Univerzitet u Novom Sadu

Ekonomski fakultet u Subotici

**Angular u razvoju poslovnih aplikacija**

DIPLOMSKI RAD

Student:

Marija Matić PIN028/17

Novi Sad, 2021. godina

Univerzitet u Novom Sadu

Ekonomski fakultet u Subotici

**Angular u razvoju poslovnih aplikacija**

DIPLOMSKI RAD

Mentor: Student:

Sakal dr Marton Marija Matić PIN028/17

Novi Sad, 2021. godina

|  |  |  |
| --- | --- | --- |
| Autor, AU: | | Marija Matić |
| Mentor, MN: | | Marton dr Sakal |
| Naslov rada, NR: | | Angular u razvoju poslovnih aplikacija |
| Jezik publikacije, JP: | | Srpski jezik |
| Zemlja publikovanja, ZP: | | Republika Srbija |
| Uže geografsko područje, UGP: | | AP Vojvodina |
| Godina, GO: | | 2021. |
| Mesto i adresa, MA: | | Subotica, Segedinski put 9-11 |
| Fizički opis rada, FO: (poglavlja/strana/citata/tabela/slika/grafika/priloga) | |  |
| Naučna oblast, NO: | | Poslovna informatika |
| Naučna disciplina, ND: | | Poslovna informatika |
| Predmetna odrednica/Ključne reči, PO: | | Angular, Typescript, poslovna veb aplikacija, komponenta, modul |
| Čuva se, ČU: | | Biblioteka Ekonomskog fakulteta u Subotici |
| Važna napomena, VN: | | - |
| Izvod, IZ: | |  |
| Datum prihvatanja teme, DP: | |  |
| Datum odbrane, DO: | |  |
| Članovi komisije, KO: | Predsednik, član: | Sakal dr Marton |
|  | Mentor, član: | Raković dr Lazar |

Sadržaj

[**1.** **Uvod** 5](#_Toc81747396)

[**2.** **Istorija razvoja veb aplikacija** 6](#_Toc81747397)

[**2.1.** **HTML** 6](#_Toc81747398)

[**2.2.** **CSS** 6](#_Toc81747399)

[**2.3.** **JavaScript** 6](#_Toc81747400)

[**3.** **Angular** 7](#_Toc81747401)

[**3.1. Prednosti korišćenja Angulara u razvoju veb aplikacija** 8](#_Toc81747402)

[**4.** **Ostale tehnologije korišćene u izradi veb aplikacije** 8](#_Toc81747403)

[**4.1.** **ASP.NET Core Framework** 8](#_Toc81747404)

[**4.2.** **Baza podataka** 9](#_Toc81747405)

[**5.** **CASE STUDY – Veb aplikacija** 10](#_Toc81747406)

[**5.1.** **Relacioni dijagram** 10](#_Toc81747407)

[**5.2.** **Struktura aplikacije** 11](#_Toc81747408)

[**5.3.** **Moduli** 12](#_Toc81747409)

[**5.4.** **Komponente** 14](#_Toc81747410)

[**5.5.** **Lifecycle hooks** 15](#_Toc81747411)

[**5.6.** **Povezivanje podataka u Angular-u** 16](#_Toc81747412)

[**5.6.1.** **Jednostrano povezivanje: Od komponente prema prikazu** 16](#_Toc81747413)

[**5.6.2.** **Jednostrano povezivanje: Od prikaza ka komponenti** 17](#_Toc81747414)

[**5.6.3.** **Obostrano povezivanje** 17](#_Toc81747415)

[**5.7.** **Stranica za prijavu** 19](#_Toc81747416)

[**5.8.** **Početna strana admin panela** 24](#_Toc81747417)

[**5.9.** **Rutiranje** 26](#_Toc81747418)

[**5.10.** **Servisi** 29](#_Toc81747419)

[**5.11.** **Komponenta „Korisnik“** 31](#_Toc81747420)

[**5.12.** **Zaključavanje stavki** 38](#_Toc81747421)

[**5.13.** **Lokalizacija** 41](#_Toc81747422)

[**6.** **Zaključak** 43](#_Toc81747423)

[**7.** **Literatura** 44](#_Toc81747424)

# **Uvod**

Praćenje prodaje robe, stanja robe na zalihama, kupaca i zaposlenih organizacije, izdatih faktura i isporuka robe kupcima, u jednoj organizaciji veoma je naporan i komplikovan posao. Veb aplikacija za upravljanje navedenim procesima služi da pojednostavi celokupan proces i da olakša osoblju da na brz i elegantan način prate podatke, da pristupe podacima, da ih dodaju, brišu i menjaju i na taj način obezbede uslove za donošenje kvalitetnih odluka.

Prilikom razvoja ove poslovne web aplikacije, koristila sam se savremenim tehnologijama:

* front-end tehnologije: HTML5, SCSS, Typescript (Framework Angular)
* back-end tehnologije: .NET Web API, SQL

S obzirom na to da je aplikacija razvijana uz pomoć Angular framework-a, sama aplikacija je modularna, jer Angular poseduje sopstveni modularni sistem – NgModules. Moduli koji se kreiraju i potom smeštaju unutar kontejnera imaju sopstvene funkcionalnosti, komponente i servise koji se mogu koristiti unutar ostalih modula. Ovo čini aplikaciju skalabilnom i lako održivom, a samim tim i lako prilagodljivom potrebama kompanije.

Ideju za razvoj ove web aplikacije dobila sam nakon prisustvovanja lošem sistemu upravljanja fakturama organizacije, gde je primarni problem bio nepostojanje centralnog repozitorijuma podataka, ali i aplikacije uz pomoć koje bi se moglo upravljati podacima celokupnog procesa. S obzirom na to da je proces upravljanja fakturama, isporukama, kupcima itd. veoma složen i zahteva mnogo vremena, odlučila sam da kreiram pojednostavljenu verziju veb aplikacije za praćenje faktura organizacije.

Komunikaciju sa bazom podataka odlučila sam da sprovedem koristeći Web Api. S obzirom na to da sam se odlučila za organizaciju podataka zasnovanu na relacionom modelu, trenutno je, uz pomoć Web Api-a, uspostavljena konekcija sa SQL serverom.

Za razvoj „prednjeg“ ili klijentskog dela aplikacije – preko kojeg će korisnici moći da pristupe sistemu, koristila sam Angular framework. To je okvir koji se koristi za razvoj mobilnih ili desktop veb aplikacija. Angular omogućava izradu takozvane jednostranične aplikacije. Reč je o aplikaciji koja, umesto da prilikom prelaska na drugu stranu učitava celokupnu aplikaciju ponovo, ona samo dinamički očitava druge podatke u zavisnosti od zahteva korisnika. Na taj način se značajno štedi na vremenu koje bi „odlazilo“ na stalnu klijent-server komunikaciju (za učitavanje nepotrebnog HTML, Typescript i CSS koda), ali i doprinosi većoj responzivnosti i fluentnosti, što rezultuje okruženjem koje korisnicima omogućava lakše snalaženje.

# **Istorija razvoja veb aplikacija**

## **HTML**

Razvoj poslovnih aplikacija kakvih danas poznajemo i koristimo prvenstveno se obavljao pomoću HTML-a – jezika za označavanje hiperteksta. HTML je krajem 1991. godine razvio Tim Berners-Li i HTML kao takav sastojao se od samo 18 različitih tagova. Struktura HTML fajlova definisana je različitim tipovima tagova, na osnovu kojih veb pretraživač formatira i prikazuje sadržaj. Veb pretraživači konstruisani su tako da primaju HTML fajlove i njihov sadržaj – tekstove, slike i multimediju interpretiraju u vidu veb stranica koje su razumljive i sa kojima mogu da komuniciraju i „obični“ korisnici.

