IZRADA WEB APLIKACIJE ZA OGLAŠAVANJE I PRONALAZAK POSLOVA

Spajić, Marin

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Stručni diplomski studij Primijenjeno računarstvo

MARIN SPAJIĆ

Z A V R Š N I R A D

IZRADA WEB APLIKACIJE ZA OGLAŠAVANJE I PRONALAZAK POSLOVA

Split, siječanj 2025.

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Stručni diplomski studij Primijenjeno računarstvo

Predmet: Napredno programiranje web aplikacija

Z A V R Š N I R A D

Kandidat: Marin Spajić

Naslov rada: Izrada web aplikacije za oglašavanje i pronalazak poslova

Mentor: Marina Rodić, viši predavač

Split, siječanj 2025.

SADRŽAJ

SAŽETAK

U sklopu ovog završnog rada razvijena je web aplikacija za traženje posla. Aplikacija se sastoji od korisničkog dijela prilagođenog posloprimcima i dijela prilagođenog poslodavcima. Posloprimcima aplikacija omogućava pregled i pretraživanje otvorenih pozicija, prijavu na oglase za posao te prilaganje životopisa (CV-a) uz svaku prijavu. Poslodavci putem aplikacije mogu objavljivati oglase za posao, upravljati prijavama kandidata i pratiti njihov status kroz selekcijski proces. Aplikacija je izrađena korištenjem suvremenih web tehnologija, uključujući *React* za razvoj korisničkog sučelja i *React Query* za upravljanje podacima na klijentskoj strani. Na poslužiteljskoj strani korišteni su *.NET Core* za implementaciju poslovne logike i *Entity Framework* za pristup podacima, dok je baza podataka implementirana u *MS SQL* okruženju. Komunikacija između klijentske i poslužiteljske strane ostvarena je putem REST API-ja.

Ključne riječi: *.NET Core, pretraga poslova, React, oglasi, web aplikacija*

SUMMARY

Advertising and finding jobs web application development

As part of this final project, a web application for job searching was developed. The application consists of a user interface tailored to job seekers and a section designed for employers. For job seekers, the application enables browsing and searching for open positions, applying for job postings, and attaching resumes with each application. Employers can use the application to post job advertisements, manage candidate applications, and track their status throughout the selection process. The application was built using modern web technologies, including *React* for developing the user interface and *React Query* for client-side data management. On the server side, *.NET Core* was used to implement business logic, and *Entity Framework* was employed for data access, while the database was implemented in an *MS SQL* environment. Communication between the client and server sides is facilitated through a REST API.

Keywords: *.NET Core, job search, React, job postings, web application*

1. UVOD

U suvremenom digitalnom dobu tržište rada prolazi kroz značajne promjene. Tradicionalni načini zapošljavanja, poput fizičkog oglašavanja radnih mjesta i prijava putem pošte ili e-maila, postaju sve manje učinkoviti. Pojava internetskih platformi za traženje posla značajno je olakšala proces zapošljavanja, pružajući poslodavcima i posloprimcima brži i jednostavniji način povezivanja. Uz načine zapošljavanja, mijenja se i način rada. *AT&T*, vodeći pružatelj telekomunikacijskih usluga, predviđa dominaciju hibridnog načina rada u nadolazećim godinama [1].

Tema ovog završnog rada je razvoj web aplikacije za traženje posla koja povezuje radnike i poslodavce putem moderne i intuitivne platforme. Cilj aplikacije je pružiti korisnicima učinkovit alat za pretraživanje otvorenih pozicija, kao i za pronalaženje odgovarajućih kandidata ili tvrtki prema specifičnim potrebama.

Platforma je strukturirana prema dvije glavne korisničke uloge: tvrtke i kandidati. Tvrtke mogu koristiti aplikaciju za objavu otvorenih pozicija, pretraživanje kandidata i upravljanje prijavama. Kandidati imaju mogućnost pretraživanja otvorenih pozicija, prijave na poslove i upravljanje svojim profilima. Svaka uloga ima specifične funkcionalnosti koje su prilagođene njihovim potrebama, čime se osigurava da aplikacija zadovolji zahtjeve oba tipa korisnika.

Iduće poglavlje bavi se arhitekturom same aplikacije, a u trećem poglavlju opisani su korišteni alati. Glavne funkcionalnosti prikazane su u četvrtom poglavlju. Poseban naglasak stavljen je na ključne značajke aplikacije, poput upravljanja prijavama i interakcije između korisnika U petom poglavlju iznosi se zaključak o radu i funkcionalnostima aplikacije, korištenim tehnologijama, te su prikazane mogućnosti daljnjeg razvoja aplikacije kako bi se zadovoljile buduće potrebe tržišta rada i korisnika.

2. DIZAJN

U ovom poglavlju opisan je postupak razvoja web aplikacije. Poglavlje sadrži opis potrebnih funkcionalnosti, pregled arhitekture projekta na poslužiteljskoj i klijentskoj strani. Na kraju poglavlja, opisani su korišteni programski jezici te pomoćne knjižnice.

2.1. Funkcionalnosti programskog rješenja

Prije početka razvoja programskog rješenja, definirane su funkcionalnosti koje aplikacija mora sadržavati:

1. Pretraživanje i filtriranje osoba: Aplikacija mora omogućiti svim korisnicima pretraživanje registriranih korisnika prema imenu i lokaciji. Također, potrebno je osigurati funkcionalnost filtriranja rezultata prema kriterijima kao što su lokacija, razina obrazovanja i vještine. Samo prijavljeni korisnici mogu vidjeti profil osobe.
2. Pretraživanje i filtriranje tvrtki: Aplikacija mora omogućiti svim korisnicima pretraživanje registriranih tvrtki prema imenu i lokaciji. Također je potrebno osigurati funkcionalnost filtriranja rezultata prema različitim kriterijima, kao što su tip organizacije tvrtke, industrija u kojoj tvrtka djeluje, te po broju zaposlenika tvrtke. Samo prijavljeni korisnici mogu vidjeti detaljne informacije o tvrtki.
3. Pretraživanje i filtriranje otvorenih oglasa: Aplikacija mora omogućiti svim korisnicima pretraživanje trenutno otvorenih oglasa za posao po naslovu i lokaciji. Također, potrebno je osigurati funkcionalnost filtriranja rezultata prema određenim kriterijima. Kriteriji uključuju industriju, način obavljanja posla (iz ureda, udaljeno ili kombinirano), traženu razinu edukacije, potrebne vještine kandidata, benefite koji se nude te raspon plaće. Prijavljeni korisnici mogu vidjeti detalje oglasa.
4. Registracija i prijava korisnika: Aplikacija mora omogućiti registraciju putem adrese elektroničke pošte i unosa osobnih podataka, te kasniju prijavu. Korisnici se dijele na dva tipa: predstavnik tvrtke i osoba koja traži posao. Nakon registracije korisnik unosi osnovne informacije.
5. Uređivanje profila: Svi prijavljeni korisnici moraju imati funkcionalnost uređivanja svog javnog profila. Kod tvrtke, profil uključuje opis tvrtke, fotografiju, godinu

osnivanja. Kod osobe koja traži posao, profil uključuje opis, povijest zaposlenja, te prijenos više životopisa u obliku dokumenta.

6. Spremanje kandidata: Aplikacija predstavniku tvrtke mora omogućiti zabilježbu potencijalnih kandidata.
7. Prijava na oglas: Aplikacija registriranoj osobi mora omogućiti prijavu za posao, koristeći prethodno preneseni životopis i propratno pismo, koje će biti dostupno predstavniku tvrtke. Predstavnik tvrtke mora imati mogućnost pregleda poslane prijave i preuzimanja priloženog životopisa.
8. Poziv za prijavu na oglas: Aplikacija mora omogućiti predstavniku tvrtke da potencijalnom kandidatu pošalje pozivnicu za prijavu na oglas, uz pismo zamolbe. Osoba mora imati pregled pozivnica, te istu prihvatiti ili izbrisati.
9. Poziv za razgovor: Aplikacija mora omogućiti predstavniku tvrtke da pozove prijavljenog kandidata na razgovor u određenom terminu. Pozvana osoba ima mogućnost prihvaćanja i odbijanja pozivnice.
10. Poruke: Aplikacija mora omogućiti pisanu komunikaciju između predstavnika tvrtke i osobe koja traži posao. Također, komunikacija između dvije osobe ili dvije tvrtke mora biti onemogućena.
11. Obavijesti putem elektroničke pošte: Aplikacija mora omogućiti osobi koja traži posao da definira kriterije po kojima može primati obavijesti kada se objavi novi oglas za posao. Mogući kriteriji uključuju: ključnu riječ, lokaciju, raspon plaće, razinu obrazovanja ili iskustva.
12. Statistika: Aplikacija mora omogućiti prikaz ključnih statistika u obliku grafa. Prikazani podaci moraju uključivati: najtraženije vještine, najčešće lokacije kandidata, distribuciju razina edukacije i godine iskustva.

2.2. Baza podataka

Baza podataka predstavlja ključan dio svake aplikacije. Koristi se za pohranu i upravljanje podacima. Upravljanje tim podacima obavlja se pomoću sustava za upravljanje bazama podataka (engl. *Database Management System*). Podaci, *DBMS* i aplikacije koje s njima rade zajedno čine cjelokupni sustav baze podataka.

Način na koji se podaci pohranjuju ovisi o vrsti baze podataka. U relacijskim bazama, primjerice, podaci su organizirani u tablicama. Svaka tablica predstavlja određeni entitet, a sastoji se od redaka i stupaca. Redci predstavljaju pojedinačne instance tog entiteta, dok stupci opisuju njihove atribute. Ovakva struktura omogućuje jednostavno pretraživanje, upravljanje, ažuriranje i organizaciju podataka. Većina baza podataka koristi SQL (engl. *Structured Query Language*) za izvršavanje upita i upravljanje podacima.

Aplikacija za traženje posla koristi bazu podataka *MS SQL* i *Microsoft SQL Server Management Studio (SMSS)* za upravljanje bazom podataka. Baza podataka sadrži 28 tablica, koje se prema primjeni mogu podijeliti na tri tipa:

- Osnovni entiteti - Ove tablice pohranjuju ključne podatke i predstavljaju osnovne poslovne objekte u sustavu.
    - Primjeri: `Candidates`, `Employers`, `Jobs`
- Vezne tablice – Tablice koje povezuju dvije glavne tablice u relaciji "više prema više".
    - Primjeri: `CandidateSkill`, `JobSkill`, `SavedCandidates`
- Pomoćne tablice – Tablice koje sadrže uglavnom fiksan sadržaj, kao što su ponuđene opcije padajućih izbornika.
    - Primjeri: `ExperienceLevel`, `EducationLevel`, `Countries`

2.3. Arhitektura klijentske aplikacije

Klijentski dio aplikacije (engl. *frontend*) napisan je u *TypeScriptu* uz korištenje razvojnog okvira *React*, čime se osigurava bolja struktura kôda, tipizacija i održivost aplikacije. *React*, kao jedna od najpopularnijih knjižnica za razvoj korisničkih sučelja, omogućuje modularni i komponentni pristup razvoju, što olakšava razvoj, održavanje i proširivanje funkcionalnosti aplikacije.

Za izgradnju korisničkog sučelja korištena je knjižnica ***Mantine UI*** koja omogućava brzu izradu responzivnih i vizualno privlačnih sučelja s minimalnim pisanjem prilagođenog CSS-a (engl. *Cascading Style Sheets*), čime se ubrzava proces razvoja i osigurava konzistentnost u dizajnu.

Za upravljanje dohvatom podataka s poslužitelja, implementirana je knjižnica ***React Query***. Ova knjižnica ne samo da pojednostavljuje upravljanje podacima dobivenim iz API-ja (engl. *Application Programming Interface*) već nudi i funkcionalnosti poput keširanja podataka, sinkronizacije podataka u pozadini, automatskog osvježavanja te optimističnog ažuriranja, čime poboljšava performanse i korisničko iskustvo. ***Axios*** je korišten kao alat za izvođenje samih API poziva. Ovaj HTTP (engl. *Hypertext Transfer Protocol*) klijent omogućuje jednostavan i fleksibilan način za slanje zahtjeva i obradu odgovora, s podrškom za presretanje zahtjeva, prilagođavanje zaglavlja te rukovanje pogreškama.

Aplikacija je organizirana u modularnu i jasno definiranu arhitekturu, što omogućava lakoću održavanja i jednostavno snalaženje unutar projekta. Prva razina je podijeljena na sljedeće direktorije:

- `common` – mapa sadrži zajedničke resurse koje koriste različiti dijelovi aplikacije
- `configs` – sadrži putanju za pozivanje poslužitelja, te popis javnih i privatnih stranica aplikacije
- `consts` – sadrži konstante za razne *stringove*, kao što su identifikatori poziva za *React Query* i tabovi za navigaciju.
- `modules` – najveći dio aplikacije organiziran je u module, od kojih svaki pokriva specifičnu funkcionalnost aplikacije. Moduli uključuju:
    - `account` – prijava i registracija
    - `candidate` – modul za upravljanje računom u ulozi osobe
    - `company` – modul za upravljanje računom u ulozi tvrtke
    - `jobs` – modul za izlist i prijavu na poslove
    - `messages` – modul za slanje i primanje poruka između osoba i tvrtki
    - `person` – modul za izlist i pretragu osoba
    - `stats` – modul za statistiku
- `routes` – ovdje se upravlja rutama aplikacije. Koristi se za definiranje putanja kroz koje se korisnici kreću, npr. javne i privatne rute.
- `services` - sadrži logiku za dohvat podataka i upravljanje sesijama i

autentifikacijom.

- `theme` - ova mapa sadrži tematske datoteke koje definiraju izgled aplikacije, uključujući boje, tipografiju i druge stilove korisničkog sučelja.

