Razvoj programske podrške za izdavanje digitalnih studentskih bedževa na javnom blockchainu

Krapić, Karlo

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Diplomski sveučilišni studij računarstva

Diplomski rad

Razvoj programske podrške za izdavanje digitalnih studentskih bedževa na javnom blockchainu

Rijeka, studeni 2023.

Karlo Krapić
0069082989

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Diplomski sveučilišni studij računarstva

Diplomski rad

Razvoj programske podrške za izdavanje digitalnih studentskih bedževa na javnom blockchainu

Mentor: prof.dr.sc. Kristijan Lenac

Rijeka, studeni 2023.

Karlo Krapić
0069082989

Umjesto ove stranice umetnuti zadatak za završni ili diplomski rad

Izjava o samostalnoj izradi rada

Sukladno članku 8. pravilnika o diplomskom radu, diplomskom ispitu i završetku diplomskih sveučilišnih studija Tehničkog fakulteta Sveučilišta u Rijeci od 31. siječnja 2020., izjavljujem da sam samostalno izradio/izradila diplomski rad prema zadatku preuzetom dana 21. ožujka 2022.

Rijeka, studeni 2023.

_____

Karlo Krapić

Zahvala

Zahvaljujem se prof. dr. sc. Kristijanu Lencu na podršci i strpljenju tijekom pisanja ovoga rada i pravovremenim savjetima. Također, zahvaljujem se svojoj obitelji na podršci tijekom studiranja.

Sadržaj

Popis slika

Popis tablica

Popis isječaka koda

Poglavlje 1

Uvod

Na Tehničkom fakultetu u Rijeci, u sklopu RiTeh Blockchain Teama pokrenut je 2021. godine projekt "Digitalni studentski bedževi" s idejom da se implementira sučelja za kreiranje i upravljanje digitalnim studentskim bedževima baziranim na nezamjenjivim tokenima.

Ideja iza projekta studentskih bedževa je korištenje karakteristika javnih lanaca blokova (eng. *blockchain*) te nezamjenjivih tokena (eng. *Non-fungible token* - NFT) kako bi se studentima omogućilo jednostavno nadgledanje informacija vezanih uz različite aspekte njihovog obrazovanja (kojim školskim klubovima pripadaju, kojim konferencijama su prisustvovali itd.). Korištenjem novčanika na određenom blockchainu studenti bi imali jednostavan pregled i povijesni zapis vlastitog sudjelovanja u različitim elementima školovanja.

Izrada nezamjenjivih tokena je implementirana na sljedećim javnim blockchain mrežama: Algorand, Near, Ethereum, Cardano i Cosmos. Navedene mreže su odabrane primarno jer podržavaju nezamjenjive tokene te jer imaju dostupna sučelja za rad s njima. Neki od dodatnih parametara koji su sagledani pri odabiru mreža su: cijene transakcija za izradu i transfer bedža, dostupna dokumentacija, popularnost blockchaina i dr. Konačni odabir je donesen na temelju rasprava održanih na sastancima RiTeh Blockchain Teama.

Cilj ovog rada bio je razviti mrežno administratorsko programsko sučelje za interakciju s više različitih javnih blockchaina u svrhu izdavanja digitalnih studentskih

bedževa. Studentski bedževi su predstavljeni nezamjenjivim tokenima koji se temelje na blockchain tehnologiji te se koriste kao dokaz vlasništva.

Programska podrška je osmišljena kao jednostavna mrežna stranica za administratore studentskih bedževa. Sučelje je dizajnirano da sadrži osnovne funkcionalnosti mrežne stranice u koje spadaju autentifikacija, pohranjivanje podataka te upravljanje dostupnim blockchain mrežama. Glavne funkcionalnosti vezane uz studentske bedževe su implementirane za više različitih mreža te uključuju: pregled svih izdanih bedževa, kreiranje novog studentskog bedža te slanje bedža na određenu adresu.

U prvom poglavlju je detaljnije opisana teorijska podloga rada u koju spadaju termini: blockchain, pametni ugovor te nezamjenjivi tokeni.

U drugom poglavlju opisuju se pojedinačne blockchain mreže te njihove najbitnije karakteristike. Dodatno, sadrži prikaz najbitnijih dijelova implementiranog programskog sučelja, poslužitelja i baze podataka. Posljednje potpoglavlje sadrži opis implementacije osnovnih funkcionalnosti za rad s bedževima u spomenutim blockchain mrežama.

Posljednje poglavlje sadrži zaključak, probleme koji su identificirani te prijedloge za daljnji razvoj.

Poglavlje 2

Opis problema i postojećih rješenja

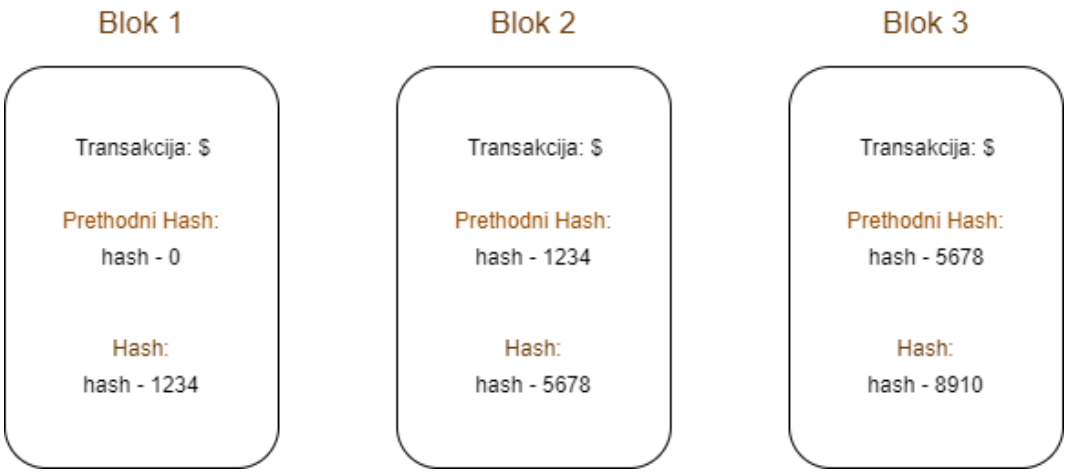
2.1 Blockchain

Blockchain je distribuirana baza podataka koja se sastoji od rastućeg broja blokova (eng. *blocks*) koji su međusobno povezani s pomoću kriptografije [6]. Blokovi u sebi sadrže informacije o transakcijama koje se odvijaju na blockchainu, vrijeme kreiranja bloka te *hash* vrijednosti prijašnjeg bloka u nizu [7]. *Hash* funkcije su matematičke funkcije koje proizvoljan broj znakova transformiraju u kompresirani niz fiksne duljine [8]. Dodatno, bitna svojstvo kvalitetnih hash funkcija je da su otporne na kolizije (eng. *collision-resistant*). To znači da su izrazito male šanse da će se dva različita unosa preslikati u isti niz te da će mala promjena ulaznog niza rezultirati velikom promjenom izlaza [9]. Na slici 2.1 prikazano je kako se *hash* funkcije koriste za povezivanje blokova. Korištenjem ovih funkcija je osigurano da je lako detektirati bilo kakve izmjene na blockchainu te time mreža postaje sigurnija.

Arhitektura blockchaina se može podijeliti na 5 konceptualna sloja: hardverski, podatkovni, mrežni, konsenzus te aplikacijski [10].

Hardverski sloj se sastoji od mreže računala koji koriste vlastite resurse za funkcioniranje mreže. Svako računalo u mreži naziva se čvor (eng. *node*) te doprinosi radu mreže [10].

Podatkovni sloj se primarno bavi pohranom podataka i struktura koje čine blockchain mrežu. Na njemu su pohranjeni svi blokovi mreže te s njim povijest svih



Slika 2.1 Prikaz povezivanja blokova u blockchainu

transakcija. Kompleksnost navedenog sloja ovisi o pojedinačnom blockchainu. Dodatno, u ovome sloju su definirani osnovni kriptografski standardi koje se koriste u protokolu poput asimetrične enkripcije korištene u generiranju javnih i privatnih ključeva. Time ovaj sloj doprinosi sigurnosti mreže [10].

Mrežni sloj je odgovoran za komunikaciju između čvorova na mreži [10]. S obzirom na to da je blockchain otvorena *peer-2-peer* mreža svaki čvor mora imati informaciju koje transakcije validiraju ostali čvorovi. Navedeni sloj omogućuje otkrivanje i povezivanje s drugim čvorovima te time osigurava komunikaciju. Sama implementacija komunikacijskog protokola ovisi o specifičnom blockchainu [11].

Konsenzus sloj postavlja pravila za sudjelovanje u radu blockchaina. Navedeni sloj definira mehanizam za postizanje konsenzusa te time osigurava validiranje ispravnih blokova. Dodatno, sudjeluje u validiranju te u zaštiti mreže od određenih vrsta napada [11].

Aplikacijski sloj služi za kreiranje raznovrsnih aplikacija te pametnih ugovora. Navedene aplikacije se koriste kao i bilo koja druga samo u pozadini komuniciraju s blockchainom te ga koriste za pohranu informacija [10]. Time ovaj sloj predstavlja sučelje za interakciju prosječnog korisnika i blockchaina.

Za kreiranje novih blokova u mreži se koriste različiti mehanizmi za postizanje

konsenzusa. Najpopularniji su: dokaz o radu (eng. *Proof of work* - PoW) i dokaz o udjelu (eng. *Proof of Stake* - PoS) [12] (Tablica 2.1). PoW mehanizam koristi kompetitivni sustav validacije gdje se sudionici natječu tko će prvi pogoditi niz znakova s određenim uzorkom koristeći *hash* funkcije. Osoba koja prva dođe do rješenja dobiva nagradu te se tako potiče sudionike da troše računalne resurse na potvrđivanje transakcija [12]. Kod PoS mehanizma koriste se nasumično odabrani korisnici kako bi se validirale transakcije i kreirali novi blokovi na mreži [12]. Postoje i drugi mehanizmi te se pri njihovom odabiru promatra veliki broj čimbenika poput cijene, utjecaja na okoliš, potrebe same mreže i drugih [13].

PoW	PoS
Kapacitet rudarenja ovisi o računalnim resursima	Kapacitet validiranja ovisi o količini uloga na mreži
Rudari primaju nagrade za rješavanje kriptografskih zadataka	Validatori primaju proviziju za svaku transakciju
Napadači trebaju imati računalo snažnije od 51% mreže kako bi napali mrežu	Napadači trebaju imati 51% kriptovalute na mreži kako bi izvršili napad

Tablica 2.1 Usporedba PoW i PoS konsenzus mehanizma

Iz arhitekture blockchaina proizlaze i njegove karakteristike od kojih su najbitnije: neizmjenjivost, distribuiranost, decentraliziranost, sigurnost i transparentnost [14].

Neizmjenjivost znači da je mreža stalna i jednom kada se nešto zapiše na nju ne može se izmijeniti ili izbrisati. Dodavanjem novog bloka, transakcije koje se nalaze u njemu postaju trajno zapisane u blockchainu [14].

Distribuiranost je vidljiva u činjenici da svi sudionici u mreži imaju kopiju zapisnika (eng. *ledger*) koji sadrži informacije o svim sudionicima na mreži te svim transakcijama. Ovime je osigurano da svi čvorovi u mreži su jednako važni te da za dodavanje novog bloka u mrežu, mora postojati konsenzus između čvorova [14].

Iz jednakosti svih čvorova proizlazi decentraliziranost sustava gdje ne postoji ni jedan centralni autoritet koji kontrolira mrežu. Svatko tko preuzme čvor na računalo postaje ravnopravni član mreže. Dodatno, decentralizirane mreže su teže za napasti

te ih je teže srušiti nego centralizirane jer svaki čvor ima kopiju cijelog zapisnika [14].

Iz distribuiranosti proizlazi transparentnost blockchaina. Sve transakcije su vidljive na osobnom čvoru korisnika ili na jednom od brojnih blockchain preglednika [15].

Navedene karakteristike su dovele do naglog razvoja blockchain tehnologije od njihovog nastanka 2008. godine kada je Satoshi Nakamoto objavio rad gdje opisuje tehnološku pozadinu prve kriptovalute, Bitcoina [16]. Najrazvijeniji aspekt tehnologije su kriptovalute čiji broj kontinuirano raste te su postale ogromno tržište. Dodatno, blockchain tehnologija se koristi u različite svrhe te broj novih primjena i inovacija kontinuirano raste [17].

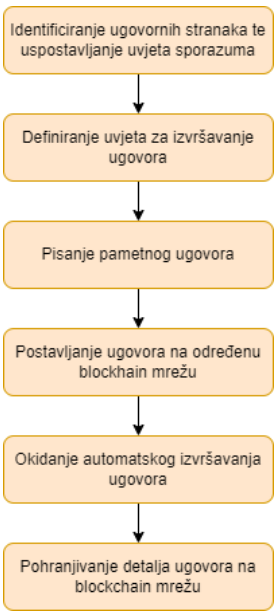
2.2 Pametni ugovori

Pametni ugovori (eng. *Smart Contracts*) su programi kodirani i pohranjeni na blockchainu koji se pokreću kada su određeni uvjeti ispunjeni [18]. Navedena tehnologija predstavlja podlogu za rad s nezamjenjivim tokenima te studentskim bedževima koji su svrha ovog projekta. Najčešće se koriste za automatiziranje sporazuma između 2 ili više sudionika [18]. Velika prednost pametnih ugovora je da ne zahtijevaju vanjske posrednike kako bi se izvršili [19]. Nakon izvršetka ugovora, sve stranke mogu odmah potvrditi njegovu validnost. Primjerice, za plaćanje preko određene kriptovalute, nije potreban posrednik poput banke već se s pomoću pametnog ugovora prijenos automatski izvršava. Ovime proces postaje brži, jeftiniji i jednostavniji [18]. Popularizirao ih je Ethereum blockchain koji koristi poseban programski jezik Solidity koji je specifično dizajniran u svrhu kreiranja pametnih ugovora [20].

Proces kreiranja pametnog ugovora se može podijeliti na korake opisane na slici 2.2.

Prvi korak je identificiranje ugovorne strane te uspostaviti uvjete sporazuma koji odgovaraju svim sudionicima. Definiraju se obaveze svih strana u postupku te se definiraju svi standardi u ugovoru [19].

Idući korak obuhvaća specificiranje uvjeta koji se moraju ispuniti da se pokrene izvršavanje ugovora. Navedeni uvjeti se izraze kao skup pravila koji se moraju zado-



Slika 2.2 Proces kreiranja pametnog ugovora

voljiti [19].

U trećem koraku se piše sam kod za pametni ugovor. Navedeni kod sadrži točno definirane uvijete koji se temelje na prethodnom koraku te opisuju kada će se ugovor izvršiti [19].

Nakon kreiranja programske implementacije ugovora potrebno ga je postaviti na blockchain mrežu. Nakon validiranja koda i postavljanja ugovora pametni ugovor je spreman i u trenutku kada se ispune unaprijed definirani uvjeti se automatski izvršava. Nakon završetka izvršavanja, detalji o ugovoru se pohranjuju na blockchain mreži [19].

Industrija pametnih ugovora pokazuje kontinuirani rast od pojave Ethereuma kao prve platforme za njihovo kreiranje [21]. Primjene pametnih ugovora su brojne i stalno se razvijaju. Prošireni su kroz različite dijelove ljudskog društva: pohranjivanje podataka, lanci opskrbe, hipoteke, trgovanje nekretninama, zdravstvo te brojni drugi [22].

2.3 NFT

Nezamjenjivi tokeni su jedinstvena kriptografska digitalna sredstva koja su pohranjena na određenom blockchainu [23]. Glavna im je karakteristika da su jedinstveni i nezamjenjivi te se zbog toga ne mogu kopirati [23]. Primjer nedigitalnih nezamjenjivih sredstava su umjetnine. Jedinstvene su, nezamjenjive te se ne mogu kopirati. S druge strane, zamjenjiva sredstva su kriptovalute i fizički novac. U tablici 2.2 prikazani su primjeri zamjenjivih i nezamjenjivih tokena [24].