## **CSS**

S nastankom potrebe da se razdvoji sadržaj samih veb strana od njenog vizuelnog prikaza, dolazi do nastanka CSS-a, koji se smatra prekretnicom u sferi veb dizajna. CSS omogućava uređivanje izgleda veb strana uz formatiranje izgleda pojedinačnih HTML elemenata – promenu njihovog izgleda, fontova, boja itd. Separacija formatiranja i strukture doprinosi rastu fleksibilnosti, padu kompleksnosti, ali i prilagođavanju veb strana uređajima različitih dimenzija.

## **JavaScript**

Prvobitne veb stranice bile su potpuno statičke, odnosno, njihov sadržaj nije mogao da se menja nakon prvobitnog učitavanja. Kao posledica želje da se ovo ograničenje otkloni, 1995. godine kompanija Netscape dodaje skriptni jezik u veb pretraživač Navigator i to u cilju ugrađivanja programskog jezika Scheme-a i programskog jezika Jave u tada najkorišćeniji veb pretraživač. Kada su uvideli probleme vezane za implementaciju, dolaze do ideje da kreiraju potpuno nov programski jezik. Iako je inicijalno nosio ime LiveScript, sa prvom objavom beta verzije Navigatora izlazi i novi programski jezik po imenu JavaScript. JavaScript i ostali skritpni jezici zapravo determinišu način na koji će se HTML primljene stranice biti preveden u objektni model dokumenta – DOM[[1]](#footnote-1). Nastankom Internet Explorer-a i njegovog skriptnog jezika JScript-a, dolazi do „rata“ između veb pretraživača. To otežava situaciju i stvara problem programerima kojima je bilo teško da kreiraju veb sajtove koji su mogli da rade na oba veb pretraživača. U godinama koje slede popularnost Microsoft-ovog Internet Explorer-a je rasla, te 2000. godine Navigator gubi trku, i skriptni jezik JSkript postaje standard za razvoj dinamičkih veb stranica.

JavaScript, kakvog danas poznajemo, nastao je 2008. godine s nastankom Microsoft-ovog Google Chrome-a, i to u sklopu V8 – open-source pokretača (biblioteke) koji je služio kao tradicionalni interpreter. Ono što je pravilo razliku u odnosu na ostale veb pretraživače i činilo ga bržim u odnosu na njih, jeste postojanje JIT-a - kompajlera koji nadgleda i optimizuje način izvršavanja kompjuterskog koda.

Danas preko 80% veb sajtova koristi neku od biblioteka JavaScripta[[2]](#footnote-2) ili veb-okvir za razvoj i podršku dinamičnih veb sajtova, veb aplikacija, veb servisa, za razvoj veba uopšte. Za rešavanje problema posmatrane kompanije, odlučila sam da veb aplikaciju za upravljanje fakturama razvijem jednim od veb okvira – Angular-om.

# **Angular**

Jedan od najpoznatijih okvira za razvoj veb aplikacija danas, pored Vue.js-a i React-a jeste Angular. Prva verzija ovog okvira pod nazivom AngularJS nastala je 2009. godine i održavana je primarno od strane kompanije Google, ali i zajednice pojedinaca i kompanija u cilju razvoja „jednostranične aplikacije“. Ideja je da korisnikovo iskustvo i interakcija sa veb-aplikacijom ili veb-sajtom bude slična interakciji sa aplikacijama na računarima bez internet konekcije. Umesto učitavanja kompletnog seta veb stranica sa veb servera, aplikacija ili veb sajt dinamično učitava ili dopunjava postojeću veb stranicu sa novim podacima sa servera. Na taj dolazi do uštede resursa, što čini samu aplikaciju mnogo efikasnijom. Pored toga, cilj je kreirati takvu aplikaciju koja će korisnicima biti jednostavna i prijatna za korišćenje. Korisniku u svakom trenutku treba da bude jasno kako aplikacija funkcioniše, čak i kada se prvi put susretne sa grafičkim korisničkim interfejsom. Ne treba je zatrpati bespotrebnim informacijama, već treba prikazati ono što je najbitnije za dati prozor.

AngularJs omogućava proširenje HTML vokabulara, dodaje mogućnost korišćenja različitih kontrola, omogućava kreiranje komponenti koje se mogu više puta koristiti, olakšava navigaciju, testiranje, komunikaciju sa backend-om itd. Pored toga, dizajniran je tako da ga je moguće korisiti sa ostalim bibliotekama, a sve predefinisane opcije se mogu prilagoditi pojedinačnim potrebama.

Angular je okvir koji je napisan u TypeScript programskom jeziku, razvijenom od strane Microsoft-a. Kreiran je s namerom da otkloni nedostatke Javasript-a, a može raditi sa bilo kojom verzijom JavaScript-a. Osnovna prednost TypeScript-a, jeste to što omogućava nametanje ograničenja vrednostima promenljivih. To doprinosi većoj jasnosti koda i sprečavanju grešaka jer varijable imaju tačno definisan tip, što kod Javascript-a nije slučaj. Pored toga, Typescript je opcionalan. Aplikacija u Angular-u može se razvijati upotrebom Vanilla JavaScript-a, jer Angular CLI već interno koristi TypeScript. Korišćenje Typescript-a u razvoju Angular veb aplikacije nije nužno ali pruža mnogo više mogućnosti Angular-a nego u slučaju korišćenja JavaScript-a.

Slika 1. Angular logo  
Izvor: https://en.wikipedia.org/wiki/Angular\_(web\_framework)

## **3.1. Prednosti korišćenja Angulara u razvoju veb aplikacija**

Razvoj aplikacija koje bi trebalo da zadovolje potrebe korisnika nije nimalo lak zadatak. Kvalitet i kompleksnost veb aplikacija raste, a sa njima rastu i očekivanja korisnika.

Prednosti korišćenja Angular okvira za razvoj veb aplikacija su brojne, a samo neke od ključnih su[[3]](#footnote-3):

* Jasno definisani veb standardi – danas postoji mnogo načina i ideja o tome kako dizajnirati aplikacije. Angular je okvir kojim se definišu standardi poput korišćenja TypeScript programskog jezika, SCSS (umesto CSS-a) i slično.
* Razvojni alati, mogućnost prilagođavanja – Angular omogućava zajedničko iskustvo za programere uz postojanje CLI[[4]](#footnote-4) – interfejsa koji omogućava automatizaciju rada. Putem interfejsa, na veoma brz i jednostavan način, mogu se kreirati moduli, komponente, servisi i sl.
* Moćan ekosistem sa velikom zajednicom – sve je veći broj novih third-party biblioteka, biblioteka za korisnički interfejs, događaja i korisnika. Velika zajednica stvara dobre temelje za učenje i uliva poverenje u to da će ostati vredna tehnologija.
* Podrška Google-a, open source zajednica – Google-ov tim inžinjera posvećen je predstavljanju Angular-a ostatku veb zajednice i podsticanju korišćenje Angular-a za razvoj aplikacija koje žive na vebu, mobilnim ili desktop uređajima.

Angular Material je biblioteka koja omogućava implementaciju elemenata Material Design-a. To je sistem dizajniran od strane Google-a, koji omogućava izgradnju visoko responzivnih i prijatnih za korišćenje korisničkih interfejsa. Takođe, Angular je univerzalni okvir, te je napor za razvijanje aplikacija na više različitih platformi smanjen jer to vodi ka uštedi vremena i novca.

