WEB APLIKACIJA ZA OBRADU PODATAKA ISTRAŽIVANJA I PRIKAZ IZVJEŠTAJA

Cervelin, Matej

VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

WEB APLIKACIJA ZA OBRADU PODATAKA ISTRAŽIVANJA I PRIKAZ IZVJEŠTAJA

Matej Cervelin

Zagreb, Lipanj 2020.

Student vlastoručno potpisuje Završni rad na prvoj stranici ispred Predgovora s datumom i oznakom mjesta završetka rada te naznakom:

„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor, te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada“.

U Zagrebu, 11.06.2020.

Predgovor

Ovim putem htio bi se zahvaliti svome mentoru na pomoći i strpljenju tijekom cijeloga procesa od prijave do izrade završnog rada, svome šefu i kolegama s posla što su me naučili mnogim vrijednim vještinama i primili u svoje poslovno okruženje otvorenih ruku, svojim roditeljima i obitelji na kontinuiranoj podršci kroz godine kroz koju su mi omogućili da upišem željeni faks (i produžim malo školovanje), te svojoj curi koja me bodrila i znatno uljepšala moje vrijeme u ova dva mjeseca karantene te time mi pomogla da ostanem fokusiran na izradu ovog rada.

Prilikom uvezivanja rada, Umjesto ove stranice ne zaboravite umetnuti original potvrde o prihvaćanju teme završnog rada kojeg ste preuzeli u studentskoj referadi

Sažetak

Ideja ovog završnog rada je izraditi dio nove softverske platforme za Peekator koji će se poglavito koristiti za prikaz izvještaja klijentima. Obrada podataka prikupljenih kroz istraživanja je iznimno bitna kako bi se klijentima omogućila kvalitetna analiza dobivenih podataka kroz izvještaje. Na osnovu tih izvještaja klijenti će donositi zaključke o navikama krajnjih kupaca te planirati sljedeće poslovne kampanje za ostvarivanje lojalnosti trenutnih korisnika i privlačenje novih. Kako bi se podaci istraživanja mogli kvalitetno obraditi potreban je adekvatan alat te se ovim radom nastoji kreirati potrebno rješenje.

Ključne riječi: nova softverska platforma, prikaz izvještaja, obrada podataka, analiza

Abstract

The idea of this final paper is to build a part of the new software platform for Peekator which will primarily be used to display market research reports. The processing of data collected through research is extremely important in order to provide clients with a quality analysis of the given dana through reports. Based on these reports clients will draw conclusions about the habits of their customers and plan their future business campaigns to achieve loyalty from their current customers and bring in new ones. In order for the research data to be properly processed an adequate tool is required and so the goal of this paper is to create the necessary solution.

Key words: new software platform, display of reports, data processing, analysis

Sadržaj

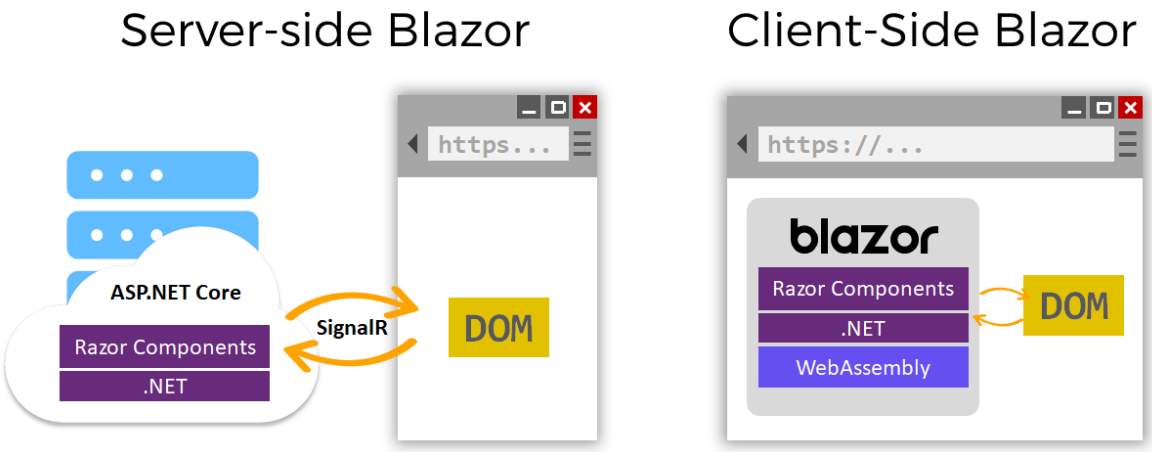
1. Uvod

Odnos s korisnicima te korisničko iskustvo je u svakoj tvrtki jedan od najbitnijih faktora pri planiranju poslovnih strategija. U svrhu analize tržišta provode se istraživanja kroz koja se prikupljaju podaci vezani uz iskustva krajnjih kupaca. Nakon što su istraživanja provedena, potrebno je agregirati prikupljene podatke te prezentirati ih na pristupačan način za analizu. Cilj ovog rada je bio implementirati stabilnu, skalabilnu i održivu web aplikaciju koja će prikupljati i obrađivati podatke iz različitih izvora, prikazati ih klijentima te omogućiti im različite načine za vlastitu analizu. Kroz tu analizu, u budućnosti će im biti olakšano donošenje konačnih poslovnih odluka. Osim analize izvještaja, klijenti također imaju mogućnost uređivati osobne podatke, te podatke vezane uz njihove urede. Aplikacija je dio veće softverske platforme (REST API), unutar koje komunicira s centralnim API sustavom te koristi podatke koji se integriraju iz MS SQL Server i MySQL baze. Podaci se prikupljaju kroz mobilnu aplikaciju nakon čega se spremaju u baze podataka odakle ih ova aplikacija povlači te priprema za prikaz u izvještaje.

U idućem poglavlju opisane su sve tehnologije korištene za izradu aplikacije. Nakon toga slijedi poglavlje „Poslovno okruženje“ koje detaljnije pojašnjava čemu služi ovaj rad te kakvu će on praktičnu svrhu imati. U poglavlju „Izrada aplikacije“ opisuje se arhitektura sustava, funkcionalnosti aplikacije, te su definirani sigurnosni aspekti aplikacije. Spomenuto poglavlje također sadrži i potpoglavlje „Analiza i vizualizacija“ u kojem se opisuju istraživanja za koja se prikazuju izvještaji unutar ove aplikacije.

2. Korištene tehnologije

Aplikacija je izrađena koristeći **Blazor Server-side** okvir (engl. *framework*) za razvoj *web* aplikacija. Blazor je nova tehnologija izgrađena na ASP.NET Core tehnologiji. Dok se u prošlosti C# koristio samo za izradu *back-end* dijela *web* aplikacija, Blazor omogućuje izradu interaktivnih *web* aplikacija, uključujući i njen *front-end* i *back-end*, a sve koristeći C# programski jezik. *Back-end* se odnosi na sloj pristupa podacima, odnosno na razvoj svih funkcionalnost koje aplikacija nudi. *Front-end* se odnosi na vizualni, tj. prezentacijski sloj u kojem se definira izgled *web* aplikacije. Za izradu internetskog sučelja često se koristi JavaScript *framework* poput Angulara. Zahvaljujući Blazoru, sada je to moguće koristeći C# umjesto JavaScripta. Ono što je posebno kod Blazora, jest da su i klijentski i poslužiteljski kod napisani koristeći C#. Postoje dva moguća načina uporabe Blazora. Prvi način uporabe je da se aplikaciju pokrene lokalno, tako da se preuzima i pokreće na strani klijenta (*Client-side*). U tome slučaju, Blazor pokreće *client-side* kod izravno u pregledniku, koristeći WebAssembly. WebAssembly (skraćeno *Wasm*) je otvoreni standard koji definira prijenosni format binarnog koda za izvršne programe. Drugi način je da se aplikacija pokrene na poslužitelju (*Server-side*). U tom slučaju, između preglednika i *server-side* aplikacije se uspostavi SignalR veza. SignalR je *framework* za razmjenu poruka u stvarnom vremenu. Pomoću njega se podaci o događaju s klijenta, poput klika miša, šalju poslužitelju. Nakon uspješne obrane potrebne izmjene korisničkog sučelja se vraćaju nazad klijentu i spajaju u DOM. (Microsoft, Blazor, 2020; Wikipedia, Blazor, 2020; Web Assembly, 2020) Razlike između navedena dva modela Blazora vidljive su na slici (Slika 2.1.).



Slika 2.1. Usporedba Blazor hosting modela (Ivan Derevianko, Blazor, 2020)

Blazor aplikacije su sastavljene od ponovo iskoristivih komponenti internetskog sučelja. Komponenta u Blazoru formalno se naziva Razor komponentom.

Razor komponenta je samostalni komad korisničkog sučelja (UI), poput stranice, dijaloga ili obrasca. Moguće ju je ugnijezditi, ponovo koristiti i dijeliti među projektima. Prema pravilu imenovanja, prvo slovo naziva komponente mora biti veliko. Implementira se kombinacijom C# programskog jezika i HTML oznake (engl. *markup*). Korisničko sučelje komponente se definira koristeći HTML. (Microsoft, Razor components, 2020) Unutar korisničkog sučelja može se implementirati dijelove programske logike (poput petlji, uvjeta, izraza) koristeći ugrađenu C# sintaksu koja se zove Razor. Razor koristi znak „@“ za prijelaz s HTML-a na C#. Razlikuju se Razor izrazi i Razor blokovi koda. Razor procjenjuje C# izraze i prikazuje ih u HTML izlazu (engl. *output*). Razor izrazi se dijele na eksplicitne i implicitne. Uz iznimku ključne riječi *await*, implicitni izrazi ne smiju sadržavati razmak. Sljedeći isječak koda prikazuje primjer implicitnog izraza:

```
<p>@DateTime.Now.ToLocalTime()</p>
```

Eksplicitni izrazi započinju znakom „@“ nakon čega slijede zagrade unutar kojih se pišu izraze. Sljedeći primjer koristi eksplicitni izraz kako bi se provjerilo ako *boolean* varijabla `IsNumbered` od objekta `Question` ima vrijednost istina (engl. *true)* ili laž (engl. *false),* i ovisno o tome ispisao određeni rezultat:

```
<p>@(Question.IsNumbered ? "%" : "")</p>
```

Razor blokovi koda započinju znakom „@“ nakon čega slijede vitičaste zagrade unutar kojih se piše kod.

```
@{
    var statement = "";

    if(5 > 1)
    {
        statement = "Five is greater than one!";
    }else{
        statement = "Five is lesser than one!";}

<p>@hello</p>
```

Kod 2.1. Primjer Razor bloka koda

U primjeru (Kod 2.1.) unutar Razor bloka je definiran tekstualni niz (engl. *string)* koji se kasnije ispisuje implicitnim izrazom Razor sintakse unutar HTML *markupa* za paragraf. Izrazi i blokovi koda dijele isti opseg, te stoga se može unutar njih koristiti iste varijable. Za razliku od izraza, C# kod unutar Razor blokova se ne prikazuje u HTML izlazu. (Microsoft, Razor syntax, 2020) Pri kompilaciji aplikacije, HTML označavanje i C# logika ispisa se pretvaraju u klasu komponente. Članovi te klase su definirani unutar `@code` bloka. Dobra praksa programiranja je odvojiti programsku logiku od označnog HTML koda. Umjesto da se piše programska logika i definiraju varijable unutar `@code` bloka, može se napraviti *code-behind* datoteka, koja će onda biti klasa komponente. Pravilo imenovanja koje se prati u ovome radu je da ako se komponenta zove `CapiComponent`, onda se njena klasa komponente mora zvati `CapiComponentBase`. Nakon što je kreirana klasa komponente, potrebno je izmijeniti komponentu tako da nasljeđuje tu klasu. To se radi tako da se komponenti na vrh doda ključna riječ `@inherits` nakon čega slijedi naziv klase komponente. Sljedeći primjer prikazuje kako to izgleda za komponentu `CapiComponent`:

```
@inherits CapiComponentBase
```

Iako je zadani način korištenja miješani pristup, u ovoj aplikaciji korišten je pristup s dvije odvojene datoteke. Razor komponentu se ne može prikazati ako joj nije zadana ruta putem koje joj se onda može pristupiti. Kako bi se postavila rutu, na vrh stranice je potrebno dodati ključnu riječ `@page`, kao što je to vidljivo u sljedećem primjeru:

```
@page "/project/{ProjectGuid}"
```

Nakon ključne riječi `@page` slijedi atribut rute (engl. *route attribute*) koji definira rutu do stranice na kojoj će se moći otvoriti i prikazati tu stranicu. Tijekom izvođenja, usmjerivač traži klase komponente koje imaju *route attribute* i prikazuje onu komponentu čiji predložak rute odgovara traženom URL-u. Za naprednije pretraživanje može se dodati parametar rute nakon *route attributea*. (Microsoft, Razor components, 2020) Opcionalni parametri nisu podržani, što znači da kako bi se moglo pristupiti komponenti bez navođenja parametra u ruti, potrebno je koristiti dvije `@page` direktive, kao u sljedećem primjeru:

```
@page "/project
@page "/project/{ProjectGuid}"
```

Razor stranica je datoteka koja ima `.cshtml` sufiks. Slične su komponentama, no postoje neke bitne razlike. Za razliku od komponenti, Razor stranice imaju zadano postavljenu direktivu `@page`, što znači da ta stranica izravno rješava zahtjeve, bez prolaska kroz

kontroler (engl. *controller*), te da joj se može pristupiti koristeći ime stranice kao rutu. Pri kreaciji Razor stranice postoji opcija da se izgenerira `PageModel` klasa. Ta klasa, jednako kao i klasa komponenti za komponente, služi kako bi se odvojio sloj korisničkog sučelja od sloja poslovne logike. Prema konvenciji, `PageModel` klasa ima isto ime kao i datoteka Razor stranice s razlikom da je njen sufiks `.cshtml.cs` (Microsoft, Introduction to Razor Pages, 2020). Primarni fokus ove aplikacije je obrada i analiza podataka prikupljenih kroz istraživanja. Nakon prikupljanja, te podatke je potrebno spremiti u bazu podataka te su stoga potrebne tehnologije za spremanje i pristup spremljenim podacima.

