

# NFT TRGOVINA

---

**Tomasović, David Leopold**

**Master's thesis / Specijalistički diplomski stručni**

**2022**

*Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj:* **University of Split / Sveučilište u Splitu**

*Permanent link / Trajna poveznica:* <https://um.nsk.hr/um:nbn:hr:228:579652>

*Rights / Prava:* [In copyright](#)/[Zaštićeno autorskim pravom.](#)

*Download date / Datum preuzimanja:* **2025-04-03**



*Repository / Repozitorij:*

[Repository of University Department of Professional Studies](#)



UNIVERSITY OF SPLIT



**SVEUČILIŠTE U SPLITU**  
**SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE**  
Specijalistički diplomski stručni studij Primijenjeno računarstvo

**DAVID LEOPOLD TOMASOVIĆ**

**ZAVRŠNI RAD**

**NFT Trgovina**

Split, rujan 2022.

**SVEUČILIŠTE U SPLITU**  
**SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE**

Specijalistički diplomski stručni studij Primijenjeno računarstvo

**Predmet:** Programsko inženjerstvo

**ZAVRŠNI RAD**

**Kandidat:** David Leopold Tomasović

**Naslov rada:** NFT trgovina

**Mentor:** mr. sc. Karmen Klarin, viši predavač

Split, rujan 2022.

# Sadržaj

Sažetak.....	1
Summary.....	1
1. Uvod.....	2
2. Tehnologija i terminologija.....	3
2.1 Blockchain terminologija.....	3
2.1.1 Ethereum blockchain.....	3
2.1.2 Što je NFT?.....	4
2.1.3 Verzije Weba.....	4
2.1.3.1 Web 1.0.....	5
2.1.3.2 Web 2.0.....	5
2.1.3.3 Web 3.0.....	5
2.2 Solidity.....	6
2.2.1 Pisanje pametnih ugovora.....	7
2.2.2 Varijable, funkcije i provjere.....	8
2.2.2.1 Varijable.....	8
2.2.2.2 Funkcije.....	8
2.2.2.3 Require.....	9
2.3 Backend razvojno okruženje.....	9
2.3.1 Hardhat.....	10
2.3.1.1 Instalacija.....	10
2.3.2 Metamask.....	11
2.3.2.1 Instalacija.....	12
2.3.3 Ethers.js.....	12
2.3.3.1 Instalacija.....	13

2.3.4 Visual Studio Code.....	13
2.4 Frontend razvojno okruženje.....	13
2.4.1 TypeScript.....	14
2.4.2 React.....	15
2.4.2.1 Props, Hooks i JSX.....	16
2.4.2.1.1 JSX.....	16
2.4.2.1.2 Props.....	17
2.4.2.1.3 Hooks.....	17
2.4.3 React + TypeScript.....	18
2.4.3.1 Sučelje.....	19
2.4.4 Tailwind.....	19
2.4.4.1 Instalacija.....	20
2.4.4.2 Primjer.....	21
2.4.5 Git.....	23
3. Backend.....	25
3.1 Kreiranje i postavljanje.....	25
3.1.1. Hardhat.....	26
3.1.2 Solidity.....	27
3.1.3 Metamask.....	27
3.1.3.1 Pokretanje.....	29
3.2 Pametni ugovori.....	29
3.2.1 OpenZeppelin.....	30
3.2.2 <i>NFT.sol</i> .....	30
3.2.3 Marketplace.sol.....	31
3.2.3.1 Struct.....	32
3.2.3.2 Event.....	32
3.3 Značajne funkcije.....	33

3.3.1 MakeItem.....	33
3.3.2 PurchaseItem.....	34
3.4 Testiranje.....	36
3.5 Test za kreiranje NFT-a.....	37
3.5.1 <i>Happy</i> testovi.....	38
3.5.2 <i>Sad</i> testovi.....	38
3.6 Implementiranje pametnih ugovora.....	39
4. Frontend.....	42
4.1 Struktura mapa.....	42
4.2 Navigacija.....	46
4.2.1 React-router-dom.....	46
4.2.2 Izgled i funkcionalnost.....	47
4.2.2.1 Funkcionalnost.....	48
4.2.2.1.1 NAV_ITEMS.....	48
4.2.2.2 Izgled.....	49
4.2.2.2.1 NavItem.....	49
4.2.3 Prikaz.....	51
4.2.4 Container.....	52
4.3 Značajne komponente.....	53
4.3.1 Headline komponenta.....	53
4.3.2 NFT komponenta.....	54
4.4 Inicijalizacija pametnih ugovora.....	56
4.4.1 Ethers.js.....	57
4.5 Stranice.....	57
4.5.1 Home.....	58
4.5.1.1 Prikazivanje.....	58
4.5.1.2 Kupovanje.....	60

4.5.2 CreateItem.....	62
4.5.2.1 IPFS.....	62
4.5.2.2 Kreiranje i postavljanje.....	63
4.5.3 MyAssets.....	66
4.5.4 CreatorDashboard.....	68
5. Generiranje nasumičnih NFT-eva.....	71
5.1 Slojevi.....	71
5.2 Generiranje rijetkosti.....	73
5.2.1 Težinska nasumičnost.....	73
5.2.1.1 Jednostavna implementacija.....	74
5.2.1.2 Učinkovita implementacija.....	74
5.2.2 Praktična primjena.....	75
5.3. ER dijagram i prikaz prolaska slijeda projekta.....	77
6. Zaključak.....	86
Literatura.....	87

## Sažetak

### **NFT tržište**

U ovom završnom radu kreirano je NFT tržište na Ethereum blockchainu s dodatnom funkcijom nasumičnog generiranja slika prema slojevima slike. Unutar samog NFT tržišta, korisnici će kreirati vlastite NFT-ove, pregledavati već stvorene NFT-ove te prodavati i pregledavati prethodne aktivnosti unutar trgovine.

Za kreiranje pametnih ugovora korišten je programski jezik Solidity uz pomoć razvojnog okruženja Hardhat. Za frontend se koristio TypeScript i React za kreiranje korisničkog sučelja. Kako bih se povezali pametni ugovori sa korisničkim sučeljem se koristio Metamask digitalni novčanik.

Ključne riječi: Blockchain, NFT, Ethereum, TypeScript, generiranje slika

## Summary

### **NFT Marketplace**

In this final paper, an NFT marketplace is created on the Ethereum blockchain with an additional function of randomly generating an image according to the image layers. Within the NFT marketplace itself, users will create their own NFTs, review already created NFTs, sell and review past activities within the store.

The Solidity programming language was used to create smart contracts, with the help of the Hardhat development environment. For the frontend, TypeScript and React were used to create the user interface. In order to connect the smart contracts with the user interface, the Metamask digital wallet was used.

Keywords: Blockchain, NFT, Ethereum, TypeScript, image generation



## 1. Uvod

U ovom završnom radu je napravljeno NFT tržište (engl. *non-fungible token*, *NFT marketplace*) s dodatnom funkcionalnošću generiranja nasumičnih slika koristeći slojeve slika. Mreža na kojoj će biti pokrenuta je Ethereum blockchain. Unutar NFT tržišta će korisnici moći kreirati vlastite NFT-eve, pregledavati stvorene, prodavati i pregledati povijest aktivnosti unutar trgovine.

Ova je tema izabrana zbog rastućeg interesa za kripto područje u posljednjih nekoliko godina. U protekle dvije godine prevladavalo je bikovsko tržište (engl. *bull market*) čime su narasle cijene. Samim time, iznimno je porasla i potražnja za *developerima*. Kao vrlo aktualna tema, NFT-evi su dobili na popularnosti.

Ovaj projekt je kreiran pomoću Create-React-App uz TypeScript i Tailwindcss. Kako bi se napisali pametni ugovori korišten je Solidity. Za pristup pametnim ugovorima korišteno je Hardhat *developersko* okruženje, Metamask kao digitalni novčanik te `ethers.js` biblioteku zbog komunikacije sa Metamask novčanikom.

Ovaj završni rad je podijeljen na:

- 1 Uvod – kratak opis i sadržaj
- 2 Tehnologija i terminologija – opisivanje tehnologija su korištene unutar projekta
- 3 Backend – podešavanje *developerskog* okruženja i pametni ugovori
- 4 Frontend – povezivanje *backenda* s web preglednikom i podešavanje frontend *developerskog* okruženja
- 5 Generator slika – opisivanje algoritma za generiranje slika od manjih dijelova, slojeva (engl. *layers*) i povezivanje slika na blockchain.
- 6 Zaključak – Opis osobnog iskustva pisanja završnog rada i moji dojmovi na stečeno znanje
- 7 Literatura – svi izvori korišteni tijekom pisanja ovog dokumenta.

## 2. Tehnologija i terminologija

Za razumijevanje rada i projekta nužno je upoznati osnovne pojmove tehnologije i terminologije koja je korištena u ovom projektu.

### 2.1 Blockchain terminologija

Blockchain je kriptografska knjiga podataka koja pohranjuje informacije poput transakcija i vremenskih oznaka. Kako bi se nove informacije dodale u “knjigu”, moraju ih provjeriti *validatori* i utemeljiti temeljni mehanizam konsenzusa. Tako se svi prethodno dodani podaci označavaju istinitima. Ovaj korak je neophodan kako bi se u postojeći lanac mogli dodati novi podaci.

#### 2.1.1 Ethereum blockchain

Ethereum je javna blockchain mreža otvorenog koda slična Bitcoinu. Razlikuju se utoliko što Ethereum uvodi nekoliko ključnih tehnologija - ponajviše za pametne ugovore i virtualni stroj poput potpunog *Turinga* (engl. *turing completeness*) [1].

Zahvaljujući pametnim ugovorima Ethereum postaje moćna platforma za izgradnju financijskih ugovora i drugih aplikacija na decentraliziran način bez dopuštenja. Aplikacije koje se rade na Ethereumu se ujedno nazivaju “dApp” - decentralizirana aplikacija (engl. *Decentralized Application*).

Korištenje Ethereum blockchaine zahtijeva kriptovalutu Ether. Ona se služi za korištenje blockchaine, ali i za razmjenu vrijednosti i usluga na Ethereumu.

Platforma je univerzalna i modularna. Korisnici iz cijelog svijeta komuniciraju izvornim kodnim jezikom Solidity. Time je čak i sama komunikacija maksimalno pojednostavljena.

## 2.1.2 Što je NFT?

NFT (engl. *non-fungible token*) je token koji se može koristiti za uspostavljanje vlasništva nad jedinstvenim stvarima - umjetninama, kolekcionarskim predmetima, nekretninama, virtualnim dobrima [2]. NFT-ovi u isto vrijeme mogu biti pod vlasništvom samo jedne osobe te su osigurani blockchainom. Najčešće je korišten Ethereum blockchain. Zapis o vlasništvu se ne može mijenjati. Isto tako, nemoguće je kopirati ili zalijepiti novi NFT u optok.

Standard koji se koristi za pisanje NFT-ova se zove ERC721, dok od nedavno postoji i ERC1155 koji podržava višestruko kovanje (engl. *minting*) NFT-eva. Nezamjenjivo je ekonomski izraz koji se koristi za opisivanje stvari poput namještaja, datoteke pjesme ili računala. Te stvari nisu zamjenjive za druge predmete jer imaju jedinstvena svojstva koja ih čini prepoznatljivima.

S druge strane, zamjenjivi predmeti se mogu razmjenjivati jer su definirani njihovom vrijednošću, a ne svojstvima. Tako su, na primjer, ETH i Američki dolari zamjenjivi jer je 1 ETH/1 USD zamjenjiv za drugi 1 ETH/1 USD.

NFT i Ethereum rješavaju neke od problema koji danas postoje na internetu. S obzirom na porast digitalizacije u posljednjih nekoliko desetljeća, nastala je potreba za repliciranjem svojstava fizičkih predmeta kao što su jedinstvenosti i dokaza vlasništva. Digitalni predmeti često funkcioniraju samo u kontekstu svog proizvoda [3]. Na primjer, već kupljen iTunes mp3 se ne može ponovno prodati, te se ne mogu zamijeniti bodovi vjernosti jedne tvrtke za kredit druge platforme, čak ako za to i postoji tržište. Svrha NFT-ova leži upravo u tome – oni vraćaju vlasništvo kupcu.

## 2.1.3 Verzije Weba

Web se, od svog nastanka do danas, zamjetno razvio. Njegove primjene danas su, u usporedbi s onima za koje se inicijalno koristio, gotovo neprepoznatljive. Razvitak weba često je podijeljena u tri odvojene faze: Web 1.0, Web 2.0 i Web 3.0. [4]

### 2.1.3.1 Web 1.0

Web 1.0 je bila prva iteracija weba. Većina sudionika bili su korisnici sadržaja, a kreatori su obično bili programeri čija je uloga bila izgradnja web-stranica koje su sadržavale informacije uglavnom u tekstualnom ili slikovnom formatu. Razdoblje Web 1.0 trajalo je otprilike od 1991. do 2004. godine.

Web 1.0 je sadržavao stranice koje poslužuju statički sadržaj, umjesto dinamičkog HTML-a. Podaci i sadržaj bili su posluženi iz statičkog datotečnog sustava, a ne iz baze podataka, dok stranice nisu imale puno interaktivnosti. Web 1.0 se može zamisliti kao web namijenjen isključivo čitanju.

### 2.1.3.2 Web 2.0

Web 2.0 je trenutno stanje weba - može se zamisliti kao interaktivni i društveni web. Nije potrebno biti programer kako bi se sudjelovalo u procesu stvaranja. Mnoge su aplikacije izgrađene na način koji svakome lako omogućuje da bude kreator njihovog sadržaja.

Osmišljenu ideju lako je podijeliti s ostatkom svijeta, a isto tako je moguće, primjerice, prenijeti videozapis i omogućiti milijunima ljudi da ga vide, komentiraju i komuniciraju s drugim korisnicima putem njega. Zbog svoje iznimne jednostavnosti korisnici diljem svijeta postaju ujedno i kreatori.

### 2.1.3.3 Web 3.0

Postoji nekoliko temeljnih razlika između Web 2.0 i Web 3.0 - primarna je decentralizacija. Web 3.0 poboljšava internet u obliku koji je većini poznat te pridodaje još nekoliko karakteristika – povjerljivost, samoupravnost, ne oslanja se na povjerenje, nema dopuštenja, raspodijeljen je i robustan. Isto tako, u Web 3.0 su ugrađena plaćanja koji omogućuju slanje i primanje tokena.

Kod Web 3.0 programeri obično ne izgrađuju i ne implementiraju aplikacije koje se pokreću na jednom serveru ili na jednom serveru te ne pohranjuju svoje podatke u jednu bazu podataka. Ustaljeno je postaviti server na jednom od poslužitelja oblaka – npr. Google Firebase, Amazon Web Services.

Web 3.0 aplikacije se pokreću na blockchainovima, tj. decentraliziranim mrežama mnogih *peer-to-peer* mreža (engl. *peer-to-peer network*) ili kombinacijom, tj. kripto-ekonomskim protokolom. Takve se aplikacije često nazivaju dApps (engl. *decentralized applications*), vrlo poznat izraz u Web3 prostoru.

Kada se govori o Web 3.0, kripto valuta je često dio razgovora. Ona igra veliku ulogu u mnogim protokolima te pruža financijski poticaj, tj. tokene, za svakoga tko želi sudjelovati u kreiranju, upravljanju ili poboljšanju nekog od projekata.

## 2.2 Solidity

Solidity je relativno nov programski jezik koji je kreiran za kreiranje pametnih ugovora na Ethereum mreži, drugo najveće tržište kriptovaluta po kapitalizaciji. Osnovao ga je Christian Reitwiessner [5]. Logo za Solidity programski jezik može se vidjeti na slici 1).

Neke ključne značajke Solidityja:

- To je programski jezik visoke razine, dizajniran za implementaciju pametnih ugovora.
- Statički je tipiziran i objektno, tj. ugovorno orijentiran jezik.
- Pod velikim je utjecajem Python-a, C++ i JavaScript-a koji rade na Ethereum Virtual Machine (EVM).
- Podržava složeno, korisnički definirano programiranje, knjižnice (engl. *library*) i nasljeđivanje.
- Primarni je jezik za blockchain platforme.

- Može koristiti za stvaranje ugovora poput glasanja, slijepe aukcije, dražbe, „novčanika s više potpisa“ (engl. *multisig wallets*), itd.



Slika 1: Solidity logo

### 2.2.1 Pisanje pametnih ugovora

Prilikom kreiranja svakog Solidity pametnog ugovora, potrebno je na samom početku utemeljiti licencu određuje tko smije ili ne smije koristiti taj kod, tj. ugovor [6]. Primjer se može vidjeti na slici 2. Iako je često na više mjesta definirano koja će se verzija Solidity programskog jezika koristiti, praksa je postaviti ju unutar pametnog ugovora (prikazano na slici 2).

```
100, 5 weeks ago | 1 author (100)  
1 // SPDX-License-Identifier: MIT  
2  
3 pragma solidity ^0.8.4;  
.
```

Slika 2: Licenca za pametni ugovor

## 2.2.2 Varijable, funkcije i provjere

### 2.2.2.1 Varijable

Kao i u svakom programskom jeziku, tako su i u Solidityju definirane varijable. Postoje tri različita tipa varijabli:

- 1 Varijable stanja – varijable čije su vrijednosti trajno pohranjene u skladištu ugovora, a njihova promjena košta Ether.
- 2 Lokalne varijable – varijable čije su vrijednosti prisutne dok se funkcija ne izvrši.
- 3 Globalne varijable – posebne varijable koje postoje u globalnom imenskom prostoru koji se koristi za dobivanje informacija o blockchainu. To su:

3.a `Msg.sender` - adresa osobe koja je pozvala pametni ugovor

3.b `Msg.value` - vrijednost koju je poslala osoba koja je pozvala pametni ugovor

U Solidity programskom jeziku potrebno je prilikom kreiranja varijable deklarirati i koje je ona vrste. Postoji nekoliko glavnih vrsta podataka:

- `Bool` – istinite ili lažne izjave
- `Int/uint` – cijeli brojevi bez i s predznakom
- `Int8` do `int256` – cijeli brojevi s ili bez predznaka, dužine od 8 do 256 bitova
- `Uint8` do `uint256` – cijeli pozitivni brojevi, dužine od 8 do 256 bitova

### 2.2.2.2 Funkcije

Funkcije mogu imati više razina vidljivosti, a one su:

- `Public` - prema zadanim postavkama (engl. *default*) svaka je funkcija javna. To znači da bilo tko (ili bilo koji drugi ugovor) može pozvati funkciju tog ugovora i izvršiti njezin kod.
- `Private` - funkciju je samo moguće pozvati samo unutar istog pametnog ugovora.

- Internal – slično funkciji *private*, osim što je dostupan ugovorima koji se nasljeđuju iz ovog ugovora
- External - sličan *publicu*, osim što se te funkcije mogu isključivo pozvati izvan ugovora, ali ne i drugim funkcijama unutar tog ugovora

Bitno je naglasiti kako funkcije trebaju imati eksplicitno napisano koji se tip podatka vraća korištenjem “returns” ključne riječi.

### 2.2.2.3 Require

Kako bih se spriječilo manipuliranje ili ubacivanje vrijednosti koje ne odgovaraju, Solidity ima način provjere stanja varijabli prije ulaska u funkciju. To se obavlja korištenjem ključne riječi “require”. Kao što u drugim programskim jezicima postoji “if” tako u Solidity postoji “require” koji, ako se ne zadovolji uvjet, zaustavlja daljnji rad programa.

Primjer “require” izjave možemo vidjeti na slici 3. Ukoliko je je cijena veća od 0 program ide dalje, no ako je cijena manja program se neće dalje izvršavati te vraća povratnu poruku.

```
66 | require(_price > 0, "Price must be greater than zero");
```

Slika 3: Primjer 'require' izjave

## 2.3 Backend razvojno okruženje

Kako bi se moglo započeti s kreiranjem pametnog ugovora potrebno je podesiti *developersko* okruženje za njihovo programiranje. U ovom radu su u tu svrhu korišteni:

- Hardhat
- Metamask
- Ethers.js
- Visual Studio Code



### 2.3.1 Hardhat

Hardhat je razvojno okruženje za sastavljanje, implementaciju, testiranje i otklanjanje pogrešaka u Ethereum softveru [7]. Pomaže programerima pri upravljanju i automatizaciji ponavljajućih zadataka koji su nužni u procesu izgradnje pametnih ugovora i dApp-ova. Hardhat čini proces dodavanja više funkcionalnosti tijekom rada projekta jednostavnijim. Dodavanje više funkcionalnosti označava sastavljanje, pokretanje i testiranje pametnih ugovora.

