Modeliranje MongoDB baze podataka za potrebe web-aplikacije osobnog portfelja

Hodak, Tonino

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Tonino Hodak

MODELIRANJE MONGODB BAZE PODATAKA ZA POTREBE WEB-APLIKACIJE OSOBNOG PORTFELJA

ZAVRŠNI RAD

Varaždin, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Tonino Hodak

Matični broj: 0016142992

Studij: Informacijski sustavi

MODELIRANJE MONGODB BAZE PODATAKA ZA POTREBE WEB-APLIKACIJE OSOBNOG PORTFELJA

ZAVRŠNI RAD

Mentor :

Dr. sc. Bogdan Okreša Đurić

Varaždin, rujan 2023.

Tonino Hodak

Izjava o izvornosti

Izjavljujem da je moj završni 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 potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Kroz ovaj završni rad se opisuju načini korištenja jedne od mnogih baza podataka, a to je MongoDB. Detaljno se razmatraju karakteristike različitih tipova baza podataka, s naglaskom na nerelacijske baze podataka, a posebno na dokumentne baze podataka. Ključni dio rada je posvećen konkretnom modeliranju MongoDB baze podataka, obuhvaćajući sve faze razvoja - od konceptualnog do fizičkog modela. Nadalje, rad detaljno analizira primjenu programskih jezika HTML, CSS, JavaScript i C# u okviru Blazor okvira, s posebnim naglaskom na Blazor Server Side. Rad također opisuje i alate poput Visual Studio-ija i NuGet-a, neophodne za efikasan razvoj web-aplikacija. U praktičnom dijelu, rad prikazuje razvoj web-aplikacije kroz različite komponente, uključujući implementaciju funkcionalnosti kao što su navigacija, mogućnost kontaktiranja, prijava i registracija te uređivanje profila i objava. Uz to sve, važno je napomenuti da rad naglašava potencijal i prednosti MongoDB-a u razvoju modernih web-aplikacija.

Ključne riječi: HTML; CSS; C#; Blazor; Blazor Server Side; MongoDB; portfolio

Sadržaj

1. Uvod

Sektori informacijskih tehnologija su vrlo poznat pojam u današnjici te su često tema svakodnevnog razgovora. Razmatraju se svakojake mogućnosti digitalizacije i optimizacije već postojećih zamornih te repetitivnih poslova. Iz tog razloga se traže rješenja kod raznoraznih web-developera. Ljudi često dođu s nekom idejom što bi mogli digitalizirati, velikom ambicijom te pozamašnim novčanim sredstvima kako bi se ta ideja naposlijetku i ostvarila, ali sve to naravno uz neko određeno vrijeme isporuke web-aplikacije. U digitalizaciju te razvoj web-aplikacija se ulaže podosta novca jer se to pokazalo kao jedno od najpotrebitijih i najkorisnijih stvari danas. Pri samoj pomisli da se prije npr. moralo otići do knjižnice koja je mogla biti udaljena 20 km kako bi se posudila neka određena knjigu za čitanje ili učenje kod kuće, a knjižničar kaže da je ta knjiga trenutno nedostupna najmanje 2 tjedna... Danas se jednostavno ode na internet gdje se može pronaći tona web-aplikacija sa već digitaliziranim knjigama i koje se mogu čitati odmah, samo ih se treba preuzeti. A to je samo jedan primjer olakšice od njih milijun!

Naravno, na papiru se sve to čini vrlo lako, no kako to zapravo sve funkcionira? Naime, čak i da imate odličnu ideju i cijelu razradu aplikacije, kako će ona izgledati, kako će funkcionirati, što će raditi, sve Vam to propada u vodu ako nemate – bazu podataka. Postoje mnoge vrste baza podataka, neke manje kompleksne za rad, neke više, ali svima je svrha ista – spremanje podataka za kasniju manipulaciju istima. Naizgled se možda čini kao mala, usputna stvar, ali je zapravo cijeli potporanj Vaše web-aplikacije. Sve podatke o nekom proizvodu ili informacije o ljudima koje vidite u pravom vremenu ispred Vas na ekranu, sve je to negdje zapisano i spremljeno u neku bazu podataka.

10.8.2023. je objavljen zanimljiv podatak sa najpopularnijim (najkorištenijim) bazama podataka [1]. Spominju se klasična imena kao što su Oracle, MySQL, Microsoft SQL Server te PostegreSQL, a peto mjesto (vidljivo u tablici 1) je zauzeo MongoDB – baza podataka koja se koristi za razvoj web-aplikacije opisane u ovom radu.

Tablica 1: Najkorištenije baze podataka 2023. godine [1]

	Baza podataka	Rang
1	Oracle	1268.84
2	MySQL	1154.27
3	Microsoft SQL Server	1040.26
4	PostgreSQL	466.11
5	MongoDB	387.18
6	IBM Db2	179.85
7	Redis	149.01
8	Elasticsearch	143.44
9	Microsoft Access	141.62
10	SQLite	126.8
11	Cassandra	122.98
12	Splunk	81.43
13	MariaDB	78.82
14	Teradata	76.19
15	Hive	69.91
16	Solr	61.48
17	HBase	60.39
18	FileMaker	57.15
19	SAP HANA	56.64
20	Amazon DynamoDB	55.09
21	SAP Adaptive Server	55.04
22	Neo4j	46.8
23	Couchbase	34.59
24	Memcached	29.54
25	Microsoft Azure SQL Database	27.2

Svrha ovog završnog rada je prikazati proces modeliranja MongoDB baze podataka te principe i koristi korištenja iste u razvoju web-aplikacija te relativno laganu implementaciju kroz aplikaciju. Teorijski dio sadrži pregled baze podatala s naglaskom na domene nerelacijskih baza podataka, a posebice dokumentne baze podataka i MongoDB, te njihovu ulogu u izradi web aplikacija.

2. Modeliranje baze podataka

Modeliranje podataka je proces kojim se definiraju i analiziraju zahtjevi podataka koji su potrebni za podršku poslovnim procesima u okviru odgovarajućih informacijskih sustava u organizacijama [2]. Stoga proces modeliranja podataka uključuje stručnjake koji blisko surađuju s poslovnim dionicima, kao i potencijalnim korisnicima informacijskog sustava. Važno je napomenuti i da modeliranje podataka ne definira samo elemente podataka, već i njihove strukture te odnose među njima [3].

Postoje tri različite vrste modela podataka koje će se koristiti za informacijski sustav [2]:

- konceptualni model podataka - zahtjevi za podatcima se početno bilježe kao skup koji je u biti specifikacija o podatcima neovisnih o tehnologiji te se koristi kao osnova za raspravu o početnim zahtjevima
- logički model podataka – u njega se prevodi konceptualni model, koji dokumentira strukture podataka koje se mogu implementirati u baze podataka. Napomena - implementacija jednog konceptualnog modela podataka može zahtijevati više logičkih modela podataka
- fizički model podataka - posljednji korak koji uključuje transformacija logičkog modela podataka u fizički model podataka koji organizira podatke u tablice i vodi računa o pristupu, izvedbi i detaljima pohrane

2.1. Baza podataka

U računalstvu, baza podataka [4] je organizirana zbirka pohranjenih podataka, kojima se pristupa elektronički korištenjem sustava za upravljanje bazom podataka. Male baze podataka mogu se pohraniti u sustav datoteka, dok se velike baze podataka nalaze na računalnim klasterima [5] ili pohrani u oblaku (engl. *cloud* [1]). Dizajn baza podataka obuhvaća formalne tehnike i praktična razmatranja, uključujući modeliranje podataka, učinkovito predstavljanje i pohranjivanje podataka, upitne jezike, sigurnost te privatnost osjetljivih podataka, uključujući podršku istovremenom pristupu i toleranciji na pogreške.[7, str. 27-28, 65–66]

2.2. Sustav za upravljanje bazom podataka

Sustav za upravljanje bazom podataka (engl. *database management system*) je softver koji komunicira s krajnjim korisnicima, aplikacijama i samom bazom podataka kako bi uhvatio i analizirao podatke. Softver dodatno obuhvaća osnovne sadržaje koji se pružaju za administriranje baze podataka. Ukupni zbroj baze podataka, softvera i pripadajućih povezanih aplikacija se može nazvati sustavom baze podataka, oni moraju biti međusobno kompatibilni [4]. Često

[1] Cloud predstavlja dostupnost resursa računalnog sustava na zahtjev, posebice pohrane podataka (pohrana u oblaku), bez izravnog aktivnog upravljanja od strane korisnika [6]

se pojam "baza podataka" također slobodno koristi za označavanje bilo kojeg sustava za upravljanje bazom podataka, sustava baze podataka ili aplikacije povezane s bazom podataka. [7, str. v][2]

2.3. Relacijske baze podataka

Relacijska baza podataka je baza podataka temeljena na relacijskom modelu podataka, kako je predložio E. F. Codd 1970. godine [8].[9] Sustav koji se koristi za održavanje relacijskih baza podataka je sustav upravljanja relacijskim bazama podataka. Mnogi sustavi relacijskih baza podataka opremljeni su opcijom korištenja SQL-a (engl. *Structured Query Language*) za postavljanje upita i ažuriranje baze podataka [7, str. 6-7].

Cilj relacijske baze podataka je osigurati [10]:

- ažurnost pohranjenih podataka,
- izgradnju sigurnosti i nadzora pristupa podatcima,
- mogućnost dnevnog sigurnosnog arhiviranja,
- mogućnost kontrole i administriranja s jednog mjesta,
- mogućnost postavljanja više upita s više kriterija u svrhu izrade analiza i sinteza podataka,
- najmanju zalihost (redundanciju),
- postojanost pohranjenih podataka,
- točnost pohranjenih podataka,
- trajno očuvanje integriteta pohranjenih podataka.

Sustavi za upravljanje relacijskim bazama podataka:

1. Prema DB-Engines [11], **MySQL** [12] je druga najpopularnija baza podataka otvorenog koda na svijetu, iza Oracle Database [13].
   MySQL ima samostalne (engl. *stand-alone*) klijente koji korisnicima omogućuju izravnu interakciju s MySQL bazom podataka koristeći SQL, ali češće se MySQL koristi s drugim programima za implementaciju aplikacija kojima je potrebna sposobnost relacijske baze podataka.
   MySQL pokreće mnoge od najčešće pristupanih aplikacija [12], uključujući Facebook [14][15], Twitter [16], Flickr [17] i Youtube [18]. Budući da je MySQL otvorenog izvora (engl. *open source*), uključuje brojne značajke razvijene u bliskoj suradnji s korisnicima tijekom više od 25 godina.

2. **Microsoft SQL Server** [19] je sustav za upravljanje relacijskim bazama podataka koji je razvio Microsoft. Kao poslužitelj baze podataka, to je softverski proizvod s primarnom funkcijom pohranjivanja i dohvaćanja podataka prema zahtjevu drugih softverskih aplikacija - koje se mogu izvoditi ili na istom računalu ili na drugom računalu preko mreže (uključujući Internet). Microsoft prodaje najmanje desetak različitih izdanja Microsoft SQL Servera, namijenjenih različitoj publici i za radna opterećenja u rasponu od malih aplikacija s jednim strojem do velikih aplikacija okrenutih prema Internetu s mnogo istodobnih korisnika.

3. **PostgreSQL** [20], također poznat kao Postgres, besplatan je sustav za upravljanje relacijskim bazama podataka otvorenog koda s naglaskom na proširivost i usklađenost sa SQL-om. Izvorno je nazvan POSTGRES, upućujući na svoje podrijetlo kao nasljednika baze podataka Ingres razvijene na Kalifornijskom sveučilištu Berkeley [21]. 1996. godine je projekt preimenovan u PostgreSQL kako bi odražavao podršku za SQL. Nakon pregleda 2007. godine, razvojni tim odlučio je zadržati naziv PostgreSQL i pseudonim (engl. *alias*) Postgres. PostgreSQL sadrži transakcije sa svojstvima atomičnosti, dosljednosti, izolacije, trajnosti, automatski ažuriranim pogledima, materijaliziranim pogledima, okidačima, stranim ključevima i pohranjenim procedurama. Dizajniran je za rukovanje nizom radnih opterećenja, od pojedinačnih strojeva do skladišta podataka ili web usluga s mnogo istodobnih korisnika. Bila je to zadana baza podataka za macOS Server, a dostupna je za Linux, FreeBSD, OpenBSD te Windows.

4. **Oracle Database** [13] je višemodelni sustav za upravljanje bazom podataka koji proizvodi i prodaje Oracle Corporation [22]. To je baza podataka koja se obično koristi za radna opterećenja baze podataka tj. za online obradu transakcija te skladištenje podataka. Može se izvoditi na poslužiteljima trećih strana kao i na Oracle hardveru te koristi SQL upitni jezik za ažuriranje i dohvaćanje baze podataka.

5. **SQLite** [23] je biblioteka napisana na C-jeziku koja implementira mali, brzi, samostalni, visokopouzdani te potpuno opremljeni SQL pogon baze podataka. SQLite je najčešće korištena baza podataka na svijetu [24]. SQLite je ugrađen u sve mobilne telefone i većinu računala te dolazi u paketu s bezbrojnim drugim aplikacijama koje ljudi koriste svaki dan. Format datoteke SQLite je stabilan, višeplatformski te kompatibilan sa starijim verzijama. Datoteke baze podataka SQLite obično se koriste kao spremnici za prijenos velikog broja sadržaja između sustava i kao dugoročni arhivski format za podatke. Postoji više bilijuna SQLite baza podataka u aktivnoj upotrebi [24]. Važno je napomenti i da je izvorni kod SQLite-a u javnoj domeni te je dostupan svim korisnicima za bilo koju svrhu.

2.4. Nerelacijske baze podataka

Nerelacijske baze podataka (često nazivane NoSQL bazama podataka) razlikuju se od tradicionalnih relacijskih baza podataka po tome što svoje podatke pohranjuju u netabularnom obliku. Umjesto toga, nerelacijske baze podataka mogu se temeljiti na podatkovnim strukturama poput dokumenata. Dokument može biti vrlo detaljan, dokument sadržava niz različitih

vrsta informacija u različitim formatima. Ova sposobnost probavljanja i organiziranja različitih vrsta informacija jedne pored drugih čini nerelacijske baze podataka mnogo fleksibilnijim od relacijskih baza podataka. [25][26]

Nerelacijske baze podataka često se koriste kada je potrebno organizirati velike količine složenih i raznolikih podataka. Na primjer, velika trgovina može imati bazu podataka u kojoj svaki kupac ima vlastiti dokument koji sadrži sve njegove podatke, od imena i adrese do povijesti narudžbi i podataka o kreditnoj kartici. Unatoč različitim formatima, svaki od ovih podataka može se pohraniti u isti dokument. [26]

Nerelacijske baze podataka često rade brže jer upit ne mora pregledavati nekoliko tablica kako bi dao odgovor, kao što to često čine relacijski skupovi podataka. Nerelacijske baze podataka stoga su idealne za pohranu podataka koji se mogu često mijenjati ili za aplikacije koje obrađuju mnoge različite vrste podataka. Oni mogu podržati aplikacije koje se brzo razvijaju i zahtijevaju dinamičku bazu podataka koja se može brzo mijenjati i koja može prihvatiti velike količine složenih, nestrukturiranih podataka. [26]

2.4.1. Vremenske

Vremenske baze podataka [7, str. 198] dizajnirane su za povezivanje vrijednosti podataka, pojedinačnih n-torki ili cjelina tablice na vremensku os. Sama specifikacija vremena ima različita značenja za objekta u bazi podataka, jer valjano vrijeme možemo shvatiti ili kao trenutak kada se dogodi određeni događaj ili kao razdoblje ako su odgovarajuće vrijednosti podataka važeće kroz određeno vremensko razdoblje. Na primjer, adresa zaposlenika vrijedi dok god nije promijenjena.

Druga vrsta vremenske specifikacije je vrijeme transakcije, bilježenje trenutka kada se određeni objekt unosi, mijenja ili briše iz baze podataka. Sustav baze podataka obično sam upravlja različitim vremenima transakcija uz pomoć dnevnika, zbog čega će se vrijeme uvijek koristiti u smislu valjanog vremena.

Kako bi zabilježili važeća vremena kao točke u vremenu, većina sustava relacijskih baza podataka podržava nekoliko tipova podataka: DATUM se koristi za datume u obliku godine, mjeseca i dana, TIME za vrijeme dana u satima, minutama i sekundama, a TIMESTAMP je kombinacija datuma i vremena. Za davanje vremenskog razdoblja nema posebne vrste podataka koja mora biti izabrana; dovoljni su cijeli brojevi i decimale. To omogućuje mogućnost izračuna datuma i vremena.

2.4.2. Objektno-relacijske

Prvo proširenje tehnologije relacijske baze podataka [7, str. 209] je eksplicitna deklaracija strukturnih svojstava sustava baze podataka, na primjer, dodjeljivanjem surogata. Surogat je stalna, nepromjenjiva ključna vrijednost definirana od strane sustava, koja jedinstveno identificira svaku n-torku u bazi podataka. Surogati, kao nepromjenjive vrijednosti, mogu se koristiti da definiraju sustavno kontrolirane odnose čak i na različitim mjestima unutar relacije baza

podataka. Oni podržavaju referentni integritet kao i generalizaciju i agregaciju strukture.

Sustav baze podataka je strukturno objektno-relacijski ako podržava strukturirane tipove objekata. Osim identifikacije objekta, opisa strukture, i dostupnosti generičkih operatora (metode poput implicitnog spajanja itd.), potpuno objektno-relacijski sustav baze podataka trebao bi podržavati definiciju novih tipova objekata (klase) i metoda. Korisnici bi trebali moći odrediti metode potrebne za pojedinačni tip objekta. Također bi se trebali moći osloniti na podršku naslijeđenih svojstava tako da ne moraju definirati sve nove tipove objekata i metoda od nule, nego da se mogu oslanjati na već postojeće koncepte.

Objektno-relacijski sustavi baza podataka omogućuju tretiranje strukturiranih objekata kao jedinice i da se upotrijebi prilagodba generičkih operatora s njima. Formiranje klasa pomoću PART_OF i IS_A strukture dopuštene su i podržane metodama za spremanje, ispitivanje i manipuliranje.

2.4.3. XML

XML [7, str. 233] (eXtensible Markup Language) je razvio World Wide Web Konzorcij (W3C). Sadržaj hipertekstualnih dokumenata označen je oznakama, baš kao u HTML-u. XML dokument je samoopisujući, jer ne sadrži samo stvarne podatke nego i informacije o strukturi podataka.

Osnovni građevni blokovi XML dokumenata nazivaju se elementi. Sastoje se od početne oznake (u kutnim zagradama <name>) i završne oznake (u kutnim zagradama s kosom crtom </name>) sa sadržajem elementa između. Identifikatori za početak i završetak oznake moraju međusobno odgovarati, također, elementi u XML dokumentima mogu biti ugniježđeni proizvoljno.

Osnovni koncept XML shema je definiranje tipova podataka te podudaranje imena i tipova podataka pomoću deklaracija. To omogućuje stvaranje potpuno proizvoljnih XML dokumenata. Dodatno, moguće je opisati pravila integriteta za ispravnost XML dokumenata.

Postoji veliki broj standardnih tipova podataka, kao što su string, boolean, integer, date, time itd., ali osim njih, mogu se uvesti i korisnički definirani tipovi podataka. Ovo omogućuje npr. ograničenje vrijednosti pomoću gornje ili donje granice, ograničenja duljine ili popise dopuštenih vrijednosti.

2.4.4. Grafovske

Grafovske baze podataka [7, str. 50-51] imaju shemu strukturiranja tj. podatci se pohranjuju kao čvorovi i rubovi koji pripadaju tipu čvor ili tipu rub te sadrže podatke u obliku parova atribut-vrijednost. Za razliku od relacijskih baza podataka, njihova shema je implicitna, tj. podatkovni objekti koji trenutno pripadaju nepostojećem čvoru ili rubu se mogu umetnuti izravno u bazu podataka bez da im se prvo definira tip. Sustav za upravljanje bazom podataka implicitno prati promjene u shemi na temelju dostupnih informacija te time stvara odgovarajuće tipove.

Graf baza podataka je sustav za upravljanje bazom podataka sa sljedećim svojstvima:

- podatci i shema prikazani su kao grafikoni ili u obliku grafikona strukture, koje generaliziraju koncept grafova (npr. hipergrafovi),
- manipulacije podatcima se izražavaju kao transformacije grafova ili operacije koje mogu izravno adresirati tipična svojstva grafova (npr. susjedstvo, podgrafovi, veze itd.),
- baza podataka podržava provjeru ograničenja integriteta kako bi se osigurala dosljednost podataka. Definicija dosljednosti izravno je povezana sa strukturama grafova (npr. tipovi čvorova i rubova, referentni integritet rubova, ...),
- rubovi grafikona pohranjuju se u zasebnom skupu podataka, kao i čvorovi. Ovo čini graf učinkovitim za analize.

Grafovske baze podataka se koriste kada su podatci organizirani u mreže, jer u tim slučajevima nije bitan pojedinačni zapis, već međusobna povezanost svih zapisa. Prednost baze podataka grafova je svojstvo susjedstva bez indeksa: za svaki čvor, sustav baze podataka može pronaći izravnog susjeda, bez potrebe za razmatranjem svih rubova, kao što bi to bio slučaj u relacijskim bazama podataka koje koriste tablicu odnosa. Stoga je napor (engl. *query*) za ispitivanje odnosa s čvorom konstantan, neovisno o količini podataka. U relacijskim bazama podataka napor za utvrđivanje referenciranih n-torki se povećava s brojem n-torki, čak i ako se koriste indeksi. Baš kao i relacijske baze podataka, baze podataka s grafovima trebaju indekse kako bi osigurale brz i izravan pristup pojedinačnim čvorovima i rubovima putem svojih svojstava.

2.4.5. Dokumentne

Dokumentne baze podataka [7, str. 230] ne pohranjuju proizvoljne dokumente kao što su web, video ili zvučni podaci, ali pohranjuju strukturirane podatke u zapisima koji se nazivaju dokumenti.

Uobičajena spremišta dokumenata razvijena su posebno za korištenje na web-u. Stoga se mogu lako integrirati s web tehnologijama kao što su JavaScript i HTTP. Dodatno, oni su lako skalabilni koristeći kombinaciju više računala u integriranom sustavu koji raspodjeljuje količinu podataka dijeljenjem. Fokus je uglavnom na obradi velikih količina heterogenih podataka, dok za većinu web podataka stalna konzistentnost podataka ne mora biti osigurana. Iznimka su sigurnosno osjetljive web usluge poput internetskog bankarstva, koji se uvelike oslanjaju na ograničenja sheme i zajamčenu dosljednost.

Spremišta dokumenata potpuno su bez shema, tj. nema potrebe za definiranjem sheme prije umetanja podatkovnih struktura. Shematska odgovornost je dakle prenesena korisniku ili aplikaciji za obradu. Nedostatak fiksne sheme je nepostojanje referentnog integriteta i normalizacije. Međutim, nepostojanje ograničenja sheme omogućuje fleksibilnost u pohranjivanju širokog raspona podataka, što također olakšava fragmentaciju i raspodjelu podataka.

Na prvoj razini, pohrane dokumenata su vrsta pohrane ključ-vrijednost. Za svaki ključ (ID dokumenta), zapis se može pohraniti kao vrijednost. Ti se zapisi nazivaju dokumentima. Na drugoj razini ti dokumenti imaju svoju unutarnju strukturu. Termin dokument nije posve prikladan, budući da oni nisu izričito multimedijski ili nestrukturirani podatci druge vrste. Dokument u kontekstu spremišta dokumenata je datoteka sa strukturiranim podatcima, npr. u JSON (vidljivo u 2.1) formatu. Struktura je popis parova vrijednosti atributa. Sve vrijednosti atributa u ovoj strukturi podataka mogu rekurzivno sadržavati popise sami parova atribut-vrijednost. Dokumenti nisu međusobno povezani, ali sadrže zatvorenu zbirku podataka.

Isječak koda 2.1: JSON dokument

```
{
    "FirstName": "Mile",
    "Address": "Zagrebacka␣33",
    "Hobby": "jedrenje"
}
```

Upiti u spremištu dokumenata mogu se paralelizirati te ubrzati s postupkom MapReduce. Takvi procesi su dvofazni, gdje Map odgovara grupiranju grupiraj po (engl. *group by*), a Reduce odgovara agregaciji npr. zbroj (engl. *count*) u SQL-u.

Tijekom prve faze, funkcija Map, koja provodi unaprijed definirani proces za svaki dokument se izvršava, izgrađuje te vraća kartu. Takva karta je asocijativni niz s jednim ili više parova ključ-vrijednost po svakom dokumentu. Faza karte se može izračunati po dokumentu neovisno o ostalim podatcima sadržaja, čime se uvijek omogućuje paralelna obrada bez ovisnosti ako je baza podataka raspoređena između različitih računala.

U neobaveznoj fazi smanjenja, funkcija se izvršava za smanjenje podataka, vraćajući jedan redak po ključu u indeksu iz funkcije karte te agregira odgovarajuće vrijednosti.

Zahtjev za cjelovitost dokumenta čini strani ključ (engl. *foreign key*) nepotrebnim. To čini učinkovitu distribuciju dokumenata u klasteru računala jer nema latencije mreže kao kod razlučivanja stranog ključa. Ovo horizontalno skaliranje kombinira različita računala u jedan cjelokupni sustav. Iz tog razloga, velike količine podataka se mogu raspodijeliti na više računala. Taj se mehanizam naziva usitnjavanje (engl. *sharding*). Stoga je u slučaju dokumentnih baza podataka fokus na obradi velikih količina heterogenih podataka [7, str. 55].

2.4.6. Sustavi za upravljanje nerelacijskim bazama podataka

1. **MongoDB** [27] je dokumentna baza podataka (NoSQL) koja se odlikuje brzinom i efikasnošću pri upravljanju podatcima. Koristi se u web-aplikacijama kojima su podatci bitni u realnom vremenu te u web-aplikacijama s ogromnim količinama podataka. Podatci u MongoDB se spremaju u takozvane “kolekcije” koje su slične tablicama u relacijskim bazama podataka, a te kolekcije sadrže podatke u obliku dokumenata. Svakoj kolekciji administrator (osoba zadužena za rad sa bazom podataka) unaprijed definira neka svojstva koja dokumenti u kolekciji mogu imati (a ne moraju nužno), a po potrebi se naravno svojstva lako mogu dodavati ili oduzimati. Uz takva unaprijed definirana svojstva, doku-

menti (kad su napravljeni) imaju izgled sličan JSON-u, što takav pregled podataka čini vrlo intuitivnim i prilagođenim korisniku (engl. *user-friendly*) te olakšava pronalazak potencijalnih grešaka ili drugih bitnih podataka čime se potiče brži razvoj. Upravo ta odlika je najviše opravdala visoko, 5. mjesto (vidljivo u tablici 1) MongoDB-a na listi [1] kao izbor za najkorišteniju bazu podataka.

2. Apache Cassandra je besplatan NoSQL sustav za upravljanje bazom podataka, otvorenog je koda te nudi pohranu širokih stupaca dizajniranih za rukovanje velikim količinama podataka na mnogim poslužiteljima, pružajući visoku dostupnost bez ijednog mjesta kvara. Također nudi podršku za klastere koji obuhvaćaju više podatkovnih centara, s asinkronom replikacijom koja omogućuje operacije niske latencije za sve klijente. Ova baza podataka je dizajnirana za implementaciju kombinacije Amazonove Dynamo tehnike distribuirane pohrane i replikacije u kombinaciji s Googleovim Bigtable [28] modelom podataka i pohrane podataka. [29]

3. Redis služi kao pohrana u memoriji, otvorenog je koda, a koristi se kao distribuirana baza podataka ključeva i vrijednosti u memoriji te kao predmemorija i broker [2] poruka. Budući da drži sve podatke u memoriji i zbog svog dizajna, Redis nudi čitanje i pisanje s malom latencijom, što ga čini posebno prikladnim za slučajeve upotrebe koji zahtijevaju predmemoriju. [31]

   Redis također podržava različite vrste apstraktnih struktura podataka kao što su nizovi, popisi, karte, skupovi, sortirani skupovi, HyperLogLogovi, bitmape, tokovi te prostorni indeksi [31].

4. Apache CouchDB [32] je NoSQL baza podataka otvorenog koda, orijentirana na dokumente, a implementirana u Erlangu [33]. CouchDB koristi više formata i protokola za pohranu, prijenos i obradu svojih podataka: koristi JSON za pohranu podataka, JavaScript kao jezik upita koristeći MapReduce [34] i HTTP za API.

   CouchDB također implementira oblik kontrole multiverzionirane kontrole (engl. *multiversion concurrency control* tako da ne zaključava datoteku baze podataka tijekom pisanja odnosno konflikti su prepušteni aplikaciji da ih riješi. Rješavanje konflikata općenito prvo uključuje spajanje podataka u jedan od dokumenata, a zatim brisanje zastarjelog. [32]

   Ostale značajke uključuju ACID [35] semantiku (engl. *atomicity, consistency, isolation, durability*, prevedeno - atomičnost, konzistencija, izolacija, trajnost) na razini dokumenta s konačnom dosljednošću, MapReduce, (inkrementalnu) replikaciju te tzv. “multi-master” replikaciju, koja omogućuje skaliranje između strojeva za izgradnju sustava visokih performansi [36].

5. Amazon DynamoDB [37] je potpuno upravljana NoSQL baza podataka koju nudi Amazon.com kao dio portfelja Amazon Web Services [38]. DynamoDB nudi brzu, postojanu ključ-vrijednost pohranu podataka s ugrađenom podrškom za replikaciju, automatsko skaliranje, šifriranje te sigurnosno kopiranje na zahtjev među ostalim značajkama.

[2] Broker poruka je posrednički računalni programski modul koji prevodi poruku iz formalnog protokola za slanje poruka pošiljatelja u formalni protokol za slanje poruka primatelja [30]

U DynamoDB-u podatci su pohranjeni u tablicama kao stavke i mogu se postavljati upiti pomoću indeksa. Stavke se sastoje od niza atributa koji mogu pripadati brojnim vrstama podataka i moraju imati ključ za koji se očekuje da bude jedinstven u tablici.

2.5. Modeliranje MongoDB baze podataka

Modeli podataka [7, str. 25] daju strukturiran i formalan opis podataka te odnosa podataka za informacijski sustav. Na temelju toga, model baze podataka ili shema definira odgovarajuće strukturiranje baze podataka. Kada su podatci potrebni za informacijske projekte, kao što su npr. podatci o zaposlenicima, odjelima i projektima, mogu se definirati potrebne kategorije podataka te njihovi odnosi. Definicija tih kategorija podataka (zvanih skupovi entiteta) i određivanje skupova odnosa se u ovom trenutku vrši bez razmatranja vrste sustava za upravljanje bazom podataka (SQL ili NoSQL) koji će se koristiti za unos, pohranjivanje te kasnije za održavanje podataka. Time se osigurava da podatci te odnosi podataka ostaju stabilni iz perspektive korisnika tijekom razvoja te širenja informacijskih sustava.

Potrebna su tri koraka za postavljanje strukture baze podataka [7, str. 25]:

- analiza zahtjeva
- konceptualno modeliranje podataka
- implementacija shema baze podataka preslikavanjem modela odnosa entiteta u SQL ili NoSQL baze podataka

Cilj analize [7, str. 25] je pronaći (u suradnji s korisnikom) podatke potrebne za informacijski sustav i njihove međusobne odnose uključujući strukturu količine. Što je bitno za rano određivanje granice sustava. Katalog zahtjeva priprema se u iterativnom procesu, na temelju intervjua, analiza potražnje, upitnika, kompilacija obrazaca itd. Katalog zahtjeva uvijek sadrži barem verbalni opis zadatka s jasno formuliranim ciljevima te popisom relevantnih informacija. Pisani opis podatkovnih veza može se nadopuniti grafičkim ilustracijama ili sažimanjem primjera. Obvezno je da analiza zahtjeva stavi činjenice potrebne za kasniji razvoj baze podataka na jeziku korisnika.