Entity Framework (EF) Core je lagana (engl. *lightweight*), proširiva, višeplatformska verzija popularne Entity Framework tehnologije pristupa podacima. EF Core je otvoreni softver (engl. *open-source*), što znači da uz njega dolazi njegov izvorni kod koji bilo tko onda može izmjenjivati prema *open-source* licenci. EF Core je *framework* objektno/relacijskog preslikavanja, što znači da programerima omogućuje pristup i upravljanje podacima iz baza podataka. EF Core podržava brojne motore baza podataka (engl. *database engine*), no za ovaj projekt su bitni MS SQL Server te MySQL. (Microsoft, Entity Framework Core, 2020)

Microsoft (MS) SQL Server je sustav za upravljanje relacijskom bazom podataka kojoj je primarni jezik za upite *Transact* SQL (T-SQL), što znači da osim osnovnih i klasičnih (SELECT tipa) SQL upita dozvoljava i složenije stvari poput mijenjanja programskog toka (IF naredba) i slično. Razvio ju je Microsoft. To je softverski proizvod s primarnom funkcijom spremanja i dohvaćanja podataka prema zahtjevima drugih softverskih aplikacija. Aplikacija može pristupiti podacima s baze ako se nalazi na istom računalu, ali i s drugog računala preko mreže (uključujući Internet). (Wikipedia, Microsoft SQL Server, 2020)

MySQL je besplatan, *open-source* sustav za upravljanje bazom podataka koji je razvila švedska tvrtka MySQL AB, kasnije preuzeta od strane Oracle Corporation. MySQL je , uz PostgreSQL, čest izbora baze za *open-source* projekte, te se distribuira kao sastavni dio serverskih *Linux* distribucija, no također postoje inačice i za ostale operacijske sustave poput Mac OS-a, Windows-a itd. MySQL baza je slobodna za većinu uporaba. Ranije u svom razvoju, MySQL baza podataka suočila se s raznim protivnicima MySQL sustava organiziranja podataka jer su joj nedostajale neke osnovne funkcije definirane SQL standardom. Naime, MySQL baza je optimizirana kako bi bila brza nauštrb funkcionalnosti. Nasuprot tome, vrlo je stabilna i ima dobro dokumentirane module i ekstenzije te podršku od brojnih programskih jezika. (Wikipedia, MySQL, 2020) EF Core u

svome osnovnom izdanju ne podržava MySQL bazu podataka. Kako bi se ova aplikacija mogla spojiti na MySQL bazu, mora se koristiti ekstenzija na EF Core, tj. dodatna biblioteka. Takva biblioteka se naziva davateljem baza podataka (engl. *database provider*), te se distribuira kao NuGet paket. Postoji službeni Oracle-ov NuGet paket pod nazivom MySql.Data.EntityFrameworkCore, ali ga nije moguće koristiti u ovome projektu jer podržava EF Core do verzije 2.1, dok ovaj projekt koristi verziju 3.1, stoga je potrebno koristiti alternativnu tehnologiju u vidu Pomelo (MySQL). Za implementaciju Pomela unutar projekta, potrebno je instalirati Pomelo.EntityFrameworkCore.MySql NuGet paket.

ASP.NET Identity je API čija implementacija korisnicima ove aplikacije omogućuje funkcionalnosti prijave kroz jednostavno korisničko sučelje. Pomoću ovog API-ja moguće je upravljati korisnicima i njihovim svojstvima, od lozinki i uloga (engl. *role*) do *tokena*, tvrdnji (engl. *claim*), email potvrda i drugog. Aplikacija koristi ASP.NET Identity korisničko sučelje s izgledom prilagođenim generalnom dizajnu ove aplikacije. Zadani način prijave kroz ovaj API je koristeći korisničke podatke stvorene putem registracije u sustav, no podržane su i eksterne prijave od raznih pružatelja prijava uključujući Facebook, Google, Microsoft Account i Twitter. (Microsoft, ASP.NET Core Identity, 2020) Postoji više načina implementacije ASP.NET Identity API-ja. Mehanizmi se dijele na autentifikaciju na temelju *tokena*, te autentifikaciju na temelju kolačića (engl. *cookie*), koja je ujedno mehanizam korišten u ovoj aplikaciji.

CSS je programski jezik pomoću kojeg se definira stil HTML elemenata *web* dokumenta. HTML je označni jezik za opis sadržaja *web* stranice u obliku formi, paragrafa, tablica, naslova, slika i drugog. Takav sadržaj je vrlo bazičan te *web* stranica u takvom osnovnom obliku ne bi privukla puno posjetitelja. Kako bi se obogatilo aplikaciju raznim stilovima neizbježno je koristiti CSS. CSS nudi razne mogućnosti za stiliziranje sadržaja, od različitih fontova, promjene pozadinskih boja, veličine slika, rasporeda elemenata i drugog. (W3C, HTML & CSS, 2020) Sljedeći primjer prikazuje CSS kod kojim se izmjenjuje font teksta unutar jednog dokumenta ove aplikacije:

```
html, body {
    font-family: 'Helvetica Neue', Helvetica, Arial, sans-
serif;
}
```

CSS je vrlo koristan, no zahtijeva puno uloženog vremena kako bi se ostvario kvalitetan dizajn *web* stranice, ponajviše što se tiče konzistentnosti na uređajima različite veličine

ekrana. Kako bi se uštedilo to vrijeme, pri izradi ove aplikacije je korišten jedan od brojnih CSS alata dostupnih na internetu, a to je Bootstrap.

Bootstrap je *framework* korišten za brzo i efikasno dizajniranje *web* stranica, te kao takav je jedan od najpopularnijih na svijetu. Bootstrap je izrađen koristeći HTML, CSS i JavaScript. Omogućuje korištenje predefiniranih stilova i komponenti koje znatno olakšavaju izradu korisničkog sučelja Internet stranica. Korištenje *front-end frameworka* za dizajn *web* stranica je postao standard te Bootstrap predvodi kao neosporni vođa među takvim alatima. Bootstrap je intuitivan i vrlo lak za uporabu te od korisnika ne zahtijeva previše uloženog vremena kako bi se upoznao s alatom. Jedna od glavnih prednosti Bootstrapa je što je izrađen s mobilnim uređajima u vidu kako bi osigurao responzivnost *web* stranica bez obzira na veličinu ekrana. Bootstrap je započeo kao interni projekt u tvrtki Twitter gdje su Mark Otto i njegov mali tim krenuli razvijati alat koji im je trebao pomoći u lakšoj izradi sučelja. Prije Bootstrapa postojao je problem repetitivnosti pri izradi sučelja u svakom novom projektu, te nedosljednosti pri korištenju različitih biblioteka stilova što je između ostalog vodilo ka velikom opterećenju održavanja. Projekt je započeo pod nazivom Twitter Blueprint te je kroz razvoj prerastao u nešto više, i ultimativno, kroz nekoliko iteracija postao ono što je dana poznato kao Bootstrap. (Wikipedia, Bootstrap, 2020; TaniaRascia, What is Bootstrap, 2020)

JavaScript je programski jezik najčešće korišten kao *client-side* skriptni jezik za implementaciju kompleksnih značajki (engl. *feature*) u *web* aplikacije. JavaScript može upravljati HTML elementima i njihovim sadržajem, mijenjati vrijednosti njihovih atributa te raditi izmjene na stilu tih elemenata. (Mozilla, What is JavaScript, 2020) Blazor omogućuje uporabu C# klijentskog koda umjesto JavaScripta te stoga JavaScript u ovoj aplikaciji nije u širokoj upotrebi, osim u nužnim situacijama. Kod 2.2. prikazuje JavaScript funkciju koja se izvodi pri učitavanju dokumenta u kojoj se manualno vrši *submit* forme kako bi se odjavilo korisnika. Ova funkcija je potrebna kako bi se izbjeglo učitavanje ove stranice na način da se odjava dogodi automatski i odmah preusmjeri korisnika nazad na ekran za prijavu.

```
<script>
    window.onload = function () {
        var form = document.getElementById('logoutForm');
        form.submit();
    }
</script>
```

Kod 2.2. Primjer JavaScript funkcije pri učitavanju dokumenta

3. Poslovno okruženje

Peekator je tvrtka koja se bavi istraživanjem tržišta i savjetovanjem u području korisničkog iskustva. Nudi prikupljanje podataka kroz različite kanale:

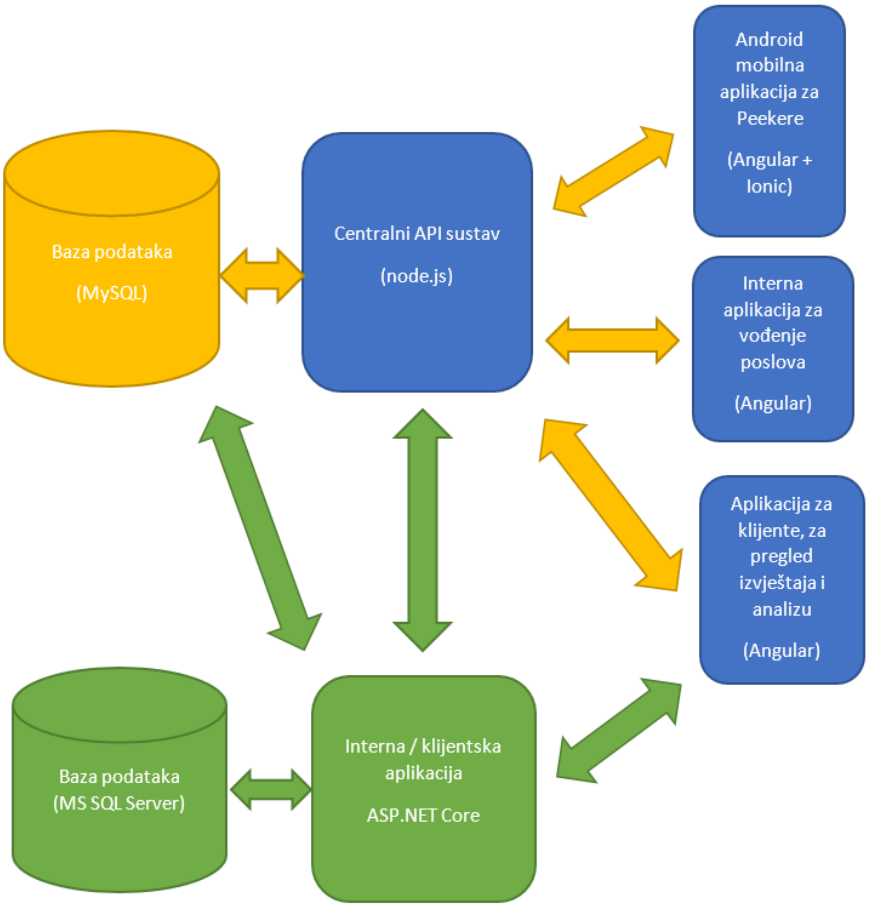
- Terenske ankete
- Dubinski intervjui
- Analiza društvenih mreža
- Izvještaj tajnih kupaca
- Telefonske ankete
- QR kod

Peekator ima svoju softversku platformu koja se sastoji od centralnog API sustava, mobilne aplikacije za zaposlenike tvrtke, interne aplikacije za vođenje poslova te aplikacije za klijente. Klijentska aplikacija klijentima omogućuje prikaz rezultata provedenih istraživanja te analizu i segmentiranje dobivenih podataka. Tijekom poslovanja pokazali su se neki problemi u Peekator platformi:

- Nestabilnost mobilne aplikacije
- Nestabilnost centralnog sustava
- Nefleksibilnost sustava da podrži nove zahtjeve klijenata (nove tipove izvještaja, nove tipove prikaza rezultata, nove tipove pitanja)
- Loša struktura baze podataka koja ne može dobro zapisati sve zahtijevane slučajeve

U svrhu rješavanja navedenih problema započet je projekt izgradnje nove softverske platforme. Kako se trenutno postojeća platforma koristi za tekuće poslovanje, nije moguće odmah prijeći na novu platformu, nego će prelazak biti postepeni te se ona razvija u modulima. Ova aplikacija predstavlja prvi modul toga projekta. Cilj je bio izgraditi novu internu/klijentsku aplikaciju koristeći Blazor Server-side *framework* za razvoj *web* aplikacija. Aplikacija će se koristiti za prikaz novog proizvoda Peekatora, a to je CX Market Research Report. Radi se o sveobuhvatnom istraživanju različitih tržišta različitim kanalima gdje svaki klijent može kupiti samo određene module. Slika (Slika 3.1.) prikazuje shemu novog modula Peekator softverske platforme, te kako se on uklapa u stari sustav. Osim interne/klijentske aplikacije, novi modul uključuje i MS SQL Server bazu podataka koja će se koristiti za pohranu podataka prikupljenih kroz analizu društvenih mreža. Uz podatke

istraživanja, ova baza također sadrži podatke za prijavu korisnika u aplikaciju uključujući njihove role s obzirom na to da je sustav napravljen za korisnike s različitim pravima pristupa. Također, MS SQL Server baza podataka je namijenjena za spremanje svih podataka koje koriste nove funkcionalnosti unutar aplikacije. Osim te baze, ova aplikacija koristi MySQL bazu podataka u koju se pohranjuju podaci vezani uz terenske ankete. Unutar ove sheme preostaju stari dijelovi softverske platforme. Centralni API sustav je API sustav koji komunicira s tri aplikacije unutar stare platforme i zapravo obavlja svu programsku logiku. Drugi dio posla koji obavlja centralni API odnosi se na zapis podataka u MySQL bazu koje spomenute tri aplikacije unose u sustav. Prva aplikacija je mobilna aplikacija koju koriste Peekator zaposlenici kako bi se prijavili na poslove i obavljali ih. Internu aplikaciju također koriste zaposlenici kako bi određivali svakodnevne poslove u vidu definicije klijenata i klijentskih korisničkih imena, definicije klijentske organizacijske strukture (uredi, područja, regije i drugo), definicije poslova, pregleda poslova u tijeku, praćenja obavljanja poslova te analize rezultata. Aplikacija za klijente je mjesto gdje klijenti mogu nakon prijave vidjeti poslove koje su naručili, provedbe u tijeku te prikaz rezultata koje mogu segmentirati.



Slika 3.1. Shema novog modula Peekator softverske platforme

4. Izrada aplikacije

4.1. Arhitektura sustava

Kako prikazuje slika (Slika 3.1.) ovaj sustav predstavlja prvi modul nadograđene Peekator softverske platforme. Glavni fokus je interna/klijentska aplikacija koja, koristeći Entity Framework, se povezuje s dvije baze podataka: MS SQL Server i MySQL bazom podataka. Aplikacija je podijeljena na više zasebnih projekata koji zajedno tvore rješenje (engl. *solution).* Spomenuti projekti su sljedeći:

- Commons
- Interfaces
- Map
- Peekator
- Peekator10Compatibility
- PeekatorDataLayer
- SocialListeningDataLayer
- Services
- ViewModels

Odabran je takav pristup kako bi se odvojila odgovornost unutar *solutiona* tako da svaki projekt odgovara za strogo određene dijelove aplikacije. Commons je biblioteka klasa koja sadržava klase koje se mogu koristiti kroz cijeli *solution* te nemaju strogo definiranu pripadnost samo jednom od projekata, poput enuma i klase za autorizacijske police (engl. *authorization policy*). Interfaces je projekt unutar kojega se spremaju sva sučelja (engl. *interface*) korištena u aplikaciji. Map je projekt posebno rezerviran za AutoMapper biblioteku te sadrži klasu u kojoj su definirana sva mapiranja koja ta biblioteka koristi unutar aplikacije. Peekator je projekt unutar kojeg su sadržane su sve komponente i stranice aplikacije koje skupa izrađuju korisničko sučelje. Dobra praksa pri izradi Blazor aplikacija jest odvojiti komponente i stranice u zasebne datoteke te je zato takav pristup zastupljen i u ovoj aplikaciji. Blazor je *framework* na jednoj stranici (engl. *single-page framework*) te stoga ova aplikacija, za razliku od aplikacija izrađenih na MVC arhitekturi, ne odvaja svoj prezentacijski od poslovnog sloja. Sukladno tome, komponente koje grade korisničko sučelje ujedno sadrže i poslovnu logiku koja definira funkcionalnosti aplikacije.

