IZRADA INTEGRIRANOG PROGRAMSKOG RJEŠENJA ZA AKTIVNOSTI PRODAJE ULAZNICA KORISTEĆI BLOCKCHAIN TEHNOLOGIJU

Ivanović, Filip

VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

IZRADA INTEGRIRANOGA PROGRAMSKOG RJEŠENJA ZA AKTIVNOSTI PRODAJE ULAZNICA KORISTEĆI BLOCKCHAIN TEHNOLOGIJU

Filip Ivanović

Zagreb, veljača 2023.

„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor, te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada“.

U Zagrebu, 28. 02. 2023.

Predgovor

Ovim putem želim se zahvaliti svima koji su mi pomogli u završetku preddiplomskog studija – svojoj obitelji, a posebno majci, baki i bratu, koji su mi uvijek bili podrška. Želim se zahvaliti svojoj partnerici i njezinoj obitelji koji su mi svojom toplinom i znanjem znatno olakšali studiranje. Također, i svojim kolegama, profesorima, asistentima i svim ostalim djelatnicima Visokog učilišta *Algebra,* bez kojih sve ovo ne bi bilo moguće.

Posebno zahvaljujem svome mentoru, Danielu Beleu, koji mi je svojom predanošću prenio svoje veliko programersko i još veće životno znanje i mudrost koji su u meni ponovno probudili onu dječju znatiželju prema istraživanju novoga i dali mi vjeru da se u životu sve može uz dovoljno truda i ljudskošću te da je jedini pravi uspjeh u životu biti dobar čovjek.

A Gradišćanska 24, HR-10000 Zagreb
T (01) 2222 182
E student@algebra.hr
www.algebra.hr

ALGEBRA

Temeljem članka 8. Pravilnika o završnom radu i završnom ispitu na preddiplomskom studiju Visokog učilišta Algebra sačinjena je ova

Potvrda o dodjeli završnog rada

kojom se potvrđuje da student Filip Ivanović, JMBAG 0067554915, OIB 59955867695 u šk. godini 2021./2022., studij: Primjenjeno računarstvo - Preddiplomski studij, smjer: Programsko inženjerstvo, od strane povjerenstva za provedbu završnog ispita, dana 24.01.2022. godine, ima odobrenu izradu završnog rada

s temom: **Izrada integriranog programskog rješenja za aktivnosti prodaje ulaznica koristeći blockchain tehnologiju**

i sažetkom rada:
Student će svojim radom prikazati napredno korištenje inovativnih tehnologija u kontekstu razvoja sustava za upravljanje aktivnostima prodaje ulaznica. Rezultat ovog završnog rada je sustav koji omogućuje interaktvino i napredno koristenje blockchain tehnologije u svrhu aktivnosti prodaje ulaznica.

Mentor je: Daniel Bele.

Odobrenjem završnog rada studentu je omogućen upis kolegija "Izrada završnog projekta/Praksa" te je sukladno članku 8. Pravilnika o završnom radu i završnom ispitu dužan najkasnije do početka nastave ljetnog semestra u sljedećoj školskoj godini, uspješno obraniti završni rad uspješnim polaganjem završnog ispita.

U protivnom student može zatražiti novog mentora/icu i temu te ponovo upisati kolegij "Izrada završnog projekta/Praksa" budući da rad koji nije predan i obranjen na završnom ispitu u roku određenom Pravilnikom završnom radu i završnom ispitu prestaje vrijediti. Izrada novog završnog rada se izvodi sukladno rokovima određenima za školsku godinu u kojoj je studentu određen novi mentor/ica i dodijeljen novi završni rad.

Potpis studenta:	Potpis mentora:	Potpis predsjednika povjerenstva:

Ova potvrda izdaje se u 4 (četiri) primjerka od kojih 3 (tri) idu kao prilog završnom radu.

Sažetak

Tema je ovog rada uočavanje neiskorištenih potencijala u aktivnosti prodaje ulaznica za događaje i prezentacija programskog rješenja pod nazivom NFT Tickets koje bi te potencijale moglo iskoristiti koristeći pametne ugovore na Ethereum blockchainu. Rezultati ovog rada su web-aplikacija i mobilna android aplikacija koje organizatorima događaja olakšavaju prodaju ulaznica te omogućuju zaradu u njihovoj preprodaji, a posjetiteljima događaja omogućuju sigurnu kupovinu ulaznica u prodaji i preprodaji te njihovu laganu preprodaju.

Ključne riječi: prodaja ulaznica, pametni ugovori, android, Ethereum, blockchain

Abstract

The topic of this paper is the observation of unused potentials in the activity of ticket sales for events and the presentation of a software solution called NFT Tickets that could utilize these potentials using smart contracts on the Ethereum blockchain. The results of this paper are a web application and a mobile Android application that makes it easier for event organizers to sell tickets and enables them to make money from their resale, and enables event visitors to safely buy tickets, both from sale and resale, and easily resell them.

Keywords: ticketing, smart contracts, Android, Ethereum, blockchain

Sadržaj

1. Uvod

Mnogi od nas vole posjećivati razne događaje, bilo nogometne utakmice, koncerte, predstave ili pak neke zanimljive konferencije. Za velik broj tih događaja ulaz nije besplatan, već je potrebno kupiti ulaznicu. To ponekad može biti problematično, posebice ako se radi o nekom popularnom događaju za koji je broj ulaznica ograničen. Ljudi se onda okreću tržištu preprodaje ulaznica u težnji da barem nekako dođu do ulaznice. To može biti riskantno jer se često dogodi da ulaznice kupljene tim putem budu falsificirane ili preprodane većem broju ljudi od kojih na sam događaj može ući samo onaj koji je prvi došao do ulaza.

Ideja je ovog projekta osmisliti i izraditi rješenje koje će postojeće načine kreiranja, prodaje i preprodaje ulaznica obogatiti dodatnim funkcionalnostima zahvaljujući blockchain tehnologiji koja se drastično razvila posljednjih desetak godina. Navedeni bi procesi bili dostupniji, sigurniji i direktniji zbog manjeg broja posrednika, kako organizatorima tako i posjetiteljima samih događaja. To će se postići implementacijom programskog rješenja koji se sastoji od web-aplikacije preko koje organizatori mogu kreirati, izmjenjivati i nadgledati informacije o svom događaju te pustiti ulaznice u prodaju, a posjetitelji mogu iste ulaznice kupiti, bilo direktno ili u preprodaji te ih i sami preprodati. Također, postojat će i android aplikacija preko koje će posjetitelji moći generirati QR kôd za ulaznicu koju posjeduju, koji se pri ulasku na događaj skenira, također preko android aplikacije, od strane organizatora događaja.

Kroz ovaj rad bit će detaljno opisan proces nastanka samoga programskog rješenja. Nakon uvodnog poglavlja, opisuje se problematika trenutačnog načina prodaje ulaznica. Nudi se prijedlog programskoga rješenja koje bi moglo riješiti velik broj problema u tom području. Također, dotiče se sličnih programskim rješenja i analizira njihove prednosti i nedostatke. Nadalje, opisuje se blockchain tehnologija i dio nje relevantan za ovaj projekt. Rad se potom bavi arhitekturom samoga programskog rješenja, od opisa rješenja do korištenih tehnologija, jezika i razvojnih okruženja, nakon čega slijedi opis implementacije rješenja i izrade različitih dijelova aplikacije. Naposljetku, analizira se uspješnost programskog rješenja i uviđaju potencijalna poboljšanja te mogućnosti nadogradnje rješenja.

2. Nedostaci klasičnog načina prodaje ulaznica i prijedlog programskog rješenja

Ovim poglavljem nastoje se objasniti problemi u aktivnostima prodaje ulaznica i opisuje se prijedlog programskog rješenja koji bi mogao ukloniti te probleme. Spominju se neka postojeća rješenja i uspoređuju s predloženim programskim rješenjem iz ovog rada.

2.1. Neiskorištene mogućnosti pri prodaji ulaznica

Proces organizacije događaja, na kojem će sudjelovati veći broj ljudi, može biti veoma kompleksan i stresan posao zbog niza koraka koji taj proces zahtijeva. Uzmimo samo donekle pojednostavljen primjer organizacije nekog koncerta. Potrebno je ugovoriti suradnju s izvođačem, unajmiti dvoranu, privući posjetitelje kroz reklame, plakate i razne druge načine. Važno je i isplanirati kakva će biti sama izvedba koncerta što se tiče ozvučenja, svjetlosnih efekata i sigurnosti. Tu je još dugi niz koraka koji dodatno otežavaju sam proces.

Ključno je, uglavnom preko nekog posrednika, organizirati i prodaju ulaznica od koje se financiraju mnogi drugi koraci, a koja je često glavni izvor prihoda samog koncerta. Posrednici uglavnom uzimaju postotak od prodane ulaznice kao kompenzaciju za svoje usluge, i to je nešto što organizator događaja teško može izbjeći jer posrednik često nudi cjelokupnu infrastrukturu – od prodaje do skeniranja ulaznica na ulazu u dvoranu. Razvoj sličnoga programskog rješenja za organizatora bi često bio preskup pa je ovisan o posrednikovu rješenju, što smanjuje njegovu ukupnu zaradu po koncertu, a i usporava proces organizacije događaja jer dogovori s posrednicima često budu na upit, a ne automatizirani preko nekog formulara na webu.

Mogućnost direktnije organizacije prodaje ulaznica na webu u formi NFT-ova organizatora uvelike bi ubrzala proces pripreme samog događaja i ponudila mu dodatni niz opcija koje do sada nije imao s tradicionalnim načinom prodaje ulaznica.

Često se dogodi da netko zbog osobnih razloga ne može prisustvovati događaju za koji je kupio ulaznicu pa ona ostaje neiskorištena jer proces njezine preprodaje može biti kompliciran pa mnogi ljudi od njega odustanu. To rezultira da novac utrošen u kupovinu ulaznice „propadne“. Također, često se dogodi da sve ulaznice za neki događaj budu rasprodane pa se ljudi, koji nisu došli do ulaznica, okreću stranicama za preprodaju. To može

biti rizično iz više razloga, ali najučestaliji je taj da na takvim stranicama zlonamjerni pojedinci mogu prodavati falsificirane ulaznice, što smanjuje povjerenje u takav način kupovine ulaznica stoga ih se često izbjegava. Upravo zato, riječ je o korisnom rješenju koje bi omogućilo preprodaju ulaznica tijekom koje bi obje strane, preprodavač i kupac, mogle vjerovati samom procesu.

Također, za organizatora bi bilo korisno da i on može na neki način profitirati od tržišta preprodaje ulaznica, a to s tradicionalnim načinom prodaje ulaznica u fizičkom i digitalnom obliku, bez korištenja blockchaina, nije moguće.

Isto tako, neke ulaznice nakon određenog vremena dobivaju kolekcionarsku vrijednost te zadrže vrijednost i nakon samog događaja. Ni od toga organizator događaja nema nikakvu korist iako to ne mora biti tako uz korištenje blockchain tehnologije.

Problem može predstavljati i fizička dostupnost ulaznica jer kada su ulaznice isključivo u fizičkoj prodaji, teško je doći do njih, posebice ako posjetitelji prisustvuju događaju u drugom gradu ili državi, a ponekad poteškoća može biti i prilikom internetske kupovine ulaznica jer kartična plaćanja znaju biti zaključana samo na zemlju u kojoj je događaj, koja ne mora uvijek biti ona u kojoj je izdana kartica koju koristi kupac, što dodatno komplicira proces kupnje ulaznice.

2.2. Prijedlog programskog rješenja za optimizaciju prodaje ulaznica

Ono što bi pomoglo organizatorima da lakše, uz što manje posrednika, organiziraju prodaju ulaznica za svoj događaj bila bi upravo prodaja ulaznica u formi NFT-ova putem NFT Tickets platforme, čiji se razvoj opisuje u ovom radu.

Za početak, proces bi bio vrlo jednostavan, a od organizatora bi se očekivalo samo da popuni formular na webu, plati transakcijsku naknadu blockchain mreže i ulaznice bi bile spremne za prodaju. Sa strane posjetitelja, proces je također jednostavan – pri kupnji ulaznice samo je potrebno uplatiti svotu koju je definirao organizator, uz neku manju naknadu blockchain mreže, i ulaznica je spremna za korištenje.

Organizator dobiva cijenu koju je definirao a da mu nitko uzima postotak od prodaje, što može povećati njegov profit, dok posjetitelj dobiva ulaznicu u formi NFT-a, koja se potom

pojavljuje u njegovu kripto novčaniku i u njegovu korisničkom pretincu na NFT Tickets aplikaciji.

Problemi koji nastaju u preprodaji ulaznica, navedeni u prethodnom podnaslovu, lako se rješavaju preko NFT Tickets platforme jer u sklopu nje bi se nalazila trgovina za preprodaju ulaznica koja bi vlasniku ulaznice omogućila da u nekoliko jednostavnih koraka stavi svoju ulaznicu na prodaju, a potencijalnom bi kupcu omogućila da zna da se radi o autentičnoj ulaznici i da neće biti prevaren u samoj transakciji. Također, od te kupoprodaje koristi može imati i organizator događaja ako je pri kreiranju događaja na stranici odlučio da želi postotak od svake preprodane ulaznice.

Ulaznice trajno ostaju u kripto novčaniku svoga vlasnika pa ih je moguće preprodati i nakon samog događaja, bilo putem NFT Tickets platforme ili bilo koje NFT burze, što može biti vrijedan dodatak u slučaju da ulaznice dobiju kolekcionarsku vrijednost. Od preprodaje ulaznica nakon događaja može profitirati i organizator događaja, ako to želi.

Zbog svoje decentraliziranosti, blockchain tehnologija nudi plaćanja s bilo kojeg mjesta gdje je dostupna internetska konekcija, što rješava problem dostupnosti kartičnog plaćanja u samo određenim državama ili regijama.

2.3. Usporedba sa sličnim programskim rješenjima

Na tržištu ima nekoliko programskih rješenja koja nude prodaju ulaznica u obliku NFT-ova, a neka od njih su SeatLabNFT, Oveit te NFT Tix. Iako svako od tih pojedinih rješenja ima svoje prednosti, zajednički nedostatak im je što ne nude direktno kreiranje događaja u svrhu prodaje ulaznica, već je takva usluga uvijek na upit. To usporava sam proces i čini taj dio procesa sličnim tradicionalnom načinu prodaje ulaznica. Uz to, mnoga rješenja ne nude trgovinu za preprodaju ulaznica, nego je kupac primoran svoje ulaznice, ako to želi, preprodati na NFT burzama.

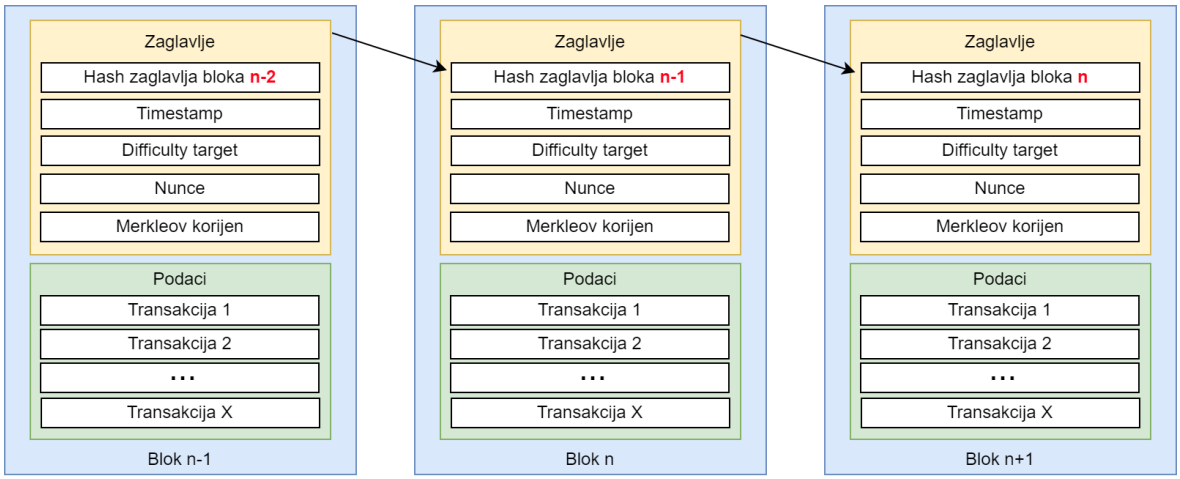
Što se tiče hrvatskog tržišta, slične aplikacije ne postoje, čime se otvara mogućnost da NFT Tickets bude pionir takvog tipa prodaje ulaznica na ovome geografskom području.

