DEBRECENI EGYETEM • INFORMATIKAI KAR

Webalkalmazás fejlesztése online szavazás megvalósítására

SZAKDOLGOZAT

KÉSZÍTETTE:

Ardai Zsolt
gazdaságinformatikus hallgató

TÉMAVEZETŐ:

Dr. Tomán Henrietta
egyetemi adjunktus

Debrecen

2022

Köszönetnyilvánítás

Ezen fejezetben elsősorban szeretném megköszönni az Dr. Tomán Henriettának a szakdolgozat elkészítése során nyújtott segítségét. A családomnak az egyetem elvégzésében nyújtott anyagi és lelki támogatást, az Ozeki Systems Kft-nek a szakmai gyakorlatom során szerzett tapasztalatokért, illetve az IT Squad Kft-nek a React és Next.js alapú ismereteim elmélyítéséért.

Tartalomjegyzék

1. Bevezetés

A koronavírus járvány életünk minden területén fennakadást okozott, és drasztikus lépéseket kellet tennünk mindennapi tevékenységeink digitalizálásra, a social distancing megvalósításának érdekében.

Szakdolgozatom témája egy olyan felhasználóbarát felület kialakítását foglalja magában, amely segítségével az állampolgárnak otthona kényelméből, vagy akár egy tömeg közlekedési eszközön is lehetősége nyílik részt venni a polgári demokrácia egyik legfontosabb folyamatában, az őket képviselő köztisztviselők megválasztásában.
Ezt egy olyan modern fullstack webfejlesztési keretrendszer használatával tervezem kivitelezni, amely magában foglalja a szerver és kliens oldali renderelés lehetőségét, és lehetőséget biztosít arra, hogy egy projekten belül készítsük el a frontend-hez és a backend-hez szükséges infrastruktúrát.

A frontend kivitelezéséhez az egyik legismeretebb webes keretrendszert a React-ot fogom használni, mely egy a Meta által fejlesztett ingyenes nyílt forráskódú JavaScript csomag. Továbbá a felhasználói adatok, illetve szavazatok eltárolását egy manapság nagyobb ismeretséget szervezett, és szélesebb körben elterjedt no-sql adatbázis használatával a MongoDB-vel valósítom meg.

A témaválasztásom indoka az, hogy jelenleg Magyarországon nincs egy olyan rendszer, amely lehetővé tenné a demokratikus folyamatok működését a személyes jelenlét nélkülözésével akár egy pandémia, vagy egy olyan esemény alatt, melynél a személyben történő szavazás nem kivitelezhető.
Az európai országok közül ezen a téren kiemelkedő eredményeket produkált már Észtország, amely függetlenségének elnyerése óta fokozatosan egy digitális társadalom kiépítésén dolgozott. Ma az észt állampolgároknak lehetősége van adóbevallásuk és bankolásuk online elintézésre, valamint szavazatuk leadására.
Észtországban a népesség 89,1%-a rendelkezik internethozzáféréssel, ami hozzávetőlegesen nem sokkal tér el a magyarországi adatoktól, ahol a lakosság 84,8%-a rendelkezik internethozzáféréssel. Az észttel ellenben viszont a magyar közszemléletben sokkal nagyobb a bizalmatlanság az online elvégzett tranzakciókkal és azon biztonságosságával szemben.

A szakdolgozatom során bemutatom a projekten belül általam használt titkosítási szabványokat, alkalmazásukat mind elméleti és programozási szinten, továbbá kitérek az

általam használt csomagokra és függőségekre, illetve felvetek néhány módszert a szavazás névtelenségének kivitelezésére, a fentebb említett titkosítási szabványok alkalmazásával. Ismertetem az általam elkészített felhasználói felületet, részletesen kitérve azokra a React specifikus elemekre, amelyeket fontosnak tartok kiemelni a projekt szempontjából. Ezután szemléltetem az általam használt módját a szavazatok megszámlálására, illetve validálására a Python, magas szintű programozási nyelv segítségével segítségével.

2. A szavazás titkosságának kérdése

A szavazás titkosságát Magyarország 2011 évi CCIII. törvénye határozza meg, mely kimondja, hogy „Az országgyűlési képviselőket a választópolgárok **általános** és **egyenlő választójog** alapján, **közvetlen** és **titkos szavazással**, a választók akaratának szabad kifejezését biztosító választáson, sarkalatos törvényben meghatározott módon választják.” [1] Programozási szempontból a **szavazás titkosságát** és **egyenlőségét** több oldalról is meg lehet közelíteni, a megközelítés függ attól is, hogy mennyi adatot kívánunk eltárolni, illetve mennyire szeretnénk biztonságossá tenni azt.

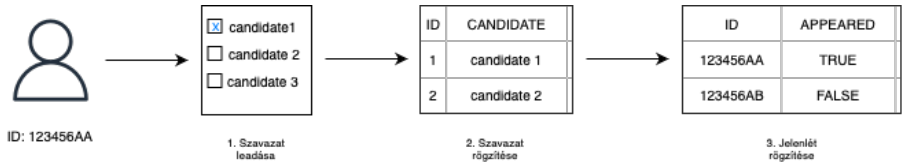
2.1 Digitális választói névjegyzék

Amennyiben a leadott szavazatokat nem kívánjuk egy létező felhasználóhoz társítani, a feladatunk egyszerű, hiszen, ha nem tároljuk el a szavazat és a választópolgár közötti kapcsolatot, akkor ezzel a megoldással egyszerűen biztosíthatjuk a **szavazás titkosságát**.

Ezzel a megoldással viszont nehezebb meghatározni azt, hogy egy adott felhasználónak joga van szavazatának leadására vagy sem, hiszen az országgyűlési képviselők választásáról szóló 2011 évi CCIII. törvénye azt is kimondja, hogy minden állampolgárt **egyenlő választójog** illet meg.

A digitális választói névjegyzék megvalósítása programozási szempontból viszonylag egyszerű. Azt a tényt, hogy egy adott választópolgár már leadta a szavazatát egyszerűen eltárolhatjuk egy relációs adatbázis táblájában egy boolean típusú oszlop értékének true-ra vagy false-ra állításával. Mivel viszont ez az érték egy külön adatbázisban kerül rögzítésre, könnyen előkerülhet az a probléma, hogy miközben a szavazat eltárolásra kerül hiba lép fel szerver oldalon. Ekkor az úgynevezett digitális választói névjegyzékben a „megjelent” oszlop értéke nem változik meg, és így habár a választópolgár már leadta szavazatát, ennek ténye nem kerül eltárolásra.

Ebben az esetben az esetleges hiba helyreállta után, habár a szavazat tárolásra került, a választópolgárnak továbbra is lehetősége nyílik szavazatának újbóli leadására, így programunk ezzel nem teljesí a fentebb említett alaptörvényi kritériumot, miszerint az állampolgároknak joga van az egyenlő és általános választójoghoz, és minden szavazatnak egyenlő értéke van. (*1.ábra*)



1. ábra: Digitális névjegyzékes szavazás sematikus ábrázolása

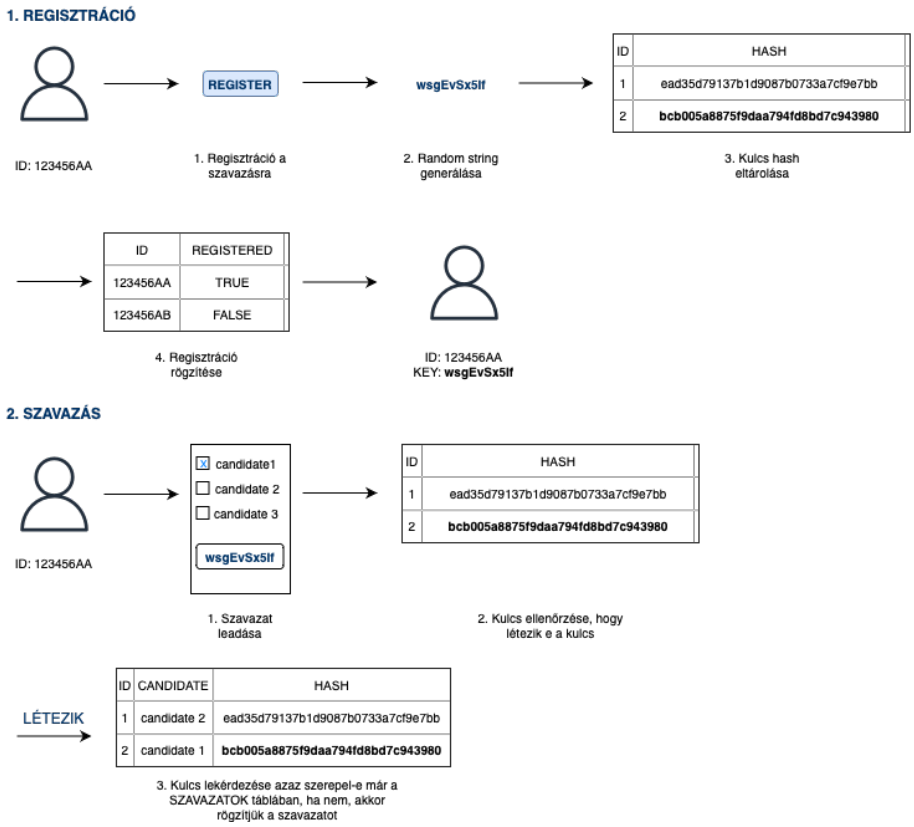
2.2 Random generált aláíró kulcs

Tulajdonképpen ez a választói névjegyzékes megoldás tovább gondolása, hiszen itt is eltároljuk azt, hogy a választópolgár regisztrált-e már az adott szavazásra, viszont a regisztráció után a felhasználó kap egy random generált string-et, amelyet szavazata leadásakor csatol a szavazatához, ezzel azonosítva magát, hogy ő egy regisztrált választópolgár, akinek joga van szavazni.

Maga a regisztráció folyamatát a szerver oldalon úgy lehetne kivitelezni, hogy a szerver generál egy random string-et, melynek az md5 hash értékét eltárolja egy adatbázis táblájában, illetve egy másik táblában eltárolja azt is, hogy az adott felhasználó már regisztrált a szavazásra, így rendelkezik egy azonosító kulccsal, majd egy 200 OK státusz kóddal visszaküldi a generált string-et a kliens oldalra.

A szavazat adattáblába való rögzítésekor egy SQL lekérdezéssel ellenőrizhető, hogy az adott kulcs létezik-e. Amennyiben létezik, akkor azt is megvizsgáljuk, hogy tartozik-e már valamelyik szavazathoz, amennyiben nem, eltároljuk a szavazatot a hozzá tartozó kulccsal együtt majd egy 200 OK státusz kóddal térünk vissza. Abban az eseten viszont, ha már az adott kulcsot felhasználták, akkor nem tároljuk el a szavazatot, és egy 409 Konfliktus (Conflict) státusz kóddal fog visszatérni az API.

A szavazatok összesítésekor a szerver oldalon futó program ellenőrizni fogja, hogy a szavazathoz társított kulcs md5 hash értéke létezik-e az regisztrált kulcsok táblájában, amennyiben létezik, a szavazat elszámolásra kerül, ha viszont nem akkor figyelmen kívül lesz hagyva. (*2.ábra*)



2. ábra: Online szavazás lépései random generált aláíró kulccsal

Az adatintegritás biztosításának érdekében lehetséges egy olyan megoldás, melynél rekordonként eltároljuk a legutóbbi rekord hash értékét, majd ezt a hash értéket használva a szavazatszámláláskor ellenőrizni tudjuk, hogy bármely rekord módosult-e. Amennyiben viszont a lánc megszakadt, akkor feltehetőleg a szavazatokat tartalmazó tábla egy rekordját módosíthatták utólag, tehát csalás történt. A jogtalan módosítások könnyedén elkerülhetőek azzal, ha az írási, illetve módosítási jogokat korlátozzuk az adatbázisban.

2.3 Szimmetrikus titkosítás

A szimmetrikus titkosítás jellemzője, hogy adott egy kriptográfiai kulcs melyet az információ titkosítására, illetve visszafejtésére alkalmaznak. A használt kulcs mérete, illetve a titkosítási procedúra, a tömb méret a használt titkosítási algoritmustól függően változik. Mivel szakdolgozatom szempontjából az AES titkosítás a releváns, ezért fontosnak éreztem a következő szakaszban azt, hogy mi is az AES titkosítás, hogyan működik, illetve miért is jobb ez, mint a korábban használt DES (Data Encryption Standard).

2.3.1 AES titkosítás

Az AES (**Advanced Encryption Standard**) váltotta fel a NIST addigi ajánlását a DES-t, amely a kulcsméretéből eredendően fokozottan ki volt téve a brute force támadásoknak. A kulcsméretet tekintve az AES esetében 128 bit, 192 bit és 256 bites kulcsok közül választhatunk, a DES-el ellentétben, ahol a kulcsméret 56 bites lehet.