Peekator10Compatibility je projekt koji predstavlja poveznicu s MySQL bazom podataka. Sadrži kontekstnu klasu (engl. *context class*) koju Entity Framework koristi kako bi izvodio CRUD operacije nad bazom podataka. Unutar *context class* su definirane tablice kojima Entity Framework može pristupiti u bazi. Spomenute tablice su definirane kao klase unutar istog projekta. Svaka klasa predstavlja jedan model ekvivalentan jednoj tablici unutar MySQL baze podataka. PeekatorDataLayer i SocialListeningDataLayer projekti su poveznice sa MS SQL Server bazom podataka. Jedina razlika je u tome što izvode CRUD operacije nad različitim tablicama unutar baze podataka. Services je projekt koji služi kao sloj pristupa podacima. Sadrži servise koji, između ostalog, koristeći Entity Framework pristupaju podacima i dobavljaju ih iz podatkovnog sloja, odnosno baza podataka. ViewModels je biblioteka klasa koja sadrži modele ekvivalentne modelima koji predstavljaju tablice iz baza podataka, no s razlikom da su namijenjeni za prikaz unutar komponenti i stranica, što znači da sadrže samo svojstva potrebna za prikaz u pogledu.

4.2. Funkcionalnosti

Funkcionalnosti aplikacije su podijeljene prema tipu korisnika koji će ih utilizirati. Aplikacija je namijenjena za uporabu od strane klijenata te u isto vrijeme internih zaposlenika tvrtke Peekator te stoga neke od funkcionalnosti su namijenjene za klijente, neke za zaposlenike, dok neke će koristiti oba tipa korisnika, uz različite mogućnosti koje će biti naznačene za svaku funkcionalnost.

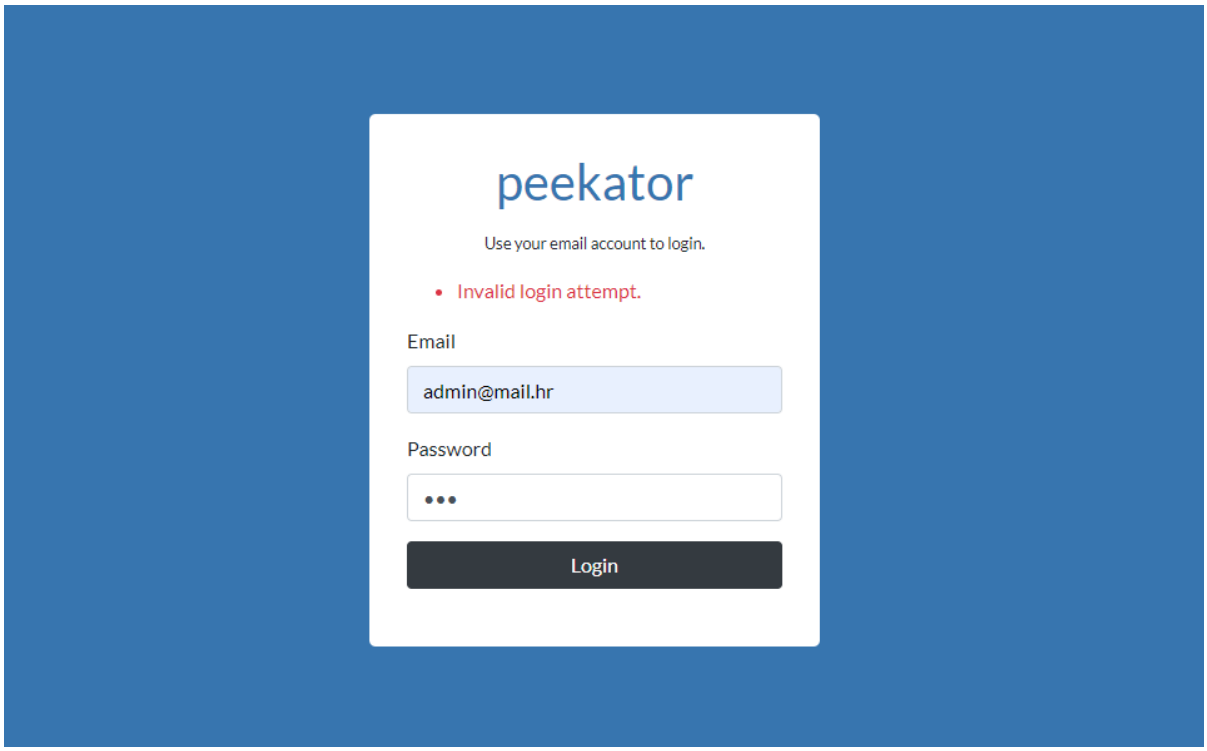
4.2.1. Autentifikacija korisnika

Kako će aplikacija u komercijalnoj upotrebi raspolagati povjerljivim podacima kojima smiju pristupiti samo ovlaštene osobe potreban je sustav za prijavu. Za ovu aplikaciju je korišten ASP.NET Identity API te njegova ugrađena funkcionalnost za prijavu u sustav. Prije nego se može koristiti API za prijavu treba se omogućiti autentifikacija unutar `ConfigureServices` metode u `Startup` klasi, kao što je to vidljivo u sljedećem isječku koda:

```
services.AddAuthentication("Identity.Application")
                .AddCookie();
```

Metoda `AddCookie()` naznačuje da je riječ o autentifikaciji na temelju *cookieja*. Dodatno je moguće unutar te metode definirati koja se autentifikacijska shema prosljeđuje, no zadana

vrijednost je `CookieAuthenticationDefaults.AuthenticationScheme` te stoga to nije potrebno posebno naznačiti s obzirom na činjenicu da je to shema korištena u ovoj aplikaciji. Prilikom prijave korisnik mora unijeti ispravne podatke ili u suprotnom mu pristup aplikaciji neće biti omogućen. Slika (Slika 4.1.) prikazuje ekran za prijavu te kako izgleda neuspješan pokušaj prijave te greška koja se pojavi pri takvom pokušaju.



Slika 4.1. Neuspješan pokušaj prijave na ekranu za prijavu

Nakon uspješne prijave, biti će prikazan glavni ekran aplikacije s vertikalnim izbornikom s lijeve strane ekrana pomoću kojeg se može pristupiti različitim funkcionalnostima koje aplikacija nudi, kao što je vidljivo na slici (Slika 4.2.). Kada se korisnik prijavi u sustav aplikacija provjerava da li je riječ o klijentu ili zaposleniku te u skladu s time izbornik prikazuje gumbove, odnosno navigacijske poveznice prema funkcionalnostima kojima taj korisnik ima prava pristupa. Svaki od tih gumbova unutar izbornika predstavlja Blazorovu `NavLink` komponentu koja sadrži `href` atribut s rutom do određene stranice. Mehanizam koji definira autorizacijska prava korisnika se naziva *authorization policy*.

```
public static class Policies
{
        public const string IsPeekatorEmployee = "IsPeekatorEmployee";
        public const string IsClientEmployee = "IsClientEmployee";
        public static AuthorizationPolicy IsPeekatorEmployeePolicy()
        {
```

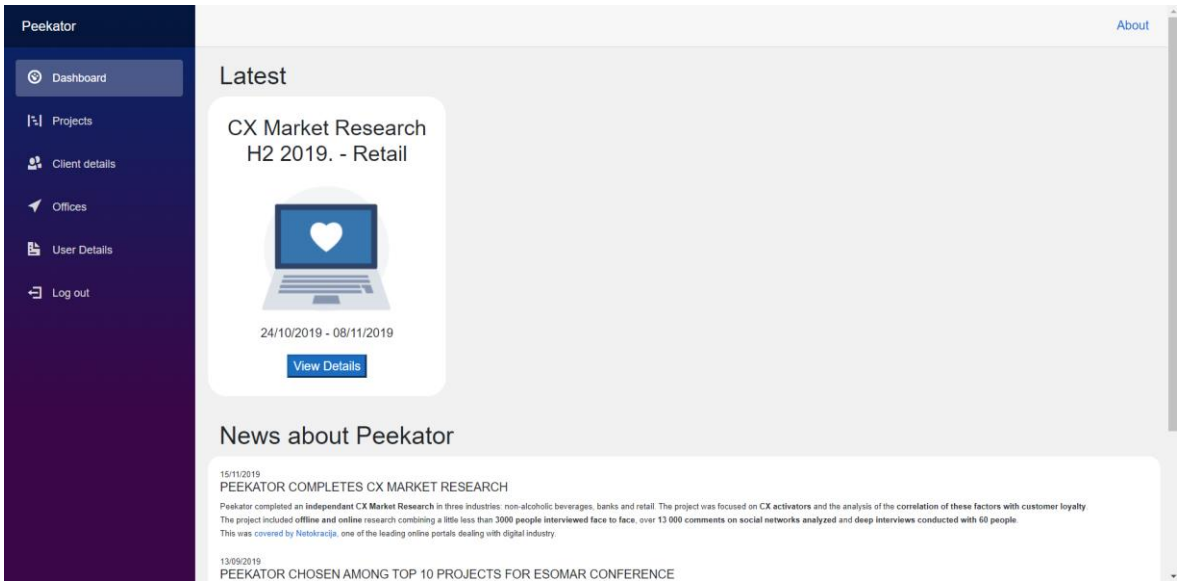
```
            return new AuthorizationPolicyBuilder()
                  .RequireAuthenticatedUser()
                  .RequireClaim("PeekatorEmployee", "true")
                  .Build();
        }
    }
```

Kod 4.1. Klasa koja definira *authorization policy*

Kod (Kod 4.1.) prikazuje klasu unutar koje je definiran *authorization policy* za ovu aplikaciju. Nakon što *je authorization policy* kreiran, treba omogućiti aplikaciji njegovo korištenje. Autorizaciju i *authorization policy* je, jednako poput autentifikacije, potrebno omogućiti unutar `ConfigureServices` metode u `Startup` klasi, kao što to prikazuje sljedeći isječak koda:

```
services.AddAuthorization(options =>
{
    options.AddPolicy(Policies.IsPeekatorEmployee,
    Policies.IsPeekatorEmployeePolicy());
});
```

Glavni izbornik (Slika 4.2.) sadrži, osim `NavLink` komponenti koje usmjeravaju prema određenim funkcionalnostima, `NavLink` za odjavu korisnika.

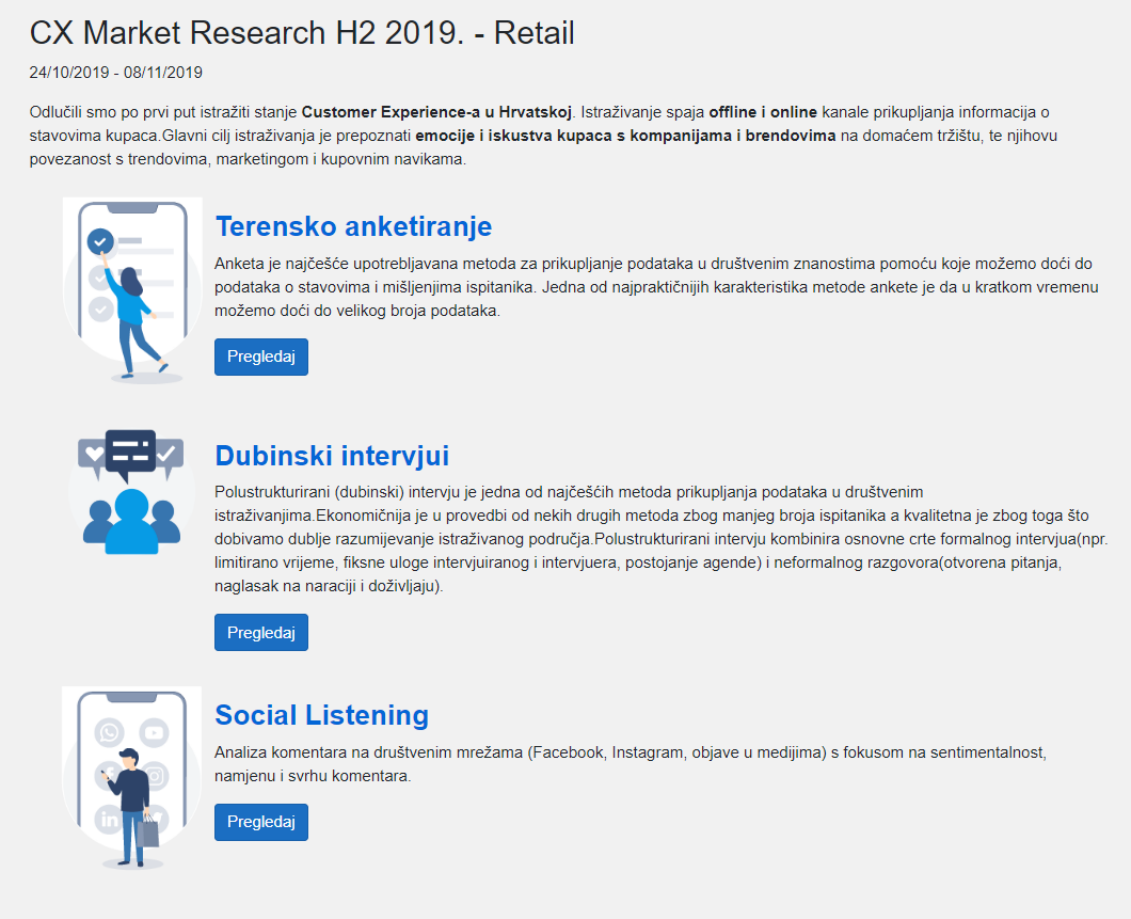


Slika 4.2. Početni ekran aplikacije s navigacijskim izbornikom

U gornjem desnom kutu stranice se nalazi poveznica (engl. *link*) „About“ koja vodi na službenu Peekatorovu stranicu. Nakon prijave, prvi ekran koji se prikaže klijentu je glavna kontrolna ploča. Na toj kontrolnoj ploči su prikazani svi projekti koji su dostupni tome

klijentu. Ispod svih navedenih projekata se nalazi sekcija koja sadrži vijesti vezane uz Peekator platformu. Nakon što klijent odabere jedan od projekata, biti će mu prikazan glavni ekran tog projekta s njegovom definicijom (Slika 4.3.). Ispod definicije popisana su sva istraživanja unutar tog projekta. Postoje tri različita tipa istraživanja, a to su sljedeća:

