dREADER-DECENTRALIZIRANI MARKETPLACE SPECIJALIZIRAN ZA DIGITALNE GRAFIČKE NOVELE

Volarević, Josip

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Prijediplomski stručni studij Informacijska tehnologija

JOSIP VOLAREVIĆ

Z A V R Š N I R A D

dReader - DECENTRALIZIRANI MARKETPLACE SPECIJALIZIRAN ZA DIGITALNE GRAFIČKE NOVELE

Split, rujan 2024.

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Prijediplomski stručni studij Informacijska tehnologija

Predmet: Programiranje na internetu

Z A V R Š N I R A D

Kandidat: Josip Volarević

Naslov rada: dReader - decentralizirani marketplace specijaliziran za digitalne grafičke novele

Mentor: Marina Rodić, viši predavač

Split, rujan 2024.

Sadržaj

Sažetak

Završni rad bavi se razvojem i implementacijom platforme dReader, koja omogućuje digitalno čitanje i prikupljanje stripova koristeći Solana blockchain. Problem koji je pokrenuo razvoj platforme jest želja strip entuzijasta za digitalnim kolekcionarstvom te nedostatak prilika za monetizaciju sadržaja od strane stvaratelja stripova.

Kroz analizu postojećih rješenja i odabir najprikladnijih tehnologija, razvijena je platforma koja omogućuje kreatorima da unovče svoj rad kroz tokenizirane stripove, uz dodatne mogućnosti poput digitalnih potpisa umjetnika. Cilj rada je bio napraviti sveobuhvatan sustav koji će pojednostaviti proces stvaranja, distribucije i prikupljanja digitalnih stripova, istovremeno osiguravajući sigurno i transparentno korisničko iskustvo na Solana blockchainu.

Ključne riječi: Blockchain, NestJS, Next.js, Solana, TypeScript

Summary

dReader - decentralized marketplace specialized for digital graphic novels

This thesis focuses on the development and implementation of the dReader platform, which enables digital reading and collection of comics using Solana blockchain technology. The driving force behind the platform's development was the desire of comic enthusiasts for digital collectibles and the lack of opportunities for comic creators to monetize their content.

Through the analysis of existing solutions and the selection of the most suitable technologies, a platform was developed which allows creators to monetize their work through tokenized comics, with additional features such as digital signing. The goal of this project was to create a comprehensive system which simplifies the process of creating, distributing, and collecting digital comics while ensuring a secure and transparent user experience on the Solana blockchain.

Keywords: Blockchain, NestJS, Next.js, Solana, TypeScript

1. Uvod

Digitalni stripovi postaju sve popularniji, ali tržište često ne nudi učinkovite načine za monetizaciju za nezavisne autore, niti platformu koja podržava decentralizirano vlasništvo i distribuciju. Unatoč rastućem interesu za digitalnim kolekcionarstvom, nedostaju platforme koje bi integrirale moderne tehnologije poput blockchaina kako bi omogućile sigurnost, transparentnost i nove modele zarade.

Motivacija za odabir ovog problema proizlazi iz rastuće potražnje za digitalnim sadržajem i nedostatka rješenja koja adresiraju specifične potrebe kreatora i kolekcionara digitalnih stripova. Blockchain tehnologija, posebno Solana, nudi jedinstvene mogućnosti za rješavanje ovih izazova, što je potaklo razvoj platforme dReader *(decentralized reader)*.

Problem je riješen razvojem platforme dReader koja koristi blockchain tehnologiju Solana za stvaranje decentraliziranog tržišta gdje korisnici mogu čitati, skupljati i preprodavati digitalne stripove. Platforma omogućuje tokenizaciju stripova, digitalne potpise autora, i digitalno kolekcionarstvo, što stvara dodanu vrijednost za korisnike.

Glavni doprinos dReader platforme je u stvaranju novog ekosustava koji omogućava stvarateljima ostvariti pravedniju zaradu i osvojiti publiku koja će komunicirati sa njima sa ciljem izgradnje međusobnog odnosa.

U nastavku rada detaljno će se opisati specifikacija projekta, korišteni razvojni alati i tehnologije, proces implementacije platforme, kao i sigurnosni i tehnički izazovi koji su se pojavili tijekom razvoja proizvoda. Zaključno poglavlje predstavlja postignuća projekta te razmatra mogućnosti za budući razvoj i poboljšanja.

2. Specifikacija aplikacije

dReader je zamišljen kao platforma koja omogućuje digitalno čitanje i prikupljanje stripova koristeći blockchain tehnologiju Solana. Proizvod predstavlja višestrano tržište: s jedne strane čitatelji i kolekcionari stripova, te s druge strane autore stripova.

Sučelje za objavljivanje stripova, namijenjen za autore, će biti izoliran od glavne aplikacije. Radi jednostavnosti čitanja završnog rada, izraz "korisnici" će se odnositi na čitatelje i kolekcionare stripova. Ovaj završni rad će se primarno fokusirati na izradu sučelja za krajnje korisnike.

Naziv aplikacije za korisnike: dReader (skraćeno za *decentralized reader*)

Naziv aplikacije za kreatore: dPublisher (skraćeno za *decentralized publisher*)

Ciljana platforma: web preglednici

Kratki opis: dReader je platforma za pretragu, čitanje, prikupljanje, i preprodaju digitalnih grafičkih novela.

Slika 1: Promotivni prikaz dReader platforme

2.1. Opis problema i zahtjevi

Srž aplikacije su prodaja i digitalno prikupljanje stripova, što implicira probleme poput: registracija korisnika i kreatora, mogućnost objavljivanja i prodaje stripova, sakupljanja stripova, potpisivanja stripova u svrhu nagrađivanja kolekcionara, te otvaranje stripova u svrhu čitanja.

2.1.1 Varijanta naslovnice stripa

U fizičkome svijetu stripovi se često prodaju u dućanima na način da imaju dostupne različite naslovnice. Strastveni ljubitelji stripova će htjeti skupljati setove, odnosno više puta kupiti primjer iste epizode, sa ciljem prikupljanja različite naslovnice.

Često to izgleda na način da 10,000 primjeraka bude izdano u naslovnici od glavnog ilustratora stripa. 4,500 primjeraka izdano u naslovnici od gostujućeg umjetnika, i 500 (jako mala količina) u naslovnici od drugog gostujućeg umjetnika (u pravilu jako poznatog i cijenjenog ilustratora).

Slika 2: Primjer stripa sa različitim varijantama naslovnice

Na dPublisher aplikaciji za autore stripova, cilj je omogućiti autorima dodavanje različite varijante naslovnica i unesu podatke o ilustratoru pojedine naslovnice. U dReader aplikaciji, želi se postići da korisnici mogu sakupljati digitalne stripove sa različitim naslovnicama.

2.1.2 Digitalno potpisivanje stripova

Digitalno potpisivanje stripova je ključna značajka dReader platforme, koja omogućuje autorima stripova dodavanje osobnog pečata na svoja digitalna djela. Ova funkcija ne samo da povećava autentičnost i originalnost digitalnih stripova, već također dodaje vrijednost stripovima kao kolekcionarskim predmetima.

Digitalni potpisi na dReader platformi realizirani su kroz integraciju s blockchain tehnologijom, koja osigurava neizbrisivost i provjerljivost potpisa. Svaki digitalni potpis se zapisuje u glavnu knjigu (engl. *ledger*), koristeći transakcije na blockchain tehnologiji Solana, što osigurava transparentnost i sigurnost transakcija.

Digitalno potpisivanje stripova pruža dodatnu vrijednost za kolekcionare jer povećava kolekcionarsku vrijednost stripa, omogućuje kolekcionarima da imaju autentičnost potpisa, te potiče interakciju između autora i kolekcionara, gradivši zajednicu oko stripova i njihovih autora.

Slika 3: Naslovnice stripova sa digitalnim potpisom

Proces digitalnog potpisivanja je sljedeći:

- **registracija potpisa:** umjetnici registriraju svoj potpis u dReader aplikaciji, koji se pohranjuje kao jedinstveni digitalni identifikator.
- **zahtjevanje potpisa:** kada kolekcionar želi zatražiti potpis za svoju (tokeniziranu) kopiju stripa, može pokreniti naredbu na Discord aplikaciji koju će registrirati i izvršiti Discord bot.
- **potpisivanje stripa:** zatim autor stripa može odlučiti da digitalno potpiše strip, čime se mijenja svojstvo digitalne kopije stripa.
- **verifikacija potpisa:** kolekcionari mogu provjeriti autentičnost potpisa pomoću blockchain preglednika

2.1.3 Otvaranje stripova

Otvaranje stripova na dReader platformi ključan je proces koji omogućava korisnicima interaktivno sudjelovanje u prikupljanju i čitanju digitalnih stripova.

Proces otvaranja stripova uključuje:

- **autentikacija i autorizacija:** prije pristupa sadržaju, korisnici moraju proći kroz sigurnosnu provjeru koja osigurava da samo ovlašteni korisnici mogu otvoriti stripove.
- **učitavanje sadržaja:** stripovi se pohranjuju na AWS S3 usluzi, te poslužuju korisnicima preko Cloudflare i CloudFront predmemorije (engl. *cache*), i to samo u slučaju ako korisnik posjeduje "otvoreni" primjerak stripa.
- **decentralizacija sadržaja:** u trenutku otvaranja stripa kolekcionar osigurava da će sadržaj stripa biti pohranjen na Arweave usluzi gdje će mu biti omogućen decentralizirani pristup sadržaju.
- **enkripcija sadržaja:** sadržaj pohranjen na Arweave usluzi mora biti enkriptiran i poslužen korisnicima na način da samo vlasnici relevantnog sadržaja (osobe koje posjeduju primjerak stripa) imaju mogućnost dekripcije i konzumacije sadržaja.

Pristupanje sadržaju preko Arweave usluge je sporije jer ne koristi predmemoriju i ostale tehnike optimizacije posluživanja digitalnog sadržaja. Proces pohranjivanja enkriptiranog sadržaja na Arweave će se napraviti preko Darkblock usluge.