# **Ostale tehnologije korišćene u izradi veb aplikacije**

## **ASP.NET Core Framework**

.NET je open-source framework razvijen od strane Microsoft-a, koji se koristi za razvoj savremenih poslovnih web aplikacija na strani servera. Inicijalna verzija 1.0. .NET framework-a, objavljena je u januaru 2002. godine, kao naslednik Microsoft-ove tehnologije ASP. Kao deo .NET framework-a, izvršava se na CLR-u i tako omogućava razvijanje dinamičkih veb stranica upotrebom bilo kog podržanog .NET jezika.

CLR (Common Language Runtime) je osnovni alat na kom se .NET framework izvršava. On pruža servise za upravljanje izuzecima, sigurnošću, upravljanje izuzecima, nitima procesora i sakupljanje otpada. Svi programi napisani upotrebom .NET framework-a izvršavaju se na CLR, a kompajliraju u CLI (Common Intermediate Language Code), dok se tokom izvršavanja sam CLI kompajlira u mašinski kod.

ASP.NET CORE je naslednik ASP.NET-a, odnosno, on predstavlja reimplementaciju ASP-NET-a kao modularnog veb framework-a i otvorenog je tipa. Ono što je još jedna prednost ASP.NET CORE-a u odnosu na njegovog prethodnika, jeste što se on, pored Windows-a, može koristi i na ostalim operativnim sistemima.

Za potrebe projekta, korišćen je ASP.NET Web API napisan u C# programskom jeziku, koji predstavlja dobru podlogu za izgradnju skalabilnih RESTful aplikacija baziranih na .NET Framework-u.

## **Baza podataka**

Baza podataka predstavlja kolekciju podataka organizovanu na način da bude pristupačna i lako prilagodljiva. Baza podataka predviđena je za upotrebu od strane većeg broja korisnika i aplikacija, te je neophodno da ona bude raspoloživa u svakom trenutku. Podacima u bazi podataka upravlja se sistemom za upravljanje bazom podataka.

Sistemi za upravljanje bazama se, u zavisnosti od modela baza koje podržavaju, mogu podeliti na sisteme za upravljanje relacionim i sisteme za upravljanje ne relacionim bazama podataka.

Relaciona baza podataka je poseban tip baze podataka, gde su podaci organizovani u skupu relacija između kojih se definišu određene veze. Naziv relacione baze podataka osmislio je E.F. Codd 1970. godine, radeći za IBM. Codd je definisao relacione baze podataka kao:

„Produkt koji prezentuje pregled podataka kao kolekcije redova i kolona“. Relacija se definiše kao skup [n-torki](https://sh.wikipedia.org/w/index.php?title=N-torka&action=edit&redlink=1) sa istim atributima, definisanih nad istim [domenima](https://sh.wikipedia.org/w/index.php?title=Domen_(matematika)&action=edit&redlink=1) iz kojih mogu da uzimaju vrednosti. U relacionim bazama podataka, svaka relacija mora da ima definisan [primarni ključ](https://sh.wikipedia.org/w/index.php?title=Primarni_klju%C4%8D&action=edit&redlink=1), koji predstavlja atribut pomoću kojeg se jedinstveno identifikuje svaka [n-torka](https://sh.wikipedia.org/w/index.php?title=N-torka&action=edit&redlink=1). Relacija opciono može da poseduje i [spoljni ključ](https://sh.wikipedia.org/w/index.php?title=Strani_klju%C4%8D&action=edit&redlink=1), preko kojeg ostvaruje vezu sa drugim relacijama.[[5]](#footnote-5)

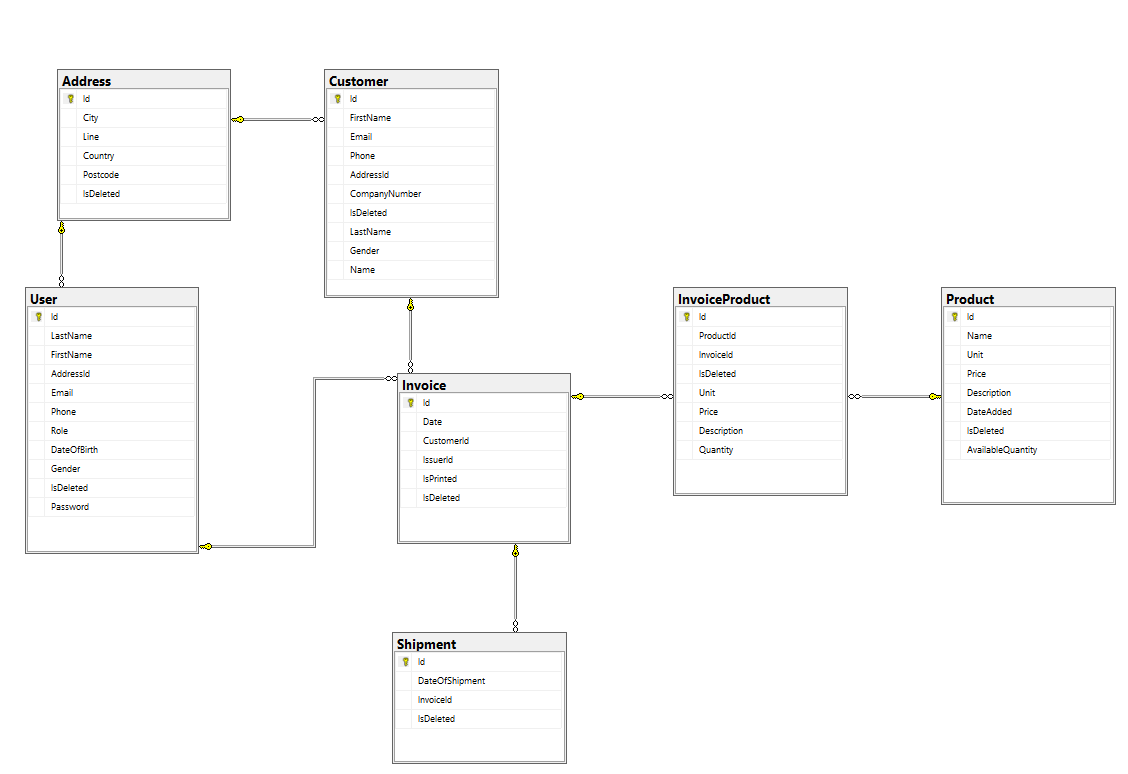
Codd je dokazao da se odnos između podataka može prikazati jednostavnim konceptima. Jedan od njih podrazumeva mogućnost menjanja i selektovanja samo jednog reda u tabeli putem primarnog ključa. Kada se u tabeli dodaje novi red, njemu se dodeljuje novi primarni ključ. Primarni ključevi moraju biti unikatni. U slučaju da primarni ključ jedne tabele „prebacimo“ u drugu tabelu, on postaje spoljni ključ, te na taj način nastaju veze između tabela.

SQL je standardni korisnički jezik za čuvanje, upravljanje i preuzimanje podataka u bazama podataka.[[6]](#footnote-6)

# **CASE STUDY – Veb aplikacija**

## **Relacioni dijagram**

Baza podataka sastoji se od 7 tabela: adresa, kupac, korisnik, faktura, proizvodi, proizvodi na fakturi i isporuka i veza između ovih tabela i ovaj dijagram predstavlja pojednostavljenu verziju realnog sistema za upravljanje fakturama kompanije.



