ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

OE-NIK	Hallgató neve:	**Bogdán Roland János**
2022	Hallgató törzskönyvi száma:	**T/006594/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Szoftvertervezés és -fejlesztés Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Bogdán Roland János**
Törzskönyvi száma: T/006594/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

Decentralizált blogger webalkalmazás Ethereum blokkláncon
Decentralized blogging web application on the Ethereum blockchain

Intézményi konzulens: Sipos Miklós László
Külső konzulens:

Beadási határidő: 2022. május 15.

A záróvizsga tárgyai: Szoftvertechnológia és grafikus felhasználói interfész tervezése
Szoftvertervezés és -fejlesztés specializáció

A feladat

A feladat egy blogger webalkalmazás készítése, decentralizált backend szolgáltatásokkal, Ethereum blokkláncra kihelyezve. Vizsgálja meg, hogy mely technológiák szükségesek egy blokklánc alapú webes rendszer készítéséhez, elemezze azokat pozitív és negatív oldalról, hasonlítsa össze a hagyományos centralizált és a decentralizált alkalmazásokat. Az adott technológia melletti döntését támassza alá. Az alkalmazást lehessen használni felhasználói fiók nélkül, vagy regisztrálva egy kriptovaluta pénztárca segítségével (pl. Metamask). A regisztrált felhasználók tudjanak blog posztokat írni, más felhasználókat követni, feliratkozni rájuk, és a posztok alá megjegyzést írni. Más felhasználók követése legyen ingyenes, feliratkozni rájuk pedig járjon havi díjjal. Egy blog poszt írásánál a felhasználó dönthesse el, hogy az adott posztot láthassa bárki, csak a követői, vagy csak a rá feliratkozott felhasználók. A fizetés okos szerződések segítségével történjen, ezek Solidity programozási nyelven jöjjenek létre. A webes felület Angular használatával készüljön el. Mivel a blokkláncra felhelyezett okos szerződések nem módosíthatók, végezzen alapos elemzést a tesztelési lehetőségekről és készítse is el az egyes teszt típusokat és teszteseteket.

A dolgozatnak tartalmaznia kell:

- a megoldandó feladat pontos leírását és részletes elemzését,
- magas szintű szerkezeti mintát az alkalmazásról,
- az egyes funkciók megvalósításához felhasználható lehetséges technológiákat,
- az elkészített rendszer tervét, blokkdiagramját,
- a megvalósítás menetét, felmerülő problémákat és az azokra adott megoldást,
- a tesztelési tervet, a tesztelés módszertanát és a tesztelés eredményeit,
- az elkészült szoftver felhasználási lehetőségeit.

Dr. Vámossy Zoltán
intézetigazgató

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

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

.....
intézményi konzulens

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

HALLGATÓI NYILATKOZAT

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

Budapest, 2022.12.07.

Bogdán Roland

hallgató aláírása

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar
Neumann Faculty of Informatics

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Neptun kód: Tagozat:

BOGDÁN ROLAND JÁNOS [redacted] NAPPALI

Telefon: Levelezési cím (pl: lakcím):

[redacted]

Szakdolgozat címe magyarul:

DECENTRALIZÁLT BLOGGER WEBALKALMAZÁS ETHEREUM BLOKKLÁNCON

Szakdolgozat címe angolul:

DECENTRALIZED BLOGGING WEB APPLICATION ON THE ETHEREUM BLOCKCHAIN

Intézményi konzulens: Külső konzulens:

SIPOS MIKLÓS LÁSZLÓ ---

Alk.	Dátum	Tartalom	Aláírás
1.	2021.09.10.	Félév teendőinek, menetrendjének és mérföldköveinek ismertetése.	
2.	2021.10.18.	Hasonló rendszerek elemzése.	
3.	2021.11.22.	Irodalomkutatás, annak összegzése valamint konzekvenciák meghatározása.	
4.	2021.12.06.	Tervezés, követelmények specifikációja. Véglegesítés. Leadással kapcsolatos teendők egyeztetése.	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

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

Budapest, 2021.12.06.

Sipos Miklós

intézményi konzulens

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve: BOGDÁN ROLAND JÁNOS

Neptun Kód: [redacted]

Tagozat: NAPPALI TAGOZAT

Telefon: [redacted]

Levelezési cím (pl.: lakcím): [redacted]

Szakdolgozat címe magyarul:

DECENTRALIZÁLT BLOGGER WEBALKALMAZÁS ETHEREUM BLOKKLÁNCON

Szakdolgozat címe angolul:

DECENTRALIZED BLOGGING WEB APPLICATION ON THE ETHEREUM BLOCKCHAIN

Intézményi konzulens: SIPOS MIKLÓS LÁSZLÓ

Külső konzulens: -

Alk.	Dátum	Tartalom	Aláírás
1.	2022.10.02.	Fejlesztés tervezett menetének revíziója, szakmai egyeztetés, fejlesztési teendők megkezdése.	
2.	2022.10.30.	Fejlesztés állapotának és haladásának áttekintése. Technikai problémák és módosítások egyeztetése.	
3.	2022.11.20.	Felmerülő problémák egyeztetése és megoldása. Tesztelési fázis megindítása.	
4.	2022.11.30.	Dokumentáció véglegesítése. Leadással kapcsolatos teendők egyeztetése.	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

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

.....

Intézményi konzulens

Budapest, 2022.12.01.

ABSZTRAKT

A 21. században az internetet olyan cégek dominálják, akik szolgáltatásokat ajánlanak személyes adatainkért cserébe. Az ilyen weboldalakat centralizált vagy Web 2.0 oldalakként ismerjük. A Web 3.0 egy blokklánc alapú, decentralizált irányba vinné a mai internetet, ezzel biztosítva a biztonságot, adatvédelmet, anonimitást, cenzúramentességet, és az elérhetőséget.

Szakdolgozatom célja, a Web 3.0-hoz kapcsolódó technológiák bemutatása, és egy olyan blogger alkalmazás készítése, ahol bárki megoszthatja gondolatait és véleményeit anélkül, hogy félnie kellene attól, hogy a jövőben ez milyen kihatással lesz az életére. Fontos az anonimitás, de azt a lehetőséget is meg szeretném adni a felhasználóknak, hogy valós névvel és arccal is használhassák az alkalmazást. Fontos az rendelkezésre állás, a decentralizált alkalmazások hatalmas előnye, hogy folyamatosan elérhetőek, mivel nem egy szervertől függenek, hanem akár több millió csomóponttól. További célom, hogy ahelyett, hogy a felhasználó adatai el lennének adva egy cég jövedelmének érdekében, inkább a felhasználó tudja monetizálni az általa létrehozott tartalmakat, blogposztokat.

Irodalomkutatás során számos Web3-hoz kapcsolódó témakört megvizsgálok, mint például a blokklánc és a kriptovaluta. Célom releváns technológiákat használni, mint például az Angular és a Solidity, ezeken felül pedig a fejlesztés során kihasználni a hozzájuk kapcsolódó eszközöket, mint például a Truffle vagy a Ganache.

A fejlesztés során elkészült alkalmazás képes számos felhasználót kiszolgálni anonim módon, megbízhatóan, egy mögöttes centrális szervezet nélkül. A fejlesztés menetével bemutatom az előnyeit, hátrányait és akadályait a Web3 fejlesztésnek.

A blokklánc véglegességénél fogva nagyon fontos lépés a tesztelés az okos szerződések fejlesztésénél. Ennek úgy érzem eleget tettem számos teszt esettel, mind manuálisan, mind unit tesztek formájában, amik mind az elvárt működés szerinti eredményt adják, szélsőséges esetekben is.

ABSTRACT

In the 21st century, the internet is dominated by companies that provide services in exchange for our personal data. These websites are known as centralized, or Web 2.0 sites. Web 3.0 is taking the internet as we know it today in a blockchain-based, decentralized direction, thus providing security, privacy, anonymity, censor resistance, and accessibility.

The goal of my thesis work is to present the technologies of Web 3.0 and creating a blogging application, where anyone can share their thoughts and ideas without fearing future consequences. Anonymity is important, but I would like to provide the users the opportunity to use their real names and face on the application. Availability is an important aspect as well, one of the huge advantages of decentralized applications is that they are always available, since they aren't dependent on a single server, rather up to millions of nodes. An additional goal is for the user to monetize their content, instead of a company selling the user's data for their profits.

During my research, I'm going to examine several topics relevant to Web3, such as blockchains and cryptocurrencies. My goal is to use relevant technologies, like Angular, and Solidity, and beyond that, to put their tools to use, for example Truffle, and Ganache.

The developed application can serve numerous people anonymously, while being trustworthy, without a central company behind it. During the development itself I showcase the pros, cons, and obstacles of Web3 development.

Data stored on blockchains are permanent, and because of this property, testing is a very important step in smart contract development. I believe I tested the system quite well, using several test cases, by manual testing, and by unit testing, all of which give the expected results, even in extreme cases.

Tartalomjegyzék:

1. BEVEZETÉS

Napjainkban a legtöbb weboldal centralizált. Ez azt jelenti, hogy egy központi szerverről fut, és általában egy cég, egy entitás áll mögötte. Ez a hagyományos megoldás egy bevált jól működő rendszer, viszont hátrányokkal is jár. Az egyik alapvető tényező egy ilyen rendszernél, hogy bíznunk kell abban, hogy a személyes adataink bizalmasan lesznek kezelve. Ha a bizalom adott, egy támadás veszélye akkor is fennáll, és általában nem ismerjük, hogy milyen védelmi intézkedéseket alkalmaz a rendszer szolgáltatója, hogy megvédje az adatainkat. Ha támadás éri a centralizált rendszert, könnyen összeomolhat, mivel általában egy pontot, egy szervert kell támadnia a támadónak. Fennáll annak a veszélye is, hogy egy központi szervezet cenzúrázza az oldalt politikai vagy pénzügyi okokból.

Ezekre a problémákra egy modern, effektív megoldás a decentralizálás. Decentralizálás esetén az alkalmazás és a felhasználón kívüli harmadik ember (aki eddig az alkalmazás szolgáltatója volt) bevonását elhagyhatjuk. Egy nagy köztes szereplő helyett, egy világszerte szétosztott csomópontokból álló hálózatról fut az adott alkalmazás. Természetesen ezt hallva felmerülhet bennünk, hogy ez biztonság szempontjából egy sokkal rosszabb megoldás, ekkor jön képbe a kriptográfia és a blokklánc technológia. A blokklánc egy nyilvános adatbázis, ami a decentralizált hálózat minden csomópontjánál ott van, és csak akkor történik rajta változás, ha a hálózat csomópontjainak a többsége egyetért a változással. Egy blokk bizonyos mennyiségű információt tud tárolni, majd egy következő blokk jön létre. Minden blokk egy kriptografikus referenciát tartalmaz a következő blokkra, innen jön a lánc elnevezés. A blokkláncon szinte bármit tárolhatunk, általában tranzakciókat, és felhasználókat az egyenlegükkel tárolunk rajta. Mivel publikus a blokklánc, felmerülhet, hogy a felhasználók tárolása így elég veszélyes. Erre megoldás, hogy nem szokásos felhasználónév és jelszó párosítást tárolunk, hanem egy publikus és privát kulcs kapcsolatot hozunk létre. A publikus kulcs tárolva van a blokkláncon, ezt bárki tudhatja. A privát kulcs csak a felhasználónál van, jelszóként funkcionál, és matematikailag, visszafejthetetlenül kapcsolódik a nyilvános kulcshoz, így a hagyományos adatbázissal szemben sokkal biztonságosabb, mert nincs eltárolva semmilyen formában a jelszó. Innentől kezdve bár minden információ publikus, mégis anonim marad minden felhasználó, és nincs egy pont, amit támadni, vagy cenzúrázni lehetne, helyette egy decentralizált, akár több millió csomópontból álló hálózat van.

A közösségi médiában manapság szerintem kifejezetten nagy probléma, hogy az oldal tulajdonosai úgy dönthetnek, hogyha valakinek a nézeteivel nem értenek egyet, elveszik a jogosultságát az oldalra való bejegyzés írástól. Ez nagyobb mértékben kihat az újságírókra és bloggerekre, akik sokszor vitatott témákról írnak. Ennél fogva, szakdolgozatomként, egy decentralizált alkalmazást szeretnék készíteni, ami kiváló bloggolásra, biztonságos, anonim, cenzúra mentes, és nyílt forráskódú.

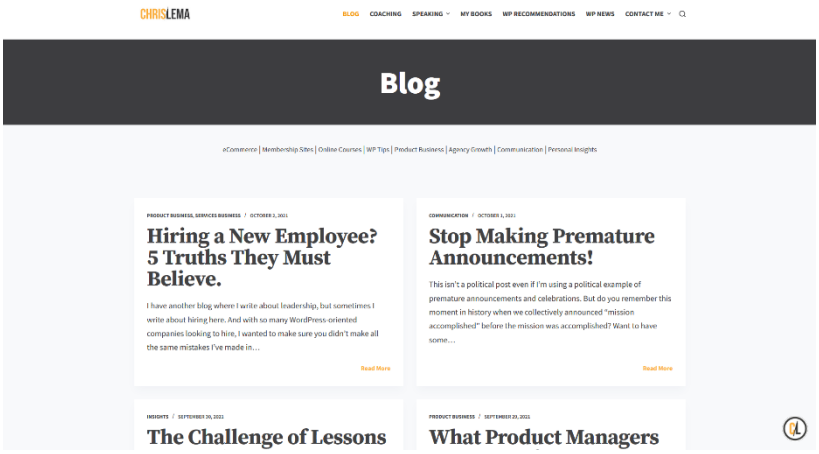
2. IRODALOMKUTATÁS

2.1. Hasonló rendszerek elemzése és bemutatása

A szakdolgozatomhoz hasonló, már létező rendszereket és szolgáltatásokat adott szempontok mentén szeretném elemezni. Két csoportra bontottam őket aszerint, hogy hagyományos centralizált megoldások, vagy decentralizáltak. Az első összehasonlítási szempont, a funkcionalitások listája, ezek közül is a felhasználói fiók létrehozása, szociális interakciók, és a posztok/blog oldal testreszabhatósága. Következő szempont, a felhasználói élmény és felhasználói interfész, tehát mennyire könnyen használható az oldal, és mennyire látványos. Fontos szempont továbbá a szolgáltatás árazása, és hogy monetizálható-e a rajta végzett tevékenységünk. Végül használt technológiák szerint is megvizsgálom a hasonló rendszereket, decentralizált rendszerek esetén kiemelve, hogy milyen blokkláncra vannak kihelyezve ezek az alkalmazások.

2.1.1. Centralizált rendszerek

Két elég népszerű, bár különböző hagyományos centralizált szolgáltatást fogok megvizsgálni: a WordPress, és a Tumblr.



1. ábra: Egy WordPressben készült blog oldal

A WordPress egy széles körben elterjedt szolgáltatás, ami lassan 20 éve létezik, számos funkcionalitással. Az oldal maga nem egy blog, hanem egy általános weboldal készítő alkalmazás, viszont ehhez az elemzéshez, a blog készítési lehetőségeit fogom vizsgálni. Lehet, és kell is felhasználói fiókot létrehozni, hogyha igénybe akarjuk venni az oldalt, hogy létrehozzuk saját weboldalunkat, weblogunkat. Tetszőleges komponensekkel, kiegészítőkkel láthatjuk el weboldalunkat, így rajtunk áll, hogy szeretnénk-e például kommentszekciót a blogposztok alá. Az oldalunk nagy mértékben testre szabható, tömérdek előre elkészített téma közül lehet választani. A WordPress oldalát használni egyszerű, viszont saját oldalunkat létrehozni eleinte nehéz lehet, viszont miután kész van, könnyű kezelni, és új posztokat létrehozni. A felhasználói interfész átláthatósága, kezelhetősége az oldalunkon rajtunk múlik, ami lehet előny és hátrány is. A szolgáltatás

alapvetően ingyenes, viszont az oldal élesbe helyezéséért, tehát a domain név, és a szerver hoszting bérlésért mi felelünk, nekünk kell intézni. A WordPress-el készített weboldal monetizálható, lehet online boltot üzemeltetni a szolgáltatás segítségével, vagy fizetős tagságot adni az oldalunkhoz. Technológiákat tekintve, a készített oldal PHP-t használ a legtöbb funkciójához, és MySQL-t adatbázis szolgáltatásokhoz.

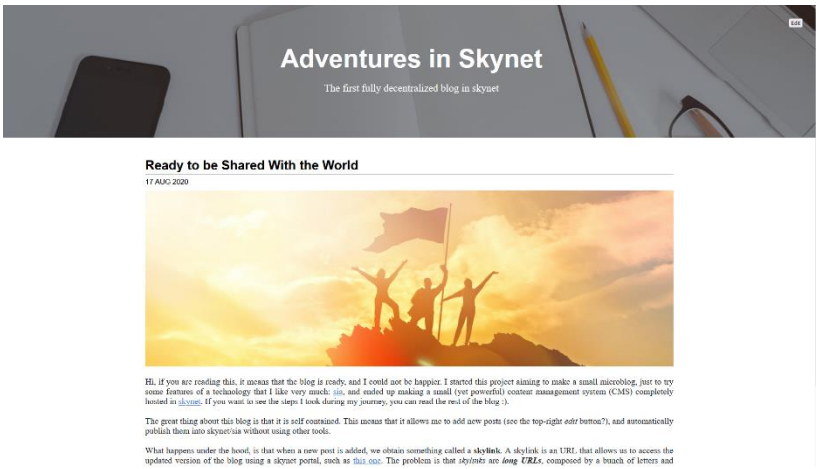
A Tumblr egy közösségi média oldal, amit mikrobloggolásra használnak. A mikrobloggolás azt jelenti, hogy hosszadalmas írások helyett, rövid terjedelmű blog posztok, esetleg csak képek kerülnek megosztásra. Az alkalmazás, és a privát blog oldalak megtekinthetőek regisztrálás és bejelentkezés nélkül, viszont interakció a posztokkal csak bejelentkezés után lehetséges. Közösségi média jellegénél fogva több opció is van beépítve a posztokkal való interakcióra. A felhasználók tudják egymás posztjait kedvelni, megjegyzéssel ellátni, megosztani, vagy más felhasználóknak privát üzenet formájában tovább küldeni, továbbá a felhasználók tudják egymást ingyenesen követni. Saját blog oldalt létre lehet hozni, ezt egy adott mennyiségű téma segítségével lehet testre szabni. A rendszer használata kifejezetten egyszerű, átlátható, felhasználóbarát. Saját blogot létrehozni és szerkeszteni is egyszerű menüpontokon keresztül lehet. Az oldal használata teljesen ingyenes, beleértve a saját oldal létrehozását, és fenntartását. Monetizálásra nincs beépített opció. A Tumblr egy úgy nevezett LAMP szoftvercsomagot használ, ennél fogva a fő használt technológiájuk a Linux, Apache, MySQL, és PHP.

