Web aplikacija za povezivanje naručitelja i izvođača poslova

Starčić, Roberta

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

Završni rad

Web aplikacija za povezivanje naručitelja i izvođača poslova

Roberta Starčić

Pula, rujan, 2024. godine

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

Završni rad

Web aplikacija za povezivanje naručitelja i izvođača poslova

(Web application for connecting clients and service providers)

Roberta Starčić

JMBAG: 0351011388

Studijski smjer: Informatika

Znanstveno područje: Društvene znanosti

Znanstveno polje: Informacijske i komunikacijske znanosti

Znanstvena grana: Informacijski sustavi i informatologija

Kolegij: Web aplikacije

Mentor: doc. dr. sc. Nikola Tanković

Pula, rujan, 2024. godine

Sažetak

Ovaj završni rad opisuje razvoj i implementaciju web aplikacije koja omogućuje povezivanje naručitelja i izvođača poslova, oslanjajući se na modernu troslojnu arhitekturu. Aplikacija je usmjerena na ljude koji traže posao te na pojedince ili tvrtke koje traže zaposlenike za jednokratne poslove na temelju Ugovora o djelu. Aplikacija koristi Vue.js i Quasar za razvoj prezentacijskog sloja koji pruža intuitivno korisničko sučelje, Node.js i Express.js za upravljanje poslovnom logikom u aplikacijskom sloju, te PostgreSQL i Sequelize ORM za efikasno upravljanje podacima u sloju podataka. U ovom radu detaljno su objašnjene i implementirane ključne funkcionalnosti web aplikacije, uključujući registraciju i prijavu korisnika, pretragu i objavu oglasa za posao, generiranje ugovora te upravljanje plaćanjima putem Stripea. Osim toga, rad detaljno prikazuje strukturalne i funkcionalne značajke sustava. Navedene funkcionalnosti su razvijene kako bi pružile cjelovitu podršku korisnicima u upravljanju poslovnim procesima unutar platforme, omogućujući im da na jednostavan način obavljaju poslovne aktivnosti u digitalnom okruženju. Važno je napomenuti da aplikacija rješava problem povezanosti naručitelja i izvođača u kontekstu digitalnog poslovanja, u kojem korisnici sve više preferiraju tehnološka rješenja umjesto tradicionalnih metoda komunikacije uživo. Time se olakšava interakcija između korisnika, što odgovara modernim potrebama tržišta.

Ključne riječi: web aplikacija, Vue.js, Quasar, Node.js, PostgreSQL, Stripe API.

Abstract

This thesis describes the development and implementation of a web application designed to connect clients and service providers, based on a modern three-tier architecture. The application is aimed at individuals seeking jobs and businesses or individuals looking for workers for temporary projects based on a Contract for Services. The application utilizes Vue.js and Quasar for the development of the presentation layer, providing an intuitive user interface, Node.js and Express.js for managing business logic in the application layer, and PostgreSQL and Sequelize ORM for efficient data management in the data layer. This thesis provides a detailed explanation and implementation of the key functionalities of the web application, including user registration and login, job posting and searching, contract generation, and payment management through Stripe. Additionally, the thesis thoroughly examines the structural and functional features of the system. Mentioned functionalities were developed to provide comprehensive support to users in managing business processes within the platform, enabling them to perform business activities in a digital environment with ease. It is important to note that the application addresses the problem of connecting clients and service providers in the context of digital business, where users increasingly prefer technological solutions over traditional methods of face-to-face communication. This facilitates interaction between users, aligning with the modern needs of the market.

Keywords: web application, Vue.js, Quasar, Node.js, PostgreSQL, Stripe API.

Sadržaj

1. Uvod

Tema ovog završnog rada je razvoj i implementacija web aplikacije koja povezuje naručitelje i izvođače poslova. Predmet rada obuhvaća cjelokupni proces dizajna, razvoja i implementacije aplikacije, uključujući odabir odgovarajućih tehnologija, te implementaciju ključnih funkcionalnosti unutar definirane arhitekture sustava.

Cilj ovog rada je izraditi funkcionalnu web aplikaciju koja će omogućiti jednostavno i efikasno povezivanje naručitelja s izvođačima za jednokratne poslove na temelju Ugovora o djelu. Aplikacija podržava dvije osnovne uloge korisnika: naručitelja i izvođača, pri čemu naručitelji mogu biti privatne osobe ili poslovni subjekti, odnosno tvrtke.

Za izvođače poslova, bez obzira na to jesu li privatne osobe ili poslovni subjekti, implementirane su funkcionalnosti koje omogućuju jednostavno upravljanje njihovim profilima, uključujući ažuriranje, pregled, brisanje i deaktivaciju profila. Izvođači također imaju pristup alatima za pretragu i filtriranje poslova, pregled obavljenih, prijavljenih i tekućih poslova, te mogućnost prijave na poslove koji odgovaraju njihovim vještinama.

Naručitelji, s druge strane, imaju mogućnost upravljanja svojim profilima na sličan način kao i izvođači. Ključne funkcionalnosti koje su razvijene za naručitelje uključuju pretragu i filtriranje poslova, pregled vlastitih objavljenih poslova, te kreiranje, ažuriranje, deaktivaciju i brisanje oglasa za posao. Ove funkcionalnosti omogućuju naručiteljima jednostavno upravljanje procesom zapošljavanja, olakšavajući im pronalazak idealnog kandidata, bilo da je riječ o privatnoj osobi ili poslovnom subjektu.

Pored toga, obje strane – i izvođači i naručitelji – mogu koristiti sustav za upravljanje uplatama i isplatama implementiran pomoću *Stripe API-ja*. Dodatno, razvijena je funkcionalnost za generiranje ugovora, što omogućuje izvođačima zaključenje poslovnog odnosa s naručiteljima.

Za razvoj navedenih funkcionalnosti korištene su sljedeće tehnologije: *Vue.js*, zajedno s njegovim razvojnim okvirom *Quasar*, za frontend; *Node.js* i *Express.js* za serversku stranu odnosno backend; te *PostgreSQL* u kombinaciji sa *Sequelize ORM-om* za upravljanje pohranom podataka u bazi podataka. Razlog odabira ovih tehnologija objašnjen je u sljedećem poglavlju ovog rada.

U konačnici, cilj je razviti rješenje koje će biti prilagođeno potrebama tržišta, te koje će korisnicima olakšati poslovnu interakciju u digitalnom okruženju.

2. Opis korištenih tehnologija

U ovome završnom radu korištene su moderne tehnologije koje su omogućile razvoj ove web aplikacije temeljene na troslojnoj arhitekturi (eng. *Three-Tier Architecture*). Troslojna arhitektura, koja se sastoji od prezentacijskog sloja (frontend), aplikacijskog sloja (backend) i sloja podataka (baza podataka), osigurava jasnu separaciju odgovornosti, lakše održavanje i jasniju komunikaciju između navedenih slojeva.

2.1. Prezentacijski sloj

Prezentacijski sloj igra ključnu ulogu u interakciji korisnika s web aplikacijom, a njegova osnovna svrha je prikazivanje informacija korisniku i prikupljanje podataka koje korisnik unosi. Za razvoj korisničkog sučelja odabran je *Vue.js*, razvojni okvir izgrađen na temeljima *HTML-a*, *CSS-a* i *JavaScript-a* [1]. U kontekstu ove web aplikacije, *Vue.js* je poslužio kao čvrsta osnova, dok je naglasak stavljen na korištenje njegovog razvojnog okvira, *Quasara*. *Quasar* je korišten radi njegove mogućnosti izrade responzivnih i brzih stranica, što se činilo idealnim izborom za razvoj korisničkog sučelja ove aplikacije [2].

2.2. Aplikacijski sloj

Aplikacijski sloj obuhvaća poslovnu logiku aplikacije. U ovoj aplikaciji, backend je izgrađen koristeći Node.js u kombinaciji s *Express.js*. *Node.js* je otvoreno okruženje koje omogućuje izvršavanje JavaScript koda na serverskoj strani, odnosno poslužitelju [3]. *Express.js* je minimalistički web razvojni okvir za *Node.js* koji olakšava razvoj *RESTful API-ja*, upravljanje HTTP zahtjevima te obradu poslovne logike aplikacije [4].

Važan dio aplikacijskog sloja je i integracija sa *Stripe API-jem* [5], koji je korišten za upravljanje plaćanjima unutar sustava. *Stripe* osigurava sigurno procesiranje uplata te upravljanje transakcijama između naručitelja i izvođača poslova.

2.3. Sloj podataka

Sloj podataka, također poznat kao backend sloj, važan je dio troslojne arhitekture koji je zadužen za pohranu, upravljanje i obradu informacija unutar aplikacije. Ovaj sloj koristi sustav za upravljanje bazom podataka kako bi osigurao učinkovito rukovanje podacima.

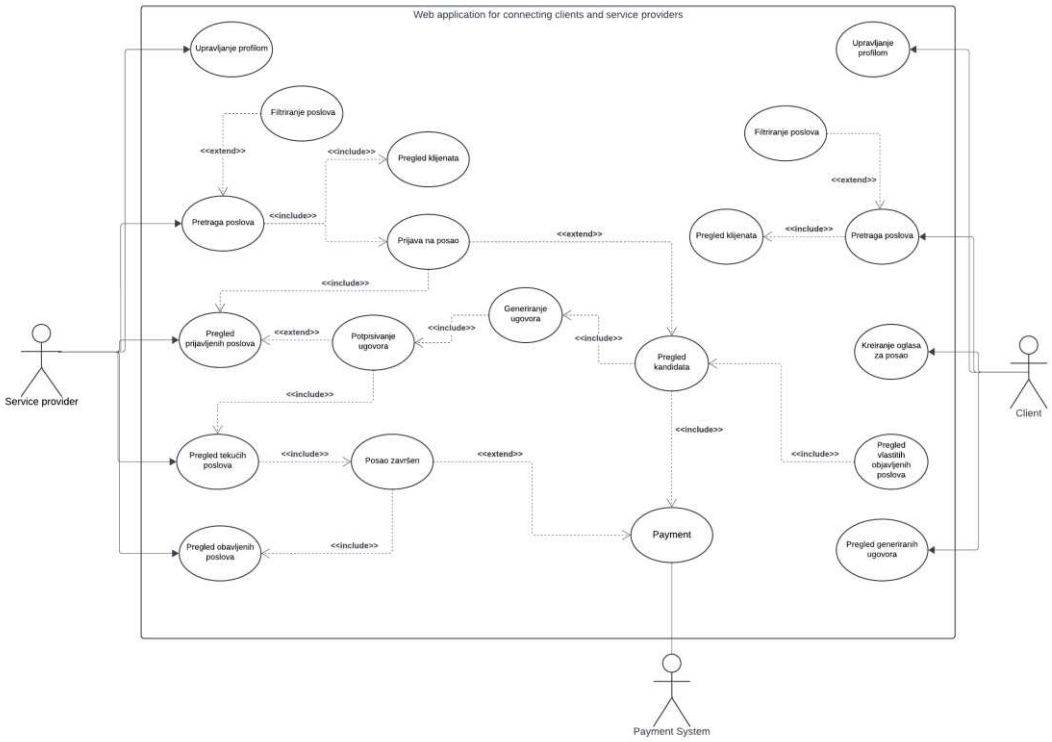
U ovom projektu korišten je *PostgreSQL* kao relacijski sustav za upravljanje bazom podataka [5], dok je *Sequelize ORM* (eng. *Object-Relational Mapping*), popularna

biblioteka za Node.js, korišten za pojednostavljenje rada s bazom podataka [6]. *Sequelize* omogućuje mapiranje složenih *SQL* upita u jednostavne *JavaScript* objekte, što značajno olakšava programiranje i manipulaciju podacima. Dodatno, za izradu ove aplikacije korišten je i alat za upravljanje bazom podataka – *pgAdmin* [7].

3. Značajke implementiranog sustava

U ovome poglavlju prikazani su dijagrami koji ilustriraju ključne aspekte sustava. Dijagram klasa (eng. *Class diagram*) prikazuje kako je aplikacija strukturirana, s naglaskom na objekte, njihove atribute, metode i međusobne odnose, dok dijagram slučaja upotrebe (eng. *Use Case diagram*) pokazuje kako korisnici komuniciraju sa sustavom i koje funkcionalnosti su im na raspolaganju.

3.1. Funkcionalne značajke sustava



SLIKA 1. UML DIJAGRAM SLUČAJA UPOTREBE

Dijagram slučaja upotrebe, prikazan na slici 1, opisuje ključne funkcionalnosti sustava za povezivanje naručitelja i izvođača posla. U dijagramu su identificirana dva glavna aktera: naručitelj (eng. *Client*) i izvođač poslova (eng. *Service provider*), te sustav plaćanja kao treći entitet.

Naručitelj ima sljedeće mogućnosti: pretraživanje i filtriranje poslova, upravljanje profilom, kreiranje oglasa za posao, pregled vlastitih objavljenih poslova, pregled generiranih ugovora i pregled profila izvođača poslova.

Izvođač poslova ima sljedeće mogućnosti: pregled profila izvođača posla, upravljanje profilom, pregled prijavljenih poslova, pregled tekućih psolova, pregled obavljenih poslova, pregled profila naručitelja te prijava na oglas za posao.

Proces započinje kada naručitelj kreira oglas za posao, čime omogućuje izvođaču usluga pretraživanje dostupnih poslova i prijavu na onaj koji mu najviše odgovara. Kada se izvođač prijavi na oglas, njegovi osnovni podaci postaju vidljivi naručitelju unutar profila kandidata, što omogućuje naručitelju da pregleda prijavljene kandidate i odabere onog koji najbolje odgovara potrebama posla. Nakon prijave, naručitelj može generirati ugovor, a izvođač može pristupiti procesu potpisivanja ugovora. Nakon što obje strane potpišu ugovor, izvođač može započeti s radom. Kartica posla za koji je ugovor sklopljen premješta se u karticu za pregled tekućih poslova, gdje izvođač može obavijestiti naručitelja da je posao završen. Zatim se kartica posla prebacuje na stranicu za pregled obavljenih poslova, dok se naručitelju pruža mogućnost plaćanja završenog posla.

Pod upravljanjem profilom za oba aktera podrazumijevaju se funkcionalnosti poput ažuriranja, brisanja, deaktivacije i prikaza profila. Ove funkcionalnosti omogućuju jednostavno i učinkovito korištenje sustava, prilagođeno specifičnim potrebama korisnika.

3.2. Strukturne značajke sustava

SLIKA 2. KLASNI DIJAGRAM

Klasni dijagram, prikazan na slici 2, opisuje strukuturu i odnose između klasa unutar sustava. Klasni dijagram omogućuje bolje razumijevanje kako se različite komponente sustava koriste u implementaciji funckionalnosti prikazanih u dijagramu slučaja upotrebe.

Svaka klasa u dijagramu predstavlja određeni entitet, u ovom slučaju radi se o sljedećim klasama:

1. Naručitelj – predstavlja osobu ili organizaciju koja koristi aplikaciju za pronalazak izvođača poslova. Klasa pohranjuje podatke kao što su ime, prezime, naziv tvrtke, kao i druge relevantne informacije. Pojedini atributi, poput naziva tvrtke, popunjavaju se ovisno o tipu naručitelja – individualni (eng. *individual*) ili poslovni (eng. *business*) tip.
2. Oglas za posao (eng. *Job advertisement*) – predstavlja oglas koji naručitelj kreira u sustavu kako bi pronašao izvođača poslova za određeni posao. Klasa pohranjuje sve relevantne informacije o poslu, poput naziva posla, kategorije, potrebne opreme, tražene kvalifikacije i mnoge druge.

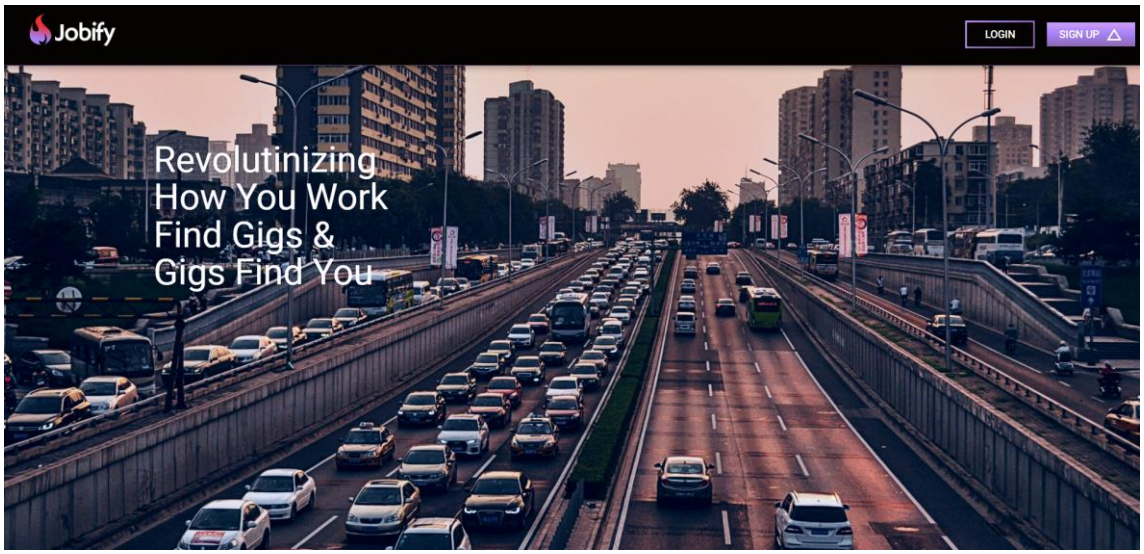
3. Slobodna radna mjesta (eng. *Job vacancies*) – predstavlja radna mjesta dostupna unutar oglasa za posao. Klasa pohranjuje informacije o statusu posla, statusu prijave, datuma prijave kao i *Stripe Account id-a* izvođača posla, koji je potreban za isplatu nakon obavljenog posla. Uz navedene, klasa sadrži i niz drugih važnih atributa.
4. Izvođač poslova – predstavlja osobu koja koristi aplikaciju za pronalazak posla. Klasa pohranjuje podatke kao što su ime, prezime, kontakt informacije i kompetencije izvođača. Ovi atributi omogućuju naručitelju da procijeni prikladnost izvođača posla za određeni posao.
5. Radno iskustvo (eng. *Work Experience*) – sadrži informacije o prijašnjim radnim iskustvima izvođača. Klasa pohranjuje podatke poput naziva tvrtke, pozicije i trajanja zaposlenja. Ovi atributi pomažu naručiteljima da procjene relevantnost radnog iskustva izvođača poslova za određeni posao.
6. Obrazovanje (eng. *Education*) – sadrži informacije o obrazovanju izvođača poslova. Klasa pohranjuje podatke poput naziva obrazovne institucije, stečenog stupnja i trajanja obrazovanja.
7. Jezici (eng. *Languages*) – pohranjuje informacije o jezicima koje izvođač poslova poznaje. Klasa uključuje atribute poput naziva jezika i razine znanja.
8. Ugovor o poslu (eng. *Job contract*) – predstavlja Ugovor o djelu sklopljen između naručitelja i izvođača poslova. Ova klasa pohranjuje nekoliko ključnih atributa, uključujući status ugovora, koji prati trenutnu fazu realizacije. Atribut contract pohranjuje ugovor u binarnom formatu, omogućujući spremanje digitalne verzije dokumenta, poput PDF-a. Klasa također sadrži digitalne potpise naručitelja i izvođača određenog posla, također pohranjene u binarnom formatu, čime se potvrđuje valjanost i pristanak na uvjete ugovora. Dodatno, atributi priceId i productId pohranjuju identifikatore povezane s integracijom ugovora s vanjskim sustavom naplate - *Stripea*.
9. Plaćanje po ugovoru (eng. *Contract payment*) – pohranjuje podatke o plaćanjima koje se izvršavaju prema ugovorima sklopljenih naručitelja i izvođača posla. Jedan od najvažnijih atributa je amount, koji bilježi iznos plaćanja u decimalnom formatu, osiguravajući preciznost u izračunu i prikazu vrijednosti transakcije. Uz to, atribut invoiceId pohranjuje jedinstveni identifikator fakture povezane s plaćanjem, dok atribut invoicePdf omogućuje spremanje same fakture u PDF formatu kao binarni podatak, što omogućuje jednostavan pristup i preuzimanje fakture.

4. Prikaz funkcionalnosti sustava

U ovom dijelu završnog rada detaljno su prikazani izgled i funkcionalnosti korisničkog sučelja aplikacije. Prikazivanje korisničkog sučelja ključno je kako bi se jasno razumjela struktura aplikacije. Ovaj pregled omogućit će bolji uvid u cjelokupni rad aplikacije, te pripremiti osnovu za tehničku analizu koda u sljedećim poglavljima.