Hardhat dolazi ugrađen u Hardhat Network, lokalnu Ethereum mrežu dizajniranu za razvoj. Njegova se funkcionalnost usredotočuje na otklanjanje pogrešaka u Solidityju. Hardhat omogućava pisanje `console.log` komande unutar pametnih ugovora i daje eksplicitne poruke o pogrešci ukoliko transakcija nije uspješna.

Za početnike, Hardhat se može činiti zastrašujućim. On je spoj svih dijelova softvera koji su ponuđeni među glavnim izborima pri pokretanju bilo kojeg Ethereum projekta. Logo Hardhata nalazi se na slici 4.



Slika 4: Hardhat logo

#### 2.3.1.1 Instalacija

Kako bi se dohvatila Hardhat instalacija u projekt, potrebno je upotrijebiti komandu:

```
npm install --save-dev hardhat.
```

Nakon što se Hardhat instalira, potrebno je inicijalizirati Hardhat projekt što se postiže komandom: `npx hardhat`. Prilikom toga, dobiva se CLI (engl. *Command Line Interface*) za odabir vrste konfiguracije projekta.

### 2.3.2 Metamask

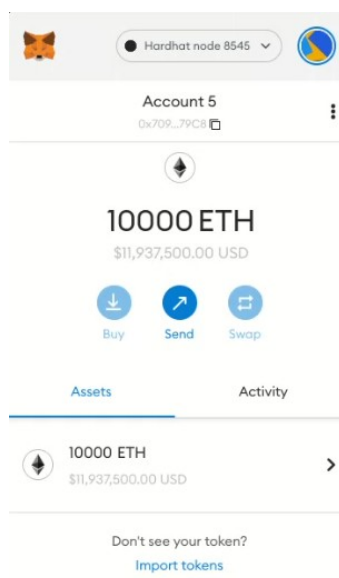
Kako bi se pristupilo bilo kojem dApp-u potreban je digitalni novčanik koji će biti spojen na njega. U *Web 3.0*, digitalni novčanik služi poput osobne iskaznice s kojom se prijavljuje na stranice i koristi različite programe. Bez digitalnog novčanika nije moguće pristupiti uslugama *Web 3.0* [8].

Daleko najpoznatiji digitalni novčanik je Metamask - dodatak za preglednik koji je instaliran kao i svaki drugi takav dodatak. Nakon instalacije, korisnicima omogućuje pohranu tokena s raznih mreža te im omogućava transakcije s bilo kojom adresom na mreži koja je odabrana. Logo je vidljivi na slici 5.



*Slika 5: Metamask logo*

Izvršna osobina Metamaska je mogućnost povezivanja na bilo koju mrežu (Ethereum, Binance Smart Chain, Polygon, Avalanche, itd.). Potrebno je na internetu pronaći podatke koji su potrebni kako bi Metamask znao uspostaviti vezu, a jednom kad je to podešeno, moguće je promijeniti mreže putem jednostavnog izbornika. Sliku Metamask sučelja je na slici 6.



Slika 6: Metamask grafičko sučelje

Povezivanjem Metamaska na dApp-ove utemeljene na Ethereumu, korisnici mogu trošiti kriptovalute u igrama, ulagati tokene u aplikacije za kockanje, ali i trgovati njima na decentraliziranim burzama (DEX). Također, korisnicima je pružena ulazna točka u novi svijet decentraliziranih financija (DeFi) te je omogućen pristup DeFi aplikacijama poput Aavea i Compounda.

### 2.3.2.1 Instalacija

Prednost Metamask novčanika je njegova velika popularnost zbog koje je vrlo dobro podržan na svim popularnim web preglednicima. Jednostavno se pretraži Chrome Web Store ili Firefox Browser Add Ons te se instalira samo jednim klikom. Nakon instalacije potrebno je napraviti račun. To zapravo čini cijeli postupak kako bi korisnikov web bio spreman koristiti dApp-ove.

### 2.3.3 Ethers.js

Nakon što se u projektu nalazi Ethereum razvojno okruženje, digitalni novčanik je posljednja postavka koja se treba povezivati kako bi dva elementa mogli komunicirati.

Postoje različite biblioteke za to, no najpopularniji su `ethers.js` i `Web3.js` [9]. Razlog odabira `ethers.js` je zbog njegove jednostavnosti prilikom korištenja i podešavanja. Iako je `Web3.js` isto popularan, `ethers.js` se češće koristiti zbog lakoće pisanja. `Ethers.js` je biblioteka dizajnirana za lakšu komunikaciju s pametnim ugovorima i digitalnim novčanicima. Temelji se na JavaScriptu te se primjenjuje za slične aplikacije.

### 2.3.3.1 Instalacija

Kako bi se instalirao `ethers.js`, u projektu se koristi komanda: `npm install --save ethers`. Za upotrebu se koristi komanda: `import { ethers } from "ethers"`.

### 2.3.4 Visual Studio Code

U ovom projektu će se koristiti VS Code (engl. *Visual Studio Code*) kao odabrani tekstualni editor. Kreirao ga je Microsoft te je daleko najpopularniji tekstualni uređivač za programere zbog svoje iznimne plastičnosti i mnogo ekstenzija pomoću kojih se mogu dodavati razne funkcionalnosti i mijenjati teme izgleda [10]. Također, dio popularnosti leži u mogućnosti formatiranja, povezivanja s različitim servisima i dr.

Odlično svojstvo VS Codea je njegov *Intellisense* koji preporučuje dovršavanje pisanja trenutne linije koda. Ujedno, postoje i *snippets*, odnosno kratice koji se mogu koristiti kako bi se neki komad koda, koji se često koristi, napisao brže.

## 2.4 Frontend razvojno okruženje

Kako bi korisnici jednostavno komunicirali s pametnim ugovorima i blockchainom, potrebno je imati lijepo grafičko sučelje koje to omogućuje. Ovaj rad je izgrađen koristeći:

- TypeScript

- React
- Tailwind

### 2.4.1 TypeScript

TypeScript je programski jezik kojeg je razvio i održava Microsoft. To je strogo sintaktički nadskup (engl. *Superset*) JavaScripta koji jeziku dodaje neobavezno statičko tipkanje. Dizajniran je za razvoj velikih aplikacija i prenosi se na JavaScript. Budući da je to nadskup JavaScripta, postojeći JavaScript programi također su valjani kao i TypeScript programi [11].



Slika 7: JavaScript i TypeScript logo

Dostupno je više opcija za transpilaciju, tj. pretvaranje TypeScript koda u čisti JavaScript. Može se koristiti zadani prevoditelj TypeScripta, ili se može pozvati Babel prevodilac. U slici se vide sličnosti i u samoj ikoni TypeScripta (desna ikona s plavom pozadinom) u usporedbi s JavaScriptom (lijeva ikona za žutom pozadinom).

U JavaScriptu postoji nekoliko tipova koji se dijele na primitivne i ne primitivne tipove [12].

Primitivni tipovi:

- Number
- BigInt

- String
- Boolean
- Null
- Undefined

I ne primitivni tipovi:

- Object

JavaScript datoteke se postavljaju s ekstenzijom `.js`, dok u TypeScriptu je ekstenzija `.ts`.

Kod kreiranja neke varijable ili funkcije, moguće je nadodati kojeg je tipa ta varijabla ili koju povratnu vrijednost će imati neka funkcija. Time se osigurava manje problema s kodom te, pogotovo u većim projektima, sve postaje puno jasnije. Na slici 8 se može vidjeti primjer funkcije koja se pojavljuje u projektu, a koja postavlja prvo početno slovo na tekst koji je proslijeđen u funkciju.

```
const capitalizeFirstLetter = (text: string): string =>
  text.charAt(0).toUpperCase() + text.slice(1);
```

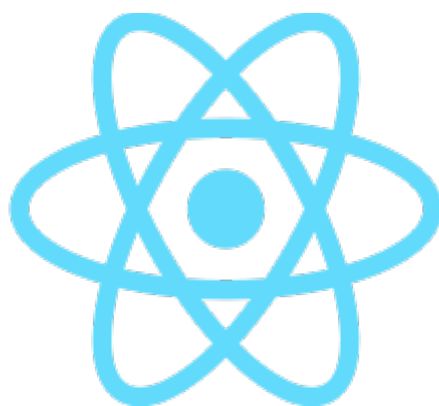
Slika 8: CapitalizeFirstLetter funkcija

Na prvi pogled može se odmah vidjeti da parametar “text” je tipa *string*, i da funkcija vraća *string* tip. Iako vrlo jednostavno i banalno, u kompleksnijim funkcionalnostima postaje vrlo korisno.

#### 2.4.2 React

React je JavaScript biblioteka otvorenog koda za izgradnju korisničkih sučelja temeljenih na komponentama. Održavaju ga Meta (bivši Facebook) te zajednica pojedinačnih programera i tvrtki koji doprinose projektu [13]. React se može koristiti kao baza u razvoju jednostraničnih, mobilnih ili poslužiteljskih aplikacija i sl.

Međutim, React se bavi isključivo upravljanjem stanjem podataka i prikazivanjem tog stanja u DOM (engl. *Document Object Model*), tako da stvaranje React aplikacija obično zahtijeva korištenje dodatnih biblioteka za usmjeravanje, kao i određene funkcionalnosti od strane klijenta. Prednost Reacta je što je iznimno popularna, ako ne i najpopularnija tehnologija za kreiranje web stranica pa je tako iznimno dobro podržan od Meta i korisnika. Logo Reacta je prikazan na slici 9.



Slika 9: React logo

#### 2.4.2.1 Props, Hooks i JSX

Neke od glavnih aspekata Reacta su:

- JSX
- Props
- Hooks

##### 2.4.2.1.1 JSX

JSX, tj. JavaScript Syntax Extension, je proširenje sintakse JavaScript jezika. Po izgledu je sličan HTML-u, ali JSX pruža način strukturiranja tako što prikazuje komponente koristeći sintaksu koja je slična HTML-u uz mogućnost ubacivanja JS-a.

React komponente se obično pišu pomoću JSX-a, iako to nije nužno. Komponente također mogu biti napisane u čistom JavaScriptu.

#### 2.4.2.1.2 Props

Props u Reactu je poput parametara funkcije ili atributa u HTML-u. To su vrijednosti koje su prosljeđene u komponentu u Reactu. U primjeru na slici 10 može se vidjeti destrukuirani *prop* koji se šalje u tu komponentu (treća linija). Na liniji jedanaest je primjer prosljeđivanja podatka u komponentu.

```
1  import React from 'react'
2
3  export const TestComponent = ({text}) => {
4    return (
5      <div>TestComponent: {text}</div>
6    )
7  }
8
9  const Untitled = () => {
10   return (
11     <TestComponent text="Demo text" /> )
12 }
```

Slika 10: Propovi u Reactu

#### 2.4.2.1.3 Hooks

*Hooks* omogućuje komponentama da pristupe stanju podataka te pristup ostalim React funkcionalnostima. Prije su se u Reactu koristile klase, no od verzije 16.8 je napravljena tranzicija prema *hookovima* i funkcionalnim komponentama. Najpoznatiji *hook* je `useState` koji omogućuje praćenje stanja u funkcijskoj komponenti. Stanje se općenito odnosi na podatke ili svojstva koja se trebaju pratiti u aplikaciji.

Primjer `useState` *hooka* je prikazan na slici 11. Prije korištenja `useState` *hooka* je potrebno uključiti komponentu koristeći naredbu koja je prikazana na prvoj liniji na slici.



useState se sastoji od vrijednosti i funkcije koja postavlja tu vrijednost (na četvrtoj liniji, numberOfClicks je vrijednost, a setNumberOfClicks je funkcija koja postavlja vrijednost). Kod inicijalizacije se destruktuira u ta dva dijela, a u zagrade se postavlja početna vrijednost. U ovom primjeru se prati koliko puta je korisnik kliknuo na ovu komponentu pri čemu će se pozvati funkcija useState *hooka* i povećati vrijednost za jedan. Prilikom pozivanja funkcije za postavljanje vrijednosti useState, cijela se komponenta ponovno “nacrtava” na zaslone. Time postižemo reaktivne komponente.

```
1  import React, {useState} from 'react'
2
3  export const TestComponent = ({ text }) => {
4    const [numberOfClicks, setNumberOfClicks] = useState(0);
5
6    return (
7      <div onClick={() => setNumberOfClicks(numberOfClicks + 1)}>
8        TestComponent: {text}
9        Number of clicks: {numberOfClicks}
10     </div>
11   );
12 };
```

Slika 11: useState hook

### 2.4.3 React + TypeScript

Korištenje TypeScripta u normalnim JavaScript programima u kombinaciji s Reactom je od iznimne koristi jer pomaže s nizom problema koje sam React donosi. Prilikom korištenja TypeScripta s Reactom mijenjamo ekstenziju komponenti s **.jsx** na **.tsx**. Time se naznačuje prevoditelju kako se zaista radi o TypeScript komponenti. Na slici 12 se može vidjeti primjer komponente koja je napisana u TypeScriptu [14].

```

1  import React from "react";
2
3  interface TestComponentProps {
4    | text: string;
5  }
6  export const TestComponent = ({ text }: TestComponentProps) => {
7    | return <div>TestComponent: {text}</div>;
8  };

```

Slika 12: TypeScript komponenta

#### 2.4.3.1 Sučelje

Sučelje (engl. *interface*) je sintaktički ugovor kojeg bi se entitet ili objekt trebao pridržavati. Sučelja ujedno definiraju svojstva, metode i događaje koji su članovi sučelja [15].

Sučelja u Reactu su iznimno bitna za definiranje kojih tipova će biti *propovi* koje primamo. Primjer možemo vidjeti na prijašnjoj slici gdje je kreirano sučelje `TestComponentProps` u kojoj je napisano da varijabla “text” treba biti tipa *string*. Ako se proslijedi bilo kakva druga vrijednost, naziva ili tipa, TypeScript će javiti grešku. To pomaže ukloniti greške tijekom pisanja koda te se time smanji vrijeme koje bi bilo utrošeno pri pokušaju otkrivanja mjesta nastanka problema unutar koda.

#### 2.4.4 Tailwind

Tailwind CSS je prvi uslužni CSS okvir (engl. *interface*) dizajniran kako bi omogućio korisnicima brže i lakše kreiranje aplikacije. Mogu se koristiti uslužne klase za upravljanje izgledom, bojom, razmakom, tipografijom, sjenama i dr., kako bi se stvorio, u potpunosti prilagođen, dizajn komponente. Sve to se radi bez napuštanja HTML-a ili pisanja prilagođenog CSS-a. Prilikom sastavljanja projekta (engl. *compiling*), sve upotrijebljene klase se postavljaju u glavnu .css datoteku, a sve neupotrijebljene klase se pročišćuju (engl. *purging*) kako bi se smanjila veličina paketa programa [16]. Logo Tailwindcssa je prikazan na slici 13.



Slika 13: Tailwindcss logo

Tailwindcss postaje industrijski standard za pisanje CSS-a zbog svoje jednostavnosti i praktičnosti. Mnogi *developeri* imaju problema s imenovanjem svojih klasa unutar projekta s obzirom na to da, ukoliko ih ima mnogo, proces brzo izmakne kontroli. Iako Tailwind na prvu izgleda vrlo neorganizirano, većina korisnika se, nakon početne prilagodbe, ne vraća drugim CSS okvirima.

Uz sve ove prednosti, Tailwindcss ima i pseudoklase poput: Hover, Focus, Group, Dark i drugih. Svi oni čine *developersko iskustvo* jednostavnijim preuzimajući komplicirane radnje poput prelaska kursora preko komponente tako što dodaju jednostavan prefiks “hover:” na bilo koju klasu.

#### 2.4.4.1 Instalacija

U ovom projektu je korišten Tailwindcss v.3.0.24, a instalacija je tekla u opisanim koracima:

- 1 Instalirati Tailwindcss - `npm install -D tailwindcss` „npx tailwindcss init
- 2 Konfigurirati putanje do datoteka – unutar kreirane `tailwind.config.js` datoteke postaviti putanje do direktorija u kojima su datoteke kojima će se koristiti Tailwind.
- 3 Nadodati `@tailwind` direktorije – unutar glavne `.css` datoteke na vrhu napisati sljedeće linije koda: `@tailwind base; @tailwind components; @tailwind utilities;`

4 Pokrenuti CLI proces – unutar terminala upisati sljedeću komandu: `npx tailwindcss -i ./src/input.css -o ./dist/output.css --watch`

Konfiguracijska datoteka Tailwindcssa za ovaj projekt je prikazana na slici 14. Gdje content ključ označava niz putanja u kojima se koristi Tailwindcss. Unutar theme ključa su postavke za ograničavanje Tailwindcssa ili nadodavanje funkcionalnosti. Ovdje se nadodaju novi fontovi. Kod plugins ključa je niz u kojem se mogu nadodati dodaci za Tailwind.

```
1  module.exports = {
2    mode: "jit",
3    content: [
4      "./src/**/*.{js,ts,jsx,tsx}",
5      "./node_modules/tw-elements/dist/js/**/*.js",
6    ],
7    theme: {
8      extend: {
9        fontFamily: {
10         roboto: ["Roboto", "sans-serif"],
11         robotoMono: ["Roboto Mono", "monospace"],
12         montserrat: ["Montserrat", "sans-serif"],
13         inter: ["Inter", "sans-serif"],
14         poppins: ["Poppins", "sans-serif"],
15       },
16     },
17   },
18   plugins: [require("tw-elements/dist/plugin")],
19 };
```

Slika 14: tailwind.config.js

#### 2.4.4.2 Primjer

Na slici 15 je prikazan izgled tipke koja će se izgraditi.



Slika 15: Primjer slike

Koristeći klasični CSS kao što je prikazano na slici 16, vidljivo je da je ovako napisan CSS poznat i da funkcioniра. Ipak, on zahtjeva kreiranje nove datoteke, povezivanje na tu datoteku i, najkompliciranije od svega, osmišljavanje imena za klasu. Imenovanje klasa nije problem kada postoji samo jedna tipka koja se zove *button*, no ukoliko postoji pedeset različitih tipki u programu, imenovanje tih varijacija postaje iznimno iscrpno.

```
1  .button {
2    padding-left: 0.5rem;
3    padding-right: 0.5rem;
4    padding-top: 0.75rem;
5    padding-bottom: 0.75rem;
6    font-weight: 700;
7    background-color: #rgb(233 213 255);
8    border-width: 2px;
9    border-radius: 0.25rem;
10   cursor: pointer;
11   border-color: #rgb(207 250 254);
```

Slika 16: Tipka primjer CSS

Primjer Tailwindcssa korištenog za kreiranje identične tipke prikazano je na slici . Iako na prvi pogled izgleda komplicirano, jasno je što se događa unutar koda. U eksternim klasama se ne treba ništa imenovati ni dijeliti, već se zapamti nekoliko jasnih imena klasa.

```
1  const ButtonExample = () => {
2    return (
3      <div className="px-2 py-3 font-bold #bg-purple-200 border-2 rounded cursor-pointer #border-cyan-100">
4        | Click me
5      </div>
6    );
7  };
```

Slika 17: Tipka primjer tailwindcss

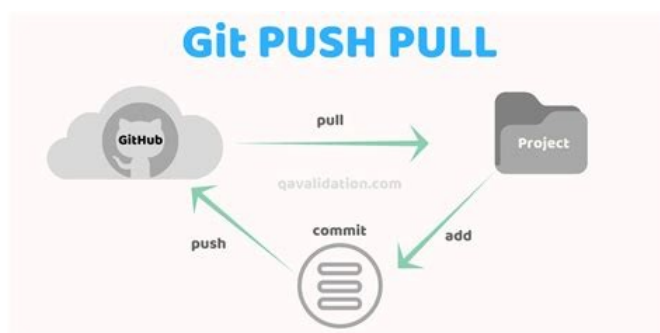
Uz sve prednosti koje Tailwind pruža, postoje i ekstenzije koje automatski predlažu imena klasa dok se one unose pa se tako još više pojednostavljuje cijeli proces.

