-okat az objektumokban a hash szimbólummal ellátott objektumokba csomagolják. A gráf képes bármilyen adatszerkezetet alkotni, ezáltal rendkívül rugalmas és hatékony.

### 3.8. Hitelesítés

A hitelesítéssel kapcsolatban néhány alapvető kérdés merül fel. Először is, hogyan lehet megakadályozni, hogy valaki más személynek tűnjön fel? Ezen kívül, hogyan osszunk meg érzékeny adatokat egy csoporttal, és hogyan kerüljük el az alkalmazás spammingjét és a DOS (Denial of Service) támadásokat.

A kulcs fogalmak közé tartozik az aszimmetrikus vagy publikus-kulcsú titkosítás. A publikus kulcs lehetővé teszi, hogy bárki képes legyen információt titkosítani, de csak a privát kulcs használható a visszafejtésre. A digitális aláírás pedig lehetővé teszi annak az ellenőrzését, hogy egy üzenet valóban egy adott felhasználótól származik.

A hitelesítés megvalósításához létrehozunk egy kulcspárt mindenkinek, akik részt vesznek a rendszerben. A nyilvános kulcsokat közzétesszük, míg a privát kulcsokat titokban tartjuk. Az üzeneteket digitális aláírással látjuk el, hogy ellenőrizhető legyen az eredetük és valódiságuk. Az érzékeny adatokat pedig aszimmetrikus titkosítással védjük, hogy csak az illetékes személyek olvashassák azokat.

Ha egy csoporttal szeretnénk megosztani az adatokat, létrehozunk egy projekt-specifikus kulcspárt, amelyet a csoport tagjai használnak az üzenetek titkosításához és visszafejtéséhez. Az új tagokat hozzáadjuk a kulcstárba, hogy hozzáférjenek a projekt adatokhoz. Ha valakinek a hozzáférését vissza szeretnénk vonni, új kulcspárt hozunk létre és újra titkosítjuk az adatokat az új kulcsokkal.

Az autentikáció és adatvédelem a GUN rendszerben számos fontos problémát old meg, és biztosítja, hogy csak az illetékes személyek férhessenek hozzá az adatokhoz, és hogy az üzenetek valódisága ellenőrizhető legyen.

## 4. fejezet

# Programok, technológiák bemutatása

### 4.1. React

Az alkalmazásom UI (felhasználói felület) tervezéséhez a React és a React Native könyvtárakat használtam. A React egy JavaScript könyvtár, amely lehetővé teszi a hatékony és újrafelhasználható felhasználói interfész komponensek fejlesztését. A React Native pedig egy keretrendszer, amely lehetővé teszi a natív mobilalkalmazások fejlesztését a Reactet használva.

A React alapját a funkcionális komponensek képezik. A funkcionális komponensek egyszerű JavaScript függvények, amelyek JSX (JavaScript Syntax Extension) elemet térítenek vissza. A JSX a JS szintaxisának egy kiegészítése, amely a HTML-hez hasonlóan leírja a komponens kinézetét és működését.

Az adatkezelés szempontjából a Reactben a szülő komponens adatokat adhat át a gyerekek komponenseknek props (tulajdonságok) objektumon keresztül. Ez lehetővé teszi a hierarchikus adatáramlást a komponensfában, ahol a szülő komponens adatai lefelé áramlanak a gyerekek komponensek felé.

A React egy sor beépített hookot biztosít, amelyek lehetővé teszik az állapot kezelését, az életciklus események figyelését és más funkciók hozzáadását a komponensekhez. A hookok kötelezően "use" előtaggal vannak ellátva, így tudja a React megkülönböztetni a többi függvénytől és speciális módon kezelni ezeket. Az alkalmazás megírásához a következőket használtam:

- A useState hook lehetővé teszi az állapot kezelését funkcionális komponensekben. Ezzel a hookkal létrehozhatunk és frissíthetünk állapotokat a komponensen belül változók formájában. Amikor az állapot változik, a React automatikusan újrarajzolja a komponenst, hogy megjelenítse az új adatokat.
- A useReducer hook hasonló módon működik, mint az useState, de alkalmasabb bonyolultabb állapotkezelésre. A useReducer egy "reducer" alapú állapotkezelést implementál, ahol a műveleteket (action) egy adott állapotra alkalmazzuk, és az új állapotot visszatérítjük. Ez különösen hasznos, ha az állapot változásait több lépésben vagy összetett módon kell kezelni.

- A `useContext` hook lehetővé teszi az adatok globális elérését a komponensfában anélkül, hogy azokat a teljes alkalmazáson keresztül továbbítanunk kéne az adott gyerek komponenseken, mivel Reactben csak szülő-gyerek irányba lehet adatot átadni. Ez azáltal valósul meg, hogy egy központi kontextust hozunk létre, amelyben az adatokat tároljuk, majd a komponensek egyszerűen hozzáférhetnek ehhez a kontextushoz és az ott tárolt adatokhoz. A `useContext` hookot ötvözhetjük más hookokkal is, ha a gyerek komponensen belül szeretnénk lentről felfelé propagálni egy globális adat változtatását egy függvény meghívásával.
- A `useEffect` hook egy függvényt vár, amely lehetőséget nyújt a funkcionális komponensekben az életciklus események figyelésére és az ezekhez kapcsolódó műveletek végrehajtására. Ez a React régi verzióiban használt osztály alapú komponensek metódusait hivatott leváltani, mint a `componentDidMount`, `componentDidUpdate` és `componentWillUnmount`, amelyeket az életciklus események kezelésére szolgáltak. A `useEffect` hook kombinálja ezeket a funkcionalitásokat egyetlen függvényben. Az `useEffect` hooknak egy olyan második tömb paramétere is van, amiben megadhatjuk, hogy mely értékek változására reagáljon, vagy ha üres tömböt adunk át, akkor csak egyszer fusson le, amikor létrejön a komponens. Visszatérítési értéként megadhatunk egy újabb függvényt, amelyet akkor futtat le, ha újrarajzolódik vagy eltűnik a komponens.