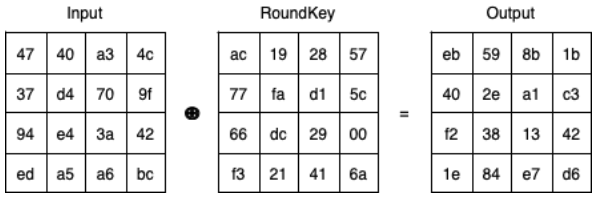
Az AES algoritmus az információt 128 bites tömbökben titkosítja, ez azt jelenti, hogy egy 128 bites bemenet egy 128 bites kimenetet eredményez, amely már a titkosított adat lesz. Az, hogy hány kört futtat végig az adaton, az a kulcsmérettől függ, egy 128 bites kulcs 10 kört, egy 192 bites kulcs 12 kört, míg egy 256 bites kulcs 14 kört jelent. Maga az algoritmus a bemeneti adatot nem bitenként hanem byte-onként dolgozza fel, azaz egyszerre 16 byte-on végez műveleteket. [2]

1. Első lépésként a kulcsból előállítja a RoundKey-eket, azokat a kulcsokat, melyek ténylegesen használva lesznek a titkosítás adott köreiben. Tehát ez egy 128 bites kulcs esetében azt jelenti, hogy a 128 bites (16 bájt) kulcsot, 11 darab egyenként 16 bájt, hosszúságú kulcsokra terjeszti ki.
2. Az első tényleges RoundKey használatával XOR (kizáró vagy) műveletet (*3.ábra*) végez a soron következő 4x4-es mátrix minden egyes bájtján bitenként.

x	y	⊕
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

3. ábra: A XOR (kizáró vagy) művelet igazságtáblája

Gyakorlatban a AddRoundKey alkalmazása egy 16 bájtos blokkra a következőképp a *4. ábrán* látható módon néz ki.

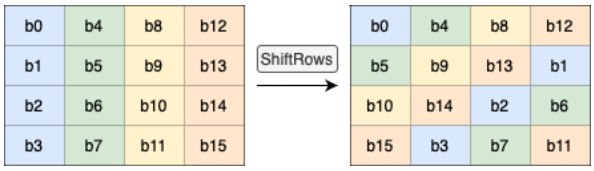


4. ábra: A tömb XOR művelet elvégzése előtt, és után

$$X_{0,0} = 47 \oplus ac = 01000111 \oplus 10101100 = 11101011 = eb$$

$$X_{0,1} = 40 \oplus 19 = 01000000 \oplus 00011001 = 01011001 = 59$$

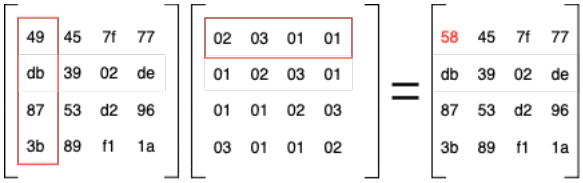
5. Ezután körönként ismétlődő lépések következnek, melyek közül az utolsó körben kimarad a MixColumns lépés:
    a. A SubBytes lépésen minden bájtot lecserélünk egy 8 bites helyettesítési tábla segítségével, melyet Rijndael S-box-nak, illetve AES S-box-nak is nevezünk.
    b. A ShiftRows szakaszban a mátrix sorait különböző mértékben, $n - 1$ bájttal tolja el, így az első sor változatlan marad míg az utolsó sort 3 bájttal fogja eltolni balra. (*5.ábra*)



5. ábra: A 16 bájt elhelyezkedése a ShiftRows művelet előtt és után

    c. A MixColumns lépés keretein belül, minden oszlop megszorzásra kerül egy előre meghatározott mátrixszal.

$$S_{0,0}^{`} = (02 \times 49) \oplus (03 \times db) \oplus (01 \times 87) \oplus (01 \times 3b)$$
$$= 92 \oplus 76 \oplus 87 \oplus 3b = 58$$



6. ábra: A bájtok értékének meghatározása mátrix szorzással

    d. A kör végén az AddRoundKey operátor pedig alkalmazza az adott körhöz tartozó tényleges kulcsot, melyet az eredeti kulcs kiterjesztéséből nyerünk. (*6.ábra*)

A visszafejtés hasonlóképp működik, mint a titkosítás, azzal a módosítással az algoritmusban, hogy a műveleteket visszafelé fogja elvégezni, kulcsmérettől függően 10, 12 vagy 14 kört fog lezajlani a kulcsmérettől függően. A visszafejtés sorrendje:

1. Az adott körhöz tartozó kulccsal az AddRoundKey operátor végrehajtja a XOR műveletet a tömbön.

2. Második lépésként pedig egy a titkosításhoz használt mátrixtól különböző mátrix használatával elvégzi a mátrix szorzást.
3. A ShiftRows művelettel soronként az elemek eltolásra kerülnek az ellentétes irányba, $n-1$-el jobbra.
4. A fordított S-box segítségével pedig, kikeresi az algoritmus azt, hogy mely byte volt használva a SubBytes procedúra keretein belül a titkosításkor.

Az eddigi tudásunk szerint, még nem volt példa arra, hogy az AES titkosítást feltörték volna, biztonságosnak bizonyult a brute force támadásokkal szemben. Ennek ellenére javasolt minél nagyobb kulcsméretekkel dolgozni, annak érekében, hogy a lehető legnagyobb védelmet biztosítsuk a külső támadásokkal szemben.[3] [4]

Az általam megvalósított projektben az **AES-256-CBC** algoritmust fogom használni, ami egy 256 bites kulcsot fog használni, melyet felhasználó által nyújtott sztring az SHA256 hash értékéből fogunk megkapni. A program esetében ez a sztring a felhasználó jelszava lesz. A CBC (**Cipher Block Chaining**) titkosítási mód használatával a titkosítás menete annyiban fog módosulni, hogy a tömbök titkosítása nem végezhető egyszerre, hiszen minden egyes soron következő tömbnek szüksége lesz az előző tömbre az első AddRoundKey művelet elvégzéséhez, így összeláncolva a tömböket. Az első tömb AddRoundKey műveletének elvégzéséhez egy úgynevezett inicializációs vektort fogunk alkalmazni, mely nem más, mint random generált 16 bájt. A körökön belül zajló AddRoundKey műveletekhez viszont már azonos kulcsot fog használni. Mivel az inicializációs vektor random generált, így a titkosított információhoz mindenképp hozzá szükséges csatolunk, annak érdekében, hogy a vissza lehessen fejteni az eredeti üzenetet.

2.3.2 AES titkosítás, mint az anonim szavazás eszköze

Habár az előző szekcióban tárgyalt megoldás használatával képesek lehetünk azonosítani a felhasználót, viszont fennáll annak a lehetősége, hogy a választópolgár elfelejti a szavazatának aláírásához szükséges kulcsot.

Ezt a problémát ki lehetne küszöbölni azzal, ha minden egyes felhasználó aláíró kulcsát eltárolnánk egy adatbázis táblában, viszont így nem teljesülhetne a **szavazás titkosságát** érintő kritérium. Hiszen, ha minden kulcsot el szeretnénk tárolni, akkor a kulcs és választópolgár közötti kapcsolatokat is rögzítenünk kellene, annak érdekében, hogy egy felhasználó ne rendelkezhessen több aláíró kulccsal. Így egy adott szavazathoz csatolt kulcs és a regisztrált

kulcsok táblájában tárolt hash érték alapján könnyen meg lehetne állapítani azt, hogy melyik szavazat mely választópolgárhoz tartozik.

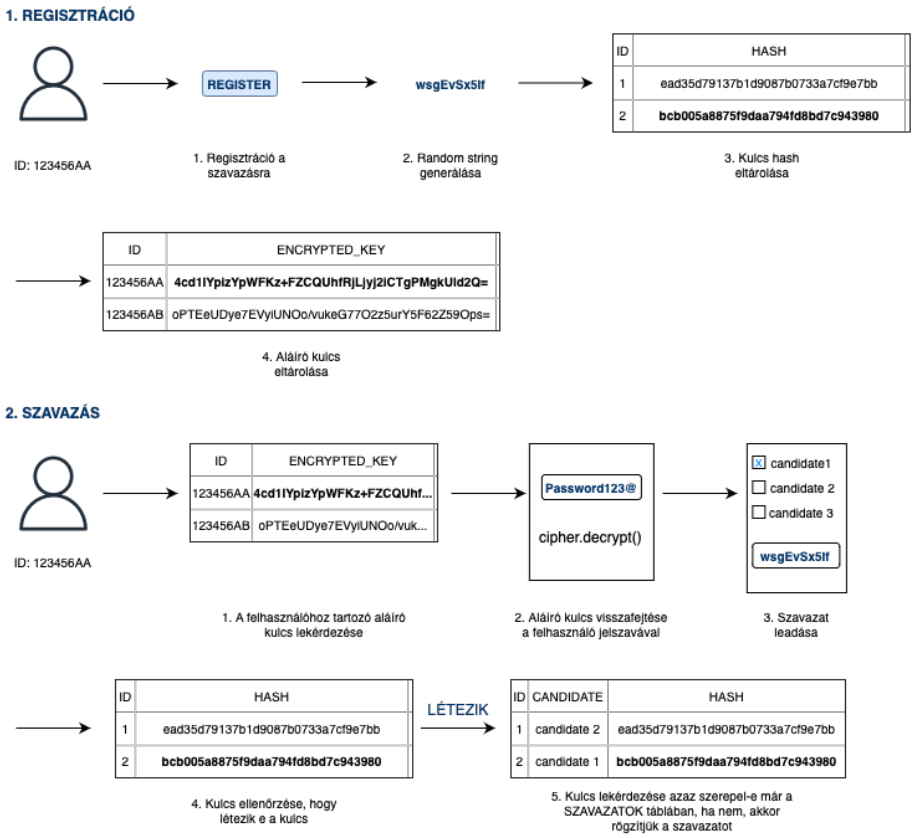
Ennek kiküszöbölésére alkalmazhatunk titkosítást annak érdekében, hogy el tudjuk tárolni a felhasználó aláíró kulcsát, anélkül, hogy identitása nyilvánosságra kerülne.

Erre a felhasználó jelszavát érdemes használni passphrase-ként, hiszen az ez egy olyan adat, ami nem gyakran változik, és ha esetleg változik is, akkor a változtatás alkalmával lehetőségünk nyílik felülírni az adatbázisban lévő aláíró kulcs titkosított értékét.

A felhasználónak így nem szükséges megjegyeznie az aláíró kulcsot, a szavazatának leadása előtt lekérhető lesz az adatbázisból. A lekérdezés után a kliens alkalmazás visszafejti az aláíró kulcsot a felhasználó jelszavának használatával, majd automatikusan csatolhatja azt a szavazathoz. Az előbbi megoldáshoz hasonlóan, a szavazat érvényesítésekor, egy SQL lekérdezéssel ellenőrizhetjük a kulcsot a regisztrált kulcsok között.

Elkerülendő azt az esetet, hogy egy felhasználó több aláíró kulcsot generálhasson magának, célszerű a választásra történő regisztráció alkalmával ellenőrizni azt, hogy a felhasználó már rendelkezik-e regisztrációval és ezáltal egy titkosított aláíró kulccsal.

A példában az **AES-256-CBC** titkosító algoritmus fogom használni, míg titkosító kulcsként a felhasználó jelszavát fogom alapul venni, ami legyen most **Password123@**. (*7.ábra*)



7. ábra: Az online szavazás menete titkosított aláíró kulcs alkalmazásával

2.4 Aszimmetrikus titkosítás

Az aszimmetrikus titkosításról akkor beszélünk, ha felhasználó rendelkezik egy nyilvános/titkos kulcspárral, melyek matematikailag összefüggőek. A titkosítás a kulcspár nyilvános kulcsával megy végbe, melyet a titkos kulcs segítségével lehet visszafejteni. Az aszimmetrikus titkosító algoritmusok közül a legelterjedtebb az RSA titkosító eljárás, melyet Ron Rivest, Adi Shamir és Len Adleman fejlesztett ki (nevük kezdőbetűiből ered az RSA elnevezés). Az eljárás alapjául a moduláris számelmélet és a prímszámelmélet egyes tételei szolgálnak.

2.4.1 RSA titkosítás

Az RSA titkosítás alapjául két random választott nagy prím szolgál, melyekből kiszámolhatjuk a titkosító (*e*), illetve visszafejtésre (*d*) szolgáló kulcsokat. Az RSA kulcspár előállítása x lépésből áll, melyek:

1. Kiválasztunk két nagy prímet, melyet esetünkben p-ként és q-ként fogunk jelölni.

2. Kiszámítjuk a titkosítás alapjául szolgáló modulust, melyet N-nel fogunk jelölni. $N = p \times q$
3. Kiszámítjuk az Euler-féle φ függvény értékét N-re. $\varphi(\mathrm{N}) = (p-1)(q-1)$
4. Választunk egy olyan egész számot, amelyre teljesülnek az alábbi feltételek: $1 < e <$ φ(*N*), és lnko(e, φ(*N*)) = 1. Ezután az *e*-t nyilvánosságra hozzuk az *N*-nel együtt és ezek együtt fogják alkotni a nyilvános kulcsot.
5. Utolsó lépésként kiszámoljuk a visszafejtésre szolgáló titkos kulcsot, melyet az alábbi kongruencia teljesülésének eredményeként kapunk: $d \times e \equiv 1\ (mod\ \varphi(\mathrm{N}))$, majd az így kapott *d* értéket titokban tartjuk.

