Proces dizajniranja web aplikacije

Mirat, Jure

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Jure Mirat

PROCES DIZAJNIRANJA WEB APLIKACIJE

DIPLOMSKI RAD

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Jure Mirat

Matični broj: 0058199670

Studij: Informacijsko i programsko inženjerstvo

PROCES DIZAJNIRANJA WEB APLIKACIJE

DIPLOMSKI RAD

Mentor:

Izv. prof. dr. sc. Sandro Gerić

Varaždin, rujan 2021.

Jure Mirat

Izjava o izvornosti

Izjavljujem da je moj diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Cilj ovog rada je prikazati cijeli proces dizajniranja web aplikacije od njene početne ideje do njene implementacije, praćenjem smjernica dobrog dizajna. Rad se sastoji od teoretskog i praktičnog dijela. U teoretskom dijelu će biti objašnjen životni ciklus razvoja web aplikacija, specifičnosti vezane za razvoj web aplikacija i metodologija koja će biti korištena u radu. Proces dizajniranja će biti prikazan na primjeru razvoja web aplikacije za pronalazak stručnjaka (*freelancera*). Koristit će se preporučena praksa za projektiranje informacijskih sustava. Koja uključuje planiranje aplikacije prepoznavanjem poslovnih zahtjeva, analizu zahtjeva, arhitekturno i grafičko dizajniranje aplikacije i implementaciju. U praktičnom dijelu će biti napravljena implementacija obrađenog teoretskog primjera i analizirat će se korisničko sučelje. Web aplikacija će biti implementirana koristeći .NET razvojnu okolinu, sa PostgreSQL bazom podataka.

Ključne riječi: web aplikacija, projektiranje informacijskih sustava, proces dizajna, grafičko sučelje, web inženjerstvo

Sadržaj

1. Uvod

U relativno kratkom roku, unutar jednog desetljeća, svjetska mreža (*World Wide Web*) je postala sveprisutni internetski servis. Njen rast se i do danas odvija eksponencijalnom brzinom što se u praksi najbolje vidi kroz cijeli niz web aplikacija i sustava o kojima sve više ovisimo. Zbog te, sve veće, ovisnosti je postalo važno da su ti sami sustavi kvalitetni i pouzdani. Količina zahtjeva i očekivanja od sustava su se znatno povećali. To je dovelo do puno veće složenosti u dizajniranju, razvoju, kodiranju i održavanju sustava. Mnogi sustavi se i dalje razvijaju *ad hoc*, nedovoljno planirano, s naglim zaokretima u smjeru razvoja sustava i prekratkim vremenskim rokovima. To stavlja veliko opterećenje na programere i dovodi do razvoja nekvalitetnih sustava s potencijalno ozbiljnim problemima. Kod manjih sustava, to nije toliko izraženo, ali kod velikih, kritičnih sustava za tvrtke, to može dovesti do financijskog gubitka, frustriranih korisnika, gubitka ugleda i slično. Kako bi se povećala kvaliteta razvoja sustava, postalo je vrlo važno da programeri usvoje disciplinirani pristup razvoju sustava koji se temelji na jasnom procesu razvoja i metodologiji. Kao i korištenje boljih alata za razvoj. Cilj, pogotovo kod razvoja sustava za web je bio da naglasak razvoja nije samo na razvoju funkcionalnosti programa ili samo vizualnom dizajnu nego da se cijeli proces razvoja gleda kao zajedničku cjelinu u kojoj je sve međusobno povezano. A za uspješnu međusobnu povezanost je potrebno kvalitetno planiranje, dizajniranje sustava i njegove arhitekture, testiranje, osiguranje kvalitete, kontinuirano održavanje.

Proces dizajniranja je zato nužan na svakom projektu kako bi se, kao prvo, postavila jasna očekivanja od sustava i to kroz komunikaciju s klijentom. Jasna komunikacija s klijentom je nužna od samog početka kako bi se postavili jasni ciljevi i rokovi za isporuku nekog sustava. Zatim, da bi se jasnije mogao pratiti napredak razvoja sustava i da sve stranke koje bi trebale moći sudjelovati u razvoju budu uključene u razvoj. Posljednje, smanjuje se rizik od neuspjeha jer se razmatraju svi aspekti sustava. S adekvatnim procesom razvoja može se razviti sustav koji je sa strane dizajna, vizualno i arhitekturno kvalitetan i u potpunosti funkcionalan. Cilj ovog rada je prikazati takav proces dizajniranja web sustava na primjeru sustava za pronalazak stručnjaka (*freelancera)* i time prikazati problematiku s kojom se suočavaju svi programeri koji su se našli u situaciji gdje je bilo potrebno dizajnirati novi sustav.

2. Programsko inženjerstvo

Razvoj softvera nije definirani proces. To je empirijski proces. I kao takav, razvoj softvera ne može biti u potpunosti automatiziran i često je vrlo teško na njega primijeniti inženjerske principe. Dio problema je što se praktično inženjerstvo jako oslanja na ponovnoj upotrebi postojećih dizajna. Iako je znatna količina ponovne upotrebe dizajna moguća u računalnom programiranju, ona zahtjeva puno više prilagođavanja nego što se može naći u drugim inženjerskim profesijama. [1] Čime se proces razvoja softvera po količini potrebne (logičke) kreativnosti katkad može povezati i s umjetnošću.

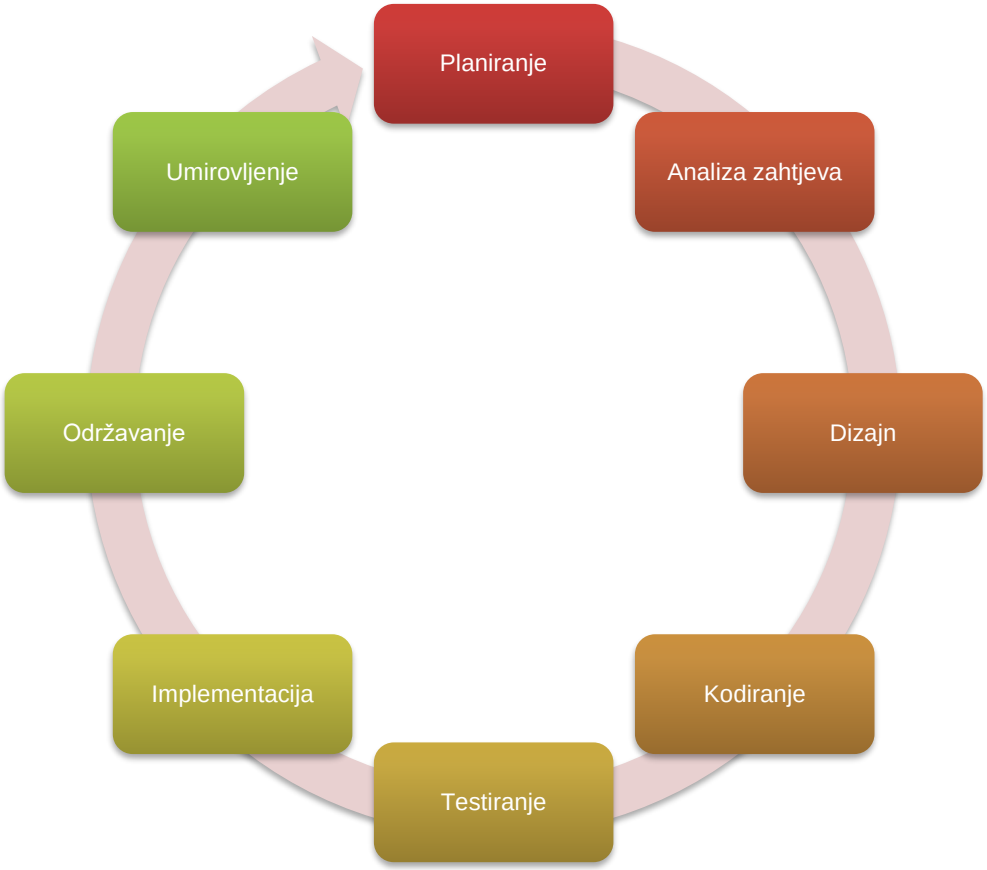
Disciplina koja se bavi procesom razvoja aplikacija je programsko inženjerstvo. Definicija programskog inženjerstva prema IEEE: „Primjena sustavnog, discipliniranog i mjerljivog pristupa na strukture, strojeve, proizvode, sustave i procese.“ [1, str. xxxi]

Cilj programskog inženjerstva je pružiti disciplinirani pristup razvoju softvera kako bi se razvijali sustavi koji su robusni, pouzdani, isporučeni na vrijeme i u okviru zadanog financijskog proračuna. Te koji bi riješili probleme zbog kojih su i razvijani. Sistematični proces za razvoj softvera je nužan kako bi se osigurala njegova kvaliteta i ispravnost. U središtu aktivnosti softverskog inženjerstva je dizajniranje koncepcije sustava koji ima svoju svrhu i planiranje njegovog razvoja. Sami dizajn softvera široko je zastupljen u razvoju softvera. Bavi se vanjskim, funkcionalnim aspektima softvera, kao i njegovim unutarnjim dijelovima. Vanjski, funkcionalni aspekti softvera definiraju njegove mogućnosti i sučelje za komunikaciju s korisnicima ili drugim softverom u okruženju. Unutarnji dizajn definira karakteristike i funkcije s pomoću kojih se postiže vanjski dizajn. [1]

U kontekstu web sustava su se pojavile specifičnosti vezane za web servise pa se u literaturi može često pronaći i termin programsko inženjerstvo za web. Cilj programskog inženjerstva za web je smanjiti rizike, poboljšati kvalitetu, održivost i skalabilnost web sustava što dovodi do kontroliranijeg okruženja za razvoj web aplikacija. Srž web inženjerstva je uspješno upravljanje raznolikošću i složenošću razvoja web sustava kroz životne cikluse web aplikacija s ciljem identificiranja i prepoznavanja potencijalnih problema prije nego se oni pojave.

2.1. Životni ciklus razvoja aplikacija

Životni ciklus razvoja aplikacija je skup koraka koji se koriste za kreiranje softvera. Oni primjenjuju standardne poslovne prakse i sadrže sve preporučene osnovne komponente za projekte razvoja softvera. [2] Za vrijeme svog života (razvoja), softver generalno prolazi kroz osam faza. To su planiranje, analiza zahtjeva, dizajn, kodiranje, testiranje, implementacija, održavanje i umirovljenje.



Slika 1. Životni ciklus razvoja softvera [autorski rad]

Prva faza s kojom počinje neki projekt razvoja aplikacije je planiranje. U ovoj fazi, klijent, menadžer ili neki drugi sudionik, u suradnji s ostalim sudionicima u projektu, osmisle ideju za aplikaciju koja se bavi određenim poslovnim slučajem. Korist te zamišljene aplikacije se zatim ocjenjuje, definira se njen opseg, računaju se troškovi izrade i određuju se koraci razvoja. [2]

Druga faza je analiza zahtjeva. Sudionici projekta nastoje odgovoriti na temeljna pitanja projekta. Primjeri takvih pitanja su: Kome je sustav namijenjen? Koje informacije ulaze u sustav? Koje informacije izlaze iz sustava? U kojem formatu sustav razmjenjuje informacije?

Očekivana vremena odaziva? Preko odgovora na ta pitanja se mogu razaznati hardverski i softverski zahtjevi koji se onda raspišu. [3]

Treća faza je dizajn. Zahtjevi iz prijašnje faze se koriste kako bi se dobio pregled svih aspekta sustava. Ova faza može uključivati i izradu prototipa. U nastavku će biti pojašnjeni neki aspekti sustava koji se definiraju u ovoj fazi. Dizajn arhitekture koji uključuje određivanje programskog jezika, cjelokupni dizajn sustava, primjenu prakse iz industrije. Dizajn korisničkog sučelja preko kojeg korisnici komuniciraju s aplikacijom. Određivanje na kojoj platformi ili platformama će softver biti implementiran. Sigurnosne mjere koje će biti implementirane kako bi aplikacija bila zaštićena. Definiranje metoda komunikacije s nekim drugim servisima ili aplikacijama. Definiranje metoda rješavanja određenih problema. [2]

Četvrta faza je kodiranje, odnosno programiranje aplikacije. Ovo je faza, za koju se smatra, da je programerima najpoznatija i „najzabavnija“. Organiziranje timova programera s potrebnim vještinama kako bi mogli programirati aplikaciju.

Peta faza je testiranje. U ovoj fazi se provjerava da li aplikacija uspješno ispunjava sve navedene zahtjeve. Testiranje smanjuje broj potencijalnih grešaka s kojima se korisnici mogu susresti. Postoji više načina testiranja. Jedinično testiranje provjerava pojedinačne module u programu i provjerava njihovo odgovaranje očekivanjima. Integracijsko testiranje potvrđuje dobru suradnju između pojedinih podsustava u aplikaciji. Samo testiranje sustava potvrđuje njegovu implementaciju. I ispitivanje prihvatljivosti provjerava da li je aplikacija prikladna za namjeravanu namjenu. [3]

Šesta faza, i posljednja koja se odnosi na razvoj sustava dok on ne dođe do korisnika, je implementacija softvera. Softver se dostavlja korisnicima na korištenje. U kontekstu web aplikacije, aplikacija se objavi na server i omogući se pristup do nje preko weba.

Sedma faza je održavanje. Otklanjaju se nedostaci iz sustava koji su otkriveni tijekom korištenja. Dodaju se nova poboljšanja ili funkcionalnosti u aplikaciju.

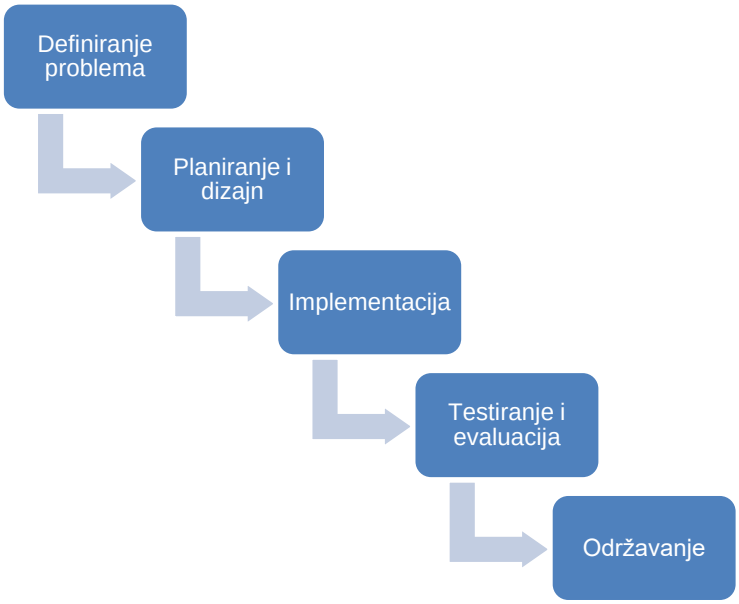
Osma i posljednja faza je umirovljenje softvera. To je donošenje poslovne odluke da se više neće pružiti podrška za sustav i da razvoj prestaje.

2.2. Modeli razvoja aplikacija

Životni ciklus razvoja aplikacija daje proces kojim se aplikacije razvijaju, a modeli razvoja aplikacija daju pristup provedbi tog procesa. Ti modeli opisuju kako se sve faze životnog ciklusa procesa razvoja softvera spajaju u softverski projekt. Postoji nekoliko uobičajenih popularnih modela razvoja i stotine, ako ne i tisuće, njihovih iteracija. Model razvoja softvera opisuje što će se raditi ali ostavlja puno prostora načinu kako će se raditi. Postoji i razlika između životnog ciklusa razvoja tradicionalnog (*desktop)* i razvoja web aplikacija. Iako oni dijele mnoge zajedničke karakteristike, web projekti obično imaju kraće vrijeme životnog ciklusa razvoja, sustave koji integriraju softver i podatke te multidisciplinarne timove. U nastavku je kratki pregled pet uobičajenih modela koji se često koriste kod razvoja sustava.

2.2.1. Metoda vodopada

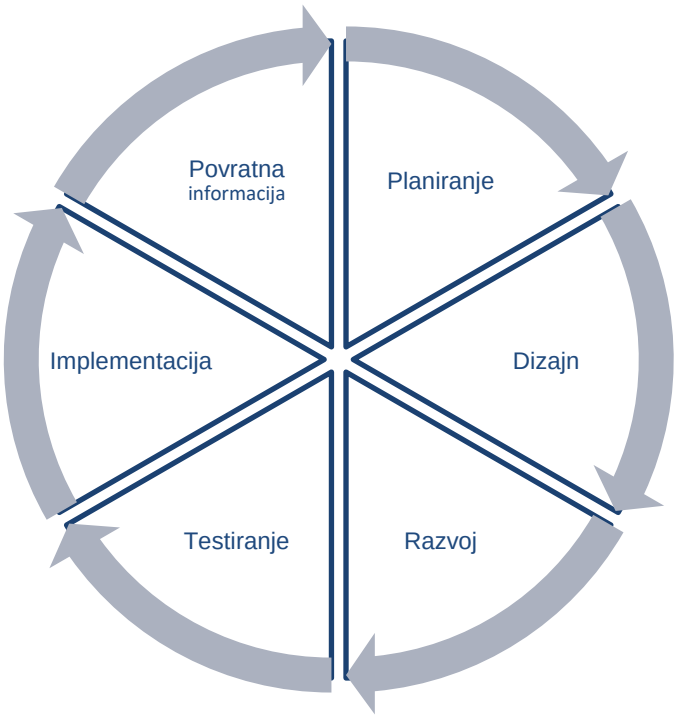
Strukturirani, tradicionalni pristup ili često poznatiji kao 'metoda vodopada' je pristup koji uključuje vrlo strukturirane faze koje se izvršavaju linearno, korak po korak. Metoda je pogodna za razvijanje softvera gdje se svi zahtjevi mogu definirati i razumjeti prije početka planiranja i dizajniranja. Negativnost je što svi zahtjevi i specifikacija moraju biti određeni prije faze dizajna. Tu često nastanu pogreške u definiranju zahtjeva, pogreške koje su tek prepoznate kada se krene s dizajniranjem ili programiranjem sustava. Što je problem kasnije otkriven, cijeli ciklus razvoja postaje sve skuplji za ispraviti. [4]



Slika 2. Koraci metode vodopada [autorski rad]

2.2.2. Agilne metode razvoja

Agilni razvojne metode su metode koje naglasak stavljaju na tim koji razvija sustav, a ne na unaprijed strukturirane razvojne procese, kao što je to slučaj u tradicionalnom modelu. Javljaju se zbog problema nepredvidivosti kod određenih projekata razvoja softvera gdje je postalo jako teško pratiti strukturirani proces razvoja. Uklanja se potreba za detaljnom analizom zahtjeva i složenim dizajnom i umjesto toga se potiče suradnja i timski rad. U agilnim metodama, umjesto planiranja i dokumentiranja implementacije softvera, zahtjevi, dizajn i implementacija pojavljuju se istovremeno i zajedno razvijaju. [1] To ih čini vrlo pogodnima za web i sličan razvoj gdje je potreba za redovitim izmjenama i modifikacijama softvera. Tipičan slijed aktivnosti u agilnom razvoju bi bio da se prvo formira razvojni tim koji onda stvori osnovni plan i opći dizajn s minimalno detalja ali dovoljan da se može početi s razvojem. Detalji se razvijaju kada postanu potrebni. Dogovoreni plan se odmah implementira i testira. I nakon toga se ponovo planiraju i zatim implementiraju nove funkcionalnosti. Ti iterativni ciklusi se stalno ponavljaju i softver se stalno nadograđuje. Jedan takav ciklus može potrajati od par dana do par tjedana. Taj pristup da se razvija dio po dio sustava nudi puno dinamike kod razvoja i umanjuje rizik jer se stalno tijekom razvoja izvodi planiranje i usklađuje s potrebama. Glavna pozitivna stvar kod agilnih metoda je ovisno o situaciji i najveća negativnost. To je nedostatak dokumentacije u vidu detaljnih zahtjeva, čime se puno teže kod projekata dugog razvoja mogu predvidjeti financijski i vremenski troškovi razvoja. U konačnici, agilne metode su samo koncept, a ne puni okvir. Na programeru je kako će slijediti njihove smjernice. [4]



Slika 3. Iterativni pristup agilne metode [autorski rad]

2.2.3.Metoda izrada prototipova

Metoda izrada prototipova, je iterativna metoda razvoja koja dijeli puno sličnosti s agilnom metodom. Planirani zahtjevi se kroz svaku iteraciju dodatnu usavršavaju, a implementirani sustav se detaljno validira. Prototipovi se mogu razvijati konceptualno i evolucijski. Konceptualni prototipovi nikada neće postati finalni sustav i njihov razvoj služi samo testiranje raznih zahtjeva ili testiranje raznih načina na koji se mogu implementirati određeni zahtjevi. Evolucijski prototip će postati finalni sustav. On je kontinuirano nadograđivan kroz sukcesivne generacije prototipova. [4]

2.2.4.Metoda brzog razvoja aplikacije

Metoda brzog razvoja aplikacije (*Rapid Application Development*) je metoda čiji je cilj smanjiti vrijeme razvoja sustava u situacijama kada se zahtjevi sustava mogu iz temelja promijeniti do trenutka kada je originalni plan sustava implementiran. Ti sustavi su obično manje kvalitete jer se kompromisno smanji količina implementiranih funkcionalnosti kako bi se izbjegle greške u radu sustava. Ova metoda se koristi s razvojnim tehnologijama koje omogućuju brzo 'vizualno' programiranje i upotrebu višekratnih komponenti. Odnosno tehnologije u kojima je naglasak na imati što manje programiranja. [4]

2.2.5.Metoda razvoja krajnjeg korisnika

Metoda razvoja krajnjeg korisnika (*End User Development*) je metoda koja korisnicima koji koriste sustav omogućuje da preko raznih 'čarobnjaka' ili generatora koda prilagođavaju i nadograđuju softver koji koriste. Ovakva rješenja manjim tvrtkama nude jeftini način da razviju softver za svoje potrebe ali negativnost je što su jako ograničeni zahtjevi koji se mogu implementirati i često sa zahtjeva prilagodba poslovnih procesa načinu rada tih sustava. Kvaliteta takvog sustava isto može biti puno niža zbog nedostatka znanja razvoja sa strane korisnika koji ga razvija.