Slika 4: Naslovnice stripova sa iskorištenim i neiskorištenim stanjem

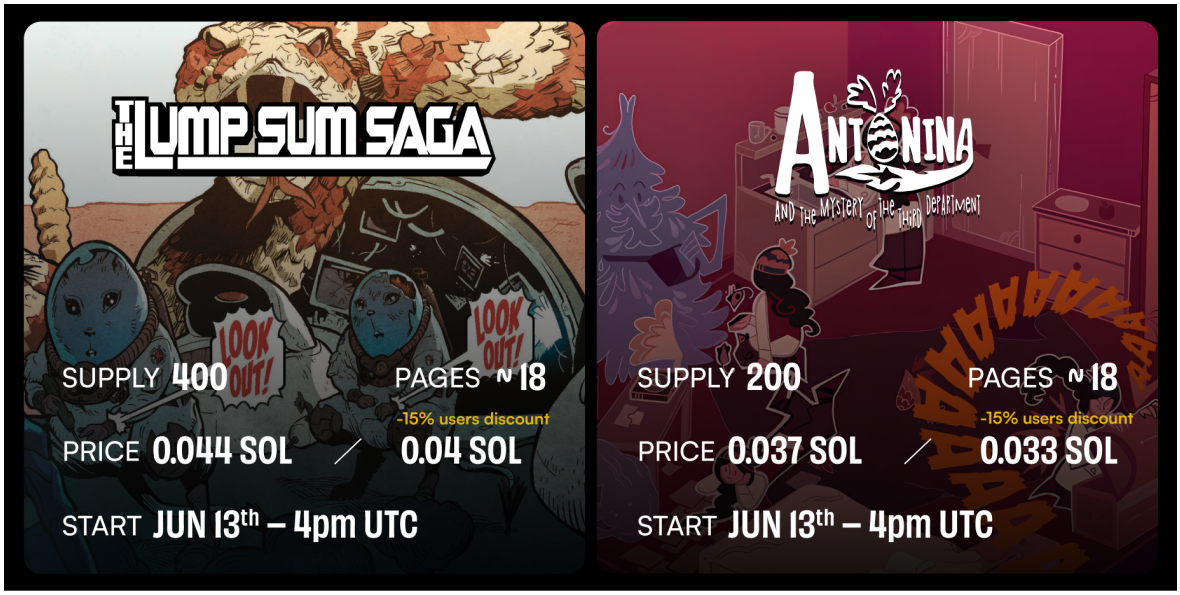
2.1.4 Prodaja stripova

Prodaja stripova ključna je komponenta dReader platforme koja omogućava autorima stripova monetizaciju svojih radova. Platforma je dizajnirana tako da maksimizira prihode autora dok istovremeno nudi privlačne opcije za kupce.

Prodaja stripova na dReader platformi koristi blockchain tehnologiju za transparentnost i sigurnost transakcija. Proces prodaje uključuje:

- **postavljanje stripa na prodaju:** autori stripova moraju moći staviti stripove na prodaju, određujući cijenu i druge relevantne podatke.
- **tokenizacija stripova:** Svaki strip se tokenizira kao NFT (engl. *non-fungible token*), što omogućava jedinstveno vlasništvo i lako praćenje povijesti transakcija.
- **decentralizacija sadržaja:** u trenutku otvaranja stripa kolekcionar osigurava da će sadržaj stripa biti pohranjen na decentraliziranoj usluzi Arweave.
- **transakcija:** kupci kupuju stripove spajajući svoj digitalni novčanik na platformu te koristeći kriptovalute kako bi platili i potpisali transakciju.

Postavke prodaje stripova potvrđuje i odobrava administrator platforme. Administrator može označiti strip spremnim za prodaju, koja će započeti na datum koji je definiran. U postavkama prodaje pojedinih stripova može se odrediti popust za registrirane članove dReader platforme.



Slika 5: Promotivni vizual za prodaju stripova

2.2. Analiza postojećih rješenja

Ova sekcija pruža analizu tržišta digitalnih stripova, fokusirajući se na ključne igrače kao što su VeVe, Webtoon i Global Comix. Svaka od ovih platformi pruža različite pristupe monetizaciji, distribuciji i korisničkom iskustvu, što direktno utječe na preference i odluke korisnika.

VeVe je digitalna platforma za kolekcionare koja nudi premium digitalne kolekcionarske predmete i stripove, te koristi blockchain tehnologiju u jako ograničenim uvjetima. Česte su pritužbe VeVe korisnika kako ne mogu izvući stripove i novce sa platforme jer ih zapravo ne posjeduju (sadržaj nije decentraliziran).

Webtoon i **Global Comix** su jedne od vodećih platformi za web stripove koje omogućuje autorima da objavljuju svoje stripove i dosegnu široku globalnu publiku. Platforme koriste tradicionalniji model distribucije bez upotrebe blockchain tehnologija i kolekcionarskih funkcionalnosti.

2.3. Odabir tehnologije i alata

Pri odabiru tehnologija za dReader, ključni su bili kriteriji kao što su skalabilnost, sigurnost, i sposobnost integracije s blockchain tehnologijom. Solana je odabrana kao temeljna tehnologija zbog svoje visoke propusnosti i niskih transakcijskih troškova, što je idealno za aplikaciju koja zahtijeva brze i česte transakcije kao što su kupnja i trgovina digitalnim stripovima.

Za razvoj korisničkog sučelja, odabran je Next.js zbog njegove učinkovitosti u generiranju statičkih stranica i prikazivanja na strani poslužitelja (engl. *server-side renderinga*), što je poboljšalo brzinu učitavanja stranica i ukupno korisničko iskustvo. React, korišten u kombinaciji s Next.js, omogućio je izgradnju interaktivnih sučelja s dinamičnim ažuriranjima sadržaja bez potrebe za ponovnim učitavanjem stranica. SCSS (engl. *Sassy Cascading Style Sheets*) je korišten za stilizaciju zbog svojih naprednih funkcionalnosti koje olakšavaju održavanje CSS koda, što je ključno u projektima velikog obima.

Na poslužitelju, NestJS je poslužio kao robusno rješenje za izgradnju poslužiteljskih aplikacija. Njegova arhitektura temeljena na modulima i injekciji ovisnosti olakšala je razvoj čistog, organiziranog koda koji je lako skalabilan. Integracija s AWS S3 omogućila je pouzdanu i skalabilnu pohranu digitalnih stripova, dok je Nodemailer korišten za upravljanje komunikacijom s korisnicima putem elektroničke pošte. Discord bot je razvijen kako bi se omogućila automatizacija interakcija s korisnicima unutar Discord zajednice, što je dodatno poboljšalo angažman i podršku korisnicima.

Sigurnost je bila od veće važnosti pri razvoju dReadera. Integracija Web3 tehnologija omogućila je sigurnu autentikaciju i transakcije koristeći blockchain, što je korisnicima pružilo sigurnost i transparentnost pri kupnji i trgovini stripovima. Metaplex protokol, korišten za stvaranje i upravljanje NFT-ovima, ključan je u implementaciji funkcionalnosti tokeniziranih digitalnih stripova.

3. Razvojni alati

U ovom poglavlju detaljno se opisuju tehnološki alati koji su korišteni za razvoj platforme dReader. Svaki odabrani alat ima ključnu ulogu u implementaciji različitih aspekata platforme, od razvoja korisničkog sučelja i poslužitelja, do sigurnosnih i infrastrukturnih rješenja.

3.1. Next.js

Next.js je razvojni okvir (engl. *framework*) koji omogućava izgradnju brzih web aplikacija koristeći moderne prakse web razvoja i oslanjajući se na razvojni okvir React. Njegova sposobnost prikazivanja na poslužiteljskoj strani (engl. *server-side rendering*, skraćeno SSR), generiranje statičnih stranica (engl. *static site generation*, skraćeno SSG) i optimizacija izvedbe čine ga idealnim izborom za projekte koji zahtijevaju visoku razinu interaktivnosti i optimalno korisničko iskustvo.

Za projekt je razvojni okvir Next.js, između ostaloga, izabran zbog njegove integracije s Reactom, robustnih opcija za routiranje, i ugrađene podrške za SEO. SSR i SSG funkcionalnosti razvojnog okvira Next.js omogućuju brzo učitavanje stranica i poboljšavaju vrijeme do interaktivnosti (engl. *time to interactive*, skraćeno TTI), što je ključno za održavanje korisničke angažiranosti na platformi koja se intenzivno oslanja na vizualni sadržaj. [1] U sklopu razvoja Next.js aplikacije koristit će se i tehnologije poput SCSS i ReactQuery.

ReactQuery je knjižnica za upravljanje asinkronim podacima i poslužiteljskog stanja u React aplikacijama. Njegova glavna funkcija je omogućavanje efikasnijeg dohvaćanja, cachinga, sinkronizacije i ažuriranja podataka, što znatno poboljšava izvođenje aplikacije i korisničko iskustvo. ReactQuery je odabran zbog svoje sposobnosti da pojednostavi rad s asinkronim podacima i smanji potrebu za složenim upravljanjem stanjem unutar komponenti Reacta. [2]

3.2. NestJS

NestJS je razvojni okvir za izgradnju učinkovitih, skalabilnih Node.js poslužiteljskih aplikacija. Koristi napredni JavaScript (ES6), izgrađen je uz potpunu podršku za TypeScript te kombinira elemente OOP-a (objektno orijentirano programiranje), FP-a (funkcionalno

programiranje) i FRP-a (funkcionalno reaktivno programiranje). [3]

NestJS ima snažnu integraciju s drugim knjižnicama i alatima, kao što su TypeORM za bazu podataka i Passport za autentikaciju, te u pozadini koristi robustne HTTP poslužiteljske okvire poput Expressa. U sklopu razvojnog okvira NestJS koristit će se i Nodemailer.

Nodemailer je modul za Node.js koji omogućava jednostavno slanje elektroničke pošte iz aplikacija. [4] Zbog svoje efikasnosti i široke podrške za različite pružatelje usluga elektroničke pošte, Nodemailer je postao popularan izbor za razvojne timove koji trebaju pouzdanu komunikaciju putem elektroničke pošte u svojim aplikacijama.

Za dReader platformu, Nodemailer je odabran zbog njegove sposobnosti da integrira različite transportne mehanizme elektroničke pošte, što omogućava timu da koristi lokalne SMTP (engl. *Simple Mail Transfer Protocol*) poslužitelje ili usluge trećih strana poput SendGrid-a za slanje elektroničke pošte.

3.3. Discord

Discord je popularna platforma za komunikaciju koja se koristi u mnogim zajednicama, uključujući obrazovne i poslovne. Discord nudi širok set alata za komunikaciju u realnom vremenu putem glasa, teksta i videa, te bogat ekosustav botova koji se mogu prilagoditi za automatizaciju različitih zadatka. Za dReader, Discord je odabran zbog svoje široke prihvaćenosti među mlađom publikom i mogućnosti izgradnje zajednice oko digitalnih stripova.

Discord botovi će se koristiti za slanje obavijesti urednicima dReader Discord poslužitelja, te za potpisivanje stripova i slične radnje koje služe svrsi izgradnje veze između autora stripova i kolekcionara, te moderatora Discord poslužitelja i članova dReader zajednice.