A kapott értékek segítségével már lehetőségünk van üzenetek titkosítására, illetve visszafejtésére. Az *m* üzenet titkosítása az alábbi egyenlettel oldható meg: $c = m^e\ mod\ N$, míg a titkosított üzenet visszafejtésére az alábbi egyenlet használható: $m = c^d\ mod\ N$. [5]

Programozási szempontból a titkosítandó üzenet *m* általában egy karakter ASCIII kódja, hogy matematikai műveleteket tudjunk vele végezni.

Így például, ha a titkosító kulcsunk értéke *e* = 5, a modulusunk pedig *N* = 23 213, míg a titkosítandó üzenet egy sztring *m* = 'a', akkor megkeressük az ASCIII táblában az 'a' értékét, ami 97 lesz, és ezzel a számmal más képesek leszünk elvégezni az RSA titkosítást.

$$c = m^e(mod\ N) = 97^5(mod\ 23\ 213) = 15\ 889$$
$$m = c^d(mod\ N) = 15\ 889^{13\ 745}(mod\ 23\ 213) = 97$$

2.4.2 RSA titkosítás, mint az anonim szavazás eszköze

Az előző megoldásban az AES titkosítás alkalmazásával még mindig nem küszöböltük ki azt a problémát, hogy így a kulcs a szerver oldalon kell generálódjon, elkerülendő azt a felállást, hogy véletlenszerűen két különböző választópolgár, ugyan azon kulcsot kapja, és mivel a felhasználó jelszavát használjuk az aláíró kulcs titkosításra, így a titkosítás végbemenetelének érdekében, a kliens alkalmazásnak el kellene küldenie azt a szerver számára

Ezt elkerülendő azzal a megoldással álltam elő, hogy minden felhasználó számára generálunk egy RSA kulcspárt a klines oldalon, és a rendelkezésre álló privát kulcs kerül titkosításra a felhasználó jelszavával, még a regisztráció idejében. Ennek csupán egy hátránya van, mégpedig az, hogy mivel csupán a jelszó hash értékét juttatjuk el az adatbázishoz, így nincs lehetőség arra, hogy szerver oldalon ellenőrizzük azt, hogy a jelszó megfelel-e az általános jelszavakat érintő kritériumoknak, mint például annak, hogy a jelszónak

- minimum 8 karakterből kell állnia,

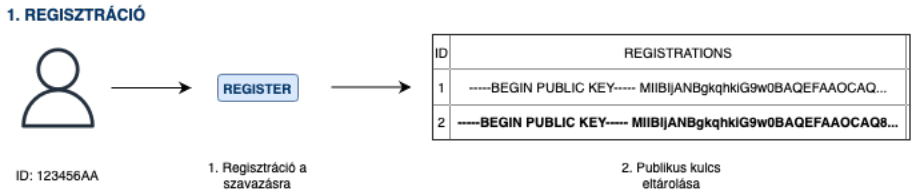
- tartalmaznia kell kis és nagy betűket,
- tartalmaznia kell számokat és betűket egyaránt,
- tartalmaznia kell speciális karaktereket.

Tehát a jelszó formai követelményeinek betartatására minimális lehetőségünk van, csupán kliens oldalon alkalmazhatunk megkötéseket, illetve feleltethetjük meg a felhasználó jelszavát reguláris kifejezéseknek.

Az előző metódusnál használt AES-256-CBC szimmetrikus titkosítást használva kerül rejtjelezésre a generált RSA kulcspár privát kulcsa. Azért szükséges csupán a privát kulcs eltárolása az adatbázisban, mert a privát kulcsból könnyedén visszafejthető a publikus kulcs, melyet a későbbiekben a szavazat hitelesítéséhez fogunk használni.

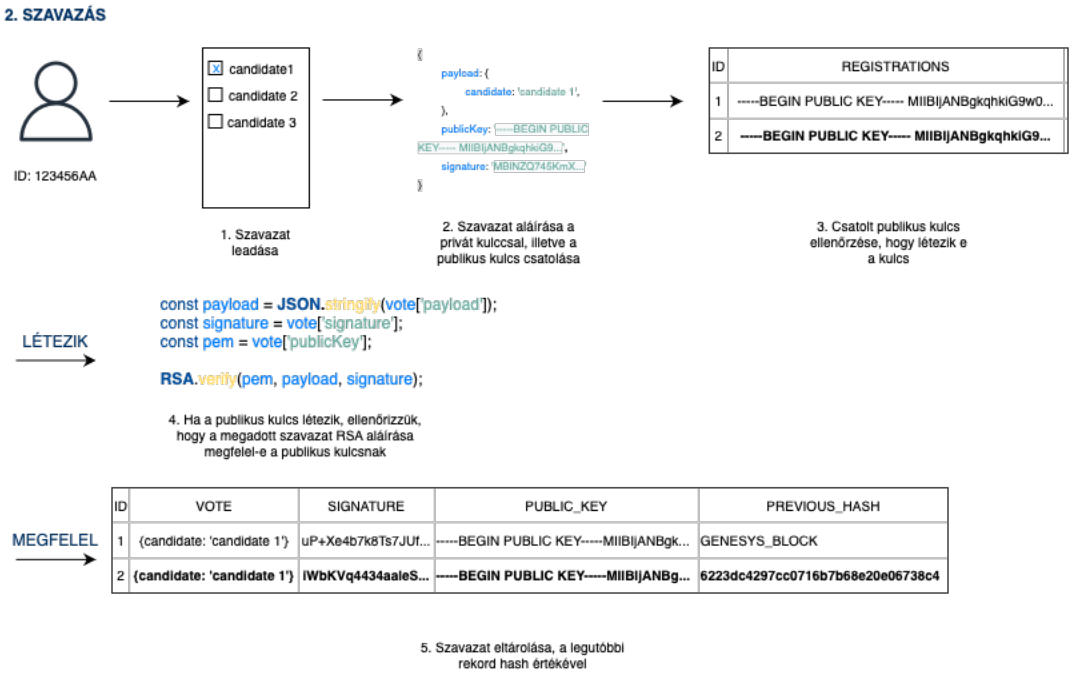
A választói anonimitás érdekében a felhasználói adatok eltárolásánál is titkosítást fogunk alkalmazni, ezt a felhasználó publikus kulcsával fogjuk kivitelezni, tehát a cél az, hogy ha valaki egy pillantást vet az adatbázisra, akkor se tudjon meg többet a felhasználóról, mint a személyi számának, illetve e-mail címének hash értéke.

A regisztráció során generált RSA titkosító kulcsunk lesz a jövőbeli azonosítás kulcspontja. Segítségével fogjuk igazolni azt, hogy a szavazat, egy az adott választásra regisztrált felhasználótól származik. (*8. ábra*)



8. ábra: RSA publikus kulcs regisztrálása

A szavazás során egy olyan JSON Objektumot fog a kliens alkalmazásunk a szerver felé küldeni, amely tartalmazni fogja a „szavazat” Objektumot, annak a privát kulcs által aláírt signature-jét, illetve magát a publikus kulcsot String-ként, amelyet a választásra való regisztrációkor is használtunk. Az HTTP kérés Header része tartalmazni fogja továbbá a felhasználó számára generált JWT token-t, amely információt biztosít a felhasználó településéről, illetve ezáltal arról, hogy melyik választókerületbe tartozik. A JWT token módosítása nem lehetséges a klines oldalon. (*9.ábra*)



9. ábra: A szavazás menete RSA kulcsok alkalmazásával

A szavazat feldolgozásakor, a szerver oldalon futó program előbb ellenőrizni fogja azt, hogy a szavazathoz csatolt publikus kulcs szerepel-e a választásra regisztrált publikus kulcsok között, amennyiben nem, akkor a szavazat rögzítése nélkül egy 400-as HTTP response kóddal fog visszatérni.

Ha viszont a publikus kulcs regisztrálva van a választásra, akkor második lépésként ellenőrizni fogja azt, hogy szerepel-e már szavazat az adott publikus kulccsal, amennyiben igen, akkor 409 Conflict response kóddal fog visszatérni az HTTP kérés, viszont, ha nincs még szavazat az adott publikus kulccsal, akkor tovább lép a program.

A következő körben azt fogja ellenőrizni, hogy a privát kulccsal aláírt objektum, azaz a Signature megfelel-e a publikus kulcsnak, tehát a szavazatot leadó fél, rendelkezett-e a privát kulccsal szavazatának leadásakor, amennyiben nem hitelesíthető az aláírás, akkor 400-as hibakóddal fog visszatérni, hiszen feltehetőleg a szavazatot nem a publikus kulcshoz tartozó privát kulccsal rendelkező fél adta le, hanem valaki olyan, aki csak a publikus kulcsot tulajdonította el.

Hogyha viszont a signature hitelesíthető, a program ellenőrizni fogja azt, hogy a felhasználó által leadott szavazat a felhasználó körzetébe tartozó egyéni képviselőjelölt-re lett leadva, amennyiben nem, a kérés szintúgy 400-as hibakóddal fog visszatérni, viszont, ha megfelelő jelölre érkezett az egyéni választókerületében, akkor 200-as OK response kóddal fog visszatérni, és a szavazat rögzítésre kerül.

A szakdolgozatom keretein belül létrehozott online szavazórendszerben az alábbi folyamatot fogom alkalmazni a szavazatok rögzítésére, és eltárolására.

3. Felhasznált technológiák

Az általam létrehozott alkalmazással, a felhasználónak lehetősége nyíliik szavazatának leadására az anonimitás, illetve annak integritása mellett. A felhasználói felület az alábbi műveletek elvégzésére ad lehetőséget a felhasználó számára:

a. Fiók létrehozása egy random generált regisztrációs kód segítségével, melyet csupán admin hozzáférhetőséggel lehet generálni, és a regisztrációs folyamat befejeztével lejár, és törlésre kerül az adatbázisból.
b. Bejelentkezésre, mely folyamat során a felhasználónak a regisztrált e-mail címre és a jelszójára van szüksége, melyek helyessége esetén a szerver 200 OK válasszal tér vissza, és beállítja a szükséges kliens oldali Cookie-kat.
c. Felhasználói adatai konfigurálására, az e-mail címének és jelszójának módosítására.
d. Szavazatának leadására, melynek érvényességét a szerver oldalon ellenőrzik majd hozzáadják a szavazatok gyűjteményéhez.

Az alkalmazás elkészítéséhez az alábbi technológiákat hívtam segítségül:

3.1 Node.js

A Node.js egy olyan szoftverrendszer [6], mely segítségével esemény alapú aszinkron alkalmazások készíthetőek a JavaScript nyelv használatával. Manapság leginkább skálázható internetes alkalmazások, webszerverek készítésére használják. A rendszer Ryan Dahl hozta létre 2009-ben és nagyrészt implementálja a CommonJS alapelveket. Ma már egész fejlesztői közössége van, további fejlesztése a GitHubon zajlik.

3.2 Next.js

A Next.js keretrendszert [7], a Vercel csapata fejlesztette ki, ami egyben egy hosting szolgáltatás, amelyet Next-js alapú weboldalak hostolására optimalizáltak. A React kiterjesztésével lehetőségünk nyílik a komponenseink szerver oldali renderelésére, így a React alapú weboldalunkat könnyedén SEO optimalizálhatjuk, habár ez webalkalmazásoknál kevésbé, de statikus weboldalaknál igazán hasznos, ha a weboldalt mindenképp React

használatával szeretnénk megvalósítani. Továbbá a Next.js keretrendszeren belül lehetőségünk nyílik API route-ok definiálására, így megszüntetve a vanilla React alkalmazások egyik szerintem elég nagy problémáját, hogy az API és a webalkalmazás kódját külön kell kezelnünk, általában egy express.js és egy vanilla React projektben.

3.3 React

A Next.js keretrendszer egyik fontos eleme a React keretrendszer [8], melyet a Meta csapata fejlesztett ki, és jelenleg is a Meta és független fejlesztők javaslatai alapján fejlődik tovább. A React lehetőséget ad reaktív felhasználói felületek építésére JSX elemek használatával. A keretrendszer régebbi verzióiban osztály alapú komponenseket lehetett készíteni, de a modern React lehetőséget ad arra, hogy függvény alapú komponenseket írjunk, mely csökkenti az egy funkció implementálásához szükséges kódmennyiséget.

3.4 react-dom

A react-dom könyvtár DOM azaz Document Object Model specifikus kódokat tartalmaz, segítségével egy id által definiált elembe rendelhetjük komponenseinket. Ezen felül további feladatokra használhatjuk, magában a projektemben ez a Notification komponensnél jelenik meg a createPortal függvény melynek használatával egy előre definiált DOM elembe fogja renderelni az adott komponenst.

3.5 react-switch

A react-switch csomag egy a mobilon futó alkalmazások switch-jéhez hasonló user interface elemet biztosít. Implementálása nem jelentene különösebb fejtörést, csupán a fejlesztés felgyorsítása indokolja használatát.

3.6 cookies-next

A cookies-next egy Next.js specifikus könyvtár mely segítségével cookie-kat helyezhetünk el a kliens oldalon. Az alkalmazásomban a token klines oldali tárolására, illetve a dark mód állapotának tárolására fogom használni. Mivel a cookie-k esetében meg lehet adni a lejárati

időt, így a token lejárta után, miután használhatatlanná vált, a böngésző automatikusan törölni fogja azt a tárhelyéből. A cookie-k tárolásának szempontjából fontos kiemelni, hogy méretét tekintve 4096 bájtban van meghatározva a felső határ, ezért bizonyos felhasználói adatokat, melyekre klines oldalon is szükség lehet, külön lesznek eltárolva.