2.3. Metode rada

Kroz kratki pregled ovih pet metoda razvoja i nekih njihovih prednosti i mana je ukazano na njihove velike razlike u pristupu razvoja. Kod odabira modela razvoja se uvijek postavlja pitanje zašto se ne može uzeti najpopularniji model razvoja i koristiti ga na svim projektima razvoja. Odgovor se krije i u razlogu zašto ima toliko puno modela razvoja. A to je da u različitim situacijama različiti modeli imaju bolje i lošije rezultate. Na primjer, praksa rada koja dobro funkcionira za pojedinca ili manje timove se često ne može dobro skalirati na veće timove. Isto tako, tehnike koje dobro rade na većim projektima se rijetko mogu uspješno skalirati na manje timove. Nijedna metodologija se ne skalira dobro prema većim ili manjim projektima. Potrebno je odabrati odgovarajući model ovisno o veličini projekta. Za male projekte je vodopadni model bez dokumentacije dobar izbor. Za veće projekte su to iterativni, agilni modeli. [3]

Primjer projekta u ovom radu, razvoj sustava za pronalazak stručnjaka je mali projekt za pojedinca. Na malim projektima su programerski resursi učinkovitije iskorišteni nego na većim. Programer može iskoristiti, po svojem mišljenju, najproduktivniji pristup rješavanju nekog problema jer ne mora dolaziti do dogovora s drugim programerima na projektu. Također se može bolje optimizirati vrijeme koje se troši na svakoj razvojnoj fazi. Što se tiče metode rada, u strukturiranom (vodopadnom) modelu razvoja bi se značajna količina vremena potrošila na dokumentiranje operacija. To nema smisla ako samo jedan programer radi na projektu. Iako, postoje situacije kada se može dogoditi da u jednoj fazi životnog ciklusa sustava drugi programer preuzme razvoj. Na malom projektu programer je potpuno odgovoran za cijeli životni ciklus razvoja, od analize zahtjeva i dizajna sustava do njegove implementacije, testiranja i održavanja. To je puno više zadataka nego što bi programer imao da radi na nekom srednjem ili velikom projektu. Zato je nužno da se zadaci podijele na manje, lakše izvodljive dijelove. Nedostatak, čak i zamka, tako malog projekta je što programer mora biti multidisciplinaran kako bi mogao moći rješavati širok raspon različitih zadataka. Puno malih projekata propadne (ili je trošak razvoja previsok) jer programer nema odgovarajuće vještine za razvoj cijelog projekta. [3]

Metoda razvoja koja će se koristiti u ovom radu će pratiti tradicionalni pristup vodopadnog modela bez dodatne dokumentacije i bez restrikcija iterativnog pristupa.

3. Analiza

U ovom poglavlju će biti opisana web aplikacija za pronalazak stručnjaka. Na temelju tog opisa i analize postojećih aplikacija sličnih funkcionalnosti će biti napisani funkcionalni i nefunkcionalni zahtjevi.

3.1. Opis aplikacije

Potrebno je izraditi web aplikaciju za pronalazak stručnjaka (*freelancera*). Cilj te aplikacije je da se preko portala aplikacije stručnjaci za određena područja i oni koji ih trebaju, mogu povezati. Povezivanje bi se odvijalo tako da korisnici mogu pretraživati profile stručnjaka i kada pronađu onog koji im odgovara, mogu mu poslati poruku i tako s njime stupiti u kontakt.

Dolaskom na stranicu, gost ima mogućnost prijave/registracije na stranicu ili pretraživanja korisničkih profila stručnjaka. Da bi mogao poslati poruku, gost se mora prijaviti na stranicu. Prijavom na stranicu, korisnik sada ima mogućnost dopisivanja s drugim korisnicima (stručnjacima) koji ga zanimaju. Također, ima mogućnost popunjavanja svog profila i aktiviranje opcije da njegov profil isto bude vidljiv kao profil stručnjaka. Čime on isto postaje korisnik stručnjak kojemu se drugi korisnici mogu javiti. Profil stručnjaka bi trebao moći imati slična polja kao i konkurentne stranice. Mogućnost dodavanja kontakt podataka, profilne slike, kratka biografija, ponuđene vještine, okvirna satnica rada. Još jedna uloga na stranici je uloga administratora. Administrator se može prijaviti na stranicu i preko administratorskog portala može administrirati profile korisnika kao i pratiti osnovnu statistiku aplikacije. Osnovna statistika aplikacije bi uključivala ukupnu listu korisnika, ukupnu listu poruka, broj poruka po korisniku, broj pregleda profila nekog korisnika.

3.2. Analiza zahtjeva

Zahtjeve je nužno raspisati kako bi se dobio pregled funkcionalnosti koje bi aplikacija morala imati i kako bi se onda ciljano razvijale samo potrebne funkcionalnosti.. Zahtjevi su podijeljeni na funkcionalne i nefunkcionalne zahtjeve. Funkcionalni zahtjevi navode operacije koje bi trebale biti dostupne korisnicima aplikacije. Nefunkcionalni zahtjevi navode sve druge zahtjeve aplikacije koji nemaju veze s funkcionalnošću.

3.2.1. Funkcionalni zahtjevi

Funkcionalni zahtjevi su logički grupirani po funkcionalnostima i grupama korisnika.

Tablica 1. Funkcionalnost: Pretraživanje i pregled profila stručnjaka

Funkcionalnost: Pretraživanje i pregled profila stručnjaka	
Gost	• Pretraživanje galerije stručnjaka • Filtriranje galerije stručnjaka
Obični korisnika	
Korisnik stručnjak	
Administrator	

(Izvor: autorski rad)

Tablica 2. Funkcionalnost: Prijava i registracija u profil

Funkcionalnost: Prijava i registracija u profil	
Gost	• Registracija novog korisnika • Prijava postojećeg korisnika
Obični korisnik	
Korisnik stručnjak	
Administrator	• Prijava postojećeg administratora

(Izvor: autorski rad)

Tablica 3. Funkcionalnost: Slanje poruke korisnicima

Funkcionalnost: Slanje poruke korisnicima	
Gost	• Ne može započeti dopisivanje
Obični korisnik	• Može započeti dopisivanje s korisnicima stručnjacima • Nakon započetog dopisivanja, može slati i primati poruke
Korisnik stručnjak	
Administrator	

(Izvor: autorski rad)

Tablica 4. Funkcionalnost: Korisnički profil

Funkcionalnost: Korisnički profil	
Gost	• Ne može pristupiti korisničkom profilu
Obični korisnik	• Ažuriranje korisničkog profila • Aktiviranje profila stručnjaka
Korisnik stručnjak	• Ažuriranje korisničkog profila • Deaktiviranje profila stručnjaka
Administrator	• Ažuriranje korisničkog profila

(Izvor: autorski rad)

Tablica 5. Funkcionalnost: Administratorski portal

Funkcionalnost: Administratorski portal	
Gost	• Ne može pristupiti portalu
Obični korisnik	
Korisnik stručnjak	
Administrator	• Može pristupiti portalu • Može dobiti listu korisnika • Može ažurirati podatke korisnika • Može dobiti osnovnu statistiku stranice

(Izvor: autorski rad)

3.2.2.Nefunkcionalni zahtjevi

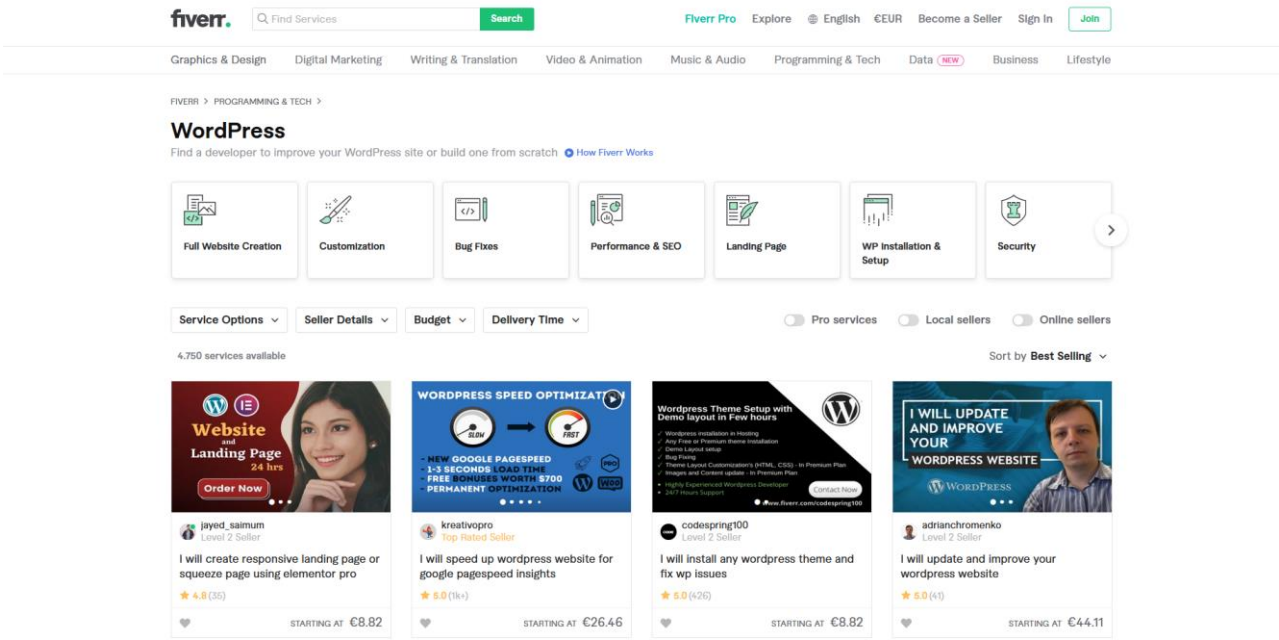
Lista nefunkcionalnih zahtjeva koja ne uključuje aspekte dizajna iz sljedećeg poglavlja.

- Pristupačnost – web aplikacija mora biti dostupna svima preko URL neovisno o Internet pregledniku koji koriste
- Dostupnost – web aplikacija mora moći biti dostupna u svim vremenima 24/7
- Performanse – funkcionalnosti web aplikacije bi se trebale izvršavati unutar razumno brzog vremena.
- Sigurnost – web aplikacija mora imati mogućnost autentifikacije korisnika kako bi ovisno o grupi korisnika omogućila ili onemogućila funkcionalnost
- Upotrebljivost – aplikacija mora biti intuitivna za korištenje svim korisnicima koji je žele koristiti

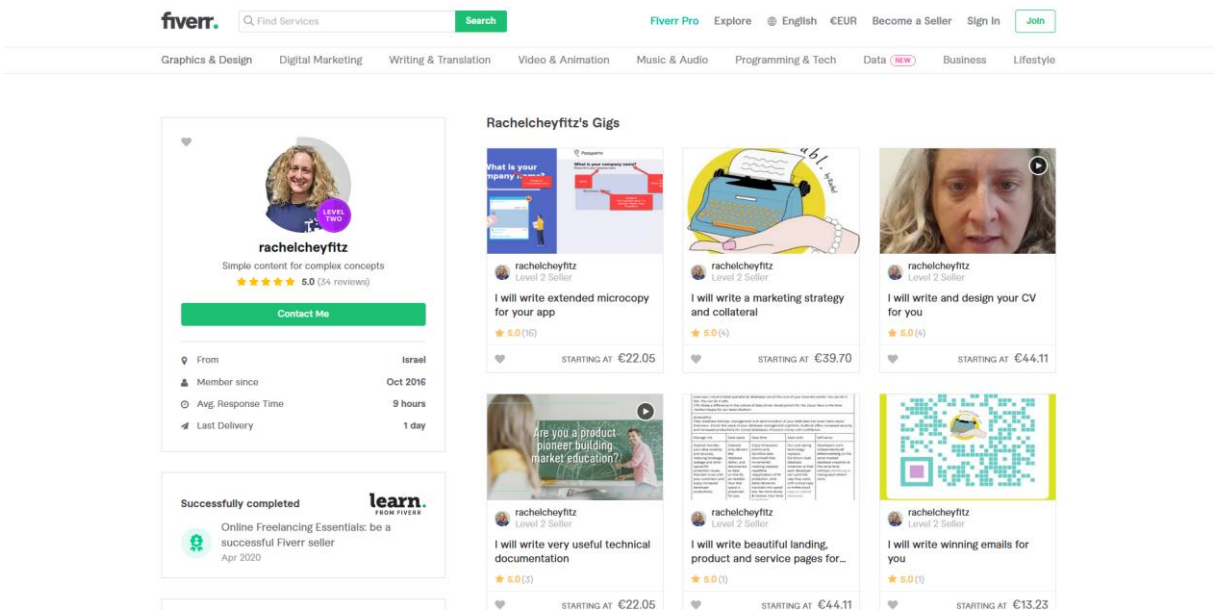
3.3. Pregled postojećih aplikacija

Traženje stručnjaka, pogotovo u digitalnoj domeni, je vrlo popularna praksa. Tako da već postoje mnogi primjeri aplikacija koje nude funkcionalnosti povezivanja sa stručnjacima. Prije dizajniranja vlastitog rješenja, dobra praksa je istražiti i konkurentna rješenja. Korisnici preferiraju (i čak očekuju) imati sličan set funkcionalnosti i slijeda funkcionalnosti kod stranica sličnog sadržaja i namjene. U nastavku će biti prikazane stranice s web aplikacije Fiverr.

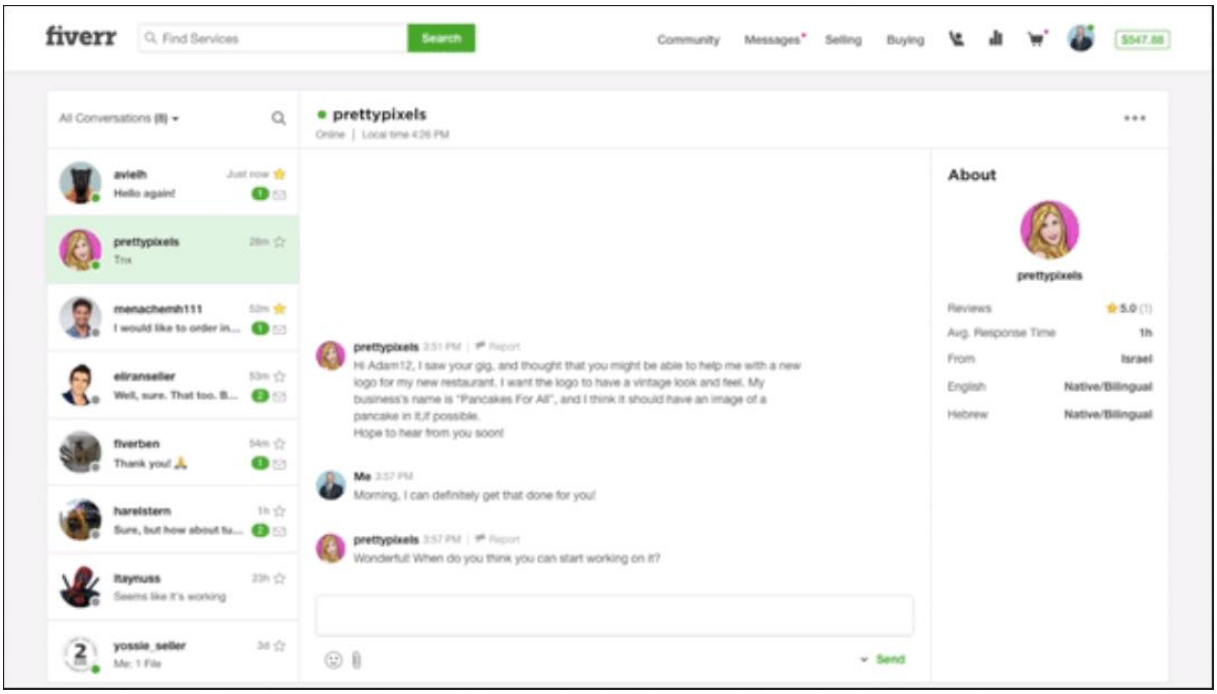
Slika 4. Lista stranica za pronalazak stručnjaka (https://penzit.com/top-10-best-freelance-websites/)



Slika 5. Pretraga korisnika stranice Fiverr (https://www.fiverr.com)



Slika 6. Primjer pregled korisničkog profila (https://www.fiverr.com/rachelcheyfitz)



Slika 7. Primjer sustava dopisivanja (https://assets-global.website-files.com/606a802fcaa89bc357508cad/606fcf4459ab5e50e297a66d_New-Inbox-GIF.gif)

Iz priloženih slika stranice Fiverr se mogu razaznati korisnički zahtjevi kakvi će biti implementirani i u ovom radu. Ali s manje funkcionalnosti. Kao i kod Fiverr-a, ostale stranice također nude pretragu stručnjaka bez potrebe za izradom računa. Sve stranice nude mogućnost dopisivanja unutar same stranice. Tako da komunikacija može započeti u vrlo kratkom roku, bez potrebe da se inicijalno odmah moraju razmjenjivati kontakt podaci.

4. Dizajn

Iz dobivenih funkcionalnih i nefunkcionalnih zahtjeva se sada može početi s dizajnom aplikacije. Postoje bezbrojni načini i tehnologije kako se aplikacija može implementirati. U ovom poglavlju će se dati pregled tehnologije, arhitekture, baze podataka i plan sučelja koji će se u sljedećem poglavlju koristiti kao temeljni plan za programiranje aplikacije.