Svaki modul podijeljen je na sljedeće podmodule:

- Modul `api` sadrži HTTP pozive korištenjem Axiosa, koji omogućuju komunikaciju s vanjskim API-jem. Ovdje se nalaze funkcije poput `getEmployer` i `getProfile`, koje su centralizirane za jednostavno dohvaćanje podataka o poslodavcima i kandidatima.
- Modul `components` obuhvaća sve komponente *React* koje čine korisničko sučelje.
- Modul `hooks` sadrži hookove u kojima se koristi *React Query*, poput `useMutationEmployer`. Ovi hookovi olakšavaju asinkrono dohvaćanje i manipulaciju podacima, upravljajući stanjima učitavanja i pogreškama.
- Modul `models` koristi *TypeScript* za definiranje tipova podataka. Ovdje su definirani modeli kao što su `Employer`, `Job` i `Candidate`, što osigurava tipizaciju i smanjuje mogućnost pogrešaka.
- Modul `routes` sadrži implementaciju i strukturu pojedinačnih stranica. Određuje glavni *layout*, poziva *hookove* i sadržaj stranice.

2.4. Arhitektura poslužiteljske aplikacije

Poslužiteljski dio aplikacije razvijen je u programskom jeziku C# koristeći razvojni okvir *.NET Core*, što omogućava visokoučinkovit rad i lakoću integracije s različitim tehnologijama za izradu korisničkog sučelja. Arhitektura se temelji na **slojevitoj arhitekturi** (engl. *layered architecture*), koja olakšava organizaciju kôda, testiranje i održavanje aplikacije.

Kontroler

Kontroleri su odgovorni za upravljanje HTTP zahtjevima i pružanje odgovora klijentima. Oni primaju zahtjeve od klijenta, obrađuju ih uz pomoć servisa i vraćaju odgovarajuće rezultate. Svaki kontroler povezan je s određenom funkcionalnošću aplikacije, poput korisničkih računa, poslova ili poruka. Kontroleri pružaju jednostavan način za

definiranje API ruta, što poboljšava čitljivost i održavanje kôda.

Osim toga, kontroleri upravljaju osnovnom autorizacijskom logikom, osiguravajući da samo ovlašteni korisnici mogu pristupiti određenim resursima. Oni također obrađuju iznimke i pružaju odgovarajuće HTTP status kôdove u slučaju pogrešaka.

Servis

Servisni sloj sadrži poslovnu logiku aplikacije i djeluje kao posrednik između kontrolera i repozitorija, osiguravajući odvajanje odgovornosti. Servisi upravljaju validacijom podataka, obradom zahtjeva i obavljanjem kompleksnijih operacija koje mogu uključivati više repozitorija.

Ovaj sloj omogućuje lako testiranje, jer se poslovna logika može testirati neovisno o kontrolerima ili repozitorijima. Servisi također koriste asinkroni pristup kako bi poboljšali performanse i korisničko iskustvo, omogućujući obradu više zahtjeva u isto vrijeme.

Repozitorij

Repozitorij je sloj koji se bavi interakcijom s bazom podataka. Koristi Entity razvojni okvir, ORM (engl. *Object-Relational Mapping*) alat koji olakšava rad s podacima i upravljanje bazom. Repozitorij omogućuje izvođenje CRUD (engl. *Create-Read-Update-Delete*) operacija i implementaciju složene logike za dohvat i manipulaciju podacima. Svaki repozitorij vezan je uz određeni entitet, kao što su korisnici, poslovi ili poruke.

U ovom sloju, ***DbContext*** predstavlja ključnu komponentu razvojnog okvira *Entity*. Predstavlja sesiju s bazom podataka i omogućava pristup i upravljanje entitetima, a koristeći *DbContext*, programeri mogu provoditi upite, dodavati, ažurirati ili brisati entitete te obavljati transakcije. Svaki repozitorij koristi instancu *DbContexta* za interakciju s podacima u bazi.

Repozitoriji centraliziraju pristup podacima i omogućuju jednostavnu promjenu načina pohrane podataka bez utjecaja na ostatak aplikacije. Primjerice, ukoliko se odluči promijeniti bazu podataka, dovoljna je promjena samo u repozitoriju, dok kontroleri i servisi ostaju nepromijenjeni.

Autentifikacija

Za autentifikaciju korisnika koristi se osnovna autentifikacija (engl. *Basic*

Authentication). Ova metoda jednostavno kodira korisničko ime i lozinku u *Base64* format, što omogućava slanje korisničkog imena i lozinke unutar HTTP zaglavlja. Iako je osnovna autentifikacija jednostavna i brza metoda za implementaciju, važno je osigurati da se podaci šalju preko HTTPS-a (engl. *HyperText Transfer Protocol Secure*) kako bi se zaštitili osjetljivi korisnički podaci od neovlaštenog pristupa.

3. KORIŠTENE TEHNOLOGIJE I ALATI

3.1. HTML i CSS

Osnovni gradivni element svake web-stranice je HTML (engl. *HyperText Markup Language*) kôd. HTML je jezik za označavanje kojim se određuje struktura, sadržaj i funkcija nekog HTML-dokumenta (web-stranice). Pomoću HTML-a određuju se važni elementi svakog HTML-dokumenta kao što su npr. naslov, odlomak, slika, poveznica/veza (engl. *hyperlink*) i sl. Dakle, HTML je jezik za označavanje pomoću kojega se može odrediti struktura elemenata unutar HTML-dokumenta. Aktualna inačica HTML-a zove se HTML5 i podržana je u svim novijim internetskim preglednicima.

Osim strukture elemenata u dokumentu definirane HTML-om, CSS omogućava da se ti elementi stilski/grafički urede. Na taj se način web-pregledniku daje do znanja kako će stranica izgledati prilikom prikaza. Trenutačna verzija CSS-a je CSS 3, koji je donio brojne novitete i poboljšanja. Ta verzija još nije konačna, nego se i dalje neprestano razvija i nadograđuje [2].

3.2. JavaScript i TypeScript

JavaScript je jednostavan, interpretiran programski jezik namijenjen ponajprije razvoju interaktivnih HTML-stranica. Jezgra *JavaScripta* uključena je u većinu današnjih preglednika (*Internet Explorer, Google Chrome, Mozilla Firefox, Opera, Safari* i drugi).

JavaScript omogućuje izvršavanje određenih radnji u inače statičnim HTML-dokumentima, npr. interakciju s korisnikom, promjenu svojstava preglednikova prozora ili dinamičko stvaranje HTML-sadržaja. *JavaScript* nije pojednostavljena inačica programskog jezika *Java*. Povezuje ih jedino slična sintaksa i to što se koriste za izvršavanje određenih radnji unutar preglednika [3].

TypeScript je nadskup *JavaScripta* [4] koji uvodi statičko tipiziranje, omogućujući programerima definiranje tipova podataka za varijable, parametre funkcija i povratne vrijednosti. Ova značajka smanjuje mogućnost pogrešaka, jer se nepravilno definirani tipovi otkrivaju tijekom faze kompilacije, umjesto tijekom izvođenja programa.

Osim toga, *TypeScript* podržava objektno orijentirano programiranje, pružajući podršku za klase, sučelja i druge napredne značajke. Ovo olakšava organizaciju i održavanje većih kodnih baza, čineći razvoj složenih aplikacija učinkovitijim.

Budući da se *TypeScript* prevodi u *JavaScript*, može se koristiti u bilo kojem okruženju gdje JavaScript radi, zadržavajući prednosti dodatne provjere tipova tijekom razvoja. Ova kompatibilnost omogućuje postupnu integraciju *TypeScripta* u postojeće *JavaScript* projekte.

3.3. .NET

.NET je popularna razvojna platforma koju je razvio Microsoft i koja omogućava izradu aplikacija za različite uređaje, sustave i platforme. *.NET* je otvorenog kôda i omogućava programiranje na više jezika, uključujući *C#, F# i VB.NET*. Jedan od najvažnijih okvira unutar *.NET* platforme je *ASP.NET Core,* koji omogućava razvoj web aplikacija i API-ja. *ASP.NET Core* je prilagođen za rad u cloud okruženjima, što ga čini idealnim izborom za moderne web aplikacije i mikroservise.

3.4. Alati u klijentskoj aplikaciji

React je knjižnica koju je razvila i održava tvrtka *Meta* (prethodno *Facebook*), a koristi se za izgradnju korisničkih sučelja (engl. *User Interface*) za web aplikacije. *React* omogućava programerima stvaranje sučelja pomoću komponenata, što su samostalni dijelovi korisničkog sučelja koji sadrže i logiku i prikaz. Komponente su modularne i mogu se ponovno koristiti, što olakšava razvoj, održavanje i skaliranje velikih aplikacija.

Jedna od ključnih značajki *Reacta* je virtualni DOM (engl. *Document Object Model*), koji omogućava efikasno upravljanje promjenama u korisničkom sučelju. Kada se stanje aplikacije promijeni, *React* ažurira virtualni DOM, uspoređuje ga sa stvarnim DOM-om, te efikasno primjenjuje samo one promjene koje su potrebne. To značajno poboljšava performanse aplikacija, posebno u usporedbi s tradicionalnim metodama manipulacije DOM-om u *JavaScriptu*.

React koristi JSX (JavaScript XML, engl. *Extensible Markup Language*), sintaksu koja omogućava pisanje HTML-a unutar *JavaScript* kôda. JSX je čitljiviji od običnog *JavaScripta* i omogućava lakše strukturiranje i preglednost kôda. Primjerice, umjesto uobičajene funkcije koja stvara HTML element, u *Reactu* možemo koristiti JSX sintaksu koja izgleda kao HTML, ali je ugrađena unutar *JavaScript* funkcije.

React također podržava tzv. *hookove*, što su funkcije koje omogućavaju korištenje stanja i drugih značajki *Reacta* u funkcionalnim komponentama. Najvažniji *hookovi* su `useState`, koji omogućava rad sa stanjem i `useEffect`, koji omogućava pokretanje

kôda kao odgovor na promjene u komponenti ili njezinom stanju. *Hookovi* su unijeli značajne promjene u *React* jer omogućavaju pisanje jednostavnijih i modularnijih komponenti bez potrebe za složenim klasnim komponentama. Osim osnovnih *hookova*, postoje i brojni drugi, poput `useContext` za upravljanje globalnim stanjem, te *hookovi* koje programeri mogu sami kreirati kako bi ponovno koristili logiku između različitih komponenti.

Mantine UI je *React* knjižnica za izgradnju korisničkih sučelja koja nudi bogat skup komponenti spremnih za korištenje, s naglaskom na jednostavnost, fleksibilnost i responzivnost. *Mantine* je stvoren kako bi ubrzao proces izrade sučelja i smanjio potrebu za pisanjem vlastitog CSS-a i prilagođavanja komponenti, pružajući već unaprijed dizajnirane komponente koje se lako prilagođavaju različitim stilovima i potrebama aplikacije.

Jedna od najistaknutijih značajki *Mantine UI* knjižnice je širok izbor komponenti koji uključuje osnovne elemente kao što su tipke (engl. *button*), obrasci (engl. *form*), modalni prozori (engl. *modal*), kartice (engl. *card*) i složenije komponente poput kalendara, tablica, notifikacija i alata za oblikovanje teksta. Sve komponente dolaze s potpunim skupom opcija za prilagodbu, a knjižnica omogućava detaljno definiranje stilova direktno putem *JavaScript* objekata, čime se eliminira potreba za korištenjem vanjskih stilskih datoteka.

React Query je popularna knjižnica za upravljanje podacima u *React* aplikacijama, posebno za dohvaćanje, predmemoriranje (engl. *caching*) i sinkronizaciju podataka s poslužitelja. Ova knjižnica rješava jedan od najčešćih izazova u razvoju web aplikacija: kako efikasno upravljati podacima i omogućiti responzivno korisničko iskustvo dok podaci dolaze s poslužitelja. Umjesto ručnog pisanja logike za dohvaćanje podataka, predmemoriranje, ponovno dohvaćanje i ažuriranje sučelja, *React Query* omogućava jednostavan i dosljedan način za sve te zadatke.

```
export const useQueryJobsList = ({ params, pageNo, pageSize }: IProps) => {
  const { data, error, isLoading } = useQuery([QUERY_KEYS.JOBS_LIST, params],
    () => getJobsList({ pageNo, pageSize, params })
  );
  return { data, error, isLoading };
};
```

Ispis 1: Primjer dohvata uz pomoć *hooka* `useQuery`

Jedna od osnovnih funkcija *React Queryja* je *hook* `useQuery`, prikazan na ispisu 1, koji služi za dohvaćanje podataka s poslužitelja. Ovaj *hook* omogućava definiranje „ključ-vrijednost“ parova koji identificiraju specifične podatke. Kada se podaci dohvate, *React Query* ih automatski predmemorira pod ovim ključem, što znači da će ih aplikacija ponovno koristiti bez potrebe za dodatnim pozivima API-ja. *Hook* također nudi opcije za različite načine rada, poput automatskog ponovnog dohvaćanja podataka nakon što se korisnik vrati na aplikaciju, periodičnog osvježavanja ili ponovnog pokušaja dohvaćanja u slučaju pogreške.

Osim `useQuery`, *React Query* pruža *hook* `useMutation` za upravljanje promjenama podataka, kao što su kreiranje, ažuriranje ili brisanje podataka na poslužitelju. Mutacije omogućavaju specifičnu logiku za slučajeve kada se podaci mijenjaju, uključujući opcije za optimističke promjene (koje odmah ažuriraju sučelje prije nego što je odgovor s poslužitelja potvrđen), poništavanje promjena u slučaju pogreške i automatsko ponovno dohvaćanje povezanih podataka.