Slika 2. Relacioni dijagram baze podataka korišćene u aplikaciji

Svaka tabela ima svoj primarni ključ – ID, dok su tabele međusobno povezane stranim ključevima.

* Tabela adresa sastoji se od primarnog ključa, grada, adrese, poštanskog broja, države i indikatora da li je adresa obrisana.
* Tabela korisnik ima primarni ključ, ime, prezime, email adresu, lozinku, broja telefona, datum rođenja, ulogu u sistemu, pol, indikatora da li je korisnik obrisan, strani ključ tabele adresa i strani ključ tabele faktura.
* Tabela kupac sadrži primarni ključ, ime, prezime, broj telefona, email adresu, pol, PIB i naziv kompanije – ukoliko je reč o kompaniji, indikator da li je kupac obrisan i strani ključ tabele adresa.
* Tabela isporuka sadrži primarni ključ, datum isporuke, indikator koji ukazuje na to da li je isporuka obrisana i strani ključ tabele faktura.
* Tabela proizvod sadrži primarni ključ, ime proizvoda, jedinicu mere, cenu proizvoda, kratak opis, raspoloživu količinu na skladištu, indikator da li je proizvod obrisan i strani ključ tabele proizvodi fakture.
* Tabela proizvodi fakture sadrži primarni ključ količinu poručenog proizvoda po fakturi, vrednost proizvoda, opis, jedinicu mere, indikator da li je proizvod obrisan sa fakture, ali i strane ključeve tabela proizvod i faktura.
* Tabela faktura sadrži primarni ključ, datum fakture, indikator da li je faktura obrisana, indikator koji ukazuje na to da li je faktura proknjižena, ali i strane ključeve tabela kupac i korisnik.

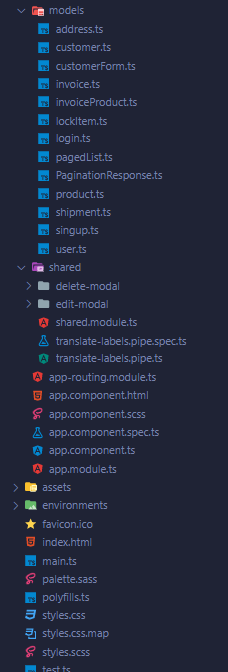
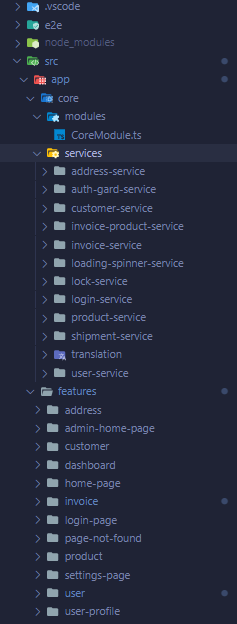
## **Struktura aplikacije**

Sav kod aplikacije nalazi se u okviru src foldera. U folderu /app smešteni su svi moduli, servisi, direktive, komponente - odnosno, logika i podaci aplikacije. Pored toga, tu se nalaze i fajlovi kojima se definiše globalni izgled aplikacije - fontovi, teme itd. Pored toga, tu se smeštaju i fajlovi poput slika, video zapisa, json fajlova i skriptova.

U posmatranom projektu, logika i kod aplikacije grupisani su po sledećim direktorijumima:

* core - direktorijum u kom se nalaze moduli i servisi aplikacije: servisi za autentifikaciju prijave na sistem, prijave na sistem, zaključavanje stavki, lokalizacije aplikacije itd.
* features - u kom se nalaze komponente aplikacije: stranica za prijavu, admin panel stranica, stranica korisničkmog profila itd.
* models - u kom se nalaze svi interfejsi koji se koriste u aplikaciji
* shared - direktorijum u kom se nalaze komponente, moduli, direktive i sl. koje se višestruko koriste u aplikaciji.

Slika 3. Struktura aplikacije



## **Moduli**

Svaka angular aplikacija sastoji se od velikog broja komponenti. Kako aplikacija "raste", tako raste i njena kompleksnost, te se ona mora podeliti u odgovarajuće logičke celine. Iz tog razloga koriste se moduli.

Moduli su klase koje se koriste za organizovanje aplikacije i grupisanje funkcionalnosti u jednu celinu. Dakle, svaki modul grupiše delove aplikacije - komponente, servise, direktive itd., u blokove funkcionalnosti, te svaki modul predstavlja jedan deo aplikacije. Moduli omogućavaju korišćenje eksternih biblioteka[[7]](#footnote-7). Osnovni modul aplikacije je App modul koji vodi računa o ostalim modulima aplikacije.

Moduli se označavaju dekoratorom @NgModule koji se nalazi u @angular/core biblioteci. U okviru dekoratora se navode koje se komponente nalaze unutar tog modula, koja je osnovna komponenta tog modula, koji se servisi koriste u tom modulu, koji se moduli uvoze u okviru njega i slično. Svi moduli mogu biti učitani prilikom startovanja aplikacije, ili se mogu "lenjo" učitavati po potrebi.

Da bi funkcionalnosti neke biblioteke mogle koristiti, neophodno ih je uvesti u odgovarajući modul. Biblioteke angular-a - poput FormsModule-a, RouterModule-a i slično, deo su NgModule-s biblioteke.

Da bi se sprečilo ponavljanje import-a svakog modula na više različitih mesta, preporuka je da se kreira zajednički modul, u kom će se nalaziti definicije svih modula koji se koriste više puta, bilo da je reč o predefinisanim ili korisnički definisanim modulima.

Uz pomoć Angular CLI modul se može kreirati upotrebom komande ng generate module naziv-modula.

@NgModule({

  declarations: [

    AppComponent,

    SettingsComponent,

    DashboardComponent,

  ],

  imports: [

    SharedModule,

    BrowserModule,

    AppRoutingModule,

    LoginPageModule,

    BrowserAnimationsModule,

    CoreModule,

    HomePageModule,

    AdminHomePageModule,

    UserModule,

    AddressModule,

    CustomerModule,

    ProductModule,

    InvoiceModule,

    TranslateModule.forRoot(

      {

      loader: {

        provide: TranslateLoader,

        useFactory: HttpLoaderFactory,

        deps: [HttpClient]

      }

    }

    ),

  ],

  providers: [TranslationService],

  bootstrap: [AppComponent]

})

export class AppModule { }

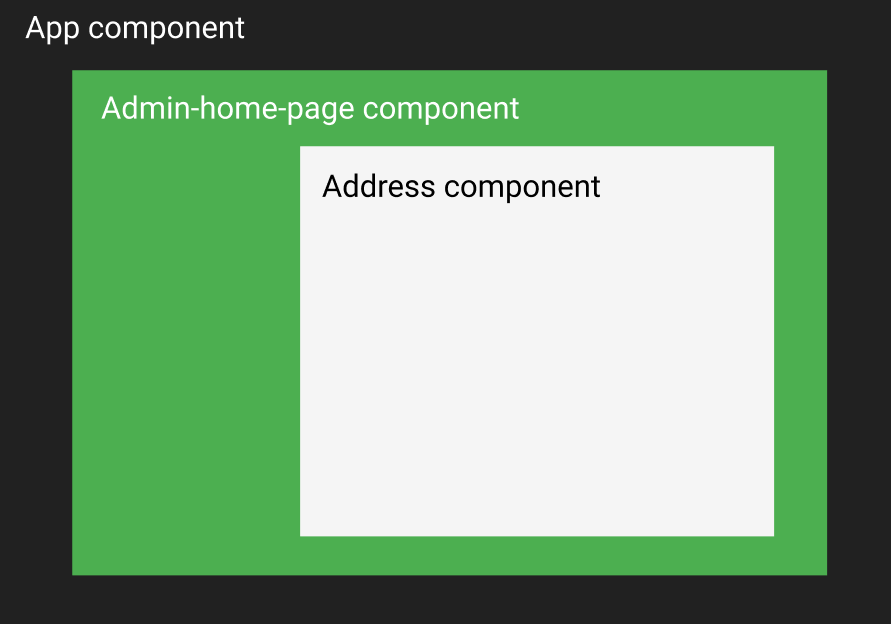
export function HttpLoaderFactory(http: HttpClient) {

  return new TranslateHttpLoader(http);

}

## **Komponente**

Klase su osnovni gradivni element svake angular aplikacije. To su klase koje predstavljaju nezavisnu celinu koja obavlja odgovarajuću funkcionalnost. Unutar komponente nalazi se njena logika, a sama klasa označava se dekoratorom @Component.