U procesu razvoja suvremene web-aplikacije, ključna faza predstavlja pravilno modeliranje baze podataka koja će omogućiti sve funkcionalnosti i osigurati dinamično korisničko iskustvo. S obzirom na dinamičnu prirodu današnjih aplikacija, postoji sve izraženija potreba za fleksibilnim i skalabilnim bazama podataka. U tom kontekstu, MongoDB se ističe kao jedan od vodećih u području NoSQL baza podataka, pružajući razvojnim timovima mogućnost brze i efikasne izrade, pohrane i upravljanja kompleksnim setovima podataka.

U ovom dijelu rada, prikazan je proces modeliranja baze podataka za web-aplikaciju koristeći MongoDB. Aplikacija, koja je razvijena u Blazor okviru, integrirajući tehnologije kao što su HTML, CSS, JavaScript, i C#, zahtijeva strukturirani i dobro dizajnirani model baze podataka koji može pravilno podržati sve segmente aplikacije.

Prvo je opisana aplikacijska domena, gdje su identificirani glavni entiteti i relacije koje čine srž (engl. *core*) aplikacije. Zatim se prelazi na konceptualno modeliranje, gdje je detaljnije analiziran svaki identificirani entitet te interakcije entiteta. Nakon toga, izrađen je logički model baze podataka, definirajući najvažnije segmente konceptualnog modela te dodatno ih opisujući. Na kraju, predstavljen je fizički model baze podataka, detaljno opisujući strukturu i organizaciju podataka unutar MongoDB-a.

Kroz ovaj proces, predstavljene su koristi i mogućnosti MongoDB-a za kreiranje fleksibilne i efikasne baze podataka koja idealno služi potrebama ove aplikacije.

2.5.1. Opis aplikacijske domene

Ova web-aplikacija služi kao platforma gdje korisnici mogu izraditi vlastiti portfelj sa unaprijed predodređenim predloškom profila. Korisnici imaju olakšicu prilikom izrade reprezentativnog portfelja jer ga ne moraju sami iz nule programirati. Mogu ga uređivati pomoću određenog seta atributa kao što su npr. zanimanje, zaposlenje, iskustvo, poslovni e-mail, ...) te promovirati sebe kroz objave na stranici “Novosti i objave”.

Cilj ove aplikacije je olakšati korisnicima promociju svojih vještina i postignuća, stvarajući vid zajednice gdje se korisnici mogu povezivati jedni s drugima, razmjenjivati informacije te potencijalno međusobno surađivati, a poseban naglasak je na međusobnoj komunikaciji.

2.5.2. Konceptualni model

Primarni entiteti u domeni aplikacije su:

- korisnici: individue koje se mogu registrirati i prijaviti na platformu, uređivati svoj profil, postaviti slike na profil te objavljivati objave (engl. *post*)
- objave: objave koje korisnici stvaraju na svojim profilima i koje su vidljive na stranici “Novosti i objave”
- slike: multimedijski elementi koje korisnici mogu dodati na svojem profilu
- profesije: različite vrste zanimanja koje korisnici mogu odabrati prilikom uređivanja svojih profila
- uloge: različite razine pristupa ili funkcionalnosti koje korisnici mogu imati na platformi (npr., neregistrirani korisnik, registrirani korisnik, administrator, ...).
- korisničke sesije: informacije koje prate individualne sesije korisnika na platformi

Povezanost entiteta:

- Korisnici mogu poslati kontakt e-mail
- Korisnici mogu kreirati korisnički račun

- Korisnici se mogu prijaviti u svoj korisnički račun
- Korisnici mogu zatražiti promjenu zaboravljene lozinke
- Korisnici mogu promijeniti zaboravljenu lozinku
- Korisnici mogu kreirati objave te ih uređivati i/ili brisati
- Korisnici mogu učitati i/ili obrisati sliku profila
- Korisnici mogu uređivati postavke profila te odabrati svoju profesiju
- Korisnici mogu imati određene uloge u aplikaciji
- Korisnici mogu pregledavati objave drugih korisnika
- Korisnici mogu otići na profile drugih korisnika i vidjeti njihove informacije i objave

2.5.3. Logički model

Sljedeći logički model (vidljivo u tablici 2) detaljno opisuje strukturu kolekcije "Users" u MongoDB bazi podataka, sa svim atributima i njihovim tipovima podataka te kratkim opisima.

Tablica 2: Kolekcija Users

Tip Naziv atributa	Tip podatka	Opis
_id	string	jedinstveni identifikator korisnika u bazi podataka
Username	string	korisničko ime koje korisnik koristi za prijavu u sustav
FirstName	string	ime korisnika
LastName	string	prezime korisnika
Email	string	e-mail adresa korisnika
Password	string	lozinka korisnika
Salt	string	nasumično generirani niz znakova koji se spaja s lozinkom prije procesa heširanja kako bi se povećala sigurnost lozinke
BusinessEmail	string	poslovna e-mail adresa korisnika
City	string	grad u kojem korisnik živi
Country	string	država u kojoj korisnik živi
JobStatus	int	trenutni status zaposlenja korisnika (npr. 1 za zaposlen, 0 za nezaposlen), korišten je tip podatka int jer je u aplikaciji definiran enum na temelju kojeg se sprema i čita vrijednost ovog atributa
Description	string	opisni tekst o korisniku, uključujući trenutne aktivnosti ili zanimanja
RememberMe	bool	indikator koji pokazuje želi li korisnik da sustav pamti njegove podatke za prijavu (u ovom slučaju samo korisničko ime)
LoggedIn	bool	indikator koji pokazuje je li korisnik trenutno prijavljen u sustav
Blocked	bool	indikator koji pokazuje je li korisnički račun blokiran
RegistrationDate	datetime	datum i vrijeme kada se korisnik registrirao na web-aplikaciji
SessionStart	datetime	datum i vrijeme početka trenutne sesije korisnika
SessionEnd	datetime	datum i (predviđeno) vrijeme završetka trenutne sesije korisnika (predviđeno - ako se korisnik odluči odjaviti, završetak trenutne sesije se postavlja na DateTime.Now odnosno vrijeme u tom trenutku)
UserSession_Id	string	jedinstveni identifikator trenutne sesije korisnika
Role_Id	string	jedinstveni identifikator uloge koju korisnik ima
Profession_Id	string	jedinstveni identifikator profesije koju korisnik ima
Company_Id	string	jedinstveni identifikator tvrtke u kojoj je korisnik zaposlen
Image_Id	string	jedinstveni identifikator slike profila korisnika

Sljedeći logički model (vidljivo u tablici 3) detaljno opisuje strukturu kolekcije "UserSessions" u MongoDB bazi podataka, sa svim atributima i njihovim tipovima podataka te kratkim opisima.

Tablica 3: Kolekcija UserSessions

Tip Naziv atributa	Tip podatka	Opis
_id	string	jedinstveni identifikator sesije korisnika u bazi podataka
Username	string	korisničko ime koje korisnik koristi tijekom sesije
Password	string	lozinka koju korisnik koristi tijekom sesije
RememberMe	bool	indikator koji pokazuje želi li korisnik da sustav pamti njegove podatke za prijavu tijekom sesije (u ovom slučaju samo korisničko ime)

Sljedeći logički model (vidljivo u tablici 4) detaljno opisuje strukturu kolekcije "Roles" u MongoDB bazi podataka, sa svim atributima i njihovim tipovima podataka te kratkim opisima.

Tablica 4: Kolekcija Roles

Tip Naziv atributa	Tip podatka	Opis
_id	string	jedinstveni identifikator uloge korisnika u bazi podataka
Name	string	naziv uloge (npr. neregistrirani korisnik, registrirani korisnik, administrator, ...)

Sljedeći logički model (vidljivo u tablici 5) detaljno opisuje strukturu kolekcije "Professions" u MongoDB bazi podataka, sa svim atributima i njihovim tipovima podataka te kratkim opisima.

Tablica 5: Kolekcija Professions

Tip Naziv atributa	Tip podatka	Opis
_id	string	jedinstveni identifikator zanimanja korisnika u bazi podataka
Name	string	naziv zanimanja (npr. front-end developer, back-end developer, ...)

Sljedeći logički model (vidljivo u tablici 6) detaljno opisuje strukturu kolekcije "Posts" u MongoDB bazi podataka, sa svim atributima i njihovim tipovima podataka te kratkim opisima.

Tablica 6: Kolekcija Posts

Tip Naziv atributa	Tip podatka	Opis
_id	string	jedinstveni identifikator objave u bazi podataka
Posted	datetime	datum i vrijeme kada je objava prvi put objavljen
Edited	datetime	datum i vrijeme kada je objava zadnji put ažurirana
Author_Id	string	jedinstveni identifikator autora objave

Sljedeći logički model (vidljivo u tablici 7) detaljno opisuje strukturu kolekcije "Images" u MongoDB bazi podataka, sa svim atributima i njihovim tipovima podataka te kratkim

opisima.

Tablica 7: Kolekcija Images

Tip Naziv atributa	Tip podatka	Opis
_id	string	jedinstveni identifikator slike u bazi podataka
Name	string	naziv slike
Data	byte[]	podatci slike u nizu znakova, svaki znak je byte (sadrži vrijednost od 0 do 255)

Sljedeći logički model (vidljivo u tablici 8) detaljno opisuje strukturu kolekcije "Companies" u MongoDB bazi podataka, sa svim atributima i njihovim tipovima podataka te kratkim opisima.

Tablica 8: Kolekcija Companies

Tip Naziv atributa	Tip podatka	Opis
_id	string	jedinstveni identifikator tvrtke u bazi podataka
Name	string	naziv tvrtke

2.5.4. Fizički model

Na sljedećim kolekcijama se može vidjeti način na koji se vršila implementacija atributa te njihova međusobna povezanost. Ovakva struktura kolekcija omogućuje brz i efikasan pristup podatcima koji se nalaze u svakom od dokumenata, što dodatno osigurava stabilnost te visoku učinkovitost web-aplikacije.

Kolekcija Users 2.2:

Isječak koda 2.2: Primjer dokumenta korisnika

```
_id: ObjectId('65f8b6baa65ab451dc9b322a')
Username: "ACMERocks"
FirstName: "Ela"
LastName: "Monsiri"
Email: "elamons@gmail.com"
Password: "$2y$10$CjHD7NNhMFCuNrCsSiYwW.4otpR.2umCpZiKhoRlllA4o4PAmMPGO"
Salt: "k1U1haaZs/D623aW9cwDeS=="
BusinessEmail: "acmebusiness@gmail.com"
City: "Madrid"
Country: "Spain"
JobStatus: 1
EmployedAt: "ACME"
Description: "10 years of experience in developing ACME"
RememberMe: true
LoggedIn: true
Blocked: false
RegistrationDate: 2023-09-06T17:28:27.772+00:00
```

```
SessionStart: 2023-09-11T23:36:29.074+00:00
SessionEnd: 2023-10-11T23:36:29.074+00:00
UserSession_Id: "14faa6712f7f81f22f7a3182"
Role_Id: "65f8b6baa65bb451dc9b322a"
Profession_Id: "65aab6b1b6574521c963877a"
Company_Id: "612abvab1b66521c9jf3877b"
Image_id: "3316326cbb4baababeef4620"
```

Kolekcija UserSessions 2.3:

Isječak koda 2.3: Primjer dokumenta sesije korisnika

```
_id: ObjectId('65f8b6baa65bb451dc9b322a')
Username: "ACMERocks"
Password: "$2y$10$CjHD7NNhMFCuNrCsSiYwW.4otpR.2umCpZiKhoRlllA4o4PAmMPGO"
RememberMe: true
```

Kolekcija Roles 2.4:

Isječak koda 2.4: Primjer dokumenta uloge

```
_id: ObjectId('65f8b6baa65bb451dc9b322a')
Name: "registeredUser"
```

Kolekcija Professions 2.5:

Isječak koda 2.5: Primjer dokumenta zanimanja

```
_id: ObjectId('65aab6b1b6574521c963877a')
Name: "Engineer"
```

Kolekcija Posts 2.6:

Isječak koda 2.6: Primjer dokumenta objave

```
_id: ObjectId('65f8b6baa65bb451dc9b322a')
Text: "Now␣hiring␣new␣engineers␣for␣our␣ACME␣corporation!"
Posted: 2023-09-10T03:35:05.104+00:00
Edited: 2023-09-10T07:46:38.549+00:00
Author_Id: "65f8b6baa65ab451dc9b322a"
```

Kolekcija Images 2.7:

Isječak koda 2.7: Primjer dokumenta slike

```
_id: ObjectId('3316326cbb4baababeef4620')
Name: "acmeprofcorporation"
Data: BinData(0, '/9j/4AAQSkZJABAQEABgAAD/wAARCAGOAAAhEBAxEB/8
    ...QBogAAAQUBAAAAECAwQFBgcICQoLEAAC')
```

Kolekcija Companies 2.8:

Isječak koda 2.8: Primjer dokumenta tvrtke

```
_id: ObjectId('3316326cbb4baababeef4620')
Name: "acmeprofcorporation"
Data: BinData(0, '/9j/4AAQSkZJABAQEABgAAD/wAARCAGOAAAhEBAxEB/8
    ...QBogAAAQUBAAAAECAwQFBgcICQoLEAAC')
```

3. Metode i tehnike rada

U ovom poglavlju se detaljno razmatraju metode i tehnike koje su primijenjene tijekom razvoja projekta. Naglasak je na planiranju, tehnologijama i alatima te resursima koji su bili ključni u ostvarivanju ove aplikacije te pomoću kojih se nastojalo zadovoljiti cilj izrade aplikacije.

3.1. Programski jezici

Tijekom opisivanja web-aplikacije, fokusirat će se podosta također na programerske jezike koji su bili korišteni u razvoju projekta kroz različite primjere programskih kodova koji će biti prikazani. Dodatno će biti opisane karakteristike svakog programskog jezika. Posebno se ističe njihova interoperabilnost i učinkovitost.

3.1.1. HTML

HyperText Markup Language ili HTML je standardni jezik za označavanje (engl. *markup*) za dokumente dizajnirane za prikaz u web-pregledniku čije je prvo izdanje izašlo 1993. godine. Definira značenje i strukturu web-sadržaja. Zajedno s njim se često koriste i pomažu tehnologije kao što su Cascading Style Sheets (CSS) te skriptni jezici kao što je npr. JavaScript ili PHP.

web-preglednici primaju HTML dokumente s web-poslužitelja ili iz lokalne pohrane i učitavaju dokumente u multimedijske web-stranice. HTML semantički opisuje strukturu web-stranice i izvorno uključuje znakove za njezin izgled.

HTML elementi su građevni blokovi HTML stranica. Uz HTML konstrukcije i slike, drugi objekti kao što su interaktivni obrasci također mogu biti ugrađeni (engl. *embedded*) u prikazanu stranicu. HTML pruža sredstva za stvaranje strukturiranih dokumenata označavajući strukturnu semantiku za tekst kao što su naslovi, odlomci, popisi, poveznice, citati i druge stavke. HTML elementi su opisani (engl. *delineated*) oznakama napisanim u "kutnim" (engl. *angle*) zagradama (< i >). Oznake kao što su <img> i <input> izravno uvode sadržaj na stranicu. Ostale oznake kao što su i okružuju i pružaju informacije o tekstu dokumenta te mogu uključivati oznake podelementa. Preglednici ne prikazuju HTML oznake, ali ih koriste za interpretaciju sadržaja stranice [39].

HyperText Markup Language 5 ili HTML5 je peta i posljednja glavna HTML verzija koja je preporuka za korištenje u svim današnjim aplikacijama čije je prvo izdanje izašlo 2008. godine. Njegovi su ciljevi bili poboljšati jezik uz podršku za najnoviju multimediju, da održi jezik lako čitljivim ljudima i dosljedno razumljivim računalima i uređajima (kao što su web-preglednici i parseri) te ostati kompatibilan sa starijim softverom. HTML5 je namijenjen da obuhvati sve ranije verzije HTML-a.

HTML5 uključuje detaljne modele obrade za poticanje interoperabilnijih implementacija: proširuje, poboljšava i racionalizira označavanje dostupno za dokumente te uvodi označavanje sučelja za programiranje aplikacija (API) za složene web-aplikacije. Iz istih razloga, HTML5

se također koristi za višeplatformske mobilne aplikacije jer uključuje značajke dizajnirane za uređaje male snage. Uključene su mnoge nove sintaktičke značajke. Za izvorno uključivanje i rukovanje multimedijskim i grafičkim sadržajem, dodani su novi <video>, <audio> i <canvas> elementi, proširivi odjeljci nativno su implementirani kroz <summary>...</summary> i <details>... </details> umjesto da ovisi o CSS-u ili JavaScriptu, a dodana je i podrška za sadržaj skalabilne vektorske grafike (SVG) i MathML za matematičke formule. Kako bi se obogatio semantički sadržaj dokumenata, dodani su novi elementi strukture stranice kao što su <main>, <section>, <article>, <header>, <footer>, <aside>, <nav> i <figure>. Uvedeni su novi atributi, neki elementi i atributi su uklonjeni, a drugi kao što su <a>, <cite> i <menu> su promijenjeni, redefinirani ili standardizirani. Uz to, API-ji i DOM (engl. *Document Object Model*)) sada su temeljni dijelovi specifikacije HTML5 [40].

3.1.2. CSS

Cascading Style Sheets ili CSS je stilski jezik koji se koristi za opisivanje prezentacije dokumenta [41][42] napisanog u označnom jeziku kao što je HTML ili XML. CSS je temeljna tehnologija web-aplikacija, uz HTML i JavaScript.

CSS je dizajniran da omogući odvajanje sadržaja i prezentacije, uključujući izgled, boje i fontove. Ovo odvajanje može poboljšati dostupnost sadržaja, pružiti veću fleksibilnost i kontrolu u specifikaciji prezentacijskih karakteristika, omogućiti višestrukim web-stranicama dijeljenje oblikovanja određivanjem relevantnog CSS-a u zasebnoj .css datoteci (što smanjuje složenost i ponavljanje u strukturnom sadržaju) te omogućiti predmemoriranje .css datoteke kako bi se poboljšala brzinua učitavanja stranice između stranica koje dijele datoteku i njezino oblikovanje.

Odvajanje oblikovanja i sadržaja također čini izvedivim predstavljanje iste označne stranice u različitim stilovima za različite metode prikazivanja, kao što je na ekranu, u ispisu, glasom (putem preglednika temeljenog na govoru ili čitača zaslona) i na temelju Brailleovog pisma taktilni uređaji. CSS također ima pravila za alternativno oblikovanje ako se sadržaju pristupa na mobilnom uređaju.

Naziv kaskadno (engl. *cascading*) dolazi od navedene sheme prioriteta za određivanje pravila stila koje se primjenjuje ako više od jednog pravila odgovara određenom elementu [41][42].

Kaskadni redoslijed se određuje prema [41][42]:

1. Izvoru stila – postoje unaprijed definirani na web-pregledniku i korisnički definirani (oni koje definira programer web-aplikacije)
2. Važnosti – ako je neko svojstvo označeno sa !important, ima veću važnost nego ono koje nije
3. Specifičnosti – ovisno koji selektor ima najveći prioritet, postoje 4 kategorije selektora, poredani od najnižeg prioriteta do najvišeg prioriteta, to su:
    - univerzalni selektor - * (i neki pseudoelementi kao što je ::before)

- selektor oznake – npr. p (za paragraf)
- klasni selektori – npr. my-class (klasa koja ima niz svojstava u sebi), atributni selektori – npr. a[target] (svaki element a, odnosno anchor, koji se služi za linkove, će imati definiran stil za svoj link), i pseudoklase – npr. :hover (kad se postavi kursor miša na neki element, taj element će poprimiti neki definiran stil)
- ID selektori – npr. navigation-menu (ID je jedinstven (engl. *unique*), odnosno samo jednom se može definirati ID sa takvim imenom)

4. Redoslijedu – ako su dvije klase jednake, primijenit će se ona koja je redoslijedno kasnije u CSS-u

3.1.3. C#

C# je programski jezik opće namjene visoke razine koji podržava višestruke paradigme. C# obuhvaća statične, snažne, leksičke, imperativne, deklarativne, funkcionalne, generičke, objektno-orijentirane (temeljene na klasi) programske discipline te programske discipline orijentirane na komponente.

Microsoft je 2004. godine predstavio prvo izdanje C# [43] zajedno s .NET Framework i Visual Studio, oba su bila zatvorenog (engl. *closed source* [1]) koda. Četiri godine kasnije, započeo je besplatni projekt otvorenog (engl. *open source* [2]) koda pod nazivom Mono, koji je pružao međuplatformski kompajler i runtime okruženje za C# programski jezik. 1997. godine je pokrenut projekt Visual Studio (uređivač koda), alat koji se koristi pri izradi web-aplikacije u ovom završnom radu, a tek je 2012. godine predstavljena njegova završna verzija (više o Visual Studio-u kasnije). 2015. godine, Microsoft je objavio Visual Studio Code (uređivač koda) i jedinstvenu .NET platformu (softverski okvir), oboje podržavaju C#, besplatni su, otvorenog koda te su dostupni na više platformi. [45]

3.1.4. JavaScript

JavaScript, često skraćeno JS, je programski jezik koji je jedan od temeljnih tehnologija web-aplikacija, uz HTML i CSS. 2023. 98.7% web-mjesta koristi JavaScript na strani klijenta [46]. Često uključuje biblioteke trećih strana. Svi glavni web-preglednici imaju namjenski JavaScript mehanizam za izvršavanje koda na uređajima korisnika. [47]

JavaScript je jezik visoke razine, često kompajliran upravo na vrijeme (engl. *just-in-time* [3]. Ima dinamičan način pisanja, objektnu orijentaciju temeljenu na prototipu te prvoklasne funkcije. Ima više paradigmi, a podržava stilove programiranja vođene događajima, funkcionalne i imperativne. Ima sučelja za programiranje aplikacija (API) za rad s tekstom, datume, regularne izraze, standardne strukture podataka i DOM model (engl. *Document Object Model*). [47]

[1] Closed source kod se odnosi na računalni softver čiji je izvorni kod zatvoren, što znači da javnost nema pristup izvornom kodu. [44]

[2] Open source kod se odnosi na računalni softver čiji je izvorni kod otvoren što znači da mu šira javnost može pristupiti te koristiti ga [44]

[3] Just-in-time [48] je kompilacija računalnog koda tijekom izvođenja programa (engl. *runtime*)

Standard ECMAScript [49] ne uključuje nikakav ulaz/izlaz (I/O), kao što su umrežavanje, pohrana ili grafički sadržaji. U praksi, web-preglednik ili drugi runtime sustav pruža JavaScript API-je za I/O.

JavaScript "pogoni" (engl. *engines*) su se izvorno koristili samo u web-preglednicima, ali sada su ključne komponente nekih poslužitelja i raznih aplikacija. Najpopularniji runtime sustav za ovu upotrebu je Node.js [50].

Javascript se često mijenja za jezik Java, misleći da su isti, no zapravo su 2 potpuno različita programska jezika, te se uvelike razlikuju u dizajnu [47].

3.2. Blazor okvir

Blazor [51][51] je moderni web-okvir (engl. *framework* [4]), čije je prvo izdanje izašlo 2018. godine, a koji omogućuje izradu interaktivnih web-aplikacija koristeći programski jezik C# zajedno u kombinaciji s HTML-om i CSS-om, bez potrebe za JavaScriptom.

Prednosti Blazora su [53][54][55]:

- Jedinstvena kodna baza (engl. *codebase* [5]): Blazor omogućava razvoj frontend i backend dijela aplikacije pomoću jedne kodne baze čime se omogućava koherentnost (logički smisao) projekta. Kodna baza se može koristiti na više načina, ovisno o kompleksnosti komponente koja se pravi. Npr., ako je komponenta manjeg obujma, koja ne zahtijeva puno koda, na istom mjestu se mogu napisati svi dijelovi koda (HTML, CSS, C# i JavaScript), dok neke druge komponente, kompliciranije, radi koherentnosti se odvajaju u posebne datoteke (engl. *file*) – komponente djecu.
- Komponentno programiranje: Blazor stavlja naglasak na komponentno programiranje, odnosno pisanje koda koji je ponovno iskoristiv (engl. *reusable*), u smislu da se cijela komponenta može ponovno pozvati na nekoj drugoj stranici i da će u potpunosti ispravno raditi, time izbjegavajući pisanje duplog odnosno redundantnog koda. Time poboljšavamo održivost koda te značajno ubrzavamo razvojni proces same aplikacije [51].

  Blazor za komponentno programiranje koristi Razor Syntax. Upravo ono omogućuje kombinaciju HTML-a sa C# ili JavaScript-om. Svaka komponenta na kraju imena ima proširenje (engl. *extension*) .razor, npr. PrijavaKomponenta.razor, a može dodatno imati svoje "pod-komponente" – djecu, ovisno o namjeni koda koji će se pisati u njima, pa možemo reći da su u odnosu roditelj - dijete. Na taj način se može imati komponentu dijete za CSS, a u tom slučaju se komponenta dijete MORA zvati isto kao i komponenta roditelj pri čemu na kraju imena dodajemo njeno proširenje, npr. PrijavaKomponenta.razor.css (.css je dodan na kraju naziva). Ako se ne dodijeli ime komponente roditelj, komponenta dijete neće znati gdje svoj kod treba izvršavati. Postoji više vrsta komponenata djece koje se

[4] Framework je softverski okvir koji je dizajniran za podršku razvoju web-aplikacija uključujući web-usluge, web-resurse i web-API-je [52]

[5] Codebase je tijelo izvornog koda nekog softverskog programa, komponente ili sustava [56]

mogu napraviti, ali najviše se koriste .cs za C# i .js za JavaScript, pri čemu se mora paziti ako se radi komponentu dijete .cs, jer ta komponenta dijete mora biti djelomična (engl. *partial*) klasa, inače će program javljati greške [51][57].

- Dvosmjerno povezivanje podataka (engl. *two-way data binding*): Blazor omogućuje dvosmjerno povezivanje podataka, čime se olakšava automatsko ažuriranje korisničkog sučelja (engl. *user interface*) kad se podatci promijene i obratno. Podatci se mogu povezivati npr. iz HTML-a u C# i obratno [58].
- Integracija s .NET: Blazor je dio .NET okvira i .NET Core-a/.NET 5+, što znači da se mogu koristiti njihove prednosti, odnosno rad sa raznim dostupnim bibliotekama (engl. *libraries* [6]) i alatima koji su na raspolaganju, a preko kojih dolazimo pomoću NuGet-a (više o NuGet-u kasnije). Važno je napomenuti da su oni otvorenog izvora. Uz te prednosti, dolaze i napredne funkcije za upravljanje memorijom, sigurnošću te radom s velikim skupovima podataka [51].
- Interoperabilnost s JavaScript-om: iako Blazor omogućava rad bez JavaScript-a, on također podržava interoperabilnost s JavaScriptom, omogućavajući korištenje JavaScript biblioteka i funkcija po potrebi, ako se pokažu kao bolje rješenje u nekim slučajevima [51].

Postoji više različitih izdanja Blazor aplikacija, ali bitnije su [51]:

- Blazor Server Side
- Blazor WebAssembly
- Blazor Hybrid

Najviše se koriste BlazorWebAssembly i Blazor Server Side [60], ovisno o potrebama korisnika te namjeni aplikacije.

3.2.1. Blazor WebAssembly

Blazor WebAssembly je klijentski model Blazora koji omogućava izvršavanje C# koda izravno u web-pregledniku korištenjem WebAssembly-ja [60].

Prednosti Blazor WebAssembly-ja [7] su [60]:

- Klijentska strana izvršavanja: Blazor WebAssembly omogućava izvršavanje koda izravno u web-pregledniku, smanjujući opterećenje na poslužitelju (engl. *server*) te potencijalno omogućavajući brži odziv aplikacije.

[6] Library je programska biblioteka tj. zbirka unaprijed napisanog koda koji programeri mogu koristiti za optimizaciju zadataka [59]

[7] WebAssembly je binarni format instrukcija za virtualni stroj temeljen na stogu [61]

- Izvanmrežni (engl. *offline*) rad: Jedna od prednosti Blazor WebAssembly-ja je mogućnost omogućavanja izvanmrežnog rada, što znači da aplikacija može nastaviti s radom čak i kad nije povezana s internetom.
- Sigurnost i izolacija: Blazor WebAssembly pruža sigurno okruženje (engl. *environment*) za izvršavanje koda, s različitim mehanizmima za izolaciju koda od preglednika domaćina (engl. *host*) i korisničkog sustava.
- Optimizirano vrijeme učitavanja: Iako prva učitavanja mogu biti duža zbog preuzimanja potrebnih resursa, Blazor WebAssembly optimizira učitavanje kroz korištenje različitih mehanizama kao što su lijeno učitavanje (engl. *lazy loading* [8]) i keširanje (engl. *caching* [9]).

3.2.2. Blazor Server Side

Blazor Server Side je poslužiteljski model Blazora koji omogućava izvršavanje Blazor komponenata na poslužiteljskoj strani, dok se korisničko sučelje učitava (engl. *render*) na klijentskoj strani [60].

Prednosti Blazor Server Side-a su [60]:

- Manje opterećenje na klijentu: Budući da se veći dio koda izvršava na poslužitelju, klijentska strana ima manje opterećenje, što rezultira bržim vremenima učitavanja i boljim ukupnim izvođenjem (engl. *performance*).
- Potpuni pristup resursima na poslužitelju: Blazor Server Side omogućava potpuni pristup resursima na poslužitelju, uključujući baze podataka i druge poslužiteljske (engl. *server-side*) resurse, čime se pojednostavljuje integracija s backend tehnologijama i alatima.
- SignalR: Blazor Server Side koristi SignalR za komunikaciju između klijenta i poslužitelja, omogućavajući brz i učinkovit dvosmjerni prijenos podataka.
- Povećana sigurnost: S obzirom na to da se veći dio koda izvršava na poslužiteljskoj strani, to pomaže u povećanju sigurnosti aplikacije, jer se osjetljivi podatci i logika čuvaju na sigurnom poslužiteljskom okruženju.

3.3. Alati

3.3.1. Visual Studio

Visual Studio [64] je integrirano razvojno okruženje (IDE) tvrtke Microsoft. Koristi se za razvoj računalnih programa, uključujući web-stranice, web-aplikacije, web-usluge i mobilne

[8] Lazy loading je tehnika kojom se čeka da se učitaju određeni dijelovi web-stranice (posebno slike) dok ne budu potrebni [62]

[9] Caching je proces pohranjivanja kopija datoteka u predmemoriju ili privremenu lokaciju za pohranu, tako da im se može brže pristupiti [63]

aplikacije. Visual Studio koristi Microsoftove platforme za razvoj softvera kao što su Windows API, Windows Forms, Windows Presentation Foundation, Windows Store i Microsoft Silverlight. Može proizvesti izvorni (engl. *native*) kod i upravljani (engl. *managed*) kod.