4.1. Tip aplikacije

U samom opisu aplikacije je navedeno da aplikacija za pronalazak stručnjaka mora biti web aplikacija. U slučaju da to nije navedeno, odluka se može donijeti na temelju nefunkcionalnih zahtjeva. Primjer su i nefunkcionalni zahtjevi za ovu aplikacija. Navedeno je da aplikacija mora biti pristupačna svim korisnicima koji joj žele pristupiti što dovodi do toga da se aplikaciji treba moći pristupiti preko određene domene. Navedeno je i da treba biti stalno dostupna što se u startu postigne ako je aplikacija na stabilnom serveru. Upotrebljivost je sljedeći zahtjev iz kojeg proizlazi da samo web aplikacija ima smisla. Aplikaciji se onda može pristupiti s bilo kojeg uređaja, mobitela, računala, tableta. Alternativa tome bi bili programski klijenti koje bi prvo trebalo instalirati. A onda bi oni komunicirali s aplikacijom na serveru. S obzirom na poslovnu domenu koju aplikacija rješava, to nije potrebno.

4.2. Tehnologije

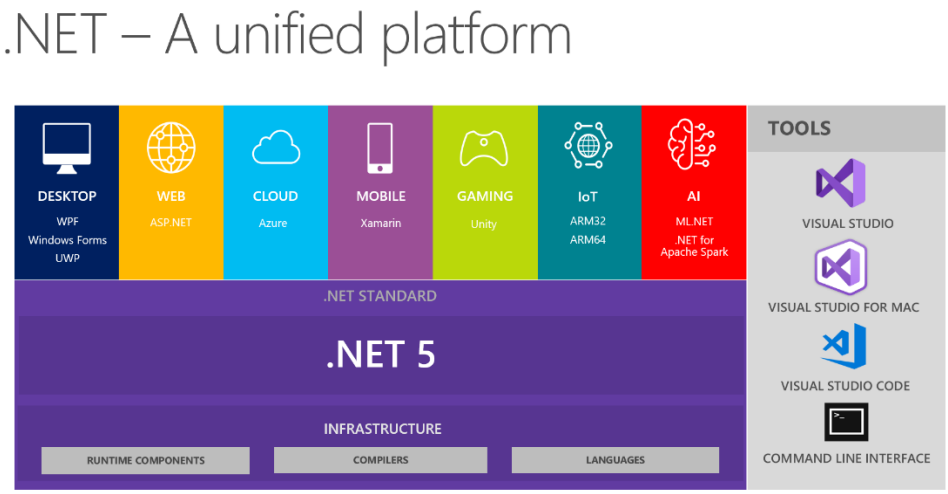
Odabir tehnologije, odnosno programskog okruženja je sljedeća bitna odluka koju je potrebno donijeti. Danas je dostupno puno kvalitetnih alata za razvoj. Ali kao što je to problem i u drugim profesijama tako je i u programiranju. Za određeni projekt treba koristiti odgovarajući alat. Tu postoje faktori o kojima treba razmisliti prije donošenja konačne odluke. Neki od faktora su tip aplikacije koja se razvija, popularnost određene tehnologije, održivost tehnologije u slučaju prebacivanja održavanja aplikacija na druge programere. Sve više programskih okruženja danas nudi višeplatformni razvoj. Odnosno, mogućnost razvoja desktop, web, mobilnih i ostalih aplikacija u jednom okruženju. Na programeru koji radi na projektu je da se odluči za tehnologiju s kojom može brzo i učinkovito realizirati aplikaciju. Popularnije tehnologije obično imaju i puno veću podršku od strane neke veće firme ili od online zajednice. To olakšava razvoj jer se lakše mogu naći ideje ili rješenja za potencijalne probleme tijekom programiranja. Kada se sve to uzme u obzir, donosi se konačna odluka o tehnologiji. U nastavku je pregled jezika, tehnologije i programskog okruženja koje će se koristiti u ovom projektu.

Primarni programski jezik ovog projekta je C#. To je moderni, objektno orijentirani, programski jezik sa sigurnim tipovima. Vuče korijene iz C jezika što ga čini lakše poznatim programerima koji su upoznati s C, C++, Java i JavaScript jezicima. Usko je vezan uz .NET razvojnu platformu. [5]

.NET je razvojna platforma koja developeru nudi alate i biblioteke koda potrebne za izgradnju različitih tipova aplikacija. Podržava razvoj za web, desktop, igre, internet stvari, cloud aplikacije i mnoge druge. Platforma je otvorenog koda i s licencom koja dopušta razvoj za komercijalnu svrhu bez dodatnih troškova. Također je *cross-platform* što znači da se .NET aplikacije mogu pokrenuti na raznim operativnim sustavima (Windows, Linux, MacOS, Android i drugi.) i različitim procesorskim arhitekturama (x86, x64, ARM). [6]

Platformu je izdao i primarno održava Microsoft. Prvi se put pojavila 2002. pod nazivom .NET Framework i podržavala je samo razvoj za desktop i web. Framework platforma nudi mnoge servise, širok set API-a, upravljanje memorijom, jezičnu interoperabilnost i razvojne okoline kao ADO.NET, ASP.NET, WCF, WinForms, WPF i druge. U početku se morala na sustav instalirati preko zasebnog instalacijskog paketa ali kasnije ju je Microsoft integrirao u Windowse. Zadnja verzija .NET Frameworka je 4.8. To je posljednja verzija platforme u tom obliku jer je Microsoft 2014. izdao kompletno novu platformu naziva .NET Core. Ta nova platforma je napisana ispočetka kako bi bila brza, modularna, *cross-platform* i pod ranije navedenom slobodnom licencom. Zadnjih nekoliko godina .NET Core se od verzije 1.0 pa do verzije 3.1 kao mlada razvojna platforma jako približila svim mogućnostima koje ima starija

platforma .NET Framework. S obzirom na mogućnosti Framework-a i njegovog dugogodišnjeg iskustva u IT industriji, to je veliki pothvat. Ove 2021. godine Microsoft je napravio i taj posljednji pothvat kada je izdao .NET verziju 5 (Core platforme) i time integrirao sve mogućnosti stare platforme u novu platformu. Time je platforma unificirana pod jednim imenom .NET. [6] [7]



Slika 8. .NET 5 moderna razvojna platforma
(https://devblogs.microsoft.com/dotnet/introducing-net-5/)

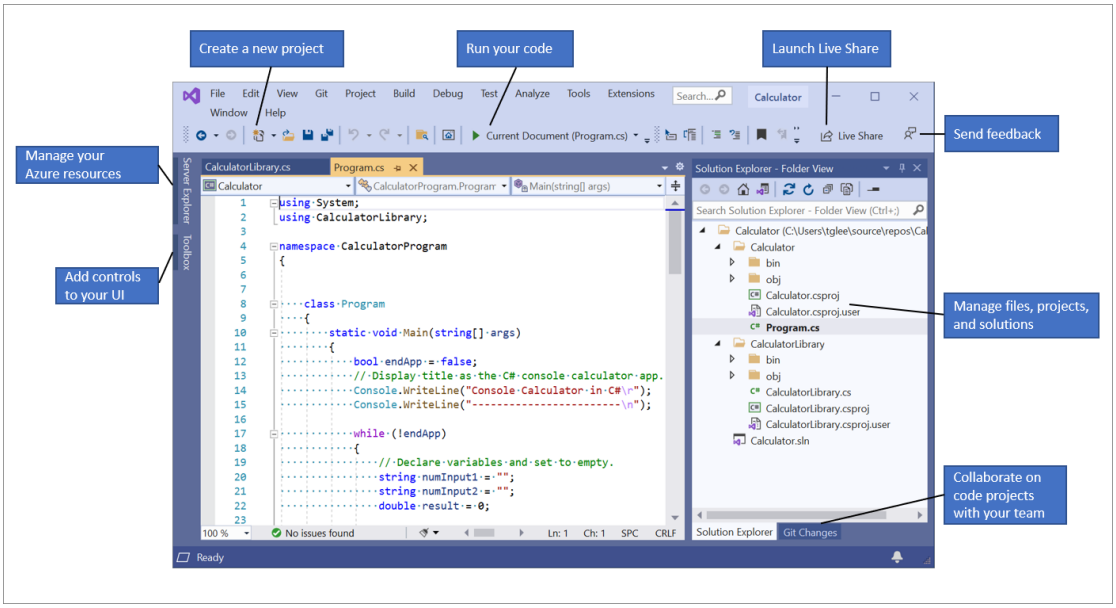
Upravo ta .NET 5 razvojna platforma će se koristiti za razvoj web aplikacije za pronalazak stručnjaka.

Za bazu podataka će se koristiti PostgreSQL. PostgreSQL je objektno-relacijski sustav baze podataka otvorenog koda. Koristi i proširuje SQL s mnogim značajkama za sigurno pohranjivanje i skaliranje najsloženijih podatkovnih opterećenja. Sustav je nastao 1986. godine u sklopu projekta POSTGRES na sveučilištu Berkeley i aktivno ga se razvija još i danas. Ima jako dobru reputaciju, dokazanu arhitekturu, pouzdanost, integritet podataka, široki spektar mogućnosti i veliku online zajednicu. [8]

Microsoft razvija svoju Microsoft SQL bazu podataka koja vrlo dobro radi sa .NET platformom. Ali za nju je potrebna licenca za naprednije opcije ili komercijalno korištenje. U kontekstu malih developera, PostgreSQL-ov otvoreni kod i licenca koja dopušta korištenje za komercijalne svrhe ga čini primamljivim odabirom za bazu podataka.

4.3. Alati

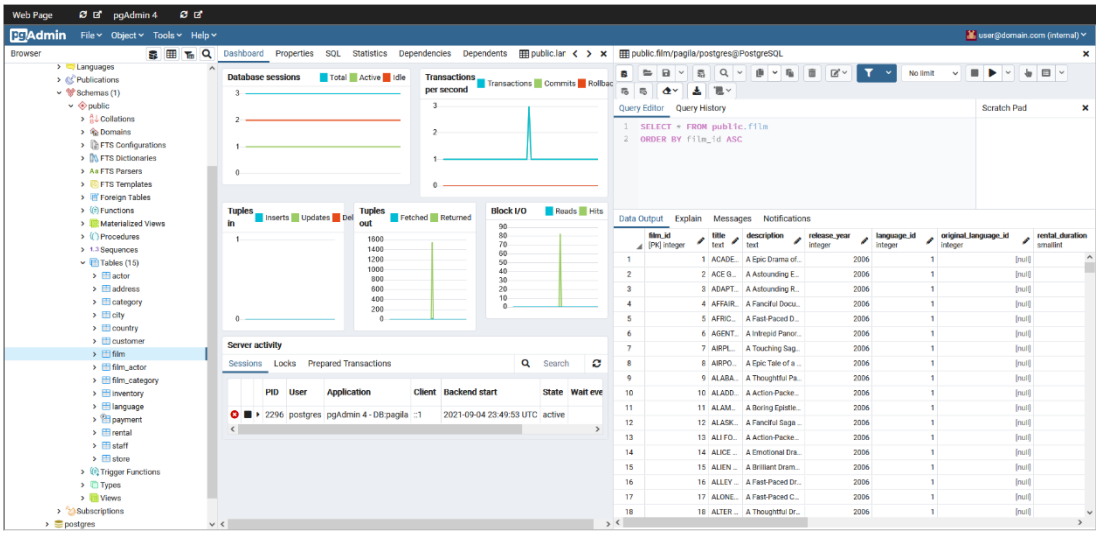
Za razvoj u .NET platformi će se koristiti alat Visual Studio IDE. To je integrirana razvojna okolina za izmjenu, debuggiranje, izgradnju koda i objavljivanje aplikacija. Sadrži razne mogućnosti kao što su razni kompajleri, alati za generiranje koda, grafički dizajneri i drugi alati za olakšavanje procesa razvoja aplikacija. [9]



Slika 9. Visual Studio IDE sučelje (https://docs.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2019)

U sklopu visual studija će se koristiti i sustav Git. Git je besplatni sustav otvorenog koda za kontrolu verzija distribuiranog koda. Brz je i učinkovit što mu omogućuje rad s malim i velikim projektima. Verzioniranje koda nije samo rezervirano za timove nego je dobra praksa i za pojedinačne developere. Razvoj aplikacije se može lakše pratiti. Potencijalne nove greške u kodu se mogu lakše proučiti pregledom izmjena u kodu. Dijeljenje koda s drugim stručnjacima je olakšano. Najpoznatija platforma koja koristi Git je GitHub (https://github.com/). [10]

Drugi važan alat je pgAdmin 4. pgAdmin je službeni alat za upravljanje PostgreSQL bazom podataka. Dostupan je na svim operativnim sustavima i može se pokrenuti u desktop i web verziji. Ima sve osnovne mogućnosti za upravljanje bazom podataka kao i ekvivalentni alati za druge baze. Jedina veća manjkavost mu je što nema mogućnost prikaza nikakvih dijagrama kompletne baze podataka ili grafičko editiranje pojedinačnih tablica. [11] Za generiranje dijagrama se može koristiti besplatni alat SchemaSpy.



Slika 10. pgAdmin 4 sučelje (https://www.katacoda.com/enterprisedb/scenarios/pgadmin-sandbox)

4.4. Arhitektura

4.4.1. Osnovni tipovi arhitekture

Postoje tri osnovna tipa arhitekture web aplikacija. To su monolitne aplikacije, servisno orijentirane aplikacije (*Software as a Service*) i aplikacije kao set mikro servisa.

Monolitna aplikacija je naziv za aplikaciju koja je u kontekstu ponašanja u potpunosti samostalna. U svom radu može sudjelovati s drugim aplikacijama ili bazama podataka ali je cijela jezgra njenog ponašanja unutar domene njenog procesa. Komponente monolitne aplikacije su međusobno jako povezane i međusobno ovisne. Ta samostalnost, ju čini vrlo popularnom za razvoj manjih aplikacija ili kao arhitektura za manje timove. Iako se danas na tu arhitekturu gleda kao tradicionalnu i zastarjelu, realnost je da je za mnoge projekte i dalje savršeno rješenje. [12] Prednosti su jednostavniji razvoj i objavljivanje aplikacije, s obzirom na to da su sve akcije aplikacije unutar jednog direktorija. Dodatne komponente kao upravljanje dnevnicima, ograničavanje prometa se mogu jednostavno integrirati jer je cijela aplikacija unutar jednog projektnog rješenja. Performanse su isto bolje, ako se usporedi s mikro servisima, jer nije potrebno čekati API komunikaciju sa setom mikro servisa. Negativne strane su što sav kod u jednom projektu može postati glomazan. To dovodi do komplicirane strukture koju je sve teže razumjeti i mijenjati. Nekoliko timova ne može raditi na istom projektu zbog konfliktnih izmjena. Prebacivanje projekta na novu tehnologiju često uključuje potrebu za pisanjem čitave aplikacije ispočetka. [13]

Servisno orijentirana arhitektura se odnosi na aplikaciju sastavljenu od diskretnih i slabo povezanih softverskih agenata koji obavljaju traženu funkciju. Koncept je da se aplikacija dizajnira i izgradi tako da su njeni moduli ponovno iskoristivi i besprijekorno integrirani. Prednosti su ponovna iskoristivost usluga aplikacije. Jednostavno održavanje jer su moduli samostalni. Veća pouzdanost jer je za traženje grešaka često potrebno testirati puno manji kod nego u monolitnoj aplikaciji. Paralelni razvoj jer su komponente neovisne. Negativna strana su komplicirano upravljanje raznim modulima i praćenje da njihova međusobna komunikacija ispravno radi. Dodatno opterećenje jer se sva interakcija između raznih komponenti, odnosno servisa, mora validirati. [13]

Mikro servisi su arhitektura u kojoj se aplikacija sastoji od niza autonomnih komponenti. Te komponente su povezane preko API-a. Kod mikro servisa usmjerenje je na poslovne prioritete i sposobnost dok je kod monolitnog pristupa usmjerenje na tehnološke slojeve, korisnička sučelja i baze podataka. Prednosti su jednostavni razvoj pojedinačnih servisa i njihovo održavanje. Bolja agilnost kod razvoja jer više timova može paralelno raditi na zasebnim servisima. Lakša horizontalna skalabilnost jer se kapaciteti pojedinačne usluge šire

umjesto kompletne aplikacije. Negativne strane su veća kompleksnost kod planiranja i praćenja svih zasebnih usluga. Ispravno sinkroniziranje podataka između zasebnih servisa s obzirom na to da imaju zasebne baze podataka. Sigurnosni problemi jer se svakom servisu može pristupiti preko API-a. Problem ako dođe do diversifikacije u korištenim tehnologijama i programskim jezicima korištenim za razvoj pojedinačnih mikro servisa. [13]

Zaključak bi bio da je monolitni pristup idealan za nove projekte na kojima radi manji tim koji treba brzo izdati aplikaciju. Servisno orijentirana arhitektura prikladna za složene poslovne sustave. Mikro servisi za razvoj složenih sustava koji su dovoljno veliki da se mogu razbiti na usluge i na kojima bi radili iskusni timovi.

Aplikacija za pronalazak stručnjaka će biti napravljena u monolitnoj arhitekturi. Argumenti su da aplikacija nije dovoljno složena da bi se dijelila na manje servise i da na njoj neće raditi nekoliko timova programera nego samo jedan. Nije potrebno povećavati kompleksnost razvoja aplikacija ako za to nema potrebe.

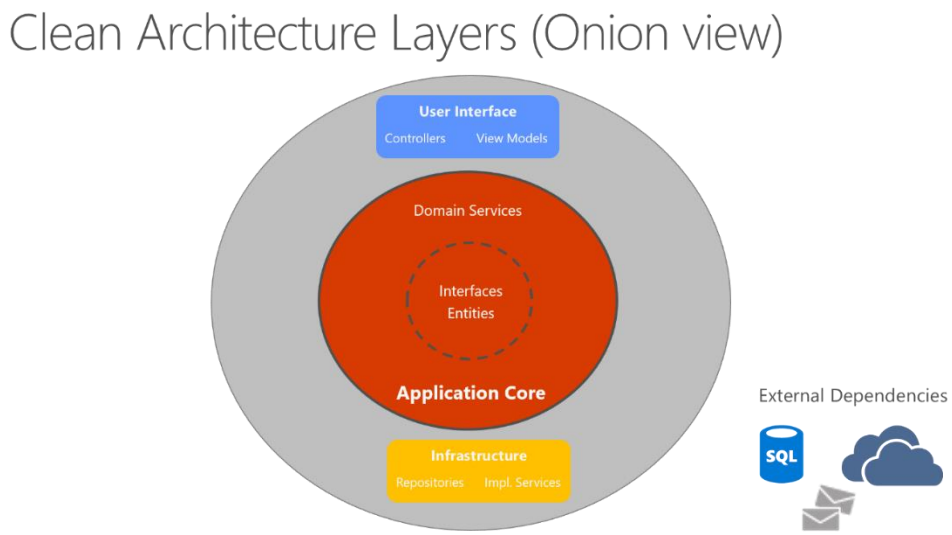
4.4.2.Čista arhitektura

Negativnost monolitne aplikacija je da sa širenjem funkcionalnosti postaje sve kompleksnija i kompleksnija. Popularno rješenje za taj problem je da se dijelovi aplikacije razdvoje ovisno o njihovim odgovornostima i ulogama u kodu. To pomaže organizaciji koda ali ta slojevita arhitektura nije jedina prednost. Organiziranjem koda u slojeve, funkcionalnosti „niže“ razine se mogu ponovo iskoristiti. To dovodi do standardizacije dijelova aplikacije i u konačnici do enkapsulacije koda. Funkcionalnosti unutar aplikacije se mogu lakše mijenjati bez da se drastično utječe na ostatak aplikacije. Primjer je da u sloju koji je zadužen za dohvat podataka promijenimo da umjesto da komunicira direktno s nekom SQL bazom, dohvaća podatke preko nekog API-ja. Slojevi aplikacije koje ovise o tim podacima ne moraju znati kako se do tih podataka došlo. [12]

Termin za takvu slojevitu arhitekturu je n-slojevita arhitektura. Uloge slojeva se mogu podijeliti na UI (korisničko sučelje), BLL (sloj poslovne logike) i DAL (sloj za dohvat podataka). Korisnik komunicira sa sučeljem koje komunicira s poslovnom logikom. Poslovna logika komunicira sa slojem za dohvat podataka koji onda izvršava određene naredbe. To je do sada bio tradicionalni pristup slojevitoj arhitekturi web aplikacija ali se pokazalo da ima jednu „veliku“ negativnost. Poslovna logika je previše ovisna o podatkovnom sloju. Primjer je da je za testiranje poslovnih funkcionalnosti potreban pristup podatkovnom sloju i testnoj bazi. [12]