S obzirom na to da je komponenta klasa, instanciranjem klase, delovi komponente mogu se deliti i koristiti u drugim komponentama unutar aplikacije. Na taj način se obezbeđuje ponovna upotrebljivost komponente. Jedna komponenta može biti izgrađena od više ugnježđenih komponenata.

Slika 4. Primer ugnježđavanja komponenata u posmatranoj aplikaciji

Komponenta se može kreirati ručno, ali i putem Angular CLI-a. Kada se komponenta kreira posredstvom Angular CLI (komandom ng generate component naziv-komponente), to kreira direktorijum u kom se nalaze sledeći dokumenti:

1. dokument komponente - naziv-komponente.component.ts

2. dokument prikaz komponente - naziv-komponente.component.html

3. CSS (SCSS) dokument - naziv-komponente.component.scss

4. dokument za specifikaciju testiranja - naziv-komponente.component.spect.ts[[8]](#footnote-8)

U typescript (ts) dokumenta, u okviru @Component dekoratora, navode se:

- selektor komponente - na osnovu kog se komponenta jednoznačno identifikuje,

- templateUrl - putanja ka HTML dokumentu komponente (ili konkretan blok koda)

- styleUrls - putanja ka SCSS dokumentu/dokumentima komponente[[9]](#footnote-9)

Prikaz dekoratora i tela osnovne komponente aplikacije:

@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

  styleUrls: ['./app.component.scss']

})

export class AppComponent {

  title = 'projekat-angular';

  logged = '';

  constructor(private spinnerService: LoadingSpinnerService) {

  }

  showSpinner = this.spinnerService.visibility.pipe(delay(0));

}

## **Lifecycle hooks**

Nakon instanciranja direktive ili komponente pozivanje konstruktora, Angular poziva metode kojima se definiše njihov životni ciklus.

Postoji više metoda životnog ciklusa komponente. Osnovne metode životnog ciklusa komponente (ili direktive) su[[10]](#footnote-10):

- ngOnInit() - metoda koja se poziva prilikom inicijalizacije komponente

- ngOnChanges() - ukoliko komponenta ima dete, ova metoda poziva se prilikom promene svojstva komponente deteta iz roditelj komponente

- ngDoCheck() - metoda koja se poziva prilikom bilo kakve promene svojstva komponente i uglavnom se ova metoda poziva nakon metode ngOnChanges() i/ili ngOnInit().

- ngOnDestroy() - metoda koja se poziva pre destrukcije instance komponente

## **Povezivanje podataka u Angular-u**

### **Jednostrano povezivanje: Od komponente prema prikazu**

#### **Povezivanje svojstava - Property binding**

HTML elementima u angularu moguće je postaviti vrednosti uz pomoć binding-a - tzv. povezivanja. Postoje različiti atributi koji se mogu dodeliti elementima, u zavisnosti od toga o kom je elementu reč. Atributi se postavljaju unutar uglastih zagrada, a vrednost se dodeljuje sa desne strane znaka jednakosti, pod navodnicima. Pod navodnicima se može postaviti statička vrednost, promenljiva koju je moguće dinamički menjati ali čak i formula.

Svojstvo sledećeg dugmeta se dinamički menja u zavisnosti od ispunjenosti uslova.

 <button mat-icon-button

[disabled]="editIndicator == true && element.id == this.lockedItem.itemId">

</button>

* + - 1. **Povezivanje atributa – Attribute binding**

Podrazumeva dodeljivanje atributa nekom od HTML elemenata. Povezivanje atributa najčešće se koristi kada ne postoji željeno svojstvo nekog HTML elementa.

* + - 1. **Povezivanje stilova – Style binding**

Koristi se za dodavanje stila HTML elementima – širina, veličina fonta, boja itd.

* + - 1. **Povezivanje klasa – Class binding**

Koristi se da bi se dodalo svojstvo CSS (ili SCSS) klase nekom od HTML elemenata.

* + - 1. **Nasleđivanje/ Interpolacija**

Pored mogućnosti postavke svojstava, moguće je prikazivati dinamičke vrednosti stringova u okviru HTML tagova[[11]](#footnote-11). Dakle, uz pomoć interpolacije, moguće je podesiti da se stringovske vrednosti na prikazu aplikacije menjaju.

Naslov prikaza komponente na kojoj se nalaze podaci o korisniku u sledećem slučaju menja se u zavisnosti od toga da li je korisnik ušao da modifikuje ili samo pogleda podatke o korisniku.

Struktura naslova u HTML fajlu:

  <mat-card-title>{{pageHeader}}</mat-card-title>

Vrednost promenljive pageHeader postavlja se u typescript fajlu komponente.

### **Jednostrano povezivanje: Od prikaza ka komponenti**

#### **Povezivanje događaja - Event binding**

Podrazumeva povezivanje u suprotnom smeru, od prikaza ka komponenti – gde događaj koji se desi bilo klikom na neki HTML element ili nekom drugom interakcijom, izvršava neku akciju.

Primer povezivanja događaja:

  <button mat-icon-button color="warn" (click)="switchLanguage('en')">  </button>

Klikom na dugme poziva se metoda switchLanguage kojom se menja jezik na veb aplikaciji.

### **Obostrano povezivanje**

Obostrano povezivanje koristi se za „prisluškivanje“ događaja - interakcije sa HTML elementima i dinamičko ažuriranje vrednosti nakon što se neki događaj desi.

Primer za obostrano povezivanje je dropdown lista iz koje se bira proizvod. Kada se proizvod izabere, događaj okida poziv metode updateData, koja, pored ostalog, popunjava i vrednost prozora dropdown liste sa izabranom stavkom.