3.4. AWS S3 (engl. *Amazon Simple Storage Service*)

AWS S3 je usluga objektnog skladištenja koja nudi industrijski vodeću skalabilnost, dostupnost podataka, sigurnost i izvedbu. Ova usluga omogućava bilo kojem programeru da lako i sigurno pohranjuje i preuzima bilo koju količinu podataka s bilo kojeg mjesta na webu. [5] Za izradu projekta potrebno je postaviti AWS infrastrukturu, uključujući i usluge S3, te integrirati usluge AWS u dReader poslužiteljsku aplikaciju preko kompleta za razvoj softvera (engl. *software development kit*, skraćeno SDK).

3.5. Solana

Solana je visoko performantna blockchain tehnologija koja je dizajnirana za podršku skalabilnih decentraliziranih aplikacija i tržišta. Svojom jedinstvenom arhitekturom omogućava obradu više tisuća transakcija u sekundi uz izuzetno niske troškove transakcije. Ključna razlika između blockchain tehnologije poput Solane i tradicionalnih centraliziranih baza podataka je u načinu pohrane i verifikacije podataka. Dok se kod centraliziranih baza podataka svi podaci pohranjuju i kontroliraju na jednom mjestu (u vlasništvu pojedinačne organizacije), blockchain koristi distribuiranu mrežu čvorova koji zajednički potvrđuju i osiguravaju integritet podataka.

Solana, kao blockchain tehnologija, pruža veću otpornost na napade i manipulacije podacima jer nema jednog središnjeg entiteta koji upravlja podacima. Sve transakcije su javno dostupne, a podaci su nepromjenjivi jednom kada su zapisani u mrežu, čime se osigurava visok stupanj transparentnosti i sigurnosti, što je ključno u kontekstu tržišta digitalnih stripova i NFT-ova (Non-Fungible Tokens).

Solana koristi JSON (engl. *JavaScript Object Notation*) RPC (engl. *Remote Procedure Call*) API (engl. *Application Programming Interface*) za omogućavanje interakcija između klijentskih aplikacija i mreže Solana. Klijentske aplikacije izvršavaju upite i šalju transakcije na efikasan način, koristeći standardni format koji je lako integrirati i koristiti.

@solana/web3.js knjižnica je ključni alat za interakciju sa Solanom, koja apstrahira korištenje osnovnih funkcionalnosti Solaninih definiranih JSON RPC metoda, te ostalih funkcionalnosti poput generiranja para ključeva. U sklopu Solana alata koristit ćemo i Metaplex i Helius.

Metaplex je protokol na blockchain tehnologiji Solana koji omogućuje brzo, sigurno i efikasno kreiranje i upravljanje NFT-ovima. Metaplex nudi set standardiziranih alata i pametnih ugovora (engl. *smart contract*) koji olakšavaju programerima implementaciju različitih poslovnih logika na Solana blockchainu. Primjerice, Metaplex ima pametni ugovor otvorenog koda zvani "Candy Machine" koji ima mnoge postavke i služi za prodaju NFT-ova (engl. *Non Fungible Token*) na Solana blockchainu.

4. Razvoj i implementacija

4.1. Arhitektura sustava

Arhitektura dReader platforme temelji se na modularnom pristupu, gdje su različite funkcionalnosti odvojene u zasebne module koji komuniciraju putem definiranih API-a. Poslužitelj je izgrađen na NestJS frameworku koji omogućava jasno definiranu strukturu aplikacije, s modulima za AWS, Discord, upravljanje korisnicima, upravljanje stripovima i integraciju s blockchainom. Klijentska aplikacija koristi Next.js za optimizirano renderiranje i brzinu učitavanja, dok ReactQuery olakšava upravljanje asinkronim podacima i komunikaciju s poslužiteljem.

4.2. Razvoj poslužiteljskih funkcionalnosti

Poslužiteljski sustav dReader-a razvijen je koristeći NestJS, što omogućuje visoku modularnost i organiziranost koda. Svaka poslovna logika, kao što su upravljanje prodajom stripova ili registracija korisnika, implementirana je unutar odgovarajućih modula. Korištenje objekta za prijenos podataka (engl. *Data Transfer Objekata*, skraćeno DTO) osigurava se validacija podataka i transformiranje prije nego što stignu do poslovne logike, čime se dodatno povećava sigurnost i konzistentnost aplikacije. Integracija s blockchainom putem Solana RPC API-a i servisa Helius omogućuje brzo i sigurno provođenje transakcija, dok AWS S3 osigurava pouzdanu pohranu medijskih datoteka.

Projekt je započet sa standardnom NestJS naredbom za sučelje naredbenog retka (engl. *Command Line Interface*, skraćeno CLI) `nest new d-reader-backend` i instalacijom ovisnosti putem naredbe `npm install`.

Različite datoteke `.env` se koriste za postavljanje okruženja, uključujući razvojno, testno i produkcijsko okruženje. Svaka datoteka `.env` sadrži konfiguracijske varijable specifične za određeno okruženje, kao što su API ključevi, URL-ovi baza podataka i postavke za povezivanje s blockchainom.

S obzirom da konačna platforma ima preko pedeset vrijednosti u okruženju, rad će se fokusirati samo na one koje su bitne za implementirati funkcionalnosti koje su obuhvaćene u radu.

```
# POSTGRES
POSTGRES_USER="postgres"
POSTGRES_PASSWORD="postgres"
POSTGRES_DB="dreader"
DB_HOST="localhost"
DB_SCHEMA="public"

# Prisma database connection
DATABASE_URL="postgresql://${POSTGRES_USER}:${POSTGRES_PASSWORD}@${
   DB_HOST}:${DB_PORT}/${POSTGRES_DB}?schema=${DB_SCHEMA}&sslmode=
   prefer"

# Environment
NODE_ENV='dev'
```

Ispis 1: Početno stanje datoteke `.env`

Nakon što su ovisnosti instalirane i okruženje postavljeno, aplikacija se može pokrenuti pomoću naredbe CLI: `npm run start:dev`.

Poslužitelj je organiziran u module, gdje svaki modul predstavlja određenu funkcionalnost unutar aplikacije. Centralni modul `AppModule` u kojeg se uključuju svi ostali moduli, servisi, kontroleri (engl. *controllers*) i ostale funkcionalnosti potrebne za izvođenje aplikacije, definiran je u datoteci `app.module.ts`.

Početna točka aplikacije je `main.ts` u kojem se definira i pokreće funkcija `bootstrap` koja stvara novu instancu poslužiteljske aplikacije, definira ponašanje poslužitelja, i pokreće ga. Funkcija `bootstrap` pokreće inačnicu Nest tvornice sa postavkama definiranim u `AppModule`.

U sljedećim isječku koda je prikazan početni izgled `main.ts` datoteke, koji će se mijenjati kako se nove konfiguracije budu dodavale na poslužiteljsku aplikaciju.

```
import { ConfigService } from '@nestjs/config';
import { HttpAdapterHost, NestFactory } from '@nestjs/core';
import { PrismaClientExceptionFilter } from 'nestjs-prisma';
import { AppModule } from './app.module';
import { NestConfig } from './configs/config.interface';
import * as express from 'express';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  app.useGlobalPipes();
  app.enableShutdownHooks();

  const { httpAdapter } = app.get(HttpAdapterHost);
  app.useGlobalFilters(new PrismaClientExceptionFilter(httpAdapter)
   );

  const configService = app.get(ConfigService);
  const nestConfig = configService.get<NestConfig>('nest');

  app.use(express.static('public'));
  await app.listen(process.env.PORT || 3005);
}

bootstrap();
```

Ispis 2: Sadržaj main.ts datoteke

Posljednji dio aplikacije koji je potreban za uspješno pokreniti poslužiteljsku aplikaciju jest `config.ts` u kojem se definiraju postojana svojstva aplikacije koja se dijele u različitim okruženjima, bilo razvojnim bilo produkcijskim.

Tijekom izrade aplikacije `config.ts` datoteka će se popunjavati sa novim svojstvima, kako se definiraju nove vrijednosti za različite funckionalnosti. Za početak, specifikacija je jednostavna i potrebno je samo definirati `port` svojstvo.

```
import { Config } from './config.interface';

const config: Config = {
  nest: { port: 3005 },
};

export default (): Config => config;
```

Ispis 3: Početni sadržaj config.ts datoteke

Ovako postavljen temelj poslužiteljske aplikacije čini cjelinu na kojoj se mogu lako dodavati nove funkcionalnosti poput registracije korisnika, potpisivanje stripova i tako dalje.

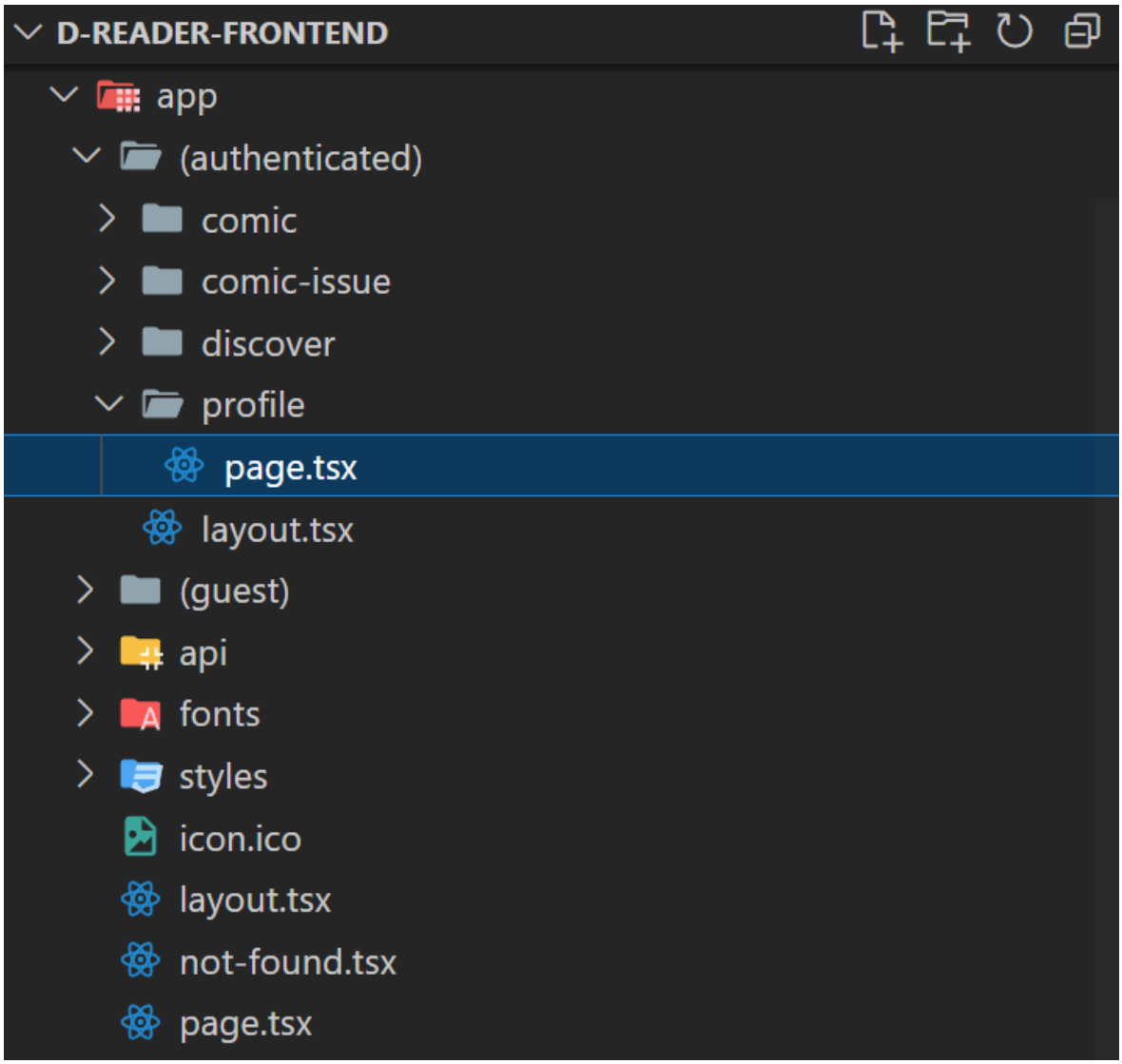
4.3. Razvoj komponenata korisničkog sučelja