Zamjenjivi	Nezamjenjivi
Euro	*Cryptokitties* [25]
Bitcoin	Umjetnine
Ethereum	Kuća/Vlasništvo

Tablica 2.2 Usporedba zamjenjivih i nezamjenjivih sredstava

Tehnologija nezamjenjivih tokena se bazira na korištenju pametnih ugovora [26]. Pametni ugovori se koriste u procesu kreiranja i prijenosa vlasništva nezamjenjivog tokena [26]. U trenutku kreiranja NFT-a, vlasnik postaje tvorac pametnog ugovora [26]. U slučaju prodaje tokena pametni ugovor prenosi vlasništvo na kupca [26]. Dodatno, pametni ugovori se mogu proširiti da imaju i druge opcije poput deaktiviranja tokena i izgaranje tokena čime se simulira brisanje [26].

Svaki NFT sadrži jedinstveni digitalni potpis [27]. Svojstva i karakteristike blockchaina definiraju karakteristike i ulogu nezamjenjivih tokena. Najčešće se koriste kao potvrda vlasništva za digitalna i fizička sredstva [27]. Prodaju se i kupuju na različitim tržištima te imaju različite primjene od kojih su neke prikazane u tablici 2.3 [28].

Dodatno, određene blockchain mreže podržavaju dijeljenje nezamjenjivih tokena na više manjih dijelova (eng. *Fractional NFT*) [29]. Navedena tehnologija omogućuje da više različitih ljudi dijele vlasništvo nad određenim nezamjenjivim tokenima.

Područje	Primjena
Digitalna umjetnost	Autentificiranje originalnosti i vlasništva digitalne umjestnosti
Videoigre	Tokeniziranje predmeta u igrici te njihova razmjena na tržištu
Društveni tokeni	Izdavanje tokena za članove zajednice kako bi se povećala angažiranost
Sport	Ponuda licenciranih digitalni zapisa popularnih sportskih trenutaka
Luksuzna dobra	Autentificiranje originalnosti i vlasništva luksuznih dobara. Smanjenje količine krivotvornina.

Tablica 2.3 Najpopularniji načini korištenja nezamjenjivih tokena

2.4 Digitalni studentski bedževi

Projekt "Digitalni studentski bedževi" Tehničkog fakulteta u Rijeci bazira se na korištenju nezamjenjivih tokena kako bi se kreirali digitalni studentski bedževi za praćenje različitih aspekata studentskog obrazovanja. Proučavajući postojeća rješenja te druge fakultete možemo vidjeti da korištenje nezamjenjivih tokena za potvrdu obrazovanja već postoji u nekim obrazovnim institucijama [30]. Unatoč tome, navedena tehnologija i dalje nije široko rasprostranjene u obrazovanju unatoč prednostima opisanim u Blockchain potpoglavlju.

Jedan od razloga za to je da veliki broj blockchain mreža zahtjeva visoko tehničko znanje kako bi se uspješno postavile i koristile. Također, najčešće koriste naredbeni redak kako bi se komuniciralo s mrežom što prosječnom korisniku predstavlja izazov. Sukladno tome, problem koji se rješava u radu je kreiranje jednostavnog sučelje koje će administratoru omogućiti pregled, stvaranje i izdavanje bedževa studentima. Dodatno, pokušava se riješiti problem integriranja navedenih funkcionalnosti na više javnih blockchain mreža u jedno korisničko sučelje.

Pri dizajnu sučelja javlja se problem odabira odgovarajuće implementacije i tehnologije. Dvije mogućnosti su kreiranje mrežne ili lokalne aplikacije te zahtijevaju potpuno različito rješenje. Lokalna aplikacija zahtjeva korištenje naredbenog retka te implementiranje lokalnih novčanika. Mrežna aplikacija koristi novčanike implemen-

tirane u pregledniku te direktno komunicira s blockchain mrežom. U slučaju mrežne aplikacije se javlja dodatno pitanje hoće li se implementirati kao korisničko sučelje i aplikacijsko programsko sučelje ili kao jedna aplikacija.

Implementiranjem sučelja koji podržava više različitih blockchain mreža javlja se problem usklađivanja njihovog rada. Svaki blockchain ima drugačiji način na koji implementira operacije nad nezamjenjivim tokenima. Dodatno, neki zahtijevaju korištenje naredbenog retka, a neki imaju implementirane knjižnice za rad s mrežom. Korisničko sučelje mora korisniku omogućiti rad s blockchain mrežama na isti način neovisno o odabranoj mreži.

Odabrana implementacija dodatno utječe na pohranu podataka vezanih uz bedževe gdje se podaci mogu pohranjivati lokalno ili korištenjem međuplanetarnog datotečnog sustava (eng. *Interplanetary file system* - IPFS).

Odabrano rješenje izmjenjuje logiku za autentifikaciju korisnika. Korištenjem blockchain novčanika, postoji potvrda identiteta korisnika. Problem se javlja u slučaju lokalnog pohranjivanja podataka. Mora postojati način za povezati korisnika s lokalnim podacima.

Dodatno, pri dizajniranju sučelja potrebno je imati na umu moguća buduća proširenja sustava. U slučaju dodavanja novih blockchain mreža potrebno je minimizirati potrebne promjene postojećem sustavu. Upravljanje ulogama i dopuštenjima korisnika predstavlja dodatni problem gdje bi buduća proširenja sustava mogla imati različite uloge za različite korisnike te pristup specifičnim blockchain mrežama.

Poglavlje 3

Razvoj programskog rješenja

Implementacija je osmišljena kao mrežno korisničko sučelje podijeljeno na dva dijela: klijentski i poslužiteljski. Mrežni pristup je odabran kako bi se izbjegla instalacija programa te pokretanje blockchain mreže na lokalnom računalu. Klijentski i poslužiteljski dio su razdvojeni kako bi se omogućilo lakše povezivanje podataka i funkcionalnosti na poslužitelju s drugim servisima u slučaju proširenja sustava. Dodatno, njihovom razdvajanjem je lakše implementirati promjene i nadogradnje na samo jednom dijelu korisničkog sučelja time smanjujući potencijalne greške.

Uloga klijentskog dijela se fokusira na prikaz rezultata te jednostavno obavljanje operacija za rad s bedževima. Koristi novčanike u poslužitelju za potvrđivanje i izvršavanje transakcija. Novčanici se razlikuju za različite blockchain mreže. Time su glavne funkcionalnosti za rad s bedževima implementirane na klijentskoj strani aplikacije. Dodatno, problem različitih implementacija za različite mreže je riješen korištenjem više razine apstrakcije kako bi sučelje radilo jednako neovisno o odabranoj blockchain mreži te kako bi standardizirali podatke koje sve mreže moraju sadržavati.

Odabran je pristup pohranjivanja slika i metapodataka na IPFS s pomoću *pinate* (IPFS sučelje). Navedeni pristup se pokazao ograničen pri velikom broju kreiranih bedževa. *Pinata* ima ograničenje na broj zahtjeva u kratkom periodu te se u tom slučaju podaci ne mogu prikazati. Sustav je proširen da se podaci pohranjuju i lokalno kako bi se osigurala dodatna kopija te kako bi se smanjio broj zahtjeva na

pinatu. Aplikacija može pristupiti lokalno pohranjenim podacima te podacima na međuplanetarnom datotečnom sustavu.

Pohranjivanjem podataka na poslužitelju javlja se problem povezivanja podataka s korisnikom. Kako bi se riješio navedeni problem implementirana je autentifikacija i prijavljivanje u korisničko sučelje. Dodatno, originalno je planiran sustav za upravljanje korisnicima, ali se u dogovoru s mentorom odbacio. Inicijalno je osmišljen kao sustav za kreiranje i brisanje korisnika te dodjeljivanje različitih uloga i dopuštenja. Time bi se u slučaju proširenja sustava na druge obrazovne institucije pružila veća kontrola administratoru. Kod za navedene funkcionalnosti je uključne u repozitoriju projekta u slučaju budućeg proširenja sustava.

Sustav za upravljanje dostupnim blockchain mrežama je dodatno izbačen iz zahtjeva projekta. Osmišljen je kao sustav za dodavanje novih mreža te brisanje, omogućavanje i onemogućavanje postojećih mreža. Postojeći kod je dostupan u repozitoriju u slučaju daljnjeg unaprjeđenja sustava.

Na temelju navedenih stavki uloga poslužiteljske strane je autentifikacija korisnika, pohranjivanje metapodataka i slika, komunikacija s IPFS-om. Dodatno, poslužitelj služi kao predmemorija (eng. *cache*) za dohvat svih kreiranih bedževa. Time se rješava problem čekanja na potvrđivanje i izvršavanje transakcija te se smanjuje broj HTTP (eng. *Hypertext Transfer Protocol*) zahtjeva. U slučaju problema s povezivanjem s blockchain mrežom ili s *pinatom*, moguć je pregled podataka jer se kopija nalazi na lokalnom poslužitelju.

3.1 Odabrane blockchain mreže

Za razvoj programskog sučelja odabrane su sljedeće blockchain mreže: Algorand, Ethereum, Near, Cardano i Cosmos. Navedene mreže su odabrane primarno jer podržavaju pametne ugovori i nezamjenjive tokene koji su centralna tema ovog rada. Dodatno, odabrani su na sastancima RiTeh Web Teama na temelju potreba cjelokupnog projekta.

3.1.1 Algorand

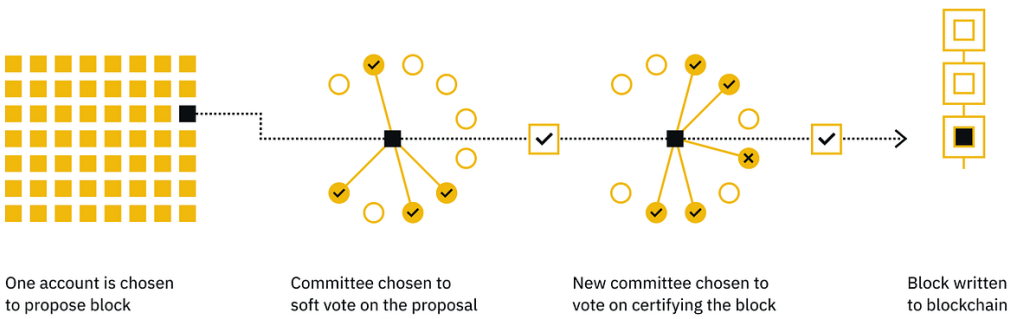
Algorand je blockchain mreža dizajnirana s fokusom na ekološku održivost. Osmišljena je s ciljem smanjenja emisija ugljikovog dioksida te zagađenja okoliša [31]. Nastala je kao odgovor na česti problem prekomjerne potrošnje energije i negativnog utjecaja na okoliš koje su česte kod drugih blockchain tehnologija [32]. Prema Pwc Crypto izvješću, količina energije potreba za napajanje Bitcoin blockchain mreže je ekvivalentna godišnjoj potrošnji omanje zemlje poput Švicarske [32]. Uzimajući u obzir broj postojećih blockchain mreža i količinu transakcija koje se odvijaju na njima vidljivo je da znatno doprinose količini stakleničkih plinova u atmosferi te globalnom zatopljenju. Općenito, PoW konsenzus za potvrđivanje transakcija zahtijeva najviše energije, dok neki drugi mehanizmi poput PoS zahtijevaju manje [32]. Unatoč tome, Algorand je razvio vlastiti unikatni konsenzus mehanizam koji dodatno smanjuje energetske potrebe za rad mreže

Pure Proof of Stake (PPoS) je konsenzus mehanizam baziran na problemu bizantskih generala. Navedeni problem je čest u komunikaciji između distribuiranih sustava. Najčešći problem je kako postići konsenzus pri donošenju odluka. PPoS je primjer protokola koji nudi rješenje na taj problem koristeći standardnu pretpostavku da su dvije trećine čvorova u sustavu iskreni sudionici te ju dodatno razvija korištenjem odabira vođe glasanja [33].

Na slici 3.1 je prikazan način na koji funkcionira PPoS mehanizam.

Algoritam se izvršava u rundama i u svakoj rundi mora doći do kreiranja novog bloka koji se dodaje u lanac. U svakoj rundi korisnici su nasumično podijeljeni u grupu koja se natječe u kreiranju novog bloka te u grupu koja glasa koji će se blok dodati u mrežu. Pri odabiru korisnika koristi *Verifiable Random Functions*(VRF), funkcije za dobivanje pseudo nasumičnog izlaza. Navedene funkcije su slične lutriji gdje svaki korisnički račun ima šansu da bude odabran, gdje šansa raste s količinom algoranda koju korisnik posjeduje [33].

Algorand podržava kreiranje različitih sredstava na blockchainu. Službeni naziva za sredstva koja se nalaze na Algorandu je Algorand Standard Assets(ASA) [34]. Podržavaju kreiranje nezamjenjivih i zamjenjivih tokena.



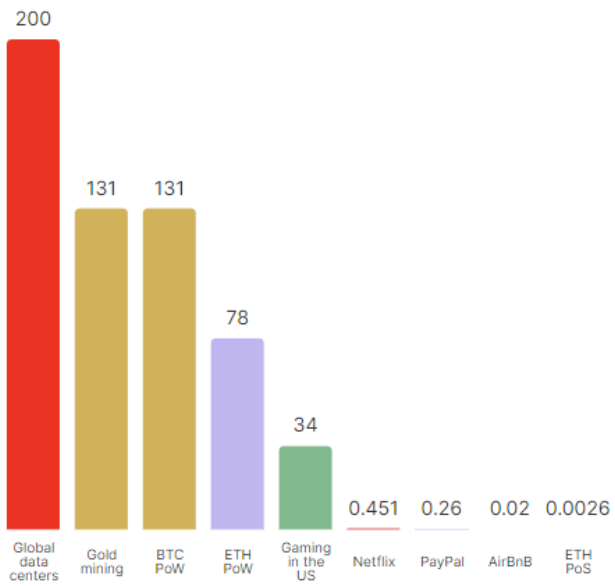
Slika 3.1 Pure Proof of Stake konsenzus mehanizam [1]

3.1.2 Ethereum

Ethereum je decentralizirani blockchain koji je nastao 2015. godine te je trenutno drugi najvrjedniji blockchain nakon bitcoina [35]. Otvorenog je koda te je jedan od prvih uveo pametne ugovore te razvoj decentraliziranih aplikacija na blockchainu [36]. Zbog svoje popularnosti i fokusa na razvoj novih proizvoda ima izrazito razvijenu zajednicu programera te veliki broj dostupnih resursa.

Ethereum je originalno koristio PoW mehanizam za potvrđivanje i dodavanje blokova u mrežu. Krajem 2022. godine došlo je do najveće promjene u povijesti Ethereuma kada su se prebacili na PoS mehanizam [2]. Time su smanjili potrošnju energije potrebnu za napajanje mreže za 99,5 posto (Slika 3.2) [2]. Time je Ethereum postao puno ekološki prihvatljivije rješenje.

Ethereum se sastoji od kriptovalute zvane ETH te velikog broja raznovrsnih aplikacija. ETH je glavni pokretač svih radnji na blockchainu. Kada korisnik odluči napraviti transakciju ili koristiti aplikaciju na mreži mora platiti proviziju. Provizija se koristi kao nagrada i inicijativa za validatore koji se brinu da se samo validni



Slika 3.2 Godišnja potrošnja energije u Twh u godini [2]

blokovi dodaju na blockchain [37].

Za rad s nepromjenjivim tokenima Ethereum koristi ERC-721 standard. Prema navedenom standardu svaki token ima svoj jedinstveni identifikator i adresu ugovora. Može postojati više različitih jedinstvenih tokena na istom ugovoru. ERC-721 pruža različite metode za kreiranje nezamjenjivih tokena, njihov prijenos, dohvat metapodataka i brojne druge [38].