## 2.4.5 Git

Git je distribuirani sustav kontrole verzija. To znači da je lokalni klon projekta cjelovito spremište kontrola verzija. Ova potpuno funkcionalna lokalna spremišta olakšavaju rad izvan mreže ili na daljinu. Programeri predaju svoj rad lokalno, a zatim sinkroniziraju svoju kopiju pohranjenu u spremištu s kopijom na poslužitelju. Gitova fleksibilnost i popularnost čine ga odličnim izborom za svaki tim. Mnogi iskusni programeri, ali i studenti i diplomanti, znaju koristiti Git. On se zato smatra općim znanjem svakoga u industriji. Gitova korisnička zajednica stvorila je razne tečajeve i izvore za obuku programera, a njegova popularnost olakšava pronalazak pomoći kada je ona potrebna. Većina razvojnih okruženja imaju podršku za Git i Git alate koji se pokreću na svakom većem operativnom sustavu [17].

Primjer jednostavnog tijeka rada za Git projekt je prikazan na slici 18. Prikazano je odvijanje nekoliko radnji:

- 1 Kreirati Git projekt s komandom: `git init`
- 2 Napraviti promjene na kodu u projektu, te ga nadodati u Git s komandom: `git add .`
- 3 Postaviti te promjene (engl. *commit*) u Gitu uz poruku s komandom: `git commit -m 'Vaša poruka'`
- 4 Gurnuti promjene na neki od Git internetskih poslužitelja kao što je GitHub s komandom: `git push`
- 5 Nakon potonjih koraka moguće je povući sve promjene koda s komandom: `git pull` s bilo kojeg drugog računala



Slika 18: Git glavne operacije

Ovaj projekt je na GitHubu te se može pronaći na sljedećem linku:  
<https://github.com/dlt98/nft-marketplace>.

GitHub je pružatelj internetskog hostinga za razvoj softvera i kontrolu verzija pomoću Gita. Nudi distribuiranu kontrolu verzija i funkcionalnost upravljanja izvornim kodom Gita, uz vlastite funkcionalnosti. Omogućuje i kontrolu pristupa te nekoliko funkcionalnosti suradnje kao što su praćenje pogreški u kodu (engl. *bug*), zahtjevi za značajkama, upravljanje zadacima, kontinuirana integracija i wikiji za svaki projekt [18].

## 3. Backend

Kod kreiranja bilo kakvog programa, vrlo je bitno imati dobru bazu i strukturu mapa uz dobro i čisto napisan kod. Ako su mape loše strukturirane i odvojene, projekt, što se više gradi, postaje eksponencijalno kompliciraniji za upravljanje.

Unutar projekta će se spomenuti trikovi koji uvelike pomažu kod strukturiranja i pisanja pravilnog koda.

### 3.1 Kreiranje i postavljanje

Pomoću alata `create-react-app` moguće je kreirati cijeli React projekt koristeći se jednom komandom. Moguće je direktno nadodati TypeScript koristeći sljedeću komandu:

```
npx create-react-app ime-projekta --template typescript
```

Nakon kreiranja projekta, `create-react-app` će kreirati projekt koji je strukturiran s idućim mapama:

- `Node_modules` - mapa koja sadrži sve potrebne biblioteke koje su potrebne u projektu
- `Package.json` - datoteka u JSON formatu koja sadrži imena biblioteka i njene verzije
- `Package-lock.json` - datoteka u JSON formatu koja sadrži imena i verzije podpaketa biblioteka koji se koriste u projektu
- `Public` - mapa u kojoj su datoteke koje će web preglednik koristiti za prikaz stranice
- `README.md` - kratak opis koji opisuje projekt, a može sadržavati tekst, slike i dijelove koda



- `src` - izvorni direktorij koji sadrži sve datoteke koje će se koristiti u Reactu
- `tsconfig.json` - datoteka u JSON formatu koja pruža opcije za konfiguriranje TypeScripta.

### 3.1.1. Hardhat

Nakon korištenja komande za inicijaliziranje, Hardhat projekt, kao što je spomenuto prije (`npm hardhat`), stvara četiri nove mape i jednu datoteku:

- 1 `artifacts` - mapa koja ima sve informacije koje su potrebne za implementaciju i interakciju s pametnim ugovorima ugovorom.
- 2 `cache` - mapa koja sadrži spremljene vrijednosti (engl. *cached values*) koje su potrebne za Hardhat
- 3 `contracts` - mapa u kojoj *developer* postavlja kreirane datoteke ugovora
- 4 `test` - mapa gdje svi testovi vezani za pametne ugovore stoje
- 5 `hardhat.config.js` - datoteka koja sadrži sve opcije potrebne za podešavanje Hardhata

Prikaz `hardhat.config.js` se može vidjeti na slici 19. Unutar ove datoteke se postavlja verzija Solidity programskog jezika koja će se koristiti u pametnim ugovorima. Postavljaju se i putanje do različitih mapa koje koristi Hardhat.

```

1  require("@nomiclabs/hardhat-waffle");
2
3  module.exports = {
4    solidity: "0.8.4",
5    paths: {
6      artifacts: "./artifacts",
7      sources: "./contracts",
8      cache: "./cache",
9      tests: "./test",
10   },
11 };

```

Slika 19: hardhat.config.js

### 3.1.2 Solidity

Za započinjanje pisanja pametnih ugovora, potrebno je prethodno instalirati ekstenziju na VS Code (Visual Studio Code) koja se zove: “Solidity + Hardhat”. Ekstenzija nadodaje podršku za Solidity programski jezik i pruža uredničku integraciju za Hardhat projekte.

### 3.1.3 Metamask

Kako bi se omogućilo povezivanje Metamaska na lokalni blockchain, potrebno je konfigurirati prilagođenu mrežu. Kad je Metamask otvoren, u gornjem desnom kutu je ikonica računala, a pritiskom na tu ikonu će se otvoriti izbornik u kojem treba ući u postavke (engl. *settings*). Prilikom ulaska u postavke potrebno je otvoriti opciju mreža (engl. *networks*), zatim nadodati mrežu (engl. *add a network*). Idući korak je ubacivanje vrijednosti koje su potrebne za povezivanje s Hardhat čvorom [19]. Opcije za konfiguraciju su prikazane na slici 20.

**Network Name**

**New RPC URL**

**Chain ID** ⓘ

**Currency Symbol**

**Block Explorer URL** (Optional)

Slika 20: Nadodavanje nove mreže

### 3.1.3.1 Pokretanje

Za pokretanje Hardhat projekta potrebno je pokrenuti naredbu: `npx hardhat node` koja će kreirati čvor na lokalnom blockchainu na koji će se povezati. Ova naredba će se redovito koristiti pa je dobro napraviti skriptu unutar `package.json` datoteke. Nova skripta se može nadodati kreiranjem nove stavke u objektu “scripts” koja se nalazi u `package.json`. Komanda se može vidjeti na liniji devet, na slici 21. Nakon kreiranja skripte, u konzolu se upiše: `npm run create-node`, i `node.js` će pokrenuti tu komandu.

```
1 {
2   "name": "nft-marketplace",
3   "version": "0.1.0",
4   "private": true,
5   "scripts": {
6     "start": "react-scripts start",
7     "build": "react-scripts build",
8     "prettier": "npx prettier --write 'c",
9     "create-node": "npx hardhat node",
10    "deplov-contracts": "npx hardhat run
```

Slika 21: `package.json` nova skripta

Zatim, nakon pokretanja prethodne skripte i ukoliko je sve uspješno postavljeno, prikazat će se lista testnih računa koji se mogu koristiti na testnom blockchainu. Na svakom je 10,000 Ethera, no oni nemaju nikakvu monetarnu vrijednost, već su samo testni tokeni. Moguće je u Metamask nadodati kopirajući privatni ključ, a zatim ga uvesti (engl. *import*) jednom kad je otvoren Metamask.

## 3.2 Pametni ugovori

Za potrebe ovog rada, unutar projekta će biti kreirana dva pametna ugovora te će ujedno biti ubačeni i drugi pametni ugovori otvorenog koda. Ugovori su od OpenZeppelina, a kreirat će se:

1 NFT.sol

2 Marketplace.sol

### 3.2.1 OpenZeppelin

OpenZeppelin pruža sigurnosne proizvode za izgradnju, automatizaciju i upravljanje decentraliziranim aplikacijama. Također, štite vodeće organizacije obavljanjem sigurnosnih revizija na njihovim sustavima i proizvodima [20].

OpenZeppelin ima ogromnu kolekciju malih pametnih ugovora koji se mogu upotrijebiti u drugim pametnim ugovorima koje korisnici koriste kako bi se zaštitili od napadača ili krivo napisanog koda.

Najpoznatiji su:

- ERC721URIStorage.sol – sadrži dodatne funkcije koje su testirane i funkcioniraju.
- ReentrancyGuard.sol – pametni ugovor koji je modifikator (nadodaje funkcionalnost) na funkciju i sprječava da napadač pozove ugovor i funkciju dok je još unutar pametnog ugovora.

### 3.2.2 NFT.sol

Prvi pametni ugovor koji će se kreirati je NFT.sol. Ovaj pametni ugovor je zadužen za NFT kreiranje ili kovanje (engl. *minting*). Vrlo je jednostavan te je prikazan na slici 22.

```

1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.4;
4
5 import "@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";
6
7 contract NFT is ERC721URIStorage {
8     uint256 public tokenCount;
9
10    constructor() ERC721("Davids NFT", "DNFT") {}
11
12    function mint(string memory _tokenURI) external returns (uint256) {
13        tokenCount++;
14        _safeMint(msg.sender, tokenCount);
15        _setTokenURI(tokenCount, _tokenURI);
16        return (tokenCount);
17    }
18 }

```

Slika 22: NFT.sol pametni ugovor

Svaki pametni ugovor unutar sebe ima ključnu riječ `contract` u kojoj stoji sva logika ugovora. Ujedno je potrebno napraviti konstruktor, tj. funkciju koja se pokreće na prvo pokretanju koda.

Kod pete linije se uvozi pametni ugovor za ERC721 od OpenZeppelina te se zatim primjenjuje na trenutni ugovor kod sedme linije s ključnom riječi `is` koja označava da ovaj pametni ugovor nasljeđuje sve funkcionalnosti uvezenog pametnog ugovora.

Funkcija `mint` prima kao parametar `_tokenURI` koji će biti JSON u obliku teksta, a on će sadržavati link na sliku, ime i opis. Unutar funkcije se poveća `tokenCount` za jedan i zatim se pozove `_safeMint` funkcija, koja je naslijeđena od OpenZeppelina, a u kojoj se prosljeđuje adresa pozivatelja funkcije i `tokenCount`. Nakon što je izvršen `_safeMint` poziv, prosljeđuje se `_tokenURI` u tu funkciju s brojem tokena.

### 3.2.3 Marketplace.sol

U ovom pametnom ugovoru, sastavljenom od mnogo manjih dijelova, nalazi se cijela logika iza trgovine. Funkcionalnost ovog pametnog ugovora je dopustiti korisnicima kreiranje, kupovanje i prodaju. Prilikom korištenja trgovine ujedno se postavlja postotak

koji će biti naplaćen kako bi trgovina bila profitabilna. Pritom svaka NFT trgovina uzme udio transakcije.

Na početku pametnog ugovora se uvozi od OpenZeppelina dva pametna ugovora - IERC721.sol i ReentrancyGuard.sol. Oba će se koristiti unutar koda kako bi se dobile već testirane funkcionalnosti koje provjereno rade.

### 3.2.3.1 Struct

Unutar Solidityja moguće je kreirati svoj tip podataka, koristeći ključnu riječ `struct`, a zatim ime novog tipa podataka. Unutar ovog pametnog ugovora se kreira tip podataka pod imenom `Item`, koji sadržava razne vrijednosti kao što je prikazano na slici 23

```
16 struct Item {
17     uint256 itemId;
18     IERC721 nft;
19     uint256 tokenId;
20     uint256 price;
21     address holder;
22     address payable seller;
23     bool sold;
24 }
```

Slika 23: Primjer proizvoljne strukture  
podataka

### 3.2.3.2 Event

Događaj (engl. *event*) je nasljedni član ugovora. Događaj se emitira i pohranjuje proslijeđene argumente u dnevnik transakcija (engl. *transaction log*) koji su pohranjeni na blockchainu. Dostupni su pomoću adrese ugovora sve dok je ugovor prisutan na blockchainu. Generirani događaj nije dostupan unutar ugovora, pa čak ni onome koji ih je stvorio i emitirao.

Event se može kreirati korištenjem ključne riječi `event` i zatim se unosi ime događaja. Primjer je prikazan na slici 24. Poziva se koristeći `emit` ključnu riječ i prosljeđene vrijednosti koje su definirane u slici.

```
26     event Offered(  
27         uint256 itemId,  
28         address indexed nft,  
29         uint256 tokenId,  
30         uint256 price,  
31         address indexed seller  
32     );
```

Slika 24: Primjer događaja

### 3.3 Značajne funkcije

U ovom pametnom ugovoru postoji pet funkcija:

- 1 `makeItem` – funkcija zadužena za kreiranje NFT-a i njegovo postavljanje na trgovinu;
- 2 `purchaseItem` – funkcija zadužena za kupovinu NFT-a s trgovine i postavljanje u novčanik kupca;
- 3 `sellItem` - slična funkciji za kupovinu, no modificirana tako da stavlja NFT natrag na trgovinu;
- 4 `fetchMyNfts` – funkcija koja vraća listu svih NFT-ova koje pozivatelj posjeduje;
- 5 `getTotalPrice` – funkcija koja vraća ukupnu cijenu NFT-a uz postotak kojeg trgovina uzima.

#### 3.3.1 `MakeItem`

Unutar ove funkcije korisnici će poslati adresu svog NFT-a, `tokenId` i cijenu. Cijela funkcija je prikazana na slici 25.



```

60 //Creating item to be placed in marketplace
61 function makeItem(
62     IERC721 _nft,
63     uint256 _tokenId,
64     uint256 _price
65 ) external nonReentrant {
66     require(_price > 0, "Price must be greater than zero");
67
68     itemCount++;
69
70     //Transfer the NFT to the marketplace
71     _nft.transferFrom(msg.sender, address(this), _tokenId);
72
73     //Add new item to items mapping
74     items[itemCount] = Item(
75         itemCount,
76         _nft,
77         _tokenId,
78         _price,
79         address(this),
80         payable(msg.sender),
81         false
82     );
83
84     emit Offered(itemCount, address(_nft), _tokenId, _price, msg.sender);
85 }

```

*Slika 25: MakeItem funkcija*

Na šezdeset i petoj liniji se vidi modifikator za funkciju `nonReentrant` koja sprječava pozivatelja funkcije da pozove istu funkciju dok se prva izvršava. Nakon što se prođe `require` koji provjerava ako je poslana vrijednost za cijenu veća od 0, na liniji sedamdeset i jedan odvija se uvezena funkcija nad proslijeđenim NFT-om i prenosi se s računa korisnika (`msg.sender`) na račun trgovine (`address(this)`). Nadalje, kreira se `Item` tip varijable i postavlja se u mapping svih `Item` objekata u pametnom ugovoru (u drugim programskim jezicima se zove niz (engl. *array*)). Na kraju, na osamdeset i četvrtoj liniji, odašilje se događaj s vrijednostima ove funkcije.

### 3.3.2 PurchaseItem

Ova funkcija je zadužena za kupovinu NFT-eva s trgovine, a prikazana je na slici 26.

```

87     function purchaseItem(uint256 _itemId) external payable nonReentrant {
88         uint256 _totalPrice = getTotalPrice(_itemId);
89         Item storage item = items[_itemId];
90         require(_itemId > 0 && _itemId <= itemCount, "item doesn't exist");
91         require(
92             msg.value >= _totalPrice,
93             "not enough ether to cover item price and market fee"
94         );
95         require(!item.sold, "item already sold");
96
97         //pay seller and feeAccount
98         item.seller.transfer(item.price);
99         feeAccount.transfer(_totalPrice - item.price);
100
101         //update item sold
102         item.sold = true;
103
104         item.holder = msg.sender;
105
106         //transfer nft to buyer
107         item.nft.transferFrom(address(this), msg.sender, item.tokenId);
108
109         items[_itemId] = item;
110
111         //emit Bought event
112         emit Bought(
113             _itemId,
114             address(item.nft),
115             item.tokenId,
116             item.price,
117             item.seller,
118             msg.sender
119         );
120     }

```

Slika 26: PurchaseItem funkcija

Kao parametar, ova funkcija prima ID NFT-a koji se želi kupiti. Ima modifikator payable jer se očekuje da korisnik šalje Ether te nonReentrant modifikator.

Prilikom ulaska u funkciju pozove se dodatna funkcija getTotalPrice kojoj je povratna vrijednost cijena NFT-a uz postotak kojeg uzima trgovina.

Kod bilo koje funkcije u kojoj se rukuje s Etherom, potrebno je biti jako oprezan jer krivo napisana funkcija može uzrokovati štete od stotine milijuna dolara ukoliko takav promet prolazi kroz pametni ugovor i ako se uspije manipulirati. Uvjeti za ovu funkciju se nalaze u kodu od devedesete do devedeset i pete linije koda. U kojem se provjerava ako postoji taj NFT, ako se šalje dovoljno Ethera za platiti NFT i ujedno ako je taj NFT dostupan za prodaju. Ako su svi uvjeti zadovoljeni, poslani Ether se šalje vlasniku NFT-a,

trgovina uzima svoj udio te se NFT šalje kupcu. Na kraju se odošilje događaj da je NFT kupljen.

### 3.4 Testiranje

Jednom kad je pametni ugovor stavljen na blockchain, tamo je zauvijek. To je dvosjekli mač. Ako je napravljen pametni ugovor dobro napisan i odrađuje svoju svrhu, program će „zauvijek“ biti savršeno funkcionalan, bez straha da će doći do ažuriranja koji će nešto promijeniti ili dovesti do prestanka rada. Problem nastaje ako je pametni ugovor loše napravljen i nije testiran. Tada će taj program „zauvijek“ biti loš te će „napadači“ moći iskoristiti svaku manu koda u svoju korist. Generalno, u programiranju je jako bitno testirati svoj kod kako bi svaka iteracija bila dobra i funkcionirala pravilno. Testiranje je pogotovo bitno u pametnim ugovorima.

Kako bi se kreirao test potrebno je postaviti datoteku unutar test-mape (ili u bilo koju drugu mapu koja se odabere - samo je bitno da se u *hardhat.config.js* navede lokacija) koja ima ekstenziju **.test.js** ili **.test.ts**

### 3.5 Test za kreiranje NFT-a

Unutar pametnih ugovora je bitno dobro testirati svaku funkciju. Kao primjer na slici 27 je testirana funkcionalnost kreiranja NFT-a.

```
55 describe("Making marketplace items", () => {
56   beforeEach(async () => {
57     // addr1 mints an nft
58     await nft.connect(addr1).mint(URI);
59
60     //addr1 approves marketplace to spend nft
61     await nft.connect(addr1).setApprovalForAll(marketplace.address, true);
62   });
63
64   it("Should track newly created item, transfer NFT from seller to marketplace and emit Offered event", async () => {
65     await expect(
66       marketplace.connect(addr1).makeItem(nft.address, 1, toWei(1))
67     )
68       .to.emit(marketplace, "Offered")
69       .withArgs(1, nft.address, 1, toWei(1), addr1.address);
70
71     //Owner of NFT should now be the marketplace
72     expect(await nft.ownerOf(1)).to.equal(marketplace.address);
73     //Item count should now equal 1
74     expect(await marketplace.itemCount()).to.equal(1);
75
76     //Get item from items mapping then check fields to ensure they are correct
77     const item = await marketplace.items(1);
78     expect(item.itemId).to.equal(1);
79     expect(item.nft).to.equal(nft.address);
80     expect(item.tokenId).to.equal(1);
81     expect(item.price).to.equal(toWei(1));
82     expect(item.sold).to.equal(false);
83   });
84
85   it("Should fail if the price is set to zero", async () => {
86     await expect(
87       marketplace.connect(addr1).makeItem(nft.address, 1, 0)
88     ).to.be.revertedWith("Price must be greater than zero");
89   });
90 });
```

Slika 27: Test kreiranja NFT-a

Za testiranje pametnih ugovora će se koristiti Chai - iznimno jednostavna i korisna, a jako popularna biblioteka za testiranje unutar Node.js programa [21].

Kada se cjelina nekog koda želi testirati, kreira se `describe` blok koji, kao prvi parametar, prima opis cjelokupne funkcionalnosti (u ovom slučaju na pedeset i petoj liniji, `Making marketplace items`), a kao drugi parametar postavlja se `callback` funkcija koja sadrži specifičnije testove koji se označuju `it` blokom. Unutar ovog testa nalazi `Should track newly created item, transfer NFT from seller to marketplace and emit Offered event` na šezdeset i četvrtoj liniji i `Should fail if the price is set to zero` na osamdeset i petoj liniji.