Razvoj klijentskih komponenata uključuje korištenje različitih tehnologija kao što su Next.js za razvoj web aplikacija, SCSS za stilizaciju, ReactQuery za upravljanje podacima, te integracija s blockchainom putem knjižnice Solana web3.js.

Projekt je započet sa naredbom CLI `npx create-next-app@latest` te koristi Node.js i npm za upravljanje ovisnostima. Instalacija ovisnosti vrši se pomoću naredbe `npm install` i pokretanje aplikacije pomoći naredbe `npm run dev`.

Klijentska aplikacija je organizirana u komponente, gdje svaka komponenta predstavlja određeni dio korisničkog sučelja. Korištenje Next.js omogućuje jednostavno upravljanje stranicama, dok React.js omogućuje kreiranje dinamičnih i interaktivnih komponenti.

Svaka stranica aplikacije definirana je kao `page.tsx` datoteka unutar odgovarajućeg poddirektorija unutar `app` direktorija. Next.js automatski generira rute na temelju strukture direktorija i datoteka, omogućujući lako upravljanje navigacijom unutar aplikacije. Primjerice, za datoteku na putanji `app/discover/page.tsx` će se izgenerirati putanja na web aplikaciji `localhost:3000/discover`.



Slika 6: Struktura `app` direktorija, sa istaknutom stranicom `/app/profile/page.tsx`

Komponente koje se koriste na različitim stranicama definirane su unutar `components` direktorija. Ove komponente su modularne i ponovno upotrebljive, što omogućuje brzu izradu i održavanje korisničkog sučelja.

Neke komponente su atomarne, odosno nemaju nikakvu logiku i služe uređivanju osnovnih elemenata web aplikacije poput gumbova, slika i sličnog. Druge komponente su "pametnije" i sadržavaju u sebi logiku, funkcionalnosti, složenije su izvedenosti, i služe implementiranju poslovne logike. Primjerice komponenta `ComicItem.tsx` koja u web aplikaciji predstavlja digitalni strip.

U primjerku ispod prikazana je komponenta `Button` u datoteci `Button.tsx`.

```
import { ButtonHTMLAttributes, DetailedHTMLProps } from 'react'
import clsx from 'clsx'

export interface ButtonProps extends DetailedHTMLProps<
   ButtonHTMLAttributes<HTMLButtonElement>, HTMLButtonElement> {}

const Button: React.FC<ButtonProps> = ({ type = 'button', className
   , ...props }) => {
  return <button className={clsx(className)} type={type} {...props}
    />
}

export default Button
```

Ispis 4: komponenta Button na putanji `/components/buttons/Button.tsx`

Različite datoteke `.env` se koriste za postavljanje klijentskog okruženja, uključujući razvojno, testno i produkcijsko okruženje.

Radi jednostavnosti, rad će se fokusirati samo na razvojnu konfiguraciju lokalnog okruženja, i izostaviti će vrijednosti okruženja koje nisu potrebne za implementaciju funkcionalnosti koje će se obrazložiti u ovome radu.

```
PORT=3000
NODE_ENV='development'
NEXT_PUBLIC_API_ENDPOINT='http://localhost:3005'
GOOGLE_AUTH_CLIENT_ID='...'
GOOGLE_AUTH_CLIENT_SECRET='...'
NEXTAUTH_SECRET='...'
```

Ispis 5: Početno stanje datoteke `.env`

4.4. Integracija s blockchainom

Komuniciranje sa blockchainom je potrebno implementirati na više razina: klijentskoj i poslužiteljskoj strani.

Na poslužiteljskoj strani će se definirati par ključeva koji će se ponašati kao digitalni novčanik koji komunicira sa mrežom Solana. Kako par ključeva sadrži osjetljive podatke poput "tajnoga ključa", ti podaci će se morati šifrirati prije nego se postave u datoteku `.env`. Na poslužitelju taj par ključeva ćemo zvati novčanik riznice (engl. *treasury wallet*).

```
TREASURY_PRIVATE_KEY="U2FsdG...rC5zm3Dg=="
TREASURY_SECRET="bBV9...G&Me"
```

Ispis 6: Primjer novododanih vrijednosti u datoteci `.env`

U datoteci `src/utils/metaplex.ts` definira se funkcija `initMetaplex()` kojoj je zadaća inicijaliziranje pomoćne klase `Metaplex`, koja u sebi sadrži RPC vezu na blockchain. Funkcionalnosti povezivanja na mrežu Solana će implementirati pomoću knjižnice `@metaplex-foundation/js`, koja se u pozadini oslanja na `@solana/web3.js`.

```
import { Metaplex, keypairIdentity } from '@metaplex-foundation/js'
import { Connection, Keypair } from '@solana/web3.js';
import * as AES from 'crypto-js/aes';
import * as Utf8 from 'crypto-js/enc-utf8';

const getTreasuryKeypair = () => {
  const treasuryWallet = AES.decrypt(
    process.env.TREASURY_PRIVATE_KEY,
    process.env.TREASURY_SECRET,
  );
  const treasuryKeypair = Keypair.fromSecretKey(
    Buffer.from(JSON.parse(treasuryWallet.toString(Utf8))),
  );
  return treasuryKeypair;
};
```

```
export function initMetaplex(endpoint?: string) {
  const connection = new Connection(endpoint, 'confirmed');
  const treasuryKeypair = getTreasuryKeypair();
  const metaplex = new Metaplex(connection).use(
    keypairIdentity(treasuryKeypair),
  );

  return metaplex;
}

export const metaplex = initMetaplex();
```

Ispis 7: sadržaj `metaplex.ts` datoteke

Ovako definirana globalna konstanta `metaplex` se može koristiti u različitim servisima u aplikaciji, kao primjerice servis za izradu blockchain transakcija, definiran u datoteci `transaction.service.ts`.

```
import { Metaplex } from '@metaplex-foundation/js';
import { metaplex } from '../utils/metaplex';
import { Injectable } from '@nestjs/common';

@Injectable()
export class TransactionService {
  private readonly metaplex: Metaplex;

  constructor() { this.metaplex = metaplex; }

  const constructCandyMachineTransaction() {
    this.metaplex.candyMachinesV2(...)
  }
}
```

Ispis 8: pojednostavljeni sadržaj `transaction.service.ts` datoteke

Klijentska integracija oslanja se na knjižnice `@solana/wallet-adapter-react` i `@solana/wallet-adapter-react-ui`. Potrebno je napraviti *React Context* koji će uključiti kontekst iz navedenih knjižnica, koji će omogućiti komuniciranje sa blockchainom i digitalnim novčanicima na web aplikaciji.

Datoteka `ClientContextProvider.tsx` će biti napravljena, koja će se koristiti u korijenskoj `layout.tsx` datoteci.

```
'use client'

import { createContext, useContext } from 'react'
import { WalletProvider } from '@solana/wallet-adapter-react'
import { WalletModalProvider } from '@solana/wallet-adapter-react-
   ui'
import { useWalletAdapter } from '@/hooks/useWalletAdapter'

export const ClientContext = createContext(null)

const ClientContextProvider: React.FC<{ children: React.ReactNode
   }> = ({ children }) => {
  const wallets = useWalletAdapter()

  return (
    <WalletProvider wallets={wallets}>
      <WalletModalProvider>{children}</WalletModalProvider>
    </WalletProvider>
  )
}

export default ClientContextProvider

export const useClientContext = (): null => useContext(
   ClientContext)
```

Ispis 9: pojednostavljeni sadržaj `ClientContextProvider.tsx` datoteke

4.5. Registracija korisnika

Za registraciju korisnika potrebno je prvo implementirati funkcionalnosti na poslužitelju, počevši sa izradom autorizacijskih klasa i `auth.module.ts` datoteke.

Tip `JwtPayload` će biti napravljen, koji će označavati sadržaj JWT-a (engl. *JSON Web Token*), odnosno osnovne podatke o autentificiranome korisniku: identifikator, adresa elektroničke pošte, ime i uloga.

```
type JwtPaylod = {
  id: number;
  email: string;
  name: string;
  role: Role;
};
```

Ispis 10: `Jwt` tip

U `auth.service.ts` je potrebno implementirati sljedeće funkcionalnosti:

- izdavanje pristupnog tokena (engl. *access token*) autoriziranim korisnicima
- izdavanje tokena za osvježavanje korisničke sesije (engl. *refresh token*)
- validacija JWT-a

`generateAccessToken` je funkcija čija je svrha izdavati pristupni token svim korisnicima koji generiraju odgovarajući `JwtPayload`.

`generateRefreshToken` je funkcija čija implementacija je manje bitna pa će, radi jednostavnosti i preglednosti završnoga rada, biti izostavljena. U pravilu pristupni token ima jako kratki životni vijek, koji se produljuje sa osvježavajućim tokenom koji ima dulju životni vijek.