HTML element:

<mat-select [formControl]="productsfrm"

(ngModelChange)="updateData($event)">

            <mat-option  \*ngFor="let product of products"

[value]="product.id" >{{product.name}}</mat-option>

  </mat-select>

Metoda u komponenti:

 updateData(target: any): void {

    if (target !== '' || this.products.length !== undefined) {

      this.products.forEach(product => {

         if (target === product.id) {

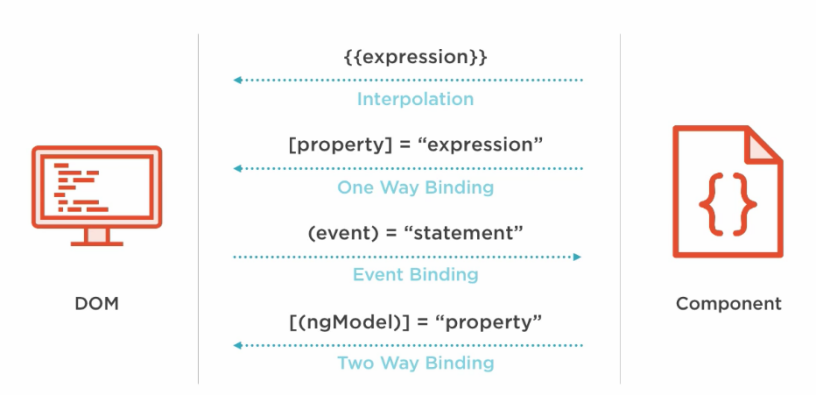
          this.selectedProductId = target;

          this.productForm.patchValue(product);

         }

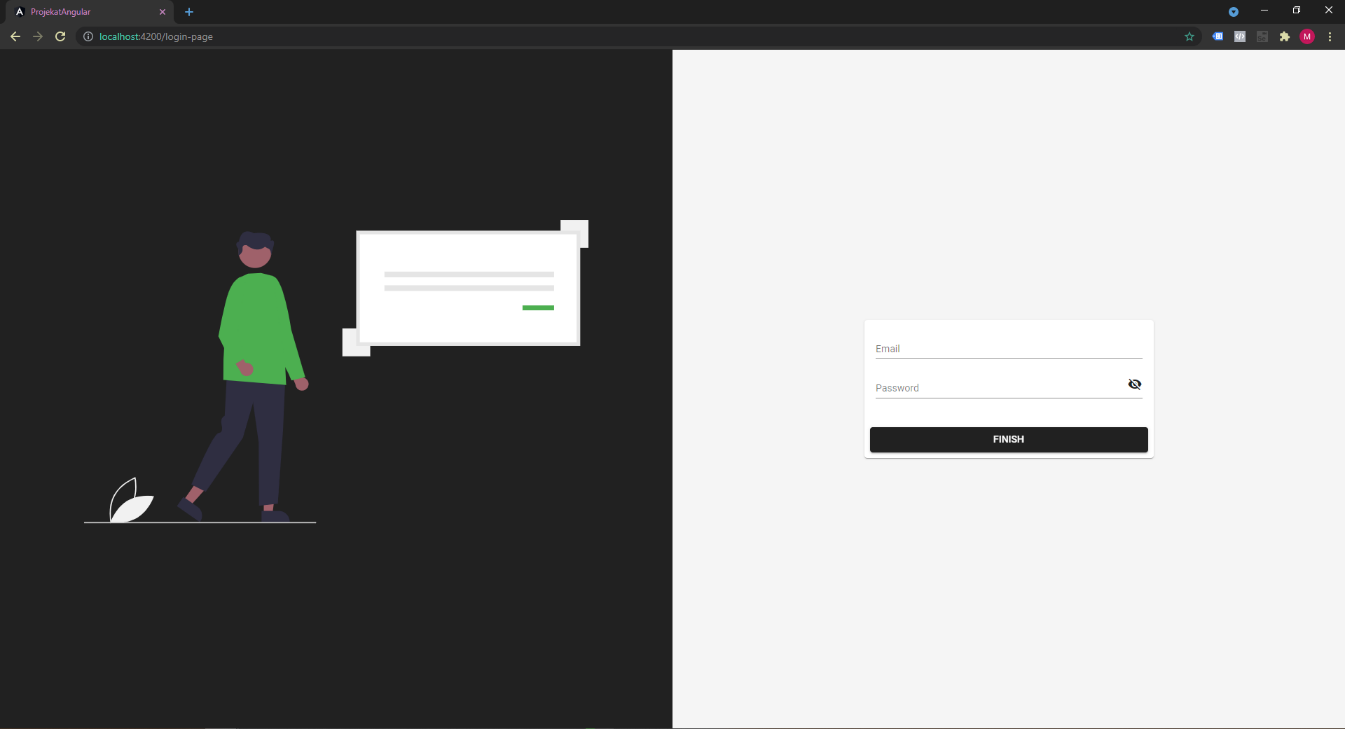
      });

    }

  }

*Slika 5. Povezivanje podataka u angular-u  
Izvor: https://ducmanhphan.github.io/2019-08-03-Data-binding-in-Angular/*

## **Stranica za prijavu**



Slika 6. Početna strana za prijavu

Prva stranica sa kojom se korisnik susreće jeste stranica za prijavu na aplikaciju. Da bi korisnik mogao da pristupi aplikaciju, dužan je da unese kombinaciju email adrese i odgovarajuće lozinke. Podaci o kredencijalima korisnika nalaze se u bazi podataka, te je za logovanje potrebno poklapanje kombinacije podataka unetih na formi sa podacima koji su u bazi. Svaki korisnik koji se nalazi u bazi podataka ima jedinstvenu email adresu, te se pri uspešnom logovanju na aplikaciju dalji rad korisnika i izgled delova aplikacije prilagođava shodno podacima ulogovanog korisnika.

Sve greške prilikom pokušaja logovanja, u smislu pogrešne kombinacije korisničkih kredencijala, email adrese neispravnog formata, nepopunjenih polja na formi i sl., prikazuju se korisniku u vidu poruka na samoj formi.

Strana za prijavu korisnika na aplikaciju struktuirana je na sledeći način:

<header></header>

<div id="content" fxLayout="row">

  <div id="left-side" fxLayout="row" fxLayoutAlign="center none">

    <img mat-card-image src="../../../assets/img/loginpic.svg" alt="Log in picture"/>

  </div>

  <div id="right-side">

    <mat-card class="login-card">

      <mat-card-content>

              <form [formGroup]="loginForm">

                <mat-form-field class="input-full-width">

                    <mat-label>Email</mat-label>

                    <input type="email" matInput [formControl]="emailInput" [errorStateMatcher]="matcher"

                           placeholder="Ex. pat@example.com">

                    <mat-error \*ngIf="emailInput.hasError('email') && !emailInput.hasError('required')">

                      Please enter a valid email address

                    </mat-error>

                    <mat-error \*ngIf="emailInput.hasError('required')">

                      Email is <strong>required</strong>

                    </mat-error>

                  </mat-form-field>

                  <mat-form-field class="input-full-width">

                    <mat-label>Password</mat-label>

                    <input type="password" matInput  placeholder="Password" [type]="hide ? 'password' : 'text'" [formControl]="passwordInput" [errorStateMatcher]="matcher">

                    <mat-icon matSuffix (click)="hide = !hide">{{

                            hide ? "visibility\_off" : "visibility"}}

                    </mat-icon>

                    <mat-hint \*ngIf="!passwordInput">Enter your password</mat-hint>

                    <mat-error \*ngIf="passwordInput.hasError('required')">

                      Password is <strong>required</strong>

                    </mat-error>

                  </mat-form-field>

              </form>

            </mat-card-content>

            <h3 class="error" \*ngIf="showMessage" fxLayout="row" fxLayoutAlign="center center">{{message}}</h3>

            <mat-card-actions>

              <button (click)="finishLogin()" id="loginButton" color="accent" mat-raised-button>FINISH</button>

            </mat-card-actions>

    </mat-card>

  </div>

</div>

Na desnom delu stranice nalazi se forma za prijavu, koja se sastoji od dva polja za slobodan unos. U prvom polju korisnik unosi email adresu, koja mora biti u formi email adrese. U drugom polju forme, korisnik unosi lozinku koja prilikom unosa ostaje skrivena. Pored dva polja za unos, na formi se još nalazi i dugme kojim se pokušava prijava na sistem.