3.5. Alati u poslužiteljskoj aplikaciji

Entity Framework Core (EF Core) je ORM alat za rad s bazama podataka unutar *.NET* okruženja. Njegova uloga je omogućiti programerima rad s podacima koristeći *C#* klase, umjesto pisanjem upita SQL. *EF Core* preuzima zadatak prevođenja *C#* izraza u upite SQL, dohvaćanja podataka iz baze, te njihovog vraćanja kao objekata u aplikaciji. Time *EF Core* omogućava jednostavnije i efikasnije rukovanje podacima, a podržava različite baze podataka, uključujući *SQL Server, MySQL, PostgreSQL* i *SQLite*. *EF Core* također podržava LINQ (engl. *Language Integrated Query*), mehanizam koji omogućava pisanje upita

koristeći sintaksu sličnu *C#*-u.

SQL Server ili skraćeno MS SQL, je relacijska baza podataka koju je razvila kompanija *Microsoft*. *SQL Server* koristi SQL (*Structured Query Language*) kao jezik za rad s podacima i podržava različite napredne značajke, kao što su pohranjene procedure, indeksiranje, *triggere* i transakcije. *MS SQL* je visoko skalabilan i osigurava visoku razinu sigurnosti podataka, čime je popularan izbor za velike poslovne aplikacije i sustave koji zahtijevaju sigurno upravljanje podacima.

Jedna od ključnih prednosti korištenja *.NET* platforme u kombinaciji s *Entity Framework Core* i *MS SQL Serverom* je integracija ovih tehnologija, koja omogućava jednostavnu i efikasnu izradu aplikacija. Kad se koristi *EF Core* s *MS SQL Serverom*, programeri mogu koristiti *C#* za definiranje modela podataka, a zatim koristiti migracije kako bi jednostavno ažurirali strukturu baze podataka prema promjenama u modelima. Migracije omogućavaju postupno uvođenje promjena u bazu podataka bez ručnog pisanja SQL skripti, čime se smanjuje mogućnost pogrešaka i povećava produktivnost.

Pomoću naredbi `Add-Migration` i `Update-Database`, programeri mogu kreirati migracijske skripte koje ažuriraju strukturu baze podataka na temelju promjena u modelima podataka. Svaka migracija predstavlja promjenu u bazi podataka, a *EF Core* upravlja redoslijedom izvršavanja migracija, osiguravajući da se sve promjene primjenjuju ispravnim redom. To omogućava dosljedno ažuriranje baze podataka na razvojnom i produkcijskom okruženju.

EF Core omogućava rad s bazom podataka na dva osnovna načina: *Code-First* i *Database-First*. Kod *Code-First* pristupa, programer prvo definira modele kao klase jezika *C#*, a zatim *EF Core* generira bazu podataka na temelju tih modela. Ovaj pristup omogućava veću kontrolu nad modelima podataka i omogućava brze promjene u bazi kroz migracije. S druge strane, *Database-First* pristup počinje s postojećom bazom podataka, a *EF Core* generira modele na temelju strukture te baze, što je korisno u slučajevima kada već postoji baza podataka koju treba povezati s *.NET* aplikacijom.

LINQ omogućava pisanje upita na način koji je prirodan za programere u *.NET* okruženju, koristeći *C#* sintaksu za dohvaćanje, filtriranje, grupiranje i sortiranje podataka. Primjerice, umjesto pisanja upita SQL, programer može koristiti izraz *LINQ* za dohvaćanje podataka iz baze koristeći *C#* metode poput `Where`, `Select`, `OrderBy` i `GroupBy`. Upiti

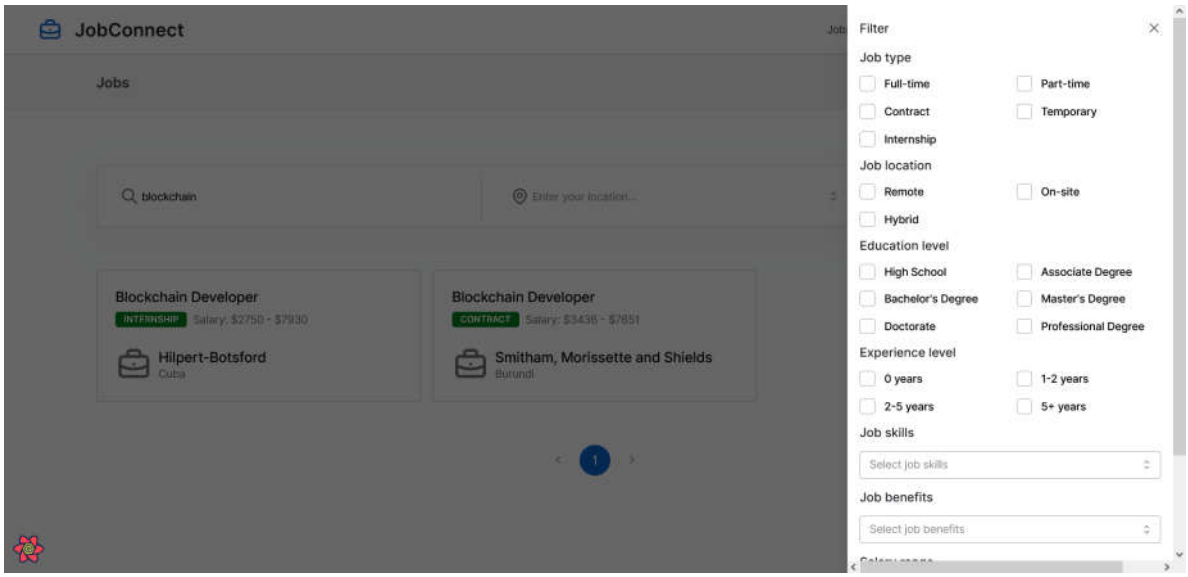
LINQ su tipno sigurni, što znači da kompajler može provjeriti ispravnost upita u fazi kompilacije, čime se smanjuje broj pogrešaka koje bi se mogle pojaviti tijekom izvršavanja.

4. IMPLEMENTACIJA FUNKCIONALNOSTI

U ovom poglavlju napravljen je osvrt na glavne funkcionalnosti aplikacije *JobConnect*. Svaka od tih funkcionalnosti je opisana te su prikazani neki od dijelova samog programskog kôda u kojem je ona izvedena.

Funkcionalnosti aplikacije su podijeljene prema tipu korisnika u tri glavna segmenta. Neprijavljeni korisnici mogu samo pretraživati dostupne poslove, tvrtke i kandidate. Registrirani korisnici spadaju u jednu od dvije kategorije: poslodavci koji objavljuju poslove ili kandidati koji traže zaposlenje. Svaka od ovih uloga nudi prilagođene funkcionalnosti specifične za perspektivu korisnika.

Na početnoj stranici aplikacije nalazi se izlist aktivnih oglasa za posao, uz mogućnost pretrage i filtriranja po parametrima koji uključuju lokaciju, tip posla, način rada, traženu razinu edukacije ili iskustva, te po traženim vještinama ili benefitima koji pozicija nudi. Slika 1 prikazuje bočnu traku s filtrima.



Slika 1: Filtri ponude oglasa

Odabrani filtri se putem *URL*-a u pregledniku šalju poslužitelju, poziva se metoda `GetJobsPage`, prikazana na ispisu 2, koja poziva servis u kojem se na temelju proslijeđenih parametara gradi upit u obliku logičkog izraza (engl. *expression*) koji *Entity Framework* ORM lako može prevesti u upit SQL na bazu podataka.

```
public async Task<IPage<JobListResponse>> GetJobsPage(IPaging paging, JobsFilterQuery
query)
        {
            // Build the predicate dynamically based on the provided filters
            Expression<Func<Job, bool>> predicate = x => true;
            if (!string.IsNullOrEmpty(query.Keyword))
            {
                predicate = predicate.And(x => x.Title.Contains(query.Keyword) ||
x.Description.Contains(query.Keyword));
            }
            if (!string.IsNullOrEmpty(query.Location))
            {
                predicate = predicate.And(x => x.Location.Contains(query.Location));
            }
            if (query.JobType != null && query.JobType.Any())
            {
                predicate = predicate.And(x => query.JobType.Contains(x.JobTypeId));
            }
          .....
            predicate = predicate.And(x => x.ExpireDate >= DateTime.Now);
            predicate = predicate.And(x => x.IsActive);
            predicate = predicate.And(x => x.IsDeleted == false);
            // Get the paginated result from the repository
            var jobs = await jobRepository.GetPage(paging, predicate, orderBy);
```

Ispis 2: Primjer filtriranja prema traženoj ključnoj riječi

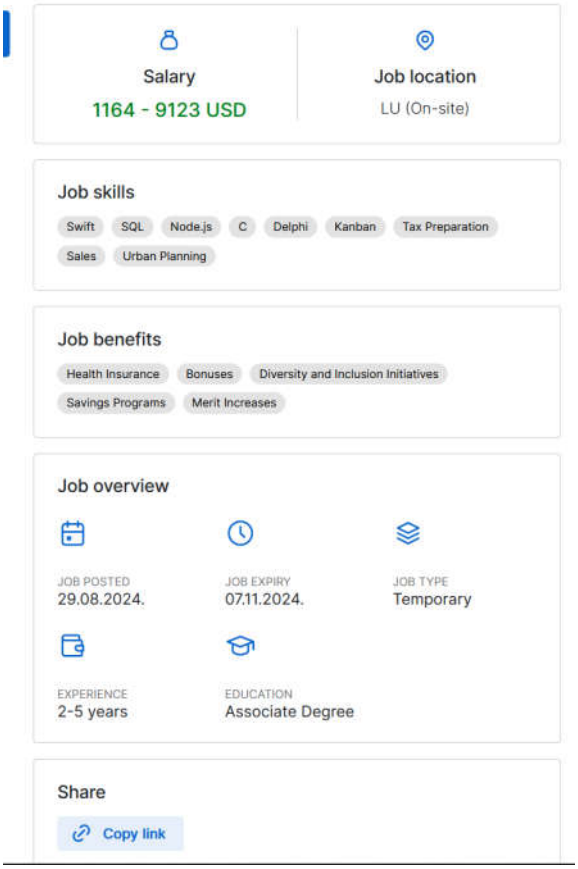
Nakon što je upit na bazu izvršen, podaci se oblikuju na način koji ne uključuje sve detalje i pogodan je za prikaz liste podijeljene na stranice. Podaci su prikazani na ispisu 3.

```
{
  "totalItems": 2,
  "pageSize": 20,
  "currentPage": 1,
  "numberOfPages": 1,
  "items": [
    {
      "id": "a7e1b2c9-56ea-4cbf-4095-08dcc83903db",
      "title": "Blockchain Developer",
      "description": "Consequatur est modi ipsum molestiae quis quia vel eos et. Qui velit corrupti suscipit in molestiae deserunt mollitia numquam. Quia excepturi distinctio reiciendis. Ipsam et velit consequatur et sit dolores eos aut voluptatem.",
      "jobType": "Internship",
      "minSalary": 2750,
      "maxSalary": 7930,
      "location": "Cuba",
      "employer": "Hilpert-Botsford",
      "photoId": null
    }
  ]
}
```


Ispis 3: Odgovor poslužitelja u obliku prilagođenom za paginaciju

4.1. Detalji i prijava na oglas

Klikom na oglas, korisnika se preusmjerava na stranicu detalja oglasa. Stranica s detaljima dijeli se na tri dijela. Naslovna traka, opis pozicije i bočni dio koji sadrži informacije kao što su lokacija, raspon plaće, tražene vještine i benefiti. Bočna traka prikazana je na slici 2.



Slika 2: Bočna traka s informacijama o poslu

Važan dio kôda za istaknuti, prikazan na ispisu 4, je uključivanje svih važnih informacija u glavni entitet `Job`, te filtriranje poslova koji su izbrisani.

```
            modelBuilder.Entity<Job>().HasQueryFilter(j => !j.IsDeleted);

            modelBuilder.Entity<Job>().Navigation(j => j.JobType).AutoInclude();

            modelBuilder.Entity<Job>().Navigation(j                                    =>
j.JobLocationType).AutoInclude();

            modelBuilder.Entity<Job>().Navigation(j                                    =>
j.EducationLevel).AutoInclude();

            modelBuilder.Entity<Job>().Navigation(j                                    =>
j.ExperienceLevel).AutoInclude();

            modelBuilder.Entity<Job>().Navigation(j => j.Employer).AutoInclude();

            modelBuilder.Entity<Job>().Navigation(j => j.JobSkills).AutoInclude();

            modelBuilder.Entity<Job>().Navigation(j                                    =>
j.JobBenefits).AutoInclude();
```

Ispis 4: Automatsko uključivanje povezanih entiteta u glavni entitet

U naslovnoj traci, osim naslova oglasa, nalazi se tipka za prijavu. Pritiskom na „*Apply*“, otvara se modalni prozor u kojem korisnik odabire jedan od prethodno unesenih životopisa, te polje za motivacijsko pismo. Modalni prozor je definiran u *React* komponenti `JobApplyModal`, prikazanoj u ispisu 5, koja sadrži poziv za dohvat dostupnih životopisa, metodu za slanje prijave, validaciju obaveznih polja, te samu *render* funkciju koja definira strukturu i sadržaj komponente.

```
export const JobApplyModal = ({ jobId, opened, close }: IProps) => {
  const { data } = useQueryCandidateResumes();
  const { mutate: apply } = useMutationApplyJob();
  const form = useForm<IFormValues>({
    initialValues: { resumeId: undefined, coverLetter: "" },
    validate: {
      resumeId: (value) => (value ? undefined : "Resume is required"),
      coverLetter: (value) =>
        value.length > 0 ? undefined : "Cover letter is required",
    },
  });

  const formOnSubmit = form.onSubmit((values) => {
    apply(
      { id: jobId, payload: values },
      {
        onSuccess: () => {
          close();
        },
      }
    );
  });
```