3.7 md5

Az md5 könyvtárat használunk a jelszavak, a blokkláncban az előző blokk hash értékének kiszámítására. Szerver és kliens oldalon egyaránt használjuk, kliens oldalon palául a jelszó titkosítására, vagy a regisztráció ellenőrzésére, hogy ne kelljen a teljes publikus kulcsot elküldeni az HTTP kérésben paraméterként.

3.8 node-forge

A node-forge package-et a privát és publikus kulcsok olvasására, létrehozására, és az adatok velük történő titkosításra és visszafejtésére használom. Ezt a könyvtárat, illetve az aes-cipher module elemeit használom az rsa.js fájlban melyben, ha az RSA osztály getPrivateKeyPem metódusában megadunk egy passphrase-t, akkor a generált RSA privát kulcsot AES-256-CBC titkosítással fogja exportálni.

3.9 crypto-js

A crypto-js könyvtárra az AES titkosítás megvalósítására van szükségünk. Segítségével érem el az AES-256-CBC titkosítás implementálását, melyet az aes-cipher.js fájlban láthatunk.

3.10 jsonwebtoken

A jsonwebtoken egy MIT licensz alatt futó node csomag, mely az RFC 7519 szabvány [9] node-js implementációja, segítségével olyan tokeneket készíthetünk, melyek egy header-ből és egy payloadból állnak.

A header általában két részből áll, az egyikben definiáljuk az aláírásához használt algoritmust, és alg néven találjuk meg, míg a másikban a token típusát nevezzük meg, ez a mi esetünkben jwt lesz, ami a JSON Web Token rövidítése.

A palyload részben claim-eket nevezünk meg, amelyek lehetnek regisztrált claim nevek, mint például iss, iat, exp, illetve lehetnek publikus és private claim-ek.

A signature részben a szerver oldalon tárolt kulccsal aláírjuk a header-ben megnevezett algoritmus segítségével a payload-ban található claim-eket, ezzel az aláírással fogja később a felhasználó igazolni azt, hogy az adott token-t a szerver bocsájtotta ki.

A jsonwebtoken verify függvénye abban az esetben tér vissza a dekódolt claim-ekkel, ha a token nem járt le, illetve a szerver által használt kulccsal volt aláírva, tehát az aláírás érvényes.

3.11 mongodb

A mongodb könyvtár segítségével tudunk kommunikálni a MongoDB [10] adatbázisunkkal, amely egy no-sql adatbázis, a projekt nagyságát tekintve nem láttam szükségét sql adatbázis alkalmazásának. Amennyiben gyorsabb lekérdezési időt kívánunk elérni, illetve nagyobb webes forgalomra számítunk, akkor érdemesebb lenne átírni a backend-et, úgy, hogy az egy sql adatbázist használjon.

A no-sql adatbázis további előnyeiről beszélve, fontos megemlíteni, hogy az adatbázissal való kommunikáció nem igényel bonyolult lekérdezések megírását, egyszerűen, a MongoClient metódusait használva kommunikálhatunk az adatbázissal.

Az adatok tárolásának módja szinte megegyezik a JavaScript-ben létező adattípusokkal, megkönnyítve ezzel a MongoDB használatát számunkra. A struktúráját tekintve egy JSON szerű dokumentum alapú állományt kell elképzelni, ezeket az adatokat tárolja el úgynevezett kollekciókban.

4. A fejlesztett alkalmazás bemutatása

Szakdolgozatom ezen fejezetében, az általam a 2.4.2 részben definiált anonim szavazási módot valósítom meg, harmadik fejezetben felsorolt technológiák alkalmazásával, az választott JavaScript programozási nyelven, illetve a Next.js keretrendszer alkalmazásával.

4. 1 Az adatbázis struktúrája

Az adatbázis kialakításához, ahogyan azt már a harmadik fejezetben kiemeltem, mongodb alkalmaztam, ennek oka a projekt mérete. A fejlesztett alkalmazásnál felmerülő adatok eltárolására négy kollekciót hoztam létre, melyeknek az alábbiak.

4.1.1 codes

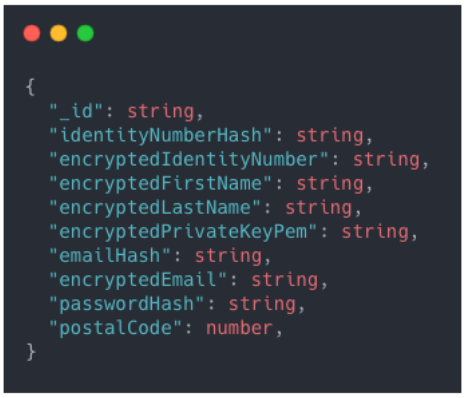
A **codes** kollekcióban kerülnek eltárolásra azok a regisztrációhoz szükséges, csupán egyszer felhasználható kódok, melyeket az /api/v1/auth/code api endpoint-ra küldött HTTP kérés segítségével lehet generálni. Az így generált kód egy 20 byte nagyságú, Base64 kódolású sztringet lesz lehetősége a felhasználónak a regisztráció során megadni. A sikeres regisztráció végeztével ez az egyszer használatos kód törlésre kerül. Az adatbázisban az adott dokumentum három sztring attribútumot fog tartalmazni. Egy automatikusan generált _id-t, egy code-ot, amely regisztrációs kód hash értéke, illetve a kód igénylésének idejét ISO 8601 formátumban. (*10.ábra*)

10. ábra: Egy code dokumentum struktúrája

4.1.2 users

A **users** kollekcióban kerülnek eltárolásra a felhasználóval kapcsolatos ingformációk. Mivel a létrejött információk csupán a felhasználó számára relevánsak, ezeket a felhasználó privát kulcsával fogjuk titkosítani, három adat kivételével, melyekre a felhasználók létrehozásánál szükségünk lesz.

Ilyen adat a választópolgár személyigazolvány számának hash értéke, melyre azért lesz szükségünk, hogy ellenőrizhessük azt, hogy a felhasználó regisztrált-e már a szavazó rendszerbe. Hasonlóan járunk el az e-mail cím eltárolásánál is, viszont itt mind a hash értéket, és a felhasználó privát kulcsa által titkosított e-mail értékét el fogjuk tárolni, elkerülve azt a szituációt, hogy két felhasználó azonos e-mail címet adjon meg. Továbbá a felhasználó jelszava olyan adat, aminek csupán a hash értékét tároljuk el, hiszen a bejelentkezés alkalmával a felhasználó által elküldött email, illetve jelszó hash párost fogjuk ellenőrizni, hogy találunk-e egyezést az adatbázisban. (*11.ábra*)



11. ábra Egy user dokumentum struktúrája

4.1.3 settlements

A settlements kollekcióban a választásokban résztvevő összes település van eltárolva. Azért ezt a megoldást választottam, mert a választókerületek megváltoztatásakor csupán a benne szereplő települések változnak, maga a település adatai nem. Egy settlement dokumentumban két sztring és egy szám típusú adat kerül eltárolásra. Az _id ismételten egy random generált azonosító, a postalCode attribútum a településhez tartozó irányítószám, a name pedig az adott település neve. Az election dokumentumok létrehozásakor, minden település irányítószáma a megfelelő választókerületbe kerül majd besorolásra. (*12.ábra*)

12. ábra Egy settlement dokumentum struktúrája

4.1.4 elections

Az elections kollekcióban árulkodó nevéből is következtetve a választások vannak eltárolva. Maga az election dokumentum szerkezetét tekintve elég tömör, egy élesben működő választási rendszer esetében javasolt lenne az attribútumok többségét SQL adatbázis esetében külön adatbázis táblában, míg no-SQL adatbázis esetében külön kollekcióban tárolni. (*13.ábra*) Magában az election dokumentumban kerülnek eltárolásra az adott választáshoz tartozó választókerületek, hiszen mint azt a 4.1.3-as szekcióban említettem a választások alkalmával változhatnak a választókerületek határai, ezáltal előfordulhat, hogy különböző választásokkor különböző választókerületbe kerülnek egyes települések. A szavazásra való regisztrációkat is az election dokumentumban tároltam el, amely egy blockchain-hez hasonló metódussal kerül eltárolásra, hiszen minden soron következő rekord tartalmazza az előző rekord md5 hash értékét, így biztosítva azt, hogy ne kerüljön törlésre egy regisztráció sem. Ha viszont mégis sor került a regisztrációk listájának módosítására, akkor erre a blockchain validáció esetében fény derül. A szavazatok tárolásának módja teljes mértékben megegyezik, minden rekord tartalmazza az előző rekord hash értékét.

```
{
  "_id": string,
  "name": string,
  "registration": {
    "start": string,
    "end": string
  },
  "voting": {
    "start": string,
    "end": string
  },
  "candidates": [{
    "_id": string,
    "firstName": string,
    "lastName": string,
    "identityNumberHash": string,
    "electionDistrictId": string || null,
    "partyListId": string || null,
    "placeOnPartyList": string || null
  }],
  "partyLists": [{
    "_id": string,
    "name": string,
    "color": string,
    "placeOnBallot": number
  }],
  "districts": [{
    "_id": string,
    "name": string,
    "postalCodes": Array[string]
  }],
  "registrations": [{
    "_id": number,
    "previous": string,
    "publicKey": string,
    "publicKeyHash": string,
  }],
  "votes": [{
    "_id": number,
    "previous": string,
    "vote": {
      "candidateId": string,
      "districtId": string
    },
    "signature": string,
    "publicKey": string,
    "publicKeyHash": string
  }]
}
```

13. ábra Egy settlement dokumentum struktúrája

4.2 Az API endpoint-ok bemutatása

A Next.js keretrendszer adta lehetőségeknek köszönhetően lehetőségünk van API endpoint-ok definiálására magában a Next.js projekten belül, így a frontend és backend kódunkat nem kell külön projektben megvalósítani. Az API endpoint-ok létrehozása egy api elnevezésű mappa létrehozásával lehetséges a projekt pages mappájában.

4.2.1 api/v1/auth

Az auth mappában három endpoint van definiálva, ezek mind az felhasználói fiók működésével és létrehozásával állnak kapcsolatban.

A regisztráció esetében felhasználói adatok a megadásuk után klines oldalon kerülnek titkosításra, ezáltal a backend ezeket az adatokat már titkosítottan Base64 kódolású szövegként fogja megkapni. Tehát ezen adatok ellenőrzése már nem lehetséges a backend-en, csupán eltárolásra kerülnek.

A regisztrációkór továbbá szükség van egy regisztrációs kód megadására, amelyet csupán vagy admin jogosultságú token birtoklásával, vagy a szerver privát kulcsával generált aláírás megadásával lehetséges. Ezeket a regisztráció befejezéséig fogja tárolni az adatbázis a *codes* kollekcióban, majd törlésre kerülnek.

A bejelentkezés a regisztrációhoz hasonlóan egy HTTP POST kéréssel történik, a kérés body részében egy JSON objektum megadása szükséges, mely rendelkezik egy email és egy passwordHash attribútummal. A passwordHash értéke a felhasználó jelszavából számított md5 hash, melyet a szerver oldalon összehasonlítja az adatbázissal tárol értékkel. Amennyiben minden megadott adat érvényes, az endpoint HTTP 200 OK válaszkóddal tér vissza, illetve klines oldalon létrehozza a felhasználó adatait tartalmazó Cookie-kat egy session tokennel egyetemben. A session token az igénylés dátumát, az igénylésének az ideje plusz egy órás lejárati időt, illetve a választópolgár irányítószámát fogja tartalmazni. (*14. ábra*)

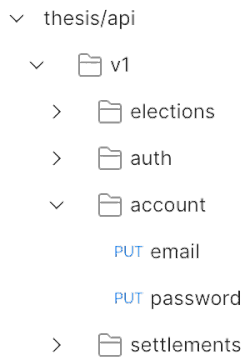
14. ábra: Az auth API endpoint és alpontjai

4.2.2 api/v1/account

Az account API végponton belül megtalálható email és password végpontok által lehetőségünk van az e-mail cím és a jelszó módosítására.

Mivel a privát kulcs a felhasználó jelszavával van titkosítva, így a jelszó módosítása esetén nem csupán a jelszó hash értékének módosítása szükséges az adatbázisban, hanem az AES-256-CBC titkosítással ellátott privát kulcs pem-et is frissítenünk kell.

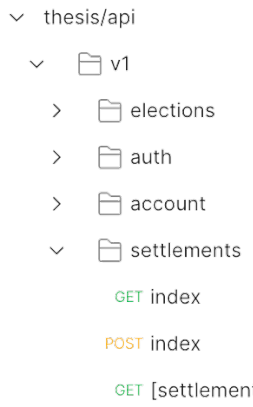
Az email módosításakor már könnyebb dolgunk van, hiszen az email esetében csak az email-t és annak titkosított értékét kell eljuttatnunk. Mivel az email nem annyira érzékeny adat, mint a jelszó így az email hash értékének kiszámítása akár szerver oldalon is megtörténhet, ezáltal lehetőségünk van reguláris kifejezések segítségével ellenőrizni, hogy az adott email megfelel-e az email címet övező kritériumoknak. (*15. ábra*)



15. ábra: Az account API endpoint és alpontjai

4.2.3 api/v1/settlements

A settlements végpont segítségével HTTP POST kérés küldése esetén településeket hozhatunk létre, GET kérés esetén pedig le kérdezhetjük a settlements kollekcióban szereplő települések listáját. A település dokumentumokat a 4.1.3 szekcióban bemutatott struktúrában tárolja el. (*16.ábra*)



16. ábra: Az settlements API endpoint és alpontjai

4.2.4 api/v1/elections

Az elections végpont a legkomplexebb, egy az api/v1/elections végpontra küldött kéréssel választást hozhatunk lére, illetve kérdezhetünk le.