Visual Studio uključuje uređivač koda koji podržava IntelliSense (komponentu za dovršetak koda), kao i refaktoriranje koda. Integrirani debugger radi i kao debugger na razini izvora i kao debugger na razini stroja. Ostali ugrađeni alati uključuju alat za profiliranje koda, dizajner za izradu GUI aplikacija, web-dizajner, dizajner klasa i dizajner sheme baze podataka. Prihvaća dodatke koji proširuju funkcionalnost na gotovo svim razinama — uključujući dodavanje podrške za sustave kontrole izvora (kao što je npr. Git) i dodavanje novih skupova alata poput uređivača i vizualnih dizajnera za jezike specifične za domenu ili skupova alata za druge aspekte razvoja softvera. [64]

Visual Studio podržava 36 različitih programskih jezika i omogućuje uređivaču koda i programu za ispravljanje pogrešaka da podrže (u različitim stupnjevima) gotovo bilo koji programski jezik, pod uvjetom da postoji usluga specifična za jezik. Ugrađeni jezici uključuju C, C++, C++/CLI, Visual Basic .NET, C#, F#, JavaScript, TypeScript, XML, XSLT, HTML i CSS. Podrška za druge jezike kao što su Python, Ruby, Node.js i M među ostalima dostupna je putem dodataka. Java (i J#) su bile podržane u prošlosti. [64]

Najosnovnije izdanje Visual Studija, izdanje Community, dostupno je besplatno. Slogan izdanja Visual Studio Community je "Besplatno, potpuno opremljeno IDE za studente, open-source i za individualne programere". [64]

3.3.2. NuGet

NuGet [65] je mehanizam za .NET (uključujući .NET Core), korišten za dijeljenje koda koji podržava Microsoft. NuGet definira kako se paketi za .NET stvaraju, poslužuju i koriste te pruža alate za svaku od tih uloga.

Bitan alat za svaku modernu razvojnu platformu je mehanizam putem kojeg programeri mogu stvarati, dijeliti i koristiti koristan kod. Često je takav kod povezan u "pakete" koji sadrže kompajlirani (engl. *compiled*) kod (kao DLL-ove) zajedno s drugim sadržajem potrebnim u projektima koji koriste te pakete.

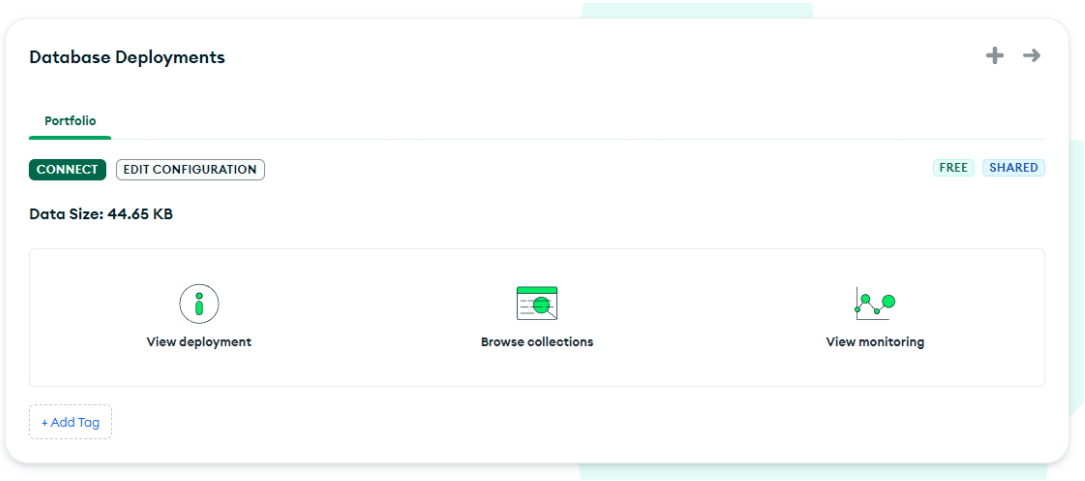
Jednostavno rečeno, NuGet paket je jedna ZIP datoteka s ekstenzijom .nupkg koja sadrži kompajlirani kod (DLL), druge datoteke povezane s tim kodom i opisni tekst koji uključuje informacije poput broja verzije paketa. Programeri s kodom za dijeljenje stvaraju pakete i objavljuju ih na javnom ili privatnom domaćinu (engl. *host*). Potrošači paketa dobivaju te pakete od odgovarajućih domaćina, dodaju ih svojim projektima, a zatim pozivaju funkcionalnost paketa u svom kodu projekta. Sam NuGet zatim obrađuje sve međupojedinosti.

Budući da NuGet podržava privatne domaćine uz javni host nuget.org, mogu se koristiti NuGet paketi za dijeljenje koda koji je ekskluzivan za organizaciju ili radnu grupu. Također se mogu koristiti NuGet paketi kao prikladan način za faktoriziranje vlastitog koda za korištenje samo u vlastitim projektima. Ukratko, NuGet paket je jedinica koda koja se može dijeliti, ali ne zahtijeva niti podrazumijeva bilo koji poseban način dijeljenja. [65]

U ovom radu će se koristiti tri NuGet paketa:

1. **MongoDB Driver**

   - Službeni MongoDB Node.js Driver NuGet paket omogućuje Node.js aplikacijama da se povežu na MongoDB i rade s podatcima. MongoDB Driver ima asinkroni API koji omogućuje interakciju s MongoDB-om koristeći povratne objekte (engl. *promise* [10]) ili putem tradicionalnih povratnih poziva. [67]
   - U ovom radu konkretno, MongoDB Driver će se koristiti kako bi se povezalo sa bazom podataka koja se nalazi na **https://www.mongodb.com/atlas/database** te kako bi se mogla vršiti manipulacija podataka kroz web-aplikaciju. MongoDB Atlas omogućuje stvaranje tzv. klastera (vidljivo na slici 1) (engl. *cluster* [68]) koji u sebi sadrže baze podataka, a te baze podataka dokumente u kolekcijama. U ovom radu, klaster ima naziv Portfolio (vidljivo na slici 2).

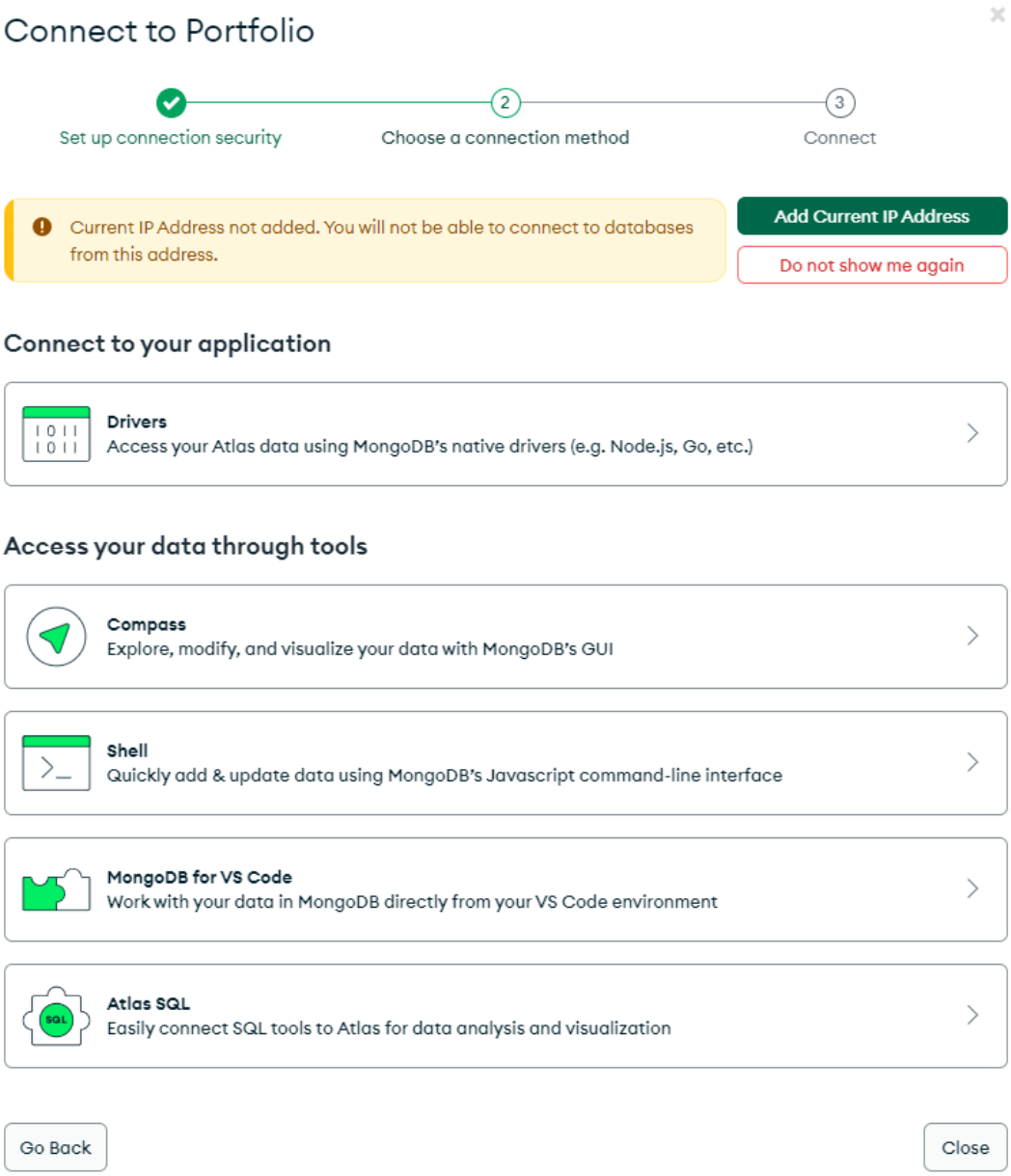


Slika 1: Primjer klastera

Klasteri imaju svoj "niz za povezivanje" (engl. *connection string* [11]) koji omogućuje spajanje na njega kako bi se mogla vršiti manipulacija podatcima u njemu.

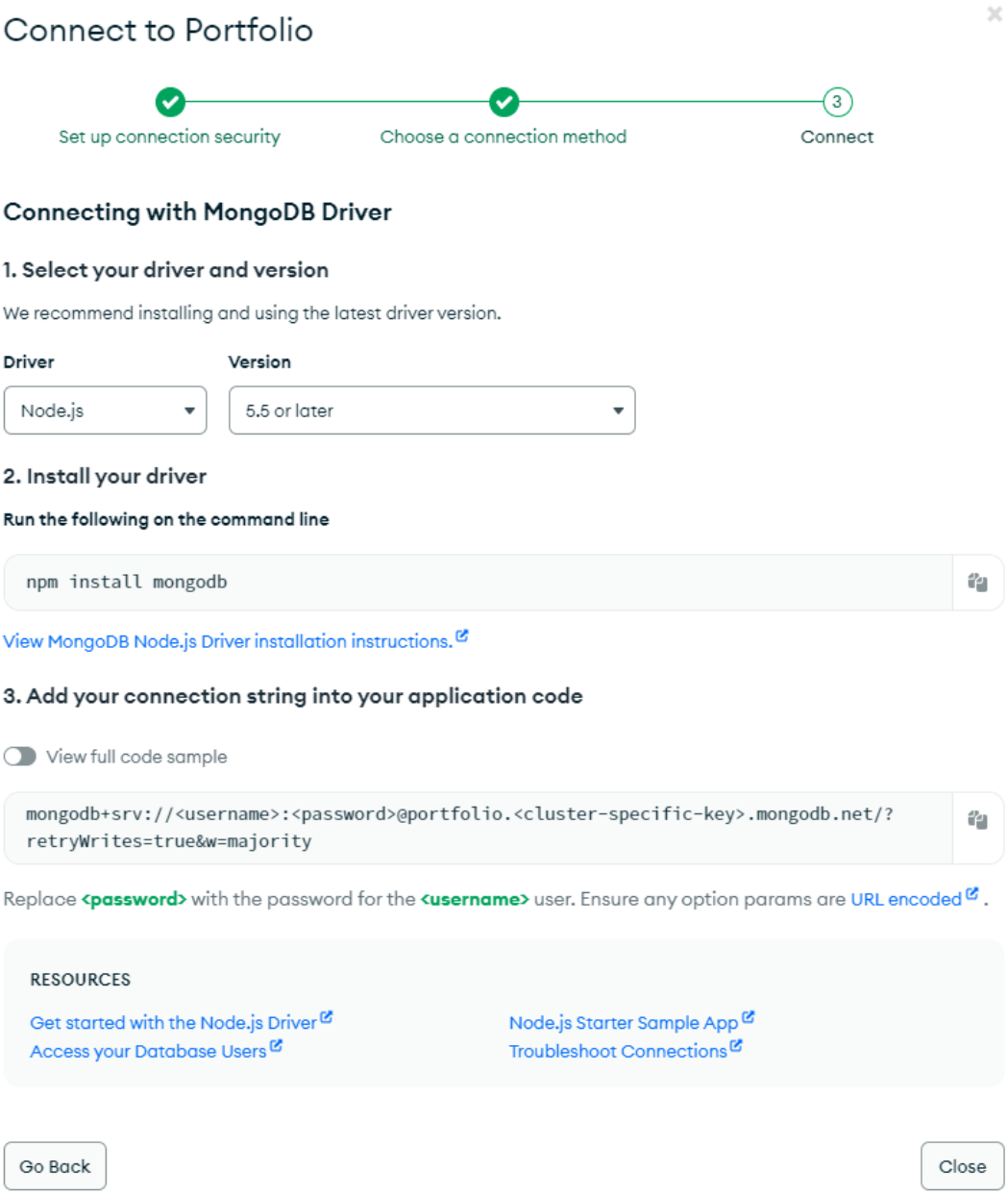
[10] Promise je objekt vraćen pozivom asinkrone metode koja omogućuje pristup informacijama ovisno o eventualnom uspjehu ili neuspjehu operacije koje obavljaju. [66]

[11] U programiranju, niz za povezivanje je niz koji specificira informacije o izvoru podataka i načinu povezivanja s njim. [69]



Slika 2: Izbornik za povezivanje na klaster

Klaster nudi opciju za spajanje na web-aplikaciju pomoću drivera (vidljivo na slici 2). Kad se ta opcija odabere, dobiju se dodatne informacije za spajanje na klaster, kao i niz za povezivanje (vidljivo na slici 3).



Slika 3: Metoda za spajanje

Niz za povezivanje se kopira (ona unaprijed postavi korisničko ime i ključ klastera (engl. *cluster-specific-key*) koji iz razloga potencijalne krađe podataka ovdje nisu navedeni, dodatno se samo <password> zamijeni sa pravom lozinkom korisničkog računa koji je trenutno prijavljen na MongoDB Atlas) te se jednostavno u projektu pozove ondje gdje je potrebno spajanje na bazu podataka.

U web-aplikaciji koja se opisuje kroz ovaj rad, niz za povezivanje je deklarirana kao konstanta, odnosno statična varijabla (princip engl. *clean code* [12]).

2. **MailKit**

[12]Clean code je kod koji je jednostavan za čitanje, održavanje, razumijevanje i mijenjanje kroz strukturu i dosljednost, ali ostaje robustan i siguran kako bi izdržao zahtjeve performansi [70]

- MailKit je višeplatformska, popularna biblioteka otvorenog koda koja se koristi kao klijent e-pošte za .NET [71].
- U ovom radu se koristi Gmail kao e-mail servis kako bi se mogao iskoristiti MailKit NuGet paket za slanje e-pošte kako korisniku, tako i "sami sebi". Mailkit funkcionira tako da se poveže na domaćina e-mail servisa pomoću imena servisa i priključka, odnosno u ovom slučaju bi domaćin e-mail servisa bio **smtp.gmail.com**, a priključak **587**. Nakon toga se dodatno MailKit-u proslijede podatci e-maila i lozinke koji će web-aplikacija koristiti za slanje e-pošte korisnicima ili "sama sebi".
- **Primjer:** slučaj slanja e-pošte korisnicima bi bio kad oni pošalju zahtjev za zaboravljenu lozinku, dok bi slučaj slanja e-pošte "sama sebi" bio ako bi korisnici poslali zahtjev za kontakt, u tom slučaju oni ne koriste svoj e-mail, nego ga web-aplikacija šalje "sama sebi".

3. **SixLabors.ImageSharp**

- ImageSharp biblioteka je nova, potpuno opremljena, višeplatformska biblioteka u obliku 2D grafičkog programskog sučelja koja je dizajnirana za pojednostavljenje obrade slika [72].
- ImageSharp je dizajniran od temelja da bude fleksibilan i proširiv. Biblioteka pruža krajnje točke (engl. *endpoint*) programskog sučelja za uobičajene operacije obrade slike i građevne blokove koji omogućuju razvoj dodatnih operacija.
- Izgrađena na temelju .NET 6, ImageSharp biblioteka se može koristiti u scenarijima uređaja, pohrane na oblaku i ugrađenih "internet stvari" (engl. *internet of things*) scenarija.

3.3.3. Git

Git je distribuirani sustav za kontrolu verzija koji prati promjene u bilo kojem skupu računalnih datoteka, obično se koristi za koordinaciju rada među programerima koji zajednički razvijaju izvorni kod tijekom razvoja softvera. Njegove karakteristike uključuju brzinu, integritet podataka i podršku za distribuirane, nelinearne tijekove rada (tisuće paralelnih grana koje rade na različitim računalima).

Kao i kod većine drugih distribuiranih sustava za kontrolu verzija, i za razliku od većine klijent-poslužiteljskih sustava, svaki Git direktorij na svakom računalu je punopravno spremište s potpunom poviješću i punim mogućnostima praćenja verzija, neovisno o mrežnom pristupu ili središnjem poslužitelju. Git je besplatan softver otvorenog koda koji se dijeli samo pod licencom GPL-2.0. [73]

4. Razvoj web-aplikacije

U ovom dijelu završnog rada je podrobnije opisano kako su tehnologije i alati primijenjeni u izradi web-aplikacije. Poglavlje je konstruirano tako da omogućuje postupno razumijevanje koraka pri izradi web-aplikacije ovog završnog rada kroz detaljne razrade svake funkcionalnosti.

U nastavku će biti opisane komponente aplikacije uz naglasak na integraciju s MongoDB bazom podataka. Također su prikazani primjeri koda koji su korišteni u praksi te koji demonstriraju kako su različite funkcionalnosti implementirane u korelaciji s bazom podataka, kao i slike zaslona (engl. *screenshot*) koje opisuju konačni izgled web-aplikacije.

Naposlijetku, cilj ovog poglavlja je pružiti uvid u proces razvoja web-aplikacije, od inicijalne ideje, planiranja i implementacije do završnog testiranja i prezentacije web-aplikacije.

4.1. Ideja i raspored komponenata

Prije samog programiranja web-aplikacije, uvijek je dobro napraviti prvotnu ideju i/ili plan, odnosno što bi aplikacija trebala moći raditi i na koji način. Nakon toga, bilo bi poželjno nacrtati inicijalni dizajn npr. na papir (što je puno brže, no dosta nepreglednije u većini slučajeva) ili ga napraviti u nekom od dostupnih alata na internetu kao što su npr. Figma [74], Balsamiq [75], Adobe XD [76]... Na taj se način osigurava puno preciznija slika konačnog izgleda aplikacije te se uvelike smanjuje vrijeme potrebno za naknadne promjene u aplikaciji u slučaju da se otkrije da je nešto pogrešno zamišljeno ili jednostavno prekomplicirano za implementirati (a to se često dogodi te kasnije iziskuje puno vremena za pronalazak alternative koje se moglo utrošiti na programiranje ostalih dijelova aplikacije).

U ovom slučaju, aplikacija je trebala moći omogućiti pregled podataka o autoru (zato i jest naziv portfelj (engl. *portfolio*), a uz to, imati interakciju s bazom podataka. Kako bi se to ostvarilo, ideja je bila implementirati prijavu i registraciju korisnika te ovisno o tome prikazati ostale komponente.

Na inicijalnom pokretanju aplikacije, korisnika dočekuje početni zaslon (engl. *home screen*) koji prikazuje osnovne komponente, odnosno stranice: Početna, O meni, Vještine, Kontakt i Prijava, a kad se korisnik prijavi (uz to da se prethodno registrirao) dobit će vizualno još jednu stranicu - Novosti i objave, a kad klikne na svoje korisničko ime, prikazat će mu se padajući izbornik (engl. *dropdown*) s opcijama (stranicama): Profil, Postavke i Odjava.

4.2. Implementacija komponenata i funkcionalnosti

Svaka stranica ne zahtijeva kompleksan posao što se dizajna tiče, za izradu su korišteni HTML, CSS i JavaScript kako bi se postavio inicijalan izgled stranice te kako bi se dinamično (ovisno o rezoluciji ekrana) postavila visina glavnog kontejnera (HTML element - div) s elementima (koja radi i kad korisnik dinamično mijenja visinu i/ili širinu preglednika (engl. *browser*).

Izuzetak su komponente za prijavu, registraciju, odjavu i zaboravljenu lozinku korisnika.

Dodatno, na svakoj stranici se na inicijalnom učitavanju svakom korisniku postavlja unikatni identifikator za sesiju, koji se potom sprema u bazu podataka (MongoDB) te se rade provjere je li korisnik ulogiran ili ne, ako je, prikažu mu se dodatne prikladne veze u navigacijskom izborniku koje mora vidjeti, a ako nije, prikažu mu se sve unaprijed zadane veze u navigacijskom izborniku.

Napomena: za prikaz ikona na navigacijskom izborniku te ostalih ikona pronađenih u aplikaciji, korištena je stranica Font Awesome [77].

4.2.1. Navigacijski izbornik

Pri kreiranju nove Blazor aplikacije, aplikacija ima neke osnovne stvari kao što su početne tri komponente te navigacijski izbornik (vidljivo na slici 4) koji se u aplikaciji može naći pod imenom "NavMenu.razor". Navigacijski izbornik se po potrebi može stilizirati upotrebom CSS-a.

Slika 4: Navigacijski izbornik (neprijavljen korisnik)

Isječak koda 4.1: Navigacijski izbornik

```
<NavLink class="navigation-link" href="o-meni" @onclick="CloseNavigationMenu">
    <i class="fa-solid␣fa-address-card" />

    O meni
</NavLink>
```

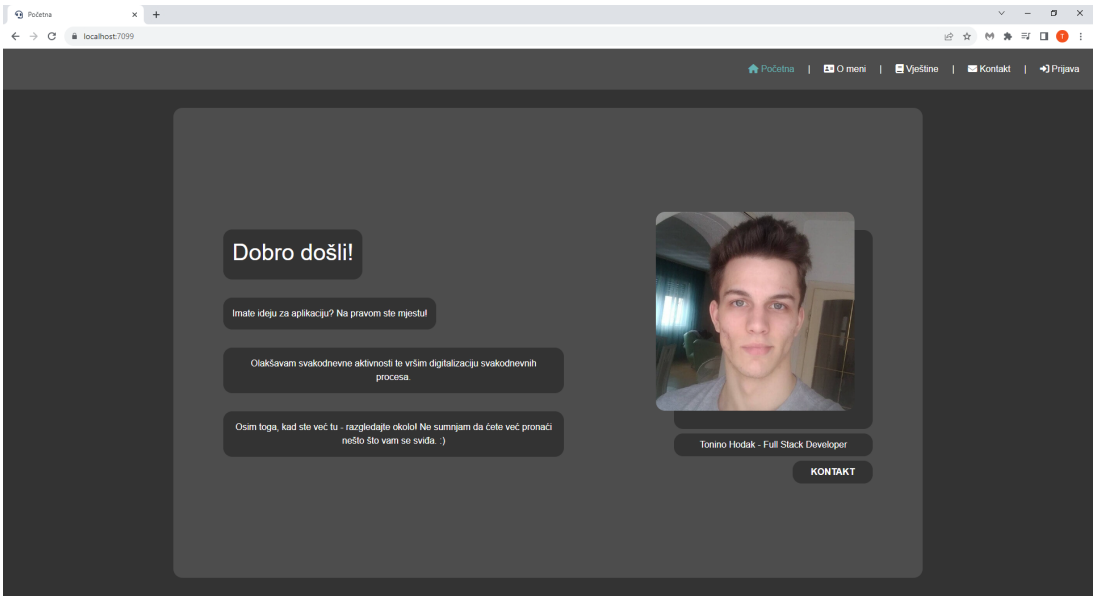
Dodavanje novih veza nije problem, potrebno ih je dodati unutar NavMenu.razor komponente na za to predviđeno mjesto, može ih se doslovno kopirati i zalijepiti (engl. *copy-paste*) i prikladno izmijeniti.

Opis komponente vidljiv na programskom kodu (4.1):

- NavLink je zadana komponenta za preusmjeravanje (engl. *routing*) Blazor frameworka
- class predstavlja klasu u CSS-u koja se primjenjuje za stiliziranje
- href predstavlja lokaciju na koju će aplikacija preusmjeriti korisnika prilikom klika
- @onclick="CloseNavigationMenu" predstavlja koja klasa će se izvršiti prilikom klika na vezu (uz osnovnu radnju, a to je preusmjeravanje na neku lokaciju)

Dodatno, unutar komponente se nalazi opis veze, odnosno u ovom slučaju ikona korisničke kartice i opis "O meni"

4.2.2. Početna



Slika 5: Početni zaslon (neprijavljen korisnik)

Isječak koda 4.2: Početna stranica - elementi

```
<div class="py-3" />

<div id="full-height-div" class="container container-holder">
    <div class="container-holder-div">
        <div>
            <h1>
                Dobro dosli!
            </h1>
        </div>

        <div>
            Imate ideju za aplikaciju? Na pravom ste mjestu!
        </div>

        <div>
            Olaksavam svakodnevne aktivnosti te vrsim digitalizaciju svakodnevnih
                procesa.
        </div>

        <div>
            Osim toga, kad ste vec tu - razgledajte okolo! Ne sumnjam da cete vec
                pronaci nesto sto vam se svida. :)
        </div>
    </div>

    <div class="image-holder">
        <div>
            <img src="/Images/Home/picture-me.jpg" />
        </div>
```

```
        <div>
            Tonino Hodak - Full Stack Developer
        </div>

        <div class="button-default-portfolio">
            <a href="kontakt">
                Kontakt
            </a>
        </div>
    </div>
</div>

<div class="py-3" />
```

U ovom slučaju su informacije "hard-kodirane", odnosno eksplicitno je napisano što treba biti vidljivo korisniku na stranici (vidljivo na 5, 4.2). Kasnije (po potrebi) se može implementirati prijevod, koji radi na principu ključ - vrijednost (engl. *key-value*). Npr. ResXManager [78] odlično obavlja tu funkcionalnost, odnosno upisalo bi se npr. "@Localizer["Welcome"] gdje je Welcome ključ, a vratila bi se vrijednost ovisno o postavljenom jeziku, ako je npr. hrvatski, pisalo bi "Dobro došli", a ako je npr. engleski, pisalo bi "Welcome".

Isječak koda 4.3: Početna stranica - dizajn

```
.image-holder {
        display: flex;
        flex-flow: column;
        align-items: end;
        width: 400px;
        height: auto;
        margin: auto;
        padding-top: 3em;
    }

        .image-holder > div:first-child {
            background-color: @Constants.ColorGrayFullDark;
            position: relative;
            width: 350px;
            height: 350px;
            border-radius: 1em;
        }

        .image-holder > div:nth-child(2) {
            display: inline-block;
            background-color: @Constants.ColorGrayFullDark;
            width: 350px;
            text-align: center;
            color: white;
            border-radius: 1em;
            padding: 0.5em 1em;
            margin: 0.5em 0;
        }
```

```
            .image-holder div > img {
                position: absolute;
                width: 350px;
                height: auto;
                top: -2em;
                left: -2em;
                border-radius: inherit;
            }
```

Pri izradi CSS-a je po potrebi korišteno gniježđenje (engl. *nesting*), odnosno manipulira se hijerarhijom elemenata kako bi se postigao određeni rezultat (vidljivo na 4.3).

Isječak koda 4.4: Početna stranica - dizajn ovisno o veličini

```
@@media (max-width: 991px) {
        .image-holder {
            width: 100%;
        }

        .image-holder div > img {
            position: unset;
            width: 100%;
        }

        .image-holder > div:first-child {
            width: 100%;
            height: auto;
            background-color: transparent;
        }

        .image-holder > div:nth-child(2) {
            width: 100%;
        }
    }
```

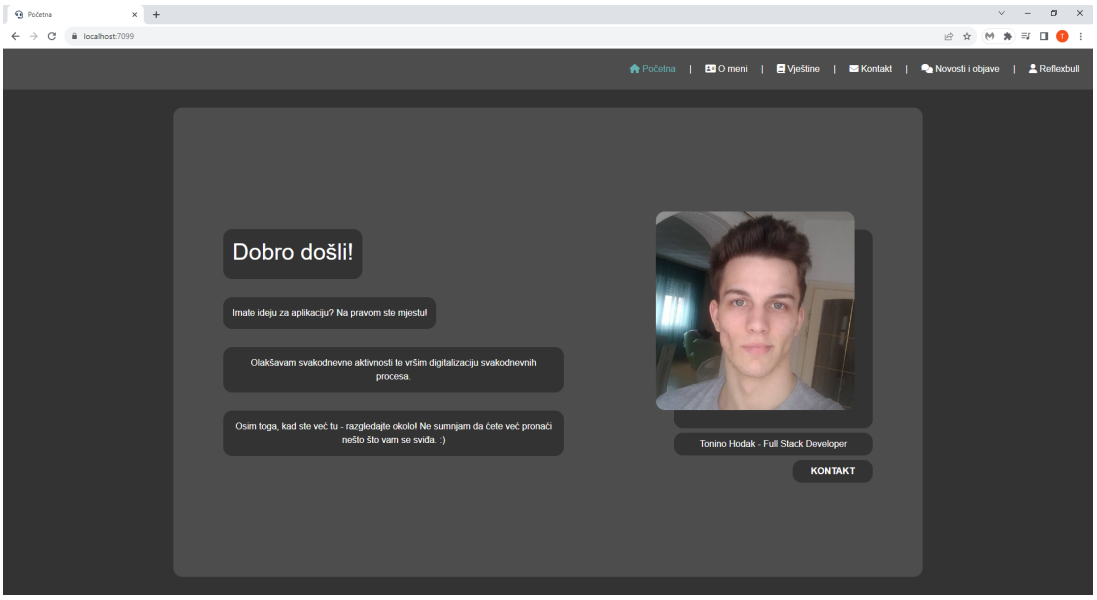
Osim toga, korištena je tehnika media-query (vidljivo na 4.4), a pomoću nje se ovisno o širini ekrana mogu promijeniti postavke već postojećih CSS klasa (npr. promijeniti raspored elemenata, postaviti druga boja, drugi font, ...).