Veliki fokus Ethereuma je razvoj decentraliziranih aplikacija (eng. *dapps*). Decentralizirane aplikacije koriste tehnologiju pametnih ugovora kako bi kod koji se tradicionalno nalazi na centraliziranom poslužitelju postavili na decentraliziranoj blockchain mreži [39]. Pametni ugovori izvršavaju logiku aplikacije, a sam blockchain se koristi za pohranu podataka [39]. Kada se aplikacija postavi na Ethereum mrežu ne može se više izmijeniti [39]. Decentralizane aplikacije imaju bitno različite karakteristike od centraliziranih te njihova primjena ovisi o zahtjevima projekta [40]. Dodatno, decentralizirane aplikacije predstavljaju novu eksperimentalnu tehnologiju gdje se i dalje razvijaju nove primjene blockchain mreža. Unatoč tome, njihov broj kontinuirano raste i pokriva različita područja ljudske aktivnosti uključujući igrice,

društvene mreže, financijske aplikacije te brojne druge [39].

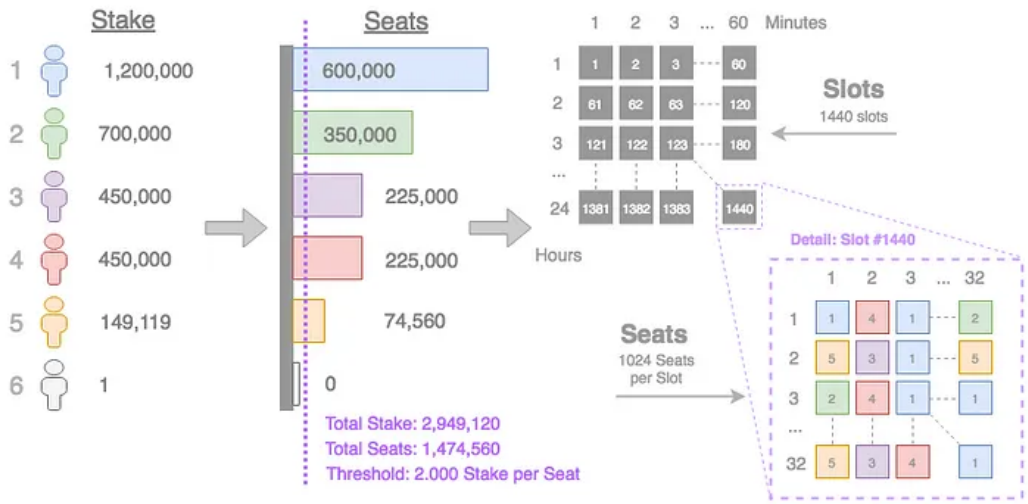
3.1.3 Near

Near je blockchain dizajniran s fokusom na jednostavnost korištenja te na smanjenje negativnog utjecaja na okoliš. Osmišljen je da ima visoke performanse, skalabilnost i sigurnost. NEAR je dizajniran kao jednostavan blockchain koji se može koristiti kao infrastruktura za daljnji razvoj. Fokus na performansu je vidljiv u brzini i jeftinosti transakcija [41]. Dodatno, kroz ljudski čitljiva imena računa, online novčanike, detaljnu dokumentaciju te mogućnost razvoja pametnih ugovora s pomoću JavaScripta dizajniran je za lako korištenje [41].

NEAR pripada skupini blockchaina prvog sloja (eng. *layer one*) [41]. Blockchain prvog sloja predstavlja baznu mrežu i njenu infrastrukturu. Dodatno, može validirati i izvršavati transakcije te se koriste kao podloga na koju se dodatno razvijaju mreže drugog sloja. Imaju nativnu kriptovalutu koja služi za plaćanje provizija i za nagrađivanje korisnika koji osiguravaju mrežu i potvrđuju transakcije. Drugi primjeri te skupine su: Bitcoin, Ethereum i BNB Chain [42].

NEAR koristi varijaciju PoS konsenzusa te time smanjuje potrošnju energije potrebnu za funkcioniranje mreže [41]. *Thresholded Proof of Stake* (TPoS) je mehanizam za postizanje konsenzusa koja se bazira na ideji povećanja broja sudionika u održavanju mreže u svrhu poboljšanja decentralizacije, sigurnosti i ostvarivanje pravedne distribucije nagrade. Cilj mehanizma je odabir velikog broja svjedoka (eng. *witnesses*) kako bi glasali u specifičnom vremenskom intervalu (zadano je jedan dan). Svaki interval se podijeli na manje dijelove (zadano je 1440) gdje svaki ima veliki broj svjedoka (zadano je 1024). Ovim postavkama se dobije 1,474,560 glasačkih mjesta koje treba popuniti kako je prikazano na slici 3.3 [43].

Kako bi sudjelovali u održavanju mreže, korisnici mogu poslati posebnu vrstu transakcije u kojoj je zapisano koliko novaca korisnik želi uložiti. Odabrani ulog se zaključava na minimalno tri dana i korisnik ga ne može koristiti [43]. Near koristi inflacijski mehanizam kako bi potaknuo svjedoke da sudjeluju u održavanju mreže. Stopa inflacije se definira kao postotak ukupnog broja tokena. Time se ohrabruju korisnici koji dugoročno posjeduju kriptovalutu (eng. *HODLers*) da sudjeluju u radu



Slika 3.3 Primjer odabira svjedoka u TPoS procesu [3]

mreže kako bi održali vrijednost vlastiti sredstava [43].

Ima izrazito visok kapacitet za skaliranje kroz korištenje *shardinga*. Dinamičko usitnjavanje znači da sustav može rekonfigurirati dijeljenje i procesiranje podataka dinamički bez potrebe za gašenjem operacijskog sustava. Ovo omogućuje balansiranje procesorske moći i memorije ovisno o potrebama mreže te se time skalira gore ili dolje [44].

Kroz Rainbow Bridge i Project Aurora, Near pruža dodatnu interoperabilnost s izrazito popularnim Ethereumom te olakšava razvoj pametnih ugovora i decentraliziranih aplikacija na njemu [41].

3.1.4 Cardano

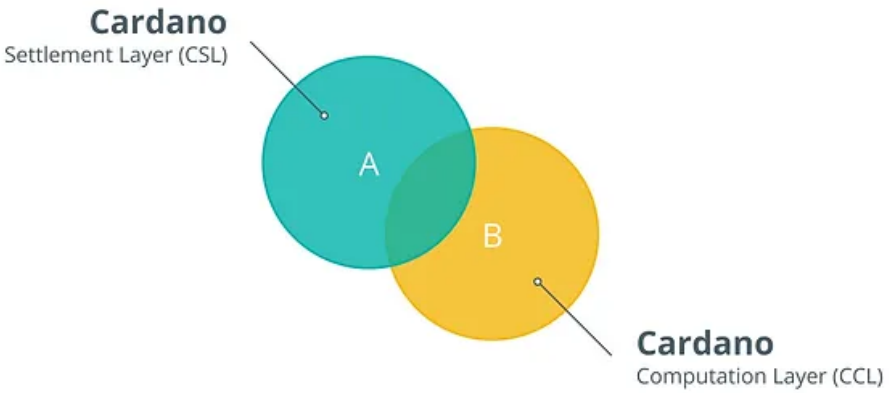
Cardano je blockchain mreža nastala 2017. godine s ciljem povećanja sigurnosti i transparentnosti transakcija [45]. Cardano se postavio kao alternativa Ethereumu, pružajući slične usluge poput pametnih ugovora i decentraliziranih aplikacija [46]. Glavna kriptovalute Cardana je ADA [46].

Cardano koristi jedinstvenu verziju delegiranog PoS konsenzus mehanizma, ta-

kozvani Ouroboros. Navedeni mehanizam podijeli vrijeme u epohe te njih u manje vremenske intervale. U svakom intervalu, vodeći čvor se nasumično odabire kako bi odlučio koje blokove dodati u blockchain. Nakon što epoha završi, biraju se novi vođe za idući vremenski period [47].

Od svojeg začetka, Cardano je imao fokus na akademska istraživanja. Razvoj navedene tehnologije su predvodili stručnjaci na području kriptografije i računarstva. Dodatno, istraživanje i razvoj je podvrgnut temeljitom *peer reviewu* što se smatra rijetkost na području blockchain tehnologije [48].

Cardano je dizajniran u više slojeva kako bi se spriječilo da određene funkcije usporavaju rad drugih (Slika 3.4). Prvi sloj se zove *settlement layer* te je njegova uloga upravljanje transakcijama i stanjima na računu. *Computation layer* se koristi za izvršavanje pametnih ugovora te decentraliziranih aplikacija [47].



Slika 3.4 Slojevi u Cardano mreži [4]

Za razliku od većine drugih blockchain mreža, Cardanom ne upravlja samo jedna organizacija nego je donošenje odluka podijeljeno na 3 različite organizacije gdje svaka ima svoju ulogu. Prva je Cardano Foundation čija je uloga razvijanje i poboljšanje blockhcain mreže. Druga organizacija je Input Output Hong Kong (IOHK) koja se bavi istraživanjem i razvojem softvera. Posljednja organizacija je Emurgo te se bavi poticanjem tvrtki i individualaca da koriste Cardano mrežu [47].

Pametni ugovori su implantirani 2021. godine s Alonzo ažuriranjem blockcha-

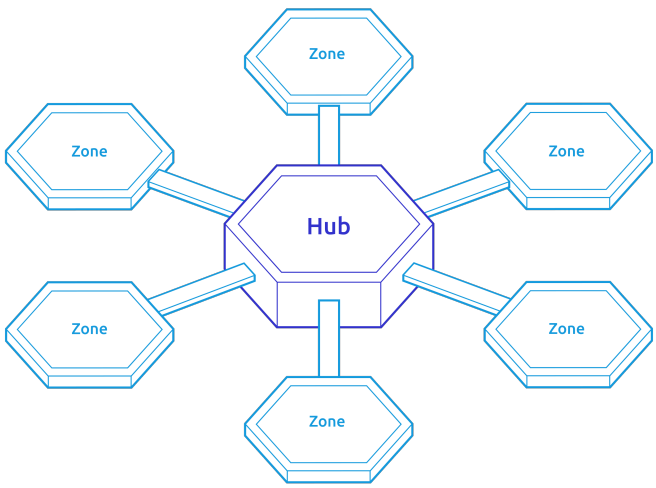
ina. Dodatno, time je omogućena podrška za kreiranje i upravljanje nezamjenjivim tokenima [46].

3.1.5 Cosmos

Cosmos je decentralizirana mreža neovisnih blockchaina koji imaju mogućnost međusobno komunicirati [5]. Trenutno postoji više od 246 aplikacija i servisa na Cosmos mreži poput Terra, Binance Chain, Cosmos Hub itd [49].

Blockchain industrijom prevladavaju samostalne neovisne blockchain mreže koje nemaju mogućnost komuniciranja s drugim mrežama te pokušavaju korisnicima ponuditi različite servise. Time je otežan razvoj, smanjen je broj transakcija koje se mogu obraditi te je moguće da dođe do prenatrpanosti blockchaina [50]. Cosmos nudi rješenje u obliku mreže interoperabilnih blockchain mreža čime se odudara od industrijskog standarda.

Krajnji cilj je kreiranje interneta blockchaina koji bi pružao veliki broj usluga korisnicima te riješio nedostatke prisutne kod izoliranih mreža [5]. Arhitektura navedene mreže je prikazana na Slici 3.5.



Slika 3.5 Dizajn Cosmos "Interneta Blockchaina" [5]

Cosmos se zalaže za kreiranje modularne arhitekture s dvije vrste blockchaina:

središta (eng. *hubs*) i zona (eng. *zones*). Zone predstavljaju heterogene blockchaine dok su središta blockchain mreže posebno dizajnirane za povezivanje zona. Kada se zona poveže sa središtem automatski može slati i primati resurse od svih zona povezane s tim središtem [5]. To znači da bi se zone morale povezati s manjim brojem središta kako bi cijeli sustav funkcionirao. Ovim dizajnom se smanjuje ukupan broj veza koji bi bio prisutan da se sve blockchain mreže povežu direktno [5].

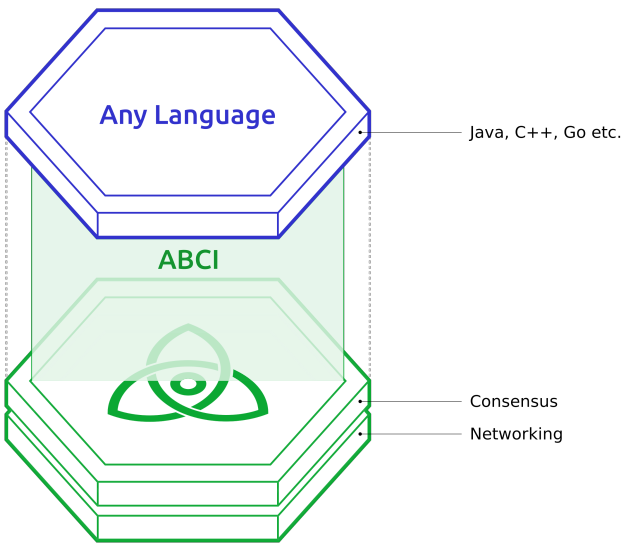
Cosmos Hub je prvi blockchain razvijen na Cosmos mreži te pruža veliki broj servisa bitnih za infrastrukturu mreže. Neke od usluga koje pruža je sigurnost, izmjena resursa između blockchaina, poveznice prema Ethereum i Bitcoin mreži te upravljanje digitalnom imovinom. Glavna valuta Cosmos Hub-a je ATOM [51].

Cosmos nudi velik broj alata otvorenog koda poput Tenderminta, Cosmos SDK-a i IBC-a koji su dizajnirani kako bi olakšali razvoj interoperabilnih, sigurnih i skalabilnih blockchain servisa [5].

Tradicionalno, razvoj blockchaina zahtijeva razvoj svih 5 slojeva spomenutih u potpoglavlju Blockchain. Ovo je dugotrajni proces koji zahtjeva veliki broj resursa. Tendermint BFT pruža rješenje na navedeni problem pakirajući mrežni i konsenzus sloj u generički servis koji dopušta programerima da se fokusiraju na razvoj aplikacija te time postignu ubrzani razvoj. Dodatno, Tendermint je povezan s aplikacijom preko ABCI Application Blockchain Interface protokola koji pruža mogućnost razvoja u bilo kojem programskom jeziku (Slika 3.6) [5].

Cosmos SDK je programski okvir koji olakšava razvoj aplikacijskog sloja na Tendermint BFT-u. Izrazito je modularan te time olakšava razvoj prema potrebama korisnika. Dodatno, olakšava dijeljenje koda među članovima zajednice. Sastoji se od velikog broja korisnih alata, knjižnica, aplikacijsko programskih sučelja te brojnih drugih materijala za razvoj [52].

Povezivanja blockchain-a u mreži postignuto je s pomoću Inter-Blockchain Communication (IBC) protokola. Navedeni protokol omogućava heterogenim blockchainima da komuniciraju te prenose podatke i tokene. Time se pruža visoka razina interoperabilnosti između inače izolirani mreža [53].



Slika 3.6 Povezivanje Tenderminta i aplikacije preko ABCI protokola [5]

3.2 Implementacija korisničkog sučelja

3.2.1 Aplikacijsko programsko sučelje

Implementacija poslužiteljskog dijela programskog sučelja je provedena korištenjem Laravela. Laravel je besplatan *framework* otvorenog koda baziran na PHP-u koji se koristi za kreiranje mrežnih aplikacija [54]. Omogućuje korisniku brzo kreiranje jednostavnog aplikacijskog programskog sučelja (eng. *Application Programming Interface* - API) korištenjem ugrađenih naredbi. Dodatno, pruža kvalitetnu podršku za kreiranje različitih elemenata potrebnih u mrežnoj stranici uključujući autentifikaciju, autorizaciju, validaciju te brojne druge [54]. Dodatna podrška za programere postoji u obliku detaljne dokumentacije [54]. Laravel koristi Composer upravitelj knjižnicama za PHP [55] te Artisan naredbeni redak koji pruža brojne opcije za ubrzani razvoj rješenja [56].