4.1. Početna stranica

Na početnoj stranici aplikacije, korisnici imaju mogućnost odabrati između dvije osnovne opcije postavljene u gornjem desnom kutu stranice: Login (prijava) ili Sign Up (registracija). Prikaz stranice vidljiv je na slici 3.

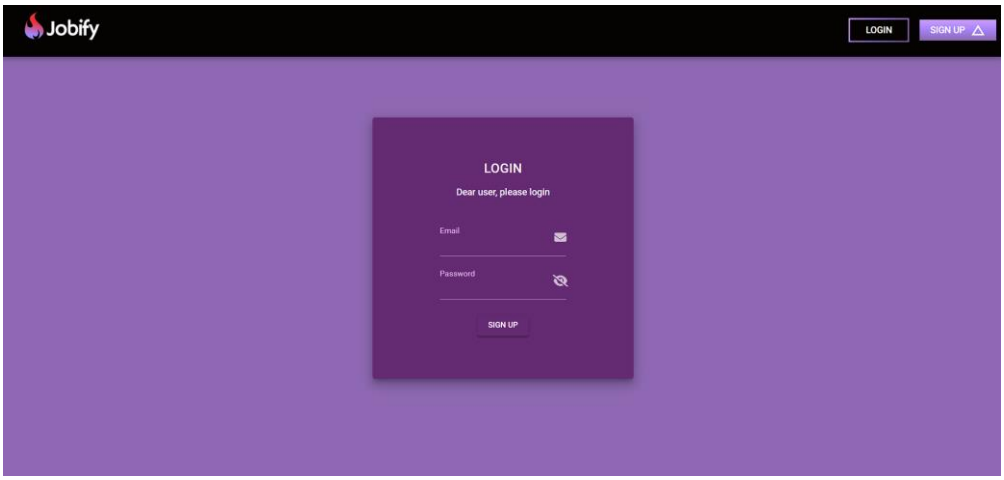


SLIKA 3. PRIKAZ POČETNE STRANICE

4.2. Proces prijave korisnika

Klikom na gumb Login za prijavu u sustav, korisnik je preusmjeren na stranicu koja je prikazana na slici 4. Ova stranica omogućuje unos podataka već registriranih korisnika, uključujući korisničko ime i lozinku. Nakon što korisnik unese svoje podatke, sustav provjerava te podatke (credentials) u bazi podataka. Ako su uneseni podaci ispravni, korisnik će biti uspješno prijavljen u sustav. U slučaju da su uneseni podaci netočni,

korisniku će biti prikazana poruka o grešci, koja ga obavještava da su uneseni podaci pogrešni i omogućuje mu se ponovni pokušaj prijave.



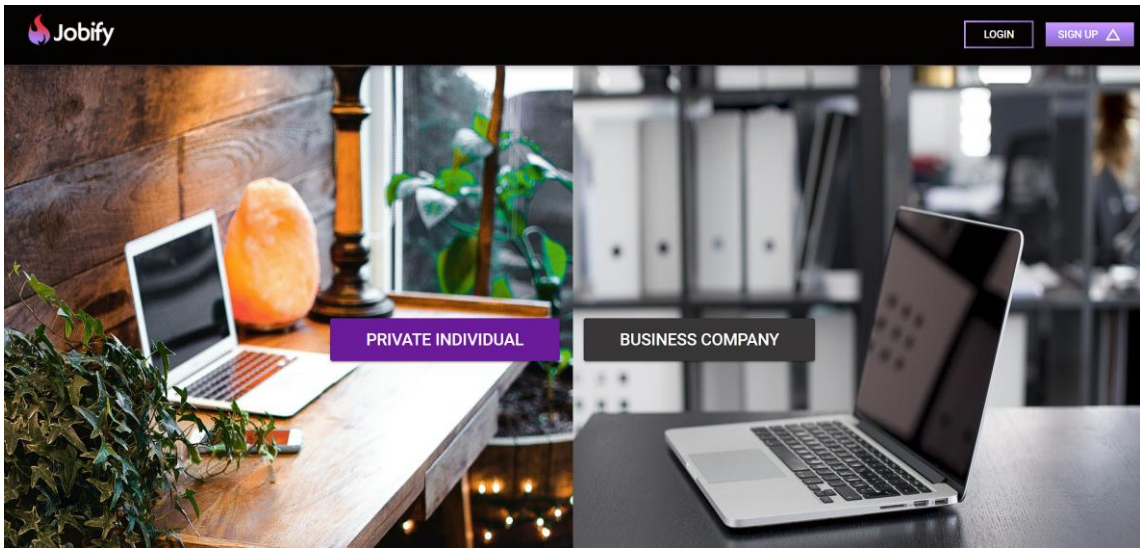
SLIKA 4. PRIKAZ FORME ZA PRIJAVU

4.3. Proces registracije korisnika

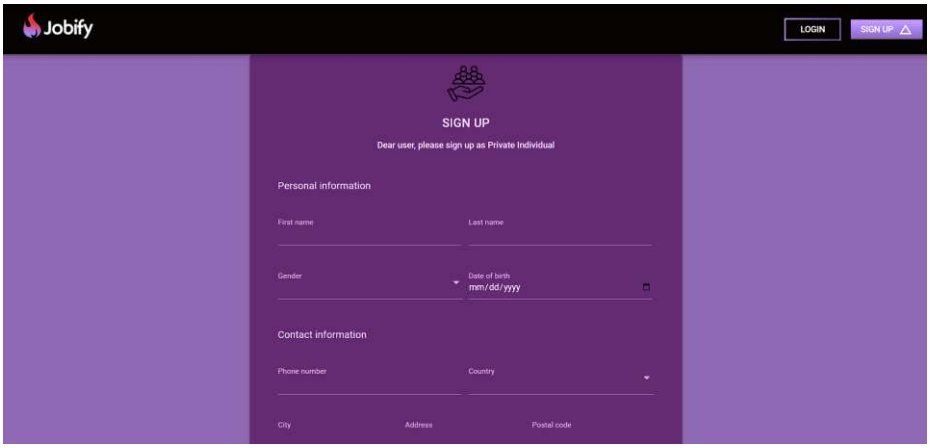
Kada korisnik klikne na gumb Sign up za registraciju u sustav, sustav mu nudi dvije osnovne opcije, koje ovise o njegovoj ulozi u aplikaciji: Client (naručitelj) ili Service Provider (izvođač poslova). Ovaj početni korak osigurava da svaki korisnik bude pravilno usmjeren prema odgovarajućoj registracijskoj formi, prilagođenoj njihovim specifičnim potrebama.

Ako korisnik odabere opciju Client (naručitelj), sustav pruža dodatni izbor između dvije vrste naručitelja, kako je prikazano na slici 5:

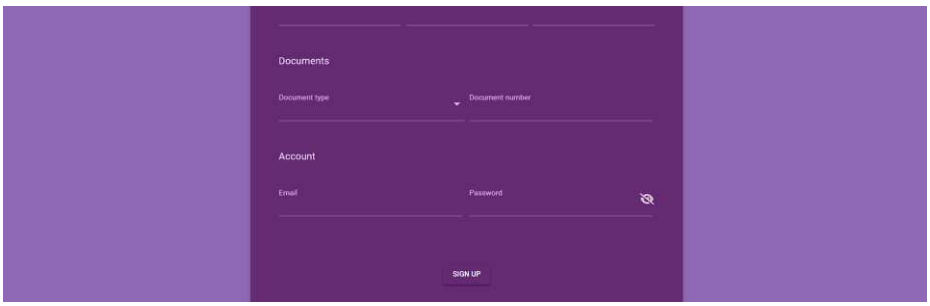
- Privatna osoba (eng. *Private individual*): Registracija, prikazana na slikama 6 i 7, prilagođena je pojedincima koji žele koristiti usluge kao fizičke osobe. Forma za privatne osobe prikuplja osnovne osobne podatke, kao što su ime, prezime, adresa, kontakt informacije i ostale relevantne informacije.
- Poslovni subjekt (eng. *Business company*): Ova opcija namijenjena je tvrtkama koje žele koristiti usluge u poslovne svrhe. Registracijska forma za poslovne subjekte uključuje dodatne informacije specifične za poslovanje, kao što su naziv tvrtke, adresa sjedišta i druge poslovne podatke. Registracijska forma namijenjena ovom tipu naručitelja prikazana je na slikama 8 i 9.



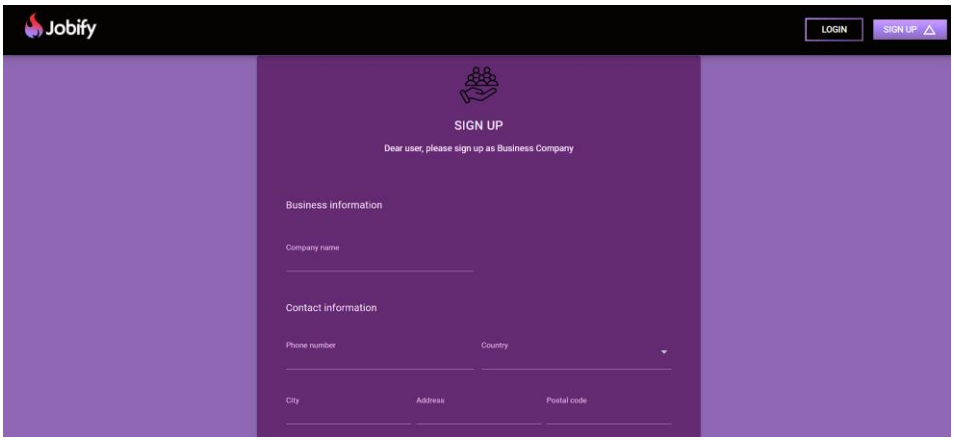
SLIKA 5. PRIKAZ ODABIRA TIPA NARUČITELJA



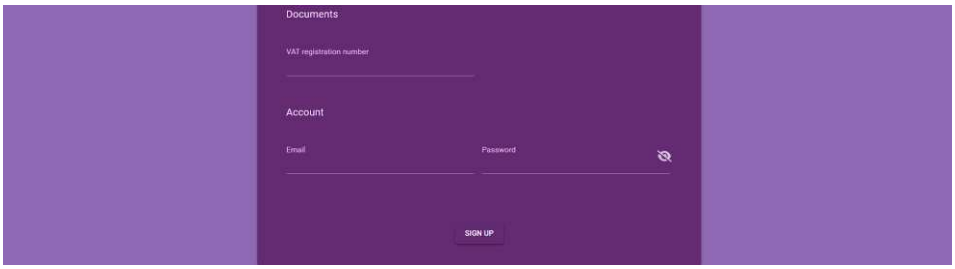
SLIKA 6. PRVI DIO REGISTRACIJE NAMIJENJEN PRIVATNOJ OSOBI



SLIKA 7. DRUGI DIO REGISTRACIJE NAMIJENJEN PRIVATNOJ OSOBI

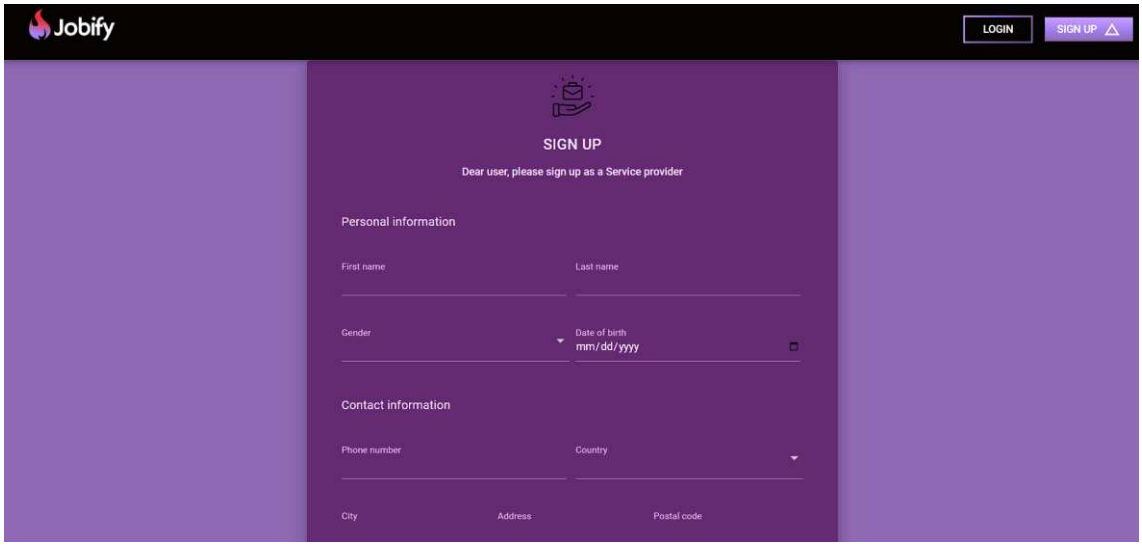


SLIKA 8. PRVI DIO REGISTRACIJE NAMIJENJEN POSLOVNOM SUBJEKTU



SLIKA 9. DRUGI DIO REGISTRACIJE NAMIJENJEN POSLOVNOM SUBJEKTU

Ukoliko korisnik u padajućem izborniku odabere opciju Service Provider preusmjeren je na registraciju prikazanu na slikama 10 i 11.



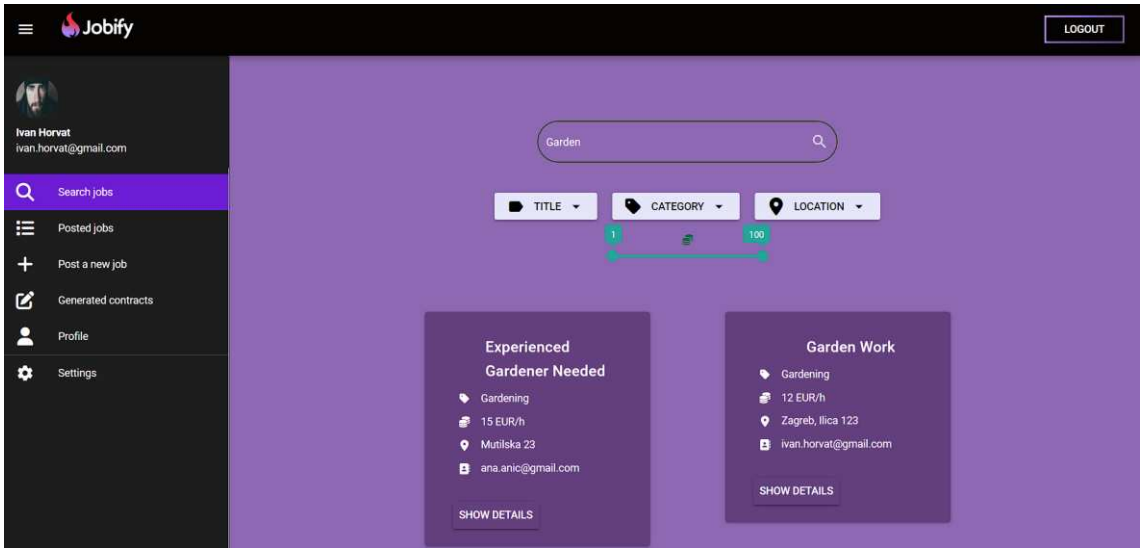
SLIKA 10. PRVI DIO PRIKAZA REGISTRACIJE NAMIJENJEN IZVOĐAČU

SLIKA 11. DRUGI DIO PRIKAZA REGISTRACIJE NAMIJENJEN IZVOĐAČU

4.4. Pretraga i pregled poslova

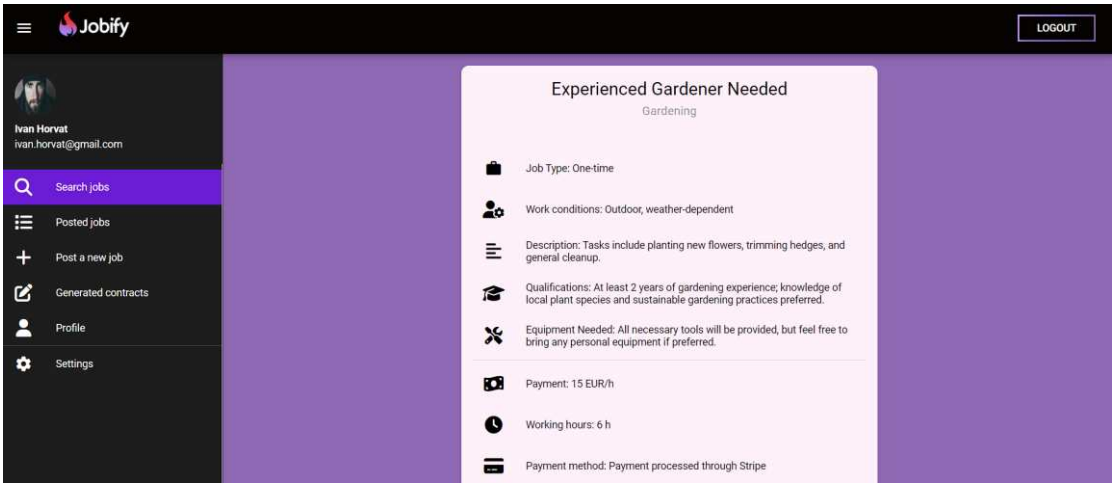
Stranica za pretragu poslova identična je za svakog korisnika aplikacije. Na slici 12 prikazana je stranica za naručitelja tipa privatna osoba. Oglasi za posao prikazani su kronološkim redoslijedom, od najnovijih ka starijim oglasima. Korisnik ima mogućnost dinamički filtrirati poslove koristeći specifične filtere kao što su naslov posla, kategorija, lokacija, i satnica. Na slici je prikazano klizanje satnice između 1 i 100, ovisno o valuti posla, što omogućuje korisniku specifično filtriranje poslova prema satnici koja im odgovara.

Pored filtriranja, platforma također nudi opciju pretrage poslova unosom ključnih riječi u polje za pretraživanje. Primjerice, unesena je riječ "Garden" u polje za pretraživanje, što rezultira prikazom poslova koji u naslovu ili kartici općenito sadrže ovu ključnu riječ. Ovo uključuje poslove kao što su "Experienced Gardener Needed" i "Garden Work".

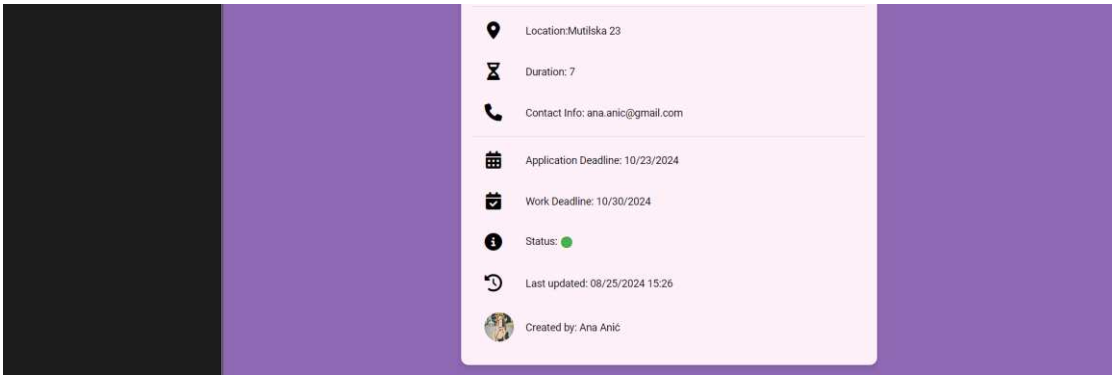


SLIKA 12. PRIKAZ STRANICE ZA PRETRAGU POSLOVA

Također, korisniku su prikazane kartice koje sadrže osnovne informacije o poslu kao što su naslov posla, kategorija, plaća, lokacija, i kontakt informacije. Svaka kartica također sadrži gumb "SHOW DETAILS" koji omogućuje pristup detaljnijim informacijama koje su prikazane na slikama 13 i 14.