Isječak koda 4.5: Konstanta za boju

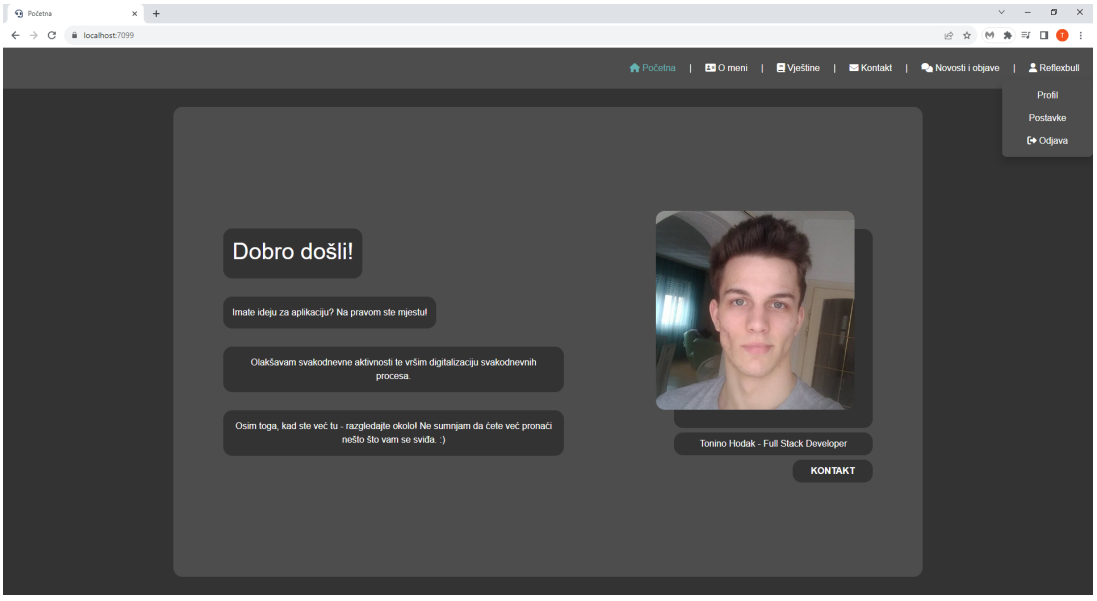
```
public static string ColorGrayFullDark { get; set; } = "#333333;";
```

@Constants.ColorGrayFullDark (vidljivo na 4.5) je posebnost Blazor frameworka koja omogućuje da se negdje u aplikaciji (u ovom slučaju u statičnoj klasi Constants) uz klasične upotrebe svojstava i funkcija (metoda), kreiraju stringovi (u ovom slučaju public string ColorGrayFullDark get; set; "#333333;"; gdje je #333333 HEX [1] - vrijednost boje) koji mogu poslužiti kao boja u nekoj css klasi (ili prilikom primjene stiliziranja direktno na element pomoću style="background-color = @Constants.ColorGrayFullDark;"...).

[1] HEX kod boje je heksadecimalni način predstavljanja boje u RGB formatu kombiniranjem tri vrijednosti – količine crvene, zelene i plave u određenoj nijansi boje. [79]



Slika 6: Početni zaslon (prijavljen korisnik)



Slika 7: Početni zaslon (prijavljen korisnik - padajući izbornik)

Početna stranica (vidljivo na 6) prijavljenog korisnika dolazi uz neke preinake. U navigacijskom izborniku se vrše provjere kako na back-endu, tako i na front-endu.

Isječak koda 4.6: Provjera je li korisnik ulogiran

```
protected override async Task OnInitializedAsync()
{
    await UserService.CheckIfUserIsLoggedIn();

    CheckingSession = false;
}
```

Na inicijalnom učitavanju stranice (vidljivo na 4.6), back-end provjerava je li korisnik

trenutno prijavljen. Dodatno, svojstvo private bool CheckingSession (hrv. *provjera sesije*) se postavlja na false (hrv. *laž*) koja odgodi (engl. *delay*) prikaz navigacijskog izbornika dok se ne utvrdi je li korisnik prijavljen ili nije.

Isječak koda 4.7: Navigacijski izbornik - elementi

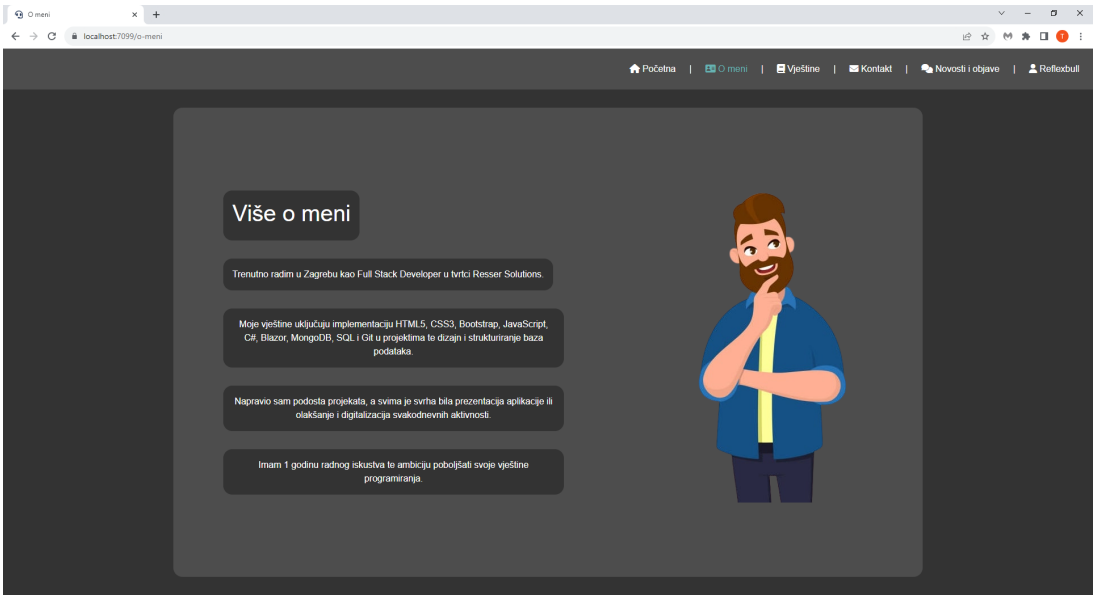
```
@if (UserService.CurrentUser != null && UserService.CurrentUser.LoggedIn)
{
    <div class="pipe-separator">
        |
    </div>

    <div>
        <NavLink class="navigation-link" href="novosti-i-objave" @onclick='async ()
            => {CloseNavigationMenu(); await OnTabChange("/novosti-i-objave");}'>
            <i class="fa-solid␣fa-comments" />

            Novosti i objave
        </NavLink>
    </div>

    <div class="pipe-separator">
        |
    </div>

    <div id="dropdown-menu-profile-parentID">
        <NavLink class="navigation-link" @onclick='ToggleProfileDropdownMenu'>
            <i class="fa-solid␣fa-user" />

            @UserService.CurrentUser.Username
        </NavLink>

        @if (ShowProfileDropdownMenu)
        {
            <div id="dropdown-menu-profileID" class="dropdown-menu-profile">
                <div>
                    <a href="profil/@UserService.CurrentUser.Username" @onclick='
                        async () => { ToggleProfileDropdownMenu();
                        CloseNavigationMenu(); await OnTabChange("/profil"); }'>
                        Profil
                    </a>
                </div>

                <div>
                    <a href="postavke" @onclick='async () => {
                        ToggleProfileDropdownMenu(); CloseNavigationMenu(); await
                        OnTabChange("/postavke");}'>
                        Postavke
                    </a>
                </div>

                <div>
                    <a href="odjava" @onclick='async () => {
```

```
                        ToggleProfileDropdownMenu(); CloseNavigationMenu(); await
                        OnTabChange("/odjava");}'>
                        <i class="fa-solid␣fa-right-from-bracket" /> @* Logout icon
                            *@

                        Odjava
                    </a>
                </div>
            </div>
        }
    </div>
}
else
{
    <div class="pipe-separator">
        |
    </div>

    <div>
        <NavLink class="navigation-link" href="prijava" @onclick="
            CloseNavigationMenu">
            <i class="fa-solid␣fa-right-to-bracket" /> @* Login icon *@

            Prijava
        </NavLink>
    </div>
}
```

Nakon toga, vrše se dodatne provjere na front-endu (vidljivo na 4.7) koje će ovisno o stanju koje je back-end utvrdio, ili prikazati "Prijava" u navigacijskom izborniku (ako korisnik nije prijavljen) ili će prikazati "Novosti i objave" uz korisničko ime korisnika (ako je korisnik prijavljen) (vidljivo na 7).

4.2.3. O meni

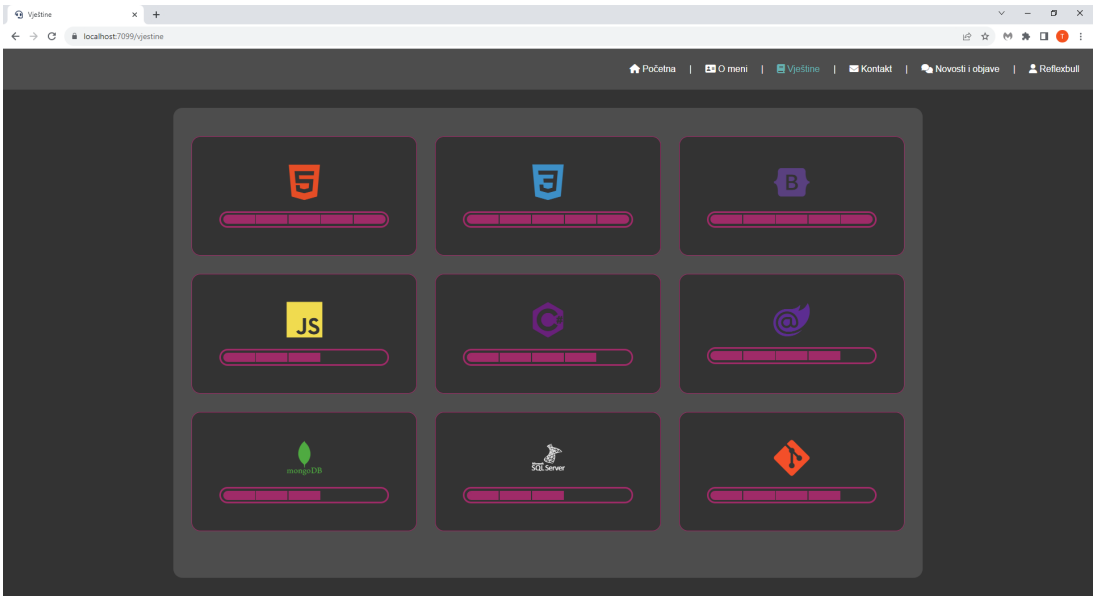
Stranica "O meni" (vidljivo na 8) je izrađena na isti princip kao i stranica "Početna" uz manje izmjene teksta te slike koja je prikazana. Također na inicijalnom učitavanju stranice koristi JavaScript kako bi se promijenila veličina elementa ovisno o visini i širini ekrana korisnika.



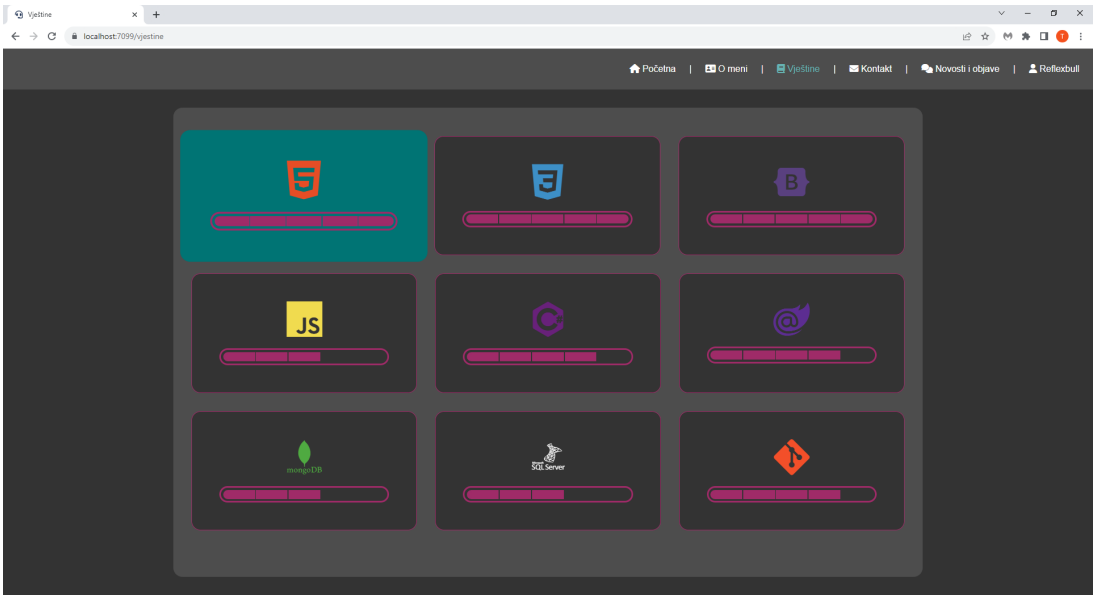
Slika 8: Stranica "O meni" (prijavljen korisnik)

4.2.4. Vještine

Stranica "Vještine" (vidljivo na 9) je više - manje izrađena na isti princip kao i stranica "Početna" uz manje izmjene teksta te slike koja je prikazana. **Dodatne** promjene uključuju upotrebu :hover selektora, kad se pokazivačem miša prijeđe preko jedne od prikazanih vještina, ona se zaplavi i poveća (vidljivo na 10).

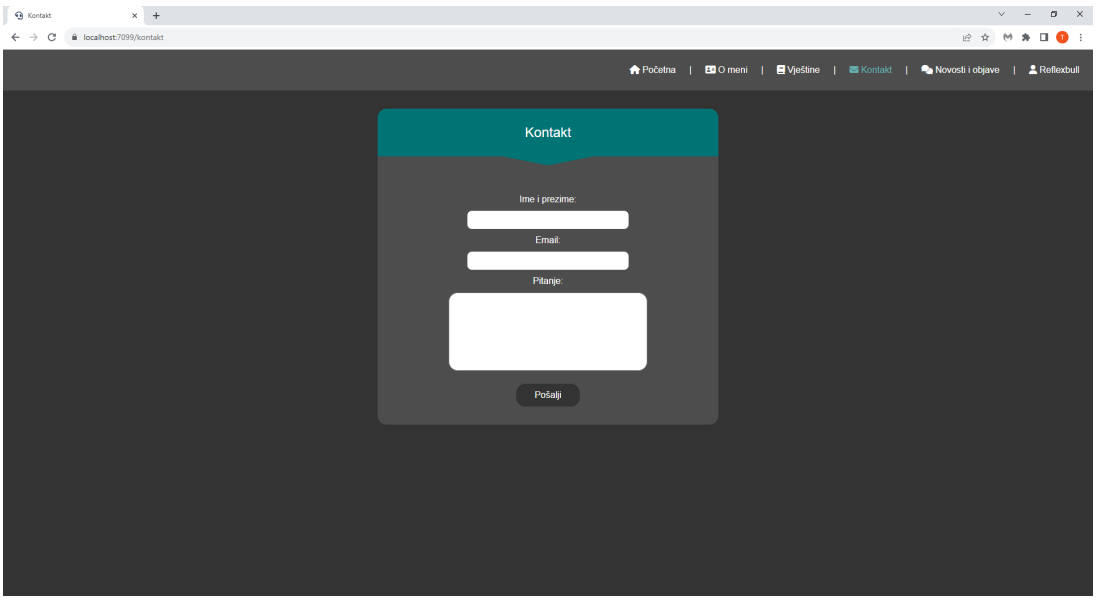


Slika 9: Stranica "Vještine" (prijavljen korisnik)



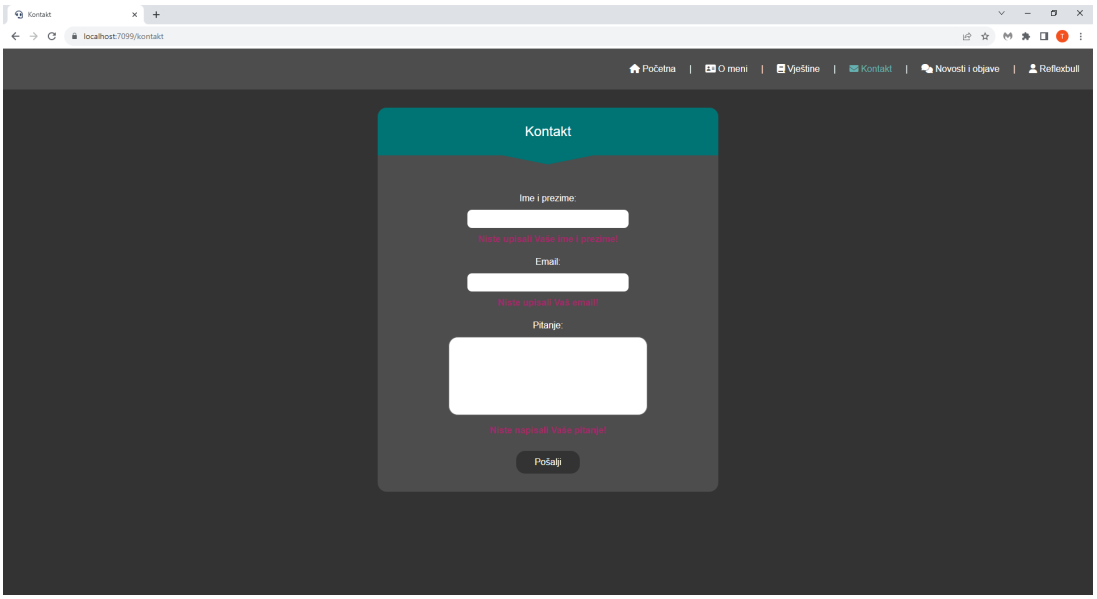
Slika 10: Vještine nakon postavljanja pokazivača miša

4.2.5. Kontakt

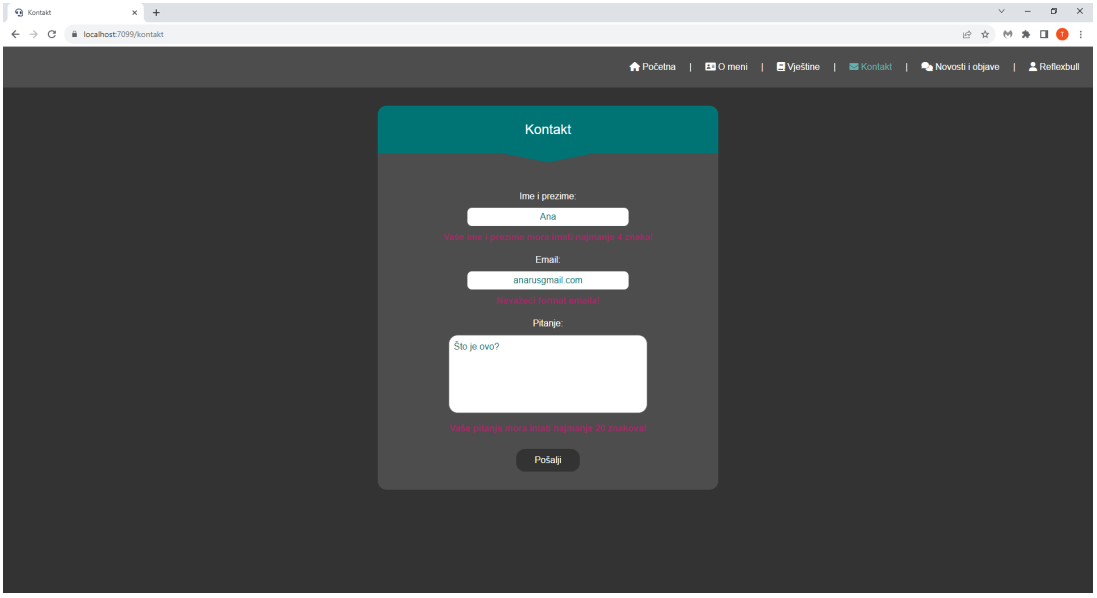


Slika 11: Stranica "Kontakt" (prijavljen korisnik)

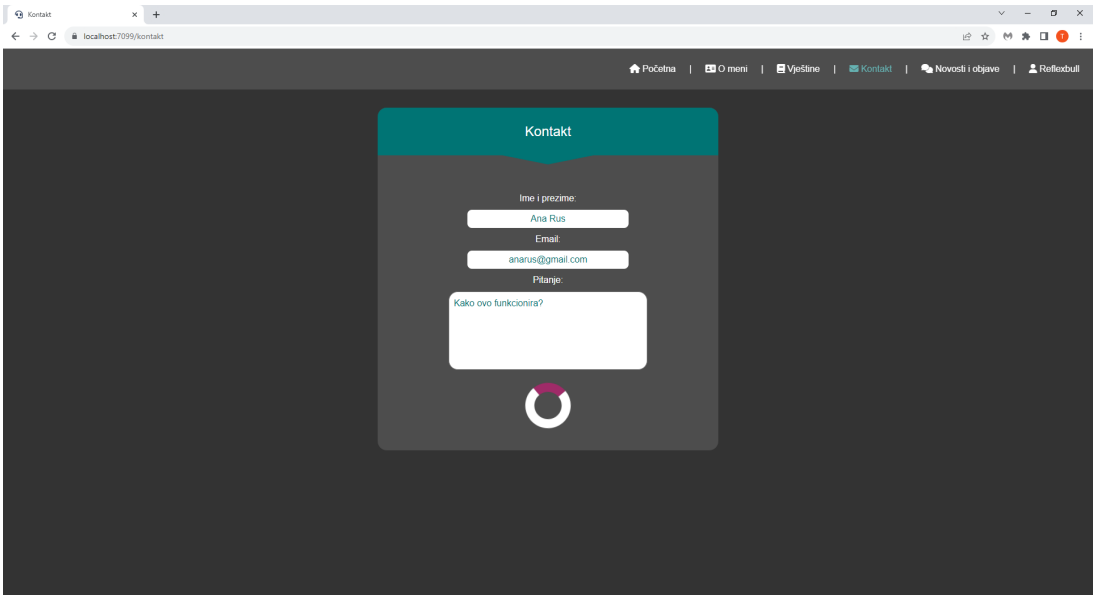
Ova stranica ima kontakt formu (vidljivo na 11) koju korisnik (bio on prijavljen ili ne) može ispuniti kako bi postavio neko određeno pitanje, npr. ako ima neki problem ili mu je potrebno nešto više razjasniti.



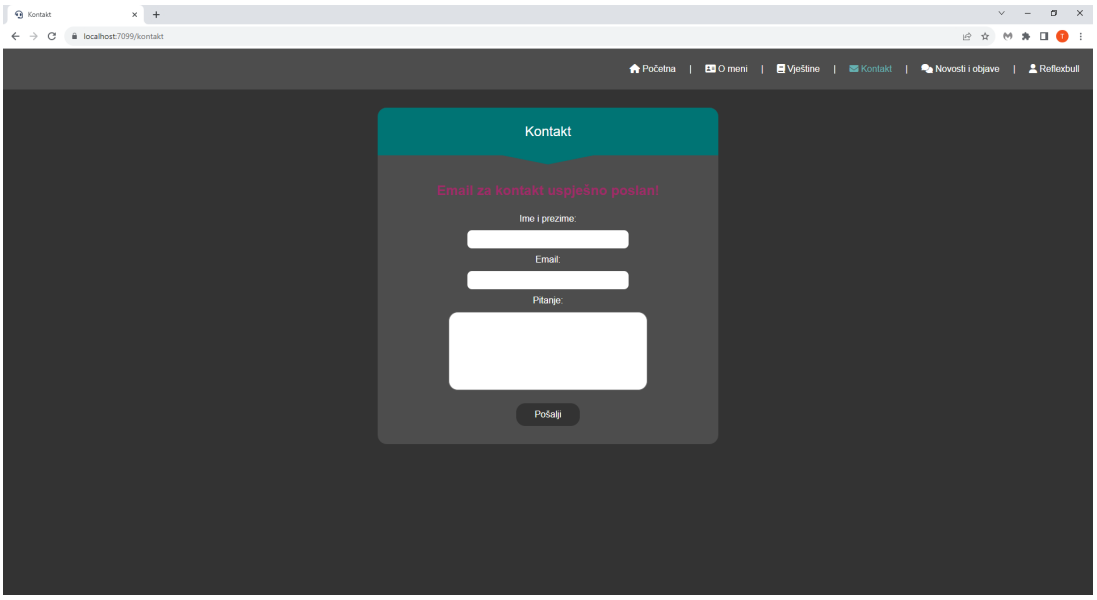
Slika 12: Forma s praznim poljima



Slika 13: Forma s nevažećim poljima



Slika 14: Forma koja je u procesu slanja



Slika 15: Forma koja je uspješno poslana

Na slikama se mogu vidjeti četiri primjera kontakt forme:

- forma s praznim poljima (vidljivo na 12)
- forma s nevažećim poljima (vidljivo na 13)
- forma koja je u procesu slanja (vidljivo na 14)
- forma koja je uspješno poslana (vidljivo na 15)

Forma sama po sebi ima određene validacije (vidljivo na 4.8) koje korisnik mora ispuniti kako bi kontakt e-mail bio uspješno poslan.

Isječak koda 4.8: Validacija

```
public class Email
{
    [Required(ErrorMessage = "Niste␣upisali␣Vase␣ime␣i␣prezime!")]
    [MinLength(6, ErrorMessage = "Vase␣ime␣i␣prezime␣mora␣imati␣najmanje␣4␣znaka!")]
    public string FullName { get; set; } = string.Empty;


    [Required(ErrorMessage = "Niste␣upisali␣Vas␣email!")]
    [RegularExpression(@"^\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*$",
        ErrorMessage = "Nevazeci␣format␣emaila!")]
    public string Email_ { get; set; } = string.Empty;


    [Required(ErrorMessage = "Niste␣napisali␣Vase␣pitanje!")]
    [RegularExpression(@"^.{20,}$", ErrorMessage = "Vase␣pitanje␣mora␣imati␣najmanje
        ␣20␣znakova!")]
    public string Question { get; set; } = string.Empty;
}
```

Validacije uz pomoć regularnog izraza (engl. *RegEx*) [2] na prikazanom modelu određuju o stanju u kojem će se forma nalaziti.

Isječak koda 4.9: Kontakt forma - elementi

```
<EditForm Model="email" OnValidSubmit="SendEmail">
    <DataAnnotationsValidator />

    <div id="full-height-div" class="container">
        <div class="contact-box">
            <div class="contact-box-header">
                Kontakt
            </div>

            <div class="triangle-down" />

            <div class="contact-box-body">
                <div class="contact-box-body-input-fields">
                    @* Contact Mail Error Box *@
                    <div style="font-size:␣1.5rem;" class="error-box">
                        <div>
                            @Message
                        </div>
                    </div>

                    @* Full Name *@
                    <div>
                        Ime i prezime:
                    </div>
```

[2] Regularni izraz (skraćeno kao regex) niz je znakova koji navodi uzorak podudaranja u tekstu. Obično se takvi uzorci koriste u algoritmima pretraživanja nizova za operacije "pronađi" ili "pronađi i zamijeni" na nizovima ili za provjeru valjanosti unosa. [80]

```
                    <input type="text" spellcheck="false" @bind-value="email.
                        FullName" />
                    <ValidationMessage For="@(() => email.FullName)" />

                    @* Email *@
                    <div>
                        Email:
                    </div>

                    <input type="text" spellcheck="false" @bind-value="email.Email_"
                         />
                    <ValidationMessage For="@(() => email.Email_)" />

                    @* Question *@
                    <div>
                        Pitanje:
                    </div>

                    <textarea rows="5" maxlength="500" spellcheck="false" @bind="
                        email.Question" />
                    <ValidationMessage For="@(() => email.Question)" />
                </div>

                @if (IsLoading)
                {
                    <LoaderCircle />
                }
                else
                {
                    <div class="button-default-portfolio">
                        <button type="submit">
                            Posalji
                        </button>
                    </div>
                }
            </div>
        </div>
    </div>
</EditForm>
```

Ovaj isječak programskog koda (vidljivo na 4.9) prikazuje formu zatvorenu unutar <Edit-Form> komponente koja se mora koristiti zajedno sa komponentom RegularExpression kako bi validacije pravilno radile, u suprotnom će forma biti važeća (engl. *valid*) pri svakom kliku na gumb za potvrdu (engl. *submit*) forme.

Pritiskom na gumb za potvrdu forme, okinut će se EventCallback [3] - OnValidSubmit koji će pozvati funkciju SendEmail.

Isječak koda 4.10: Slanje e-maila vlasniku stranice

```
private async Task SendEmail()
```

[3] EventCallback je posebna klasa u Blazor-u koja se može prikazati kao parametar, kako bi komponente mogle obaviti neke funkcije kada se dogodi neki slučaj. [81]

```
{
    IsLoading = true;
    Message = string.Empty;

    if (editContext.Validate())
    {
        ErrorsDetected = false;
        Message = await EmailSenderService.SendContactEmailToWebPageDeveloper(email)
            ;
        email = new Email();
        IsLoading = false;
        StateHasChanged();

        Message = await Constants.ClearSuccessMessage();
    }
    else
    {
        ErrorsDetected = true;
        Message = "Neuspjesno␣slanje␣maila!";
    }

    IsLoading = false;
}
```

U slučaju da je forma važeća, odnosno da je zadovoljila sve potrebne validacije, SendEmail funkcija će poslati e-mail vlasniku web-stranice (vidljivo na 16) te će korisnika propisno obavijestiti o uspjehu, a ako forma nije važeća, korisnik će jednostavno dobiti povratnu poruku o neuspjehu (vidljivo na 4.10, 4.11).

Za slanje e-maila je napravljen poseban servis, IEmailSenderService, koji će biti korišten u procesu slanja e-mailova.