2.1.2. Decentralizált rendszerek

Három alkalmazást találtam ebben a kategóriában, amik nem térnek el túlságosan a blogolás témájától: Wakio, Flote, és Sigle.

A Wakio egy egyszerű webalkalmazás, amiben statikus blog oldalakat lehet létrehozni pár kattintással. Regisztrációra nincs lehetőség, bárki létrehozhat egy blogot szimplán egy cím, egy leírás, és egy borítókép megadásával. A létrehozott oldalon lehetséges egyszerű posztokat hozzáadni a bloghoz, a szöveget alapvető funkciókkal formázni (például félkövér, dőlt betűs, áthúzott szöveg), hivatkozásokat beágyazni, és képeket csatolni. Korábbi posztokat, a címet, és a leírást is bármikor tudjuk módosítani. Az oldalt nem lehet kifejezetten testre szabni, a felosztás és a stílus szinte mindig ugyan az marad. Az oldal interfésze átlátható, kevés gomb és opció van, így a használata is nagyon egyszerű. Az alkalmazás használata ingyenes teljesen, viszont monetizációra nincs beépített opció. A weboldal egy statikus HTML oldal, CSS formázással, a funkcionalitások pedig JavaScript-tel vannak megvalósítva. Az alkalmazás is, és a létrehozott blogok is, a Sia blokkláncra kerülnek fel. A Sia Skynet egy decentralizált hoszting platform, ahol ingyenesen lehet weboldalakat és fájlokat lehet tárolni, amik bárki számára elérhetőek

egy az oldal által generált linken. Ennek a platformnak a saját blokkláncа a Sia, a lánc natív fizetőeszköze a Siacoin.



2. ábra Egy Wakio-ban készült blog oldal

A Flote egy közösségi média, mikroblogger alkalmazás. Az oldalra hagyományos módon email cím, felhasználónév és jelszó kombinációval lehet regisztrálni, majd bejelentkezni. Regisztrált fiók nélkül nem tudunk tartalmat megtekinteni az oldalon. Közösségi média jellegénél fogva a felhasználók közötti interakcióra számos opció van, többek között egymás követése, posztokra való válaszolás, megosztás és kedvelés. Testre szabni nem tudjuk az oldalunkat, a közösségi médiákon megszokott profilkép, borítókép, és profil leíráson kívül. Az alkalmazás interfésze egyszerű, más közösségi média használata után ismerős és intuitív. Az oldal használata ingyenes. A posztjaink monetizálhatóak, a profil beállítások között megadható több feliratkozási szint is egyedi árazással, majd a posztnál opcionálisan beállítható mely feliratkozási szinttel rendelkező felhasználók láthatják. A Flote alkalmazás két blokklánccal van interakcióban, az Ethereum-mal és a Polygon-nal, natív fizetőeszköze a FLOTE.

A Sigle egy teljesen decentralizált blog oldal. Regisztrációkor a rendszer generál nekünk egy titkos kulcsot (ez fog jelszóként funkcionálni), ami 12 véletlenszerű angol szóból áll, majd egy általunk választott felhasználónevet is megadhatunk. Más felhasználók blogjaival tudunk interaktálni a megszokott módokon: fel lehet iratkozni a blogokra, követhetőek, a posztok megoszthatók, és lehet megjegyzéseket hagyni. Testre szabni az oldalunkat nem tudjuk, posztok csak szimpla szövegek, vagy képek lehetnek. A felhasználói interfész egyértelmű, könnyen használható. Egy funkció, amit a Sigle alkalmaz, de a többi alkalmazás nem, az a piszkozatok. Ha félbehagytuk a szerkesztését egy blog posztnak, akkor piszkozatként elmenti a rendszer egy külön fül alá. Az oldal használata ingyenes. A feltöltött tartalmak monetizálhatóak az alapul használt technológia, a Stacks segítségével. A Stacks egy olyan blokklánc, ami kiegészíti a Bitcoin

blokkláncát extra funkcionalitásokkal, közben kihasználva a Bitcoin tranzakció lebonyolítási képességeit.

3. ábra Egy Sigle-ben készült Blog oldal

2.1.3. Összegzés

A centralizáltságon kívül, többféleképpen is csoportosíthatók a megvizsgált rendszerek: vannak alkalmazások, amikkel a saját blog oldalunkat létre tuídjuk hozni, teljesen testre szabva, és vannak, ahol egy közösségi felületen kapunk egy oldalt, ahová kihelyezhetjük a blog posztjainkat. Hasonlóképpen vannak szolgáltatások, amik hosszú, újságcikk szerű posztokra vannak tervezve, és vannak, amik mikroblog-posztokra lettek tervezve. Az tervezett alkalmazásom ezeket szeretné összevonni úgy, hogy közösségi média jellegű legyen az oldal, megtartva az interakciók lehetőségét, és egyaránt optimális legyen hosszabb terjedelmű írásokra és rövid terjedelmű posztokra is.

	Egyszerűen használható	**Szociális**	**Testre szabható**	**Monetizálható**	**Decentralizált**
WordPress[1]	Általában	Lehet	Teljesen	Lehet	Nem
Tumblr	Igen	Igen	Teljesen	Nem	Nem
Wakio	Igen	Nem	Nem	Nem	Igen
Flote	Igen	Nem	Részben	Igen	Igen
Sigle	Részben	Igen	Nem	Igen	Igen
Tervezett alkalmazás	Igen	Igen	Részben	Igen	Igen

1. Táblázat Hasonló rendszerek összehasonlítása Web3 és a decentralizáltság

[1] A WordPress-ben készült blog oldalak

2.2. Web3 és a decentralizáltság

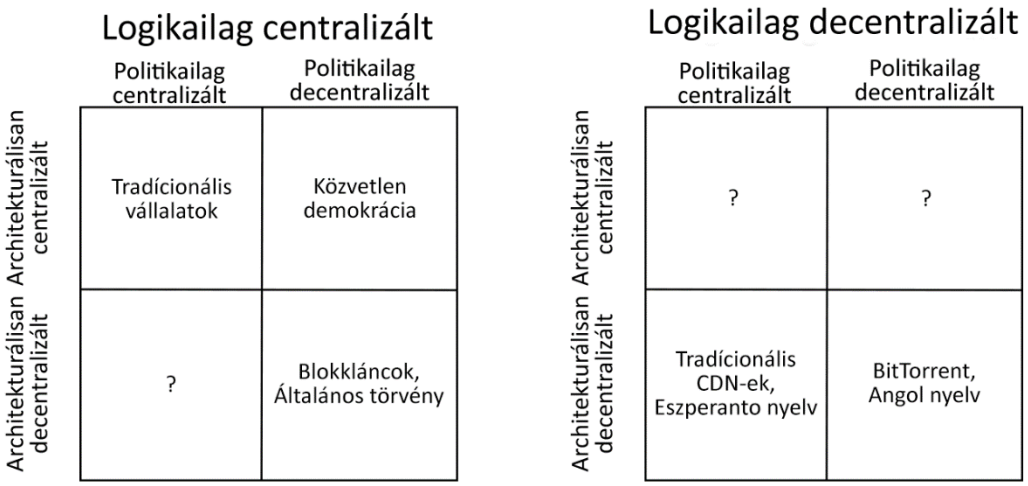
Az internetet jelenleg 3 generációba lehetne sorolni. A Web1.0 a 90-es évekre jellemző, statikus weboldalakkal volt tele, amiket csak asztali számítógépekről lehetett elérni.

A Web2.0 manapság is az internet nagyrészét kiteszi. A weboldalak szinte bármilyen eszközről elérhetőek a nap bármely pontjában, közösségi platformok hatalmasra nőttek, könnyebb emberekkel online kapcsolatba lépni, mint valaha, és a felhő szolgáltatások miatt, egyre olcsóbb és elérhetőbb mindenki számára egy weboldal létrehozása, és üzemeltetése. Az utóbbi pontot kiemelve, egy dolog szinte minden Web2.0 webalkalmazásban közös: egy centralizált szervezet vagy entitás áll mögöttük.

A Web3.0 három szempontban előrelépés a mostani generációhoz képest, az első a bizalom szükségletének hiánya. Ezeknél az alkalmazásoknál nincs szükség arra, hogy bíznunk kelljen egy harmadik félben, egy köztes szervezetre. A hálózat résztvevői egy előre meghatározott szabályrendszer szerint léphetnek interakcióba egymással. A második az engedélyek hiánya. Ez azt jelenti, hogy bárki használhatja a rendszert, és nem egy szerv dönti ezt el. A harmadik, a nyitottság. Legyen minden webalkalmazás nyílt forráskódú, hogy mindenki pontos képet kaphasson róluk, hátsó szándékokat ezzel kiszűrve.

A Web3 fő célja, a decentralizáltság, ami alapvetően egy nehéz fogalom, mivel többféle decentralizáltságról beszélhetünk. Az, hogy egy szolgáltatás centralizált-e vagy sem, azt három szempontból tudjuk megvizsgálni [4]. Első, az architektúrai (de)centralizáltság, ennél azt kell figyelembe venni, hogy hány fizikai számítógépből áll a rendszer, és ezek közül hánynak a hiányát tudja tolerálni, hánynak a hiányával tud tovább funkcionálni a szolgáltatás. A második, az a politikai (de)centralizáltság. Itt azt kell figyelembe venni, hogy hány egyénnek, vagy szervezetnek van jogosultsága hozzáférni a hoszt számítógéphez/számítógépekhez. A harmadik szempont, a logikai (de)centralizáció: az alkalmazás struktúrája egy nagy monolitikus rendszerre hasonlít, vagy inkább több részből álló szolgáltatások sokaságára. Egy másik megközelítés ehhez a kérdéshez, hogyha a rendszert félbe vágjuk, beleértve a szolgáltatókat és a felhasználókat, tud-e a

rendszernek mindkét fele hibátlanul működni, mint egyedülálló rendszer. Ezeket két táblázatba helyezve tudjuk ábrázolni:



4. ábra: Centralizáltság vizsgáló táblázat

A tradicionális vállalatok minden szempontból centralizáltak: egy vezérigazgatóval rendelkezik, egy irodából igazgatják, és általában nem tartja meg működőképességét, ha kettévágják. A nyelvek logikailag decentralizáltak: nincs egy központi szervezet, aminek léteznie kell, hogy egy nyelv is létezzen, és a nyelvi szabályok (nyelvtan) nem egy ember által volt készítve, és nincs irányítva. Részben kivétel ez alól az eszperantó nyelv, mivel azt Ludwig Zamenhof egyedül készítette. A blokkláncok politikailag decentralizáltak, mivel senki sem irányítja őket, és architekturálisan, mivel egy blokklánc infrastruktúrájában több számítógép elvesztése mellett is tud funkcionálni, viszont logikailag centralizáltak, mivel egy előre meghatározott szabályrendszer szerint működnek, és egy egységes virtuális számítógépként funkcionálnak. A BitTorrent egy decentralizált fájlmegosztó alkalmazás, ami logikailag teljesen decentralizált.

Felmerülhet az a kérdés, hogy miért jó az, ha egy decentralizált irányba halad az internet. Első sorban biztonságosabb, és nagyobb az elérhetőség. Természetesen nem lehetetlen támadni a decentralizált rendszereket, viszont nagy mértékben drágább, és erőforrás igényesebb egy támadónak egy decentralizált hálózatot támadnia, mivel nincs egy pont, amit hogyha sikeresen lebénítanak, megbénulna az egész rendszer. Ennél fogva jobb az elérhetőség is, mivel hogyha egy hagyományos rendszerrel hasonlítjuk össze, ott lehet, hogy a szerver épületében, ha áramszünet van, a szolgáltatás nem lesz elérhető, míg egy decentralizált hálózat esetén, ez akár több millió különböző rendszer áramszünetének esetében sem állna fenn. Ezen felül a hálózaton belüli kártékony tevékenység is nehezebb, és a cenzúrától is véd. Sokkal költséghatékonyabb, mivel a felhasználó közvetlenül a szolgáltatásért fizet, és nincs egy harmadik fél, aki levonna szolgáltatási díjakat. Adatvédelem szempontjából is nagyon nagy előrelépés, mivel a saját adataink a mi kezünkben vannak, és nem tudja egy harmadik fél eladni a saját profitjára.

2.3. Blokklánc

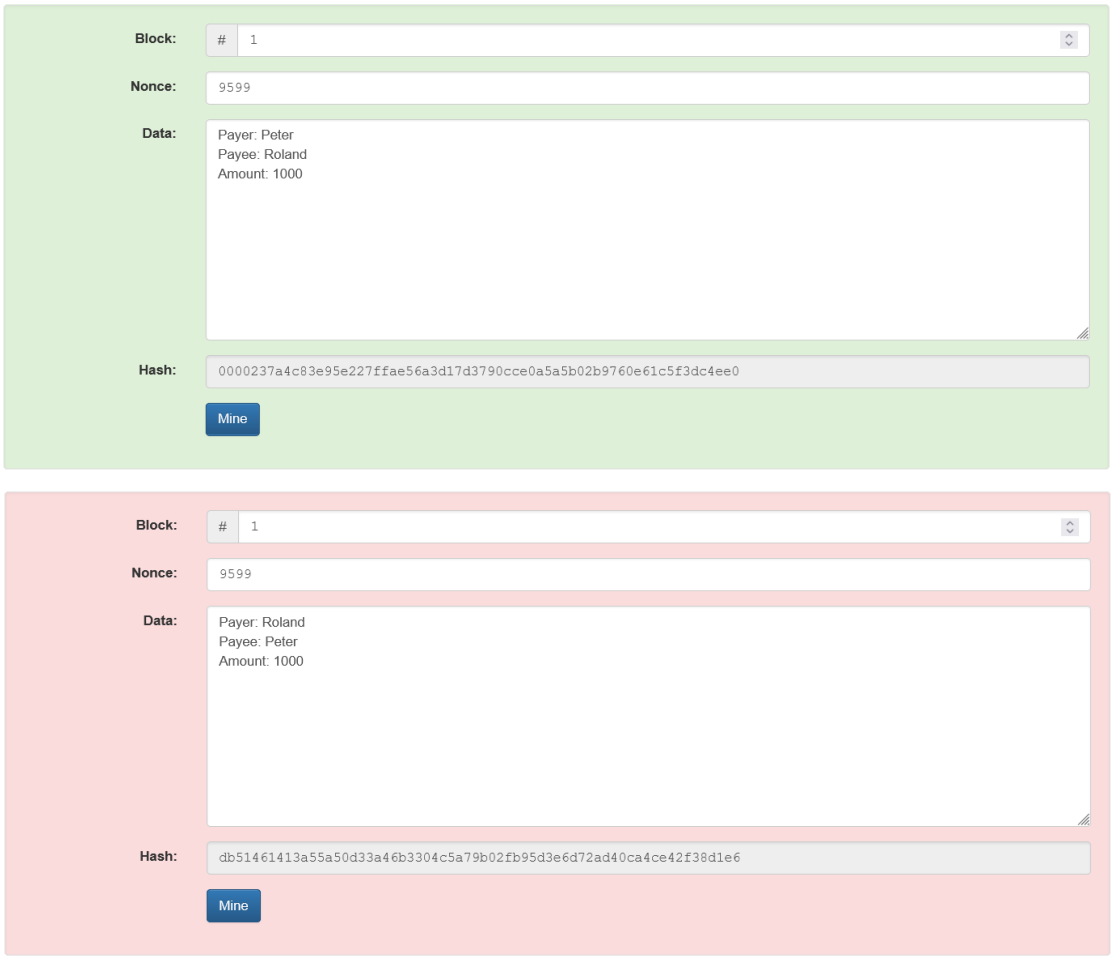
A blokklánc egy láncolt listához nagyon hasonló adatszerkezet, amiben leggyakrabban tranzakciókat tárolnak. Ahhoz, hogy a blokkláncot megértsük, először tisztában kell lennünk néhány egyéb fogalommal, ezeknek a bemutatására Anders Brownworth alkalmazását fogom használni. Az első fogalom a hasítófüggvény, példaként a SHA256 kriptográfiai hasítófüggvényt fogom alapul venni. Ez egy olyan eljárás, ami bármilyen bemenetre, egy 256 bites, 64 karakter hosszúságú hexadecimális karaktersorozatot fog visszaadni. A titkosítási algoritmusok közül a SHA256 az egyik legelterjedtebb. Egy fontos megjegyzendő működési mechanizmusa, hogy a bemenet minimális változtatásának esetében, a kimenet teljes mértékben változik. Példa:

Bemenet	Hash (a SHA256 függvény kimenete)
példa	5ddd1c1121ca5d3a113489522a915c74ebbbabd095d85f69e452a10e2e162b87
pélDa	979bb550d21f7a5219d4dcc952fdf34167ef5e2cbc77b6d236b8a57cf0a810bd
helló	8418d62930f726a3328cac8372dc1c82b20a093dc4f3c0594c04e37d6409724c

2. Táblázat SHA256 függvény példa

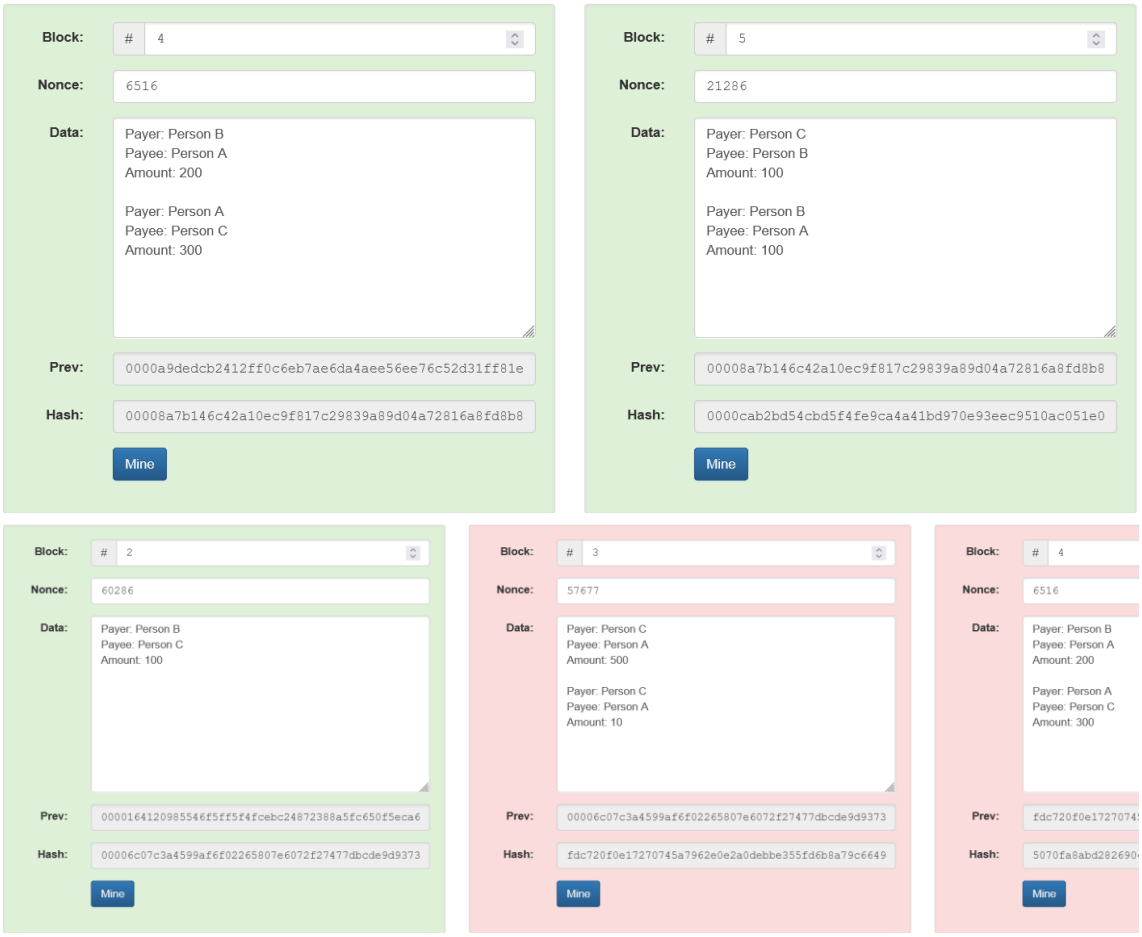
Fontos megjegyezni, hogy a SHA256 egy irányú függvény, így visszafejteni nem lehet.