Ako je potrebno izvršiti neki komad koda, prije svakog testa se može kreirati `beforeEach` funkcija koja će se izvršiti nakon svakog `it` poziva. Unutar ovog `beforeEach`, povezujemo se s dva individualna računa na trgovinu te se zatim testiraju `it` blokovi.

### 3.5.1 Happy testovi

Kod pisanja testova, oni za koje se očekuje da dobro prođu zovu se „sretni“ (engl. *happy*) testovi. Unutar prvog `it` bloka se testira kada je uspješno kreiran NFT.

Ključnom riječi `expect` naznačuje se `Chai` biblioteci što se očekuje. Rezultat se postavlja s nastavkom `.to.emit` i zatim se prosljeđuje `'Offered'` kao parametar. S ovim se testira da će transakcija odaslati događaj koji se postavio unutar pametnog ugovora, prikazano na slici 28. Unutar testa se testiraju još razni slučajevi za provjeru ako se dobije dobar ID, naziv, cijena i ako je prodan (sedamdeset i osma linija do osamdeset i druge linije).

```
64   it("Should track newly created item, transfer NFT from seller to marketplace and emit Offered event", async () => {
65     await expect(
66       marketplace.connect(addr1).makeItem(nft.address, 1, toWei(1))
67     )
68       .to.emit(marketplace, "Offered")
69       .withArgs(1, nft.address, 1, toWei(1), addr1.address);
70
71     //Owner of NFT should now be the marketplace
72     expect(await nft.ownerOf(1)).to.equal(marketplace.address);
73     //Item count should now equal 1
74     expect(await marketplace.itemCount()).to.equal(1);
75
76     //Get item from items mapping then check fields to ensure they are correct
77     const item = await marketplace.items(1);
78     expect(item.itemId).to.equal(1);
79     expect(item.nft).to.equal(nft.address);
80     expect(item.tokenId).to.equal(1);
81     expect(item.price).to.equal(toWei(1));
82     expect(item.sold).to.equal(false);
83   });
```

Slika 28: Happy test

### 3.5.2 Sad testovi

Testovi za koje se očekuje da ne prođu dobro zovu se „tužni“ (engl. *sad*) testovi. Unutar drugog `it` bloka očekujemo neuspjeh transakcija ukoliko je cijena 0. Time

expect dovršavamo s `.toBe.revertedWith` (osamdeset i osma linija, na slici 29) čime Chai sada očekuje da će odgovor na taj poziv u kojem je postavljena cijena da je 0, da će povratna poruka biti jednaka kao ona postavljena u pametnom ugovoru.

```
85     it("Should fail if the price is set to zero", async () => {
86         await expect(
87             marketplace.connect(addr1).makeItem(nft.address, 1, 0)
88         ).toBe.revertedWith("Price must be greater than zero");
89     });
90 }
```

Slika 29: Sad test

Nakon završetka pisanja testova, u konzoli se upiše komanda `npx hardhat test` kako bi Hardhat pronašao sve testove unutar programa i zatim ih izvršio. Ako je sve uspješno provedeno, u konzoli se dobiva rezultat svih testova. Primjer uspješno odrađenih testova nalazi se na slici 28.

```
user@david:~/Documents/Uni/nft-marketplace$ npx hardhat test

NFTMarketplace
Deployment
  ✓ Should track and and symbol of the NFT collection
  ✓ Should track feeAccount and feePercent of marketplace
Minting NFTs
  ✓ Should track each minted NFT (88ms)
Making marketplace items
  ✓ Should track newly created item, transfer NFT from seller to marketplace and emit Offered event (69ms)
  ✓ Should fail if the price is set to zero
Purchasing marketplace items
  ✓ Should update item as sold, pay seller, transfer NFT to buyer, charge fees and emit and Bought event (68ms)
  ✓ Should fail for invalid item ids, sold items and when not enough ther is paid (89ms)

7 passing (2s)
```

Slika 30: Uspješan test

### 3.6 Implementiranje pametnih ugovora

Nakon dovršetka i testiranja svih pametnih ugovora, potrebno ih je ubaciti u blockchain. Kako bi se to ostvarilo, potrebno je kreirati `scripts` mapu unutar koje je kreirana `deploy.js` skripta. Cijela skripta je prikazana na slici 31, a dijeli se na dva dijela:

- 1 Main – glavni dio koda koji pomoću *ethers.js* dohvati ta dva ugovora i postavlja ih na lokalni blockchain
- 2 SaveFrontendFiles – kod koji preuzima dio potrebnih podataka koji će biti korisni prilikom *frontend* dijela projekta te ih sprema u datoteku koja se nalazi u *src/contracts* direktoriju. To je bitno kako bi se te datoteke mogle dohvatiti s *frontend* sučelja.

```
2  async function main() {
3      const [deployer] = await ethers.getSigners();
4
5      console.log("Deploying contracts with the account:", deployer.address);
6      console.log("Account balance:", (await deployer.getBalance()).toString());
7
8      // Get the ContractFactories and Signers here.
9      const NFT = await ethers.getContractFactory("NFT");
10     const Marketplace = await ethers.getContractFactory("Marketplace");
11
12     // deploy contracts
13     const marketplace = await Marketplace.deploy(1);
14     const nft = await NFT.deploy();
15
16     // Save copies of each contracts abi and address to the frontend.
17     saveFrontendFiles(marketplace, "Marketplace");
18     saveFrontendFiles(nft, "NFT");
19 }
20
21 function saveFrontendFiles(contract, name) {
22     const fs = require("fs");
23     const contractsDir = "src/contracts";
24
25     console.log("__dirname", __dirname);
26     if (!fs.existsSync(contractsDir)) {
27         fs.mkdirSync(contractsDir);
28     }
29
30     fs.writeFileSync(
31         contractsDir + `/${name}-address.json`,
32         JSON.stringify({ address: contract.address }, undefined, 2)
33     );
34
35     const contractArtifact = artifacts.readArtifactSync(name);
36
37     fs.writeFileSync(
38         contractsDir + `/${name}.json`,
39         JSON.stringify(contractArtifact, null, 2)
40     );
41 }
```

Slika 31: Implementiranje pametnih ugovora

Nakon kreiranja te datoteke, potrebno ju je i pokrenuti. Zato što će se *deploy.js* datoteka morati pokrenuti više puta je dobro napraviti unutar *package.json* skriptu koja će to napraviti za nas.

Unutar `scripts` objekta u `package.json` se upisuje: `"deploy-contracts": "npx hardhat run scripts/deploy.js --network localhost"`. Zatim se naredbom `npm run deploy-contracts` pametni ugovori postavlja na blockchain.



## 4. Frontend

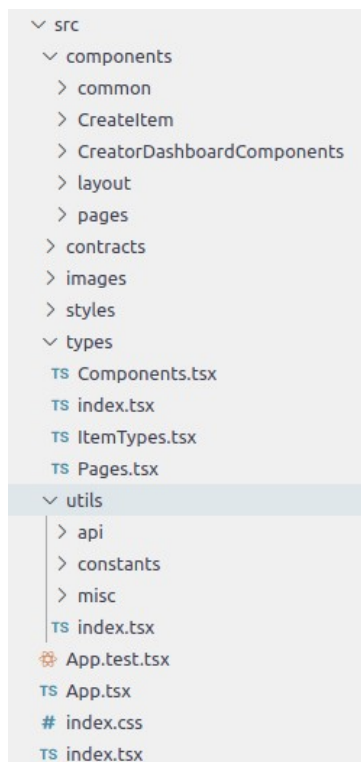
Svaki program mora imati jednostavan način za komunikaciju s logikom programa. Upravo to ima i ovaj. U ovom se projektu koristi React sa TypeScriptom, Tailwindcss za lakše ukrašavanje i ethers.js za komunikaciju s *backendom*.

Glavne stavke u *frontendu* koje će se proći su:

- Struktura mapa
- Navigacija
  - o Sidebar
  - o Container
- Značajne komponente
- Stranice
  - o „Home“
  - o „Create Item“
  - o „MyAssets“
  - o „Creator dashboard“

### 4.1 Struktura mapa

Od neizmjerne je važnosti izrada dobre strukture mapa unutar svakog projekta. U protivnom, kôd postaje težak za održavanje i nespretno je snalaziti se u njemu. Kada se koristi create-react-app, kreirana je `src` mapa koju React može “vidjeti”, dok bilo koja druga datoteka ili mapa izvan `src` nije dobavljiva Reactu. Struktura datoteka je vidljiva na slici 32.



Slika 32: Struktura mapa

Glavni dijelovi strukturirane mape su:

- `Components` mapa - sadrži sve dijelove koje će React koristiti, a unutar njih datoteke dijele na još manje dijelove
- `Contacts` – mapa u kojoj stoje podaci pametnih ugovora koji se nalaze na lokalnom *blockchainu*
- `Styles` – stilovi koji se koriste unutar komponenti
- `Types` – mapa koja postoji isključivo zbog TypeScripta, a sadržava sve tipove koji će biti korišteni unutar projekta

- `Utils` – mapa u kojoj se nalaze razne funkcije i konstante koje će se više puta koristiti unutar projekta
- `App.tsx` - Prva “komponenta” projekta iz koje se stavljaju druge komponente
- `Index.css` - primarna datoteka za stilove
- `Index.tsx` - primarna datoteka koja pokreće cijeli *frontend* dio projekta

Jedan od problema koji može nastati s ovolikim brojem mapa i podmapa je dohvaćanje podataka koji se nalaze unutar njih. Moguće kreirati unutar svake mape `index.ts` koji će pri sastavljanju kôda biti prepoznat kao primarna datoteka u toj mapi. Takav uvoz olakšava daljnji izvoz svih datoteka te mape. Primjer `index.ts` datoteke za `components/common` direktorij je prikazan na slici 33. Prikazan je uvoz svih komponenti u jednu datoteku te izvoz u sklopu jednog objekta. Na prvu izgleda neuredno, iako je u praksi vrlo korisno.

```
src > components > common > TS index.tsx
You, 4 days ago | 1 author (You)
 1  import Spinner from "./Spinner";
 2  import NFT from "./NFT";
 3  import UserAnnouncement from "./UserAnnouncement";
 4  import Headline from "./Headline";
 5  import SelectComponent from "./SelectComponent";
 6  import Copy from "./Copy";
 7  import NewNft from "./NewNft";
 8  import Button from "./Button";
 9  import Label from "./Label";
10  import NftContainer from "./NftContainer";
11  import Modal from "./Modal";
12
13  export {
14      Spinner,
15      NFT,
16      UserAnnouncement,
17      Headline,
18      SelectComponent,
19      Copy,
20      NewNft,
21      Button,
22      Label,
23      NftContainer,
24      Modal,
25  };

```

Slika 33: Primjer uvoza i izvoza

Kod uvoza u datoteku gdje je mnogo datoteka pozvano iz iste mape, moguće je destrukuiranjem pozvati samo one podatke koji u tom trenu potrebni. Primjer se može vidjeti na slici 34. Na četvrtoj liniji koda korištena je metoda destrukuiranja, a ispod su iste komponente uvedene klasičnom metodom uvoza (od šeste do desete linije koda).

```
4  import {Spinner, UserAnnouncement, NewNft, NftContainer, Button } from "../common
5
6  import Spinner from "../common/Spinner";
7  import UserAnnouncement from "../common/UserAnnouncement";
8  import NewNft from "../common/NewNft";
9  import NftContainer from "../common/NftContainer";
10 import Button from "../common/Button";
```

Slika 34: Usporedba metoda uvoza i izvoza

Kod destrukuiranja je kôd puno čišći i jasnije je što se događa. Ukoliko je potrebno još jednu datoteku uvesti iz `common` mape, ona se jednostavno nadoda na kraj. Klasičnom metodom trebala bi se kreirati cijela nova linija i zatim pozvati s prave putanje. Ovaj „trik“ s uvozom i izvozom datoteka se koristi tijekom cijelog projekta.

## 4.2 Navigacija

Kako bi se korisnici mogli navigirati po web stranici potreban im je sustav navigacije. React je inicijalno dizajniran za web stranice sa samo jednom stranicom, no postoje i razna rješenja kako bi se omogućila navigacija i kompleksnijim stranicama. Jedno od rješenja je korištenje biblioteke koja se naziva `react-router-dom`. Koristeći se svojim komponentama, ona omogućava mijenjanje stranica. Web preglednik se ne obnavlja promjenom stranica, već se samo zamijeni jedna komponenta stranice za drugu.

### 4.2.1 React-router-dom

Unutar `App.tsx` datoteke uvoze se potrebne komponente od `react-router-dom` biblioteke: `BrowserRouter`, `Route`, `Routes`. `BrowserRouter` će biti omotač (engl. *wrapper*) oko `Route` komponente koja onda „omotava“ individualne `Routes` komponente te kao parametar prima link i element koji će biti prikazan.

Prikaz `App.tsx`-a je vidljiv na slici 35. Na sto osmoj liniji je vidljivo da glavni element `BrowserRouter` obuhvaća sve druge komponente, a zatim na stoosamnaestoj liniji `Routes` komponenta omotava sve pojedinačne `Route` komponente koje primaju `path`, odnosno poveznicu i komponentu stranice koja se prikazuje.

```

107   return (
108     <BrowserRouter>
109       <Sidebar
110         walletAddress={walletAddress}
111         profileImage={profileImage}
112         ethPrice={ethInfo?.market_data.current_price.usd}
113       />
114     <Container>
115       {isLoading ? (
116         <Spinner label="Awaiting metamask connection..." />
117       ) : (
118         <Routes>
119           <Route
120             path="/"
121             element={
122               <>
123                 <Headline
124                   text="NFT Marketplace"
125                   description="This is the page where you can buy NFT's that users have created."
126                 />
127                 <Home marketplace={marketplace} nft={nft} />
128               </>
129             }
130           />
131           <Route
132             path="/create-item"
133             element={
134               <>--
135             }
136           />
137           <Route
138             path="/my-assets"
139             element={
140               <>--
141             }
142           />
143           <Route
144             path="/creator-dashboard"
145             element={
146               <>--
147             }
148           />
149         </Routes>
150       )}
151     </Modal name="ethModal" title="Ethereum">--
152   </Modal>
153 </Container>
154 </BrowserRouter>

```

Slika 35: Primjer strukture BrowserRouter-a

#### 4.2.2 Izgled i funkcionalnost

Doslovni smisao bočnog izbornika (engl. *sidebar*) govori o smještaju navigacije u web pregledniku. Cilj je napraviti bočnu traku koja ima tipku za svaku stranicu koju korisnici moguće trebaju doseći. Bočna traka mora ostati “fiksirana” i uvijek vidljiva na lijevoj strani. Takav se izgled jednostavno postiže koristeći tailwindcss. Dovršeni izgled koda je vidljiv na slici 34.

```

9 > <nav className="fixed top-0 left-0 flex flex-col w-16 h-screen m-0 text-white bg-cyan-100">--
35 </nav>

```

Slika 36: Sidebar tailwindcss

#### 4.2.2.1 Funkcionalnost

Unutar *Sidebar* komponente je potrebno prikazati različite tipke koje će dopustiti korisnicima samostalnu navigaciju stranicom. Sve tipke su jako slične izgledom. Razlikuju se drugačijim ikonama i tekstem. Primamljivo je kopirati i zalijepiti isti komad kôda te nakon ručno promijeniti dijelove. Ipak, kako bi se pratila dobra struktura i čistoća kôda, kreirano je niz objekata nazvanih `NavItem`.

##### 4.2.2.1.1 NAV\_ITEMS

Unutar mape `src/utils/constants` kreirana je datoteka nazvana `navigation.tsx` u kojoj je kreirana konstanta za navigaciju te tako pojednostavljeno buduće mijenjanje. Kôd je tada potrebno promijeniti samo na jednom. Objekt je prikazan na slici 37.

```
9   export const NAV_ITEMS: NavItemType[] = [
10  {
11    title: "Home",
12    url: "/",
13    icon: home,
14  },
15  {
16    title: "Sell digital asset",
17    url: "/create-item",
18    icon: marketplace,
19  },
20  {
21    title: "My digital asset",
22    url: "/my-assets",
23    icon: wallet,
24  },
25  {
26    title: "Creator dashboard",
27    url: "/creator-dashboard",
28    icon: dashboard,
29  },
30  ];
```

Slika 37: NAV\_ITEMS konstanta

Na vrhu datoteke uvoze se sve ikone koje su preuzete s interneta te se ovdje postavljaju njihove vrijednosti. Može se primijetiti, pored naziva konstante, da je riječ o nizu tipa `NavItemType`, prikazanoj na slici 38.

Unutar `NavItemType` odmah se može vidjeti koje su stavke bitne i kojeg su tipa. Oznaka „?” pored imena, označava da te stavke nisu nužne.

```
4 export interface NavItemType {  
5   title: string;  
6   url: string;  
7   icon: string;  
8   onClick?: (text: string) => void;  
9   profile?: boolean;  
10 }
```

Slika 38: `NavItemType` tip

#### 4.2.2.2 Izgled

Kako bi se mogle korisnicima prikazati vrijednosti koje su postavljene u `NAV_ITEMS` konstanti, potrebno je za to kreirati React komponentu. Cilj je napraviti ikonicu koja će, prilikom dolaska miša na nju, malo promijeniti oblik i boju, a zatim tekst će izaći sa strane te će prikazati korisniku o kojoj je stavki riječ.

##### 4.2.2.2.1 NavItem

Kôd za `NavItem` komponentu je prikazan na slici 37. Može se primijetiti kako cijelu komponentu omotava `Link` komponenta, dok ta komponenta dolazi iz `react-router-dom` biblioteke. Pomoću te `Link` komponente ta biblioteka zna promijeniti na kojoj stranici se trenutno nalazi korisnik.

Kao ulazne vrijednosti se šalju:

- `Url` – tekst; link na koji link stranice će se otvoriti pritiskom na komponentu
- `Title` – tekst; naslov ikone
- `Icon` – tekst; putanja do ikone koja će biti prikazana
- `OnClick` – funkcija; funkcija koja će biti pokrenuta prilikom klika



- Profile – *boolean*; ako je ikona namijenjena da bude profilna slika

```

4  const NavItem = ({
5      url,
6      title,
7      icon,
8      onClick,
9      profile = false,
10 }): NavItemType) => {
11     return (
12         <Link to={url || "#"}>
13             <div className="sidebar-icon group" onClick={() => onClick?.(title)}>
14                 <img
15                     src={icon}
16                     alt={` ${title} icon`}
17                     className={`w-8 h-8 ${profile ? "rounded-xl shadow-lg" : ""}`}
18                 />
19                 <span className="sidebar-tooltip group-hover:scale-100">{title}</span>
20             </div>
21         </Link>
22     );
23 };

```

Slika 39: NavItem kôd

Kako bi se postigao željeni izgled komponente, moguće je unutar same komponente ispisati sve tailwindcss klase. Također, moguće je i kreirati proizvoljne klase te zatim, unutar tih klasa, postaviti tailwindcss koristeći `@apply` ključnu riječ. Korišteni stilovi su prikazani na slici 40.

Ovim postavkama je prikazana navigacija i animacije koje se postignu prilikom prelaska mišem preko pojedinačne ikone.

```

1  .sidebar-icon {
2      /*Positioning*/
3      @apply relative flex items-center justify-center;
4
5      /*Size and margins*/
6      @apply mx-auto mt-2 mb-2 w-14 h-14 rounded-3xl;
7
8      /*Aesthetics*/
9      @apply bg-blue-200 shadow-lg;
10
11     /*Hover effects*/
12     @apply hover:bg-cyan-50 hover:text-white hover:rounded-xl;
13
14     /*Animation*/
15     @apply transition-all ease-linear cursor-pointer;
16 }
17
18 .sidebar-tooltip {
19     @apply absolute z-50 w-auto p-2 m-2 text-xs font-bold text-white transition-all origin-left scale-0 bg-gray-900 rounded-md shadow-md min-w-max left-14;
20 }

```

Slika 40: Tailwindcss primjer za `@apply`

Završeni izgled individualne komponente uz dodatnu animaciju kada je miš na tipki je prikazan na slici 41.



Slika 41: Izgled NavItem komponente

### 4.2.3 Prikaz

Kako bi se iskombinirali svi aspekti, od funkcionalnosti do izgleda pojedinačne ikone unutar *sidebar* komponente, uvozi se kreirana konstanta, te NavItem komponenta koja će prikazati te vrijednosti. Za korištenje JavaScripta unutar *sidebara* potrebno je koristiti jsx funkcionalnost te ubaciti kôd koristeći vitičaste zagrade. Upravo u tome leži moć jsx-a -on olakšava posao ubacivanja koda unutar HTML-a.