Prilikom provere ispravnosti podataka unetih na formi, korišćen je interfejs sa dva svojstva: email adresom i lozinkom korisnika.

export interface Login {

  email: string;

  password: string;

}

U okviru login-page komponente korišćen je objekat tipa Login (interfejs) čije vrednosti se popunjavaju get metodama, koje kao povratnu vrednost vraćaju vrednosti korisničkog unosa sa forme za prijavu. Prilikom pritiska dugmeta za završetak prijave („FINISH“), uz pomoć login servisa, kojem se prosleđuju podaci uneti na formi, poziva se metoda login, koja proverava postojanje prosleđene kombinacije u bazi. Ukoliko su podaci na formi ispravno uneti, i ukoliko korisnik sa takvim podacima postoji u bazi, to korisnika automatski preusmerava na sledeću stranu i upisuje njegove podatke u lokalno skladište.

U slučaju da korisnik napravi grešku prilikom popunjavanja forme – ne unese email adresu i/ili lozinku u polje na formi, ili unese pogrešnu kombinaciju navedenih polja, metoda login vratiće grešku, i na samoj formi biće ispisan tekst greške shodno tome šta je korisnik pogrešio.

export class LoginPageComponent implements OnInit {

  public login: Login = {

    email: '',

    password: '',

  };

  showMessage = false;

  message = '';

  hide = true;

  loginForm: FormGroup = new FormGroup({

    email: new FormControl('', [Validators.email, Validators.required]),

    password: new FormControl('', [Validators.required, Validators.min(3)]),

  });

  matcher = new ErrorStateMatcher();

  isErrorState(control: FormControl | null, form: FormGroupDirective | NgForm | null): boolean {

    const isSubmitted = form && form.submitted;

    return !!(control && control.invalid && (control.dirty || control.touched || isSubmitted));

  }

  get emailInput(): any {

    return this.loginForm.get('email');

  }

  get passwordInput(): any {

    return this.loginForm.get('password');

  }

  constructor(private loginService: LoginService, private router: Router) {}

  ngOnInit(): void {

  }

f

  finishLogin(): void {

    this.loginService.login(this.loginForm.value).subscribe(

      (response) => {

        if (response != null) {

          if (response.role === true) {

            this.router.navigate([`/admin-home-page`]);

          }

          this.storeToLocalStorage(response);

        }

      },

      (error) => {

        this.showMessage = true;

        this.message = `The credentials you've entered don't seem to be valid. Try again.`;

        localStorage.setItem('isLoggedIn', 'false');

        console.log(error.error);

      }

    );

  }

  storeToLocalStorage(user): void {

    localStorage.setItem('isLoggedIn', 'true');

    localStorage.setItem('userObject', JSON.stringify(user));

  }

}

U okviru servisa za login postoji samo jedna metoda, koja komunicira sa serverom putem HTTP protokola, kako bi pristupila podacima na backend-u aplikacije.

U ovom slučaju, šalje post zahtev - metodi se prosleđuju podaci sa forme, dok povratna vrednost može biti:

1. objekat tipa korisnik – u slučaju da postoji takav korisnik u bazi podataka
2. greška o neispravnom zahtevu

Servis za prijavu na sistem:

export class LoginService {

  constructor(private http: HttpClient) { }

  login(login: Login): Observable<User> {

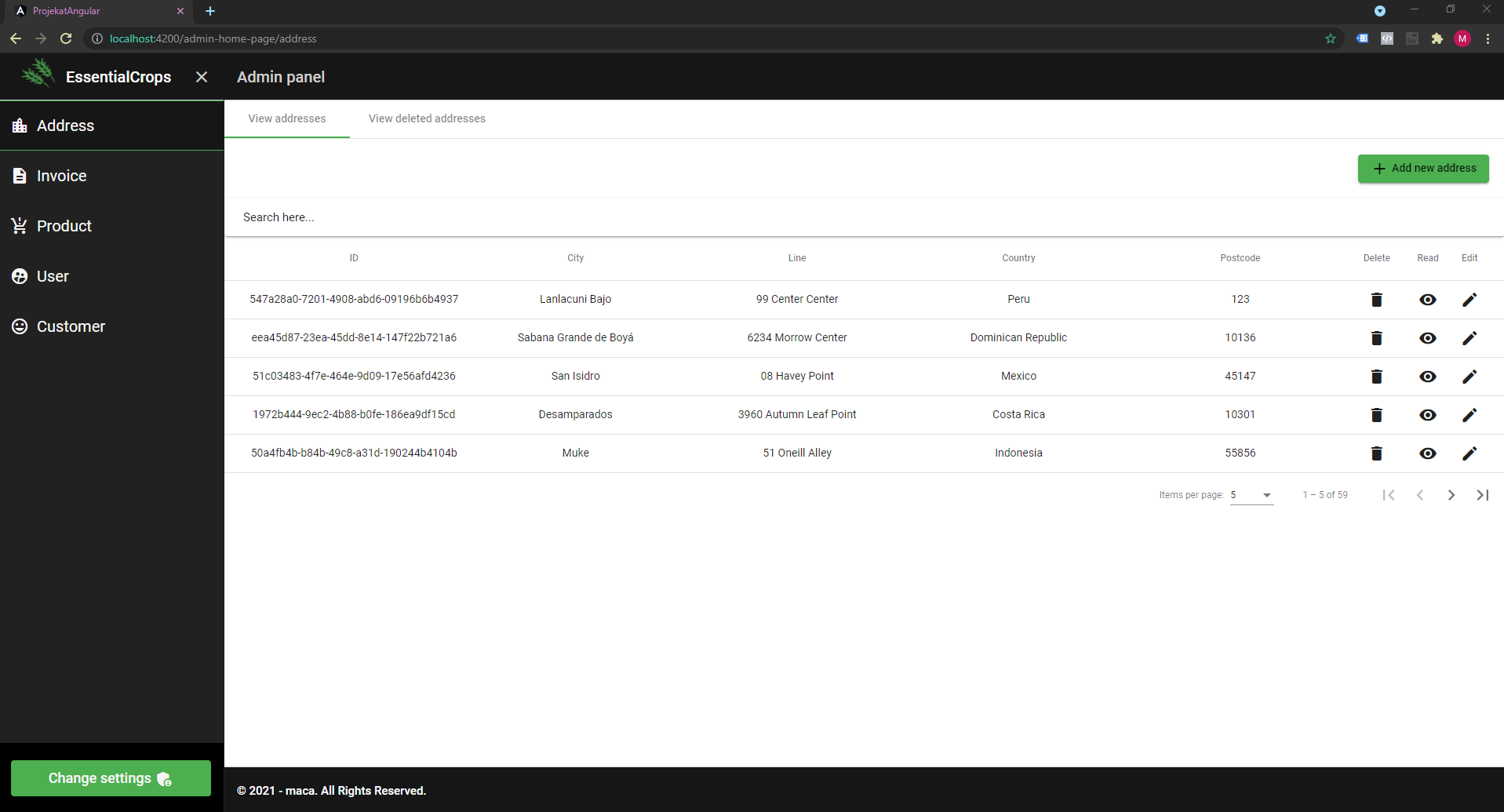
    return this.http.post<User>(`http://localhost:28846/api/User/logIn`, login);

  }

}

Ukoliko je prijava uspešna, korisnik se rutiranjem preusmerava na početnu stranu admin panela, a njegovi podaci pamte se u lokalnom skladištu veb pretraživača.