A hasítófüggvény ismeretében, a blokk a következő fogalom, amit meg kell ismerni. Egy blokk lényegében egy objektum, ami több dolgot is tartalmaz magában. A fontosabb adatok, amiket tartalmaz, a blokk sorszáma (ez azt jelenti, hogy hányadik blokk a blokkláncban), egy úgy nevezett „nonce” érték, ami a hash érték számításánál jön majd képbe, maga az adat, amit a blokk tárol (gyakran tranzakciók listája), és a blokkhoz tartozó hash érték, amit jelen esetben a SHA256 hasítófüggvény segítségével adunk meg. Egy blokk hash értékének, bizonyos mennyiségű nullával kell kezdődjen, hogy a blokk érvényes legyen. Ennek a kiszámításánál jön képbe a nonce érték. A blokkban tárolt adat és egy bizonyos nonce érték (egy szám, jelen példa esetén pár ezres nagyságrendtől pár százezres nagyságrendig tartó intervallumon belül) együtt átfuttatva a hasítófüggvényen, egy szükséges mennyiségű nullával kezdődő hash értéket fog eredményezni. A példában ez 4 darab nulla, valós blokkláncoknál ez egy dinamikusan változó darabszám. Ha a nonce érték és az adatok hash-elt értéke megfelelő, akkor érvényes a blokk. Ha változtatunk akár az adaton, akár a nonce értéken akár egy karaktert is, a hash érték nem lesz megfelelő, így a blokk érvénytelen lesz. Ezt a keresett értéket akkor kell kiszámolni, amikor „megtelik a blokk”. Minden blokklánc esetén van egy fix blokkméret meghatározva, ez lehet tranzakciók száma, vagy maximum fájlméret határ.



5. ábra: Egy érvényes, és egy érvénytelen blokk

Hogyha ezeket a blokkokat egy láncolt lista szerkezetbe rendezzük, akkor kapunk egy blokkláncot. Viszont lényegében egy visszafele láncolt listát fogunk csinálni, mivel a blokkláncnál nem a következő elemre mutatunk a lista elemeknél, hanem az előzőre, így az eddig blokkban tárolt adatok mellé fel kell vegyünk még egy „előző elem hash értéke” adatot is (példákban „prev” néven, az első blokk esetén ez 64 darab nulla). Ahhoz, hogy érvényes legyen a blokklánc, minden blokknak benne érvényesnek kell lennie. Hogyha van öt blokkunk a láncon, és a harmadikban megváltoztatjuk az adatokat, akkor az a blokk mindaddig érvénytelen lesz, ameddig nem számoljuk ki újra a hozzátartozó nonce és hash értékeket. Mivel ezután meg fog változni a hash értéke, a következő blokkban is meg kell hogy változtassuk a „prev” hash értéket. Egy blokklánc esetén, a blokkok hash értéke már az előző hash értékétől is függ, így a következőknél is újra ki kell ezt számolni. Ennek a számolásnak a folyamatát nevezik bányászásnak.

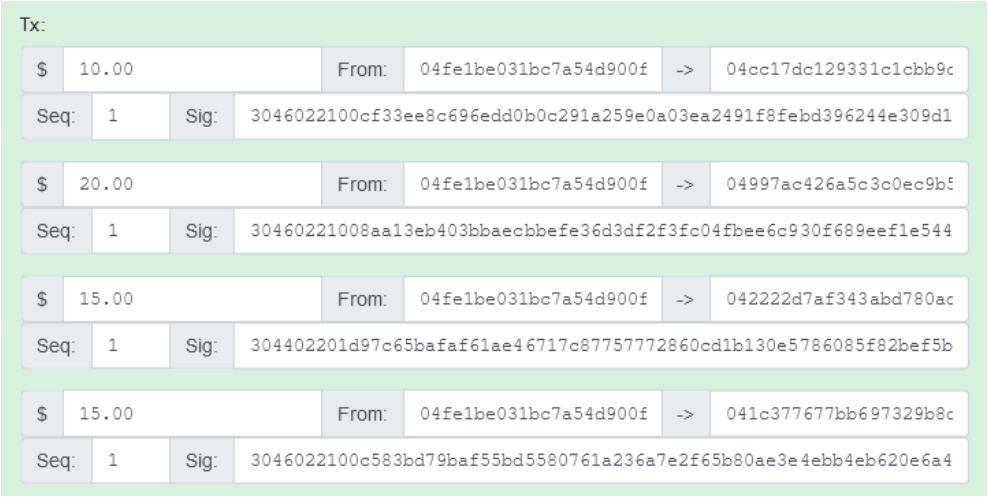


6. ábra: Egy érvényes és egy érvénytelen blokklánc

Jelenleg ezeknek az ismeretében, ez csak egy túlbonyolított láncolt lista. Az igazi előnye, erőssége ennek akkor jön elő, ha decentralizáljuk, és több felhasználó között szétosztjuk a blokkláncot. Tegyük fel, hogy van 10 emberünk, akinél ott van ez a blokklánc. Ha egy blokk megtelik, mindenki elkezdi kibányászni (azaz kitalálni, hogy melyik nonce értékkel együtt adja ki az elvárt mennyiségű nullával kezdődő hash értéket). Az első ember, aki ezt megtalálja, kap egy kis mennyiségű jutalmat, ezt a legtöbb rendszer a tranzakciók költségeiből vonja le, és adja a sikeres felhasználónak. Ezután az összes felhasználó le ellenőrzi, hogy az a nonce érték és az az adat, tényleg a kellő hash értéket adja-e ki az adott blokkban, és hogy ugyanazokkal az adatokkal jön-e ki ez az érték, mint amivel ők dolgoztak. Ez egy igen gyors folyamat, így leellenőrizni a sikerességet nem erőforrásigényes. Ha valaki rossz szándékkal változtat bármely blokk adatain, azt csak úgy teheti meg, ha a mostani példában szereplő 10 felhasználóból a többséget ráveszi, hogy tegyen ugyanígy, mivel minden blokk után ellenőrzik, hogy egyforma blokklánccal dolgoznak-e, és azt a blokkot fogják konszenzusosan elfogadni, ami a felhasználók legalább ötvenegy százalékánál szerepel. Tehát csak akkor lehet átverni a rendszert, hogyha az erőforrások több mint fele az adott felhasználó irányítása alatt van. Ez nagyobb blokkláncoknál egy rendkívüli mód nehéz feladat, mivel több millió számítógépnyi

erőforrásról is szó lehet. Továbbá hogyha régebbi blokkon szeretne változtatni valaki, akkor az összes azóta megfejtett blokkon is újra el kell végeznie a hash kiszámításának folyamatát, ami hatalmas mértékben növeli az erőforrás igényeket.

További biztonsági tényező, hogy egy valós blokklánc esetében az adatok között nem az szerepel, hogy „Péter küld Rolandnak 1000 pénzt”, hanem egy egyedi azonosító. Egyes felhasználók egyenlege úgynevezett pénztárcákon van tárolva. Egy pénztárcának 3 jelentős tartalma van: egy nyilvános kulcs, ez a pénztárca címe, egy privát kulcs, ez a pénztárca jelszója, és egy egyenleg. Egy tranzakciónál azt látni, hogy egy pénztárca cím küld egy másik pénztárca címnek adott mennyiségű pénzt, így egyszerre nyitott és publikus a rendszer, mégis anonim. Továbbá, hogy biztos legyen, hogy ezt az összeget tényleg a küldő személy küldte, ezt egy digitális aláírással kell biztosítani. A digitális aláírás egy aszimmetrikus kulcsú titkosításon alapszik. A tranzakció adatait aláírja a küldő fél a privát kulcsának használatával, majd az aláírás csatolva lesz a tranzakcióhoz a blokkban. Ezt a nyilvános kulcsával vissza lehet fejteni, így ellenőrizve, hogy tényleg ő indította a tranzakciót. Ha egy kártékony szándékú felhasználó változtatna például a küldött pénzösszegen, akkor az aláírást visszafejtve rájövünk, hogy ez nem egy érvényes tranzakció. Az aláírást hamisítani nem lehet, csak akkor, ha a felhasználó privát kulcsának az integritása sérül, és más kezébe kerül. Ha korábbi blokkon szeretne változtatni valaki, akkor azon felül, hogy a tranzakció érvénytelen lesz, még a blokkok hash értéke is megváltozik, és újra kellene bányászni őket, így tovább nehezítve a kártékony felhasználó dolgát.



7. ábra: Egy érvényes blokk, aláírt tranzakciókkal

2.4. Kriptovaluta

A kriptovaluták az elmúlt évtizedben folyamatosan egyre elterjedtebbek lettek, az elmúlt évben pedig eddig nem látott népszerűségnek örvendenek. Ez két fő indoknak tudható be: a mögötte lévő technológia, és az értékük. Vannak olyanok, akik azért foglalkoznak a kriptovalutákkal, mert érdekli őket és hisznek a mögötte rejlő technológiában. Jelenleg több mint tizenháromezer különböző kriptovaluta létezik, és ez napról napra bővül. Ezek mind decentralizált blokkláncokon vannak, saját natív fizetőeszközükkel (maga a kriptovaluta), és sokszor széles körű használattal. Ezeket a valutákat legtöbbször fiat pénznemből (tehát olyan pénzből, amit az állam, vagy központi bankok bocsájtanak ki) vásároljuk, ezért egy árfolyamuk is van, és ez a másik indok, ami miatt rengetegen foglalkoznak velük: manapság a kriptovaluta egy befektetési forma. A legnagyobb kriptovaluta, és egyben az első is, a Bitcoin. Ennek a fiat pénzbeli értéke 2016-ban nagyjából 1000 dolláros csúcsot ért el, 2021-re pedig már a 65 ezer dollárt is átlépte egy bizonyos ideig [5]. A szakdolgozatomban a jelenlegi két legnagyobb piaci kapitalizációval rendelkező kriptovalutákat, a Bitcoint, és az Ethereumot fogom mélyebben megvizsgálni.

2.4.1. Bitcoin

A Bitcoin az első kriptovaluta, amit egy Satoshi Nakamoto nevű ember (vagy szervezet, az identitása ezen a néven kívül a mai napig ismeretlen) készített. Az úgynevezett „whitepaper”-jét (fehér könyvét) 2008-ban adta ki, ez lényegében a Bitcoin működésének leírása volt. „Egy peer-to-peer elektronikus pénznem, ami lehetővé tesz online fizetéseket egyenesen egyik féltől a másiknak, egy pénzügyi intézet bevonása nélkül.”[6]. A Bitcoin blokklánc egy decentralizált főkönyv, amiben a tranzakciók blokkokba vannak szervezve, és kriptografikusan biztosítva van a sértetlenségük.

Egyediségét és fontosságát az adja, hogy az első volt a sok közül, és így belőle született egy egész iparág. A blokklánc natív fizetőeszköze a BTC (azaz Bitcoin), amit a hálózaton keresztül globálisan, szinte azonnal, minimális kezelési költséggel lehet utalni. A blokkok bányászatáért Bitcoin kriptovaluta jár jutalomként. Egy blokkot általában több személy bányászik ki egyszerre, ezek a személyek úgynevezett „pool”-okba vannak szerveződve, így megsokszorozva az erőforrásaikat, ezzel együtt az esélyeiket, hogy elsőként számítsák ki egy blokk megfelelő hash értékéhez tartozó nonce, adat kombinációt. Régebben videókártyák segítségével, manapság erre kifejlesztett dedikált bányászgépekkel dolgoznak. Ezt a módszert „proof of work”, azaz munka bizonyítéka módszernek nevezik. A kibányászható Bitcoin mennyisége véges, az utolsó kibányászott valutát 2140 februárjában fogják megszerezni, ekkor 21 millió Bitcoin lesz elérhető világszerte, ennél több sose lesz. Jelenleg több mint 19 millió kibányászott Bitcoin van. Azért van már közel a kilencven százaléka kibányászva a kriptovalutáknak (annak ellenére, hogy több mint száz év még ameddig az utolsót is kibányásszák), mert 4 évente

a jutalmat minden kibányászott blokkért megfelezik. Az úgy nevezett blokk jutalom eleinte 50 BTC volt, ez azóta lecsökkent 6.25 BTC-re.

A Bitcoin blokkláncon csak tranzakciók vannak tárolva, és a blokklánccal kompatibilis pénztárcákon (például az Armory Secure Wallet) a felhasználók egyenlege. Léteznek olyan blokkláncok, amik a Bitcoinra épülnek, kiegészítve a funkcionalitásait. Például a Stacks egy olyan blokklánc, ami a Bitcoinra épült (azaz kompatibilis vele), és lehetővé teszi, hogy okos szerződéseket írjunk, amik BTC-t kezelnek. Az okos szerződéseket az Ethereum bekezdés alatt részletesebben kifejtem.

Manapság számos helyen (mind online, mind offline) lehet már Bitcoinnal fizetni, legtöbbször egy beszkennelhető QR kód segítségével.

2.4.2. Ethereum

Az Ethereum blokklánc fehér könyve 2013-ban került publikálásra Vitalik Buterin orosz-kanadai programozó által, a blokklánc maga pedig 2015-ben indult el. Lényege, hogy ahelyett, hogy csak tranzakciók lennének tárolva a blokkláncok, konkrét logikát is lehessen tárolni rajta, okos szerződések formájában. Egy okos szerződést valahogy úgy kell elképzelni, mint egy ital automatát: meg van adva egy feltétel, például „HA bedobnak 300 forintot, ÉS megnyomják az ötös gombot”, ami hogyha teljesül, akkor az előre definiált esemény megtörténik: „AKKOR kiad az automata egy üdítőt”. Ennek segítségével bárki élesbe tud helyezni decentralizált alkalmazásokat, amik ameddig a blokklánc létezik, futni fognak. Ezek az okos szerződések, az Ethereum saját programozási nyelvében, a Solidity-ben íródnak. Emellett továbbra is vezet a blokklánc főkönyv szerűen tranzakciókat a hálózat natív fizetőeszközén az Ether-en (ETH).

Egy fontos koncepció az Ethereum-mal kapcsolatban, az EVM, azaz Ethereum Virtual Machine (Ethereum Virtuális Gép). A hálózatot bányászók erőforrásaiból egy virtuális szuperszámítógép fut, az EVM. Ezen a virtuális gépen fut minden okos szerződés, ez gondoskodik arról, hogy ténylegesen megtörténjen, amit a szerződés előír. Az EVM-et csomópontokon keresztül érjük el, a bányászok lényegében ezekként a csomópontokként működnek.

A Bitcoinnal ellentétben nincs véges számú Ether. 2021-ben nagyjából 120 millió ETH van forgalomban, és elérte már a 4000 dollár / Ether árat, így a második legnagyobb piaci kapitalizációjú kriptovaluta. A Bitcoinhoz hasonlóan rengeteg helyen lehet használni már fizetőeszközként, beleértve centralizált szolgáltatásokat is. Hasonlóan a Bitcoinhoz, az Ethereumra is számos másik blokklánc épül, viszont nagy mértékben több itt ezeknek a száma, mivel úgy volt tervezte az Ethereum, hogy könnyű legyen rá építeni új technológiákat. A kriptovaluta fehér könyve szerint, akár már 20 sor kód segítségével is lehet új kriptovalutát kiépíteni az Ethereumból. [7]

2.5. Solidity

A Solidity egy objektum orientált, magas szintű programozási nyelv, okos szerződések írására. A nyelv erősen típusos, többek között támogatja a leszármaztatást, könyvtárakat, és felhasználók által definiált típusokat. A Solidity segítségével létre lehet hozni olyan szerződéseket, amik képesek lebonyolítani szavazásokat vagy aukciókat. A szerződéseket megírásuk után nem emberek által vannak kezelve, így nem manipulálhatóak. [8]

Egy Solidity fájl kiterjesztése „.sol”. A fájl tetején a „pragma” kulcsszó segítségével meg kell határozzuk a fordítóprogramnak, hogy melyik solidity verziót használja. Egy okos szerződés létrehozásához a „contract” kulcsszót kell megadnunk, majd a szerződés nevét. A szerződés részleteit kapcsos zárójelek között tudjuk megadni. Egy szerződés lényegében megegyezik egy osztállyal. A szerződésen belül létre tudunk hozni változókat, ezeket Solidity esetében állapotoknak hívjuk. A programozási nyelvnek van egy különleges adattípusa az „address”. Ezzel az adattípussal tudunk tárolni 20 bájtos Ethereum pénztárca címeket. A „struct” kulcsszóval tudunk létrehozni komplexebb, saját adattípusokat. Metódusokat a „function” kulcsszóval tudunk létrehozni, ezek lényegében ugyanúgy funkcionálnak, mint a más programozási nyelvekben megszokott metódusok. Konstruktort a szerződéshez a „constructor” kulcsszóval tudunk létrehozni, ez egyszer fut le, amikor a szerződés létrejön.