Az alkalmazásom multiplatform lehetőségei a React Native használatával valósulnak meg. A React Native lehetővé teszi, hogy ugyanazt a kódot használjuk az Android és az iOS platformokon is, így csökkentve a fejlesztési időt és erőforrásokat. A React Native alkalmazások natívan futnak a készüléken, így a felhasználók hasonló teljesítményt és élményt tapasztalnak, mint az eredeti natív alkalmazások esetében.

A React és a React Native népszerűsége folyamatosan nő, mivel könnyű tanulni és fejleszteni velük, és széles körű közösség és dokumentáció áll rendelkezésre. Ezáltal a React és a React Native a modern alkalmazásfejlesztés egyik meghatározó technológiájává vált.

## 4.2. Tailwind CSS

A komponensek designjának kialakításához a Tailwind CSS frameworköt használtam. A Tailwind CSS egy utility-first CSS keretrendszer, amely lehetővé teszi a gyors és hatékony design készítést. Az alapelve az, hogy a CSS osztályokat közvetlenül használjuk a HTML kódban, anélkül hogy saját stíluslapokat kellene írunk.

A Tailwind CSS számos előnnyel rendelkezik a hagyományos CSS-hez képest. Először is, a Tailwind CSS egy átfogó osztálykönyvtárat biztosít, amely tartalmazza a leggyakrabban használt design elemeket, például betűtípusokat, színeket, távolságokat és elrendezéseket. Ez jelentősen lerövidíti a fejlesztési időt, mivel nem kell minden stílust saját magunknak definiálni. Ehelyett egyszerűen hozzáadhatjuk a megfelelő osztályokat a HTML elemekhez.

A Tailwind CSS más megközelítést alkalmaz, mint más CSS keretrendszerek, például a Bootstrap. Míg a Bootstrap előre meghatározott komponenseket és stílusokat kínál, a Tailwind CSS inkább az alacsony szintű építőköveket adja meg. Ez azt jelenti, hogy az elemek stílusát közvetlenül a HTML kódban definiáljuk az osztályokkal. Ez nagyobb

rugalmasságot és testreszabhatóságot biztosít, mivel minden elemet egyedi módon formázhatunk.

A Tailwind CSS segíti elő egy jó design létrehozását, mert egy jól strukturált osztályrendszerrel rendelkezik, amely logikus és konzisztens. Az osztályok intuitívak és könnyen érthetőek, így egyszerűen átláthatjuk, hogy milyen stílusokat alkalmaznak az elemekre. Emellett a Tailwind CSS lehetővé teszi az egységes megjelenés fenntartását az egész projektben, mivel minden fejlesztő azonos osztályokat használ.

A Tailwind CSS további előnye, hogy lehetővé teszi a könnyű testreszabást és a rezponzív design kialakítását, amely segítségével a weboldalak és alkalmazások szépen adaptálódnak a különböző eszközökön és kijelzőméretekre.

### 4.3. TypeScript

TypeScript egy Microsoft által fejlesztett programozási nyelv, amely a JavaScript szintaktikáját és funkcionalitását hivatott kiterjeszteni. A JavaScript kódhoz típusellenőrzést és statikus analízist ad hozzá, ezáltal növelve a fejlesztés sebességét és a kód minőségét.

A TypeScript fő előnyei a JavaScripthez képest:

- **Típusbiztonság:** A TypeScript erőteljes típusrendszerrel rendelkezik, ami lehetővé teszi a hibák felderítését a fejlesztési időben, így megelőzve a futási időben bekövetkező hibákat és hibás működést.
- **Jobb fejlesztői élmény:** A TypeScript erős statikus analízist és fejlesztői eszközöket nyújt, például intelligens kódkiegészítést, automatikus refaktorálást és hibakeresést. Ez javítja a fejlesztői produktivitását, illetve segít gyorsabban és hatékonyabban fejleszteni.
- **Kódolási segítség:** A TypeScript lehetővé teszi az objektumorientált programozási paradigmák, például az osztályok és interfészek használatát. Ez lehetővé teszi a kód jobb strukturálását, újrafelhasználhatóságát és karbantarthatóságát.
- **JavaScript-kompatibilitás:** A TypeScript a JavaScript kiterjesztése, tehát minden érvényes JavaScript kód TypeScript kódként is futtatható. Ez azt jelenti, hogy egy meglévő JavaScript projektet fokozatosan át tudunk alakítani Typescriptre, és használhatjuk a TypeScript funkcionalitását anélkül, hogy minden meglévő kódot meg kellene változtatni.
- **Nagy közösség és támogatás:** A TypeScript népszerűvé vált a fejlesztők körében, és egyre több nagy projektben használják. Ennek eredményeként sok dokumentáció, cikk, könyv és online erőforrás áll rendelkezésre. Emellett számos fejlesztői eszköz és keretrendszer, köztük a React is támogatja a Typescriptet.

### 4.4. Vite

A Vite egy modern fejlesztői eszköz és környezet a webalkalmazások számára. Eredetileg a Vue.js-hez készült, de a React keretrendszert is támogatja. A fő előnye, hogy gyors és hatékony fejlesztési élményt biztosít.

A Vite lehetővé teszi a fejlesztés közbeni gyors visszajelzést, mert alacsony indítási idővel rendelkezik, és a fejlesztői környezet és a böngésző azonnal válaszol a kódmódosításokra.

Támogatja a natív ES modulokat, ami azt jelenti, hogy a kódot kisebb modulokra bonthatjuk, és csak a szükséges modulokat kell letöltenie a böngészőnek, amikor azokra szükség van. Ez növeli az alkalmazás teljesítményét és csökkenti a letöltési időt.

A Vite rendelkezik egy beépített fejlesztői szerverrel, amely közvetlenül futtatja a kódot a fejlesztői környezetben. Ez lehetővé teszi a moduláris fejlesztést és azonnali frissítéseket a böngészőben anélkül, hogy újra kellene építeni az egész alkalmazást.

