Razvoj web aplikacije za generiranje slika pomoću radnog okvira MERN

Kalajžić, Luka

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Prijediplomski sveučilišni studij računarstva

Završni rad

Razvoj web aplikacije za generiranje slika pomoću radnog okvira MERN

Rijeka, rujan 2023.

Luka Kalajžić

0069090342

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Prijediplomski sveučilišni studij računarstva

Završni rad

Razvoj web aplikacije za generiranje slika pomoću radnog okvira MERN

Mentor: Doc. dr. sc. Marko Gulić

Rijeka, rujan 2023.

Luka Kalajžić

006909034

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA ZAVRŠNE ISPITE

Rijeka, 12. srpnja 2023.

Zavod: **Zavod za računarstvo**
Predmet: **Razvoj web aplikacija**
Grana: **2.09.06 programsko inženjerstvo**

ZADATAK ZA ZAVRŠNI RAD

Pristupnik: **Luka Kalajžić (0069090342)**
Studij: Sveučilišni prijediplomski studij računarstva

Zadatak: **Razvoj web aplikacije za generiranje slika pomoću radnog okvira MERN / Development of a web application for image generation using the MERN framework**

Opis zadatka:

Razviti web aplikaciju za generiranje slika na temelju unesenih citata i slika od strane korisnika. Uz generiranje slika iz unesenog sadržaja, aplikacija mora imati mogućnost mijenjanja generirane slike s obzirom na veličinu slike kao i veličinu, oblik te boju slova na slici. Nadalje, aplikacija mora imati funkcionalnost praćenja broja generiranih slika svakog korisnika uz limitiranje korištenja za korisnike bez pretplate. Također, treba implementirati cjelovitu autentifikaciju unutar spomenute web aplikacije. Aplikacija treba imati odvojeni administracijski dio i korisnički dio. Aplikaciju treba izraditi upotrebom MERN Stack razvojnog okvira koji se sastoji od 4 tehnologije (MongoDB, Express, React, Node.js). Za vizualni dizajn web aplikacije koristiti neki od React UI radnih okvira.

Rad mora biti napisan prema Uputama za pisanje diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Zadatak uručen pristupniku: 17. srpnja 2023.

Mentor:

Doc. dr. sc. Marko Gulić

Predsjednik povjerenstva za završni ispit:

Prof. dr. sc. Miroslav Joler

Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradio ovaj rad.

Rijeka, rujan 2023.

_____

Luka Kalajžić

Zahvala

So long & thanks for all the trading cards.

Sadržaj

Popis isječaka koda

Popis slika

1. Uvod

1.1 Opis aplikacije

U današnjem digitalnom dobu tehnološka pismenost je izuzetno važna. Važnost tehnološke pismenosti očituje se u znatno većoj produktivnosti i širim opcijama tehnološki pismenih. Jedan od primjera tehnološke pismenosti jest aplikacija koja je temelj ovog završnog rada.

Quotify je web aplikacija za generiranje slika na temelju citata i slika koje unese korisnik. Osim generiranja slika na temelju unesenog sadržaja, aplikacija ima mogućnost modificiranja generirane slike u vidu fonta, boje teksta, veličine teksta, i boje pozadinskog pravokutnika. Aplikacija omogućuje kreiranje stotine slika u minuti. U slučaju tehnološke nepismenosti ista radnja bi zahtijevala više sati ručnim radom u uređivačima slika poput *Photoshopa* ili *Canve.* Sama aplikacija spoj je suvremenih tehnologija sa klijentske strane i poslužiteljske strane. Korištene tehnologije uključuju MERN arhitekturu, Next.js, Clerk i Paddle. MERN arhitektura sastoji se od četiri tehnologije: MongoDB, Express.js, React.js, i Node.js. Sve tehnologije dio su JavaScript ekosustava. MongoDB popularan je sustav za upravljanje bazama podataka, odnosno koristi se kao nerelacijska baza podataka. Express.js je Node.js okvir koji olakšava programiranje poslužiteljske strane aplikacije koristeći programski jezik JavaScript. Velika prednost Expressa jest pisanje koda koristeći isti programski jezik za klijentsku i poslužiteljsku stranu. React.js je najpopularnija knjižnica za JavaScript i koristi se za izradu brzih i interaktivnih korisničkih sučelja. Koristi JSX sintaksu, odnosno spoj JavaScripta i HTML-a. Node.js jest višeplatformsko izvršno okruženje (eng. *runtime environment*) za JavaScript. Također omogućava preuzimanje, brisanje, i upravljanje ovisnostima vlastitim alatom imena *Node Package Manager* (NPM). Next.js je najpopularniji React okvir (eng. *framework*) i prvi okvir kojeg React preporuča u svojoj dokumentaciji. Next.js uvelike olakšava programiranje s novim *app* direktorijem i usmjeravanjem baziranim na datotekama (eng. *file-based routing*). Clerk je relativno nova autentifikacijska platforma koja pojednostavljuje autentifikaciju i upravljanje korisnicima u web aplikacijama. Paddle je platforma za e-trgovinu (eng. *e-commerce*) koja pruža korisnicima sve potrebno za prodaju digitalnih proizvoda i usluga putem interneta.

2. Korištene tehnologije

Za izradu kompleksnijih web aplikacija ključno je znanje *frontend* i *backend* strana razvijanja aplikacija. *Frontend*, odnosno klijentska strana odnosi se na sve što se događa ili prikazuje na strani korisnika, to jest na krajnjem korisničkom uređaju. *Backend*, odnosno poslužiteljska strana odnosi se na sve što se događa što se odvija na poslužiteljskog, odnosno strani *servera*. Kombinacija te dvije strane omogućuje modularne i sigurne aplikacije. Popularni primjeri *full-stack* arhitektura uključuju:

• MERN(MongoDB, ExpressJS, ReactJS, NodeJS)

•LAMP(Linux, Apache, MySQL, PHP)

• MEAN (MongoDB, ExpressJS, AngularJS, NodeJS)

Full-stack razvoj izuzetno je koristan tvrtkama koje ušteđuju na troškovima zapošljavanja više programera ili timova programera jer pri *full-stack* razvoju potreban je samo jedan tim. Ušteda na troškovima i vremenu razvijanja omogućava ispunjavanje više projekata u nekom vremenskom razdoblju, što rezultira u većoj produktivnosti i time profitabilnosti. *Full-stack* razvoj također je koristan programerima koji mogu izgraditi aplikacije jednoručno, učinkovitije, i samostalnije. U primjeru MERN arhitekture potrebno je znati jedan programski jezik, JavaScript. Time je lakša implementacija, održavanje, nadogradnja i skaliranje. Valja spomenuti da je rješavanja problema i pogrešaka u kodu također brže zbog veće upoznatosti s kodom. Na kraju je korisno i korisnicima koji dobivaju pristup aplikacijama brže uz manji trošak. S korisničke stane brža je i implementacija zatraženih značajki i ispravljanja pogrešaka u radu aplikacija.

2.1 Frontend

Frontend, odnosno klijentska strana jest vidljivi dio web aplikacije. Korisnici vide i interaktiraju s klijentskom stranom. *Frontend* omogućava programerima da naprave sučelja kojima se korisnici mogu intuitivno, brzo, i bezbolno koristiti. Lako korištenje sučelja i dobro dizajnirano i implementirano sučelje određuju hoće li korisnik koristiti aplikaciju ili izaći iz nje. *Frontend* uključuje tehnologije HTML, CSS i JavaScript, od kojih zadnja tehnologija od kojih omogućava korisnicima da koriste web aplikaciju na dinamičan način poput uređivanja videa ili slika.

2.2. Backend

Backend, odnosno poslužiteljska strana je kritična strana koja izvršava funkcije „iza scene“. Poslužiteljska strana omogućava programiranje kompleksnih funkcije u aplikacijama. To uključuje pohranu, obradu i upravljanje podacima te osigurava sigurnost aplikacije. Poslužiteljska strana omogućava bolju performansu aplikacije u slučaju da se logika aplikacije odvija na strani poslužitelja čime se ne opterećuje klijentska strana. Drugim riječima, uređaj krajnjeg korisnika nije opterećen što omogućava bolju performansu s klijentske strane. Naravno, bitna je ravnoteža između opterećenja korisnika i poslužitelja i bitno je biti svjestan mogućnosti klijentske i poslužiteljske strane pri donošenju odluka. Postoji više popularnih opcija pri biranju jezika i tehnologije na poslužiteljskoj strani, ali česte opcije uključuju JavaScript i Python.

2.3. MERN

MERN okvir je akronim za četiri tehnologije koje čine jednu od najpopularnijih opcija za izgradnju *full-stack* web aplikacija. Te tehnologije su MongoDB [1], Express.js [2], React [3] i Node.js [4]. Svaka od tehnologija ima važnu ulogu u izgradnji *full-stack* aplikacija koje su brze i pružaju izvrsno korisničko iskustvo. Kombinacija tih tehnologija rezultira u lakšoj i brzoj implementaciji ideje u gotove aplikacije koje su isporučene korisnicima. Velika prednost MERN okvira jest što je korištenje besplatno. Uz to, MERN okvir je vrlo raširen što rezultira u bogatstvu resursa koji su korisni programerima u svrhe edukacije, ispravljanja pogrešaka u kodu, implementacije novih značajki i slično. MERN je po izboru tehnologija efikasan, odnosno ne postoji nepotrebna tehnologija što rezultira u manjim veličinama aplikacija koje se brže izvršavaju. MERN se posebice ističe u izgradnji dinamičkih web sučelja radi korištenja React.js knjižnice. Najvažnija prednost pri korištenju MERN okvira jest korištenje samo jednog programskog jezika za klijentsku i poslužiteljsku stranu – JavaScripta. Neki od primjera za koji je prikladno korištenje MERN okvira jesu kalendari, interaktivne aplikacije, društvene mreže, i slično.

2.3.1 MongoDB

MongoDB je NoSQL sustav za upravljanje bazom podataka koji se inače koristi u razvoju modernih web aplikacija. Razlika između SQL i NoSQL sustava jest u načinu modeliranja i pohrane podataka te korištenom jeziku za pristupu podacima. SQL sustavi su općeniti fiksniji zbog

korištenja fiksnih *shema*, odnosno tablica s redovima i stupcima. NoSQL sustavi podatke pohranjuju u različitim, ali fleksibilnijim strukturama. MongoDB omogućuje fleksibilnu i skalabilnu pohranu podataka i ima mogućnost brze prilagodbe promjenjivim zahtjevima aplikacija, što je bitno kod npr. softverskih tvrtki. MongoDB temelji se na konceptu baze podataka orijentiranoj na dokumente. To znači da su podaci pohranjeni u dokumentima u formatu podataka sličnom JSON-u poznatom kao BSON (Binarni JSON, tj. *binary encoded Javascript Object Notation*). Svaki dokument može sadržavati različita polja, a njegova se struktura može mijenjati kroz vrijeme bez promjene sheme baze podataka. Ova fleksibilnost čini MongoDB prikladno za aplikacije čiji se zahtjevi mijenjaju tijekom vremena. MongoDB nudi širok jezik upita (eng. *query language*) koji olakšava pretraživanje i analizu podataka. Moguće je pisati složene upite, indeksirati podatke i koristiti okvire agregacije za analizu podataka. Jedna karakteristika koja izdvaja MongoDB od ostalih baza podataka jest sposobnost skaliranja. Baza podataka podržava horizontalno skaliranje kroz replikaciju i fragmentaciju (eng. *sharding*), što omogućuje visoku dostupnost i distribuciju podataka na više poslužitelja za poboljšanje performansi te toleranciju na pogreške. MongoDB se obično koristi za pohranu podataka kao što su korisnički profili i proizvodi. Ima veliku zajednicu programera koji razvijaju razne dodatke i knjižnice koje pružaju podršku i olakšavaju rad. Sve ovo čini MongoDB odličnim alatom za pohranu podataka u razvoju modernih *full-stack* web aplikacija.

2.3.2. Express.js

Express.js je popularan okvir za Node.js koji se često koristi za razvoj skalabilnih i brzih web aplikacija. Jedna od glavnih karakteristika Express.jsa je njegova jednostavnost i fleksibilnost. To znači da programeri mogu slobodno graditi aplikacije prema svojim specifičnim potrebama i zahtjevima bez nametanja previše unaprijed definiranih pravila i struktura poput nekih drugih okvira za Node.js. Middleware je ključna značajka Express.js okvira. Middleware programerima omogućuje dodavanje funkcija koje se izvršavaju tijekom zahtjeva za stvaranje slojevitih aplikacija. Valjda spomenuti da Express.js olakšava definiranje ruta za različite HTTP metode kao što su POST, PUT, GET i DELETE. Jedan od ključnih dijelova Express-a je podrška za autentifikaciju, koja je neophodna za sigurnost web aplikacija. Programeri mogu implementirati ove značajke koristeći različite knjižnice i dodatke, ovisno o potrebama svog projekta. Konačno, Express.js ima aktivnu zajednicu koja redovito ažurira i proširuje tehnologiju. Primjer postavljanja Express.js aplikacije može se vidjeti na slici 2.1.

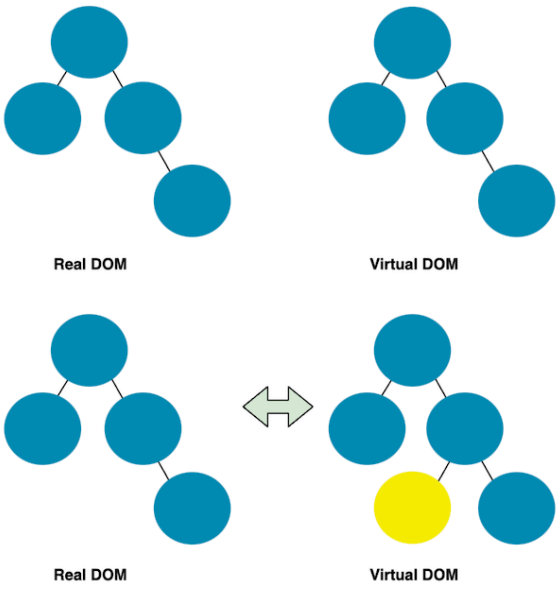
Slika 2.1 – Postavljanje Express.js aplikacije

2.3.3. React

React.js je izuzetno popularna JavaScript knjižnica za kreiranje korisničkih sučelja koju je razvio Facebook (sada Meta) u 2013. godini. Ovaj okvir je iznimno važan i čest alat u razvoju modernih web i mobilnih aplikacija. Neke od najvažnijih značajka Reacta su:

1. Komponente. React se temelji na komponentnom pristupu. To znači da je korisničko sučelje izgrađeno pomoću malih samostalnih komponenti. To olakšava ponovnu upotrebu koda i održavanje aplikacije.