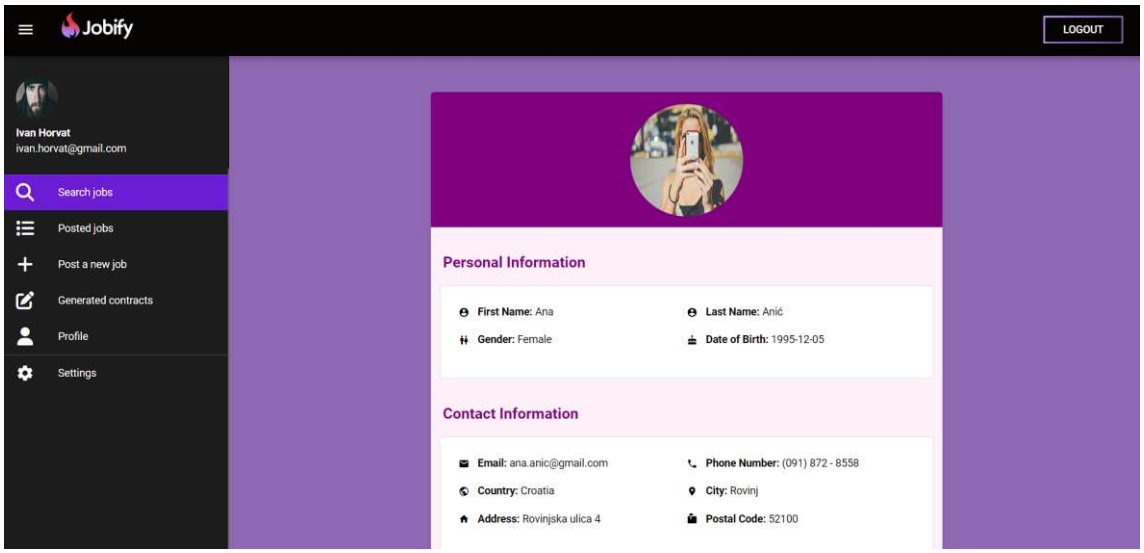


SLIKA 13. PRVI DIO PRIKAZA DETALJNIJIH INFROMACIJA O POSLU

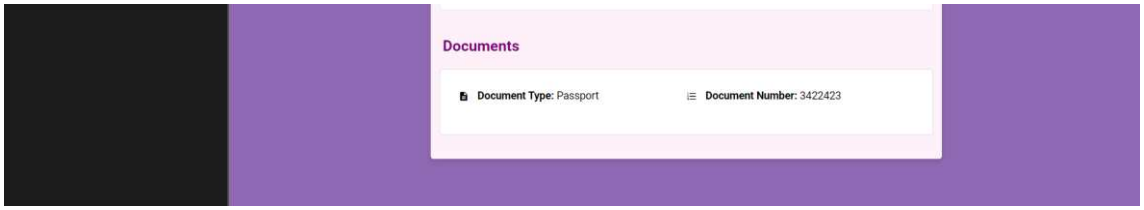


SLIKA 14. DRUGI DIO PRIKAZA DETALJNIJIH INFORMACIJA O POSLU

Korisnici, u ovom slučaju, naručitelji imaju mogućnost kliknuti na avatar bilo kojeg drugog naručitelja poslova kako bi pristupili njihovom profilu, što je prikazano na slikama 15 i 16. Ova funkcionalnost je korisna za izgradnju mreže kontakata**.** Naručiteljima je na taj način omogućeno uspostavljanje kontakata s drugima u svojoj industriji ili onima koji dijele slične interese.



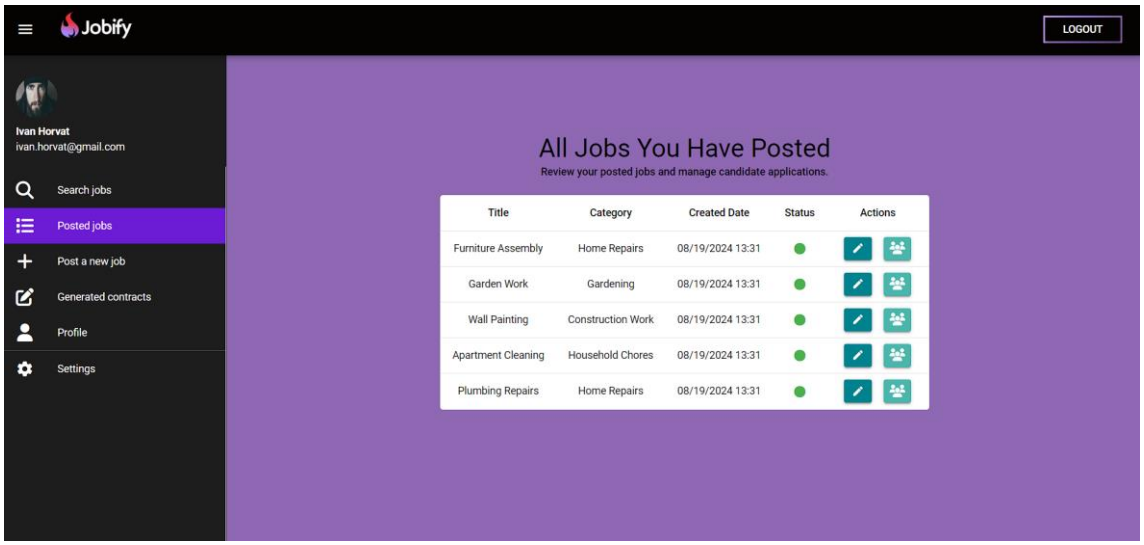
SLIKA 15. PRVI DIO PRIKAZA PROFILA NARUČITELJA



SLIKA 16. DRUGI DIO PRIKAZA PROFILA NARUČITELJA

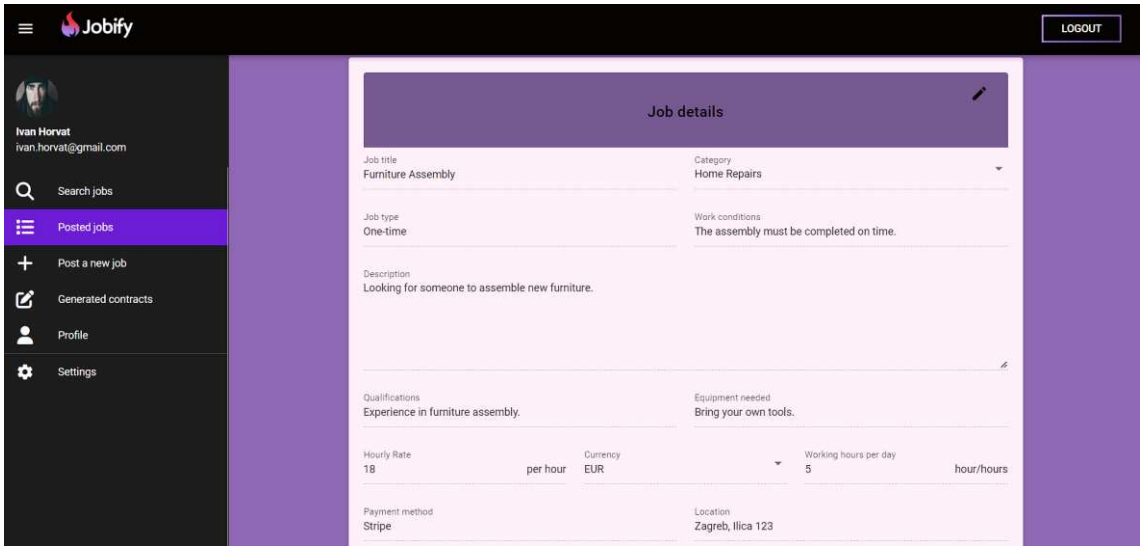
4.5. Upravljanje objavljenim poslovima za naručitelje

Na stranici Posted Jobs, koja je ilustrirana na slici 17, naručitelji posla imaju priliku pregledati sve poslove koje su prethodno objavili. Ova funkcionalnost im omogućuje upravljanje i praćenje svojih objavljenih oglasa za posao.

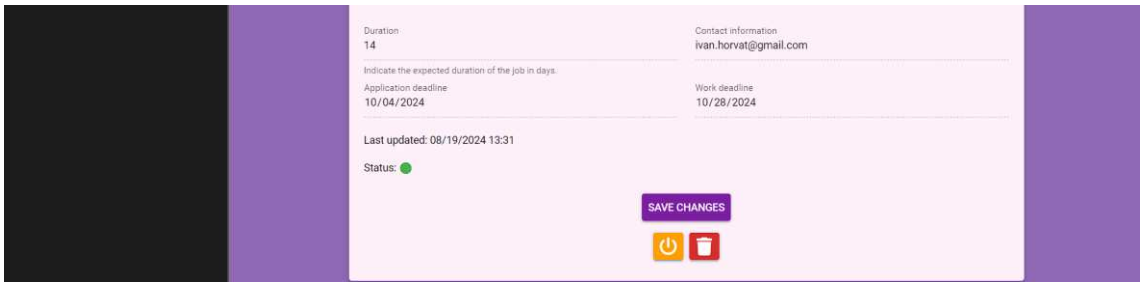


SLIKA 17. PRIKAZ POSLOVA KOJE JE NARUČITELJ OBJAVIO

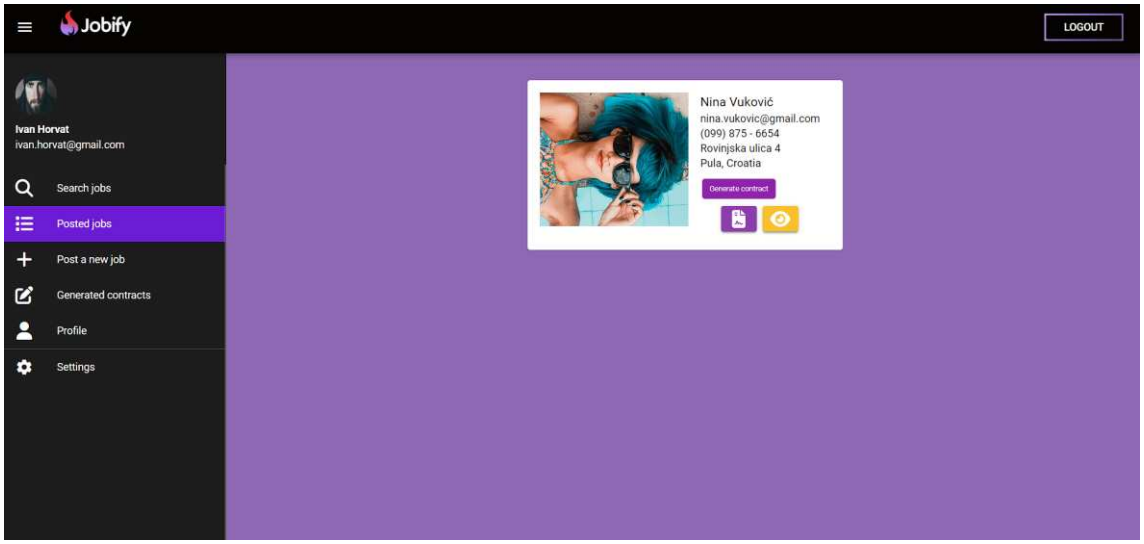
Tablica na ovoj stranici sadrži sve ključne informacije o poslovima uključujući naslov, kategoriju, datum objave, trenutni status, te akcijske gumbove koji omogućuju naručitelju da pristupi dodatnim opcijama. Gumb na kojem je prikazan ikona olovke vodi do stranice na kojoj naručitelj može pregledati detalje posla, ažurirati, izbrisati oglas ili ga deaktivirati. Ova stranica je prikazana na slikama 18 i 19. Drugi gumb omogućuje pregled kandidata koji su se prijavili na određeni posao, što naručiteljima pruža mogućnost da lako pregledaju i upravljaju prijavama izvođača poslova. Ova stranica je prikazana na slici 20.



SLIKA 18. PRVI DIO PRIKAZA STRANICE ZA UPRAVLJANJE OGLASOM

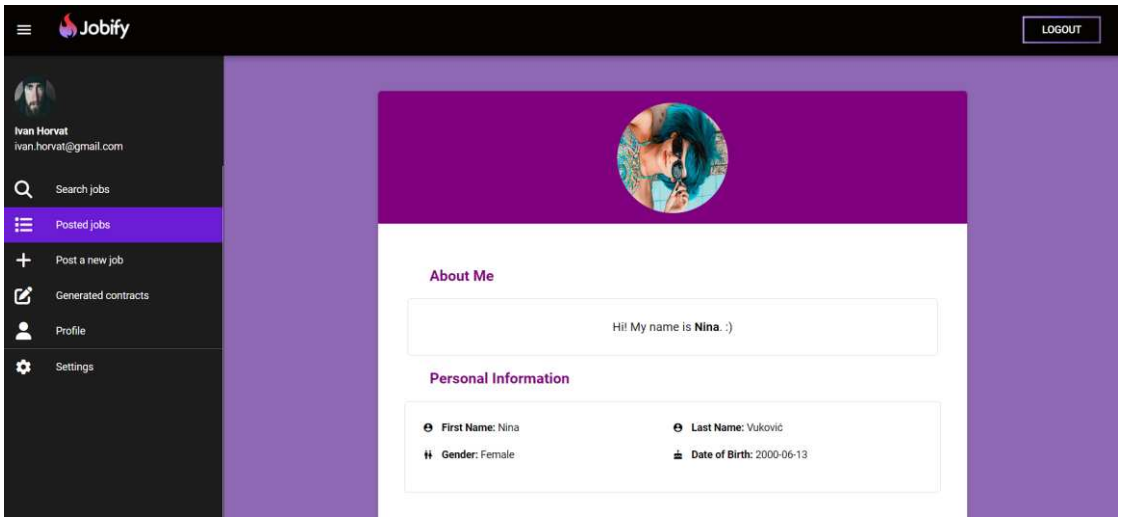


SLIKA 19. DRUGI DIO PRIKAZA STRANICE ZA UPRAVLJANJE OGLASOM

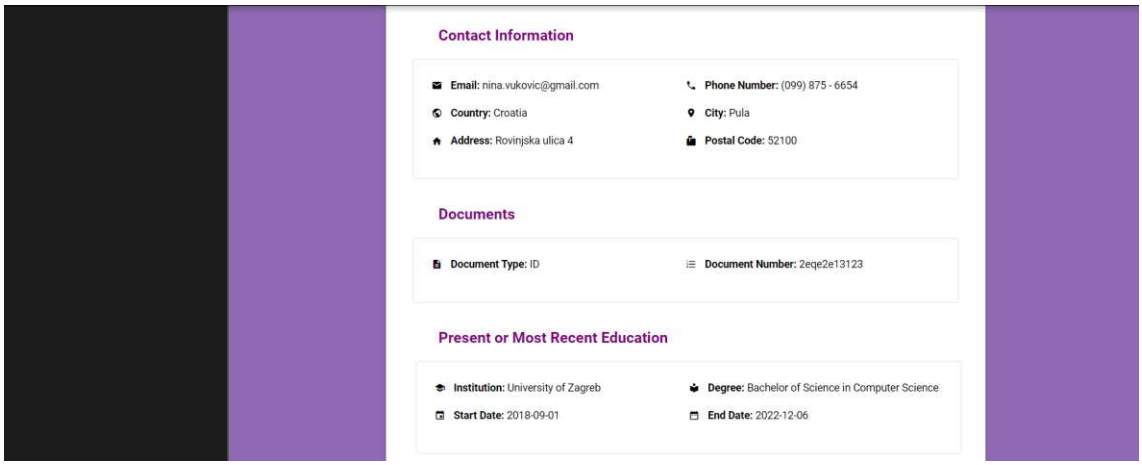


SLIKA 20. PRIKAZ PRIJAVLJENIH IZVOĐAČA

Klikom na žuti gumb s ikonom oka, naručitelju je omogućen pregled profila kandidata koji su se prijavili na njegov objavljeni oglas za posao. Ova funkcionalnost ključna je jer naručiteljima omogućuje detaljan uvid u kvalifikacije, iskustvo i vještine kandidata, što je presudno za donošenje informirane odluke o tome tko najbolje odgovara zahtjevima posla. Profil kandidata prikazan je na slikama 21, 22, 23 i 24.

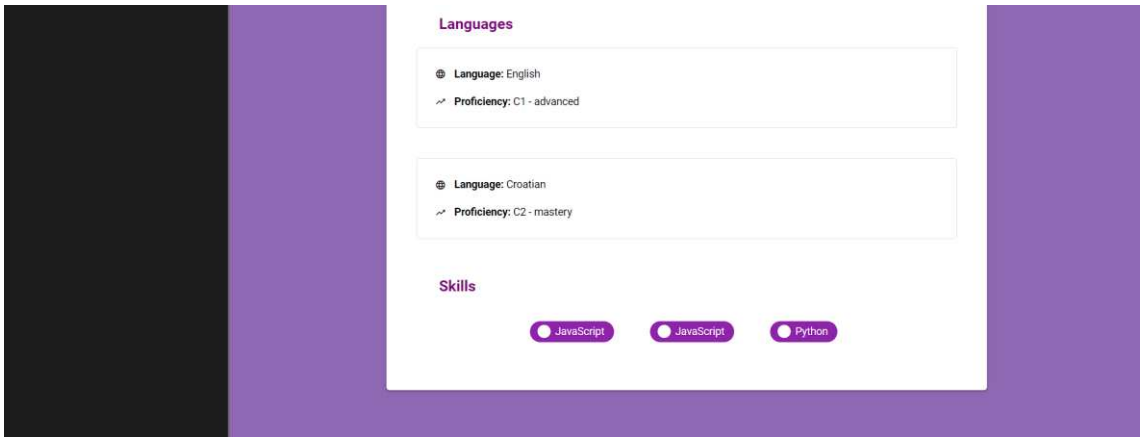


SLIKA 21. PRVI DIO PRIKAZA PROFILA KANDIDATA (IZVOĐAČA POSLOVA)



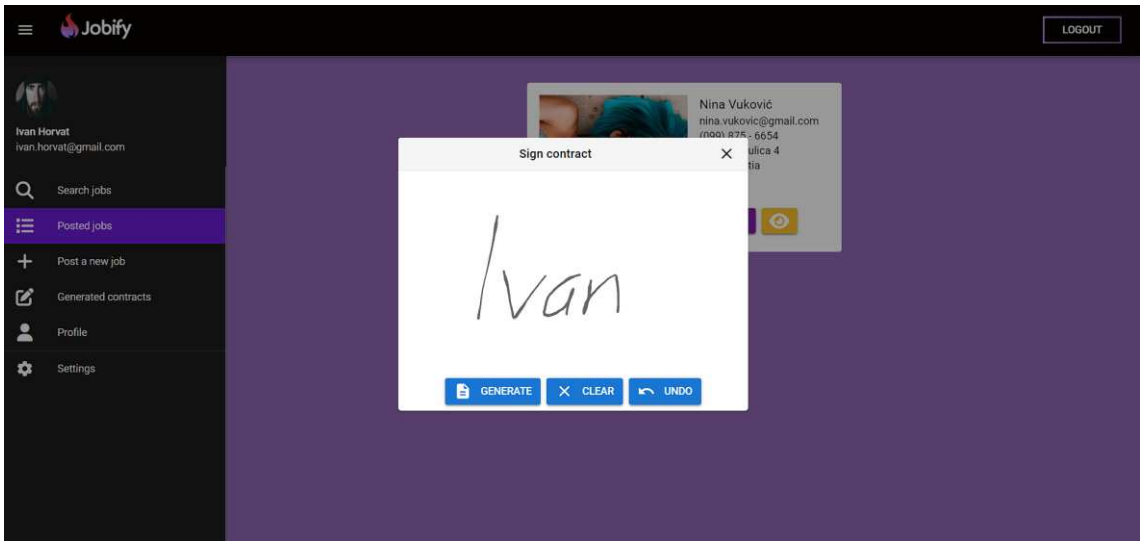
SLIKA 22. DRUGI DIO PRIKAZA PROFILA KANDIDATA (IZVOĐAČA POSLOVA)

SLIKA 23. TREĆI DIO PRIKAZA PROFILA KANDIDATA (IZVOĐAČA POSLOVA)

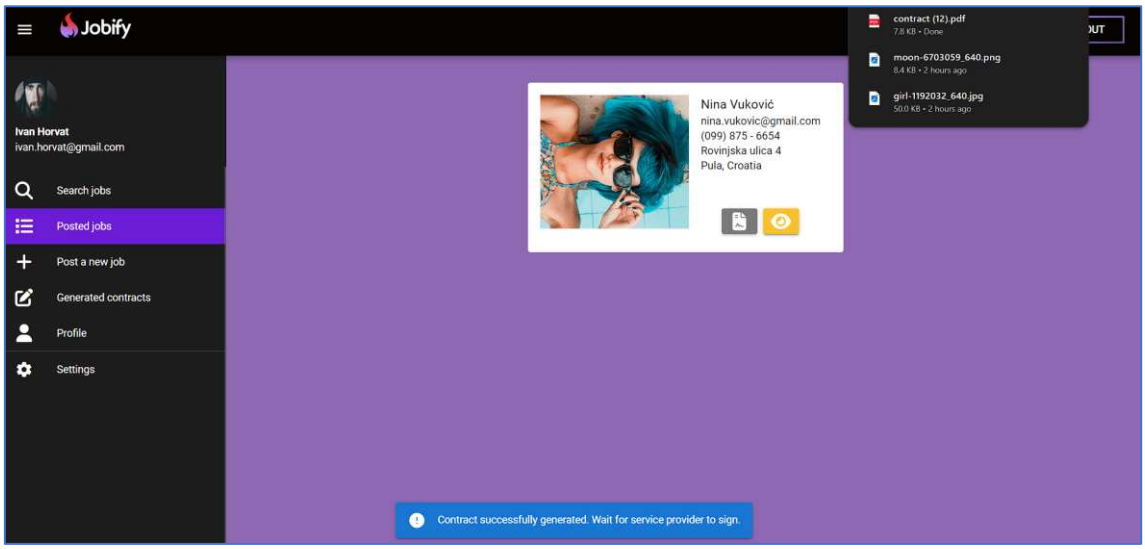


SLIKA 24. TREĆI DIO PRIKAZA PROFILA KANDIDATA (IZVOĐAČA POSLOVA)