A Vite lehetőséget nyújt bővítmények használatára, amelyek kiterjesztik a funkcionalitását. Ez segít testreszabni a fejlesztői környezetet, valamint különböző optimalizációkat végezni, például tömörítheti a megírt kódot.

## 4.5. Prettier

A Prettier egy nyílt forráskódú kódformázó, amely segít egységes formázást elérni a projekteken. Célja, hogy automatikusan meghatározza és alkalmazza a kód formázási szabályokat, így elkerülhetők a felesleges viták a stílusról és az egységesített kódolási konvenciók használata.

A Prettier az egyszerű konfiguráció és az alapértelmezett formázási szabályok miatt kényelmesen használható. Az eszköz több programozási nyelvet támogat, beleértve a JavaScriptet, a TypeScriptet, illetve a leíró nyelveket is, a CSS és a HTML. Automatikusan megváltoztatja a forráskódot, hogy megfeleljen a beállított szabályoknak, például behúzást, sorvégeket, zárójelek elhelyezkedését és egyéb stilisztikai előírásokat tekintve.

A prettier-plugin-tailwindcss egy Prettier plugint jelent, amely specifikusan a Tailwind CSS keretrendszert célozza meg. Ez a plugin biztosítja a kódformázás támogatását a Tailwind CSS stílusosztályok esetén, ennek eredményeként a kód formázása egységes marad a projektben. Másrészt a CSS osztályok meghatározási sorrendje hatással van arra, hogy milyen sorrendben lesznek alkalmazva a futás során, így ha két osztálynak ugyanannak a HTML elemnek a stílusát határozza meg

## 4.6. ESLint

Az ESLint egy nyílt forráskódú statikus kódelemző eszköz JavaScript és TypeScript projektekhöz. Az ESLint segít azonosítani és jelenteni a potenciális hibákat, stilisztikai problémákat és a kódminőséggel kapcsolatos javaslatokat a forráskódban.

A TypeScript, Prettier és Tailwind CSS eszközökkel együtt használva az ESLint-et, különböző pluginokat lehet telepíteni és konfigurálni, hogy ezek az eszközök együttműködjenek és a lehető legjobb eredményt érjék el.

A TypeScript-hez való használat során az @typescript-eslint/eslint-plugin plugin szükséges, amely kiterjeszti az ESLint funkcionalitását a TypeScript specifikus ellenőrzésekkel és szabályokkal. Ez lehetővé teszi a TypeScript típusrendszerének és jellemzőinek kihasználását az eredményesebb kódelemzés érdekében.

A Prettier egy formázási eszköz, amely segít egységes stílust és formázást elérni a kódprojektben. Az eslint-plugin-prettier és az eslint-config-prettier pluginok segítenek

az ESLint integrálásában a Prettier-rel, hogy a kódformázási szabályok automatikusan érvényesüljenek az ESLint ellenőrzése során.

A Tailwind CSS egy CSS keretrendszer, amely segít gyorsan és hatékonyan kialakítani a felhasználói felületet. Az eslint-plugin-tailwind plugin lehetővé teszi az ESLint konfigurálását a Tailwind CSS használata során, hogy ellenőrizze a helyes osztályneveket és szintaktikákat, valamint segítsen az átfogó kódellenőrzésben.

Az említett pluginok együttes használatával az ESLint képes integrálni a TypeScript típusellenőrzést, a Prettier automatikus formázást és a Tailwind CSS specifikus ellenőrzéseket a projektbe, ezáltal elősegítve a kód minőségét, konzisztenciáját és olvashatóságát.

## 4.7. npm és pnpm

Az npm egy csomagkezelő a JavaScript programozási nyelvhez, amelyet az npm, Inc. tart fenn. Az npm a Node.js JavaScript futtatási környezet alapértelmezett csomagkezelője. Egy parancssori kliensből, és egy nyilvános online adatbázisából, az npm registry-ből áll.

A pnpm ennek egy új, gyorsabb alternatívája, amely azonos funkciókkal rendelkezik. Ezt fogom használni az alkalmazáshoz felhasznált csomagok telepítésére.

## 5. fejezet

# Gyakorlati megvalósítás

### 5.1. Projekt létrehozása és konfigurálása

Az alkalmazás az `Elost` nevet kapja, ami a magyar eloszt szóból jön, ami a felhasználni kívánt GUN adatbázis természetére utal.

A projektet az előző fejezetben bemutatott Vite segítségével hozzuk létre. Ehhez a `pnpm create vite elost --template vue` parancsot használjuk, amely létrehoz egy TypeScript nyelvben írt React sablont, amely a futtatáshoz szükséges összes függőséget beállítja.

Ezután a `pnpm` segítségével még telepítjük és konfiguráljuk a GUN, Tailwind CSS és Prettier csomagokat. Mivel minden szükséges függőség települt, elkezdhetjük írni a kódot.

### 5.2. GUN inicializálása

Az `src` könyvtárban belül írom a forráskódot. Itt hozom létre a `user.ts` fájlt, itt inicializálom a GUN adatbázist, és ennek a segítségével hozom létre a `user` változót, amely az adatbázisban tárolt felhasználói profilhoz nyújt hozzáférést. A `.recall({ sessionStorage: true })` metódushívás megpróbálja kiolvasni a lokális adattárból a felhasználói adatokat, amelyeket a munkamenet végéig tárol, így egy oldalfrissítés után sem kell újra bejelentkeznünk.

Az importolt SEA és AXE könyvtárakra szüksége van a GUN adatbázisnak titkosításhoz és a többi felhasználóhoz való csatlakozáshoz.

```
import GUN from 'gun';
import 'gun/sea';
import 'gun/axe';

export const gun = new GUN();

export const user = gun.user().recall({ sessionStorage: true });
```

#### 5.1. kódrészlet. `user.ts`



## 5.3. Felhasználói felület inicializálása

Minde webalkalmazás alapja az index.html fájl, amely tartalmazza a kezdeti statikus struktúrát. Itt létrehozunk egy "root" (gyökér) div elemet, amelyen belül a React majd létrehozza felhasználói felület többi részét.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Elost</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.tsx"></script>
  </body>
</html>
```

### 5.2. kódrészlet. index.html

Ezt követően a main.tsx fájlban létrehozzuk a React Virtuális DOM-ját. A "root" div taget állítjuk be gyökérnek, majd ezen belül kirajzoljuk az App komponenst. Az App tartalmazza a felhasználói felület többi részét, és JSX elemként, a HTML szintaxisához hasonlóan tagként adjuk meg.