Prvi korak pri implementaciji aplikacijskog korisničkog sučelja u Laravelu je kreiranje projekta s pomoću sljedeće naredbe [54]:

```
composer create-project laravel/laravel ime_projekta
```

Baza podataka korištena pri testiranju programskog sučelja je SQLite. SQLite je knjižnica napisana u C-u koja implementira malenu, brzi i pouzdanu bazu podataka [57]. Konfiguracija je izuzetno lagana te je cijela baza podataka pohranjena u jednoj lokalnoj datoteci [57]. Iako je primjerena za testiranje, u produkcijskom okruženju se preporučuje korištenje robusnijih baza podataka poput PostgreSQL-a [58].

Konfiguracija baze podataka se izvršava s pomoću *.env* datoteke kao što je prikazano u isječku koda 3.1.

```
DB_CONNECTION=sqlite
#DB_HOST=127.0.0.1
#B_PORT=3306
#DB_DATABASE=laravel
#DB_USERNAME=root
#DB_PASSWORD=
```

Isječak koda 3.1 Konfiguriranje baze podataka

Kako bi se ubrzao razvoj aplikacijskog programskog sučelja instalirana je knjižnica Breeze koja nudi jednostavnu implementaciju logike potrebne za autentifikaciju korisnika uključujući: registracija, prijava, potvrda email adrese i druge [59].

Kako bi se proširila mogućnost kontrole pristupa instalirana je i Spatie knjižnica koja se nadovezuje na Breeze te omogućuje kontrolu nad ulogama i dopuštenjima korisnika [60]. Naredbe instaliranje navedenih projekta su:

```
composer require laravel/breeze --dev
composer require spatie/laravel-permission
php artisan breeze:install
```

Laravel koristi migracije kako bi izvršio promjene u strukturi baze podataka. Migracije služe kako bi se osigurala konzistentno stanje strukture baze podataka

između više članova razvojnog tima [61]. Time je svrha migracija slična programima za kontrolu verzije. Migracija se može dodati koristeći sljedeću Artisan komandu [61]:

```
php artisan make:migration ime_migracije
```

U samoj migraciji definiraju se dvije funkcije: *up* i *down*. Prva funkcija se izvršava kada korisnik želi izvršiti migraciju i primijeniti definirane promjene u bazi. Druga funkcija se izvršava kada korisnik želi vratiti bazu u prijašnje stanje te bi ona trebala biti suprotna od funkcije *up* [61]. Isječak koda 3.2 prikazuje migraciju za kreiranje tablice korisnika koju izgenerira Breeze knjižnica.

```
return new class extends Migration {
    public function up(): void{
        Schema::create('users', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('email')->unique();
            $table->timestamp('email_verified_at')->nullable();
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();
        });
    }

    public function down(): void{
        Schema::dropIfExists('users');
    }
};
```

Isječak koda 3.2 Migracija za kreiranje tablice *users*

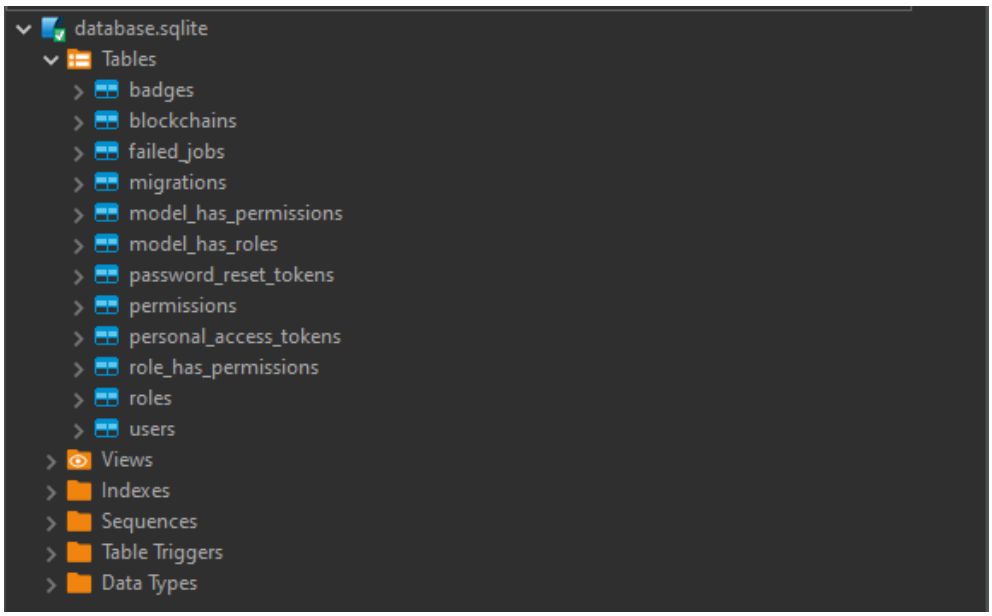
Uz automatski generirane migracije iz Breeze i Spatie knjižnica, dodane su migracije za kreiranje *blockchains* i *badges* tablica. Tablica *blockchains* sadrži sve podržane

blockchain mreže. Tablica *badges* sadrži zapise svih kreiranih bedževa. Kroz razvoj su dodane još dodatne migracije za dodavanje i izmjenu navedenih tablica.

Kako bi se izvršile sve migracije koje se nalaze u mapi *migrations* potrebno je izvršiti sljedeću naredbu:

```
php artisan migrate
```

Na slici 3.7 prikazana je struktura baze nakon izvršenja migracija.



Slika 3.7 Stanje baze nakon izvršenja migracija

Inicijalno punjenje baze izvršeno je s pomoću *seedera* koje Laravel generira s pomoću naredbe [62]:

```
php artisan make:seeder ime_seedera
```

Izgenerirani su *seederi* za uloge korisnika, korisnike i dostupne blockchain tehnologije. Svaka generirana klasa sadrži funkciju *run* u kojoj se definiraju podaci kojima će se populirati baza podataka. U isječku koda 3.3 prikazan je *seeder* za kreiranje korisnika u bazi podataka.

```
class UserSeeder extends Seeder {
    public function run(): void
    {
        User::truncate();
        $user = User::factory()->create([
            'name' => 'admin',
            'email' => 'test@example.com',
        ]);

        $user->assignRole(Role::whereName('admin')->first());
        $user->save();
    }
}
```

Isječak koda 3.3 *Seeder* za kreiranje korisnika

Naredba za pokretanje *seedera* je sljedeća:

```
php artisan db:seed
```

Komuniciranje s bazom je u Laravelu implementirano s pomoću Eloquent ORM-a (eng. *Object relational mapper*) [63]. Svaka tablica u bazi podataka mora imati odgovarajući Eloquent model koji se koristi za interakciju s navedenom tablicom [63]. Podražava sve standardne operacije uključujući dohvata, kreiranja, brisanja te ažuriranja zapisa u bazi. U modelu se mogu dodatno definirati različiti parametri te relacije između modela na isti način kao i u bazi [63].

Za generiranje modela se koristi sljedeća naredba [63]:

```
php artisan make:model ime_modela
```

Za potrebe projekta su kreirani modeli *User*, *Blockchain* te *Badge* (Isječak koda 3.4).

```
class Badge extends Model {
    use HasFactory;

    protected $fillable = ['name', 'description',
    'created_amount', 'blockchain_id',
    'img_path', 'current_amount', 'user_id',
    'token_id', 'original_address',
    'sent_to_address', 'properties',
    'local_img_path', 'badge_created_at'];

    protected $casts = [
        'badge_created_at' => 'datetime:d-m-Y',
    ];

    public function user(): BelongsTo {
        return $this->belongsTo(Badge::class);
    }

    public function blockchain(): BelongsTo {
        return $this->belongsTo(Blockchain::class);
    }
}
```

Isječak koda 3.4 Model koji predstavlja *badges* tablicu

Sljedeći korak u razvoju aplikacijskog programskog sučelja je definirati krajnje točke (eng. *endpoint*) koje će korisničko sučelje pozivati. U Laravelu je to implementirano s pomoću kontrolera (eng. *Controller*) [64]. Kontroleri su posebne klase u kojima se nalazi sva logika vezana uz upravljanje HTTP zahtjevima [64].

Za generiranje kontrolera je korištena sljedeća naredba [64]:

```
php artisan make:controller ime_kontrolera --api --model=ImeModela
```

Dodatne zastavice *api* i *model* su korištene kako bi se specificiralo koju vrstu kontrolera izgenerirati. Zastavica *api* generira kontroler koji sadrži sve funkcije potrebne za implementiranje aplikacijskog programskog sučelja. (Isječak koda 3.5) [64]. Zastavica *model* specificira koji model se treba koristiti u generiranim zahtjevima [64]. Time se ubrzava dohvat željenih modela te je potrebno manje koda.

```
class BlockchainController extends Controller {
    public function index()
    {
        try {
            $blockchains = Blockchain::get();
            return response()->json([
                'message'=> 'Blockchains fetched successfully',
                'blockchains'=>$blockchains
            ]);
        }
        catch (\Exception $e) {
            report($e);
            return response()->json([
                'There was an error while fetching blockchains'
            ], 500);
        }
    }

    public function store(Request $request) {}

    public function show(string $id) {}

    public function update(Request $request, string $id){}

    public function destroy(string $id){}
}
```

Isječak koda 3.5 Kontroler za upravljanje implementiranim blockchain mrežama

Za potrebe projekta kreirani su kontroleri *BlockchainController*, *UserController* te *BadgeController*. *UserController* sadrži osnovne CRUD operacije te služi za upravljanje korisnicima. Ova funkcionalnost je trenutno onemogućena jer je naknadno utvrđeno da ne spada u opseg projekta te je uključena u kodu u slučaju buduće nadogradnje.

BlockchainController sadrži CRUD operacije za upravljanje dostupnim blockchain mrežama te je trenutno omogućena samo opcija za dohvat svih dostupnih mreža.

BadgeController sadrži logiku vezanu uz kreiranje, slanje i prikaz bedževa. Funkcija *index* prikazana u isječku koda 3.6 koristi se za dohvat svih bedževa koji su pohranjeni u bazi podataka za određeni blockchain. Dodatno, bitna je funkcija *paginate(12)* koja na dohvat dodaje automatsku paginaciju od 12 stavki. Koriste za prikaz bedževa na korisničkom sučelju.

Funkcija *getAllIds* u isječku koda 3.6 dohvaća samo *id* stupac svih bedževa za određeni blockchain. Navedena funkcija je implementirana kako bi se smanjio broj poziva prema vanjskim aplikacijsko programskim sučeljima pri ažuriranju svih bedževa.

```
public function index(Blockchain $blockchain) {
    try {
        $badges = Badge::where('blockchain_id', $blockchain->id)
        ->paginate(12);

        return response()->json([
            'message' => 'Badges fetched successfully',
            'badges' => $badges
        ], 200);
```

```
        } catch (\Exception $e) {
            report($e);
            return response()->json([
                'There was an error while fetching badges'
            ], 500);
        }
    }

    public function getAllIds(Blockchain $blockchain) {
        try {
            $badges = Badge::where('blockchain_id', $blockchain->id)
            ->get()->map(function ($el) {
                return (int)$el->token_id;
            });

            return response()->json([
                'message' => 'Badge ids fetched successfully',
                'badges' => $badges
            ], 200);
        } catch (\Exception $e) {
            report($e);
            return response()->json([
                'There was an error while uploading data to IPFS'
            ], 500);
        }
    }
```

Isječak koda 3.6 Funkcije za dohvat bedževa i dohvat *id-a*

Funkcija prikazana u isječku koda 3.7 koristi se pri kreiranju bedža za pohranu slike i metapodataka. Na slici nisu prikazana pravila za validaciju. Ako je zahtjev

uspješno validiran, slika za bedž se pohranjuje lokalno te učitava na *pinatu*. Za interakciju s pinatom se koristi pomoćna klasa *Pinata* te varijable definirane u *.env* datoteci.

U slučaju da se slika uspješno pohrani, kreira se objekt s metapodacima koje je korisnik upisao preko korisničkog sučelja. Dodatno, metapodaci sadrže poveznicu do slike na *pinati*, putanju do lokalno pohranjene slike te vrijeme kada je kreiran bedž. Navedeni objekt se učitava na *pinatu*. U slučaju da bilo koja od navedenih operacija ne uspije, zahtjev vraća grešku te ju zapisuje u *.log* datoteku. Funkija vraća odgovor od *pinata* API-a.

```
$path = Storage::disk('public')->put('images/', $request->image);
$fullPath = Storage::disk('public')->path($path);

$pinata = new Pinata(env('PINATA_API_KEY'),
    env('PINATA_SECRET_API_KEY'));
$responseFile = $pinata->pinFileToIPFS($fullPath);

$metaData = [
    "name" => $request->name,
    "description" => $request->description,
    "image" => "https://gateway.pinata.cloud/ipfs/"
        . $responseFile['IpfsHash'],
    "properties" => json_decode($request->properties),
    "created_at" => Carbon::now()->addHour(),
    "local_image" => $path
];

$fileName = substr($path, strrpos($path, '/')
+ 1, strrpos($path, '.') - strpos($path, '/') + 1 - 3);
Storage::disk('public')
->put('metadata/' . $fileName . '.json', json_encode($metaData));

$responseJson = $pinata->pinJSONToIPFS($metaData);
```

```
return response()->json(['
    message' => 'Badge created successfully',
    'responseJson' => $responseJson,
    'responseFile' => $responseFile
], 200);
```

Isječak koda 3.7 Funkcija za pohranu slike i metapodataka

Funkcija *updateSentToAddress* se koristi pri slanju bedževa drugom korisniku. Sadrži validator za adresu na koju se poslao bedž. Ako je zahtjev uspješno validiran ažurira se atribut *sent_to_address* (Isječak koda 3.8).

```
DB::beginTransaction();
Badge::find($id)->update(
    ['sent_to_address' => $request->sent_to_address]
);
DB::commit();
return response()->json([
    'message' => 'Sent to address updated successfully'
], 200);
```

Isječak koda 3.8 Funkcija za ažuriranje adrese na koju je poslan bedž

Funkcija *refresh* se koristi za osvježivanje postojećih bedževa (Isječak koda 3.9) za određenu blockchain mrežu. Prvo se dohvaća autentificirani korisnik. Nakon toga se validiraju podaci u zahtjevu. U slučaju da je zahtjev validan, za svaki bedž se poziva funkcija *firstOrNew*. Navedena funkcija dohvaća bedž ovisno o atributima *token_id* i *blockchain_id*. U slučaju da ne postoji traženi redak u bazi podataka kreira se novi. Time je osigurano da se svaki bedž zapiše samo jednom. Funkcija vraća listu bedževa korisniku kako bi se osvježio prikaz na korisničkom sučelju.

```
DB::beginTransaction();

if ($request->assets) {
    foreach ($request->assets as $asset) {
        $badge = Badge::firstOrNew(
        [
            'token_id' => (int)$asset['tokenId'],
            'blockchain_id' => $request->blockchain_id
        ],
        [
            'blockchain_id' => $request->blockchain_id,
            'name' => isset($asset['name']) ? $asset['name'] : '',
            'description' => $asset['description']
            ? $asset['description'] : '',
            'current_amount' => $asset['current_amount'],
            'created_amount' => $asset['created_amount'],
            'img_path' => isset($asset['image_url'])
            ? $asset['image_url'] : '',
            'token_id' => $asset['tokenId'],
            'original_address' => $request->original_address,
            'properties' => isset($asset['properties'])
            ? $asset['properties'] : null,
            'user_id' => $user->id,
            'local_img_path' => isset($asset['local_img_path'])
            ? env('APP_URL') . '/storage//'
            . $asset['local_img_path'] : null,
            'badge_created_at' => $asset['badge_created_at']
        ]);
        $badge->save();
    }
}
$badges = Badge::whereBlockchainId($request->blockchain_id)
    ->paginate(12);
```

```
DB::commit();
return response()->json([
    'message' => 'Badges refreshed successfully',
    'badges' => $badges
], 200);
```

Isječak koda 3.9 Funkcija za osvježivanje postojećih bedževa

Posljednji korak potreban za implementaciju API-a je postavljanje ruta te njihovo povezivanje s odgovarajućim funkcijama u kontrolerima. Rješenje je prikazano u isječku koda 3.10.