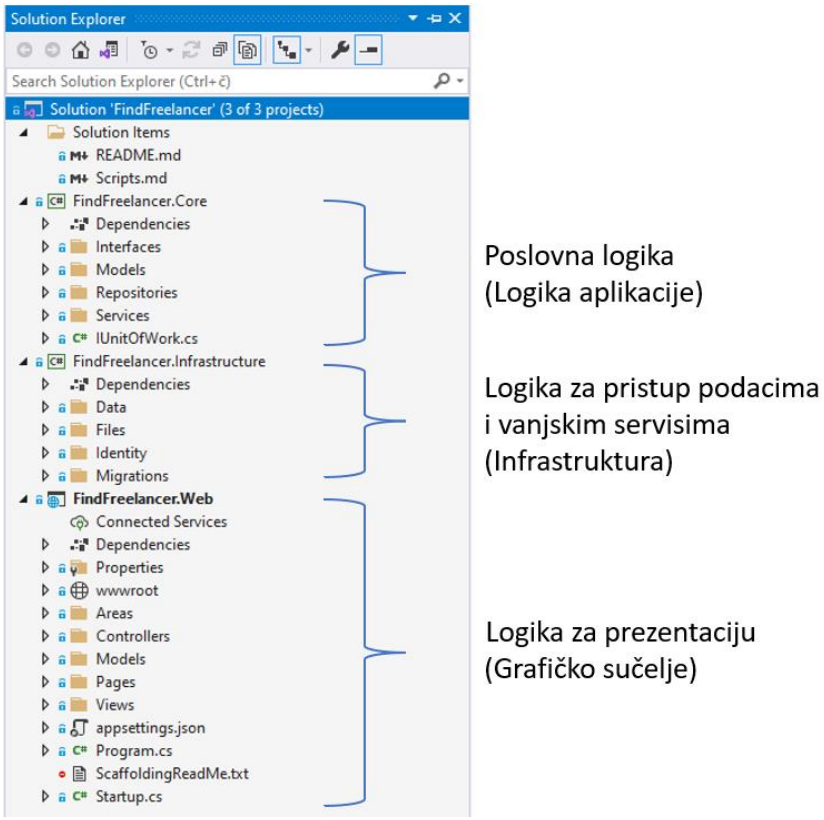
Rješenje tom problemu prevelike ovisnosti jednog sloja o drugom je pojam načela inverzije ovisnosti (*Dependency Inversion Principle*). To je načelo ključno za izgradnju labavo povezanih dijelova aplikacije. Aplikacija je implementirana tako da su detalji implementacije

komponenti ovisni o apstrakcijama više razine, a ne obrnuto. Naziv za takav uzorak dizajna je *Dependency Injection*. Arhitektura koja se dobiva tim pristupom ima razne nazive. Ali često se može naći pod nazivom čista (*Clean*) arhitektura ili arhitektura luka (*Onion Architecture*). [12]



Slika 11. Prikaz čiste arhitekture. (https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures)

Čista arhitektura stavlja poslovnu logiku i modele u jezgru aplikacije. Infrastruktura i implementacija ovise o toj jezgri. Kao što je to već ranije spomenuto, to se postiže apstrakcijama više razine. U jezgri aplikacije se definiraju apstrakcije ili sučelja koje se implementiraju u višim slojevima. Najčešće u infrastrukturnom sloju jer je on zadužen za pristup podacima. [12]



Slika 12. Organizacija koda u čistoj arhitekturi [autorski rad]

U čistoj arhitekturi slojevi su ovisno o odgovornostima podijeljeni na jezgru (*Core*), infrastrukturu, i sučelje. Jezgra aplikacije sadrži čitavu poslovnu domenu. To uključuje entitete, usluge, sučelja. Sučelja sadrže apstrakcije koje se implementiraju u infrastrukturi. To uključuje pristup datotečnom sustavu, pristup mreži, pristup podacima i druge. Ako su za potrebe aplikacije potrebni i jednostavni objekti za prijenos podataka (*DTO*), oni se isto definiraju u jezgri. Infrastrukturni sloj obično implementira pristup podacima. Tu su definirani i objekti za migracije, klase koje implementiraju pristup podacima kao i uzorak dizajna repozitorija. Posljednje imamo i UI sloj odnosno korisničko sučelje. To je sama aplikacija cijelog projekta. Sve akcije koje radi, radi preko sučelja definiranih u jezgri aplikacije. [12]

4.5. Dizajn baze podataka

Kao što je to navedeno i u opisu rada, za bazu podataka će se koristiti PostgreSQL baza podataka. Po svojoj ulozi tablice možemo podijeliti u dvije kategorije. Prva kategorija su tablice s prefiksom AspNet koje su auto generirane od strane gotovog paketa za autentifikaciju i autorizaciju .NET Identity. Druga kategorija su tablice koje se bave poslovnom domenom aplikacije.

Tablica 6. AspNet Identity tablice

Tablica	Opis
AspNetUsers	Tablica koja sadrži registriranog korisnika.
AspNetRoles	Pomoćne tablice za tablicu AspNetUsers koje spremaju podatke o pravima pojedinačnog korisnika, tokenima za login.
AspNetUserRoles	
AspNetRoleClaims	
AspNetClaims	
AspNetUserClaims	
AspNetLogins	

Izvor [autorski rad]

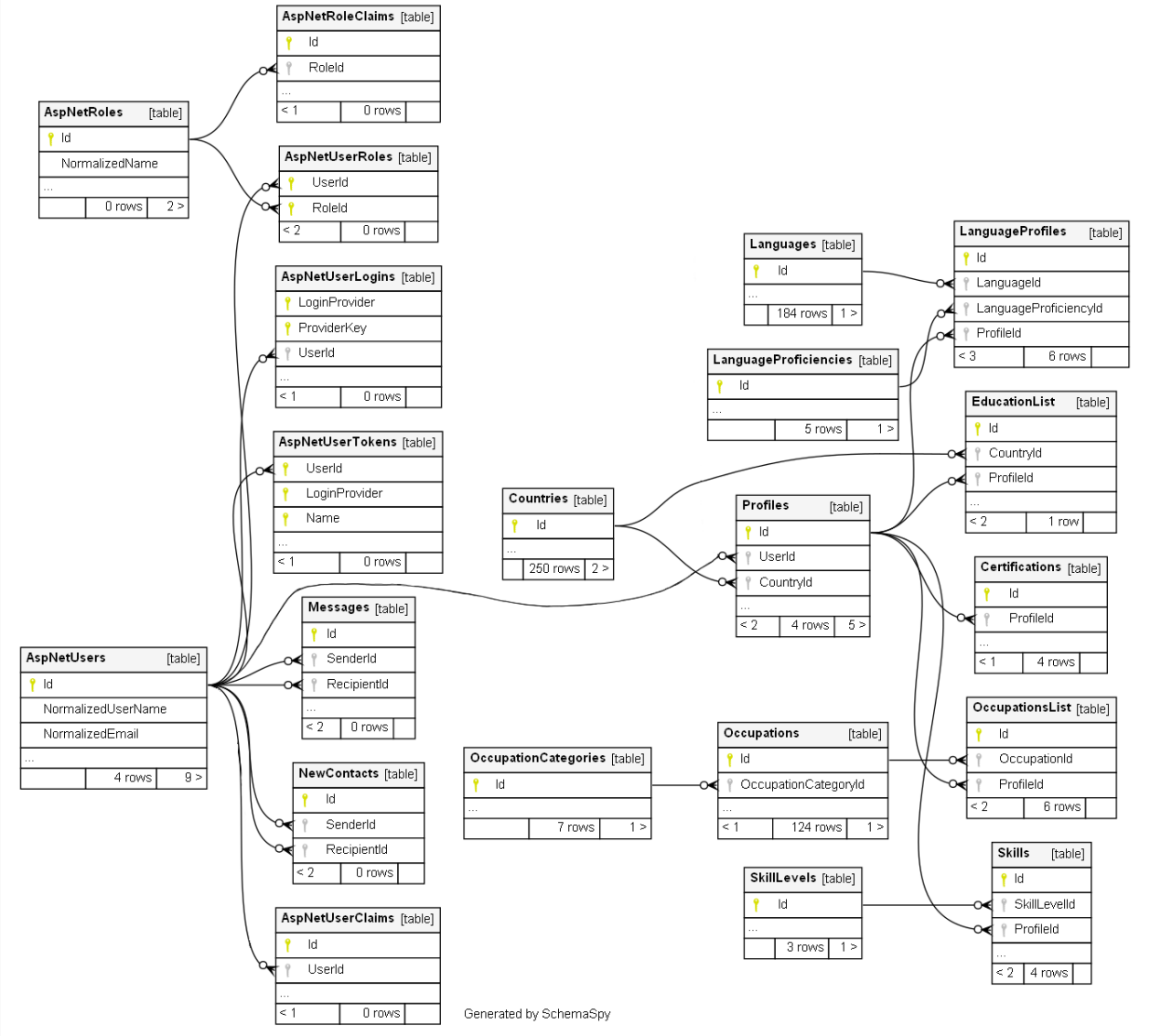
Baza podataka je zamišljena da svaki korisnik ima svoj korisnički profil. Profil sadrži primarni ključ korisnika kao strani ključ. Na profilu se nalaze atributi ime i prezime korisnika, zemlja podrijetla, datum registracije na stranicu, osobna web adresa, profilna slika. I liste za trenutni posao, poznavanje jezika, vještine, obrazovanje i dobiveni certifikati ili nagrade. Puna lista atributa tablica se nalazi u prilogu 1.

Tablica 7. Tablica domene sustava za pronalazak stručnjaka

Tablica	Opis
Profiles	Profil korisnika koji je vezan za registriranog korisnika.
LanguageProfiles	Među tablica za listu jezika i poznavanje jezika na profilu.
Languages	Lista svjetskih jezika.
LanguageProficiencies	Kategorije poznavanja jezika.
EducationList	Među tablica za listu edukacija na profilu.
Countries	Lista svih zemalja u svijetu.
Certifications	Lista certifikata ili nagrada na profilu.
OccupationsList	Među tablica za listu zaposlenja na profilu.
Occupations	Predodređena lista grupiranih zaposlenja za profil.
OccupationCategories	Lista kategorija zaposlenja.
Skills	Lista vještina na profilu.
SkillLevels	Kategorije stručnosti u vještini.
NewContact	Pomoćna tablica za sustav poruka. Omogućuje slanje poruka drugom korisniku bez da se odmah prikaže kontakt primatelju.
Messages	Spremište za poruke.

Izvor [autorski rad]

Na slici 13. je prikazan potpuni dijagram baze podataka koju će koristiti aplikacija.



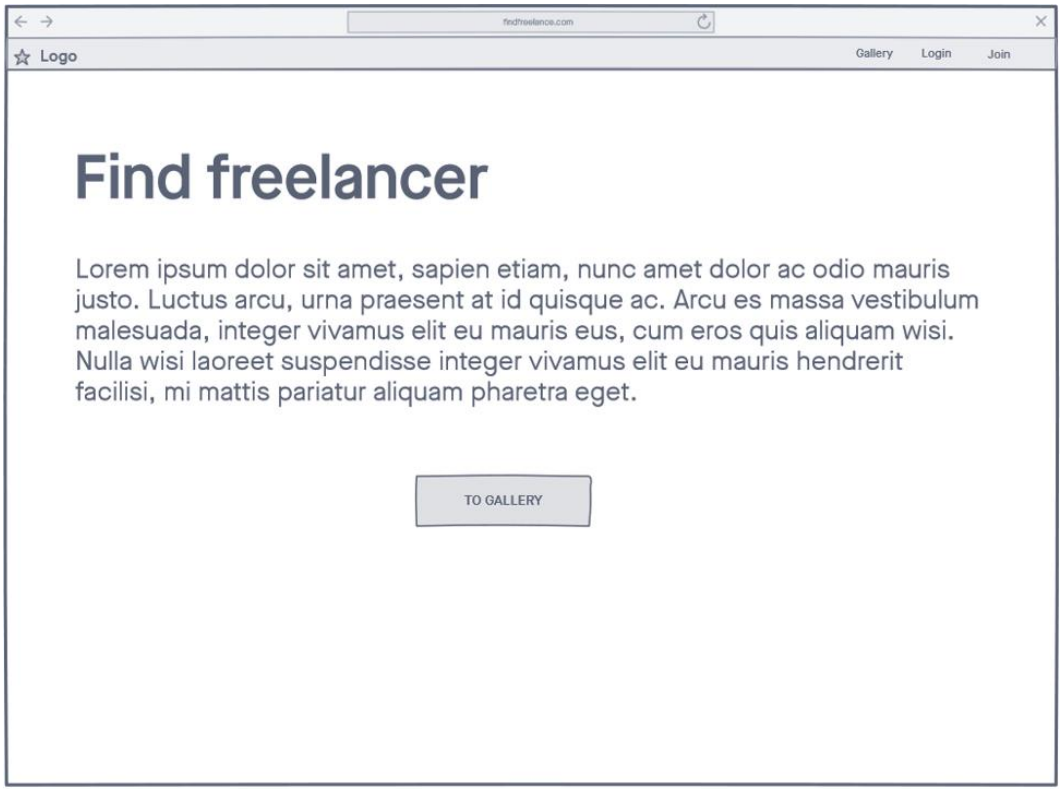
Slika 13. Dijagram tablica

4.6. Dizajn sučelja

Prije početka programiranja dobra je praksa stranice aplikacija prvo skicirati. Za izradu skica je potrebno razmisliti o kontrolama i načinu kako će se određene funkcionalnosti prikazati korisniku. Skicu je vrlo lako za mijenjati pa je svako takvo planiranje velika ušteda vremena kasnije, u fazi programiranja aplikacije.

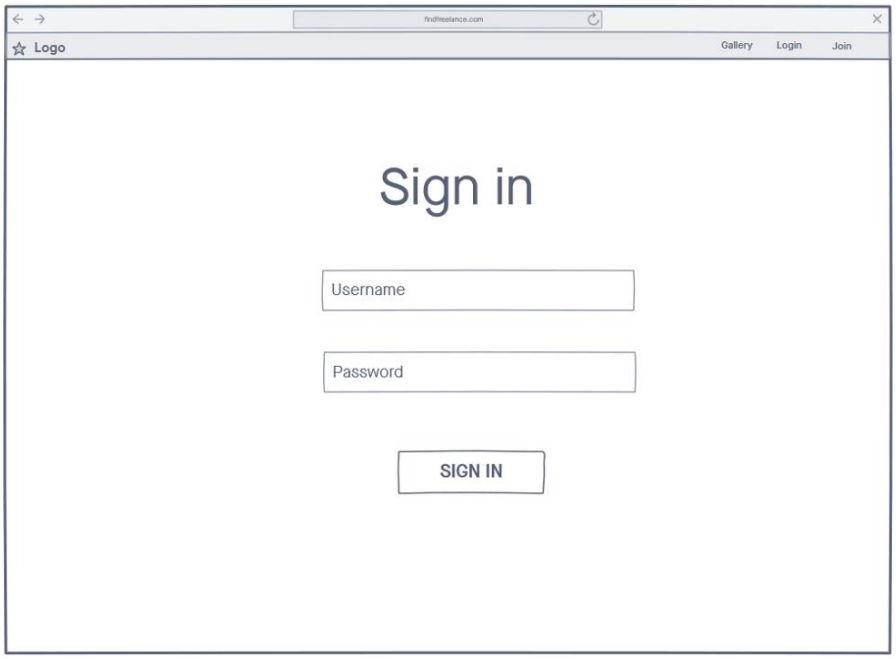
Skice aplikacije za pronalazak stručnjaka će biti napravljene u online besplatnom (dodatne opcije se plaćaju) alatu InvisionApp (https://www.invisionapp.com/). Invision nudi set mogućnosti skiciranja. Od skiciranja rukom do slaganja gotovih grafičkih elemenata. Za skice će biti korišteni gotovi elementi.

Aplikacija je zamišljena da ima početnu stranicu koja bi korisniku prikazala čemu aplikacija služi.

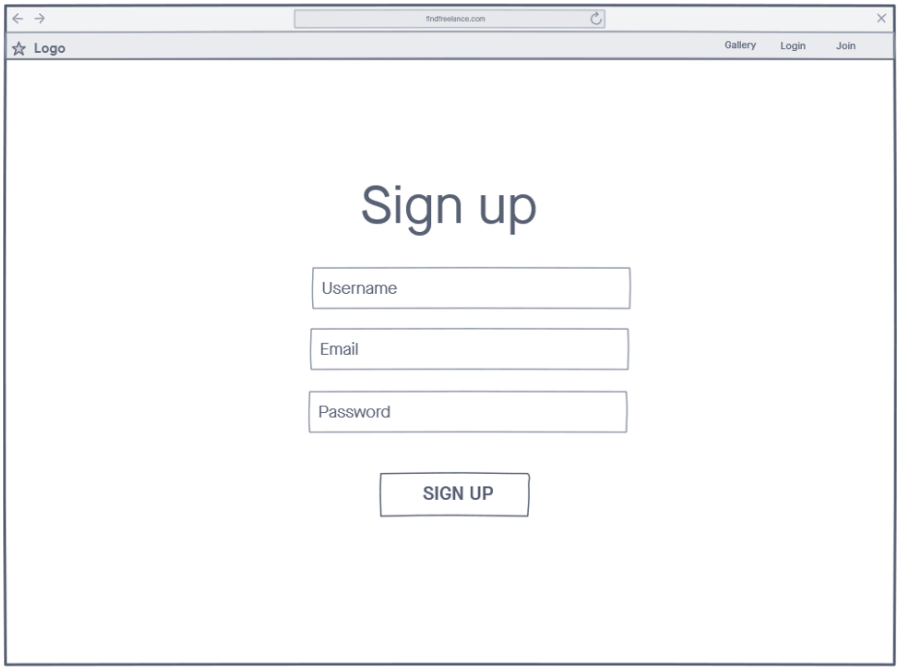


Slika 14. Skica početne stranice [autorski rad]

Stranica sadrži naslov web aplikacije, kratki opis aplikacije i poveznicu na galeriju profila stručnjaka. Na vrhu se nalazi navigacija u stanju kada korisnik nije ulogiran. Ponuđene opcije navigacije su odlazak u galeriju profila stručnjaka, registracija i prijava.