```
ReactDOM.createRoot(document.getElementById('root') as HTMLElement).render(
  <App />
);
```

### 5.3. kódrészlet. main.tsx

Az App komponens attól függően, hogy be van-e jelentkezve a felhasználó, vagy a Login vagy a Home komponens fogja kirajzolni.

```
export default function App() {
  // ...
  return isLoggedIn ? <Home /> : <Login />;
}
```

### 5.4. kódrészlet. App komponens

A felhasználónevet globális adatként használjuk, mivel több helyen is szükség lesz rá az alkalmazáson belül. Ehhez egy saját hookot hozunk létre, amelyet setUsername-nek nevezünk. Itt a useState segítségével létrehozzuk a felhasználónév állapotát és feliratkozunk a GUN "auth" eseményére, amely a bejelentkezés során hívódik meg. Ekkor megváltoztatjuk a globális felhasználónevet a bejelentkezett felhasználó nevére.

```
export function setUsername() {
  const [username, setUsername] = useState('');

  useEffect(() => {
    let listener: IGunOnEvent | null = null;
```

```

gun.on('auth', async () => {
  const alias: any = await user.get('alias');
  console.log('auth', alias);
  setUsername(alias);
  console.log('signed in as', alias);
});
}, []);

return [username, setUsername] as const;
}

```

### 5.5. kódrészlet. setUsername hook

Ahhoz, hogy bárhol elérjük ezt az állapotot, a `UsernameContext` kontextusba helyezzük. Ehhez létrehozunk egy saját kontextus szolgáltatót, hogy könnyebb legyen a használata.

```

export const UsernameContext = createContext('');

export const UsernameSetterContext = createContext<UsernameSetter>(
  () => undefined,
);

export function UsernameProvider({ children }: { children: React.ReactNode })
{
  const [username, setUsername] = setUsername();

  return (
    <UsernameContext.Provider value={username}>
      <UsernameSetterContext.Provider value={setUsername}>
        {children}
      </UsernameSetterContext.Provider>
    </UsernameContext.Provider>
  );
}

```

### 5.6. kódrészlet. UsernameContext.tsx

A `main.tsx` fájlunkban pedig kiegészítjük a JSX-et a szolgáltatóval.

```

<UsernameProvider>
  <App />
</UsernameProvider>

```

### 5.7. kódrészlet. App komponens a Username kontextussal

Így már bármelyik komponensben elérhetjük a felhasználónevet: `const username = useContext(UsernameContext);`

## 5.4. Bejelentkezési oldal

A bejelentkezési oldalon a megadott felhasználónévvel és jelszóval tudunk bejelentkezni vagy regisztrálni. A bejelentkezéshez a `user.auth` függvényt használjuk, amelynek átadjuk a felhasználónevet és a jelszót, harmadik paraméternek pedig egy függvényt adunk át, ami hiba esetén tájékoztatja a felhasználót.

Regisztrálás esetén a `user.create` függvényt hívjuk, aminek szintén átadjuk az adatokat, majd sikeres regisztrálás esetén a már említett módon bejelentkezünk.

```
function handleLogin() {
  console.log('login with', username, password);
  user.auth(username, password, ({ err }) => {
    if (err) alert(err);
  });
}

function handleSignUp() {
  console.log('sign up with', username, password);
  user.create(username, password, ({ err }) => {
    if (err) {
      alert(err);
    } else {
      handleLogin();
    }
  });
}
```

### 5.8. kódrészlet. Regisztrálás és bejelentkezés

## 5.5. Főoldal

A főoldal két fő elemből áll: az oldalmenü, amit a `Drawer` (fiók) nevű komponens tartalmaz, és a chat.

Ezen az oldalon mindig csatlakozva leszünk egy chathez. A beszélgetés állapotának könnyebb kezeléséhez itt is létrehozunk egy saját kontextus szolgáltatót.

```
export const defaultContact: UserContact = {
  publicKey: 'elost-public-test',
  name: 'Elost Public Test',
  image: getAvatar('Elost Public Test'),
};

export const ContactContext = createContext<UserContact>(defaultContact);

export const ContactSetterContext = createContext<(value: UserContact) =>
  void>(
    () => undefined,
  );
```

```
export function ContactProvider({ children }: { children: ReactNode }) {
  const [contact, setContact] = useState(defaultContact);

  return (
    <ContactContext.Provider value={contact}>
      <ContactSetterContext.Provider value={setContact}>
        {children}
      </ContactSetterContext.Provider>
    </ContactContext.Provider>
  );
}
```

### 5.9. kódrészlet. ContactContext.tsx

Kiegészítjük az App komponensünket a kontextus szolgáltatóval.

```
export default function App() {
  const isLoggedIn = !!useContext(UsernameContext);

  return isLoggedIn ? (
    <ContactProvider>
      <Home />
    </ContactProvider>
  ) : (
    <Login />
  );
}
```

### 5.10. kódrészlet. Contact kontextus átadása a Home komponensnek

## 5.5.1. Oldalmenü

Az oldalmenühöz hozzáadjuk a kijelentkezéshez szükséges gombot, amely a `user.leave` függvényt hívja, ami jelzi az adatbázisnak a szándékunkat.

Az oldamenüben helyet kap egy keresősáv. Reguláris kifejezés segítségével ellenőrizzük, hogy az itt megadott érték egy URL az adott linkhez kapcsolódó nyilvános beszélgetéshez vagy egy másik felhasználó publikus kulcsa privát beszélgetéshez.