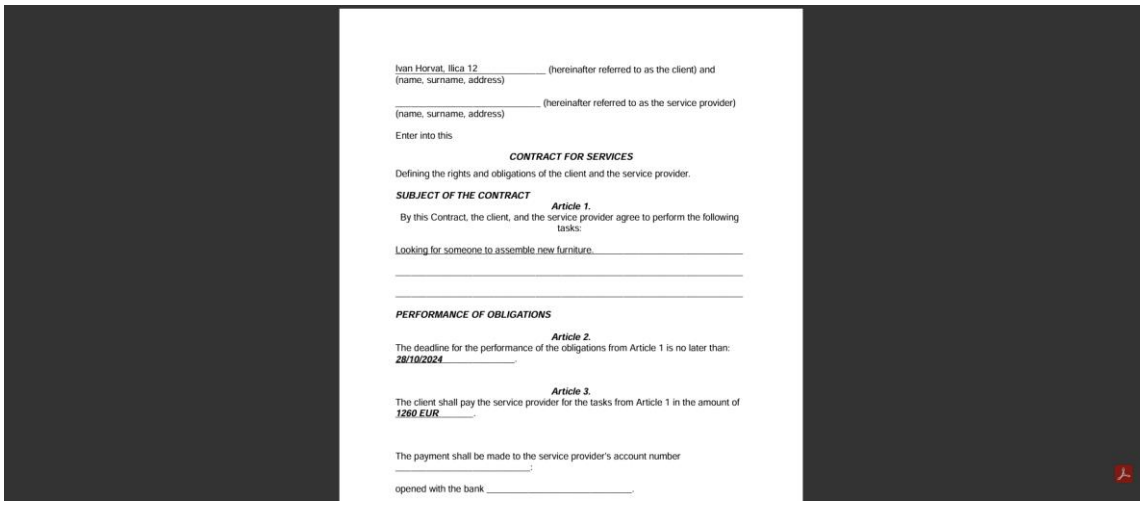
Stranica namijenjena za pregled kandidata, odnosno prijavljenih izvođača jedna je od ključnih stranica u aplikaciji. Na ovoj stranici naručitelj odabire kandidata na način da generira ugovor. Generiranje ugovora započinje pritiskom na gumb i tada se otvara digitalni potpisni blok, poznat kao Vue Signature [9]. Ova funkcionalnost omogućuje digitalno potpisivanje ugovora. Istovremeno, dio ugovora, koji se odnosi na naručitelja, ispunjava se automatski s njegovim podacima. Ugovor je napravljen pomoću bibilioteke PDFKit koja se koristi za generiranje PDF dokumenata na programatski način [10].



SLIKA 25. PRIKAZ POTPISIVANJA UGOVORA OD STRANE NARUČITELJA



SLIKA 26. PRIKAZ PREUZETOG UGOVORA



Ivan Horvat, Ilica 12 (hereinafter referred to as the client) and
(name, surname, address)

(hereinafter referred to as the service provider)
(name, surname, address)

Enter into this

CONTRACT FOR SERVICES

Defining the rights and obligations of the client and the service provider.

SUBJECT OF THE CONTRACT

Article 1.

By this Contract, the client, and the service provider agree to perform the following tasks:

Looking for someone to assemble new furniture.

PERFORMANCE OF OBLIGATIONS

Article 2.

The deadline for the performance of the obligations from Article 1 is no later than:
28/10/2024.

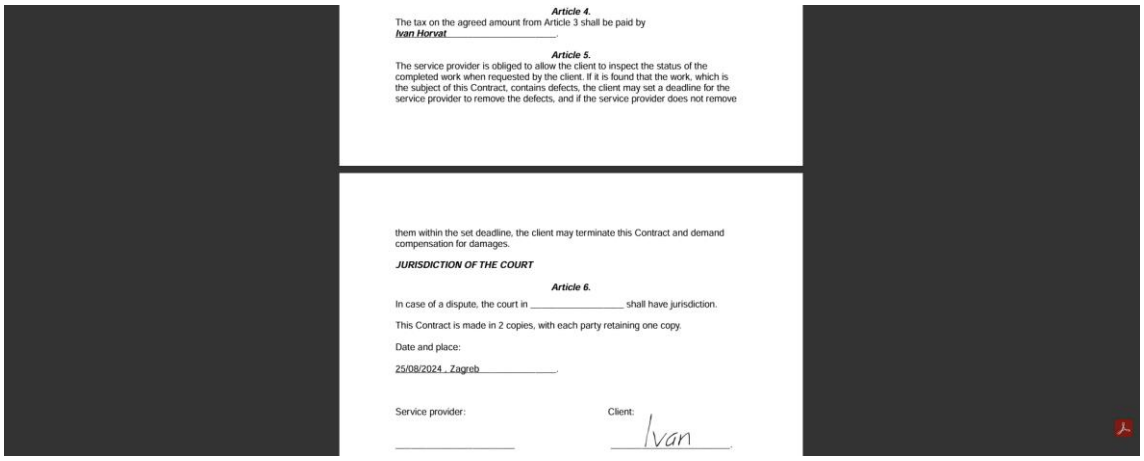
Article 3.

The client shall pay the service provider for the tasks from Article 1 in the amount of
1260 EUR.

The payment shall be made to the service provider's account number
;

opened with the bank .

SLIKA 27. PRIKAZ PRVOG DIJELA UGOVORA



Article 4.

The tax on the agreed amount from Article 3 shall be paid by
Ivan Horvat.

Article 5.

The service provider is obliged to allow the client to inspect the status of the completed work when requested by the client. If it is found that the work, which is the subject of this Contract, contains defects, the client may set a deadline for the service provider to remove the defects, and if the service provider does not remove them within the set deadline, the client may terminate this Contract and demand compensation for damages.

JURISDICTION OF THE COURT

Article 6.

In case of a dispute, the court in shall have jurisdiction.

This Contract is made in 2 copies, with each party retaining one copy.

Date and place:

25/08/2024 , Zagreb.

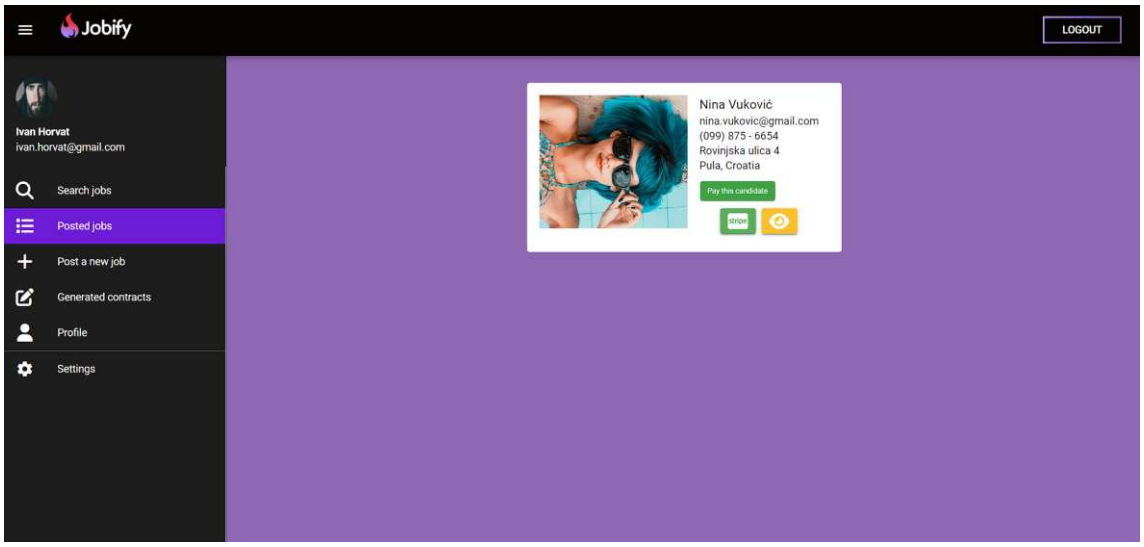
Service provider:

Client:

Ivan

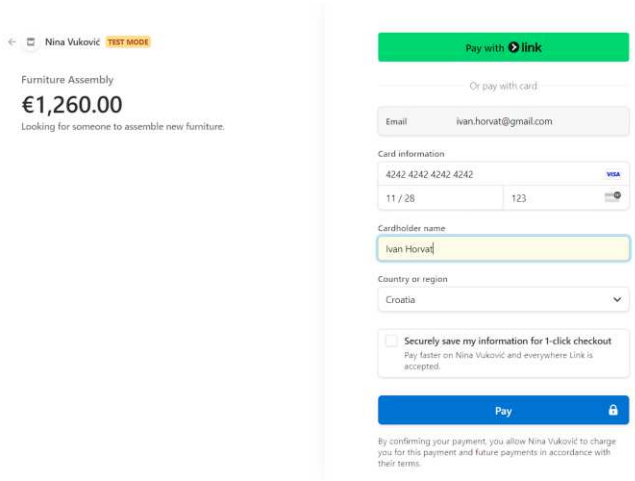
SLIKA 28. PRIKAZ DRUGOG DIJELA UGOVORA

Nakon što izvođač potpiše ugovor generiran od strane naručitelja, obavi posao te ga označi kao dovršen, naručitelju se otvara mogućnost plaćanja za obavljeni posao.

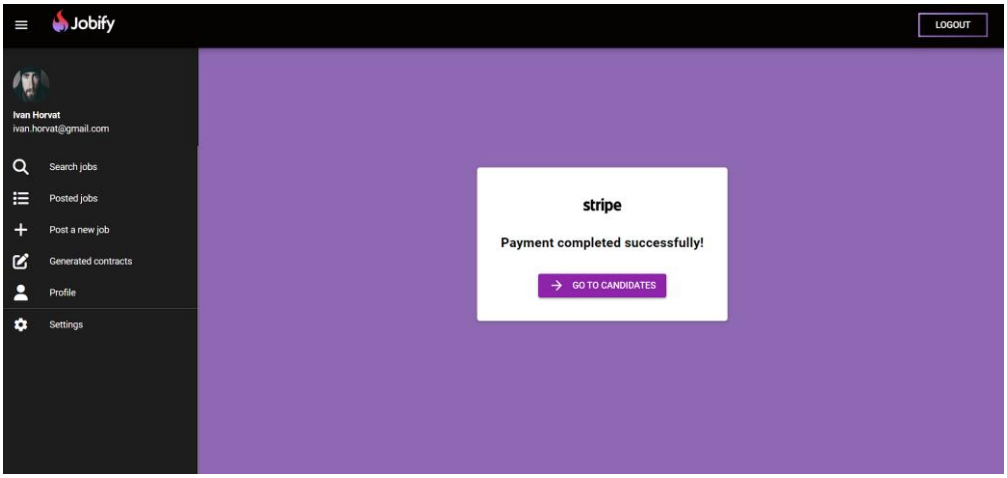


SLIKA 29. PRIKAZ MOGUĆNOSTI PLAĆANJA OBAVLJENOG POSLA

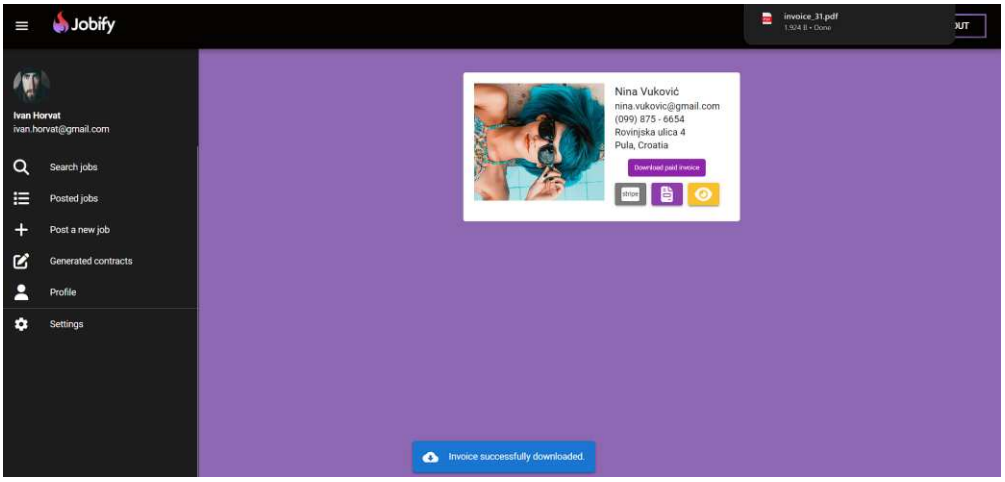
Klikom na gumb s ikonom *Stripe*, započinje proces kreiranja Checkout sesije, koja omogućuje korisniku sigurno i jednostavno izvršenje plaćanja za određenu uslugu ili proizvod, u ovom slučaju za obavljeni posao. Ova sesija automatski generira sve potrebne podatke za plaćanje, uključujući informacije o korisniku, cijeni usluge definiranoj ugovorom, te detalje o naručenom poslu, kako bi se osiguralo da transakcija prođe glatko i bez poteškoća. Nakon što je Checkout sesija uspješno kreirana, korisnik je preusmjeren na *Stripe-ovo* sučelje, gdje može unijeti svoje podatke o plaćanju i finalizirati transakciju [11]. Ova funkcionalnost omogućuje naručitelju da jednostavno i sigurno izvrši uplatu izvođaču, čime se završava proces plaćanja i izvršenja posla.



SLIKA 30. PRIKAZ CHECKOUT SESIJE



SLIKA 31. PRIKAZ PORUKE NAKON USPJEŠNE TRANSAKCIJE



SLIKA 32. PRIKAZ PREUZIMANJA FAKTURE

Invoice

Invoice number: 2CF97A11-0003
Date of issue: August 25, 2024
Date due: August 25, 2024

Nina Vukovi

Bill to
Ivan Horvat
ivan.horvat@gmail.com

1.260,00 € due August 25, 2024

Invoice for completed job - Furniture Assembly, job ID: 31

Description	Qty	Unit price	Amount
Furniture Assembly	1	1.260,00 €	1.260,00 €

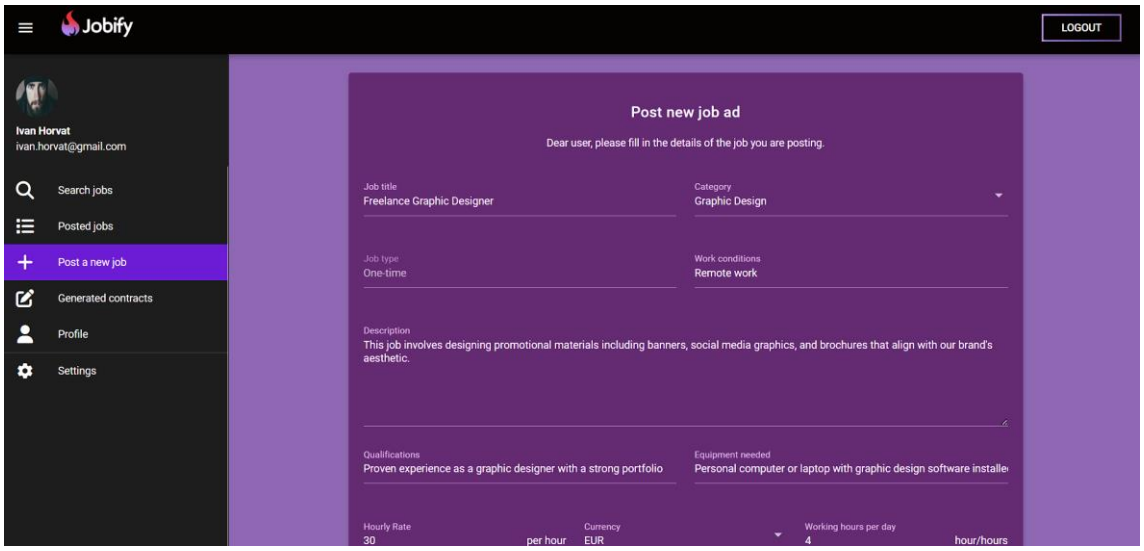
Subtotal	1.260,00 €
Total	1.260,00 €
Amount due	**1.260,00 €**

SLIKA 33. PRIKAZ PREUZETE FAKTURE

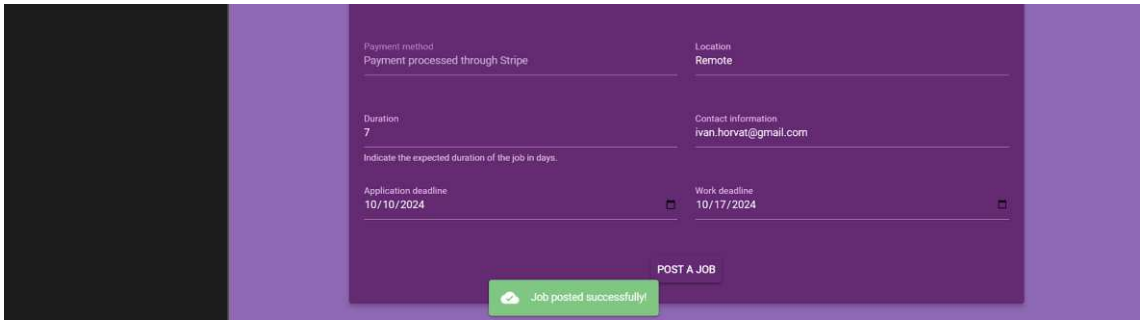
U kasnijem dijelu rada bit će detaljno prikazan i izvođačev dio aplikacije, uključujući proces kojim je izvođač potpisao ugovor te označio posao kao dovršen. Ovaj pregled će pružiti cjelovito razumijevanje kako su ovi koraci implementirani u aplikaciji, i kako se svi elementi međusobno povezuju.

4.6. Kreiranje oglasa za posao od strane naručitelja

Naručitelj mora popuniti sva potrebna polja kako bi kreirao oglas za posao. Nakon što oglas bude uspješno kreiran, u tablici na stranici "Posted Jobs" (slika 17) automatski se dodaje novi redak s podacima o tom oglasu. Oglas se zatim prikazuje pri vrhu stranice za pretraživanje poslova, omogućavajući korisnicima da ga brzo pronađu. Detaljan primjer procesa kreiranja oglasa za posao prikazan je na slikama 34 i 35.



SLIKA 34. PRVI DIO PRIMJERA KREIRANJA OGLASA ZA POSAO



SLIKA 35. DRUGI DIO PRIMJERA KREIRANJA OGLASA ZA POSAO

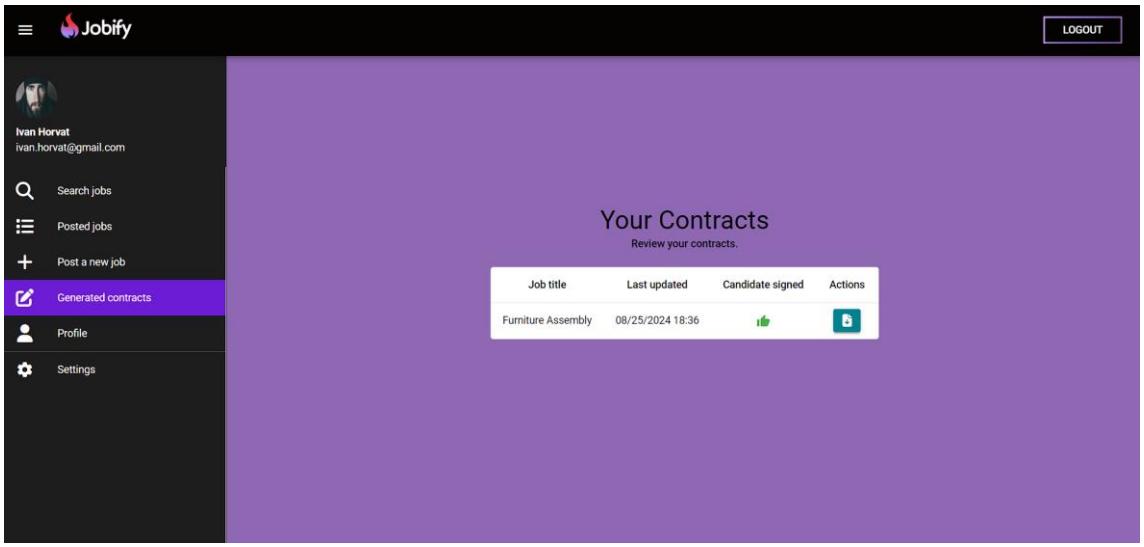
4.7. Generirani ugovori – naručitelj

U aplikaciji, naručitelj ima mogućnost preuzimanja ugovora koji je potpisan s izvođačem. Status potpisivanja ugovora jasno je prikazan kako bi naručitelj bio upoznat s trenutnim stanjem ugovora.