Kreirana konstanta je niz, pa je tako moguće iskoristiti map funkciju koju ima svaki niz. Map funkcija uzima svaki element u nizu i dopušta manipuliranje s njim, a povratna vrijednost je niz manipuliranih vrijednosti. To se može iskoristiti ubacivanjem kreiranog NavItema tako da on bude povratna vrijednost. Time se prikazuju sve ikone sa svojim vrijednostima. Prikaz jsx-a je na slici 42. Unutar map funkcije se dobije pojedinačan element, te se jednostavno popuni NavItem komponenta sa svim vrijednostima u mapi.

```
8   return []
9   <nav className="fixed top-0 left-0 flex flex-col w-16 h-screen m-0 text-white bg-cyan-100">
10     {NAV_ITEMS.map((navItem, idx) => (
11       <NavItem
12         title={navItem.title}
13         url={navItem.url}
14         icon={navItem.icon}
15         key={idx}
16       />
17     )
18   )}
```

Slika 42: Prikaz konstante u kôdu

Konačan izgled cijele navigacije je vidljiv na slici 43.



Slika 43: Konačan izgled navigacije

#### 4.2.4 Container

Postavljena konstantna navigacija na lijevom rubu zaslona stvara probleme sa stranicama jer ju ostali elementi ne registriraju, tj. „ne prepoznaju“ na tom mjestu. Potrebno je, stoga, kreirati element koji će obuhvatiti sve stranice tako da ne dolazi do preklapanja s bočnom navigacijom.

Kreirani okvir (engl. *container*) je vrlo koristan jer omogućava primjenu raznih stilova koji će biti primijenjeni na svim stranicama koje se nalaze unutar te komponente. Prikaz kôda `Container` komponente je prikazan na slici 41. Koristeći `p1-20` klasu,

postavlja se odmak od lijeve strane ekrana za 5rem ili 80px te se tako sprječava poklapanje navigacije i ostatka stranice.

```
3  const Container: FC<{ children: ReactNode }> = ({ children }) => {  
4    return (  
5      <div className="flex flex-col items-center justify-start h-screen p-5 pb-10 pl-20 mx-auto overflow-auto max-w-7xl">  
6        {children}  
7      </div>  
8    );  
9  };
```

Slika 44: Container kôd

## 4.3 Značajne komponente

React je od neprocjenjive važnosti jer posjeduje komponente koje se mogu iskoristiti beskonačno mnogo puta unutar projekta. Teoretski cijeli projekt bi trebao biti gomila ponovno iskoristivih komponenti. Unutar ovog projekta nalazi se nekoliko bitnih komponenti koje bi se trebale istaknuti, a to su:

- **Headline**
- **NFT**

### 4.3.1 Headline komponenta

Kako bi korisnici mogli vidjeti na kojoj se stranici nalaze i kako bi se dalje mogli informirati, potrebno je kreirati komponentu koja će biti pozicionirana na vrhu svake stranice. Tako, ukoliko se prilikom prelaska miša preko teksta korisnik zaustavi nad jednim dijelom, otvori se malen blok teksta koji opisuje stranicu.

Kôd komponente je prikazan na slici 45. Problematična stvar ovakvog izgleda je pitanje izrade reaktivnog dječjeg elementa na lebdenje (engl. *hover*) iznad roditeljskog elementa. Zahvaljujući tailwindcssu nudi korištenje `group` klase, to neće biti teško.

```

3  const Headline = ({ text, description }: HeadLineProps) => {
4    return (
5      <div className="relative mx-auto my-2 mb-6 transition-all border-b-2 group w-max">
6        <h1 className="text-5xl font-medium font-poppins">{text}</h1>
7        <p className="absolute right-0 z-50 p-2 font-bold text-white transition-all origin-top scale-0 rounded-md bg-slate-500 top-14 text-md group-hover:scale-100">
8          {description}
9        </p>
10     </div>
11   );
12 };

```

Slika 45: Headline komponenta kôd

Postavljanje klase `group` na roditeljski element, omogućuje da bilo koji dječji element može koristiti pseudo klasu `group-` uz gotovo bilo koji modifikator i postići željeni efekt. Na samom kraju klase, na paragraf elementu (podebljano) postavljen je `group-hover:scale-100` koji povećava veličinu tog elementa s početne vrijednosti od 0% na 100%. Uz klasu `transition-all` se odvija uz lijepu animaciju koja daje efekt da tekst izlazi iz samog naslova.

Prikaz izgleda komponente je vidljiv na slici 46



Slika 46: Headline izgled

### 4.3.2 NFT komponenta

Korisnicima je nakon kreiranja svog NFT-a i postavljanja na blockchain jako bitno da mogu odmah i vidjeti kreirani NFT.

*Propovi* ove komponente čini primanje svih vrijednosti spremljenih u NFT-u i njihov prikazati na lijep način. To se obavlja zahvaljujući animaciji `tailwindcss` i posebnih ukrasa koji nisu teški za implementirati.

Ulazne vrijednosti su:

- Image – tekst; link na sliku
- Name – tekst; ime NFT-a

- Price – tekst; cijena NFT-a
- Collection – tekst; ime kolekcije
- Description – tekst; opis NFT-a
- OnClick – funkcija; funkcija koja će se pozvati kad se klikne na tipku
- PriceText – tekst; naslov cijene koja je prikazana
- ButtonText – tekst; tekst koji će biti prikazan na tipki

Kôd cijele komponente je prikazan na slici 47

```

4  const NFT = ({image, name, price, collection = "Collection", description, onClick, priceText, buttonText}: NFTProps) => {
5  return (
6    <div className="p-1 transition-all border-2 rounded-lg hover:translate-y-1 max-w-[300px] w-max group hover:shadow-2xl">
7      <div className="relative max-w-sm overflow-hidden rounded-lg">
8        <img
9          src={image}
10         alt="NFT img"
11         className="w-full h-full transition-all ease-in-out group-hover:scale-110"
12       />
13      <div
14        className="absolute top-0 bottom-0 left-0 right-0 w-full h-full overflow-hidden transition duration-300 ease-in-out opacity-0 bg-black/70 hover:opacity-100"
15        data-mdb-ripple="true"
16        data-mdb-ripple-color="light"
17      >
18        <div className="m-3">
19          <h3 className="mb-1 text-sm text-white">Description: </h3>
20          <p className="text-sm font-bold md:text-base lg:text-lg text-gray-50">
21            {description}
22          </p>
23        </div>
24      </div>
25    </div>
26    <div className="m-2">
27      <p className="text-xs text-gray-50 font-monsterrat">{collection}</p>
28      <h3 className="mt-1 font-semibold font-monsterrat">{name}</h3>
29    </div>
30    <div className="flex items-center justify-between px-3 py-4 bg-gray-100 rounded-xl">
31      <div>
32        <h3 className="mb-1 text-xs font-medium text-gray-800 font-monsterrat">
33          {priceText || "Price"}
34        </h3>
35        <p className="text-sm font-semibold font-inter">{price} ETH</p>
36      </div>
37      <div>
38        <buttonText && {
39          <div className="transition-all origin-right scale-0 group-hover:scale-100">
40            <button text={buttonText} onClick={onClick} />
41          </div>
42        }
43      </div>
44    </div>
45  );
46  };

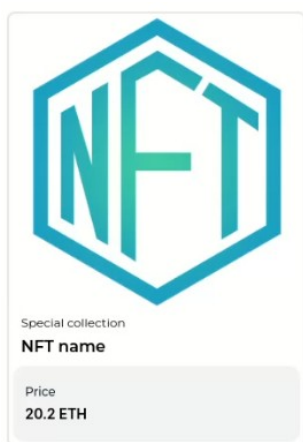
```

Slika 47: NFT komponenta kôd

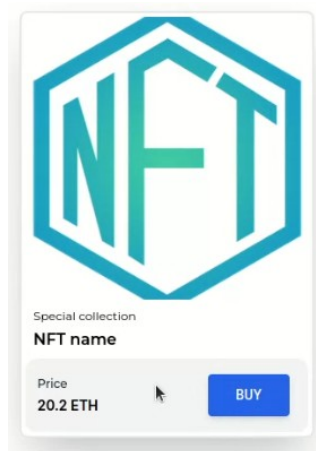
zapravo, nuspojava korištenja tailwindcssa. Unutar ove komponente se isto koristi `group` i `group-hover`.

Na četrdeset i sedmoj liniji koda se mogu primijetiti dupli uskličnici i dupli „&“ simboli. Dupli uskličnici govore JavaScriptu da pretvori tu vrijednost u *boolean* (istina ili laž) – ako je poslan tekst za tipku, bit će istina, a ako nije će biti laž. Dupli „&“ simboli označuju da je riječ o AND operatoru, koji će se odvititi isključivo ako su svi uvjeti istiniti. Dakle, u ovom slučaju će tipka biti prikazana samo ako je poslan tekst za tipku.

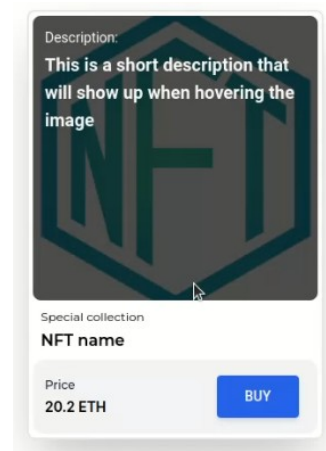
Gotov izgled komponenti i varijacije su prikazane na slikama: 48, 49 i 50.



Slika 48: NFT komponenta, normalno



Slika 49: NFT komponenta, lebdenje na kartici



Slika 50: NFT komponenta, lebdenje na slici

#### 4.4 Inicijalizacija pametnih ugovora

Na početku, prije učitavanja bilo koje stranice, potrebno je inicijalizirati pametne ugovore. Unutar App.tsx je prvo potrebno postaviti konekciju s Metamask digitalnim novčanikom. Metamask, prilikom učitavanja stranice, prepoznaje da stranica sadrži web3 funkcionalnosti i ubaci objekt u globalnu varijablu window. Adresa se zatim dohvaća putem tog objekta i postavlja u useState za adresu novčanika. Potonje opisan postupak je vidljiv na slici 51.

```
44 //Metamask get wallet address
45 const connectMetamask = async () => {
46   const accounts = await window.ethereum.request({
47     method: "eth_requestAccounts",
48   });
49   setWalletAddress(accounts[0]);
50 };
```

Slika 51: Dohvaćanje Metamask adrese

#### 4.4.1 Ethers.js

Prilikom otvaranja stranice, Metamask će upitati korisnika ukoliko želi spojiti svoj digitalni novčanik sa stranicom. Prihvatanjem tog upita inicijaliziraju se pametni ugovori, prikazano na slici 52.

```
78   const web3provider = async () => {
79     // Get provider from Metamask
80     const provider = new ethers.providers.Web3Provider(window.ethereum);
81
82     //Get signer
83     const signer: ethers.providers.JsonRpcSigner = provider.getSigner();
84
85     loadContracts(signer);
86   };
87
88   const loadContracts = async (signer: ethers.providers.JsonRpcSigner) => {
89     // Get deployed copies of contracts
90     const marketplace = new ethers.Contract(
91       MarketplaceAddress.address,
92       Marketplace.abi,
93       signer
94     );
95     setMarketplace(marketplace);
96     const nft = new ethers.Contract(NFTAddress.address, NFT.abi, signer);
97     setNft(nft);
98     setIsLoading(false);
99   };
```

Slika 52: Dohvaćanje pametnih ugovora

Ethers.js ovdje dohvaća podatke iz digitalnog novčanika kako bi se pametni ugovor mogao inicijalizirati (od osamdesete do osamdeset i treće linije), a zatim pozove `loadContracts` funkciju i proslijedi `signer` objekt. Koristeći vrijednosti koje su prilikom implementiranja pametnih ugovora u lokalni blockchain bile spremljene u `src/contracts`, kreiran je objekt pomoću kojeg se pristupa funkcijama u pametnim ugovorima.

#### 4.5 Stranice

Unutar ovog projekta postoje četiri stranice:

- 1 Home - početna stranica koja ujedno služi i kao trgovina
- 2 CreateItem – stranica u kojoj korisnici mogu kreirati svoj NFT



- 3 MyAssets – stranica na kojoj će korisnici moći vidjeti svoje kupljene NFT-eve
- 4 CreatorDashboard – stranica na kojoj će korisnici moći vidjeti prodane i kreirane NFT-eve, te mogu promijeniti izgled svoje profilne slike

Svaka stranica je kao *prop* primila *NFT* i *marketplace* pametne ugovore koji su prije bili inicijalizirani prilikom učitavanja stranice.

#### 4.5.1 Home

Početna stranica, tj. engl. *home*, je prva stranica koju korisnici vide otvaranjem web stranice. Unutar nje je moguće vidjeti sve izlistane NFT-eve koji su postavljeni na trgovini i moguće ih kupiti.

##### 4.5.1.1 Prikazivanje

Unutar ove komponente je potrebno dohvatiti sve NFT-eve koji su trenutno na trgovini. Kôd koji dohvaća NFT-eve je prikazan na slici 53.

```

20 const loadMarketplaceItems = async () => {
21   const itemCount: number = await marketplace.itemCount();
22
23   let tempItems: NFTtype[] = [];
24   for (let i = 1; i <= itemCount; i++) {
25     const item = await marketplace.items(i);
26     if (!item.sold) {
27       //get uri from nft contract
28       const uri = await nft.tokenURI(item.tokenId);
29
30       //use uri to fetch the nft metadata store on ipfs
31       const res = await fetch(uri);
32
33       const metadata = await res.json();
34       //get total price of item (item price + fee)
35       const totalPrice: string = await marketplace.getTotalPrice(item.itemId);
36
37       tempItems.push({
38         totalPrice,
39         itemId: item.itemId,
40         seller: item.seller,
41         name: metadata.name,
42         description: metadata.description,
43         image: metadata.image,
44       });
45     }
46   }
47
48   setItems(tempItems);
49   setIsLoading(false);
50 };

```

Slika 53: Učitavanje NFT-eva

Koristeći marketplace ugovor koji je prosljeđen, dobavlja se informacija o količini stavki u trgovini (vidljivo na liniji dvadeset i osam). Nadalje, napravi se for petlja koja povlači iz marketplace pametnog ugovora sve stavke koje nisu prodane (linija dvadeset i šest), te zatim povlači podatke za taj specifični NFT koji se sprema u niz sastavljen od NFT objekata. Taj objekt je prikazan na slici 54. Nakon svega, niz se postavlja u useState te se isLoading postavlja na laž.

```

3 export interface NFTtype {
4   totalPrice: BigNumberish;
5   price?: string;
6   itemId: string;
7   seller?: string;
8   name: string;
9   description: string;
10  image: string;
11 }

```

Slika 54: NFTType tip podatka

Nakon što se povuku svi NFT-evi, potrebno ih je prikazati. Koristeći `map` funkciju, koja se nalazi na svakom objektu niza u JavaScriptu, iskorištava se NFT komponenta i ispisuje se na zaslone. Prikaz je na slici 55.

```
83 {items.map((nft: NFTtype, idx) => (  
84   <NFT  
85     name={nft.name}  
86     description={nft.description}  
87     image={nft.image}  
88     price={formatBigNumber(nft.totalPrice)}  
89     onClick={() => buyMarketItems(nft)}  
90     collection={"Special collection"}  
91     buttonText="Buy"  
92     key={idx}  
93   />  
94 )}}
```

Slika 55: Izlistavanje NFT-eva

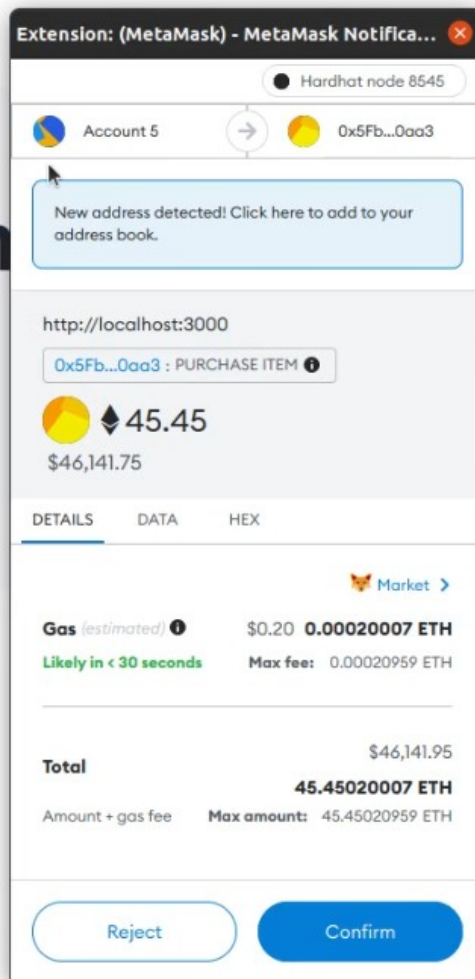
#### 4.5.1.2 Kupovanje

Prilikom izvršenja `map` funkcije, na `onClick prop` se postavlja funkcija koja pozove `buyMarketItems` i proslijedi trenutni NFT. Kôd je prikazan na slici 56. Prilikom pritiska na tipku za kupnju NFT-a, pozove se funkcija unutar pametnog ugovora koja prebaci vlasništvo na kupca, a zatim se pozove funkcija koja ponovno učita sve NFT-ove.

```
56 const buyMarketItems = async (item: NFTtype) => {  
57   await (  
58     marketplace.purchaseItem(item.itemId, { value: item.totalPrice })  
59   ).wait();  
60  
61   loadMarketplaceItems();  
62   showAlert(true);  
63 };
```

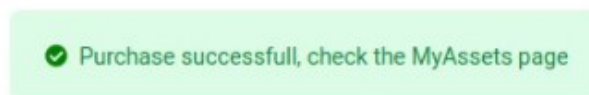
Slika 56: Kupnja NFT-eva

Nakon odabira NFT-a, te pritiskom na tipku *Buy* izvršava se proces kupnje NFT-a te se ishod kao što je vidljivo na slici 57.



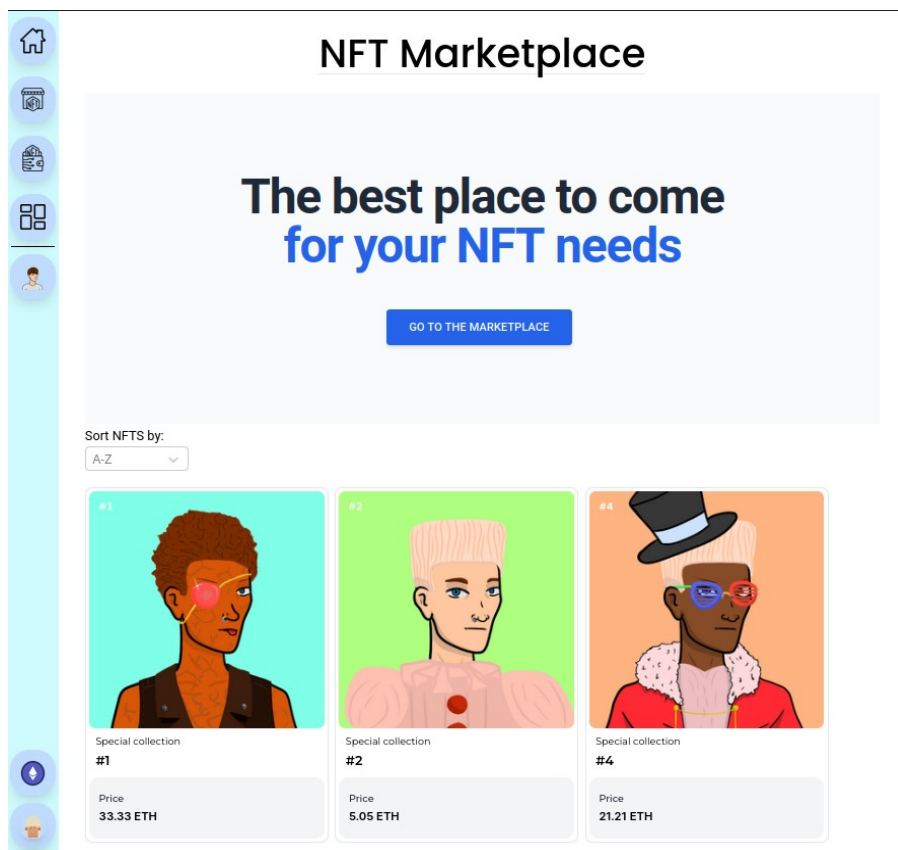
Slika 57: Metamask potvrda za  
kupovanje NFT-a

Ukoliko je kupovina bila uspješna, u donjem desnom kutu se pojavi obavijest poput one prikazane na slici 58. Kupljeni NFT se nalazi na MyAssets stranici.