- Terensko anketiranje
- Dubinski intervjui
- Analiza društvenih mreža

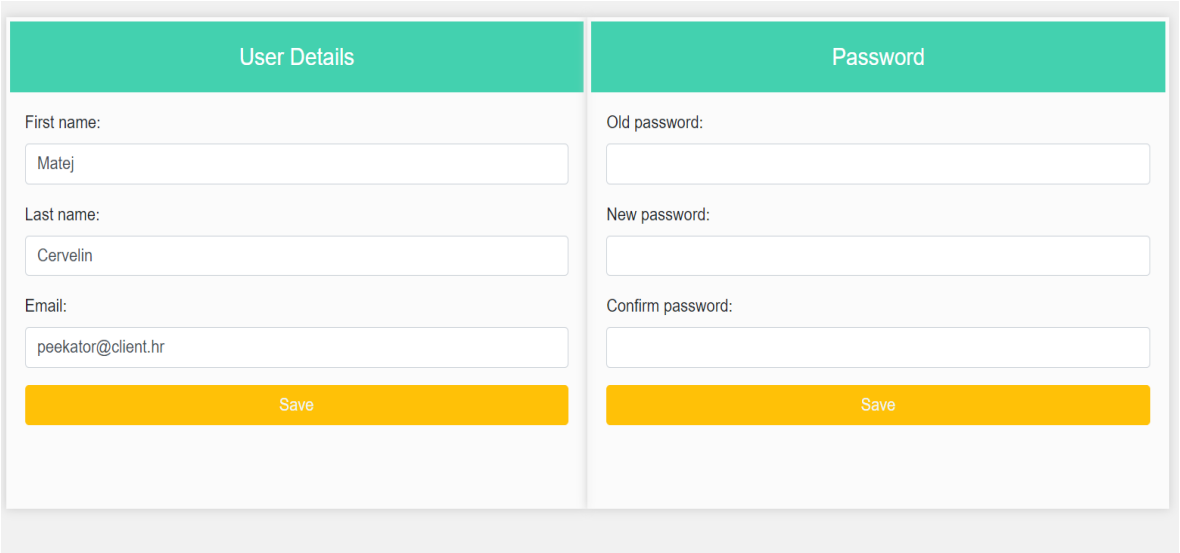


Slika 4.3. Ekran sa svim istraživanjima projekta

4.2.2. Moj profil

Svaki korisnik ima mogućnost izmijeniti osnovne podatke vezane uz svoj profil (Slika 4.4.). Stranica za izmjenu korisničkih podataka je podijeljena na dvije komponente. Unutar prve komponente korisnik može urediti svoje ime, prezime te adresu elektronske pošte (engl. *email*). Unutar druge komponente korisnik može izmijeniti svoju lozinku. Nakon što korisnik učita stranicu, aplikacija automatski dobavi podatke o trenutno prijavljenom

korisniku, te ispuni polja za ime, prezime i *email*. Sva polja su obvezna, a polje za *email* ima dodatnu validaciju kako bi korisnik unio ispravnu *email* adresu. U suprotnom slučaju, aplikacija neće prihvatiti promjene te će se pojaviti poruka upozorenja o krivo unesenim podacima. Aplikacija prihvaća unos promjena tek nakon što su sva polja ispunjena te svi podaci ispravno upisani.



Slika 4.4. Ekran za uređivanje osobnih podataka korisnika

Jednako kao za izmjenu *email* adrese, korisnik mora zadovoljiti određene uvjete kako bi promijenio svoju lozinku. Aplikacija prvo provjerava da li je upisana ispravna stara lozinka (Kod 4.2.).

```
public async Task<bool> VerifyOldPassword(string oldPassword)
{
        var currentUser = await GetCurrentUserAsync();

        var result =
        _userManager.PasswordHasher.VerifyHashedPassword(currentUser,
        currentUser.PasswordHash, oldPassword);

        if (result == PasswordVerificationResult.Success)
        {
           return true;
        }
        else
        {
           return false;
        }
```

```
    }
```

Kod 4.2. Metoda za verifikaciju unesene stare lozinke

Navedena metoda (Kod 4.2.) funkcionira na način da uzme proslijeđenu staru lozinku koju je korisnik unio te je usporedi s korisnikovom lozinkom spremljenom u bazi podataka. Ovisno o tome da li proces validacije bude uspješan, metoda vraća *true* ili *false* vrijednost. Validacija unesene stare lozinke predstavlja prvi uvjet za izmjenu lozinke, no mora se zadovoljiti i drugi uvjet. Osim unesene stare lozinke, validacija se također vrši i nad novom lozinkom koju korisnik unese. Nova lozinka mora imati minimalno 6 i maksimalno 50 karaktera. Također, lozinka mora sadržavati najmanje jedno malo slovo, jedno veliko slovo, jedan broj te jedan poseban znak. Pravila validacije su definirana koristeći anotacije u modelu prikaza (engl. *view model*) za lozinke.

```
public class UserPasswordViewModel
{
      [Required(ErrorMessage = "Old password is required")]
      [Display(Name = "Old password")]
       public string OldPassword { get; set; }

      [Required(ErrorMessage = "New password is required")]
      [DataType(DataType.Password)]
      [StringLength(50, ErrorMessage = "The new password must be at least
      {2} and at max {1} characters long.", MinimumLength = 6)]
      [RegularExpression(@"^(?=.*[a-z])(?=.*[A-
      Z])(?=.*\d)(?=.*[\W]).{8,}$",ErrorMessage = "Password must contain
      at least one upper case character, one lower case character, one
      number and one special character")]
      [Display(Name = "New password")]
      public string NewPassword { get; set; }

      [Required(ErrorMessage = "Password confirm is required")]
      [DataType(DataType.Password)]
      [Compare("NewPassword", ErrorMessage = "The password and
      confirmation password do not match.")]
      [Display(Name = "Confirm password")]
      public string ConfirmNewPassword { get; set; }
}
```

Kod 4.3. *View model* s validacijskim pravilima za korisničke lozinke

Ako unesena stara lozinka i nova lozinka zadovoljavaju sve uvjete validacije, onda slijedi korak prevođenja nove lozinke u *hash* oblik. Lozinka se sprema u takvom obliku iz sigurnosnih razloga kako bi bila neprepoznatljiva bilo kome tko ju pokuša ukrasti. *Hash* oblik lozinke dobije se šifriranjem lozinke koristeći matematički algoritam. Navedeni proces je ugrađeni mehanizam ASP.NET Identity API-ja te ga stoga nije potrebno dodatno implementirati. Nakon što je lozinka prevedena u *hash* oblik, slijedi zapis lozinke u bazu podataka, pri čemu se stara lozinka briše (Kod 4.4.).

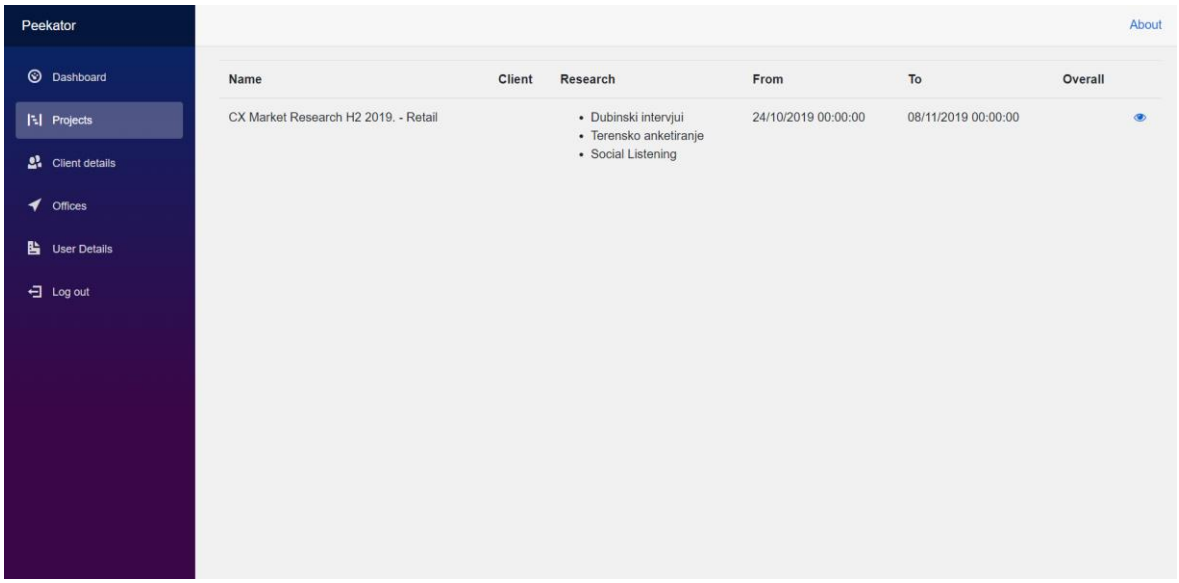
```
public async Task ChangePasswordForUser(string newHashedPassword)
{
        var userToUpdate = await GetCurrentUserAsync();
        userToUpdate.PasswordHash = newHashedPassword;

        await _userManager.UpdateAsync(userToUpdate);
}
```

Kod 4.4. Metoda za zapis izmijenjene lozinke u bazu podataka

4.2.3. Prikaz projekata

Stranica projekata je stranica na kojoj korisnici mogu vidjeti pregled svih projekata u sustavu (Slika 4.5.). Ovisno o autorizacijskim pravima korisnika prikazati će se različiti projekti.



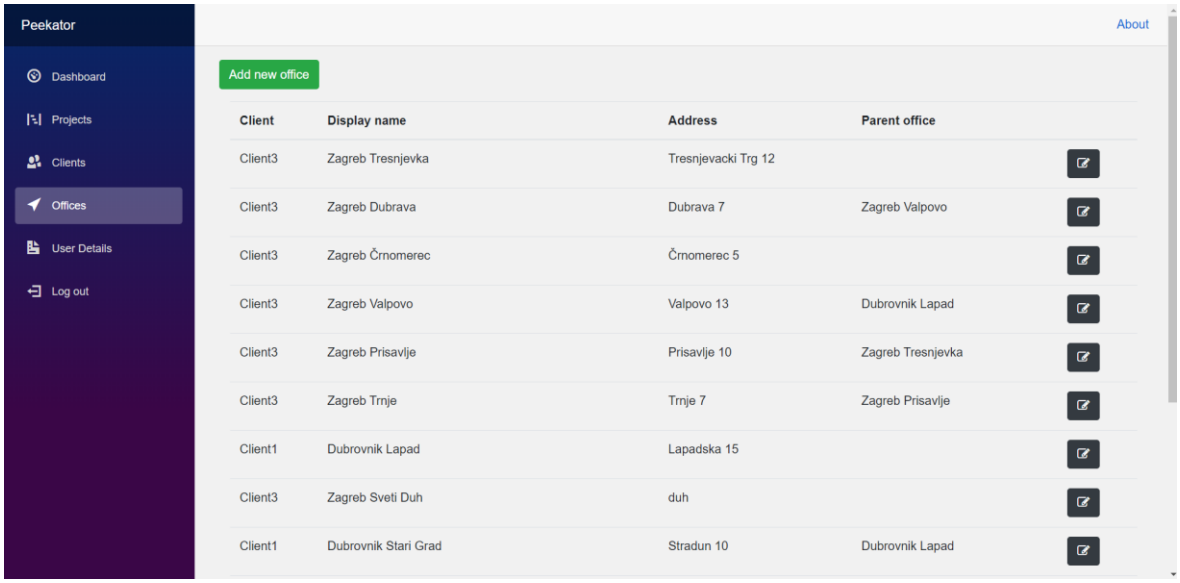
Slika 4.5. Prikaz projekata

U slučaju da je korisnik zaposlenik Peekatora, biti će prikazani svi projekti. U suprotnom slučaju, ako je korisnik klijent, biti će mu prikazani samo projekti koji pripadaju tome

klijentu kako bi se zaštitili podaci između različitih klijenata. Za svaki projekt definirani su osnovni podaci kao ime, klijent koji je vlasnik projekta, istraživanja koja spadaju pod taj projekt te vremenski period trajanja projekta. Za svaki projekt u desnoj strani retka se nalazi ikonica oka. Klikom na tu ikonicu aplikacija preusmjerava korisnika na glavni ekran odabranog projekta s njegovom definicijom i istraživanjima.

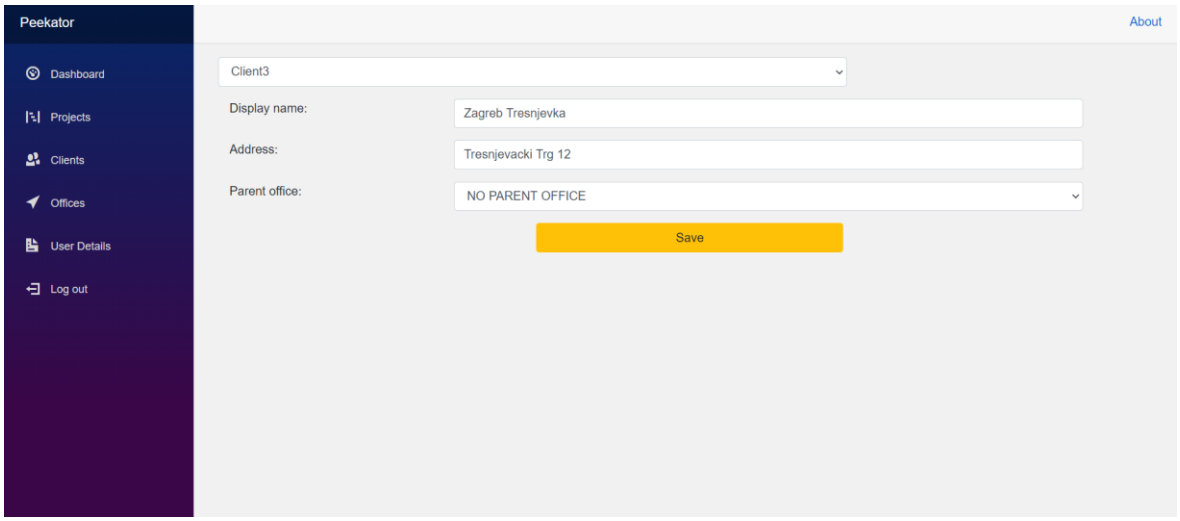
4.2.4. Ažuriranje ureda klijenata

Svaki klijent ima više ureda čije je podatke potrebno moći ažurirati. Svaki ured je definiran klijentom koji je vlasnik ureda, imenom za prikaz te adresom (Slika 4.6.).



Slika 4.6. Prikaz ureda

Za svaki ured dodatno može biti definiran i njegov roditeljski ured (engl. *parent office).* Stranica za prikaz ureda prezentira različit sadržaj ovisno o tipu prijavljenog korisnika. Ako je trenutno prijavljen zaposlenik Peekatora, biti će prikazani svi uredi svih klijenata. S druge strane, ako je u sustav prijavljen klijent, biti će prikazani samo uredi koji pripadaju tome klijentu. Osim toga, za klijenta neće biti prikazana kolumna s imenom klijenta kojem pripada ured, iz razloga što će tome klijentu biti prikazani samo njegovi uredi tako da je nepotrebno istaknuti za svaki ured vlasničkog klijenta. Osim pregleda svih ureda, na vrhu stranice se nalazi gumb „Add new office“ za kreiranje novog ureda. Klikom na njega aplikacija preusmjerava korisnika na ekran za dodavanje novog ureda (Slika 4.7.), no taj ekran ima i drugu funkcionalnost.