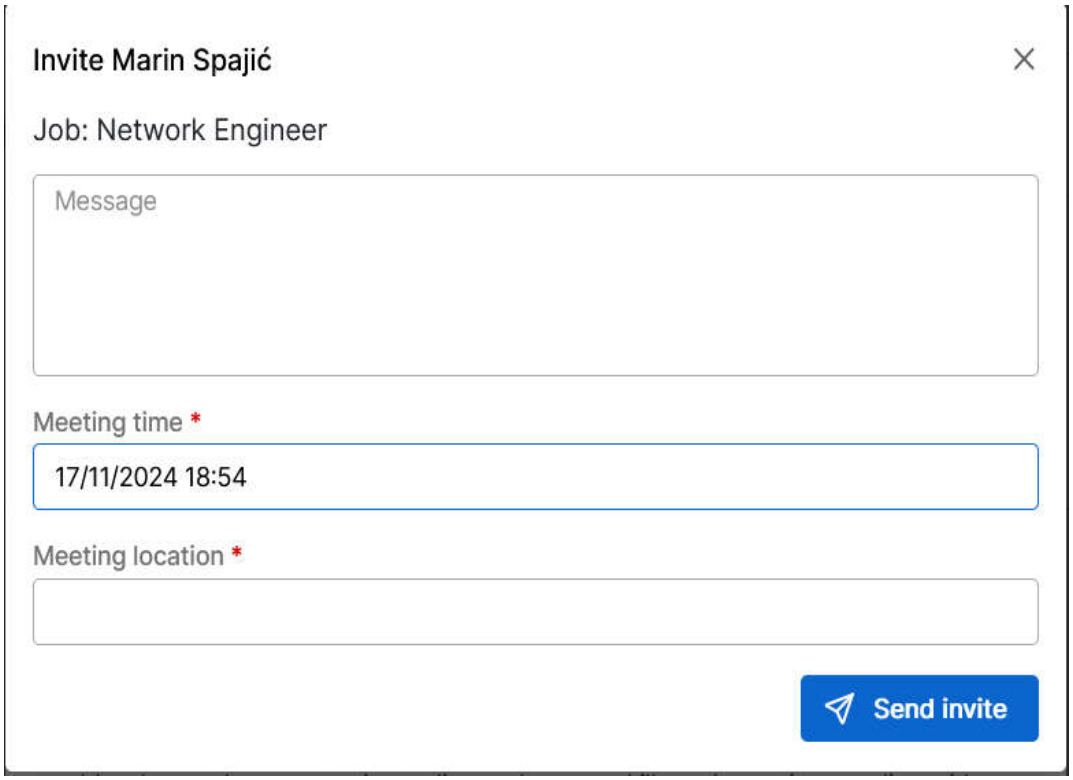
Ispis 5: Dio koda modalnog prozora za prijavu na oglas

Slanje prijave uključuje provjeru postoji li već prijava za isti oglas od istog korisnika, kreiranje novog objekta `JobApplication` koji se sprema u istoimenu tablicu, provjeru te brisanje prethodnih pozivnica za prijavu ukoliko postoje.

Zaprimljene prijave predstavnik tvrtke vidi na svom kontrolnom sučelju. Svaku od prijava može pregledati, preuzeti priloženi životopis, staviti u određeni status, odbaciti ili zakazati sastanak s kandidatom.

4.2. Sastanci

Ako tvrtka odluči pozitivno odgovoriti na poslanu prijavu, ima opciju zakazivanja sastanka. Prilikom zakazivanja sastanka, unosi se poruka i datum sastanka, te se provjerava se je li odabrani termin slobodan, odnosno ima li preklapanja s postojećim terminima kod poslodavca ili kandidata. Metoda za provjeru preklapanja prikazana je na ispisu 6. Ako preklapanje nije pronađeno, unosi se novi zapis u tablicu `Meetings`. Slika 3 prikazuje modalni prozor za slanje pozivnice kandidatu.

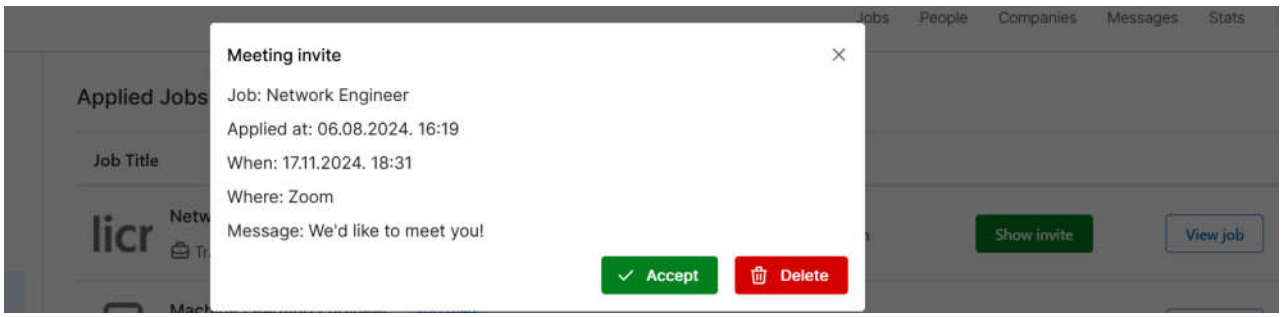


Slika 3: Modalni prozor za slanje pozivnice

```
        public async Task<bool> CheckForOverlap(MeetingRequest meetingRequest,
Guid currentEmployerId)
        {


            var meetingStart = meetingRequest.MeetingTime;
            var meetingEnd = meetingRequest.MeetingTime.AddHours(1);


            return await context.Meetings
                .Include(x => x.JobApplication)
                .ThenInclude(x => x.Candidate)
                .Where(m           =>           m.MeetingTime.Date           ==
meetingRequest.MeetingTime.Date &&
                           (m.JobApplication.Job.EmployerId                 ==
currentEmployerId ||
                            m.JobApplication.CandidateId                    ==
m.JobApplication.CandidateId) &&
                           m.MeetingTime        <        meetingEnd        &&
m.MeetingTime.AddHours(1) > meetingStart)
                .AnyAsync();
        }
```

Ispis 6: Provjera preklapanja sastanaka

Kandidat kroz svoju nadzornu ploču vidi pozivnicu i može je prihvatiti ili odbiti, što se zapisuje u polje `IsMeetingAccepted` u tablici `Meetings`. Slika 4 prikazuje pregled primljene pozivnice za sastanak.

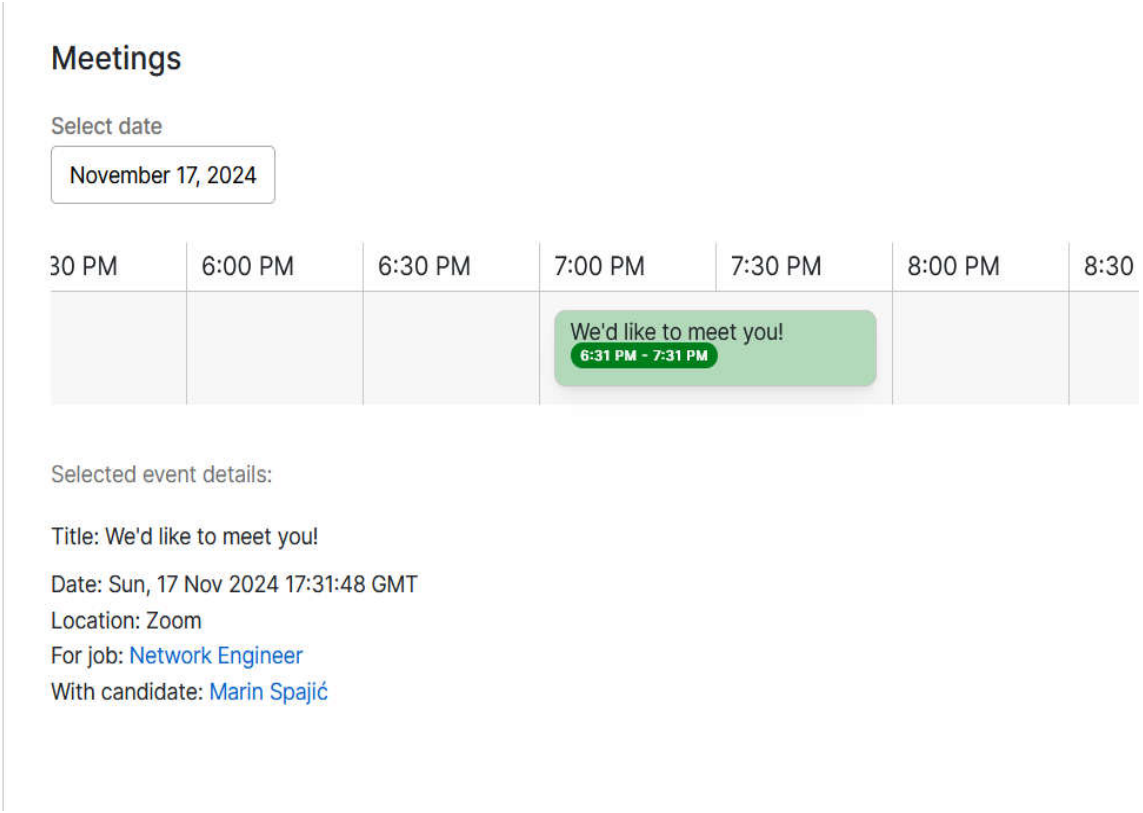


Slika 4: Modalni prozor pozivnice za sastanak

Poslodavac klikom na „*Meetings*“ vidi kalendar u kojem se prikazuju svi kandidati s kojima ima razgovor na odabrani dan, prikazano u slici 5, uz pripadajući kôd za dohvat sastanaka na ispisu 7.

```
public async Task<List<Meeting>> GetMeetingsByDate(DateTime date)
{
    return await context.Meetings
        .Include(x => x.JobApplication)
        .ThenInclude(x => x.Candidate)
        .Where(x => x.MeetingTime.Date == date.Date)
        .ToListAsync();
}
```

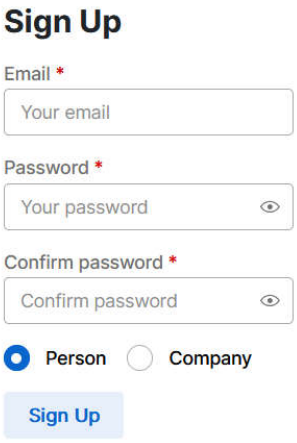
Ispis 7: Dohvat sastanaka na odabrani dan



Slika 5: Kalendar zakazanih sastanaka

4.3. Registracija korisnika

Za pristup većini funkcionalnosti u aplikaciji *JobConnect*, korisnici moraju biti registrirani, te imaju jednu od dvije uloge – poslodavac ili kandidat. Proces registracije započinje klikom na tipku „*Sign Up*“, koji korisnika vodi na stranicu za unos elektroničke pošte i lozinke. Forma za početak registracije prikazana je u slici 6.



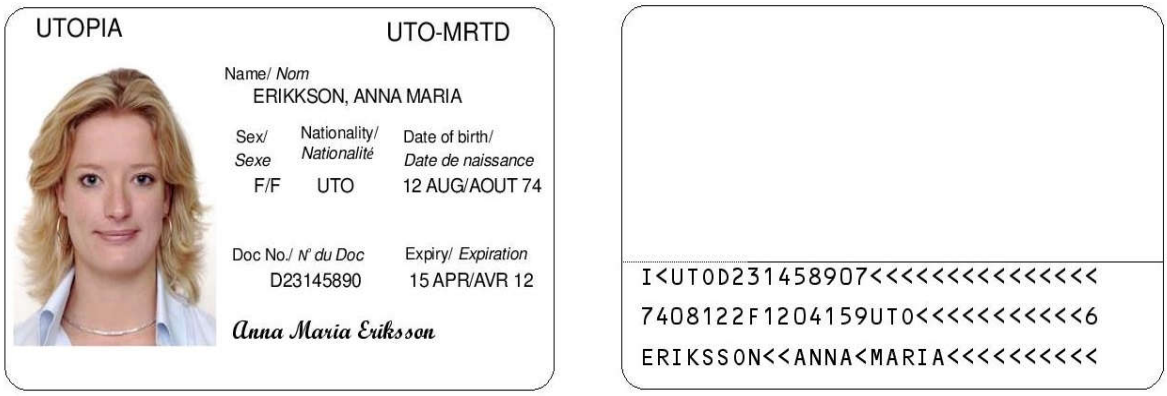
Slika 6: Forma za početak registracije

Nakon odabira željene uloge šalje se POST zahtjev na poslužitelj i poziva se metoda `SignUp` unutar `AuthControllera`, prikazana na ispisu 8. Bitno je označiti metodu atributom `AllowAnonymous`, jer u trenutku registracije korisnik nije autoriziran. Na samom početku prijave, izvršava se provjera postoji li već korisnik s istom adresom elektroničke pošte. Ako je adresa elektroničke pošte slobodna, kreira se objekt `User`, te pripadajući prazan profil ovisno o odabranoj ulozi.