Isječak koda 4.11: Servis za slanje kontakt e-maila

```
public async Task<string> SendContactEmailToWebPageDeveloper(Email contactEmail)
{
    MimeMessage email = new MimeMessage();
    email.From.Add(MailboxAddress.Parse(_configuration.GetSection("GmailEmail").
        Value));
    email.To.Add(MailboxAddress.Parse(_configuration.GetSection("GmailEmail").Value)
        );
    email.Subject = "Kontakt␣sa␣Portfolio␣stranice";
    email.Body = new TextPart(TextFormat.Html)
    {
        Text = await ConvertContactDataToHtmlTemplate(contactEmail)
    };

    using (SmtpClient smtp = new SmtpClient())
    {
        smtp.Connect(_configuration.GetSection("GmailHost").Value, int.Parse(
            _configuration.GetSection("GmailPort").Value), SecureSocketOptions.
            StartTls);
        smtp.Authenticate(_configuration.GetSection("GmailEmail").Value,
```

```
            _configuration.GetSection("GmailPassword").Value);
        smtp.Send(email);
        smtp.Disconnect(true);
    }

    return "Email za kontakt uspjesno poslan!";
}
```

Logika slanja e-maila (vidljivo na 4.11):

- From - e-mail pošiljatelja
- To - e-mail primatelja
- smtp.Connect - spajanje na domaćina e-mail servisa, uz određeni port
- smtp.Authenticate - autentifikacija osobe koja ima korisnički račun na domaćinu e-mail servisa, s čijeg će se korisničkog računa slati e-mailovi

Isječak koda 4.12: Zamjena HTML elemenata u predlošku sa stvarnim vrijednostima

```
public async Task<string> ConvertContactDataToHtmlTemplate(Email contactEmail)
{
    string template = await (new FileInfo(Constants.Root + "/wwwroot/Templates/
        Account/ContactEmailTemplate.html")).OpenText().ReadToEndAsync();

    // Body
    template = template.Replace("@{NAME}", contactEmail.FullName);
    template = template.Replace("@{EMAIL}", contactEmail.Email_);
    template = template.Replace("@{QUESTION}", contactEmail.Question);

    return template;
}
```

Isječak koda 4.13: HTML predložak

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />

        <style>
            body {
                margin: 0;
            }

            .wrapper {
                background-color: whitesmoke;
                padding: 2em;
            }

            .content-wrapper {
```

```
    width: 50%;
    border: 1px solid #eee;
    border-radius: 1em;
    box-shadow: 0 0 1em rgba(0, 0, 0, 0.15);
    max-width: 900px;
    margin: auto;
}

.header {
    padding: 1em;
    background-color: #007474; /* Constants.ColorTealDark */
    color: white;
    text-align: center;
    margin: 0;
}

.triangle-down {
    width: 0;
    height: 0;
    border-left: 5em solid transparent;
    border-right: 5em solid transparent;
    border-top: 1em solid #007474; /* Constants.ColorTealDark */
    margin: auto;
}

.content {
    background-color: #333333; /* Constants.ColorGrayFullDark */
    color: white;
    padding: 2em;
    padding-top: 0;
    text-align: center;
}

    .content > div {
        padding: 0.5em 0;
    }

        .content div > div:first-child {
            color: #9F2B68 !important; /* Constants.ColorPinkDark */
        }

        .content div > div:nth-child(2) {
            color: white;
        }

div.div-width-contact {
    width: 60%;
    margin: auto;
}

@media only screen and (max-width: 600px) {
    .content-wrapper {
        width: 100%;
```

```
            }
        }
    </style>

    <title>
        Kontakt sa Portfolio stranice
    </title>
</head>

<body>
    <div class="wrapper">
        <div class="content-wrapper">
            <h3 class="header">
                Kontakt sa Portfolio stranice
            </h3>

            <main>
                <div class="content">
                    <div class="triangle-down" />

                    <div>
                        <div>
                            Korisnik:
                        </div>

                        <div class="div-width-contact">
                            @{NAME}
                        </div>
                    </div>

                    <div>
                        <div>
                            Email korisnika:
                        </div>

                        <div class="div-width-contact">
                            @{EMAIL}
                        </div>
                    </div>

                    <div>
                        <div>
                            Pitanje:
                        </div>

                        <div class="div-width-contact">
                            @{QUESTION}
                        </div>
                    </div>
                </div>
            </main>
        </div>
    </div>
```

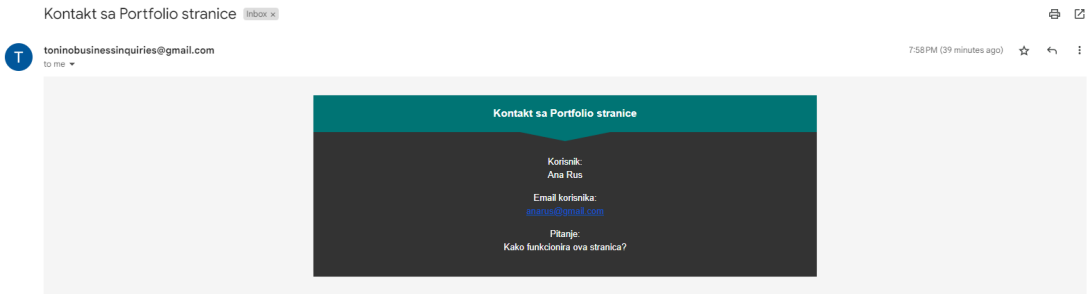
```
    </body>
</html>
```

Prije nego se pošalje e-mail, prvo se konvertira HTML predložak koji će se koristiti za stiliziranost emaila koji će biti vidljiv na nekom od e-mail servisa prilikom otvaranja (vidljivo na 4.12, 4.13).

Isječak koda 4.14: Primjer autentifikacije za e-mail domaćina

```
"GmailHost": "smtp.gmail.com",
"GmailPort": "587",
```

_configuration.GetSection("GmailHost").Value označava sekciju koja se dohvaća iz postavki aplikacije koje se nalaze u datoteci **appsettings.json**, dok Value vraća vrijednost tog parametra (smtp.gmail.com) (vidljivo na 4.13).



Slika 16: Primjer uspješno poslanog maila za kontakt

4.2.6. Prijava

Stranica prijava je jedna, glavna (engl. *parent*) komponenta koja sadrži tri podkomponente (vidljivo na 4.15), tzv. komponente djecu (engl. *child components*).

Isječak koda 4.15: Login.razor komponenta

```
@page "/prijava"
@using Portfolio.Shared.Components

<PageTitle>
    Prijava
</PageTitle>

@if (CurrentComponent == Component.Login)
{
    <LoginComponent OnForgottenPasswordButton="ShowForgottenPasswordComponent"
        OnRegisterButton="ShowRegisterComponent" />
}
else if (CurrentComponent == Component.ForgottenPassword)
{
    <ForgottenPasswordComponent OnLoginButton="ShowLoginComponent" />
}
else if (CurrentComponent == Component.Register)
{
```

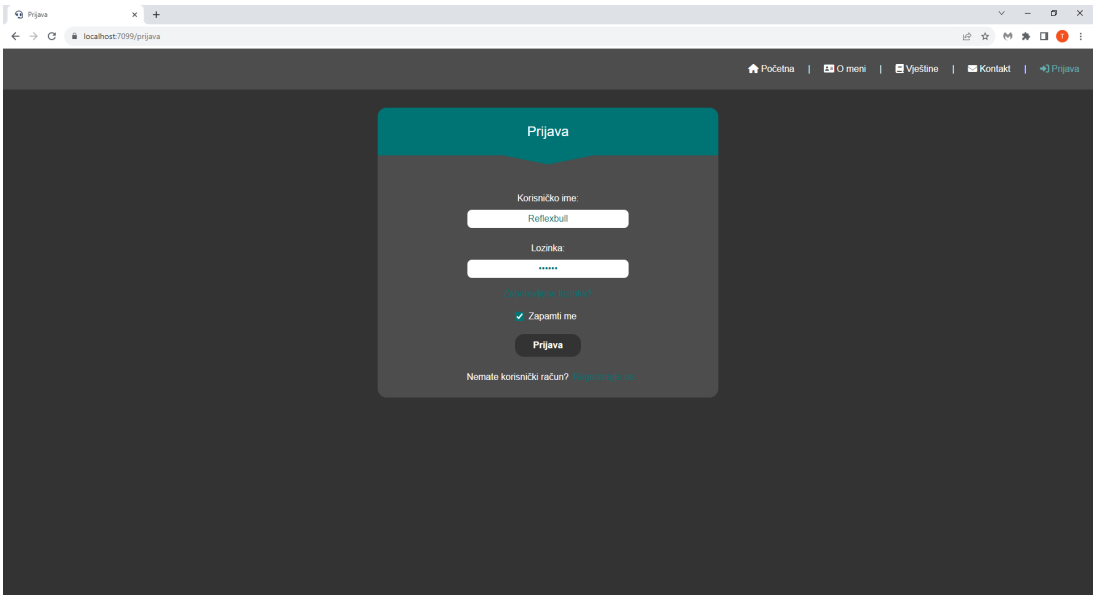
```
    <RegisterComponent OnLoginButton="ShowLoginComponent" />
}
```

Isječak koda 4.16: Login.razor.cs komponenta

```
private Component CurrentComponent { get; set; }
private enum Component
{
    Login,
    Register,
    ForgottenPassword
}

private void ShowLoginComponent()
{
    CurrentComponent = Component.Login;
}

private void ShowRegisterComponent()
{
    CurrentComponent = Component.Register;
}

private void ShowForgottenPasswordComponent()
{
    CurrentComponent = Component.ForgottenPassword;
}
```

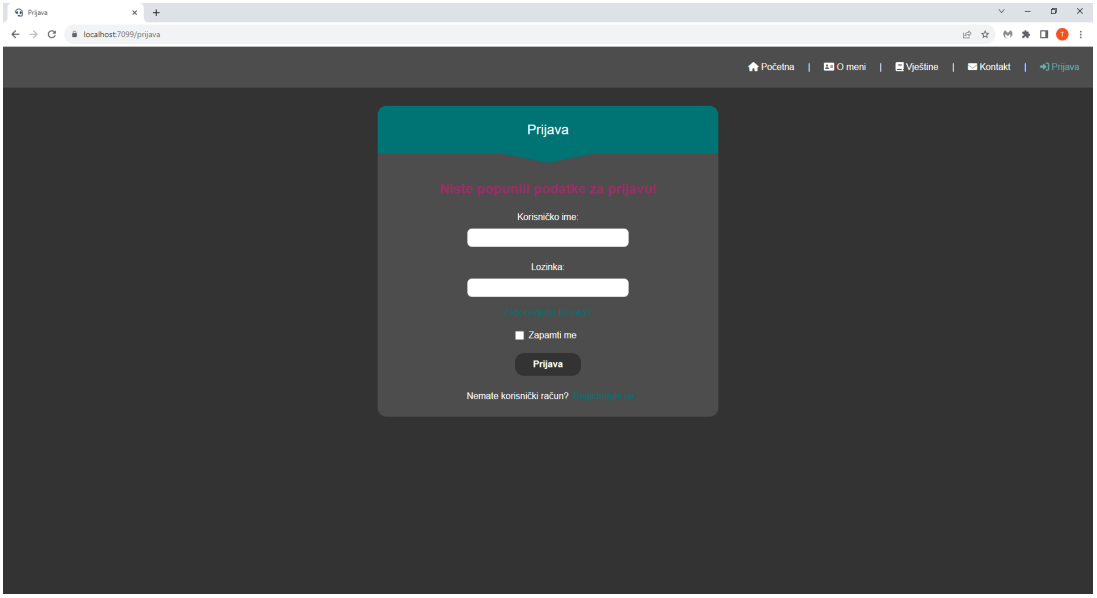
LoginComponent, ForgottenPasswordComponent i RegisterComponent su zasebne komponente koje imaju vlastiti kod, a koje se mogu pozivati gdje god je to potrebno (u ovom slučaju, samo na stranici prijava). Ovisno u kojoj se komponenti korisnik nalazi od navedene tri, okinut će se zaseban EventCallback parametar koji je definiran u njima. Ako npr. korisnik stisne "registrirajte se", unutar LoginComponent.razor.cs komponente će se opaliti EventCallback OnRegisterButton, nakon što se on okine, Login.razor komponenta će dobiti signal i pozvat će funkciju ShowRegisterComponent koja će enum CurrentComponent prebaciti na enum Register i tako će se prikazati komponenta za registraciju (vidljivo na 4.16).

4.2.6.1. Prijava

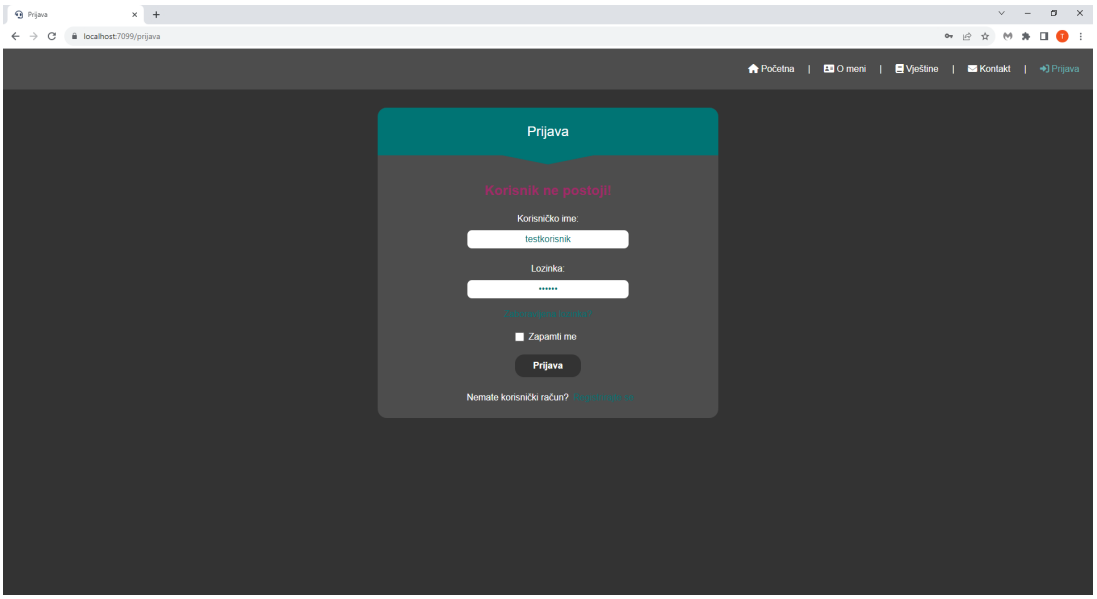


Slika 17: Forma za prijavu

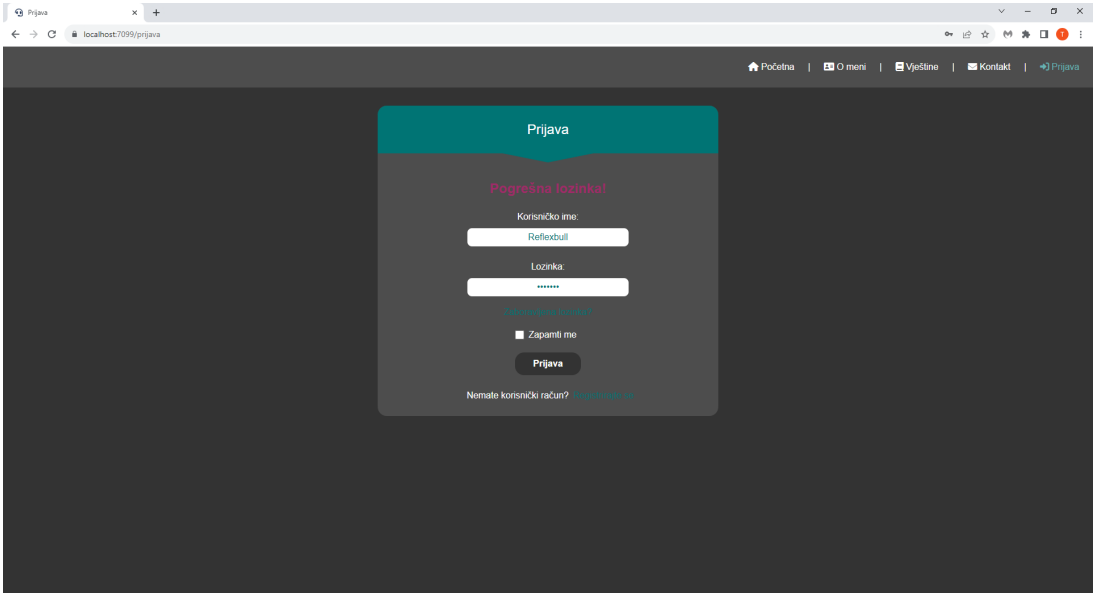
Pri inicijalnom učitavanju stranice, prikazuje se forma za prijavu gdje se korisnik (ako je registriran) može prijaviti (vidljivo na 17).



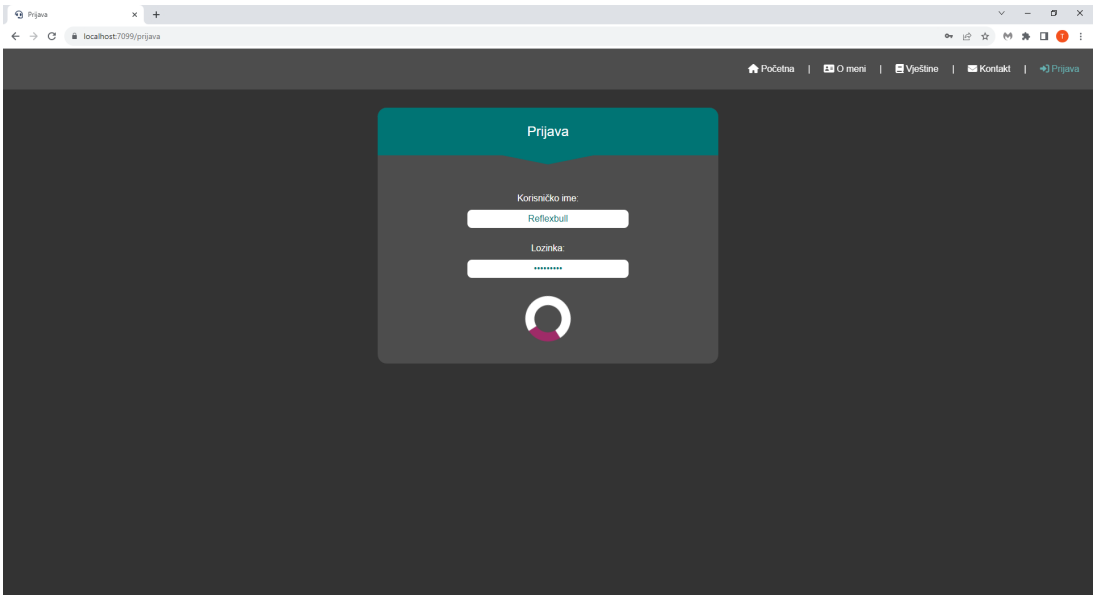
Slika 18: Forma s praznim poljima



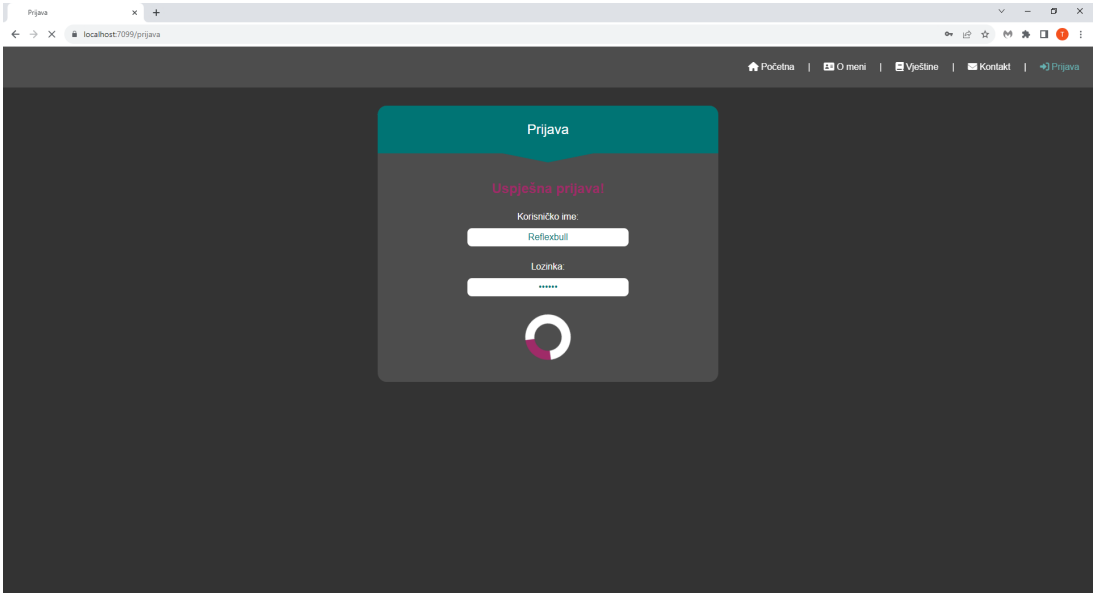
Slika 19: Korisnik nije pronađen



Slika 20: Pogrešna lozinka



Slika 21: Forma koja je u procesu slanja



Slika 22: Forma koja je uspješno poslana

Na slikama se može vidjeti pet primjera forme za prijavu:

- forma s praznim poljima (vidljivo na 18)
- korisnik nije pronađen (vidljivo na 19)
- pogrešna lozinka (vidljivo na 20)
- forma koja je u procesu slanja (vidljivo na 21)
- forma koja je uspješno poslana (vidljivo na 22)

Isječak koda 4.17: Prijava na komponenti

```
private async Task Login()
{
    if (ShouldAttemptLogin && CheckInput())
    {
        if (Checked)
        {
            RememberMe = true;
            await UserSessionService.SaveUserDataToDatabase(JSRuntime, Username,
                Password, true);
        }
        else
        {
            RememberMe = false;
            await UserSessionService.SaveUserDataToDatabase(JSRuntime, string.Empty,
                 string.Empty, false);
        }

        Tuple<bool, string> loginUserResult = await UserService.LoginUser(Username,
            Password, RememberMe);

        if (loginUserResult.Item1)
        {
            ErrorsDetected = false;
            Message = loginUserResult.Item2;
            StateHasChanged();

            if (UserService.CurrentUser != null)
            {
                await JSRuntime.InvokeVoidAsync("localStorage.setItem", "username",
                    $"{Username}");
                NavigationManager.NavigateTo("/", forceLoad: true);
            }
        }
        else
        {
            ErrorsDetected = true;
            Message = loginUserResult.Item2;
            IsLoading = false;
        }
    }
    else
    {
        ErrorsDetected = true;
        Message = "Niste␣popunili␣podatke␣za␣prijavu!";
        IsLoading = false;
    }
}
```

Isječak koda 4.18: Spremanje korisničkog imena u lokalnu pohranu web-preglednika

```
public async Task SaveUserDataToDatabase(IJSRuntime jSRuntime, string username,
    string password, bool rememberMe)
```

```
{
    string userSessionID = await GetUserSessionIDFromLocalStorage(jSRuntime);

    UserSession userSession = new UserSession
    {
        Id = userSessionID,
        Username = username,
        Password = password,
        RememberMe = rememberMe
    };

    await userSessions.ReplaceOneAsync(us => us.Id == userSessionID, userSession);
}

public async Task<string> GetUserSessionIDFromLocalStorage(IJSRuntime jSRuntime)
{
    return await jSRuntime.InvokeAsync<string>("localStorage.getItem", "
        userSessionID");
}
```

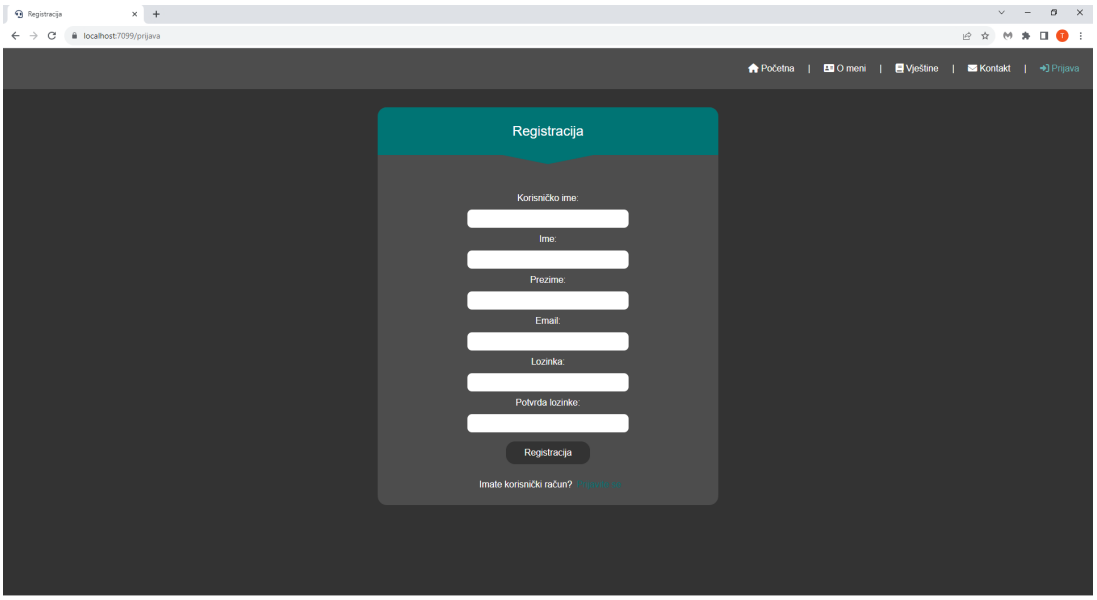
Isječak koda 4.19: Prijava korisnika - servis

```
public async Task<Tuple<bool, string>> LoginUser(string username, string password,
    bool rememberMe)
{
    bool correctCredentials = false;
    string message = string.Empty;
    User user = await users.Find(u => u.Username == username).FirstOrDefaultAsync();

    if (user == null)
    {
        correctCredentials = false;
        message = "Korisnik␣ne␣postoji!";
    }
    else if (user != null && user.Password != password)
    {
        correctCredentials = false;
        message = "šPogrena␣lozinka!";
    }
    else if (user != null && user.Password == password)
    {
        user.RememberMe = rememberMe;
        await users.ReplaceOneAsync(u => u.Username == username, user);

        await UpdateUserSessionStatus(true, user);
        correctCredentials = true;
        message = "šUspjena␣prijava!";
    }

    return Tuple.Create(correctCredentials, message);
}
```

Nakon što je forma za prijavu uspješno ispunjena (vidljivo na 4.17), ako je korisnik

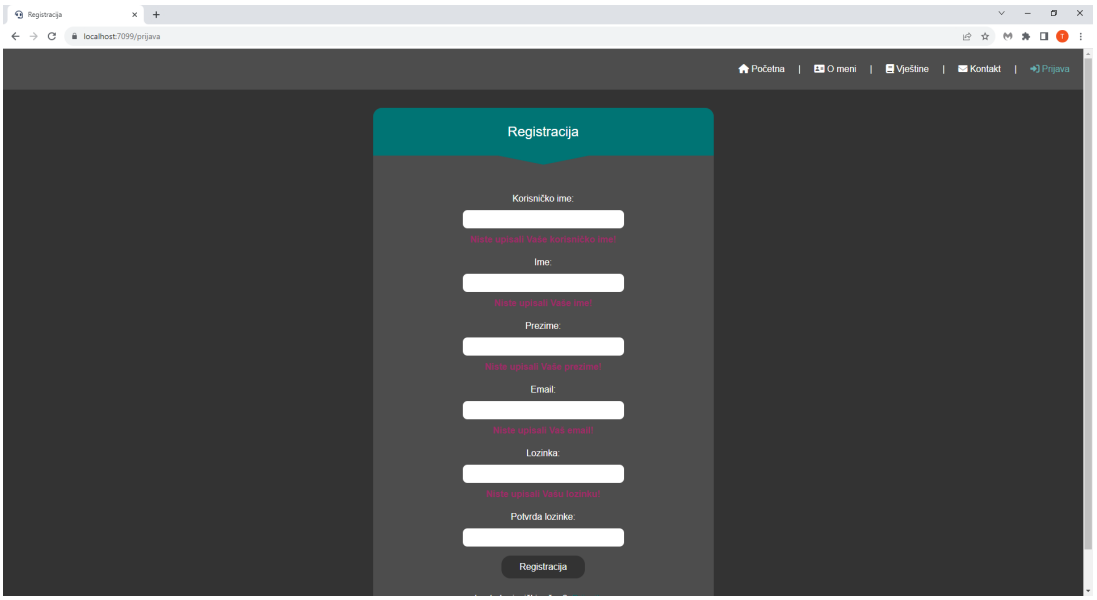
označio da želi da njegovi podatci budu zapamćeni pri sljedećoj prijavi, njegovo korisničko ime će se zapisati u lokalnu pohranu na web-pregledniku (vidljivo na 4.18), početak njegove sesije se postavlja na DateTime.Now odnosno današnji datum i vrijeme sad, kraj njegove sesije se postavlja na jedan mjesec od sad (znači ako se korisnik ne izlogira sam mjesec dana, prilikom sljedećeg posjeta stranici nakon mjesec dana će se morati ponovno prijaviti), te se korisnik uspješno prijavljuje (vidljivo na 4.19) i dobiva pristup ostalim značajkama aplikacije.

4.2.6.2. Registracija

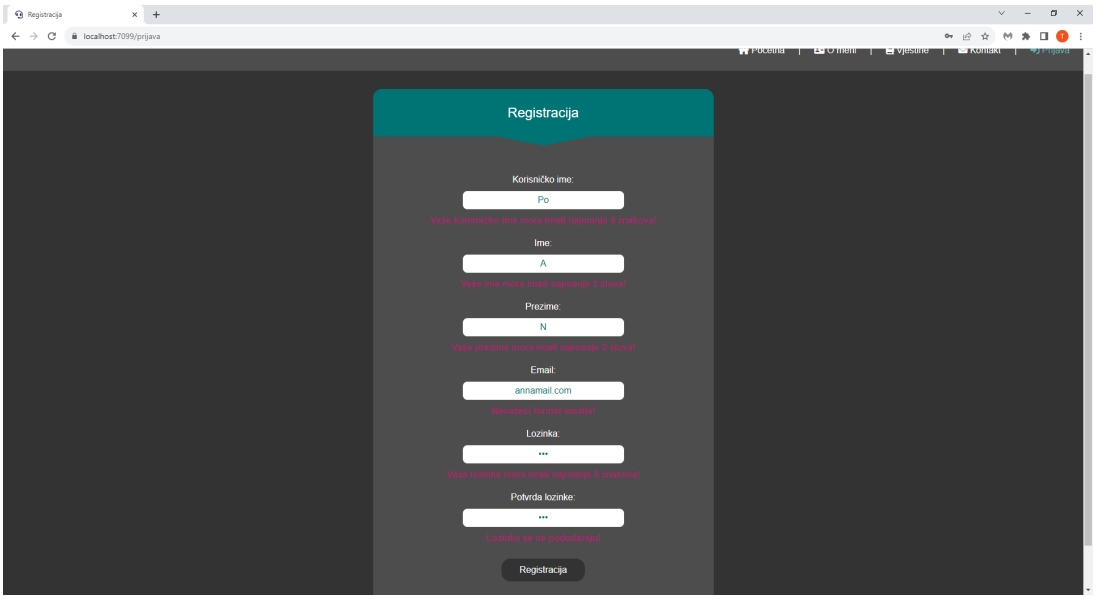


Slika 23: Forma za registraciju

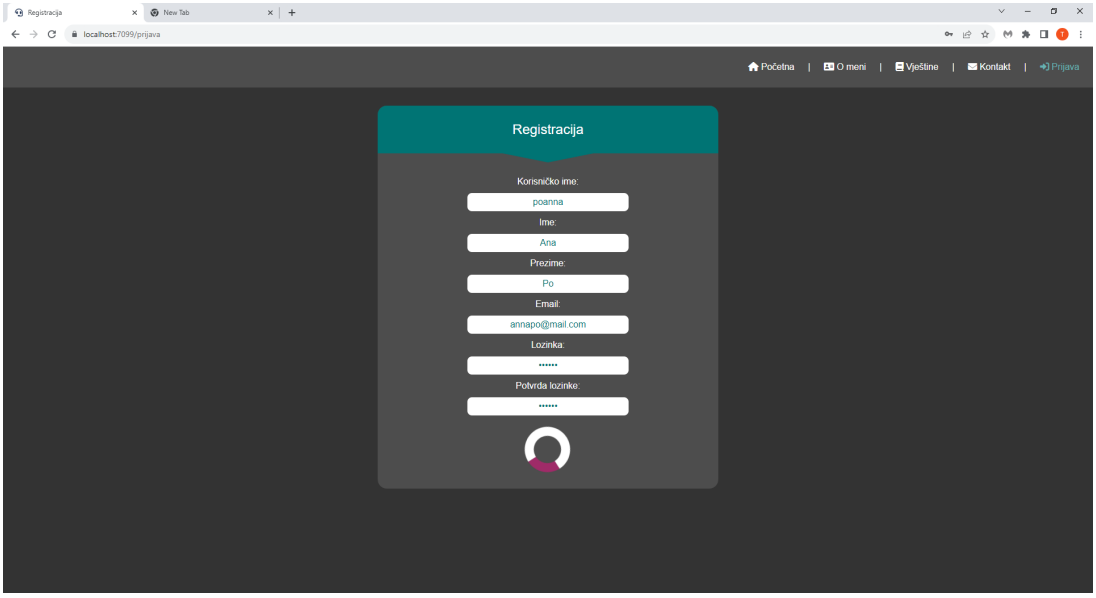
Pri inicijalnom učitavanju stranice, prikazuje se forma za registraciju gdje se korisnik može registrirati (vidljivo na 23).



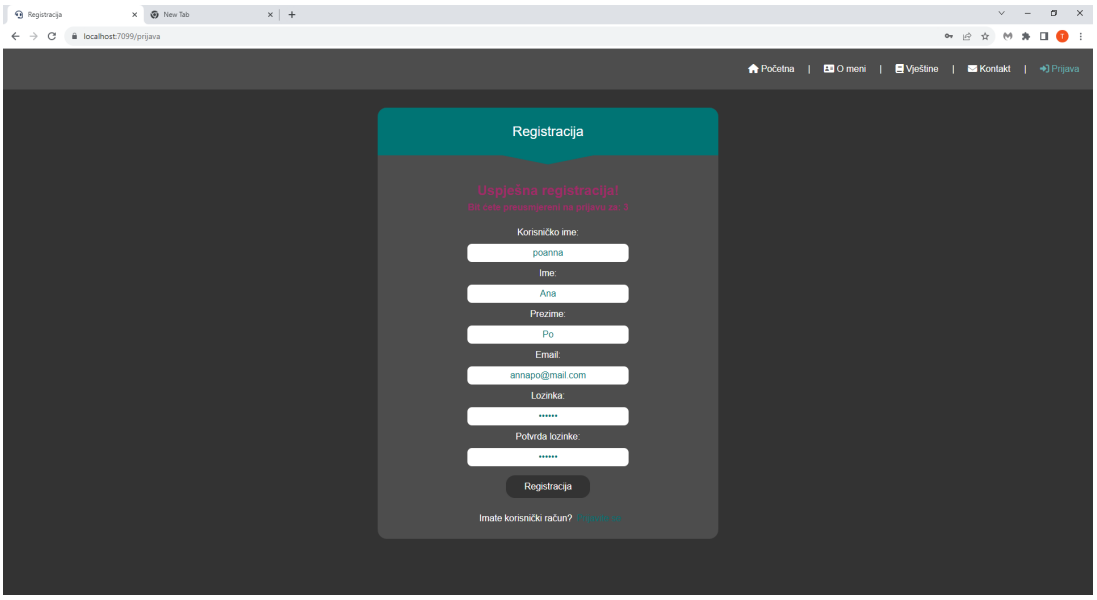
Slika 24: Forma s praznim poljima



Slika 25: Forma s nevažećim poljima



Slika 26: Forma koja je u procesu slanja



Slika 27: Forma koja je uspješno poslana

Na slikama se mogu vidjeti četiri primjera forme za registraciju:

- forma s praznim poljima (vidljivo na 24)
- forma s nevažećim poljima (vidljivo na 25)
- forma koja je u procesu slanja (vidljivo na 26)
- forma koja je uspješno poslana (vidljivo na 27)

Isječak koda 4.20: Validacije kod registracije korisnika