`validateJwt` funkcija uzima `JwtPayload` od korisnika, validira podatke, i traži odgovarajućeg korisnika na osnovu pruženih podataka. U slučaju da je *payload* iskvaren ili zastario, moguće da se ne pronađe odgovarajući korisnik, što će prouzročiti grešku.

```
import { Injectable } from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';
import { PrismaService } from 'nestjs-prisma';
import { JwtPayload } from './dto/authorization.dto';

@Injectable()
export class AuthService {
  constructor(
    private readonly prisma: PrismaService,
    private readonly jwtService: JwtService,
  ) {}

  private generateAccessToken(payload: JwtPayload): string {
    return this.jwtService.sign(payload);
  }

  private generateRefreshToken(payload: JwtPayload) {
    // arbitrary implementation
  }

  async validateJwt(payload: JwtPayload): Promise<JwtPayload> {
    const user = await this.prisma.user.findUnique({
      where: { id: payload.id },
    });

    if (!user) throw new Error('User not found');
    return user;
  }
}
```

Ispis 11: Sadržaj `auth.service.ts` datoteke

Zatim je potrebno specificirati JWT `PassportStrategy` u datoteci `jwt.strategy.ts`. *Passport* je najpopularnija *node.js* knjižnica za autentikaciju, dobro poznata u *JavaScript* za-

jednici. Ta knjižnica se implementira pomoću modula `@nestjs/passport`.

Prvo se specificira nova vrijednost `JWT_ACCESS_SECRET='MNEY...wtnJ'` u datoteci `.env`. Zatim se definira nova datoteka `jwt.strategy.ts` sa klasom `JwtStrategy` koja nasljeđuje `PassportStrategy` i implementira funkcionalnost za validaciju JWT-a.

```
import { Strategy, ExtractJwt } from 'passport-jwt';
import { PassportStrategy } from '@nestjs/passport';
import { Injectable } from '@nestjs/common';
import { AuthService } from './auth.service';
import { ConfigService } from '@nestjs/config';
import { JwtPayload } from './dto/authorization.dto';

@Injectable()
export class JwtStrategy extends PassportStrategy(Strategy) {
  constructor(
    private readonly authService: AuthService,
    readonly configService: ConfigService,
  ) {
    super({
      jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearer(),
      secretOrKey: configService.get('JWT_ACCESS_SECRET'),
    });
  }

  async validate(payload: JwtPayload) {
    return await this.authService.validateJwt(payload);
  }
}
```

Ispis 12: Sadržaj `jwt.strategy.ts` datoteke

Preostalo je implementirati navedene funkcionalnosti u `auth.controller.ts` te spojiti autorizacijske servise i kontroler u `auth.module.ts` koji će se u konačnici, uvesti u centralni `AppModule` koji se nalazi u `app.module.ts` datoteci.

```
import { Controller, Post, Get, Body } from '@nestjs/common';
import { AuthService } from './auth.service';
import { UserService } from '../user/user.service';
import { LoginDto } from '../types/login.dto';
import { RegisterDto } from '../types/register.dto';

@Controller('auth')
export class AuthController {
  constructor(
    private readonly authService: AuthService,
    private readonly userService: UserService,
  ) {}

  /* register a new user */
  @Post('user/register')
  async registerUser(@Body() registerDto: RegisterDto) {
    const user = await this.userService.register(registerDto)
    return this.authService.authorizeUser(user);
  }

  /* login an existing user */
  @Get('user/login')
  async loginUser(@Body() loginDto: LoginDto) {
    const user = await this.userService.login(loginDto);
    return this.authService.authorizeUser(user);
  }
}
```

Ispis 13: Sadržaj `auth.controller.ts` datoteke

Radi očuvanja jednostavnosti završnog rada, funkcije `userService.register` i `userService.login` će ostati neodređene i slobodne za osobnu interpretaciju implementacije. U funkciji `register` će se validirati unos korisnika i unijeti novi korisnik u bazu podataka, dok će za `login` se validirati unos korisničkog identifikatora i lozinke.

Prilikom registracije novog korisnika poslužiteljske aplikacije često šalju elektroničku poštu potvrde izrade novog računa. U slučaju dReader poslužitelja, funkcionalnost je implementirana u novome servisu `mail.service.ts` sa sljedećom funkcijom:

```
async sendUserRegisteredEmail(user: User) {
const verificationToken = this.authService.generateEmailToken(user)
   ;

await this.mailerService.sendMail({
  to: user.email,
  subject: 'Account created!',
  template: 'userRegistered',
  context: {
    name: user.name,
    actionUrl: 'https://localhost:3005/verify-email/' +
   verificationToken,
  },
});
}
```

Ispis 14: `sendUserRegisteredEmail` funkcija

`this.authService.generateEmailToken(user)` je arbitrarna funkcija koja na osnovu korisničke elektroničke pošte generira privremeni pristupni token.

`this.mailerService` je instanca `MailerService` klase koja se uvozi iz knjižnice `@nestjs-modules/mailer`. `MailerService` u pozadini implementira `Pug` framework za definiranje oblika elektroničke pošte.

Pug je jednostavan i moćan predložak (engl. *templating*) jezik za Node.js koji omogućuje brže i intuitivnije pisanje HTML-a (engl. *Hypertext Mark-up Language*) s ugrađenom podrškom za varijable, petlje i uvjetne izjave, što ga čini idealnim za dinamično generiranje elektroničke pošte.

Primjer Pug datoteke je sljedeći:

```
extends userBase.pug

block title
    h1(style='font-size: 28px;') Welcome #{name}!
block content
    p(style='font-size: 18px;') Your account has been created!
block action
    a.action-link.yellow-bg(href=actionUrl) Verify email
```

Ispis 15: Pojednostavljeni sadržaj `userRegistered.pug` datoteke

Tako definirani autentikacijski servisi i kontroleri su spremni za registriranje u novi modul `auth.module.ts`.

```
import { Module } from '@nestjs/common';
import { AuthService } from './auth.service';
import { PassportModule } from '@nestjs/passport';
import { JwtModule } from '@nestjs/jwt';
import { AuthController } from './auth.controller';
import { JwtStrategy } from './jwt.strategy';
import { PasswordService } from './password.service';

@Module({
  imports: [
    PassportModule.register({ defaultStrategy: 'jwt' }),
    JwtModule.register({ secret: process.env.JWT_SECRET }),
  ],
  controllers: [AuthController],
  providers: [AuthService, PasswordService, JwtStrategy],
})
```

Ispis 16: Sadržaj `auth.module.ts` datoteke

Preostala je implementacija autorizacije (registracije) korisnika sa klijentske strane, koja započinje sa definiranjem tipa `RegisterData`.

```
export interface RegisterData {
  name: string
  email: string
  password: string
}
```

Ispis 17: `RegisterData` tip

React Hooks su posebne funkcije u Reactu koje omogućuju korištenje stanja i drugih React značajki u funkcijskim komponentama. Prije uvođenja Hooks-a, state i životni ciklus komponenti bili su dostupni samo u klasnim komponentama.

Sa *React hooks* se smanjuje potreba za korištenjem klasa i omogućuju pisanje kompaktnijeg i čitljivijeg koda, omogućuje lakša podjela i ponovno korištenje logike stanja i efekata između komponenti. Nadalje, potiču pisanje manjih, specifičnih funkcijskih komponenti, što dovodi do bolje organizacije i održavanja koda.

React Query je snažan alat za upravljanje poslužiteljskim stanjem u React aplikacijama, koji pojednostavljuje rukovanje podatkovnim dohvatom, predmemoriranjem (engl. *caching*), sinkronizacijom i ažuriranjem podataka. Umjesto ručnog pisanja kompleksnih logika za dohvat podataka i rukovanje asinkronim operacijama, React Query omogućuje jednostavnu integraciju s API-ima putem intuitivnog sučelja, automatizirajući proces dohvaćanja i osvježavanja podataka.

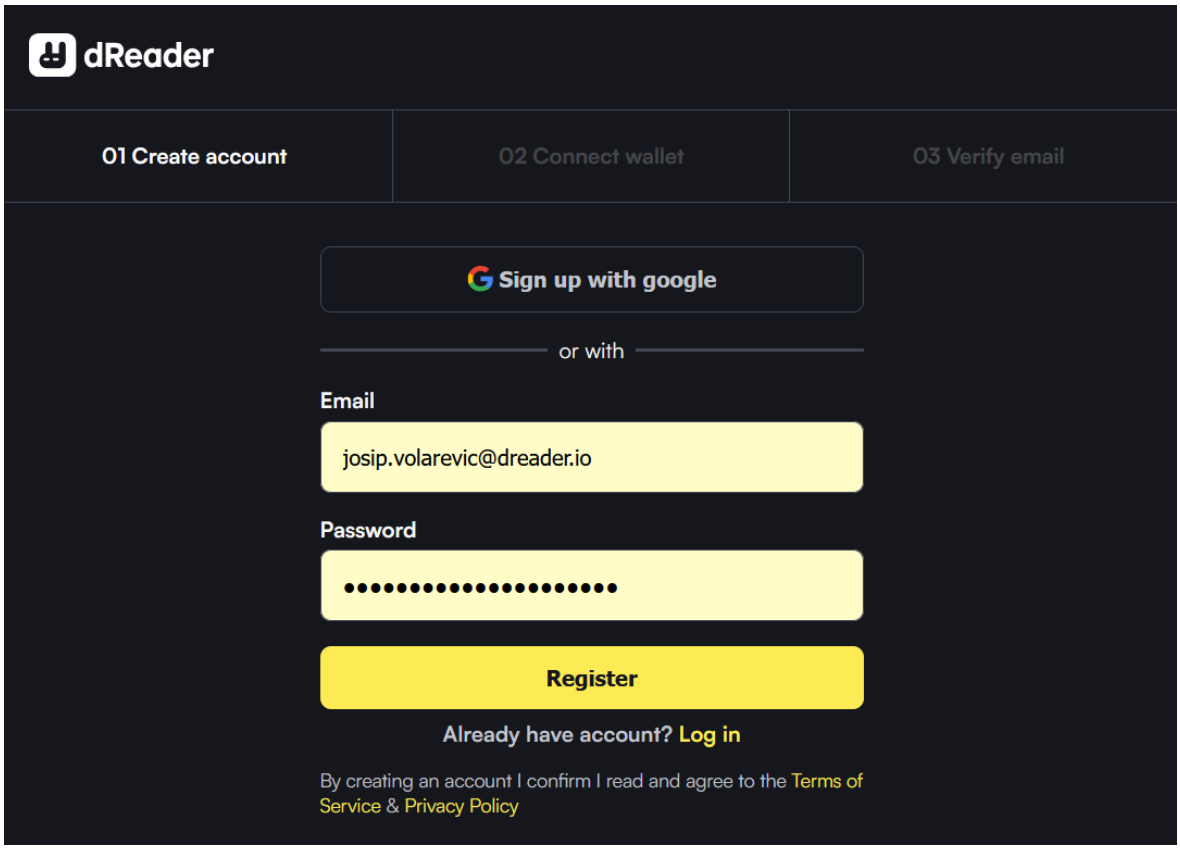
U klijentskoj aplikaciji definira se `useRegisterUser` hook koji implementira logiku komuniciranja sa poslužiteljem u svrhu registracije korisnika i dobivanja pristupnog tokena.