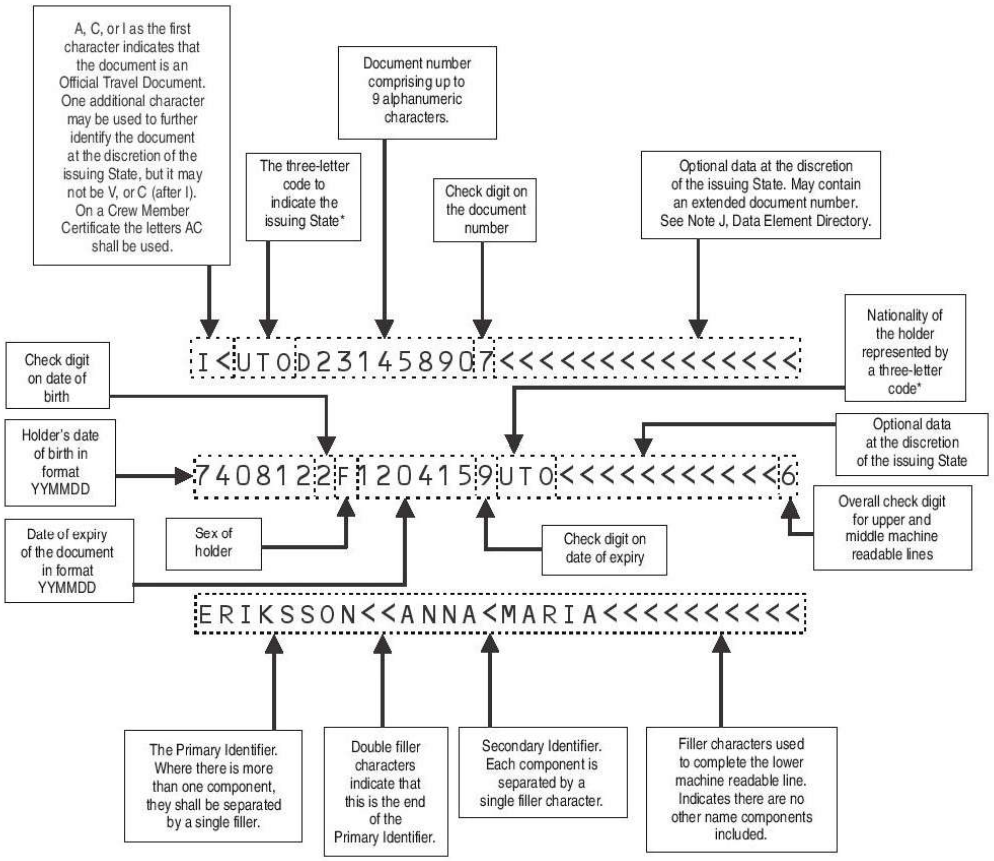
```
[AllowAnonymous]
[HttpPost]
[Route("register")]
public async Task<IActionResult> Register(Register register)
{
    if (await authService.ExistsWithEmail(register.Email))
    {
        return BadRequest("User with that e-mail already exists");
    }

    User user = await authService.Register(register);
    if (user.UserType == UserType.Company)
    {
        Employer profile = await companyService.CreateNewCompany(user);
        return Ok(profile);
    }
    else if (user.UserType == UserType.Person)
    {
        Candidate profile = await candidateService.CreateNewPerson(user);
        return Ok(profile);
    }
```

Ispis 8: Kôd za registraciju novog korisnika

Idući korak registracije sastoji se od unosa osnovnih podataka. U slučaju tvrtke, obavezan je unos imena, elektroničke pošte, države, te industrije u kojoj djeluje tvrtka. U slučaju registracije osobe, uz osnovne osobne podatke, obavezna je i provjera osobne iskaznice. Potrebno je odabrati sliku koja sadrži MRZ (engl. *Machine Readable Zone*) kôd koji se nalazi na poleđini osobne iskaznice. MRZ je format za računalno očitavanje putnih dokumenata, standardiziran kod međunarodne organizacije za standardizaciju (ISO) [5]. Postoji nekoliko MRZ formata: TD1 (*Travel Document 1*) je najčešći i sadrži 3 linije znakova, dok TD2 i TD3 sadrže dvije linije znakova. Slike 7 i 8 ispod prikazuju format TD1.



Slika 7: Format TD1 na osobnoj iskaznici
(Izvor: ICAO, Doc 9303 – Machine Readable Travel Documents)



Construction of the 3-line MRZ data on a TD1 Size MROTD

Note 1.— (*) Three-letter codes are given in Doc 9303-3.
Note 2.— Dotted lines indicate data fields; these, together with arrows and comments boxes, are shown for the reader's understanding only and are not printed on the document.
Note 3.— Data is inserted into a field beginning at the first character position starting from the left. Any unused character positions shall be occupied by filler characters (<).

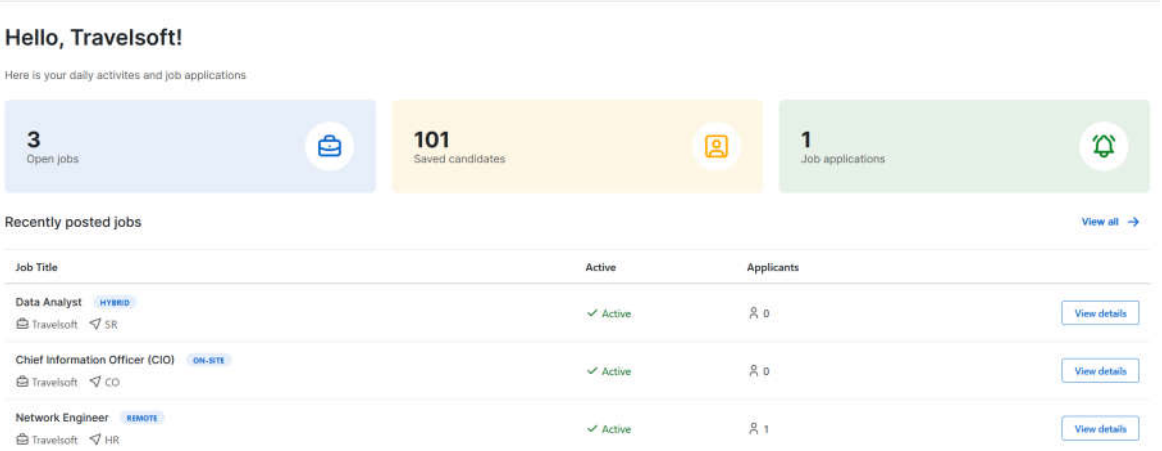
— — — — — — — —

Slika 8: Osobne informacije u TD1 formatu
(Izvor: ICAO, Doc 9303 – Machine Readable Travel Documents)

Aplikacija koristi *Tesseract* OCR (engl. *Optical Character Recognition*) knjižnicu za prepoznavanje teksta sa slike, u kombinaciji s paketom `mrz` koji sadrži *regex* definicije za pronalazak informacija unutar MRZ kôda. Kod registracije se provjerava podudara li se upisano prezime s prezimenom na osobnoj iskaznici. Po spremanju, flag `SetupCompleted` na objektu `User` se stavlja na `true`, a uneseni podaci se zapisuju u odgovarajuću tablicu `Employer` ili `Candidate` i nakon toga aplikacija preusmjerava korisnika na njegovu nadzornu ploču.

4.4. Nadzorna ploča

U aplikaciji se nalazi nadzorna ploča za kandidata i nadzorna ploča za tvrtku. Nadzorna ploča korisniku daje brzi uvid u nekoliko ključnih podataka. Tvrtka može vidjeti svojih 5 najnovije otvorenih oglasa, broj prijava i spremljene kandidate kojima se možda planira javiti. Nadzorna ploča poslodavca prikazana je na slici 9.



Slika 9: Nadzorna ploča poslodavca

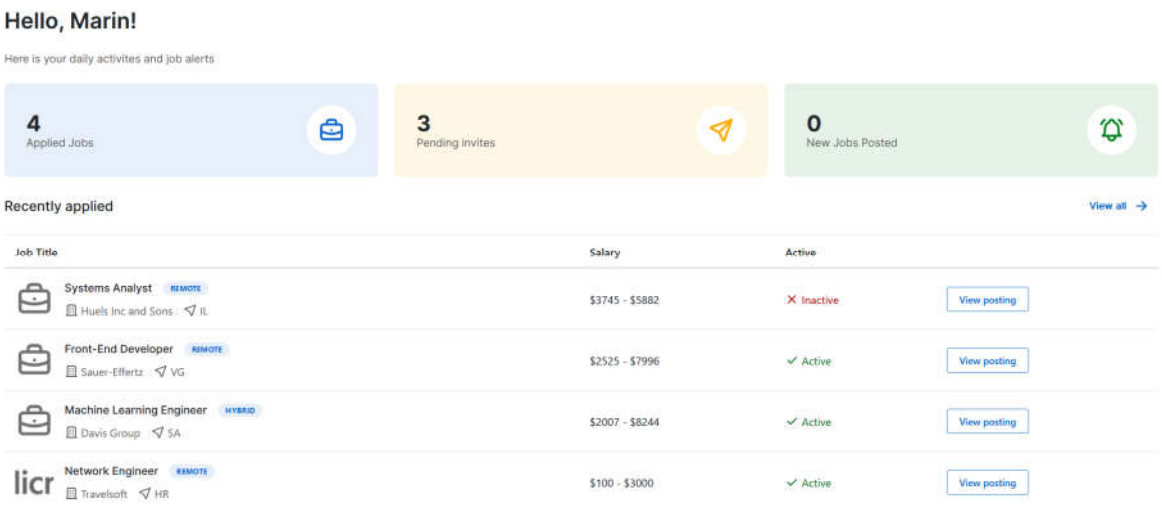
Nakon otvaranja nadzorne ploče, klijent šalje GET zahtjev prema poslužitelju. Ovisno o ulozi korisnika, poziva se metoda unutar servisa `EmployerService` ili `CandidateService`, te uz nekoliko upita na bazu prikuplja sve potrebne podatke. Potom ih šalje poslužitelju u obliku jednostavnom za prikaz. U ispisu 9 prikazan je upit za listu

nedavnih oglasa koji uključuje i broj prijava.

```
        public             async             Task<List<JobWithApplicationCount>>
GetRecentJobsByEmployerId(Guid employerId)
        {
            var jobsWithCounts = await context.Jobs
                .Where(j => j.EmployerId == employerId)
                .OrderByDescending(j => j.CreatedAt)
                .Take(5)
                .Select(job => new JobWithApplicationCount
                {
                    Id = job.Id,
                    Title = job.Title,
                    Location = job.Location,
                    CompanyName = job.Employer.Name,
                    IsActive = job.IsActive,
                    JobLocationType = job.JobLocationType.Name,
                    ApplicationCount   =   context.JobApplications.Count(ja   =>
ja.JobId == job.Id)
                })
                .ToListAsync();

            return jobsWithCounts;
        }
```

Ispis 9: Dohvat liste oglasa s brojem prijava za nadzornu ploču poslodavca

U slučaju kandidata, nadzorna ploča ima istu strukturu uz različite informacije. Kandidatu je istaknuto zadnjih 5 poslova na koje se prijavio, ukupan broj poslanih prijava, broj neodgovorenih pozivnica i broj oglasa objavljenih u zadnjih sedam dana. Nadzorna ploča kandidata prikazana je na slici 10.



Slika 10: Nadzorna ploča kandidata

Slično kao i za poslodavca, poziva se metoda `GetDashboardStats` unutar klase `CandidateService` koja dohvaća sve podatke u obliku pogodnom za prikaz. Metoda je prikazana u ispisu 10.

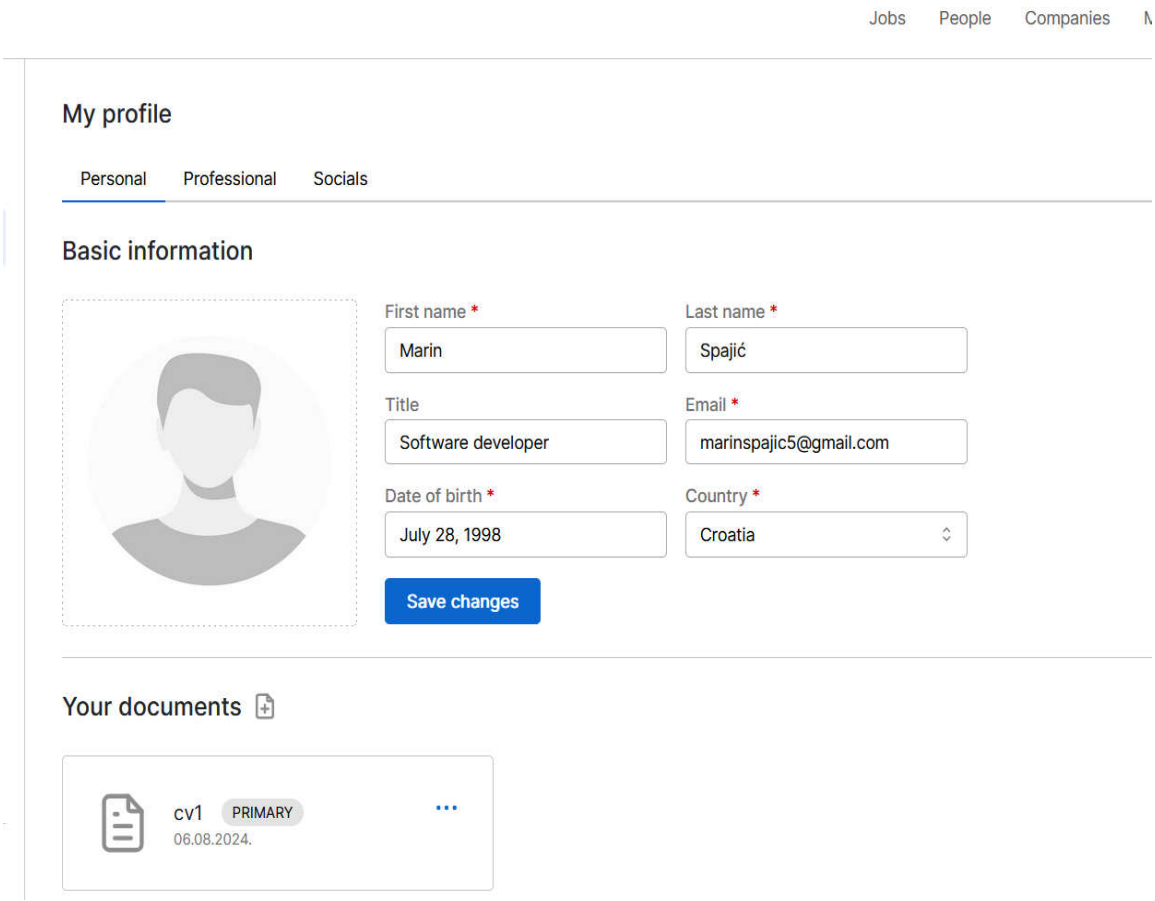
```
        public async Task<CandidateDashboardStats> GetDashboardStats()
        {
            var candidateId = identityContext.CandidateId ?? throw new
Exception("Candidate not found");

            var appliedJobs = await jobApplicationRepository.FindByCondition(x
=> x.Candidate.UserId == identityContext.UserId && !x.IsDeleted);
            var invites = await inviteRepository.FindByCondition(x =>
x.CandidateId == candidateId && !x.IsDeleted);
            var newJobs = await jobRepository.FindByCondition(x => x.CreatedAt >
DateTime.Now.AddDays(-7) && x.IsActive);
            var appliedJobsCount = appliedJobs.Count();
            var invitesCount = invites.Count();
            var newJobsCount = newJobs.Count();

            var recentlyAppliedJobs = await
jobApplicationRepository.GetRecentJobApplicationByCandidateId(candidateId);


            var stats = new CandidateDashboardStats
            {
                AppliedJobs = appliedJobsCount,
                Invites = invitesCount,
                JobAlerts = newJobsCount,
                RecentlyAppliedJobs = recentlyAppliedJobs.Select(jobApplication
=> new CandidateDashboardJobResponse
                {
                    Id = jobApplication.Job.Id,
                    Title = jobApplication.Job.Title,
                    CompanyName = jobApplication.Job.Employer.Name,
                    CompanyLogo = jobApplication.Job.Employer.Logo,
                    Location = jobApplication.Job.Location,
```