2. Virtualni DOM. React koristi virtualni DOM (*Document Object Model*) za učinkovito ažuriranje korisničkog sučelja. Naime, umjesto da svaki put izravno mijenja stvarni DOM, React stvara virtualni prikaz i uspoređuje ga s postojećim DOM-om kako bi pronašao samo potrebne promjene. Samo izmijenjeno sučelje se ažurira. To poboljšava performanse aplikacije, pogotovo kod većih aplikacija. Na primjeru slike 2.2 vidljiva je razlika između pravog DOM-a i virtualnog. Za jednako vrijeme virtualni DOM stići će napraviti promjenu, u ovom slučaju dodavanje komponente na stablo, dok pravi DOM neće. Virtualni DOM je brži.



Slika 2.2 – Pravi DOM i virtualni DOM

3. JSX. JSX je programski jezik koji React koristi za pisanje svog korisničkog sučelja. JSX je spoj JavaScripta i HTML-a zato da bi kod bio lakši za čitanje i pisanje. JSX omogućuje stvaranje elemenata i komponenti kao da se piše HTML, ali uz mogućnost uključivanja dinamičkih podataka i funkcionalnosti korištenjem JavaScripta. Važno je spomenuti da JSX i HTML nisu isti. Naime, jedna očita razlika je u tome što su neke rijeli rezervirane za JavaScript. Jedan takav primjer je *for* u HTML-u, koji u JSX-u je preimenovan u *HTMLFor*. Razlog tome jest dakako JavaScriptova *for* petlja.

4. Prilagodljivost. React je prilagodljiv i može se koristiti s brojnim alatima i knjižnicama. Nadalje, može se koristiti i koristi se za razvoj malih, jednostavnih aplikacija od školskih projekata kao i za izgradnju velikih, složenih sustava poput bankovnih sustava.

5. Stanje (eng. *state*). React omogućava komponentama da imaju svoje unutarnje stanje koje se može dinamički mijenjati. Kada se stanje promijeni, React automatski ažurira korisničko sučelje kako bi odražavalo promjenu. To omogućuje stvaranje reaktivnih aplikacija. Jedan primjer stanja jest tema stranice – svijetla ili tamna. Sačuvano je stanje (na primjer 0 za tamno i 1 za svijetlo). Stanje se mijenja *toggle* gumbom. Stanje aplikacije bi se u tom primjeru mijenjalo kukom (eng. *hook) useState*. Neke od drugih popularnih React kuka su *useEffect, useMemo, useContext,* i *useCallback.*

6. Jednostranične aplikacije (eng. *Single Page Applications*, skraćeno SPA). React se često koristi za izradu jednostraničnih aplikacija. U ovoj se aplikaciji učitava samo HTML te

stranice. Sve ostale promjene prikazuju se dinamički bez ponovnog učitavanja stranice. To omogućuje brže, efikasnije i ugodnije korisničko iskustvo.

7. Velika zajednica i ekosustav. React ima iznimno veliku zajednicu programera koji rade i održavaju mnogo dostupnih knjižnica i alata. Valja spomenuti i React Native, koji omogućuje pisanje koda za web aplikacije i za mobilne uređaje u JavaScriptu i Reactu.

2.3.4 Node.js

Node.js izvršno je okruženje (eng. *runtime environment*) za JavaScript koje omogućava izvršavanje JavaScript koda na poslužiteljskoj strani. Izvršavanje JS koda na poslužiteljskoj stranici uvelike je korisno sa strane programera jer omogućava programerima da koriste jedan programski jezik za sve potrebe aplikacije, umjesto da se, na primjer, koristi JavaScript sa strane klijenta i Python s poslužiteljske strane. Drugim riječima, postoji znatna ušteda u resursima obrazovanja i ljudi ako je cijela aplikacija u istom ekosustavu.

Node.js okruženje temelji se na V8 JavaScript *engineu* [5]. To ga čini efikasnim i brzim pri obradi zahtjeva s na primjer klijentske strane. Budući da je Node.js iznimno popularan odabir okruženja postoje brojne knjižnice koje olakšavaju programiranje poslovne logike. Bitno je spomenuti *Node Package Manager* (NPM) koji se koristi za upravljanje ovisnostima i paketima u Node.js ekosustavu. Moguće je koristiti i ostale alate za upravljanje ovisnostima i paketima u Node.js ekosustavu poput *yarna*.

2.4 Next.js

Next.js je najpopularniji React okvir (eng. *framework*) [6] i prvi okvir kojeg React preporuča u svojoj dokumentaciji. Next.js uvelike olakšava programiranje *server-side* generiranih komponenata (eng. *server-side rendering*, skraćeno SSR), s novim *app* direktorijem i usmjeravanjem baziranim na datotekama (eng. *file-based routing*). Osim toga, Next.js znatno ubrzava proces razvoja modernih web aplikacija zbog dodatnih mogućnosti koje dodaje na bazu React.js-a. Zbog SSR mogućnosti, poboljšava se SEO (eng. *Search Engine Optimization*), performansa aplikacije i brzina prikaza sadržaja. To je pogotovo važno kod korisnika sa slabijim

uređajima i lošom brzinom interneta koji su načelno u nerazvijenim zemljama. Odlična značajka Next.js-a jest *Hot Module Reloading (*skraćeno HMR*)*, što omogućava da se promjene u kodu odmah naprave u pregledniku bez da se cijela stranica ponovno učita tako što se ažurira samo promijenjeni dio, tj. *module*. *Static Site Generation* (skraćeno SSG*)* je također olakšan Next.js-om. SSG omogućava izradu statičkih stranica u vremenu izgradnje (eng. *build time*) koje su kasnije poslužene korisnicima od strane *Content Delivery Networka* (skraćeno CDN). To je izuzetno korisno kod statičnih stranica poput blogova i stranica s člancima. Takve stranice se brže učitavaju jer su poslužene preko CDN-a i time bliže krajnjem korisniku.

2.5 Clerk

Clerk [7] je relativno nova autentifikacijska platforma koja pojednostavljuje autentifikaciju i upravljanje korisnicima u web aplikacijama. Najčešće se koristi uz Next.js, iako se može koristiti uz ostale React UI knjižnice. Clerk pruža alate i komponente koji znatno olakšavaju i ubrzavaju implementaciju autentifikacije i poboljšavaju sigurnost. Ključne značajke i mogućnosti platforme Clerk uključuju:

1. Jednostavna integracija. Clerk se može jednostavno integrirati u postojeće web aplikacije bez složenih prilagodbi. Programeri mogu brzo dodati autentifikaciju korisnika svojim projektima. Za jednostavnu postavu potrebno je svega pet minuta, što čini Clerk odličnim izborom za projekte koji su kratki na vremenu.

2. Fleksibilna prilagodba. programeri mogu prilagoditi autentifikaciju kako bi odgovarala njihovim specifičnim potrebama i dizajnu aplikacije. Clerk nudi prilagodljive komponente za dizajniranje obrazaca za registraciju i stranica za registraciju. Primjeri komponenata su *<SignedIn />, <SignedOut />.* Ovisno o tome je li korisnik autentificiran ili ne, prikazuju se komponente koje su stavljene u jednu od te dvije komponente.

3. Sigurnost. Clerk upravlja sigurnošću autentifikacije, uključujući zaštitu od *Cross-Site Request Forgery* (skraćeno CSRF) napada, *Cross-site scripting* (skraćeno XSS) napada i ostalih sigurnosnih prijetnji. Bitno je spomenuti i *Multi-factor authentication* (skraćeno MFA) opciju koja pruža dodatnu razinu zaštite za korisničke račune.

4. Brojne mogućnosti prijave. Osim e-maila, Clerk podržava različite metode prijave kao što su Google Auth, Facebook, Twitter, Discord, i još mnogo opcija. Clerk nudi i *Multi-factor authentication* (skraćeno MFA) za dodatnu sigurnost.

5. Upravljanje korisnicima. Clerk pruža sučelje za jednostavno upravljanje korisnicima koje je lako dostupno i uključuje mogućnost provjere i promjene njihovih osobnih podataka, lozinke i postavki.

6. Zaboravljena lozinka. Platforma programerima olakšava implementaciju funkcije poništavanja zaboravljene lozinke, čime se poboljšava korisničko iskustvo. Clerk i tu dužnost uzima iz ruka programera što im daje više vremena za npr. rad na poslovnoj logici.

7. Praćenje aktivnosti. Clerk omogućuje praćenje aktivnosti korisnika kao što su prijave, registracije i promjene lozinki kako bi pomogao u praćenju sigurnosti. Praćenje aktivnosti je moguće u Clerk *admin* sučelju na njihovoj web stranici.

Sve navedene karakteristike čine Clerk odličnim rješenjem za implementaciju autentifikacije u web aplikacijama. Programeri danas često koriste Clerk za ubrzavanje razvoja aplikacija i poboljšanje sigurnosti autentifikacije korisnika. Jedina mana jest što je Clerk plaćeni alat te ga je moguće koristiti besplatno za samo do 5000 korisnika.

2.5.1 Instalacija Clerka

Implementacija Clerka u Next.js-u izuzetno je brza i nekomplicirana [8]. Na početku potrebna je Next.js aplikacija koju je moguće inicijalizirati pokretanjem naredbe u terminalu (Isječak koda 2.1).

```
npx create-next-app@latest
```

Isječak koda 2.1 – inicijalizacija Next.js aplikacije

Nakon odabira ponuđenih postavki tijekom instalacije, potrebno je pokrenuti sljedeću naredbu za inicijalizaciju Clerka (Isječak koda 2.2).

```
npm install @clerk/nextjs
```

Isječak koda 2.2 – instalacija Clerk-a

2.5.2 Postavljanje *.env.local* datoteke

Slijedi postavljanje *.env.local* datoteke u korijenskom direktoriju što se može vidjeti na isječku koda 2.3. *.env* varijable dostupne su u Clerk upravljačkom sučelju koji se nalazi na adresi https://dashboard.clerk.com/. Te varijable su važne jer omogućuju postavljanje osnovnih parametara i tajna u tekstualnoj datoteci i njihovo korištenje u aplikaciji. Time je odvojena konfiguracija varijabli od koda i štite se tajne lozinke i ključevi od nepoželjnog pristupa.

```
NEXT_PUBLIC_CLERK_PUBLISHABLE_KEY=pk_test_d2l0dHktd29vZGNvY2stMzMuY2xlcmsuYWN
jb3VudHMuZGV2JA
CLERK_SECRET_KEY=sk_test_9gbTHRKqOk9ANtwg0ZyuyzBVZ2crRsWqkBqdRjkLQW
```

Isječak koda 2.3 - primjer *.env* varijabli:

2.5.3 Ažuriranje korijena aplikacije

Potrebno je uključiti korijensku *layout* datoteku tako da uključuje Clerk pomoću komponenta *<ClerkProvider />*. Time se daje Clerku pristup kontekstu cijele aplikacije što omogućava funkcionalnosti poput sesija (Isječak koda 2.4).

```
import { ClerkProvider } from '@clerk/nextjs'

export const metadata = {
  title: 'Next.js 13 with Clerk',
}

export default function RootLayout({
  children,
}: {
  children: React.ReactNode
}) {
  return (
```

```
    <ClerkProvider>
      <html lang="en">
        <body>{children}</body>
      </html>
    </ClerkProvider>
  )
```

Isječak koda 2.4 – primjer */app/layout.js* datoteke:

2.5.4 Zaštita aplikacije uz Middleware

Nakon što je instaliran Clerku korijenu aplikacije, potrebno je uspostaviti *middleware* datoteku koji određuje u kojem se određuje koje stranice su javne a koje privatne, odnosno dostupne samo autentificiranim korisnicima (Isječak koda 2.5).

```
import { authMiddleware } from "@clerk/nextjs";

// This example protects all routes including api/trpc routes
// Please edit this to allow other routes to be public as needed.
// See https://clerk.com/docs/references/nextjs/auth-middleware for more
information about configuring your middleware
export default authMiddleware({});

export const config = {
  matcher: ["/((?!.*\\..*|_next).*)", "/", "/(api|trpc)(.*)"],
};
```

Isječak koda 2.5 – primjer middleware.ts datoteke

2.5.5 Izgradnja stranica za *sign up* i *sign in*

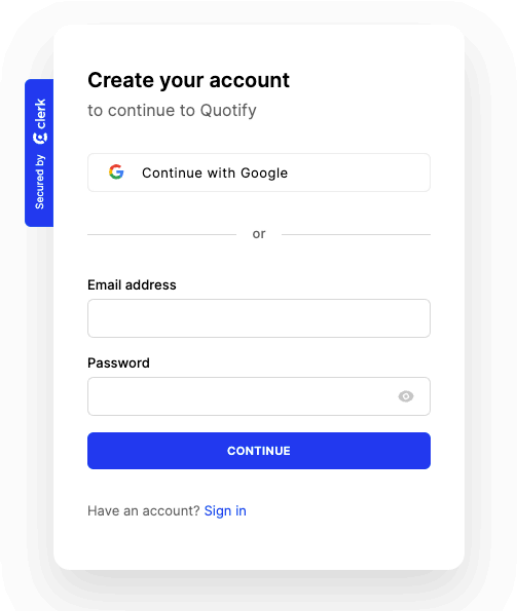
Clerk nudi vlastite komponente koje mogu biti iskorištene za *sign in* i *sign up.* Ovo uvelike olakšava i ubrzava postupak postavljanja autentifikacije (Isječak koda 2.6).

```
import { SignUp } from "@clerk/nextjs";

export default function Page() {
  return <SignUp />;
}
```

Isječak koda 2.6 – primjer *Sign up* stranice

Na slici 2.2 moguće je vidjeti izgled Clerkove komponentu za *Sign up*. Bitno je spomenuti da izgled ovisi o odabranim metodama autentifikacije. Ako programer doda još opcija, te će opcije biti prikazane na istom mjestu gdje je Google Auth opcija u ovoj slici.