`useUserAuth.addAuthorization` je funkcija koja dodaje korisničku autorizaciju u lokalno stanje (*context*) i HTTP (engl. *Hypertext Transfer Protocol*) zaglavlje. Iz knjižnice `react-query` uzima se funkcija `useMutation` koje se koristi za optimiziranje `POST` i `PATCH` HTTP zahtjeva na poslužitelj. Za ostvarivanje veze na poslužitelj koristi se instanca axios-a iz knjižnice `axios`.

```
import { useUserAuth } from 'providers/UserAuthProvider'
import { RegisterData } from 'models/auth/register'
import { useMutation } from 'react-query'
import axios from 'axios'

export const http = axios.create({ baseURL: 'http://localhost:3005'
    })

const register = async (request: RegisterData) => {
  const response = await http.post('auth/user/register', request)
  return response.data
}

export const useRegisterUser = () => {
  const { addAuthorization } = useUserAuth()

  return useMutation({
    mutationFn: (request: RegisterData) => register(request),
    onSuccess: (data) => {
      const user = addAuthorization(data)
      console.log(`Welcome ${user.name}!`)
    },
  })
}
```

Ispis 18: `useRegisterUser` hook

Za kraj je preostalo pozvati `useRegisterUser` *hook* sa korisničkim podacima unesenim u klijentsku formu, što je implementirano preko `react-hook-form` knjižnice. Navedena implementacija ima nekoliko stotina linija koda pa će, radi čitljivosti završnoga rada, biti preskočena.



Slika 7: Izgled stranice za registraciju korisnika

4.6. Digitalno potpisivanje stripova

Digitalno potpisivanje je omogućeno autorima stripova kako bi nagradili kolekcionare svojih stripova. Kako je dReader zajednica okupljena na Discord poslužitelju, ova funkcionalnost je napravljena preko Discord botova.

Sveukupna implementacija je kompleksna, pa je u svrhu lakšeg razumijevanja rada fokus stavljen na najbitnije dijelove implementacije. Korištena je knjižnica `discord.js` za olakšanu interakciju sa Discord API-ima.

Za početak se definira funkcija `CREATE_COMIC_RESPONSE` koja oblikuje poruku koju će Discord bot izbaciti nakon zahtjevanja digitalnog potpisa i nakon potpisivanja. Funkcija vraća objekt koji je definiran tipom `InteractionReplyOptions` iz `discord.js` knjižnice. Iza se definira `sign-comic.command.ts` datoteka sa funkcijom za zahtjevanje potpisivanja i funkcijom za potpisivanjem stripa.

```
import { InteractionReplyOptions } from 'discord.js';

export const CREATE_COMIC_RESPONSE = ({
  content,
  imageUrl,
  comicName,
  components,
  ephemeral,
  mentionedUsers,
}): InteractionReplyOptions => {
  return {
    content,
    embeds: [
      {
        image: { url: imageUrl },
        fields: [
          {
            name: 'Comic Name',
            value: comicName,
            inline: true,
            fetchReply: false,
          },
        ],
      },
    ],
    components,
    ephemeral,
    allowedMentions: { users: mentionedUsers },
  };
};
```

Ispis 19: `CREATE_COMIC_RESPONSE` funkcija

U datoteci `sign-comic.command.ts` definira se klasa `SignComicCommand` koja u sebi sadrži polje `metaplex` koje označava instancu `Metaplex` klase. Nadalje, klasa sadrži i instancu `prisma` servisa koji komunicira sa bazom, te instancu `transactionService` koji predstavlja servis za manipuliranje Solana transakcijama.

```
// ... imports

@Command({ name: 'get-signature' })
export class SignComicCommand {
  private readonly metaplex: Metaplex;

  constructor(
    private readonly transactionService: TransactionService,
    private readonly prisma: PrismaService,
  ) {
    this.metaplex = metaplex;
  }

  ...
}
```

Ispis 20: Postavljanje `sign-comic.command.ts` datoteke

Sljedi implementacija `onRequestSignature` i `onComicSignedButtonClick` funkcija. U klasi `SignComicCommand` definiraju se dvije funkcije, od kojih prva funkcija, `onRequestSignature` omogućuje korisniku da zahtjeva potpis od autora stripa.

```
@Handler()
async onRequestSignature(@IA() options) {
  const params = options;
  const { interaction } = params;
  await interaction.deferReply({ ephemeral: true });

  const { user, address } = params;
```

```
  const response = await fetchDigitalAsset(address);
  const { offChainMetadata, name } = response;

  const component =
    new ActionRowBuilder().addComponents(
      new ButtonBuilder()
        .setCustomId(`${user.id};${address}`)
        .setLabel(`Sign comic`),
    );

  await interaction.followUp(CREATE_COMIC_RESPONSE({ ... }));
}
```

Ispis 21: Pojednostavljeni sadržaj `onRequestSignature` funkcije

Druga funkcija, `onComicSignedButtonClick` definira rezultat pritiska na *Sign comic* tipku od strane autora stripa.

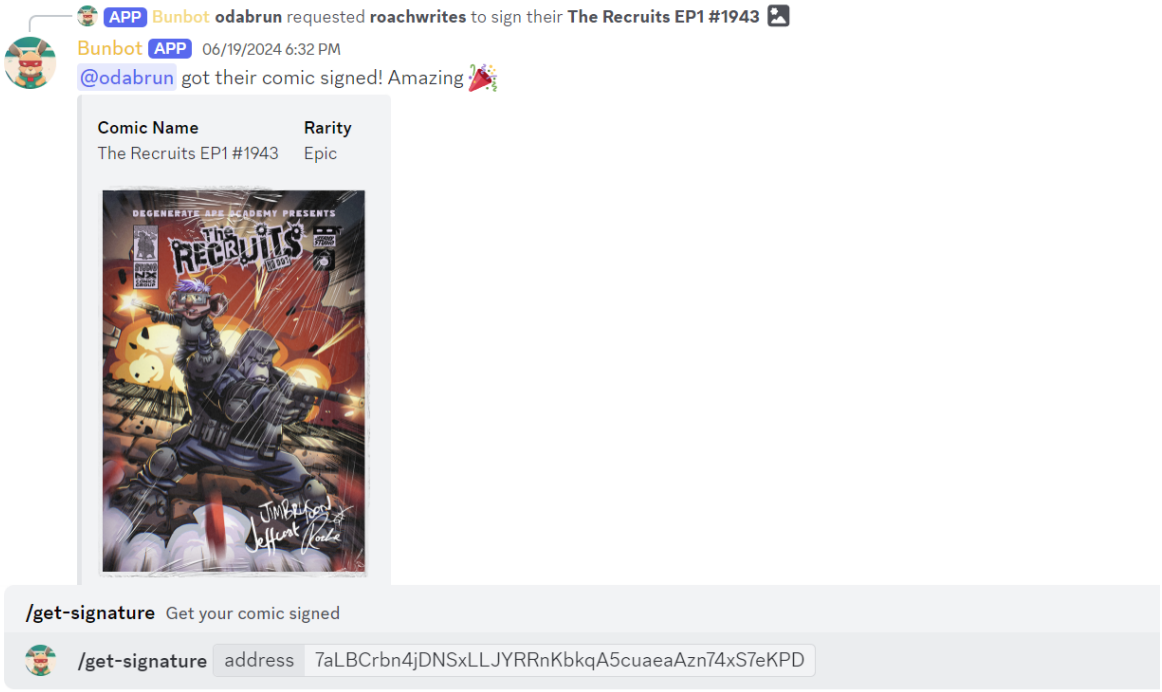
```
@On('interactionCreate')
async onComicSignedButtonClick(
  @EventParams() eventArgs: ClientEvents['interactionCreate'],
) {
  const buttonInteraction = eventArgs[0] as ButtonInteraction;
  await buttonInteraction.deferReply({ ephemeral: true });
  const user = buttonInteraction.customId.split(';')[0];
  const address = buttonInteraction.customId.split(';')[1];

  const latestBlock = await this.metaplex.connection.
   getLatestBlockhash();

  const asset = await this.prisma.digitalAsset.findUnique({
    where: { address: address } });
  const rarity = asset.metadata.rarity;
```

```
  const collection = asset.metadata.collection;

  const rawTransaction =
    await this.transactionService.createSignComicTransaction(
      new PublicKey(address),
      this.metaplex.identity().publicKey,
    );

  const tx = VersionedTransaction.deserialize(
    Buffer.from(rawTransaction, 'base64'),
  );

  tx.sign([this.metaplex.identity()]);

  const signature = await this.metaplex.connection.
   sendRawTransaction(tx.serialize());
  await this.metaplex.connection.confirmTransaction({ signature,
   ...latestBlock });

  await buttonInteraction.followUp(CREATE_COMIC_RESPONSE({ ... }));
}
```

Ispis 22: Pojednostavljeni sadržaj `onComicSignedButtonClick` funkcije

Osim implementacije na poslužitelju, potrebno je namjestiti Discord bota na dReader Discord poslužitelju kako korisnici imali priliku koristiti naredbe, koje bi taj bot osluškivao.

Kao krajnji rezultat, kolekcionarima stripova je omogućeno na Discord poslužitelju dReader zajednice pokreniti naredbu `/get-signature`, na koju autor stripa može reagirati sa klikom na `sign comic` tipku, koja će posljedično aktivirati digitalno potpisivanje stripa.