Ispis 10: Dohvat podataka nadzorne ploče posloprimca

4.5. Profil kandidata

Profil kandidata je dostupan za pregled svim registriranim korisnicima, tvrtkama i drugim kandidatima, te sadrži osnovne informacije o osobi, sliku profila, opis, povijest zaposlenja, životopis dostupan za preuzimanje, vještine i poveznice na društvene mreže ili web stranicu.

Kandidat može urediti vlastiti profil klikom na „*My profile*“ u navigacijskoj traci s lijeve strane. Na prvom tabu se uređuju osobne informacije, profilna slika i prijenos dokumenata, prikazano na slici 11.



Slika 11: Uređivanje osobnih informacija kandidata

Pritiskom na tipku „*Save changes*“, radi se PUT poziv na `UpdateBasicInfo` metodu i ažuriraju se informacije korisnika u tablici `Candidates`. U sekciji ispod nalaze se dokumenti koje korisnik prenese. Dokumenti se koriste za pohranu različitih životopisa,

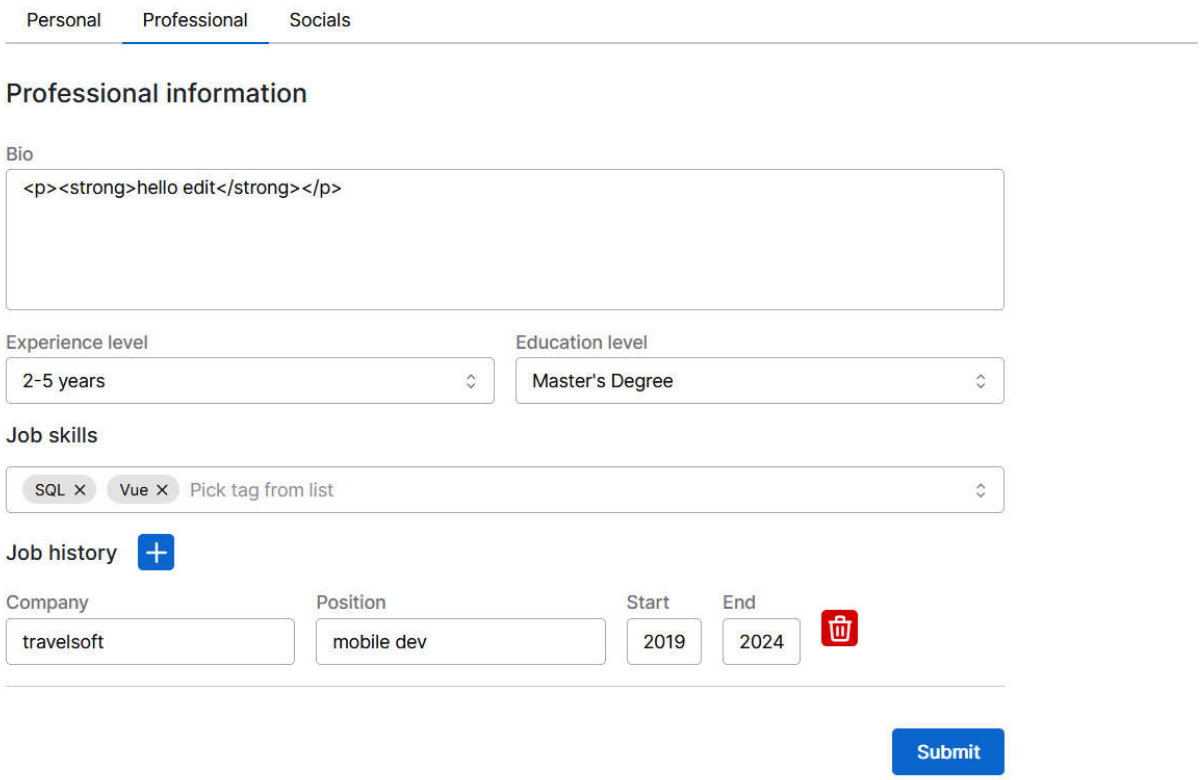
odnosno lakšu prijavu na različite pozicije. Životopis koji je odabran kao primarni prikazuje se na profilu korisnika. Svaki dokument ima zapis u tablici `Resumes`, koji sadrži naziv, datum i vezu na tablicu `Files` u kojoj se nalaze informacije o prenesenoj datoteci. Sama datoteka sprema se na disk prema određenoj putanji, u ovom slučaju u mapu `uploads` na particiji C. Ispis 11 prikazuje metodu za spremanje datoteka.

```
public async Task<Models.File> SaveFileAsync(IFormFile file, Guid userId)
        {
            var uploadsRootFolder = Path.Combine(@"C:\", "uploads");
            var uniqueFileName = $"{Guid.NewGuid()}_{file.FileName}";
            var filePath = Path.Combine(uploadsRootFolder, uniqueFileName);

            if (!Directory.Exists(uploadsRootFolder))
                Directory.CreateDirectory(uploadsRootFolder);

            using (var fileStream = new FileStream(filePath, FileMode.Create))
            {
                await file.CopyToAsync(fileStream);
            }

            var fileEntity = new Models.File
            {
                FileName = file.FileName,
                FileType = file.ContentType,
                FilePath = filePath,
                UserId = userId
            };

            var response = await fileRepository.Insert(fileEntity);

            return response;
        }
```

Ispis 11: Kôd za spremanje odabrane datoteke i evidenciju datoteke u bazu podataka

Na sljedećem tabu unutar uređivanja profila, prikazanom na slici 12, nalazi se opis odnosno biografija, razina obrazovanja i iskustva te lista vještina i prethodnih poslova kandidata. Biografija kandidata podržava HTML *tagove*, kako bi se mogli istaknuti ili

formatirati željeni dijelovi teksta. Na zadnjem tabu se nalaze poveznice na društvene mreže ukoliko ih kandidat odluči dodati.

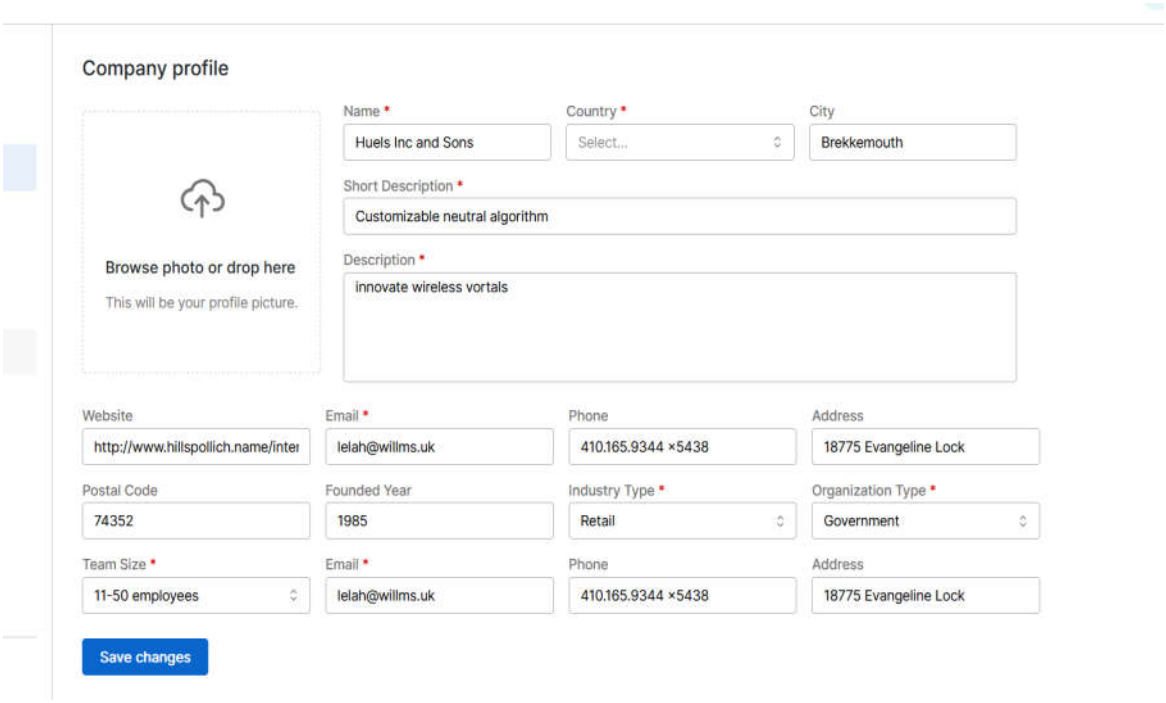


Slika 12: Uređivanje podataka o vještinama i iskustvu kandidata

4.6. Profil tvrtke

Profil tvrtke, je dostupan za pregled svim registriranim korisnicima, tvrtkama i drugim kandidatima, te sadrži opis tvrtke, tip i veličinu tvrtke, industriju kojoj tvrtka pripada te poveznice na društvene mreže tvrtke.

Predstavnik tvrtke može urediti profil klikom na „*My profile*“ u navigacijskoj traci s lijeve strane, nakon čega se otvara forma prikazana na slici 13.

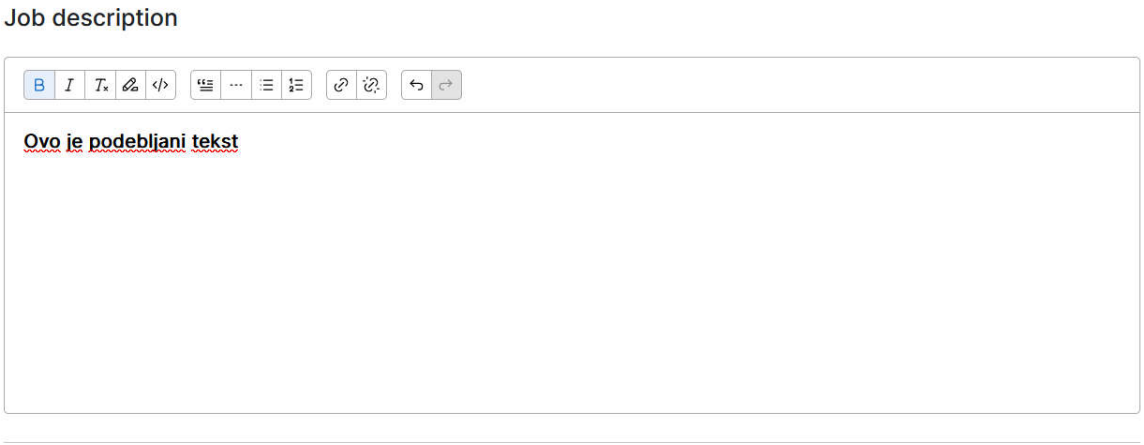


Slika 13: Forma za uređivanje profila tvrtke

Klikom na tipku „*Save changes*“, kroz PUT zahtjev, promjena se ažurira u tablici `Companies`.

4.7. Objava oglasa

Tvrtka objavljuje oglase na svoje otvorene pozicije klikom na „*Post a Job*“ u navigacijskoj traci unutar nadzorne ploče. Kod otvaranja stranice, izvršava se nekoliko GET zahtjeva da bi se padajući izbornici popunili dostupnim odabirima. Izbornici koji se moraju popuniti su: države, tip posla, tražena razina iskustva, tražena razina obrazovanja, lista vještina i lista benefita. Podaci koji se koriste za padajuće izbornike spremljeni su u jednostavnim tablicama koje se sastoje samo od identifikatora i naziva. Opis pozicije unosi se uz pomoć uređivača teksta *TipTap* koji olakšava formatiranje HTML-a, prikazano na slici 14.