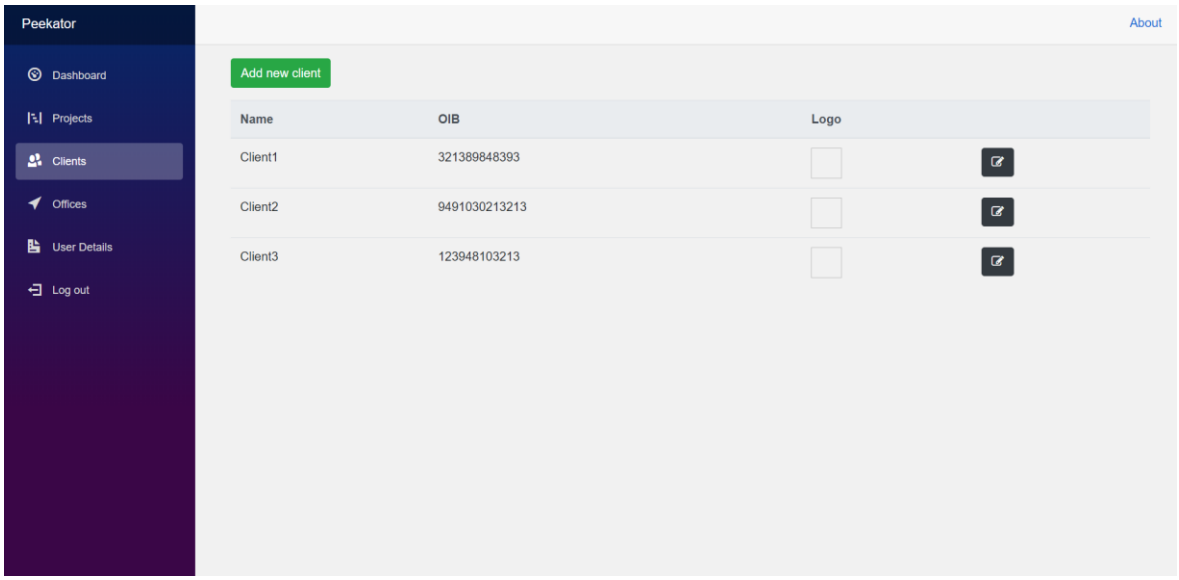


Slika 4.7. Ekran za ažuriranje postojećeg ili dodavanje novog ureda

Osim za dodavanje novog ureda, ekran na slici (Slika 4.7.) služi i za ažuriranje postojećih ureda. Unutar tablice ureda za svaki ured postoji ikona u posljednjem stupcu koja korisnika preusmjerava na ekran za ažuriranje odabranog ureda. Za svaki ured je moguće izmijeniti njegov naziv, adresu te *parent office*. Za izmjenu *parent officea* postoje dodatna pravila koja je važno napomenuti. Ako postoje uredi kojima je odabrani ured *parent office*, onda niti jedan od njih se ne može postaviti kao njegov *parent office*. Aplikacija automatski vrši tu provjeru tako da se takvi uredi niti ne pojavljuju u padajućem izborniku za odabir *parent officea*. Sljedeće pravilo se odnosi samo na zaposlenike Peekatora. Ako je prijavljeni korisnik zaposlenik Peekatora, onda ima dodatnu opciju da promijeni klijenta za odabrani ured. Pri odabiru klijenta aplikacija dobavi svaki *parent office* koji pripada tome klijentu i ujedno zadovoljava prvo pravilo te onda napuni *parent office* padajući izbornik s dobavljenim uredima. *Parent office* nije nužno odabrati te stoga moguće je postaviti da neki ured nema svoj *parent office*. U slučaju da je prijavljeni korisnik klijent, onda se automatski dobave samo uredi koji pripadaju tome klijentu.

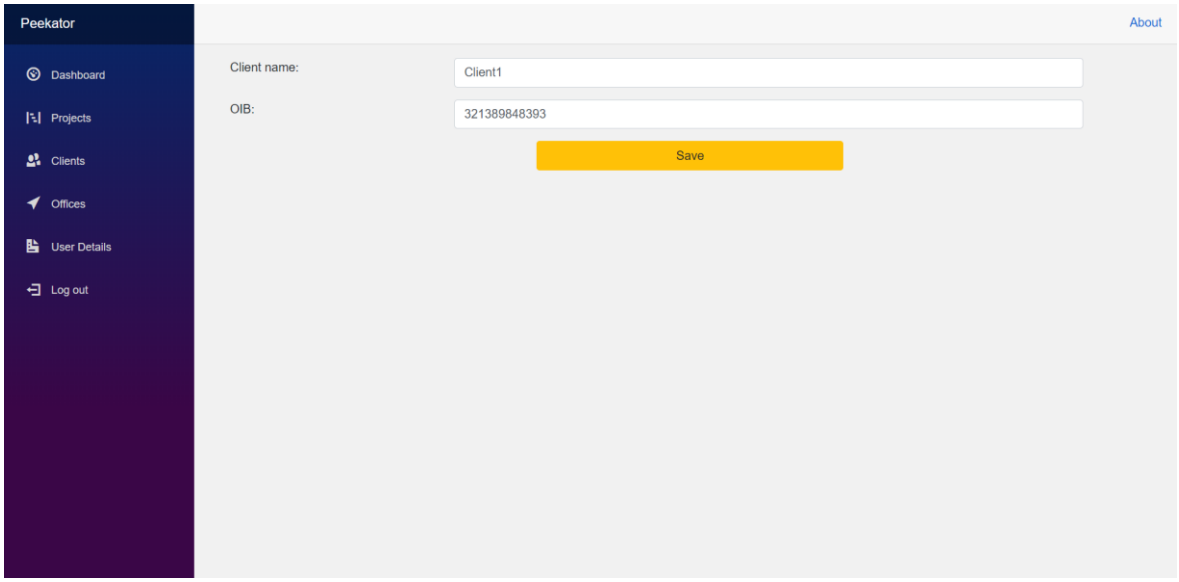
4.2.5. Ažuriranje klijenata

Stranica za prikaz klijenata sadrži tablični pregled klijenata i njihovih osnovnih podataka (Slika 4.8.). U slučaju da aplikaciju koristi zaposlenik Peekatora, u tablici će biti prikazani svi klijenti i njihovi podaci. Unutar tablice je prikazano ime klijenta, njegov OIB i logo te gumb koji preusmjerava korisnik na ekran za uređivanje podataka.



Slika 4.8. Prikaz klijenata

Zaposlenik ima dodatni gumb „Add new client“ iznad tablice koji ga preusmjerava na ekran za dodavanje novog klijenta u sustav (Slika 4.9.). Ako je prijavljeni korisnik klijent, onda će biti prikazani samo njegovi podaci te on nema mogućnost dodavanja novog klijenta u sustav.



Slika 4.9. Ekran za dodavanje novog ili ažuriranje postojećeg klijenta

4.2.6. Prikaz izvještaja terenskog anketiranja

Izvještaj terenskog anketiranja prikazuje rezultate provedenog istraživanja u obliku pitanja i dobivenih odgovora. Svako pitanje spada pod određenu kategoriju te shodno tome sami izvještaj je podijeljen na kategorije. Na vrhu izvještaja su prvo navedeni detalji istraživanja

kao npr. broj ispitanika, razdoblje prikupljanja, lokacija i slično. Pitanja se dijele prema mogućim odgovorima. Tipovi pitanja prema mogućim odgovorima su sljedeći:

- Pitanja s jednim mogućim odgovorom
- Pitanja s više mogućih odgovora
- Pitanja s brojevnim odgovorom
- Pitanja s tekstualnim odgovorom

Pitanja su izrađena koristeći Razor komponente. Prva dva tipa pitanja su izrađena koristeći istu komponentu pod nazivom `CapiComponent`, no s nekoliko bitnih razlika koje su navedene u nastavku teksta. Preostala dva tipa pitanja su izrađena koristeći zasebne komponente. Kako bi se definirale razlike unutar `CapiComponent` komponente, koriste se *if* iskazi. Ponovna iskoristivost je velika odlika Razor komponenti te se stoga to svojstvo često utilizira kroz ovaj projekt. Jedan od primjera je komponenta `TextComponent` koja se koristi unutar dvije druge komponente. Isječak koda (Kod 4.5.) prikazuje *if* iskaz unutar `CapiComponent` komponente. Taj iskaz kao svoj uvjet provjerava istinito stanje `IsNumbered` svojstva koje pripada `Question` objektu. `Question` je objekt klase `StandardQuestionViewModel` koja služi kao model za sve tipove pitanja ovog istraživanja. `IsNumbered` je svojstvo na temelju kojeg aplikacija prepoznaje o kojem je tipu pitanja riječ. U slučaju da je to svojstvo istinito, odnosno da ima vrijednost *true,* onda je riječ o prvom tipu pitanja, odnosno pitanjima s jednim mogućim odgovorom. U suprotnom slučaju, ako ima vrijednost *false*, onda je riječ o drugom tipu pitanja, odnosno njegovom podtipu koji sadrži i „ostale“ odgovore. U ovome slučaju se dodatno prikazuje komponenta `TextComponent` koja prikazuje „ostale“ odgovore.

```
@if (!Question.IsNumbered)
{
<br />
<TextComponent CollapseDivId="Question.Id"
Answers="Question.Answers.Where(a =>
a.IsText).ToList()">Drugo: </TextComponent>
}
```

Kod 4.5. *If* iskaz za određivanje tipa pitanja

Pitanja s tekstualnim odgovorom su drugi tip pitanja koji koristi `TextComponent` komponentu za prikaz odgovora. Kod (Kod 4.6.) prikazuje uporabu `TextComponent` komponente za prikaz pitanja s tekstualnim odgovorom. Za razliku od pitanja s više mogućih

odgovora, u ovom slučaju se `TextComponent` komponenta koristi na način da ispisuje i redni broj pitanja, te sami tekst pitanja.

```
<TextComponent CollapseDivId="Question.Id" Answers="Question.Answers">
     @QuestionOrdinal. @Question.QuestionText
</TextComponent>
```

Kod 4.6. Uporaba `TextComponent` komponente za prikaz tekstualnih odgovora

Izvještaj terenskog anketiranja, kao i druga dva istraživanja koja ova aplikacija prikazuje, je detaljnije opisan te vizualno prikazan u poglavlju „Analiza i vizualizacija“.

4.2.7. Prikaz analize društvenih mreža

Ovaj ekran služi za prikaz rezultata istraživanja analize društvenih mreža. Izvještaj prikazuje kategoriziran tablični pregled prikupljenih podataka koji se može filtrirati prema sljedećim parametrima:

- Kategorija komentara
- Tip komentara
- Sentiment komentara
- Dob
- Spol

Kod (Kod 4.7.) prikazuje `OnInitializedAsync` metodu koja se priziva kada je komponenta spremna za prikaz. Unutar navedene metode se instanciraju svi parametri pomoću kojih korisnik može prilagoditi podatke svojim potrebama za analizu. Podaci se dohvaćaju koristeći `SocialListeningService` servis.

```
protected override async Task OnInitializedAsync()
{
 _slService = new SocialListeningService(Ctx);
 AgeGroups = await _slService.GetAgeGroups();
 Sentiments = await _slService.GetSentiments();
 CommentCategories = await _slService.GetCommentCategories();
 CommentTypes = await _slService.GetCommentTypes();
 Data = await _slService.GetDataForSocialListening(ResearchId, _filters);
 await base.OnInitializedAsync();
}
```

Kod 4.7. Instanciranje parametara pri učitavanju stranice

Jednako kao za izvještaj terenskog anketiranja, na početku ovog izvještaja se prvo navode detalji istraživanja. Za svaki od prethodno navedenih parametara se izgenerira tablica koja prikazuje analizu objava, komentara i komentatora pojedinih objava s obzirom na različite opcije parametara. Unutar svake tablice su prikazani podaci klijenta i svih konkurenata kako bi klijent mogao kvalitetno proučiti i usporediti iskustva komentatora na vlastitim profilima društvenih mreža i onim od njihovih konkurenata. Tablice prikazuju broj objava za svaku opciju nekog parametra. Podaci iz prve dvije tablice dodatno su vizualno prikazani grafovima. Nakon što korisnik odabere jednu od opcija određenog parametra, sve tablice unutar izvještaja se filtriraju prema navedenoj opciji parametra.

```
public async Task FilterChanged(ChangeEventArgs e, string senderType)
{
        UpdateFilters(e, senderType);

        IsMessageOn = true;
        Message = "Refreshing data, please wait...";
        StateHasChanged();
        await RefreshData();
}
```

Kod 4.8. Metoda koja se okida pri promjeni opcije određenog parametra

Kod (Kod 4.8.) prikazuje metodu koja se okida svaki puta kada korisnik promjeni opciju bilo kojeg parametra. Aplikacija prikaže poruku da se podaci osvježavaju, te nakon toga obavijesti komponentu da se promijenilo njeno stanje te da je potrebno prikazati novo stanje komponente. Metoda tada učitava nove podatke, odnosno filtrira trenutne podatke na osnovu odabrane opcije parametra. Osim analize tablica prema parametrima, korisnik na kraju izvještaja ima opciju odabrati jedan od subjekata istraživanja iz padajućeg izbornika, čime dobiva pregled detalja o svim objavama toga subjekta te ima opciju prikazati komentare za bilo koju od objava. Metoda unutar koda (Kod 4.9.) provjerava ako je odabran subjekt čiji jedinstveni identifikator nije prazan *string*. Nakon te provjere, koristeći `SocialListeningService` servis, metoda dobavlja sve objave za odabranog subjekta te popunjava tablicu i obavještava komponentu da se promijenilo njeno stanje.

```
public async Task FillPostsTable(ChangeEventArgs e)
{
   _slService = new SocialListeningService(Ctx);

   SubjectSelectedId = new Guid(e.Value.ToString());
```

```
    if (SubjectSelectedId != Guid.Empty)
    {
        Posts = await _slService.GetPostsForSubject(SubjectSelectedId);
        CommentViewModels = null;

        PostViewModels = Mapper.Map<List<PostViewModel>>(Posts);
    }
        StateHasChanged();
}
```

Kod 4.9. Metoda za popunjavanje tablice s detaljima objava odabranog subjekta

4.2.8. Prikaz izvještaja dubinskog intervjua

Ovaj ekran sadrži ugrađeni dokument koji predstavlja izvještaj dubinskog intervjua. Kako bi se izvještaj ugradio u stranicu koristi se element `embed`, kao što je to vidljivo u kodu (Kod 4.10.). *If* iskaz unutar ove komponente provjerava ako je proslijeđen string koji sadrži putanju na lokalnom računalu do dokumenta. U slučaju da nije proslijeđena putanja, stranica neće prikazati HTML elemente koji se nalaze unutar petlje. Prije dokumenta su navedeni svi detalji vezani uz istraživanje, kao što je to slučaj i za prikaz ostalih istraživanja.

```
@if (!string.IsNullOrWhiteSpace(Pdf))
{
    <div class="row">
        <div class="col-md-12">
            <embed src="@Pdf" width="100%" height="750px" />
        </div>
    </div>
}
```