Slika 2.3 – Clerk *Sign Up* komponenta

2.5.6 Ažuriranje *.env* varijabli

Nakon izrade ruta za *sign in* i *sign up*, potrebno je specificirati te rute u *.env* datoteci. Potrebno je i odrediti rutu na koju će korisnik biti preusmjeren nakon uspješne autentifikacije (Isječak koda 2.7).

```
-----------------------------------------------------------------------------------------------------------
NEXT_PUBLIC_CLERK_SIGN_IN_URL=/sign-in
NEXT_PUBLIC_CLERK_SIGN_UP_URL=/sign-up
NEXT_PUBLIC_CLERK_AFTER_SIGN_IN_URL=/
NEXT_PUBLIC_CLERK_AFTER_SIGN_UP_URL=/
-----------------------------------------------------------------------------------------------------------
```

Isječak koda 2.7. – primjer *.env* varijabli

2.5.7 Dodavanje *<UserButton />* komponente

Dodavanje ove komponente omogućava korisnicima lako upravljanje informacijama svog računa te odjavu iz računa. Ova komponenta česti je dodatak navigacijskoj traci. Važno je

specificirati `afterSignOutUrl`, koji određuje gdje će korisnik biti usmjeren nakon odjave (Isječak koda 2.8).

```
import { UserButton } from "@clerk/nextjs";

export default function Home() {
  return (
    <div>
      <UserButton afterSignOutUrl="/"/>
    </div>
  )
}
```

Isječak koda 2.8 – primjer korištenja *<UserButton />* komponente

2.6 Paddle

Paddle [9] je platforma za e-trgovinu (eng. *e-commerce*) koja pruža korisnicima sve potrebno za prodaju digitalnih proizvoda i usluga putem interneta. Neke od tih funkcija su funkcionalnosti e-trgovine, naplatnih vrata (eng. *payment gateway*), analitike i upravljanja korisnicima. Sve u svemu Paddle čini proces prodaje digitalnih proizvoda vrlo jednostavnim.

Paddle je jedna od popularnih alternativa Stripeu koja ima veći fokus na digitalne proizvode. Glavna razlika je što Paddle ne procesira samo plaćanja, već je i odgovoran za procesiranje poreza (eng. *Merchant of record*). Neki od poznatijih tvrtaka koje koriste Paddle su Tailwind Labs, Geoguessr, i Laravel.

Nekoliko ključnih karakteristika i značajki Paddle platforme:

1. Omogućavanje brojnih načina plaćanja. Paddle prihvaća kreditne kartice, PayPal i lokalne metoda plaćanja. To omogućava lako plaćanje krajnjim korisnicima.

2. Opcije plaćanja. Paddle nudi jednokratna plaćanja te plaćanja pretplatama, što je savršeno za softverske tvrtke koje često imaju opcije plaćanja pretplatama.

3. Brza i jednostavna integracija. Programiranje Paddle integracija, od jednostavnog plaćanja do kompleksnijih značajki je općenito bezbolan i kratkotrajan proces.

4. Sigurnost. Paddleova dužnost je brinuti se za sigurnost i zaštitu od prevara korisnika.

5. Globalnost. Paddle omogućuje prodaju proizvoda u brojnim državama i valutama. To omogućuje rješavanje kompliciranih poreznih problema i oslobađa programere od administrativnih dužnosti, omogućavajući da vrijeme usmjere na važnije zadaće.

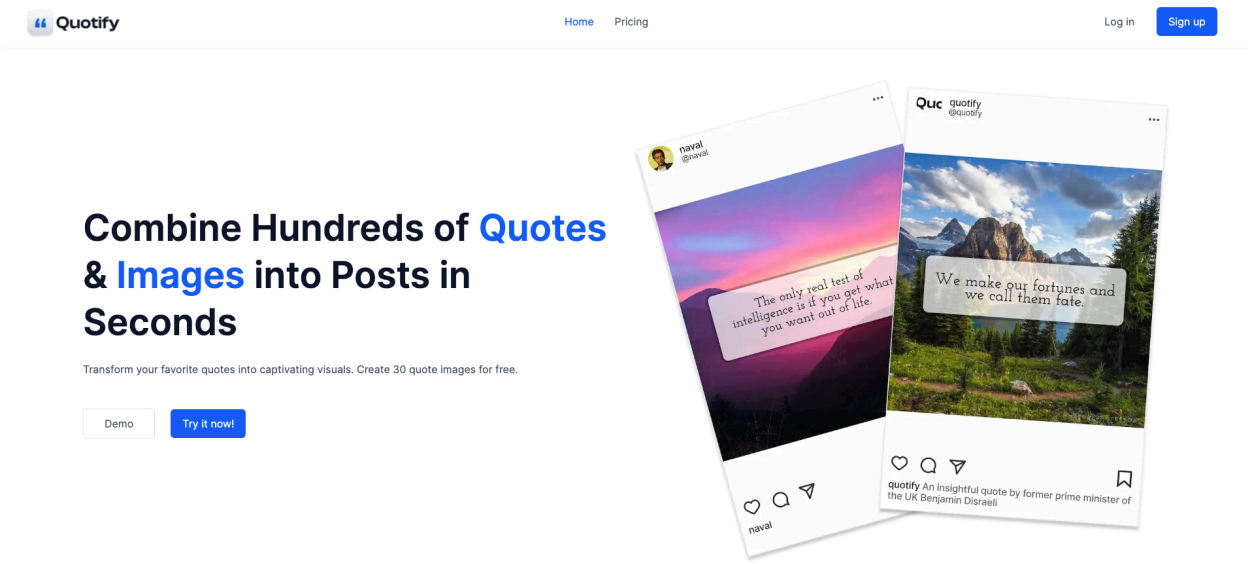
6. Brojne mogućnosti. Osim jednostavnog inicijalnog postavljanja, Paddle nudi mnogo mogućnosti što osigurava da programer nije ograničen alatom.

3. Opis aplikacije

3.1 Početna stranica

Quotify je web aplikacija za generiranje slika na temelju citata i slika koje unese korisnik. Osim generiranja slika od unesenog sadržaja, aplikacija ima mogućnost modificiranja generirane slike u vidu fonta, boje fonta i pravokutnika koji se nalazi iza teksta u boji. Nadalje, aplikacija ima funkciju praćenja broja generiranih slika za svakog korisnika s ograničenjima korištenja za nepretplaćene korisnike. U navedenoj web aplikaciji implementirana je potpuna autentifikacija. Aplikacija ima zaseban administrativni i korisnički dio.

Ulaskom na početnu stranicu korisniku se prikazuje opis aplikacije i primjeri generiranih slika (Slika 3.1).



Slika 3.1 – Početna stranica

Nakon dolaska na stranicu, korisnik ima mogućnosti prijave i kreiranja korisničkog računa pritiskom na sljedeće opcije (Slika 3.2).

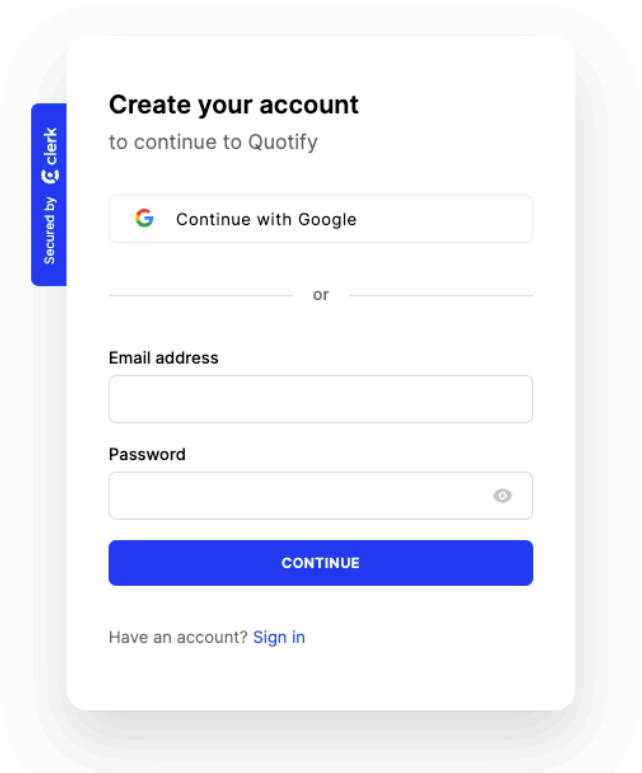
Slika 3.2 – Gumbovi za prijavu u navigacijskoj traci

Na sljedećoj slici vidljiv je gumb za prijavu u *hero* sekciji (Slika 3.3).

Slika 3.3 – Gumb za prijavu u *hero* sekciji

3.2 Prijava

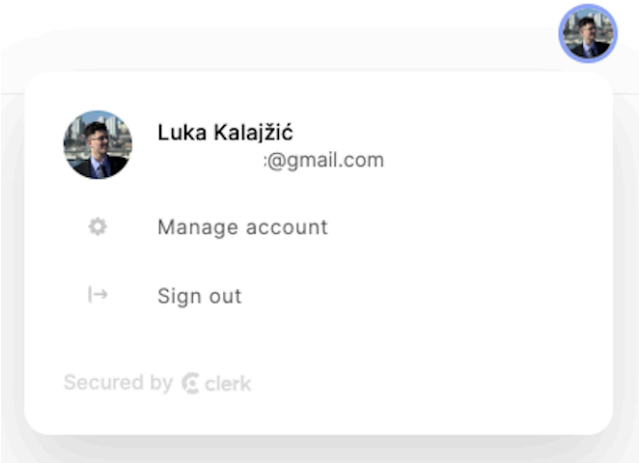
Ukoliko korisnik pritisne neki od gumbova za prijavu, korisnik će biti preusmjeren na stranicu za prijavu. Na toj stranici korisnik ima mogućnosti prijave i izrade računa pomoću email adrese ili Google Autha (Slika 3.4). Nakon prijave, korisnik je preusmjeren na stranicu za podnošenje slika i citata.



Slika 3.4 – Stranica za prijavu

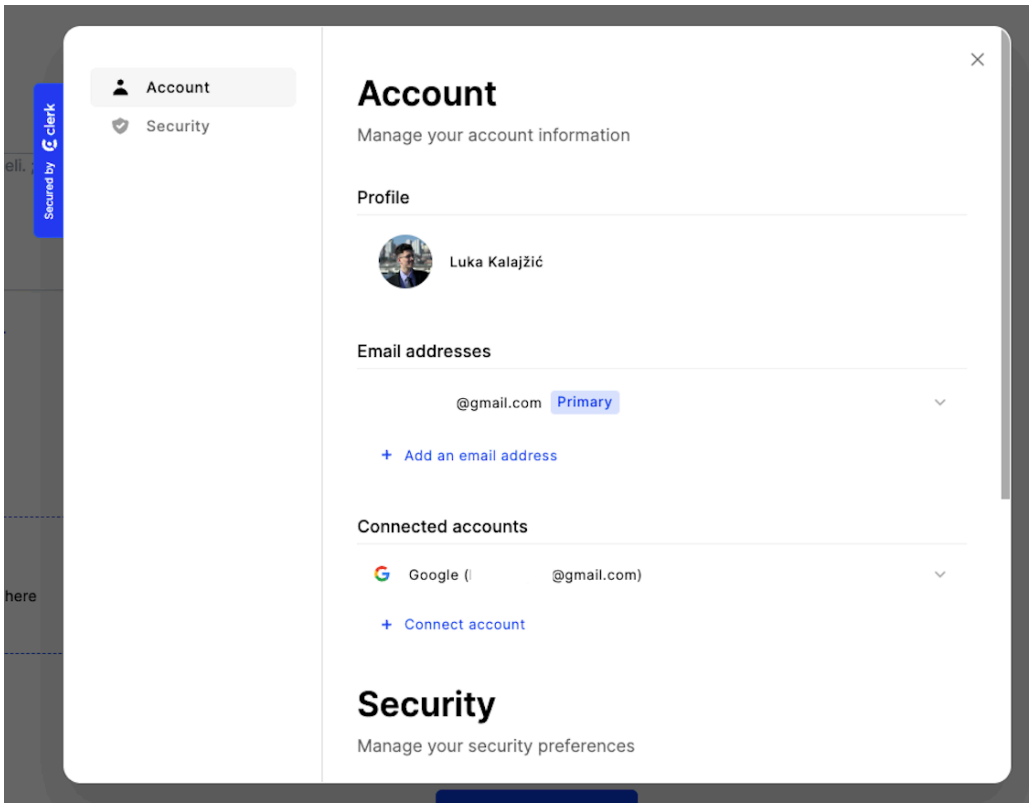
3.3 Uređivanje profila

Nakon što je korisnik prijavljen, korisnik može u desnom gornjem kutu pristupiti svome profilu, ažurirati ga te se odjaviti (Slika 3.5).