- Ako izvođač još nije potpisao ugovor**:** Status *Candidate Signed* bit će prikazan crvenom bojom u obliku ikone *dislike*. U ovoj fazi, naručitelj može preuzeti ugovor, ali će u preuzetom ugovoru biti ispunjen samo dio koji se odnosi na njega. Ugovor o djelu će sadržavati samo naručiteljeve podatke i njegov potpis, dok će dio namijenjen izvođaču ostati prazan dok on ne potpiše.
- Ako je izvođač potpisao ugovor**:** Status *Candidate Signed* promijenit će se u zelenu ikonu *like*, kao što je prikazano na slici 36. Kada naručitelj preuzme ugovor

u ovoj fazi, ugovor će sadržavati potpise obje strane, što znači da je ugovor validan i obostrano prihvaćen.

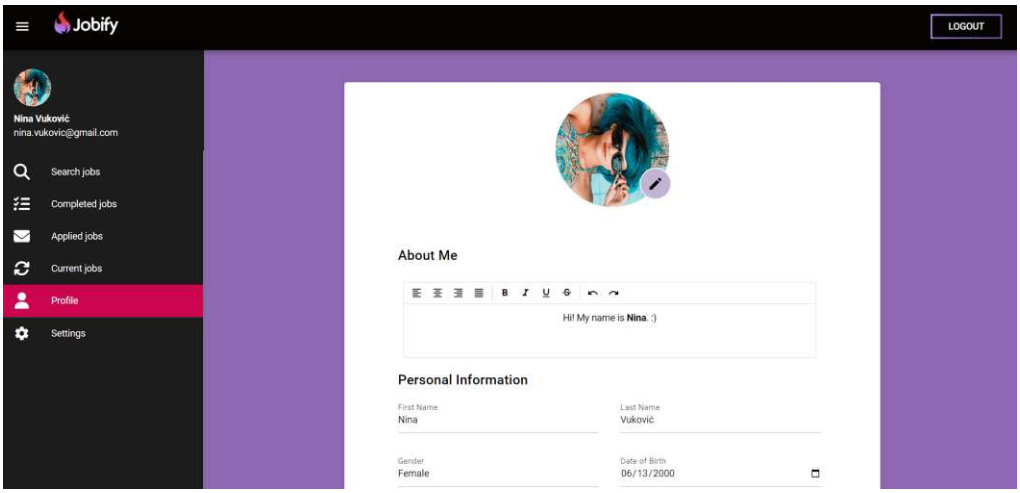
Ovaj sustav omogućuje naručitelju da jednostavno prati status ugovora i osigura da su svi potrebni koraci ispunjeni prije nego što se preuzeti ugovor smatra konačnim i obvezujućim za obje strane. Kada je ugovor potpisan, to također daje naručitelju do znanja da je izvođač započeo s radom, jer je time potvrđeno obostrano prihvaćanje uvjeta i početak realizacije dogovorenih zadataka.



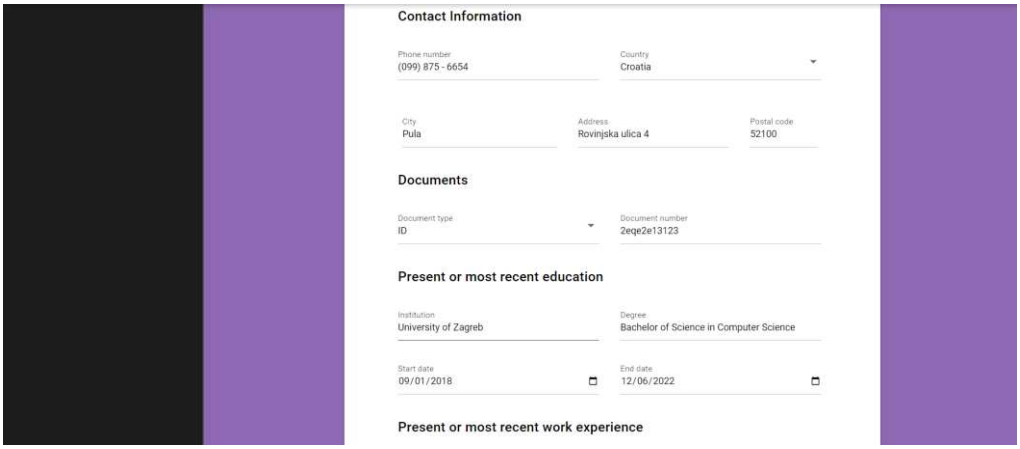
SLIKA 36. PRIKAZ GENERIRANIH UGOVORA

4.8. Upravljanje profilom izvođača

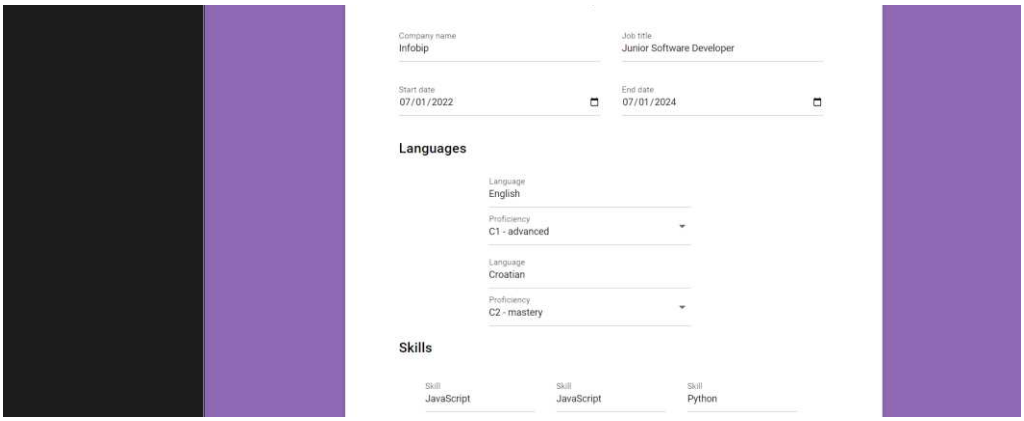
U ovom poglavlju bit će prikazan profil izvođača, uz napomenu da naručitelj ima sličan profil. Razlika je u tome što profil izvođača sadrži dodatna polja za unos informacija, uključujući "O meni", obrazovanje, radno iskustvo, jezike, vještine, i druge relevantne detalje. Važno je napomenuti da su nakon registracije ove sekcije inicijalno prazne. Preporučuje se da izvođači ispune ove sekcije kako bi naručitelji dobili više informacija o njima, što povećava šanse za dobivanje posla i poboljšanje profila unutar aplikacije. Također, izvođači mogu ažurirati svoj profil i prilagoditi ga ovisno o tome na koji se posao prijavljuju, čime mogu bolje istaknuti svoje kvalifikacije i relevantno iskustvo za određeni posao.



SLIKA 37. PRVI DIO PRIKAZA PROFILA IZVOĐAČA S MOGUĆNOŠĆU AŽURIRANJA INFORMACIJA



SLIKA 38. DRUGI DIO PRIKAZA PROFILA IZVOĐAČA S MOGUĆNOŠĆU AŽURIRANJA INFORMACIJA



SLIKA 39. TREĆI DIO PRIKAZA PROFILA IZVOĐAČA S MOGUĆNOŠĆU AŽURIRANJA INFORMACIJA

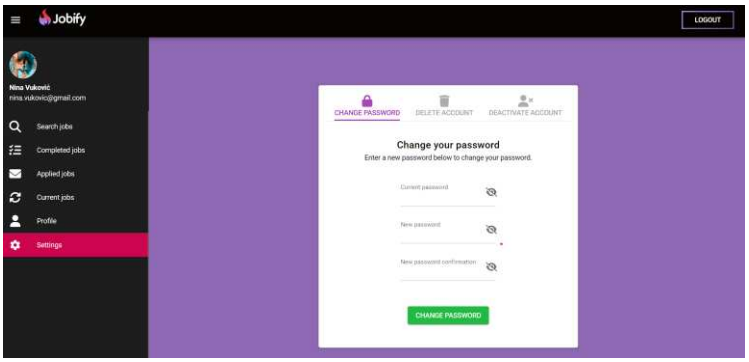
SLIKA 40. ČETVRTI DIO PRIKAZA PROFILA IZVOĐAČA S MOGUĆNOŠĆU AŽURIRANJA INFORMACIJA

4.9. Upravljanje postavkama računa izvođača

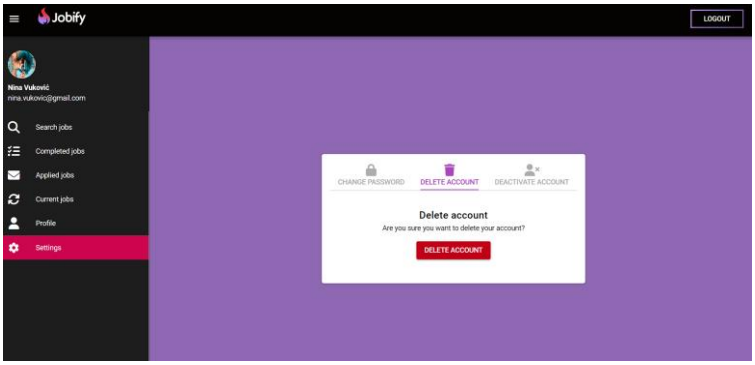
Postavke računa omogućuju izvođačima, kao i naručiteljima, potpunu kontrolu nad njihovim profilima unutar aplikacije. Kao i u prethodnom poglavlju, obje korisničke skupine imaju iste funkcionalnosti u postavkama računa. Korisnicima je omogućeno promijeniti lozinku, izbrisati ili deaktivirati svoj račun, čime imaju potpunu kontrolu nad svojim profilom.

Kada korisnik odluči deaktivirati ili izbrisati svoj račun, aplikacija ga prvo pita želi li zaista poduzeti tu radnju, nudeći jasne opcije za deaktivaciju ili brisanje računa. Ukoliko korisnik klikne na opciju *Deactivate Account* ili *Delete Account*, otvara se dodatni dijaloški okvir koji ponovno traži potvrdu od korisnika kako bi se osiguralo da je odluka konačna. Ova dodatna potvrda služi za smanjenje rizika od greške ili zabune, osiguravajući da korisnik svjesno donosi odluku o deaktivaciji ili brisanju računa.

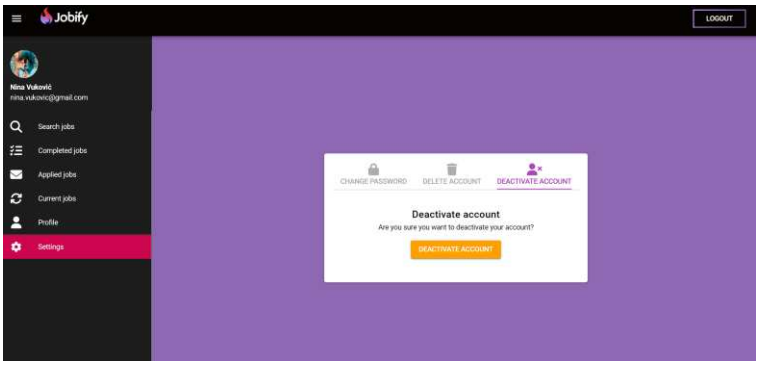
Nakon što je račun deaktiviran, korisnici mogu ponovno aktivirati svoj račun jednostavnim pokušajem prijave. Prilikom pokušaja prijave, aplikacija provjerava status računa. Ako je račun neaktivan, korisniku će se prikazati posebna forma za ponovno aktiviranje računa, gdje treba unijeti svoju email adresu i posljednju korištenu lozinku. Nakon unosa ovih podataka, račun će biti reaktiviran i korisnik će ponovno imati pristup svim funkcionalnostima aplikacije.



SLIKA 41. PRVI DIO PRIKAZA POSTAVKI PROFILA



SLIKA 42. DRUGI DIO PRIKAZA POSTAVKI PROFILA



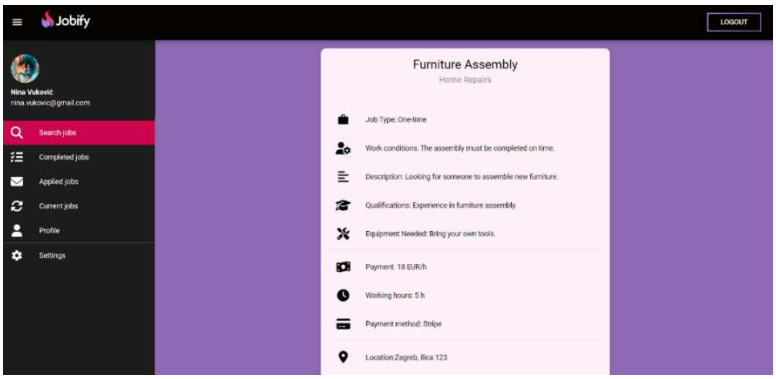
SLIKA 43. TREĆI DIO PRIKAZA POSTAVKI PROFILA

4.10. Postupak prijave na oglas za posao od strane izvođača

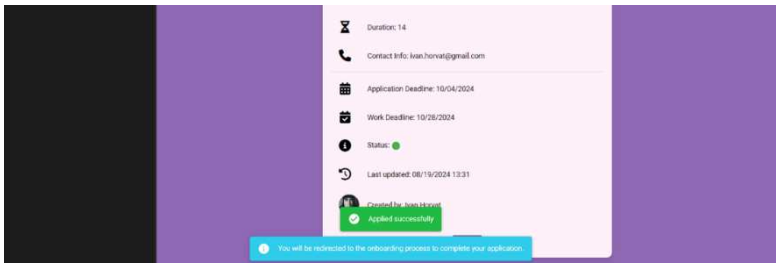
Izvođač pretražuje dostupne poslove na stranici *Search Jobs*, kao što je prikazano na slici 12. Klikom na gumb *Show Details*, izvođač može vidjeti detalje o odabranom poslu. Ako je zainteresiran, može se prijaviti na oglas klikom na gumb *Apply*. U tom trenutku, njegova prijava se šalje naručitelju, gdje se pojavljuje kao kandidat za taj posao, što je prikazano na slici 20.

Istovremeno, pokreće se proces onboardinga za *Stripe*. Prvi korak ovog procesa prikazan je na slici 46. Na kraju onboarding procesa, korisnik se vraća u aplikaciju, gdje dobiva poruku o tome je li proces uspješno ili neuspješno završen. Ova poruka prikazana je na slici 47.

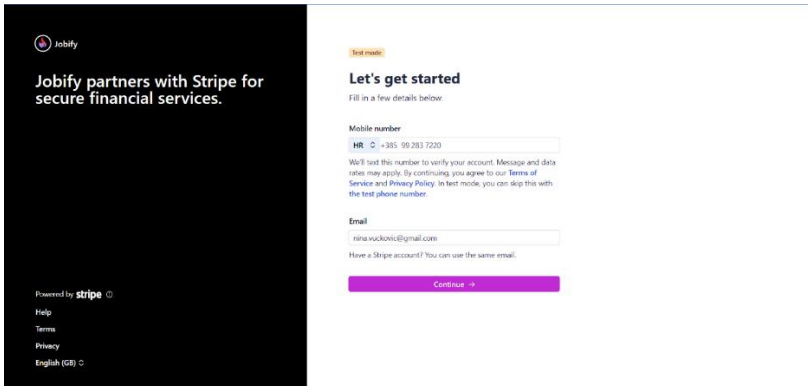
Onboarding proces je ključan pošto omogućuje stvaranje povezanih računa za izvođače unutar *Stripe* sustava [12]. Platforma koristi *Stripe Connect* kako bi upravljala plaćanjima između naručitelja (putem checkout sessiona) i izvođača (putem povezanih računa).



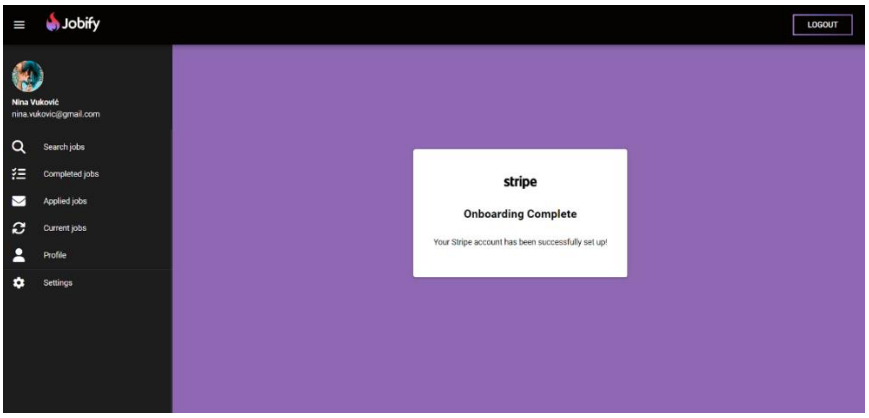
SLIKA 44. PRVI DIO PRIKAZA PRIJAVE NA OGLAS ZA POSAO



SLIKA 45. DRUGI DIO PRIKAZA PRIJAVE NA OGLAS ZA POSAO



SLIKA 46. POČETAK ONBOARDING PROCESA



SLIKA 47. PRIKAZ PORUKE NAKON USPJEŠNOG ZAVRŠETKA ONBOARDING PROCESA

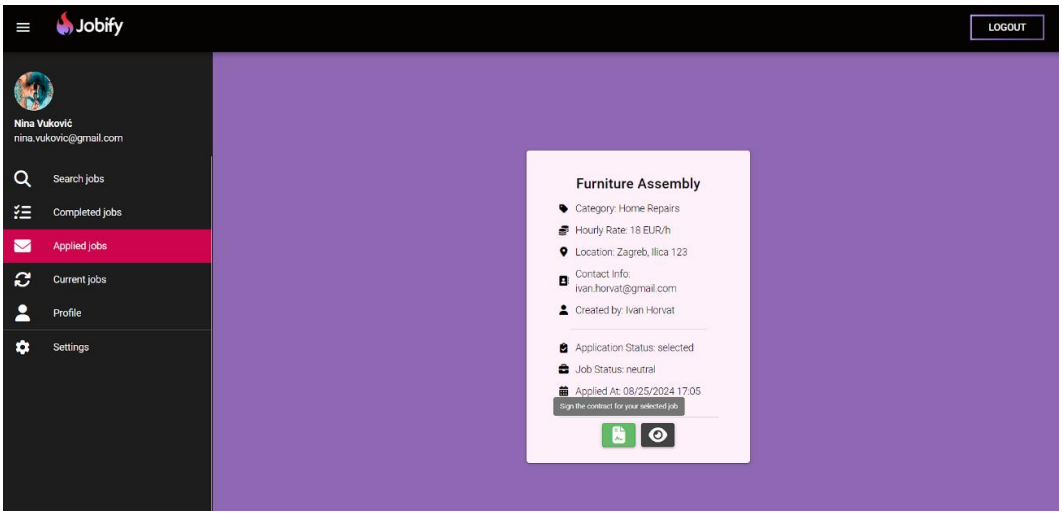
4.11. Pregled prijavljenih poslova izvođača

Nakon uspješnog onboarding procesa, izvođač na stranici *Current Jobs* može pronaći prikaz tog posla u obliku kartice.