Ovime su implementirane osnovne funkcionalnosti potrebne za upravljanje bedževima. Određeni manje relevantni detalji su izostavljeni te se nalaze u git repozitoriju.

```
Route::middleware(['auth:sanctum', 'isAdmin'])->group(function () {
    Route::get('/blockchains', [BlockchainController::class, 'index']);
    Route::get('/badges/{blockchain}',
        [BadgeController::class, 'index']);
    Route::get('/badge-ids/{blockchain}',
        [BadgeController::class, 'getAllIds']);
    Route::post('/badges', [BadgeController::class,
        'store']);
    Route::post('/badges/refresh', [BadgeController::class,
        'refresh']);
    Route::put('/badges/sentToAddress/{id}',
        [BadgeController::class, 'updateSentToAddress']
    );
});
```

Isječak koda 3.10 Definiranje ruta u aplikaciji

3.2.2 Korisničko sučelje

Korisničko sučelje je implementirana s pomoću Vue-a. Vue je brz, svestran i intuitivan JavaScript *framework* za kreiranje korisničkih sučelja [65]. Bazira se na kreiranju komponenti koje se po potrebi slažu na stranicu. Time se povećava modularnost te ponovno korištenje koda. Također, ima jako opširnu dokumentaciju te brojne dodatne pakete za različite potrebe korisnika [65].

Izgled korisničkog sučelja je implementiran korištenjem Vuetify knjižnice otvorenog koda [66]. Navedena knjižnica sadrži gotove često korištene komponente za Vue čime se ubrzava razvojni proces sučelja.

Prvi korak pri kreiranju korisničkog sučelja je generiranje Vue projekta s pomoću sljdeće naredbe [65]:

```
npm create vue@latest
```

Pri pokretanju naredbe se pojavljuju dodatne opcije za konfiguriranje objekta prikazane na slici 3.8. Odabrane su dodatne opcije *pinia* te *vue router*. *Pinia* služi za upravljanje stanjem korisničkog sučelja te je u sklopu projekta korišteno za pohranu trenutno ulogiranog korisnika. *Vue router* služi za upravljane rutama na korisničkom sučelju.

Dodatno, za instaliranje *Vuetify* knjižnice potrebno je pokrenuti sljedeću naredbu [66]:

```
npm install vuetify --save
```

Ostali paketi su dodani po potrebi kroz razvoj rješenja te se nalaze u *package.json* datoteci.

Korisničko sučelje se sastoji od nekoliko stranica, gdje svaka od njih ima više komponenti. Stranice vezane uz upravljanje korisnicima i dostupnim blockchain mrežama su sakrivene jer je dogovoreno da navedene funkcionalnosti nisu potrebne u sklopu projekta. Njihov kod je ostavljen u slučaju buduće nadogradnje projekta. U radu će se opisati samo najbitniji dijelovi stranica i komponenti. Konfiguriranje klijenta

```
✓ Project name: … <your-project-name>
✓ Add TypeScript? … No / Yes
✓ Add JSX Support? … No / Yes
✓ Add Vue Router for Single Page Application development? … No / Yes
✓ Add Pinia for state management? … No / Yes
✓ Add Vitest for Unit testing? … No / Yes
✓ Add an End-to-End Testing Solution? … No / Cypress / Playwright
✓ Add ESLint for code quality? … No / Yes
✓ Add Prettier for code formatting? … No / Yes

Scaffolding project in ./<your-project-name>...
Done.
```

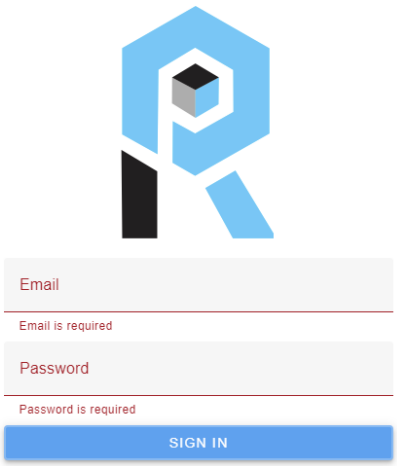
Slika 3.8 Kreiranje Vue projekta

za slanje zahtjeva, konfiguriranje rutera te slični dijelovi aplikacije koji se ne odnose direktno na rad s bedževima se mogu detaljnije pregledati na git repozitoriju.

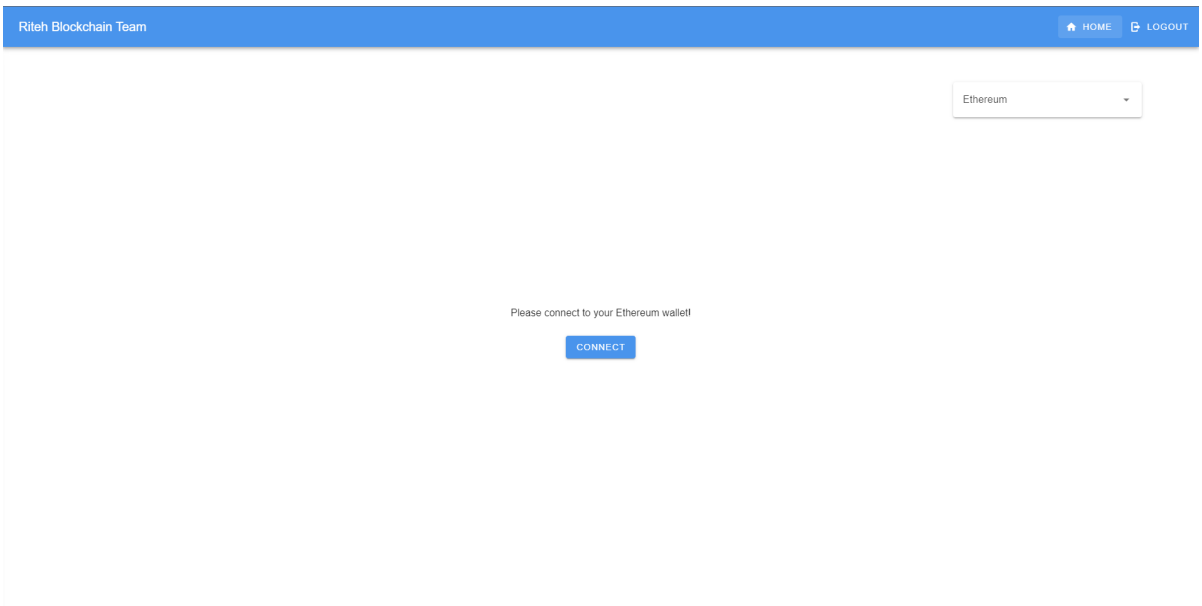
Glavna stranica koja se prikazuje kada korisnik nije prijavljen je *Login*. Navedena stranica služi za autentifikaciju korisnika te se sastoji od jednostavne forme za unos email adrese i korisničke šifre (Slika 3.9).

Stranica sadrži validaciju za navedena polja. Pritiskom na gumb *Sign in* šalje se zahtjev na poslužitelj. Poslužitelj izvršava autentifikaciju korisnika. U slučaju uspješnog zahtjeva dohvaćaju se podaci o trenutnom korisniku te s pomoću *pinie* pohranjuju u *local storage*. Nakon toga korisnik se preusmjerava na stranicu *BadgeGridView*. U slučaju greške ili neuspješne autentifikacije, ispisuje se poruka korisniku.

Stranica *BadgeGridView* je glavna stranica za rad s bedževima te se prikazuje kada je korisnik uspješno prijavljen. Stranica se sastoji od padajućeg izbornika za odabir blockchain mreže. Izbornik se populira s podacima koji se nalaze u bazi u tablici *blockchains* kako je spomenuto u poglavlju Aplikacijsko programsko sučelje. Glavni dio stranice sadrži poruku da se korisnik mora povezati s odgovarajućim novčanikom (Slika 3.10). Pritiskom na gumb *Connect* poziva se funkcija za prijavu u novčanik. Ovisno o odabranoj blockchain mreži, korisnik se mora ulogirati u različiti novčanik te se otvara nova stranica ili ekstenzija za prijavu.

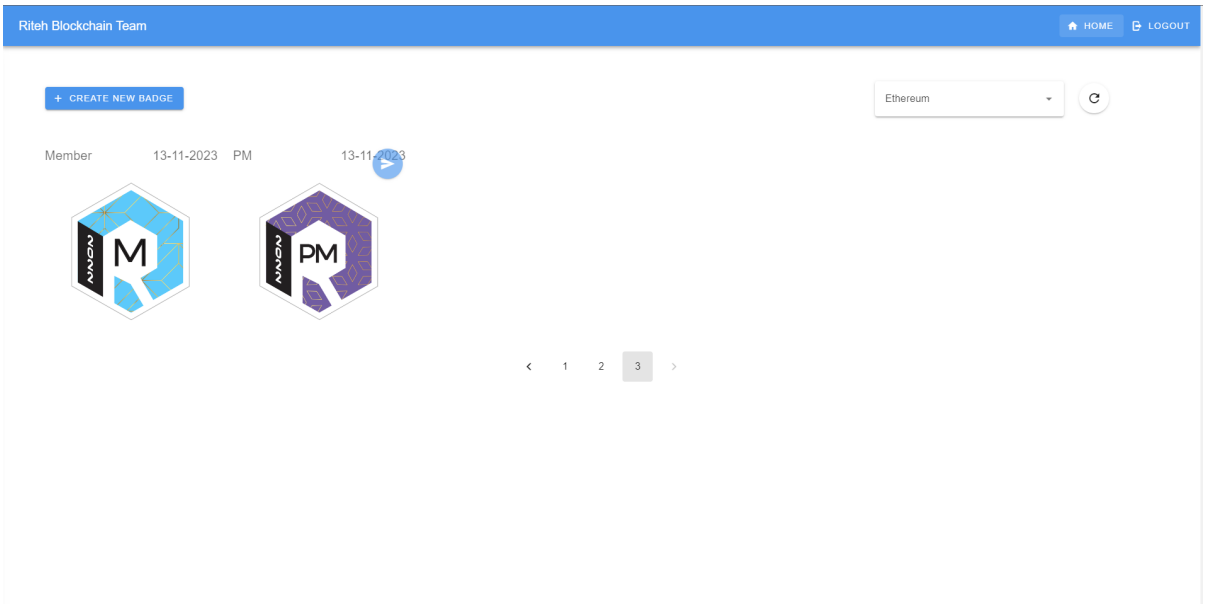


Slika 3.9 Stranica za prijavu



Slika 3.10 Stranica za povezivanje na novčanik

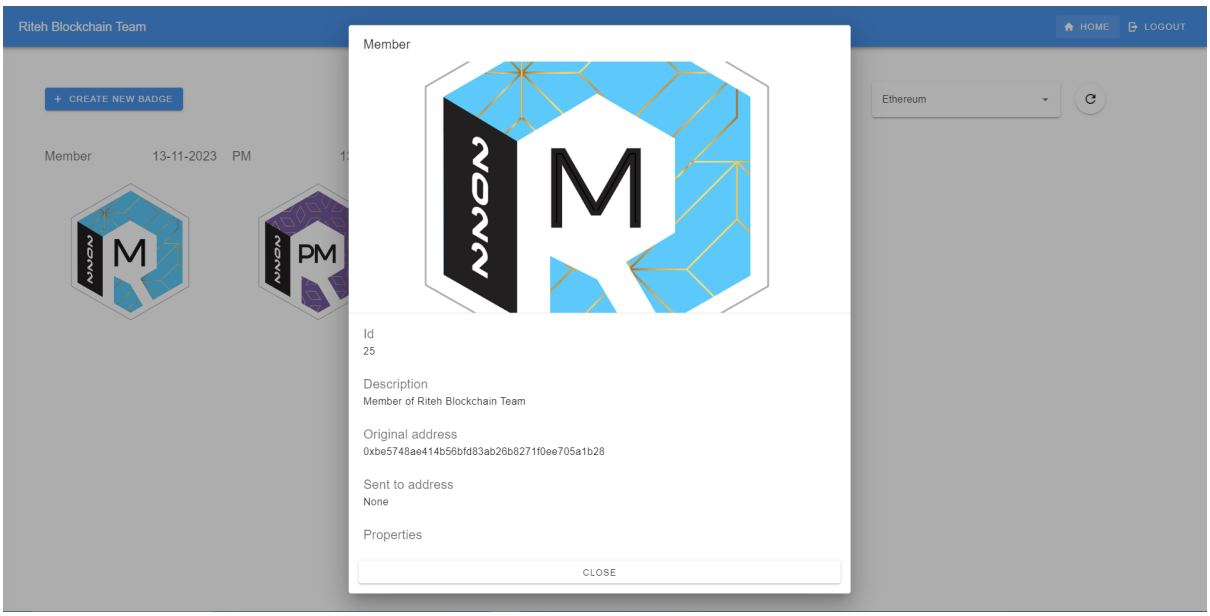
Nakon što se korisnik spoji na novčanik, poziva se funkcija za dohvat svih bedževa pohranjenih u lokalnoj bazi za trenutnog korisnika i za trenutno odabranu blockchain mrežu. Bedževi su prikazani kao responzivna mreža kartica koja sadrži sliku, ime te datum kreiranja bedža. Dodatno, implementirana je paginacija zbog preglednosti te kako bi se izbjeglo nepotrebno dohvaćanje svih bedževa (Slika 3.11).



Slika 3.11 Stranica s prikazom kreiranih bedževa

Pritiskom na određenu karticu otvara se prozor koji prikazuje sve podatke vezane uz određeni bedž. Sadrži sliku, jedinstveni identifikacijski broj bedža, ime, opis, originalnu adresu koja ga je kreirala, adresu na koju je poslan te listu svojstava. Navedeni prozor definiran je u komponenti *BadgeDetailsDialog*, ne sadrži dodatnu logiku te služi samo za prikaz podataka. (Slika 3.12).