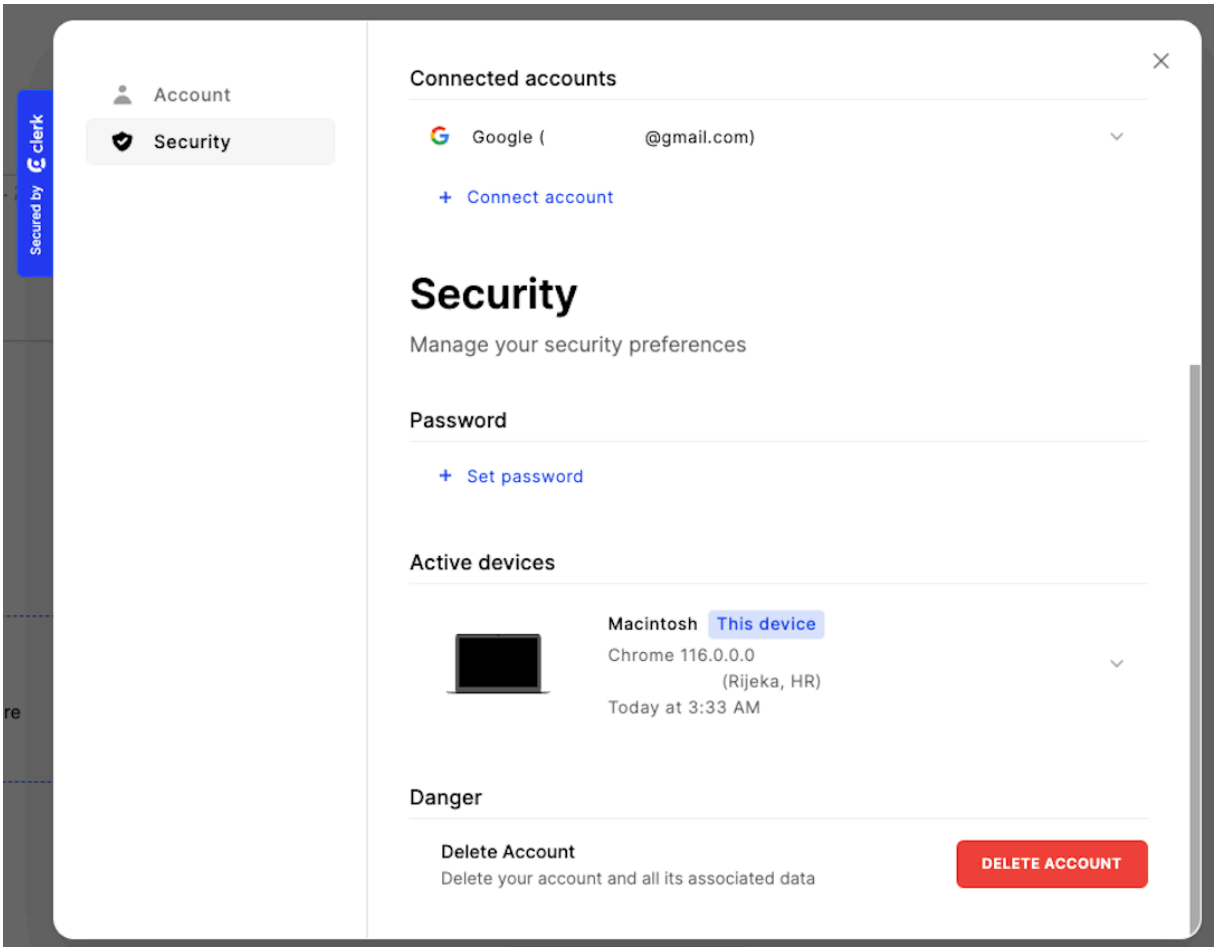
Slika 3.5 – Korisnikov profil

Pritiskom na *Manage account*, korisnik može vidjeti informacije o svom računu (Slika 3.6). Korisnik može dodati i brisati email adrese te spojiti i odspojiti račune s integracijama treće strane.



Slika 3.6 – Upravljanje računom

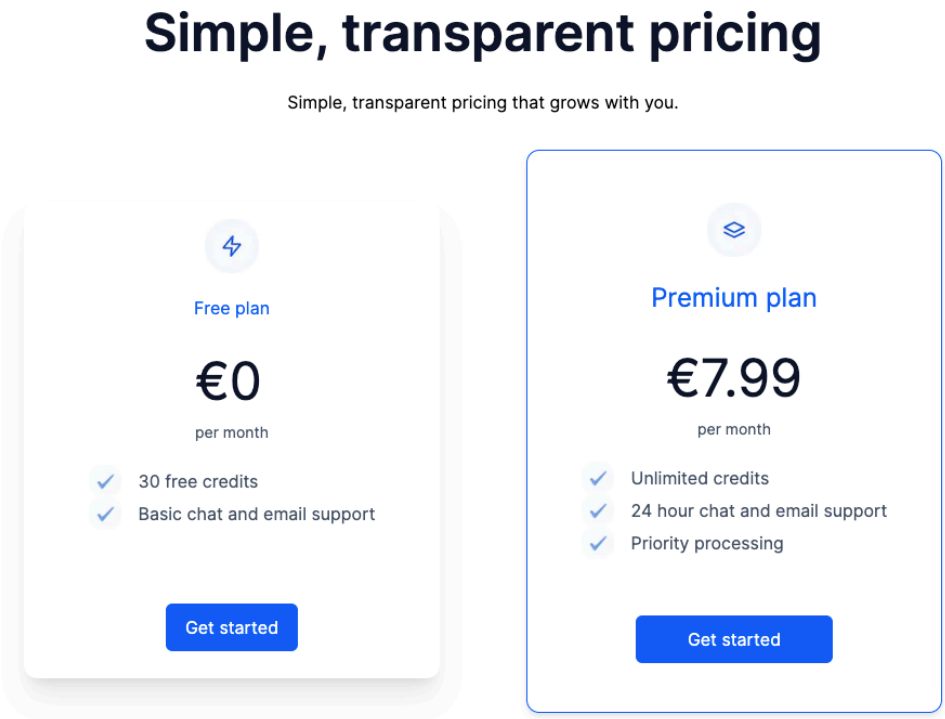
Ispod informacija o računu korisnik može promijeniti svoju lozinku. U slučaju da je korisnik autentificiran metodom poput Google Autha, korisnik nema opciju promijeniti zaporku, već je prvo mora zadati. Korisnik također može vidjeti podatke o aktivnim uređajima, što uključuje operativni sustav, preglednik, IP adresu, lokaciju i trenutno vrijeme. Na samom dnu korisnik ima opciju brisanja vlastitog računa i svih podataka vezanih uz račun poput povijesti korištenja aplikacije (Slika 3.7).



Slika 3.7 – Sigurnosne postavke

3.4 *Pricing*

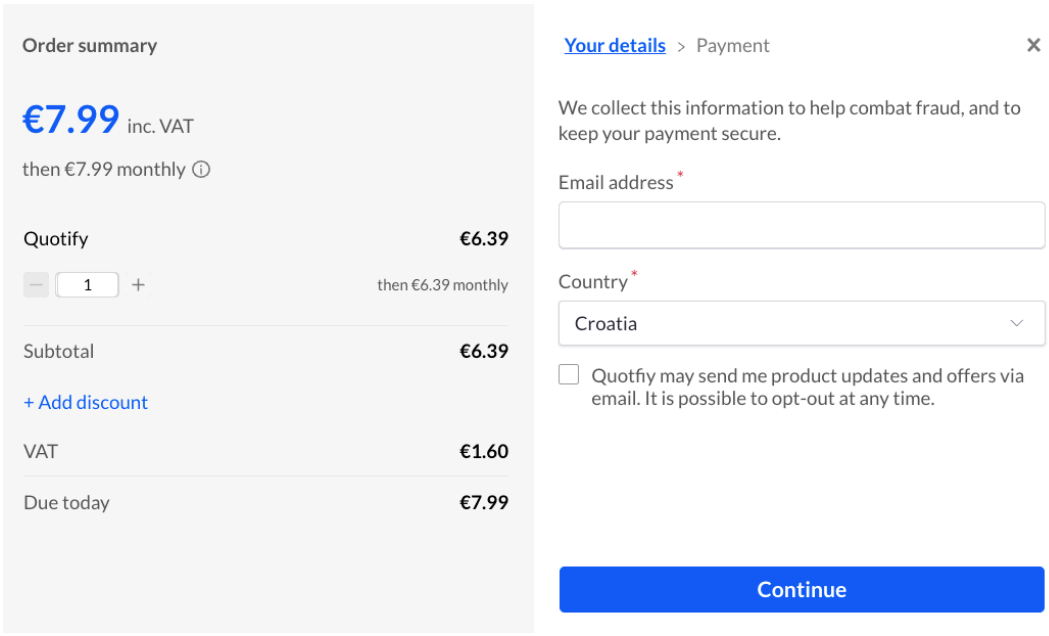
Odlaskom na stranicu s planovima pretplata putem klika na *Pricing* na navigacijskoj traci korisnik postoji opcija plaćanja pretplate i dobivanja plaćenog računa koji ima neograničeni broj korištenja. Besplatni plan je ograničen na 30 korištenja. Detalji o samim planovima vidljivi su u karticama tih planova (Slika 3.8).



Slika 3.8 – Pricing stranica

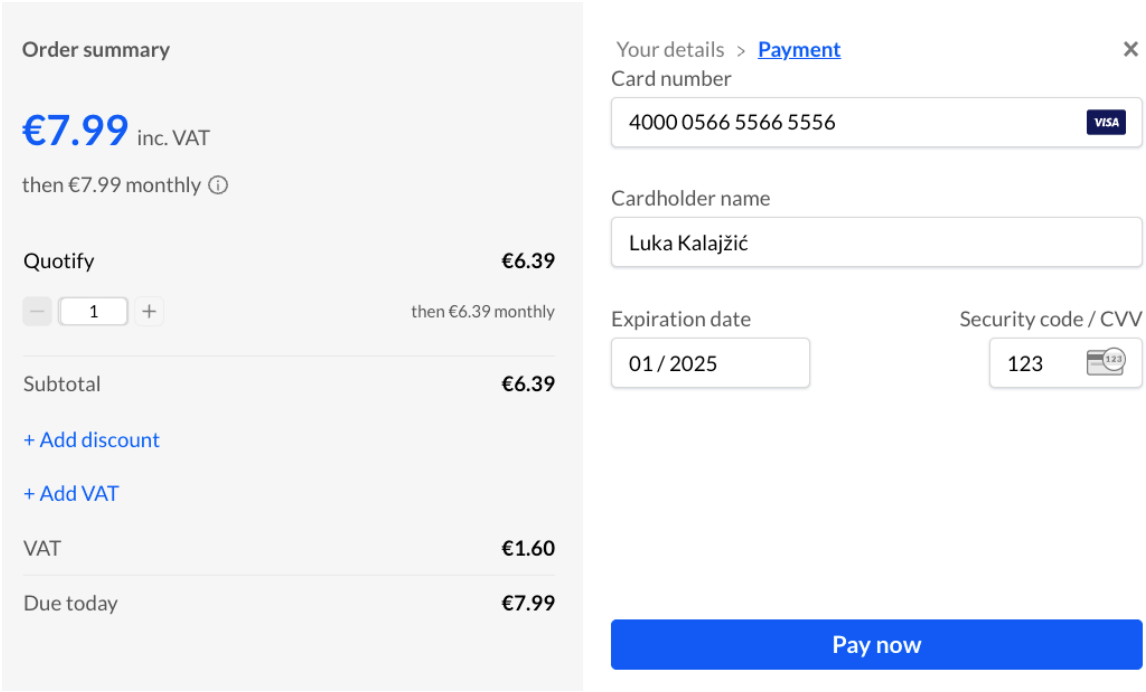
3.5 *Checkout*

Pritiskom na *Get started* gumb na *Pricing* stranici pokreće se *checkout* proces (Slika 3.9).



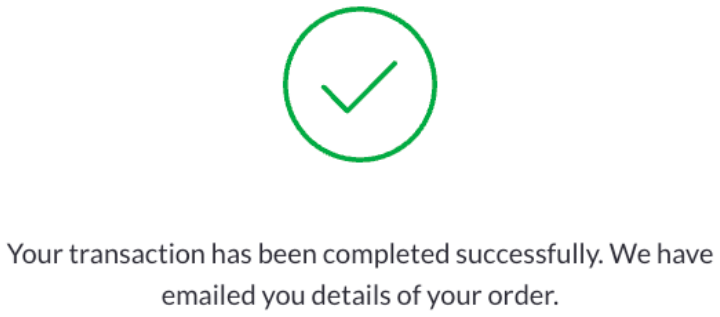
Slika 3.9 – *Checkout* – korak 1

Nakon unošenja email adrese, odabira svoje zemlje i pritiska na *Continue* gumb potrebno je unošenje podataka za plaćanje (Slika 3.10). U slici je korištena testna kartica koju pruža Paddle.



Slika 3.10 – *Checkout* - korak 2

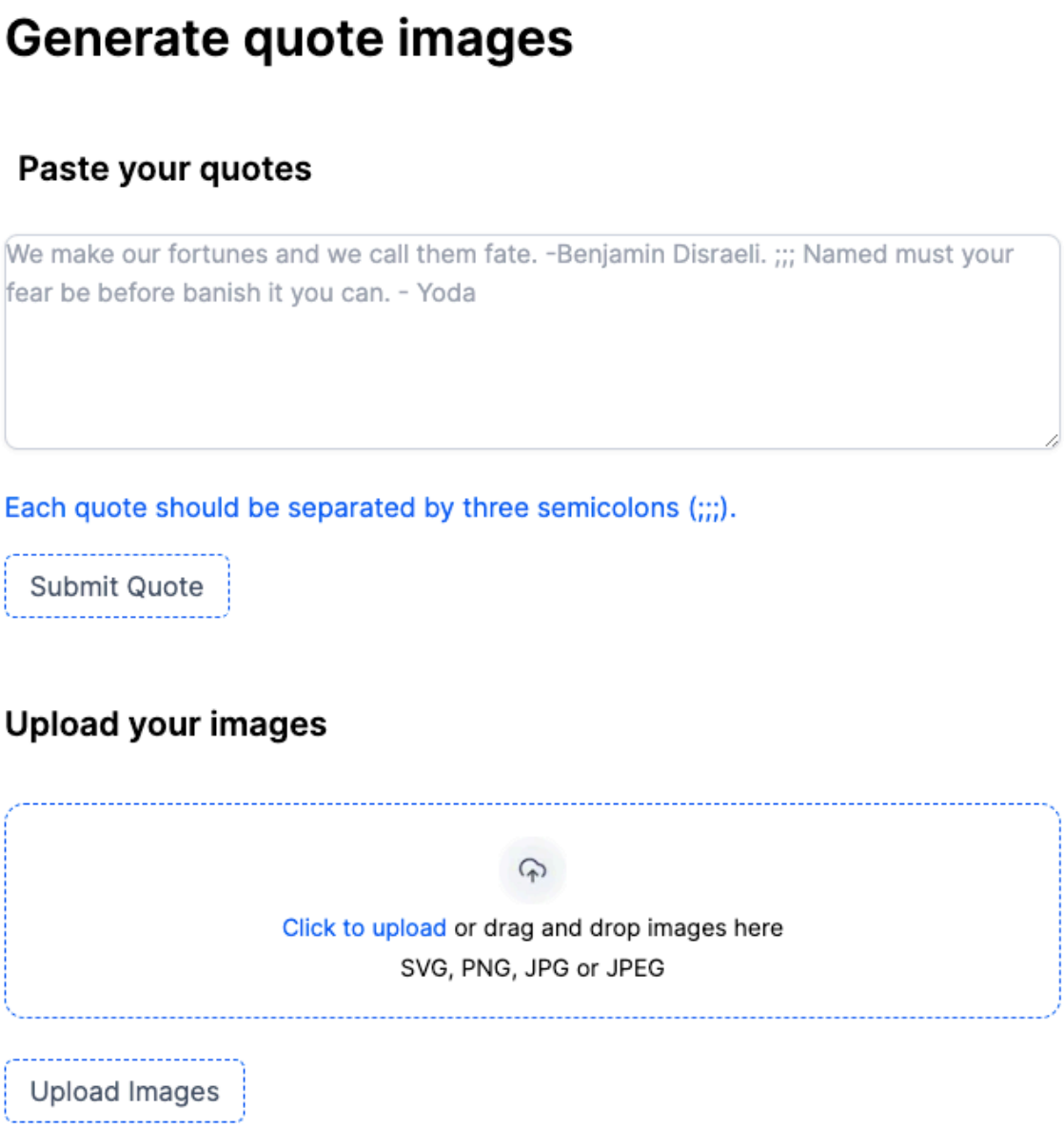
Pritiskom na *Pay now* gumb vidi se ekran za uspjeh plaćanja (Slika 3.11). Uspješnim plaćanjem korisnik dobiva dva emaila od Paddlea.