MyContract.sol

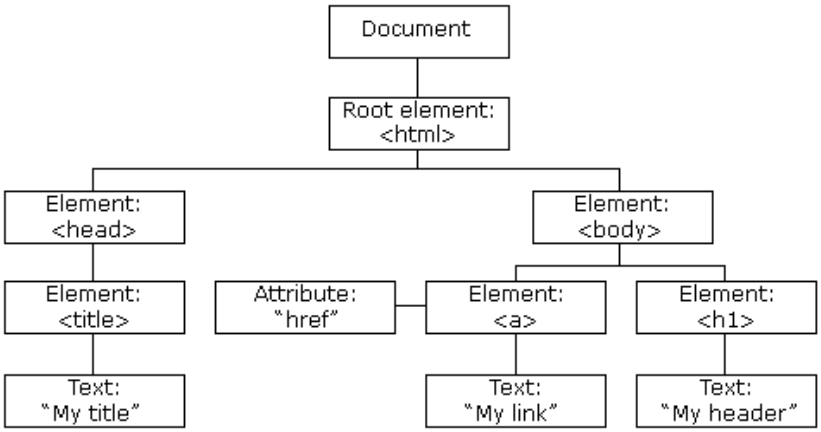
```
1   //SPDX-License-Identifier: MIT
2   pragma solidity ^0.8.10;
3
4   contract MyContract{
5       string public hello = "Hello Solidity world!";
6       address payable seller;
7       address payable buyer;
8
9       struct Order {
10          string description;
11          bool completed;
12      }
13
14      function confirmOrder() public {
15          buyer = payable(msg.sender);
16      }
17
18      constructor(string memory _text){
19          hello = _text;
20      }
21  }
```

8. ábra: Solidity példa kód

2.6. JavaScript

A JavaScript egy magas szintű, gyengén típusos, futási időben fordított programozási nyelv. Mivel magas szintű a nyelv, ezért nem kell foglalkoznunk erőforrások menedzselésével, és erre az is rásegít, hogy a JavaScript motorban automatikus memória felszabadítás van beépítve. Gyengén típusos, dinamikus a nyelv, így a változóknak nem kell a típusát megjelölni értékadáskor, ez csak fordításkor dől el, viszont lehetséges később változtatni a típusán. Ha változóknak típust szeretnénk adni, a TypeScript nyújt erre megoldást, erről az Angular fejezetben részletesebben írok. A fordítása futási időben történik, és folyamatosan optimalizálja magát futás közben, így gyorsabb és effektívebb futási időt el tud érni. A nyelv objektum orientáltnak mondható, mivel lehetséges benne osztályokat definiálni, viszont ezt a prototípus tervezési minta megvalósításával éri el. Minden objektum az ős prototípus objektum leszármazottja. Ennek egy olyan előnye van, hogy tudunk olyan objektumokat is létrehozni, aminek az osztályát nem definiáltuk előre, de emellett tudunk új osztályokat is létrehozni, és azokat példányosítani.

A DOM egy fontos fogalom a nyelvvel kapcsolatban, mivel ez az összekötő pont a HTML dokumentum, és a JavaScript között. A DOM az angol document object model kifejezés rövidítése, és a dokumentum részeit modellezi, egy fa adatszerkezetbe. Ennek segítségével tudja a JavaScript manipulálni a HTML dokumentum adott részeit.

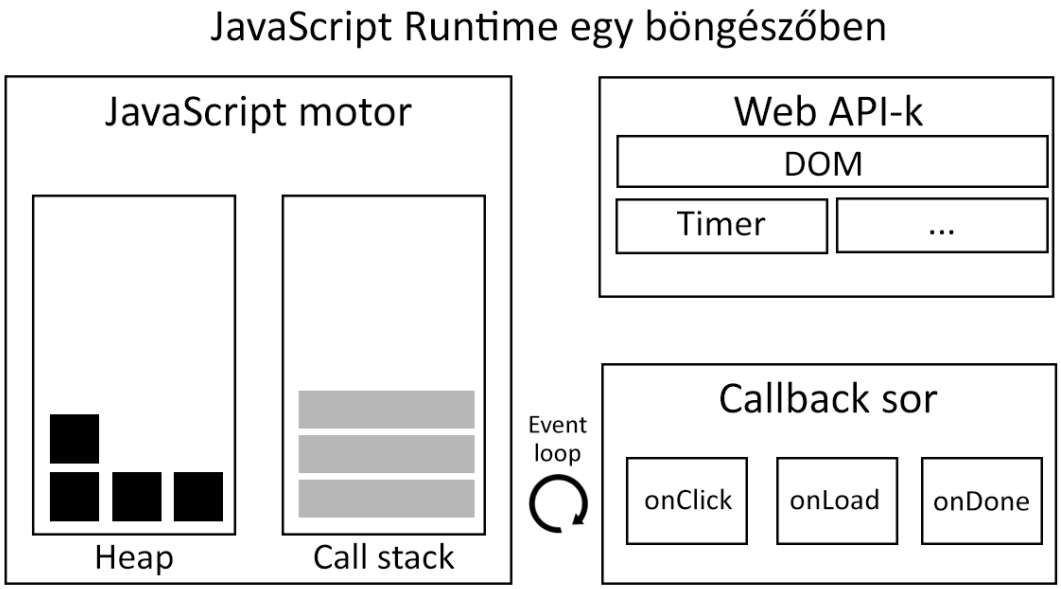


9. ábra: HTML DOM fa

A kód futtatási környezete általában a böngésző, és a futtatásért a JavaScript Runtime felel. Ez egy konténer, ami tartalmazza a szükséges dolgokat a kód futtatásához, és ezt általában a böngésző biztosítja. Ennek a szíve a JavaScript motor, ez hajtja végre ténylegesen a kódot. A motorban van a call stack (hívássorozat), ebben tárolódnak el a lefutásra váró kód blokkok, és a memória terület, ahol minden mást tárolunk. A runtime tartalmaz még web API-kat is, ezek bizonyos funkcionalitásokat biztosítanak, de nem részei a JavaScript nyelvnek, csak elérhetőek a segítségével. A runtime része még a callback (visszahívási) sor, itt tárolódnak a lefutásra kész kódrészek. Bár a futási környezetet általában a böngésző biztosítja, ez megvalósítható azon kívül is, erre jelenleg

a legelterjedtebb eszköz a Node.js, ami egy backend szerver oldali JavaScript runtime, a böngésző által biztosított verzióhoz képest annyi különbséggel, hogy nem tartalmaz Web API-kat.

A JavaScript egy szálon dolgozik, így egyszerre csak egy feladatot tudna ellátni, viszont egy úgynevezett event-loop segítségével (esemény hurok) lényegében képes párhuzamosításra. Ha egy kód rész hosszabb ideig tartana, hogy lefusson, nem tartja fel a programnak a futását, helyette bekerül egy sorba a háttérben, amit az event-loop folyamatosan ellenőriz, hogy tartalmaz-e kész feladatot, és ha igen, akkor a call stack-be visszahelyezi, és lefuttatja.



10. ábra: JavaScript runtime vizualizálása

Az alap JavaScript kód önmagában is sok dologra képes, viszont elérhető számos publikus könyvtár hozzá. Egy könyvtár, előre megírt kódok gyűjteménye, amivel lényegesen ki tudjuk bővíteni az elérhető funkcionalitásokat. Léteznek még keretrendszerek, az egész kódunkat képesek meghatározni, és számos új kód lehetőséget behozni. Ilyen keretrendszer például az Angular.

2.6.1. Angular

Az Angular egy TypeScript frontend keretrendszer. A TypeScript lényegében egy erősen típusos kód, szintaxisban szinte mindenhol megegyezik a JavaScript kóddal, és fordításkor először átfordul JavaScript kóddá, hogy a böngészők is tudják értelmezni. A keretrendszert a Google-nél készítették. Elődje az AngularJS ami 2010-ben indult, viszont a jelenlegi TypeScript alapú verzió 2016 óta létezik.

A keretrendszer segítségével van lehetőségünk feltételes megjelenítésre. Ezt a HTML kódon belül írt *ngIf=”feltétel” használatával tudjuk megadni. Iterációra is van lehetőségünk, az *ngFor=”let item in items” segítségével. Az Angular egy komponens alapú keretrendszer, ami egy könnyen átlátható kódot, és skálázható alkalmazást fog jelenteni. Egy TypeScript osztályból tudunk komponenst csinálni a @Component({}) dekorátorral. Innentől, az osztály tulajdonságai állapotokká válnak, és ha egy állapot változik, a felhasználói interfészen a hozzá tartozó megjelenített adat is újra töltődik. Ezt adatkötés segítségével tudjuk megvalósítani, amit a HTML kódban dupla kapcsos zárójelek között megadott változó, vagy állapot nevekkel tudunk megadni. Ezen felül, metódusokat tudunk eseményekhez kapcsolni. Például, ha egy metódust szeretnénk végrehajtani egy gombra kattintásnál azt a gomb HTML kódjába beillesztett (click)=”metódus()” kóddal tudjuk megadni.

TS demo-comp.component.ts U ×

src > app > demo-comp > TS demo-comp.component.ts > ...

```
1   import { Component } from '@angular/core';  110.4K (gzipped: 35K)
2
3   @Component({
4     selector: 'app-demo-comp',
5     templateUrl: './demo-comp.component.html',
6     styleUrls: ['./demo-comp.component.css']
7   })
8   export class DemoCompComponent {
9
10    fruitBasket : Array<string> = ['Banana', 'Apple']
11
12    addNewFruit(){
13      this.fruitBasket.push(this.getRandomFruit())
14    }
15
16    getRandomFruit(){
17      const allFruits = ['Apple', 'Strawberry', 'Banana', 'Grape', 'Melon']
18      return allFruits[Math.floor(Math.random() * allFruits.length)]
19    }
20  }
21
```

11. ábra: Angular TypeScript példa kód

12. ábra Angular HTML példa kód

A függőségi injekció tervezési mintát megvalósítja az Angular. Ez kifejezetten előnyös, mivel nagy méretű kód esetén sokkal könnyebben átlátható, tesztelhető, módosítható és bővíthető lesz az alkalmazás. A tervezési minta úgy valósul meg, hogy a komponenseket szolgáltatásokká alakítjuk a @Injectable({}) dekorátorral. Ezután, más osztályoknak a konstruktorába beinjektálhatjuk a szolgáltatást, és így elérjük annak a tulajdonságait, és metódusait anélkül, hogy példányosítottuk volna az osztályt.

2.6.2. Node.js

A Node.js egy JavaScript futási környezet, ami lehetővé teszi azt, hogy böngészőn kívül, szerver oldalon futtathassunk JavaScript kódot. Egy hatalmas előnye a Node.js-nek a versenytársaihoz képest (például ASP vagy PHP), hogy aszinkron. Ha egy feladat érkezik egy webszerverre, például egy fájlnak a feldolgozása, azt általában úgy dolgozza fel a szerver, hogy miután megkapta a fájlt, elkezdi feldolgozni, ha végzett vele, visszaküldi a feldolgozott tartalmat vagy a várt választ, majd készen áll, hogy a következő kérést is feldolgozza. Ezzel szemben a Node.js mivel aszinkron, képes több kérést is fogadni egyszerre, vagy új kéréseket fogadni miközben épp feldolgozik egy fájlt. A Node.js segítségével képesek vagyunk generálni weboldalaknak dinamikus tartalmat, űrlapok adatait begyűjteni, szerver oldalon fájlokkal dolgozni, és adatbázist is szerkeszteni.

Node.js-hez tartoznak beépített modulok. A modulok lényegében olyanok, mint JavaScript esetében a könyvtárak. Modulok használatához a require() metódust kell használjuk. Az egyik alapvető modul Node.js esetében, a http, ennek segítségével tudunk szervert létrehozni (ehhez a http.createServer() metódust kell használjuk) , és adatokat fogadni és küldeni http-n keresztül.

index.js

Backend > index.js > ...

```
1   const http = require('http');
2
3   const server = http.createServer((req, res) => {
4       res.statusCode = 200;
5       res.setHeader('Content-Type', 'text/plain');
6       res.end('Hello nodejs.');
7   });
8
9   server.listen(port, hostname, () => {
10      console.log('Server running at http://localhost:3000/');
11  });
```

13. ábra: Node.Js példa kód

A Node.js az egyik legnépszerűbb szerver oldali programozási környezet, mivel lehetővé teszi, hogy webfejlesztés során kliens oldalon, és szerver oldalon is ugyanazt a programozási nyelvet (JavaScript) használjuk.

2.6.3. Web3.js

A web3.js egy JavaScript könyvtár gyűjtemény, ami lehetővé teszi az interakciót Ethereum csomópontokkal. Ezeket az interakciókat általában http-n történő kommunikációval tesszük meg. A web3.js aszinkron működéséért, úgynevezett „promiEvent” -ek felelnek. Mivel az Ethereum blokklánc esetén egy interakció (például egy tranzakció) több eseményt is kivált, több metódusnak is a visszatérési értéke egy promiEvent. Ezeknek a segítségével, az interakciók különböző fázisaiban, különböző függvényeket tudunk végrehajtani. Új kulcsszavak, amik a promiEvent-ekhez tartoznak, az on, once és off metódusok.

A Node.js-hez hasonlóan, modulokkal dolgozik a web3.js, amiket a require() metódussal lehet elérni. Ezek közül a legfontosabb modul, a web3 modul. Ezen keresztül érhetünk el minden metódust, amivel interakcióba tudunk lépni az Ethereum blokklánccal.

JS web3demo.js ×

Backend > JS web3demo.js > ...

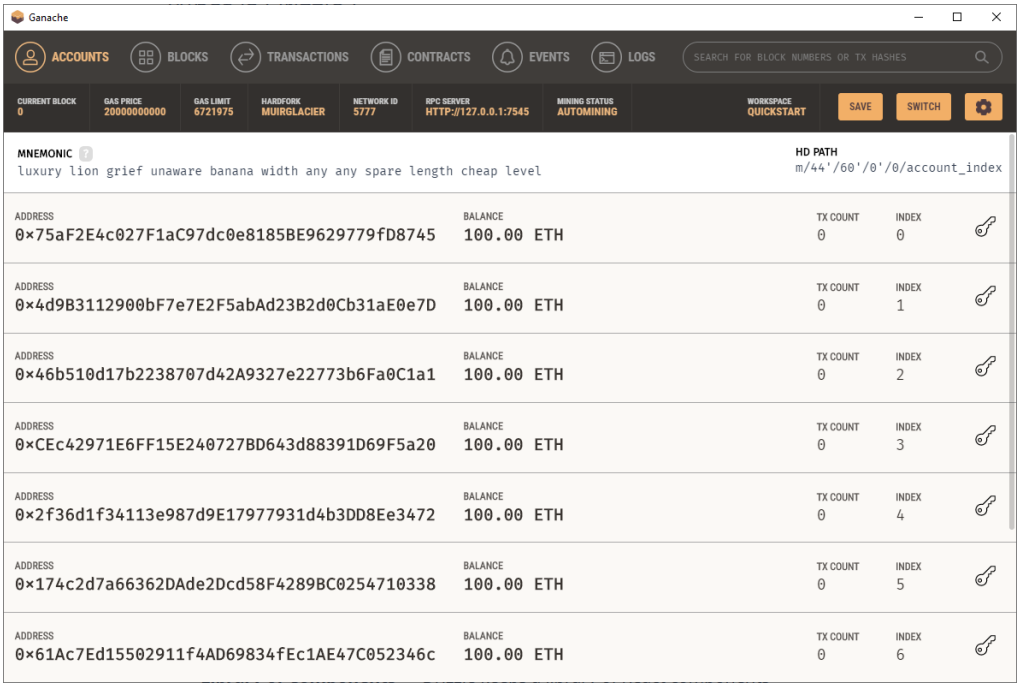
```
1   const web3 = require('web3');
2
3   web3.eth.sendTransaction({from: '0x123...', data: '0x987...'})
4       .once('sending', function(payload){ /* Do something... */ })
5       .once('sent', function(payload){ /* Do something... */ })
6       .once('transactionHash', function(hash){ /* Do something... */ })
7       .once('receipt', function(receipt){ /* Do something... */ })
8       .on('confirmation', function(confNumber, receipt, latestBlockHash){ /*
9       .on('error', function(error){ /* Do something... */ })
10      .then(function(receipt){ /* Finally */ });
```

14. ábra: Web3.Js példa kód

2.7. Truffle Suite

A Truffle Suite három fő eszközt foglal magában. Az első a Truffle, ami egy JavaScript könyvtár gyűjtemény, fejlesztői környezet, és teszt keretrendszer okos szerződések fejlesztéséhez. Fő célja, hogy minél egyszerűbbé tegye az okos szerződések fejlesztését. Segít a szerződések menedzselésében és a blokkláncra kihelyezésükben. Lehetővé teszi az automatizált tesztelését az okos szerződéseknek, ami azért nagyon fontos, mivel blokklánc fejlesztés esetén az alapos tesztelés nélkülözhetetlen, mivel a kihelyezett kódot nem lehet utólag szerkeszteni. A Truffle továbbá lehetővé teszi a blokkláncra kihelyezés, és a migráció automatizálását, ezzel is megkönnyítve a fejlesztési folyamatot. A Truffle segít továbbá a decentralizált hálózat menedzselésében is, és implementál egy interaktív konzolt is, amivel az összes Truffle utasítást könnyen tudjuk használni.

A második a Ganache, ami egy különálló program, és Ethereum blokklánc szimulálására alkalmas. Ennek az a hatalmas előnye, hogy amíg a fő blokkláncra kihelyezni kódot pénzbe kerül, a teszt blokkláncra pedig időigényes a kihelyezés, a Ganache segítségével lokálisan és biztonságosan tudunk ingyen és azonnal tesztelni okos szerződéseket, egy szimulált helyi Ethereum blokkláncon.

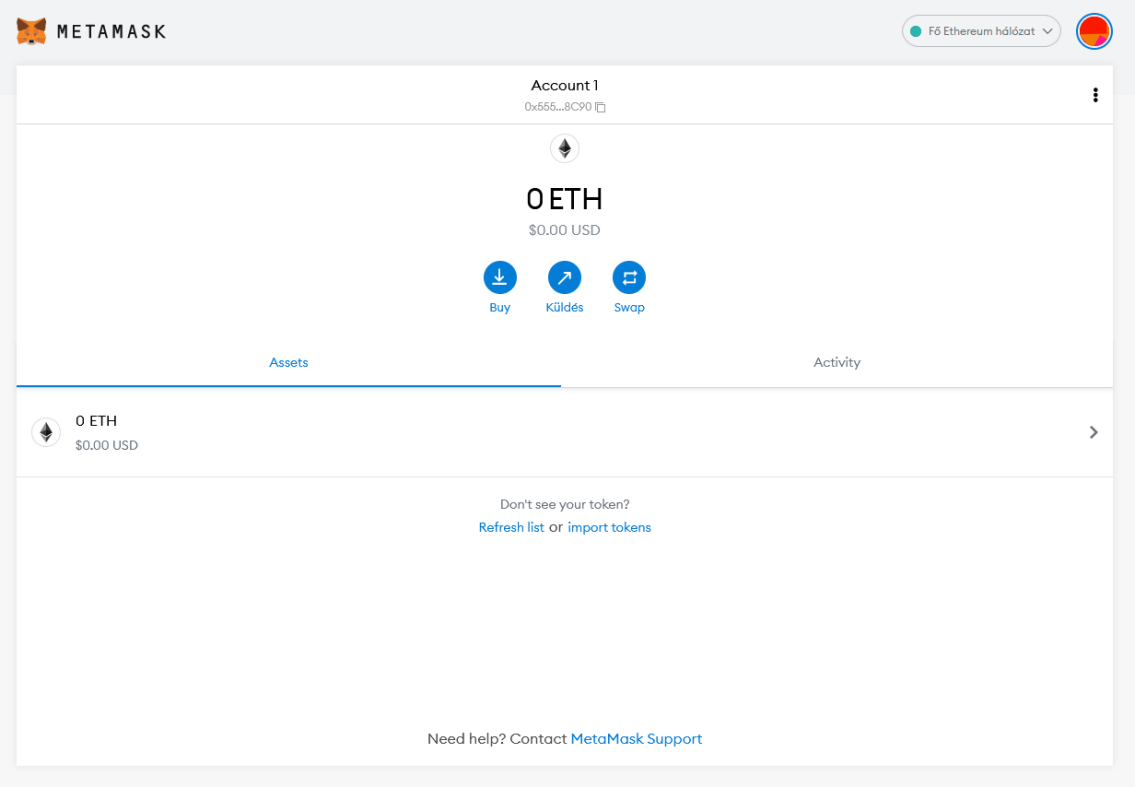


15. ábra: Ganache lokális blokklánc

A harmadik eszköz a Drizzle, ami felhasználói interfész könyvtárak gyűjteménye, elsősorban React keretrendszerhez. Célja, hogy megkönnyítse a web3 frontend fejlesztés menetét. Számos decentralizált alkalmazás funkcionalitást ad hozzá a keretrendszerünkhöz, de a fő előnye, hogy egyszerűen lehet vele élő adatokat megjeleníteni a blokkláncról, és annak felhasználóiról.