3. Tehnologija blockchaina

U ovom poglavlju nalazi se pregled blockchain tehnologije i nezamjenjivih tokena. Ne ulazi se previše u samu dubinu tehnologije, nego se teži objasniti dio relevantan za temu ovoga završnog rada.

3.1. Općenito o tehnologiji i njezinoj primjeni

Blockchain je decentralizirana, distribuirana baza podataka koja sadrži listu zapisa smještenih u blokove. Ovi blokovi, konceptualno prikazani na slici (Slika 3.1), nastaju jedan za drugim, a sastoje se od zaglavlja (engl. *header*) i podataka (engl. *data*). Zaglavlje sadrži kriptografski *hash* zaglavlja prethodnog bloka, vremensku oznaku nastanka bloka (engl. *timestamp*), težinu kreiranja bloka (engl. *difficulty target*), unikatni broj naziva *nunce* te Merkleov korijen (engl. *root*) koji je *hash* nastao hashiranjem svih transakcija koje se nalaze unutar podataka bloka [1]. Hashiranje je proces pretvaranja proizvoljnog ulaznog podatka u fiksnu duljinu naziva „hash vrijednost“ putem „hash funkcije“ te se svakim prolaskom kroz „hash funkciju“ istog ulaznog podatka dobiva jednaka „hash vrijednost“ [2].



Slika 3.1 Koncept blockchain blokova

Težnja blockchaina je izbjeći centraliziranost tradicionalnih baza podataka u kojima administracija nad bazom ovisi o centralnom entitetu koji njome upravlja. Također, cilj je postići distribuiranost podataka smještanjem podataka na velik broj izvora, što smanjuje ovisnost o pojedinom izvoru podatka.

Iako su se neki slični koncepti spominjali već 80-ih i 90-ih godina 20. stoljeća, prvi decentralizirani blockchain pojavio se 2009. godine i naziva se Bitcoin blockchain. Kreiran je od strane Satoshi Nakamota, za kojeg se na zna je li osoba ili grupa ljudi, koji je principe rada blockchaina objavio 2008. godine u dokumentu pod nazivom „Bitcoin whitepaper“.

Da bi postigao decentraliziranost i distribuiranost, blockchain se sastoji od niza čvorova (engl. *nodes*) koji čine tzv. mrežu. Jedni od najbitnijih sudionika u toj mreži nazivaju se rudari (engl. *miners*). Koristeći svoje resurse – struju i sklopovlje (engl. *hardware*), oni rješavaju kriptografske algoritme koji su zaslužni za provođenje transakcija na blockchainu, za što su nagrađeni kripto valutama. Kripto valute su nativne digitalne valute blockchaina kojem pripadaju. Rudari, koji sudjeluju u Bitcoin blockchainu, nagrađeni su Bitcoinom – nativnom kripto valutom Bitcoin blockchaina.

Uz Bitcoin blockchain tu je još mnogo blockchaina, a najpopularniji je, ujedno onaj koji se koristi u ovom radu, Ethereum blockchain čija je nativna kripto valuta ether. Za razliku od Bitcoin blockchaina, koji služi isključivo za slanje i primanje Bitcoina, Ethereum blockchain nudi jednu važnu dodatnu funkcionalnost – pametne ugovore (engl. *smart contracts*).

Pametni ugovori su programi na blockchainu koji se izvršavaju samo kada su ispunjeni određeni uvjeti. Ideja iza njih je pružiti dodatne funkcionalnosti samom blockchainu i omogućiti da za provedbu određenih odluka nije potrebno centralno tijelo koje određuje jesu li uvjeti ispunjeni ili ne, nego taj sud sam donosi pametni ugovor na temelju podataka koji se nalaze na blockchainu. Tako, npr. pametnim ugovorom možemo napraviti veoma učinkovito grupno financiranje (engl. *crowdfunding*), definirajući u pametnom ugovoru željeni iznos i krajnje vrijeme prikupljanja sredstava. Korisnici mogu putem svog kripto novčanika (engl. *crypto wallet*) uplaćivati kripto valute na taj pametni ugovor do određenog vremena. Kada dođe krajnje vrijeme prikupljana sredstava, pametni ugovor provjerava je li prikupljen željeni iznos. Ako jest, financiranje je uspješno i kripto valute se mogu, npr. automatski poslati na kripto novčanik organizatora grupnog financiranja, a ako nije, onda pametni ugovor može, npr. poslati sve kripto valute natrag na kripto novčanike sudionika u grupnom financiranju. Mogućnosti su mnogobrojne, a jedna od najčešćih primjena pametnih ugovora su tokeni.

Tokeni predstavljaju reprezentaciju *nečega* na blockchainu. Ovisno o vrsti tokena, to *nešto* najčešće je novac ili slika, ali može biti i druga stvar, kao npr. video ili glazba. Dijelimo ih u dvije vrste – zamjenjivi (engl. *fungible*) i nezamjenjivi (engl. *non-fungible*). Zamjenjivi tokeni su oni koji se mogu zamijeniti jedan za drugi bez gubitka vrijednosti, npr. prilikom

razmjene deset eura između dvoje korisnika vrijednost koju posjeduju na kraju te razmjene ostaje ista. Kod nezamjenjivih tokena to nije slučaj, što će biti objašnjeno u nastavku.

U kontekstu novca važno je razlikovati kripto valute od tokena. Tokeni su digitalna valuta čije se ime, simbol i podatak o svoti u novčaniku spremaju unutar „token ugovora“, koji je zapravo pametni ugovor koji upravlja tim tokenom. To je različito od kripto valute koja je nativna valuta nekog blockchaina jer kripto valutom upravlja sam blockchain i ona je sredstvo plaćanja svih transakcija unutar nekog blockchaina, neovisno radi li se o transakcijama prijenosa vlasništva nekog tokena, npr. slanju USDT tokena na neki novčanik, ili slanju same kripto valute na novčanik, npr. slanju ethera s jednog novčanika na drugi. U oba slučaja, troškovi transakcije na Ethereum blockchainu plaćaju se u etheru.

S vremenom je zbog težnje za standardizacijom došlo do standarda pod nazivom ERC-20 koji programerima olakšava interakciju s tom vrstom tokena pa se oni često nazivaju ERC-20 tokeni.

3.2. Nezamjenjivi tokeni

Nezamjenjivi tokeni razlikuju se od zamjenjivih po tome što prilikom razmjene dvaju nezamjenjivih tokena između dvoje korisnika vrijednost koju posjeduju na kraju razmjene nije ista kao prije razmjene jer se vrijednost nezamjenjivih tokena ne može precizno odrediti. Tako, npr. prilikom razmjene umjetničkih slika između dvoje korisnika, vrijednost koju posjeduju nakon te razmjene nije ista kao i prije razmjene jer se radi o dvije različite slike.

Nezamjenjivi tokeni popularno se zovu NFT-ovi. Slično kao i s ERC-20 tokenima, NFT-ovima upravlja pametni ugovor koji definira kako će pojedini NFT-ovi biti nazvani, kreirani, korišteni i prema potrebi – izbrisani. Popularno se kaže da su pojedini NFT-ovi dio neke kolekcije, što zapravo znači da su dio svog NFT pametnog ugovora.

Slično kao kod ERC-20 tokena, postojala je težnja za standardizacijom pa je tako izdan ERC-721 standard koji definira što je NFT. Stoga, ponekad se umjesto naziva NFT-ovi koristi naziv ERC-721 tokeni.

Najčešće se koriste za spremanje poveznica na metapodatke tako da svaki pojedini NFT unutar kolekcije može imati različite metapodatke. Ovi metapodaci gotovo uvijek uključuju dodatne poveznice na neku sliku na internetu. Kasnije web-stranice, koje prikazuju NFT-ove, učitaju tu sliku pa često dolazi do zablude da su NFT-ovi sami po sebi slike.

Ono što ih čini pogodnima za spremanje digitalnog vlasništva upravo je to što su na Ethereum blockchainu sve transakcije javne pa se može jasno utvrditi vlasništvo nad pojedinim NFT-jem. Isto tako, lako se može utvrditi autentičnost svih NFT-ova u kolekciji jer svi imaju jednaku javnu adresu, kao i njihova pripadajuća kolekcija, te je glavna razlikovna karakteristika, koja je ujedno i unikatna, njihov identifikacijski broj, poznat kao „token ID“.

Također, treba naglasiti jednu specifičnost kod preprodaje samih NFT-ova. Kreator NFT kolekcije ima mogućnost dobivanja tantijeme (engl. *royalties*) od svake transakcije preprodaje NFT-a [3]. Iznos tantijeme u postocima može se prema želji definirati prilikom kreiranja NFT-ja, što omogućava ERC-2981 protokol. Npr. ako se definira da tantijema bude 20%, a netko preproda NFT iz kreatorove kolekcije za 1 ether, on će dobiti 0,2 ethera od te transakcije, a prodavatelj 0,8 ethera. Jedino što ga može u tome spriječiti je to što nisu sve burze NFT-ova implementirale taj protokol, ali unutar Ethereum zajednice postoji težnja za kažnjavanjem burza koje su to izbjegle tako da je postotak burzi koje su ga implementirale svakim danom sve veći.

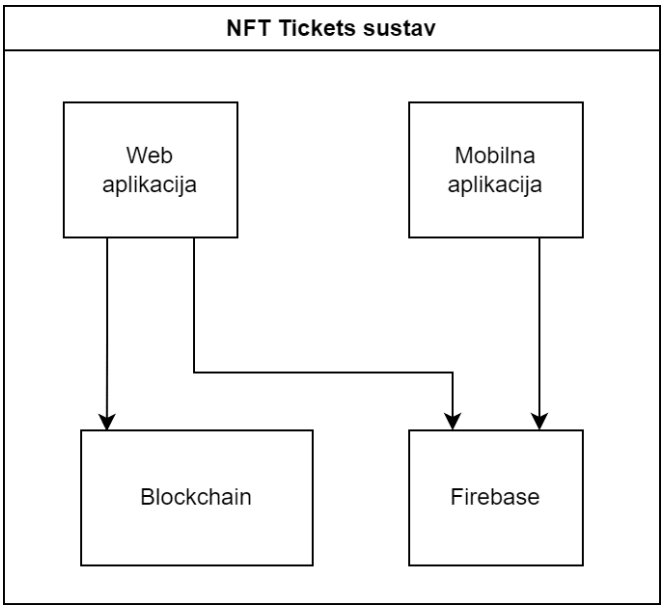
Navedene značajke čine NFT-ove pogodne za aktivnosti prodaje i preprodaje ulaznica jer se na jednostavan način može utvrditi autentičnost same ulaznice, što je ključno kod problema preprodaje krivotvorenih ili već iskorištenih ulaznica. Također, može se na siguran način provesti sama transakcija prodaje i preprodaje ulaznice jer blockchain svojom infrastrukturom garantira da obje strane prilikom kupoprodaje dobiju dogovorenu uslugu. I mogućnost dobivanja tantijeme od svake preprodaje je nešto što bi moglo mnoge organizatore događaja privući da za prodaju ulaznica koriste baš NFT-ove.

4. Arhitektura programskog rješenja

Ovo poglavlje sadrži pregled arhitekture programskog rješenja pri čemu se ne ulazi u duboko u implementacijske detalje. Arhitektura podrazumijeva način organiziranja komponenti programskog rješenja te odnosa i komunikacije među tim komponentama.

4.1. Skica i opis programskog rješenja

Programsko rješenje je osmišljeno kako bi organizatorima događaja i posjetiteljima omogućilo jednostavnu prodaju i kupovinu te preprodaju ulaznica za događaje, a to se ostvaruje kroz međusobnu interakciju više komponenti programskog rješenja. Na sljedećoj slici (Slika 4.1) prikazan je uvid u to kako je programsko rješenje idejno zamišljeno.



Slika 4.1 Model strukture NFT Tickets programskog rješenja

Arhitektura ovoga programskog rješenja sastoji se od niza komponenti. Što se tiče aplikacija, tu su web-aplikacija i mobilna aplikacija. Obje aplikacije vrše interakciju s komponentom Firebasea tako da joj šalju podatke i primaju podatke od nje, dok samo web-aplikacija vrši interakciju s komponentom blockchaina.

Web-aplikacija je glavna komponenta programskog rješenja. Zadužena je za kreiranje pametnih ugovora na Ethereum blockchainu preko kojih se kasnije kreiraju NFT-ovi koji predstavljaju ulaznice. Zadužena je i za prodaju te preprodaju ulaznica potencijalnim posjetiteljima događaja. Većina njezinih funkcionalnosti svodi se na pozivanje funkcija na

pametnom ugovoru koje potom vraćaju određene podatke ili ih zapisuju na blockchain uz pomoć MetaMask novčanika koji se povezuje na nju, a zadužen je za potvrđivanje transakcija i plaćanje transakcijskih troškova pri zapisivanju na blockchain.

Ostatak njezinih funkcionalnosti vezan je uglavnom uz dodavanje omiljenih događaja za pojedinu Ethereum adresu te spremanje adresa pametnih ugovora koji su postavljeni (engl. *deployed*) na blockchain putem NFT Tickets aplikacije u svrhu prikaza isključivo onih NFT kolekcija koje su kreirane putem nje. To se radi putem Firebase platforme. Također, putem Firebase platforme bilježi se koja je ulaznica skenirana na ulazu u sam događaj kako bi se znalo koja je iskorištena.

Mobilna je aplikacija zadužena isključivo za generiranje QR kôdova od strane vlasnika ulaznice i za skeniranje QR kôdova od strane organizatora događaja, a te podatke bilježi putem Firebase platforme.

Što se tiče pametnih ugovora na blockchainu, pametni ugovor koji je vezan uz preprodaju ulaznica postavljen je na blockchain prilikom razvoja aplikacije i kontrolira postavljanje ulaznica u preprodaju te samo izvršenje preprodaje tih ulaznica. Također, zadužen je i za povlačenje ulaznica iz preprodaje. Pametni ugovori vezani uz prodaju ulaznica za događaj kreiraju se putem web-aplikacije od strane organizatora događaja i njihova je zadaća držati podatke o događaju, kao što su broj dostupnih ulaznica, vrijeme događaja, naziv događaja, cijena ulaznice i slično.

Interakcija s pametnim ugovorima vrši se putem Ethers.js biblioteke unutar web-aplikacije.

4.2. Korištene tehnologije, jezici i razvojna okruženja

Za izradu programskog rješenja korišteno je niz tehnologija, jezika i razvojnih okruženja. Od tehnologija treba naglasiti blockchain putem kojeg je izveden velik dio poslužiteljsko-podatkovnog sloja. Ostatak tog sloja izveden je putem Firebase platforme. Što se tiče klijentskog dijela, web-aplikacija je izvedena putem React okruženja, a za izradu mobilne aplikacije korištenja je android tehnologija. U nastavku će navedene tehnologije biti detaljnije opisane.

4.2.1. Klijentski sloj

Klijentski sloj čine web i mobilna aplikacija. Unutar klijentskog sloja nalazi se vizualni, tj. prezentacijski dio aplikacije, te dio poslovne logike aplikacije koji je uglavnom vezan uz dohvaćanje i slanje podataka poslovno-poslužiteljskom sloju. Web-aplikacija napisana je u React okruženju, koristeći *Visual Studio Code*. Programski jezik korišten za pisanje web-aplikacije je TypeScript koji je zapravo inačica JavaScript programskog jezika s nekim dodatnim funkcionalnostima od kojih je najistaknutije tipiziranje, što znači da se sve varijable, funkcije i objekti mogu definirati tipom podataka. TypeScript se prevodi u JavaScript, što znači da je kompatibilan sa svim JavaScript datotekama. Također, koristi se Node.js, koji omogućava razvijanje JavaScript aplikacija izvan internetskog preglednika. Mobilna aplikacija napisana je u Kotlin programskom jeziku koristeći *Android Studio*.

React je JavaScript biblioteka za izradu korisničkih sučelja, koja je danas jedna od najpopularnijih i najtraženijih tehnologija u web-razvoju, razvijena je i održavana od strane Facebooka.

React je dizajniran da omogući razvoj aplikacija s dinamičkim sadržajem i interaktivnim funkcionalnostima. Glavna gradivna jedinica React aplikacije je komponenta (engl. *component*), koja predstavlja jedan dio aplikacije i ima vlastito stanje i prikaz. Komponente se mogu kombinirati i ponovno koristiti tijekom izrade aplikacije, što čini razvoj jednostavnijim i učinkovitijim [4].