Slika 3.11 – *Checkout* uspjeh

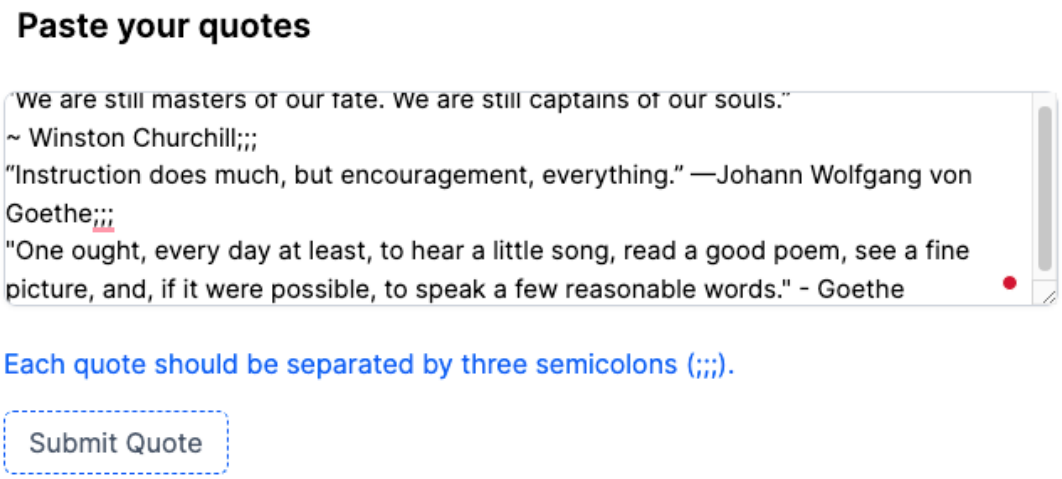
3.6 Podnošenje slika i citata

Korištenjem navigacijske trake za odlazak na *submission* stranicu, odnosno stranicu za podnošenje slika i citata, može se podnijeti slike i citati te obraditi te slike i citate (Slika 3.12).



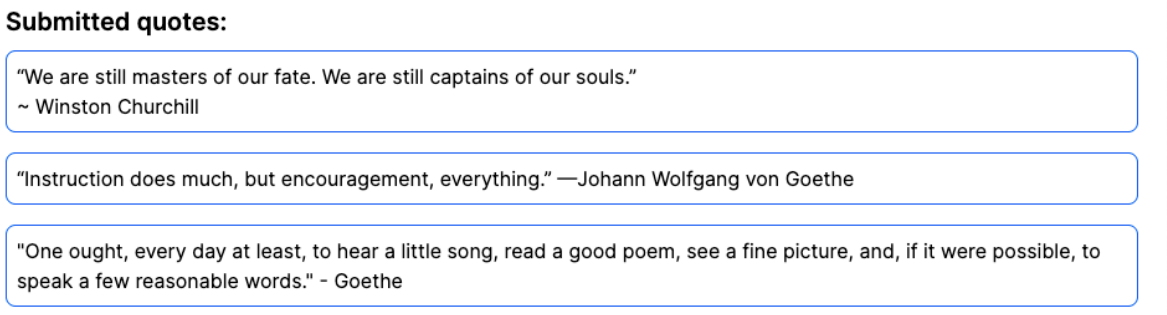
Slika 3.12 – Podnošenje slika i citata

Potrebno je unijeti citate u tekstni okvir. Citate je potrebno odvojiti korištenjem tri točke sa zarezom. Ta uputa je vidljiva ispod tekstnog okvira (Slika 3.13).



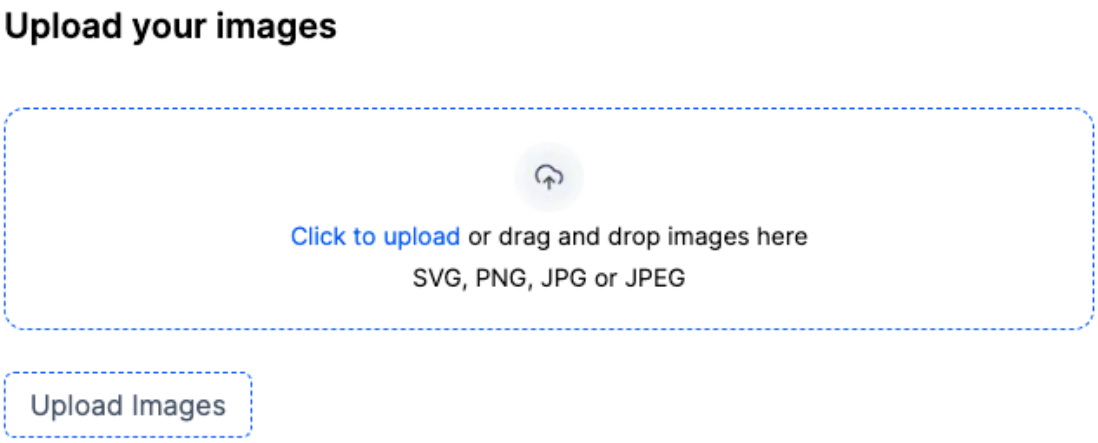
Slika 3.13 – Podnošenje citata

Pritiskom na gumb *Submit Quote* citat je dodan na drugu stranu ekrana uz sve ostale podnesene citate (Slika 3.14).



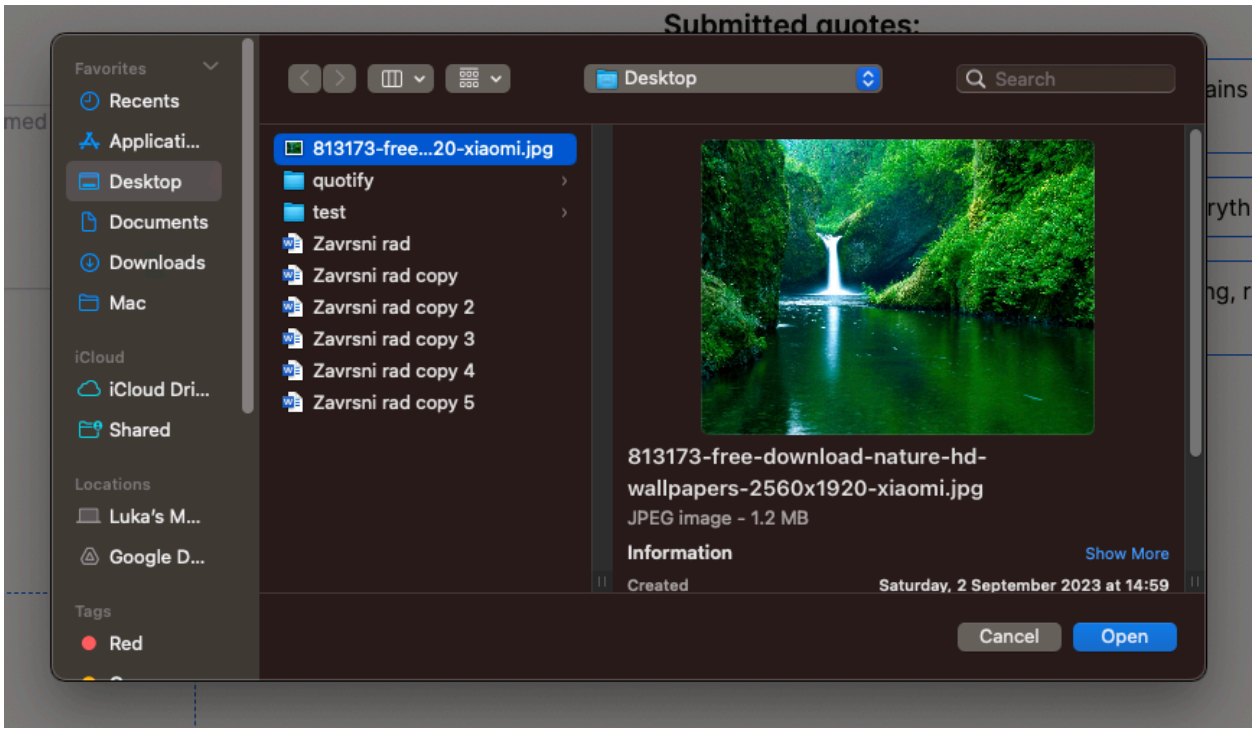
Slika 3.14 – Podneseni citati

Nakon podnošenja citata moguće je podnijeti slike pritiskom na *dropbox* zonu, koja je vidljiva na slici 3.15.



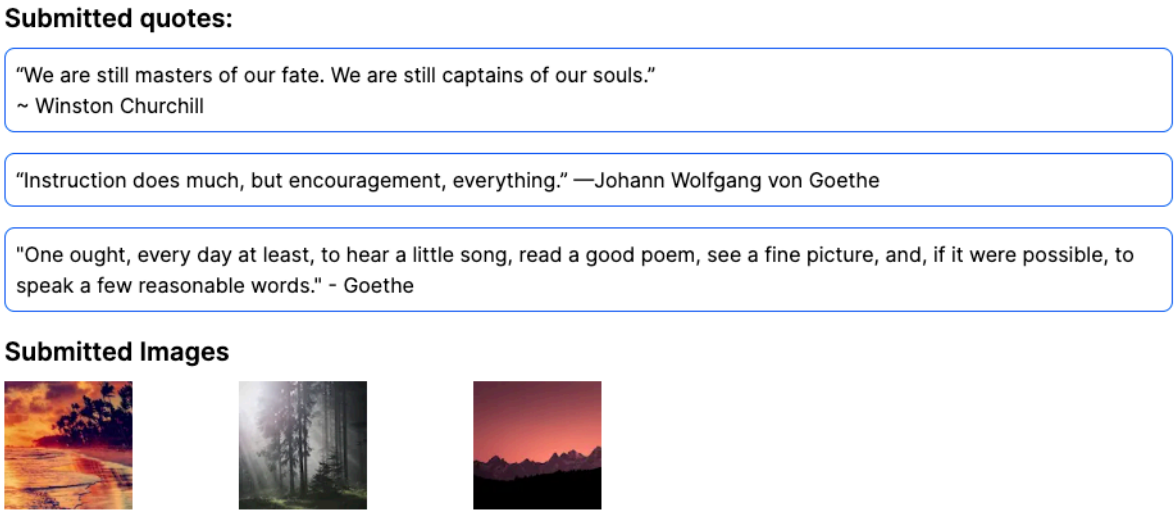
Slika 3.15 – Podnošenje slika

Pritiskom na *dropbox* zonu otvara se sustav za biranje slika korisnikovog operativnog sustava (Slika 3.16).



Slika 3.16 – Odabir slika

Pritiskom na *Upload images* gumb podnesene slike vidljive su sa desne strane ekrana (Slika 3.17).



Slika 3.17 – Podnesene slike

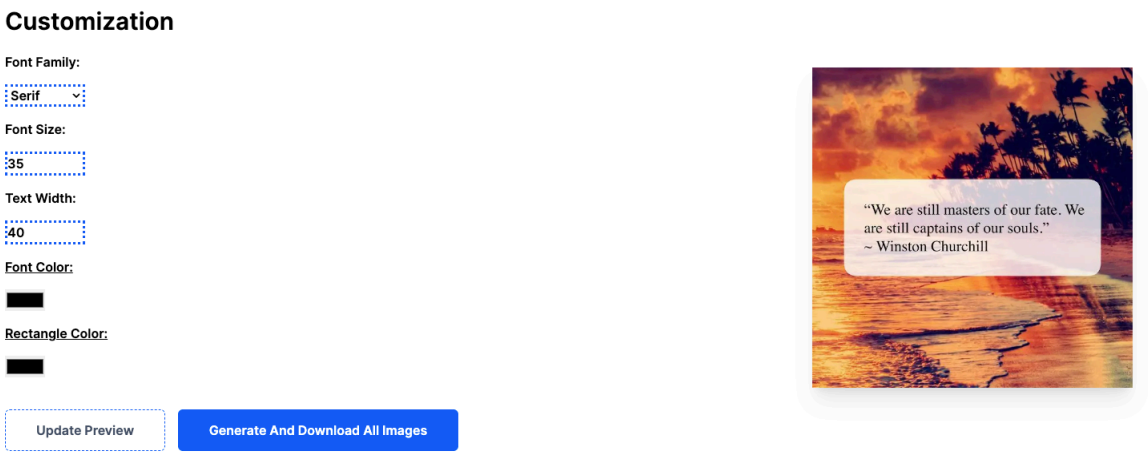
Pritiskom na gumb *Continue* moguće je doći na ekran na kojem se uređuju slike. Istim pritiskom na gumb stvara se slika koja služi kao pretpregled (eng. *preview*) napravljenih promjena.