2.8. Metamask

A Metamask egy böngésző kiegészítőként feltelepíthető, és mobil alkalmazás. Egy többfunkciós eszköz, elsősorban egy pénztárca Ethereum, és Ethereum alapú kriptovaluták tárolására. Regisztrációhoz kötött, viszont nem a megszokott email cím és jelszó szükséges hozzá, hanem a rendszer által legenerált 12 véletlenszerű angol szóból álló jelmondat, aminek a segítségével a jövőben be tudunk regisztrálni. Az extra védelem érdekében, egy saját jelszót is meg kell adnunk emellé a jelmondat mellé. A jelmondatra csak akkor van szükség, ha új eszközön szeretnénk elérni a Metamask felhasználónkat, a jelszóra pedig minden alkalommal amikor elindítjuk a böngészőnket és először használni szeretnénk az alkalmazást. Ezeken kívül, a Metamask generál nekünk egy nyilvános kulcsot, és egy privát kulcsot. A nyilvános kulcs lesz a pénztárca címe, ez megfelel például egy bankszámla számnak, a privát kulcs pedig a hozzátartozó aláírásnak, amivel biztosítani tudjuk, hogy hozzánk tartozik a pénztárca. Számos decentralizált oldalra lehetséges Metamask pénztárca segítségével beregisztrálni, és bejelentkezni. Ennek előnye például, hogy ha fizetni szeretnénk az oldalon valamiért, tudunk egyből a pénztárcánkon keresztül biztonságosan kriptovalutával.



16. ábra: Metamask pénztárca

2.9. Összegzés

Irodalom kutatásom során megvizsgáltam az internet néhány jelenlegi, és lehetségesen jövőbeli technológiáit. Az internet evolúciójában egy teljesen lehetséges út a decentralizáltság, jelenleg még kezdetlegi fázisban van a web3.0, viszont minden lehetőség adott, hogy ennek az irányába haladjunk. Jelenleg a legtöbb internetes szolgáltatás, ami ingyenes, az adataink eladásából gazdagszik meg. A web3.0 koncepciója, hogy ne más kezében legyenek a személyes adataink és internetes tevékenységünk, hanem a saját kezünkben. Egyet értek ezzel az elképzeléssel, mivel nem szeretném, hogy pénz kérdése legyen, hogy ki férhet hozzá az online tevékenységeimhez. További elgondolás a web3.0-val kapcsolatban, hogy a technológiai óriások helyett, akik manapság is dominálják az online teret, a nyílt forráskódú, közösség által fenntartott és fejlesztett alkalmazások legyenek a középpontban.

A blokklánc technológia sok remek előnnyel jár, elsősorban a biztonság, és az anonimitással. Nincs szükség megbízandó harmadik félre ahhoz, hogy biztonságban legyenek az adataink, helyette bízhatunk a kriptográfiában és a matematikában. A kriptovalutákkal alacsony tranzakciós költséggel, bármikor lehet szinte azonnal kereskedni világszerte, emiatt évről évre egyre elterjedtebbek, mint fizetőeszközök. Elég gyakran kereskednek velük tőzsde-szerűen, profit céljából, ami véleményem szerint azért pozitívum, mert a népszerűségüket lényegesen növelik. Jelenleg sajnos elég gyakran használják a Bitcoint és hasonló kriptovalutákat illegális kereskedelemre az anonimitásuk miatt.

Megvizsgáltam kapcsolódó programozási nyelveket is. Okos szerződés fejlesztésre jelenleg a legelterjedtebb opció a Solidity nyelv, ez annak köszönhető, hogy az Ethereum szerződéseinek programozási nyelve, és az Ethereum a legnagyobb okos szerződés blokklánc. A Solidity egy könnyen tanulható nyelv, megszokott szintaxissal. Az okos szerződések legnagyobb előnye, hogy nyílt forráskódúak, a viselkedésük determinisztikus és nem lehet módosítani őket, így teljesen megbízhatóak. A JavaScript jelenleg a világ legelterjedtebb programozási nyelve webfejlesztéshez, így hatalmas előnye, hogy rengeteg forrás, és ráépülő technológia elérhető hozzá, továbbá hatalmas a kereslet is rá. Ráépülő technológiák közé tartozik a TypeScript és arra pedig az Angular, ami az egyik legnépszerűbb felhasználói interfész keretrendszer manapság. Szerveroldalon a Node.js rendkívül népszerű, és előnyként felsorolható, hogy használata esetén egy webfejlesztőnek általában nincs szüksége más programozási nyelv ismeretére a JavaScripten kívül. Decentralizált alkalmazások fejlesztésénél az okos szerződéseket, és a webes felületet összekötni JavaScript segítségével tudjuk, erre szolgál segítségül a web3.js, ami rengeteg funkciót biztosít a blokklánccal való interakcióhoz. Fejlesztésnél a Truffle Suite megoldásai rendkívüli mód megkönnyíthetik a fejlesztés menetét. A Metamask egy sokoldalú, egyszerű eszköz, szinte kikerülhetetlen web3.0 használatánál.

3. KÖVETELMÉNY SPECIFIKÁCIÓ

Elvárások funkcionalitásokban

Az alkalmazás legyen képes felhasználókat, és hozzájuk tartozó tartalmakat kezelni. Regisztrációra Metamask virtuális pénztárca segítségével legyen lehetőség, de legyen lehetőség ezen felül email cím, és kijelzett név beállítására is. A felhasználók legyenek képesek blogposztokat létrehozni, ezekhez lehessen csatolni képeket is, és más felhasználók tudjanak reagálni a posztokra. Két felhasználó tudja egymást követni ingyen, ezáltal rendszeresen a főoldalán megjelenítve a követett személy posztjait, vagy tudjanak feliratkozni, ETH kriptovalutáért cserébe. A tranzakciók a Metamask pénztárcán keresztül történjenek. A blogposztoknál továbbá opcionális beállításként legyen lehetőség csak feliratkozott felhasználók számára láthatóvá tenni a posztot.

Elvárások a frontend alkalmazással kapcsolatban

A frontend alkalmazásnál szükségünk lesz négy fő felületre: egy regisztráció/bejelentkezés oldalra, egy főoldalra, egy poszt létrehozó és egy felhasználó beállításai oldalra. A bejelentkezés legyen egyszerű egy gomb segítségével induljon el a Metamask autentikáció, majd sikeres bejelentkezés után továbbítson a főoldalra. A főoldalon legyenek láthatóak a követett vagy feliratkozott felhasználók blogposztjai, idő szerint a legfrissebbel legfelül. A főoldalon legyen lehetőség blogposzt írásának kezdésére is. A poszt létrehozó oldal egy formázható szöveges bemeneti mező legyen, csatolmányok hozzáadásának lehetőségével. A felhasználó beállításainál legyen lehetőség email címet és becenevet vagy valós nevet megadni, és törölni a felhasználói fiókunkat.

Elvárások a backend szolgáltatásokkal kapcsolatban

A szerveroldali funkciók törekedjenek a decentralizáltságra. Készüljenek el funkciók okos szerződések keretein belül, vagy használjon az alkalmazás már elkészült publikus okos szerződéseket. Mivel a fő Ethereum blokklánccal interakcióba lépni rendkívül költséges, ezért a szakdolgozatom keretein belül az alkalmazás fusson lokálisan, teszt Ethereum hálózaton, de legyen technikailag alkalmas a fő hálózatra való kihelyezésre is. Legyenek az okos szerződések alaposan tesztelve, mivel a blokkláncra kihelyezett kód utólag nem módosítható, csak ha a módosított verziót újra kihelyezzük, ami ahogy feljebb is említettem, költséges folyamat éles hálózat esetén.

4. TERVEZÉS

4.1. Használt technológiák

Irodalomkutatásom során megvizsgáltam a technológiákat, amiket használni tervezek a fejlesztés során, viszont nem hasonlítottam össze őket egyéb, kompetitív technológiákkal. Fontos megjegyezni, hogy nem volt teljesen objektív a választásom minden kategóriában, mivel személyes érdeklődés és egyetemi források elérhetősége is beleszámított a döntéseimbe.

Blokklánc

A legfontosabb döntés a technológiák választásakor a blokklánc volt. Ez határozza meg a decentralizált lehetőségek nagyrészét, az elérhetőséget, a sebességet, az árat, és még sok mást. A CoinMarketCap alapján, okos szerződések írására jelenleg több mint száz blokklánc biztosít lehetőséget. Ezek közül megvizsgáltam a négy legnagyobb piaci kapitalizációval rendelkező blokkláncot, ezek név szerint az Ethereum, a Binance, a Cardano és az Avalanche láncok. Az alábbi táblázatban szereplő számok, 2021. november 28.-ai adatokat használnak, kerekítve a legközelebbi egész, tízes vagy százas értékhez. [10]

	Ethereum	**Binance**	**Cardano**	**Avalanche**
Okos szerződések programozási nyelve	Solidity	Solidity	Plutus	Solidity
Okos szerződések kezdete	2015	2020	2021	2020
Projektek száma az ökoszisztémában	135	1848	6	113
Piaci kapitalizáció[2]	$509	$102	$53	$25
Kriptovaluta ára	$4294	$611	$1.6	$110
Napi tranzakció szám	1.1 millió	15.3 millió	140 ezer	600 ezer
Átlagos tranzakció költsége	$16	$0.06	$0.4	$0.05
Konszenzus mechanizmus	PoW[3]	PoS[4]	PoS	Avalanche

3. Táblázat: Okos szerződés blokkláncok összehasonlítása

A táblázat alapján a következőket emelném ki. A Solidity messze a legnépszerűbb okos szerződés programozási nyelv, így ez a választás elég egyértelmű volt. Az Ethereum sokkal több ideje létezik, és stabilan népszerű volt mindig is. Forradalmár a technológiában, első volt a sok közül. Előnye, hogy piaci kapitalizációban messze a

[2] Milliárd dollárban
[3] Proof of work: a munka bizonyítéka
[4] Proof of Stake: tétbiztosítás

legnagyobb, az utána következő Binance hálózatnál ötször akkora. Hatalmas hátránya, hogy rendkívül magasak a tranzakciós költségek. Hagyományos bankoknál 16 dolláros tranzakciós költség több milliós tételeknél fordul csak elő, Ethereum esetén pedig a legkisebbtől a legnagyobb tranzakcióig ennyit kell fizetni. Ez a magas összeg a konszenzus mechanizmusának köszönhető. A konszenzus mechanizmus azt jelenti, hogy a decentralizált hálózat tagjai, csomópontjai, ezen protokoll szerint döntik el, hogy egy blokk érvényes-e. A proof of work módszer a hagyományos, amit a Bitcoin, az Ethereum, és sok más bányászott blokklánc használ. Itt a szakdolgozatomban korábban említett hasheléses folyamat segítségével döntik el, hogy mi kerülhet fel a blokkláncra. Ez meglehetősen erőforrásigényes, és lassú. Ezzel szemben a proof of stake nem a munkát jutalmazza, hanem a feltett tétet. Egy adott kriptovaluta tulajdonosa felteheti az általa birtokolt fizetőeszközt arra, hogy ő validálja a tranzakciókat. Minél több kriptovalutával rendelkezik, annál nagyobb szava van az adott blokk elfogadásában. Ha a blokklánc felé ártó szándékkal cselekszik, elveszítheti a tétként biztosított kriptovalutáját, így arra van bíztatva, hogy becsülettel hitelesítse a blokkokat. Az Avalanche hálózat a saját konszenzus mechanizmusát alkalmazza. Ez a protokoll lényegében hasonlít a proof of stake módszerre, viszont ahelyett, hogy minden csomópontot megkérdezne a rendszer egy blokk hitelességéről, csak egy bizonyos (kisebb) mennyiségűt választ ki, és azok alapján dönt. Ha nagy az eltérés a megkérdezettek között, akkor újra megkérdez más csomópontokat, és ezt addig csinálja, ameddig nem ér el egy konszenzus állapotot. Azért hasonlít a proof of stake módszerre, mert itt is azok közül választ a rendszer, akik tétet raktak fel, hogy hitelesíthessenek, és minél nagyobb tétet, annál nagyobb eséllyel lesznek véletlenszerűen kiválasztva. Tranzakció és projektszámban a Binance hálózata vezet toronymagasan, ez annak köszönhető, hogy a Binance alapból egy kriptovaluta kereskedő oldal volt, ráadásul az egyik legnépszerűbb a világon, majd 2020-ban elindították a saját okos szerződés kompatibilis blokkláncukat. A hálózaton rendkívül olcsók a tranzakciók, és akár percek alatt létre lehet hozni, és élesbe lehet helyezni egy új projektet. A választásom többek között azért nem a Binance hálózatára esett, mivel egy nagyobb cég áll az egész mögött, így meglehetősen centralizált a blokklánc. A Cardano blokkláncra okos szerződéseket fejleszteni 2021 szeptembere óta lehet, így még nem túl nagy az ökoszisztéma, és nincs is eléggé kiforrva. További indok, hogy ne a Cardano-t válasszam, hogy Solidity helyett a saját, Haskell alapú programozási nyelvükben íródnak az okos szerződéseik. Az Avalanche blokklánc hasonlóan nincs még annyira kiforrva, egy elég fiatal projekt, és bár vannak rendkívül jó ötleteik, mint például a konszenzus mechanizmusuk, nincs mögötte akkora stabil hálózat, mint például az Ethereum mögött. Ez abban is meglátszik, hogy a piaci kapitalizációja huszad akkora, mint az Ethereum-nak. Ezeknek a függvényében, a legkiforrottabb, legnagyobb múlttal rendelkező, legismertebb hálózatot választottam blokkláncként az okos szerződésekhez: az Ethereum-ot.

Frontend

A felhasználó által használt frontend felület elkészítésére három keretrendszert fontoltam meg, ezek az Angular, a React és a Vue. Jelenleg ez a három keretrendszer a legnépszerűbb. Nem JavaScript-es megoldásokat azért nem vettem figyelembe, mivel a legegyszerűbb módja az Ethereum blokklánccal való kommunikációnak a Web3.js könyvtár segítségével történik, ami nem elérhető például Blazor-ban. A három keretrendszer közül jelenleg a legnépszerűbb a React, az Angular és a Vue pedig nagyjából egyforma népszerűséggel áll második helyen. A React-ot fejlesztő cég a Meta (korábban Facebook), az Angular fejlesztője a Google, a Vue pedig egy független fejlesztő, Evan You és csapata által van fejlesztve. A hosszútávú támogatást esélyesebbnek látom a nagyobb cégek esetén, így a React és az Angular kerültek előtérbe. A három közül az Angular jött létre legelőször, viszont mindhárom legalább 7 éves és folyamatos fejlesztés alatt áll.

	Előnyök	**Hátrányok**
Angular	• Jó háttér, valószínűleg hosszútávon támogatott lesz a jövőben is • A legtöbb ideje kint van • TypeScript keretrendszer • Munkaerő piacon nagyon keresett • Külső könyvtárak nélkül is rengeteg lehetőséget biztosít • Jól dokumentált	• Komplex • Nagy fájlméretek • Sok fájlal kell dolgozni • Cégeknek és nagy csapatoknak jobban ajánlott
React	• Jó háttér, valószínűleg hosszútávon támogatott lesz a jövőben is • Jelenleg a legnépszerűbb • Nagyon keresett a munkaerő piacon • Jó UI lehetőségek • Rendkívül gyors	• MVC hiánya • Folyamatosan változik, alkalmazások hamar elavulnak • Kevés megfelelő dokumentáció
Vue	• Egyszerű • Hamar látványos oldalakat lehet produkálni vele • Rendkívül gyors	• Kisebb háttér • Legfiatalabb • Nem annyira keresett a munkaerőpiacon

4. Táblázat Frontend keretrendszerek összehasonlítása

A választott keretrendszerem az Angular, mivel nagyon jól dokumentált és támogatott nyelv, továbbá egyetemi és munkahelyi környezetben is ezt volt lehetőségem tanulni és használni.

Backend

Szerver oldalon az egyik fő választást a blokklánc volt, amik közül az Ethereum hálózatát választottam, ezzel együtt a Solidity programozási nyelvet az okos szerződésekhez. Ezt a döntést részletesebben kifejtettem már a blokkláncokról szóló bekezdésben, így itt nem részletezném újra.

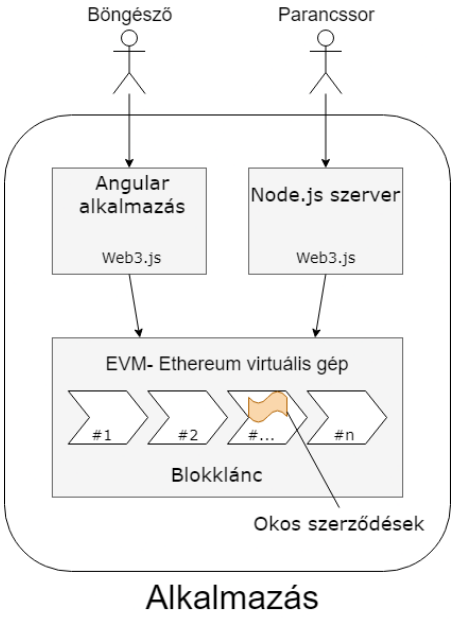
Okos szerződéseken kívül szükség van még szerver oldali szolgáltatásokra, leginkább a frontend alkalmazás miatt. Itt a legegyértelműbb döntés a JavaScript nyelv volt, így ennek futási környezetként a Node.js. Azért volt a legegyértelműbb döntés, mivel mint a frontendnél is, a blokklánccal való kommunikáció JavaScript segítségével történik, így a Node.js volt a legkézenfekvőbb megoldás. Rendkívül elterjedt technológia, részletesebben a róla szóló szekcióban írtam. Összehasonlítani a Deno futási környezettel lehetne. A Deno egy modern futási környezet JavaScript és TypeScript kódnak. A Chrome V8 motorját használja, és Rust programozási nyelvben íródott a környezet (a Node.js C++-ban). Előnye a Node.js-el szemben, hogy sokkal biztonságosabb. Környezeti változókhoz, fájlokhoz vagy hálózathoz csak akkor van hozzáférése, ha engedélyt adtunk rá. Támogatja a TypeScript kódot, így szerver oldalon is lehet TypeScript-et használni. Modulokkal dolgozik, amiket lehet exportálni és importálni. [9] A döntésem azért a Node.js-re esett, mivel jelenleg a munkaerőpiacon egy lényegesen többet keresett tudás, mint a modernebb Deno. Ezen felül, a Web3.js könyvtár stabilan működik Node.js környezetben, ami nem biztos, hogy elmondható Deno környezetre.