Kod 4.10. *If* iskaz koja provjerava da li je proslijeđena putanja dokumenta

4.2.9. Dohvat i priprema (agregacija) svih podataka za prikaz

Kako bi se korisniku aplikacije prikazali svi podaci u izvještajima, podatke je prvo potrebno dohvatiti i pripremiti, odnosno agregirati. U slučaju izvještaja terenskog anketiranja, podaci se dohvaćaju iz MySQL baze podataka pod nazivom „peekator“. Entity Framework Core omogućuje tri obrasca (engl. *pattern*) za pristup podacima, od kojih su za ovu aplikaciju bitni *lazy loading* te *eager loading*. Prije nego što Entity Framework može pristupiti bazi

podataka, potrebno je konfigurirati kontekst baze podataka (engl. `DbContext`) s odgovarajućim konekcijskim *stringom*. `DbContext` klasa sadrži objekte za pristup tablicama unutar određene baze podataka. Konekcijski *string* definira na koju bazu podataka se treba spojiti, te koje je korisničko ime i lozinka za pristup bazi. Moguće je definirati još neke dodatne opcije konekcijskog *stringa* poput *porta*, no generalno su samo bitna prije navedena tri podatka. `DbContext` se konfigurira unutar `IdentityHostingStartup` klase, kao što je to vidljivo u sljedećem isječku koda:

```
services.AddDbContext<peekatorContext>(opts =>
opts.UseMySql(context.Configuration.GetConnectionString("Peek
ator10")));
```

Kao sigurnosna mjera, postavke konekcijskog *stringa* se ne spremaju ovdje, već unutar `User Secrets` skrivenog dokumenta specifično namijenjenog za pohranu takvih podataka. Nakon što je konfiguriran `DbContext`, potrebno je registrirati servis unutar `Startup` klase (Kod 4.11.).

```
services.AddScoped<IAuthService, AuthService>();
services.AddScoped<IProjectService, ProjectService>();
services.AddScoped<INewsService, NewsService>();
services.AddScoped<IResearchService, ResearchService>();
services.AddScoped<IClientService, ClientService>();
services.AddScoped<IOfficeService, OfficeService>();
services.AddScoped<IUserService, UserService>();
services.AddScoped<ICAPIService, CAPIService>();
```

Kod 4.11. *Dependency injection* servisa unutar `Startup` klase

Servis može imati jedan od tri navedena životna vijeka (engl. *lifetime*):

- *Scoped*
- *Singleton*
- *Transient*

S obzirom na zahtjeve ove aplikacije, najprikladniji je *scoped lifetime*. Stoga se u primjeru (Kod 4.11.) koristi `AddScoped` metoda za registraciju servisa. Servis koji ima *scoped lifetime* se kreira jednom po konekciji, što znači da se instancira nanovo svaki put kad se uspostavi nova konekcija. `DbContext` te navedeni servisi su konfigurirani koristeći *dependency injection*. Za servis to znači da nakon što je registriran moguće ga je koristiti unutar neke klase bez prethodnog instanciranja. Kako bi Blazorov davatelj usluga (engl.

service provider) pri inicijalizaciji aplikacije znao da unutar neke klase treba ubaciti (engl. *inject*) servis, potrebno je koristiti anotaciju `Inject`. Pristup „peekator“ bazi podataka obavlja `CAPIService` servis unutar `Services` projekta. Sljedeći isječak koda prikazuje primjer *injectanja* spomenutog servisa unutar `CapiComponentBase` klase:

```
[Inject]
protected CAPIService { get; set; }
```

Aplikacija sada ima instancu servisa te ga se može koristiti za dobavljanje podataka:

```
AllDataViewModel = await
capiService.GetSurveyDataFromOldDb(Id, filter);
```

`CAPIService` servis koristi `peekatorContext` *context class* koja sadrži objekte za pristup svim tablicama „peekator“ baze podataka. Servis pristupa objektima unutar *context class* kroz LINQ upite i na taj način dohvaća podatke iz baze podataka.

```
var questionsQuery = from job in db.Job
          join template in db.Templates on job.Template
equals template.TemplateId
          join template_question in db.TemplateQuestion on
template.TemplateId equals template_question.Template
          join question in db.Questions on
template_question.Question equals question.QuestionId
          where job.JobId == jobId
          select new { Questions = question,
TemplateQuestion = template_question };
```

Kod 4.12. LINQ upit za dohvat podataka

Kod (Kod 4.12.) prikazuje primjer LINQ upita koji dohvaća sva pitanja za trenutno istraživanje. Unutar baze podataka, tablica koja definira istraživanja se zove `job`. Svaki `job` ima svoj jedinstveni `jobId` prema kojemu prethodno navedeni upit filtrira podatke tako da dobavi samo pitanja koja se odnose na trenutni `job`. Ovaj primjer također demonstrira *eager loading pattern* za dohvat podataka. Upit dohvaća podatke iz više tablica odjednom te ih sprema unutar `questionsQuery` varijable. Podaci se učitavaju odmah, a kasnije se kroz varijablu `questionsQuery` može pristupiti svakoj tablici koju je upit dohvatio (Kod 4.13.).

```
List<StandardQuestionViewModel> questions = questionsQuery.Select(c
=> new StandardQuestionViewModel
{
```

```
            Id = c.Questions.QuestionId,
            QuestionText = c.Questions.QuestionText,
            CategoryID = c.TemplateQuestion.Category.Value,
            Type = (QuestionTypeEnum)c.Questions.QuestionType,
            Ordinal = c.Questions.QuestionId,
            IsNumbered = (QuestionTypeEnum)c.Questions.QuestionType  ==
            QuestionTypeEnum.StandardList ? true : false
    }).ToList();
```

Kod 4.13. Primjer spremanje dohvaćenih podataka u listu objekata

Jednako tako, koristeći LINQ upite servis dobavlja odgovore za sva pitanja. Ovisno o tipu pitanja, odgovori se dobavljaju koristeći drukčije metode. Pitanja s jednim mogućim odgovorom imaju određenu „težinu“ koja predstavlja maksimalni broj bodova koji se može postići odgovorom na to pitanje. Jednako tome, svaki ponuđeni odgovor na to pitanje ima „težinu“, koja predstavlja postignuti broj bodova. Kako bi se izračunao postotak kategorije, aplikacija kalkulira ukupni broj postignutih bodova unutar kategorije i uspoređuje ga s maksimalnim mogućim brojem bodova.

Za prikaz analize društvenih mreža podaci se dohvaćaju iz MS SQL Server „Peekator“ baze podataka. Pri izradi MS SQL Server baze podataka korišten je *Code First* pristup, što znači da su prvo izrađeni modeli na osnovu kojih je izgenerirana baza podataka koristeći migracije. Migracijske klase definiraju koje sve tablice trebaju biti sadržane u bazi podataka, koje su relacije između tablica te također moguće je definirati neke podatke koji će biti spremljeni. Pri inicijalnoj izradi modela te svaki put kada se rade izmjene na modelima unutar projekta, potrebno je izgenerirati novu migracijsku klasu te nakon toga ažurirati bazu podataka s promjenama iz najnovije migracijske klase. Za generiranje migracijske klase, potrebno je izvršiti sljedeću komandu unutar *Package Manager* konzole: `Add-Migration`. Nakon toga sve promjene se ažuriraju u bazu podataka izvršavanjem komande `Update-Database` unutar konzole. Kako bi se moglo pristupiti bazi podataka, potrebno je konfigurirati `DbContext`.

```
            services.AddDbContext<PeekatorContext>(options =>
                 options.UseSqlServer(
            context.Configuration.GetConnectionString("PeekatorAuthentica
            tionContextConnection")));

            services.AddDbContext<SocialListeningContext>(options =>
```

```
{
options.UseSqlServer(context.Configuration.GetConnectionStrin
g("SocialListeningContextConnection"),
sqlServerOptions => sqlServerOptions.CommandTimeout(60));
});
```

Kod 4.14. Registracija `DbContexta` za pristup MS SQL Server bazi podataka

Kod (Kod 4.14.) prikazuje konfiguraciju dva `DbContexta` za pristup MS SQL Server bazi podataka. `SocialListeningContext` klasa sadrži objekte za pristup tablicama koje se odnose na prikaz analize društvenih mreža, dok `PeekatorContext` klasa sadrži objekte za pristup tablicama koje se odnose na preostale funkcionalnosti. Aplikacija koristi `SocialListeningService` servis za dohvat svih podataka potrebnih za prikaz analize društvenih mreža. Sljedeći isječak koda prikazuje jedan primjer metode za dohvat podataka u kojoj se dobavljaju podaci o parametru dobne skupine:

```
public async Task<List<AgeGroup>> GetAgeGroups()
{
    return await _ctx.AgeGroups.OrderBy(a => a.From).ToListAsync();
}
```

Navedeni kod također demonstrira *lazy loading pattern* pristupa podacima. Metoda pristupa svojstvima objekta `AgeGroups` unutar `SocialListeningContext` klase. Za razliku od *eager loading patterna*, prema ovom *patternu* se ne dobavljaju podaci iz više tablica odjednom. Generalno *eager loading* puno bolje optimizira performanse. Razlog tome je da koristeći *lazy loading* servis svaki put izvršava novi upit prema bazi podataka kad pristupa svojstvima objekta iz *context classa*. S druge strane, kroz *eager loading* je moguće odmah dohvatiti podatke iz više tablica, te stoga nema potrebe za ponovnim upitima prema bazi podataka. Za usporedbu, kod (Kod 4.15.) prikazuje još jednu metodu napisanu prema *eager loading patter*nu koja dohvaća sve komentare za odabranu objavu unutar tablice objava.

```
public async Task<List<Comment>> GetCommentsForPost(Guid postId)
{
    return await _ctx.Comments.
        Include(c => c.Sentiment).
        Include(c => c.CommentType).
        Include(c => c.CommentCategory).
        Include(c => c.Author).
        ThenInclude(a => a.AgeGroup).
```

```
            Where(c => c.PostId == postId).OrderBy(c =>
            c.CommentCategoryId).ToListAsync();
    }
```

Kod 4.15. Metoda za dohvat svih komentara za odabranu objavu

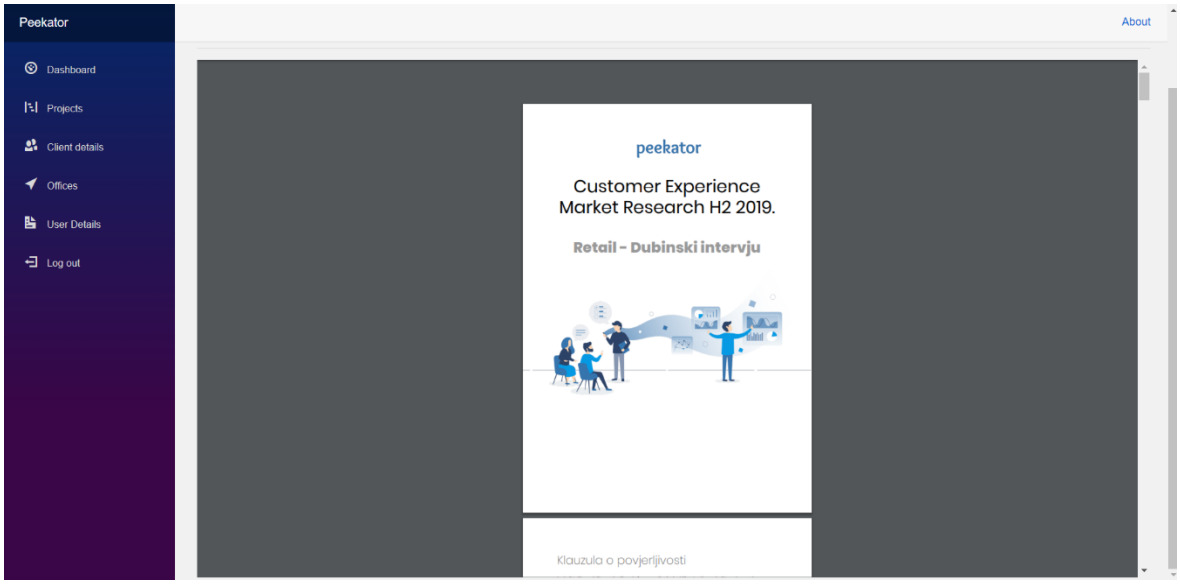
Postoje dva načina za pisanje LINQ sintakse. Prvi način je korištenje LINQ upita koji se koristi u kodu (Kod 4.12.). Drugi način je pisanje LINQ sintakse kroz *lambda* izraze kao što je to vidljivo u kodu (Kod 4.15.). Moguće je dobiti iste rezultate koristeći oba načina, stoga odabir LINQ sintakse generalno se svodi na osobne sklonosti. U nekim slučajevima je moguće da zbog dizajna baze podataka više odgovara jedan od načina LINQ sintakse te je stoga u ovoj aplikaciji korišten drukčiji pristup za dvije različite baze podataka.

4.3. Analiza i vizualizacija

Aplikacija nudi analizu podataka prikupljenih kroz tri tipa istraživanja:

- Dubinski intervju
- Terensko anketiranje
- Analiza društvenih mreža

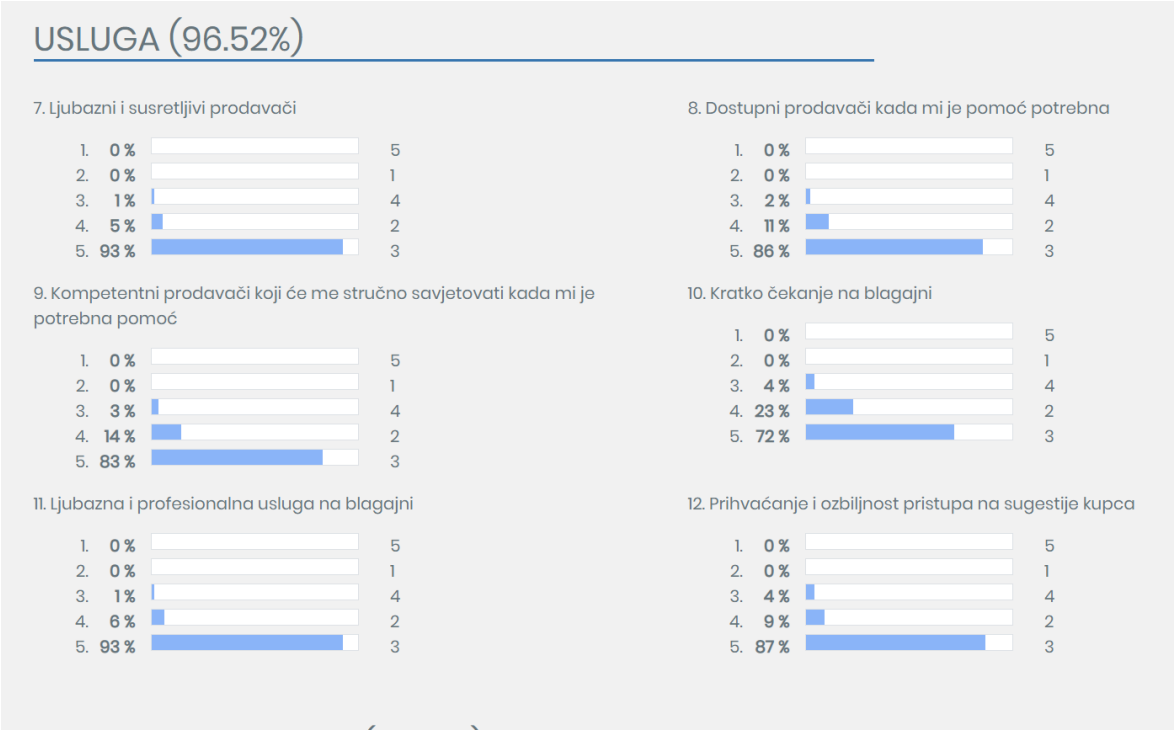
Dubinski intervju je kvalitativna metoda prikupljanja podataka koja se vrlo često koristi u društvenim istraživanjima. Za razliku od ankete, dubinski intervju uključuje manji broj ispitanika, no pitanja su opsežnija i otvorena čime se dobiva dublje razumijevanje istraživanog područja.