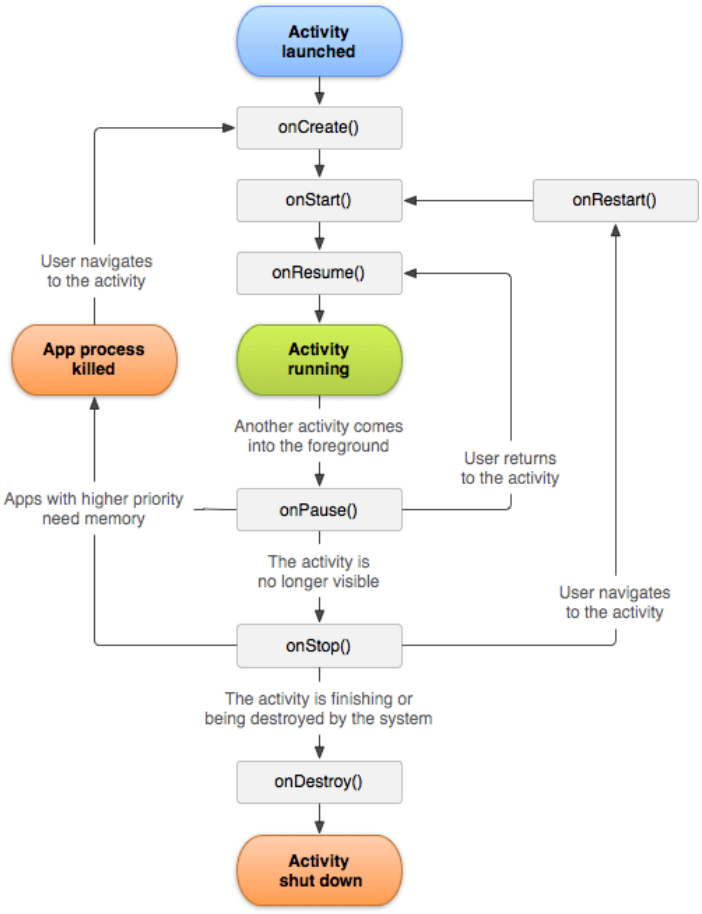
Veoma koristan dodatak u React ekosustavu je React Native koji omogućava izradu mobilnih aplikacija koristeći React, što znači da isti kôd može biti korišten za izradu aplikacija za mobilne uređaje i web. React je jednostavno integrirati s drugim tehnologijama i alatima, što čini razvoj aplikacija još efikasnijim. Postoji mnogo biblioteka i dodataka koji se mogu koristiti s Reactom, a u ovom radu od tih biblioteka će se istaknuti Ethers.js, koja je biblioteka za rad s Ethereum mrežom.

Ethers.js omogućuje lagano korištenje Ethereum funkcionalnosti u web-aplikacijama, kao što su potpisivanje transakcija, dohvaćanje podataka s blockchaina i interakcija s pametnim ugovorima [5]. Mogu se koristiti s različitim Ethereum klijentima, kao što je npr. Alchemy koji se koristi u ovom programskom rješenju. Ethereum klijenti predstavljaju čvor na Ethereum mreži, a zahvaljujući njima nije potrebno na svom lokalnom računalu imati cijelu bazu podataka Ethereum blockchaina koja iznosi nekoliko stotina gigabajta. Biblioteka

također podržava funkcije sigurnosti, kao što su enkripcija privatnih ključeva i jednostavnu integraciju s postojećim Ethereum alatima i bibliotekama.

Android aplikacija je software koji je namijenjen pokretanju i izvršavanju na android uređajima, a sastoji se od različitih dijelova koji funkcioniraju zajedno kako bi stvorili jedinstveno korisničko sučelje i funkcionalnosti. Ključne komponente uključuju aktivnosti, fragmente, servise, opskrbljivače sadržaja i prijamnike.

Aktivnosti su glavne komponente android aplikacije koje predstavljaju interaktivne prozore na kojima se iscrtava korisničko sučelje. Tijekom životnog ciklusa aktivnosti pozivaju se metode onCreate(), onStart(), onResume(), onPause(), onStop() i onDestroy(), kao što je vidljivo na slici (Slika 4.2). OnCreate() se poziva kada se aktivnost prvi put instancira, a onStart() i onResume()pozivaju se kada se aktivnost prikazuje korisniku. OnPause()poziva se kada aktivnost gubi fokus, a onStop() kada se aktivnost potpuno zaustavlja. OnDestroy()poziva se kada se aktivnost uništava.



Slika 4.2 Metode unutar životnog ciklusa aktivnosti [6]

Fragmenti su dijelovi sučelja koji se najčešće koriste u svrhu smanjivanja broja aktivnosti, što smanjuje opterećenje procesora i pridonosi uštedi baterije kod mobilnih uređaja koji koriste android operativni sustav. Mogu se koristiti za prikaz različitih dijelova sučelja u različitim situacijama, kao npr. prikaz različitog sučelja na tabletima i telefonima. Servisi su komponente android aplikacije koje se izvode u pozadini i ne prikazuju korisniku. Koriste se za obavljanje zadataka koji se ne moraju prikazivati korisniku, poput pozadinskog preuzimanja podataka s interneta. Opskrbljivači sadržaja su komponente android aplikacije koje omogućuju pristup podacima poput baze podataka. Koriste se za dohvaćanje podataka za prikaz korisniku. Prijamnici su komponente android aplikacije koje se koriste za primanje obavijesti i događaja poput poziva ili poruka. Manifest je datoteka u android aplikaciji koja definira komponente aplikacije i njihove značajke, poput dozvola i ovisnosti o drugim komponentama [7].

Programski jezici za pisanje android aplikacija su Java i Kotlin. U ovom radu se za razvoj android aplikacije koristi Kotlin, čije su odlike jednostavnost, sigurnost i interoperabilnost s Java kôdom. Kvalitete Kotlina prepoznao je i Google koji ga je od 2017. godine proglasio službenim jezikom za razvoj aplikacija android operacijskog sustava, čiji su vlasnik.

4.2.2. Poslužiteljsko-podatkovni sloj

Poslužiteljsko-podatkovni sloj čine Ethereum blockchain i Firebase. Iako rade na drukčiji način, konceptualno nude slične funkcionalnosti u kontekstu rada NFT Tickets programskog rješenja. Ethereum blockchain i Firebase služe kao baza podataka i API u jednome, uz to da Ethereum blockchain uz pomoć pametnih ugovora još u sebi sadrži dio programske logike za kreiranje, slanje i preprodaju NFT-ova.

O blockchain tehnologiji i NFT-ovima bilo je riječi u trećem poglavlju, ali ukratko, ona omogućava decentraliziranu evidenciju transakcija i podataka, a što znači da više nije potrebna centralna instanca za potvrđivanje transakcija.

MetaMask je jedan od najpopularnijih kripto novčanika, a najčešće se koristi u obliku ekstenzije za web-preglednik koja omogućava korištenje aplikacija na Ethereum blockchainu. Glavni benefit korištenja MetaMaska je mogućnost kreiranja i potpisivanja transakcija na Ethereum blockchainu te čuvanje kripto valuta i tokena, kao što su na primjer NFT-ovi [8]. Upravo zbog svih navedenih značajki, koristi se u ovom programskom rješenju.

Važno je spomenuti i JSON-RPC, a riječ je o protokolu za komunikaciju između aplikacija i blockchain mreže. Omogućava pozivanje funkcija na blockchainu putem HTTP zahtjeva i primanje odgovora u JSON formatu preko pružatelja usluge, već spomenutog Alchemyja. Ako se želi izbjeći korištenje pružatelja usluge, potrebno je na lokalnom računalu pokrenuti čvor koji će biti dio Ethereum mreže pa se JSON-RPC funkcije pozivaju direktno na tom čvoru. Tako se omogućava jednostavna izgradnja aplikacija koje se integriraju s blockchainom.

Pametni ugovori korišteni u ovom programskom rješenju napisani su u *Remix* razvojnom okruženju, koristeći Solidity programski jezik koji je objektno orijentirani jezik kreiran u svrhu izrade pametnih ugovora za Ethereum blockchain.

Firebase je platforma za razvoj aplikacija koja se temelji na konceptu „Backend-as-a-Service“, što je model usluge koji pruža gotova rješenja za upravljanje i izgradnju poslužiteljsko-podatkovnog sloja aplikacije, a razvijena je od strane Googlea. Firebase uključuje bazu podataka, API, autentifikaciju korisnika, slanje obavijesti i druge funkcionalnosti koje se obično razvijaju izvan samoga klijentskog sloja aplikacije [9]. Također, količina postavki koje treba konfigurirati svodi se na minimum, što omogućava brže lansiranje aplikacije na tržište.

Firebase koristi NoSQL bazu podataka za pohranu podataka aplikacije. Vrsta NoSQL baze podataka, koja se koristi u ovom programskom rješenju, omogućuje pohranu i upravljanje podacima u obliku nestrukturiranih dokumenata. Na taj se način brže i jednostavnije čitaju i ažuriraju podaci, bez potrebe za definiranjem stupaca i redaka unaprijed, kao što je slučaj kod relacijskih baza podataka. Firebase također omogućuje korištenje sinkronizacije u realnom vremenu i brze izmjene na bazi podataka, što ga čini idealnim rješenjem za mobilne i web-aplikacije koje zahtijevaju brzu i efikasnu bazu podataka. Bazi podataka se pristupa prilično jednostavno i brzo putem API-ja kojeg također generira Firebase.

5. Implementacija programskog rješenja

U ovom poglavlju opisuje se implementacija NFT Tickets programskog rješenja. Rješenje je kategorizirano u manje cjeline pa je detaljnije opisana izrada svih cjelina zajedno sa slikama ekrana i kôdom koji prate njihovu izradu.

5.1. Izrada poslužiteljskog dijela

Poslužiteljski dio programskog rješenja sastoji se od pametnih ugovora postavljenih na blockchain te Firebase projekta koji predstavlja bazu podataka i API u jednome. Glavni je zadatak poslužiteljskog dijela skladištenje podataka i izvedba dijela programske logike prema aplikaciji. U nastavku se opisuje izrada pametnih ugovora i Firebase projekta.

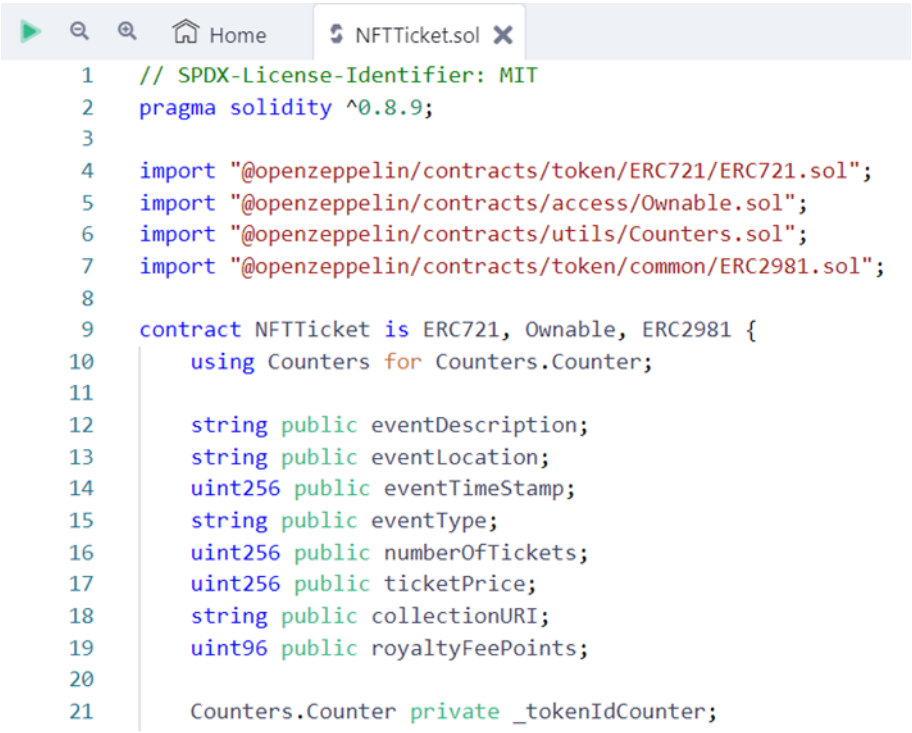
5.1.1. Izrada pametnih ugovora

Pametni ugovori izvedeni su koristeći Solidity programski jezik unutar *Remix* razvojnog okruženja i Goerli testni Ethereum blockchain na kojeg su pametni ugovori nakon razvoja postavljeni.

Glavna odlika Ethereum pametnih ugovora jest to što je njihov kôd nepromjenjiv (engl. *immutable*) [10]. Zbog toga je važno zaštititi kôd od što je više moguće sigurnosnih prijetnji i rizika jer ako se kasnije otkrije neka prijetnja, jedina mogućnost koja preostaje jest kreirati novi pametni ugovor i prebaciti sva sredstva na njega, što može biti veliki trošak.

U ovom programskom rješenju koriste se dva pametna ugovora. Prvi pametni ugovor, „NFTTicket“, zadužen je za prodaju ulaznica u formi NFT-ova, dok je drugi, „Marketplace“, zadužen za preprodaju ulaznica. Pametni ugovor „NFTTicket“ postavlja se na blockchain prilikom kreiranja događaja od strane organizatora, dok se pametni ugovor „Marketplace“ postavlja na blockchain prilikom razvoja aplikacije.

Prilikom izrade „NFTTicket“ pametnog ugovora korištena je „OpenZeppelin“ biblioteka koja nudi niz već napisanih pametnih ugovora namijenjenih nasljeđivanju prilikom razvoja pametnih ugovora. Na slici (Slika 5.1) prikazan je dio kôda „NFTTicket“ pametnog ugovora. Nakon definiranja po kojim je kriterijima licenciran kôd u prvoj liniji, navodi se verzija Solidity programskog jezika kojom je napisan pametni ugovor.



Slika 5.1 Dio kôda „NFTTicket“ pametnog ugovora

Nakon ključne riječi „contract“, kojom se definira da se radi o pametnom ugovoru te nazivu pametnog ugovora, koristi se ključna riječ „is“ koja definira da pametni ugovor nasljeđuje „ERC-721“, „Ownable“ i „ERC-2981“ pametne ugovore iz OpenZeppelin projekta. „ERC-721“ pametni ugovor definira što pametni ugovor mora implementirati da bi se smatrao NFT pametnim ugovorom. „Ownable“ pametni ugovor daje pametnom ugovoru mogućnost da uz pomoć „onlyOwner“ modifikatora na metodama omogući njihovo pozivanje samo putem kripto novčanika kreatora toga pametnog ugovora. „ERC-2981“ pametni ugovor definira što pametni ugovor mora implementirati da bi podržavao mogućnost da od svake preprodaje ulaznica organizator događaja dobije željeni postotak.

Pametni ugovor definira nekoliko javnih varijabli zaduženih za čuvanje opisa, lokacije, vremena početka, kategorije, broja ulaznica, cijene i slike događaja, a također koristi biblioteku „Counters“ za generiranje jedinstvenih token ID-jeva.