Slika 14: Uređivanje opisa oglasa

Prilikom unosa opisa oglasa, provjerava se odgovara li sadržaj odabranoj razini iskustva. Svaka od razina iskustva sadrži popis riječi koje nisu dozvoljene, primjerice, ako se odabere razina 1-2 godine iskustva, u opisu nije dozvoljena riječ „*senior*“. Za poboljšanje ovog ograničenja, uvedena je i tolerancija na razliku u slovima unutar riječi. Za provjeru riječi koristi se izračun Levenshteinove udaljenosti – broj koji predstavlja udaljenost dva *stringa*, odnosno broj slova koji je potrebno dodati, oduzeti ili zamijeniti da bi se od jedne riječi došlo do druge. Ispis 12 prikazuje kôd za pozivanje provjere nad svakom riječi u tekstu.

```
const useJobDescriptionValidator = () => {
  const [warnings, setWarnings] = useState<string[]>([]);

  const similarityThreshold = 2; // Tolerates up to 2 character differences

  const checkForInconsistentWords = (
    description: string,
    jobLevelId: number
  ): void => {
    let flaggedWords: string[] = [];

    const jobLevel = jobLevelId as JobLevel;
    const wordsToCheck = levelWordsMap[jobLevel] || [];

    const descriptionWords = description.toLowerCase().split(/\s+/);

    wordsToCheck.forEach((wordToCheck) => {
      descriptionWords.forEach((descriptionWord) => {
        if (
          levenshteinDistance(wordToCheck, descriptionWord) <=
          similarityThreshold
        ) {
          flaggedWords.push(wordToCheck);
        }
      });
    });
```

Ispis 12: Provjera dozvoljenih riječi u opisu oglasa

Levenshteinova udaljenost je algoritam koji je 1965. godine objavio ruski matematičar Vladimir Levenshtein. Ovaj koncept je s vremenom postao ključan dio sustava za provjeru pravopisa (engl. *spell-checking*) na velikom broju modernih uređaja. Udaljenost se može

izračunati uz pomoć matrice i poznatog Wagner–Fischer algoritma, koji definira sljedeće korake:

1. Kreira se matrica koja ima veličinu prema broju slova riječi koje se uspoređuju.
2. Prvi red i prvi stupac popunjavaju se rednim brojevima.
3. Za svako polje matrice analiziraju se susjedne vrijednosti (iznad, s lijeva i dijagonalno) i uzima se najmanja od tih tri vrijednosti. Ako slovo u stupcu i retku nije isto, na najmanju vrijednost dodaje se jedan (izvođenje jedne od operacija). Ako je slovo u stupcu i retku isto, vrijednost ostaje ista, odnosno jednaka je najmanjoj od tri susjedne vrijednosti.
4. Ponavljanjem opisanog postupka popunjava se cijela matrica. Broj u donjem desnom kutu matrice predstavlja rezultat, odnosno udaljenost između dvije riječi.

Primjer traženja udaljenosti između riječi „*junior*“ i „*senior*“, prikazan u tablici 1.

Tablica 1: Primjer određivanja Levenshteinove udaljenosti između dvije riječi

	_	**J**	**U**	**N**	**I**	**O**	**R**
_	0	1	2	3	4	5	6
S	1	1	2	3	4	5	6
E	2	2	2	3	4	5	6
N	3	3	3	2	3	4	5
I	4	4	4	3	2	3	4
O	5	5	5	4	3	2	3
R	6	6	6	5	4	3	2

Opisani postupak implementiran u kôdu prikazan je u ispisu 13.

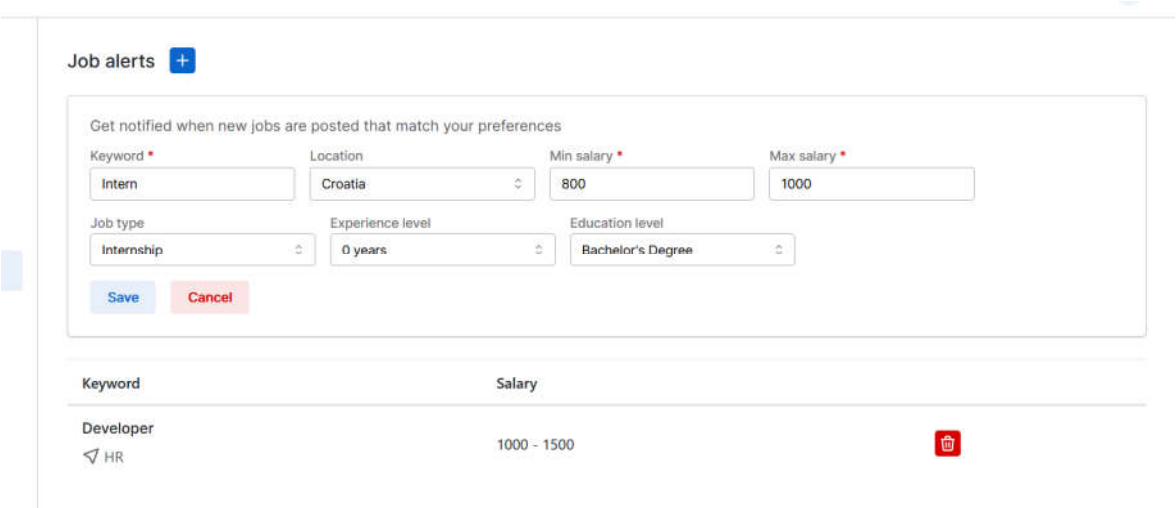
```
const levenshteinDistance = (a: string, b: string): number => {
  const matrix: number[][] = Array.from({ length: a.length + 1 }, (_, i) =>
    Array(b.length + 1).fill(i)
  );

  for (let j = 0; j <= b.length; j++) {
    matrix[0][j] = j;
  }

  for (let i = 1; i <= a.length; i++) {
    for (let j = 1; j <= b.length; j++) {
      if (a[i - 1] === b[j - 1]) {
        matrix[i][j] = matrix[i - 1][j - 1];
      } else {
        matrix[i][j] = Math.min(
          matrix[i - 1][j] + 1,
          matrix[i][j - 1] + 1,
          matrix[i - 1][j - 1] + 1
        );
      }
    }
  }
```

Ispis 13: Implementacija Wagner–Fischer algoritma

Kod spremanja oglasa šalje se POST zahtjev kroz `JobController`, poziva se metoda `CreateJob` u klasi `JobService` koja unosi zapis u tablicu `Jobs`, te šalje obavijest putem elektroničke pošte o novom poslu svim kandidatima koji su prethodno dodali da žele primati poruku ukoliko se objavi oglas s traženim uvjetima (ključna riječ u naslovu, lokacije, razina iskustva).

4.8 Obavijesti putem elektroničke pošte o novim oglasima

Kandidati mogu navesti uvjete za primanje obavijesti putem elektroničke pošte u slučaju kada se objavi novi oglas koji odgovara traženim uvjetima. Da bi pristupili ovoj funkcionalnosti, kandidati u svojoj nadzornoj ploči odaberu „*Job Alerts*“. Otvara se tablica, prikazana na slici 15, s trenutno postavljenim upozorenjima, ukoliko postoje i tipka za otvaranje forme u kojoj se unose uvjeti koje oglas mora ispuniti da korisnik dobije elektroničku poštu. Dodana upozorenja zajedno s uvjetima se upisuju u tablicu `JobAlerts`. Kada tvrtka objavljuje oglas, zadnji korak poslužitelja je provjera spomenute tablice i slanje pošte korisnicima. Kôd za sastavljanje uvjeta za slanje pošte, prikazan je u ispisu 14.



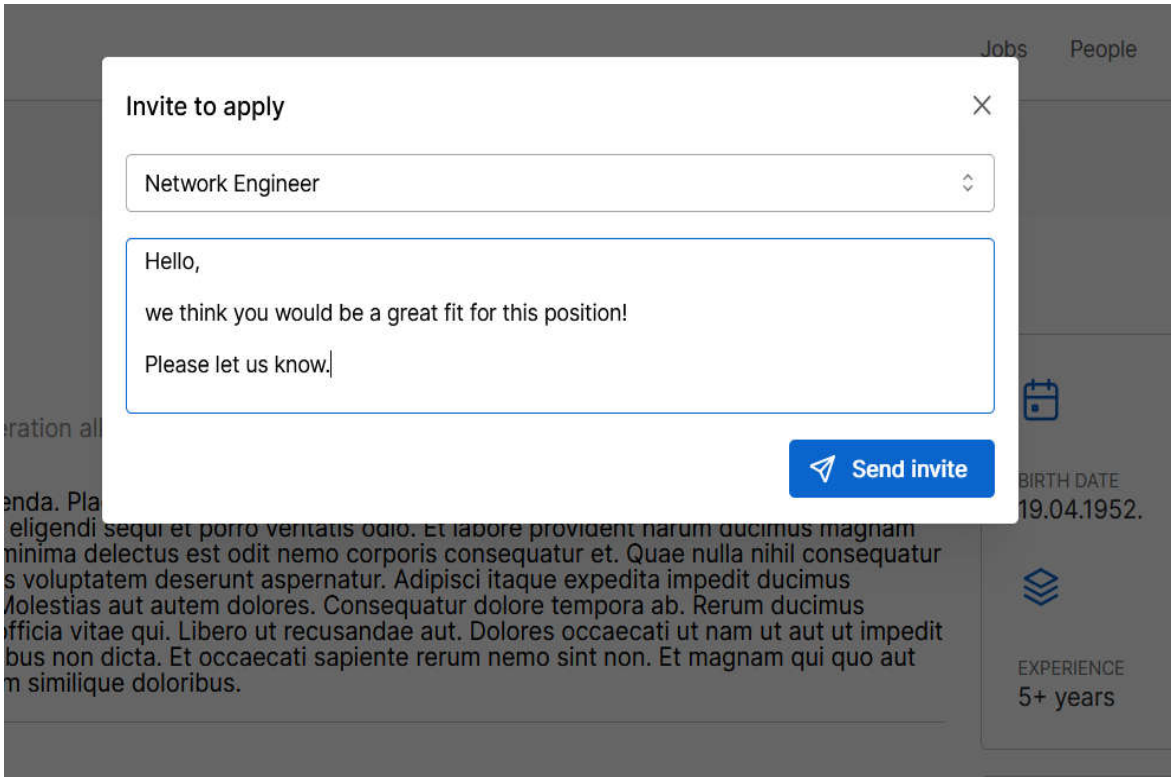
Slika 15: Sučelje za obavijesti o novim oglasima

```
...
            predicate = predicate.And(x => x.JobTypeId.HasValue && x.JobTypeId
== job.JobTypeId);
            predicate = predicate.And(x => x.MinSalary.HasValue && x.MinSalary
>= job.MinSalary);
            predicate = predicate.And(x => x.MaxSalary.HasValue && x.MaxSalary
<= job.MaxSalary);
            predicate = predicate.And(x => x.IsDeleted == false);
var jobAlerts = await jobAlertRepository.FindByCondition(predicate);


            if (jobAlerts.Count() == 0)
            {
                return false;
            }
            var subject = $"New Job Posted: {job.Title}";
            var body = $@"
            <h1>New Job Posted</h1>
            <p><strong>Title:</strong> {job.Title}</p>
            <p><strong>Location:</strong> {job.Location}</p>
            <p>See                more                details                <a
href='https://localhost:3000/jobs/{job.Id}'>here</a></p>
            <p>Thank you for using our job platform!</p>";
            var emails = jobAlerts.Select(x => x.Candidate.Email).ToArray();
            EmailSender.SendEmail(emails, "New Job Alert", body);
            return true;
```

Ispis 14: Kôd za slanje obavijesti o novom oglasu koji zadovoljava odabrane kriterije

4.9. Pozivnica za prijavu na oglas

Ukoliko tvrtka nađe kandidata za kojeg smatra da bi mogao odgovarati traženom profilu zaposlenika za poziciju koju traži, može pozvati kandidata na prijavu. Na profilu svakog kandidata nalazi se tipka *Invite* koja otvara modalni prozor, prikazan na slici 16, u kojem se odabire jedan od aktivnih oglasa, te poruka koju se šalje u pozivnici.



Slika 16: Modalni prozor slanja pozivnice za prijavu na oglas

Na profilu kandidata, pozivnice se prikazuju na stranici „*Job Invites*“ unutar nadzorne ploče, te kandidat ima opciju prijave ili brisanja pozivnice. Pozivnice se zapisuju u tablicu `Invites`.

4.10. Poruke

Kandidati i poslodavci mogu međusobno komunicirati putem poruka. Razgovor može inicirati kandidat prema tvrtki ili tvrtka prema kandidatu, klikom na tipku „*Send message*“ na profilu korisnika. Sučelju za poruke pristupa se klikom na „*Messages*“ u glavnoj navigacijskoj traci. Nakon otvaranja stranice, klijent šalje zahtjev za listu razgovora prema poslužitelju. Zahtjev dolazi na metodu `GetConversations` , prikazanu na ispisu 15, koja dohvaća zapise iz tablice `Conversations`. Tablica `Conversations` sadrži vezu između dva sugovornika, zapis kada je poslana zadnja poruka u razgovoru, te informaciju kad je koji korisnik zadnji put otvorio taj razgovor. Ove informacije su potrebne da bi se korisniku mogla prikazati informacija o još nepročitanim porukama. Prilikom

dohvata, `MessageService` na temelju prijavljenog korisnika određuje koja osoba u razgovoru je sugovornik i sukladno tome dohvaća ime, sliku profila i vrijeme posljednje poruke.