Slika 4.10. Prikaz dokumenta dubinskog intervjua

Izvještaj je priložen kao dokument kojeg se može preuzeti te ispisati (Slika 4.10.).

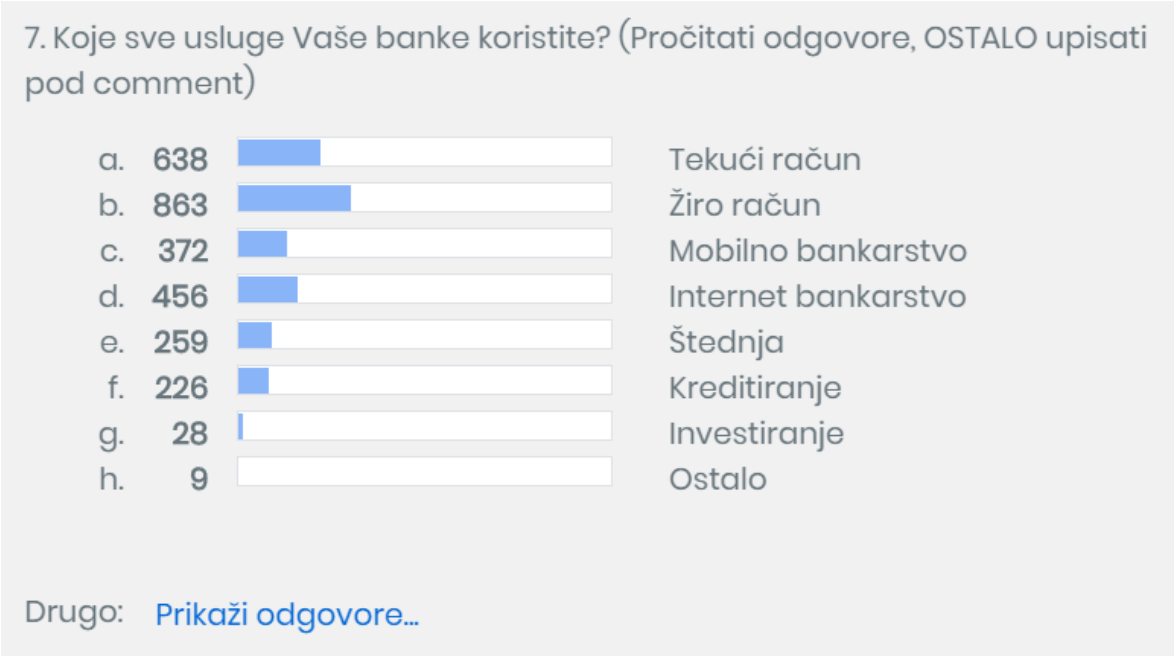
Izvještaj terenskog anketiranja prikazuje podatke prikupljene kroz ankete koje provode „peekeri“ – honorarni zaposlenici Peekatora. Njihov zadatak je postaviti zadana pitanja korisnicima usluga od klijenta koji je naručio istraživanje. Terenske ankete su brze za ispuniti te ne zahtijevaju veliku angažiranost kupaca.



Slika 4.11. Primjer kategorije koja sadrži pitanja s jednim mogućim odgovorom

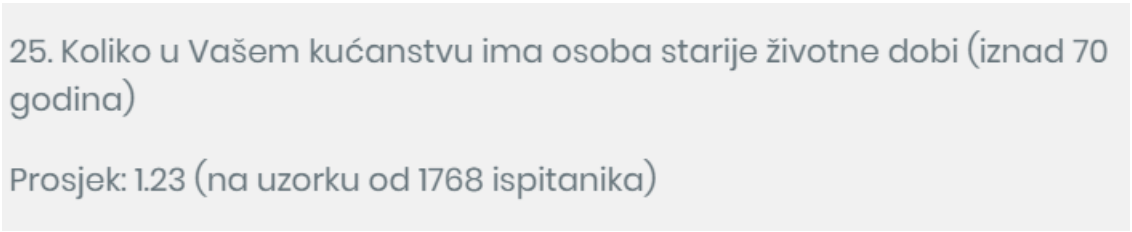
Pitanja s jednim mogućim odgovorom su pitanja gdje ispitivači ponude moguće odgovore od kojih ispitanik mora odabrati jedan odgovor. Slika (Slika 4.11.Slika 4.12) prikazuje više takvih pitanja unutar jedne kategorije. Nakon što je prikupljen dovoljan uzorak ispitanika, odgovori se zbrajaju te se za svaki ponuđeni odgovor računa postotak udjela od ukupnog broja odgovora. Ispod svakog pitanja su navedeni odgovori s njihovim postotkom te traka za napredak (engl. *progress bar*) koja služi kao vizualni prikaz postotka. S desne strane *progress bara* je napisan sami odgovor. Sva pitanja su numerirana po redoslijedu navođenja. Jednako tako i svi odgovori ispod pitanja su numerirani. Kategorije koje sadrže pitanja s jednim mogućim odgovorom također imaju izračunat postotak. U slučaju da određena kategorija nema niti jedno pitanje s jednim mogućim odgovorom, postotak kategorija se ne računa niti ispisuje. Više o načinu izračuna navedenih postotaka i agregaciji podataka je objašnjeno u poglavlju o dohvatu i pripremi podataka iz baza podataka. Na pitanja s više mogućih odgovora ispitanici mogu odabrati jedan ili više od ponuđenih odgovora (Slika

4.12.). Odgovori nisu numerirani koristeći brojeve, već se umjesto brojeva koriste slova navedena abecednim redoslijedom. Za ovakav tip pitanja se ne računa postotak svakom odgovoru, već umjesto postotka se piše ukupan broj ispitanika koji je odabrao taj odgovor. *Progress bar*, jednako kao za prethodno navedeni tip pitanja, vizualno prikazuje koliki udio svaki odgovor ima u usporedbi s ukupnim brojem odgovora.



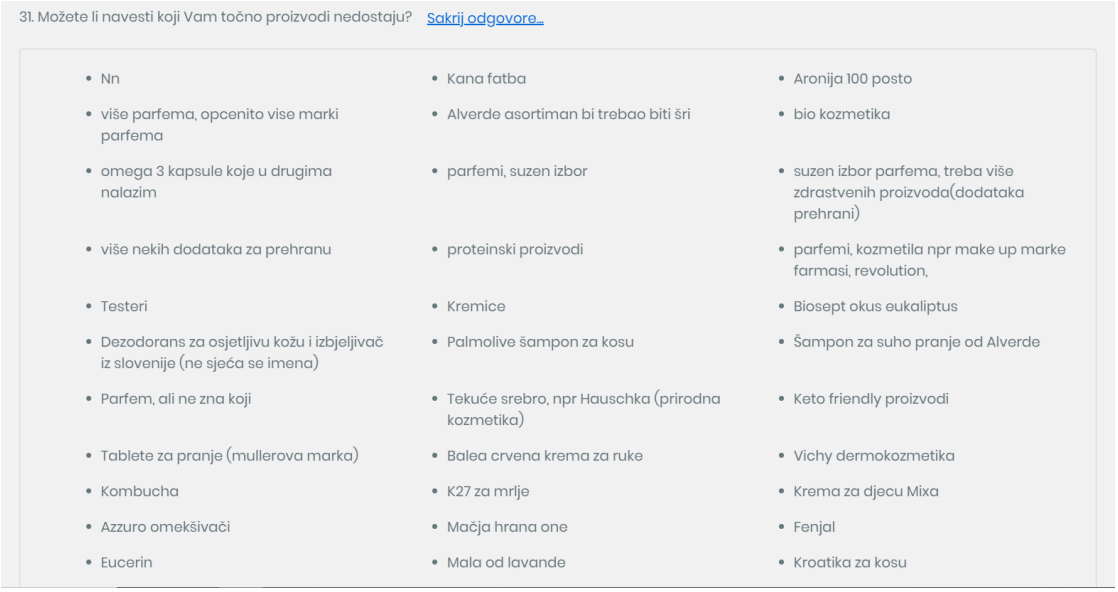
Slika 4.12. Primjer pitanja s više mogućih odgovora

Ovaj tip pitanja ima i svoj podtip koji osim ponuđenih odgovara može sadržavati i „ostale“ odgovore. Ostali odgovori su proizvoljni odgovori koje je ispitanik dao. Zadano su takvi odgovori skriveni, te klikom na gumb „Prikaži odgovore“ ti se odgovori prikažu u tri stupca ispod svih ostalih odgovora. Pitanja s brojevnim odgovorom su pitanja na koja nema ponuđenih odgovora, već ispitanici daju proizvoljan broj kao odgovor na pitanje. Unutar komponente se kao odgovor ispisuje prosjek dobivenih odgovora, zajedno s ukupnim brojem ispitanika (Slika 4.13.).



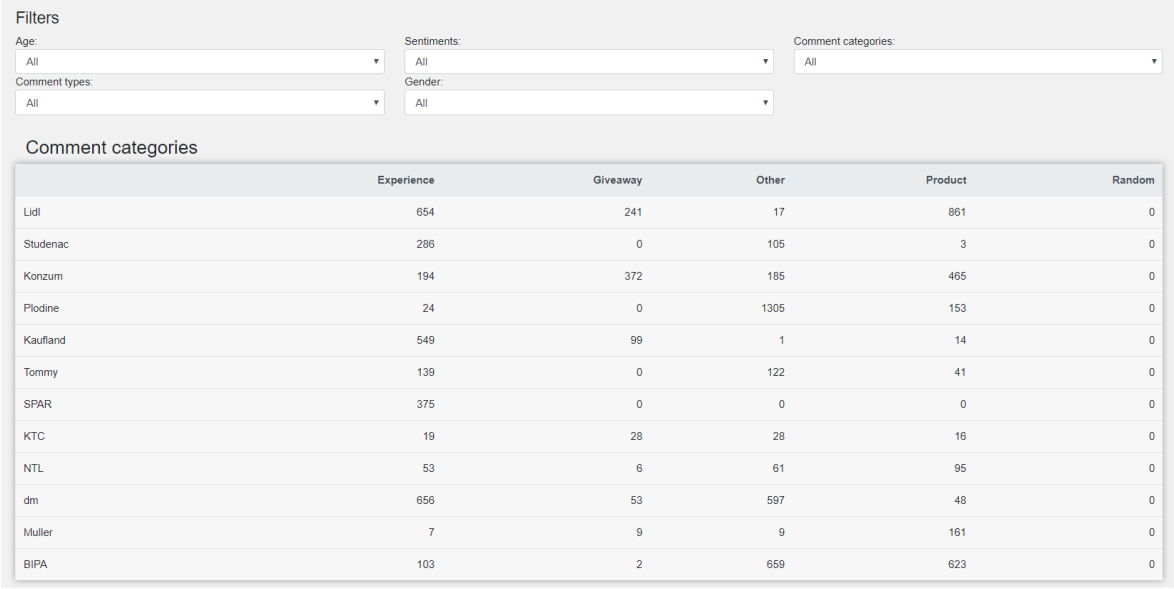
Slika 4.13. Primjer pitanja s brojevnim odgovorima

Pitanja s tekstualnim odgovorom također nemaju ponuđenih odgovora, već ispitanici daju proizvoljan tekstualni odgovor (Slika 4.14.).



Slika 4.14. Primjer pitanja s tekstualnim odgovorima

Kroz analizu društvenih mreža vrši se analiza objava i komentara s društvenih mreža. Istraživanje naručuje klijent te se analiza vrši na profilima društvenih mreža klijenta i njegovih konkurenata.



	Experience	Giveaway	Other	Product	Random
Lidl	654	241	17	861	0
Studenac	286	0	105	3	0
Konzum	194	372	185	465	0
Plodine	24	0	1305	153	0
Kaufland	549	99	1	14	0
Tommy	139	0	122	41	0
SPAR	375	0	0	0	0
KTC	19	28	28	16	0
NTL	53	6	61	95	0
dm	656	53	597	48	0
Muller	7	9	9	161	0
BIPA	103	2	659	623	0

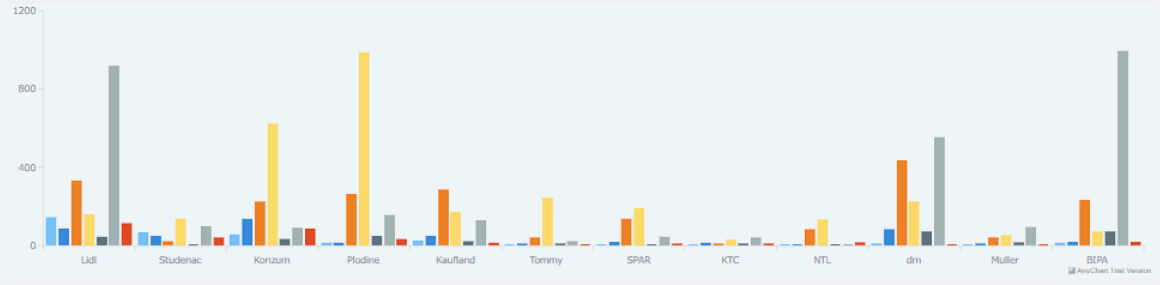
Slika 4.15. Filteri i tablica s brojem komentara prema kategorijama

Prva tablica, koja je vidljiva na slici (Slika 4.15.), prikazuje broj komentara na objave analiziranih subjekata prema svrsi objave. Na spomenutoj slici su također vidljivi filteri koji klijentima omogućuju detaljniju analizu podataka. Podaci unutar ove tablice su odličan

pokazatelj o tendencijama komentatora da ostavljaju komentare na objave određenog tipa. Tablica pokazuje kakve objave postižu veću angažiranost komentatora, što može pomoći u vođenju stranica na društvenim mrežama. Sljedeća tablica (Slika 4.16.) prikazuje broj komentara prema tipu komentara. Tip komentara je bitan kako bi se isfiltrirali *troll* komentari, uvažili *hater* komentari, cijenili komentari ljubitelja, no najbitnije kako bi se iskoristile prilike za poboljšanje predložene od strane komentatora, te prilike za prodaju proizvoda. Osim navedenih tipova preostaju komentari u kojima komentatori označavaju druge osobe te „ostali“ komentari pod koje spadaju svi komentari koje se ne može drukčije kategorizirati.