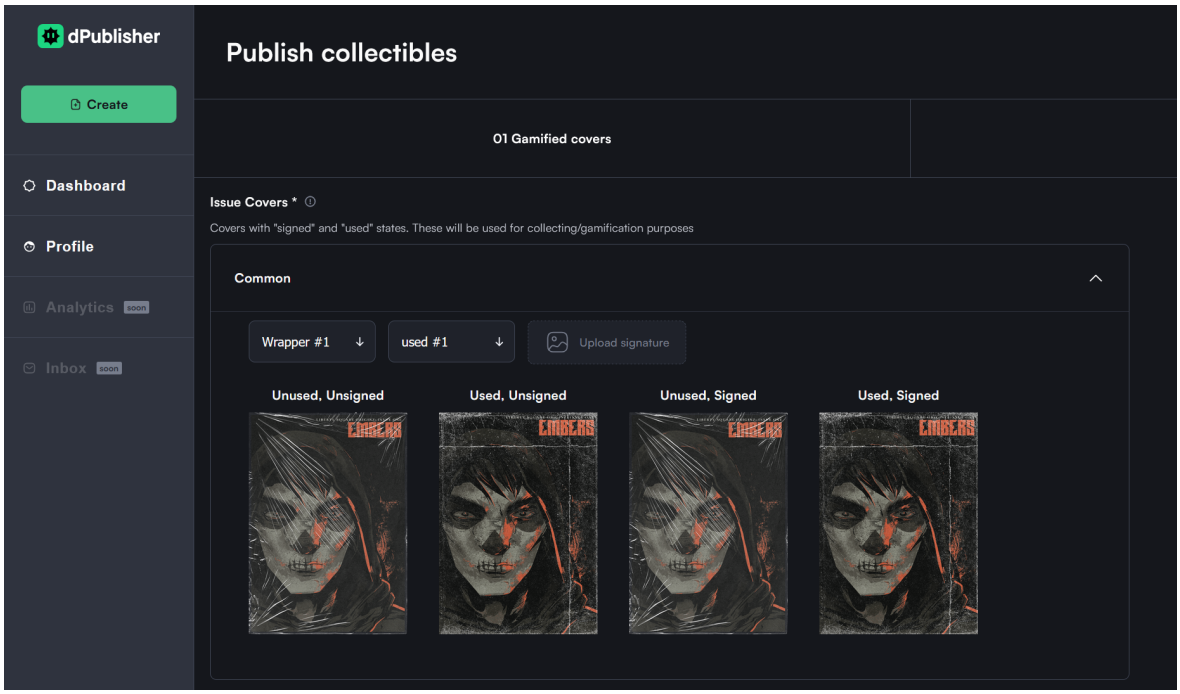
Slika 8: Primjer potpisivanja stripova na Discord poslužitelju

4.7. Prodaja stripova

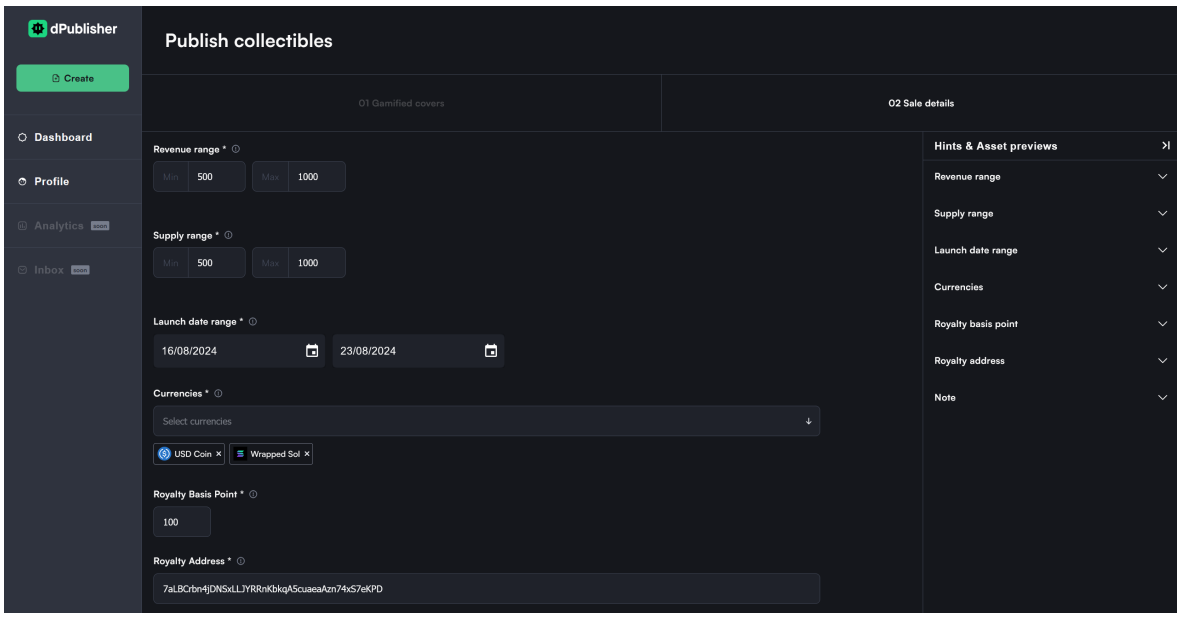
Prodaja stripova na dReader platformi integrira nekoliko ključnih tehnologija kako bi omogućila sigurno i transparentno iskustvo za korisnike. Cijeli proces prodaje temelji se na blockchain tehnologiji Solana, čime se osigurava decentralizacija i nepovredivost podataka.

Prvi korak u procesu prodaje stripova jest njihova tokenizacija. Svaki strip na dReader platformi se tokenizira kao NFT, što omogućava jedinstveno vlasništvo nad stripom. Ovaj proces je implementiran koristeći Metaplex `Candy Machine` pametni ugovor, koji automatizira kreiranje i distribuciju NFT-ova.

Autori stripova imaju mogućnost postaviti svoje stripove na prodaju kroz dPublisher sučelje. Pri tome definiraju cijenu, varijante naslovnica, količina stripova u opskrbi, te ostale relevantne informacije. Nakon što su postavke prodaje definirane, i prodaja odobrena od strane administratora aplikacije, stripovi se stavljaju na prodaju i dostupni su kolekcionarima.



Slika 9: Sučelje za uređivanje naslovnica stripa



Slika 10: Sučelje za određivanje postavki prodaje stripa

Nakon što su postavke prodaje stripa konfigurirane i odobrene, započet će proces postavljanja stripa na blockchain tehnologiju Solana i mrežu Arweave. Strip će biti dostupan na prodaju tek nakon što se je cijeli sadržaj učitao na Arweave i `Candy Machine` pametni ugovor je postavljen na blockchain.

Kako je cijeli proces objavljivanja stripova i pripreme za prodaju jako kompleksan, zavr-

šni rad će se fokusirati samo na pojedine dijelove koji se tiču prodaje stripa, a koji uključuju korištenje tehnologije koja se dosad nije spomenila u završnome radu.

U `comic-issue.controller.ts` definira se klasa `ComicIssueController` koja ima funkciju `updateStatefulCovers`, koja autorima stripova omogućuje da učitaju različite varijante naslovnice, sa različitim svojstvima (sa potpisom ili bez, sa omotom ili sa istrošenim naslovnicama).

```
@ComicIssueOwnerAuth()
@Post('update/:id/stateful-covers')
@UseInterceptors(AnyFilesInterceptor({}))
async updateStatefulCovers(
  @Param('id') id: string,
  statefulCoverDto: [CreateStatefulCoverDto],
) {
  await this.comicIssueService.updateStatefulCovers(
   statefulCoverDto, +id);
}
```

Ispis 23: Funkcija `updateStatefulCovers` u `comic-issue.controller.ts`

Gornji isječak kôda prikazuje sljedeće:

- `@ComicIssueOwnerAuth()`: dekorator koji validira da korisnik koji pristupa API funkciji je autor stripa
- `@UseInterceptors()` funkcija koja validira primljeni objekt kao objekt koji sadrži datoteke
- `comicIssueService.updateStatefulCovers()` funkcija koja u bazi ažurira naslovnice stripa

U funkciji `comicIssueService.updateStatefulCovers()` je definirana poslovna logika zamjenjivanja neažurnih varijanti naslovnica stripa sa novim varijantama. Demonstrira se korištenje `prisma.$transaction` funkcije koja osigurava da se stare naslovnice izbrišu samo i samo ako su nove naslovnice upisane.

```
async updateStatefulCovers(
  coversDto: CreateStatefulCoverDto[],
  comicIssueId: number,
) {
  const comicIssue = await this.prisma.comicIssue.findUnique({
    where: { id: comicIssueId },
    include: { statefulCovers: true },
  });
  const oldCovers = comicIssue.statefulCovers;
  const areCoversUpdated = !!oldCovers;

  const newCoversData = await this.createManyCoversData(coversDto,
   comicIssue);
  const oldFileKeys = oldCovers.map((cover) => cover.image);

  if (areCoversUpdated) {
    const deleteCovers = this.prisma.statefulCover.deleteMany({
      where: { comicIssueId },
    });

    const createCovers = this.prisma.statefulCover.createMany({
      data: newCoversData,
    });

    await this.prisma.$transaction([deleteCovers, createCovers]);
  } else {
    await this.prisma.statefulCover.createMany({ data:
   newCoversData });
  }
  await this.s3.deleteObjects(oldFileKeys);
}
```

Ispis 24: Funkcija `updateStatefulCovers` u `comic-issue.service.ts`

`s3.service.ts` datoteka definira `s3Service` koji u sebi sadrži podatke o servisu AWS S3, i funkcionalnosti za manipuliranje datotekama pohranjenim na S3 *buckets*.

U `.env` definiraju se nova svojstva koja su vezana za funkcionalnosti S3. Jedan dio podataka se odnosi na osjetljive informacije potrebne za pristup autoriziranome servisu AWS S3, a drugi dio podataka je vezan za CDN (engl. *content delivery network*) koji sprema datoteke S3 u predmemoriju radi optimizacije resursa.

```
CDN_URL="https://redacted123.cloudfront.net"
AWS_ACCESS_KEY_ID="AKI...CE5"
AWS_SECRET_ACCESS_KEY="vSL...9R4"
AWS_BUCKET_NAME=d-reader-local
AWS_BUCKET_REGION=us-east-1
```

Ispis 25: Novododane postavke u datoteci `.env`

Dodatno, novododane vrijednosti okruženja specificiraju se i u `config.ts` postavkama kao svosjtva koja su vezana za funkcionalnosti S3, tako što se pozivaju vrijednosti `process.env` i spremaju ih u `config` konstantu.

```
  s3: {
    region: process.env.AWS_BUCKET_REGION,
    bucket: process.env.AWS_BUCKET_NAME,
    cdn: process.env.CDN_URL,
    credentials: {
      accessKeyId: process.env.AWS_ACCESS_KEY_ID,
      secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY,
    },
  },
```

Ispis 26: Novododane postavke u `config.ts` datoteci

U konačnici, pojednostavljena verzija `s3.service.ts` datoteke i `s3Service` servisa izgleda na sljedeći način:

```
import { S3Client } from '@aws-sdk/client-s3';
import { Injectable } from '@nestjs/common';
import config from '../configs/config';

@Injectable()
export class s3Service {
  readonly region: string;
  readonly bucket: string;
  readonly cdn: string;
  readonly client: S3Client;

  constructor() {
    this.region = config().s3.region;
    this.bucket = config().s3.bucket;
    this.cdn = config().s3.cdn;

    this.client = new S3Client({
      region: this.region,
      credentials: config().s3.credentials,
    });
  }

  getPublicUrl = (key: string) => {
    if (this.cdn) return `${config().s3.cdn}/${key}`;
    else return `https://${this.bucket}.s3.amazonaws.com/${key}`;
  };

  // ... other functionalities
}
```

Ispis 27: Sadržaj `s3.service.ts` datoteke

Zatim na klijentskoj aplikaciji, na sličan način kao što se prethodno koristio *hook* za registraciju korisnika, koristit ćemo i *hook* za dohvaćanje podataka o pojedinoj strip epizodi i njenoj prodaji.