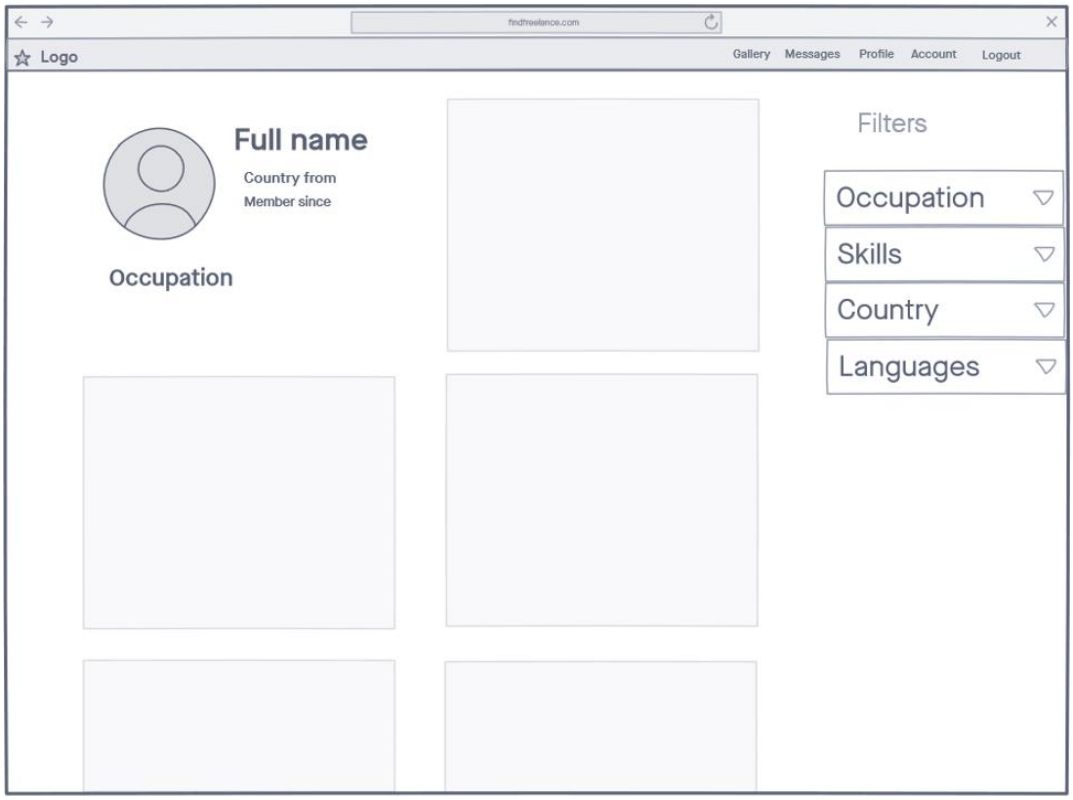


Slika 15. Skica stranice za prijavu [autorski rad]



Slika 16. Skica stranice za registraciju [autorski rad]

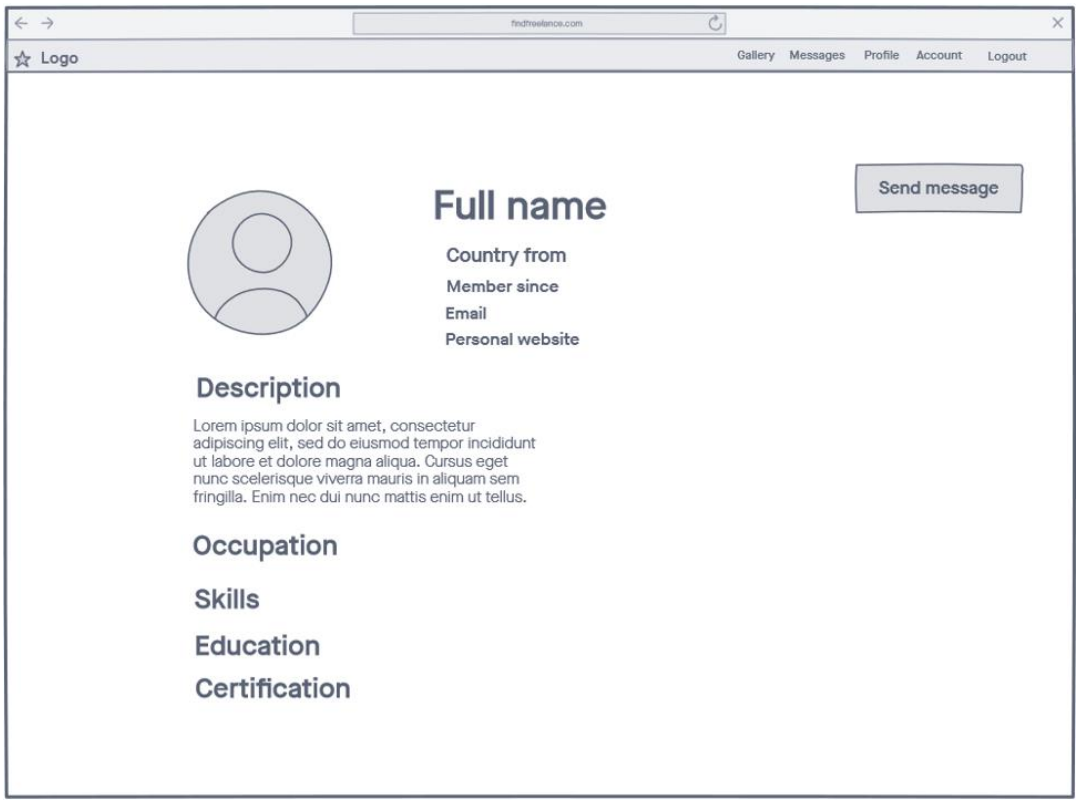
Na slikama 15. i 16. su stranice za prijavu i registraciju. Kod registracije se traži najosnovniji set podataka kako bi registracija bila što brža i jednostavnija.



Slika 17. Skica galerije [autorski rad]

Klikom u navigaciji ili na početnoj stranici se dolazi do galerija na slici 17. Galerija sadrži kartice svih korisnika koji su aktivirali svoj profil stručnjaka. Na karticama su prikazani osnovni podaci kao što su puno ime i prezime, zemlja podrijetla stručnjaka, kada je profil registriran, zaposlenje i profilna slika. S desne strane se nalaze filteri za filtriranje profila.

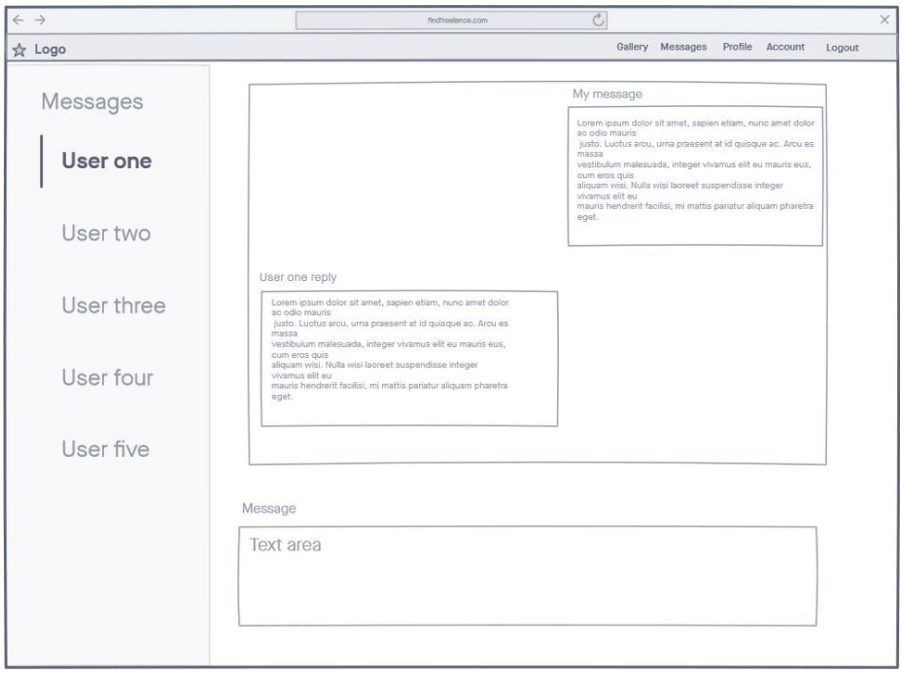
Navigacija na vrhu je sada u stanju prijavljenog korisnika. Poveznice su na galeriju, sustav poruka, upravljanje profilom prijavljenog korisnika, upravljanje korisničkim računom prijavljenog korisnika i odjava iz aplikacije.



Slika 18. Skica profila [autorski rad]

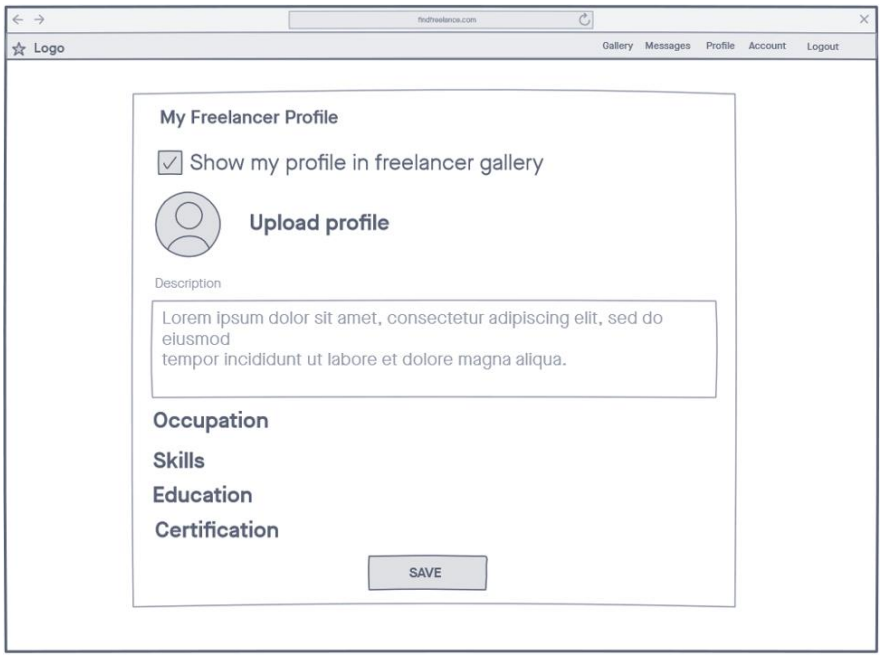
Klikom na karticu profila u galeriji se dolazi do detaljnog prikaza profila određenog stručnjaka. Tu su prikazane sve informacije o profilu koje se spremaju u bazu. Osim osnovnih informacija koje su već dostupne na karticama u galeriji tu se još može vidjeti link do osobne stranice stručnjaka, proizvoljni opis profila, lista zaposlenja, lista vještina, obrazovanje i certifikati.

Osim informacija, stranica ima i poveznicu za slanje poruke stručnjaku. Klikom na slanje poruke se otvara ekran za slanje poruke na stranici za poruke.



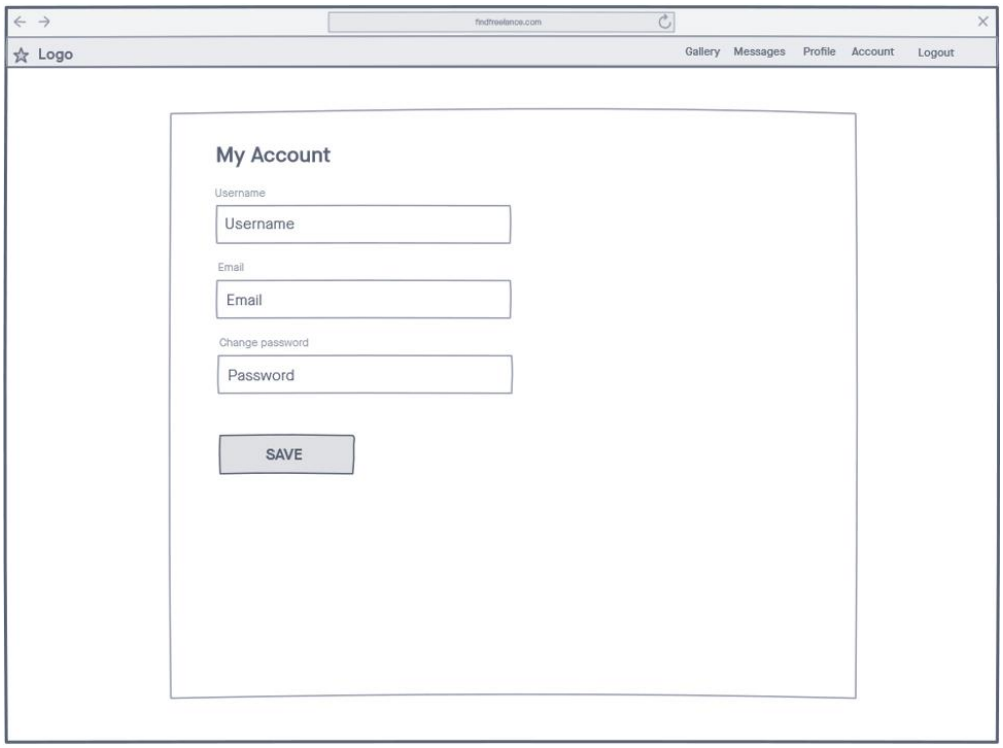
Slika 19. Skica stranice za razmjenu poruka [autorski rad]

Stranica sustava za poruke se sastoji od dva dijela. Lijevo se nalazi lista kontakata. Klikom na kontakt se otvara razgovor s njim. U centru ekrana se nalaze razmijenjene poruke s korisnikom u modernom izgledu chata. Ispod poruka se nalazi polje za pisanje i slanje poruka korisniku.



Slika 20. Skica stranice za izmjenu profila [autorski rad]

Sljedeća poveznica u navigaciji je do stranice za izmjenu osobnog profila. Stranica nudi izmjenu svih atributa profila osim onih koji se automatski računaju. Tu spada datum registracije. Opcija prikaza profila u galeriji može sakriti ili prikazati karticu profila u galeriji.



Slika 21. Skica stranice za izmjenu korisničkog računa [autorski rad]

Zadnja skica je skica stranice za izmjenu korisničkog računa. Stranica omogućuje promjenu šifre i potencijalno promjene drugih sigurnosnih postavki. S obzirom na to da se ne radi o opcijama koje su vezane za profil stručnjaka korisnika, te opcije su odijeljene na ovu zasebnu stranicu.

Posljednja stavka dizajna je nešto što svaka stranica mora imati, a to je logo. Logo za aplikaciju je generiran preko besplatnog online generatora logotipova hatchful.shopify (https://hatchful.shopify.com/).

Slika 22. Logo stranice [autorski rad]

Logo će se koristiti u navigaciji stranice kao poveznica na početnu stranicu.

5. Izrada aplikacije

U ovom poglavlju će biti prikazani ključni dijelovi implementacije web aplikacije za pronalazak stručnjaka. Ključni dijelovi su oni koji implementiraju neku veću funkcionalnost ili koji su bitni za organizaciju koda po pravilima čiste arhitekture. U visual studio alatu je otvoreno novo rješenje projekta (*Solution*) pod nazivom FindFreelancer. Rješenje projekta se sastoji od .NET 5 web aplikacije koja reprezentira grafičko sučelje za korisnika i dvije biblioteke koda koje reprezentiraju sloj arhitekture jezgre i infrastrukture. Struktura projekta je prikazana već ranije u radu na slici 12.

5.1. Autentifikacija

Autentifikacija i autorizacija web aplikacije su implementirane pomoću .NET Identity paketa. To je gotovi sustav članstva za .NET aplikacije s provjerom autentičnosti i autorizacije korisnika. Prilagođen je da se može koristiti za sve moderne aplikacije od web aplikacija do mobilnih. Generira stranice za prijavu, odjavu, upravljanje profilom i sigurnosnim postavkama. Pruža moderne sigurnosne značajke kao što su dvofaktorna provjera autentičnosti, automatsko zaključavanje računa, provjera računa preko email potvrde, brisanje podataka računa i druge. [14] Sve značajke se mogu jednostavno prilagoditi. Preporuka je da ga se koristi jer pruža siguran i brz način implementiranja sigurnih korisničkih računa, što ne mora biti slučaj ako ih implementiramo sami.

```
services.AddDefaultIdentity<IdentityUser>(options => {
    options.SignIn.RequireConfirmedAccount = false;
    options.Password.RequireUppercase = false;
    options.Password.RequireDigit = false;
    options.Password.RequireNonAlphanumeric = false;
}).AddEntityFrameworkStores<ApplicationDbContext>();
```

Priloženi kod prikazuje konfiguriranje Identity paketa tako da se zadane postavke Identity servisa zamjene drugim postavkama. Sve globalne izmjene servisa se u .NET web aplikacijama postavljaju u funkciju *ConfigureServices* u klasi Startup. Posljednja linija postavlja da se tablice baze podataka za autentifikaciju kreiraju u bazi FindFreelancer aplikacije. Više detalja o EntityFramework i ApplicationDbContext (kontekst koji reprezentira cijelu bazu) će biti u sljedećem poglavlju.

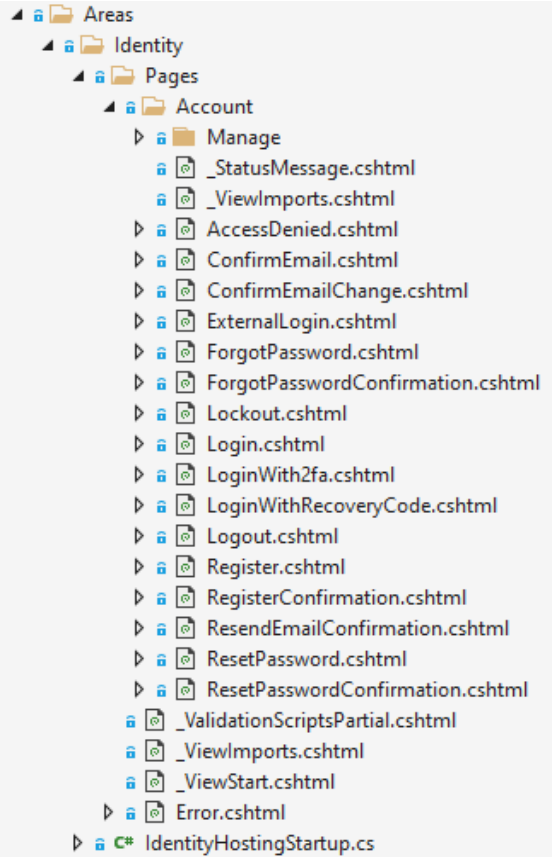
Za punu mogućnost izmjena stranice prijave i registracije, potrebno je zadane stranice prvo generirati u projektu. Sustav koristi svoje gotove stranice, ali je ostavljena mogućnost programeru da te stranice može generirati i izmijeniti. Za to se koristi alat dotnet-aspnet-codegenerator kojeg je prvo potrebno instalirati ubacivanjem sljedeće komande u PowerShell konzolu u korijenu projekta.

```
dotnet tool install -g dotnet-aspnet-codegenerator
```

Zatim se može koristiti sljedeća komanda za generiranje stranica.

```
dotnet aspnet-codegenerator identity -dc
FindFreelancer.Infrastructure.Data.ApplicationDbContext --files
"Account.AccessDenied;Account.Register;Account.Login;Account.Logout"
```

Komanda generira stranice za nedozvoljen pristup, registraciju, prijavu i odjavu. Prilagođena je za trenutni projekt jer joj se kao parametar daje lokacija konteksta baze podataka. Ako se izbriše parametar *files*, alat će generirati sve moguće stranice Identity sustava.



Slika 23. Generirane stranice Identity sustava [autorski rad]

Na slici je prikaz svih mogućih generiranih stranica. Pojedinačne stranice je onda moguće, po volji programera, mijenjati.

5.2. Entity Framework

Entity Framework je ORM (*Object - relational mapper*) okvir koji omogućuje programerima rad s podacima pomoću objekata klasa specifičnih za domenu bez fokusiranja na tablice baze podataka i stupce gdje se podaci pohranjuju. [15] Rad s podacima se radi na višoj razini apstrakcije. Nije potrebno razmišljati o načinima kako se neki podatak preuzima ili sprema u bazu. Time je i potreban kod za upravljanje podacima u aplikaciji manji.

Postoje dva načina rada s Entity Framework-om. Prvi način je način rada gdje baza dolazi prva (*database first)*. Klase objekta i postavke za postojeću bazu podataka se generiraju iz baze podataka u projekt. Za svaku izmjenu u shemi baze je klase potrebno opet generirati. Drugi način rada je gdje kod dolazi prvi (*code first*). Klase objekta i postavke za bazu se izrade u projektu i onda se migriraju u bazu. Svaka promjena nad klasama zahtijeva da se migracijom ažurira shema baze.