Különálló adatbázisra nincs szükség, mivel a blokklánc lényegében adatbázisként fog funkcionálni. A blokkláncon tároljuk az okos szerződésünket, és a szerződésen belül tároljuk a hozzá tartozó adatokat (például a posztokat).

Teszteléshez a Truffle Suite szoftvercsomagot fogom használni. Itt a Truffle felel a konkrét tesztelésért, és a Ganache egy teszt Ethereum blokklánc szimulálásáért, hogy ne járjon költségekkel a fejlesztés és tesztelés. Egyéb választási lehetőség a Remix lett volna. A Remix egy online fejlesztői környezet, alkalmas tesztelésre, és még akár élesben kihelyezésre is. Előnye, hogy minden szükséges eszközt biztosít felhasználói interfészen keresztül, viszont nagy hátránya, hogy nem egy szoftver, hanem online felület, így a fejlesztési élményt lényegesen rontva.

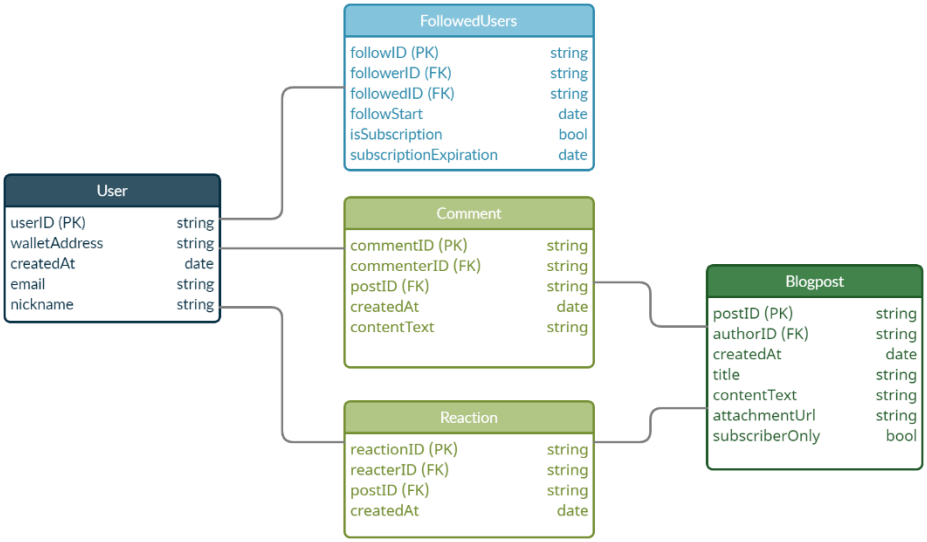
4.2. Rendszerterv

A rendszer relatíve egyszerű. A felhasználó a böngészőjén keresztül lép interakcióba a frontend alkalmazással, ez az alkalmazás Web3.js segítségével pedig a blokklánccal, és az azon elhelyezkedő okos szerződésekkel. Szerver oldalról a Node.js szerverrel tudunk kommunikálni, ez pedig szintúgy a Web3.js könyvtár segítségével kommunikál a blokklánccal. Ez a kommunikáció JSON-RPC-n keresztül történik. Az adatbázissal szintúgy a Web3.js segítségével tudunk kommunikálni.



17. ábra: Rendszerterv

Az adatbázis felépítése a következőképp fog kinézni:



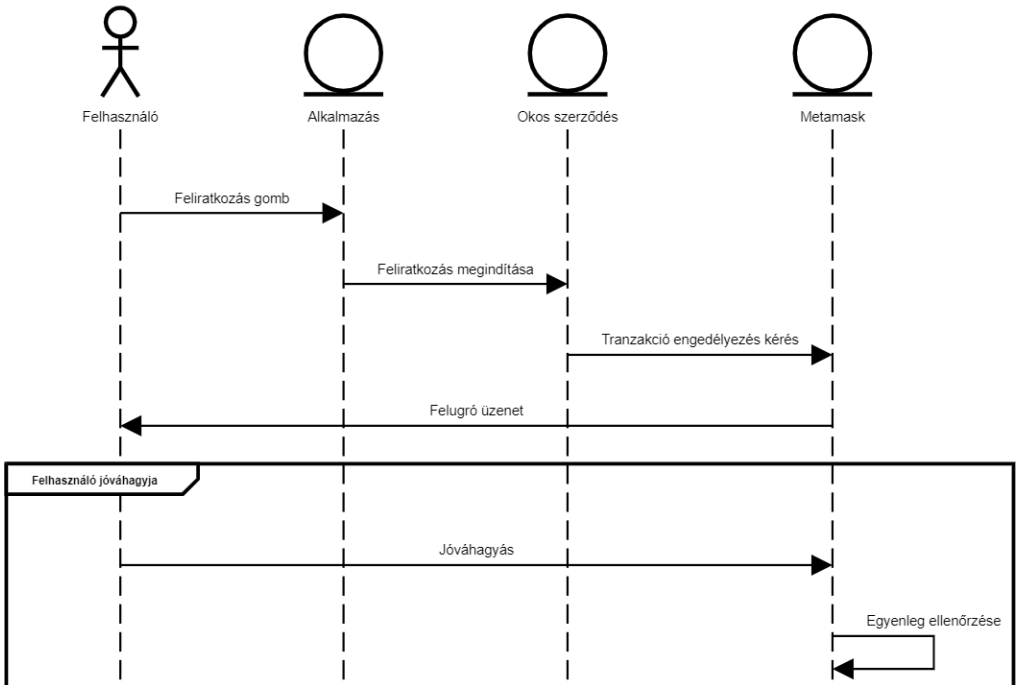
18. ábra: Adatbázis modell

4.3. Funkciólista

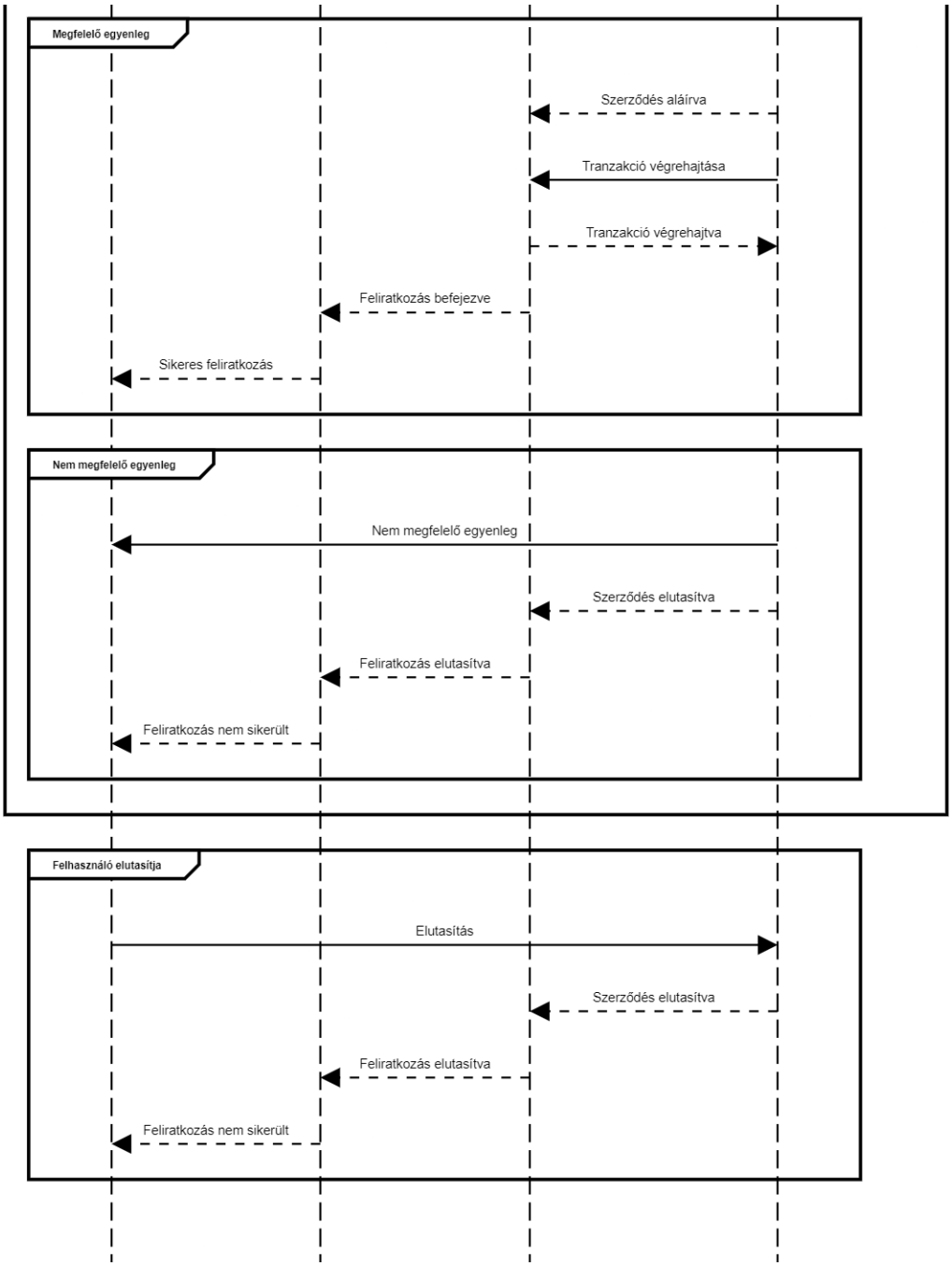
Az alkalmazás a következő funkciókkal rendelkezzen, ahol lehet, ott okos szerződés formájában:

- Bejelentkezés Metamask pénztárca segítségével, Ethereum hálózaton
- A felhasználói fiókban lehessen megadni és szerkeszteni email címet, és felhasználónevet
- Ha van rá lehetőség, email cím megadása után vissza kelljen igazolni az email címet
- A felhasználók tudjanak szöveges blogposztokat publikálni
- A felhasználók tudjanak a blogposztokhoz képeket csatolni
- Felhasználók tudjanak követni más felhasználókat, így a főoldalukon jelenjenek meg a követett felhasználók posztjai, idő szerint csökkenő sorrendben
- Követésen kívül lehessen feliratkozni egy másik felhasználóra fix havidíjért cserébe
- A tranzakció pénztárcán keresztül történjen, Ethereum blokkláncon
- A blogposztoknál opcionálisan bekapcsolható legyen, hogy csak feliratkozott felhasználók láthassák
- A blogposztokat a bejelentkezett felhasználók tudják kedvelni, és tudjanak kommentet írni

Az alábbi diagrammon, a kommunikáció a felhasználó, alkalmazás, az okos szerződés és a pénztárca között látható, amikor egy felhasználó feliratkozik egy másik felhasználóra.



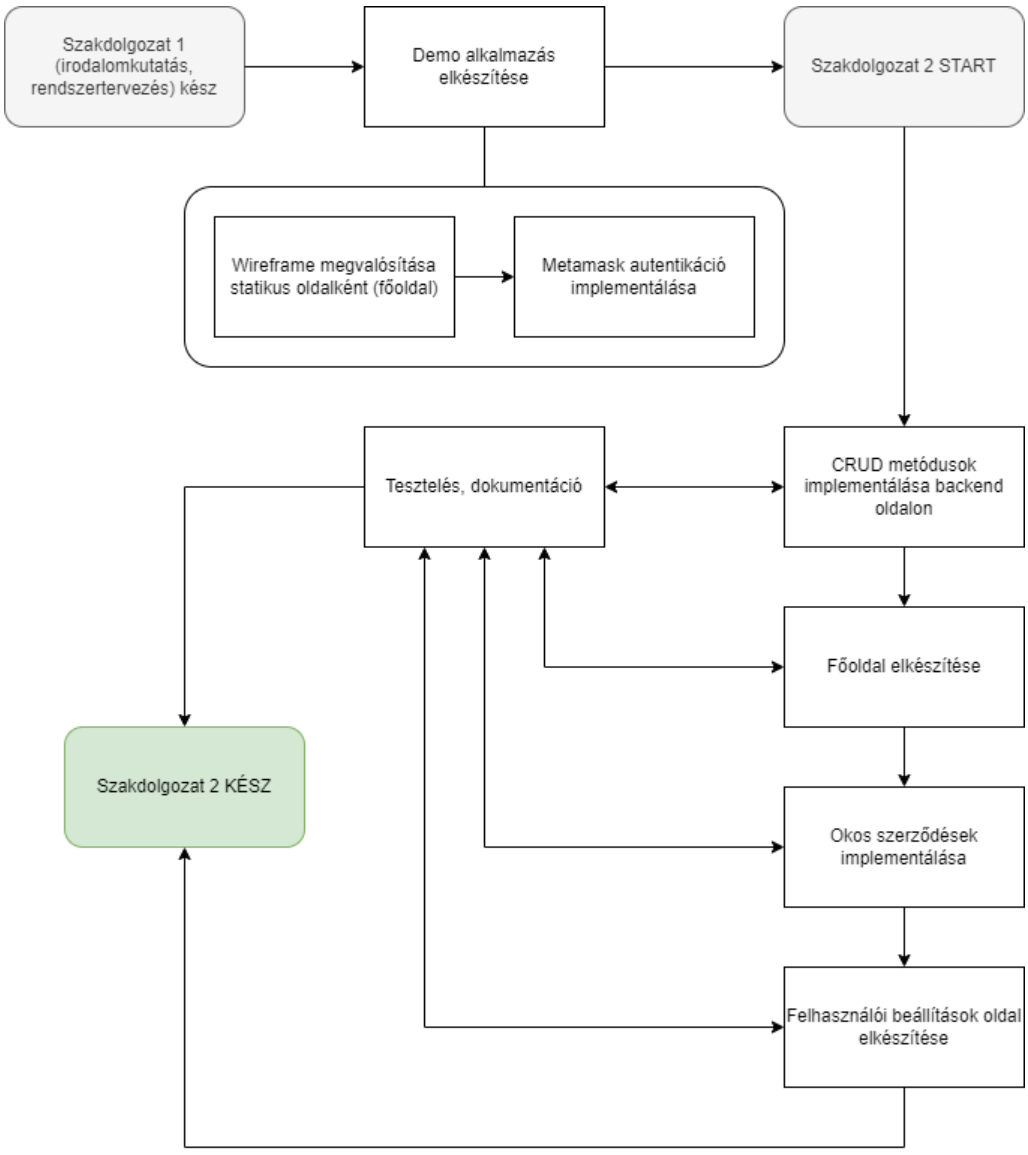
19. ábra: Feliratkozás szekvencia diagram 1/2



20. ábra: Feliratkozás szekvencia diagram 2/2

4.4. Fejlesztés tervezett menete

Az irodalomkutatás végeztével már el tervezek kezdeni dolgozni az alkalmazáson, hogy minél hamarabb felmerüljenek olyan problémák, amik egy decentralizált alkalmazás fejlesztésekor felmerülhetnek. Ezután, a teljes megvalósítás előtt szeretnék mélyebb tudást szerezni Angular keretrendszerben való fejlesztésben, és Solidity fejlesztésben. Szakdolgozatom második felének elkezdésével a teljes alkalmazás fejlesztését is elkezdem, ezt folyamatosan tesztelem, és dokumentálom. A teljes fejlesztés időtartamát nagyjából 10-12 hétre becsülöm.



21. ábra: Fejlesztés tervezett menete

5. FEJLESZTÉS

5.1. Fejlesztési napló bevezetése

Az alkalmazás készítése során nagy figyelmet kell fordítsak arra, hogy biztonsági szempontból alaposan ellenőrizzem az alkalmazást, mivel egy esetleges sérülékenység a rendszerben hatalmas problémákat okozhat. Mivel a blokklánc nem módosítható, így az okos szerződés kódján való módosítás is nehézkes. Ha változtatunk a kódon, újra ki kell helyezni a blokkláncra. Ez alapvetően nem lenne probléma, de mivel nálam az adatbázis is az okos szerződésen belül van tárolva, így ez a teljes adatbázis tartalmának elvesztésével járna. Természetesen ezt át lehetne migrálni, viszont az extra lépéseket, és tranzakciós költségeket igényelne. Ez tesztelés során nem probléma, mivel egy helyi teszt blokkláncon futtatom az okos szerződést, így ingyen vannak a tranzakciók, viszont éles program esetén ez jelentős problémákat, és pénzügyi költségeket okozhat. Ennél fogva a tesztelés, és biztonsági ellenőrzés rendkívül fontos részei az alkalmazásom fejlesztésének.

A biztonságon kívül effektívnek tartom fejlesztéshez a verziókövetést, így az alkalmazás készítése során a git verziókövetőt fogom használni, és a változásokat folyamatosan vezetni rajta keresztül. Ez lehetőséget ad arra, hogy ha egy funkció elromlik, a korábbi működő állapotot visszaállítsam. Mivel a decentralizált alkalmazásoknál a nyílt forráskód szinte alapértelmezett (mivel a blokkláncon megtekinthető), ezért ez a verziókövetett állomány is nyílt forráskódú lesz, ami, ha élő verzióba lenne publikálva megengedné azt, hogy bárki a világon hozzátehessen a kódhoz, fennálló hibákat kijavítson (természetesen jóváhagyás mellett).

Fontos azt is megemlíteni, hogy az alkalmazás leginkább Proof-of-concept céllal készül, tehát a célja, hogy bemutassa a decentralizált alkalmazások lehetőségeit, és a fejlesztés menetét. Ennek alapján olyan funkcionalitások, amik a legtöbb Web2-es alkalmazásban megtalálhatóak, és nem feltétlen szükségesek az alkalmazás működéséhez, azok kisebb prioritással rendelkeznek, és inkább a Web3 funkcionalitásokra fókuszál a fejlesztés.