```
[BsonRepresentation(BsonType.ObjectId)]
public string Id { get; set; } = ObjectId.GenerateNewId().Tostring();


[Required(ErrorMessage = "Niste␣upisali␣Vase␣korisnicko␣ime!")]
[MinLength(6, ErrorMessage = "Vase␣korisnicko␣ime␣mora␣imati␣najmanje␣6␣znakova!")]
[MaxLength(20, ErrorMessage = "Vase␣korisnicko␣ime␣moze␣imati␣najvise␣20␣znakova!")]
[RegularExpression(@"^\S*$", ErrorMessage = "Korisnicko␣ime␣ne␣smije␣sadrzavati␣
    razmake!")]
public string Username { get; set; } = string.Empty;


[Required(ErrorMessage = "Niste␣upisali␣Vase␣ime!")]
[MinLength(2, ErrorMessage = "Vase␣ime␣mora␣imati␣najmanje␣2␣slova!")]
[MaxLength(20, ErrorMessage = "Vase␣ime␣moze␣imati␣najvise␣20␣slova!")]
[RegularExpression(@"^\S*$", ErrorMessage = "Ime␣ne␣smije␣sadrzavati␣razmake!")]
public string FirstName { get; set; } = string.Empty;


[Required(ErrorMessage = "Niste␣upisali␣Vase␣prezime!")]
[MinLength(2, ErrorMessage = "Vase␣prezime␣mora␣imati␣najmanje␣2␣slova!")]
```

```
[MaxLength(30, ErrorMessage = "Vase␣prezime␣moze␣imati␣najvise␣30␣slova!")]
[RegularExpression(@"^\S*$", ErrorMessage = "Prezime␣ne␣smije␣sadrzavati␣razmake!")]
public string LastName { get; set; } = string.Empty;


[Required(ErrorMessage = "Niste␣upisali␣Vas␣email!")]
[RegularExpression(@"^\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*$", ErrorMessage =
    "Nevazeci␣format␣emaila!")]
public string Email { get; set; } = string.Empty;


[Required(ErrorMessage = "Niste␣upisali␣Vasu␣lozinku!")]
[MinLength(6, ErrorMessage = "Vasa␣lozinka␣mora␣imati␣najmanje␣6␣znakova!")]
[RegularExpression(@"^\S*$", ErrorMessage = "Lozinka␣ne␣smije␣sadrzavati␣razmake!")]
public string Password { get; set; } = string.Empty;
```

Kako bi forma bila uspješna, mora proći zadane validacije (vidljivo na 4.20).

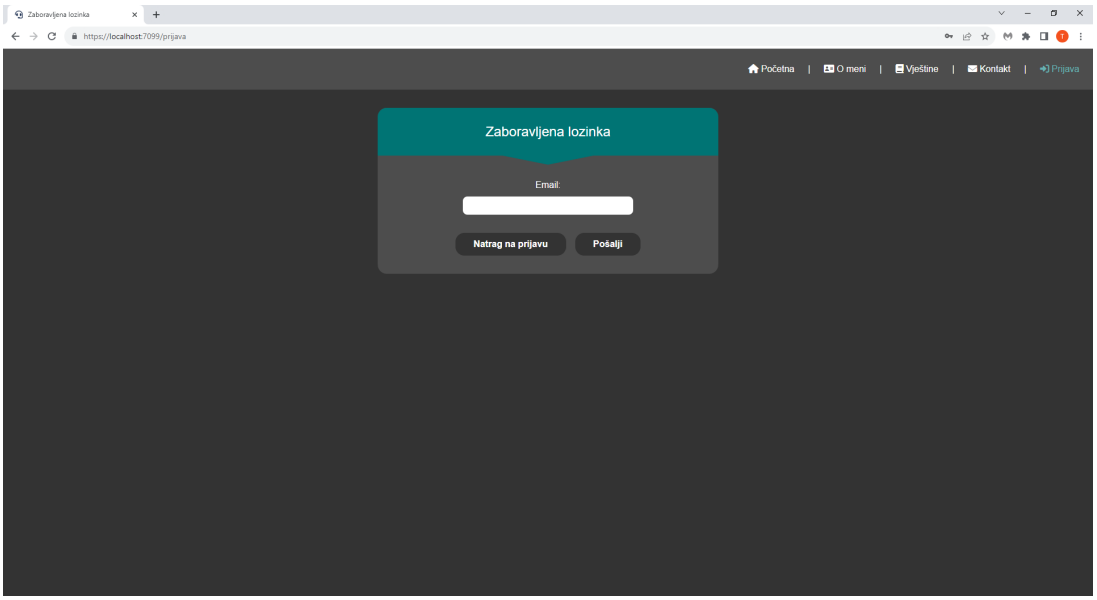
Isječak koda 4.21: Registracija korisnika

```
private async Task Register()
{
    CheckPasswordConfirmed();
    if (!PasswordConfirmed)
    {
        return;
    }

    TrimInput();

    if (EditContext.Validate() && PasswordConfirmed)
    {
        User findUserByUsername = await UserService.GetUserByUsername(NewUser.
            Username);
        if (findUserByUsername != null)
        {
            Message = "Korisnik␣sa␣odabranim␣čkorisnikim␣imenom␣će␣postoji!";
            return;
        }

        User findUserByEmail = await UserService.GetUserByEmail(NewUser.Email);
        if (findUserByEmail != null)
        {
            Message = "Korisnik␣sa␣odabranim␣emailom␣će␣postoji!";
            return;
        }

        string userSessionID = await UserSessionService.
            GetUserSessionIDFromLocalStorage(JSRuntime);
        NewUser.UserSession_Id = userSessionID;

        SuccessMessage = await UserService.RegisterUser(NewUser);
        await EmailSenderService.SendNewUserRegisteredEmailToUser(NewUser.Username,
            NewUser.Email);
```

```
            await EmailSenderService.SendNewUserRegisteredEmailToWebPageDeveloper(
                NewUser.Username, NewUser.Email);
            string userRegisteredMessage = "Bit␣ćete␣preusmjereni␣na␣prijavu␣za:";
            for (int i = 5; i > 0; i--)
            {
                Message = string.Empty;
                Message = $"{userRegisteredMessage}␣{i}";
                StateHasChanged();
                await Task.Delay(1000);
            }

            await OnLoginButtonClick();
        }
    }
```

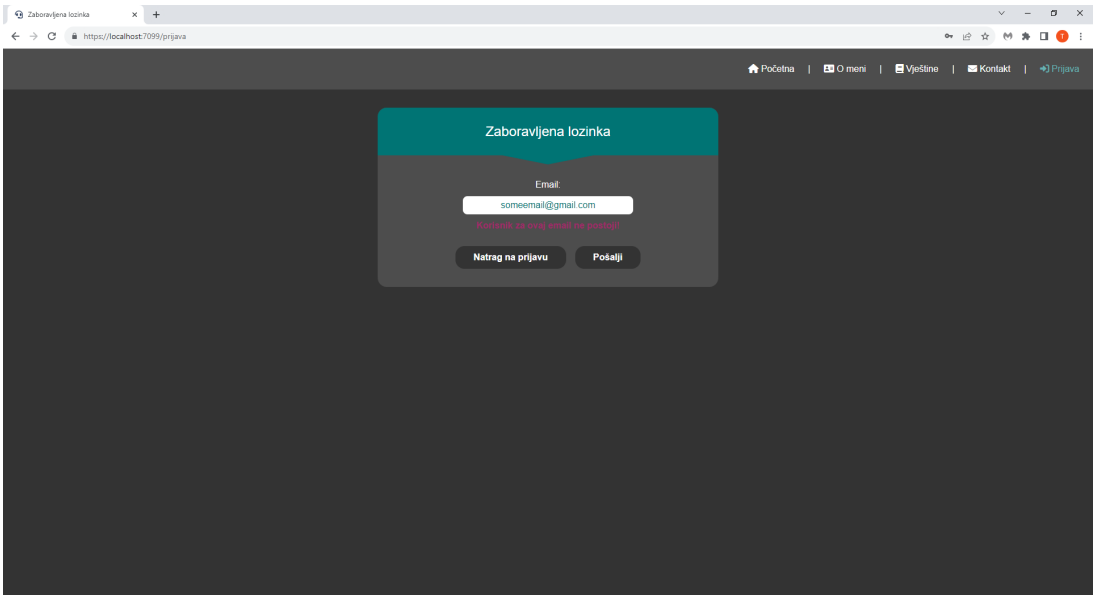
Ako upisani podatci valjaju te ako korisnik sa istim korisničkom imenom i/ili e-mailom ne postoji u bazi podataka, poziva se servis koji registrira korisnika u bazu podataka te servis koji šalje prikladne emailove za uspješnu registraciju kako vlasniku web-stranice, tako i samom korisniku koji se registrirao (vidljivo na 4.21).

4.2.6.3. Zaboravljena lozinka i promjena lozinke

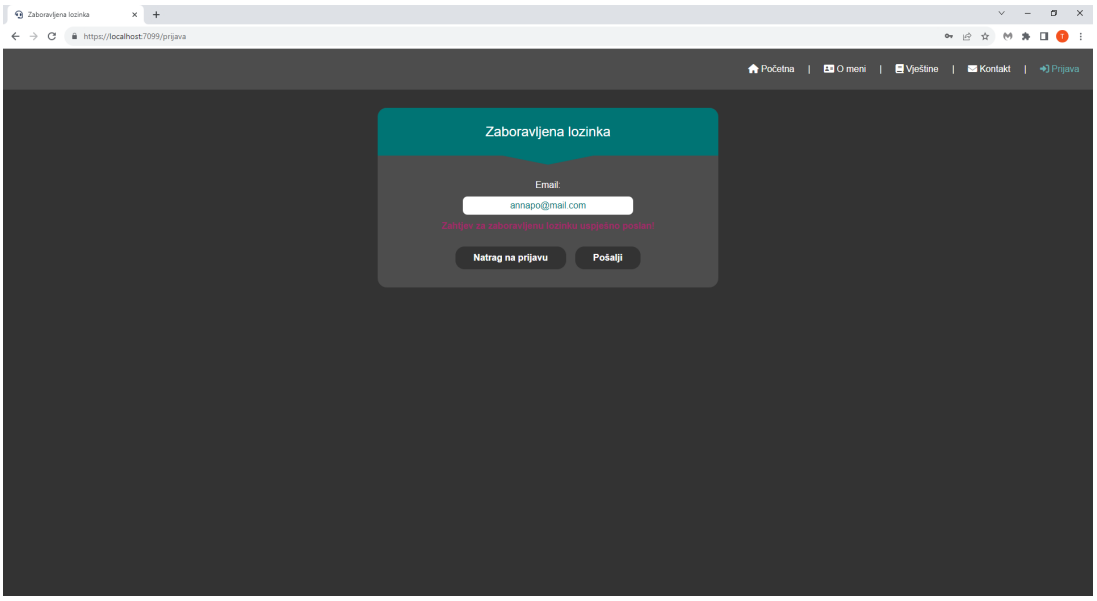


Slika 28: Forma za zaboravljenu lozinku

U slučaju da je korisnik zaboravio lozinku, a zna svoj e-mail, može ga upisati (vidljivo na 28) te tako na svoj e-mail dobiti upute za ponovno postavljanje zaboravljene lozinke.



Slika 29: Poruka za nepostojeći e-mail



Slika 30: Uspješno slanje zahtjeva za zaboravljenu lozinku

Na slikama se mogu vidjeti dva primjera forme za zaboravljenu lozinku:

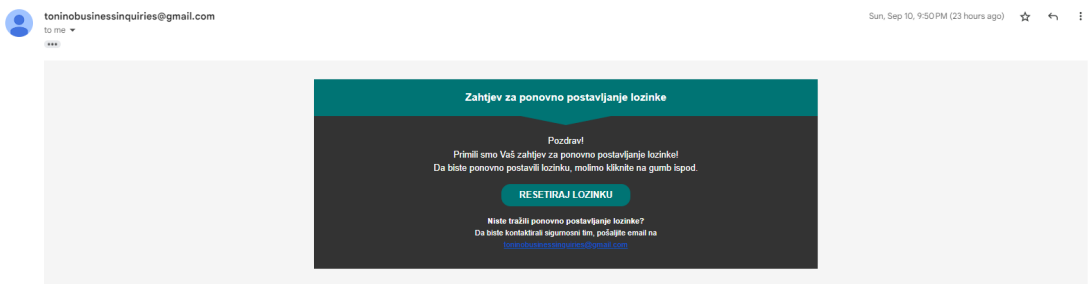
- poruka za nepostojeći e-mail (vidljivo na 29)
- uspješno slanje zahtjeva za zaboravljenu lozinku (vidljivo na 30)

Isječak koda 4.22: Slanje e-mail za zaboravljenu lozinku korisniku

```
private async Task SendEmail()
{
    IsLoading = true;
```

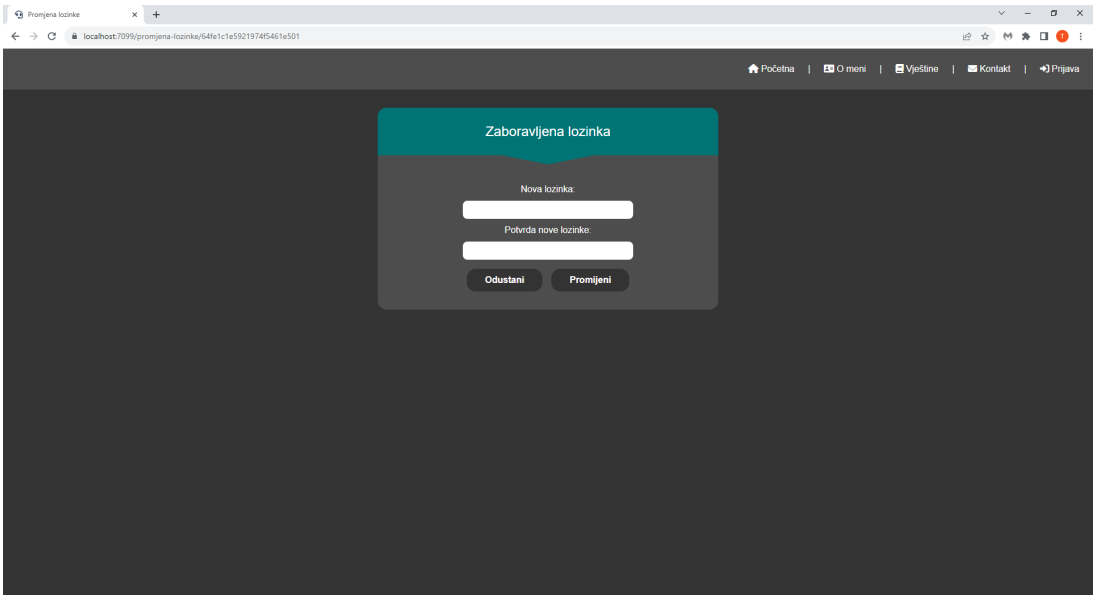
```
    User user = await UserService.GetUserByEmail(Email);

    if (user == null)
    {
        ErrorsDetected = true;
        ErrorMessage = "Korisnik za ovaj email ne postoji!";
    }
    else
    {
        ErrorsDetected = false;
        SuccessMessage = await EmailSenderService.SendForgottenPasswordEmailToUser(
            user);
    }

    IsLoading = false;
}
```

Kad korisnik upiše svoj e-mail, pozove se funkcija SendEmail (vidljivo na 4.22) koja prvo provjeri postoji li korisnik s takvim e-mailom u bazi podataka, ako postoji, pozvat će se servis za slanje e-maila koji će ponovno pretvoriti prikladan predložak te će poslati zahtjev za ponovno postavljanje lozinke na korisnikov e-mail (vidljivo na 31).

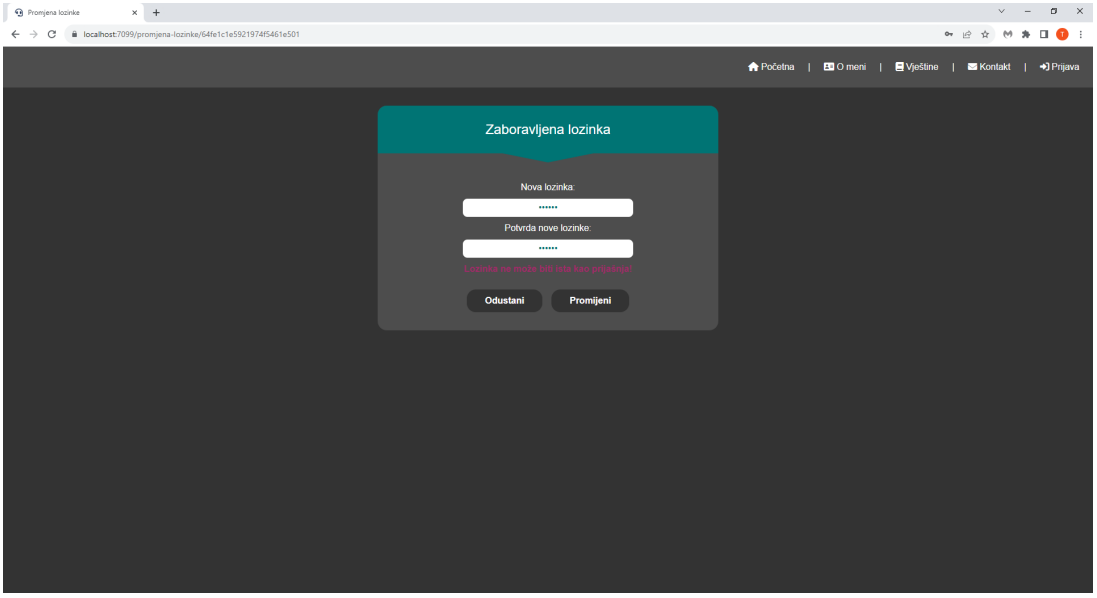


Slika 31: E-mail sa zahtjevom za ponovno postavljanje zaboravljene lozinke

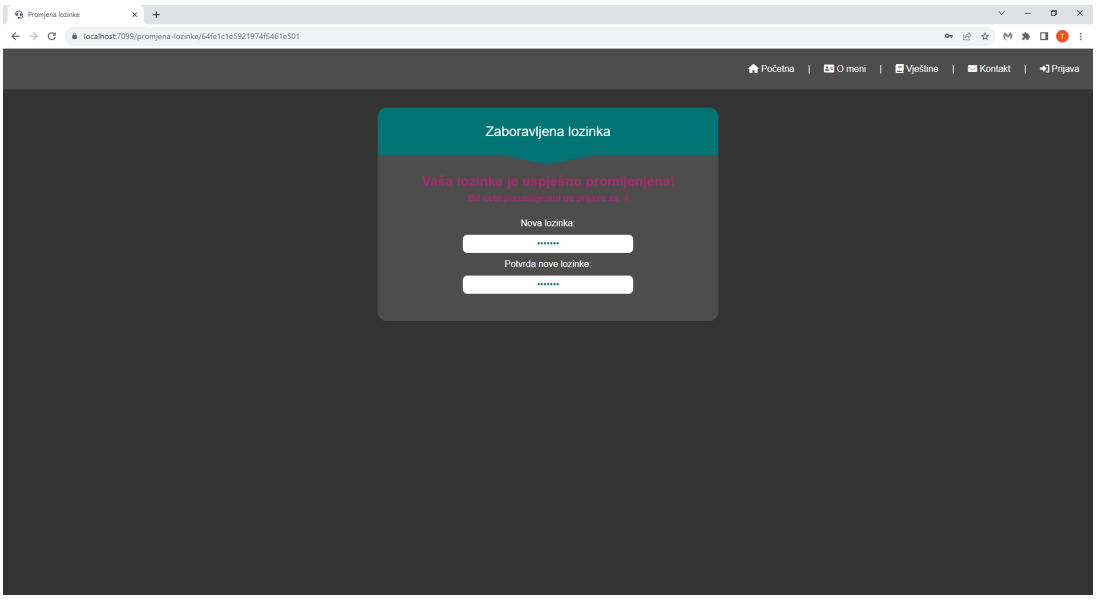
Nakon što korisnik stisne gumb resetiraj lozinku, dolazi na stranicu za promjenu zaboravljene lozinke gdje ponovno mora proći određene validacije.



Slika 32: Forma s praznim poljima



Slika 33: Prikaz kad je lozinka ista kao prijašnja



Slika 34: Uspješna promjena zaboravljene lozinke

Na slikama se mogu vidjeti tri primjera forme za promjenu zaboravljene lozinke:

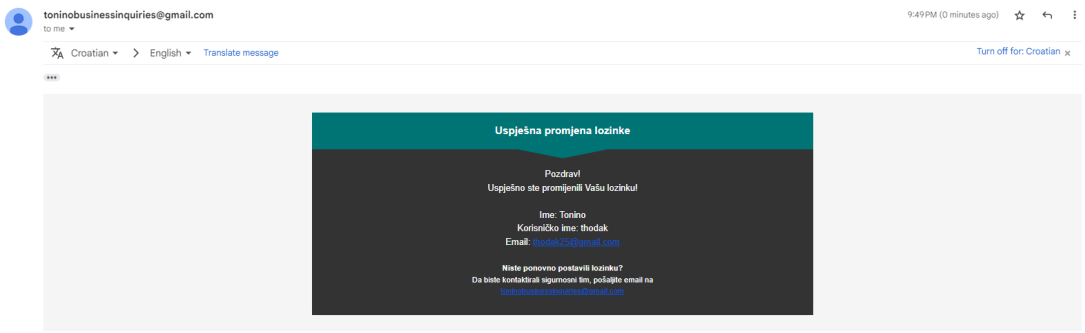
- forma s praznim poljima (vidljivo na 32)
- prikaz kad je lozinka ista kao prijašnja (vidljivo na 33)
- uspješna promjena zaboravljene lozinke (vidljivo na 34)

Isječak koda 4.23: Promjena lozinke

```
private async Task ConfirmPasswordChange()
{
    FirstPageLoad = false;

    CheckIfPasswordIsDifferentFromCurrent();
    if (!PasswordDifferent)
    {
        return;
    }

    CheckPasswordConfirmed();
    if (!PasswordConfirmed)
    {
        return;
    }

    HideButtons = true;
    IsLoading = true;

    if (EditContext.Validate() && PasswordConfirmed)
    {
        User.Password = NewPassword;
```

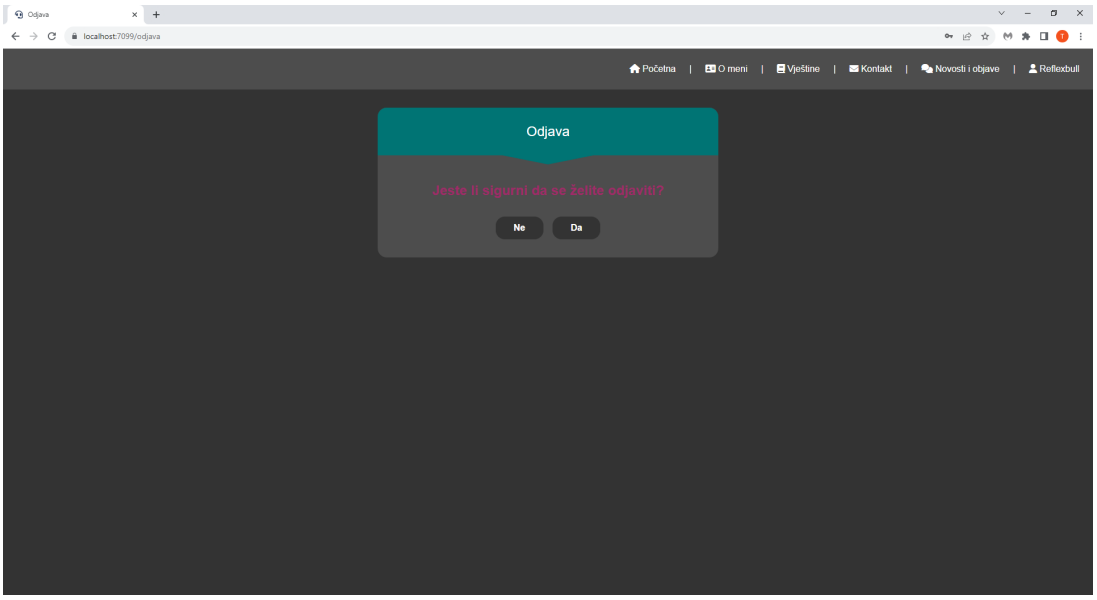
```
            await UserService.ChangePassword(User);
            await EmailSenderService.SendPasswordSuccessfullyChangedEmailToUser(User);
            IsLoading = false;

            SuccessMessage = "šVaa␣lozinka␣je␣šuspjeno␣promijenjena!";
            string userRegisteredMessage = "Bit␣ćete␣preusmjereni␣na␣prijavu␣za:";
            for (int i = 5; i > 0; i--)
            {
                Message = string.Empty;
                Message = $"{userRegisteredMessage}␣{i}";
                StateHasChanged();
                await Task.Delay(1000);
            }

            NavigationManager.NavigateTo("/prijava");
        }

        IsLoading = false;
    }
```

Nakon što korisnikov unos prođe zadane validacije (vidljivo na 4.23), njegova lozinka bude uspješno promijenjena te ga aplikacija odvede na stranicu za prijavu, nakon čega će ubrzo dobiti e-mail sa uspješnom promjenom lozinke (vidljivo na 35).



Slika 35: E-mail za uspješno postavljanje zaboravljene lozinke

4.2.6.4. Odjava



Slika 36: Odjava korisnika

Isječak koda 4.24: Odjava

```
private async Task Logout()
{
    if (ShouldAttemptLogout && UserService.CurrentUser != null)
    {
        Message = await UserService.LogoutUser(JSRuntime);
        StateHasChanged();
        NavigationManager.NavigateTo("/", forceLoad: true);
    }
}
```

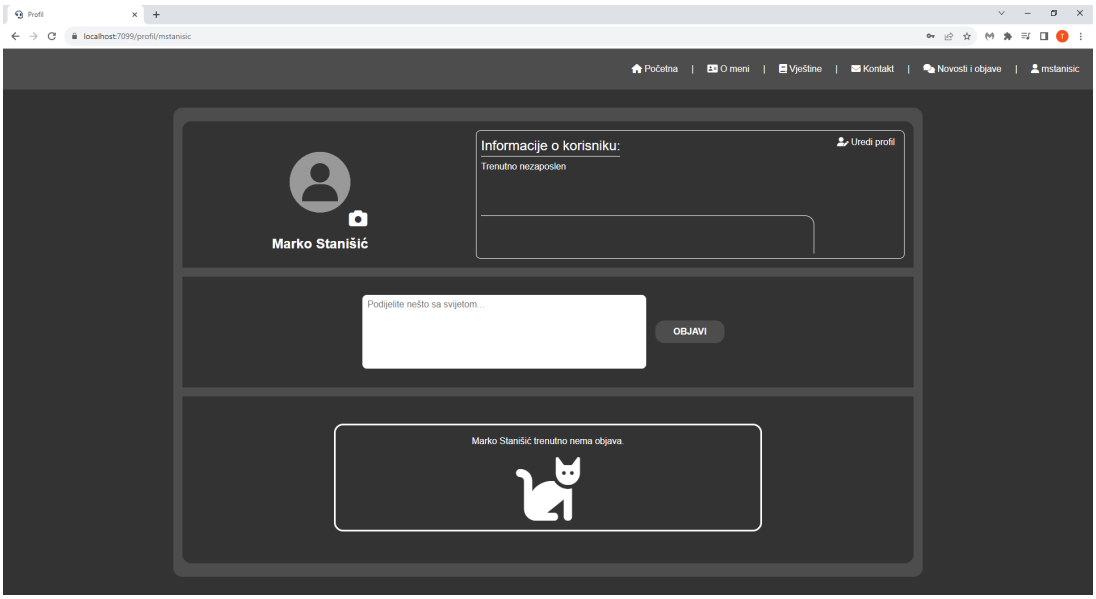
Isječak koda 4.25: Servis za odjavu

```
public async Task<string> LogoutUser(IJSRuntime jSRuntime)
{
    User user = await users.Find(u => u.Username == CurrentUser.Username).
        FirstOrDefaultAsync();

    if (user != null)
    {
        user.LoggedIn = false;
        user.SessionEnd = DateTime.Now;
        await users.ReplaceOneAsync(u => u.Username == user.Username, user);