U ovom projektu će se koristiti način rada gdje kod dolazi prvi. Prvi korak je izraditi modele koji će u bazu predstavljati tablice. Modeli se dodaju u biblioteku Core koja predstavlja sloj jezgre projekta. S obzirom na to da sve klase imaju atribut Id kao primarni ključ, napravi se apstraktna klasa koja predstavlja osnovni entitet. Njega druge klase nasljeđuju.

```
public abstract class BaseEntity
{
	[Key]
	[Required(ErrorMessage = "Field is required.")]
	[DatabaseGenerated(DatabaseGeneratedOption.Identity)]
	public int Id { get; set; }
}
```

Zatim se naprave i sve ostale klase za profil, jezike, države, vještine, obrazovanje itd.

```
public class Skill : BaseEntity
{
	[Required]
	[Display(Name="Skill Name")]
	public string Name { get; set; }
	[Required]
	[Display(Name="Skill Level")]
	public SkillLevel SkillLevel { get; set; }
	public Profile Profile { get; set; }
	public Skill(){}
}
```

Iznad je primjer za klasu vještine. Klasa ima atribute za ime vještine, razinu vještine i kojem korisničkom profilu pripada. U klasu je dodan i generički prazni konstruktor.

```
public class ApplicationDbContext : IdentityDbContext
{
      public ApplicationDbContext(DbContextOptions<ApplicationDbContext>
options) : base(options)
      public DbSet<Certification> Certifications { get; set; }
      public DbSet<Country> Countries { get; set; }
      public DbSet<Education> EducationList { get; set; }
      public DbSet<Language> Languages { get; set; }
      public DbSet<LanguageProficiency> LanguageProficiencies {get; set;}
      public DbSet<LanguageProfile> LanguageProfiles { get; set; }
      public DbSet<Message> Messages { get; set; }
      public DbSet<NewContact> NewContacts { get; set; }
      public DbSet<OccupationCategory> OccupationCategories { get; set; }
      public DbSet<Occupation> Occupations { get; set; }
      public DbSet<OccupationProfile> OccupationsList { get; set; }
      public DbSet<Profile> Profiles { get; set; }
      public DbSet<SkillLevel> SkillLevels { get; set; }
      public DbSet<Skill> Skills { get; set; }
}
```

Sljedeće se u infrastrukturnom sloju napravi klasa *ApplicationDbContext* u kojoj se definiraju klase modela kao tablice u bazi. Identity je također ranije konfiguriran da svoje tablice poveže s tablicama u ovoj klasi.

```
services.AddDbContext<ApplicationDbContext>(options =>
    options.UseNpgsql(
          Configuration.GetConnectionString("DefaultConnection"),
          x => x.MigrationsAssembly("FindFreelancer.Infrastructure")));
```

Na kraju je samo potrebno konfigurirati poveznicu na server. Kod iznad se nalazi u klasi *Startup* u dijelu za konfiguriranje globalnih servisa. Isto mjesto gdje je i Identity konfiguriran, odnosno točno iznad. U kodu piše da je *ApplicationDbContext* shema za PostgreSQL bazu. Ovo je i jedino mjesto u cijeloj aplikaciji gdje je definirano s kojom bazom podataka se radi. U budućnosti, ako je potrebno, zamjenom samo tog koda, aplikacija može raditi s bilo kojom bazom podataka. Dalje je definirano da se na bazu može spojiti preko *DefaultConnection* varijable koja se nalazi u appsetting.json datoteci. Posljednje, zapisi o migracijama se trebaju spremati u infrastrukturni sloj.

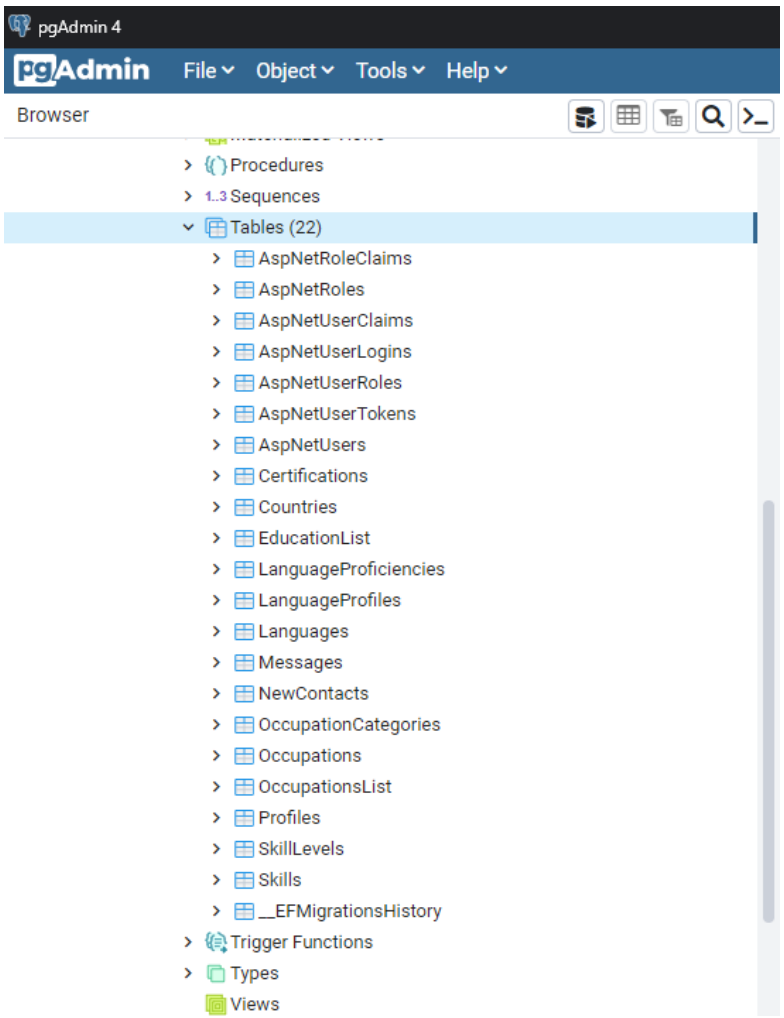
Projekt je sada spreman za migraciju baze podataka. Sljedećom naredbom u korijenu projekta se izradi nova migracija proizvoljnog imena.

```
dotnet ef migrations add MigrationName -s FindFreelancer.Web -p
FindFreelancer.Infrastructure
```

Migracija se zatim ažurira na samoj bazi podataka.

```
dotnet ef database update -s FindFreelancer.Web -p
FindFreelancer.Infrastructure
```

Rezultat su generirane tablice u bazi podataka. Ispravnost tablica se može provjeriti sa pgAdmin alatom.



Slika 24. Lista tablica u bazi podataka [autorski rad]

U slučaju da se u aplikaciju dodaju nove klase modela ili se napravi izmjene na postojećima, potrebno je ponoviti gornje dvije naredbe za migraciju i ažuriranje baze podataka.

5.3. Dependency Injection

Dependency Injection uzorak dizajna je integriran u .NET web aplikacije. A ranije u radu je spomenuto da je to i uvjet za implementaciju čiste arhitekture. Konfigurira se tako što se u funkciju *ConfigureServices* u *Startup* klasi dodaju novi servisi. Tamo je već dodan servis Identity za autentifikaciju i kontekst baze podataka. Ti servisi se zatim mogu koristiti na bilo kojoj stranici web aplikacije tako da se ubace u njen konstruktor. U nastavku je primjer za dohvaćanje profila stručnjaka na stranici galerija.

```
private readonly ApplicationDbContext _context;
private readonly UserManager<IdentityUser> _userManager;


public IndexModel(UserManager<IdentityUser> userManager,
ApplicationDbContext context)
{
      _userManager = userManager;
      _context = context;
}
```

Deklariraju se varijable za kontekst i Identity. Postavljanjem njih u konstruktor, .NET sustav zna da ih treba ubaciti metodom Dependency Injection.

```
public async Task<IList<Profile>> GetProfileGallery()
{
      return await _context.Profiles
      .Include(x => x.OccupationList)
      .ThenInclude(x => x.Occupation)
      .ThenInclude(x => x.OccupationCategory)
      .Include(x => x.Country)
      .Where(x => x.ProfileEnabled == true)
      .ToListAsync();
}
```

Profili se zatim mogu povući kao lista koristeći objekt kontekst. S obzirom na to da je profil složeni objekt koji se sastoji od drugih objekata i lista, potrebno je konfigurirati da se i ti objekti učitaju. To se u gornjem primjeru vidi sa sufiks naredbama *Include* i *ThenInclude*. Nije potrebno uzimati sve, uzimaju se samo potrebni objekti.

5.4. Repository

U prethodnom potpoglavlju je do kraja implementiran i konfiguriran Entity Framework kao glavni način rada s bazom podataka u web aplikaciji. Ali takav način implementacije je stvorio preveliku ovisnost između slojeva aplikacije. Bolji arhitekturni dizajn je da se prati čista arhitektura i da su slojevi više izolirani. To se postiže implementacijom uzorka dizajna *Repository* i *Unit of Work*. Prvi korak je implementirati generički repozitorij koji će sadržavati osnovne naredbe za kreiranje, ažuriranje i brisanje bilo kojeg entiteta. U jezgri (sloj) aplikacije se kreira sučelje *IRepository* s generičkim funkcijama.

```
public interface IRepository<TEntity> where TEntity : class
{
      ValueTask<TEntity> GetByIdAsync(int id);
      Task<IEnumerable<TEntity>> GetAllAsync();
      IEnumerable<TEntity> Find(Expression<Func<TEntity, bool>> predicate);
      Task<TEntity> SingleOrDefaultAsync(Expression<Func<TEntity, bool>>
predicate);
      Task AddAsync(TEntity entity);
      Task AddRangeAsync(IEnumerable<TEntity> entities);
      void Remove(TEntity entity);
      void RemoveRange(IEnumerable<TEntity> entities);
}
```

U infrastrukturnom sloju se te naredbe potpuno implementiraju koristeći kontekst.

```
public class Repository<TEntity> : IRepository<TEntity> where TEntity :
class {
protected readonly ApplicationDbContext _context;

public Repository(ApplicationDbContext context)
{_context = context; }

public async Task AddAsync(TEntity entity)
{await _context.Set<TEntity>().AddAsync(entity); }

public async Task AddRangeAsync(IEnumerable<TEntity> entities)
{await _context.Set<TEntity>().AddRangeAsync(entities);}

public IEnumerable<TEntity> Find(Expression<Func<TEntity, bool>> predicate)
{return _context.Set<TEntity>().Where(predicate);}
```

```
public async Task<IEnumerable<TEntity>> GetAllAsync()
{return await _context.Set<TEntity>().ToListAsync();}

public ValueTask<TEntity> GetByIdAsync(int id)
{return _context.Set<TEntity>().FindAsync(id); }

public void Remove(TEntity entity)
{_context.Set<TEntity>().Remove(entity); }

public void RemoveRange(IEnumerable<TEntity> entities)
{_context.Set<TEntity>().RemoveRange(entities); }

public async Task<TEntity> SingleOrDefaultAsync(Expression<Func<TEntity,
bool>> predicate)
{return await _context.Set<TEntity>().SingleOrDefaultAsync(predicate); }
}
```

Zatim se u jezgri implementira sučelje *IProfileRepository*. Tu će se nalaziti sve funkcije za upravljanje i pregled profila. Uključujući i funkcije galerije. U primjeru su prikazane samo prve tri funkcije.

```
public interface IProfileRepository : IRepository<Profile>
{
      Task<IList<Profile>> GetProfileGallery();
      Task<IList<Certification>> GetUserCertificationProfiles(int
profileId);
      Task<Profile> GetUserProfile(IdentityUser user);
}
```

Sučelje je zatim implementirano u infrastrukturnom sloju.

```
public class ProfileRepository : Repository<Profile>, IProfileRepository
{
      public ProfileRepository(ApplicationDbContext context) :
base(context) { }

public async Task<IList<Profile>> GetProfileGallery()
{return await _context.Profiles.Include(x =>
x.OccupationList).ThenInclude(x => x.Occupation).ThenInclude(x =>
x.OccupationCategory).Include(x => x.Country).Where(x => x.ProfileEnabled
== true).ToListAsync();}
```

```
public async Task<IList<Certification>> GetUserCertificationProfiles(int
profileId)
{return await _context.Certifications.Where(x => x.Profile.Id ==
profileId).ToListAsync();}


public async Task<IList<Education>> GetUserEducationProfiles(int profileId)
{return await _context.EducationList.Where(x => x.Profile.Id ==
profileId).Include(x => x.Country).ToListAsync();}
}
```

Time je glavni repozitorij pripremljen, sljedeći korak je implementirati *Unit of Work* koji će služiti kao sučelje za taj repozitorij. Sučelje za njega se implementira u sloju jezgre.

```
public interface IUnitOfWork : IDisposable
{
      IProfileRepository Profiles { get; }
      Task<int> CommitAsync();
}
```

Dok se pune funkcije implementiraju u infrastrukturnom sloju.

```
public class UnitOfWork : IUnitOfWork
{
      private readonly ApplicationDbContext _context;
      private ProfileRepository _profileRepository;

      public UnitOfWork(ApplicationDbContext context)
      {_context = context;

      public IProfileRepository Profiles => _profileRepository ??= new
      ProfileRepository(_context);

      public async Task<int> CommitAsync()
      {return await _context.SaveChangesAsync();}

      public void Dispose()
      {_context.Dispose();}
}
```

Sada je i *Unit of Work* završen. Preostaje napraviti servis *ProfileService* koji će se implementirati preko *Dependency Injection-a* i kojeg će same stranice koristiti za pristup bazi. I sučelje i puna klasa se implementiraju u sloju jezgre. Na primjeru će biti prikazane prve tri funkcije servisa.

```
public interface IProfileRepository : IRepository<Profile>
{
      Task<IList<Profile>> GetProfileGallery();
      Task<IList<Certification>> GetUserCertificationProfiles(int
profileId);
      Task<Profile> GetUserProfile(IdentityUser user);
}

public class ProfileService : IProfileService
{
      private readonly IUnitOfWork _unitOfWork;
      private readonly UserManager<IdentityUser> _userManager;

public ProfileService(IUnitOfWork unitOfWork, UserManager<IdentityUser>
userManager)
{
      _unitOfWork = unitOfWork;
      _userManager = userManager;
}

public async Task<IList<Profile>> GetProfileGallery()
{IList<Profile> profiles = await _unitOfWork.Profiles.GetProfileGallery();
return profiles; }

public async Task<Profile> GetUserProfile(IdentityUser user)
{Profile profile = await _unitOfWork.Profiles.GetUserProfile(user);
return profile; }

public async Task<IList<Certification>>
GetUserCertificationProfiles(ClaimsPrincipal principalUser)
{var profile = await GetUserProfile(principalUser);

IList<Certification> certificationProfiles = await
_unitOfWork.Profiles.GetUserCertificationProfiles(profile.Id);
return certificationProfiles; }
}
```

Preostaje registracija sučelja i klase servisa.

```
services.AddScoped<IUnitOfWork, UnitOfWork>();
services.AddTransient<IProfileService, ProfileService>();
```

Primjer korištenja na dohvatu kartica galerije.

```
private readonly IProfileService _profileService;

public IndexModel(IProfileService profileService)
{
      _profileService = profileService;
}
public IList< Profile> Profile { get;set; }

public async Task OnGetAsync()
{
      Profile = await _profileService.GetProfileGallery();
}
```

Dodavanjem svih ovih slojeva apstrakcije na onaj inicijalni jednostavni primjer dohvata kartica galerija preko *Entity Frameworka,* dobiva se isti rezultat. Postavlja se pitanje „da li je sva ta apstrakcija potrebna i vrijedna uloženog vremena?“. I to je zapravo glavni argument protiv implementiranja dodatnih uzoraka dizajna na *Entity Framework,* koji već sam po sebi ima sve potrebne mogućnosti i time se povećava kompleksnost projekta. Ali ako se gleda s aspekta projekta koji bi se mogao širiti, dobivena je izoliranost jezgre. Jezgra je sada neovisna o drugim slojevima. Može sadržavati svu poslovnu logiku. Što prije nije bilo moguće bez stavljanja dijela poslovne logike u infrastrukturni i aplikativnom sloj da bi sve radilo. Sada je moguće i provoditi automatska jedinična testiranja ispravnosti funkcija poslovne logike. Zamjena aplikacije nekom novom ili drugom vrstom (mobilna umjesto web) se svela samo na korištenje servisa za profil u toj novoj. U alternativnoj situaciji bi bilo potrebno kopirati, analizirati i odjeljivati dijelove koda koji su striktno poslovna logika od aplikacijskog koda prošle aplikacije.

5.5. Inicijalni podaci

U bazi podataka se trebaju nalaziti liste država, jezika i druge kategorije. To su podaci koji se neće mijenjati. Dobra ideja je da automatikom odmah budu upisani kod prvog pokretanja web aplikacije. Napravi se zasebna klasa u infrastrukturnom sloju s funkcijom koja sadrži sve podatke. U nastavku je primjer za razine poznavanja jezika.

```
private static IEnumerable<LanguageProficiency>
GetPreconfiguredLanguageProficiency()
{
	return new List<LanguageProficiency>()
	{
		new LanguageProficiency("Unspecified"),
		new LanguageProficiency("Basic"),
		new LanguageProficiency("Conversational"),
		new LanguageProficiency("Fluent"),
		new LanguageProficiency("Native/Bilingual"),
	};
}
```

Funkcija se zatim može pozvati. Napravi se provjera da nikakve promjene baze nisu u pripremi. Dohvati se sve iz tablice za razinu poznavanja jezika. Ako je lista prazna, popuni ju podacima.

```
context.Database.Migrate();
if (!await context.LanguageProficiencies.AnyAsync())
{
	await context.LanguageProficiencies.AddRangeAsync(
GetPreconfiguredLanguageProficiency());
	await context.SaveChangesAsync();
}
```

Za vrijeme pisanja ovog rada, postoji dugogodišnja greška u *Entity Frameworku* i PostgreSQL bazi podataka. Nakon dodavanja inicijalnih podataka na ovaj način, sekvence primarnih ključeva nisu ispravno sinkronizirane s *Entity Frameworkom* i nije moguće unositi složene objekte u bazu koji su vezane za tablice u koje su dodavani podaci. Postoji cijeli niz uspješnih i neuspješnih rješenja kojima se može doskočiti tom problemu. Ali možda najjednostavnije rješenje je koristiti sirove SQL upite za unos podataka. U nastavku je primjer za unos liste vještina nekog profila sirovim upitom.

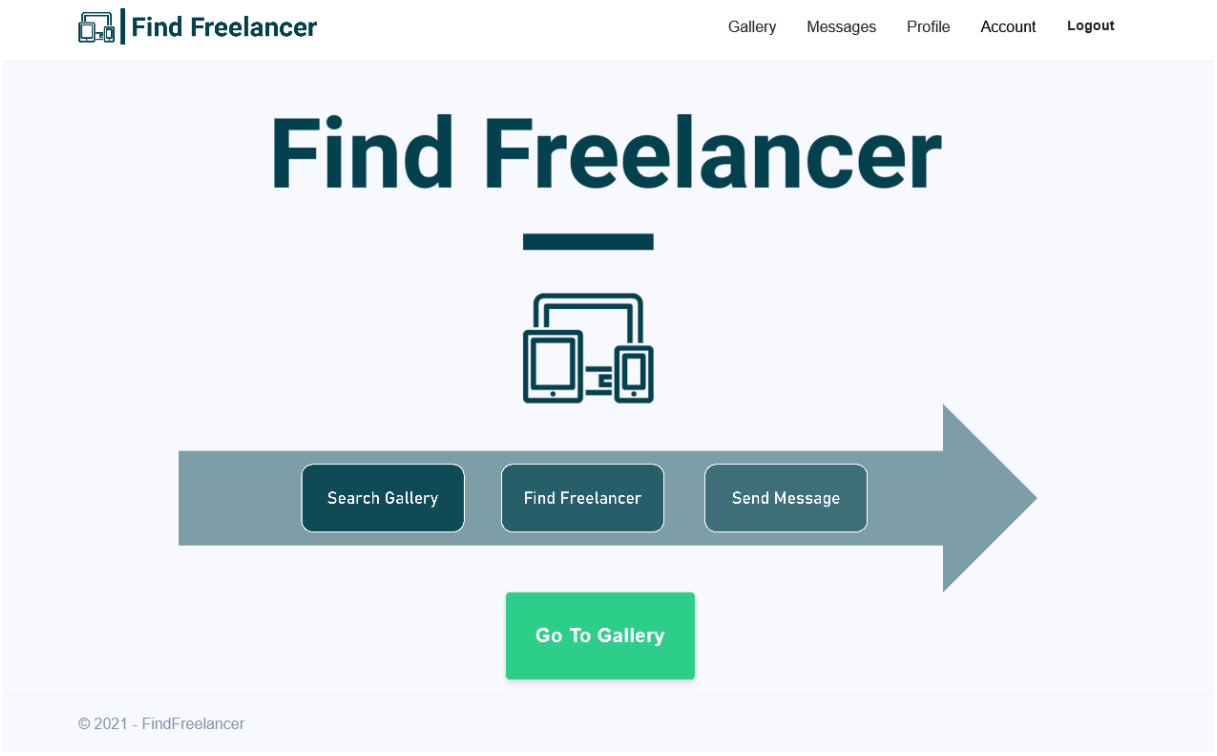
```
string commandText = $"INSERT INTO \"Skills\" VALUES
(default,'{Skill.Name}',{Skill.SkillLevel.Id},{Skill.Profile.Id})";
_context.Database.ExecuteSqlRaw(commandText);
await _context.SaveChangesAsync();
```

6. Testiranje

U ovom poglavlju će biti napravljen pregled kreirane web aplikacije i implementiranih funkcionalnosti. Napravit će se i kratka evaluacija grafičkog sučelja kao i ideja za daljnji smjer razvoja aplikacije. Za potrebe testiranja je otvoreno par korisničkih profila s nasumičnim podacima. Profilne slike korisnika nisu od stvarnih ljudi neko profilne slike ljudi generiranih umjetnom inteligencijom. Generator je dostupan na https://generated.photos.