5.2. Demó alkalmazás

A fejlesztést a korábban megtervezett menetrend szerint (lásd 21. ábra) kezdtem el, egy demó, bemutató alkalmazás készítésével, ami bár egyszerű, de jól bemutatja az előnyeit egy decentralizált alkalmazásnak, mint például az anonimitást. Ez a demó alkalmazás egy egyszerű bejelentkezési felületet, és egy nagyjából üres főoldalt foglalt magába. A bejelentkezést a Metamask, a legnépszerűbb kriptovaluta pénztárca, és egyben az általam választott technológia segítségével oldottam meg. Így a felhasználókról csak a pénztárca címüket tudja az alkalmazás a regisztrálás pillanatától. Ezen a ponton az alkalmazás nagyon egyszerű, még komolyabb routing megoldásokat sem tartalmaz, így lényeges probléma sem merült fel.

5.3. CRUD implementálás backend oldalon

Ahogyan a tervben is szerepelt, a fejlesztést backend oldalon folytattam, az alapvető létrehozás, olvasás, módosítás és törlés funkciók implementálásával, ezzel együtt járt az adatbázis struktúrájának felípétése is. A Truffle Suite keretrendszer segítségével létrehoztam a projekt backendjét, ez a következő mappaszerkezetet jelentette:

- Egy contracts mappa, amiben az okos szerződések vannak tárolva (Solidity)
- Egy migrations mappa, ami a blokkláncra való kihelyezésért felel (JavaScript)
- Egy test mappa, amiben a tesztek lesznek tárolva (JavaScript)

Ezeken felül generálódik egy build mappa az első futtatásnál, amiben JSON formátumú fájlokként tárolja az okos szerződéseket. A blokkláncra kihelyezéshez tudunk scripteket írni, ezt egy scripts mappában tárolhatjuk. Létrehoztam még egy mappát, amibe az alkalmazás frontendje kerül.

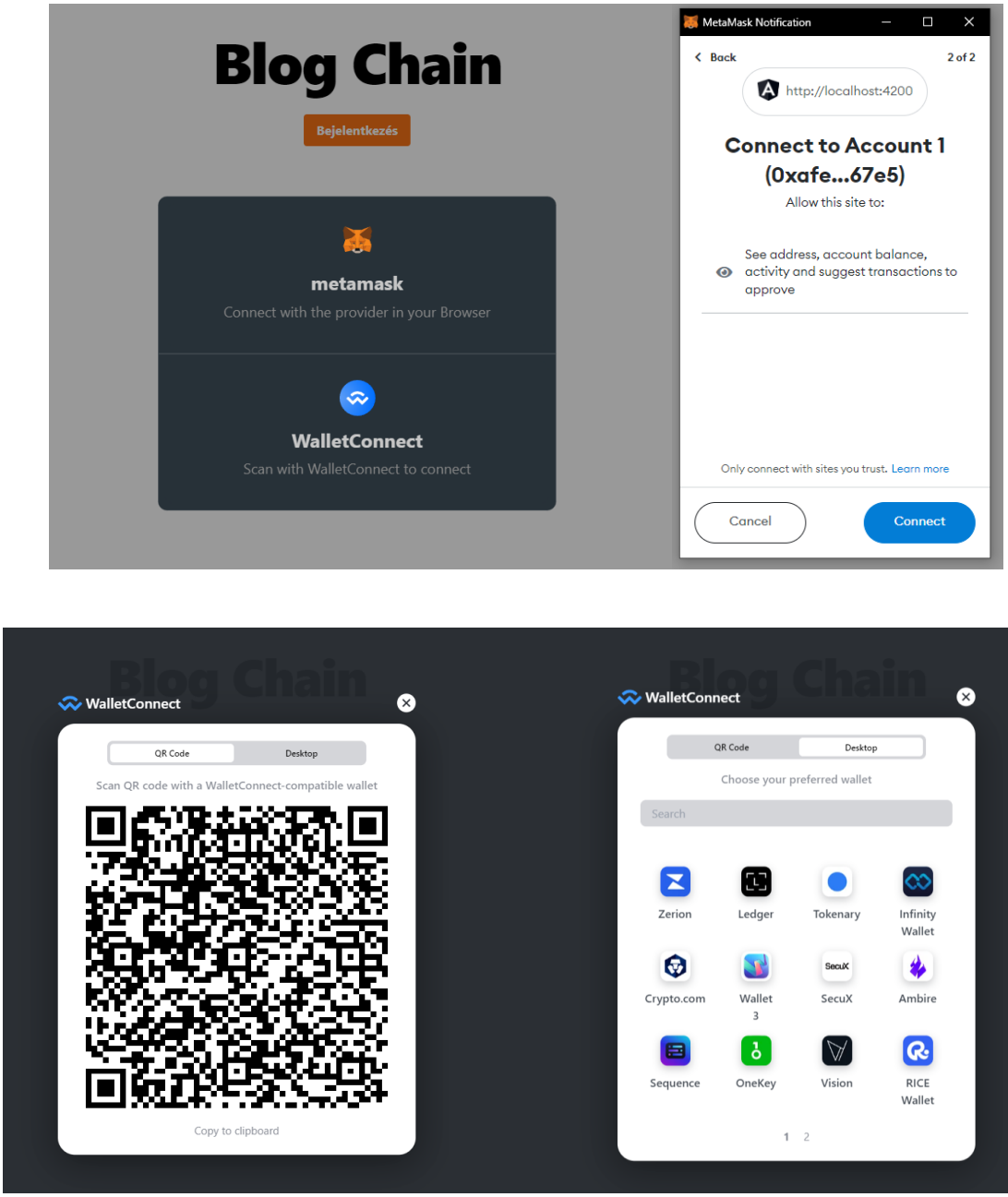
Az okos szerződést magát öt részre lehet jelenleg felosztani: struktúra felépítése, adattagok, események, konstruktor és metódusok. A struktúra felépítése, az lényegében az adatbázis tábláinak kialakítása objektumokhoz hasonló struct-ok használatával. Még hagyományos alkalmazásokban az, hogy alkalmazás szinten tároljuk az adatokat azzal jár, hogy minden újraindításnál elvesznek az adatok, decentralizált alkalmazásoknál a blokkláncon tárolás miatt ez a probléma nem áll fenn, így lényegében a szerződésen belüli adattagokban tudjuk tárolni az adatainkat. Erre a Solidity nyelvben az egyik legjobb megoldás, a mapping nyelvi elem használata. A mapping lényegében kulcs-érték párok tárolására alkalmas, és ezekből egy dinamikus hosszúságú listát tárolhatunk benne. Ez lesz lényegében az alkalmazás adatbázisa. Számos eseményt is definiáltam a kódban, ezek minden adatokban való változtatás (legyen az írás vagy olvasás) elsülnek, ezeket majd később frontend oldalon fogjuk figyelni. A konstruktorba pár inicializálásra szolgáló sor és pár teszt adat került. A metódusok az alapvető létrehozás, szerkesztés és törlés metódusok jelenleg, viszont ide kerül majd minden későbbi funkciót megvalósító programkód is.

Ezen a ponton már számos probléma felmerült, a legtöbb a Solidity nyelv sajátosságaiból adódott. Jelenleg két fő adattag van a programban, az egyik a felhasználókat tároló Users mapping, ahol a kulcs egy address (egyedi pénztárca cím) és a hozzá tartozó érték egy User objektum. A másik a posztokat tároló Posts mapping, ahol a kulcs egy address (így felhasználónként vannak tárolva a posztok), és a hozzá tartozó érték egy másik mapping, amiben a kulcs egy szám, az érték pedig egy BlogPost objektum. Egy kisebb probléma származott a mapping használatából, mégpedig, hogy ezen az adatszerkezeten nem lehet iterálni. Ismernünk kell a kulcsát az elemnek, amit vizsgálni szeretnénk. Ennek a problémának a kiküszöbölésére két megoldás létezik. Az egyik, hogy használunk egy változót, amivel számon tartjuk, hogy hány elemet tartalmaz a mapping. Ha kulcsként ezt a számot adjuk meg, úgy végig tudunk az adatok listáján később iterálni. A másik

megoldás a tömb használata. Solidity-ben létezik fix elemszámú, és dinamikus méretű tömb is, így alkalmas lett volna a tömb is ezeknek az adatoknak a tárolására. Megoldásként az első megoldást választottam, mivel így elérhetem azt is, hogy az adataimat be tudjam járni, és megtartom a mapping-nek azt az előnyét, hogy kulcs alapján egyből rá tudok mutatni adott elemekre, így gyorsítva a legtöbb művelet futási idejét. A Solidity általam használt verziója (0.8.10) nem nyújt lehetőséget arra, hogy struct példányokat használjunk kulcsként egy mapping-ben, így az adatok tárolását számomra lényegesen megnehezítette. Másik probléma szintúgy az adatok tárolásával, hogy nem lehet mapping-ben érték olyan objektum, aminek van egy dinamikus tömb adattagja, így megint megakasztva a fejlesztésem menetét. A megoldás ezekre a problémákra az volt, hogy a struct-okon belül is mapping használatával tárolok adatokat, ha azoknak listájára van szükség. Ennek az a legnagyobb hátránya, hogy minden mapping mellé szükséges egy számláló tulajdonság a struct-ba, hogy egyedi kulcsot tudjunk adni az értékekhez. Egyedi kulcs generálására alapvető módszer nincs (mint például C# esetén a GUID használata). Még egy nyelvi elem, ami problémát okoz, az idő és dátum tárolása. Solidity-ben nincs beépített megoldás dátum tárolására, így egyedi megoldást kell alkalmaznom. Dátumokat Unix-időben fogom tárolni, és ezt frontend oldalon JavaScript segítségével átalakítani tényleges dátumokká. Úgy gondolom, hogy ezeknél a problémáknál kellően átgondolt, és effektív megoldást alkalmaztam.

5.4. Bejelentkezés oldal

A demó alkalmazáshoz képest átalakítottam a bejelentkezést, így már nem csak Metamask, hanem más tárcák (pl. WalletConnect) használatával is be lehet jelentkezni (Lásd 22. ábra). Ha van telepítve egy tárca, például Metamask, a bejelentkezés gombra kattintva opcióként megjelenik a telepített, és a WalletConnect opciója. Utóbbi a Metamask-kal ellentétesen nem egy böngésző kiegészítő, helyette számos opciót ad nekünk. Egy QR kódot generál, amit telefonos tárca esetén tudunk használni, vagy asztali alkalmazásokat tud felnyitni, így bármilyen böngészőn kívüli tárcával is használható az oldal. Ha nincs telepítve tárca a böngészőbe, akkor csak a WalletConnect opciója jelenik meg. Ha a Metamask opcióját választjuk, megjelenik egy ablak, ahol engedélyt adunk az oldalnak látni a tárca címünket, egyenlegünket, és hogy tranzakciót indítson (jóváhagyni ugyanúgy nekünk kell, tehát a beleegyezésünk nélkül nem tud végrehajtani tranzakciót).



22. ábra: Bejelentkezés képernyő, funkció

5.5. Főoldal elkészítése

Következő lépésként a főoldallal dolgoztam. Ahogyan a tervezés során meghatároztam, Angular keretrendszerben dolgoztam. Az oldal kinézetéhez az alap HTML és CSS mellett egy Tailwind nevű CSS keretrendszert is használtam, ez lényegesen megkönnyítette az elemek kinézetének manipulálását. Az oldal felül egy navigációs sávval rendelkezik, ahonnan a főoldalra, az új poszt készítő képernyőre, és a beállítások képernyőre lehet átlépni. Ezek mellett egy felhasználó kereső, és egy kijelentkezés opció is felkerült a menüre. Az oldal nagyrészét a blogposztok teszik ki. Itt a követett felhasználók posztjait láthatjuk. A fizetős posztoknál külön ikonnal van jelezve, hogy az fizetős tartalom, azaz csak feliratkozott felhasználók láthatják. Az oldalak közötti navigációt az Angularba beépített routing módszerekkel valósítottam meg.

5.6. Okos szerződések implementálása

A Ganache nevű programban futtatott lokális blokkláncra kihelyeztem az okos szerződés jelenlegi állapotát a Truffle funkcióinak segítségével. Ennek a fő lépései, a Visual Studio Code konzoljának használatával a következő parancsok:

- *truffle compile*
- *truffle migrate*
- *truffle deploy*

A *compile* parancs készít a Solidity kódból egy úgynevezett artifact-ot, ez a következő lépésekhez szükséges JSON file. A *migrate* parancs lefuttatja a migrations mappában szereplő JavaScript fájlokat, amik elkészítik a JavaScript által is értelmezhető formáját a szerződésnek, ez kerül a build mappába. Az utolsó lépés kihelyezi a konfigurációs fájlban meghatározott blokkláncra (jelen esetben lokálisan futtatott teszt lánc) a szerződést.

Ezután az Angular projektemben a Solidity-ben meghatározott struct-okat modellek formájában megvalósítottam TypeScript-ben. A szerződés metódusait ezután az úgynevezett ABI (Application Binary Interface) használatával határoztam meg a TypeScript osztályaimnak. Az ABI egy JSON formátumú leírása a szerződés metódusainak, és ezt a Solidity kód fordítása során generált JSON fájlból lehet kinyerni. Ezen felül szükség van még a szerződés címére (például ’0x577b207dF41ee6a1E38ea06a7FBF3075A3f1558B’ a jelenlegi állapotában). Ezeknek a segítségével egy Contract objektumot létrehoztam, amin keresztül meg tudom hívni az okos szerződés metódusait. Web3js segítségével 2 féle módon tudjuk ezeket meghívni. Az első lehetőség a *call(),* ekkor csak egy olvasási művelet történik, ez nem indít tranzakciót, így nem kerül pénzbe, és szinte egyből megtörténik. A másik lehetőség a *send()*, itt tranzakció indítás történik, amit MetaMask-ban jóvá kell hagynunk, a blokklánc sebességétől függően egy tranzakció idejéig tart, és tranzakciós költséggel is jár. A *call()* hívást a posztok listázásánál használtam, a *send()* metódust pedig a poszt írásnál.

A frontend és backend összekötése lényeges nehézségeket okozott, mivel a legtöbb Web3 JavaScript könyvtár dokumentációja React használatához készült, így sok helyen nem stimmelt az én alkalmazásomra, ami a hivatalos leírásban szerepelt. Erre remek példa, hogy az Angular TypeScript-et használ, ahol kötelező a változók típusának megadása, erre viszont például a Web3js dokumentációjában sehol se volt példa, így mindenhol magamtól kellett a típusokat meghatározni.

Ahhoz, hogy a blokkláncról metódusokat hívhassunk meg, három dologra van szükségünk. Az első egy úgynevezett provider, ez magát a hálózatot (például Ethereum, vagy Binance) határozza meg. A második egy signer, azaz aláíró ez a felhasználót határozza meg a csatlakoztatott tárcán keresztül. Mivel be vagyunk már jelentkezve, ezért nem kell újra jóváhagynunk ezt, de ha még nem lennénk (például egy olyan alkalmazásnál, ahol nem kötelező a bejelentkezéshez MetaMask-os autentikáció), akkor ezt engedélyeznünk kellene külön. A harmadik pedig maga a szerződés, ezen keresztül fogjuk tudni meghívni a Solidity-ben elkészített metódusokat. Ehhez meg kell adnunk a szerződés címét (pl.’ 0x723496A6BDEefEBa9B4647B78BC0288438d0C060’), az ABI-ját (korábban generált Truffle segítségével), és az előző lépésben meghatározott signer-t. Ezeknek a megadásával az okos szerződés publikus metódusait és elemeit elérjük a frontend (de akár backend) kódbázisunkból is.

5.7. Felhasználói beállítások oldal elkészítése

A beállítások oldal nem járt túl sok problémával, mindössze egy kijelzett név, és egy email mező vannak rajta, egy mentés gombbal. Itt a felhasználó szabadon tudja akárhányszor változtatni a nevét és email címét, viszont mivel ez blokkláncon való szerkesztéssel jár, így minden módosítás kriptovalutába kerül. Ha már meg van adva kijelzett név vagy email cím a felhasználónak, az oldal automatikusan tölti a mezőket ezekkel az értékekkel, viszont ez szabadon szerkeszthető.

5.8. Felhasználók keresése oldal elkészítése

Következőként egy olyan oldalt készítettem, ahol a felhasználók más felhasználókat tudnak megkeresni, és követni őket, hogy lássák a posztjaikat. Ez a felület egészen egyszerű, egy lista az összes felhasználóról (leszámítva az aktuális felhasználót), egy kereső mezővel ellátva. A kereső mező különlegessége, hogy lehet a beállításokban beállítható becenév alapján, vagy pénztárca cím alapján is keresni felhasználókat. Ezen a felületen lehet követni vagy kikövetni más felhasználókat, vagy feliratkozni rájuk, leiratkozni róluk.

6. TESZTELÉS

A fejlesztés során manuálisan különböző eszközök (pl. Truffle, Remix, Ganache) használatával teszteltem az alkalmazást, továbbá Truffle használatával végeztem Unit tesztelést. A fejlesztéssel párhuzamosan folyamatosan teszteltem az elkészült funkciókat.

A fent említett teszt eszközök közül kiemelném a Ganache nevű szoftvert, ami az Ethereum blokkláncot lokálisan szimulálja. Ez rendkívül hasznos volt számomra, többek között azért, mert fejlesztés során nagyjából 1 ETH (azaz közel fél millió forintnyi) költsége volt a tranzakcióknak. Ez egy nyilvánvaló indok, hogy éles blokkláncon miért nem érdemes fejleszteni, tesztelni. A szoftverben továbbá tudtam vizuálisan is ellenőrizni a szerződés tartalmát, például, hogy egy adott posztnak megfelelően töltődtek-e a tulajdonságai. Így folyamatosan tudtam monitorozni, hogy minden érték megfelelően töltődik-e a rendszerben. Lásd 23. ábra.