        CurrentUser = null;
    }

    return "šUspjena␣odjava!";
}
```

Razumljivo samo po sebi, ako se korisnik želi odjaviti (vidljivo na 36), stisne da. pozvat će se servis za odjavu korisnika (vidljivo na 4.25), kraj njegove sesije će se postaviti na DateTime.Now, odnosno na današnji datum i vrijeme sad, te će ga aplikacija preusmjeriti na početnu stranicu (vidljivo na 4.24), a ako ne želi, ostat će prijavljen.

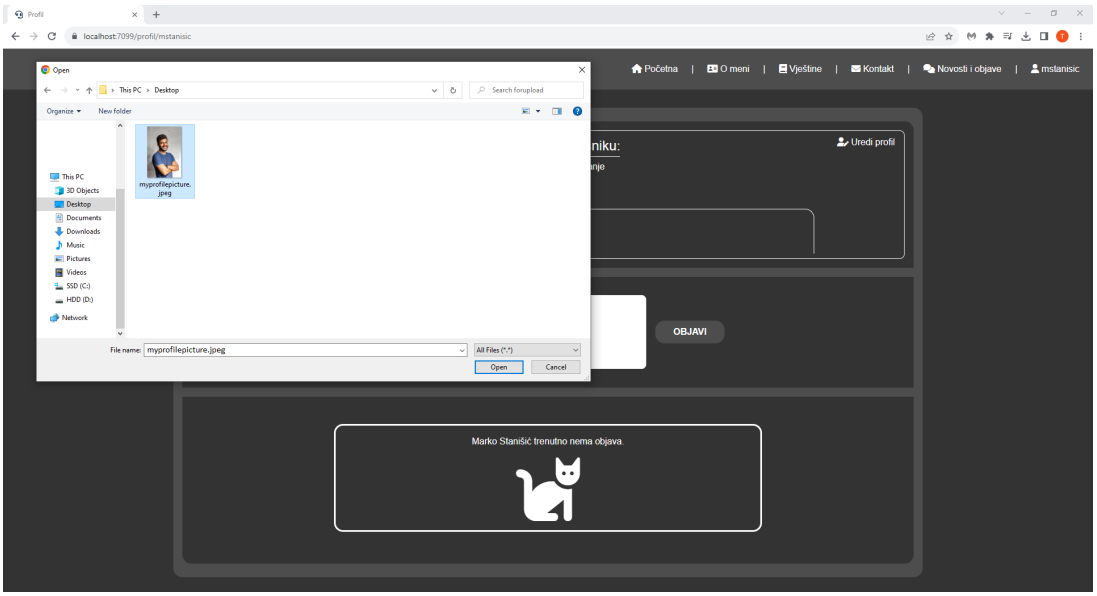
4.2.7. Profil

Prilikom prijave u svoj korisnički račun, korisnik može otići na svoj profil gdje će ga dočekati početni predložak profila (vidljivo na 37) koji on kasnije može naknadno uređivati kroz postavke dok profilnu sliku može učitati direktno na profilu.



Slika 37: Prikaz profila tek registriranog korisnika

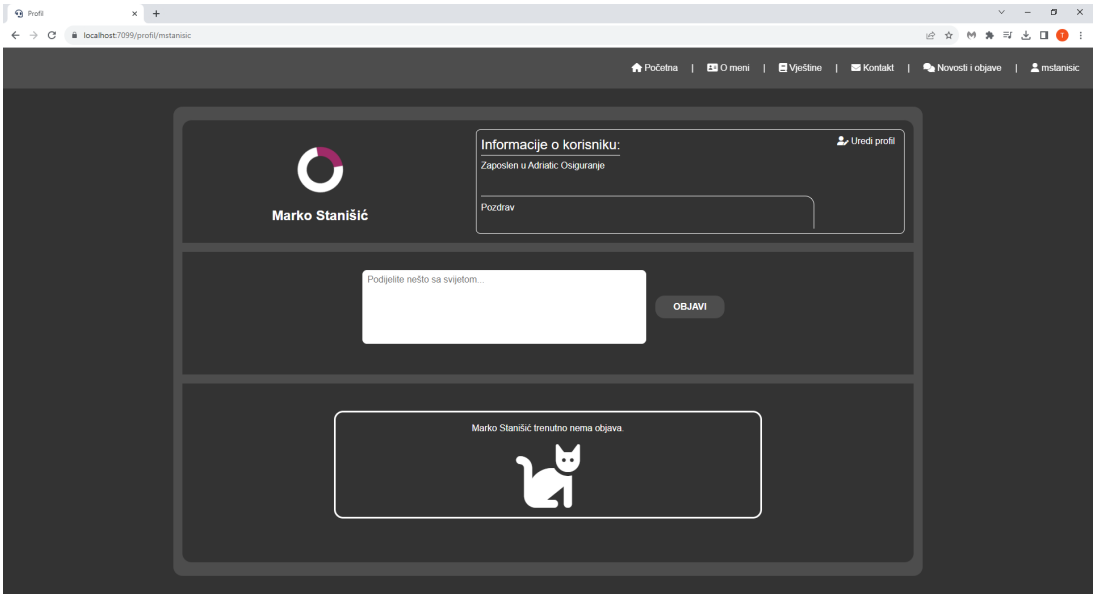
Korisniku se klikom na ikonu fotoaparata otvara prozor za učitavanje profilne slike te može učitati svoju željenu profilnu sliku (vidljivo na 38, 39, 4.26) koja se automatski ažurira nakon uspješnog učitavanja (vidljivo na 40, 4.27. 4.28).



Slika 38: Prozor za učitavanje profilne slike, preuzeto sa [82]

Isječak koda 4.26: Učitavanje profilne slike

```
private async Task UploadProfilePicture(InputFileChangeEventArgs e)
{
    IsLoadingProfilePicture = true;

    uploadedImage = e.File;
    await SaveProfilePicture();

    IsLoadingProfilePicture = false;
}
```

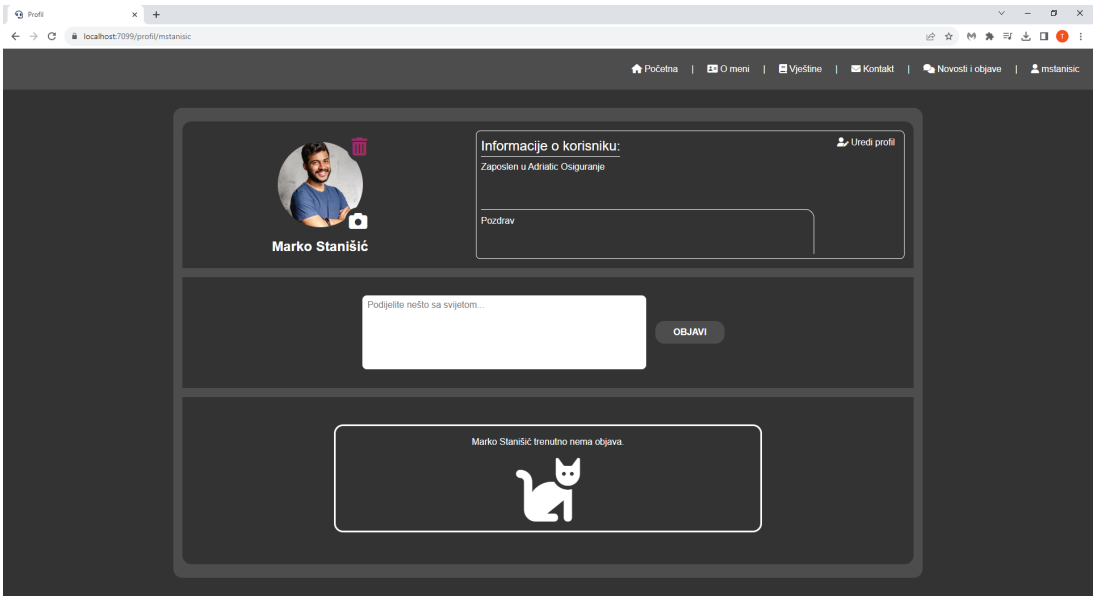


Slika 39: Proces učitavanja profilne slike

Isječak koda 4.27: Proces učitavanja profilne slike

```
public async Task<bool> UploadImageToDatabase(Stream fileStream, string fileName)
{
    try
    {
        using MemoryStream memoryStreamForRead = new MemoryStream();
        await fileStream.CopyToAsync(memoryStreamForRead);
        memoryStreamForRead.Seek(0, SeekOrigin.Begin);

        using Image image = Image.Load(memoryStreamForRead);

        image.Mutate(x => x.Resize(new ResizeOptions
        {
            Size = new Size(400, 400),
            Mode = ResizeMode.Max
        }));

        using MemoryStream memoryStreamForWrite = new MemoryStream();

        image.Save(memoryStreamForWrite, new JpegEncoder());
        byte[] imageToByte = memoryStreamForWrite.ToArray();

        PortfolioImage portfolioImage = new PortfolioImage
        {
            Name = fileName,
            Data = imageToByte
        };

        PortfolioImage imageInDatabase = new PortfolioImage();
        if (!string.IsNullOrEmpty(_userService.CurrentUser.Image_id))
        {
            imageInDatabase = await images.Find(i => i.Id == _userService.
                CurrentUser.Image_id).FirstOrDefaultAsync();
        }

        if (imageInDatabase.Data == null)
        {
            _userService.CurrentUser.Image_id = portfolioImage.Id;
            await images.InsertOneAsync(portfolioImage);
            await users.ReplaceOneAsync(u => u.Id == _userService.CurrentUser.Id,
                _userService.CurrentUser);
        }
        else
        {
            portfolioImage.Id = imageInDatabase.Id;
            await images.ReplaceOneAsync(i => i.Id == imageInDatabase.Id,
                portfolioImage);
        }

        return true;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Greska:␣{ex.Message}");
```

```
    }

    return false;
}
```

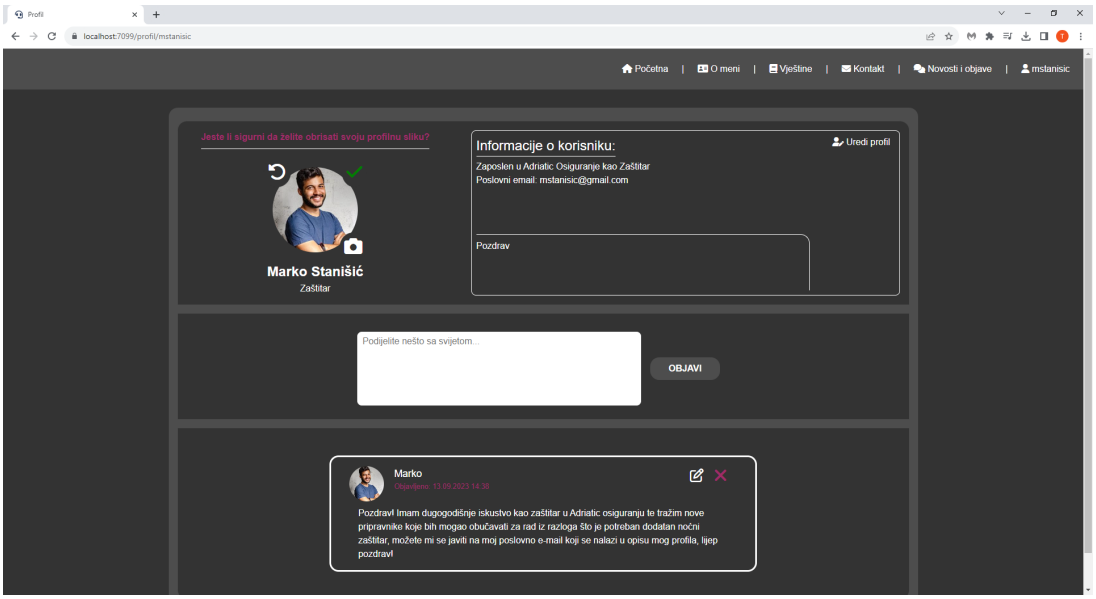


Slika 40: Prikaz učitane profilne slike

Isječak koda 4.28: Učitana profilna slika

```
private async Task SaveProfilePicture()
{
    if (uploadedImage != null)
    {
        var maxFileSize = 5 * 1024 * 1024; // 5 MB

        if (uploadedImage.Size > maxFileSize)
        {
            ProfilePhotoTooBigMessage = "Fotografija je prevelika! Maksimalna
                dozvoljena veliina fotografije je 5 MB.";
            ProfilePhotoTooBig = true;
            return;
        }

        ProfilePhotoTooBig = false;
        var stream = uploadedImage.OpenReadStream(maxFileSize);
        bool isImageUploaded = await ImageService.UploadImageToDatabase(stream,
            uploadedImage.Name);
        if (isImageUploaded)
        {
            UserProfileImage = await ImageService.GetImageFromDatabase(UserService.
                CurrentUser);
            StateHasChanged();
        }
    }
}
```

Korisnik klikom na ikonu kante za smeće ima mogućnost obrisati svoju profilnu sliku. Nakon klika na ikonu (vidljivo na 4.29), dobit će poruku za potvrdu brisanja profilne slike (vidljivo na 41) te dvije nove ikone (vidljivo na 4.30). Korisnik može ili potvrditi brisanje profilne slike klikom na ikonu za potvrdu (vidljivo na 4.32) ili otkazati brisanje profilne slike klikom na ikonu za nazad (vidljivo na 4.31).



Slika 41: Prikaz poruke za brisanje profilne slike

Isječak koda 4.29: Prikaz poruke za brisanje profilne slike - kod

```
public void OnProfilePictureDeletionButtonClick()
{
    ShowDeleteProfilePicturePrompt = true;
}
```

Isječak koda 4.30: Prikaz poruke za brisanje profilne slike - dizajn

```
if (!ShowDeleteProfilePicturePrompt)
{
    <label class="image-holder-top-right">
        <i style="color:␣@Constants.ColorPinkDark;␣cursor:␣pointer;" class="fas␣fa-
            trash-alt␣fa-2x" @onclick="OnProfilePictureDeletionButtonClick" />
    </label>
}
else
{
    <label class="image-holder-top-left">
        <i style="cursor:␣pointer;" class="fa-solid␣fa-rotate-left␣fa-2x" @onclick="
            CancelProfilePictureDeletion" />
    </label>

    <label class="image-holder-top-right">
        <i style="color:␣green;␣cursor:␣pointer;" class="fas␣fa-check␣fa-2x"
            @onclick="DeleteProfilePicture" />
    </label>
```

```
}
```

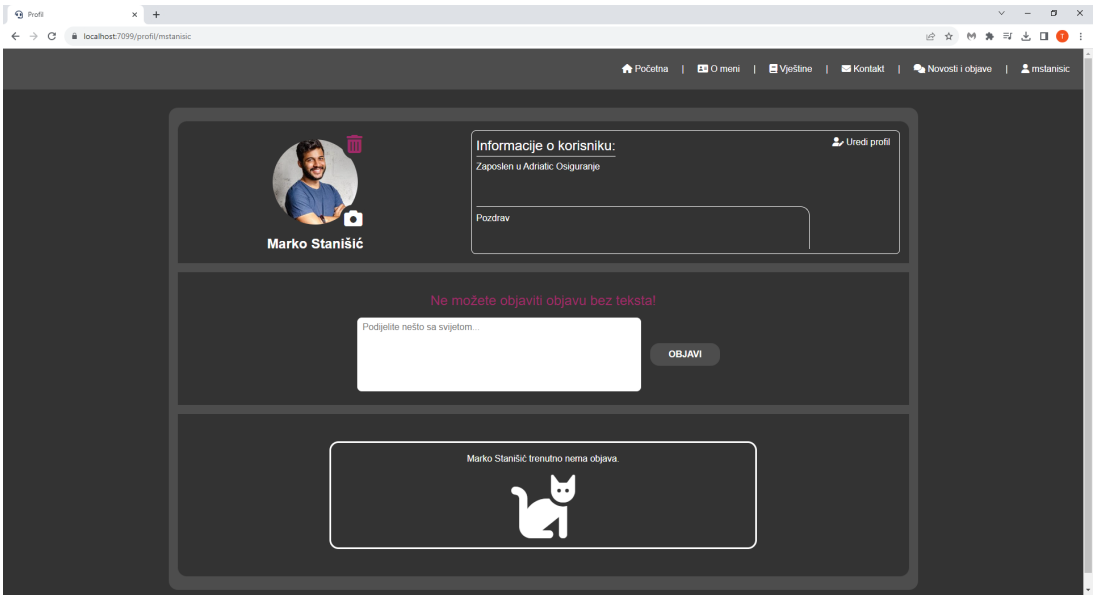
Isječak koda 4.31: Otkazivanje brisanja profilne slike

```
private void CancelProfilePictureDeletion()
{
    ShowDeleteProfilePicturePrompt = false;
}
```

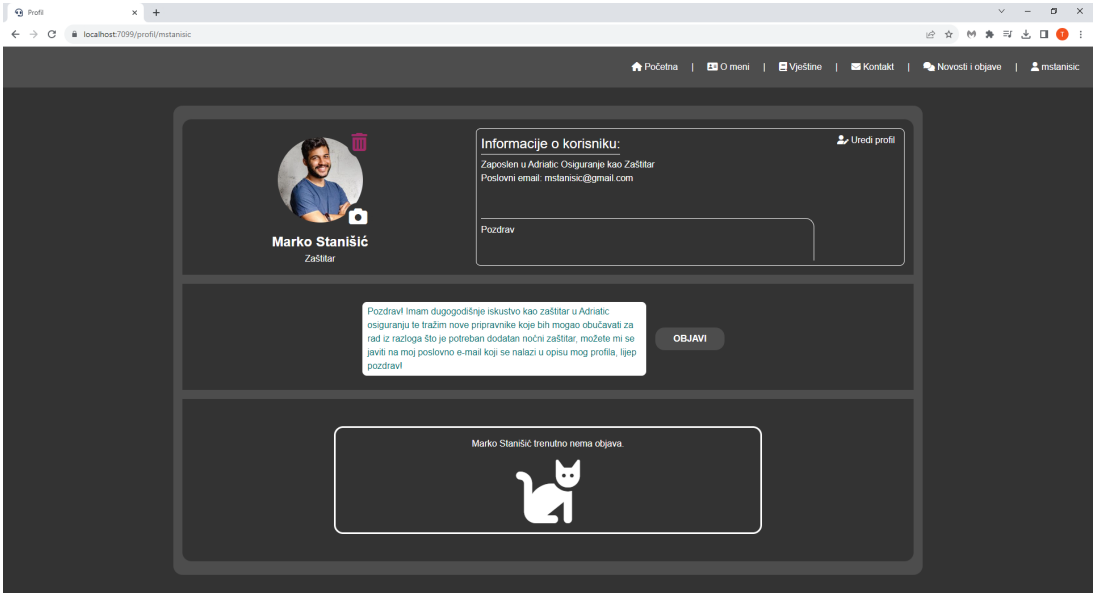
Isječak koda 4.32: Potvrda brisanja profilne slike

```
private async Task DeleteProfilePicture()
{
    IsLoadingProfilePicture = true;

    await ImageService.DeleteImageFromDatabase();
    UserService.CurrentUser.Image_id = string.Empty;
    await UserService.EditUser(UserService.CurrentUser);
    UserProfileImage = string.Empty;

    ShowDeleteProfilePicturePrompt = false;
    IsLoadingProfilePicture = false;

    StateHasChanged();
}

public async Task DeleteImageFromDatabase()
{
    try
    {
        await images.DeleteOneAsync(i => i.Id == _userService.CurrentUser.Image_id);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"šGreka:␣{ex.Message}");
        throw;
    }
}
```

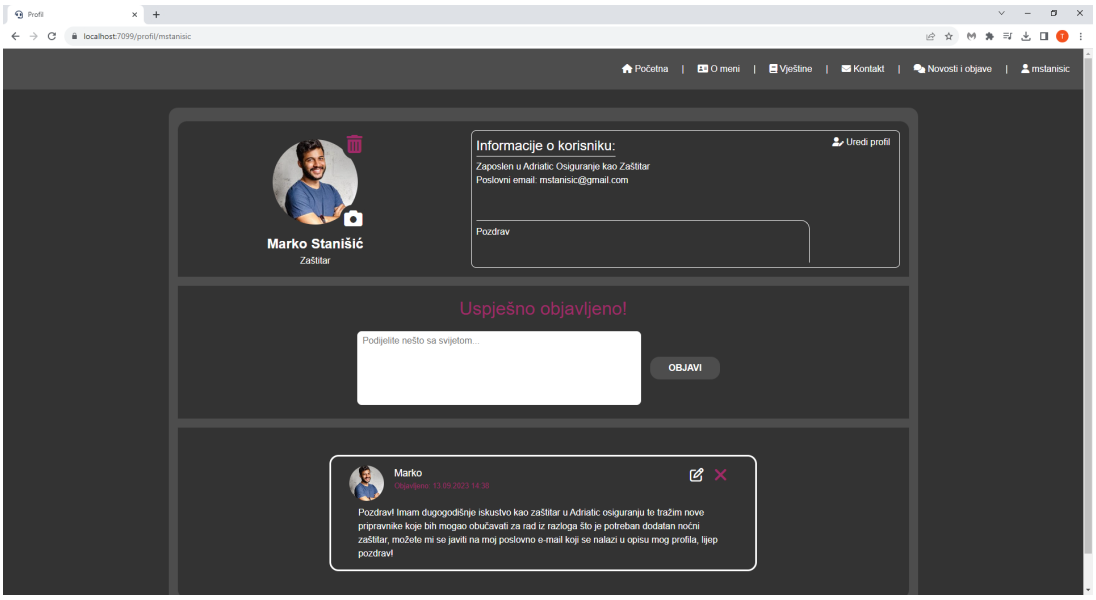
Korisnik može objaviti objavu na svom profilu tako što će upisati nešto u prozor za objavljivanje (vidljivo na 43, 4.33) te klikom na gumb (vidljivo na 4.34) objaviti tu objavu. Ako objava ne sadrži nikakav tekst, prikladna poruka će biti ispisana (vidljivo na 42). Ta objava kasnije postaje vidljiva (vidljivo na 44, 4.35) na njegovom profilu te na stranici Novosti i objave.



Slika 42: Poruka koja se javi ako korisnik pokuša objaviti prazan tekst



Slika 43: Primjer napisane objave



Slika 44: Prikaz poruke za uspjeh i nove objave na profilu

Isječak koda 4.33: Provjera unesenog teksta

```
private bool CheckInput()
{
    if (Text == string.Empty)
    {
        IsPostTextEmpty = true;
        return false;
    }

    IsPostTextEmpty = false;
    return true;
}
```

Isječak koda 4.34: Objavljivanje objave

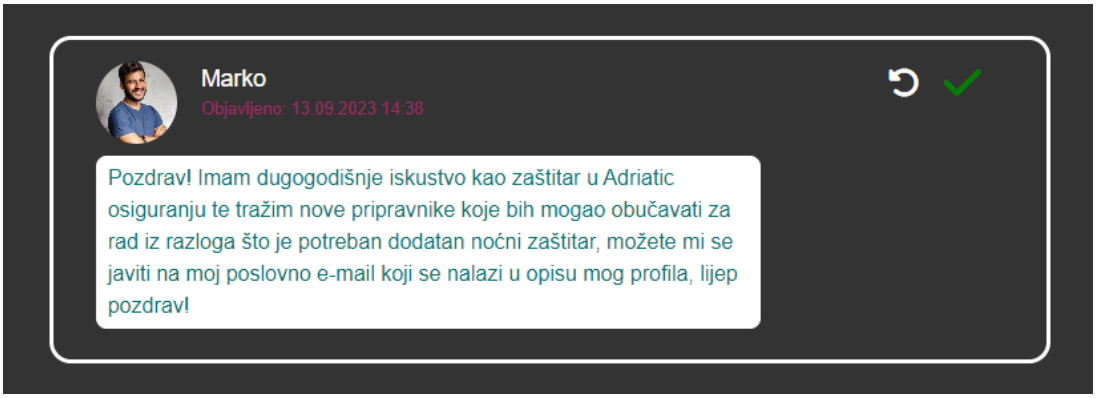
```
public async Task<string> CreatePost(Post post)
{
    await posts.InsertOneAsync(post);

    return "šUspjeno␣objavljeno!";
}
```

Isječak koda 4.35: Prikaz objave

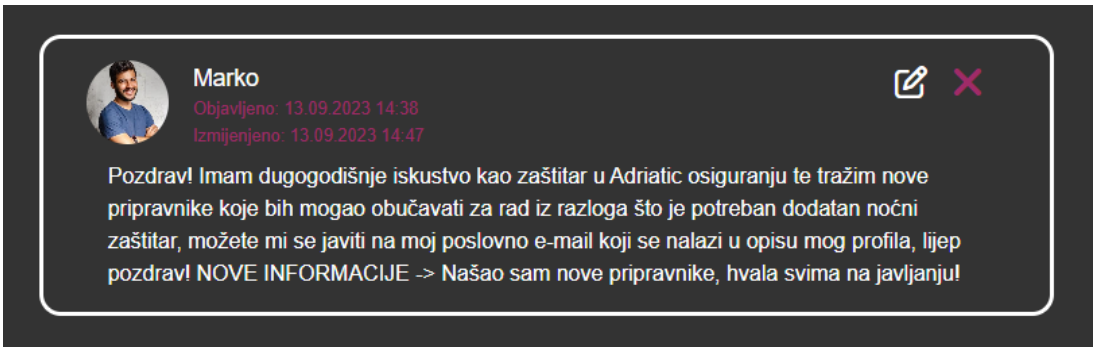
```
private async Task CreatePost()
{
    IsCreatingNewPost = true;

    if (CheckInput())
    {
        Post post = new Post
        {
            Text = Text,
```

```
            Posted = DateTime.Now,
            Author_Id = UserService.CurrentUser.Id
        };

        Text = string.Empty;

        SuccessMessage = await PostService.CreatePost(post);
        await GetAllPostsForUser();
        await JSRuntime.InvokeVoidAsync("setDivFullHeight");
        StateHasChanged();
        ClearSuccessMessage();
    }

    IsCreatingNewPost = false;
}
```

Korisnik nakon objave objave ima dodatne mogućnosti: može naknadno uređivati (vidljivo na 45) ili obrisati (vidljivo na 47, 4.38) svoju objavu. Prilikom uređivanja, novi tekst je spremljen i ažuriran (vidljivo na 4.36) te će na objavi pisati kad je ona bila zadnji put ažurirana (vidljivo na 46, 4.37).



Slika 45: Prikaz ažuriranja objave



Slika 46: Prikaz ažurirane objave

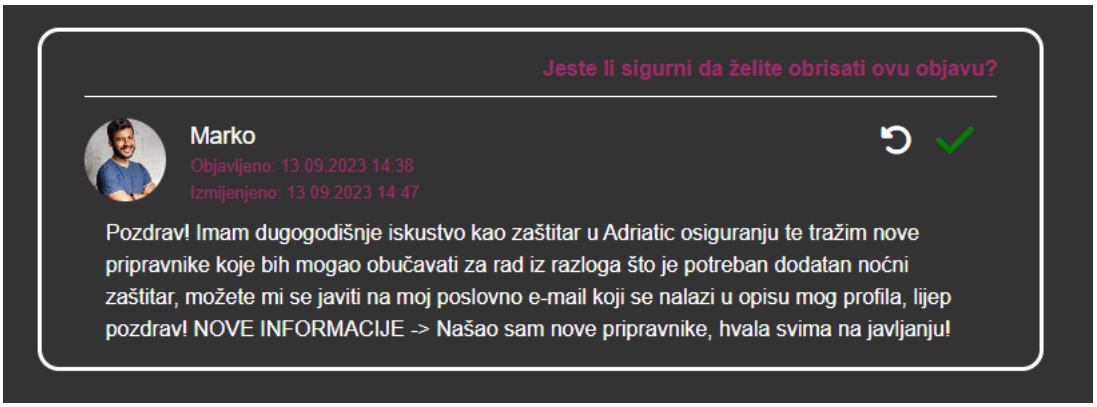
Isječak koda 4.36: Proces uređivanja teksta objave

```
public async Task EditPost(Post post)
{
```

```
    await posts.ReplaceOneAsync(p => p.Id == post.Id, post);
}
```

Isječak koda 4.37: Prikaz uređenog teksta objave

```
private async Task EditPost(Post post)
{
    IsEditingOrDeletingAPost = true;

    post.Text = PostForEditing.Text;
    post.Edited = DateTime.Now;

    await PostService.EditPost(post);
    await GetAllPostsForUser();
    await JSRuntime.InvokeVoidAsync("setDivFullHeight");
    StateHasChanged();

    EditMode = false;
    PostForEditing = null;
    PostForEditingOriginalText = string.Empty;

    IsEditingOrDeletingAPost = false;
}
```



Slika 47: Prikaz brisanja objave

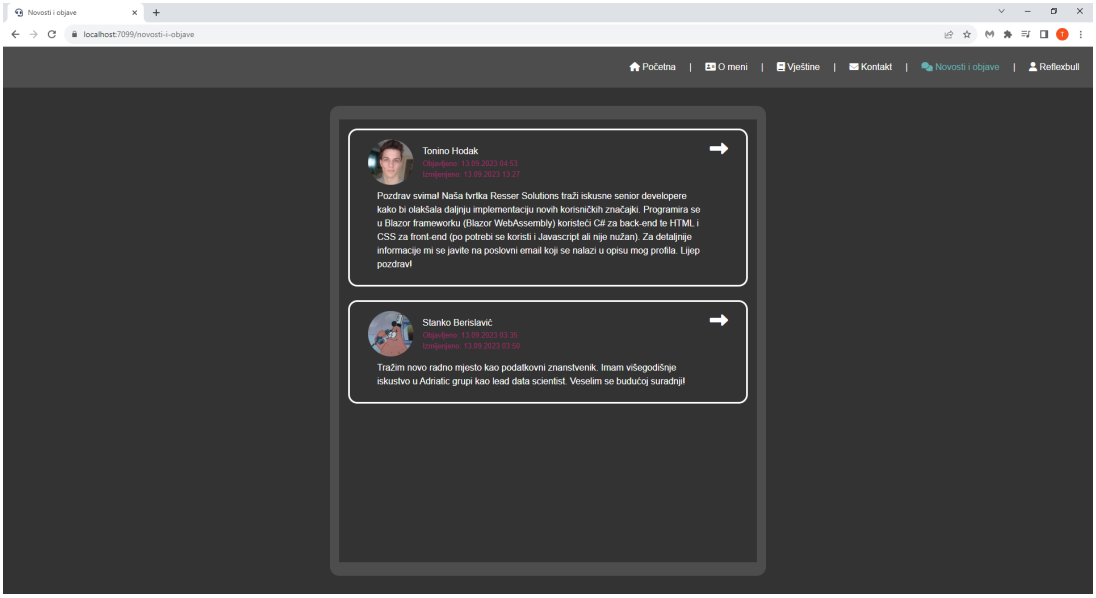
Isječak koda 4.38: Brisanje objave

```
private async Task DeletePost(Post post)
{
    IsEditingOrDeletingAPost = true;

    await PostService.DeletePost(post);
    await GetAllPostsForUser();
    await JSRuntime.InvokeVoidAsync("setDivFullHeight");
    StateHasChanged();

    ShowDeletePostPrompt = false;
    DeleteMode = false;
    PostForDeletion = null;
```