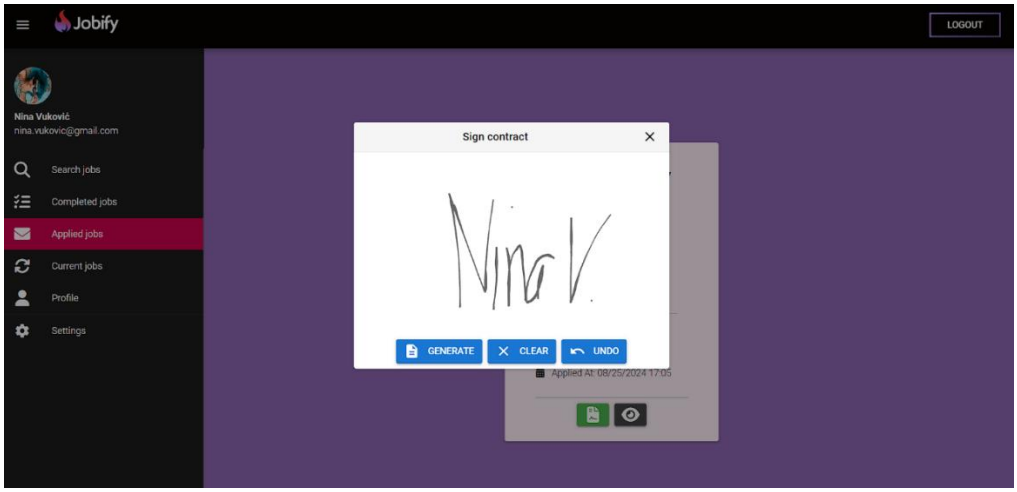
Ovisno o statusu prijave, izvođaču su prikazani sljedeći gumbovi:

- Zeleni gumb *Sign the contract for your selected job*: Pojavljuje se ako je naručitelj već potpisao ugovor za tog izvođača. Izvođač tada može potpisati ugovor i započeti s radom. Prikaz potpisivanja ugovora i koraka koji slijede nakon toga može se vidjeti na idućim slikama.
- Plavi gumb *Waiting for client decision***:** Pojavljuje se ako naručitelj još nije potpisao ugovor ni s jednim izvođačem za taj posao. U tom slučaju, izvođač mora pričekati odluku naručitelja.
- Crveni gumb *You have been rejected for this job:* Pojavljuje se ako je naručitelj odlučio potpisati ugovor s nekim drugim izvođačem. Ovo znači da je prijava tog izvođača odbijena za taj posao.

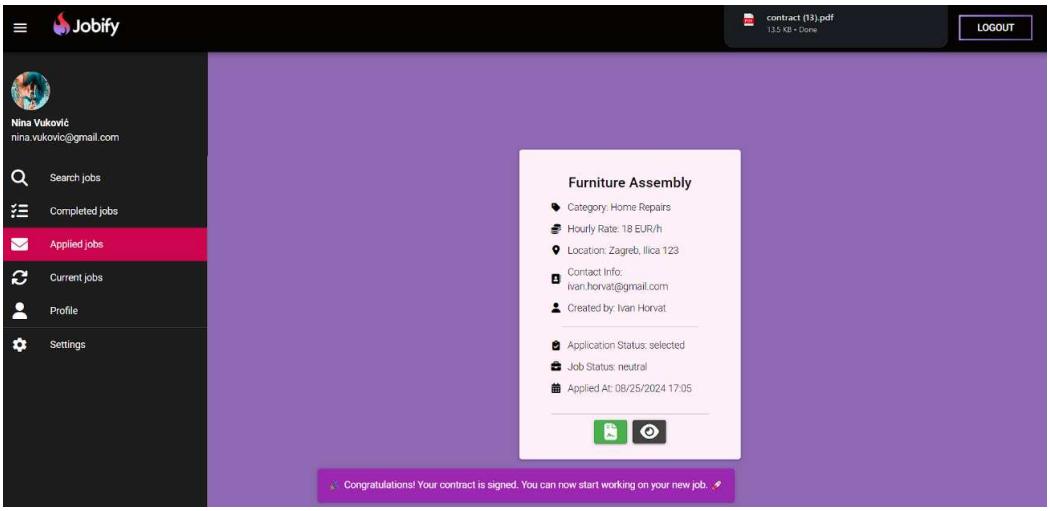
Kada izvođač potpiše ugovor, kartica posla se premješta iz sekcije *Applied Jobs* u sekciju *Current jobs*, signalizirajući da je posao sada u fazi realizacije.



SLIKA 48. PRIKAZ GUMBA ZA POTPISIVANJE UGOVORA O DJELU



SLIKA 49. PRIKAZ POTPISISVANJA UGOVORA OD STRANE IZVOĐAČA



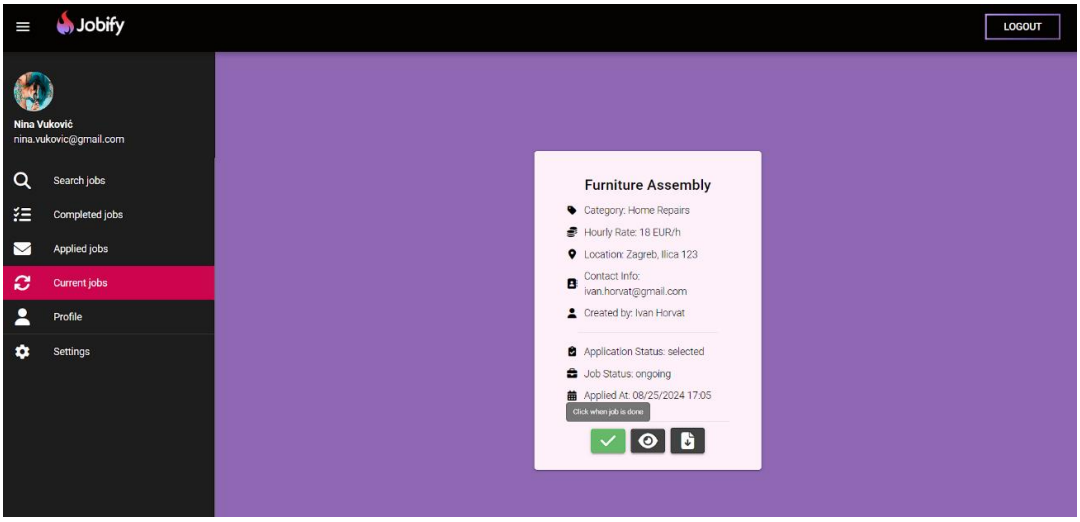
SLIKA 50. PRIKAZ PORUKE NAKON USPJEŠNO POTPISANOG UGOVORA

4.12. Upravljanje tekućim poslovima izvođača

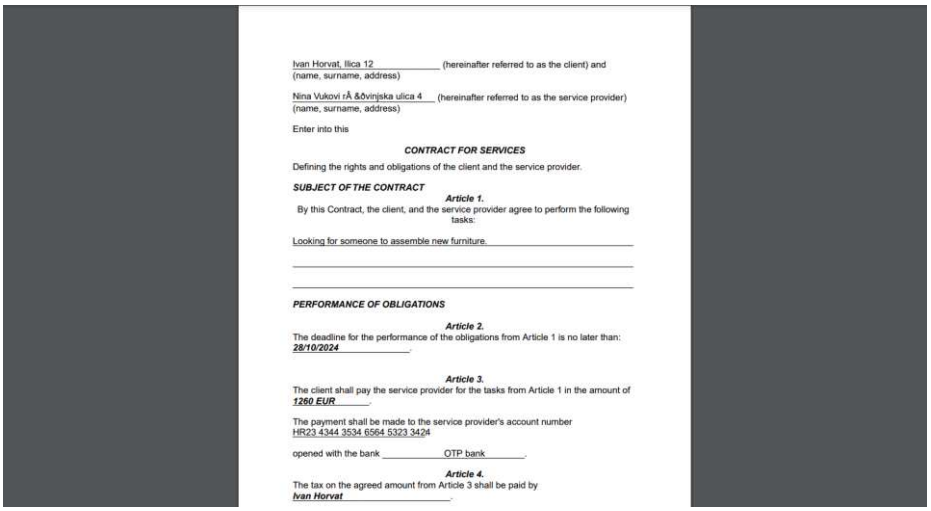
U sekciji *Current Jobs* prikazan je trenutni posao izvođača. Nakon što je ugovor potpisan, posao se pojavljuje u ovoj sekciji, omogućujući izvođaču praćenje svih relevantnih detalja vezanih za posao, uključujući kategoriju, satnicu, lokaciju, kontakt informacije naručitelja, status prijave, i trenutni status posla.

Izvođač također ima mogućnost preuzeti ugovor i pregledati sve uvjete i informacije o poslu. Primjer ugovora koji su potpisale obje strane prikazan je na slikama 52 i 53.

Kada izvođač završi posao, može kliknuti na gumb označen zelenom kvačicom kako bi označio posao kao završen. Nakon toga, kartica posla nestaje iz sekcije "Current Jobs" i premješta se u sekciju “Completed Jobs”. Na dnu ekrana pojavljuje se obavijest koja potvrđuje da je posao uspješno dovršen i da izvođač može očekivati isplatu. Istovremeno, kao što je navedeno u ranijim poglavljima, naručitelju se prikazuje opcija za plaćanje dovršenog posla nakon što ga izvođač označi kao dovršen (slika 29).



SLIKA 51. PRIKAZ GUMBA ZA OZNAČAVANJE POSLA KAO DOVRŠENOG



Ivan Horvat, Ilica 12 (hereinafter referred to as the client) and
(name, surname, address)

Nina Vukovi rÅ &ðvinjska ulica 4 (hereinafter referred to as the service provider)
(name, surname, address)

Enter into this

CONTRACT FOR SERVICES

Defining the rights and obligations of the client and the service provider.

SUBJECT OF THE CONTRACT

Article 1.

By this Contract, the client, and the service provider agree to perform the following tasks:

Looking for someone to assemble new furniture.

PERFORMANCE OF OBLIGATIONS

Article 2.

The deadline for the performance of the obligations from Article 1 is no later than:
28/10/2024.

Article 3.

The client shall pay the service provider for the tasks from Article 1 in the amount of
1260 EUR.

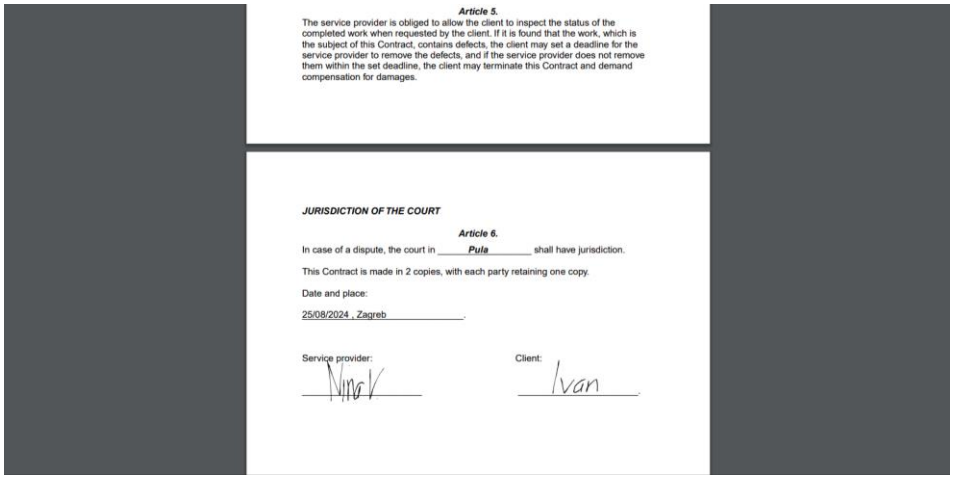
The payment shall be made to the service provider's account number
HR23 4344 3534 6564 5323 3424

opened with the bank OTP bank.

Article 4.

The tax on the agreed amount from Article 3 shall be paid by
Ivan Horvat.

SLIKA 52 PRVI DIO UGOVORA POTPISAN OD OBA KORISNIKA



Article 5.

The service provider is obliged to allow the client to inspect the status of the completed work when requested by the client. If it is found that the work, which is the subject of this Contract, contains defects, the client may set a deadline for the service provider to remove the defects, and if the service provider does not remove them within the set deadline, the client may terminate this Contract and demand compensation for damages.

JURISDICTION OF THE COURT

Article 6.

In case of a dispute, the court in ***Pula*** shall have jurisdiction.

This Contract is made in 2 copies, with each party retaining one copy.

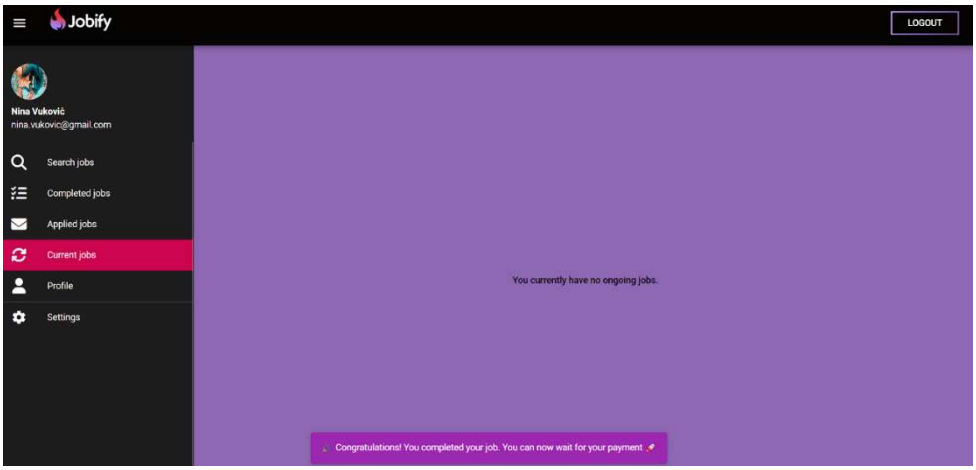
Date and place:

25/08/2024 , Zagreb.

Service provider: Nina V

Client: Ivan

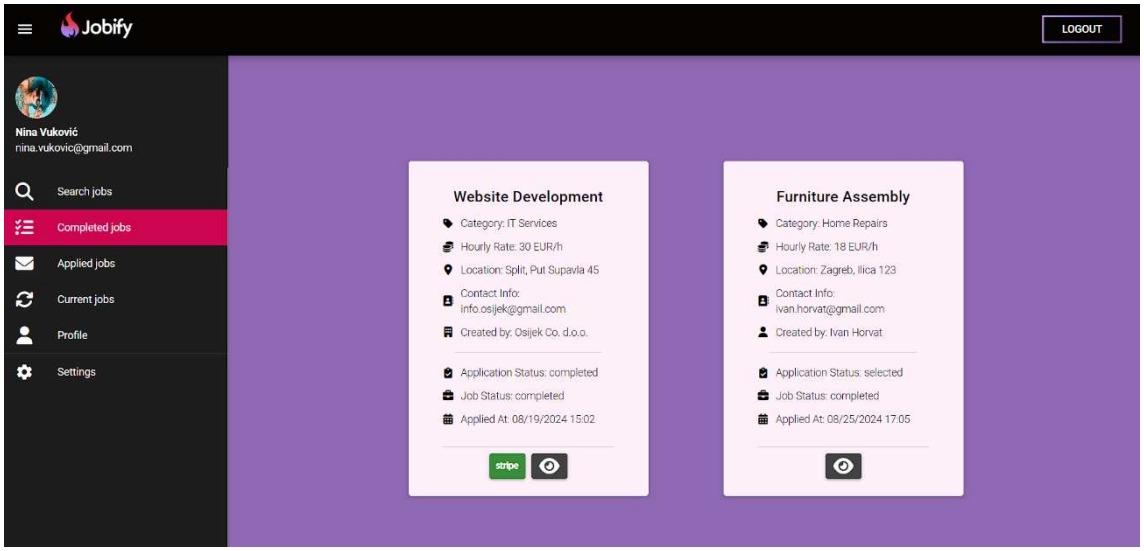
SLIKA 53. DRUGI DIO UGOVORA POTPISAN OD OBA KORISNIKA



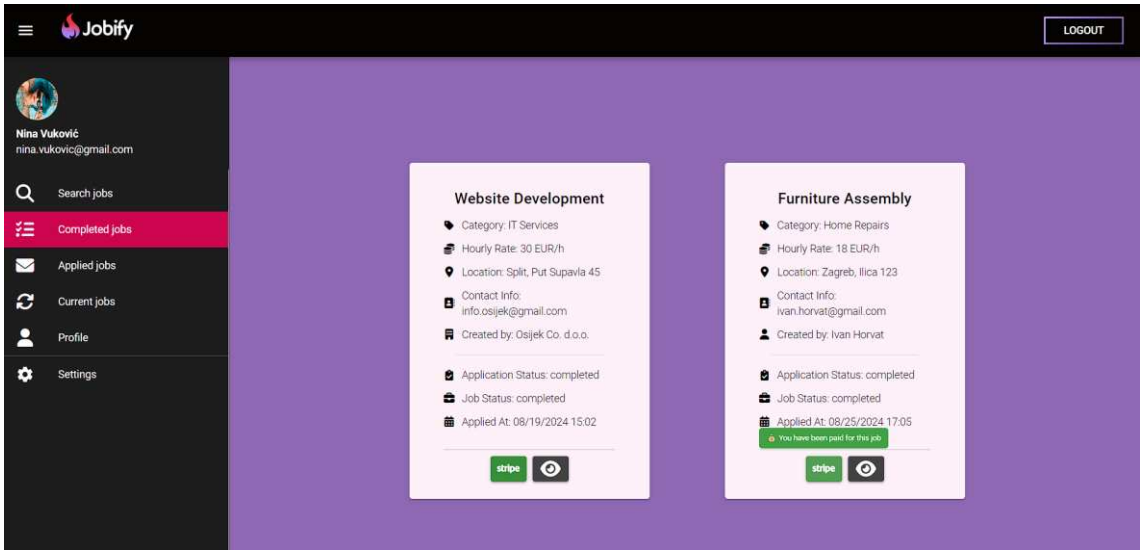
SLIKA 54. PRIKAZ SEKCIJE "CURRENT JOBS" NAKON OZNAČIVANJA POSLA KAO DOVRŠENOG

4.13. Pregled obavljenih poslova izvođača

Sekcija *Completed Jobs* prikazuje sve poslove koje je izvođač označio kao dovršene. Ako naručitelj još nije izvršio uplatu za posao prema Ugovoru o djelu, izvođaču se prikazuje posao kao na slici 55. S druge strane, ako je posao plaćen, izvođaču se prikazuje stanje kao na slici 56.



SLIKA 55. PRIKAZ NEPLAĆENOG POSLA U SEKCIJI "COMPLETED JOBS"



SLIKA 56. PRIKAZ PLAĆENOG POSLA U SEKCIJI "COMPLETED JOBS"

5. Programsko rješenje

U ovom poglavlju opisana su najvažnija programska rješenja razvijena u sklopu ovog rada, s posebnim naglaskom na ključne aspekte implementacije. Objašnjen je način implementacije frontenda i backenda, te analiziran način integracije različitih komponenti aplikacije.

5.1. Registracija korisnika

U ovom poglavlju prikazane su metode koje prikupljaju korisničke informacije tijekom procesa registracije naručitelja, prosljeđuju ih na backend te pohranjuju u bazu podataka PostgreSQL. Za vizualizaciju procesa pohrane podataka korišten je alat pgAdmin.

```
signUp() {
    const baseData = {
        email: this.email,
        password: this.password,
        phoneNumber: this.phoneNumber,
        country: this.country,
        city: this.city,
        address: this.address,
        postalCode: this.postalCode
    };

    let userData = this.isBusiness
        ? {
            ...baseData,
            companyName: this.companyName,
            VATnumber: this.VATnumber,
            type: 'business'
          }
        : {
            ...baseData,
            firstName: this.firstName,
            lastName: this.lastName,
            gender: this.gender,
            dateOfBirth: this.dateOfBirth,
            documentType: this.documentType,
            documentNumber: this.documentNumber,
            type: 'individual'
          };
    this.$api
        .post('/auth/signup/client', userData)
        .then((response) => {
            localStorage.setItem('token', response.data.token);
            const user = response.data.user;

            sessionStorage.setItem('user', JSON.stringify(user));
            sessionStorage.setItem('userId', user.id);

            if (user.role === 'client' && user.type === 'individual') {
                this.$router.push('/client/individual/search-jobs');
            } else if (user.role === 'client' && user.type === 'business') {
                this.$router.push('/client/business/search-jobs');
            }
        })
        .catch((error) => {
            if (error.response && error.response.status === 409) {
                Notify.create({
                    color: 'negative',
                    position: 'bottom',
                    message: 'User with this email already exists. Please try with a different email.',
                    icon: 'error'
                });
            } else {
                Notify.create({
                    color: 'negative',
                    position: 'bottom',
                    message: 'There was an error! ' + error.message,
                    icon: 'error'
                });
            }
            console.error('There was an error!', error);
        });
}
```

SLIKA 57. PRIKAZ FRONTEND FUNKCIJE ZA REGISTRACIJU NARUČITELJA

Na slici 57 prikazana je metoda za registraciju naručitelja, koja prikuplja osnovne podatke unijete u registracijsku formu. Ovisno o tome je li korisnik poslovni subjekt ili fizička

osoba, kreira se odgovarajući objekt s relevantnim podacima. Kreirani objekt zatim se šalje na backend server putem POST zahtjeva na *endpoint* /auth/signup/client koristeći *$api.post()*, što je prilagođena instanca Axios-a u Vue.js aplikaciji. Ovaj API osigurava slanje zahtjeva s automatskim dodavanjem JWT tokena za autentifikaciju.