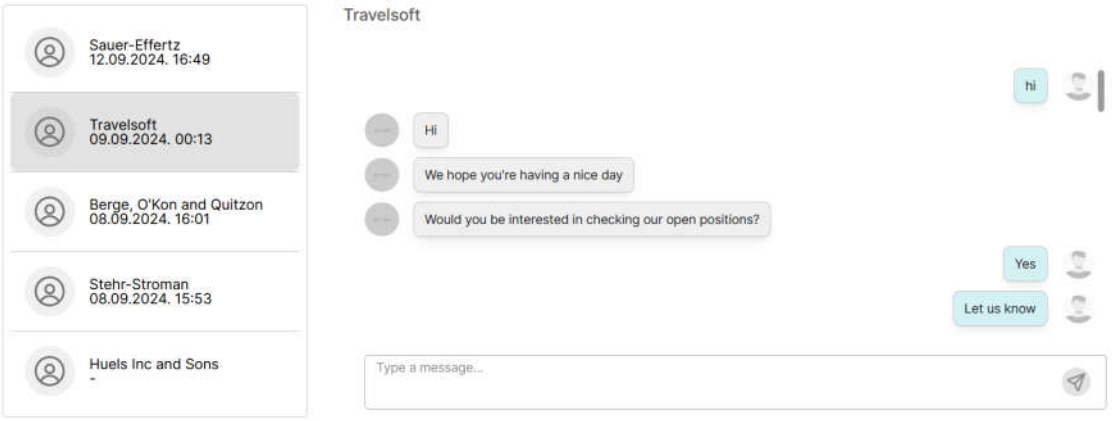
```
        public async Task<IEnumerable<ConversationResponse>> GetConversations()
        {
            var conversations = await
messageRepository.GetConversations(identityContext.UserId);

            return conversations.Select(x => new ConversationResponse
            {
                Id = x.Id,
                PhotoId = GetConversationPhoto(x),
                Name = GetConversationName(x),
```

Ispis 15: Dohvat liste razgovora

Nakon dohvata, razgovori su prikazani u listi s lijeve strane, dok desna strana služi za prikaz odabranog razgovora, prikazano na slici 17. Kod odabira razgovora, šalje se zahtjev koji poziva metodu `GetMessages`. Unutar servisa za dohvat poruka, uzimaju se podaci iz tablice `ConversationMessages`. Komponenta `ConversationBox`, prikazana na ispisu 16, formatira odgovor ovisno o logiranom korisniku koji sadrži ime sugovornika, sliku profila, samu poruku i vrijeme slanja. U tablici `Conversations` ažurira vrijeme kada se razgovor posljednji put otvoren.

Na klijentskoj strani aplikacije se prikazuje izlist poruka, stiliziran upotrebom CSS-a kako bi poruke sugovornika bile lijevo, a poruke logiranog korisnika desno, kao što je uobičajeno u aplikacijama za komunikaciju. Ispod liste poruka nalazi se tekstualno polje za upis poruke. Polje za upis poruke, na pritisak tipke *Enter* šalje poruku, a kada je istovremeno pritisnuta i tipka *Shift*, kada se upisuje novi red. Implementacija je postignuta korištenjem HTML događaja (engl. *event*) *onKeyDown*, koji omogućava reakciju na pritisak određene tipke ili kombinacije tipki.

Kod slanja poruke se uz POST akciju poziva metoda `SendMessage` na poslužitelju koja u tablicu `Conversations` unosi novi redak sa sadržajem poruke i ID-jem pošiljatelja. Na klijentskoj strani aplikacije se ta poruka dodaje u izlist, te se korištenjem *JavaScript* metode `scrollIntoView()` prozor s porukama pomiče na dno, odnosno na najnoviju poslanu poruku.



Slika 17: Sučelje za razgovore između korisnika

Kako bi korisnik znao da je primio novu poruku, u navigacijsku traku je uz poruke dodan i broj nepročitanih poruka. Kod prikaza trake, šalje se GET zahtjev koji poziva metodu `GetUnreadCount` koja na temelju informacija o vremenu zadnje poslane poruci i zadnjem otvaranju razgovora, dohvaća broj nepročitanih poruka.

```
<ConversationBox
                    messages={d ?? []}
                    currentUser={user}
                   />
                   <Textarea
                    onKeyDown={(e) => {
                      if (e.key === "Enter" && !e.shiftKey) {
                        e.preventDefault();
                        sendMessage();
                      }
                    }}
                    value={newMessage}
                    onChange={(event) =>
                      setNewMessage(event.currentTarget.value)
                    }
                    autosize
                    minRows={3}
                    maxRows={5}
                    placeholder="Type a message..."
                    ....
/>
```

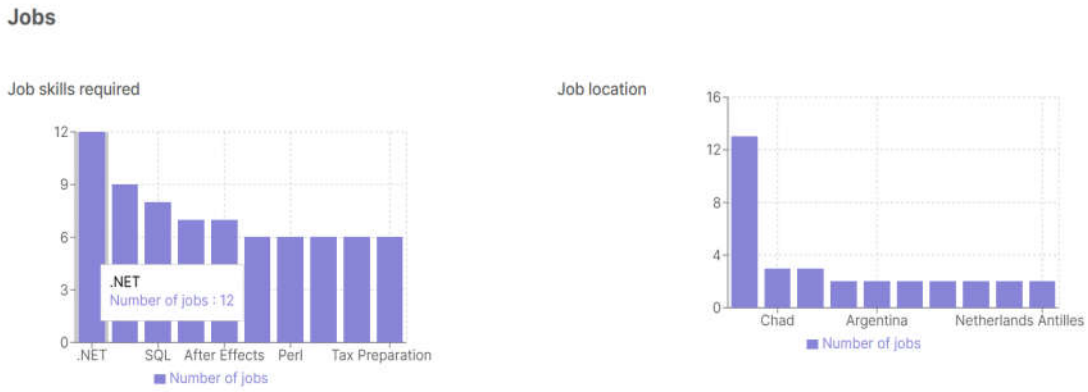
Ispis 16: Komponenta za poruke

4.11. Statistika

Kako bi svi korisnici imali bolji uvid u stanje na tržištu, implementiran je prikaz nekoliko stupčastih i kružnih grafikona. Prikazuju se sljedeće informacije:

- vještine koje se najviše traže u objavljenim oglasima
- države s najvećim brojem oglasa
- broj oglasa prema traženoj razini obrazovanja

- broj oglasa prema traženoj razini iskustva
- države s najvećim brojem tvrtki
- najzastupljeniji tip industrije prijavljenih tvrtki
- distribucija tvrtki po tipu organizacije
- distribucija tvrtki po veličini
- vještine koje posjeduje najveći broj kandidata
- države prema lokaciji kandidata
- distribucija razine obrazovanja kandidata
- distribucija razine iskustva kandidata



Slika 18: Stupčasti grafikoni za prikaz statistike

Za implementaciju grafova koristi se *Recharts* knjižnica. *Recharts* nudi širok raspon grafičkih prikaza, uključujući stupčaste grafikone (prikazano na slici 18), linijske grafikone, tortne grafikone, raspršene dijagrame, radarske dijagrame i mnoge druge. Svaka komponenta unutar knjižnice je modularna, što znači da se grafikoni mogu prilagoditi prema vlastitim potrebama dodavanjem ili uklanjanjem elemenata poput legendi i tooltipova.

Knjižnica je pogodna za rad s velikim skupovima podataka jer omogućava interaktivnost poput prikaza detalja na tooltipovima ili isticanja elemenata pri prelasku mišem. Također, podržava responzivnost, što je čini prikladnom za aplikacije koje se koriste na različitim uređajima.

Dohvat podataka je izveden korištenjem tri odvojena poziva, kako bi se postigao

paralelan dohvat tri sekcije, poslovi, kandidati. Za svaki od grafikona izvršava se upit SQL na bazu koji uzima 10 najzastupljenijih rezultata, poreda rezultate po veličini koristeći `OrderBy` funkciju te ih dodaje u objekt u obliku ime-vrijednost. Kôd za dohvat prikazan je na ispisu 17, a rezultat dohvata je na ispisu 18.

```
var location = context.Employers
    .Join(
        context.Countries,
        employer => employer.Country,
        country => country.Code,
        (employer, country) => new { employer, country }
    )
    .GroupBy(x => x.country.Name)
    .Select(g => new DataPoint
    {
        Name = g.Key,
        Value = g.Count()
    })
    .OrderByDescending(dp => dp.Value)
    .Take(10)
    .ToArray();
```

Ispis 17: Dohvat podatkovnih točaka za prikaz grafova statistike

```
"skills":[
      {
         "name":".NET",
         "value":12
      },
      {
         "name":"Foundation",
         "value":9
      },
```

Ispis 18: Odgovor poslužitelja za prikaz statistike

4.12. Generiranje podataka

Kako bi projekt sadržavao podatke koji nalikuju stvarnim, potreban je unos velike količine informacija. Za automatizaciju tog postupka, koristi se knjižnica *Faker.NET*, koja sadrži lažna imena, adrese i opise i druge često korištene informacije. Implementirane su tri metode, za generiranje tvrtki, osoba i oglasa. Metoda `SeedCompanies` kao parametar prima broj koji predstavlja koliko tvrtki je potrebno generirati. Prvi korak je kreiranje novog korisnika uz pomoć lažne elektroničke pošte, koji se unosi u tablicu `User`. Nakon toga, koristi se *Faker* kako bi se generiralo ime, opis, poveznica na web stranicu, adresa tvrtke i ostale informacije. Metoda za stvaranje podataka tvrtki prikazana je prikazana je u ispisu 19.

Na sličan način funkcioniraju i metode `SeedPeople` za generiranje osoba i `SeedJobs` za generiranje oglasa za posao.

```
        public async Task<IEnumerable<Employer>> SeedCompanies(int count)
        {
            var companies = new List<Employer>();

            for (int i = 0; i < count; i++)
            {
                var email = Faker.Internet.Email();
                var password = "123456";
                User user = new User
                {
                    Email = email,
                    UserType = UserType.Company,
                    SetupCompleted = true
                };
                Hashing.CreatePasswordHash(password, out string passwordHash,
out string passwordSalt);
                user.PasswordHash = passwordHash;
                user.PasswordSalt = passwordSalt;

                User createdUser = await userRepository.Insert(user);
                var countries = await countryRepository.GetAll();


                var company = new Employer
                {
                    Name = Faker.Company.Name(),
                    ShortDescription = Faker.Company.CatchPhrase(),
                    Description = Faker.Company.BS(),
```

Ispis 19: Kôd za generiranje lažnih podataka

5. ZAKLJUČAK

U ovom završnom radu opisan je postupak izrade web aplikacije za traženje posla *JobConnect* te su prikazane ključne tehnologije korištene u njenoj izradi. Detaljno su opisane arhitektura aplikacije, korištene tehnologije i glavne funkcionalnosti. *JobConnect* omogućuje brzo pretraživanje dostupnih radnih mjesta i potencijalnih kandidata te uspostavljanje kontakta između poslodavaca i onih koji traže posao.

Korisnicima su pružene različite mogućnosti pretraživanja, uključujući filtriranje prema ključnim riječima, lokaciji, vrsti zaposlenja i potrebnim vještinama. Svaki oglas za posao sadrži detaljne informacije koje pomažu posloprimcima u procjeni odgovara li određena pozicija njihovim profesionalnim ciljevima i kvalifikacijama.

Ključna funkcionalnost aplikacije je sustav prijave na oglase za posao. Posloprimci mogu jednostavno podnijeti prijavu na željeno radno mjesto, priložiti svoj životopis i dodatne informacije. Nakon prijave, poslodavci mogu pratiti sve prijave kroz sučelje koje omogućuje pregled i upravljanje kandidatima, zakazivanje intervjua.

Dodatna funkcionalnost aplikacije je sustav razmjene poruka koji omogućuje poslodavcima i posloprimcima uspostavljanje direktnog kontakta. Ova opcija olakšava brzu komunikaciju, razmjenu dodatnih informacija i dogovaranje detalja vezanih uz zapošljavanje.

Sve funkcionalnosti implementirane su korištenjem suvremenih web tehnologija. Aplikacija je razvijena koristeći *React* za izradu korisničkog sučelja, *React Query* za upravljanje podatkovnim upitima i predmemorijom podataka, te *.NET Core* za poslužiteljsku stranu aplikacije. Podaci su pohranjeni u *MS SQL* bazi podataka, a pristup podacima realiziran je pomoću razvojnog okvira *Entity*. Autentifikacija korisnika provedena je putem *Basic Authentication* protokola, dok komunikacija između korisničkog sučelja i poslužitelja koristi REST API. Provjerom osobne iskaznice prilikom registracije postigla se dodatna sigurnost ispravnosti identiteta registriranih korisnika.

U budućnosti, aplikacija *JobConnect* može se proširiti novim funkcionalnostima. Primjerice, može se implementirati sustav naplate koji oglase plaćenih poslodavaca dodatno ističe na vrh rezultata pretraživanja. Također, moguće je uvesti i sustav preporuka kandidata. Dodatno, aplikacija može biti koristiti umjetnu inteligenciju za lakši unos opisa oglasa.

LITERATURA

[1] Incisiv, AT&T, Dubber: Future of Work Study

https://www.business.att.com/learn/research-reports/is-corporate-america-ready-for-the-future-of-work.html (posjećeno 5.11.2024.)

[2] Kurtović, G: Uvod u HTML, Sveučilišni računski centar, 2016.

[3] Stančer, D: Osnove JavaScripta, Sveučilišni računski centar, 2019.

[4] TypeScript dokumentacija https://www.typescriptlang.org/docs/ (posjećeno 20.10. 2024.)

[5] UltimateMRZ dokumentacija https://www.doubango.org/SDKs/mrz/docs/MRZ_formats.html (posjećeno 22.10. 2024.)