Konstruktor se poziva prilikom postavljanja pametnog ugovora na blockchain, a kao parametre prima vrijednosti koje postavlja kao vrijednosti prethodno spomenutih javnih varijabli. Također, prima i željeni postotak koji kreator pametnog ugovora želi imati od svake preprodaje ulaznice, što je vidljivo na prikazu kôda konstruktora (Slika 5.2). Ujedno, vrijednost postavlja kroz „_setDefaultRoyalty“ funkciju koja uz taj podatak prima i „msg.sender“, što predstavlja adresu kripto novčanika pozivatelja funkcije, u ovom slučaju

kreatora pametnog ugovora. Zbog toga što pametni ugovor nasljeđuje ERC-721, pametni ugovor poziva se također i njegov konstruktor.

```
constructor(    infinite gas 3179600 gas
    string memory _eventName,
    string memory _eventSymbol,
    string memory _eventDescription,
    string memory _eventLocation,
    uint256 _eventTimeStamp,
    string memory _eventType,
    uint256 _numberOfTickets,
    uint256 _ticketPrice,
    string memory _collectionURI,
    uint96 _royaltyFeePoints
) ERC721(_eventName, _eventSymbol) {
    eventDescription = _eventDescription;
    eventLocation = _eventLocation;
    eventTimeStamp = _eventTimeStamp;
    eventType = _eventType;
    numberOfTickets = _numberOfTickets;
    ticketPrice = _ticketPrice;
    collectionURI = _collectionURI;
    royaltyFeePoints = _royaltyFeePoints;
    _setDefaultRoyalty(msg.sender, _royaltyFeePoints);
}
```

Slika 5.2 Konstruktor „NFTTicket“ pametnog ugovora

Za javne varijable Solidity programski jezik automatski generira istoimenu funkciju za dohvat vrijednosti određene varijable, dok je funkciju za postavljanje neke vrijednosti potrebno ručno napisati. Funkcija (Kôd 5.1) služi za postavljanje vrijednosti javne varijable „eventDescription“ koja je zadužena za spremanje opisa događaja.

```
function setEventDescription(string memory _newDescription)
public onlyOwner {
        eventDescription = _newDescription;
}
```

Kôd 5.1 Funkcija za postavljanje opisa eventa

Funkcija zadužena za kreiranje same ulaznice, tj. ERC-721 tokena, jest funkcija „safeMint“ koja je prikazana u (Kôd 5.2).

```
    function safeMint() public payable {
        _tokenIdCounter.increment();
        uint256 tokenId = _tokenIdCounter.current();
        require(tokenId <= numberOfTickets,"Maximum number of
tickets are already minted");
        require( msg.value == ticketPrice, "Ether amount must
equal ticket price");
        _safeMint(msg.sender, tokenId);
    }
```

Kôd 5.2 Funkcija za kreiranje ERC-721 tokena

Funkcija je javna, što znači da je svatko može pozivati, a modifikatorom „payable“ označava se da ova funkcija prima ethere. Funkcija na početku povećava brojač ID-jeva tokena pa dohvaća novu vrijednost brojača i uspoređuje ga s maksimalnim dozvoljenim brojem ulaznica definiranim od strane organizatora događaja pomoću „require“ ključne koja provjerava je li uvjet ispunjen. Ako to nije slučaj, funkcija prestaje s izvođenjem. Također, ako je nova vrijednost brojača veća od najvećeg dostupnog broja ulaznica, funkcija prestaje s izvođenjem, a ako nije, provjerava se je li iznos ethera, koje je poslao korisnik, a koja je dostupna u „msg.value“ varijabli, jednak cijeni ulaznice. Ako jest, poziva se ERC-721 funkcija „_safeMint“ koja kreira novi NFT s ID-jem definiranim u „tokenId“ varijabli, a njegovo vlasništvo dodjeljuje pozivatelju funkcije.

„Marketplace“ pametni ugovor zadužen je za funkcionalnosti trgovine za preprodaju ulaznica. Kao što je vidljivo na slici (Slika 5.3), nakon informacija o licenciranju i korištenoj verziji Solidity programskog jezika prilikom izrade „Marketplace“ pametnog ugovora, uvezen je samo „NFTTicket“ pametni ugovor.

Marketplace.sol

```
1  // SPDX-License-Identifier: Unlicensed
2  pragma solidity ^0.8.9;
3
4  import "./NFTTicket.sol";
5
6  contract Marketplace {
7      struct Item {
8          uint256 itemId;
9          NFTTicket nft;
10         uint256 tokenId;
11         uint256 price;
12         address payable seller;
13         bool sold;
14     }
15
16     mapping(uint256 => Item) public items;
17     uint256 public itemCount;
```

Slika 5.3 Dio kôda „Marketplace“ pametnog ugovora

Nakon ključne riječi „contract“, kojom se definira da se radi o pametnom ugovoru te nazivu pametnog ugovora, definirana je struktura „Item“ koja u sebi čuva podatke o pojedinom oglasu, kao što su ID tog oglasa, adresa pametnog ugovora NFT-a koji se prodaje i njegov ID, cijena, adresa kripto novčanika prodavača te informacija je li ulaznica prodana.

Podaci o oglasima spremaju se unutar strukture podataka naziva „mapping“ koji omogućava mapiranje jedne vrijednosti na drugu, tj. ključa i vrijednosti, slično rječniku u drugim programskim jezicima. U ovom slučaju, ključ predstavlja ID pojedinog oglasa koji mora biti veći od nula, što je odlika podatkovnog tipa uint256, dok vrijednost predstavlja sam oglas koji je tipa „Item“. Javna varijabla „itemCount“ služi za praćenje broja oglasa postavljenih u trgovini za preprodaju.

Pametni ugovor sastoji se od tri funkcije zadužene za kreiranje oglasa, kupovinu ulaznica putem oglasa te vraćanje ulaznice u vlasništvo kreatora oglasa ako odustane od preprodaje.

Funkcija za kreiranje novog oglasa naziva „makeItem“, a prikazana je u nastavku (Kôd 5.3).

```
    function makeItem(
        NFTTicket _nft,
        uint256 _tokenId,
        uint256 _price
    ) external {
        require(_price > 0, "Price must be greater than
zero");
        itemCount++;
        _nft.transferFrom(msg.sender, address(this),
_tokenId);
        items[itemCount] = Item(
            itemCount,
            _nft,
            _tokenId,
            _price,
            payable(msg.sender),
            false
        );
    }
```

Kôd 5.3 Funkcija za kreiranje novog oglasa za preprodaju ERC-721 tokena

Ona kao argumente prima instancu „NFTTicket“ pametnog ugovora postavljenog na blockchain ID NFT-a, koji se prodaje, i željenu cijenu. Funkciju je moguće pozvati samo izvan pametnog ugovora, što definira modifikator „external“. Na početku se provjerava da je cijena veća od nula; kod pozitivnog odgovora, nastavlja se izvršavanje funkcije. Funkcija zatim povećava „itemCount“ u svrhu praćenja broja ukupnog broja kreiranih oglasa, kreira oglas i smješta ga kao vrijednost unutar „mappinga“ pod ključem „itemCount“.

U nastavku je prikazana funkcija „purchaseItem“ (Kôd 5.4), zadužena za kupovinu pojedine ulaznice u preprodaji, a koja kao parametar prima ID oglasa za ulaznicu koju korisnik želi kupiti. Funkcija prvo provjerava je li ID manji ili jednak broju dostupnih oglasa; u slučaju pozitivnog odgovora dohvaća željeni oglas i provjerava je li prodan; u slučaju negativnog odgovora provjerava je li iznos ethera koje je korisnik poslao jednak cijeni ulaznice u preprodaji i tek s pozitivnim odgovorom funkcija nastavlja izvođenje.

```
    function purchaseItem(uint256 _itemId) external payable {

        require(_itemId > 0 && _itemId <= itemCount, "Item
doesn't exist");
        Item storage item = items[_itemId];
        require(!item.sold, "item already sold");
        require( msg.value == item.price, "Not enough Ether
sent to buy a ticket" );

        (address feeReceiver, uint256 feeAmount) =
NFTTicket(items[_itemId].nft)
            .royaltyInfo(items[_itemId].tokenId,
items[_itemId].price);

        item.seller.transfer(item.price - feeAmount);
        payable(feeReceiver).transfer(feeAmount);
        item.sold = true;
        item.nft.transferFrom(address(this), msg.sender,
item.tokenId);
    }
```

Kôd 5.4 Funkcija za kupovinu ERC-721 tokena u preprodaji

Funkcija zatim dohvaća adresu kripto novčanika primatelja tantijeme kao i njen iznos u postocima. Zatim na kripto novčanik kreatora oglasa šalje iznos prodajne cijene oglasa umanjen za iznos tantijeme pa na adresu kripto novčanika primatelja tantijeme šalje iznos tantijeme. Naposljetku, funkcija označava oglas prodanim i šalje kupcu kupljenu ulaznicu u formi NFT-a.

Funkcija „delistItem“, koja je prikazana u nastavku (Kôd 5.5), kao parametar prima ID oglasa ulaznice koju kreator oglasa želi vratiti u svoje vlasništvo.

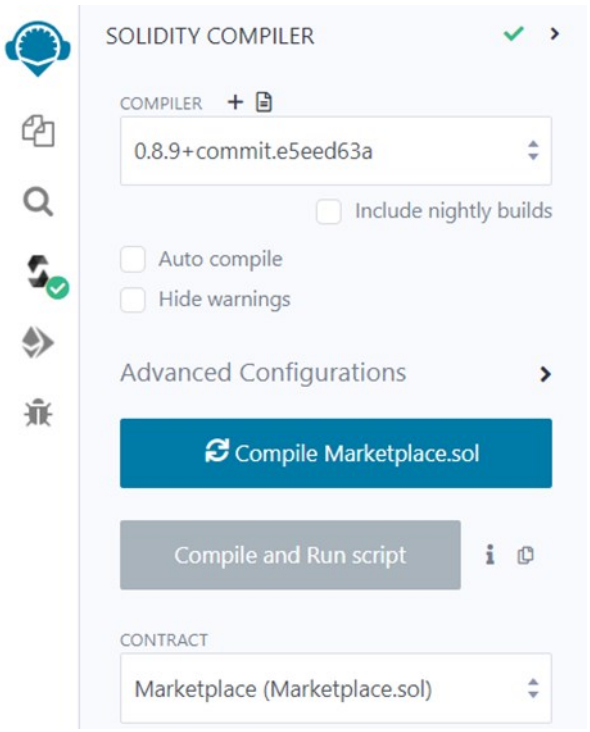
```
    function delistItem(uint256 _itemId) external {
        require(_itemId > 0 && _itemId <= itemCount, "Item
doesn't exist");
```

```
        Item storage item = items[_itemId];
        require(!item.sold, "Item already sold");
        require(msg.sender == item.seller, "Only seller can
delist item");

        item.nft.transferFrom(address(this), msg.sender,
item.tokenId);
        item.sold = true;
    }
```

Kôd 5.5 Funkcija za povlačenje ERC-721 tokena iz preprodaje

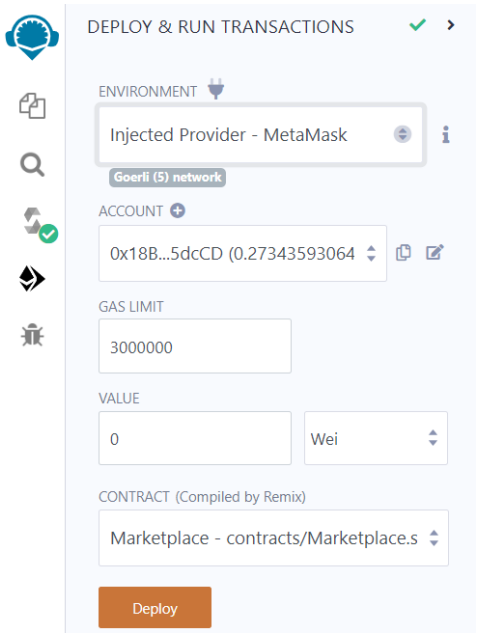
Funkcija također prvo provjerava je li ID manji ili jednak broju dostupnih oglasa, a u slučaju pozitivnog odgovora dohvaća željeni oglas i provjerava je li prodan; u slučaju negativnog odgovora provjerava je li pozivatelj funkcije ujedno i kreator oglasa pa ako je odgovor pozitivan, funkcija nastavlja s izvođenjem. U nastavku funkcija šalje ulaznicu u formi NFT-a natrag kreatoru oglasa, a oglas označava prodanim kako ne bi i dalje bio u prodaji.

Pametni ugovor „Marketplace“ potrebno je postaviti na Goerli testni Ethereum blockchain. Prvi korak u tome je prevođenje (engl. *compile*) pametnog ugovora iz Solidity programskog jezika u jezik koji koristi Ethereum virtualna mašina koja je, među ostalim, zadužena za postavljanje pametnih ugovora na blockchain. To se unutar *Remix* razvojnog okruženja postiže klikom na „Solidity Compiler“ izbornik i odabirom željenoga pametnog ugovora, odabirom verzije Solidity programskog jezika te klikom na gumb „Compile Marketplace.sol“, što je vidljivo na slici (Slika 5.4). Time se ujedno kreiraju i ABI te Bytecode koji se kasnije koriste u web-aplikaciji.



Slika 5.4 Izbornik za prevođenje Solidity kôda

Nakon prevođenja pametnog ugovora, potrebno ga je postaviti na Goerli testni Ethereum blockchain. Korisnik unutar *Remix* razvojnog okruženja to postiže klikom na „Deploy & Run Transactions“ izbornik i odabirom „Injected Provider“ opcije koja automatski dohvaća blockchain na koji je spojen MetaMask kripto novčanik, kao što je vidljivo na slici (Slika 5.5).



Slika 5.5 Izbornik za postavljanje pametnog ugovora na blockchain

U ovom slučaju to je Goerli testni Ethereum blockchain. Također, potrebno je odabrati pametni ugovor koji se postavlja na blockchain. Nakon odabira „Marketplace“ pametnog ugovora, potrebno je kliknuti gumb „Deploy“ i potvrditi transakciju na MetaMask ekstenziji, čime se kreira transakcija.

Nakon uspješnog izvršenja transakcije, pametni ugovor je postavljen na blockchain i dobiva svoju adresu koja je unikatna i razlikuje ga od svih drugih pametnih ugovora na blockchainu. Sada je moguće s njim komunicirati putem klijentskog dijela koristeći Ethers.js biblioteku.

5.1.2. Izrada i konfiguracija Firebase projekta

Baza podataka te API izvedeni su koristeći Firebase platformu. Budući da je Firebase usluga na principu „Backend-as-a-Service“, kreiranjem baze podataka automatski se kreira i API kojem možemo jednostavno pristupiti koristeći Firebase SDK.

Za izradu Firebase projekta, potrebno je samo imati Google račun. Prilikom kreiranja projekta korisnik unosi naziv projekta i odabire želi li koristiti Google Analytics u svom projektu koji, među ostalim, pruža mogućnosti testiranja i distribucije aplikacije. U ovom programskom rješenju nije korištena ta mogućnost. Nakon tog koraka, klikom na gumb kreira se Firebase projekt.

Nakon kreiranja projekta, korisnik ga treba dodati u svoju aplikaciju, bilo to android ili web-aplikacija. Prvo odabire u koji tip aplikacije ga želi dodati, a potom ispunjava neke osnovne informacije o aplikaciji, koje se razlikuju ovisno o tipu aplikacije, i odabire opciju registracija. Odabirom na tu opciju otvaraju mu se upute kako dodati Firebase SDK u svoj projekt, ovisno o platformi na kojoj ga koristi. Firebase SDK omogućava jednostavnu interakciju s Firebaseom pozivom funkcija unutar kôda

Dodavanje Firebase SDK-a u web-aplikaciju je jednostavno i sve što je potrebno je u terminalu unutar projekta u *Visual Studio Codeu* pokrenuti komandu `npm install firebase` koja dodaje sve potrebne datoteke u „node_modules“ direktorij. Nakon toga, potrebno je kopirati kôd iz uputa za uporabu Firebasea, koji je prikazan u zadnjem koraku kreiranja projekta, unutar „firebase.config.ts“ datoteke u *Visual Studio Codeu*, kao što je vidljivo na slici (Slika 5.6).