```
export async function getContact(newKey: string): Promise<UserContact | null>
{
  if (!newKey) {
    return defaultContact;
  }

  const urlRegex =
    /^(https?:\/\/\/)(?!localhost)[a-zA-Z0-9-_.]+(:\d+)?(\/[a-zA-Z0-9-_.]+)*\/*$/;
  if (urlRegex.test(newKey)) {
    return {
      name: newKey,
```

```

        image: newKey + '/favicon.ico',
        publicKey: newKey,
    };
}

const to: any = gun.user(newKey);
const who = await to.then();
if (who?.alias) {
    const alias: string = who.alias;
    return {
        name: alias,
        image: getAvatar(alias),
        publicKey: newKey,
    };
}

return null;
}

```

### 5.11. kódrészlet. getContact.ts

A saját és a többi felhasználó profilképének meghatározásához a saját a DiceBear nyilvános API-ját használom, ami a név kezdőbetűiből alkot egy profilképet.

```

export function getAvatar(username: string) {
    return 'https://avatars.dicebear.com/api/initials/${username}.svg';
}

```

### 5.12. kódrészlet. getAvatar.ts

A keresősáv alatt fognak helyet kapni a kiegészítők, amelyeket GUN adatbázisába tölthetünk fel mint JavaScript kódok. Minden új linkre való csatlakozáskor lefuttatjuk ezeket a szkripteket, így tudunk új funkciókat hozzáadni a weboldalakhoz.

A `gun.get(ext.id)` segítségével kiolvassuk az adatbázisból ID alapján a kiegészítőt. Ez egy aszinkron függvény, amit az `await` kulcsszó segítségével tudunk bevárni. A visszakapott szkriptet az `eval` segítségével tudjuk futtatni. Ha ez a kód egy függvényt térít vissza, akkor a jelenlegi linket átadva futtatjuk.

```

if (lastContact !== contact) {
    (async function () {
        for (const ext of installedExtensions) {
            const content = await gun.get(ext.id);
            const result = eval(content);
            if (typeof result === 'function') {
                await result(contact.publicKey);
            }
        }
    })();
    lastContact = contact;
}

```

---

### 5.13. kódrészlet. Bővitmények futtatása

Hozzáadjuk azt a funkcionalitást az alkalmazáshoz, hogy egy adott kiegészítőkre klikkelve tudjuk telepíteni, vagy ha már telepítve van, akkor törölni. Mivel Reactben nem ajánlott közvetlenül módosítani a változók állapotát, szükség van a telepített kiegészítőket tartalmazó tömb teljes bejárására és lemásolására.

```
function handleExtensionClick(extension: ExtensionInfo) {
  setInstalledExtensions(prev => {
    const next = prev.filter(ext => ext !== extension);
    if (next.length === prev.length) {
      next.push(extension);
    }
    return next;
  });
}
```

---

### 5.14. kódrészlet. Bővitmények telepítése/törlése

## 5.5.2. Chat

A chat ablakban elolvashatjuk az eddigi üzeneteket és egy új szöveget vagy fájlt küldhetünk.

Az üzenetek használatának megkönnyítésére készítünk egy `useMessages` hookot, ami kiolvassa az adatbázisból az üzeneteket és továbbítja a komponensek felé. A `gun.get(key).map().once` metódus minden üzenetre lefuttatja a megadott függvényt. Ebben a függvényben egy tömbben eltároljuk, hogy az adott üzenet tartalmát, illetve hogy kitől és mikor érkezett. A hook pedig visszatéríti ezen üzeneteket a komponensekben használt típusra transzformálva.

```
export function useMessages() {
  const username = useContext(UsernameContext);
  const contact = useContext(ContactContext);
  const [fetchedMessages, setFetchedMessages] =
    useState<FetchedMessage[]>([]);

  useEffect(() => {
    gun
      .get(contact.publicKey)
      .map()
      .once(async data => {
        if (!data) {
          return;
        }

        const message = {
          who: await gun.user(data).get('alias'),
```

```

    what: String(await Gun.SEA.decrypt(data.what, SECRET_KEY)),
    when: (GUN.state as any).is(data, 'what'),
  };

  if (message.what) {
    setFetchedMessages(prevMessages => {
      const nextMessages = prevMessages.slice(-100);
      nextMessages.push(message);
      return nextMessages.sort((a, b) => a.when - b.when);
    });
  }
});
}, [contact.publicKey]);

return fetchedMessages.map<Message>(message => ({
  status: message.who === username ? 'sent' : 'received',
  content: message.what,
  image: getAvatar(message.who),
  timestamp: new Date(message.when).toLocaleTimeString(),
})));
}

```

#### 5.15. kódrészlet. useMessages.ts

Az üzenetküldések során a GUN SEA protokollját használva titkosítjuk az üzeneteket, majd tároljuk az adatbázisban. A useMessages hookban ismertetett eseményfigyelő érzékeli a változást és kiegészíti az üzenetek tömbjét a saját üzenetünkkel