Slika 58: Obavijest uspješnosti

Izgled početne stranice je vidljiv na slici 59.



Slika 59: Početna stranica

#### 4.5.2 CreateItem

Unutar ove komponente korisnici mogu kreirati NFT-ove koristeći svoju proizvoljnu sliku, njezino ime i opis. Stranica se sastoji od različitih tipova unosa – onih za sliku, za tekst i za tipku za kovanje (engl. *mint*).

##### 4.5.2.1 IPFS

IPFS je kratica za InterPlanetary File System. To je protokol i *peer-to-peer* mreža za pohranjivanje i dijeljenje podataka u distribuiranom datotečnom sustavu. IPFS koristi adresiranje sadržaja radi jedinstvene identifikacije svake datoteke u globalnom imenskom prostoru koja povezuje sve računalne uređaje. Unutar ove komponente se koristi biblioteka

koja se zove *ipfs-http-client* koja omogućuje komuniciranje sa IPFS sustavom. Na početku se inicijalizira IPFS klijent koristeći bazni URL: <https://ipfs.infura.io:5001/api/v0>.

Prilikom ubacivanja slike u unos za slike, poziva se funkcija koja, koristeći inicijaliziranog klijenta, postavlja sliku na IPFS. Kôd je vidljiv na slici 58.

```
20   const uploadToIpfs = async (e: React.ChangeEvent<HTMLInputElement>) => {
21     e.preventDefault();
22     const file = e.target.files![0];
23     if (typeof file !== "undefined") {
24       try {
25         const res = await client.add(file);
26         console.log(res);
27         setImage(`${IPFS_BASE_URL}${res.path}`);
28       } catch (error) {
29         console.log("There was an issue uploading to ipfs: ", error);
30       }
31     }
32   };
```

Slika 60: Spremanje na IPFS

#### 4.5.2.2 Kreiranje i postavljanje

Nakon postavljanja slike na IPFS se dodatno provjeravaju vrijednosti (slika, cijena, ime i opis) ako su valjanje. Kôd se nalazi na slici 59.

```
47   const mintAndList = async (res: any) => {
48     if (!image && !price && !name && !description) return;
49
50     const uri = `${IPFS_BASE_URL}${res.path}`;
51
52     //minting nft
53     await (await nft.mint(uri)).wait();
54
55     //get tokenId of new nft
56     const id = await nft.tokenCount();
57
58     //approve marketplace to spend nft
59     await (await nft.setApprovalForAll(marketplace.address, true)).wait();
60
61     // add nft to marketplace
62     const listingPrice = ethers.utils.parseEther(price!.toString());
63
64     await (await marketplace.makeItem(nft.address, id, listingPrice)).wait();
65   };
```

Slika 61: Kreiranje i postavljanje NFT-a


Kasnije se link IPFS-a prosljeđuje u pametni ugovor NFT-a u funkciju `mint` (linija pedeset i tri), te se dohvaća ID tog NFT-a (linija pedeset i šest). Kako bi trgovina imala dopuštenje premjestiti vlasništvo NFT-a potrebno je postaviti to dopuštenje (linija pedeset i devet). Nakon prethodnih koraka, kreira se NFT i postavlja se na trgovinu (linija šezdeset i četiri).

Konačan izgled sučelja za kreiranje NFT-a se može vidjeti na slici 62.

## Create New Item

\* Required fields

**Image, Video, Audio, or 3D Model \***  
File types supported: JPG, PNG, GIF, SVG, MP4, WEBM, MP3, WAV, OGG, GLB, GLTF. Max size: 100 MB



**Name \***  
This is the name of your NFT

Asset name  
Name of the NFT

**Description \***  
The description will be included on the item's card view.

Description of the NFT

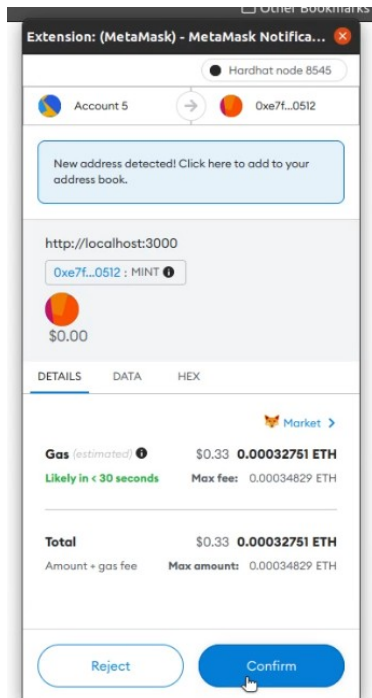
**Price \***  
The amount users you ask for your NFT, the currency is ETH

Asset price  
42

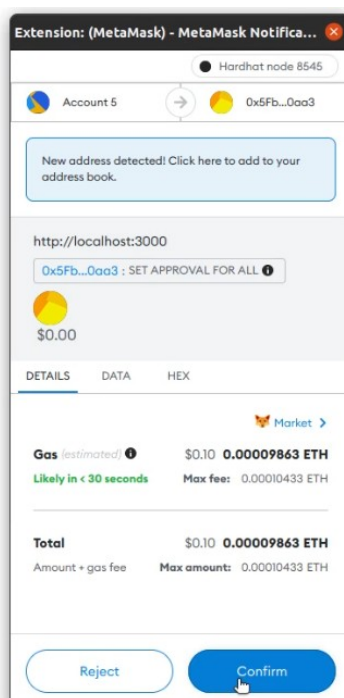
**MINT NFT**

Slika 62: Kreiranje novog NFT-a

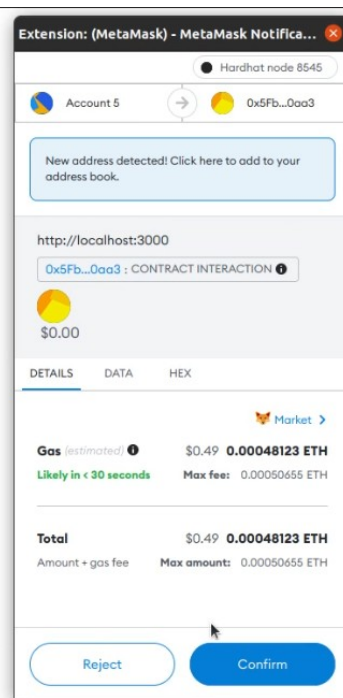
Na sljedećim slikama je prikazan proces potvrđivanja sučelja koristeći Metamask na sljedećim slikama: 63, 64, 65.



Slika 63: Metamask Mint funkcija



Slika 64: Metamask domuštenie potvrda



Slika 65: Metamask interakcija sa ugovorom



### 4.5.3 MyAssets

Unutar ove stranice korisnici će moći vidjeti pregled svih svojih NFT-eva koje su kupili, koristeći prosljeđeni marketplace objekt i `fetchMyNfts` funkciju. Dohvaćanje NFT-eva je slično već objašnjenom i opisanom primjeru za dohvaćanje. Kôd je prikazan na slici 66. Kao što je i prije napravljeno, dohvaćeni su podaci s pametnog ugovora te su zatim povučene sve vrijednosti u `NFTtype` tip podatka i spremljene u `useState`.

```
14  const loadMyAssets = async () => {
15      const data = await marketplace.fetchMyNfts();
16
17      const items: NFTtype[] = await Promise.all(
18          data.map(async (i: any) => {
19              const tokenUri = await nft.tokenURI(i.tokenId);
20              const res = await fetch(tokenUri);
21              const metadata = await res.json();
22
23              //get total price
24              const totalPrice = await marketplace.getTotalPrice(i.itemId);
25
26              const item: NFTtype = {
27                  totalPrice,
28                  price: i.price,
29                  itemId: i.itemId,
30                  name: metadata.name,
31                  description: metadata.description,
32                  image: metadata.image,
33              };
34
35              return item;
36          })
37      );
38
39      setIsLoading(false);
40      setPurchases(items);
41  };
```

Slika 66: Učitavanje vlastitih NFT-eva

#### 4.5.3.1 Prodaja

Prilikom ispisa svih NFT-eva na `onClick prop`, postavljeno je pozivanje funkcije koja prodaje NFT-eve. Kôd `sellNFT` funkcije je vidljiv na slici 67. Logika je slična kao i kod kupovanja, postavlja se dopuštenje da trgovina može prebacivati NFT u ime vlasnika te se na kraju poziva funkcija pametnog ugovora koji prodaje NFT.

```

43   const sellNFT = async (nftForSale: any) => {
44     const price = 42;
45     //approve marketplace to spend nft
46     await (await nft.setApprovalForAll(marketplace.address, true)).wait();
47
48     // add nft to marketplace
49     const listingPrice = formatToEth(price.toString());
50
51     await (await marketplace.sellItem(nftForSale.itemId, listingPrice)).wait();
52     loadMyAssets();
53   };

```

Slika 67: Prodaja NFT-a

Na četrdeset i devetnoj liniji je funkcija koja se zove `formatToEth`. Ona je jedna od mnogih funkcija koje se višestruko koriste u projektu. Uvozi na početku datoteke destrukuiranjem, kao što je prikazano prije. Korisno je imati odvojenu datoteku gdje postoje pomoćne funkcije jer se tako sprječava nepotrebno dupliciranje kôda. Kod je prikazan na slici 68.

Zahvaljujući TypeScriptu, jasno je vidljivo da je ulazna vrijednost funkcije tipa *string*, a povratna *BigNumber*. Unutar same funkcije se koristi `ethers` funkcija koja pretvara kôd u *BigNumber*. Ipak, jednostavnije je napisati svoju funkciju koja to iskorištava, nego zapamtiti cijelu putanju do `ethers` biblioteke pa njene funkcije.

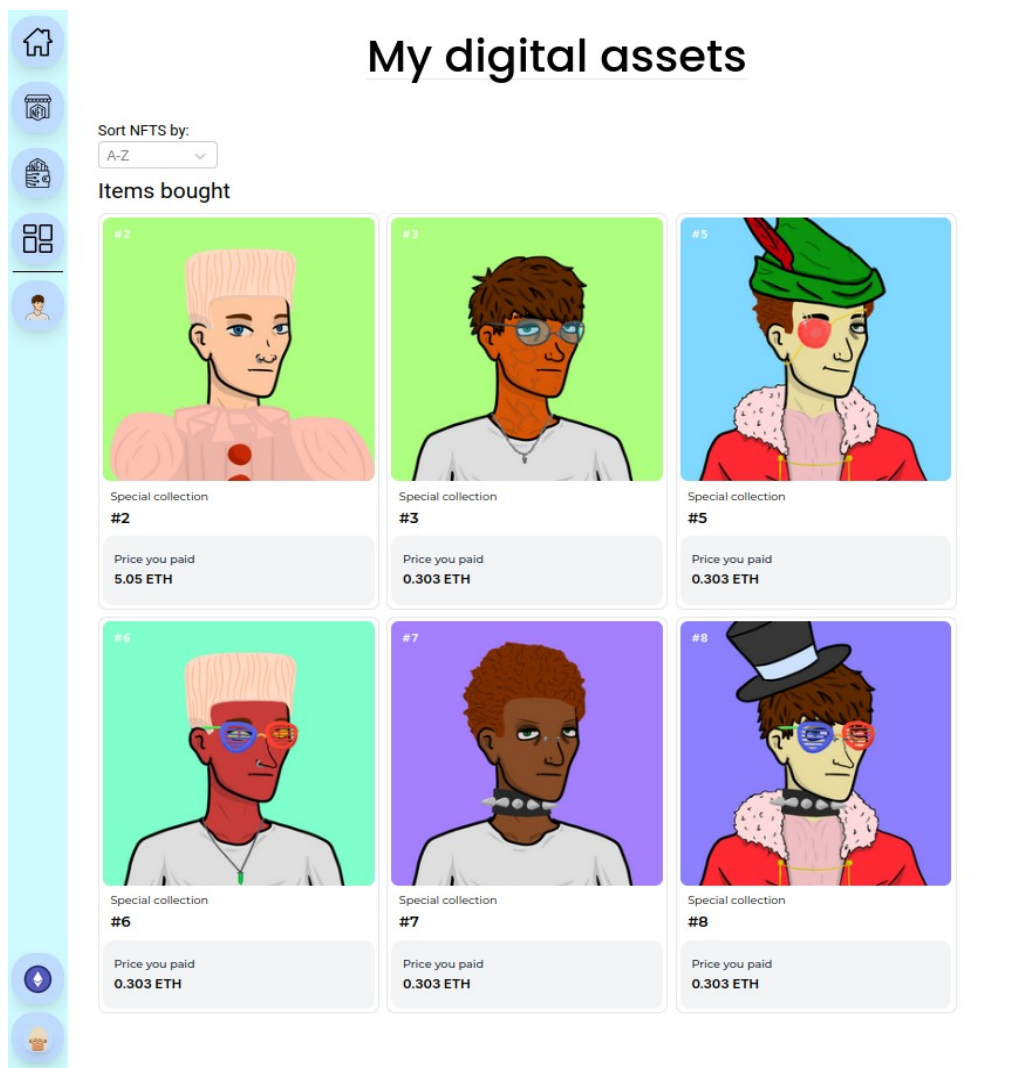
```

6   export const formatToEth = (price: string): BigNumber =>
7     ethers.utils.parseEther(price.toString());

```

Slika 68: `formatToEth` funkcija

Finalni izgled *My Digital Assets* stranice je prikazana na slici 69.



Slika 69: My Digital Assets

#### 4.5.4 CreatorDashboard

Ova je stranica namijenjena kako bi korisnici mogli vidjeti svoju aktivnost unutar cijele web stranice. Tako mogu vidjeti koje su NFT-eve postavili na trgovinu, koje su prodali te mogu i nadodati neke NFT-eve kao favorite. Ujedno, korisnici mogu iz ovog zaslona promijeniti izgled svoje avatar ikone.

Kod ostalih su stranica prilikom dohvaćanja korištene metode unutar pametnih ugovora koje dohvaćaju NFT-eve, no moguće je koristeći „filters” metodu na marketplace

objektu filtrirati po događajima koji su se odasli unutar pametnog ugovora. Time je omogućeno povlačenje samo onih NFT-eva koji su aktivirali određene događaje.

Prikaz kôda je na slici 70. Prikazano je kako se na `offeredFilter` i `boughtFilter` postavlja adresa trenutnog digitalnog novčanika na parametar gdje je `seller` parametar u događaju unutar pametnog ugovora. Događaji u pametnom ugovoru su prikazani na slici 69.

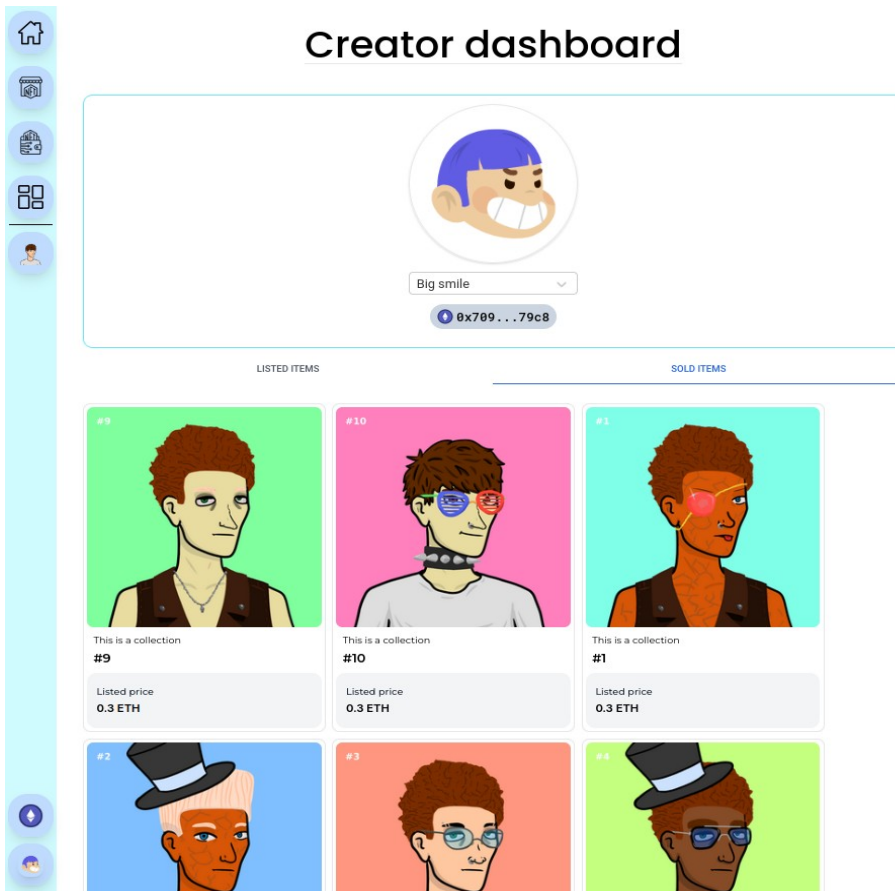
```
28     const offeredFilter = marketplace.filters.Offered(  
29         null,  
30         null,  
31         null,  
32         null,  
33         account  
34     );  
35     const boughtFilter = marketplace.filters.Bought(  
36         null,  
37         null,  
38         null,  
39         null,  
40         account,  
41         null  
42     );  
43     const offeredResults = await marketplace.queryFilter(offeredFilter);  
44     const boughtResults = await marketplace.queryFilter(boughtFilter);
```

Slika 70: Filtriranje preko događaja

```
26     event Offered(  
27         uint256 itemId,  
28         address indexed nft,  
29         uint256 tokenId,  
30         uint256 price,  
31         address indexed seller  
32     );  
33  
34     event Bought(  
35         uint256 itemId,  
36         address indexed nft,  
37         uint256 tokenId,  
38         uint256 price,  
39         address indexed seller,  
40         address indexed buyer  
41     );
```

Slika 71: Događaji u pametnom ugovoru

Nakon što se povuku svi NFT-evi, postavljaju se poput svake druge prijašnje stranice koristeći `useState` te zatim `.map` funkciju. Konačan izgled ove stranice je vidljiv na slici 72.



Slika 72: Creator Dashboard izgled

## 5. Generiranje nasumičnih NFT-eva

Kako bi se iskazala jedinstvenost NFT-eva, aktualan trend je njihova razmjena u obliku slika. Moguće je ručno nacrtati svaki NFT, no većina kolekcija ima po 10,000 NFT-eva u jednoj kolekciji. To bi zahtijevalo iznimno puno vremena jednog umjetnika kako bi kreirao toliki broj potpuno originalnih crteža.

Kako bi se to izbjeglo, koristi se trenutno vrlo popularna za kreiranje velike količine jedinstvenih slika. Ona upotrebljava slojeve slika, koji se zatim spoje u jednu sliku nasumičnim biranjem „karakteristika”. Ove karakteristike (kapa, oči, obrve, nos, usta, nakit i slično) mogu imati različitu vjerojatnost pojave, tj. u kojem postotku će se dodijeliti određena karakteristika. Pri generiranju slike nasumično se odaberu karakteristike ovisno o postotku koji im je pridodan te se generira jedinstvena slika kombinacijom karakteristika koja se pretvori u NFT. Naziv ove nasumično generirane kolekcije će se zvati *Coolio*.

### 5.1 Slojevi

Kako bi se generirale slike koristeći algoritam, potrebno je kreirati različite slojeve slika. Navedene su glavne karakteristike u ovoj kolekciji koje se ujedno slažu se u ovom redosljedu:

- Tijelo
- Obrve
- Hair
- Kapa
- Majica
- Lančić
- Oči

- Naočale
- Usta
- Nos
- Piercing

Slike su dimenzije 1000 piksela puta 1000 piksela. Kako bi se sve slike mogle postaviti jedna na drugu bez preklapanja, potrebna je prozirna pozadina svakog elementa te da se element nalazi gdje bi se i inače nalazio na slici da su prisutni i svi drugi elementi Time se osigurava da npr. nos bude na mjestu gdje bi nos trebao bit, a ne negdje sa strane.

Na slici 73 se može vidjeti kako izgleda primjer sloja kose.