Postavljanjem kursora iznad određenog bedža, u slučaju da još nije poslan, pojavljuje se strelica koja otvara prozor za slanje bedža (Slika 3.13). Prozor se nalazi u komponenti *SendBadgeDialog* te sadrži polje za unos adrese. Pritiskom na tipku *send* poziva se metodu za slanje bedža koja se razlikuje ovisno o odabranoj blockchain mreži. Određene blockchain mreže će otvoriti novčanik kako bi se potvrdila transakcija. U slučaju uspješne transakcije šalje se dodatni zahtjev na poslužitelj

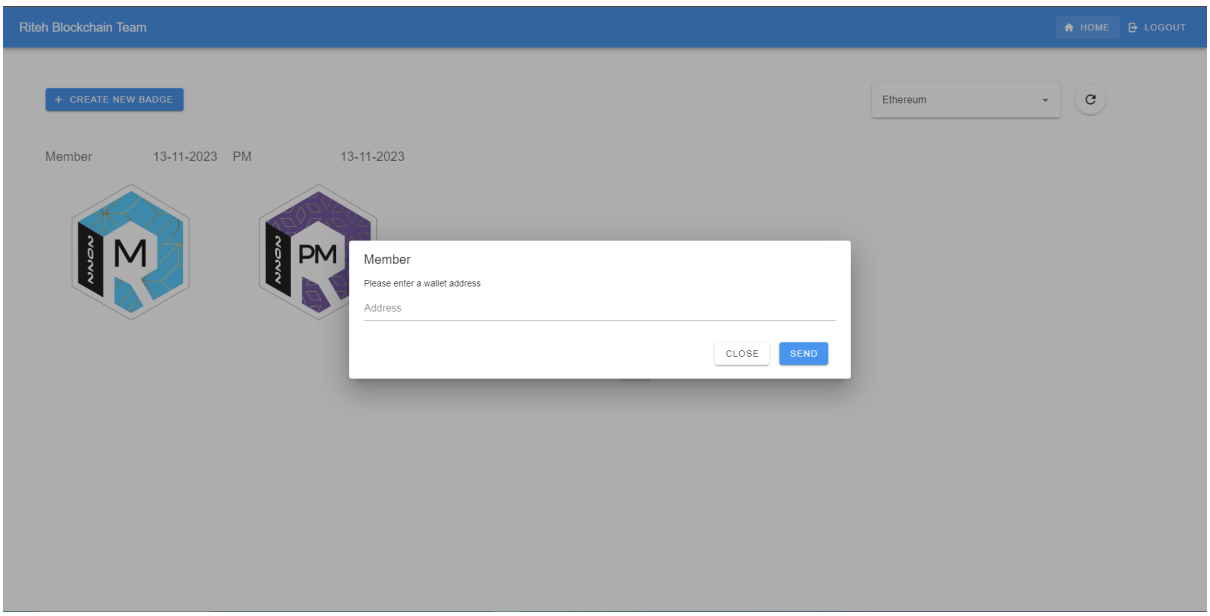


Slika 3.12 Prozor s detaljima bedža

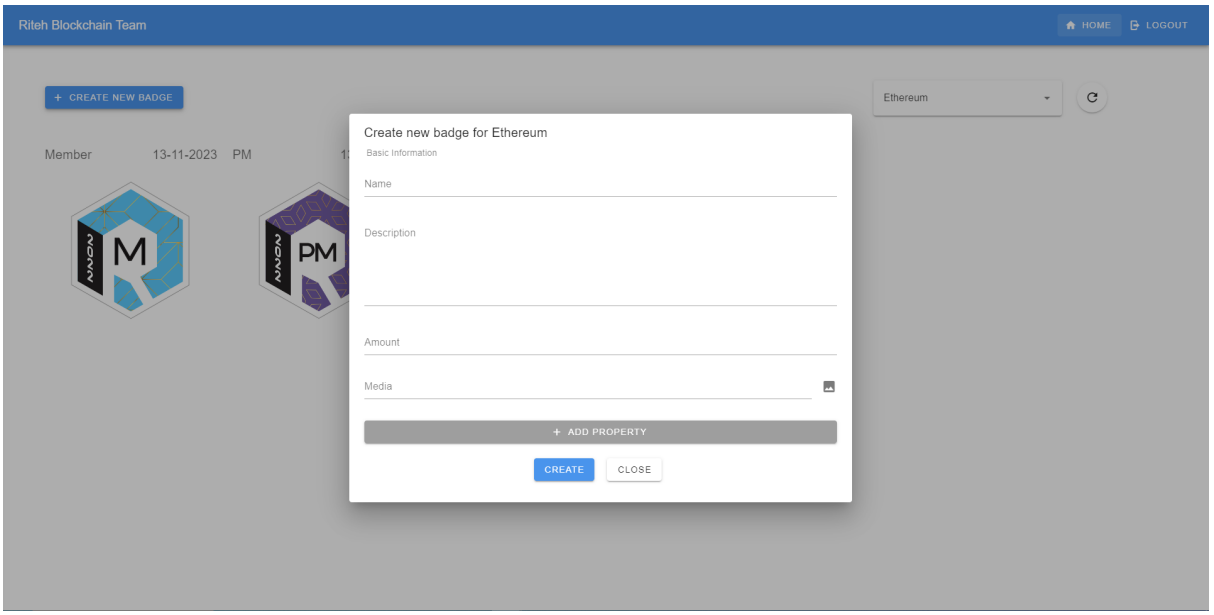
kako bi se ažurirala adresa na koju je poslan bedž u bazi podataka (potpoglavlje Aplikacijsko programsko sučelje). Dodatno, prikazuje se poruka o uspješnom slanju, ažurira se vrijednost adrese na koju je poslan bedž na korisničkom sučelju te se zatvara prozor. U slučaju greške ili neuspješne transakcije, prikazuje se odgovarajuća poruka te se greška ispisuje u konzolu.

Kada se korisnik poveže na novčanik, prikazuju se i gumb za kreiranje novog bedža te gumb za osvježivanje bedževa pohranjenih u bazi podataka.

Pritiskom na tipku *Create new badge* otvara se prozor koji sadrži formu za kreiranje novih bedževa (Slika 3.14). Forma sadrži validaciju za obavezne podatke. Nakon što korisnik upiše željene vrijednosti, pritiskom na tipku *Create* pokreće se kreiranje bedževa. Navedena funkcija šalje zahtjev na poslužitelj koji je opisan u potpoglavlju Aplikacijsko programsko sučelje. Poslužitelj izvršava logiku za pohranu slike i metapodataka lokalno te na IPFS. Ako je zahtjev uspješan poziva se odgovarajuća funkcija za kreiranje novog bedža ovisno o odabranoj blockchain mreži. Određene mreže zahtijevaju od korisnika da potvrdi transakciju. U slučaju da je transakcija uspješna ili neuspješna, zatvara se dijalog te se prikazuje odgovarajuća poruka.



Slika 3.13 Prozor za slanje postojećih bedževa



Slika 3.14 Prozor za kreiranje novih bedževa

Pritiskom na tipku za osvježivanje prikazanu na slici 3.11 dohvaćaju se podaci o tokenima s blockchain mreže te se pohranjuju u lokalnu bazu podataka. Navedena funkcija dohvaća sve bedževe koje je korisnik kreirao na specifičnoj blockchain mreži, a koji ne postoje u lokalnoj bazi (tu se koristi dodatni zahtjev *getAllIds* opisan u potpoglavlju Aplikacijsko programsko sučelje). Svaka mreža strukturira podatke na drugačiji način te neke zahtijevaju slanje dodatnih zahtjeva na *pinatu* kako bi se dohvatili svi podaci. Neovisno o mreži, podaci se strukturiraju u listu objekata koji odgovaraju modelu definiranom u Laravelu te se šalju kako bi se pohranili u bazu podataka. U slučaju da je zahtjev uspješan, osvježuju se stavke prikazane u mreži bedževa. U slučaju greške pri dohvatu informacija s blockchain mreže ili pri pohrani na poslužitelj, prikazuje se odgovarajuća poruka te se greška ispisuje u konzolu.

3.2.3 Implementacija različitih blockchaina

Sve operacije za rad s blockchain mrežama implementirane su na način da se poziva odgovarajuća metoda u klasi *GeneralBlockchainService*. Navedena klasa na temelju trenutno odabrane mreže poziva implementaciju metode za specifičnu blockchain mrežu koja se nalazi u zasebnom servisu (Isječak koda 3.11). Opisane su metode za dohvat, kreiranje i slanje nezamjenjivih tokena.

```
async login(blockhain) {
    return await this.getService(blockhain.code).login()
},

async isSignedIn(blockhain) {
    return await this.getService(blockhain.code).isSignedIn()
},

async getAccount(blockchain) {
    return await this.getService(blockchain.code).getAccount()
},

async getAllAssets(blockchain, existingAssets) {
```

```
        return await this.getService(blockchain.code)
            .getAllAssets({existingAssets})
    },

    logout() {},

    async sendAsset(blockchain, options) {
        return this.getService(blockchain.code).sendAsset(options)
    },

    async mintAsset(blockchain, options, badge) {
        return this.getService(blockchain.code).mintAsset(options, badge)
    },
```

Isječak koda 3.11 Klasa *GeneralBlockchainService*

Ethereum

Klasa *EthereumService* sadrži svu logiku potrebnu za generiranje i upravljanje bedževima na Ethereum mreži. Korišten je Metamask novčanik u pregledniku. Dodatno, korišten je Alchemy API (*alchemy-sdk*) kako bi se ubrzao i pojednostavio razvoj. Potrebno je kreirati Alchemy projekt te postaviti API ključ u konfiguracijsku datoteku. Dodatno, korištene su *Web3.js* i *@alch/alchemy-web3* knjižnice kako bi se uspješno poslali zahtjevi za Ethereum.

Potrebno je napisati i podignuti ugovor koji podržava kreiranje novih bedževa. Navedeni procesi detaljnije je opisan u Prilogu 1.

Funkcija za dohvat svih bedževa na Ethereum mreži prikazana je u isječku koda 3.12. Prvi korak je kreiranje *Alchemy* objekta te njegovo konfiguriranje. Za konfiguriranje je potrebno odrediti mrežu (u našem slučaju Testnet) te API ključ. Oba parametre su definirani putem *.env* datoteke. Nakon toga poziva se funkcija za

dohvat svih neizmjenjivih tokena koji su kreirani na ugovoru. Lista nezamjenjivih tokena se iterira te se pretvara u objekt koji odgovara modelu u bazi podataka. Funkcija vraća listu objekata. Navedeni rezultati se šalju na funkciju *refresh* opisanu u potpoglavlju Aplikacijsko programsko sučelje kako bi se pohranili u lokalnu bazu.

```
const nfts = await alchemy.nft.getNftsForContract(contractAddress);
let ownedNfts = nfts['nfts'];
let assetList = [];
for(let nft of ownedNfts) {
    if(options.existingAssets.includes(parseInt(nft['tokenId'])))
        continue
    let asset =
    {
        'name': nft['title'],
        'description': nft['description'] ? nft['description'] : '',
        'current_amount': 1,
        'created_amount': 1,
        'image_url': nft['rawMetadata']['image'],
        'properties': JSON.stringify(nft['rawMetadata']
        ['properties']),
        'tokenId': nft['tokenId'],
        'local_img_path': nft['rawMetadata']['local_image']
        ? nft['rawMetadata']['local_image'] : null,
        'badge_created_at': nft['rawMetadata']['created_at']
        ? nft['rawMetadata']['created_at'] : null
    }

    assetList.push(asset);
}

return assetList;
```

Isječak koda 3.12 Funkcija za dohvat svih bedževa na Ethereumu

Kreiranje novih bedževa prikazano je u isječku koda 3.13. Funkcija prvo dohvaća trenutno prijavljenog korisnika novčanika te generira jedinstveni broj *nonce* na temelju broja poslanih transakcija s navedenog računa. Generira se objekt za transakciju koji se sastoji od adrese koja generira bedž, adrese ugovora, *nonce* broja te metode koja je definirana u pametnom ugovoru. Nakon toga poziva se funkcija *eth_sendTransaction* kako bi se poslala transakcija na mrežu koristeći Metamask novčanik.

```
let account = await this.getAccount();
const nonce = await web3.eth.getTransactionCount(account, 'latest');
let tokenURI = 'https://gateway.pinata.cloud/ipfs/'
    + options.responseJson['IpfsHash'];
let strNonce = String(nonce);
const params = [{
    'from': account,
    'to': contractAddress,
    'nonce': strNonce,
    'gas': '500000',
    'data': nftContract.methods
        .mintMultipleNFT(account, tokenURI, badge.created_amount)
        .encodeABI()
}];

return await window['ethereum']
.request({
    method: 'eth_sendTransaction',
    params,
})
```

Isječak koda 3.13 Funkcija za kreiranje novih bedževa na Ethereumu

Prijenos postojeći bedževa prikazan je u isječku koda 3.14. Kao i kod generiranja dohvaća se trenutni korisnički račun te se generira *nonce*. Funkcija prima kao dodatne

parametre adresu na koju se šalje te identifikacijski broj bedža. Kreira se objekt koji predstavlja parametre transakcije. Na kraju poziva se funkcija *eth_sendTransaction* kako bi se poslala transakcija.

```
let account = await this.getAccount();
const nonce = await web3.eth.getTransactionCount(account, 'latest');
let strNonce = String(nonce);
let to = options.to;
let tokenID = options.tokenID;

const params = [{
    'from': account,
    'to': contractAddress,
    'nonce': strNonce,
    'data': nftContract.methods
        .safeTransferFrom(account, to, tokenID)
        .encodeABI()
}];

return await window['ethereum']
.request({
    method: 'eth_sendTransaction',
    params,
})
```

Isječak koda 3.14 Funkcija za prijenos bedževa na Ethereumu

Algorand

Logika za Algorand se nalazi u klasi *AlgorandService*. Dodatno, korišten je PeraWallet za potpisivanje i potvrđivanje transakcija. Korišten je Purestake API kako bi se izbjeglo lokalno pokretanje čvora te time ubrzao razvoj. Ključ za API je pohranjen

u *.env* datoteci. Za razvoj rješenja korištene su knjižnice *algosdk* i *@perawallet/connect*. Prva knjižnica se koristi za interakciju s blockchain mrežom, odnosno slanje transakcija. Druga knjižnica služi za povezivanje s novčanikom te potpisivanje transakcija.

Funkcija za dohvat svih bedževa prikazana je u isječku koda 3.15. Prvi korak je kreiranje klijenta za slanje zahtjeva s pomoću *algosdk* knjižnice te dohvat javnog ključa pohranjenog u pregledniku. Pozivom na funkciju *accountInformation(account)* dohvaćaju se svi tokeni vezani uz prijavljenog korisnika. Idući korak je iteriranje kroz sve stavke, dohvaćanje podataka pohranjenih na IPFS-u te njihovo formatiranje u objekt koji odgovara modelu u bazi. Za slučaj da token s određenim identifikacijskim brojem već postoji u bazi podataka, preskače se kako bi se smanjio broj nepotrebnih zahtjeva.

```
let client = new algosdk.Algodv2({ 'X-API-Key' : apiKey},
'https://testnet-algorand.api.purestake.io/ps2', '');

let accountString = window.sessionStorage.getItem(ALGORAND_KEY);
let account = JSON.parse(accountString);
let assets = await client.accountInformation(account).do();

let createdAssets = assets['created-assets'];
let assetList = [];

for(let asset of createdAssets) {
    if(options.existingAssets.includes(asset.index)) continue
    let params = asset['params'];
    let url = params['url'];
    let metadata = await axios.get(url)
    let badge =
    {
      'name': metadata.data.name,
      'description': metadata.data.name ? metadata.data.name : '',
      'current_amount': 1,
```

```
            'created_amount': 1,
            'image_url': metadata.data.image,
            'properties': JSON.stringify(metadata.data.properties),
            'tokenId': asset.index,
            'local_img_path': metadata.data.local_image
            ? metadata.data.local_image : null,
            'badge_created_at': metadata.data.created_at
            ? metadata.data.created_at : null
        }
      assetList.push(badge);
    }
    return assetList;
```

Isječak koda 3.15 Funkcija za dohvat svih bedževa u Algorandu

Funkcija za kreiranje novih bedževa prikazana je u isječku koda 3.16. Prvi korak je kreiranje klijenta za *algosdk*, *PeraWallet* te dohvat javnog ključa trenutno prijavljenog korisnika. Algorand tretira sve tokene kao izmjenjive tokene. To znači da je moguće kreirati više bedževe s istim jedinstvenim identifikatorom. Navedeno svojstvo ne odgovara željenim specifikacijama projekta te je zbog toga kreiranje više bedževa izvedeno u petlji koja postavlja parametre bedža te ih dodaje u grupu transakcija. Time je osigurano da svaki token ima jedinstveni *id*. Nakon toga koristi se novčanik kako bi se potpisala grupa transakcija. U slučaju uspješnog potpisivanja šalju se sve transakcije na blockchain mrežu.

```
const peraWallet = new PeraWalletConnect({chainId: chainId});
let client = new algosdk.Algodv2({ 'X-API-Key' : apiKey},
'https://testnet-algorand.api.purestake.io/ps2', '');

let account = await this.getAccount();
const multipleTxnGroups = []
const txns = []
```

```
for (let index = 0; index < parseInt(badge.created_amount); index++) {
    let params = await client.getTransactionParams().do();
    let unitName = "NFT";
    let assetName = badge.name;
    let assetURL = 'https://gateway.pinata.cloud/ipfs/'
    + options.responseJson['IpfsHash'];
    let assetMetadataHash = options.assetMetadataHash;

    let note = undefined;
    let defaultFrozen = false;
    let decimals = 0;
    let total = 1;

    let mintTxn = algosdk.makeAssetCreateTxnWithSuggestedParams(
      account,
      note,
      total,
      decimals,
      defaultFrozen,
      account,
      account,
      account,
      account,
      unitName,
      assetName + index,
      assetURL,
      assetMetadataHash,
      params,
    );
    multipleTxnGroups.push({txn: mintTxn, signers: [account]});
  }
```

```
    const signedTxn = await peraWallet
    .signTransaction([multipleTxnGroups]);

    for (const signedTxnGroup of signedTxn) {
      const {txId} = await client
      .sendRawTransaction(signedTxnGroup).do();
      console.log(txId)
    }
  }
```