Nakon obrade zahtjeva na serveru, ako je registracija uspješna, dobiveni JWT token pohranjuje se u *local storage*, dok se korisnički podaci pohranjuju u *session storage* radi daljnje upotrebe u aplikaciji. Zatim se provjerava uloga i tip korisnika kako bi se korisnika preusmjerilo na odgovarajuću stranicu: fizičke osobe na /client/individual/search-jobs, a poslovne subjekte na /client/business/search-jobs.

U slučaju greške, metoda hvata izuzetak. Ako je greška povezana s već postojećom email adresom (status 409), korisniku se prikazuje obavijest o tome putem Quasarove komponente *Notify*. Za sve ostale greške prikazuje se generička poruka kako bi korisnik bio informiran o problemu.

Na backend strani, endpoint prikazan na slici 58 sprema podatke dohvaćene iz tijela zahtjeva u objekt *userData*. Provjerava se tip korisnika, ukoliko je korisnik određenog tipa, atributi specifičnog tipa postavljaju se na null.

```
router.route("/auth/signup/client").post(async (req, res) => {
  try {
    const userData = req.body;
    console.log(userData);
    if (userData.type === "individual") {
      if (!userData.dateOfBirth) {
        return res.status(400).json({ error: "Date of birth is required for individual clients." });
      }
      userData.companyName = null;
      userData.VATnumber = null;
    }
    if (userData.type === "business") {
      userData.firstName = null;
      userData.lastName = null;
      userData.gender = null;
      userData.dateOfBirth = null;
      userData.documentType = null;
      userData.documentNumber = null;
    }

    const client = await registerUser(userData, Client);
    console.log("client auth signup : ", client);
    if (client) {
      const token = generateToken(client);
      res.setHeader("Authorization", `Bearer ${token}`);
      return res.status(201).json({
        message: "Client/Employer registered successfully",
        token,
        role: userData.role,
        user: client,
      });
    } else {
      return res.status(400).json({ message: "Client/Employer registration failed" });
    }
  } catch (error) {
    if (error.message.includes("User with this email already exists")) {
      return res.status(409).json({ message: error.message });
    }
    console.error("Error registering client:", error.message);
    return res.status(500).send("Error registering client: " + error.message);
  }
});
```

SLIKA 58. PRIKAZ ENDPOINTA ZA REGISTRACIJU NARUČITELJA

Poziva se handler za registraciju korisnika prikazan na slici 59. Koja provjerava postoji li već korisnik u bazi podataka s unesenom email adresom. Koristi se metoda *findOne* iz Sequelize ORM-a da bi se pronašao korisnik čiji email odgovara onom koji je poslao korisnik. Ukoliko postoji korisnik s tim emailom, vraća se greška.

Nadalje, lozinka se prije pohranjivanja u bazu podataka hashira pomoću *bcrypt* funkcije. Funkcija zatim stvara novog korisnika u bazi podataka. Ovisno o modelu koji je proslijeđen (bilo *ServiceProvider* ili *Client*), funkcija poziva odgovarajuću metodu create iz Sequelize ORM-a kako bi pohranila podatke korisnika u bazu.

Ako je kreiranje korisnika uspješno, funkcija poziva *_excludeProperties*, koja uklanja određena svojstva iz objekta, uključujući lozinku, osiguravajući da se osjetljivi podaci ne vraćaju niti prikazuju u odgovoru.

Nakon uspješne registracije, poziva se funkcija *generateTo*ken(), prikazana na slici 60, koja prima objekt *client* i generira JWT (JSON Web Token) na temelju informacija o korisniku. Ovaj token sadrži šifrirane informacije kao što su korisnički id (*userId*), uloga korisnika (*role*), te ako je korisnik naručitelj, tip korisnika (*type*).

Generirani JWT token koristi se za autentifikaciju korisnika prilikom budućih zahtjeva prema serveru. U ovom sustavu, token se pohranjuje u localStorage, kao što je već spomenuto u prethodnom poglavlju, i šalje se u zaglavlju HTTP zahtjeva za autorizaciju.

```
async function registerUser(userData, model) {
  try {
    const existingUser = await model.findOne({ where: { email: userData.email } });

    if (existingUser) {
      throw new Error('User with this email already exists.');
    }

    userData.password = await _hashPassword(userData.password);
    let user;

    if (model === ServiceProvider) {
      user = await ServiceProvider.create(userData);
    } else if (model === Client) {
      user = await Client.create(userData, null);
    }
    else {
      throw new Error('Invalid model provided.');
    }
    console.log(user);
    if (user.dataValues) {
      return _excludeProperties(user.dataValues, 'password');
    } else {
      throw new Error('User not created.');
    }
  } catch (error) {
    throw new Error('Error in user registration: ' + error);
  }
}
```

SLIKA 59. PRIKAZ BACKEND FUNKCIJE REGISSTERUSER()

SLIKA 60. PRIKAZ BACKEND FUNKCIJE GENERATETOKEN()

5.2. Prijava korisnika

U ovom poglavlju opisane su frontend i backend metode implementirane za proces prijave korisnika u sustav. Korisničko sučelje za prijavu prikazano je na slici 4, gdje korisnici unose svoje vjerodajnice kako bi pristupili aplikaciji. Prijava korisnika uključuje validaciju unesenih podataka, autentifikaciju korisnika pomoću email adrese i lozinke, te upravljanje sesijama i korisničkim podacima.

Metoda *login()* na frontend strani, prikazana na slici 61, poziva se kada korisnik pokuša prijaviti se unosom svoje email adrese i lozinke. Unutar metode koristi se *$api* objekt za slanje HTTP POST zahtjeva na endpoint /auth/login na serveru. Tijelo zahtjeva sadrži korisnikove vjerodajnice, koje se šalju backendu radi provjere identiteta korisnika.

Ukoliko je prijava uspješna, JWT token se sprema u lokalnu memoriju, a korisnikovi podaci u sesijsku memoriju radi ponovne upotrebe kroz aplikaciju. Ovisno o ulozi i tipu korisnika, korisnik je preusmjeren na odgovarajuće rute unutar aplikacije.

Ako prijava nije uspješna, Quasarova *Notify* komponenta koristi se za obavještavanje korisnika o grešci prilikom prijave. U slučaju da je korisnički račun deaktiviran, vraća se

statusni kod 403, čime se korisniku daje do znanja da je njegov račun deaktiviran i omogućuje se ponovno aktiviranje računa.

```
login() {
    this.$api
        .post('/auth/login', {
            email: this.email,
            password: this.password
        })
        .then((response) => {
            localStorage.setItem('token', response.data.token);
            sessionStorage.setItem('user', JSON.stringify(response.data.user));
            const user = response.data.user;
            sessionStorage.setItem('userId', user.id);
            if (user.role === 'service provider') {
                this.$router.push('/service-provider/search-jobs');
            } else if (user.role === 'client' && user.type === 'individual') {
                this.$router.push('/client/individual/search-jobs');
            } else if (user.role === 'client' && user.type === 'business') {
                this.$router.push('/client/business/search-jobs');
            } else {
                console.error('Unknown role:', user.role);
            }
        })
        .catch((error) => {
            if (error.response && error.response.status === 403) {
                this.deactivationMessage = error.response.data.message;
            } else if (error.response.status === 401) {
                Notify.create({
                    color: 'negative',
                    position: 'bottom',
                    message: 'Incorrect email or password. Please try again.',
                    icon: 'error'
                });
            } else {
                Notify.create({
                    color: 'negative',
                    position: 'bottom',
                    message: 'Failed to login: ' + error.message,
                    icon: 'error'
                });
            }
        });
},
```

SLIKA 61. PRIKAZ FRONTEND FUNKCIJE ZA PRIJAVU KORISNIKA

Na backend strani, endpoint prikazan na slici 62 koristi se za autentifikaciju korisnika prilikom prijave. Nakon što zahtjev stigne na endpoint, iz tijela zahtjeva izvlače se email i password. Zatim se poziva funkcija *checkCredentials()* koja provjerava vjerodajnice korisnika u bazi podataka. Funkcija prvo traži korisnika u tablici naručitelja, a potom u tablici izvođača Ako je račun deaktiviran, vraća se greška i poruka o deaktivaciji. Ako je račun aktivan i lozinka odgovara pohranjenoj, vraćaju se korisnički podaci bez lozinke; u suprotnom se vraća null.

Ako je autentifikacija uspješna, generira se JWT token pomoću funkcije generateToken i dodaje u Authorization header odgovora. Ako korisnik ima profilnu sliku, ona se enkodira u *Base64* format pomoću funkcije *encodeBase64Image*() kako bi se prikazala na frontendu. Ako vjerodajnice nisu ispravne, vraća se status 401 (*Unauthorized*), a za deaktivirane račune status 403. U slučaju greške tijekom provjere vjerodajnica ili generiranja tokena, vraća se status 500 s porukom o grešci na serveru.

```
router.route("/auth/login").post(async (req, res) => {
  try {
    const { email, password } = req.body;
    const user = await checkCredentials(email, password);

    if (user) {
      if (user.error === "deactivated") {
        return res.status(403).json({
          message: user.message,
        });
      }
      if (user.profileImage !== null && user.imageType !== null) {
        const encodedImage = await encodeBase64Image(user.profileImage, user.imageType);
        user.profileImage = encodedImage;
      }
      const token = generateToken(user);
      res.setHeader("authorization", `Bearer ${token}`);
      res.status(200).json({ message: "Successfully logged in", token, user });
    } else {
      return res.status(401).json({ message: "Unauthorized" });
    }
  } catch (error) {
    res.status(500).json({ error: "Server error" });
  }
});
```

SLIKA 62. PRIKAZ ENDPOINTA ZA PRIJAVU KORISNIKA

```
async function checkCredentials(email, password) {
  try {
    let userInstance = await Client.findOne({ where: { email: email } });

    if (userInstance) {
      let user = userInstance.get({ plain: true });
      if (user.status === 'deactivated') {
        return { error: 'deactivated', message: 'Your account is deactivated. Do you want to activate it?' };
      }
      const isPasswordMatch = await _comparePasswords(password, user.password);
      if (isPasswordMatch) {
        return _excludeProperties(user, "password");
      } else {
        return null;
      }
    }

    userInstance = await ServiceProvider.findOne({ where: { email: email } });
    if (userInstance) {
      let user = userInstance.get({ plain: true });
      if (user.status === 'deactivated') {
        return { error: 'deactivated', message: 'Your account is deactivated. Do you want to activate it?' };
      }
      const isPasswordMatch = await _comparePasswords(password, user.password);
      if (isPasswordMatch) {
        return _excludeProperties(user, "password");
      } else {
        return null;
      }
    }
    return null;
  } catch (error) {
    console.error("Error in checkCredentials:", error.message);
    throw new Error("Error during credentials check");
  }
}
```

SLIKA 63. PRIKAZ BACKEND FUNKCIJE ZA PROVJERU VJERODAJNICA

```
async function encodeBase64Image(profileImage, imageType) {
  const imageBase64 = profileImage.toString('base64');
  const imageDataUrl = `data:${imageType};base64,${imageBase64}`;
  console.log(imageDataUrl);
  return imageDataUrl;
}
```

SLIKA 64. PRIKAZ BACKEND FUNKCIJE ZA ENKODIRANJE SLIKE U BASE64 FORMAT

5.3. Kreiranje oglasa za posao

U ovom poglavlju opisane su metode implementirane za kreiranje oglasa za posao u sustavu. Proces kreiranja oglasa obuhvaća unos potrebnih informacija putem korisničkog sučelja, prikazan na slikama 34 i 35, slanje tih podataka na backend server, njihovu obradu i pohranjivanje u bazu podataka. Detaljno su objašnjene frontend i backend komponente koje omogućuju korisnicima da jednostavno kreiraju i objavljuju nove oglase za posao, uz odgovarajuće obavijesti o uspješnosti ili neuspjehu tog procesa.

Funkcija *postNewJob()*, na frontend strani, šalje HTTP POST zahtjev na /client/jobs s podacima o poslu iz forme. Ovisno o uspješnosti zahtjeva, korisniku prikazuje odgovarajuću obavijest.

```
async postNewJob() {
    this.$api
        .post('/client/jobs', this.jobData)
        .then((response) => {
            Notify.create({
                color: 'green-4',
                textColor: 'white',
                icon: 'cloud_done',
                message: 'Job posted successfully!'
            });
        })
        .catch((error) => {
            Notify.create({
                color: 'red-5',
                textColor: 'white',
                icon: 'error',
                message: 'Failed to post job.'
            });
        });
}
```

SLIKA 65. PRIKAZ FRONTEND FUNKCIJE ZA KREIRANJE OGLASA ZA POSAO

Endpoint /client/jobs na backend strani koristi metodu POST za obradu zahtjeva za kreiranje novog oglasa za posao. Handler prvo izvlači podatke o poslu iz tijela zahtjeva i id klijenta (*clientId*) iz JWT tokena korisnika. Zatim se poziva funkcija *createJobAd()* koja kreira novi oglas u bazi podataka.

```
router.route("/client/jobs").post(authenticateToken, async (req, res) => {
  const jobAdData = req.body;
  const clientId = req.user.userId;
  try {
    const jobCreated = await createJobAd(jobAdData, clientId);
    console.log("Job created:", jobCreated);
    res.status(201).send({ message: "Job created successfully", jobCreated });
  } catch (error) {
    console.error("Error creating job:", error);
    res.status(400).send({ error: "Failed to create job ad" });
  }
});
```

SLIKA 66. PRIKAZ ENDPOINTA ZA KREIRANJE OGLASA ZA POSAO

Funkcija *createJobAd()* koristi Sequelize ORM za stvaranje novog zapisa u tablici oglasa za posao. Ako je stvaranje uspješno, vraća se novokreirani oglas (*newCreatedJob*). U slučaju greške, bacanje novog izuzetka osigurava da se greška pravilno prenese natrag kroz lanac funkcija do korisnika.

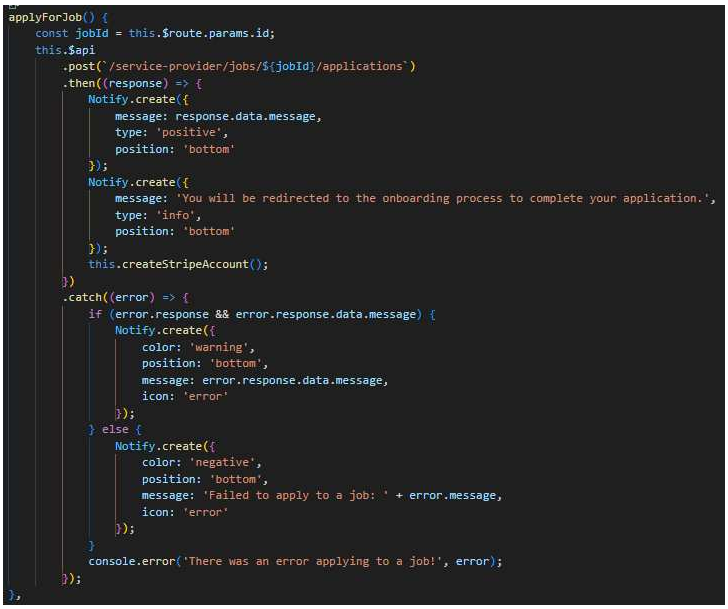
```
async function createJobAd(jobAdData, clientId) {
  try {
    const newCreatedJob = await JobAd.create({
      ...jobAdData,
      clientId: clientId,
    });
    return newCreatedJob;
  } catch (error) {
    throw new Error(error.message);
  }
}
```

SLIKA 67. PRIKAZ BACKEND FUNKCIJE ZA KREIRANJE OGLASA ZA POSAO

5.4. Prijava na oglas za posao od strane izvođača

U ovom poglavlju opisane su metode implementirane za omogućavanje prijave izvođača na oglase za posao unutar sustava. Obrađeni su ključni aspekti frontend i backend funkcionalnosti, uključujući slanje prijave, kreiranje povezanog Stripe računa te rukovanje greškama.

Na frontend strani, funkcija *applyForJob()* koristi se za podnošenje prijave na određeni posao. Na početku funkcije, dohvaća se id posla iz URL parametara pomoću *this.$route.params.id.* Nakon toga, funkcija šalje HTTP POST zahtjev na endpoint /service-provider/jobs/${jobId}/applications koristeći *$api.post()*. Ovaj zahtjev podnosi prijavu za posao. Ako je zahtjev uspješan, korisniku se prikazuje obavijest s porukom preuzetom iz odgovora servera, koja se prikazuje na dnu ekrana. Nadalje, druga obavijest informira korisnika da će biti preusmjeren na onboarding proces za dovršetak prijave. Nakon prikaza obavijesti, funkcija poziva *this.createStripeAccount()* kako bi započela proces kreiranja *Stripe* računa. U slučaju greške, ako server vrati poruku o grešci, poruka se prikazuje korisniku u obliku obavijesti. Ako greška nije specifična, prikazuje se generička obavijest o neuspješnoj prijavi za posao.



SLIKA 68. PRIKAZ FRONTEND FUNKCIJE ZA PRIJAVU NA OGLAS ZA POSAO

Funkcija *createStripeAccount()* koristi se za pokretanje procesa kreiranja *Stripe* računa za povezivanje s određenim poslom. Kao i u funkciji *applyForJob()*, id posla se dohvaća iz URL parametara pomoću *this.$route.params.id*. Nakon toga, funkcija šalje zahtjev na endpoint /service-provider/jobs/${jobId}/stripe-connected-account. Ovaj zahtjev inicira proces kreiranja *Stripe* računa. Ako odgovor sadrži *url* i *accountId*, funkcija sprema *accountId* u *sessionStorage* pod ključem *stripeAccountId*. Zatim se korisnik preusmjerava na URL koji je server vratio kao dio odgovora. Početak ovog procesa ilustriran je na slici 46.

SLIKA 69. PRIKAZ FRONTEND FUNKCIJE ZA KREIRANJE STRIPE RAČUNA ZA IZVOĐAČA POSLOVA

Obje funkcije zajedno omogućuju izvođačima da se prijave za posao i započnu proces povezivanja *Stripe* računa za plaćanje.

Na backend strani, endpoint prikazan na slici 70 obrađuje prijave izvođača na određeni posao. Nakon što se HTTP POST zahtjev pošalje na ovaj endpoint, funkcija dohvaća

jobId iz URL parametara i *serviceProviderId* iz JWT tokena korisnika. Endpoint zatim poziva funkciju *applyForAJob()* kako bi se prijava obradila. Ako je prijava već podnesena za taj posao, vraća se poruka s greškom. U slučaju uspješne prijave, server odgovara s porukom o uspjehu.

```
router.route("/service-provider/jobs/:jobId/applications").post(authenticateToken, async (req, res) => {
  try {
    const jobId = parseInt(req.params.jobId);
    const serviceProviderId = req.user.userId;
    const application = await applyForAJob(jobId, serviceProviderId);

    if (application.message) {
      return res.status(401).json({ message: application.message });
    }

    console.log("Application created:", application);
    res.status(201).json({ message: "Applied successfully", application });
  } catch (error) {
    console.error("Error applying for job:", error.message);
    res.status(404).send({ error: `Failed to apply for job: ${error.message}` });
  }
});
```

SLIKA 70. PRIKAZ ENDPOINTA ZA KREIRANJE OGLASA ZA POSAO

Funkcija *applyForAJob()* koristi se za prijavu izvođača na određeni posao. Prvo provjerava postoji li oglas za posao s određenim jobId u bazi podataka. Ako oglas ne postoji, vraća se greška. Zatim se provjerava je li isti izvođač već podnio prijavu za taj posao. Ako prijava već postoji, funkcija vraća poruku o tome. Ako prijava ne postoji, funkcija kreira novu prijavu pomoću ORM metode *create()* i vraća rezultat. Sve greške koje se pojave obrađuju se i prikazuju korisniku na frontendu.

```
async function applyForAJob(jobId, serviceProviderId) {
  try {
    const jobAd = await JobAd.findByPk(jobId);
    if (!jobAd) {
      throw new Error("Job advertisement not found");
    }
    const existingApplication = await JobVacancy.findOne({
      where: {
        jobAdId: jobId,
        serviceProviderId,
      },
    });

    if (existingApplication) {
      return { message: "You have already applied for this job" };
    }
    const application = await JobVacancy.create({
      jobAdId: jobId,
      serviceProviderId,
    });

    return application;
  } catch (error) {
    throw new Error(error.message);
  }
}
```

SLIKA 71. PRIKAZ BACKEND FUNKCIJE ZA KREIRANJE PRIJAVE ZA OGLAS NA POSAO

Endpoint prikazan na slici 72 koristi se za kreiranje Stripe računa povezanog s određenim poslom. Kada se HTTP POST zahtjev pošalje na ovaj endpoint, prvo se dohvaćaju serviceProviderId iz JWT tokena korisnika i jobId iz URL parametara. Endpoint zatim poziva nekoliko funkcija kako bi obavio različite zadatke. Funkcija *fetchServiceProviderById()* dohvaća podatke o izvođaču iz baze. Nakon toga, funkcija *createServiceProviderStripeAccount()* stvara novi *Stripe* račun za izvođača. Ako je kreiranje računa uspješno, funkcija *updateJobVacanciesWithStripeAccountId()* ažurira oglas za posao s id-jem *Stripe* računa određenog izvođača.