23. ábra: Ganache szerződés elemzés

#	Követelmény	Tesztleírás	Mérés és cél	Eredmény
1.	A bejelentkezés gombra kattintva megjelenik a pénztárca választó	Első belépés az oldalra, első kattintás a belépésre	A gombra kattintva megjelenik a WalletConnect opciója, és a feltelepített tárca opciója	Siker
2.	Sikeres bejelentkezés Metamask segítségével	Első belépés Metamask használatával	Feltelepített Metamask használatával az opciót kiválasztva megjelenik egy felugró ablak, ahol jóvá hagyhatjuk a bejelentkezést	Siker
3.	Sikeres bejelentkezés WalletConnect segítségével, mobil tárcát használva	Első belépés WalletConnect segítségével, mobil tárcáról	Mobilon feltelepített tárca segítségével, a WalletConnect által generált QR kód segítségével sikeres a bejelentkezés	Siker
4.	Sikeres bejelentkezés WalletConnect segítségével, asztali tárcát használva	Első belépés WalletConnect segítségével, asztali tárcáról	Asztali alkalmazásként feltelepített tárca segítségével, a WalletConnect-en keresztül sikeres bejelentkezés	Siker
5.	Sikertelen bejelentkezés elutasított engedély adás miatt	Első belépés, engedélyek megadásának elutasítása	Tetszőleges tárca használatával elutasítjuk az engedélyek megadását, így nem jutunk el a főoldalra	Siker
6.	Sikeres poszt publikálás	Új poszt írása tetszőleges felhasználóról	Az új poszt készítés fülről írunk egy posztot, amit kihelyezünk a blokkláncra Tetszőleges tárcával autorizáljuk a tranzakciót, kikerül a blokkláncra, és a főoldalra a poszt	Siker
7.	Felhasználók keresése név alapján	A keresés fülre navigálva, a kereső mezővel a felhasználók között kijelzett név szerint keresünk	A találatok szűkülnek csak a keresett névvel rendelkező felhasználókra	Siker

8.	Felhasználók keresése tárca cím alapján	A keresés fülre navigálva, a kereső mezővel a felhasználók között tárca cím szerint keresünk	A találatok szűkülnek csak a keresett címmel rendelkező felhasználóra	Siker
9.	Felhasználó követése	A keresés fülön a követés funkciót használjuk egy felhasználónál	A tárcánkra engedélykérést kapunk, majd azt jóváhagyva, a követés gomb kikövetésre változik, azaz sikeresen bekövettük a felhasználót	Siker
10.	Követett felhasználó posztjainak megtekintése	Felhasználó követése után a főoldalra kattintva megtekintjük az ingyenes posztjait	Felhasználó követése után a főoldalra kattintva a megjelenített posztok között látjuk a frissen bekövetett felhasználó ingyenes posztjait is	Siker
11.	Feliratkozás felhasználóra	A keresés fülön a feliratkozás funkciót használjuk egy felhasználónál	A tárcánkra engedélykérést kapunk, majd azt jóváhagyva, a feliratkozás gomb leiratkozásra változik, azaz sikeresen feliratkoztunk a felhasználóra	Siker
12.	Feliratkozott felhasználó posztjainak megtekintése	Felhasználóra feliratkozás után a főoldalra kattintva megtekintjük a fizetős posztjait	Felhasználó követése után a főoldalra kattintva a megjelenített posztok között látjuk a frissen bekövetett felhasználó fizetős posztjait is	Siker

13.	Saját felhasználó követése/feliratkozása	A keresés fülön a feliratkozás/követés funkciót használjuk a saját felhasználónknál	Nem láthatjuk a saját felhasználónkat a keresés fülön, ezzel meggátolva a saját magunkra való feliratkozást	Sikertelen (elvárt)
14.	Sikeres fizetős poszt publikálás	Új poszt írása tetszőleges felhasználóról, fizetős tartalom opció aktiválásával	Az új poszt készítés fülről írunk egy posztot, amit kihelyezünk a blokkláncra Tetszőleges tárcával autorizáljuk a tranzakciót, kikerül a blokkláncra, és a főoldalra a poszt Csak feliratkozott felhasználók láthatják	Siker
15.	Sikeres név és email cím váltás	A beállítások fülön kijelzett név és email cím váltása	A beállítások fülön megadunk egy tetszőleges nevet és emailt, majd a főoldalon és a keresőben megváltozik a kijelzett nevünk	Siker
16.	Felhasználó kikövetése	A keresés fülön a kikövetés funkciót használjuk egy felhasználónál	A tárcánkra engedélykérést kapunk, majd azt jóváhagyva, a kikövetés gomb követésre változik, azaz sikeresen kikövettük a felhasználót	Siker
17.	Leiratkozás felhasználóról	A keresés fülön a feliratkozás funkciót használjuk egy felhasználónál	A tárcánkra engedélykérést kapunk, majd azt jóváhagyva, a leiratkozás gomb feliratkozásra változik, azaz sikeresen leiratkoztunk a felhasználóról	Siker
18.	Kijelentkezés	A kijelentkezés gomb a bejelentkezésre irányít	A kijelentkezés gomb használatával visszakerülünk a bejelentkezés felületre, és nem érjük el újra bejelentkezés nélkül a többi oldalt	Siker

5. Táblázat: Tesztelések

7. TOVÁBBFEJLESZTÉSI LEHETŐSÉGEK

Az alkalmazás fejlesztése alatt elsődleges szempontnak a blokkláncok és az okos szerződések lehetőségeinek bemutatását vettem szempontként, így az ezen kívüli funkciókban számos továbbfejlesztési lehetőség van, de a Web3 fejlesztés mentén is van lehetőség további funkciókat implementálni.

A frontend alkalmazás nem túl látványos jelenleg, alapvető funkciókat szolgál ki, alapvető és egyszerű UI elemekkel. Ezt ki lehetne bővíteni számos kényelmi funkcióval, például a főoldalra a posztok mentén szűrés, tetszőleges, vagy algoritmus szerinti rendezés. Ezen felül reszponzivitás terén is lehetne az alkalmazást tovább fejleszteni, mivel jelenleg egy átlagos magas felbontású monitoron működik optimálisan, mobilon, vagy kisebb kijelzőn összecsúsznának az elemek.

Jelenleg Ethereum blokkláncon fut, Ethereum kriptovalutát használ a rendszer. Ezt többféle módon is lehetne kibővíteni. Első sorban megvalósítható lenne, hogy több különböző blokkláncon is működjön az alkalmazás, például Binance, Avalanche, vagy Cardano. Előbbi kettővel relatíve egyszerű lenne ezt megvalósítani, mivel ezek EVM kompatibilis láncok, de a Cardano esetén érdekes kihívás lenne ezt lefejleszteni. Ezen felül akár saját kriptovalutát is használhatna tetszőleges láncon. Okos szerződés keretein belül akár egy saját kitalált, általunk létrehozott kriptovalutával is működhetnének a tranzakciók.

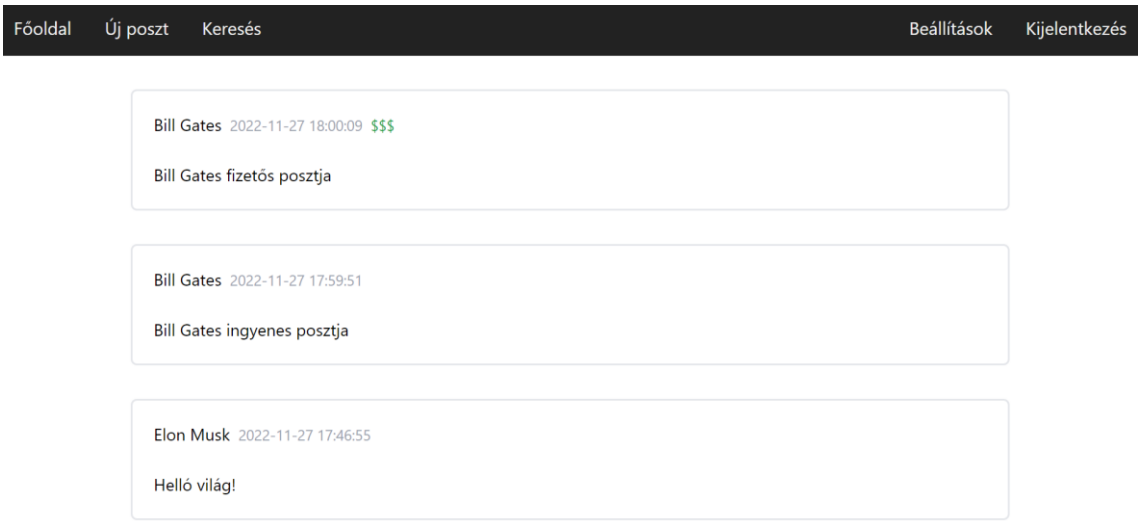
Backend oldalon lényegében csak blokklánc és okos szerződés van, ezt ki lehetne bővíteni hagyományos, centralizált backend-el, ami bár csökkentené a decentralizáltságot, de határozottan növelné az oldal egyszerű használhatóságát. Jelenleg minden írás típusú funkció tranzakció, azaz kriptovalutába kerül. Azokat a funkciókat, amik nem feltétlen szükségesek a blokkláncra (például becenév, email cím megadása), ki lehetne szervezni egy hagyományos adatbázisba és backend szolgáltatásba, így csökkenteni a szükséges tranzakciók számát és a rendszer sebességét. Ami ez esetben rendkívül fontos lenne, hogy a kritikus adatokat, például maga a felhasználó és a posztjai továbbra is a blokkláncon tároljuk, így védve őket bármi féle támadástól vagy cenzúrától. El nem készült funkció a kommentelés és a reagálás, ez annál fogva nem készült el, hogy rendkívül költséges lenne egy tetszést kifejező reakcióért is fizetni, ez kifejezetten olyan funkció, ami centralizált alkalmazás esetén sokkal effektívebb lenne.

A posztok jelenleg nincsenek szűrve, így bármilyen tartalom publikálható. Erre egy akár saját, akár külsős rendszer használatát érdemes lenne bevonni, hogy például az illegális, vagy szexuális tartalmakat kiszűrhessük, ne lehessen ilyeneket posztolni.

Végül a rendszer élesítése, az igazi Ethereum blokkláncra való kihelyezés lenne természetesen egy továbbfejlesztési lehetőség, mert jelenleg csak lokálisan fut a rendszer, lokális teszt láncról. Ez jelentős költségekkel járna.

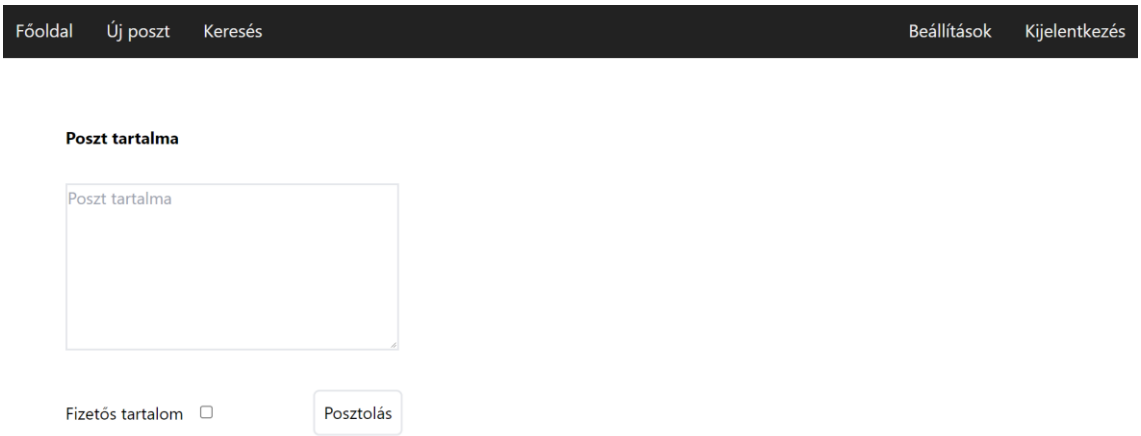
8. KÉPEK AZ ELKÉSZÜLT ALKALMAZÁSRÓL

Az alkalmazásba belépve a bejelentkezési felület fogad (lásd 22. ábra). Bejelentkezve a főoldal fogad minket, amiről egy kinagyított képet illesztettem be (lásd 24. ábra) a dolgozatban való könnyebb olvashatóságért, de az alkalmazásban normál felbontás esetén alap esetben is teljesen olvasható az oldal. Látható a felső posztnál, a dátum melletti jelzésből, hogy az egy csak fizető feliratkozóknak elérhető poszt.



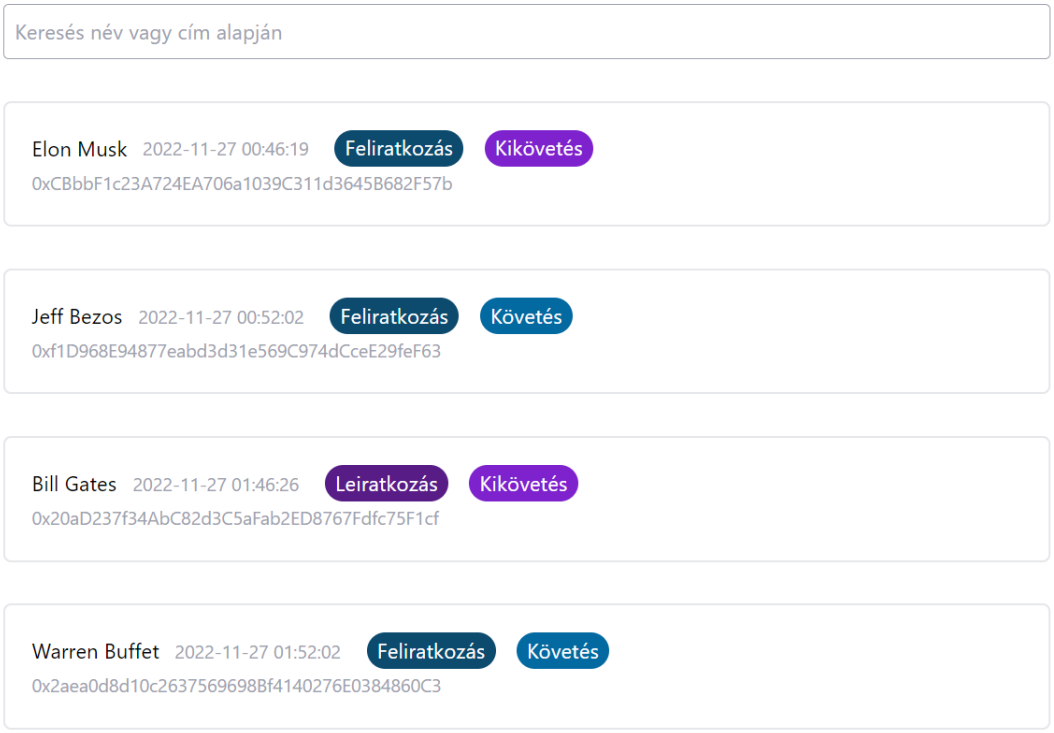
24. ábra: Az alkalmazás főoldala

Az új poszt fül alatt egy egyszerű oldal fogad, ahol tetszőleges szöveget lehet kiposztolni, választhatóan fizetős tartalomként. Lásd 25. ábra.



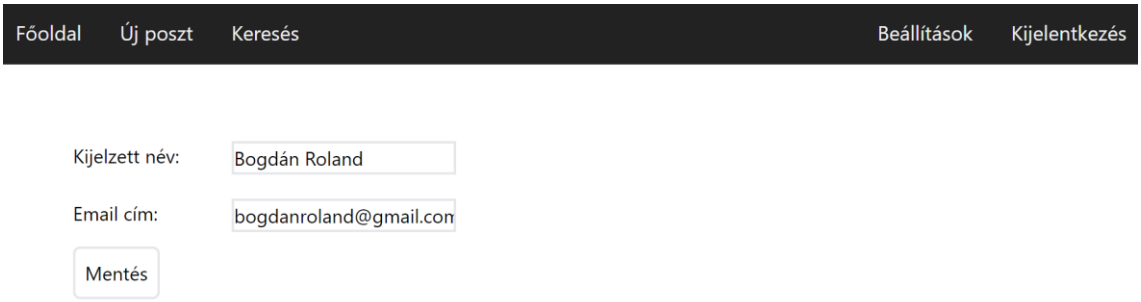
25. ábra: Új poszt oldal

A keresés fülön láthatjuk a rendszerben szereplő felhasználókat, illetve a feliratkozások és követések állapotát. A felhasználóknál látható továbbá a regisztráció pontos ideje, és a pénztárca címük. Ha nincs kijelzett nevük beállítva, helyette a tárca cím látszik, és a tárca cím helyén egy felirat, hogy még nem állított be kijelzett nevet. Lásd 26. ábra.



26. ábra: Felhasználó keresés fül

A beállítások fülön meg tudjuk adni, vagy ki tudjuk törölni a kijelzett nevet, és az email címet. Lásd 27. ábra.



27. ábra: Beállítások fül

9. ÖSSZEGZÉS

A szakdolgozatom során kellően elmélyítettem a tudásomat a blokklánc technológiákban, és rendkívül sokat tanultam a Web3 fejlesztés menetéről, akadályairól, lehetőségeiről és problémáiról. A Solidity még nem egy rendkívül elterjedt nyelv, számos esetben dokumentáción kívül sehol sem volt elérhető információ a problémákról, amikbe belefutottam. A fejlesztés során megismerkedtem több eszközzel, ami használható okos szerződések fejlesztésénél, ilyen volt például a Truffle, Remix és Ganache. Megtanultam a használatát több JavaScript könyvtárnak, mint például a Web3.js és az Ethers.js. Rendkívül tanulságos tapasztalat volt egy rendszert a semmiből megtervezni, majd felépíteni, miután irodalomkutatást végeztem a témában.

Az elkészített rendszer jelenleg, élesbe helyezés esetén képes lenne akár több millió felhasználót kiszolgálni, az Ethereum blokklánc rendkívül nagy teljesítménye miatt. Az alkalmazáshoz tervezett funkciók javarészt elkészültek, az alkalmazás képes lenne teljesen decentralizálva működni, ami fontos szempont volt a tervezés során. Adatvédelem szempontjából is megfelel az alkalmazás a tervezett állapotnak, mivel egy felhasználónak semmi más adatát nem kell megadnia kötelezően, mint a pénztárca címét, így anonim maradhat.

Úgy gondolom, hogy a következő évtizedekben hasonló növekedést fog látni a Web3 mint amit a Web2 látott az elmúlt évtizedekben, és rengeteg az enyémhez hasonló projekt fog napvilágot látni a közeljövőben, így örülök, hogy korán belefogtam ennek a fejlesztésébe.

Irodalomjegyzék

[1] Lema, C.: Chris Lema’s blog, (https://chrislema.com/blog/), utoljára megtekintve: 2021. 09. 29.

[2] Wakio Microblog: Adventures in Skynet, (https://siasky.net/hns/microblogsc/), utoljára megtekintve: 2021. 09. 30.

[3] Sigle: Decentralized writing platform, (https://app.sigle.io/sigleapp.id.blockstack), utoljára megtekintve: 2021. 10. 01.

[4] Buterin, V.: The Meaning of Decentralization, (https://medium.com/@VitalikButerin/the-meaning-of-decentralization-a0c92b76a274), utoljára megtekintve: 2021. 10. 22.

[5] Sephton, C.: Bitcoin price today, (https://coinmarketcap.com/currencies/bitcoin/), utoljára megtekintve: 2021. 10. 22.

[6] Nakamoto, S.: Bitcoin Whitepaper,(https://bitcoin.org/bitcoin.pdf), utoljára megtekintve: 2021. 10. 29.

[7] Ethereum: Ethereum Whitepaper, (https://ethereum.org/en/whitepaper/), utoljára megtekintve: 2021. 10. 29.

[8] Solidity: Solidity Documentation, (https://docs.soliditylang.org/en/v0.8.10/), utoljára megtekintve: 2021. 10. 29.

[9] W3Schools: JavaScript HTML DOM, (https://www.w3schools.com/js/js_htmldom.asp), utoljára megtekintve: 2021. 11. 12.

[10] CoinMarketCap: Historical Snapshot (2021. 11. 28.), (https://coinmarketcap.com/historical/20211128/), utoljára megtekintve: 2021. 12. 03.

Ábrajegyzék

Táblajegyzék