3.7 Uređivanje slike

Na ovoj stranici moguće je uređivati sliku koristeći sljedeće opcije:

1. Promjena fonta tekst
2. Promjena boje teksta
3. Promjena veličine teksta
4. Promjena širine tekstnog okvira
5. Promjena boje pozadinskog četverokuta

Stranica je vidljiva na slici ispod (Slika 3.18).

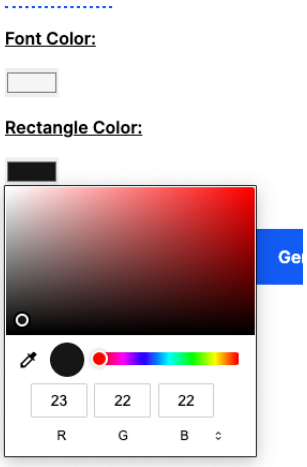


Slika 3.18 – Uređivanje slika

Korisniku su dane opcija biranje fonta i veličine teksta (Slika 3.19).

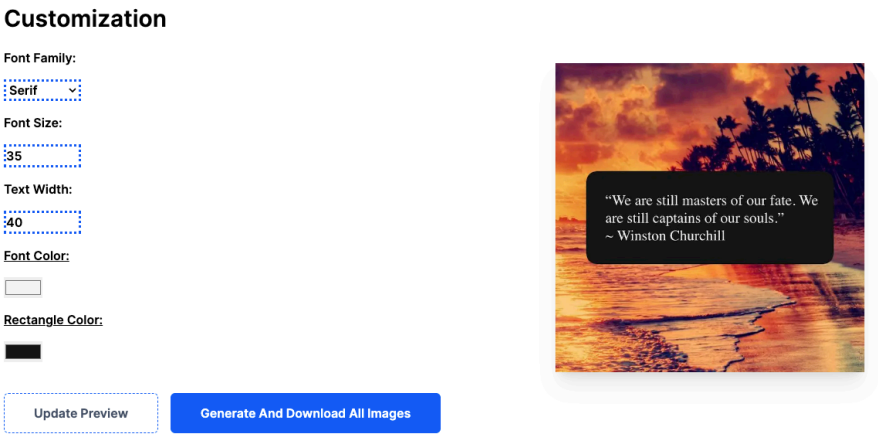
Slika 3.19 – Odabir fonta

Korisnik također može birati širinu tekstnog okvira, boju teksta i boju pozadinskog četverokuta (Slika 3.20).



Slika 3.20 – Odabir boje pozadinskog četverokuta

Rezultat nakon napravljenih promjena dobiven je pritiskom na gumb *Update preview* (Slika 3.21).

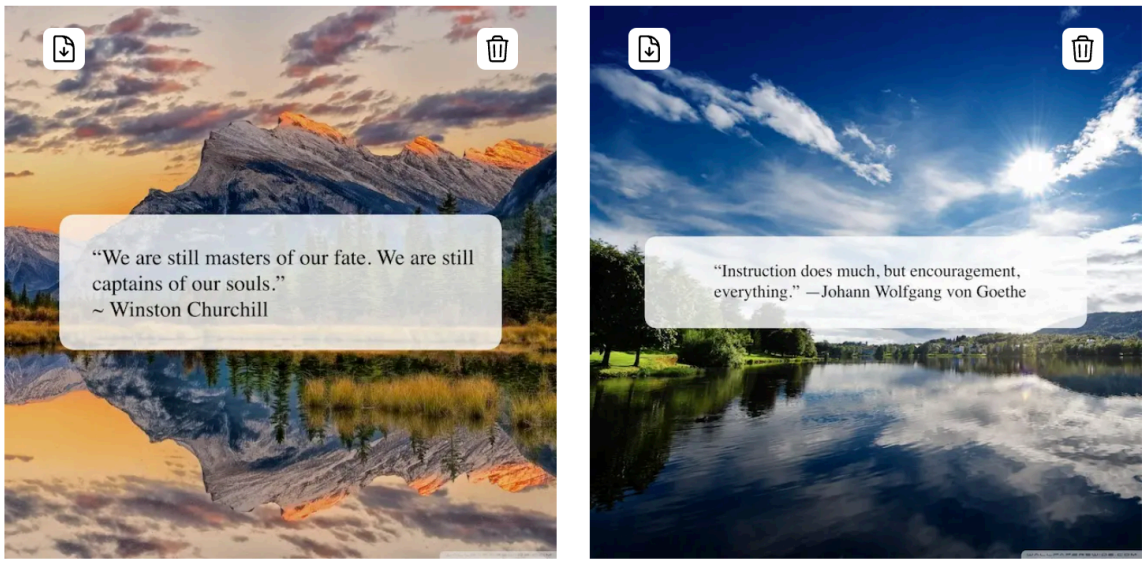


Slika 3.21 – Primjer uređene slike

Pritiskom na gumb *Generate and Download All Images* procesiraju se sve slike, odnosno primjene se citati sa odgovarajućim postavkama i skidaju se na korisnikov uređaj.

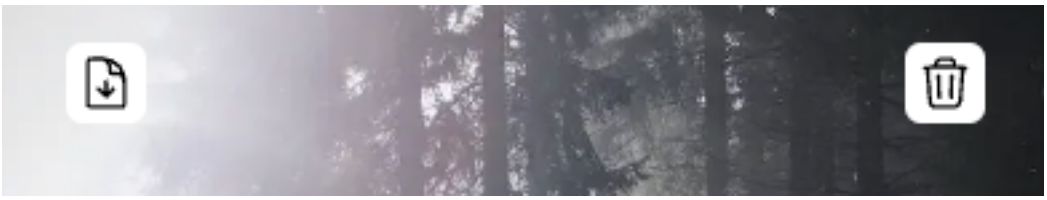
3.8 Generirane slike

Odlaskom na stranicu sa generiranih slikama moguće je vidjeti slike koje je korisnik prije napravio (Slika 3.22).



Slika 3.22 – Generirane slike

Te slike moguće je skinuti ponovno ili izbrisati (Slika 3.23).

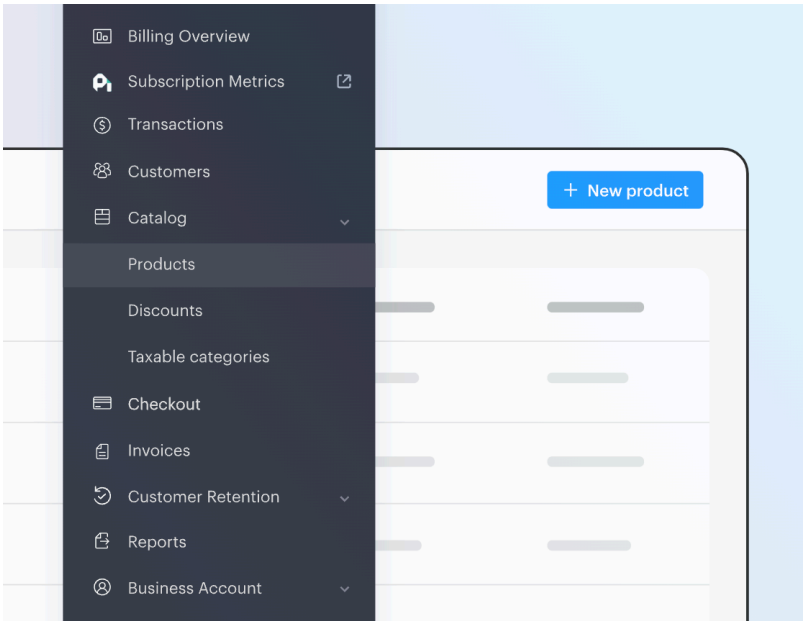


Slika 3.23 – Opcije skidanja i brisanja generiranih slika

4. Opis funkcionalnosti

4.1 Paddle *checkout*

Za implementacija Paddle plaćanja potrebno je prvo napraviti račun na Paddleu. Zatim, potrebno je napraviti proizvode da bi se plaćanje moglo izvršiti (Slika 4.1).



Slika 4.1 – Kreiranje proizvoda

4.1.1 *Pricing*

Na *pricing* stranici prikazano je korisničko sučelje koje vodi na *checkout* stranicu (Isječak koda 4.1).

```
---------------------------------------------------------------------------------------------------------------
import Image from 'next/image';
import Link from 'next/link';

export default function Home() {
  return (
    <main className="mt-12">
      <div className="flex flex-col justify-center items-center gap-4">
        <h1 className="font-semibold text-[48px] text-gray-900">
          Simple, transparent pricing
        </h1>
```

```
        <div>Simple, transparent pricing that grows with you.</div>
      </div>
      <div className="flex flex-row m-8 gap-20 justify-center">
        <div className="flex flex-col w-[384px] h-[440px] gap-5 items-center
shadow-xl rounded-xl mt-12">
          <Image
            className="mt-4"
            src="/free.png"
            width={50}
            height={50}
            alt=""
          />
          <div className="text-blue-600">Free plan</div>
          <div className="text-[48px] text-gray-900 flex flex-col">
            €0
            <span className="text-gray-600 text-[14px]">per month</span>
          </div>
          <div>
            <div className="flex flex-row items-center gap-4 text-gray-600">
              <Image
                src="/check.png"
                width={30}
                height={30}
                alt=""
                className="items-left"
              />
              <div>30 free credits</div>
            </div>
            <div className="flex flex-row items-center gap-4 text-gray-600">
              <Image
                src="/check.png"
                width={30}
                height={30}
                alt=""
                className="items-left"
              />
              <div>Basic chat and email support</div>
            </div>
          </div>
          <Link href="sign-up">
            <button className="rounded-md bg-blue-600 px-[18px] py-[10px]
text-white  hover:shadow-lg mt-12">
              Get started
            </button>
          </Link>
        </div>
        <div className="flex flex-col w-[384px] h-[520px] gap-5 items-center
shadow-md border border-blue-600 rounded-xl">
          <Image
            className="mt-12"
```

```
            src="/premium.png"
            width={50}
            height={50}
            alt=""
          />
          <div className="text-blue-600 text-[24px]">Premium plan</div>
          <div className="flex flex-col items-center">
            <div className="text-[48px] text-gray-900 flex flex-
col">€7.99</div>
            <span className="text-gray-600 text-[14px]">per month</span>
          </div>
          <div>
            <div className="flex flex-row items-center gap-4 text-gray-600">
              <Image
                src="/check.png"
                width={30}
                height={30}
                alt=""
                className="items-left"
              />
              <div>Unlimited credits</div>
            </div>
            <div className="flex flex-row items-center gap-4 text-gray-600">
              <Image
                src="/check.png"
                width={30}
                height={30}
                alt=""
                className="items-left"
              />
              <div>24 hour chat and email support</div>
            </div>
            <div className="flex flex-row items-center gap-4 text-gray-600">
              <Image
                src="/check.png"
                width={30}
                height={30}
                alt=""
                className="items-left"
              />
              <div>Priority processing</div>
            </div>
          </div>
          <a
            href="/checkout"
            className="rounded-md bg-blue-600 px-[48px] py-[10px] text-white
hover:shadow-lg mt-8 "
          >
            Get started
          </a>
```

```
        </div>
      </div>
    </main>
  );
}
```

Isječak koda 4.1 – *Pricing* stranica

Prvo su uvezeni *Image* i *Link* komponente iz *next* repozitorija. *Image* služi za prikazivanje slika koje su optimizirane za brzo učitavanje te se po zadanoj, *out-of-the-box* konfiguraciji *lazy-loadaju*, odnosno učitavaju se tek kada su potrebne. Na primjer, ako je slika na dnu stranice, onda neće biti učitana pri učitavanju stranice na početku što rezultira u bržem učitavanju stranice i time boljem iskustvu za korisnika. Slijedi *Home* komponentna u kojoj se nalazi ostatak koda. <*main*> oznaka sadrži sve ostale oznake. Nakon definiranja naslova i podnaslova definirana je oznaka <*div className="flex flex-row m-8 gap-20 justify-center"*> koja sadrži *pricing* kartice. U toj oznaci su definirane dodatne oznake za svaku karticu koja sadrži *Image* komponentu u svrhu prikazivanja ikone, <*div*> i <*span*> oznake za tekst koje sadrže kartice te <*button*> oznake za definiranje gumbova Pritiskom na gumb korisnik je odveden na *checkout*.