```

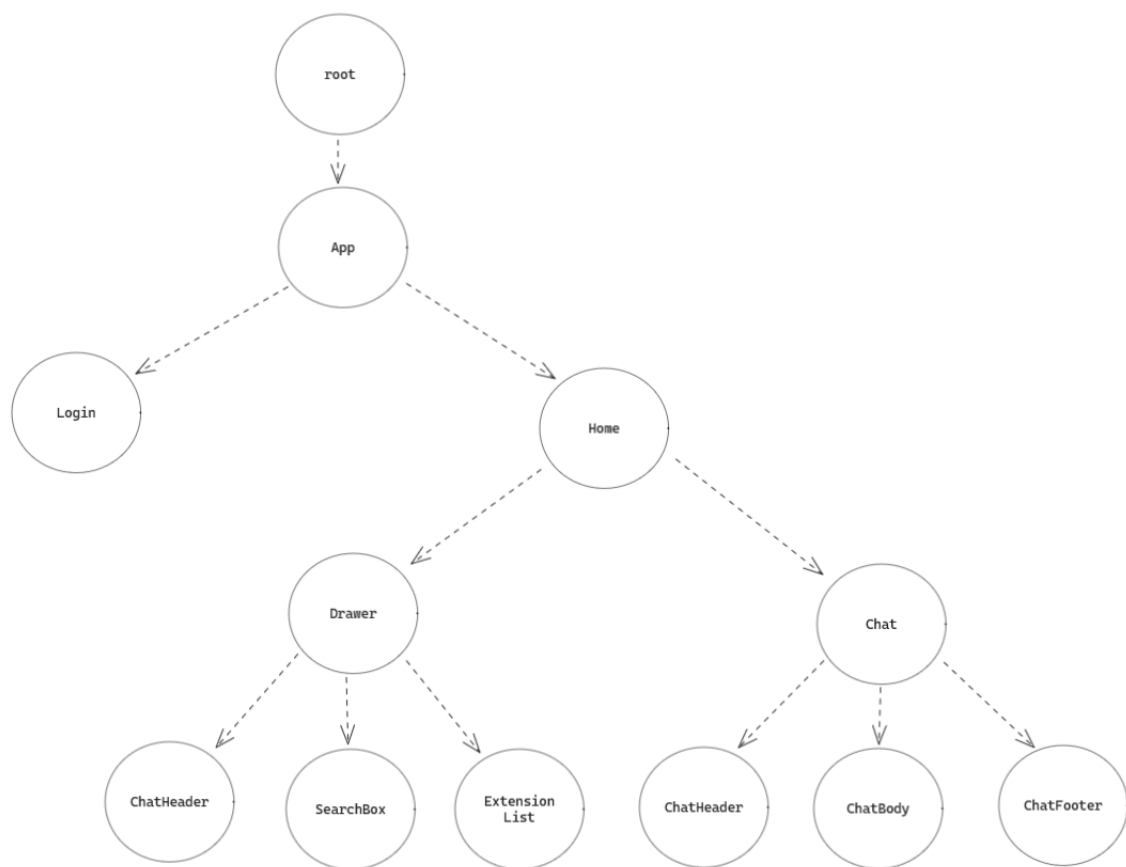
async function sendMessage(e: FormEvent<HTMLFormElement>) {
  e.preventDefault();
  const secret: any = await Gun.SEA.encrypt(message, SECRET_KEY);
  const encrypted = user.get('all').set({ what: secret });
  const index = new Date().toISOString();
  gun.get(contact.publicKey).get(index).put(encrypted);
}

```

#### 5.16. kódrészlet. Üzenet küldése

## 5.6. Felhasználói felület megjelenítése

Az alkalmazás kinézetéhez a már említett Tailwind CSS könyvtárat használjuk, amely segítségével osztályokkal írhatjuk le a komponensek stílusát. A végső eredményt a következő fejezetben mutatom be.



5.1. ábra. Komponensek felépítése



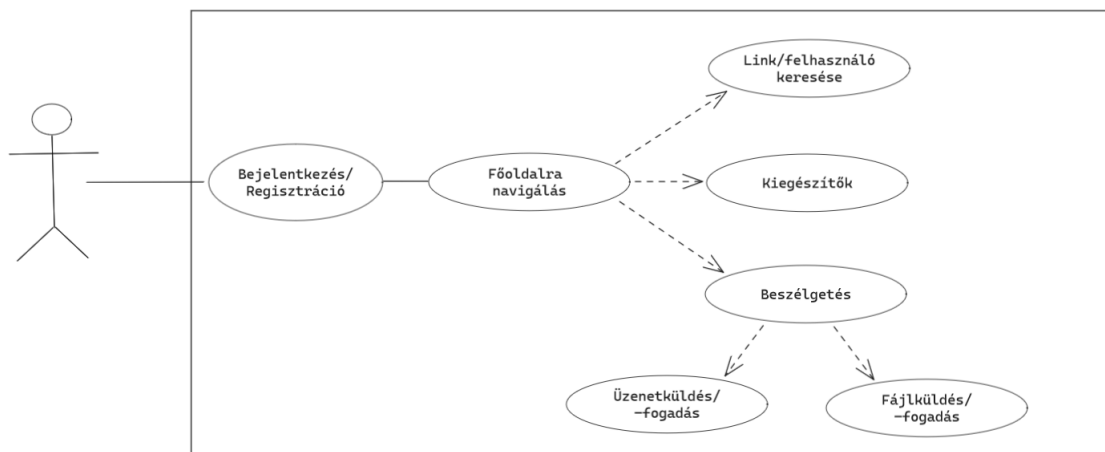
## 6. fejezet

# Felhasználói útmutatók

### 6.1. Nem funkcionális követelmények

A weboldalon futó vagy böngésző kiegészítő változathoz egy modern böngészőre (például Firefox, Google Chrome, Microsoft Edge), a natív változathoz Windows operációs rendszerű asztali számítógépre vagy Androidot futtató telefonra van szükség. Az alkalmazásnak szüksége van internet-hozzáférésre és engedélyre, hogy használhassa a helyi adattárat.

### 6.2. Az alkalmazás menete

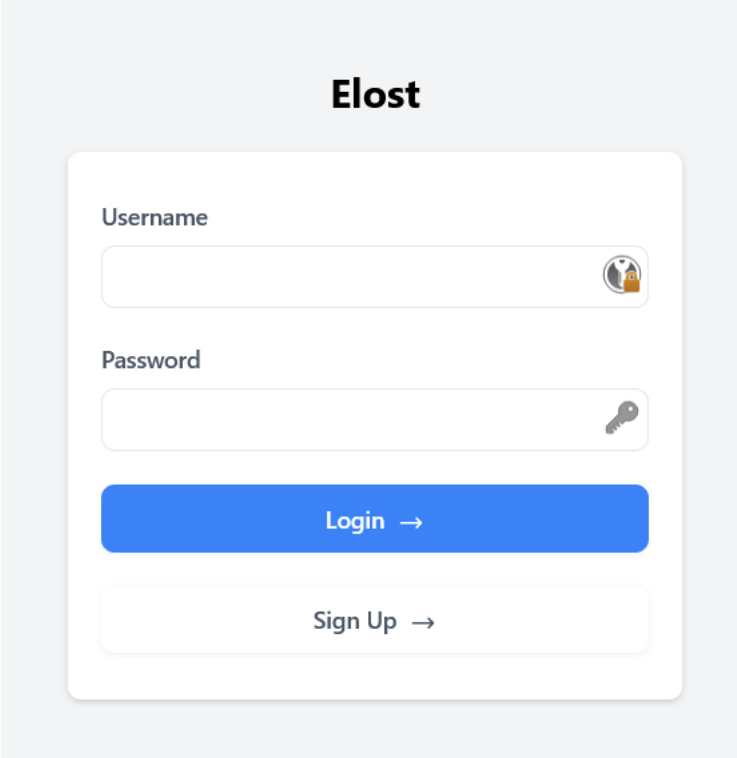


6.1. ábra. Use case diagram

#### 6.2.1. Bejelentkezési folyamat

Az felhasználó kezdetben a bejelentkezési oldalra érkezik. Ha a jelenlegi munkamenetben előzőleg már belépett, akkor a GUN könyvtár a lokális adattárból kiolvassa a felhasználói azonosítókat és automatikusan belépteti a felhasználót. Ha nem találhatók

ezek az adatok, akkor a felhasználó megadhat egy nevet és egy jelszót, a Sign Up (regisztrálás) gombra kattintva regisztrálhat vagy a Login (bejelentkezés) gombra kattintva bejelentkezhet, ha már előzőleg regisztrált.



The image shows a login and sign-up interface for a platform named "Elost". The title "Elost" is centered at the top in a bold, black font. Below the title is a white rectangular form with rounded corners. Inside the form, there are two input fields: "Username" and "Password". The "Username" field has a small icon of a person with a lock on the right side. The "Password" field has a small icon of a key on the right side. Below the input fields are two buttons: a blue button labeled "Login →" and a white button labeled "Sign Up →".

**6.2. ábra.** Bejelentkezési oldal

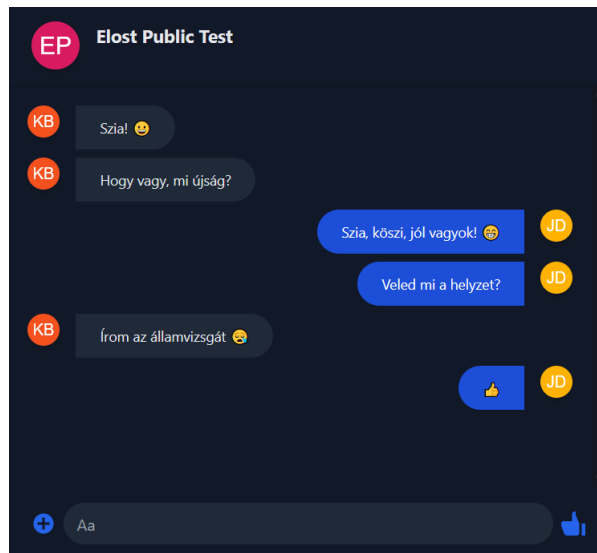
### 6.2.2. Beszélgetés

Ezt követően a felhasználó a főoldalra kerül. Itt a jobb oldalon található a chat ablak. Ez alapértelmezetten a nyilvános Elost chat, amihez bárki hozzáfér és hozzászólhat. A fejlécben látható a chat neve, baloldalon a kapott és jobb oldalon a küldött üzenetek. A lábléc baloldalán található plusz ikonra kattintva a felhasználó fájlt tölthet fel, középen a bemeneti mezőben megírhatja az üzenetét, a jobb oldali ikonnal pedig like emojit küldhet.