Comment types

	Hater	Improvement opportunity	Lover	Other	Sales opportunity	Tag	Troller
Lidl	141	84	327	156	40	915	110
Studenac	65	44	18	133	2	95	37
Konzum	50	131	219	619	28	85	84
Plodine	8	9	260	983	43	150	29
Kaufland	20	43	280	167	16	126	11
Tommy	1	7	35	238	4	16	1
SPAR	1	12	131	186	2	39	4
KTC	1	9	6	26	7	36	6
NTL	2	1	79	130	1	2	0
dm	5	77	430	219	69	551	3
Muller	2	7	36	47	0	92	2
BIPA	9	13	227	66	67	990	15



Slika 4.16. Tablica s brojem komentara prema tipu komentara i graf za vizualni prikaz

U nastavku izvještaja prikazane su tablice za ostatak parametara. Tablica s brojem komentara prema sentimentu uspoređuje količine pozitivnih, neutralnih i negativnih komentara. Tablica s parametrom dobi uspoređuje broj komentara prema dobnim skupinama komentatora. Tablica s parametrom spola uspoređuje broj komentara koje su ostavile osobe muškog, ženskog ili drugog spola. Dodatno je izgenerirana tablica koja prikazuje broj komentara samih subjekata u usporedbi s brojem komentara korisnika društvenih mreža. Osim tablica za usporedbu komentara i iskustava komentatora, izvještaj prikazuje i tablice koje

uspoređuju broj objava za svaki subjekt prema društvenoj mreži na kojoj je objava izvršena te samoj svrsi objave (Slika 4.17.).

Posts per Social Network

	Media	Facebook	Instagram
Lidl	0	65	7
Studenac	0	43	10
Konzum	3	56	6
Plodine	0	103	17
Kaufland	0	62	20
Tommy	0	64	0
SPAR	0	33	0
KTC	0	28	0
NTL	0	49	0
dm	0	102	55
Muller	0	37	15
BIPA	0	75	11

Posts per Post Purpose

	Experience	Giveaway	Other	Product	Random
Lidl	16	7	0	44	5
Studenac	21	0	0	2	30
Konzum	0	8	5	26	26
Plodine	1	1	0	68	50
Kaufland	3	7	0	27	45
Tommy	0	16	0	24	24
SPAR	2	4	0	7	20
KTC	0	8	1	10	9
NTL	0	2	0	29	18
dm	16	4	0	130	7
Muller	2	2	0	46	2
BIPA	2	0	0	59	25

Slika 4.17. Tablice s brojem objava prema društvenoj mreži i svrsi objave

Na kraju izvještaja nalazi se padajući izbornik za odabir jednog od subjekata. Odabirom subjekta popuni se tablica koja se nalazi ispod izbornika (Slika 4.18.).

Posts

Subject: Kaufland

Network Name	Post Purpose	Num of Reactions	Num of shares	Num of Comments	Link	
Facebook	Random	571	16	24	Post link	Show comments
Facebook	Giveaway	364	8	40	Post link	Show comments
Instagram	Random	397	0	10	Post link	Show comments
Facebook	Product	314	5	19	Post link	Show comments
Facebook	Product	1000	30	19	Post link	Show comments
Instagram	Random	384	0	4	Post link	Show comments
Facebook	Experience	86	3	80	Post link	Show comments
Instagram	Random	976	0	4	Post link	Show comments
Facebook	Product	369	9	10	Post link	Show comments

Slika 4.18. Tablica s generalnim podacima za svaku objavu odabranog subjekta

Tablica sadrži podatke o svakoj objavi odabranog subjekta te predstavlja kombinaciju prethodne dvije tablice. Uz opcije prethodnih dviju tablica sadrži i dodatne opcije u vidu

ukupnog broja reakcija na objavu, podjela objave te komentara. Uz to, za svaku objavu unutar tablice dostupan je *link* koji preusmjerava korisnika na originalnu objavu. Pored linka objave nalazi se gumb „Show comments“. Klikom na taj gumb se, ispod tablice s objavama subjekta, generira tablica s komentarima na odabranu objavu (Slika 4.19.). Unutar tablice su prikazani svi komentari odabrane objave te vrijednosti komentara za svaki parametar istraživanja. Ova tablica je iznimno korisna za pojedinačnu analizu komentara na osnovu svih parametara. Nakon što je pritisnut gumb za prikaz komentara određene objave, redak unutar tablice za tu objavu promijeni svoju pozadinsku boju kako bi bilo jasno koja je objava odabrana. Jednako tome i sami gumb promijeni svoju pozadinsku boju. Promjenom odabranog subjekta tablica s komentarima nestane.

Posts

Content	Num of Reactions	Num of shares	Num of Comments	Author Gender	Author Age From	Author Age To	Sentiment	Comment Type	Comment Category
Bravo! Svaka cast <	0	0	0	Female	46	55	Positive	Lover	Giveaway
Superrrrr	0	0	0	Female	46	55	Positive	Lover	Giveaway
Koliko puta se smije prijaviti tj.da li je veća šansa ako se više puta prijavi ili je dovoljno samo jednom? Hvala lijepa.	1	0	0	Female	36	45	Neutral	Other	Giveaway
super	0	0	0	Female	46	55	Positive	Lover	Giveaway
Ma odlično,a sada vas molim da to objavite na HRVATSKOM jeziku.	0	0	0	Male	46	55	Negative	Improvement opportunity	Giveaway
Di se može provjerit da sam osvojila nagradu današnju plej stejšon konzolu bio prozor da je to današnja nagrada	0	0	0	Female	46	55	Neutral	Other	Giveaway
Ne radi	1	0	0	Female	46	55	Negative	Improvement opportunity	Giveaway
Ne radi link	0	0	0	Female	26	35	Negative	Improvement opportunity	Giveaway

Slika 4.19. Komentari na odabranu objavu

4.4. Sigurnosni aspekti aplikacije

Aplikacija je osmišljena imajući na umu sigurnost klijenata kao jedan od najvećih prioriteta, te su stoga kroz implementaciju sustava autorizacije i autentifikacije primijenjene sigurnosne tehnike najbolje prakse. Prijava u sustav ujedno čini proces autentifikacije u svrhu sprečavanja pristupa aplikaciji neovlaštenim osobama. Autentifikacija je potrebna kako bi osoba koja nastoji pristupiti sustavu dokazala svoj identitet. Kao što je spomenuto u prijašnjim poglavljima, aplikacija koristi autentifikaciju na temelju *cookieja* – jedan od najčešće korištenih mehanizama za autentifikaciju korisnika. Autentifikacijski podaci

korisnika su sadržani unutar *cookieja* te po tom *cookieju* aplikacija prepoznaje korisnika sve dok je prijavljen u sustav. Kada se korisnik odjavi iz aplikacije, *cookie* se briše i autentifikacijski podaci korisnika koji su bili sadržani unutar njega nestaju, što znači da taj korisnik više nema pristupa aplikaciji i njenim funkcionalnostima. Ako želi ponovo dobiti pristup aplikaciji, korisnik se mora ispočetka prijaviti u sustav. Tijekom prijave, podaci osobe se validiraju, nakon čega, ako je validacija uspješna, korisnik dobiva pristup aplikaciji. Nakon prijave slijedi proces autorizacije kako bi se utvrdilo kojim podacima unutar aplikacije korisnik ima pristup. Tijekom autorizacije aplikacija dobavlja podatke o korisniku iz baze podataka na osnovu kojih se kreira `ClaimsPrincipal` objekt. Navedeni objekt predstavlja sigurnosni kontekst korisnika te sadrži njegov identitet. Unutar ove aplikacije identitet korisnika definira *authorization policy*, kao što to prikazuje kod (Kod 4.1.). Prema navedenoj klasi koja definira *authorization policy* korisnik može imati jedan od dva *claima*: `IsPeekatorEmployee` ili `IsClientEmployee`. Na osnovu tih *claimova* korisnik ima pristup različitim dijelovima aplikacije te mu se unutar određenih funkcionalnosti prikazuju drukčije opcije. Kako bi aplikacija provjerila koji *claim* ima prijavljeni korisnik, koristi se `AuthenticationState` – ugrađeni mehanizam koji pruža informacije o autentificiranom korisniku. Provjera autorizacijskih prava, odnosno *claima* korisnika, u svrhu prikaza različitog sadržaja unutar određene funkcionalnosti je vidljiva u kodu (Kod 4.16.).

```
[CascadingParameter]
Task<AuthenticationState> authenticationStateTask { get; set; }

protected override async Task OnInitializedAsync()
{
    var x = await authenticationStateTask;
    if (x.User.HasClaim("IsPeekatorEmployee", "true"))
    {
         IsPeekatorEmployee = true;
    }
    else if (x.User.Claims.Any(c => c.Type == "IsClientEmployee"))
    {
         IsPeekatorEmployee = false;
    }

    await base.OnInitializedAsync();
```

```
}
```

Kod 4.16. Provjera autorizacijskih prava korisnika

Kako bi se `AuthenticationState` mogao koristiti unutar cijele aplikacije, potrebno ga je definirati kao kaskadni parametar. Kaskadni parametri se definiraju unutar `App.razor` klase. Unutar te klase se nalazi `Router` komponenta koja omogućuje usmjeravanje na komponente s definiranom `@page` direktivom. Navedenu komponentu je potrebno okružiti `CascadingAuthenticationState` objektom, kao što je to vidljivo u kodu (Kod 4.17.).

```
<CascadingAuthenticationState>
   <Router AppAssembly="@typeof(Program).Assembly">
      <Found Context="routeData">
            <AuthorizeRouteView RouteData="@routeData"
            DefaultLayout="@typeof(MainLayout)">
                  <NotAuthorized>
                        <Peekator.Components.RedirectToLogin />
                  </NotAuthorized>
            </AuthorizeRouteView>
      </Found>
      <NotFound>
            <LayoutView Layout="@typeof(MainLayout)">
                  <p>Sorry, there's nothing at this address.</p>
            </LayoutView>
      </NotFound>
   </Router>
</CascadingAuthenticationState>
```

Kod 4.17. Sadržaj `App.razor` klase

Procesi autentifikacije i autorizacije su iznimno bitni za sigurnost ovog sustava kako bi se zaštitili bitni podaci klijenata i održalo njihovo povjerenje. Bilo kakvi sigurnosni propusti mogu imati ozbiljne posljedice za klijenta te je stoga neophodno da sustav bude zaštićen.

Zaključak

Pri planiranju poslovnih strategija potrebno je provesti istraživanja tržišta kako bi se upoznale potrebe korisnika te procijenilo njihovo zadovoljstvo trenutnim proizvodima ili uslugama. Rezultate provedenih istraživanja je potrebno efikasno obraditi te vizualno prikazati na način da su jasno vidljivi kako bi se klijentima omogućilo da dođu do relevantnih zaključaka o iskustvima njihovih krajnjih korisnika.

Peekator već nudi softversku platformu koja klijentima omogućuje pregled kupljenih projekata s pripadajućim istraživanjima, no u svrhu poboljšanja usluga započet je prijelaz na novu platformu temeljenu na novijim tehnologijama. Ova aplikacija je izrađena kao prvi od planiranih modula nove platforme te kao takva će se u budućnosti koristiti na produkcijskom nivou kod velikih klijenata, kao što su Zagrebačka banka, DM, Coca-cola i drugi.

Aplikacija je dio veće softverske platforme te predstavlja samo jedan od planiranih modula. Također, shodno tome, planirane su brojne druge funkcionalnost za nadogradnju aplikacije, prilagođene budućim potrebama Peekatora kako bi klijentima omogućili najbolji mogući uvid u stanje na tržištu te iskustva korisnika njihovih usluga i proizvoda.

Popis kratica

API	*Application Programming Interface*	sučelje za programiranje aplikacija
CRUD	*Create, Read, Update, Delete*	stvoriti, čitati, ažurirati, izbrisati
CSS	*Cascading Style Sheets*	kaskadni stilovi
CX	*Customer Experience*	korisničko iskustvo
DOM	*Document Object Model*	objektni model dokumenata
EF	*Entity Framework*	okvir entiteta
HTML	*Hpertext Markup Language*	hipertekstualni označni jezik
LINQ	*Language Integrated Query*	jezično integrirani upit
MS	*Microsoft*	Microsoft
SQL	*Structured Query Language*	jezik strukturiranih upita
T-SQL	*Transact-Structured Query Language*	jezik transakcijsko strukturiranih upita
URL	*Uniform Resource Locator*	usklađeni lokator sadržaja

Popis slika

Popis kôdova

Literatura

[1] MICROSOFT, Blazor, https://dotnet.microsoft.com/apps/aspnet/web-apps/blazor, 3. svibnja 2020.

[2] WIKIPEDIA, Blazor, https://en.wikipedia.org/wiki/Blazor, 4. svibnja 2020.

[3] WIKIPEDIA, WebAssembly, https://en.wikipedia.org/wiki/WebAssembly, 4. svibnja 2020.

[4] IVAN DEREVIANKO, Blazor – It's Time To Forget JavaScript, https://ivanderevianko.com/2019/07/blazor-its-time-to-forget-javascript, 4. svibnja 2020.

[5] MICROSOFT, Create and use ASP.NET Core Razor components, https://docs.microsoft.com/en-us/aspnet/core/blazor/components?view=aspnetcore-3.1, 6. svibnja 2020.

[6] MICROSOFT, Razor syntax reference for ASP.NET Core, https://docs.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-3.1, 6. svibnja 2020.

[7] MICROSOFT, Introduction to Razor Pages in ASP.NET Core, https://docs.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-3.1&tabs=visual-studio, 8. svibnja 2020.

[8] MICROSOFT, Entity Framework Core, https://docs.microsoft.com/en-us/ef/core/, 10. svibnja 2020.

[9] WIKIPEDIA, Microsoft SQL Server, https://hr.wikipedia.org/wiki/Microsoft_SQL_Server, 13. svibnja 2020.

[10] WIKIPEDIA, MySQL, https://hr.wikipedia.org/wiki/MySQL, 13. svibnja 2020.

[11] W3C, HTML & CSS, https://www.w3.org/standards/webdesign/htmlcss.html, 14. svibnja 2020.

[12] WIKIPEDIA, Bootstrap (front-end framework), https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework), 14. svibnja 2020.

[13] TANIA RASCIA, What is Bootstrap and How Do i Use It?, https://www.taniarascia.com/what-is-bootstrap-and-how-do-i-use-it/, 14. svibnja 2020.

[14] MOZILLA, What is Javascript?, https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript, 14. svibnja 2020.

ALGEBRA
VISOKO UČILIŠTE

WEB APLIKACIJA ZA OBRADU PODATAKA ISTRAŽIVANJA I PRIKAZ IZVJEŠTAJA

Pristupnik: Matej Cervelin, 0321004912

Mentor: prof. Aleksander Radovan

Datum: 11. 06. 2020.