A dinamikus route paramétereknek köszönhetően lehetőségünk van, egy-egy választást beazonosítani, majd módosításokat végezni rajtuk. Azon paraméterek esetében melyek módosulhatnak, lehetőségünk van teljes CURD-ot végezni, és mivel egy dokumentumban van eltárolva az egész választás, mint objektum, ezért egyes listák bővítése gyakran be fog következni.

4.2.4.1 Választókerületek rögzítése

A választókerületek rögzítésekor szükségünk lesz a korábban rögzített településekre, hiszen minden egyes választókerületben el lesz tárolva azoknak a településeknek az irányítószáma, melyek az adott választókerületbe tartoznak. Ez igazából a szavazatok összesítéséhez szükséges, hogy minden egyes szavazatot egy választókerülethez tudjunk társítani. Amennyiben a HTTP POST kérés törzsében megadott település irányítószáma nem létezik a settlements kollekcióban, akkor 404 Not found válasszal fog visszatérni a backend.

4.2.4.2 Pártlisták rögzítése

A párlisták rögzítésekor figyelembe veszi a program, hogy létezik-e már pártlista azonos névvel a pártlisták körében, amennyiben igen, akkor 409 Conflict válasszal fog visszatérni az HTTP kérés, amennyiben viszont nem, akkor rögzítésre kerül a pártlista. (*13. ábra*)

4.2.4.3 Képviselőjelöltek rögzítése

A képviselőjelölteknél lehetőségünk van definiálni egy placeOnPartyList és egy partyListId paramétert, melyek azt fogják elárulni a backend számára, hogy az adott id-val rendelkező párt listáján hányadikként szerepel a rögzíteni kívánt képviselőjelölt. Amennyiben ez nincs megadva akkor úgy veszi a program, hogy csak egyéni választókerületben szerepel az adott jelölt, így csak az electionDistrictId paramétert fogja megadni, mely az adott választás egyik egyéni választókerületére utal.

Ha viszont mind a három paraméter meg lett adva azt az eshetőséget úgy fogja kezelni programunk, hogy mind egyéni választókerületben indul, és pártlistán is szerepel az adott jelölt. Így a szavazatok rögzítésekor, ha az adott jelölt nem győz a saját választókerületében, még mindig van lehetősége arra, hogy saját pártjának listájáról bekerüljön.

4.2.4.4 Regisztrációk rögzítése

A regisztrációk rögzítésekor figyelembe vesszük az előző regisztrációt, hiszen ahogyan azt a 4.1.4-es szekcióban is említettem a regisztrációk eltárolásakor az korábbi regisztráció md5 hash-ét is eltároljuk. A regisztrációk törlése, illetve módosítása viszont nem lehetséges, csupán lekérdezni lehet, illetve a publikus kulcs hash értékével lekérdezni.

Fontos kiemelni, hogy a választásra történő regisztráció csak a regisztrációs időszakban lehetséges.

4.2.4.5 Szavazatok rögzítése

A szavazás megvalósítása a regisztrációhoz hasonló, azzal a különbséggel, hogy a program ellenőrizni fogja azt, hogy az adott választásra regisztrált-e a szavazatát leadó felhasználó. A szavazat leadása a regisztrációhoz hasonlóan a szavazási időszakban lehetséges. A szavazatok esetében se végezhető teljes CURD a szavazatokon, csupán létrehozni, és lekérdezni lehetséges, a felhasználó publikus kulcsának hash értékével.

4.2 A felhasználói felület bemutatása

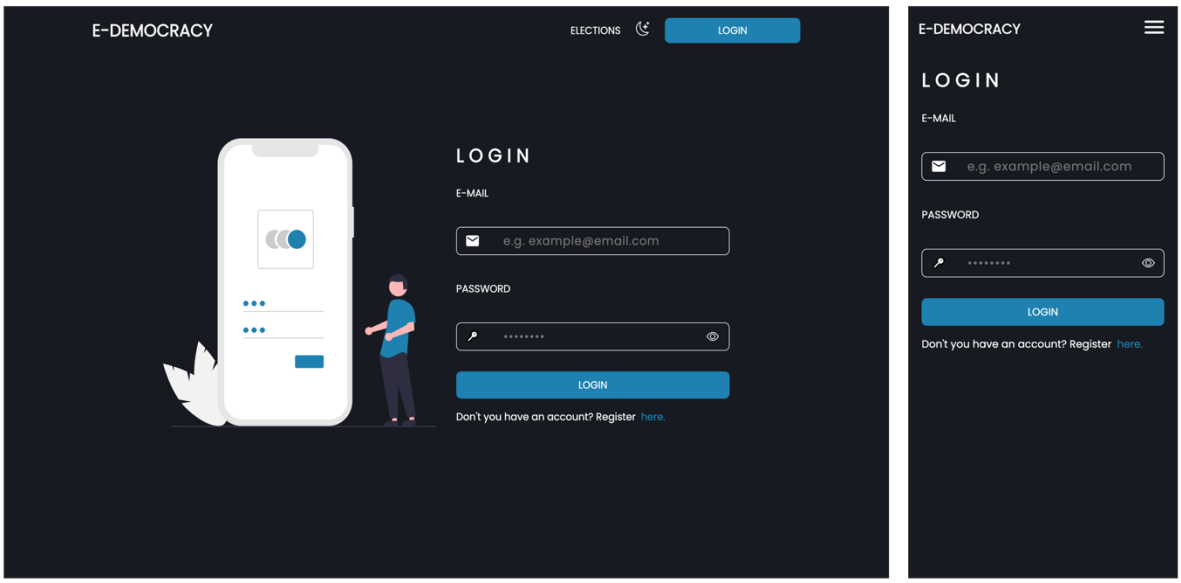
A webalkalmazásom frontend részét a backend-hez hasonlóan, a Next.js nyújtotta keretrendszeren keresztül valósítottam meg. A felhasználói felülethez használt technikák között kiemelkedő szerepet kapott a React keretrendszer.

4.2.1 /login

A bejelentkezési oldalon számos újra használható komponens jelenik meg, melyek css modulokból nyerik el a képen látható formájukat. A formula kitöltése után a klines oldal egy HTTP GET kérést küld a szerver *api/v1/auth/login* végpontjára, mely a következő adatokat tartalmazza: email-cím, illetve a felhasználó jelszavának hash értéke.

A beviteli ’password’ típusú beviteli mezőnek adtam egy olyan funkciót, melyet a legtöbb ma ismertebb weboldalak mindegyike tartalmaz, a felhasználónak lehetősége van megjeleníteni, illetve elrejteni jelszavát.

Amennyiben a megadott bejelentkezési adatok hibásak, a szerver 403 Forbidden válasszal tér vissza. Ezt a felhasználóval viszont vizuálisan is szükséges közölnünk, ezt egy úgynevezett notification üzenettel oldottam meg, amely az oldal alján válik láthatóvá, majd 3 másodperccel a megjelenése után automatikusan eltűnik. Magáról a Notification komponensről és a NotficactionConext-ről a *4.3.1.3*-as szekcióban fogok részletesebben beszámolni. (*17.ábra*)

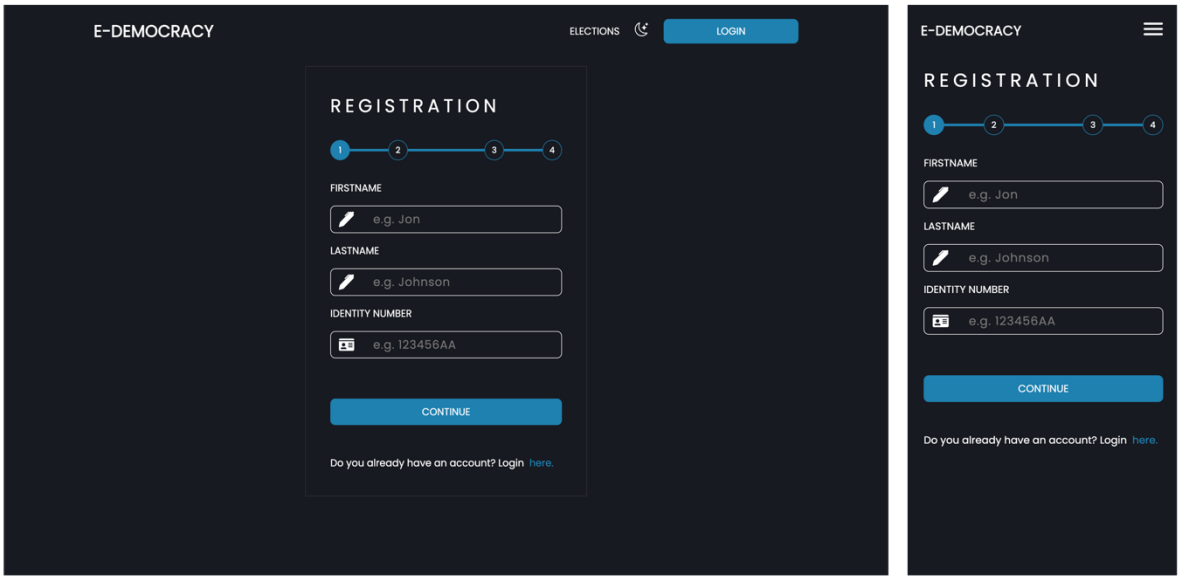


17. ábra: A /login oldal mobil nézettel

4.2.2 /register

A regisztrációs oldalon egy több oldalas formulát készítettem, mely kialakítást azért tartottam fontosnak, hogy könnyített legyen a felhasználó számára a kitöltése. Ezzel a megoldással minden egyes lépés után ellenőrzésre kerül a bevitt adat, és amennyiben nem felel meg a reguláris kifejezésnek, akkor nem engedi továbblépni a felhasználót.

Hasonlóan jár el abban az esetben is, ha a felhasználó beküldi adatait viszont nem megfelelő az általa megadott regisztrációs kód, amit viszont már szerver oldalon ellenőrzünk. Lehetőség van a kérdések közötti lépegetésre, viszont amennyiben a jelszó megadása után lép vissza, akkor a jelszó beviteli mezők értékei törlésre kerülnek. (*18.ábra*)

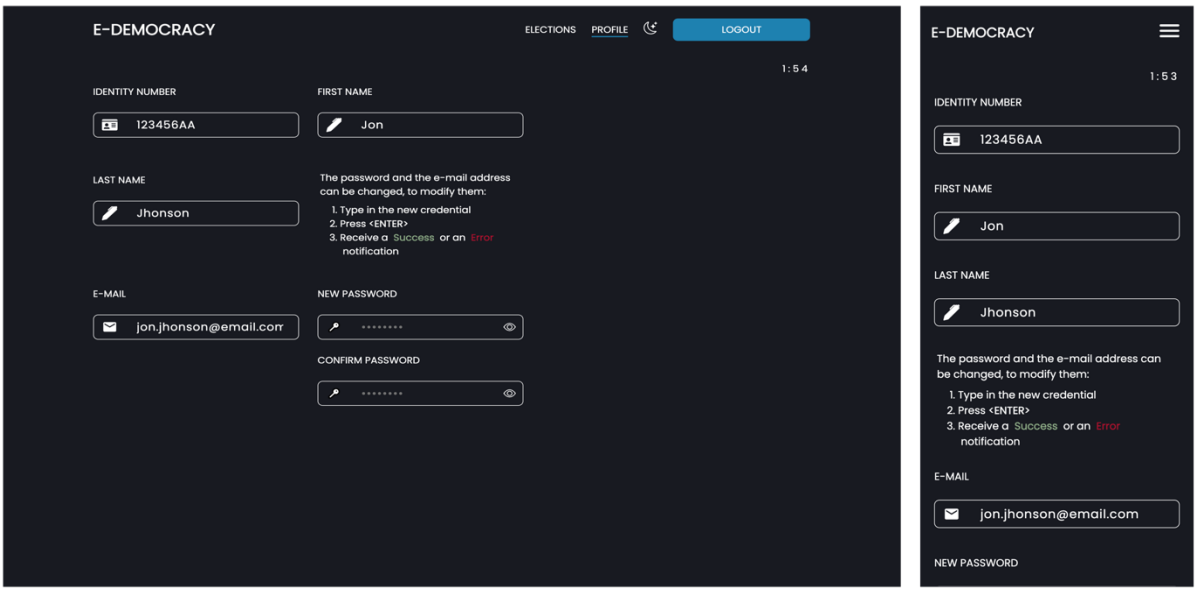


18. ábra: A /register oldal mobilnézettel

4.2.3 /account

Az */account* oldalon a felhasználónak lehetősége van felülírni bizonyos adatokat. Ilyen adatok a felhasználó jelszava, illetve email címe. Az oldalra való látogatáskor egy Modal komponenst dob fel, melyen bekéri a felhasználó jelszavát. Erre azért van szükség, hogy a felhasználó jelszavát használva vissza tudja fejteni az *AES-256-CBC* móddal titkosított privát kulcsot. Erre a felhasználói adatok visszafejtésére van szükségünk, illetve jelszóváltoztatás esetén felül kell írnunk a titkosított privát kulcs értékét is. Ehhez újból titkosítani kell az AESCipher osztály felhasználásával az új jelszót használva passphrase-ként.