```
    IsEditingOrDeletingAPost = false;
}

public async Task DeletePost(Post post)
{
    await posts.DeleteOneAsync(p => p.Id == post.Id);
}
```

I naposlijetku, korisniku se otvaraju postavke profila klikom na gumb Uredi profil.

4.2.8. Novosti i objave



Slika 48: Stranica "Novosti i objave"

Na ovoj se stranici nalaze sve objave (vidljivo na 48) koje su korisnici unutar aplikacije napravili na svojim profilima. Stranica također sadrži funkcionalnost odlaska na profil korisnika klikom na strelicu koja se nalazi u desnom gornjem uglu svake objave. Stranice ima više - manje iste funkcionalnosti kao i profil, samo što se u ovom slučaju ne dohvaćaju objave jednog korisnika nego svih korisnika. Jedini dodatak je mogućnost odlaska na profil drugog korisnika prilikom klika na ikonu strelice (vidljivo na 49, 4.39, 4.40, 4.41).

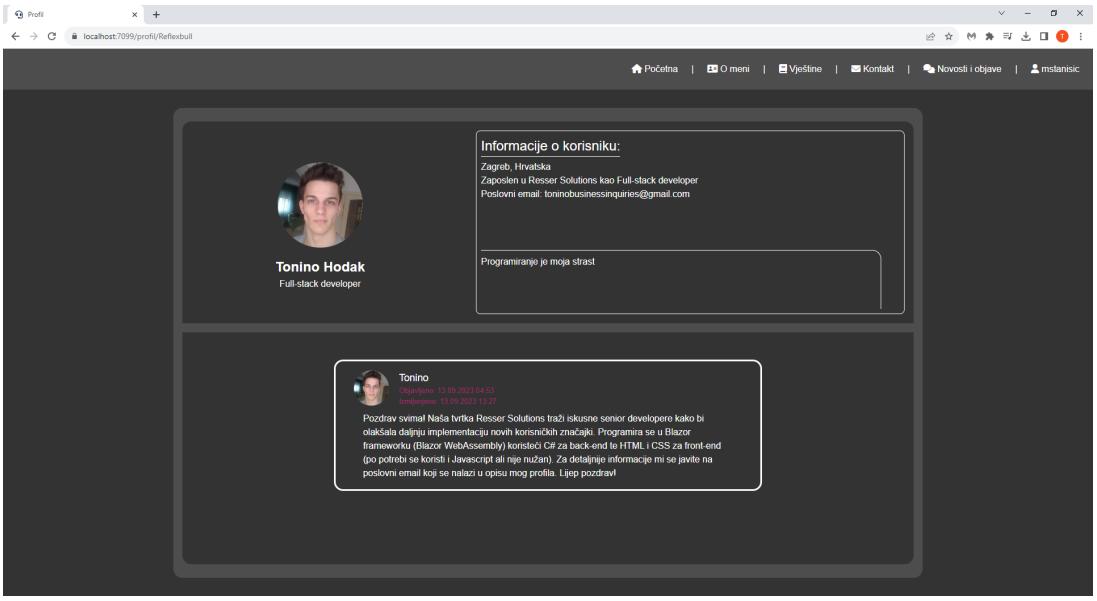
Ovisno o postavljen parametrima na inicijalnom učitavanju stranice, takvi će se podatci drugog korisnika prikazati (npr. trenutno korisnik ne može brisati / ažurirati objave, profil ili profilnu sliku drugog korisnika).

Isječak koda 4.39: Navigacija na profil drugog korisnika - okidač

```
<div class="goto-user-profile-arrow">
    <i style="cursor:␣pointer;" class="fas␣fa-long-arrow-alt-right␣fa-2x" @onclick="
        ()␣=>␣GoToUserProfile(postAuthorNames[post.Author_Id].Username)" />
</div>
```

Isječak koda 4.40: Navigacija na profil drugog korisnika - kod

```
private void GoToUserProfile(string username)
{
    NavigationManager.NavigateTo($"/profil/{username}");
}
```



Slika 49: Prikaz profila drugog korisnika

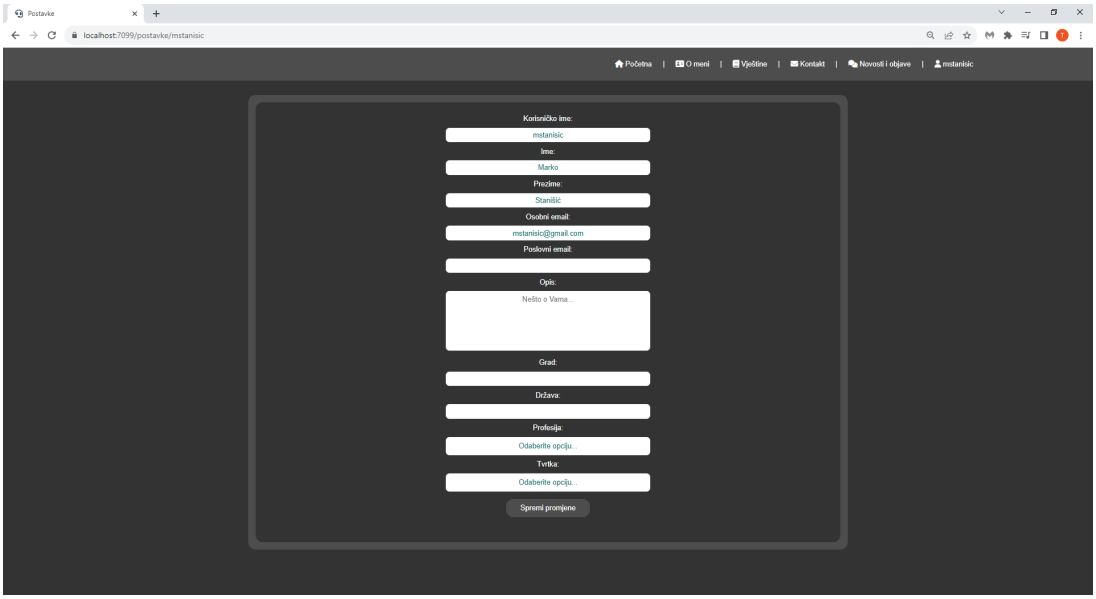
Isječak koda 4.41: Parametri postavljeni ovisnu o korisničkom imenu drugog korisnika

```
protected override async Task OnParametersSetAsync()
{
    IsLoading = true;

    if (!string.IsNullOrEmpty(Username))
    {
        User = await UserService.GetUserByUsername(Username);

        if (User != null)
        {
            UserProfileImage = await ImageService.GetImageFromDatabase(User);

            UserPosts = await PostService.GetAllPostsForUser(Username);
            UserPosts = UserPosts.OrderByDescending(p => p.Posted).ToList();

            Profession = await ProfessionService.GetUserProfession(User.
                Profession_Id);
            Company = await CompanyService.GetUserCompany(User.Company_Id);

            if (UserPosts.Count == 0)
            {
                Message = $"{User.FullName} trenutno nema objava.";
            }
```

```
            StateHasChanged();
        }
        else
        {
            Message = $"Korisnik sa čkorisnikim imenom '{Username}' ne postoji!";
        }
    }

    await JSRuntime.InvokeVoidAsync("setDivFullHeight");

    IsLoading = false;
}
```

4.2.9. Postavke

Prikaz stranice postavke do koje je moguće doći klikom na postavke u padajućem izborniku profila ili klikom na gumb "Uredi profil" na profilu korisnika.



Slika 50: Inicijalne postavke

Na inicijalnom učitavanju postavki, izvuku se podatci korisnika iz baze podataka, svojstva za to predodređena se popune vrijednostima te se prikažu korisniku (vidljivo na 50, 4.42, 4.43). Korisnik može polja mijenjati ovisno kako mu odgovara te će moći spremiti promjene samo ako su validacije valjane.

Isječak koda 4.42: Primjer polja koja poprimaju vrijednost

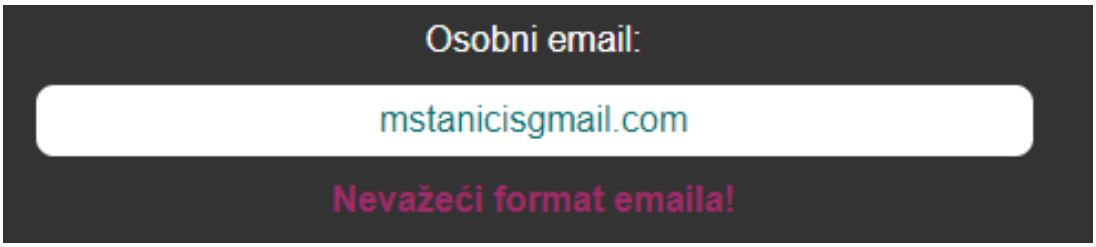
```
@* Username *@
<div>
    Korisnicko ime:
</div>

<input type="text" @bind="EditedUsername" />
```

```
<ValidationMessage For="@(() => User.Username)" />

@* First Name *@
<div>
    Ime:
</div>

<input type="text" @bind-value="EditedFirstName" />
<ValidationMessage For="@(() => User.FirstName)" />
```

Isječak koda 4.43: Popunjavanje postojećih polja

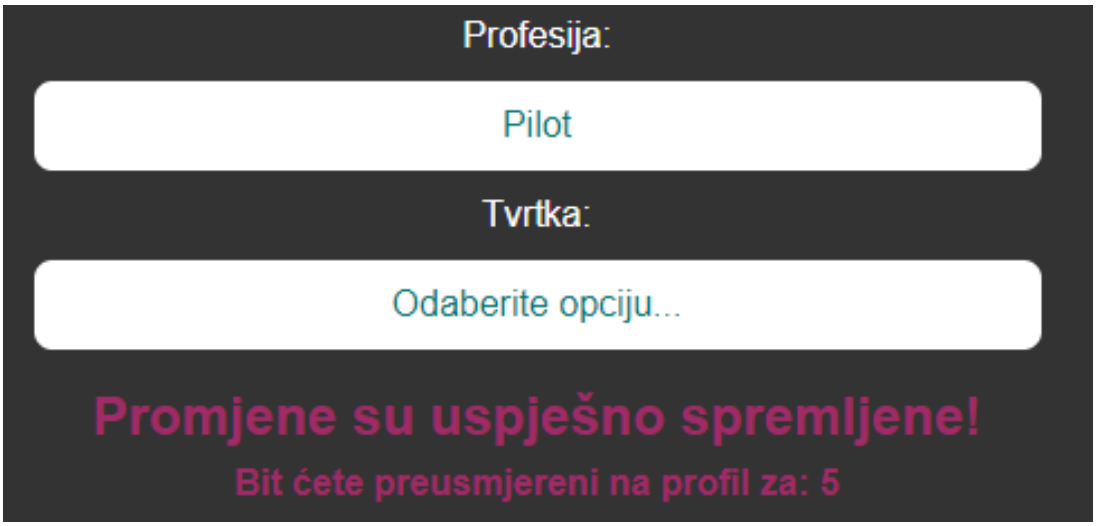
```
public async Task PrefillFields()
{
    EditedDescription = User.Description;
    EditedCity = User.City;
    EditedCountry = User.Country;
    if (!string.IsNullOrEmpty(User.Profession_Id))
    {
        EditedProfession = await ProfessionService.GetUserProfession(User.
            Profession_Id);
    }
    if (!string.IsNullOrEmpty(User.Company_Id))
    {
        EditedCompany = await CompanyService.GetUserCompany(User.Company_Id);
    }
    EditedUsername = User.Username;
    EditedFirstName = User.FirstName;
    EditedLastName = User.LastName;
    EditedPersonalEmail = User.Email;
    EditedBusinessEmail = User.BusinessEmail;
}
```

Ako korisnik pokuša spremiti postavke koje nisu važeće, dobit će prikladnu poruku za to (vidljivo na 51).



Slika 51: Poruka za nevažeći format e-maila

Ako uneseni podatci korisnika prođu sve potrebe validacije, spremit će se (vidljivo na 52, 4.44) te će ga stranica sama preusmjeriti na profil nakon 5 sekundi od uspješne promjene (vidljivo na 53, 4.45).



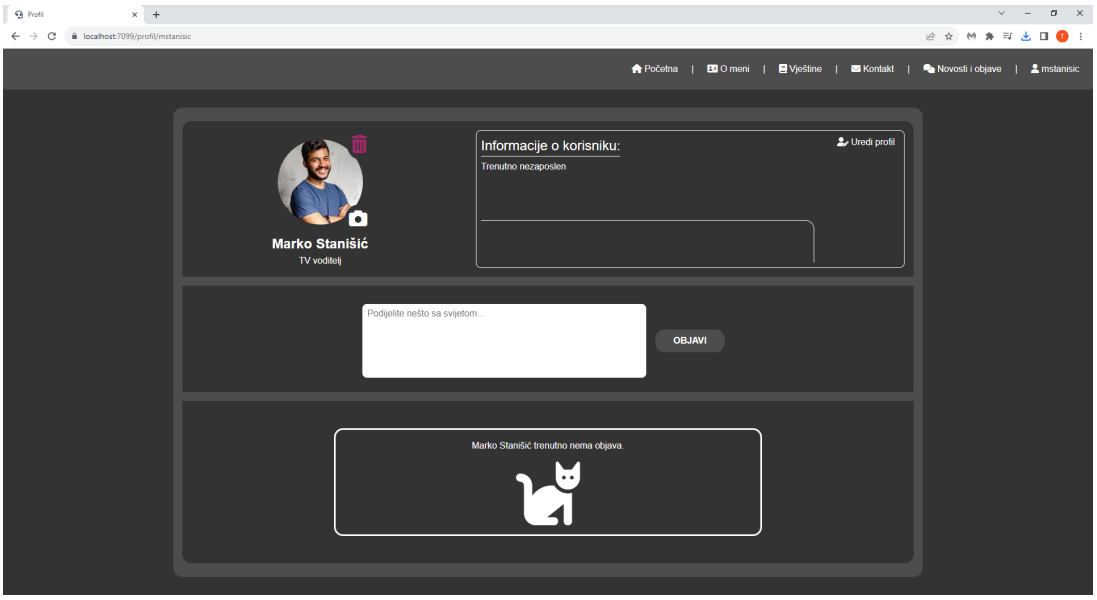
Slika 52: Uspješno spremljene postavke

Isječak koda 4.44: Ažuriranje korisnika u bazi podataka

```
public async Task EditUser(User user)
{
    await users.ReplaceOneAsync(u => u.Id == user.Id, user);
}
```

Isječak koda 4.45: Ažuriranje korisnika

```
private async Task SaveChanges()
{
    IsSavingChanges = true;

    if (EditContext.Validate())
    {
        SetNewValues();
        await UserService.EditUser(User);
        IsSavingChanges = false;
        IsRedirecting = true;

        SuccessMessage = "Promjene su šuspjeno spremljene!";
        string successfulProfileUpdate = "Bit ćete preusmjereni na profil za:";
        for (int i = 5; i > 0; i--)
        {
            Message = string.Empty;
            Message = $"{successfulProfileUpdate} {i}";
            StateHasChanged();
            await Task.Delay(1000);
        }

        NavigationManager.NavigateTo($"/profil/{Username}");
    }

    IsSavingChanges = false;
}
```



Slika 53: Uspješno spremljene postavke (postavljeno je zanimanje)

5. Zaključak

U ovom radu, naglasak je bio na izradi web-aplikacije osobnog portfelja, koristeći MongoDB bazu podataka. Predmet rada bio je razvoj funkcionalne i vizualno privlačne web-aplikacije, koristeći tehnologije poput HTML-a, CSS-a, JavaScript-a i C# u kombinaciji s Blazor okvirom.

Tokom procesa izgradnje web-stranice, korištene su razne metode i tehnike za modeliranje baze podataka, dok su se istovremeno implementirale funkcionalnosti koje su ključne za efikasno funkcioniranje web-aplikacije. Programski alati koji su korišteni uključuju Blazor framework za izgradnju aplikacije i MongoDB kao osnovu za spremanje i upravljanje podacima.

U radu, potvrđena je pretpostavku da je moguće stvoriti efikasnu i estetski privlačnu web-aplikaciju koristeći navedene tehnologije i alate. Dokazano je da da Blazor nudi odličnu platformu za razvoj web-aplikacija, a MongoDB se pokazao kao fleksibilna te efikasna baza podataka za upravljanje informacijama unutar aplikacije.

Kroz razradu rada, uspješno su obrađena sva najavljena gledišta iz uvoda, prezentirajući čitatelju korake i postupke uključene u razvoj moderne web-aplikacije. Rezultat rada je funkcionalna web-aplikacija koja ne samo da zadovoljava osnovne kriterije za osobni portfelj već i demonstrira mogućnosti suvremenih tehnologija u stvaranju kvalitetnih web-aplikacija.

Popis literature

[1] M. Chand, „Most Popular Databases In The World (2023),” *Most Popular Databases In The World*, 2023. adresa: https://www.c-sharpcorner.com/article/what-is-the-most-popular-database-in-the-world/ (pogledano 4. 9. 2023.).

[2] IBM, „Data modeling,” 2023. adresa: https://www.ibm.com/topics/data-modeling (pogledano 12. 9. 2023.).

[3] Microsoft, „Data modeling,” *Overview*, 2023. adresa: https://powerbi.microsoft.com/en-us/what-is-data-modeling/ (pogledano 14. 9. 2023.).

[4] Oracle, „Database,” 2023. adresa: https://www.oracle.com/database/what-is-database/#:~:text=Is%20a%20Database%3F-,Database%20defined,database%20management%20system%20(DBMS). (pogledano 14. 9. 2023.).

[5] M. Ibrahim, „What is a database cluster?,” 2023. adresa: https://www.harperdb.io/post/what-is-database-clustering (pogledano 12. 9. 2023.).

[6] A. Microsoft, „Cloud computing,” 2023. adresa: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-cloud-computing (pogledano 14. 9. 2023.).

[7] M. Kaufmann i A. Meier, *SQL and NoSQL Databases: Modeling, Languages, Security and Architectures for Big Data Management*, 2. izdanje. Cham CH: Springer, 2023., ISBN: 978-3-031-27907-2. DOI: 10.1007/978-3-031-27908-9. (pogledano 14. 9. 2023.).

[8] Wikipedia, „Edgar F. Codd,” 2023. adresa: https://en.wikipedia.org/wiki/Edgar_F._Codd (pogledano 13. 9. 2023.).

[9] E. F. Codd, „A relational model of data for large shared data banks,” 2023. adresa: https://dl.acm.org/doi/10.1145/362384.362685 (pogledano 13. 9. 2023.).

[10] P. Pedamkar, „Relational Database Advantages,” 2023. adresa: https://www.educba.com/relational-database-advantages/ (pogledano 14. 9. 2023.).

[11] MongoDB, „DB-Engines Ranking,” 2023. adresa: https://db-engines.com/en/ranking (pogledano 12. 9. 2023.).

[12] Oracle, „What is MySQL?,” 2023. adresa: https://www.oracle.com/mysql/what-is-mysql/ (pogledano 14. 9. 2023.).

[13] Oracle, „Database,” 2023. adresa: https://www.oracle.com/database/ (pogledano 12. 9. 2023.).

[14] Y. Matsunobu, „MyRocks: A space- and write-optimized MySQL database," 2016. adresa: https://engineering.fb.com/2016/08/31/core-data/myrocks-a-space-and-write-optimized-mysql-database/ (pogledano 13. 9. 2023.).

[15] J. Sobel, „Keeping Up," 2007. adresa: https://web.archive.org/web/20090618191723/http://blog.facebook.com/blog.php?post=7899307130 (pogledano 13. 9. 2023.).

[16] M. Hashemi, „The Infrastructure Behind Twitter: Scale," 2017. adresa: https://blog.twitter.com/engineering/en_us/topics/infrastructure/2017/the-infrastructure-behind-twitter-scale (pogledano 13. 9. 2023.).

[17] K. Kremerskothen, „Using, Abusing and Scaling MySQL at Flickr," 2010. adresa: https://code.flickr.net/2010/02/08/using-abusing-and-scaling-mysql-at-flickr/ (pogledano 13. 9. 2023.).

[18] A. B. Chintan Mehta, *MYSQL 8 Administrator's Guide*. Packt Publishing Ltd., 2018., ISBN: 978-1-78839-519-9. adresa: https://thedevelopmentstages.com/wp-content/uploads/2021/08/MySQL-8-Administrators-Guide-PDF-Room.pdf.

[19] TechTarget, „Microsoft SQL Server," 2023. adresa: https://www.techtarget.com/searchdatamanagement/definition/SQL-Server (pogledano 14. 9. 2023.).

[20] PostgreSQL, „PostgreSQL: The World's Most Advanced Open Source Relational Database," 2023. adresa: https://www.postgresql.org/ (pogledano 14. 9. 2023.).

[21] M. Stonebraker i L. A. Rowe, „THE DESIGN OF POSTGRES," adresa: https://dsf.berkeley.edu/papers/ERL-M85-95.pdf (pogledano 13. 9. 2023.).

[22] Wikipedia, „Oracle Corporation," 2023. adresa: https://en.wikipedia.org/wiki/Oracle_Corporation (pogledano 12. 9. 2023.).

[23] SQLite, „What Is SQLite?," 2023. adresa: https://www.sqlite.org/index.html (pogledano 12. 9. 2023.).

[24] SQLite, „Most Widely Deployed and Used Database Engine," 2023. adresa: https://www.sqlite.org/mostdeployed.html (pogledano 13. 9. 2023.).

[25] MongoDB, „What is NoSQL?," 2023. adresa: https://www.mongodb.com/nosql-explained (pogledano 14. 9. 2023.).

[26] MongoDB, „Non-relational databases," 2023. adresa: https://www.mongodb.com/databases/non-relational (pogledano 12. 9. 2023.).

[27] MongoDB, „What Is MongoDB?," 2023. adresa: https://www.mongodb.com/what-is-mongodb (pogledano 14. 9. 2023.).

[28] Google, „Cloud Bigtable," 2023. adresa: https://cloud.google.com/bigtable (pogledano 14. 9. 2023.).

[29] A. Cassandra, „What is Apache CassandraRegistered?," 2023. adresa: https://cassandra.apache.org/_/index.html (pogledano 14. 9. 2023.).

[30] IBM, „What is a message broker?," 2023. adresa: https://www.ibm.com/topics/message-brokers (pogledano 14. 9. 2023.).

[31] Redis, „Introduction to Redis,” 2023. adresa: https://redis.io/docs/about/ (pogledano 12. 9. 2023.).

[32] IBM, „What is CouchDB?,” 2023. adresa: https://www.ibm.com/topics/couchdb (pogledano 14. 9. 2023.).

[33] Erlang, „Practical functional programming for a parallel world,” 2023. adresa: https://www.erlang.org/ (pogledano 14. 9. 2023.).

[34] Talend, „What is MapReduce?,” 2023. adresa: https://www.talend.com/resources/what-is-mapreduce/#:~:text=achieving%20quicker%20results.-,What%20is%20MapReduce%3F,functioning%20of%20the%20Hadoop%20framework. (pogledano 14. 9. 2023.).

[35] IBM, „ACID properties of transactions,” 2023. adresa: https://www.ibm.com/docs/en/cics-ts/5.4?topic=processing-acid-properties-transactions (pogledano 14. 9. 2023.).

[36] Javatpoint, „Features of CouchDB,” 2023. adresa: https://www.javatpoint.com/features-of-couchdb (pogledano 14. 9. 2023.).

[37] Amazon, „Amazon DynamoDB,” 2023. adresa: https://aws.amazon.com/dynamodb/ (pogledano 14. 9. 2023.).

[38] Wikipedia, „Amazon Web Services,” 2023. adresa: https://en.wikipedia.org/wiki/Amazon_Web_Services (pogledano 12. 9. 2023.).

[39] A. S., „What Is HTML? Hypertext Markup Language Basics Explained,” 2023. adresa: https://www.hostinger.com/tutorials/what-is-html (pogledano 14. 9. 2023.).

[40] H. Ashtari, „What Is HTML5? Meaning, Elements, and Benefits,” 2023. adresa: https://www.spiceworks.com/tech/tech-general/articles/what-is-html-five/ (pogledano 14. 9. 2023.).

[41] M. W. Docs, „CSS basics,” 2023. adresa: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/CSS_basics (pogledano 14. 9. 2023.).

[42] M. W. Docs, „CSS,” *CSS*, 2023. adresa: https://developer.mozilla.org/en-US/docs/Web/CSS (pogledano 14. 9. 2023.).

[43] Wikipedia, „CSHARP,” *CSHARP*, 2023. adresa: https://en.wikipedia.org/wiki/C_Sharp_(programming_language) (pogledano 5. 9. 2023.).

[44] S. Jena, „Difference between Open Source Software and Closed Source Software,” *Difference between Open Source Software and Closed Source Software*, 2023. adresa: https://www.geeksforgeeks.org/difference-between-open-source-software-and-closed-source-software/ (pogledano 5. 9. 2023.).

[45] Microsoft, „A tour of the C# language,” 2023. adresa: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/ (pogledano 14. 9. 2023.).

[46] N. Raval, „Top JavaScript Usage Statistics to Prove Its Awesomeness in 2023,” 2023. adresa: https://radixweb.com/blog/top-javascript-usage-statistics (pogledano 14. 9. 2023.).

[47] M. W. Docs, „What is JavaScript?," 2023. adresa: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript (pogledano 14. 9. 2023.).

[48] FreeCodeCamp, „Just in Time Compilation Explained," 2020. adresa: https://www.freecodecamp.org/news/just-in-time-compilation-explained/ (pogledano 14. 9. 2023.).

[49] Wikipedia, „ECMAScript," 2023. adresa: https://en.wikipedia.org/wiki/ECMAScript (pogledano 12. 9. 2023.).

[50] NODEJs, „NODEJs," 2023. adresa: https://nodejs.org/en (pogledano 12. 9. 2023.).

[51] Microsoft, „ASP.NET Core Blazor," *Overview*, 2023. adresa: https://learn.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-7.0 (pogledano 5. 9. 2023.).

[52] C. Team, „What Is a Framework?," 2021. adresa: https://www.codecademy.com/resources/blog/what-is-a-framework/ (pogledano 14. 9. 2023.).

[53] F. Sazanavets, „Pros and cons of Blazor for web development," 2023. adresa: https://scientificprogrammer.net/2019/08/18/pros-and-cons-of-blazor-for-web-development/ (pogledano 12. 9. 2023.).

[54] M. Spasojevic, „Blazor Server vs Blazor WebAssembly, Pros and Cons," 2023. adresa: https://code-maze.com/blazor-webassembly-introduction/ (pogledano 12. 9. 2023.).

[55] N. Technologies, „Blazor framework: what is it, examples and benefits," 2023. adresa: https://nolt-technologies.com/blog/blazor-what-is-it-and-what-are-its-advantages (pogledano 12. 9. 2023.).

[56] R. Sheldon, „What is a codebase (code base)?" *What is a codebase (code base)?*, 2023. adresa: https://www.techtarget.com/whatis/definition/codebase-code-base (pogledano 5. 9. 2023.).

[57] Microsoft, „ASP.NET Core Blazor," *Razor Syntax*, 2023. adresa: https://learn.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-7.0 (pogledano 5. 9. 2023.).

[58] Microsoft, „ASP.NET Core Blazor," *Data Binding*, 2023. adresa: https://learn.microsoft.com/en-us/aspnet/core/blazor/components/data-binding?view=aspnetcore-7.0 (pogledano 5. 9. 2023.).

[59] R. Meltzer, „What is a Programming Library? A Beginner's Guide," *What is a Programming Library? A Beginner's Guide*, 2023. adresa: https://careerfoundry.com/en/blog/web-development/programming-library-guide/#:~:text=A%20programming%20library%20is%20a,few%20different%20pre%2Dcoded%20components. (pogledano 5. 9. 2023.).

[60] Microsoft, „ASP.NET Core Blazor," *Hosting models*, 2023. adresa: https://learn.microsoft.com/en-us/aspnet/core/blazor/hosting-models?view=aspnetcore-7.0 (pogledano 5. 9. 2023.).

[61] WebAssembly, „WebAssembly," *WebAssembly*, 2023. adresa: https://webassembly.org/ (pogledano 5. 9. 2023.).

[62] Cloudflare, „https://www.cloudflare.com/learning/performance/what-is-lazy-loading/," *https://www.cloudfl is-lazy-loading/*, 2023. adresa: https://www.cloudflare.com/learning/performance/what-is-lazy-loading/ (pogledano 5. 9. 2023.).

[63] Cloudflare, „What is caching?" *What is caching?*, 2023. adresa: https://www.cloudflare.com/learning/cdn/what-is-caching/#:~:text=Caching%20is%20the%20process%20of,in%20reference%20to%20Internet%20technologies. (pogledano 5. 9. 2023.).

[64] A. Aggarwal, „Introduction to Visual Studio," 2023. adresa: https://www.geeksforgeeks.org/introduction-to-visual-studio/ (pogledano 14. 9. 2023.).

[65] Microsoft, „What is NuGet?" *An introduction to NuGet*, 2023. adresa: https://learn.microsoft.com/en-us/nuget/what-is-nuget (pogledano 5. 9. 2023.).

[66] MongoDB, „MongoDB Documentation," *Promises*, 2023. adresa: https://www.mongodb.com/docs/drivers/node/current/fundamentals/promises/#:~:text=A%20Promise%20is%20an%20object,the%20operation%20that%20they%20wrap. (pogledano 5. 9. 2023.).

[67] MongoDB, „MongoDB Node.js Driver - The next generation Node.js driver for MongoDB," *Introduction*, 2023. adresa: https://mongodb.github.io/node-mongodb-native/#:~:text=The%20official%20MongoDB%20Node.,Promises%20or%20via%20traditional%20callbacks. (pogledano 5. 9. 2023.).

[68] MongoDB, „MongoDB Clusters," *What are clusters in MongoDB?*, 2023. adresa: https://www.mongodb.com/basics/clusters#:~:text=MongoDB%20Atlas%20Cluster%20is%20a,from%20your%20favorite%20web%20browser. (pogledano 5. 9. 2023.).

[69] E. Erkec, „SQL Connection Strings tips," *What are SQL connection strings?*, 2023. adresa: https://www.sqlshack.com/sql-connection-strings-tips/ (pogledano 14. 9. 2023.).

[70] SonarSource, „WHAT IS CLEAN CODE," *the standard for all code*, 2023. adresa: https://www.sonarsource.com/solutions/clean-code/#:~:text=Clean%20Code%20is%20code%20that's,value%20out%20of%20your%20software. (pogledano 5. 9. 2023.).

[71] A. Documents, „APB Documents," *MailKit Integration*, 2023. adresa: https://docs.abp.io/en/abp/latest/MailKit#:~:text=MailKit%20is%20a%20cross%2Dplatform,MailKit%20as%20the%20email%20sender. (pogledano 5. 9. 2023.).

[72] NuGet, „SixLabors.ImageSharp," 2023. adresa: https://www.nuget.org/packages/SixLabors.ImageSharp/3.0.0#:~:text=ImageSharp%20is%20a%20new%2C%20fully,powerful%20yet%20beautifully%20simple%20API. (pogledano 13. 9. 2023.).

[73] A. Bitbucket, „What is Git," 2023. adresa: https://www.atlassian.com/git/tutorials/what-is-git (pogledano 14. 9. 2023.).

[74] „Figma,” 2023. adresa: https://www.figma.com/ (pogledano 6. 9. 2023.).

[75] „Balsamiq,” 2023. adresa: https://balsamiq.com/ (pogledano 6. 9. 2023.).

[76] „AdobeXD,” 2023. adresa: https://helpx.adobe.com/support/xd.html (pogledano 6. 9. 2023.).

[77] F. Awesome, „Take the hassle out of icons in your project,” 2023. adresa: https://fontawesome.com/ (pogledano 14. 9. 2023.).

[78] TomEnglert, „ResXManager,” 2023. adresa: https://marketplace.visualstudio.com/items?itemName=TomEnglert.ResXManager (pogledano 11. 9. 2023.).

[79] M. Rouse, „Color Hex Code,” *What Does Color Hex Code Mean?*, 2023. adresa: https://www.techopedia.com/definition/29788/color-hex-code (pogledano 11. 9. 2023.).

[80] RegExr, „RegEx,” 2023. adresa: https://regexr.com/ (pogledano 11. 9. 2023.).

[81] Blazor, „Component events,” 2023. adresa: https://blazor-university.com/components/component-events/ (pogledano 11. 9. 2023.).

[82] I. Melo, „Happy Man,” 2023. adresa: https://www.pexels.com/photo/portrait-photo-of-smiling-man-with-his-arms-crossed-standing-in-front-of-white-wall-2379004/ (pogledano 13. 9. 2023.).

Popis slika

Popis tablica