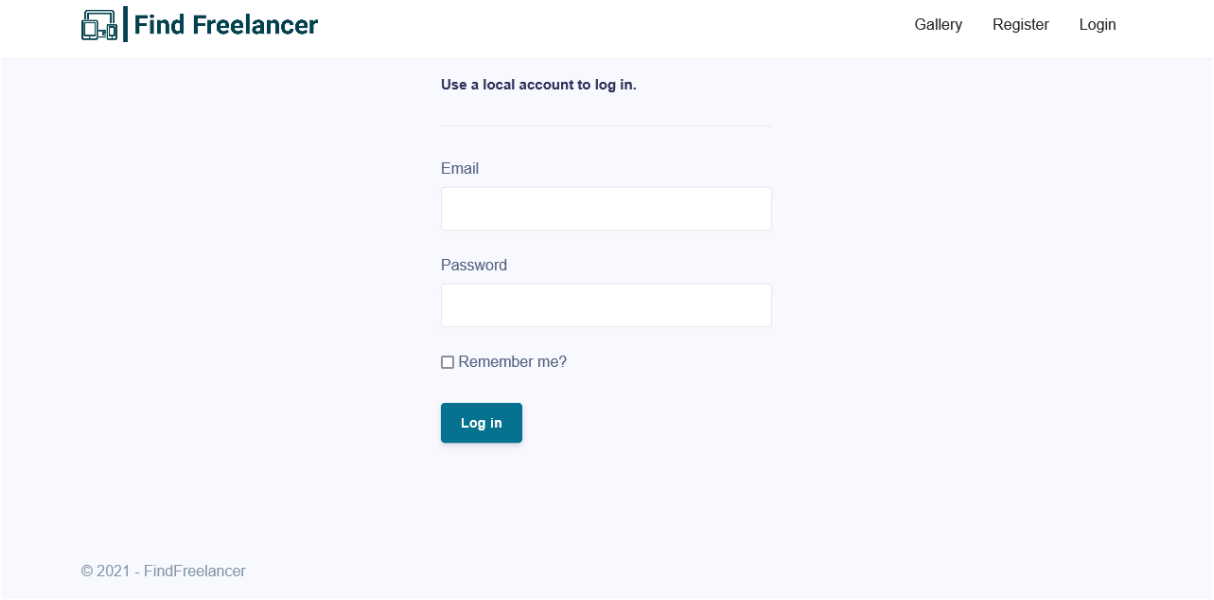
6.1. Pregled web aplikacije

Pokretanjem web aplikacije se prvo pojavi ekran početne stranice. Na stranici se nalazi veliki logo i naziv stranice. Zatim grafika koja korisniku preporučuje kako bi aplikaciju trebao koristiti. I posljednje, direktna poveznica na galeriju.

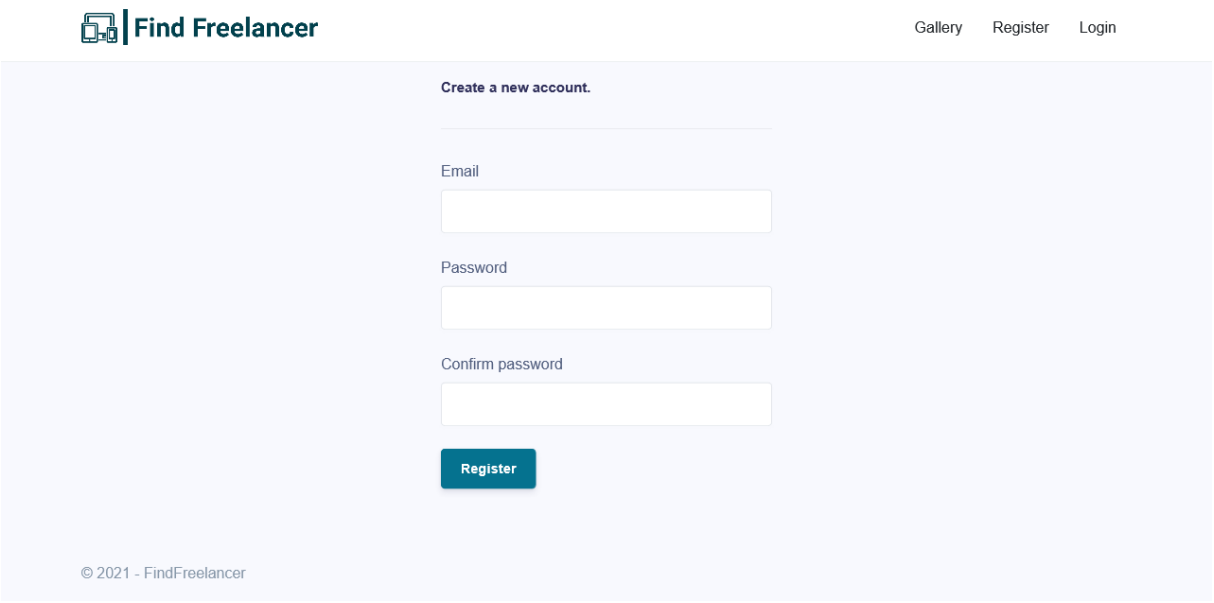


Slika 25. Početna stranica [autorski rad]

Ako korisnik želi na stranici koristiti neku funkcionalnost koja nije samo pregled galerije, potrebno je prijaviti se ili registrirati. Stranice su dio Identity paketa i prijava na njima je pojednostavljena da bude brza i bez problema.

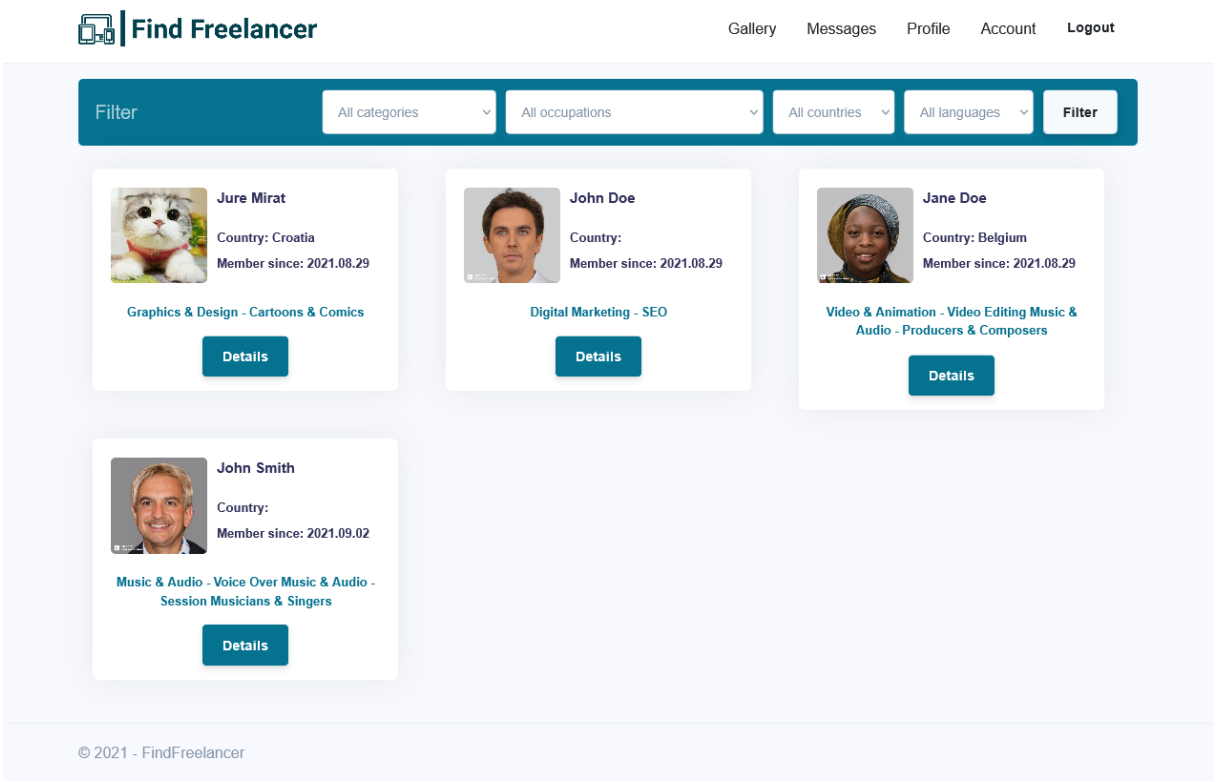


Slika 26. Stranica za prijavu [autorski rad]



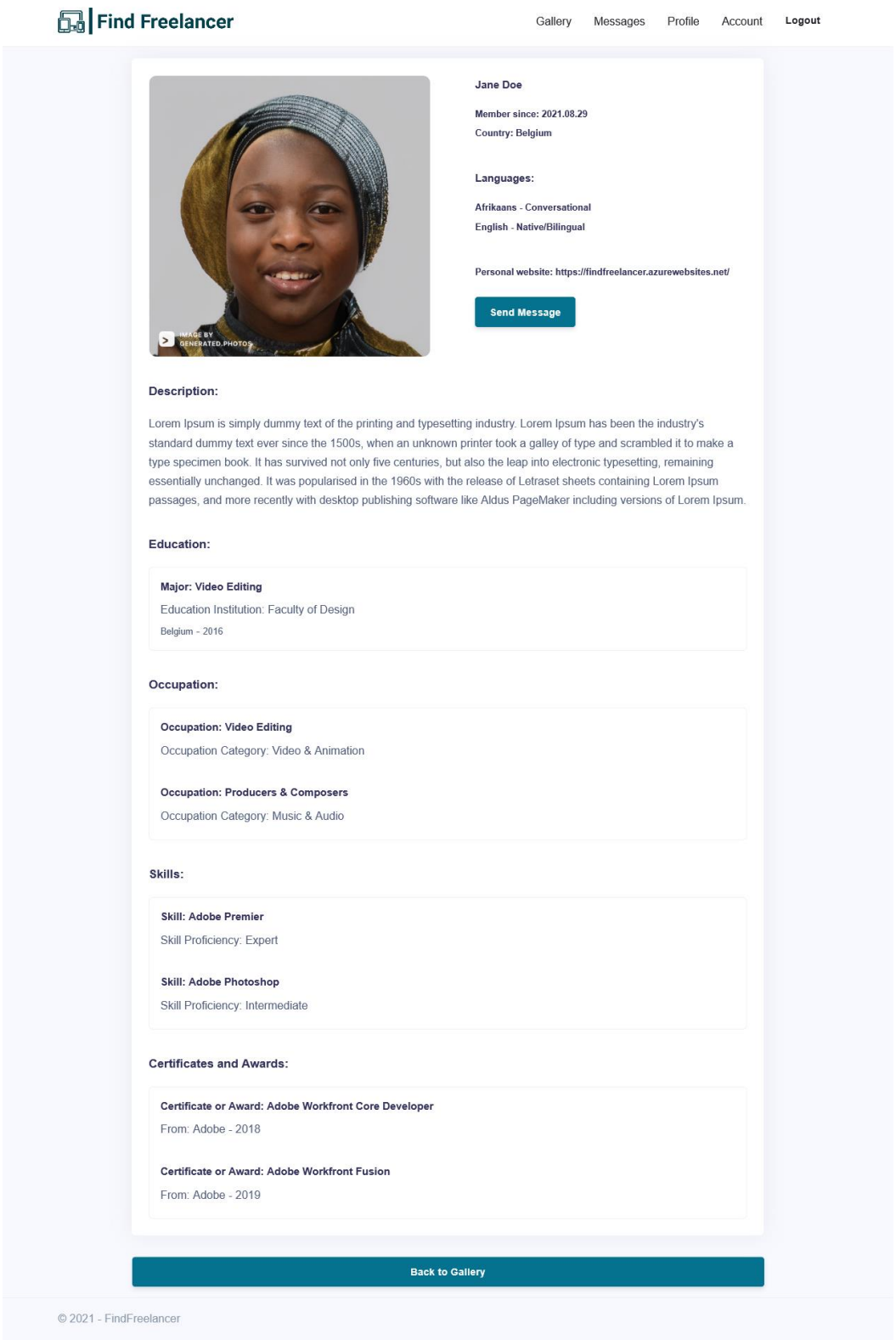
Slika 27. Stranica za registraciju [autorski rad]

Jedna od najključnijih funkcionalnosti cijele aplikacije je definitivno pregled galerije profila stručnjaka. Galerija ima osnovni filter koji pretežito služi za filtriranje kategorije zaposlenja ili samog naziva zaposlenja. Na karticama su dostupne osnovne informacije profila, naziv korisnika, profilna slika. Gumb za više detalja o pojedinom profilu je jasno označen.



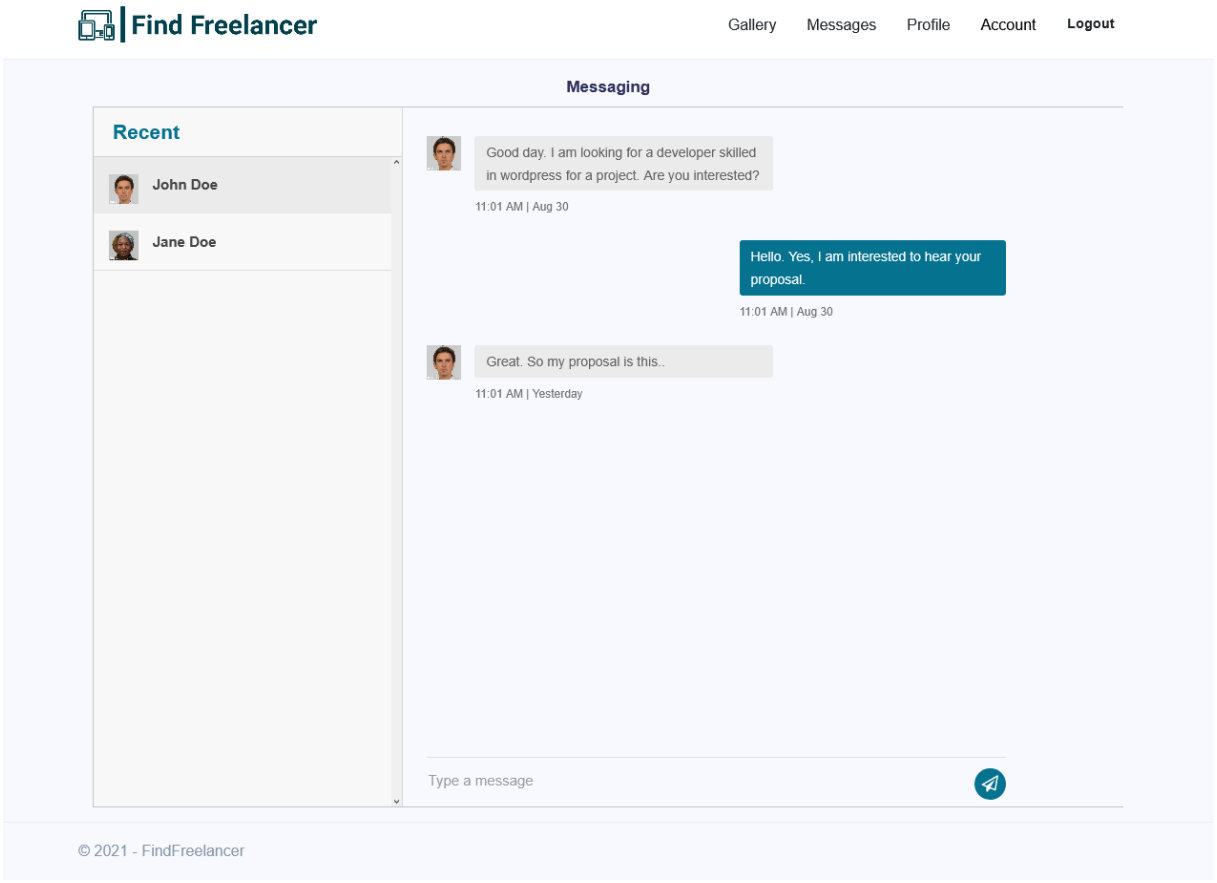
Slika 28. Stranica galerije [autorski rad]

Klikom na više detalja se otvara puni profil korisnika. Tu su dostupne sve informacije profila dostupne u bazi podataka. Klikom na gumb za slanje poruke se otvara sustav poruka s tim korisnikom.



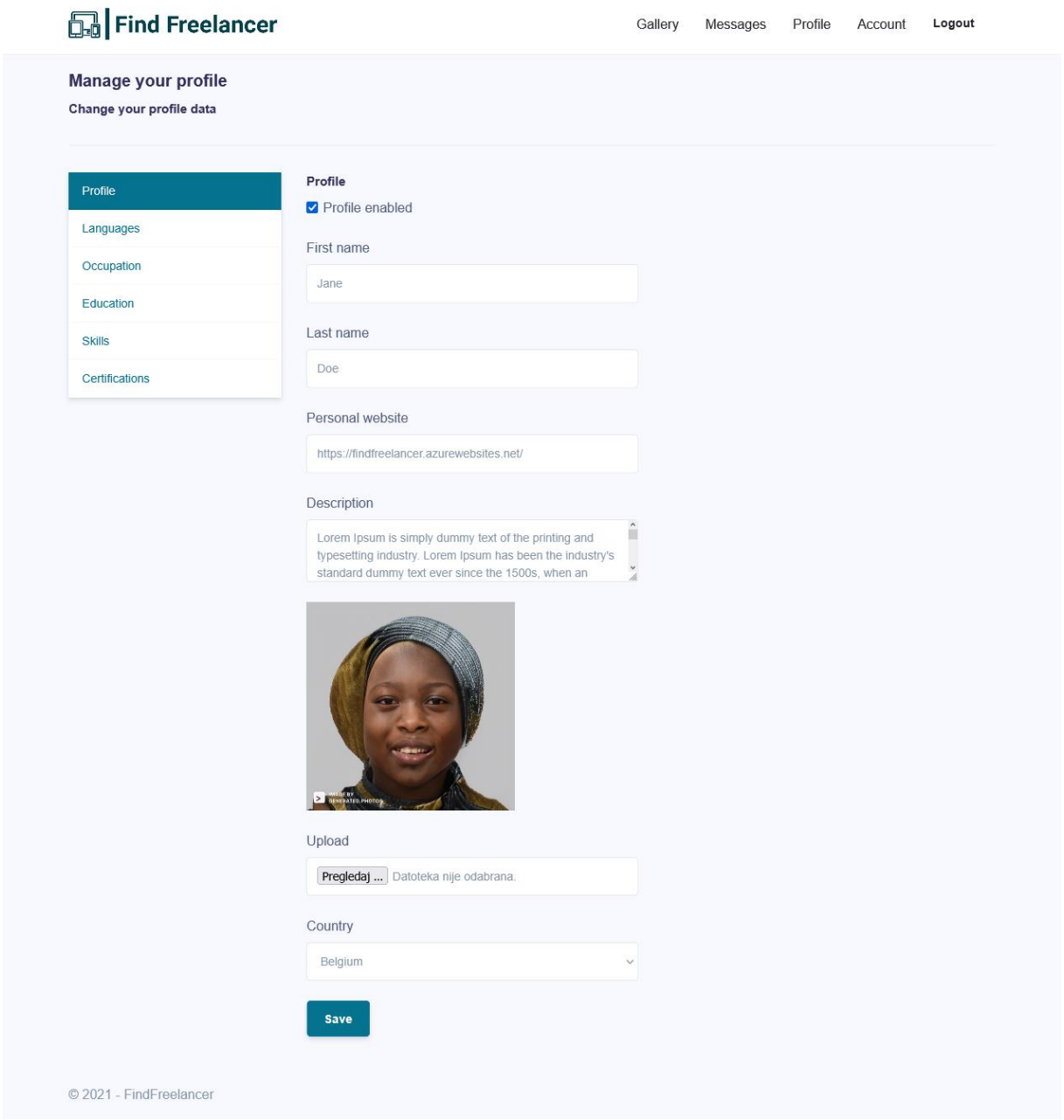
Slika 29. Stranica pregleda profila [autorski rad]

Sustav poruka je implementiran na tehnički jednostavan način. Poruke se dohvaćaju i spremaju u bazu. Sami prikaz poruka se u redovitim intervalima osvježava. Sustav nije chat u stvarnom vremenu. Namjena mu je samo za povezivanje s drugim korisnicima.