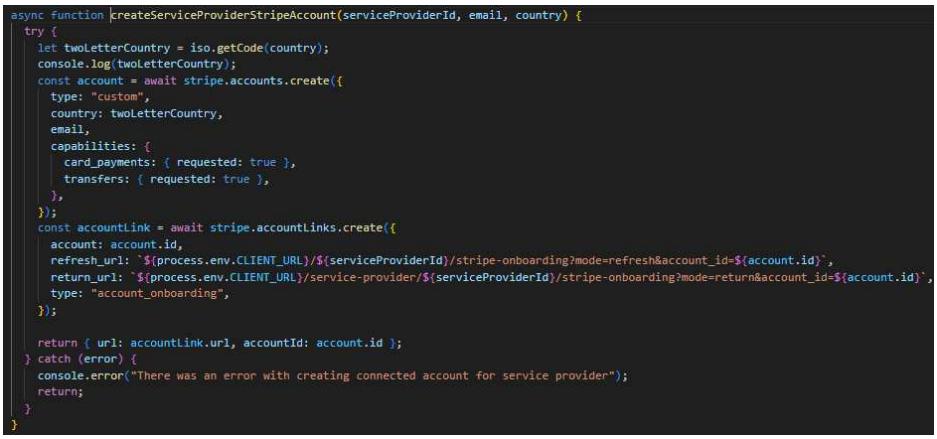
```
router.route("/service-provider/jobs/:jobId/stripe-connected-account").post(authenticateToken, async (req, res) => {
  const serviceProviderId = req.user.userId;
  const jobId = req.params.jobId;
  try {
    const user = await fetchServiceProviderById(serviceProviderId);
    if (!user) {
      return res.status(404).json({ error: "Service provider not found" });
    }

    const stripeData = await createServiceProviderStripeAccount(serviceProviderId, user.email, user.country);

    if (stripeData && stripeData.accountId) {
      const updatedJobVacancy = await updateJobVacanciesWithStripeAccountId(serviceProviderId, jobId, stripeData.accountId);

      if (updatedJobVacancy) {
        return res.status(201).json(stripeData);
      } else {
        return res.status(500).json({ error: "Failed to update service provider with Stripe account ID" });
      }
    }

    return res.status(500).json({ error: "Failed to create Stripe connected account" });
  } catch (error) {
    console.error("Error in Stripe account creation:", error.message);
    res.status(500).json({ error: "Internal server error" });
  }
});
```

SLIKA 72. PRIKAZ ENDPOINTA ZA KREIRANJE STRIPE RAČUNA ZA IZVOĐAČA

Funkcija *createServiceProviderStripeAccount()* koristi se za kreiranje povezanog *Stripe* računa za pružatelja usluga. Na početku funkcije koristi se biblioteka *iso-3166-1-alpha-2* (uvezena kao *iso*) kako bi se dohvatila dvoznamenkasta oznaka zemlje na temelju unesenog naziva države.

Nakon toga, funkcija koristi Stripe API za kreiranje novog Stripe računa. Račun se kreira kao *custom* tip, s omogućenom opcijom za prihvaćanje kartičnih uplata i prijenos sredstava. Podaci o računu, uključujući accountId, koriste se za generiranje URL-a za onboarding proces pomoću *stripe.accountLinks.create*. Onboarding URL-i za osvježavanje i povratak na aplikaciju generiraju se dinamički na temelju ID-a korisnika.

Na kraju funkcija vraća objekt koji sadrži URL za onboarding proces (url) i id računa (*accountId*). U slučaju greške tijekom kreiranja računa ili onboarding procesa, funkcija hvata izuzetak i ispisuje grešku u konzoli, čime se zaustavlja daljnji proces. Primjer uspješnog success URL-a koji vraća *onboarding* proces prikazan je na slici 47.



SLIKA 73. PRIKAZ BACKEND FUNKCIJE ZA KREIRANJE STRIPE RAČUNA ZA IZVOĐAČA

Za ovu aplikaciju odabran je *custom* tip povezanog računa zbog potrebe za potpunom kontrolom nad korisničkim iskustvom i interakcijama. S obzirom da vlasnici *custom* povezanih računa nemaju direktan pristup *Stripe* platformi, ovo rješenje omogućuje aplikaciji prilagodbu onboarding procesa.

5.5. Generiranje ugovora

U ovom poglavlju opisane su metode implementirane za generiranje i upravljanje ugovorima između naručitelja (klijenta) i izvođača. Proces uključuje kreiranje PDF ugovora, njegovo digitalno potpisivanje od strane izvođača, te pohranjivanje ugovora i ažuriranje statusa posla na backendu. Prikazane su ključne funkcionalnosti koje osiguravaju da ugovori budu pravilno generirani, potpisani i pohranjeni, uz jasan prikaz odgovarajućih obavijesti korisniku o uspjehu ili neuspjehu tih operacija.

Na frontend strani, funkcija *save()* je asinkrona funkcija koja se koristi za potpisivanje ugovora i njegovo preuzimanje u PDF formatu. Funkcija sprema potpis korisnika pomoću metode signature.value.save(), koja generira *dataURL* varijablu s potpisom u odgovarajućem formatu. Ova varijabla signature dolazi iz komponente vue-signature koja omogućuje korisnicima unos digitalnog potpisa unutar aplikacije.

Nakon spremanja potpisa, funkcija šalje zahtjev na endpoint koji uključuje potpis u tijelu zahtjeva i traži da se odgovor vrati u formatu blob, što omogućava preuzimanje ugovora u PDF formatu. Kada backend odgovori s generiranim PDF-om, funkcija stvara URL pomoću *window.URL.createObjectURL()* i koristi ovaj URL za kreiranje privremenog

HTML elementa <a>, koji automatski pokreće preuzimanje PDF-a nazvanog contract.pdf.

Ako je proces uspješan, korisniku se prikazuje obavijest pomoću *Notify.create*, koja informira da je ugovor uspješno potpisan i da može početi raditi na novom poslu.

Na backend strani, funkcija *generateServiceProviderContract()* je odgovorna za generiranje PDF ugovora između naručitelja (klijenta) i izvođača. Funkcija prvo dohvaća potrebne podatke, uključujući podatke o klijentu, izvođaču, poslu i iznosu ugovora, iz objekta *contractData*. Također, dohvaća se potpis klijenta ako ugovor već postoji, te se taj potpis koristi za generiranje potpisa unutar PDF dokumenta.

Kroz biblioteku *PDFDocument*, funkcija generira PDF dokument veličine A4. Dok se PDF generira, svi podaci se pohranjuju u buffers niz, kako bi se kasnije mogli spojiti u jedinstveni PDF dokument.

Kada se dokument završi, generirani PDF se pohranjuje u bazu podataka koristeći funkciju *_updatePDFContract*. Ova funkcija ažurira sadržaj PDF-a (*contractData*), dodaje potpis izvođača (*serviceProviderSignature*), i mijenja status ugovora na "*completed*". Također, funkcija *generateServiceProviderContract()* ažurira status posla na "*ongoing*" pomoću funkcije *_updateJobVacancyJobStatus*.

Što se tiče formata ugovora, on sadrži standardne elemente poput naziva klijenta i izvođača, opisa posla, roka isporuke, iznosa plaćanja, bankovnih podataka i ostalih relevantnih klauzula. Na kraju dokumenta, dodaju se potpisi izvođača i klijenta, što završava proces generiranja ugovora.

Bitno je napomenuti da biblioteka PDFKit koja se koristi ne dopušta direktno modificiranje već postojećih PDF dokumenata. Zbog toga se mora generirati novi dokument i pregaziti postojeći ugovor (koji je inicijalno generirao klijent). Ova strategija osigurava da su svi podaci točno ažurirani i da ugovor odražava najnovije izmjene i potpise.

Na slici 50 prikazano je što se događa na frontendu nakon što je ugovor generiran od strane izvođača.

```
async function _updateJobVacancyJobStatus(jobAdId, serviceProviderId ) {
    try {
        await JobVacancy.update({
            jobStatus: 'ongoing'
        }, {
            where: {
                jobAdId,
                serviceProviderId
            }
        });
        console.log(`Job Vacancy job status updated for jobAdId ${jobAdId}: ongoing for ID ${serviceProviderId}`);
    } catch (error) {
        console.error('Error updating job vacancy job status:', error);
        throw error;
    }
}
```

SLIKA 74. PRIKAZ BACKEND FUNKCIJE KOJA AŽURIRA STATUS POSLA IZVOĐAČA

```
async function _updatePDFContract(contractData, serviceProviderSignature, jobAdId) {
    try {
        const contract = await JobContract.findOne({ where: { jobAdId } });
        if (!contract) {
            throw new Error('Contract not found');
        }
        await contract.update({
            contract: contractData,
            serviceProviderSignature,
            status: 'completed'
        });
        console.log('Contract updated successfully');
    } catch (error) {
        console.error('Error updating the contract:', error);
        throw error;
    }
}
```

SLIKA 75. PRIKAZ BACKEND FUNKCIJE KOJA AŽURIRA UGOVOR

5.6. Plaćanje putem Stripea

U ovom poglavlju opisani su ključni koraci i funkcionalnosti vezani za proces isplate unutar aplikacije. Proces isplate implementiran je korištenjem *Stripea*, koji omogućuje sigurno i efikasno upravljanje plaćanjima između korisnika i izvođača. Objašnjeni su i frontend i backend dijelovi aplikacije, od iniciranja plaćanja do potvrde i generiranja faktura.

Na frontend strani, funkcija *handlePayClick()* je asinkrona funkcija koja pokreće process plaćanja. Kada se funkcija pozove, dohvaća id posla (*jobId*) iz URL parametara. Zatim šalje zahtjev na endpoint /client/jobs/${jobId}/pay koristeći axios instancu.

Ako je odgovor sa servera uspješan i sadrži URL, funkcija automatski preusmjerava korisnika na taj URL postavljanjem *window.location.href* na *response.data.url*. Ovaj URL vodi korisnika na *Stripe* formu za plaćanje, gdje može unijeti potrebne podatke za izvršenje transakcije. Primjer popunjavanja te forme prikazan je na slici 30.

Međutim, ako odgovor sa servera ne sadrži URL, funkcija koristi *Notify.create* za prikazivanje obavijest korisniku s odgovarajućom porukom.

U slučaju da dođe do greške tijekom izvršavanja funkcije, bilo tijekom dohvaćanja podataka ili obrade odgovora, korisniku se prikazuje obavijest s odgovarajućom porukom.

Ova funkcija je ključna za upravljanje procesom plaćanja, omogućujući korisnicima da započnu transakciju i dobiju informaciju o njenom statusu, bilo da je ona uspješno pokrenuta ili ne.

```
const handlePayClick = async () => {
    try {
        const jobId = route.params.jobId;
        const response = await $api.post(`/client/jobs/${jobId}/pay`);
        if (response.data.url) {
            window.location.href = response.data.url;
        } else {
            Notify.create({
                type: 'negative',
                message: 'Failed to initiate payment.'
            });
        }
    } catch (error) {
        console.error('Error initiating payment:', error);
        Notify.create({
            type: 'negative',
            message: 'An error occurred while initiating the payment.'
        });
    }
};
```

SLIKA 76. PRIKAZ FRONTEND FUNKCIJE ZA INICIRANJE ISPLATE OBAVLJENOG POSLA

Backend implementacija za plaćanje započinje zahtjevom na /client/jobs/:jobId/pay, gdje se koristi autentifikacija tokenom za osiguranje podataka. Prvo se dohvaća *jobId* iz URL-a i *clientId* iz JWT tokena. Funkcija *fetchDataForPayment*() dohvaća potrebne podatke, poput *serviceProviderAccountId*, *customerId*, *priceId* i *title*. Ako neki od tih podataka nedostaje, vraća se HTTP odgovor sa statusom 400. Ako su svi podaci prisutni, funkcija *createCheckoutSession()* kreira *Stripe* sesiju plaćanja putem *Stripe API-ja*, uključujući URL-ove za uspješno i neuspješno plaćanje koji vraćaju korisnika natrag u aplikaciju. Primjer uspješnog plaćanja i preusmjeravanja korisnika natrag u aplikaciju prikazan je na slici 31.

Nakon što *Stripe* uspješno kreira sesiju, vraća se *URL* i *session_id*, koji omogućuju korisniku preusmjeravanje na *Stripe* stranicu za plaćanje. Budući da aplikacija zahtijeva

upravljanje cijelim procesom unutar nje same, bez korištenja Stripe dashboarda za fakture, fakture se generiraju unutar aplikacije koristeći *PDFKit* biblioteku s podacima iz *Stripe invoice* objekta.

```
router.route("/client/jobs/:jobId/pay").post(authenticateToken, async (req, res) => {
  const jobId = req.params.jobId;
  const clientId = req.user.userId;
  try {
    const data = await fetchDataForPayment(jobId, clientId);
    if (!data || !data.serviceProviderAccountId || !data.customerId || !data.priceId || !data.title) {
      return res.status(400).json({ success: false, message: "Required data for payment is missing." });
    }

    const paymentData = await createCheckoutSession(jobId, data.type, data.serviceProviderAccountId, data.customerId, data.priceId, data.title);

    if (paymentData && paymentData.url) {
      return res.status(200).json({ success: true, url: paymentData.url, jobId: jobId });
    } else {
      return res.status(500).json({ success: false, message: "Failed to create Stripe payment session." });
    }
  } catch (error) {
    console.error("Error processing payment:", error.message);
    return res.status(500).json({ success: false, message: "Internal server error.", error: error.message });
  }
});
```

SLIKA 77. PRIKAZ ENDPOINTA ZA PLAĆANJE

```
async function createCheckoutSession(jobAdId, type, serviceProviderAccountId, customerId, priceId, title) {
  try {
    const session = await stripe.checkout.sessions.create(
      {
        payment_method_types: ["card"],
        mode: "payment",
        payment_intent_data: {
          application_fee_amount: 123,
          setup_future_usage: "off_session",
        },
        invoice_creation: {
          enabled: true,
          invoice_data: {
            description: `Invoice for completed job - ${title}, job ID: ${jobAdId}`,
            rendering_options: {
              amount_tax_display: "include_inclusive_tax",
            },
          },
        },
        customer: customerId,
        line_items: [
          {
            price: priceId,
            quantity: 1,
          },
        ],
        metadata: {
          price_id: priceId,
          customer_id: customerId,
          jobAdId: jobAdId,
        },
        success_url: `${process.env.CLIENT_URL}/client/${type}/payment-status?session_id={CHECKOUT_SESSION_ID}&jobId=${jobAdId}`,
        cancel_url: `${process.env.CLIENT_URL}/client/${type}/payment-status?jobId=${jobAdId}`,
      },
      {
        stripeAccount: serviceProviderAccountId,
      }
    );
    return { session_id: session.id, url: session.url };
  } catch (error) {
    console.error("Error creating Stripe session:", error.message);
    throw new Error("Failed to create Stripe payment session.");
  }
}
```

SLIKA 78. PRIKAZ BACKEND FUNKCIJE ZA KREIRANJE CHECKOUT SESIJE

Nakon što korisnik uspješno izvrši plaćanje, preusmjerava se natrag u aplikaciju gdje se potvrđuje plaćanje. Aplikacija dohvaća detalje o sesiji i ugovoru, provjerava status transakcije putem Stripe API-ja, te pokušava dohvatiti i provjeriti status fakture više puta. Ova višestruka provjera je potrebna jer Stripeu treba dodatno vrijeme za obradu informacije da je faktura plaćena nakon što je posao plaćen u checkout sesiji. Ako je faktura uspješno plaćena, podaci o plaćanju se ažuriraju u bazi, a generira se PDF faktura za klijenta. Ovaj proces omogućuje automatizaciju plaćanja unutar aplikacije.

6. Zaključak

U vrijeme kada sve više poslovnih aktivnosti prelazi u digitalno okruženje, korištenje interneta za stvaranje korisnih rješenja postaje ključna prilika. Iz tog razloga, rad na ovoj aplikaciji bio je ne samo vrlo zanimljiv, već i zahtjevan proces.

Jedan od najizazovnijih aspekata razvoja bio je implementacija Stripe API-ja za upravljanje plaćanjima. Iako Stripe API nudi širok spektar mogućnosti, njegova složenost i određeni nedostaci, poput zahtjevnosti u integraciji s postojećim sustavima, predstavljao je izazov tijekom razvoja. Unatoč tim poteškoćama, ostvarena je osnovna integracija koja omogućuje sigurno upravljanje plaćanjem unutar aplikacije.

Tijekom izrade ovog rada, otkriveno je da je ključ za uspješno razvijanje ovakve aplikacije pažljivo planiranje i izbor tehnologija koje najbolje odgovaraju specifičnim potrebama projekta. Iako je aplikacija funkcionalna i zadovoljava postavljene ciljeve, rad je otvorio i nova pitanja, poput potrebe za proširenjem funkcionalnosti te istraživanjem dodatnih mogućnosti, kao što su uvođenje algoritma za prilagodbu prikazanih poslova prema korisnikovim interesima i prethodnim prijavama te dodavanje sustava recenzija za izvođače i naručitelje.

Ova poboljšanja omogućila bi aplikaciji da zadovolji trenutne potrebe tržišta i ostane relevantna u budućnosti. Fleksibilnost za prilagodbu novim izazovima i korisničkim zahtjevima ključno je za dugoročan uspjeh svake aplikacije.

Literatura

[1] Vue.Js [Internet]. Vuejs.org. [pristupljeno 23. kolovoza 2024.]. Dostupno na: https://vuejs.org/

[2] Quasar Framework - Izradite visokoučinkovita VueJS korisnička sučelja u rekordnom vremenu [Internet]. Quasar Framework. [pristupljeno 23. kolovoza 2024.]. Dostupno na: https://quasar.dev/

[3] Express - Node.js web aplikacijski okvir [Internet]. Expressjs.com. [pristupljeno 23. kolovoza 2024.]. Dostupno na: https://expressjs.com/

[4] Run JavaScript everywhere [Internet]. Nodejs.org. [pristupljeno 23. kolovoza 2024.]. Available from: https://nodejs.org/en

[5] Stripe [Internet]. Stripe.com. [pristupljeno 23. kolovoza 2024.]. Dostupno na: https://stripe.com/en-hr

[6] PostgreSQL [Internet]. PostgreSQL. 2024 [pristupljeno 23. kolovoza 2024.]. Dostupno na: https://www.postgresql.org/

[7] Sequelize [Internet]. Sequelize.org. [pristupljeno 23. kolovoza 2024.]. Dostupno na: https://sequelize.org/

[8] PgAdmin - PostgreSQL Tools [Internet]. Pgadmin.org. [pristupljeno 23. kolovoza 2024.]. Dostupno na: https://www.pgadmin.org/

[9] Npm: Vue-signature [Internet]. npm. [pristupljeno 25. kolovoza 2024.]. Dostupno na: https://www.npmjs.com/package/vue-signature?activeTab=readme

[10] PDFKit [Internet]. Pdfkit.org. [pristupljeno 25. kolovoza 2024.]. Dostupno na: https://pdfkit.org/

[11] Sessions [Internet]. Stripe.com. [pristupljeno 25. kolovoza 2024.]. Dostupno na: https://docs.stripe.com/api/checkout/sessions

[12] Build a connect integration [Internet]. Stripe.com. [pristupljeno 26. kolovoza 2024.]. Dostupno na: https://docs.stripe.com/connect/onboarding/quickstart

Tablica slika

Prilozi

Github repozitorij klijenta: https://github.com/rstarcic/zavrsni-rad-frontend

Github repozitorij servisa: https://github.com/rstarcic/zavrsni-rad-backend

Link aplikacije: https://jobify-wwit.onrender.com/