Isječak koda 3.16 Funkcija za dohvat svih bedževa u Algorandu

Slanje postojećih bedževa je prikazano u isječku koda 3.17. Potrebno je instancirati klijent za potpisivanje i slanje transakcija. U funkciju su proslijeđeni adresa na koju se šalje bedž te jedinstveni identifikacijski broj bedža. Zatim se kreira transakcija, potpisuje se te šalje na željenu adresu. Bitno je istaknuti da Algorand zahtjeva od korisnika kojem se šalje bedž da *opt-ina* da želi prihvatiti željeni bedž. U suprotnom, doći će do greške koja će biti ispisana u konzolu.

```
const peraWallet = new PeraWalletConnect({chainId: chainId});

let client = new algosdk.Algodv2({ 'X-API-Key' : apiKey},
'https://testnet-algorand.api.purestake.io/ps2', '');
let params = await client.getTransactionParams().do();

let account = await this.getAccount();
let toAddress = options.to;
let tokenID = parseInt(options.tokenID);

let sendTxn = algosdk.makeAssetTransferTxnWithSuggestedParams(
    account, toAddress, undefined,
    undefined, 1, undefined, tokenID,
```

```
        params, undefined
    );

    const singleTxnGroups = [{txn: sendTxn, signers: [account]}];
    const signedTxn = await peraWallet.signTransaction([singleTxnGroups]);
    const {txId} = await client.sendRawTransaction(signedTxn).do();
```

Isječak koda 3.17 Funkcija za dohvat svih bedževa u Algorandu

Near

Za implementiranje Near blockchain mreže korištena je naredbeni redak te *near-api-js* knjižnica. Naredbeni redak je korišten za kreiranje korisničkog računa te generiranje i podizanje pametnog ugovora. Informacije potrebne na korisničkom sučelju kao što su *networkId* i *contractId* pohranjeni su u *.env* datoteku. Dodatno, za kreiranje i prijenos bedževa potrebno je poslati određenu količinu *NEAR* valute kako bi se transakcija uspješno izvršila. Navedeni parametri su pohranjeni u *.env* datoteci. Pri testiranju je utvrđenu da se znaju mijenjati s vremenom te da potrebna količina raste. Ako dođe do toga transakcija neće proći te će se odgovarajuća greška ispisati u konzolu. Iz navedenih razloga kontrola nad tim parametrima je omogućena s pomoću *.env* varijabli.

Funkcija za dohvat svih bedževa prikazana je u isječku koda 3.18. Potrebno je konfigurirati i uspostaviti konekciju u svrhu dobivanja *account* objekta koji predstavlja trenutno povezani korisnički račun novčanika. Navedeni objekt se koristi za slanje zahtjeva blockchain mreži. Pozivom na funkciju *nft_tokens_for_owner* dohvaća se lista svih tokena koji postoje za prijavljenog korisnika. Lista tokena se iterira te u slučaju da već ne postoji u bazi podatka se formatira u objekt koji odgovara modelu u bazi podataka. Nakon toga se vraća lista novokreiranih objekata.

```
    let nearConnection = await connect(connectionConfig);
    const walletConnection = new nearAPI
```

```
.WalletConnection(nearConnection, appKeyPrefix);
let account = walletConnection.account();

const messages = await account.viewFunction(
{
    contractId: contractId,
    methodName: "nft_tokens_for_owner",
    args: {account_id: account.accountId}
});
let assetList = [];
for(let asset of messages) {
  if(options.existingAssets.includes(parseInt(asset['token_id'])))
    continue

  let metadata = asset['metadata']['reference']
  ? await axios.get(asset['metadata']['reference']) : null
  let badge =
  {
      'name': asset['metadata']['title'],
      'description': asset['metadata']['description']
      ? asset['metadata']['description'] : '',
      'current_amount': 1,
      'created_amount': 1,
      'image_url': asset['metadata']['media'],
      'properties': metadata
      ? JSON.stringify(metadata.data.properties) : null,
      'tokenId': asset['tokenId'],
  }
  badge['tokenId'] = asset['token_id']
  assetList.push(badge);
}

return assetList;
```

Isječak koda 3.18 Isječak funkcije za dohvat svih bedževa u Nearu

Kreiranje novih bedževa je prikazano u isječku koda 3.19. Ponavlja se već opisani postupak konfiguriranje i uspostavljanja konekcije. Nakon toga kreira se objekt *contract* koji predstavlja ugovor koji je generiran preko naredbenog retka. Near zahtjeva ručno postavljanje jedinstvenog identifikacijskog broja te se zbog toga dohvaća trenutni maksimalni identifikacijski broj. Kao i Algorand, Near ne podržava kreiranje više kopija bedža što ne odgovara potrebama projekta. Sukladno tome, podrška za kreiranje više bedževa je implementirana s pomoću slanja asinkronih zahtjeva gdje svaki kreira zasebni novi token. Svi tokeni su identični jedino se razlikuju po jedinstvenom identifikacijskom broju. Naredba *Promises.allSettled(promises)* osigurava da se izvršavanje koda nastavi samo nakon što se svi asinkroni zahtjevi izvrše.

```
let nearConnection = await connect(connectionConfig);
const walletConnection = new nearAPI
.WalletConnection(nearConnection, appKeyPrefix);
let account = walletConnection.account();

const contract = new Contract(
  account,
  contractId,
  {
      viewMethods: ["getMessages"],
      changeMethods: ["addMessage"],
  }
);

let assetIdList = await this.getAllAssetIds();
let maxAssetId = Math.max(...assetIdList.map(o => parseInt(o)));

let promises = []
for (let index = 0; index < badge.created_amount; index++) {
    let metadata = {
```

```
            account_id:account.accountId,
            token_id: (maxAssetId + index + 1).toString(),
            receiver_id: account.accountId,
            token_metadata: {
                title: badge.name,
                description: badge.description,
                media: "https://gateway.pinata.cloud/ipfs/"
                + options['responseFile']['IpfsHash'],
                copies: 1,
                reference: "https://gateway.pinata.cloud/ipfs/"
                + options['responseJson']['IpfsHash']
            }
        }

        promises.push(account.functionCall({
            contractId: contract.contractId,
            methodName: "nft_mint", args: metadata,
            attachedDeposit: new BN(import.meta.env.VITE_API_MINT_DEPOSIT)
        }))
    }
    return Promise.allSettled(promises)
```

Isječak koda 3.19 Isječak funkcije za generiranje bedža u Nearu

Prijenos postojećih bedževa prikazan je u isječku koda 3.20. Kao i kod preostalih funkcija potrebno je uspostaviti Near konekciju kako bi se došlo do objekta koji predstavlja trenutno prijavljeni korisnički novčanik. Nakon toga kreira se objekt koji predstavlja ugovor. Funkcija prihvaća parametre za adresu na koju se šalje bedž te njegov jedinstveni identifikacijski broj. Pozivom na funkciju *nft_transfer* željeni bedž se prebacuje na unesenu adresu.

```
let nearConnection = await connect(connectionConfig);
const walletConnection = new nearAPI
```

```
.WalletConnection(nearConnection, appKeyPrefix);
let account = walletConnection.account();

let to = options.to;
let tokenID = options.tokenID;

const contract = new Contract(
    account,
    account.accountId,
    {
        viewMethods: ["getMessages"],
        changeMethods: ["addMessage"],
    }
);
let metadata = {
    account_id: account.accountId,
    token_id: tokenID,
    receiver_id: to,
    amount: 1,
}
return await account.functionCall({
    contractId: contract.contractId,
    methodName: "nft_transfer",
    args: metadata,
    attachedDeposit: new BN(import.meta.env.VITE_API_SEND_DEPOSIT)
})
```

Isječak koda 3.20 Isječak funkcije za slanje bedža u Nearu

Cosmos

Cosmos blockchain mreža je dizajnirana za kreiranje i povezivanje više različitih blockchain mreža. Sukladno tome, postoji veliki broj distribuiranih sustava dizajniranih u različite svrhe. Tijekom istraživanja odlučeno je koristiti Stargaze jer najbolje zadovoljava potrebe projekta te jer ima jednu od najboljih dokumentacija.

Pri kreiranju bedževa praćena je dokumentacija opisana na njihovoj stranici [67]. Uspješno su kreirane kolekcije i bedževi kao što je prikazano na slici 3.15. Unatoč tome, došlo je do greške gdje nije moguće vidjeti metapodatke i sliku povezanu uz bedž. Testirano je kreiranje bedža koristeći *pinatu* i *NFT storage* za pohranu podataka. U oba slučaja dolazi do iste greške. Cjelokupni postupak kreiranja bedža je ponovljen s više razlitih računa, ali daje iste rezultate. Testirane su različite putanje do slika i metapodataka te različite slike i metapodaci, ali se javlja ista greška. Nisu pronađeni korisnici koji su naišli na slični problem te se zbog toga odustalo od navedenog pristupa.

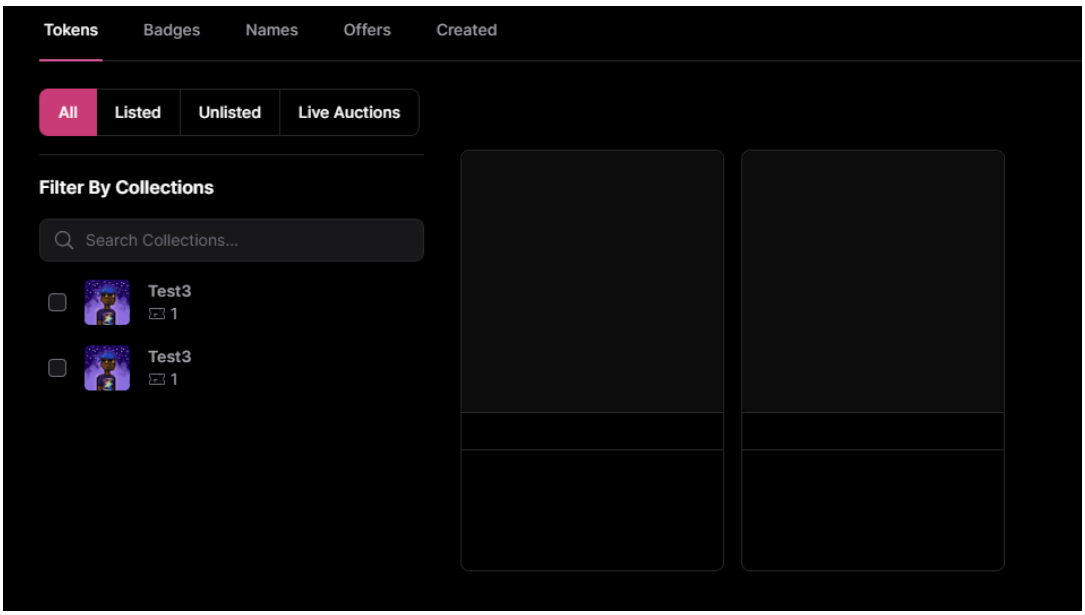
Daljnjim istraživanjem nisu pronađene mreže koje zadovoljavaju specifikacije definirane u projektu te imaju dovoljno opširnu dokumentaciju i dovoljno *online* resursa kako bi se uspješno implementirala željena logika.

Cardano

Implementacija logike za Cardano nije uspješno provedena. Pri rješavanju problema pokušana su dva pristupa: pokretanje lokalnog čvora te korištenje aplikacijsko programskog sučelja za komuniciranje s blockchain mrežom.

Pokretanje lokalnog čvora zahtjeva sinkroniziranje čvora koje može potrajati cijeli dan. Testirano je više konfiguracija opisanih u dokumentaciji [68]. U svim slučajevima sinkronizacija se zaglavila. Nije se ispisala nikakva greška nego se napredak sinkronizacije zaustavio. Mogući razlog za ovo je nedovoljna količina resursa [69]. Iz navedenih razloga nije bilo moguće pokretanje lokalnog čvora te korištenje *cardano-cli* naredbenog retka za interakciju s mrežom.

Drugi pristup je bilo korištenje Tangocrypto aplikacijsko programskog sučelja za kreiranje i upravljanje bedževima. Potrebno je napravi korisnički račun te generirati



Slika 3.15 Prikaze bedževa kreiranih na Stargaze mreži

API ključ za autentificiranje HTTP zahtjeva. Pokušano je generiranje bedža na način opisan u dokumentaciji [70]. Pri kreiranju se otvara prozor za slanje transakcije kojim se finalizira kreiranje bedža. Pritiskom na tipku za potvrdu transakcije pojavljuje se *undefined* greška. Na temelju greške nije bilo moguće utvrditi gdje je nastao problem. Dodatno, sustav za potvrdu transakcije je dio Tangocrypto API-a te nije moguće izmijeniti kod. Iz navedenih razloga pristup baziran na korištenju API-a je odbačen.

Poglavlje 4

Zaključak

Cilj ovog rada bio je implementirati mrežno korisničko sučelje za upravljanje studentskim bedževima na više različitih blockchain mreža. Operacije nad bedževima koje su bile definirane su dohvat, kreiranje te slanje bedževa.

Implementirano je korisničko sučelje za izvođenje navedenih operacija te za komunikaciju s blockchain mrežama preko mrežnih novčanika. Dodatno, implementirano je aplikacijsko programsko sučelje koje je zaduženo za autentifikaciju korisnika te pohranu metapodataka bedževa. Slike i metapodaci su pohranjeni na IPFS te na lokalni poslužitelj. Time se smanjuje broj zahtjeva na IPFS te osigurava redundantnost podataka.

Uspješno su implementirani Ethereum, Algorand i Near. Korisničko sučelje omogućuje korisniku brzo kreiranje te slanje bedževa za navedene blockchain mreže. Zbog već opisanih problema Cardano i Cosmos(Stargaze) nisu uspješno implementirani te predstavljaju potencijalno područje za buduće proširenje projekta. U slučaju Cardana, moguće je da bi novi API ili jači hardver uspješno riješili probleme koji su se pojavili. Za Cosmos je moguće da će se kreirati nova blockchain mreža koja više odgovara problematici ovog projekta.

Dodatni način za proširenje projekta je podrška za kreiranje i podizanje više različitih pametnih ugovora. Navedeni pristup bi osigurao podršku za veći broj korisnika te omogućio da svaki korisnički račun bude vezan uz vlastiti pametni ugovor.

Preporučeno je i dovršenje sustava za upravljanje dostupnim mrežama te koris-

nicima. Time bi se osiguralo da sustav podržava veći broj korisnika te bi pružio administratorima bolju kontrolu nad sustavom.

Dodatno, dizajn sustava podržava dodavanje novih blockchain mreža uz relativno malo promjena te je povećanje broja podržanih mreža moguće u budućnosti.

Bibliografija

[1] Algorand. , s Interneta, https://medium.com/coinmonks/algorand-4018a42025dd 2023.10.11.

[2] Ethereum's energy expenditure. , s Interneta, https://ethereum.org/en/energy-consumption/ 2023.05.02.

[3] Thresholded Proof Of Stake. , s Interneta, https://medium.com/nearprotocol/thresholded-proof-of-stake-67b74e616a92 2023.10.11.

[4] What is Cardano Settlement Layer Computation Layers? , s Interneta, https://www.cardanesia.com/post/what-is-cardano-settlement-layer-computation-layers 2023.10.11.

[5] What is Cosmos? , s Interneta, https://v1.cosmos.network/intro 2023.05.15.