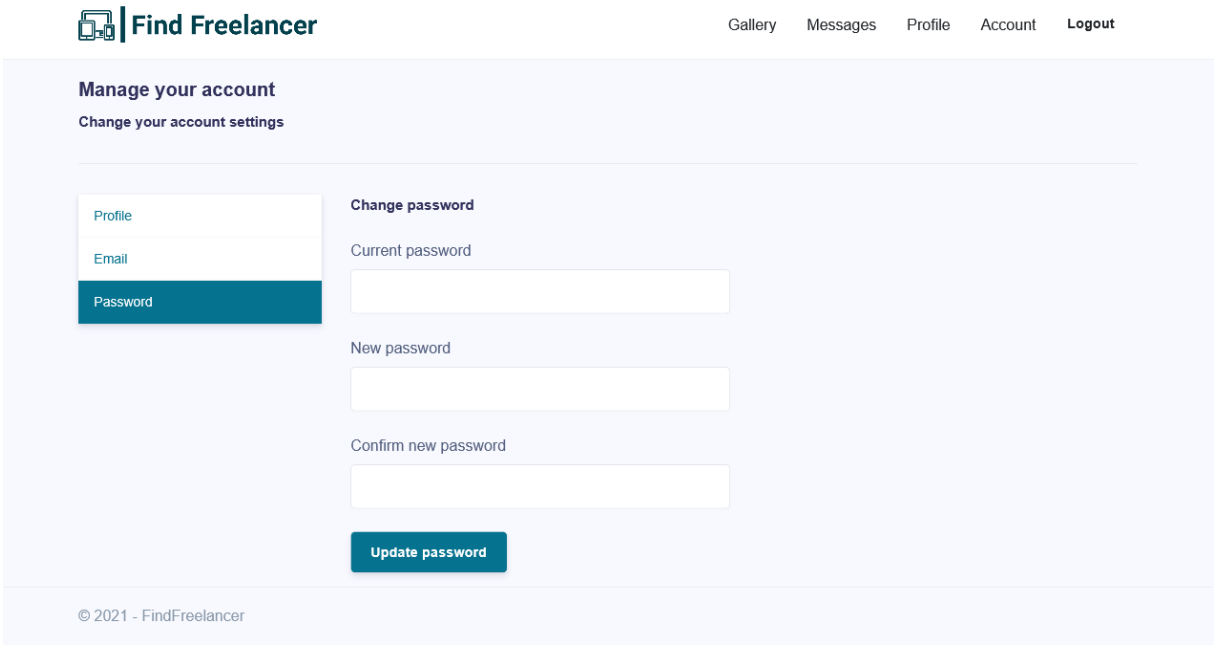
Slika 30. Stranica poruka [autorski rad]

Uređivanje profila se odvija preko seta stranica koje postanu dostupne na zasebnom meniju. Na primjeru je prikazan početni ekran za uređivanje profila. Sa strane su dostupne dodatne opcije za dodavanje detalja profilu.



Slika 31. Stranica za izmjenu profila [autorski rad]

Posljednja poveznica u navigaciji je za uređivanje računa korisnika. To su stranice koje su dio Identity paketa. Ponuđene su osnovne opcije promjene šifre, promjene emaila, pregled korisničkog imena.



Slika 32. Stranica za izmjenu korisničkog računa [autorski rad]

Time je pregled funkcionalnosti aplikacije završen. Prolazak kroz čitavu aplikaciju je ujedno i testiranje cijele aplikacije.

6.2. Analiza grafičkog sučelja

Osim testiranja funkcionalnosti web aplikacije, potrebno je provjeriti i korisničko iskustvo koje je usko vezano za kvalitetu grafičkog sučelja. Kvaliteta grafičkog sučelja se određuje metodom vrednovanja sučelja. Postoje dvije vrste vrednovanja grafičkog sučelja. Prvo je empirijsko vrednovanje, gdje sami korisnici testiraju. Drugo je heurističko vrednovanje koje prati set pravila. U nastavku će biti napravljeno vrlo jednostavno heurističko istraživanje praćenjem seta pravila koje opisuje Jakob Nielsen. To su vidljivost statusa u sustavu, podudaranje sustava i stvarnog svijeta, korisnička kontrola i sloboda, dosljednost i standardi, sprječavanje pogrešaka, prepoznavanje (a ne prisjećanje), fleksibilnost i učinkovitost korištenja, estetika i minimalistički dizajn, pomaganje korisnicima u dijagnosticiranju i oporavku od grešaka, pomoć i dokumentacija. [16]

Vidljivost statusa u sustavu se samo javlja u obliku greške kod validacije podataka forme. Ako neko polje nije popunjeno, sustav će javiti da je polje obavezno i ispisati da ga je potrebno popuniti. Nema javljanja statusa da su izmjene uspješno pohranjene. To bi bilo dobro imati na formi za promjenu osnovnih podataka profila.

Podudaranje sustava i stvarnog svijeta znači da se sustav treba koristiti pojmovima vezanima za domenu. Odnosno terminologijom i konvencijama iz stvarnog svijeta, kako bi bio poznat korisnicima. S obzirom na vrlo malu količinu teksta na stranicama i jednostavnim formama, sustav zadovoljava to pravilo.

Korisnička kontrola i sloboda se vidi u obliku mogućnosti prekida akcija ili brzog izlaska s određenih stranica sustava. Glavni meni je uvijek dostupan korisniku za brzi prijelaz. Sve stranice za uređivanje podataka, kao i pregled detalja galerije, nude mogućnost vraćanja na prethodni meni. Nema mogućnosti vraćanja obrisanih podataka (profila) ali postoji mogućnost brisanja i ponovnog dodavanja. Kod sustava poruka nema mogućnosti brisanja poruka nego samo njihovo slanje.

Dosljednost i standard znači da sustav prati neku konvenciju platforme. U ovom slučaju, nema akcija u aplikaciji koje bi mogle zbuniti korisnika tako da rade nešto skroz suprotno.

Sprječavanje pogrešaka se može vidjeti kod validacije podataka forma. Validacija sprječava da se šalju nepotpuni podaci i da dođe do greške.

Prepoznavanje, a ne prisjećanje je vidljivo na formama za uređivanje profila. Kod dodavanja određenih podataka su dostupni *dropdown* meniji s već dostupnim podacima. Na formama koje traže slobodni upis, a nisu intuitivne, je postavljen primjer unosa.

Fleksibilnost i učinkovitost korištenja nije vidljivo u sustavu. Nema kratica ili drugih načina da korisničko iskustvo bude učinkovitije.

Estetika i minimalistički dizajn se odnose na poštivanja načela kontrasta, poravnanja, blizine elemenata, ispravnih (a ne nevažnih) informacija. Stranice aplikacije koriste su složene s *Bootstrap* elementima. To stranicama daje vrlo čisti izgled i lijepi izgled kontrola. Boje aplikacije su u tamnijim nijansama plave boje, što lijepo izgleda, ali nije svugdje ujednačeno. *Bootstrap* omogućuje i responzivnost stranice za različite veličine ekrana. Kod responzivnosti na određenim stranicama postoje manje greške ali upotrebljivost je i dalje na visokoj razini.

Pomaganje korisnicima u dijagnosticiranju i oporavku od grešaka u obliku da se korisnika objasni gdje je greške postoji samo kod validacije podataka forme. Ako dođe do iznimke u radu web aplikacije, sami korisnik nema mogućnosti oporavka.

Pomoć i dokumentacija na stranici ne postoje. Aplikacija je vrlo jednostavna za korištenje i nema pretjerano veliki set mogućnosti.

Kvaliteta grafičkog sučelja je i više nego adekvatna kada se stavi u kontekst da se radi o inicijalnom prototipu aplikacije. Početna stranica zahtjeva bolji dizajn kako bi se korisnike privuklo. Ostale stranice aplikacije su vrlo jasne i funkcionalne.

6.3. Daljnji razvoj

Rezultat ovog rada je inicijalni, ali u potpunosti funkcionalni, prototip web aplikacije za pronalazak stručnjaka. Da bi aplikacija bila spremna za stvarne korisnike, potrebno je još dosta rada na korisničkom iskustvu. Odnosno malim stvarima koje korisnici očekuju u aplikaciji. Potrebno je doraditi upravljanje korisničkim računom. U današnjem dobu gdje je GDPR pravilo, a ne samo dobra praksa, potrebno je omogućiti korisnicima brisanje svojih računa i dohvat svih podataka koje sustav sprema o njima. Te mogućnosti su dostupne i u Identity paketu. Dalje bi bilo dobro proširiti mogućnosti sustava poruka. Da korisnici mogu upravljati svojim kontaktima i uređivati/brisati svoje poruke. Alternativa je prijeći na sustav poruka u realnom vremenu koristeći dostupnu tehnologiju kao *SignalR*. S poslovne strane, nema uopće informacija tko stoji iza aplikacije i kome se korisnici ili drugi zainteresirani mogu javiti. Također, bilo bi dobro omogućiti i sustav obavijesti preko mailova. Vrlo teško da će korisnici stalno provjeravati stranicu za nove poruke ili nove mogućnosti. To je jedan dodatni set mogućnosti koji bi se mogao razviti kao druga iteracija web aplikacije. Mogućnosti i ideja za smjer napredovanja aplikacija, naravno, ima napretek.

Drugi rezultat aplikacije je možda manje vidljiv ali i dalje vrlo bitan. A to je da je način na koji je ova aplikacija sastavljena vrlo fleksibilan i primjenjiv na sličnim projektima. Struktura se može doslovno kopirati u novi projekt i prenamijeniti za aplikaciju druge domene. Isto tako, arhitektura je zbog izoliranosti slojeva dovoljno fleksibilna da nije isključena ni opcija da se uz web aplikaciju razvija mobilna, pa čak i desktop verzija aplikacije. U tom slučaju bi trebalo složiti web API, ali većina potrebne strukture je već složena. Ovo je samo primjer kako praćenje dobre prakse razvoja uveliko ubrzava i olakšava daljnji razvoj projekta.

7. Zaključak

Cilj ovog diplomskog rada je bio prikazati cijeli proces razvoja web aplikacije od njene ideje pa do njene implementacije koristeći dobru praksu i aktualne metode. U teoretskom dijelu rada je objašnjen životni ciklus razvoja (web) aplikacije. Također, objašnjeni su modeli i metode razvoja. Bitno je da programer bude upoznat s postojećim znanjima programskog inženjerstva kako bi kvaliteta i uspješnost projekata rasla. Puno projekata propadne ili se jako zakompliciraju iz vrlo sličnih razloga. Razloga koji su se pravovremeno mogli izbjeći. Zato je uvijek potrebno imati generalni plan rada kako bi se rizik projekta minimizirao.

Izrada konkretnog plana je prikazana u ovom radu. Na primjeru aplikacije za pronalazak stručnjaka. Ideja aplikacije je zapisana i zatim analizirana. Pronađene su ključne funkcionalnosti buduće aplikacije iz njenog opisa. Također su proučene i postojeće aplikacije koje se bave istom domenom, pronalaska stručnjaka. Iz svega toga su proizašli konkretni zahtjevi za novu aplikaciju.

Sljedeći korak je bio dizajniranje same aplikacije. Razrada i dokumentiranje same aplikacije, prije nego što se počne sa samim programiranjem. Određena je tehnologija aplikacije, njena arhitektura, dizajnirana je baza podataka i napravljene su skice stranica aplikacije. Sve su to ključni koraci dizajniranja koje je puno lakše mijenjati i proučavati u fazi dizajniranja nego kasnije metodom pokušaja i pogreške u kodu.

Zatim je uslijedio razvoj same aplikacije. Web aplikacije je razvijena u najnovijoj .NET 5 razvojnoj okolini. Prikazano je kako se implementira čista arhitektura. Što je jako dobra praksa koju i sam Microsoft želi prenijeti na .NET developere. Razvoj web aplikacije je bio bez većih komplikacija. Za izradu stranica su praćene skice aplikacije, što je jako ubrzalo razvoj. Kod povezivanja na PostgreSQL bazu je dokumentirana i greška vezana za Entity Framework.

Gotova aplikacija je testirana. Sve funkcionalnosti ispravno rade. Napravljeno je i jednostavno vrednovanje grafičkog sučelja aplikacije. Navedene su buduće mogućnosti razvoja aplikacije. Sama web aplikacija uz još jednu ili dvije iteracije razvoja može biti spremna na korištenje vrlo kritičnoj masi korisnika.

Za kraj, prolazak kroz cijeli proces razvoja aplikacije je jedno od važnijih iskustava bilo kojeg mladog programera. Potrebno je raditi na vještinama projektiranja kako bi projekti bili sve uspješniji i kako bi se stekla određena širina gledanja na razvoj projekta. Ta vještina se po pričama i iskustvima programera često zna izgubiti u masi posla i velikih timovima gdje svaki pojedinac vidi i razvija samo jedan mali djelić aplikacije.

Popis literature

[1] Sungdeok Cha, Richard N. Taylor, and Kyochul Kang, Eds., *Handbook of Software Engineering*, 1. izd.: Springer, Cham, 2019.

[2] Goran Jevtic. (2019) What is SDLC? Phases of Software Development, Models, & Best Practices. Preuzeto sa: https://phoenixnap.com/blog/software-development-life-cycle

[3] Randall Hyde, *Write Great Code Volume 3: Engineering Software*, 1. izd.: no starch press, 2020.

[4] Samuel Davis, *Software Design and Development: the HSC course*, Drugo. izd., Janine Fendall, Ed.: Parramatta Education Center, 2012.

[5] Microsoft. (2021) A tour of the C# language. Preuzeto sa: https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/#net-architecture

[6] Microsoft. (2020) Introduction to.NET. Preuzeto sa: https://docs.microsoft.com/en-us/dotnet/core/introduction

[7] Mark J. Price, *C# 9 and.NET 5 - Modern Cross-Platform Development*, 5. izd.: Packt Publishing, 2020.

[8] The PostgreSQL Global Development Group. (2021) What is PostgreSQL? Preuzeto sa: https://www.postgresql.org/about/

[9] Microsoft. (2021) Welcome to the Visual Studio IDE. Preuzeto sa: https://docs.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2019

[10] Git Community. (2021) Git. Preuzeto sa: https://git-scm.com/

[11] pgAdmin Development Team. (2021) pgAdmin FAQ. Preuzeto sa: https://www.pgadmin.org/faq/#1

[12] Steve "ardalis" Smith, *Architecting Modern Web Applications with ASp.NET Core and Microsoft Azure*.: Microsoft Corporation, 2021.

[13] Anastasia D. (2019) Best Architecture for an MVP: Monolith, SOA, Microservices, or Serverless? Preuzeto sa: https://rubygarage.org/blog/monolith-soa-microservices-serverless

[14] Abhishek Arora. (2021) Introduction to ASP.Net Identity 2.0. Preuzeto sa: https://www.c-sharpcorner.com/UploadFile/16101a/introduction-to-Asp-Net-identity-2-0/

[15] EntityFrameWork Tutorial. (2020) What is Entity Framework? Preuzeto sa: https://www.entityframeworktutorial.net/what-is-entityframework.aspx

[16] Jakob Nielsen. Ten Usability Heuristics. [Online]. https://pdfs.semanticscholar.org/5f03/b251093aee730ab9772db2e1a8a7eb8522cb.pdf

[17] Microsoft. Visual Studio IDE. Preuzeto sa: https://docs.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2019

[18] IEEE, , Pierre Bourque and E. (Dick) Richard Fairley, Eds., 2014.

[19] San Murugesan and Athula Ginige, *Web Engineering: Introduction and Perspectives*. Australia.

Popis slika

Popis tablica

Prilog 1. Tablice u bazi podataka

Tablica. AspNetUsers

Stupac	Tip
Id	Text
UserName	Varchar
NormalizedUserName	Varchar
Email	Varchar
EmailConfirmed	Bool
PasswordHash	Text
SecurityStamp	Text
ConcurrencyStamp	Text
PhoneNumber	Text
PhoneNumberConfirmed	Bool
TwoFactorEnabled	Bool
LockoutEnf	Timestamptz
LockoutEnabled	Bool
AccessFailedCount	Int4

Tablica. AspNetUserTokens

Stupac	Tip
UserId	Text
LoginProvider	Varchar
Name	Varchar
Value	text

Tablica. AspNetUserRoles

Stupac	Tip
UserId	Text
RoleId	text

Tablica. AspNetRoles

Stupac	Tip
Id	Text
Name	Varchar
NormalizedName	Varchar
ConcurrencyStamp	text

Tablica. AspNetUserLogins

Stupac	Tip
LoginProvider	Varchar
ProviderKey	Varchar
ProviderDisplayName	Text
UserId	text

Tablica. AspNetUserClaims

Stupac	Tip
Id	Int4
UserId	Text
ClaimType	Text
ClaimValue	Text

Tablica. AspNetRoleClaims

Stupac	Tip
Id	Int4
RoleId	Text
ClaimType	Text
ClaimValue	text

Tablica. Certifications

Stupac	Tip
ID	Int4
Name	Text
From	Text
Year	Int4
ProfileId	Int4

Tablica. Countries

Stupac	Tip
Id	Int4
Name	Text

Tablica. EducationList

Stupac	Tip
Id	Int4
EducationInstitution	Text
CountryId	Int4
Major	Text
Year	Int4
ProfileId	Int4

Tablica. LanguageProficiencies

Stupac	Tip
Id	Int4
Name	Text

Tablica. LanguageProfiles

Stupac	Tip
Id	Int4
LanguageId	Int4
LanguageProficiencyId	Int4
ProfileId	Int4

Tablica. Languages

Stupac	Tip
Id	Int4
Name	Text

Tablica. Messages

Stupac	Tip
Id	Int4
SenderId	Text
RecipientId	Text
MessageText	Text
TimeSent	Timestamp

Tablica. NewContacts

Stupac	Tip
Id	Int4
SenderId	Text
RecipientId	Text

Tablica. OccupationCategories

Stupac	Tip
Id	Int4
Name	Text

Tablica. Occupations

Stupac	Tip
Id	Int4
Name	Text
OccupationCategoryId	Int4

Tablica. OccupationsList

Stupac	Tip
Id	Int4
OccupationId	Int4
ProfileId	Int4

Tablica. Profiles

Stupac	Id
Id	Int4
UserId	Text
ProfileEnabled	Bool
Name	Text
Surname	Text
CountryId	Int4
MemberSince	Timestamp
PersonalWebsite	Text
Description	Text
ProfileImage	text

Tablica. SkillLevels

Stupac	Tip
Id	Int4
Name	Text

Tablica. Skills

Stupac	Tip
Id	Int4
Name	Text
SkillLevelId	Int4
ProfileId	Int4