Slika 73: Sloj kose

Dio koda koji generira samu sliku je prikazana na slici 74.

```
56  const drawElement = (el) => {
57    ctx.drawImage(
58      el.loadedImage,
59      el.layer.position.x,
60      el.layer.position.y,
61      el.layer.size.width,
62      el.layer.size.height
63    );
64    addAttributes(el);
65  };
66
67  const drawImage = (layers, id) => {
68    drawBackground();
69    layers.forEach((el) => drawElement(el));
70    signImage(`#${id}`);
71  };
```

Slika 74: Kod za iscrtavanje slike

Kako bi se spojili slojevi unutar kôda, prvo ih je potrebno proslijediti unutar funkcije koja nacрта pozadinu (linija šezdeset i osam) i učita sve slojeve. Nakon tog koraka, sve se prosljeđuje u funkciju koja slojeve posloži po odabranom redu (linija pedeset i sedam do linije šezdeset i tri) te zatim u funkciju koja ispiše redni broj slike kako bi se označilo izdanje slike.

## 5.2 Generiranje rijetkosti

Postavljaju se različite rijetkosti unutar generiranih slika kako bih se postigla dodatna umjetna rijetkost (engl. *artificial rarity*). Ako svi slojevi imaju jednaku šansu za biti prikazani, nema nikakvog smisla imati jedan atribut iznad drugog osim zbog drugačijeg izgleda.

No, nadodavanjem varijabilnih šansi za dobiti neki atribut daje mogućnost dobivanja generirane slike s vrlo rijetkim karakteristikama, time vrijedi više na tržištu. Kako bi se postigao taj efekt, koristit će se algoritam koji se zove težinska nasumičnost (engl. *weighted random*).

### 5.2.1 Težinska nasumičnost

Težinska nasumičnost je algoritam koji omogućava nasumični odabir brojeva. Vjerojatnost odabira nekog broja ovisi o težini tog broja.

Primjer odjevnih predmeta:

Lista odjevnih predmeta bi izgledala ovako: [„majica”, „jakna”, „džemper”].

A lista težina bi bila prikazana brojčano: [3,7,1]. Time je prikazano da se majica želi obući 3 od 11 puta ( $3 + 7 + 1 = 11$ ), jakna 7 od 11 puta i džemper 1 od 11 puta.



### 5.2.1.1 Jednostavna implementacija

Jednostavan način za prikazati ovaj algoritam bi bio kreiranjem liste koja ima količinu stavki pomnoženu sa svojom težinom. Dakle prethodni niz bi izgledao ovako:

```
[„majica”, „majica”, „majica”, „jakna”, „jakna ”, „jakna ”, „jakna ”, „jakna ”, „jakna ”, „jakna ”, „džemper”]
```

Sada je jasnije prikazano koja stavka ima veću vjerojatnost za biti odabrana. Nažalost, ovakav pristup nije efikasan za pohranu.

### 5.2.1.2 Učinkovita implementacija

Moguće je postići isti rezultat s jednostavnom implementacijom ako se radi o manjim količinama. Ipak, ukoliko je riječ o tisućama stavki potreban je bolji pristup [22].

Pristup je prikazan u idućim koracima:

- 1 **Suma težina** – potrebno je kreirati novu listu sa sumom svih težina stavki u listi. Ona se zove „sumaTezina”, a izgledat će ovako:  $\text{sumaTezina} = [3, 3+7, 3+7+1] = [3,10,11]$
- 2 **Generiranje nasumičnog broja** – potrebno je generirati nasumični broj u rasponu od 0 do ukupne sume svih težina; u ovom slučaju do 11. Kao primjer će nasumični broj biti 5.
- 3 **Odabir stavke** – potrebno je proći kroz listu „sumaTezina” i s lijeva na desno uzeti indeks prve stavke kojoj je broj veći ili jednak nasumično generiranom broju. Koristeći taj indeks odabire se stavka.

Logika ovog pristupa je da će stavke s većom težinom zauzeti brojčano više mjesta i tako imaju veću vjerojatnost da će biti odabrane. Primjer ovakve jednostavne implementacije je vidljiv na slici 75.

```

const tezine = [3, 7, 1];
const sunmaTezina = [3, 10, 11];
const sumaTezinaDemonstracija = [
  1, 2, 3,           // <-- 3 broja
  4, 5, 6, 7, 8, 9, 10, // <-- 7 brojeva
  11                // <-- 1 broj
];

```

Slika 75: Težinska vjerojatnost

### 5.2.2 Praktična primjena

Nije uvijek moguće kreirati odvojenu listu u kojoj će se upisivati različite težine stavki. Kako bi se pridodale različite vrijednosti slikama, kod imena slike svakog sloja se nalazi nastavak.

Zbog jednostavnosti postoji stupnjevanje rijetkosti:

- 1 Originalno – naznačeno praznim sufiksom
- 2 Rijetko – naznačeno sufiksom „\_r”
- 3 Vrlo rijetko – naznačeno sufiksom „\_sr”

Na slici 76 je vidljivo takvo stupnjevanje unutar kôda. Uz taj se objekt direktno nadodaje težina koja ide uz tu rijetkost.

```

14 const RARITY = [
15   { key: "", val: "original", weight: 50 },
16   { key: "_r", val: "rare", weight: 10 },
17   { key: "_sr", val: "super rare", weight: 1 },
18 ];

```

Slika 76: Rijetkosti

Koristeći ovaj pristup, na imenima datoteke se postavlja sufiks ovisno o rijetkosti toga sloja. Primjer je prikazan na slici 24.



Slika 77: Primjer imena slojeva

Kako bi se svi ovi elementi težinske nasumičnosti i generiranja slike spojili u gotov proizvod, na slici 29 je prikazan kôd koji učitava sve slojeve, iščitava ime i pridodaje sloju težinu. Isti kôd zbraja težine tog tipa sloja (linija devedeset i pet) i generira nasumični broj kojem je maksimalna vrijednosti suma težina brojeva (linija devedeset i šest) te se zatim nasumično odabire ovisno o težini.

Povratna vrijednost ove funkcije je objekt koji sadrži informacije koje će biti iskorištene za iscrtavanje lika, kao na početku ovog potpoglavlja.

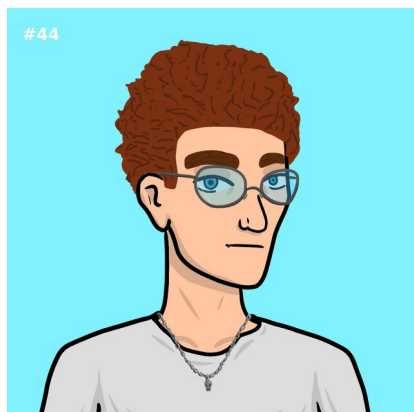
```
89  const getWeightSum = (array) =>
90    array.reduce((prevVal, currVal) => prevVal + currVal.weight, 0);
91
92  const createCharacter = (characterType) => {
93    let character = [];
94    characterType.layers.forEach((layer) => {
95      const weightSum = getWeightSum(layer.elements); //Used to calculate weight of traits
96      const randElementNum = generateRandomNumber(weightSum);
97
98      let num = 0;
99      layer.elements.forEach((element) => {
100        //Looping through all the elements weights to see which one will be randomly picked
101        if (randElementNum >= weightSum - element.weight) {
102          num = element.id;
103        }
104      });
105      character.push(num);
106    });
107    return character;
108  };
```

Slika 78: Kôd kreiranja lika

Pri generiranja lika se provjerava kako bi se izbjegao duplikat, tj. provjerava se postojanje takvog istog lika od prije. Provjera se provodi prvo spremanjem generiranog objekta lika i

zatim se uspoređuje sa svim prethodno kreiranim likovima. Tako se osigurava jedinstvenost svakog lika.

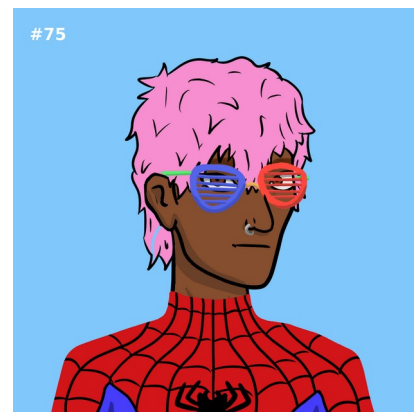
Primjer nekoliko generiranih likova je prikazano na slikama 75, 76, 64.



Slika 79: Primjer slike 1



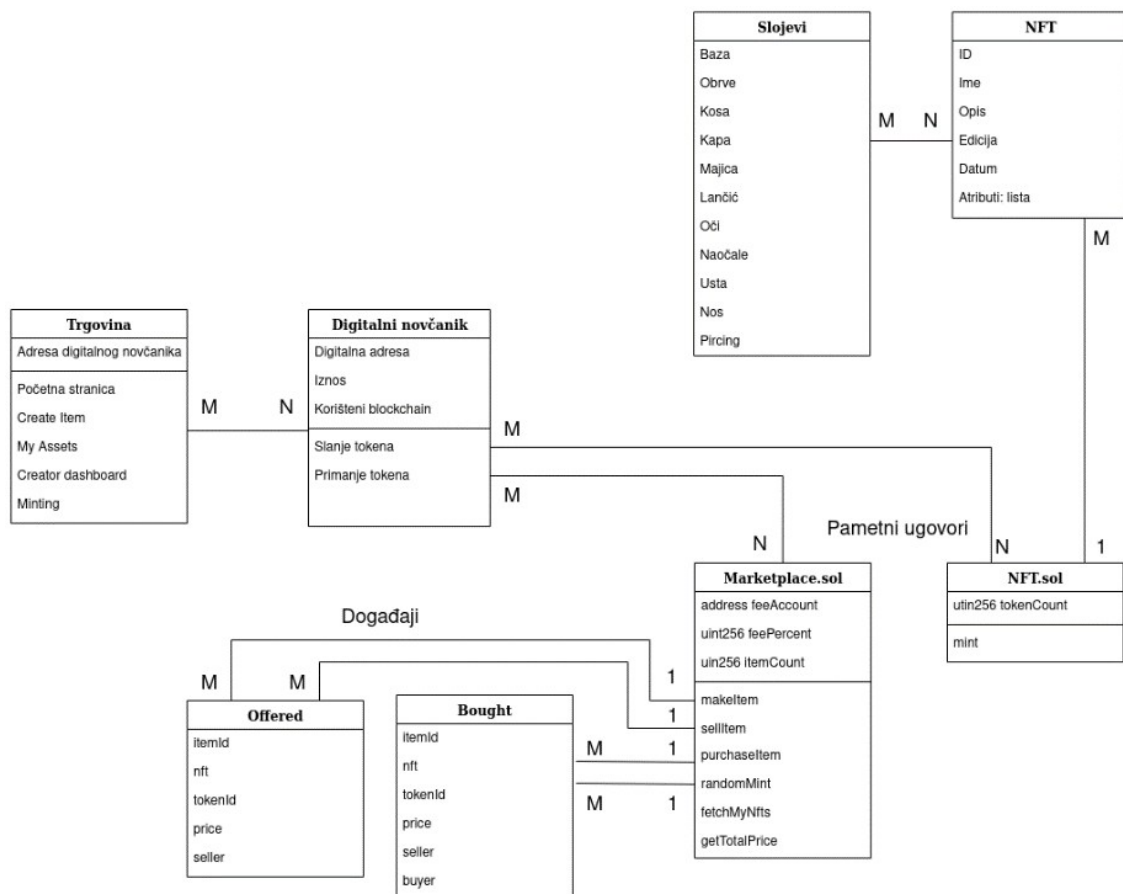
Slika 80: Primjer slike 2



Slika 81: Primjer slike 3

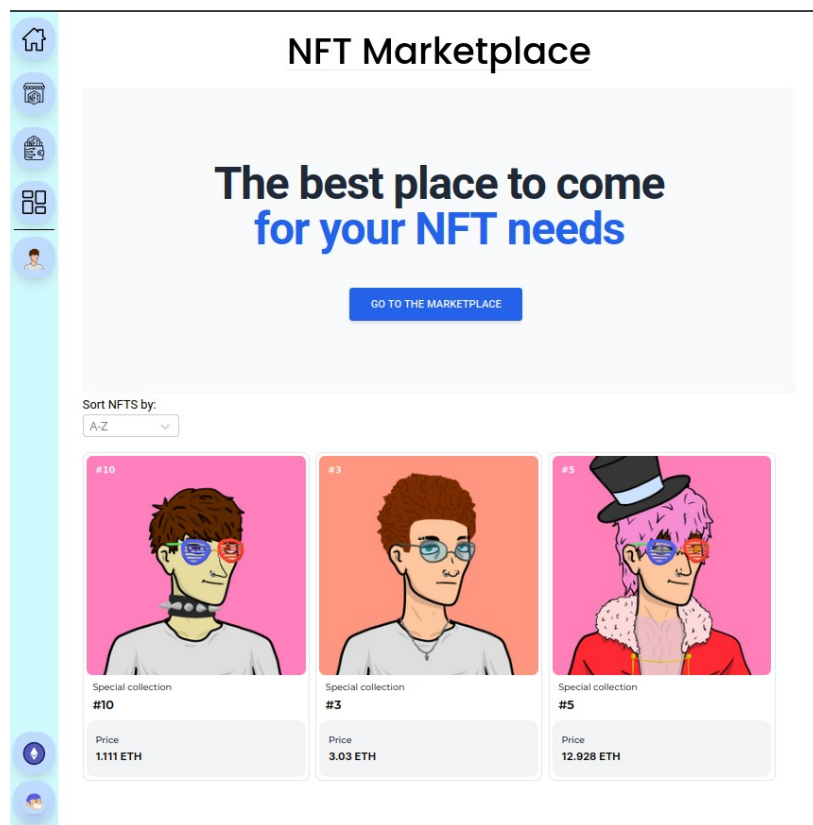
### 5.3. ER dijagram i prikaz prolaska slijeda projekta

Na slici 65 je prikazan ER dijagram cijelog projekta, unutar kojeg se vidi relacija trgovine sa digitalnim novčanikom i kako se događa komunikacija između pametnih ugovora i odašiljanje događaja.



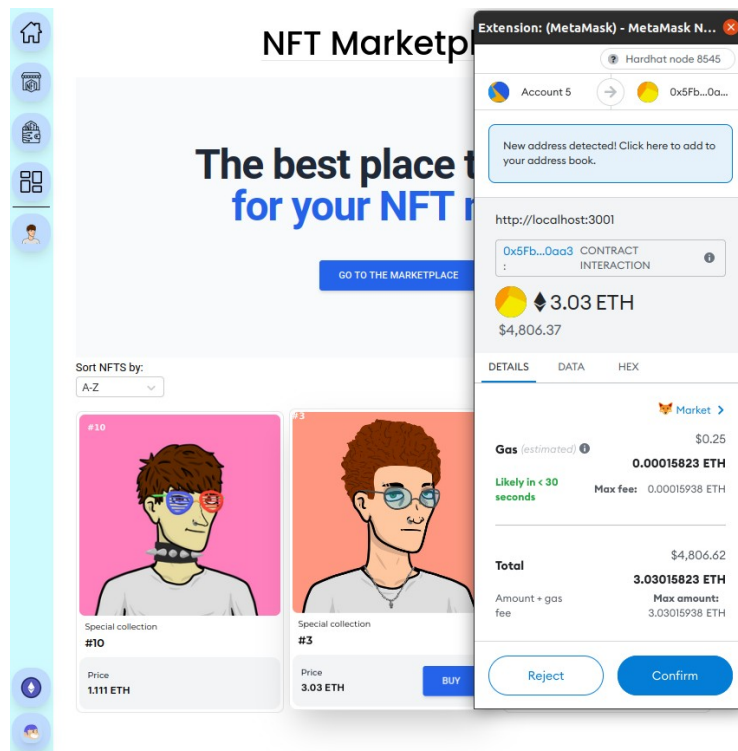
Slika 82: ER dijagram

Prilikom ulaska u NFT trgovinu korisniku će biti prikazana početna stranica na kojoj su svi raspoloživi NFT-evi, vidljivo na slici 83.



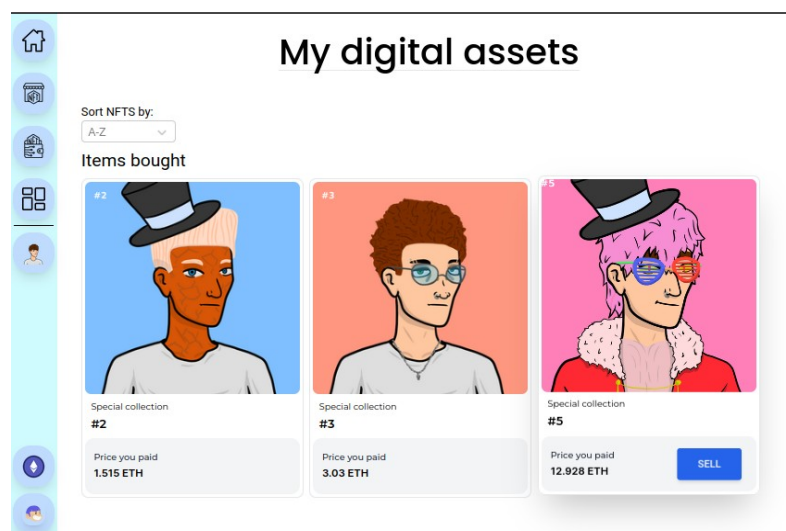
Slika 83: Početna stranica trgovine

Ako korisnik želi kupiti NFT, prelaskom miša na sliku se pojavi tipka koja nakon pritiska otvara digitalni novčanik pomoću kojeg se prihvaćaju cijene usluge korištenja Ethereum mreže i trgovine. Prihvaćanje kupovine korištenjem digitalnog novčanika je vidljiva na slici 84.



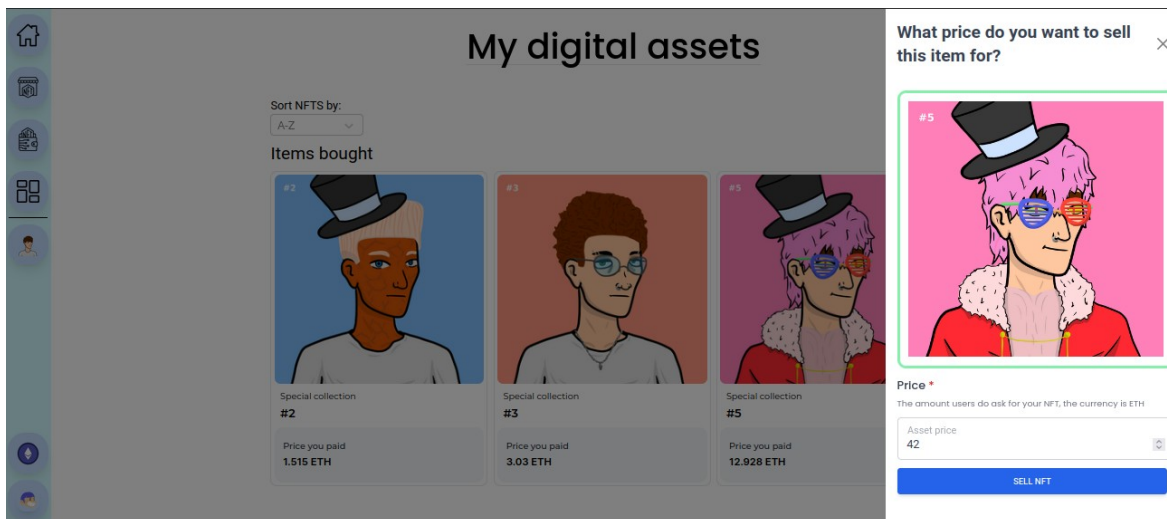
Slika 84: Primjer kupovanja NFT-a

Prilikom prihvaćanja cijene koja je prikazana u prozoru digitalnog novčanika, korisniku se prenosi NFT u novčanik korisnika iz novčanika trgovine. Pregled kupljenih NFT-eva je vidljiv na stranici *My digital assets*, prikazan na slici 85.