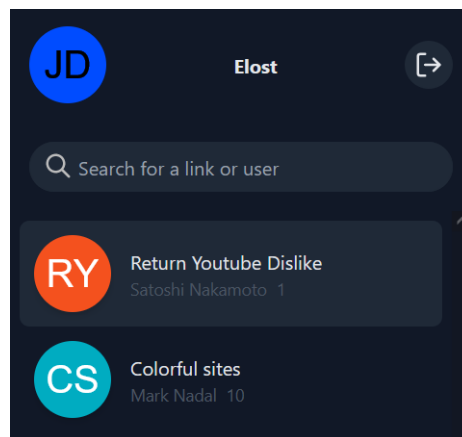
### 6.2.3. Nyilvános beszélgetések és felhasználók keresése

Bal oldalon található az oldalmenü, ahol a felhasználó a URL-ekre keresve más nyilvános beszélgetéseket találhat, vagy egy másik felhasználó publikus kulcsát megadva privát beszélgetést indíthat.

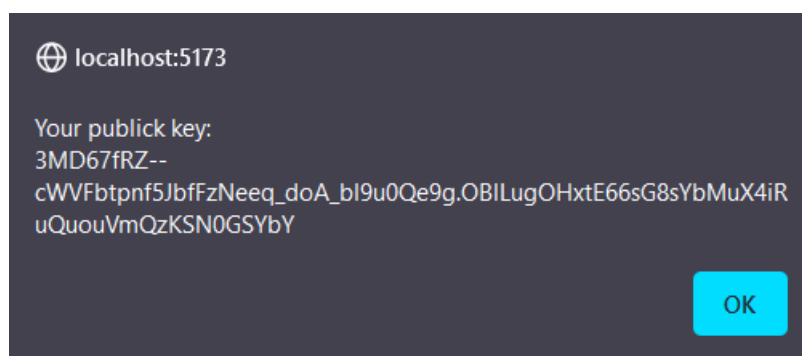
Bal felső sarokban a felhasználó a profilképére kattintva előhozhatja a saját publikus kulcsát, amit megoszthat másokkal.



6.3. ábra. Chat ablak



6.4. ábra. Oldalmenü



6.5. ábra. Nyilvános kulcs megtekintése

#### 6.2.4. Kiegészítők

Az oldalmenü alján találhatóak a telepíthető kiegészítők, amelyeknek a tartalma szintén a GUN adatbázisába van tárolva. A kiegészítők minden új linkhez csatlakozva

lefutnak, így új funkcionalitásokkal egészíthetik ki a weboldalakakat. A felhasználó a kiegészítőre kattintva telepítheti azt, vagy ha már telepítve volt, akkor törölheti.

### **6.2.5. Kijelentkezés**

Az oldalmenü tetején található a kijelentkezéshez szükséges gomb. A felhasználó a kijelentkezést követően újra a bejelentkezési oldalra kerül.

## 7. fejezet

# Továbbfejlesztési lehetőségek

Az alkalmazás továbbfejlesztése során számos új funkcióval és javítással bővíthető lenne. Egy lehetséges kiegészítés lehetne a csoportos beszélgetés lehetőségének implementálása. Ez lehetővé tenné több felhasználó közötti interakciót és egyidejű kommunikációt egy adott témában vagy csoportban. A csoportos beszélgetés funkció hozzáadása bővítené az alkalmazás funkcionalitását és vonzóbbá tenné a felhasználók számára.

Továbbá, az alkalmazásnak lehetőséget kellene biztosítania az előző beszélgetések és üzenetek tartós tárolására. Ez lehetővé tenné a felhasználók számára, hogy később visszatérjenek az előzőleg folytatott beszélgetésekhez, vagy áttekintést nyerjenek korábbi üzeneteikről. A tartós tárolás megvalósítása segítené az adatok hosszabb távú megőrzésében és könnyebb elérésében.

Az alkalmazás jelenlegi bővítmény rendszere nem biztonságos, mivel az `eval` függvényt használja a bővítmények futtatására. Ez sebezhetőséget jelenthet a rosszindulatú kódok és támadások számára. Az alkalmazást fejlesztés során biztonságosabb bővítmény rendszerre kellene átállítani, amely lehetővé teszi a kiegészítők biztonságos futtatását és védelmet nyújt a kártékony tevékenységek ellen. Ennek egyik módja a megbízható közvetítő szerverek használata volna, amelyek szintén a GUN adatbázis csomópontjaiként működnének, de a tartalmuk ellenőrizhető volna. Ezen szerverek segítségével biztosítható lenne a bővítmények tárolása és ellenőrzése, így minimalizálva a biztonsági kockázatokat és fokozva a rendszer megbízhatóságát.

Ezek a továbbfejlesztési lehetőségek segítenének az alkalmazás funkcionalitásának javításában, a felhasználói élmény növelésében és a biztonságosság biztosításában. Az ilyen fejlesztések hozzájárulnának ahhoz, hogy a decentralizált üzenetküldő és adatmegosztó alkalmazás még vonzóbbá váljon a felhasználók számára, és növelje az elterjedését a jövőben.