Továbbá létrehoztam egy Timer komponenst, melynek célja az, hogy ha összefüggően két percen keresztül nem végez a felhasználó interakciót az adott oldalon, akkor lezár és újból jelszavának megadására szólítja fel azt a Modal komponens által. Ennek az időtartamnak a vizualizálást szolgálja a Timer komponens. A jelszó beírása után a visszafejtett adatokat egy state-ben tárolja, melynek értékét az idő esetleges lejárta után újból null értékre állítja. (*19.ábra*)

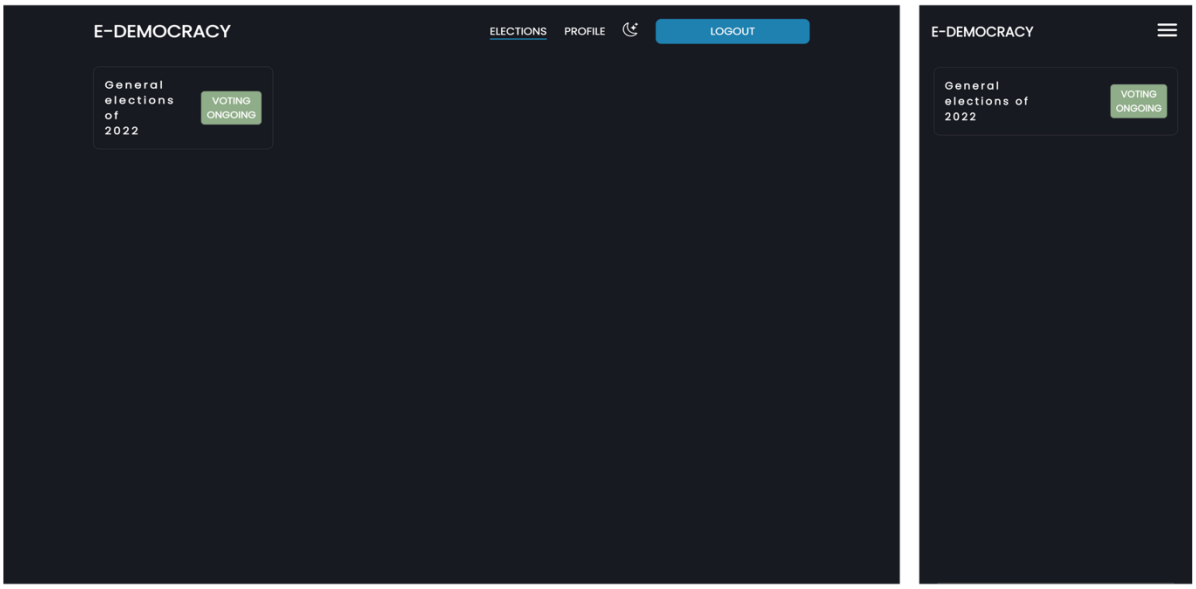


19. ábra: A /account oldal mobilnézettel

4.2.4 /elections

Az /elections oldal egy gyűjtő oldal, melyekről lehetősége van a felhasználónak az adott választás oldalára navigálni. A választások listája egy react hook segítségével fog megérkezni a szerverről. Mivel csupán alapvető adatokat akarunk mutatni az adott soron következő vagy már lezárult választásokról, mint a választás aktuális státusza, amely lehet [’Upcoming’, ’Registration ongoing’, ’Registration ended’, ’Voting ongoing’, ’Ended’] státuszú, a election dokumentum registration és voting paraméterei alapján megadva

A RegistrationPreview komponensre kattintva átirányít az adott választás oldalára, melyhez a Next.js keretrendszer beépített Link komponensét fogjuk használni, ezzel elkerülve az oldal újra töltését. (*20.ábra*)

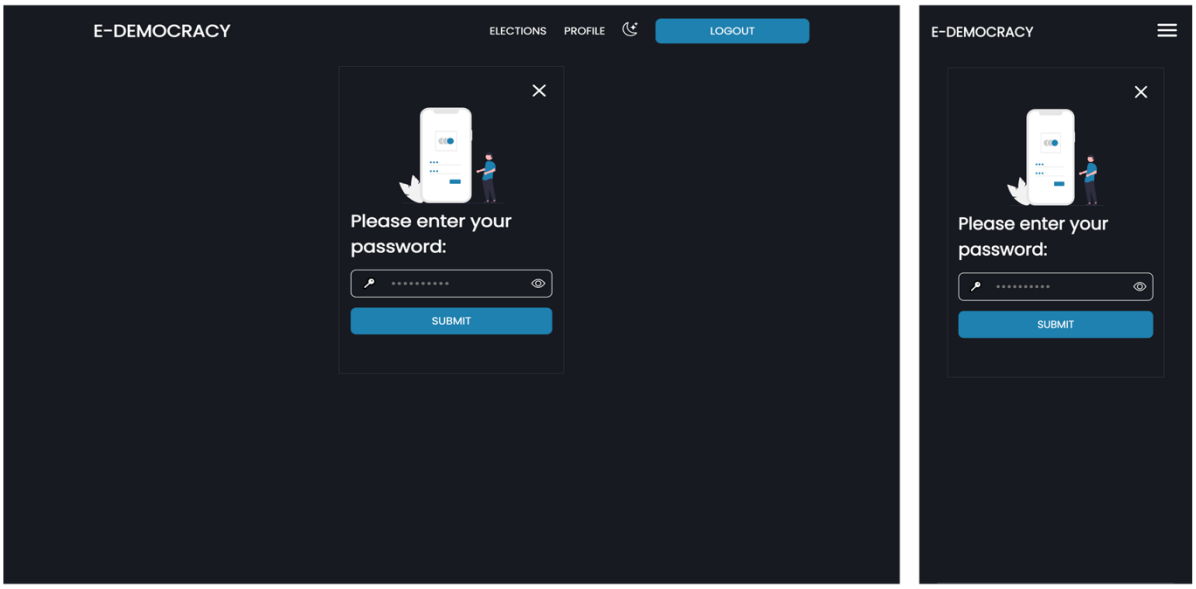


20. ábra: A /elections oldal mobilnézettel

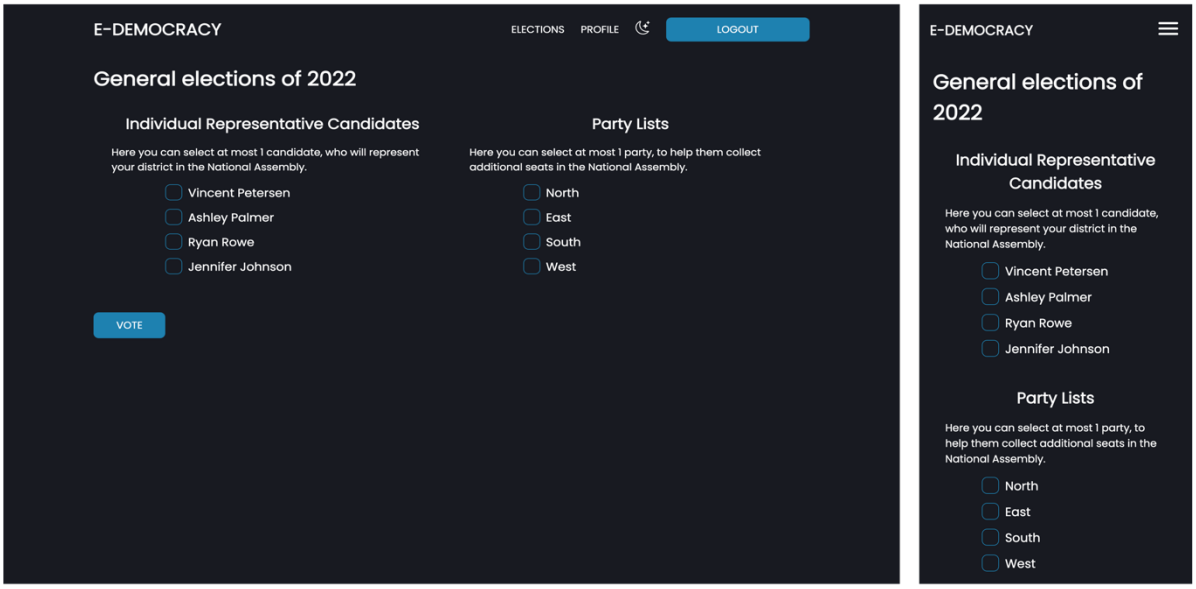
4.2.5 /elections/[electionid]

Az */election/[electionid]* oldal a */account* oldalhoz hasonlóan egy Modal komponens segítségével bekéri a felhasználó jelszavát, megadása nélkül nem lehet továbblépni a szavazás folyamatában (*21.ábra*). A jelszó megadása után a felhasználói jelszót az RSA privát kulcs feloldására fogja felhasználni, amely a szavazás folyamatában kiemelkedő jelentőségű lesz.

Az RSA kulcspár beolvasása után felsorolja a választható egyéni képviselőjelölteket és a pártlistákat. (*22.ábra*) A jelölt, illetve a pártlista kiválasztása után a szavazás gomb segítségével elküldhetjük a szavazatunkat. A szavazat formátumát tekintve, egy JavaScript objektum lesz, mely tartalmazni fogja a szavazat objektumot (az egyéni képviselő id-jával, illetve a pártlista id-jával), a szavazat RSA aláírását, illetve a felhasználó publikus kulcsát. Ezután a kliens oldali programunk egy HTTP POST kérést küld, a */api/v1/elections/[electionid]/votes* végpontra, melynek törzse tartalmazni fogja a fentebb említett JavaScript objektumot.



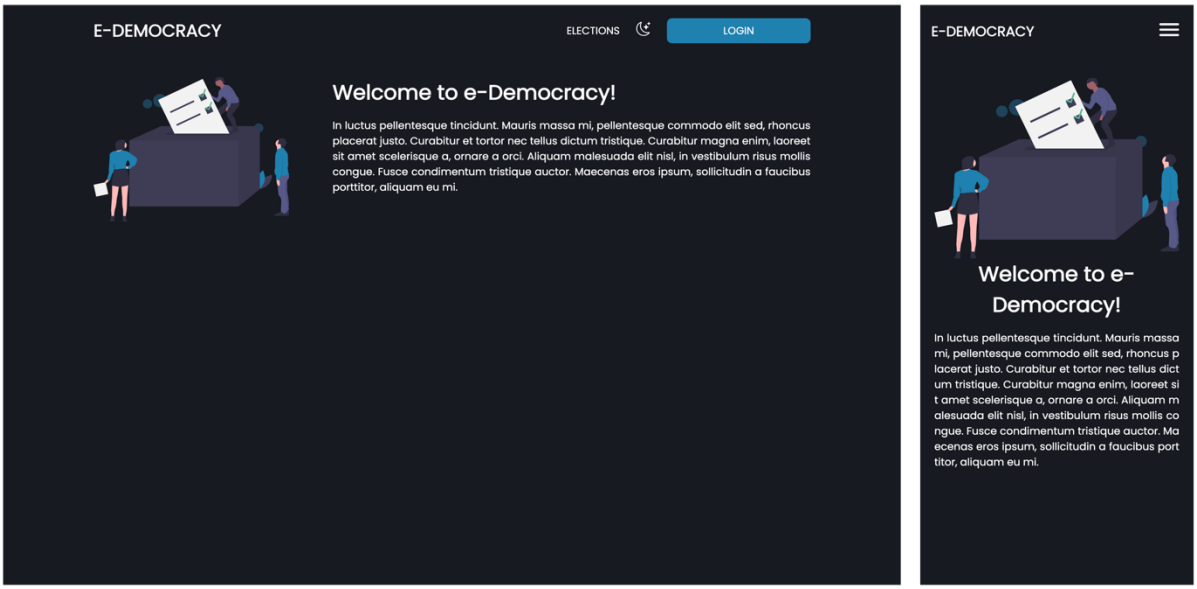
21. ábra: A lezárt /election/[electionid] oldal



22. ábra: A nyitott /election/[electionid] oldal

4.2.6 Az index oldal

Az index oldal tartalmilag csupán egy Lorem Ipsum szöveget tartalmaz, viszont ez lehet az az oldal, ahol elérhetővé tehetjük a választási eredményeket, mivel ez az az elérési útvonal, amely nincs aktív session-hoz kötve.



23. ábra: Az index oldal

4.3 React specifikus elemek bemutatása

A soron következő szakasz a webalkalmazásom néhány fontosabb React specifikus elemének bemutatására szolgál.