4.1.2 *Checkout*

Na *checkout* stranici nalazi se *CheckoutLoader* komponenta (Isječak koda 4.2). Ta komponenta prvo je uvezena iz datoteke *components,* a zatim je samo dodana u *page* komponentu. Nakon toga je ta *page* komponenta izvezena za daljnje korištenje.

```
import CheckoutLoader from '@/components/CheckoutLoader';

const page = () => {
  return <CheckoutLoader />;
};

export default page;
```

Isječak koda 4.2 – *Checkout* stranica

4.1.3 *.env* varijable

Prije nastavljanja važno je uključiti *.env* varijable (*VENDORID* i *API_KEY*) od Paddle računa (Isječak koda 4.3). Ove varijable su ključne u uspješnoj izvedbi Paddle *checkouta*.

```
NEXT_PUBLIC_PADDLE_VENDORID=14052
NEXT_PUBLIC_PADDLE_API_KEY=926e5c552b22f7fc7ce69496d8a2b22b964b1ee288b0e86ca6
```

Isječak koda 4.3 – *Paddle .env* varijable

4.1.4 CheckoutLoader komponenta

Ova komponenta je implementacija *overlay checkouta* [9]. Prvo je potrebno na vrhu napisati *„use client“;*, što označava da ovu JavaScript datoteku je potrebno prikazati na stranici klijenta, odnosno korisnikovog preglednika. Zatim je potrebno uključiti potrebne ovisnosti (Isječak koda 4.4).

```
'use client';

import Script from 'next/script';
import { useRouter } from 'next/navigation';
import { useUser } from '@clerk/nextjs';

import { createNewUser } from '@/utils/actions';
import { connectToDB } from '@/utils/database';
```

Isječak koda 4.4 – Uključivanje ovisnosti

Zatim je definirana React funkcionalna komponenta *CheckoutLoader* (Isječak koda 4.5). Korisnik se dobije koristeći funkciju *useUser* od Clerka, te je definiran *router* koristeći funkciju *userRouter* od Next.jsa. Router je korišten za usmjeravanje korisnika na određenu rutu, u ovom slučaju */pricing.* Slijedi definiranje *loading* stanja i prikazivanje * * oznake u slučaju da korisnik nije učitan .

```
const CheckoutLoader = () => {
  const { user, isLoaded } = useUser();
  const router = useRouter();

  if (!isLoaded)
    return (
      <div className="h-screen w-screen flex flex-row items-center justify-
center text-blue-600 text-[20px]">
        Loading...
```

```
      </div>
    );

  if (isLoaded) {
    return (
      <Script
        src="https://cdn.paddle.com/paddle/v2/paddle.js"
        onLoad={() => {
          if (process.env.NEXT_PUBLIC_NODE_ENV !== 'production') {
            Paddle.Environment.set('sandbox');
          }

          Paddle.Setup({
            seller: Number(process.env.NEXT_PUBLIC_PADDLE_VENDORID),
            eventCallback: async (data) => {
              if (data.name === 'checkout.error') {
                console.log('paddle checkout error');
                alert('An error has occured! Please try agin!');
                router.push('/pricing');
              }

              if (data.name === 'checkout.completed') {
                console.log('Checkout completed!');

                const response = await fetch('/api/create-user', {
                  method: 'POST',
                  body: JSON.stringify({
                    email: user.emailAddresses[0].emailAddress,
                  }),
                });
              }

              if (data.name === 'checkout.closed') {
                router.push('/pricing');
              }
            },
          });

          Paddle.Checkout.open({
            items: [
              {
                priceId: 'pri_01h8aaq7m3zagfs1p7ybcjzrdf',
                quantity: 1,
              },
            ],
            frameInitialHeight: 1000,
            frameStyle:
              'width:100%; min-width:312px; background-color: transparent;
border: none; min-height: 100vh; height: 100%; ',
            displayModeTheme: 'light',
```

```
          });
        }}
      />
    );
  }
};

export default CheckoutLoader;
```

Isječak koda 4.5 – *CheckoutLoader*

Prvi bitni korak uključivanje je *Paddle.js* skripte. Koristi se *Script* komponenta od Next.js-a, a *src* atribut vodi na skriptu koju je moguće pronaći u dokumentaciji. Slijedi provjera stadija aplikacije usporedbom s *.env* varijablom – *production, sandbox, development* (Isječak koda 4.5).

```
<Script
        src="https://cdn.paddle.com/paddle/v2/paddle.js"
        onLoad={() => {
          if (process.env.NEXT_PUBLIC_NODE_ENV !== 'production') {
            Paddle.Environment.set('sandbox');
          }
…
```

Isječak koda 4.5 – Paddle skripta

Paddleu potrebno je dati *VENDORID* varijablu. Zatim slijedi provjera i rješavanje pogrešaka. U slučaju pogreške, pomoću Next Routera korisnik je preusmjeren na *pricing* stranicu. U slučaju uspjeha, ispisana je poruka u konzolu. (Isječak koda 4.6).

```
Paddle.Setup({
            seller: Number(process.env.NEXT_PUBLIC_PADDLE_VENDORID),
            eventCallback: async (data) => {
              if (data.name === 'checkout.error') {
                console.log('paddle checkout error');
                alert('An error has occured! Please try agin!');
                router.push('/pricing');
              }

              if (data.name === 'checkout.completed') {
                console.log('Checkout completed!');
```

Isječak koda 4.6 – Postavljanje Paddlea

Zatim je korisnik je dodan u MongoDB bazu podataka koristeći email adresu kao primarni ključ i metodu POST (Isječak koda 4.7).

```
                const response = await fetch('/api/create-user', {
                  method: 'POST',
                  body: JSON.stringify({
                    email: user.emailAddresses[0].emailAddress,
                  }),
                });
              }

              if (data.name === 'checkout.closed') {
                router.push('/pricing');
              }
            },
          });
```

Isječak koda 4.7 – Dodavanje korisnika u bazu podataka

Podaci korisnika u MongoDB bazi podataka izgledaju ovako (Slika 4.2).

Slika 4.2 – MongoDB baza podataka

U kodu slijedu otvaranje, odnosno pokretanje Paddle *checkout-a* (Isječak koda 4.8). Potrebno je specificirati:

1. Proizvode – varijabla *items*
2. Početnu količinu proizvoda - *quantity: 1*

3. *id* cijene - *priceId: 'pri_01h8aaq7m3zagfs1p7ybcjzrdf'*
4. Visinu okvira u pikselima - *frameInitialHeight: 1000,*

Moguće je promijeniti stil okvira pomoću CSS koda te temu (svijetla ili tamna) modificiranjem *frameStyle* varijable.

```
          Paddle.Checkout.open({
            items: [
              {
                priceId: 'pri_01h8aaq7m3zagfs1p7ybcjzrdf',
                quantity: 1,
              },
            ],
            frameInitialHeight: 1000,
            frameStyle:
              'width:100%; min-width:312px; background-color: transparent;
border: none; min-height: 100vh; height: 100%; ',
            displayModeTheme: 'light',
          });
        }}
      />
    );
  }
};

export default CheckoutLoader;
```

Isječak koda 4.8 – Pokretanje Paddle *checkouta*

4.2 Primjena citata na slike

U ovom dijelu objašnjena je poslužiteljska strana na kojoj se primaju citati, obrađuju, primjenjuju na slike.

4.2.1 Inicijaliziranje poslužiteljske strane

Na početku potrebno je uvesti sve potrebne ovisnosti (Isječak koda 4.9).

```
import express from "express";
const app = express();
const port = 8000;
import cors from "cors";
import path from "path";
import multer from "multer";
import { fileURLToPath } from "url";

import { applyQuoteToImage } from "./quoteImageProcessor.js";

// Allow requests from http://localhost:3000
app.use(cors({ origin: "http://localhost:3000" }));

app.use(express.json());
app.use(express.urlencoded({ extended: true }));

// GET route for the root path
app.get("/", (req, res) => {
  res.send("Hello!");
});
```

Isječak koda 4.9 – Inicijaliziranje poslužiteljske strane

Uveden je *express*. Zatim je pokrenuta *express* aplikacija. Definiran je *port* na kojoj će biti poslužiteljska strana, u ovom slučaju 8000. Slijedi uvoz *cors* paketa, odnosno *Cross-origin resource sharing*, što omogućava komunikaciju između aplikacija na različitim *portovima*. U ovom slučaju to je između klijentske strane (3000) i poslužiteljske strane (8000). Zatim je uvoz *path* paketa koji služi za rad s putanjama datoteka. Slijedi *multer* koji služi za obradu datoteka. Uvezen je i *fileURLToPath*, koji je ključan pri funkcioniranju rada s datotekama u ES6 načinu modul uvezivanja. Zadnji se uvozi *applyQuoteToImage* iz istog direktorija. To je funkcija koja služi za primjenu citata na sliku, a vraća *path* rezultirajuće slike. Nakon uvoza, potrebno je iskoristiti cors paket i dopustiti zahtjeve s klijentske strane, odnosno http://localhost:3000. U sljedećim dvjema linijama koda postavlja se *middleware* koji omogućava parsiranje JSON i URL-enkodiranih podataka iz dolaznih HTTP zahtjeva. Konačno, definirana je GET ruta na korijenu aplikacije (“/”). Pri posjećivanju korijenske aplikacije, dakle http://localhost:8000, poslužitelj će odgovoriti s porukom “Hello!” (Slika 4.3).

Slika 4.3 – Poruka dobivena posjećivanjem korijena *express* aplikacije

4.2.2 Citati i slike

Slijedi definiranje polja za citate i za puteve dobivenih slika (Isječak koda 4.10).

```
// Array to store user-submitted quotes
let quotes = [];

// Array to store submitted image paths
let images = [];

// Custom trim function to remove leading spaces and spaces between the last
character and the delimiter
const customTrim = (str, delimiter) => {
  let trimmed = str.trim(); // Remove leading and trailing spaces
  if (delimiter) {
    const delimiterIndex = trimmed.lastIndexOf(delimiter);
    if (delimiterIndex !== -1) {
      const lastNonSpaceIndex = trimmed.length - 1;
      if (delimiterIndex === lastNonSpaceIndex) {
        // If the delimiter is at the last non-space character, remove any
spaces before it
        trimmed = trimmed.substring(0, delimiterIndex).trim();
      } else {
        // If the delimiter is not at the last non-space character, keep the
string as is
        trimmed = trimmed;
      }
    }
  }
  return trimmed;
};
```

Isječak koda 4.10 – Citati i slike

Bitno je spomenuti *customTrim* funkciju. Ta funkcija prima niz znakova i *delimiter.* Zatim rastavlja taj niz znakova na zasebne citate ovisno o vrijednosti *delimitera.* Na početku se miču nepotrebni razmaci na početku i kraju niza znakova. Ako ne postoji *delimiter*, odmah se vraća niz znakova *trimmed*. Ako postoji, ispituje se postoji li uopće *delimiter* u tekstu, tražeći poziciju posljednjeg pojavljivanja *delimiter* u tekstu *trimmed.* Ako *delimiter* nije prisutan u tekstu, *delimiterIndex* poprimiti će vrijednost -1.

const lastNonSpaceIndex = trimmed.length - 1; računa indeks posljednjeg znaka u tekstu *trimmed* ignorirajući razmake na kraju teksta. Ako *delimiterIndex* nije -1, odnosno *delimiter* je pronađen u tekstu, i ako je *delimiter* na kraju teksta, odnosno na poziciji *lastNonSpaceIndex*, funkcija otklanja sve razmake koji se nalaze prije *delimitera* pomoću *substring* metode kako bi dobio dio teksta prije *delimitera.* Zatim se primjenjuje funkcija *trim* na taj dio teksta. Na kraju vraća se *trimmed* niz znakova.

4.2.3 Primanje citata

Za primanje citata napravljen je API *endpoint* koji prima *quote* polje znakova. Zatim se *quote* razdvaja na individualne citate i stavlja u *quotes* polje znakova, gdje je svako polje jedan citata. Na kraju se ispisuju rezultirajući citati u polje (Isječak koda 4.11).

```
-----------------------------------------------------------------------------------------------------------------
// Route for submitting quotes
app.post("/api/submit-quotes", (req, res) => {
  const { quote } = req.body;

  if (quote) {
    if (Array.isArray(quote)) {
      // If multiple quotes are submitted as an array
      quote.forEach((individualQuote) => {
        const quotesArray = individualQuote.split(";;;");
        quotes.push(
          ...quotesArray
            .map((q) => customTrim(q, ";;;"))
            .filter((q) => q !== "")
        ); // Trim and filter out empty strings
      });
    } else {
      // If a single quote is submitted
      const quotesArray = quote.split(";;;");
      quotes.push(
        ...quotesArray.map((q) => customTrim(q, ";;;")).filter((q) => q !== "")
      ); // Trim and filter out empty strings
```

```
    }

    console.log("Submitted Quotes:", quotes);
    res.json({ success: true, message: "Quotes submitted successfully." });
  } else {
    res.status(400).json({ success: false, message: "Invalid input." });
  }
});
```

Isječak koda 4.11 – Primanje citata

4.2.4 Konfiguracija *multer storagea*

Potrebno je postaviti *multer storage* kako bi poslužitelj spremao slike koje dobiva od korisnika. Stvara se *upload*s direktorij u kojem se spremaju slike. Slike se imenuju tako da se na originalno ime slike dodaje trenutni datum (npr. wallpaper_1693824198007.jpg). Kod za konfiguraciju vidljiv je na isječku koda 4.12.

```
// Multer configuration for handling file uploads
const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, "uploads/"); // The directory where uploaded images will be stored
  },
  filename: function (req, file, cb) {
    cb(null, Date.now() + "-" + file.originalname); // Set the file name to be
unique
  },
});

const upload = multer({ storage: storage }); // Set up the multer middleware
```

Isječak koda 4.12 – Konfiguracija *multer storagea*

4.2.5 Primanje slika

Za primanje slika napravljen je API *endpoint* koji prima slike. Prvo se provjerava postoje li slike u zahtjevu. Poslužitelj odgovara sa statusom 400 ako ne postoje slike. Ako postoje slike, putanje slika dodaju se u polje *images.* To se postiže korištenjem *req.files.map()* kako bi se dobile putanje za svaku dostavljenu sliku, a zatim se te putanje dodaju u *images* array korištenjem *images.push.* Slijedi ispisivanje putanja slika u konzolu. Cijeli kod je obuhvaćen *try catch* blokom

kako bi u slučaju neuspjeha poslužitelj mogao naznačiti klijentu da je došlo do pogreške (status 500). Spomenuti kod vidljiv je u isječku koda 4.13.

```
// Route for submitting images
app.post("/api/submit-images", upload.array("images"), (req, res) => {
  try {
    if (!req.files || req.files.length === 0) {
      return res
        .status(400)
        .json({ success: false, message: "No images uploaded." });
    }

    // Update the images array with the new image paths
    images.push(...req.files.map((file) => file.path));

    // Access the array of uploaded files using req.files
    console.log("Submitted Images:", images);

    res.json({
      success: true,
      message: "Images submitted successfully.",
      images: images,
    });
  } catch (error) {
    console.error("Error uploading images:", error);
    res.status(500).json({
      success: false,
      message: "An error occurred while uploading the images.",
    });
  }
});
```

Isječak koda 4.13 – Primanje slika

4.2.6 *quoteimageprocessor.js*

Ova funkcija zaslužna je za logiku primjene citata na slike. Prvo se uvoze sve potrebne ovisnosti (Isječak 4.14).

```
import * as fs from "fs";
import * as path from "path";
import { fileURLToPath } from "url";
```

```
import { createCanvas, loadImage } from "canvas";
import wrap from "word-wrap";
```

Isječak koda 4.14 – Uvoz ovisnosti

Zatim su definirani potrebni parametri.