# Összefoglaló

A dolgozatomban bemutatottam egy decentralizált üzenetküldő és adatmegosztó alkalmazás koncepcióját, amely az innovatív technológiák és a decentralizáció erejét felhasználva készült. A decentralizáció elve, amelyet az alkalmazás tervezésénél szem előtt tartottam, rendkívül fontos a digitális világ jövője szempontjából.

A decentralizáció számos előnyt kínál a hagyományos, központosított rendszerekkel szemben. Az adatok elosztott tárolása és a hálózati erőforrások megosztása nagyobb biztonságot és ellenálló képességet biztosít a kiberfenyegetésekkel és a ponti hibapontokkal szemben. Emellett a felhasználók nagyobb irányítást és adatvédelmet élveznek, mivel az adatokat saját eszközeiken tárolják, és azokhoz csak ők férhetnek hozzá.

A decentralizált alkalmazások jövője ígéretesnek tűnik, és számos területen megoldást kínálhatnak a központosított rendszerek korlátaira. Az adatvédelem és a felhasználói szabadság mellett a decentralizáció lehetőséget ad az innovációra és az együttműködésre, miközben csökkenti az átláthatatlanságot és a függőséget a nagy tech vállalatoktól.

Személy szerint, én is szeretnék tovább foglalkozni a decentralizált alkalmazásokkal a jövőben. Hiszek abban, hogy az ilyen típusú alkalmazásoknak jelentős hatásuk lehet a digitális világra, és lehetőséget teremtenek az emberek számára, hogy nagyobb kontrollt gyakoroljanak saját adataik felett. Az általam létrehozott alkalmazás népszerűsítése és elterjedése fontos célom, és remélem, hogy az emberek felismerik a decentralizáció előnyeit és előnyben részesítik az ilyen típusú megoldásokat a jövőben.

Összességében a dolgozatomban bemutatott decentralizált üzenetküldő és adatmegosztó alkalmazás kiemeli a decentralizáció előnyeit és lehetőségeit a digitális világban. Az alkalmazás tervezése és fejlesztése során szem előtt tartottam a felhasználók adatvédelmét és irányítását, és remélem, hogy a jövőben egyre többen kezdenek el foglalkozni a decentralizált technológiákkal és alkalmazásokkal.

GitHub link: <https://github.com/prlnd/elost>

# Ábrák jegyzéke

3.1. A GUN rendszere . . . . .	15
5.1. Komponensek felépítése . . . . .	32
6.1. Use case diagram . . . . .	33
6.2. Bejelentkezési oldal . . . . .	34
6.3. Chat ablak . . . . .	35
6.4. Oldalmenü . . . . .	35
6.5. Nyilvános kulcs megtekintése . . . . .	35

# Irodalomjegyzék

- [NC] Mark Nadal and Dr. Amber Cazzell. A trustless decentralized bandwidth incentive.
- [OO21] Woko Ovunda and Asagba Prince Oghenekaro. Database systems model: Distributed. *J. Algebra*, 7, 2021.
- [WLA] Michael Kaminsky Wyatt Lloyd, Michael J. Freedman and David G. Andersen. Don't settle for eventual consistency, stronger properties for low-latency geo-replicated storage.