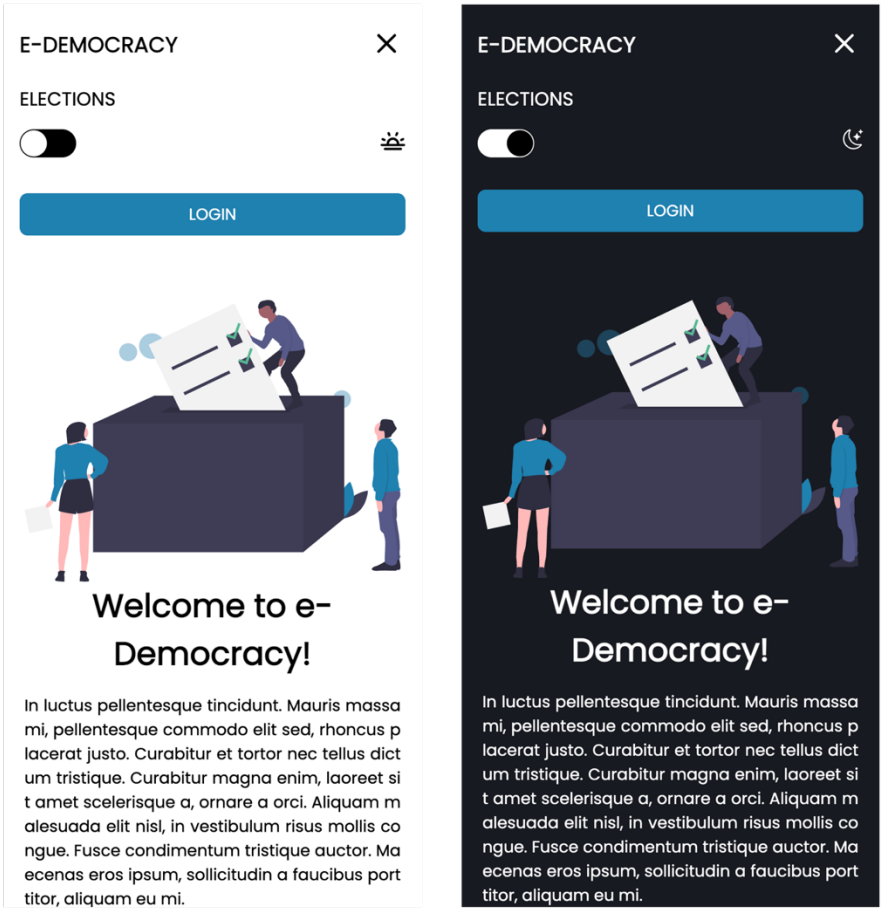
4.3.1 Context API

A context API a React keretrendszer egyik beépített funkciója, a state-hez hasonlóan egy értéket tárol el, azzal kiegészítve, hogy ez az érték a useContext függvény használatával a projekt egészében használhatóvá válik. Az alkalmazás fejlesztése során három context-et használok, melyeket aztán useHook-ok használatával hívok be az alkalmazásba.

4.3.1.1 ModeContext

A ModeContext a felhasználói felület használati módját tartalmazza, mely felvehet ’dark’ és ’light’ értékeket. Ezen értékek változtatásával módosíthatjuk a felhasználói felület megjelenését, asztali gép esetén a hold, illetve nap ikonra való kattintással, mobil nézetben pedig a Switch komponens csúsztatásával.

A mód változtatásának esetén a react-cookies csomag használatával egy klines oldali Cookie-t állítunk be mely tartalmazni fogja a felhasználó által preferált módot, és a következő oldalletöltéskor már a preferált módon fog megjelenni a felhasználói felület. (*24.ábra*)



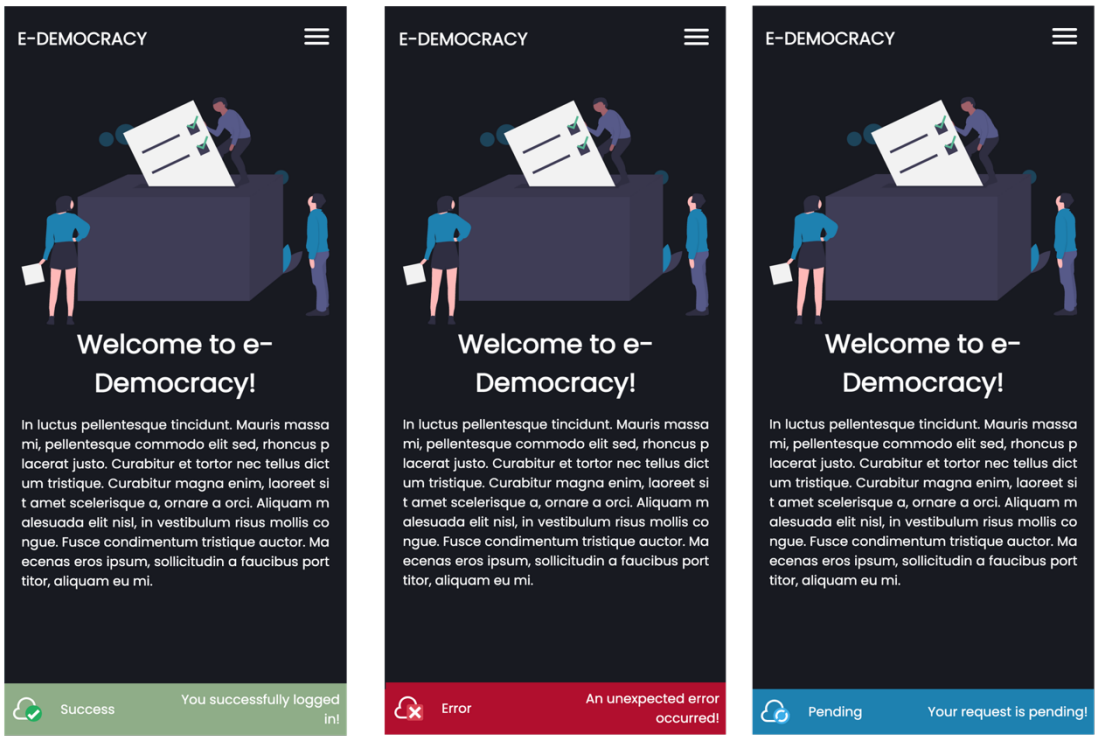
24. ábra: A megjelenési módok közötti váltás a ModeContext segítségével (mobilnézet)

4.3.1.2 SessionContext

A SessionContext ahogy a neve is árulkodó lehet a felhasználó aktuális session-jének eltárolására szolgál. A felhasználói adatokat titkosítottan tároljuk el a session-ben a session-tokennel egyetemben, amely a token lejáratát, igénylésének idejét és a felhasználó irányítószámát tartalmazza. A ModeContext-hez hasonlóan lekéri a sessionnel kapcsolatos információkat a Cookie-k közül, ha nincs ilyen információ akkor a session null értéket fog tartalmazni. Ezáltal a session context tökéletes annak meghatározására, hogy a React rendeljen-e vagy sem bizonyos komponenseket. Ezt az adottságot használom ki a /elections/[electionid] oldalnál, ahol, ha nincs egy aktív session, akkor egy bejelentkezésre felszólító üzentet fogunk kiírni az oldalon. A szerver oldali renderelésnek köszönhetően pedig az oldal klines oldali betöltése előtt lehetőségünk van megállapítani azt, hogy az adott felhasználónak be kell-e tölteni az oldalt, ezt a megoldást alkalmazom a */account* oldal esetében is.

4.3.1.3 NotificationContext

A NotificationContext segítségével lehetőségünk van a felhasználó számára értesítéseket küldhetünk az aktuálisan folyamatban lévő HTTP kérésekről, vagy azok eredményéről. Ehhez hoztam létre egy Notification komponenst (*25.ábra*), melyben meg tudjuk adni az értesítést típusát, szövegét és címét. Három értesítés típust különböztetünk meg [’pending’, ’success’, ’error’].



25. ábra: A success, az error és a pending típusú notification-ok (mobilnézet)

4.3.2 Komponensek

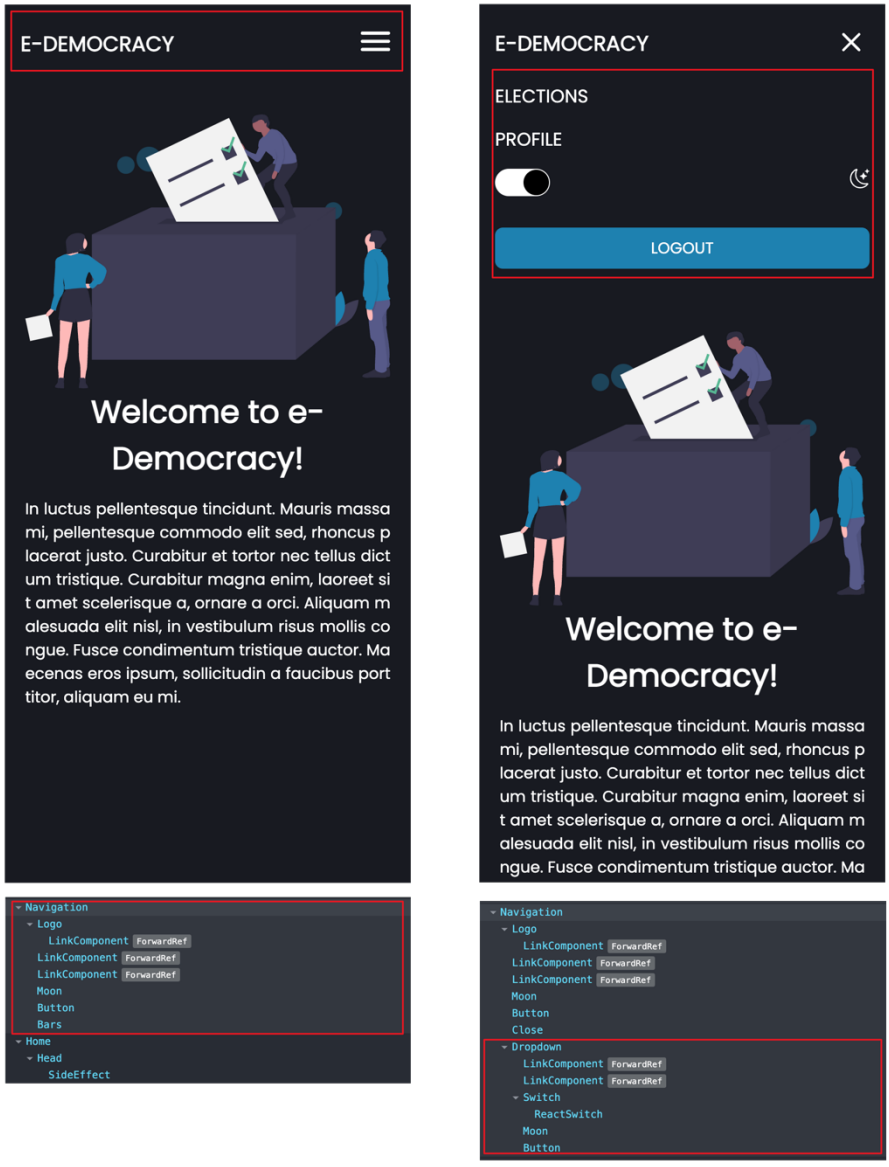
A React keretrendszer legfontosabb jellemzője az, hogy komponensek segítségével építhetjük fel az általunk készített webalkalmazást, ezzel redukálva a használt kódmennyiséget, és elszeparálva tudjuk tárolni az egyes komponensek forráskódját. A CSS modulok segítségével lehetőségünk nyílik arra is, hogy minden egyes komponens külön CSS kódot használjon, így egy letisztult kódbázist elérve.

4.3.2.1 Navigation

A Navigation komponens lefőbb célja az oldalak közötti navigáció elősegítése. Mivel a webes, illetve mobil nézet között nem kis eltérés van, ezért media query-k alkalmazásával oldottam meg azt, hogy a mobil nézet megjelenítésekor elrejtse a webes navigáció linkjeit és

helyettük a szendvics menüt szimbolizáló ikont mutassa. Bizonyos esetekben viszont a media query-k alkalmazása nem volt megoldható, ilyen például a mobilos legördülő menü megjelenítése. Ehhez a React keretrendszerbe beépített state-eket alkalmaztam. State-et létrehozni a useState hook alkalmazásával lehetséges, mely az adott state lekérdezésére alkalmas változóval, illetve a state módosítására szolgáló függvénnyel tér vissza.

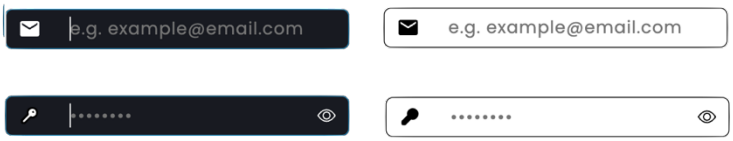
Az így létrehozott state értékének lekérdezésével már lehetőségünk van feltételes renderelésére, így, ha a legördülő menü aktív, akkor rendeljük azt, ellenkező esetben viszont nem. (*26.ábra*)



26. ábra: A Navigation komponens a Dropdown komponenssel (mobilnézet)

4.3.2.1 Input

Az Input komponens a beépített input komponens kiterjesztése, a kiterjesztett komponens által lehetőségem lett egy olyan személyre szabott beviteli mező létrehozására, amely mellé ikonokat társíthatok. Továbbá abban az esetben, ha a beviteli mező típusa ’password’ akkor renderel mellé egy Visibility komponenst, amivel lehetőségünk van a bevitt szöveg elrejtésére, és megjelenítésére. (*27.ábra*)



27. ábra: Az Input komponens ’email’ és ’password’ típusa (mobilnézet)

4.4 Szavazatok összesítése, érvényesítése

A szavazatok összesítésének bemutatását az elterjedt Python magas szintű programozási nyelven keresztül fogom bemutatni kódrészletek segítségével. Ahogyan azt említettem már a 2. fejezetben a szavazatok tárolását blockchain-hez hasonlóan összeláncolt formában valósítottam meg, melynek célja az, hogy ne lehessen szavazatot törölni, anélkül, hogy ez ki ne derülne a blockchain validációkor. Ezt a validációt akár a szavazás alatt is többször le lehet futtatni, hogy megbizonyosodjunk arról, hogy a szavazás folyamata alatt nem történt csalás. Ennek érdekében alkalmassá tettem az /api/v1/elections/[electionid]/votes végpontot arra, hogy képes legyen HTTP GET kérések fogadására, melyek eredményeként visszaküldi az összes addig a pontig megérkezett szavazatot. Habár ezt a végpontot érdemes levédeni, annak érdekében, hogy ne lehessen túlterhelni a végpontot, akár a lekérdezések számának megkötésével, vagy a jogosultságok korlátozásával, utóbbi viszont csökkentheti a rendszerbe vetett bizalmat.