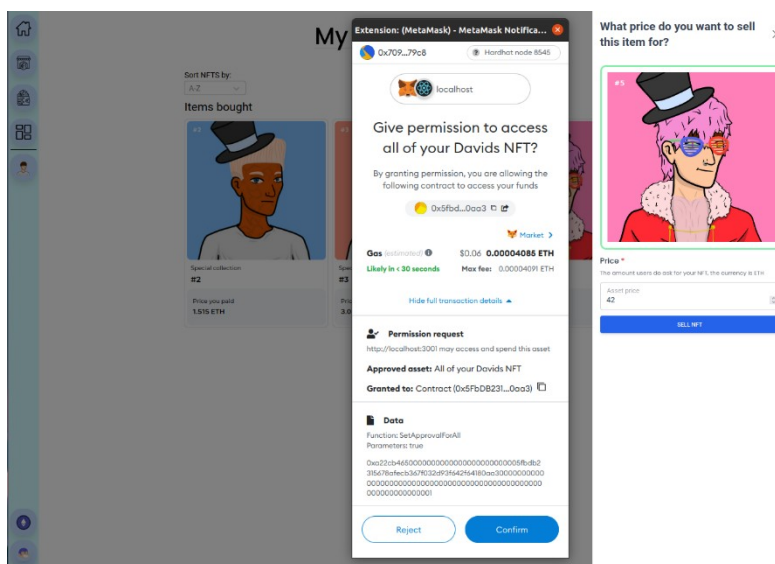
Slika 85: Kupljeni NFT na My Assets stranici

Iz ovog zaslona je moguće prodati NFT, istim postupkom kao što je kupljen NFT. Pritiskom na gumb za prodavanje se otvara sporedni prozor unutar kojeg je moguće podesiti traženu cijenu za NFT, prikazano na slici 86.



Slika 86: Prodaja NFT-a

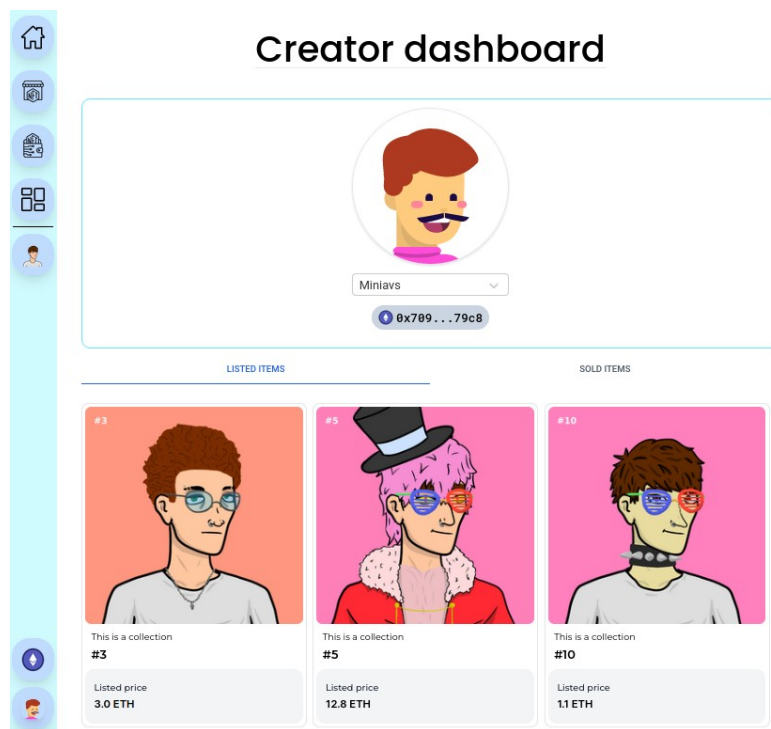
Prilikom pritiska na gumb za prodavanje se otvaraju slični prozori kao i za kupnju, u kojoj se potvrđuje prebacivanje NFT-a iz korisničkog novčanika u novčanik trgovine. Prikazano na slici .



Slika 87: Potvrda prodaje NFT-a

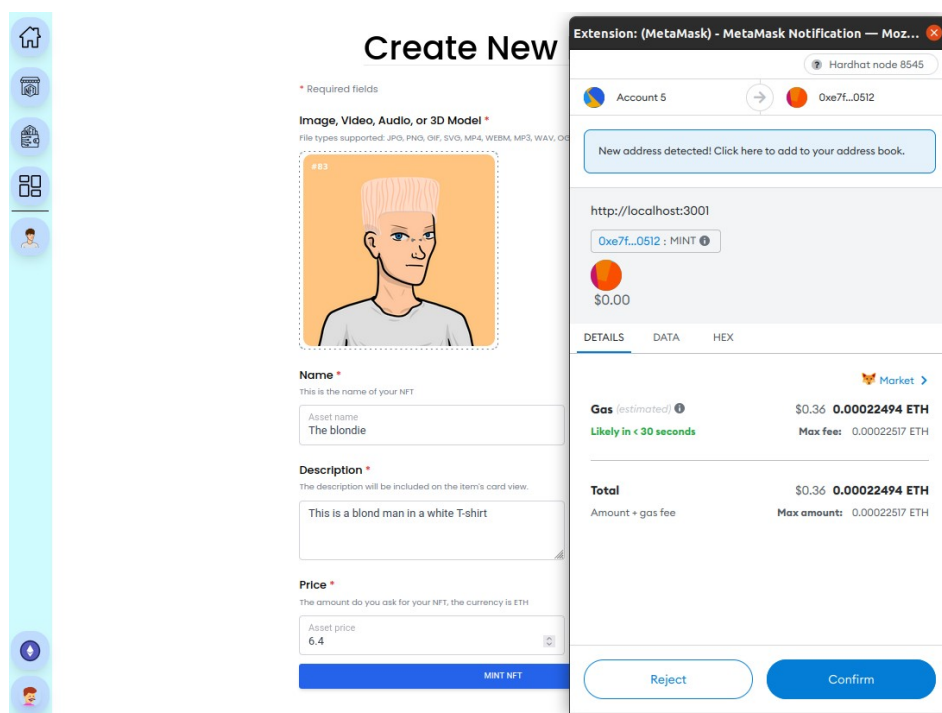


Unutar *Creator Dashboard* stranice korisnik ima uvid u NFT-eve koji su postavljeni za prodaju i prodane NFT-eve. Unutar ove stranice je moguće promjeniti koja će se ikonica prikazivati korisniku, kao što je prikazano na slici 88.



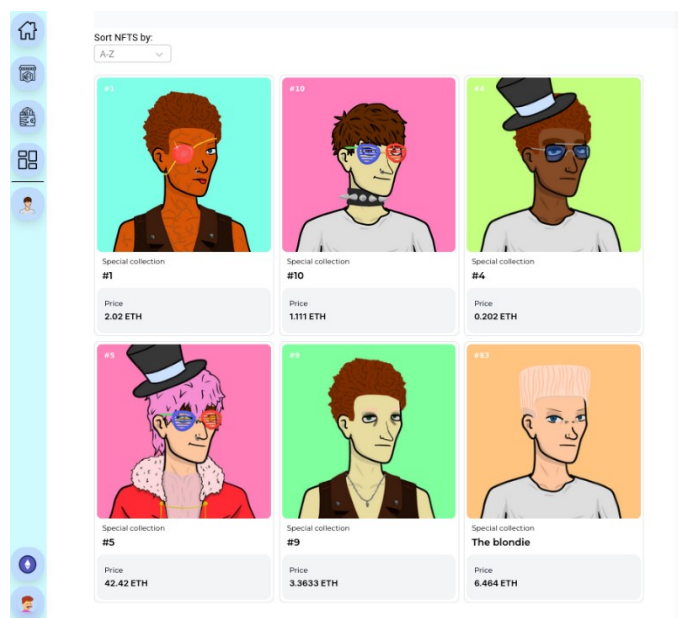
Slika 88: Prikaz Creator Dashboard stranice

Moguće je kreirati NFT od proizvoljne slike unutar *Create New Item* stranice. Unutar koje se postavlja slika i dodatne informacije o NFT-u. Zatim pritiskom na *Mint NFT* tipku se prikaže digitalni novčanik za potvrdu, kao što je prikazano na slici 89.



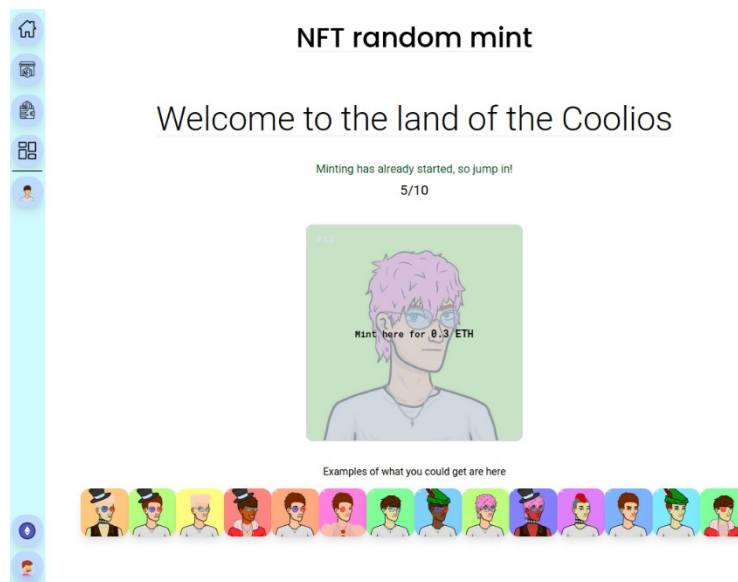
Slika 89: Kreiranje NFT-a

Nakon prihvaćanja na početnoj stranici je prikazana slika koja je prodana uz sliku upravo kreiranog NFT-a. Vidljivo na slici 90.



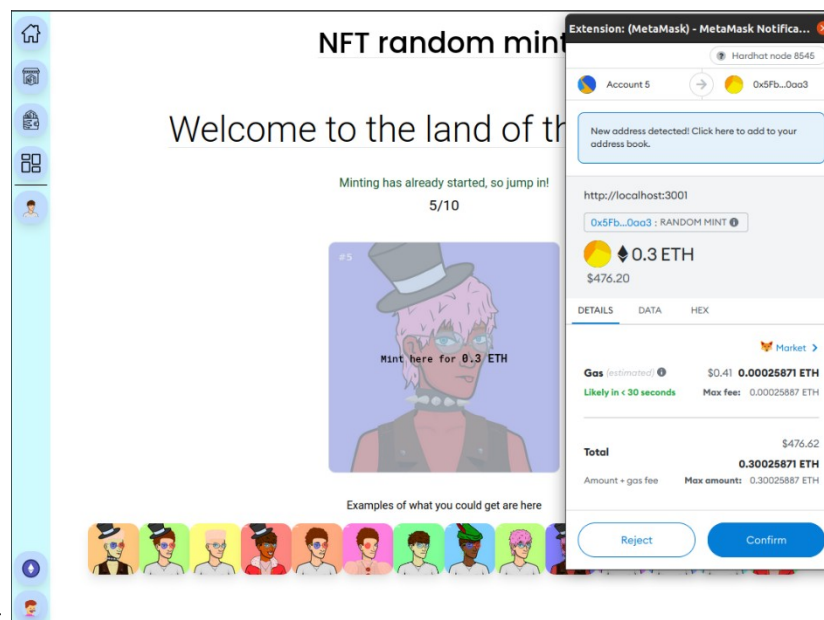
Slika 90: Prikaz kreiranog NFT-a u trgovini

Posljednji zaslon u trgovini je za generiranje nasumičnih NFT-eva. Prilikom ulaska je vidljiva tipka koja prikazuje primjere nekih mogućih NFT-eva. Prikazano na slici 91.



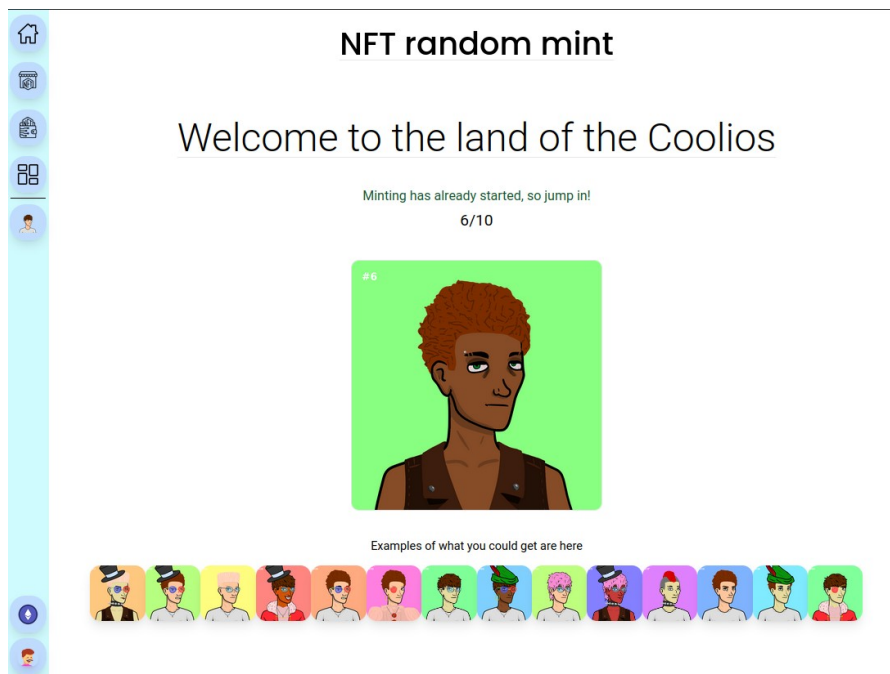
Slika 91: Nasumični NFT

Pritiskom na tipku za kreiranje NFT-a se otvara digitalni novčanik u kojem se potvrđuje kupnja NFT-a. Digitalni novčanik je prikazan na slici 92.



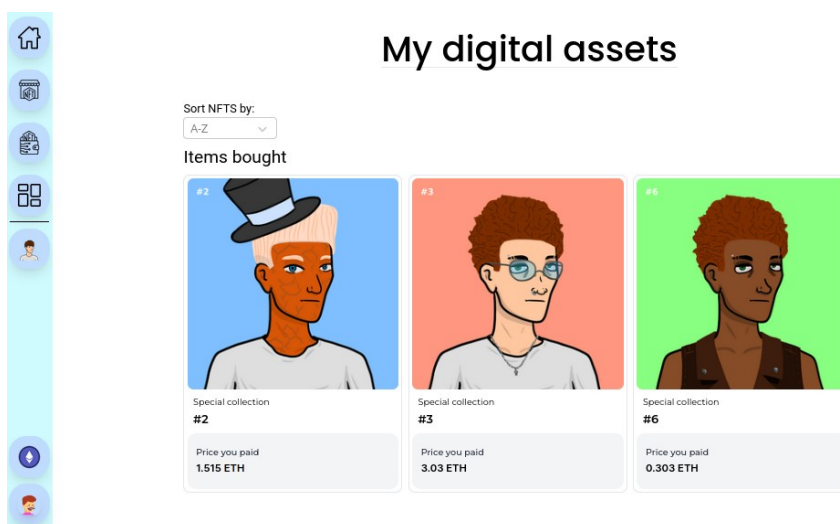
Slika 92: Potvrda kupnje nasumičnog NFT-a

Nakon prihvaćanja cijene kreiranje nasumičnog NFT-a, dobiveni NFT-a se prikazuje na mjestu gdje je tipka za kupnju nasumičnog NFT-a. Prikazano na slici 93.



Slika 93: Prikaza generiranog NFT-a

Upravo dobiveni NFT je vidljiv na stranici *My digital assets*, kao što je prikazano na slici 94.



Slika 94: Generirani NFT u My Assets stranici

## 6. Zaključak

U ovom završnom radu opisane su mnoge aktualne i nove teme trenutno zanimljive svijetu tehnologije i programiranja.

Pisanje samih pametnih ugovora, uz blockchain tehnologiju, subjektivno smatram najzanimljivijim dijelom pisanja čitavog završnog rada. Pametni ugovori su u prethodnih nekoliko godina postali iznimno popularni. Upravo ta popularnost doprinijela je porastu broja izvora za učenje, od kojih mi je dio pomogao pri istraživanju i pisanju ovog rada. Najveći je bio postavljanje *backend developerskog okruženja*, što je, vjerujem, bilo zbog relativno malog osobnog iskustva u tom području. Zasiurno, stečeno znanje je nešto što ću koristiti u budućim projektima na kojima budem radio. U ovom završnom radu koristio se i TypeScript koji je, pogotovo u većim projektima, iznimno tražen programski jezik. Smatram da sam, izlaskom iz vlastite „zone komfora“ i „onog poznatog“, za vrijeme pisanja projekta i rada unaprijedio svoje korištenje ovim programskim jezikom.

Jedno od ključnih dijelova ovog završnog rada je bilo kreiranje NFT-eva od slojeva slika. Pomoću ove metode kreirali su se neki od najvećih NFT projekta na svijetu, kao što su: „Bored Ape Yacht Club“, „Azuki“, „Cool Cats“ i mnogi drugi. Ta je vještina tako od neizmjerne važnosti za bilo kojeg budućeg Web 3.0 developera. Zahvaljujući lokalnom umjetniku Brunu Nikoliću koji je dizajnirao slojeve i dao mi dopuštenje za njihovo korištenje u ovom projektu, bilo je moguće kreirati prekrasne NFT-eve upravo kombinacijom manjih dijelova te njihovim spajanjem poput Lego kockica.

Uvjeren sam da će pametni ugovori u budućnosti biti ukomponirani u puno aspekata svakodnevnog života. Uz sve veća ulaganja od ogromnih kompanija kao što su Meta, Twitter, Spotify, Microsoft i dr., teško je ne predviđati budući uspjeh Web 3.0. Iako je trenutno u samom početku, tehnologija i ljudska želja za sigurnijom i boljom budućnosti nikad neće prestati.

## Literatura

- 1 What is ethereum?, <https://ethereum.org/en/what-is-ethereum/>, dohvaćeno 24. 6. 2022.
- 2 Non-Fungible tokens (NFT), <https://ethereum.org/en/nft/#what-are-nfts>, dohvaćeno 24. 6. 2022.
- 3 What are NFTs, Dhruv Karthik, <https://mytnft.com/learn/what-are-nfts/>, dohvaćeno 24.6.2022
- 4 What is Web3? The Decentralized Internet of the Future Explained, Nader Dabit, <https://www.freecodecamp.org/news/what-is-web3/>, dohvaćeno 24. 6. 2022.
- 5 Solidity, <https://soliditylang.org/>, dohvaćeno 24. 6. 2022.
- 6 What Is Solidity and How Is It Used to Develop Smart Contracts?, Calvin Ebun-Amu, <https://www.makeuseof.com/what-is-solidity/>, dohvaćeno 24. 6. 2022.
- 7 Hardhat Getting Started - Overview, <https://hardhat.org/getting-started#overview>, dohvaćeno 24. 6. 2022.
- 8 What is MetaMask? How to Use the Top Ethereum Wallet, What is MetaMask? How to Use the Top Ethereum Wallet, Daniel Phillips i Matt Hussey, <https://decrypt.co/resources/metamask>, dohvaćeno 24. 6. 2022.
- 9 What is ethers.js – <https://docs.ethers.io/v4/>, dohvaćeno 24. 6. 2022.
- 10 Visual Studio Code, <https://code.visualstudio.com/>, dohvaćeno 25. 6. 2022.
- 11 TypeScript, <https://www.typescriptlang.org/>, dohvaćeno 25. 6. 2022.
- 12 Data Types, <https://javascript.info/types>, dohvaćeno 25. 6. 2022.
- 13 ReactJS, <https://reactjs.org> , dohvaćeno 25. 6. 2022.
- 14 Getting started with React and TypeScript, Amanda Fawcett, <https://www.educative.io/blog/react-and-typescript>, dohvaćeno 25. 6. 2022.

- 15 What is an interface in React, <https://askinglot.com/what-is-interface-in-react>, dohvaćeno 25. 6. 2022.
- 16 Get started with Tailwind CSS, <https://tailwindcss.com/docs/installation>, dohvaćeno 25. 6. 2022.
- 17 What is Git?, <https://docs.microsoft.com/en-us/devops/develop/git/what-is-git>, dohvaćeno 25. 6. 2022.
- 18 What is GitHub? A beginner's introduction to GitHub, <https://kinsta.com/knowledgebase/what-is-github/>, dohvaćeno 25. 6. 2022.
- 19 Metamask – Custom Networks, <https://metamask.zendesk.com/hc/en-us/articles/4404424659995-User-Guide-Custom-networks-and-sidechains>, dohvaćeno 24. 6. 2022.
- 20 OpenZeppelin, <https://www.openzeppelin.com/>, dohvaćeno 24. 6. 2022.
- 21 Chai, <https://www.chaijs.com/>, dohvaćeno 24. 6. 2022.
- 22 Weighted Random in JavaScript, <https://trekhele.medium.com/weighted-random-in-javascript-4748ab3a1500>, dohvaćeno 5.7. 2022.