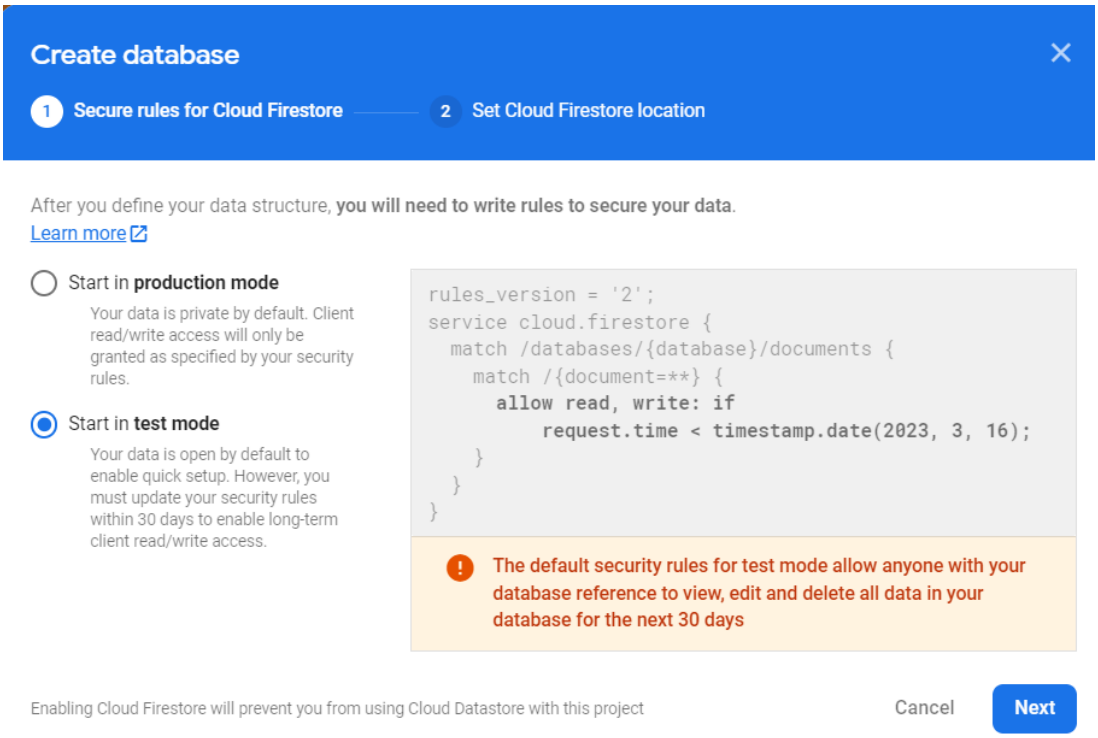
TS firebase.config.ts ●

src > TS firebase.config.ts > ...

```ts
// Import the functions you need from the SDKs you need
import { initializeApp } from 'firebase/app';
import { getFirestore } from 'firebase/firestore';

// Your web app's Firebase configuration
const firebaseConfig = {
  apiKey: 'AIzaSyDYvP3vGX2W68JfMVQD7HtctoTxJvSGc8E',
  authDomain: 'nft-tickets-4e184.firebaseapp.com',
  projectId: 'nft-tickets-4e184',
  storageBucket: 'nft-tickets-4e184.appspot.com',
  messagingSenderId: '992072079673',
  appId: '1:992072079673:web:08589171b90128fae1b0e8',
};

// Initialize Firebase
initializeApp(firebaseConfig);
export const db = getFirestore();
```

Slika 5.6 Kôd za konfiguraciju Firebase projekta

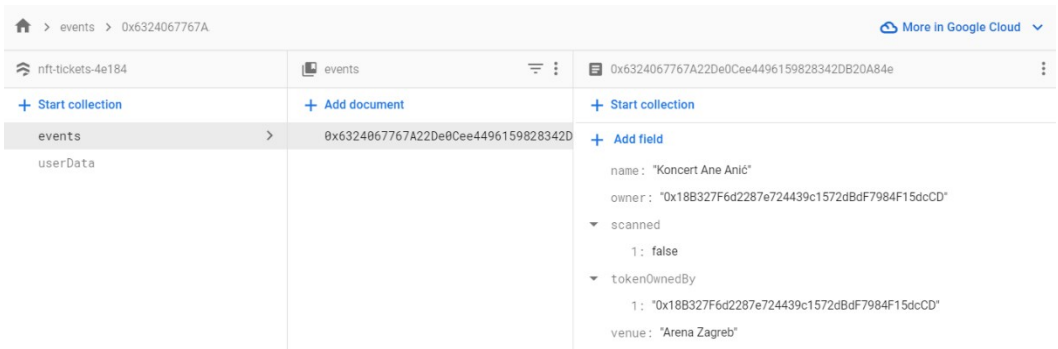
Nakon dodavanja Firebase SDK-a u aplikaciju, potrebno je kreirati bazu podataka koja se naziva Firestore. Klikom na gumb „Create Database“ unutar Firebase web-aplikacije otvara se prozor u kojem je potrebno odabrati između produkcijskog i testnog načina rada Firestore baze podataka, kao što je vidljivo na slici (Slika 5.7).



Slika 5.7 Prozor za odabir načina rada Firestore baze podatka

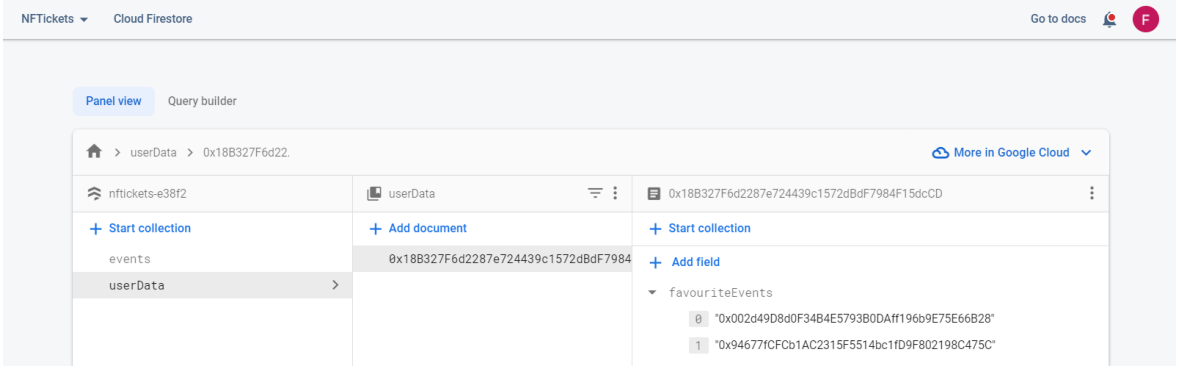
Glavna razlika između ta dva načina rada jest što produkcijski način rada korisnicima omogućava samo čitanje podatka po zadanim vrijednostima, dok testni način rada, koji je ujedno i odabran u ovome programskom rješenju, omogućava svima da čitaju, ažuriraju i brišu podatke do 30 dana nakon kreiranja aplikacije. Daljnji je korak odabir lokacije servera, za što je odabrana europska lokacija, nakon čega se otvara stranica za kreiranje baze podataka.

Klikom na gumb „Start collection“ otvara se prozor u koji je potrebno unijeti „Collection ID“, što predstavlja unikatno ime za kolekciju, a za to je odabran naziv „Events“ jer je cilj ove kolekcije sačuvati adrese samo onih pametnih ugovora događaja kreiranih putem NFT Tickets web-aplikacije u svrhu daljnjeg prikaza. Slična funkcionalnost mogla se postići kreiranjem još jednoga pametnog ugovora koji čuva te podatke, ali zbog troška unosa novih podataka na blockchainu, korištenje Firebasea je znatno jeftinije. Nakon toga, potrebno je kreirati prvi dokument u kolekciji. Dizajnom baze podataka odlučeno je da „Document ID“ bude adresa pametnog ugovora nekog događaja, ali pošto web-aplikacija nije još kreirana, taj podatak se na početku uzima se iz pametnog ugovora postavljenog na blockchain preko *Remix* razvojnog okruženja. Također, potrebno je dodati barem jedno polje unutar dokumenta pa je dizajnom baze odlučeno da to polje bude „owner“, što predstavlja adresu kripto novčanika kreatora toga pametnog ugovora koji je zapravo organizator događaja. Iako je istom podatku moguće pristupiti koristeći blockchain, ovakav je način mnogo brži prilikom iteracije kroz veću količinu kreiranih događaja u svrhu prikaza samo onih događaja koji je kreirao pojedini korisnik. Zbog težeg pristupa blockchain podacima unutar mobilne aplikacije, dodana su polja koja sadrže ime događaja, mjesto održavanja, adresu vlasnika pojedine ulaznice, kao i podatak o tome je li pojedina ulaznica skenirana. Nakon klika na gumb „Save“, dobivena je struktura kolekcije, dokumenta i polja prikazana na slici (Slika 5.8).



Slika 5.8 Struktura „events“ kolekcije unutar Firestorea

Druga kolekcija koja je dodana vezana je uz prikaz omiljenih događaja pojedinog korisnika, odnosno adrese kripto novčanika. Struktura kolekcije, dokumenta i njegova polja prikazana je na slici (Slika 5.9).



Slika 5.9 Struktura „userData“ kolekcije unutar Firestorea

Podatak „Collection ID“ naziva je „userData“, a „Document ID“ dokumenta unutar te kolekcije predstavlja adresa kripto novčanika pojedinog korisnika. Jedino polje unutar tog dokumenta naziva je „favouriteEvents“ i podatkovnog je tipa (engl. *array*). Elementi tog polja predstavljaju adrese pametnih ugovora omiljenih događaja pojedinog korisnika. Kreiranjem „events“ i „userData“ kolekcija završena je izrada baze podataka te ujedno i API-ja.

Nakon kreiranja baze podataka i API-ja, korištenje Firebasea unutar projekta je jednostavno pa je samo potrebno uvesti (engl. *import*) Firebase instancu iz „firebase.config.ts“ datoteke u TypeScript datoteku, točnije modul koji je gradivna jedinica Node.js projekata. U ovom slučaju, Firebase instanca je naziva „db“. Također, potrebno je uvesti željene Firestore funkcije, kao npr. funkciju za dohvaćanje reference na kolekciju naziva „collection“, funkciju za kreiranje upita naziva „query“ i funkciju za dohvaćanje dokumenata naziva „getDocs“, vidljive na slici (Slika 5.10).

```
1  import { collection, query, getDocs } from 'firebase/firestore';
2  import { db } from '../firebase.config';
```

Slika 5.10 Uvođenje Firebase instance i Firestore funkcija

Korištenjem pojedinih funkcija unutar Firebase SDK-a vrši se interakcija s bazom podataka, tj. čitanje, pisanje, ažuriranje i brisanje podataka iz nje.

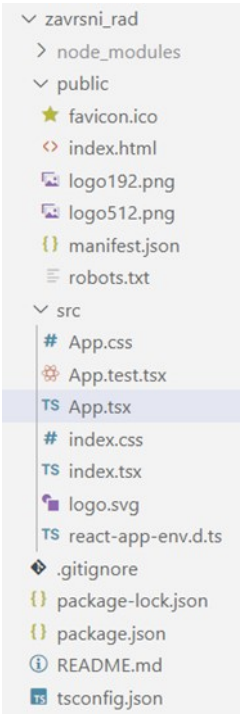
5.2. Izrada klijentskog dijela

Klijentski dio programskog rješenja predstavlja web-aplikacija i android mobilna aplikacija. Glavni je zadatak klijentskog dijela prikaz podataka prema korisniku. U nastavku se opisuje izrada tih dviju aplikacija.

5.2.1. Izrada web-aplikacije

Web-aplikacija izrađena je koristeći TypeScript programski jezik i React razvojni okvir unutar *Visual Studio Code* razvojnog okruženja.

Kreiranje React aplikacije unutar *Visual Studio Code* razvojnog okruženja je poprilično jednostavno, a sve što je potrebno je otvoriti terminal i upisati komadu `npx create-react-app ime_projekta --template typescript` te pričekati da se kreira aplikacija pod unesenim nazivom. Ime aplikacije u ovom slučaju je „zavrsni_rad“. Komanda „create-react-app“ uvelike ubrzava kreiranje aplikacije jer automatski kreira datoteke potrebne za njezino prvo pokretanje. Alternativa takvoj, ili sličnoj komandi, ručno je kreiranje i konfiguriranje potrebnih datoteka kako bi aplikacija bila spremna za prvo pokretanje i daljnji razvoj. Nakon izvršenja „create-react-app“ komande, na slici (Slika 5.11) možemo vidjeti novokreiranu strukturu aplikacije.



Slika 5.11 Struktura direktorija nakon izvršenja „create-react-app“ komande

Ona se ukratko sastoji od „node_modules“ direktorija u koji se spremaju sve biblioteke korištene prilikom izrade aplikacije. Direktorij pod nazivom „public“ sadrži „index.html“ datoteku koja je jedina HTML datoteka unutar aplikacije, a koja sadrži „div“ tag s atributom „id“ koji ima vrijednost „root“, kao što je vidljivo na slici (Slika 5.11).

index.html

zavrsni_rad > public > index.html > ...

```
1   <!DOCTYPE html>
2   <html lang="en">
3     <head>
4       <meta charset="utf-8" />
5       <meta name="viewport" content="width=device-width, initial-scale=1" />
6       <meta name="theme-color" content="#000000" />
7       <title>React App</title>
8     </head>
9     <body>
10      <noscript>You need to enable JavaScript to run this app.</noscript>
11      <div id="root"></div>
12    </body>
13  </html>
```

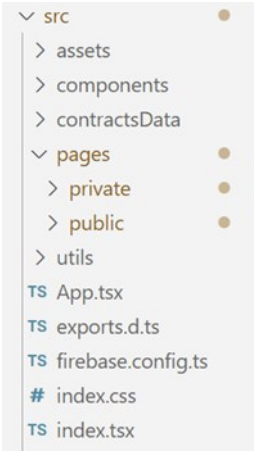
Slika 5.12. Datoteka „indeks.html“

React pri interakciji korisnika s aplikacijom ažurira sadržaj tog taga pa korisnik dobiva dojam da je učitana nova HTML datoteka, što nije slučaj, nego je samo sadržaj taga postao neki HTML kôd generiran kao povratna vrijednost TypeScript funkcije. Ta TypeScript funkcija uglavnom se nalazi unutar „App.tsx“ datoteke, a njezina je uporaba definirana unutar „index.ts“ datoteke u „src“ direktoriju. Web-aplikacije, koje funkcioniraju na ovakav način, nazivaju se jednostrane aplikacije (engl. *single page application*). React je zapravo zadužen da na optimalan način izmijeni sadržaj unutar sadržaja spomenutog „div“ taga. Datoteka „App.tsx“ je u ovom slučaju glavna TypeScript datoteka projekta unutar koje je funkcija unutar koje se potom pozivaju različite stranice (engl. *pages*) i komponente koje predstavljaju manje cjeline koje se mogu ponovno koristiti, što čini razvoj aplikacija u Reactu jednostavnim. Razlika između stranice i komponente s tehničke strane zapravo ne postoji, a ono što ih dijeli način je primjene. Stranica je namijenjena simuliranju HTML datoteke u klasičnom načinu izrade web-aplikacija kojoj se može pristupiti putem nekog URL-a, dok se komponente više koriste u svrhu prikazivanja upravo unutar tih stranica, kako bi bile gradivne jedinice od kojih je ta stranica izrađena. Komponente podržavaju ugnježđivanje jedne ili više njih unutar druge.

Od ostalih datoteka u strukturi React aplikacije treba spomenuti „package.json“ datoteku u kojoj su spremljeni nazivi i verzije svih biblioteka korištenih u aplikaciji. Koristi se ako primjerice kreirana aplikacija trenutno nije u uporabi i želi ako se uštedjeti prostor za pohranu na računalu. U tom je slučaju moguće izbrisati „node_modules“ direktorij koji često zauzima najviše prostora u cijeloj aplikaciji, a prilikom ponovnog korištenja aplikacije dovoljno je samo pokrenuti komandu „`npm install`“ u terminalu, gdje će se uz pomoć „package.json“ datoteke ponovno rekreirati „node_modules“ direktorij sa svim potrebnim bibliotekama u njemu.

Nakon uspješne izrade jednostavne predefinirane aplikacije uz pomoć „create-react-app“ naredbe, moguće ju je pokrenuti koristeći `npm start` komandu koja diže razvojni server na određenom portu i prikazuje kreiranu aplikaciju.

Nakon toga, odabrana je struktura „src“ direktorija, prikazana na slici (Slika 5.13), u kojem se odvija većina razvoja aplikacije. Direktorij „assets“ sadrži medijske datoteke korištene prilikom razvoja aplikacije, „components“ sadrži TypeScript datoteke koje sadrže prethodno objašnjene komponente, a „contractsData“ sadrži „json“ datoteke koje opisuju *sučelje* pametnih ugovora i njihov *bytecode,* što je ključno za interakciju s njima putem klijentskog dijela. Direktorij „pages“ sadrži već spomenute TypeScript datoteke, koje sadrže stranice, a „utils“ direktorij sadrži TypeScript datoteke koje sadrže pomoćne funkcije. U „src“ direktoriju još se nalaze već spomenute „App.tsx“ i „index.ts“ datoteke, kao i CSS datoteka te „firebase.config.ts“ datoteka u kojoj se nalazi kôd potreban da bi se Firebase mogao uspješno povezati s web-aplikacijom.



Slika 5.13 Struktura direktorija NFT Tickets aplikacije

Nakon izrade strukture „src“ direktorija, kreće razvoj vizualnog dijela NFT Tickets web-aplikacije. Razvoj vizualnog dijela obuhvaća kreiranje stranica i komponenti te njihovo

stiliziranje, uglavnom uz pomoć HTML tagova i CSS atributa. Kako bi razvoj bio jednostavniji, uz što manje pisanja CSS-a, korištena je biblioteka *Material UI* verzije 5. Njezinim dodavanjem u projekt na raspolaganju je velik broj već gotovih i vizualno atraktivnih React komponenti koje je jednostavno implementirati u projekt, što čini razvoj aplikacije znatno bržim.