1. *quote* -> citat, znakovni niz
2. imagePath -> putanja slike, znakovni niz
3. fontFamily -> font, znakovni niz
4. fontSize -> veličina teksta, *integer*
5. fontColor -> boja teksta, znakovni niz
6. textWidth -> veličina tekstnog okvira, *integer*
7. rectangleColor -> boja pozadinskog četverokuta, znakovni niz

Učitava se slika pomoću njene putanje. Stvara se *canvas* element dimenzija slike *(img.width, img.height).* Definira se *ctx* konstanta, i crta slika počevši od pozicije (0, 0), a sve do visine i širine *canvasa.* Definira se font koristeći *ctx.font = `${fontSize}px ${fontFamily}`;* Određuje se maksimalna širina pozadinskog četverokuta koji služi za bolju vidljivost teksta na svim slikama svih boja. Definira se *lineHeight* faktor koji određuju visinu tekstnih linija. Citat se zatim *wrapa* na maksimalnu širinu zadanu u pikselima parametrom *textWidth*. Rezultirajući citat se ispisuje u konzolu. Sve ovo vidljivo je na isječku koda 4.16.

```
// Function to apply a quote to an image using canvas
export const applyQuoteToImage = async (
  quote,
  imagePath,
  fontFamily,
  fontSize,
  fontColor,
  textWidth,
  rectangleColor
) => {
  try {
    const img = await loadImage(imagePath);
    const canvas = createCanvas(img.width, img.height);
    const ctx = canvas.getContext("2d");
    ctx.drawImage(img, 0, 0, canvas.width, canvas.height);

    // Use fontFamily, fontSize, and fontColor for customization
    ctx.font = `${fontSize}px ${fontFamily}`;
```

```
    const maxRectWidth = canvas.width * 0.8;
    const lineHeight = 1.2; // Line height factor

    const wrappedQuote = wrap(quote, { width: textWidth });
    console.log(wrappedQuote);
```

Isječak koda 4.16 – Početak funkcije i obrada citata

Slijedi brojanje novih redova. Ovo je ključno pri određivanje točne visine četverokuta jer *wrap* funkcija stvara više redova. Na početku je definiran *newlineCount*, a zatim se provjerava postojanje dvostrukih redova (na primjer "\n\n"). Zatim se iterira kroz svaki znak citata i povećava se *newlineCount* za svaki novi red. Zatim se određuje visina četverokuta mjerenjem visine teksta u pikselima i množenjem s *lineHeight* konstantom. Za dobivanje visine teksta u pikeselima koriste se metode *canvas* knjižnice (Isječak koda 4.17).

```
    let newlineCount = 0;
    if (/[\r\n]{2,}/.test(quote)) newlineCount++;
    for (let i = 0; i < quote.length; i++) {
      if (quote[i] === "\n") {
        newlineCount++;
      }
    }

    const totalHeight =
      (ctx.measureText(wrappedQuote).actualBoundingBoxAscent +
        ctx.measureText(wrappedQuote).actualBoundingBoxDescent) *
      lineHeight;
```

Isječak koda 4.17 – Računanje broja novih redova i visine teksta

Slijedi crtanje pozadinskog četverokuta. Određuje se visina četverokuta zbrajanjem visine teksta i dodavanjem visine jednog reda pomnožene s 3. Određuju se x i y koordinate u pikselima. Tamo će krenuti crtanje četverokuta. Ispunjava se bojom koja je zadana kao parameter u funkciji *applyQuoteToImage*, i postoji *radius* varijable vrijednosti 30 koja omogućava zaobljene rubove. Koristeći funkcije *canvas* knjižnice crta se četvorkut iz jednog kuta u drugi (Isječak koda 4.18).

```
// Determine rectangle height and position
    const rectHeight = totalHeight + fontSize * 3;
    const rectX = (canvas.width - maxRectWidth) / 2;
    const rectY = (canvas.height - rectHeight) / 2;
```

```
    // Draw rounded rectangle with semi-transparent fill
    ctx.fillStyle = rectangleColor;
    ctx.strokeStyle = rectangleColor;
    const radius = 30;

    ctx.beginPath();
    ctx.moveTo(rectX + radius, rectY);
    ctx.lineTo(rectX + maxRectWidth - radius, rectY);
    ctx.quadraticCurveTo(
      rectX + maxRectWidth,
      rectY,
      rectX + maxRectWidth,
      rectY + radius
    );
    ctx.lineTo(rectX + maxRectWidth, rectY + rectHeight - radius);
    ctx.quadraticCurveTo(
      rectX + maxRectWidth,
      rectY + rectHeight,
      rectX + maxRectWidth - radius,
      rectY + rectHeight
    );
    ctx.lineTo(rectX + radius, rectY + rectHeight);
    ctx.quadraticCurveTo(
      rectX,
      rectY + rectHeight,
      rectX,
      rectY + rectHeight - radius
    );
    ctx.lineTo(rectX, rectY + radius);
    ctx.quadraticCurveTo(rectX, rectY, rectX + radius, rectY);
    ctx.closePath();

    ctx.fillStyle = rectangleColor;
    ctx.fill();
    ctx.strokeStyle = rectangleColor;
    ctx.stroke();
```

Isječak koda 4.18 – Crtanje pozadinskog četverokuta

Slijedi obično ispisivanje teksta korištenjem funkcija *canvas* knjižnice. Koristi se boja teksta dana u funkciji, određuje širina teksta i time x i y koordinate gdje kreće tekst. Zatim se ispisuje tekst (Isječak koda 4.19).

```
    // Draw text
    ctx.fillStyle = fontColor;
```

```
    // Calculate the width of the wrapped text
    const wrappedTextWidth = ctx.measureText(wrappedQuote).width;

    // Draw the centered wrapped text
    const x = (canvas.width - wrappedTextWidth) / 2;
    let y = (canvas.height - totalHeight) / 2 + fontSize * 0.8;

    let lines = wrappedQuote.split("\n");
    console.log("lines", lines);
    for (let i = 0; i < lines.length; i++) {
      ctx.fillText(lines[i], x, y);
      y += fontSize * lineHeight;
    }
```

Isječak koda 4.19 – Ispisivanje teksta

Na kraju rezultirajuću sliku potrebno je spremiti, a sprema se u *output* direktorij. Ako ne postoji direktorij, stvara se. U slučaju pogreške u konzoli se ispisuje pogreška. Kod je vidljiv na isječku koda 4.20.

```
    // Save the image with the quote applied
    const __filename = fileURLToPath(import.meta.url);
    const __dirname = path.dirname(__filename);
    const outputDir = path.join(__dirname, "output");
    if (!fs.existsSync(outputDir)) {
      fs.mkdirSync(outputDir);
    }
    const outputImagePath = `output/wallpaper_${Date.now()}.jpg`; // Relative
path
    const out = fs.createWriteStream(outputImagePath);
    const stream = canvas.createJPEGStream();
    stream.pipe(out);
    console.log(`Image with quote applied saved at: ${outputImagePath}`);

    return outputImagePath; // Return the relative URL path
  } catch (error) {
    console.error("Error applying quote to image:", error);
  }
};
```

Isječak koda 4.20 – Spremanje slike

4.2.7 Dobivanje slika s klijentske strane

Kako bi korisnik dobio slike s klijentske strane se poziva *api endpoint /api/get-quotes*. Korisnik u tom zahtjevu određuje font, veličinu i boju teksta, širinu tekstnog okvira te boju pozadinskog četverokuta. Te modifikacije može raditi s klijentske strane u korisničkom sučelju. Ovaj kod prolazi kroz sve citate i primjenjuje ih na slike korištenje *applyQuoteToImage* funkcije dajući joj parametre koje je korisnik odabrao. U slučaju da je korisnik dao više citata od slika, program će ponovno početi od prve slike i primijeniti citate iznova sve dok svi citati nisu primjenjeni. Prvo je potrebno definirati polje znakova *generatedImagePaths* koje će poslije biti modificirano. Rezultirajuće putanje spremaju se u polje znakova *generatedImagePaths*, koje su dio odgovora od strane poslužitelja. Ovo je vidljivo na isječku koda 4.21.

```
app.get("/api/get-quotes", async (req, res) => {
  const { fontFamily, fontSize, fontColor, textWidth, rectangleColor } =
    req.query;
  let generatedImagePaths = [];
  let generatedImagePath;

  try {
    // Loop through each submitted quote
    for (let i = 0; i < quotes.length; i++) {
      const quote = quotes[i];
      const imagePath = images[i % images.length]; // Use modulo to cycle
through images

      generatedImagePath = await applyQuoteToImage(
        quote,
        imagePath,
        fontFamily,
        fontSize,
        fontColor,
        textWidth,
        rectangleColor
      );
      generatedImagePaths.push(generatedImagePath);
    }
    if (generatedImagePaths) {
      res.json({ success: true, generatedImagePaths });
    } else {
      res.json({
        success: false,
        message: "An error occurred while processing the images .",
      });
    }
```

```
  } catch (error) {
    console.error("Error applying quotes to images:", error);
  }
});
```

Isječak koda 4.21 – Dobivanje slika s klijentske strane

Zaključak

U suvremenom digitalnom i brzo napredujućem dobu tehnološka pismenost ključni je faktor u produktivnosti čovjeka. Tehnološki sposobni pojedinci mogu olakšati i ubrzati repetitivne zadatke i poslove i time uštediti vrijeme i resurse sebi i društvu, rezultirajući u višoj kvaliteti života.

Jedan takav primjer tehnološke sposobnosti jest aplikacija koja je temelj ovog završnog rada, koja može tjedno uštediti sate kreatorima sadržaja, oglašavačima, i za sve one koje žele brzu inspiraciju i motivaciju na pozadini svojih uređaja. Skaliranjem nekoliko sata tjedno na razdoblje od godinu dana moguće je uštediti stotine sati. Sa strane nekih tvrtki to rezultira u znatnim uštedama.

Uzevši u obzir da ljudi danas provjeravaju svoj mobilni uređaj više desetina, ako ne i stotinu i više puta dnevno, ova aplikacija omogućava stvaranje vrijednih podsjetnika na ideje i misli koje se lako zametnu u žurnom modernom životu kada primijenjene na pozadine uređaja. Engleski pisac Samuel Johnson rekao je: *„People need to be reminded more often than they need to be instructed“* [10]. Ova aplikacija jest lagan, brz, i pogodan način prisjećavanja bilo kakvih lekcija i ideja izrečenih u tekstualnom obliku.

U razvoju aplikacije korištena je MERN arhitektura (MongoDB, Express.js, React i Node.js), Next.js kao UI okvir za React, Clerk za autentifikaciju korisnika i Paddle za procesiranje plaćanja i upravljanje pretplatama.

U budućnosti moguće je unaprijediti aplikaciju dodatkom novih značajki i ubrzanjem aplikacije implementacijom boljih praksi. Jedan takav primjer dodatne značajke bi bio automatski raspored i objava rezultirajućih slika na socijalne mreže poput *Instagrama* i *Twittera*. Ta značajka bi efektivno u nekoliko minuta stvorila dovoljno sadržaja za godinu dana objavljivanja, što sigurno uključuje više od sto uštođenih sati na godinu.

Softverski projekt nije nikad zaista gotov, te aplikacija ima dobar potencijalan u uštedi vremena i resursa brojnim korisnicima.

Literatura

[1] „Mongo DB”. (2023.), adresa: https://www.mongodb.com (pogledano 3. 9. 2023.).

[2] „Express JS”. (2023.), adresa: https://expressjs.com (pogledano 3. 9. 2023.).

[3] „React JS”. (2023.), adresa: https://reactjs.org (pogledano 3. 9. 2023.).

[4] „Node JS”. (2023.), adresa: https://nodejs.dev/en/ (pogledano 3. 9. 2023.).

[5] „The V8 JavaScript Engine”. (2023.), adresa: https://nodejs.dev/en/learn/the-v8-javascript-engine (pogledano 3. 9. 2023.).

[6] „Next.js”. (2023.), adresa: https://nextjs.org/ (pogledano 3. 9. 2023.).

[7] “Clerk”, (2023.), adresa: https://clerk.com/ (pogledano 3. 9. 2023.).

[8] “Clerk Docs”, (2023.), adresa: https://clerk.com/docs (pogledano 3. 9. 2023.).

[9] “Paddle”, (2023.), adresa: https://www.paddle.com/ (pogledano 3. 9. 2023.).

[10] „Goodreads“, (2023.), adresa: https://www.goodreads.com/quotes/578531-people-need-to-be-reminded-more-often-than-they-need (pogledano 11. 9. 2023.).

Sažetak

U današnjem digitalnom dobu tehnološka pismenost je izuzetno važna. Važnost tehnološke pismenosti očituje se u znatno većoj produktivnosti i širim opcijama tehnološki pismenih. Korištenjem modernih alata za izradu programa poput MERN arhitekture, Next.jsa, Clerka, i Paddlea razvijena je web aplikacija za generiranje slika na temelju citata i slika koje unese korisnik. Aplikacija omogućuje kreiranje stotine slika u minuti. U slučaju tehnološke nepismenosti ista radnja bi zahtijevala više sati ručnim radom u uređivačima slika poput *Photoshopa* ili *Canve*. Osim generiranja slika od unesenog sadržaja, aplikacija ima mogućnost modificiranja generirane slike u vidu fonta, boje fonta i pravokutnika koji se nalazi iza teksta u boji.

***Ključne riječi* --- MERN, Next.js, Clerk, Paddle, React.js**

Abstract

In today's digital age, technological literacy is extremely important. The importance of technological literacy is reflected in the significantly higher productivity and broader options of the technologically literate. Using modern tools for creating programs such as MERN architecture, Next.js, Clerk, and Paddle, a web application was developed for generating images based on quotes and images entered by the user. The application enables users to create hundreds of images per minute. In the case of technological illiteracy, the same action would require many hours of manual work in image editors such as *Photoshop* or *Canva*. In addition to generating images from the entered content, the application has the ability to modify the generated image in the form of font, font color and the rectangle that is behind the colored text.

***Keywords* --- MERN, Next.js, Clerk, Paddle, React.js**