[6] What is Blockchain. , s Interneta, https://www.synopsys.com/glossary/what-is-blockchain.html 2022.03.12.

[7] What is blockchain technology? . , s Interneta, https://www.ibm.com/topics/blockchain 2022.05.12.

[8] Cryptography Hash functions. , s Interneta, https://www.tutorialspoint.com/cryptography/cryptography_hash_functions.htm 2022.10.12.

[9] Hash Functions. , s Interneta, https://cryptobook.nakov.com/cryptographic-hash-functions 2022.15.12.

[10] What Are Blockchain Layers? , s Interneta, https://zebpay.com/blog/what-is-blockchain-layer-0-1-2-and-3 2023.4.01.

[11] Blockchain Architecture Layers: A Comprehensive Guide. , s Interneta, https://hacken.io/discover/blockchain-architecture-layers/#:~:text=A%20blockchain%20needs%205%20main,%2C%20and%20Hardware%2FInfrastructure%20Layer. 2023.4.01.

[12] What Are Consensus Mechanisms in Blockchain and Cryptocurrency? , s Interneta, https://www.investopedia.com/terms/c/consensus-mechanism-cryptocurrency.asp#:~:text=A%20consensus%20mechanism%20is%20any,the%20most%20prevalent%20consensus%20mechanisms. 2023.10.01.

[13] How To Pick The Best Consensus Algorithm For Blockchain? , s Interneta, https://www.blockchain-council.org/blockchain/how-to-pick-the-best-consensus-algorithm-for-blockchain/ 2023.10.01.

[14] Features of Blockchain. , s Interneta, https://www.geeksforgeeks.org/features-of-blockchain/ 2023.10.01.

[15] Features of Blockchain. , s Interneta, https://www.educative.io/answers/what-are-the-characteristics-of-blockchain 2023.10.01.

[16] Bitcoin: A Peer-to-Peer Electronic Cash System. , s Interneta, https://bitcoin.org/bitcoin.pdf 2023.20.01.

[17] An Analysis of the Current State of the Blockchain Industry and Its Future Outlook. , s Interneta, https://www.linkedin.com/pulse/analysis-current-state-blockchain-industry-its-future-outlook#:~:text=According%20to%20a%20recent%20report,at%20a%20CAGR%20of%2066.2%25. 2023.01.02.

[18] What are smart contracts on blockchain? , s Interneta, https://www.ibm.com/topics/smart-contracts 2022.15.12.

[19] What is a smart contract, and how does it work? , s Interneta, https://cointelegraph.com/learn/what-are-smart-contracts-a-beginners-guide-to-automated-agreements 2022.15.12.

[20] Solidity. , s Interneta, https://soliditylang.org/ 2022.15.12.

[21] Smart contracts market overview. , s Interneta, https://www.marketresearchfuture.com/reports/smart-contracts-market-4588 2022.15.12.

[22] A Complete Guide to Smart Contracts and Their Use Cases. , s Interneta, https://blog.bingx.com/insights/bingx-insights/a-complete-guide-to-smart-contracts-and-their-use-cases/ 2023.10.11.

[23] Non-Fungible Token (NFT): What It Means and How It Works. , s Interneta, https://www.investopedia.com/non-fungible-tokens-nft-5115211 2022.20.12.

[24] Understanding Fungible Non-Fungible tokens. , s Interneta, https://blockchainsimplified.com/blog/understanding-fungible-non-fungible-tokens/ 2023.10.11.

[25] CryptoKitties. , s Interneta, https://www.cryptokitties.co/ 2023.14.11.

[26] What Is an NFT Smart Contract? , s Interneta, https://hedera.com/learning/smart-contracts/nft-smart-contract 2022.20.12.

[27] What Is An NFT? Non-Fungible Tokens Explained. , s Interneta, https://www.forbes.com/advisor/investing/cryptocurrency/nft-non-fungible-token/ 2022.20.12.

[28] 6 Popular NFT Use Cases Across Industries. , s Interneta, https://artlabs.ai/blog/6-popular-nft-use-cases-across-industries 2023.10.11.

[29] FRACTIONAL NFTS. , s Interneta, https://www.leewayhertz.com/fractional-nft/ 2022.20.12.

[30] NFTs in Education: 10 Ways to Implement In Education Industry in 2023. , s Interneta, https://www.jetlearn.com/blog/nfts-in-education 2023.02.11.

[31] Algorand's Commitment to Sustainable Blockchain. , s Interneta, https://algorand.com/about/sustainability 2023.04.05.

[32] Ecological Impact of Blockchain. , s Interneta, https://insideecology.com/2023/04/17/ecological-impact-of-blockchain/#:~:text=Environmental%20Issues%20with%20Blockchain&text=According%20to%20a%20report%20by,the%20issue%20of%20climate%20change. 2023.04.12.

[33] Algorand consensus. , s Interneta, https://developer.algorand.org/docs/get-details/algorand_consensus/ 2023.04.07.

[34] Algorand Standard Assets (ASAs). , s Interneta, https://developer.algorand.org/docs/get-details/asa/ 2023.04.10.

[35] Ethereum Whitepaper. , s Interneta, https://ethereum.org/en/whitepaper/ 2023.05.02.

[36] Smart Contract Platforms: Past, Present and Future. , s Interneta, https://www.coindesk.com/layer2/2022/10/14/smart-contract-platforms-past-present-and-future/#:~:text=Eighteen%20years%20after%20Szabo's%20musings,smart%20contract%20platform%2C%20was%20launched. 2023.05.02.

[37] What is ether(eth)? , s Interneta, https://ethereum.org/en/eth/ 2023.05.02.

[38] ERC-721 NON-FUNGIBLE TOKEN STANDARD. , s Interneta, https://ethereum.org/en/developers/docs/standards/tokens/erc-721/ 2023.05.02.

[39] DECENTRALIZED APPLICATIONS (DAPPS). , s Interneta, https://ethereum.org/en/dapps/ 2023.05.02.

[40] Centralized vs Decentralized Applications. , s Interneta, https://www.geeksforgeeks.org/centralized-vs-decentralized-applications/ 2023.10.11.

[41] What is NEAR?r. , s Interneta, https://docs.near.org/concepts/basics/protocol 2023.05.28.

[42] What is a layer 1 blockchain? , s Interneta, https://www.bitstamp.net/learn/blockchain/what-is-a-layer-1-blockchain/ 2023.05.29.

[43] Economics in a Sharded Blockchain. , s Interneta, https://near.org/blog/thresholded-proof-of-stake 2023.05.29.

[44] Economics in a Sharded Blockchain. , s Interneta, https://near.org/papers/economics-in-sharded-blockchain 2023.05.29.

[45] Discover Cardano. , s Interneta, https://cardano.org/discover-cardano/ 2023.05.20.

[46] Cardano (ADA): What It Is, How It Differs From Bitcoin. , s Interneta, https://www.investopedia.com/cardano-definition-4683961 2023.05.15.

[47] What makes Cardano so special? , s Interneta, https://www.linkedin.com/pulse/what-makes-cardano-so-special-bitoasis-technologies-fze 2023.05.15.

[48] What's So Special About Cardano And Why Is It Currently The Top 7 Cryptocurrency By Market Cap? , s Interneta, https://thevrsoldier.com/whats-so-special-about-cardano-and-why-is-it-currently-the-top-7-cryptocurrency-by-marke
#:~:text=One%20of%20the%20key%20features,used%20by%20other%
20popular%20cryptocurrencies. 2023.05.20.

[49] Build on the Interchain. , s Interneta, https://cosmos.network/ 2023.05.15.

[50] Welcome to the Interchain. , s Interneta, https://cosmos.network/intro/ 2023.05.15.

[51] Cosmos Hub. , s Interneta, https://hub.cosmos.network/main/hub-overview/overview.html 2023.05.15.

[52] Cosmos SDK. , s Interneta, https://docs.cosmos.network/main/intro/overview 2023.05.15.

[53] What is IBC? , s Interneta, https://tutorials.cosmos.network/academy/3-ibc/1-what-is-ibc.html 2023.05.15.

[54] Laravel. , s Interneta, https://laravel.com/ 2023.07.16.

[55] Composer. , s Interneta, https://getcomposer.org/ 2023.02.11.

[56] Artisan console.

[57] SQLite. , s Interneta, https://www.sqlite.org/index.html 2023.02.11.

[58] PostgreSQL. , s Interneta, https://www.postgresql.org/ 2023.02.11.

[59] Breeze. , s Interneta, https://github.com/laravel/breeze 2023.02.11.

[60] Laravel permission. , s Interneta, https://spatie.be/docs/laravel-permission/v6/introduction 2023.02.11.

[61] Database: Migrations. , s Interneta, https://laravel.com/docs/10.x/migrations 2023.07.16.

[62] Database: Seeding. , s Interneta, https://laravel.com/docs/10.x/seeding 2023.07.16.

[63] Eloquent: Getting Started. , s Interneta, https://laravel.com/docs/10.x/eloquent 2023.07.16.

[64] Controllers. , s Interneta, https://laravel.com/docs/10.x/controllers 2023.07.16.

[65] Vue. , s Interneta, https://vuejs.org/ 2023.02.11.

[66] Vuetify. , s Interneta, https://vuetifyjs.com/en/ 2023.02.11.

[67] Launching an NFT project. , s Interneta, https://docs.stargaze.zone/creators/readme 2023.04.11.

[68] Installing cardano-node and cardano-cli from source. , s Interneta, https://developers.cardano.org/docs/get-started/installing-cardano-node/ 2023.04.11.

[69] How to run cardano-node. , s Interneta, https://developers.cardano.org/docs/get-started/running-cardano/ 2023.04.11.

[70] Tangocrypto SDK. , s Interneta, https://docs.tangocrypto.com/sdk/tangocrypto-sdk 2023.04.11.

[71] NEAR CLI. , s Interneta, https://docs.near.org/tools/near-cli#near-deploy 2023.08.11.

[72] HOW TO WRITE DEPLOY AN NFT (PART 1/3 OF NFT TUTORIAL SERIES). , s Interneta, https://ethereum.org/en/developers/tutorials/how-to-write-and-deploy-an-nft/#initialize-project 2023.08.11.

Sažetak

U ovom radu opisana je implementacija korisničkog sučelja za upravlja, kreiranje i slanje studentski bedževa (neizmjenjivi tokeni) na Ethereum, Algorand, Near, Cardano i Cosmos blockchain mrežama. Prvi dio rada sadrži objašnjenje teoretske podloge rada te detaljniji opis zahtjeva projekta. Nakon toga slijedi opis implementacije najbitnijih dijelova korisničkog sučelja i aplikacijskog programskog sučelja. Dodatno, opisana je logika za različite blockchain mreže. Na kraja nalazi se zaključak s rezultatima, problemima te daljnjim preporukama.

Ključne riječi **— blockchain, NFT, Ethereum, Algorand, Near, Cosmos, Cardano**

Abstract

In this paper the task was to implement a user interface for managing, creating and sending student badges (non fungible tokens) on Ethereum, Algorand, Near, Cardano and Cosmos blockchain networks. The first part of the paper describes the theoretical foundations of the project and it's goals. The next part describes the implementation of the user interface and the application programming interface (API). Additionally, it describes the logic for each of the implemented blockchain networks. At the end there is a conclusion with results, problems and future suggestions.

Ključne riječi **— *blockchain*, NFT, Ethereum, Algorand, Near, Cosmos, Cardano**

Prilog: Incijaliziranje API-a

Inicijaliziranje aplikacijsko programskog sučelja zahtjeva da su instalirani PHP, Composer menadžer paketa, Laravel te Artisan naredbeni redak. Pri implementiranju je korištena najnovija verzija navedenih paketa.

Priv korak pri postavljanju projekta je kloniranje projekta s git repozitorija:

```
https://github.com/Tehkkrapic/rbt-badges-back
```

Nakon kloniranja projekta potrebno je otvoriti naredbeni redak te postaviti se u direktorij u kojem se nalazi projekt. Instaliranje svih potrebnih paketa se pokreće s pomoću naredbe:

```
composer install
```

Nakon instaliranja paketa potrebno je kreirati *.env* datoteku te konfigurirati projekt. Preporučuje se kopiranje te izmjena *.env.example* datoteke. Također, u ovom koraku je bitno konfigurirati konekciju prema bazi podataka. Sljedeći korak je izvršavanje svih kreiranih migracija pomoću naredbe:

```
php artisan migrate
```

Nakon izvršavanja migracija struktura baze podataka je uspostavljena. Dodatno, u slučaju da se koristi SQLite kreira se datoteka koja predstavlja bazu podataka. Potrebno je populirati bazu s podacima te je to moguće s pomoću naredbe:

```
php artisan db:seed
```

Posljednji korak je pokretanje poslužitelja te se izvršava pomoću naredbe:

```
php artisan serve
```

Prilog: Incijaliziranje korisničkog sučelja

Inicijaliziranje korisničkog sučelja zahtjeva da su instalirani node.js te npm. Pri implementiranju je korištena najnovija verzija navedenih paketa. Dodatno, potrebno je postaviti ugovore za Ethereum i Near blockchain mreže.

Prvi korak pri postavljanju projekta je kloniranje projekta s git repozitorija:

```
https://github.com/Tehkkrapic/rbt-badges-front
```

Pametni ugovor za Near blockchain mrežu je kreiran preko near-cli naredbenog retka. Za njegovu instalaciju na Windows operacijskom sustavu potrebno je prov instalirati WSL (eng. *Windows Subsystem for Linux*). Nakon toga potrebno je dodati node.js i npm ako već nisu instalirani.

Idući korak je pokretanje WSL-a te instalacija near-cli naredbenog retka s pomoću naredbe:

```
npm install -g near-cli
```

Kako bi kreirali pametni ugovor potrebno je prijaviti se u NEAR novčanik s pomoću naredbe:

```
near login
```

Izvršenjem naredbe se otvara prozor za prijavu u novčanik. Nakon unosa imena računa, korisnik može kreirati pametni ugovor. Ugovor se kreira s pomoću naredbe:

```
near deploy --accountId ime.testnet --wasmFile out/example.wasm
```

Kako bi se naredba uspješno izvršila potrebno je upisati validni *id* korisničkog računa te kreirati praznu *.wasm* datoteku tako da odgovora putanji unesenoj u naredbi. Naredba vraća informacije o transakciji te o pametnom ugovoru. Identifikacijski broj ugovora i mreže se upisuje u *.env* datoteku korisničkog sučelja. Dodatne opcije za rad s near-cli su prikazane u dokumentaciji [71].

Pametni ugovor za Ethereum blockchain mrežu se kreira s pomoću ritehbadge direktorija uključenog u repozitorij projekta. Unutar navedenog direktorija potrebno je pokrenuti naredbu:

```
npm install
```

Nakon instaliranja potrebnih paketa potrebno je kreirati *.env* datoteku unutar navedenog direktorija. Preporučuje se kopiranje odgovarajuće *.env.example* datoteke te izmjena podataka unutar nje. Idući korak je kompilirati pametni ugovor s pomoću naredbe:

```
npx hardhat compile
```

Pametni ugovor se postavlja korištenjem naredbe:

```
npx hardhat --network sepolia run scripts/deploy.js
```

Naredba vraća adresu novokreiranog pametnog ugovora. Navedena adresa se mora kopirati u *.env* datoteku korisničkog sučelja. U slučaju da se mijenja ugovor, skripta ili mreža potrebno je prilagoditi naredbe te EthereumService kako bi pokazivao na artefakte pametnog ugovora (u slučaju promjene imena). Dodatne informacije se nalaze u dokumentaciji [72].

Nakon što se postave ugovori potrebno je pokrenuti korisničko sučelje. Potrebno je pokrenuti sljedeću naredbu za instaliranje potrebnih paketa:

```
npm install
```

Idući korak je kreiranje *.env* datoteke u korisničkom sučelju. Preporučuje se kopiranje te izmjena postojećeg primjera navedene datoteke. Posljedni korak je pokretanje korisničkog sučelja s pomoću naredbe:

```
npm run dev
```