Komponenta se najčešće definira unutar TypeScript datoteke, tj. Node modula, s ekstenzijom „tsx“, a riječ je zapravo o TypeScript funkciji koja vraća HTML kôd. Primjer strukture komponente je vidljiv u kôdu (Kôd 5.6).

```
function Primjer({ mnozenik }) {
  const [mnozitelj, setMnozitelj] = useState(5);

  useEffect(() => {
    setMnozitelj(6);
  }, []);

  const pomnoziSaMnoziteljem = (mnozenik) => {
    return mnozenik * mnozitelj;
  };

  return (
    <div>
      <h1>
        Množenje broja {mnozenik} sa {mnozitelj}
      </h1>
      <h3>Rezultat je {pomnoziSaMnoziteljem(mnozenik)}</h3>
    </div>
  );
}
```

Kôd 5.6 Primjer React komponente

Ono što komponenta mora imati jest naziv, u ovom slučaju „Primjer“, te „return“ izraz koji vraća HTML kôd. Komponenta ne mora imati varijable i funkcije. Također, ne mora imati svojstva (engl. *props*), ali i može, kao što je slučaj s komponentom „Primjer“ koja ima svojstvo „mnozenik“. U slučaju poziva neke funkcije ili ispisa vrijednosti varijable unutar HTML kôda koristimo vitičaste zagrade. Komponenta može, među ostalim, koristiti „useState“ i „useEffect“ kuke (engl. *hooks*). Kuka „useState“ uglavnom se koristi za spremanje nekog stanja unutar varijable, primjerice broja 5 unutar varijable „mnozitelj“, a

pri njenom deklariranju često se odmah deklarira i funkcija koja postavlja njenu vrijednost, u ovom slučaju „setMnozitelj“. Kuka „useEffect“ je korisna jer se uvijek izvrši prije prvog prikaza komponente pa je moguće definirati i druge događaje prilikom kojih će se izvršiti, npr. promjena vrijednosti neke varijable. Funkcija se koristi pri dohvaćanju podataka s API-ja prije prikazivanja podataka. U ovom slučaju varijabla „mnozitelj“ na početku ima vrijednost 5, ali „useEffect“ kuka odmah pri prikazu komponente mijenja njenu vrijednost na 6. Ako je komponentu potrebno prikazati, to se radi tako da se u drugoj komponenti ili stranici, u ovom slučaju „App“ komponenti, pozove na način prikazan u kôdu (Kôd 5.7).

```
import Primjer from './Navbar';

function App() {
  return (
    <>
      <Primjer mnozenik={10}/>
    </>
  );
}

export default App;
```

Kôd 5.7 Primjer poziva React komponente sa svojstvom

Nakon pokretanja aplikacije u pregledniku varijabla „mnozitelj“ poprima vrijednost 6 te je svojstvo „mnozenik“ pri pozivu komponente definirano kao 10, što rezultat množenja čini 60 i potvrđuje da useEffect djeluje prije prikaza komponente.

Ovakav način korištenja komponenti je zapravo temelj izgradnje velikog dijela NFT Tickets web-aplikacije i znatno se koristi u sljedećim koracima.

Prva komponenta koja je kreirana je „Navbar.tsx“ komponenta. Ona, kao i cijela web-aplikacija, izrađena je u težnji za responzivnim dizajnom koji će se prilagoditi veličini i orijentaciji zaslona korisnika. Vizualni dio komponente namijenjen ekranima visoke razlučivosti prikazan je na slici (Slika 5.14).

Slika 5.14 Prikaz „Navbar“ komponente na ekranima visoke razlučivosti

Veći je dio komponente izrađen koristeći upravo Material UI biblioteku. U nastavku (Kôd 5.8) je prikazana jedna Material UI komponenta zadužena za pisanje „NFT Tickets“ teksta unutar Navbar komponente.

```
<Typography
  variant='h6'
  noWrap
  component='a'
  sx={{
    mr: 2,
    display: { xs: 'none', md: 'flex' },
    fontFamily: 'monospace',
    fontWeight: 700,
    color: 'inherit',
    textDecoration: 'none',
  }}
>
  NFT Tickets
</Typography>
```

Kôd 5.8 Material UI komponenta za prikaz stiliziranog teksta

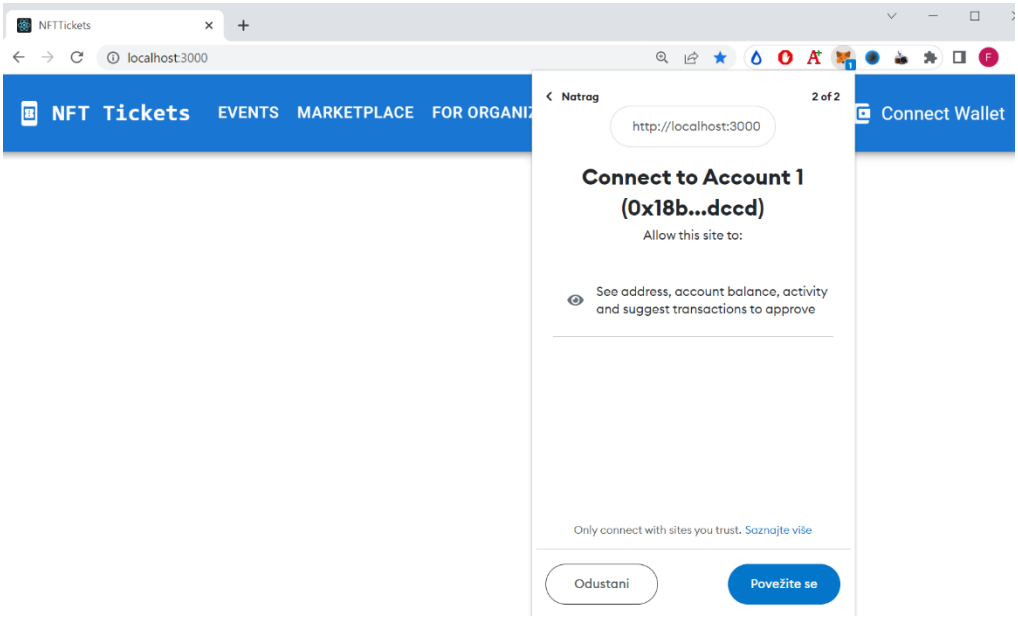
Sintaksa, koja je veoma slična standardnoj HTML sintaksi, tako da je jednostavno razumljiva.

Ono što razlikuje „Navbar“ komponentu od većine drugih je to što sadrži gumb „Connect Wallet“. Klikom na taj gumb poziva se „web3Handler“ funkcija prikazana na slici (Slika 5.15).

```
const web3Handler = async () => {
  let accounts = await window.ethereum.request({
    method: 'eth_requestAccounts',
  });
  setAccount(accounts[0]);

  window.ethereum.on('chainChanged', () => {
    window.location.reload();
  });

  window.ethereum.on('accountsChanged', async () => {
    setLoading(true);
    web3Handler();
  });
};
```

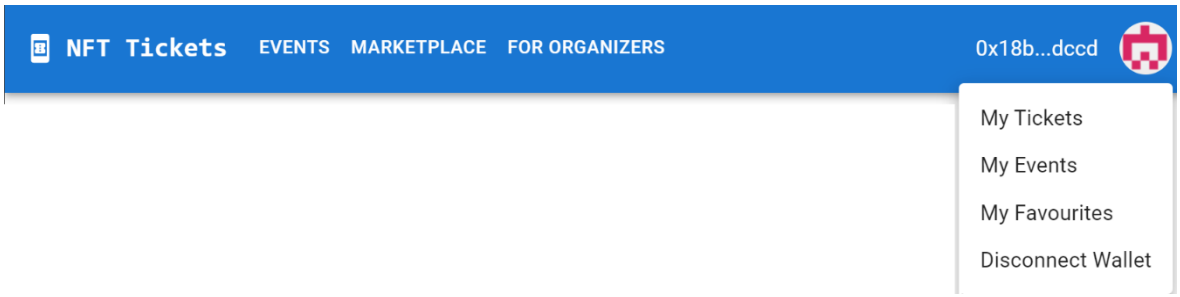
Slika 5.15 Funkcija za spajanje MetaMask kripto novčanika na NFT Tickets aplikaciju

Funkcija web3Handler zadužena je za povezivanje MetaMask kripto novčanika na web-aplikaciju. Prilikom prvog povezivanja na web-aplikaciju korisnik je dužan potvrditi da se uistinu i želi povezati, čime ga se želi zaštititi od zlonamjernih web-aplikacija. Taj korak prikazan je na slici (Slika 5.16).



Slika 5.16 Potvrda povezivanja MetaMask kripto novčanika na aplikaciju

Nakon što je korisnik povezao novčanik s web-aplikacijom, mijenja se izgled njegove „Navbar“ komponente i pojavljuje se skraćena verzija njegove Ethereum adrese uz sliku, što je najčešće korišteni način kako korisniku dati do znanja da je uspješno spojio novčanik s aplikacijom, kao što je prikazano na slici (Slika 5.17).



Slika 5.17 Komponenta „Navbar“ nakon povezivanja s MetaMask kripto novčanikom

Također, kada je korisnik spojen na aplikaciju, klikom na sliku pokraj njegove Ethereum adrese otvara se dodatni izbornik u kojem se nalaze funkcionalnosti kojima mogu pristupiti samo oni korisnici koji su spojili svoj kripto novčanik s aplikacijom.

Nakon izrade „Navbar“ komponente i njenog prikaza unutar „App“ komponente, sljedeći je korak omogućiti rad izbornika – uz pomoć biblioteke „react-router-dom“, stvarajući rute

preko kojih se putem URL-a može direktno pristupiti određenoj stranici. Najlakši način za ostvarivanje toga je unutar „App“ komponente definirati željene rute koristeći „Route“ tag koji ima atribute „element“ i „path“, kao što je prikazano na slici (Slika 5.18).

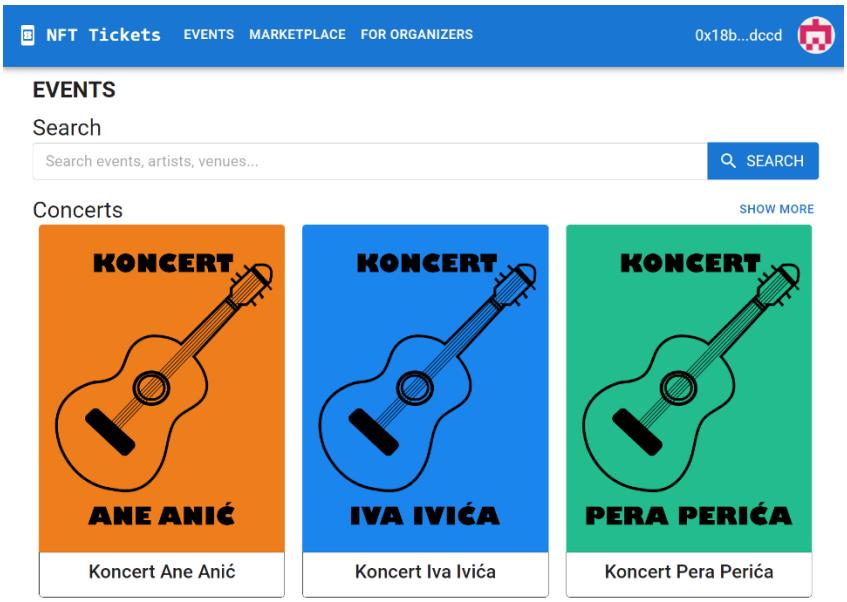
```
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import Events from './pages/public/Events';
import Marketplace from './pages/public/Marketplace';
import EventsOfType from './pages/public/EventsOfType';
import NFTMintDisplay from './pages/public/NFTMintDisplay';
import SearchResults from './pages/public/SearchResults';

function App() {
  return (
    <>
      <Router>
        <Routes>
          <Route path='/' element={<Events />} />
          <Route path='/search/:searchTerm' element={<SearchResults />} />
          <Route path='/events/:typeOfEvent' element={<EventsOfType />} />
          <Route
            path='/event/:collectionAddress'
            element={<NFTMintDisplay />}
          />
          <Route path='/marketplace' element={<Marketplace />} />
        </Routes>
      </Router>
    </>
  );
}

export default App;
```

Slika 5.18 Kôd za kreiranje ruta unutar „App“ komponente

Atribut „element“ definira koja će se stranica prikazati kada URL bude jednak vrijednosti „path“ atributa. Oko „Route“ taga potrebno je dodati „Router“ i „Routes“ tagove.

Nakon što su definirane rute, potrebno je samo unutar „Navbar“ komponente definirati koji gumb unutar izbornika vodi na koju rutu tako da se, npr. klikom na gumb „Marketplace“ otvara ruta „http://localhost:3000/marketplace“.

Daljnji razvoj aplikacije svodi se na kreiranje stranica, komponenti i njihov prikaz, što je veoma slično izradi „Navbar“ komponente. Na početnoj stranici, kao što je vidljivo na slici (Slika 5.19), prikazani su dostupni događaji poredani po kategorijama pa korisnik ima mogućnost pretrage određenog događaja ili prikaza svih događaja iz neke kategorije.



Slika 5.19 Početna stranica NFT Tickets web-aplikacije

Popis događaja je dohvaćen direktno s Ethereum testnog blockchaina pod nazivom Goerli, a u ovom se rješenju koristi jer je besplatan za razliku od glavnog Ethereum blockchaina. Dohvaćanje je postignuto uz pomoć Alchemya, koji je Ethereum klijent, i čija se uloga objašnjava u 3. poglavlju „Tehnologija blockchaina. Podaci su dohvaćeni koristeći „useEffect“ kuku, kao što je prikazano u kôdu (Kôd 5.9).

```
useEffect(() => {
    const fetchEventsURIs = async () => {
      const allEvents = await
getAllItemsInCollection('events');
      const allConnectionURIsEventTypesAndNames =
        await getCollectionURIsEventTypesAndNames(allEvents);
      setCollectionsImages(allConnectionURIsEventTypesAndName
s);
      setLoading(false);
    };

    fetchEventsURIs();
  }, []);
```

Kôd 5.9 Korištenje „useEffect“ kuke za dohvat podatka o događajima

Kuka „useEffect“ poziva asinkronu funkciju „fetchEventsURIs“ koja unutar sebe poziva asinkronu funkciju „getAllItemsInCollection“ koja s Firebasea dohvaća adrese svih pametnih ugovora postavljenih na blockchain preko NFT Tickets aplikacije. Nakon dohvata

svih adresa, poziva se asinkrona funkcija „getCollectionURIsEventTypesAndNames“ s tim adresama kao jedinim parametrom prikazana na slici (Slika 5.20).

```
export const getCollectionURIsEventTypesAndNames = async (events: any) => {
  let collectionURIs: {
    id: string;
    image: string;
    eventType: string;
    name: string;
  }[] = [];
  await Promise.all(
    events.map(async (event: any) => {
      const nftContractInstance = getContractInstanceWithAlchemy(
        event.id,
        JSON.stringify(NFTTicketABI)
      );

      const collectionURI = await nftContractInstance.collectionURI();
      const eventType = await nftContractInstance.eventType();
      const name = await nftContractInstance.name();

      collectionURIs.push({
        id: event.id,
        image: collectionURI,
        eventType: eventType,
        name: name,
      });
    })
  );

  return collectionURIs;
};
```

Slika 5.20. Funkcija za dohvat podataka pametnih ugovora

Unutar asinkrone funkcije „getCollectionURIsEventTypesAndNames“ kreira se „collectionURIs“ polje. Unutar „await Promise.all“ bloka vrši se iteracija po adresama događaja smještenih u „events“ varijabli pa se za svaku adresu poziva funkcija „getContractInstanceWithAlchemy“. Funkcija koristi adresu pametnog ugovora, datoteku ekstenzije „json“, koja definira funkcije i potpise funkcija na pametnom ugovoru, poznata kao ABI [11]. Funkcija koristi i Alchemy Ethereum klijenta kako bi kreirala apstrakciju pametnog ugovora preko koje se potom mogu zvati funkcije direktno na blockchainu. Nakon poziva „getContractInstanceWithAlchemy“ funkcije, dalje se pozivaju funkcije koje dohvaćaju podatke s pametnog ugovora i dohvaćeni se podaci spremaju u „collectionURIs“ polje koje je ujedno i vrijednost funkcije. Dio kôda „await Promise.all“ koristi se kako bi se izvršavanje programa pauziralo dok asinkrone funkcije ne vrate svoje povratne vrijednosti, što sprječava grešku u kojoj se pokušavaju prikazati podaci koji nisu dohvaćeni.

Klikom na pojedini događaj otvaraju se detalji tog događaja koji se dohvaćaju na sličan način kao što je prethodno objašnjeno te se pojavljuju gumbi za kupnju ulaznica za događaj i za dodavanje događaja u kategoriju „omiljeni“.