A blockchain validálásának elsődleges lépése a szavazatok lekérdezése, erre egy egyszerű Python függvényt írtam, mely egy HTTP GET kérést küld a fentebb említett végpontra, majd egy szavazat objektumokból álló listát küld vissza a kliens alkalmazásunk felé. (*28.ábra*)

```python
from typing import Dict, Any, List
from requests import get, Response


BASE_URL: str = 'http://localhost:3000/api/v1'


def read_votes(election_id: str) -> Tuple[bool,
List[Dict[str, any]]]:
    url: str = f'{BASE_URL}/elections/{election_id}/votes'
    response: Response = get(url)
    if not response.ok:
        return False, []
    return True, response.json()
```

28. ábra: A szavazatok lekérése HTTP GET kéréssel Python-ban

A szavazatok lekérése után lehetőségünk van felhasználásukkal a blokklánc validációjára, de ehhez szükségünk lesz minden egyes soron következő blokk md5 hash értékére, a soron következő függvény bekér egy szavazat objektumot, melynek a md5 hash értékével tér vissza. (*29.ábra*)

```python
from typing import Dict, Any
from hashlib import md5
from json import dumps


def hash_vote(vote: Dict[str, Any]) -> str:
    return md5(bytes(dumps(vote, separators=(',', ':')),
'utf-8')).hexdigest()
```

29. ábra: Egy szavazat objektum md5 hash értékének kiszámítása

A hash_vote függvény segítségével már megírhatjuk azt a függvényünket, amely a blokklánc validációért lesz felelős A blokklánc validációhoz csupán egy for ciklus használatával végig haladunk a szavazatok listáján, majd minden egyes kör végén elmentjük a soron következő blokk md5 hash értékét. Az alábbi Python függvény ezek alapján fog visszaadni egy bool típusú értéket. A visszatérési érték True lesz, ha a blokklánc érvényes, tehát nem lett módosítva, viszont False értéket fog visszaadni abban az esetben, ha bármilyen adat módosításra került. (*30. ábra*)

```python
from typing import List, Dict, Any


def validate_chain(votes: List[Dict[str, Any]]) -> bool:
    previous_hash: str = hash_vote(votes[0])
    for vote in votes[1:]:
        if vote['previous'] != previous_hash:
            return False
        previous_hash: str = hash_vote(vote)
    return True
```

30. ábra: A blokklánc validációért felelős Python kódrészlet

Amennyiben a blokklánc valid, tovább lehet lépni a szavazatok összesítésére. Mivel a blokklánc valid, tehát nem lett módosítva, így nem feltétlen fontos a szavazatok signature-jének ellenőrzése. A szavazatok összesítését megkönnyítendő érdemes azokat elkülöníteni, és az eredmény megállapításáig csupán minimális adatmennyiséggel dolgozni, mint például a jelöltek, illetve a pártlisták id-jával. (*31.ábra*)

```python
def collect_votes_by_district(districts: List[Dict[str, Any]],
votes: List[Dict[str, Any]]) -> Dict[str, Any]:
    votes_by_district: Dict[str, Any] = {}
    [votes_by_district.__setitem__(district['_id'], []) for
district in districts]
    [votes_by_district[vote['vote']
['districtId']].append(vote['vote']['candidateId']) for vote in
votes]
    return votes_by_district
```

31. ábra: A szavazatok rendezésére szolgáló függvény

Az így kapott szótár segítségével már egyszerűen összesíthetjük a szavazatokat. Ahogyan azt manuálisan is tennénk, minden egyes érvényes szavazat esetén hozzászámoljuk a szavazatot az adott jelölt kapott szavazataihoz. A magyar választási rendszerben létezik mind vesztes és győztes kompenzáció, így esetünkben azokat a szavazatokat is fontos lenne számításban tartani, amelyek már vagy nem kellettek a győzelem elnyerésére, vagy nem segítették az egyéni képviselőjelölt győzelmét elő. Viszont esetemben a kód leegyszerűsítésének érdekében ezt most nem veszem figyelembe. (*32.ábra*)

```python
def calculate_votes_by_district_and_candidate(votes_by_district:
Dict[str, Any]) -> Dict[str, Any]:
    for district in votes_by_district:
        votes_by_candidate: Dict[str, int] = {}
        for v in votes_by_district[district]:
            if v not in votes_by_candidate:
                votes_by_candidate[v] = 0
            votes_by_candidate[v] += 1
        votes_by_district[district] = votes_by_candidate
    return votes_by_district
```

32.ábra: A jelöltekre érkezett szavazatok megszámlálásáért felelős függvény

A függvény által kapott szótár egy olyan district_id szótár párokat tartalmazó változó lesz, mely a 33.ábrán látható módon fogja eltárolni az egyes szavazó körökbe tartozó jelöltek szavazatainak számát.

```
{
  "p8CXzkRyRvTlS1bS": {
    "2qEp-68WfE2gz9cG": 4,
    "1lD42q3gOAhjLjEY": 2,
    "G3h6Pueq2CpDgjlF": 2,
    "Ud6VI91P9GJd28r_": 2
  },
  ...
}
```

33.ábra: A fenti függvény által visszaadott szótár

A pártlistás szavazatok összegzése ennél egyszerűbben zajlik, hiszen nincs szükségünk arra, hogy választókerületekre bontsuk a szavazatokat, egyszerűen egy for ciklus és alapvető matematikai műveletek segítségével összegezhetjük pártlistákra leadott szavazatokat. (*34.ábra*)

```python
def calculate_votes_by_party_list(votes: List[Dict[str, Any]]) ->
Dict[str, Any]:
    votes_by_party_list: Dict[str, Any] = {}
    for vote in votes:
        if vote['vote']['partyListId'] not in
votes_by_party_list:
            votes_by_party_list[vote['vote']['partyListId']] = 0
        votes_by_party_list[vote['vote']['partyListId']] += 1
    return votes_by_party_list
```

34.ábra:A pártlistás szavazatok összegzésére szolgáló függvény

A függvény által visszaadott szótár kulcsai a pártlisták azonosítói, míg értékei az adott pártlistára érkezett szavazatok számát fogják tartalmazni.

Az így kapott eredmények alapján megkapjuk, hogy melyik párt hány százalékban kell részesüljön a pártlistás parlamenti helyekből, illetve mely egyéni választókerületben, melyik jelölt került ki győztesként. Az eredmények megállapítását egy éles helyzetben zajló választás alkalmával nyilvánvalóan tovább bonyolítaná a magyar választási rendszerben létező győztes, illetve veszteskompenzáció.

Magyarországon a parlamenti bejutási küszöb jelenleg 5%, mely az a határ, amely alatt nem kap listás mandátumot az adott parlamenti párt, így a pártlistás parlamenti székek megoszlását ez is módosítja.

A veszteskompenzáció töredék szavazatainak kiszámítása egyszerű, hiszen ez azon szavazatok egésze, amelyeket az adott jelölt megszerzett viszont nem volt elegendő a parlamenti mandátum megszerzéséhez.

$$x_2^{`} = x_2$$

A győzteskompenzáció töredék szavazatainak kiszámítása az előbbivel szemben annyiban módosul, hogy a győztes jelölt szavazatszámából kivonjuk a második helyen álló jelölt szavazatszáma plusz egy értéket, így megkapva a győztes jelölt pártjának járó töredék szavazatokat. A lentebb látható képlet adja meg a győztes jelölt töredék szavazatainak kiszámításának módját, ahol x1 a győztes jelölt által megszerzett szavazatok száma, x2 a második helyen végzett vesztes jelölt szavazatainak száma, x1` pedig a győztes jelölt töredék szavazatainak száma lesz.

$$x_1^{`} = x_1 - (x_2 + 1)$$

Tehát amennyiben a győztes megállapítását az aktuális magyar jogszabályok szerint szeretnénk megtenni, figyelembe kellene helyeznünk a fentebb megemlített szabályokat, és esetleges körülményeket. [11]

5. Összegzés

A szakdolgozat keretein belül az ismertebb titkosítási szabványok alkalmazásával közelítettem meg az online szavazás titkosságának megvalósítását.

A szavazás titkosságának kérdését a második fejezetben taglalva bemutattam négy megközelítését arról, hogy hogyan lehet elérni azt, hogy a választói anonimitás megvalósuljon. Ebben a fejezetben bemutattam azt, hogy hogyan jutottam el addig a metódusig, amely ténylegesen is alkalmazásra került az általam készített webalkalmazásban.
A harmadik fejezet csupán az alkalmazott technológiáknak, illetve node csomagoknak lett szentelve, azt taglalom ebben a fejezetben, illetve alfejezeteiben, hogy mely csomagok alkalmazása, milyen célok elérése érdekében volt indokolt.

A negyedik fejezet a webes felhasználói felület, az adatbázis felépítésének bemutatását szolgálta, továbbá néhány React specifikusabb elem ismertetését, mint a useState hook, illetve a Context API ismertetését. A fejezeten belül képeken keresztül mutattam be a felhasználói felületet és az egyes oldalakhoz tartozó komponenseket. A 4.4-es alfejezetben pedig egy módszert mutattam be a szavazatok megszámolására Python kódrészletek segítségével.

A projekt keretein belül a szavazófelület kialakítása töltötte ki a legtöbb időt, illetve az alkalmazott adatstruktúrák megalkotása. Szerintem a jelenleg meglévő rendszer nem lenne képes egy nagyobb szavazás megvalósítására, hiszen minden egyes lekérdezésnél szükséges a teljes választás objektum lekérdezése, amely lassítja a lekérdezéseket.

A projekt tovább gondolásakor mindenképp az adatok eltárolásának módja az, amit módosítanék, illetve a tervezett viszont meg nem valósult admin felület kialakítását venném fontolóra, illetve a nagyobb felhasználó bázis kiépítésének érdekében célratörő lenne egy React Native alkalmazás fejlesztése is, mely lehetővé tenné az online szavazást mobil eszközön is a böngésző használatának mellőzése mellett.

Továbbá jelenleg a beérkezett szavazatok csupán az HTTPS protokoll védelme alatt állnak, ezt azzal bővíteném, hogy az adatokat a szerverre való elküldés előtt a már a JWT token aláírására használt RSA kulcspár publikus kulcsával titkosítanám, majd a szerverre való beérkezés után visszafejteném a privát kulccsal, ezzel növelve a szavazás titkosságának védelmét.

Az elért eredményeket összességében pozitívan értékelem, hiszen a projekt keretein belül mélyebben beleáshattam magam az titkosítási szabványok működésébe, illetve a webfejlesztés

egyik legmodernebb keretrendszerének alkalmazásával valósíthattam meg a projektemet, mellyel szélesítettem azt általam ismert JavaScript keretrendszerek körét.

6. Irodalomjegyzék

[1] 2011. évi CCIII. törvény az országgyűlési képviselők választásáról, 2011. december 30.

[2] Townsend Security: AES vs DES Encryption: Why Advanced Encryption Standard (AES) has replaced DES, 3DES and TDEA, 2021. május 03. https://www.precisely.com/blog/data-security/aes-vs-des-encryption-standard-3des-tdea

[3] Advanced Encryption Standard (AES), 2022. február 11. https://www.geeksforgeeks.org/advanced-encryption-standard-aes/

[4] AES steps (SubBytes, ShiftRows, MixColumns), https://www.youtube.com/watch?v=Tx_37dF03ig&t=1s, 2020. május 25.

[5] What is the RSA algorithm? https://www.educative.io/answers/what-is-the-rsa-algorithm

[6] Node.js dokumentáció, https://nodejs.org/en/docs/

[7] Kirill Konshin: Next.js Quick Start Guide, Packt Publishing, 2018. július 26.

[8] Robin Wieruch: The Road to React: Your journey to master plain yet pragmatic React.js, 2018. szeptember 14.

[9] JSON Web Token (JWT), https://www.rfc-editor.org/rfc/rfc7519, 2015 május

[10] Bradshaw Shannon: Mongodb: The Definitive Guide: Powerful and Scalable Data Storage, Oreilly Media, 2019

[11] A győzteskompenzáció, https://jogaszvilag.hu/szakma/a-gyozteskompenzacio/, 2022. május 9.