Stranica za kupnju stripova (engl. *mint page*) omogućuje korisnicima povezivanje svojih digitalnih novčanika preko funkcionalnosti iz `useWallet` koji se uvozi iz klijentske `@solana/wallet-adapter-react` knjižnice. Komunikacija sa blockchainom se vrši preko instance `Connection` objekta koji se uvozi preko funkcije `useConnection` iz iste knjižnice, `@solana/wallet-adapter-react`.

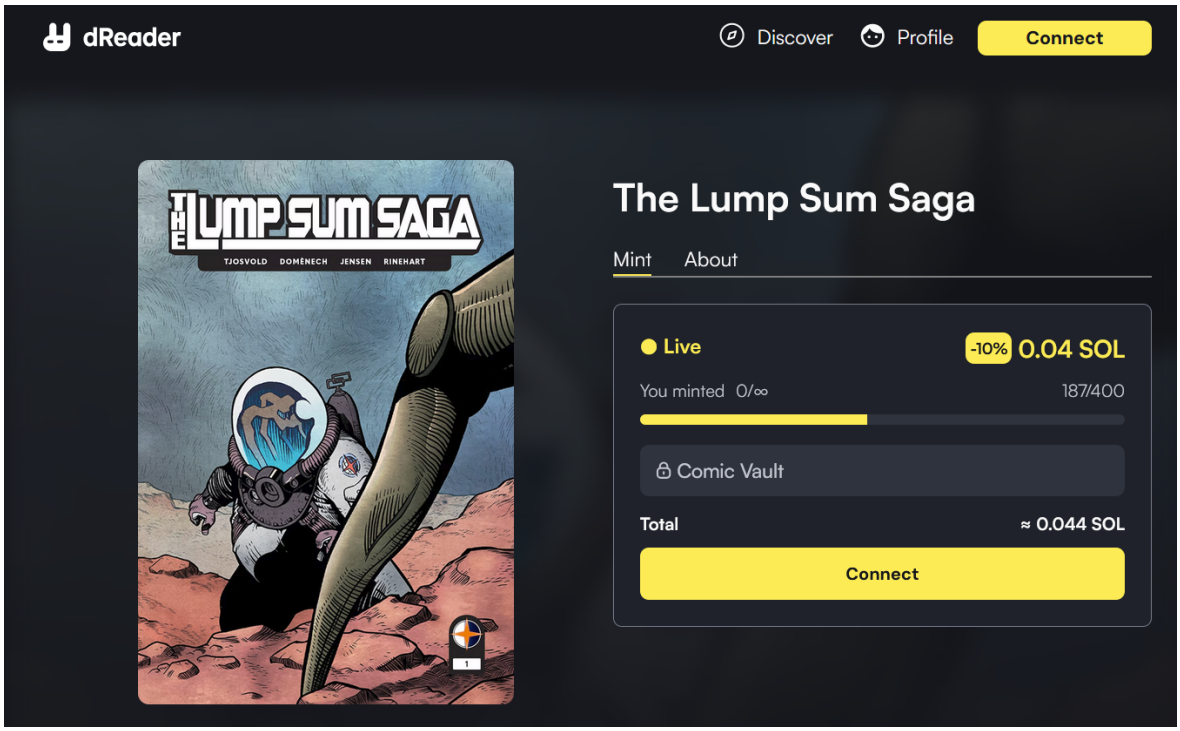
Nakon što se dohvate podaci o stripu i njegovoj prodaji, te nakon što korisnik spoji svoj novčanik, potrebno je dohvatiti kodiranu transakciju za kupnju stripa sa funkcijom `useFetchMintOneTransaction`. Transakcija se zatim dekodira, parsira, potpisuje, i šalje na blockchain preko `signTransaction` i `sendTransaction` funkcija.

```
import { useFetchPublicComicIssue } from 'api/comicIssue'
import { useConnection, useWallet } from '@solana/wallet-adapter-
   react'
import { useFetchMintTransaction } from '@/api/transaction'
import { CandyMachine } from '@/components/CandyMachine'

const MintPage = ({ params }) => {
  const { connection } = useConnection()
  const { publicKey, signTransaction } = useWallet()

  const { data: comicIssue } = useFetchPublicComicIssue(params.id)
  const { data: transaction } = useFetchMintTransaction(publicKey)
  const { data: candyMachine } = useFetchCandyMachine(comicIssue.
   candyMachineAddress)

  const handleMint = async () => {
    const signedTransaction = await signTransaction(transaction)
    await connection.sendTransaction(signedTransaction)
  }
```

```
  return (
    <main className='mint-page'>
      <p className='mint-title'>{comicIssue.title}</p>
      <div className='mint-details'>
        <CandyMachine data={candyMachine} mint={handleMint} />
      </div>
    </main>
  )
}

export default MintPage
```

Ispis 28: Pojednostavljeni sadržaj `mint/[id]/page.tsx` datoteke

Konačni izgled stranice za kupnju stripova uključuje nekoliko vizualnih komponenti koje su van dosega ovog završnog rada.



Slika 11: Stranica za prodaju stripova

4.8. Sigurnosni i tehnički izazovi

Tijekom razvoja dReader platforme naišlo se na niz sigurnosnih i tehničkih izazova, posebno s obzirom na osjetljivost podataka i potrebe za zaštitom korisničkih informacija te osiguravanjem integriteta transakcija na blockchainu.

Platforma dReader je za vrijeme pisanja ovoga rada dostupna u produkciji i broji 10,000 prijavljenih korisnika. Otvoren je i poslovni subjekt u hrvatskoj koji je podložan hrvatskim zakonima i europskim regulacijama koje štite potrošače, poput GDPR (engl. *The general data protection regulation*).

Korisnički podaci na dReader platformi uključuju osjetljive informacije poput adrese elektroničke pošte, povijesti kupovine, te povezanih Solana novčanika.

Autentikacija korisnika je kritičan dio dReader platforme, a provedena je korištenjem Passport.js knjižnice u kombinaciji s NestJS-om. Autorizacija pristupa različitim dijelovima platforme postavljena je na temelju uloga korisnika.

Platforma dReader suočavala se s tehničkim izazovima vezanim za izvođenje, osobito tijekom većih događaja kao što su masovna otvaranja prodaja ili promocije. Korištenje orm-a (Object Relational Mapper) Prisma omogućilo je optimizaciju upita prema bazi podataka, smanjujući opterećenje poslužitelja tijekom sati sa intezivnim prometom na platformi. Implementirana je predmemorija na više razina, sa svrhom optimizacije upita za često korištenim podacima (najviše slikama), što je značajno ubrzalo učitavanje stranica i smanjilo broj zahtjeva prema poslužitelju.

Korištenje AWS infrastrukture omogućilo je dinamičko skaliranje resursa, čime je platforma mogla učinkovito odgovoriti na povećane zahtjeve korisnika bez pogoršanja izvedbe.

Posebna pozornost se stavila na sigurnost pametnih ugovora (engl. *smart contract*), na način da autoritet za ažuriranje ugovora je višepotpisni novčanik (engl. *multi-sig wallet*).

5. Zaključak

U ovom završnom radu predstavljen je razvoj i implementacija dReader platforme, koja omogućuje digitalno čitanje, prikupljanje i trgovanje stripovima koristeći blockchain tehnologiju Solana. Projekt je započet analizom postojećih rješenja, definiranjem zahtjeva i odabirom odgovarajućih tehnologija, nakon čega je razvijen funkcionalni prototip koji integrira najnovije web i blockchain tehnologije.

Razvijena je jedinstvena platforma koja koristi NFT tehnologiju za tokenizaciju stripova, čime se omogućuje kolekcionarstvo i trgovina digitalnim stripovima na globalnoj razini.

dReader platforma je uspješno integrirala blockchain tehnologiju Solana kako bi osigurala transparentnost, sigurnost i nepovredivost podataka vezanih uz vlasništvo i transakcije stripova. Platforma je implementirana s naglaskom na sigurnost korisničkih podataka i transakcija, te je dizajnirana tako da može skalirati i odgovarati na promjene u broju korisnika i količini transakcija.

Tijekom razvoja platforme, rješen je niz tehničkih izazova, posebice u vezi s integracijom s blockchainom i održavanjem kvalitetne izvedbe tijekom visokog opterećenja. Pažljivo planiranje, optimizacija upita prema bazi podataka, te korištenje predmemorije i skalabilnih rješenja u oblaku, omogućili su uspješno prevladavanje ovih izazova.

Iako je dReader platforma funkcionalna, postoje dodatne mogućnosti za daljnji razvoj, kao što su proširenje na druge blockchaine, poboljšanje korisničkog sučelja, integracija s društvenim mrežama itd.

dReader platforma predstavlja inovativan korak naprijed u svijetu digitalnog izdavaštva i kolekcionarstva, pružajući korisnicima jedinstveno iskustvo čitanja i trgovanja digitalnim stripovima. Platforma je postavila temelje za daljnje istraživanje i razvoj, s ciljem unapređenja digitalne distribucije sadržaja u eri blockchain tehnologije.

Literatura

[1] Next.js, “Next.js documentation,” https://nextjs.org/docs, [Online; accessed 24-August-2024].

[2] T. Query, “Tanstack query documentation,” https://tanstack.com/query/v3/docs/framework/react/overview, [Online; posjećeno 24.8.2024].

[3] NestJS, “Nestjs documentation,” https://docs.nestjs.com/, [Online; posjećeno 24.8.2024].

[4] Nodemailer, “Nodemailer website,” https://nodemailer.com/, [Online; posjećeno 24.8.2024].

[5] AWS, “Aws s3 website,” https://aws.amazon.com/s3/, [Online; posjećeno 24.8.2024].