Klikom na gumb „Buy Now“ poziva se asinkrona funkcija „mintNFTTicket“ prikazana na slici (Slika 5.21) koja kao argumente prima adresu kolekcije i cijenu NFT-a te umjesto Alchemy Ethereum klijenta koristi MetaMask kripto novčanik. Na taj način kreira apstrakciju pametnog ugovora nad kojim potom zove asinkronu funkciju „safeMint“ koja kreira Ethereum transakciju od strane korisnika, što je i razlog zašto se koristi MetaMask umjesto Alchemya, a potom čeka izvršenje iste transakcije, što može trajati i do minutu.

```
export const mintNFTTicket = async (
  collectionAddress: any,
  price: BigNumberish
) => {
  const nftContractInstance = getContractInstanceWithSigner(collectionAddress);

  await (await nftContractInstance.safeMint({ value: price })).wait();
};
```

Slika 5.21 Funkcija za kreiranje NFT-a

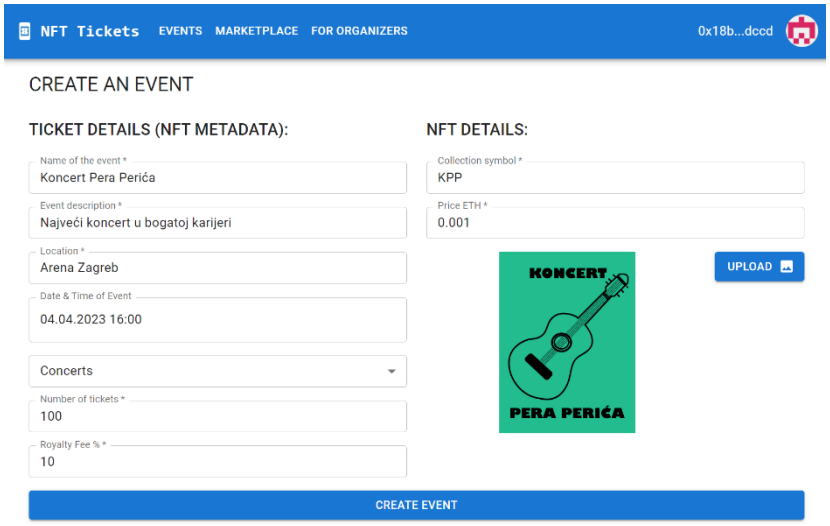
Pošto transakciju kreira korisnik, njeno izvršenje odobrava putem MetaMask ekstenzije, za što plaća i naknadu. Blockchain po uspješnom završetku transakcije kreira novi NFT i postavlja korisnika kao njegova vlasnika, čime je on uspješno kupio ulaznicu za događaj.

Što se tiče gumba za dodavanje i brisanje događaja iz „omiljenih“, klikom na njega se poziva asinkrona funkcija „updateFavouriteEvents“ vidljiva na slici (Slika 5.22) koja kao argumente prima ažuriranu verziju polja koje sadrži omiljene događaje i adresu kripto novčanika korisnika koji želi dodati ili izbrisati neki događaj iz omiljenih. Funkcija unutar Firebase kolekcije „userData“ ažurira dokument s „Document ID-jem“ jednakim adresi kripto novčanika korisnika i postavlja vrijednost „favouriteEvents“ polja jednaku ažuriranoj verziji polja omiljenih događaja, čime se postiže željena funkcionalnost.

```
export const updateFavouriteEvents = async (
  walletAddress: string,
  updatedEvents: string[]
) => {
  const userRef = doc(db, 'userData', walletAddress);
  await updateDoc(userRef, {
    favouriteEvents: updatedEvents,
  });
};
```

Slika 5.22 Funkcija za ažuriranje liste omiljenih događaja korisnika

Stranica za kreiranje novog događaja nudi polja za unos u koja korisnik unosi željene podatke o događaju koji želi kreirati, kao što je vidljivo na slici (Slika 5.23).



Slika 5.23 Stranica za kreiranje novog događaja

Nakon unosa željenih podataka, potrebno je kliknuti gumb „Create Event“. Tako se poziva asinkrona funkcija „submitForm koja, nakon početne validacije polja za unos, odabranu sliku prenosi na IPFS uslugu, a koja nudi pohranjivanje slika pomoću asinkrone funkcije „uploadToIPFS“ čija je uloga vratiti poveznicu na novoprenesenu sliku. U nastavku funkcija „submitForm“ poziva asinkronu funkciju „createNFTContract“ vidljivu na slici (Slika 5.24), koja prima podatke unesene u polja za unos i poveznicu na sliku, te pomoću ABI-ja, Bytecodea i MetaMaska, ako to odobri i plati korisnik, kreira apstrakciju pametnog ugovora.

```
export const createNFTContract = async (
  name: string,
  symbol: string,
  description: string,
  location: string,
  dateTime: number,
  eventType: string,
  numberOfTickets: number,
  priceInEth: BigNumberish,
  image: string,
  royaltyFeePercentage: number,
) => {
  try {
    const NFTTicketFactory = new ethers.ContractFactory(
      NFTTicketABI,
      NFTTicketBytecode,
      getMetamaskSigner()
    );

    const nftTicketInstance = await NFTTicketFactory.deploy(
      name,
      symbol,
      description,
      location,
      dateTime,
      eventType,
      numberOfTickets,
      fromEtherToWeiBigNumber(priceInEth.toString()),
      image,
      royaltyFeePercentage * 100
    );

    await nftTicketInstance.deployed();

    return nftTicketInstance;
  } catch (error) {
    return null;
  }
};
```

Slika 5.24 Funkcija za kreiranje novog događaja

Nakon kreiranja apstrakcije pametnog ugovora, funkcija poziva njegov konstruktor, čime kreira transakciju čije izvršenje i čeka. Nakon što je transakcija uspješno procesuirana, funkcija vraća adresu novokreiranog pametnoga ugovora u „submitForm“ funkciju koja tu adresu sprema u Firebase kolekciju „events“ u dokument čiji je „Document ID“ jednak adresi pametnog ugovora. Ondje je smješteno i polje „owner“ u kojem se zapisuje adresa kripto novčanika korisnika koji je kreirao pametni ugovor tog događaja.

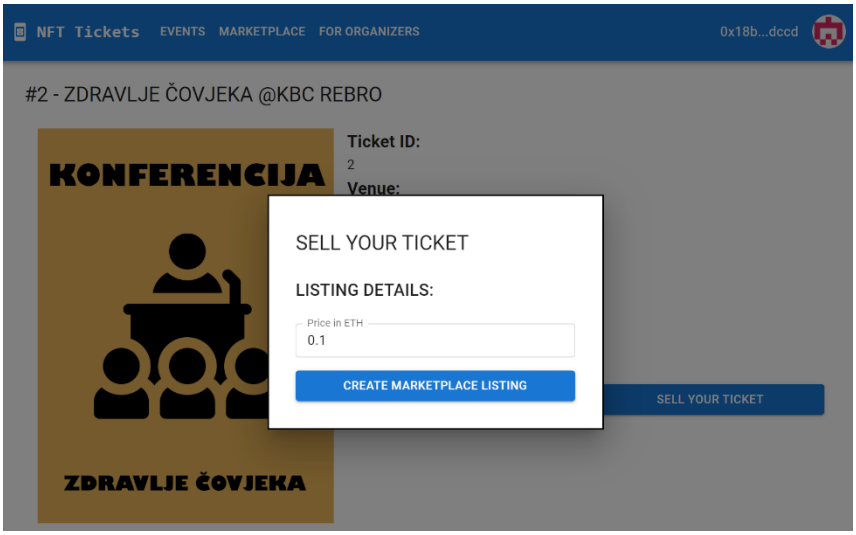
Funkcionalnost trgovine za preprodaju ulaznica realizirana je također koristeći pametni ugovor, ali taj pametni ugovor se ne kreira preko NFT Tickets aplikacije, kao pametni ugovori za događaje, već je postavljen na blockchain tijekom izrade aplikacije.

Stranica za preprodaju ulaznica vidljiva je na slici (Slika 5.25).

Slika 5.25 Stranica trgovine za preprodaju ulaznica

S lijeve strane nalazi se mogućnost filtriranja dostupnih ulaznica po kategorijama i trajanju događaja, što se postiže jednostavnom iteracijom po dohvaćenih ulaznicama te filtriranjem po zadanom kriteriju.

Klikom na gumb „Sell your ticket“, otvara se stranica „My tickets“ na kojoj je potrebno odabrati ulaznicu koja se želi preprodati i stisnuti gumb „Sell your ticket“ koji otvara prozor na kojemu je potrebno unijeti željenu cijenu ulaznice u preprodaji, kao što je vidljivo na slici (Slika 5.26).



Slika 5.26 Prozor za unos cijene ulaznice u preprodaji

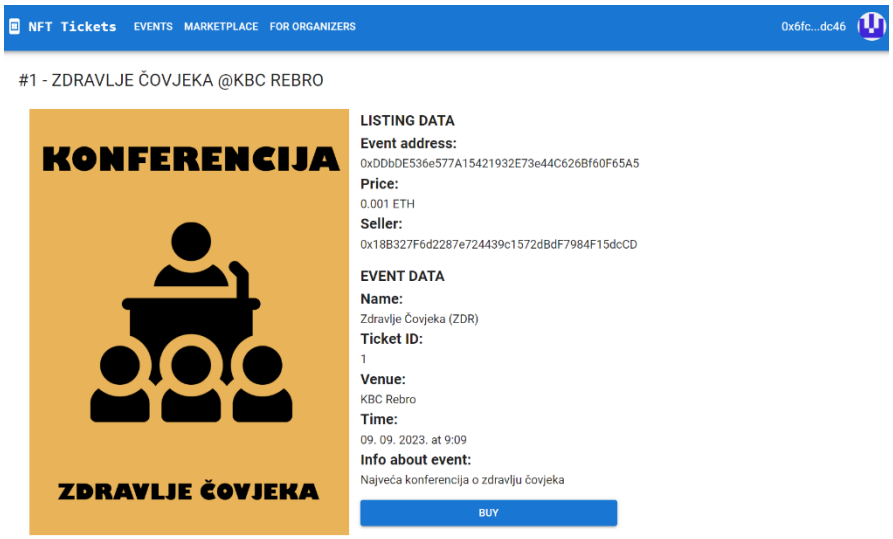
Nakon unosa željene cijene klikom na gumb „Create marketplace listing“, poziva se asinkrona funkcija „createMarketplaceListing“ vidljiva na slici (Slika 5.27).

```
export const createMarketplaceListing = async (
  collectionAddress: string,
  tokenId: number,
  price: number
) => {
  const marketplaceInstance = getMarketplaceInstanceWithSigner();
  const nftContractInstance = getContractInstanceWithSigner(collectionAddress);

  //approve
  await (await nftContractInstance.approve(marketplaceAddress, tokenId)).wait();

  //list
  await (
    await marketplaceInstance.makeItem(
      collectionAddress,
      tokenId,
      fromEtherToWeiBigNumber(price.toString())
    )
  ).wait();
};
```

Slika 5.27 Funkcija za kreiranje oglasa za preprodaju ulaznice

Funkcija „createMarketplaceListing“ kao argumente prima adresu pametnog ugovora događaja, ID ulaznice, koja se stavlja u prodaju, i željenu cijenu. Funkcija zatim dohvaća instancu „Marketplace“ pametnog ugovora koristeći MetaMask, što joj daje mogućnost kreiranja transakcija. Uz to, dohvaća i instancu pametnog ugovora pojedinog događaja, također koristeći MetaMask. Nakon toga od korisnika se traži potvrda transakcije putem „approve“ funkcije u kojoj dopušta „Marketplace“ pametnom ugovoru mogućnost prebacivanja ulaznice u svoje vlasništvo te čeka izvršenje transakcije. Posljednji korak u funkciji je kreiranje oglasa i potvrda korisnika unutar MetaMask ekstenzije za prebacivanje ulaznice u vlasništvo „Marketplace“ pametnog ugovora putem funkcije „makeItem“. Ulaznica je nakon izvršenja funkcije puštena u preprodaju i vidljiva je u trgovini za preprodaju ulaznica.

Ako je stranicu otvorio korisnik koji je taj oglas ujedno i postavio, pojavljuje mu se gumb „Delist“ kojim ulaznicu može vratiti iz „Marketplace“ pametnog ugovora u svoje vlasništvo. Ako je stranicu otvorio neki drugi korisnik, pojavljuje mu se prikaz kao na slici (Slika 5.28), gdje umjesto gumba „Delist“ stoji gumb „Buy“.



Slika 5.28 Stranica za kupovinu pojedine ulaznice u preprodaji

Klikom na gumb „Buy“ i potvrdom transakcije unutar MetaMask ekstenzije, korisnik postaje vlasnik ulaznice, a novčanik kreatora događaja uvećan je za vrijednost tantijeme.

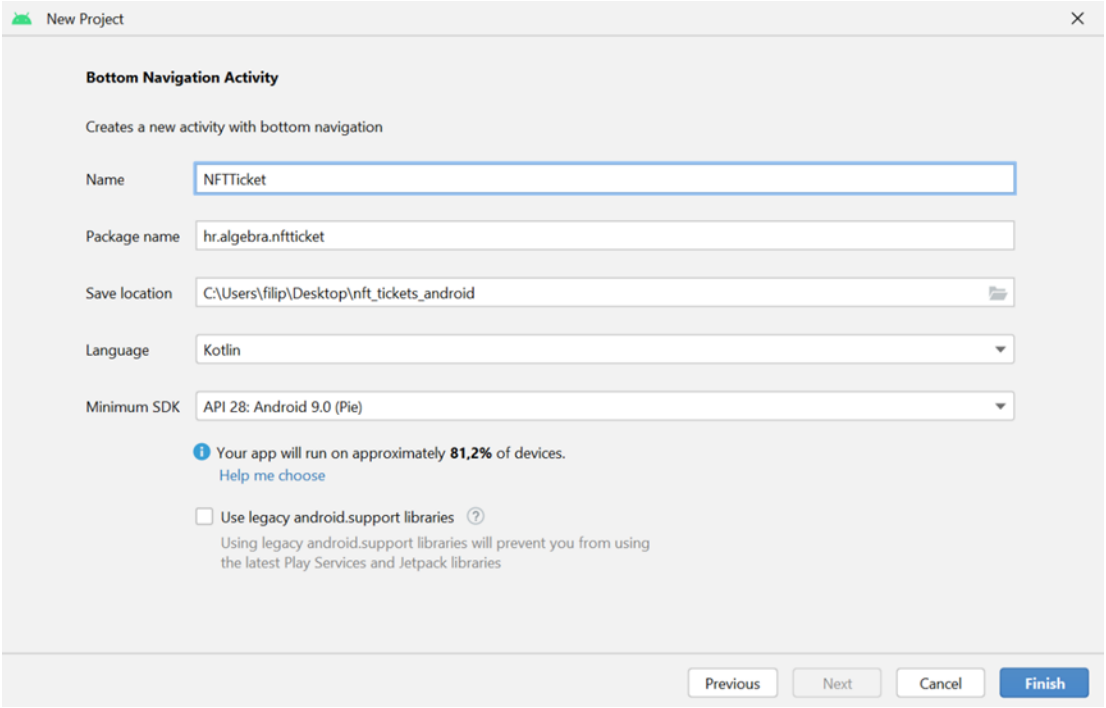
Što se tiče ostalih stranica na web-aplikaciji, one su vezane uz prikaz korisnikovih ulaznica, za prikaz događaja koji je kreirao korisnik i za prikaz korisnikovih omiljenih događaja. Njihova implementacija svodi se na dohvaćanje i prikaz podataka načinom koji je već objašnjen u ovom poglavlju.

5.2.2. Izrada android mobilne aplikacije

Android mobilna aplikacija sadrži manje funkcionalnosti od web-aplikacije, na kojoj je glavni fokus, pa je njezina izrada znatno jednostavnija.

Prilikom izrade android aplikacije korišten je predefinirani predložak naziva „Bottom Navigation Activity“. Navedeni predložak sastoji se od jedne glavne aktivnosti i nekoliko fragmenata zaslužnih za prikaz korisničkog sučelja te interakciju s korisnicima. Svaka aktivnost i fragment ima pripadajuće XML datoteke, zadužene za kreiranje korisničkog sučelja, i domenski razred zadužen za dohvat podatka i manipulaciju njima.

Nakon odabira predloška, potrebno je unijeti ime aplikacije i paketa (engl. *package name*) te odabrati minimalnu verziju SDK na kojoj će aplikacija raditi, kao što je vidljivo na slici (Slika 5.29). Za ovaj projekt odabrana je verzija „API 28: Android 9.0“.



Slika 5.29 Konfiguracija prilikom izrade android projekta

Nakon kreiranja projekta, potrebno mu je dodati Firebase SDK koji omogućava komunikaciju s bazom podataka. Slično kao i kod web-aplikacije, prvo je potrebno kopirati podatke s Firebase web-stranice i zalijepiti ih u „google-services.json“ datoteku unutar „src“ direktorija projekta. Nakon toga su dodane ovisnosti (engl. *dependencies*) za Firebase i Firestore, pokrenuta je sinkronizacija projekta s Gradle datotekama, koji je zadužen za automatizaciju izgradnje android projekata.

Funkcionalnosti android aplikacije su dohvat kupljenih ulaznica za korisnika, dohvat kreiranih događaja za organizatora i označavanje ulaznica iskorištenima prilikom skeniranja QR kôda od strane organizatora.

Podaci o pojedinom događaju nalaze se unutar podatkovnog razreda „Event“, prikazanog u kôdu (**Pogreška! Izvor reference nije pronađen.**), koji sadrži podatke jednake kao i dokumenti unutar Firestore kolekcije „events“.

```
data class Event(
    val name: String,
    val tokenOwnedBy: Map<Int, String>,
    val owner: String,
    val scanned: Map<Int, Boolean>,
    val venue: String
)
```

Kôd 5.10 Podatkovni razred „Event“

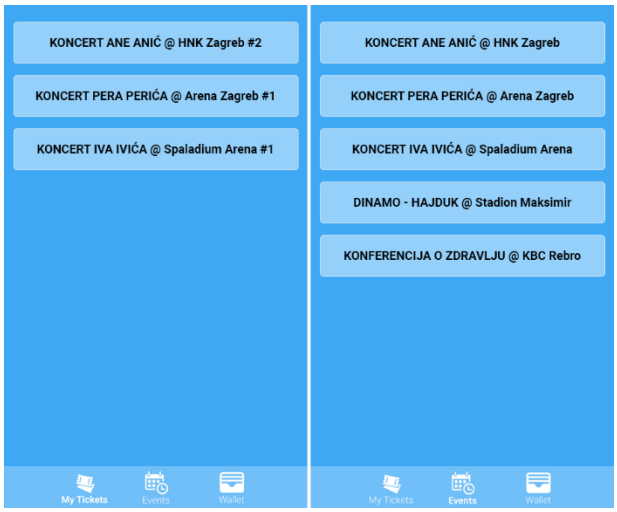
Za funkcionalnosti dohvata kupljenih ulaznica za korisnika i kreiranih događaja za organizatora koristi se jednaka funkcija prikazana kôdom (Kôd 5.11) zbog toga što „events“ kolekcija unutar Firebasea sadrži podatak tko je kreirao događaj i tko je vlasnik pojedine ulaznice.

```
fun getEvents() {

Firebase.firestore.collection("events").get().addOnSuccessLis
tener {
        val events: MutableList<Event> =
it.toObjects(Event::class.java)
        postEvents(events)
    }
}
```

Kôd 5.11 Funkcija za dohvat događaja

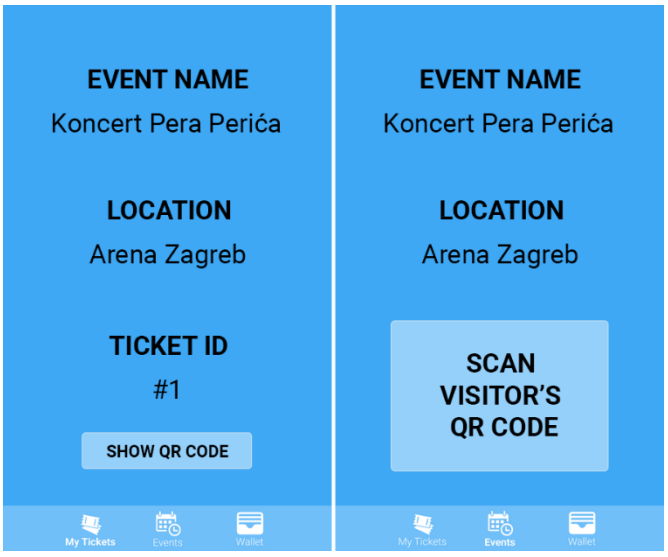
Funkcija „getEvents“ dohvaća sve dokumente iz kolekcije „events“ s Firestorea te nakon dohvata svih kolekcija pretvara podatke u listu podatkovnog tipa „Event“, koristeći refleksiju pomoću „toObjects“ metode [12]. Nakon što su događaji dohvaćeni, poziva se „postEvents“ funkcija koja postavlja vrijednost _events članske varijable na vrijednost upravo dohvaćenih događaja.

Nakon što su događaji dohvaćeni, potrebno ih je i prikazati. Prikaz kupljenih ulaznica i organiziranih događaja ostvaruje se koristeći RecyclerView biblioteku koja uz pomoć adaptera prikazuje podatke u formi liste, kao što je vidljivo na slici (Slika 5.30).



Slika 5.30 Prikaz kupljenih ulaznica (lijevo) i organiziranih događaja (desno)

Klikom na pojedinu ulaznicu ili događaj, otvara se stranica za detaljniji prikaz na kojoj je moguće, ovisno o tome radi li se o posjetitelju ili organizatoru događaja, prikazati QR kôd i skenirati ga, kao što je vidljivo na slici (Slika 5.31).



Slika 5.31 Prikaz pojedine ulaznice (lijevo) i događaja (desno)

Skeniranje ulaznica postiže se koristeći biblioteku Zebra Crossing (skraćeno ZXing) koja nudi funkcionalnosti generiranja i skeniranja QR kôdova. Skeniranjem QR kôda poziva se funkcija „setScanned“ prikazana kôdom (Kôd 5.12)

```
fun setScanned(event: Event, eventId: String, tokenId: Int) {
    val map = event.scanned.toMutableMap()
    map[tokenId] = true
    val modifiedEvent = Event(event.name, event.ownedBy,
event.owner, map, event.venue)
```

```
Firebase.firestore.collection("events").document(eventId).set
(modifiedEvent)
}
```

Kôd 5.12 Funkcija za postavljanje ulaznice iskorištenom

Glavna zadaća ove funkcije je trajno označiti pojedinu ulaznicu iskorištenom, što se postiže uz pomoć parametra „tokenId“ koji predstavlja ID ulaznice. Od mape „scanned“, koja je dio „event“ parametra, kreira se izmjenjiva (engl. *immutable*) mapa i u njoj se postavlja pod ključem jednakom ID-u ulaznice vrijednost „true“, i to označava da je ulaznica iskorištena. Zatim se zajedno s tim podatkom kreira instanca razreda „Event“ koja predstavlja pojedini događaj kojim se ažurira stari događaj unutar Firestorea, čime je završen proces skeniranja pojedine ulaznice.

6. Testiranje i analiza programskog rješenja

Kod programskog rješenja uspješnost je moguće promatrati kroz više aspekata, od kojih se izdvaja kvaliteta procesa razvoja samoga programskog rješenja i kvaliteta gotovog programskoga rješenja.

Zbog razvoja pojedinca kao programskog inženjera, potrebno je po završetku izrade programskog rješenja analizirati njegovu uspješnost u odnosu na očekivanja koja su postojala u početku razvoja tog rješenja. Važno je razumjeti i važnost organizacije pri izradi rješenja jer poboljšanjem kvalitete organizacije dolazi do kvalitetnijeg finalnog proizvoda, tj. u ovom slučaju aplikacije.

U cilju jasnijeg definiranja opsega ovoga programskog rješenja, u 2. poglavlju „Nedostaci klasičnog načina prodaje ulaznica i prijedlog programskog rješenja“, definirani su okvirni zahtjevi koje bi programsko rješenje trebalo imati da iskoristi potencijale u aktivnosti prodaje ulaznica koji su se stvorili pojavom blockchain tehnologije.

Razvijeno rješenje zadovoljava zahtjeve definirane u tom poglavlju, ali je za njegov razvoj trebalo više vremena od planiranog, što ukazuje da je neki drugi pristup razvojnom procesu možda mogao biti učinkovitiji.

Glavna ideja projekta bila je pojednostavljenje procesa puštanja ulaznica u prodaju i preprodaju te sigurnija i jednostavnija preprodaja ulaznica za preprodavača i kupca te je sve to ostvareno ovim programskim rješenjem.

Funkcionalnost, koja organizatoru omogućava dobivanje tantijeme od svake ulaznice prodane u preprodaji, može uvelike motivirati organizatore događaja da ulaznice prodaju upravo putem NFT Tickets aplikacije. Međutim, nedostatak statistike vezane uz preprodaju ulaznica, kao što je ukupna zarada od preprodaje, nešto je na čemu se može poraditi jer trenutno je praćenje profita od preprodaje moguće jedino ručnim praćenjem pojedinih transakcija u kripto novčaniku.

Od ostalih proširenja, koja bi bilo korisno dodati u nekim drugim iteracijama, tu je definitivno aplikacija za iOS sustave koja bi omogućila puno većem broju korisnika korištenje NFT Tickets aplikacije. Drugi koristan dodatak bio bi prijevod aplikacije na što više jezika, čime bi ona postala pristupačnija za korištenje korisnicima u više zemalja. Nadalje, treba poraditi na poboljšanju korisničkog iskustva i stvaranju vizualnog identiteta,

što bi programskom rješenju dalo dodatnu dozu ozbiljnosti koja također može privući korisnike.

Zadnje poboljšanje vrijedno isticanja je mogućnost odabira nekog jeftinijeg blockchaina pri kreiranju događaja od strane organizatora. Organizator bi onda odlučivao želi li svoje ulaznice pustiti u prodaju na Ethereum blockchainu, što je skuplje rješenje, ali u isto vrijeme jednostavnije i popularnije za posjetitelje događaja. Druga mogućnost mu je neki jeftiniji i manje popularni blockchain, kao što je Polygon, koji ima manje transakcijske naknade, što čini onda cijeli proces jeftinijim, ali malo kompleksnijim i samim time manje atraktivnim za posjetitelje događaja.

Unatoč navedenim područjima u kojima su moguća poboljšanja, uzevši u obzir da se radi tek o prvoj verziji NFT Tickets aplikacije, može se zaključiti da je postignuta poprilična razina uspjeha u razvoju ovoga programskog rješenja.

Zaključak

Blockchain je revolucionarna nova tehnologija čijom se primjenom ostvaruje veća doza neovisnosti o centralnom entitetu, tj. decentralizacija, koja vodi do toga da glavnu riječ ima kôd koji je napisan, a ne volja određenog pojedinca. Kao u mnogim drugim područjima, tako i u aktivnosti prodaje ulaznica, to vodi do otvaranja niza novih mogućnosti u izvedbi aplikacija.

Upravo je odatle potekla ideja za ovaj rad, čiji je cilj bio analizirati neiskorištene potencijale u aktivnosti prodaje ulaznica, koji su se stvorili pojavom blockchain tehnologije, te predložiti i napraviti rješenje koje bi te potencijale u što većoj mjeri iskoristilo.

Programsko rješenje je osmišljeno kako bi organizatorima događaja omogućilo da na jednostavan način puste ulaznice za svoj događaj u prodaju i da mogu, ako to žele, zarađivati i od preprodaje tih ulaznica. Posjetiteljima događaja ovo rješenje nudi jednostavan i siguran način dolaska do ulaznice za neki događaj, bilo u prodaji ili preprodaji, i preprodaje te ulaznice ako je iz nekog razloga ne planiraju sami iskoristiti.

NFT Tickets programsko rješenje je u finalnoj fazi razvoja te su sve početne ideje i realizirane tako da se programsko rješenje može smatrati uspješnim. Što se tiče potencijalnih nadogradnji programskog rješenja, one su iznesene u 6. poglavlju „Testiranje i analiza programskog rješenja“. Od tih nadogradnji može se istaknuti mogućnost odabira nekog jeftinijega blockchaina prilikom kreiranja događaja koji bi sam proces prodaje ulaznica učinio još jeftinijim.

Izradom ovoga završnog rada stekao sam korisno iskustvo u razvoju cjelokupnoga programskog rješenja koje se temelji na blockchain tehnologiji. Koristio sam niz novih biblioteka, alata, programskih jezika, razvojnih okvira i okruženja s kojima sam do sada imao malo ili niti malo iskustva, koji su produbili moje shvaćanje programskog inženjerstva i pomogli mi da se puno lakše odvažim pristupiti novim tehnologijama.

Ovaj rad je samo mala demonstracija moći blockchain tehnologije i smatram da uz kvalitetnu, ali i njezinu moralnu primjenu, možemo znatno poboljšati kvalitetu života i sustava u kojem živimo te istovremeno smanjiti troškove samog sustava.

Popis kratica

ABI	*Application binary interface*	*Aplikacijsko binarno sučelje*
API	*Application Programming Interface*	*Aplikacijsko programsko sučelje*
CSS	*Cascading Style Sheets*	*Cascading Style Sheets*
ERC-20	*Ethereum Request for Comment 20*	*Ethereum Request for Comment 20*
ERC-721	*Ethereum Request for Comment 721*	*Ethereum Request for Comment 721*
ERC-2981	*Ethereum Request for Comment 2981*	*Ethereum Request for Comment 2981*
HTML	*Hypertext Markup Language*	*Hypertext Markup Language*
HTTP	*Hypertext Transfer Protocol*	*Hypertext Transfer Protocol*
ID	*Identifier*	*Identifikator*
IPFS	*InterPlanetary File System*	*InterPlanetary File System*
JSON	*JavaScript Object Notation*	*JavaScript Object Notation*
JSON-RPC	*JavaScript Object Notation Remote Procedure Call*	*JavaScript Object Notation Remote Procedure Call*
NFT	*Non fungible token*	*Nezamjenjivi token*
NOSQL	*Not Only SQL*	*Ne relacijska baza podataka*
QR	*Quick response*	*Quick response*
SDK	*Software development kit*	*Paket za razvoj programa*
UI	*User interface*	*Korisničko sučelje*
UINT256	*Unsigned integer 256*	*Unsigned integer 256*
URL	*Universal Resource Link*	*Ujednačeni lokator sadržaja*
USDT	*United States Dollar Tether*	*United States Dollar Tether*
XML	*Extensible Markup Language*	*Extensible Markup Language*

Popis slika

Popis kôdova

Literatura

[1] Khandelwal, R. (2021.) “A Simple Guide to Understanding Blockchain” Dostupno na: https://medium.com/swlh/a-simple-guide-to-understanding-blockchain-8dd09356b153 [7. veljače 2023.]

[2] Jayathilake, I. (2018.) “Introduction to Hashing” Dostupno na: https://medium.com/@isuruj/introduction-to-hashing-5b4daf343889 [7. veljače 2023.]

[3] Ethereum Fundation (2023.) “Non-fungible tokens (NFT)” Dostupno na: https://ethereum.org/en/nft/ [7. veljače 2023.]

[4] Banks, A. (2020.) "Learning React: Modern Patterns for Developing React Apps" O'Reilly Media

[5] Howell, J. (2023.) “What Is Ethers.Js – A Detailed Guide” Dostupno na: https://101blockchains.com/ethers-js-tutorial/ [9. veljače 2023.]

[6] Javapoint (2023.) “Android Activity Lifecycle” Dostupno na: https://www.javatpoint.com/android-life-cycle-of-activity [9. veljače 2023.]

[7] Phillips, B., Stewart, C., Marsicano, K. (2017.) "Android Programming : The Big Nerd Ranch Guide" Big Nerd Ranch Guides

[8] Hussey, M., Phillips, D. (2022.) “What is MetaMask? How to Use the Top Ethereum Wallet” Dostupno na: https://decrypt.co/resources/metamask [17. veljače 2023.]

[9] Stevenson, D. (2018.) “What is Firebase? The complete story, abridged.” Dostupno na: https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0 [12. veljače 2023.]

[10] Solorio, K., Kanna, R., Hoover, D.H. (2019.) "Hands-On Smart Contract Development with Solidity and Ethereum: From Fundamentals to Deployment" O'Reilly Media

[11] Sen, S. (2023.) “What is an ABI?” Dostupno na: https://www.quicknode.com/guides/smart-contract-development/what-is-an-abi [15. veljače 2023.]

[12] Jemerov, D., Isakova, S. (2017.) "Kotlin in action" Manning

[13] Lewis, A. (2021.) "The Basics of Bitcoins and Blockchains: An Introduction to Cryptocurrencies and the Technology that Powers Them (Cryptography, Derivatives Investments, Futures Trading, Digital Assets, NFT)" Mango

[14] Griffiths, D., Griffiths, D. (2021.) "Head First Android Development: A Learner's Guide to Building Android Apps with Kotlin" O'Reilly Media