## **Početna strana admin panela**



Slika 7. Admin panel

Na početnoj strani admin panela, korisnik je u mogućnosti da, uz pomoć „fioke“ na levoj strani ekrana, sagleda sve komponente aplikacije. To je ujedno i meni za navigaciju kroz komponente veb aplikacije.

Admin panel je početna i ujedno i glavna strana aplikacije. Njen sadržaj je struktuiran na način da samu komponentu admin panela čine fioka sa leve strane, zaglavlje i podnožje strane. Centralni deo komponente dinamički učitava sadržaj drugog modula i njegove komponente, i to u zavisnosti od toga šta korisnik bira u meniju sa leve strane. Korisnik može sakriti vertikalni meni klikom na X dugme na samom vrhu menija. Takođe, pri dnu menija nalazi se dugme koje otvara dodatne opcije i podešavanjima, odakle korisnik može promeniti jezik veb aplikacije, otići na korisnički profil ili se može odjaviti sa sistema. Ukoliko je meni minimiziran, opcija za dodatna podešavanja i korisnički profil pojaviće se u desnom gornjem uglu zaglavlja aplikacije.

Svaki put kada korisnik uđe na admin panel, komponenta koja će se naći učitana u okviru centralnog dela jeste Adresa. Komponente se dinamički učitavaju putem router-outleta, koji „čuva“ mesto komponenti koja se učitava kada korisnik klikne na neku od stavki menija.

Klasa admin-home-page:

export class AdminHomePageComponent implements OnInit {

  @ViewChild('viewContainer', { read: ViewContainerRef }) viewContainer:

ViewContainerRef;

  selectedLang = true;

  row = '';

  selectedRow = 'Admin';

  showFiller = false;

  status = false;

constructor(private router: Router, private resolver:

ComponentFactoryResolver, private translate: TranslationService) {}

  ngOnInit(): void {

  }

  logOut(): void {

    localStorage.removeItem('isLoggedIn');

    this.router.navigate(['/login-page']);

  }

  switchLanguage(language: string): void {

    this.translate.switchLanguage(language);

  }

}

Struktura stavki menija u kojima se, u okviru direktive [routerLink] prosleđuje ruta:

        <mat-toolbar-row id="Address" [routerLink]="['address']" [routerLinkActive]="['highlight']">

          <mat-icon aria-hidden="false">location\_city</mat-icon>

          <mat-label class="mat-icon-span">{{"Address" | translate}}  </mat-label>

        </mat-toolbar-row>

        <mat-toolbar-row id="Invoice" [routerLink]="['invoice']" [routerLinkActive]="['highlight']">

          <mat-icon aria-hidden="false">description</mat-icon>

          <span  class="mat-icon-span">{{"Invoice" | translate}}</span>

        </mat-toolbar-row>

        <mat-toolbar-row id="Product" [routerLink]="['product']"  [routerLinkActive]="['highlight']"> <mat-icon aria-hidden="false" >production\_quantity\_limits</mat-icon><span  class="mat-icon-span">{{"Product" | translate}}</span>

        </mat-toolbar-row>

        <mat-toolbar-row id="User" [routerLink]="['user']" [routerLinkActive]="['highlight']">

          <mat-icon aria-hidden="false" >supervised\_user\_circle</mat-icon>

          <span class="mat-icon-span">{{"User" | translate}}</span>

        </mat-toolbar-row>

        <mat-toolbar-row id="Customer" [routerLink]="['customer']" [routerLinkActive]="['highlight']">

          <mat-icon aria-hidden="false">tag\_faces</mat-icon>

          <span  class="mat-icon-span">{{"Customer" | translate}}</span>

        </mat-toolbar-row>

## **Rutiranje**

Kretanje korisnika kroz aplikaciju, odnosno, navigacija od jedne do druge komponente klikom na link ili dugme sprovodi se upotrebom rutiranja. Rutiranje u Angular-u omogućava Router modul koji je smešten u okviru biblioteke @angular/router. U modulima za rutiranje definišu se rute aplikacije.

Da bi rutiranje funkcionisalo, neophodno je uvesti RouterModule i Router unutar modula za rutiranje koji se koriste. Rute govore Angularu koja se komponenta (prikaz) treba učitati kada se promeni URL putanja u pretraživaču. Ruta na koju korisnik prvo dolazi, je <http://localhost:4200>, odnosno, to je ruta na kojoj se nalazi stranica za prijavu. Sve ostale rute se definišu u okviru niza objekata koje imaju dva ili više svojstava. Prvo svojstvo je URL putanja, dok je drugo svojstvo komponenta koja će se učitati odlaskom na definisanu putanju. Poslednji korak jeste deklarisanje mesta gde će se ruta renderovati, ali i dodati linkove koji će voditi do željenih putanja. Za to se koriste direktiva router-outlet – koja definiše mesto na kom će se učitati komponenta, i direktiva routerLink koja povezuje dugme ili bilo koji drugi HMTL element sa rutom.

U posmatranoj aplikaciji rutiranje je odrađeno upotrebom „lenjog“ rutiranja. To podrazumeva da se u modulu za rutiranje aplikacije, rute učitavaju u zavisnosti od toga koji se modul koristi. U ovom slučaju, trenutno su učitane rute admin-home-page modula koje su se učitale u trenutku kada je korisnik uspeo da se uloguje na aplikaciju. Ovo se smatra dobrom praksom kada su u pitanju kompleksne aplikacije koje imaju veliki broj ruta. Ukoliko korisnik pokuša otići na neku rutu koja nije definisana ni u jednom modulu, on će automatski biti preusmeren na sadržaj PageNotFound komponente.

import { NgModule } from '@angular/core';

import { Routes, RouterModule } from '@angular/router';

import { HomePageComponent } from './features/home-page/home-page.component';

import { AdminHomePageComponent } from './features/admin-home-page/admin-home-page.component';

import { PageNotFoundComponent } from './features/page-not-found/page-not-found.component';

import { LoginPageComponent } from './features/login-page/login-page.component';

const routes: Routes = [

  { path: '', redirectTo: '/login-page', pathMatch: 'full' },

  { path: 'login-page', component: LoginPageComponent },

  { path: 'home-page', component: HomePageComponent },

  { path: 'admin-home-page', component: AdminHomePageComponent,

  loadChildren: () => import('../app/features/admin-home-page/admin-home-page.module').then(m => m.AdminHomePageModule)},

  { path: '\*\*', component: PageNotFoundComponent},

];

@NgModule({

  imports: [RouterModule.forRoot(routes)],

  exports: [RouterModule],

  providers: []

})

export class AppRoutingModule { }

Učitavanje nekog modula tek prilikom njegovog korišćenja značajno može uticati na performanse aplikacije. U slučaju da se ne koristi lenjo učitavanje modula, svi moduli se učitavaju odmah prilikom učitavanja aplikacije, iako većina modula u tom trenutku nepotrebna. U okviru promenljive routes navode se sve komponente i njihove rute. U okviru glavnog modula za rutiranje aplikacije (AppRoutingModule), vrši se lenjo učitavanje modula početne strane admin panela (AdminHomePageModule) upotrebom loadChildren sintakse i putanje do modula kojeg želimo da učitamo.

Po istoj analogiji odrađeno je i lenjo učitavanje ostalih modula koji ste koriste u okviru admin panela. Klikom na neki od linkova sa vertikalnog menija, učitava se odgovarajući modul i komponente u okviru njega. Definisanjem modula na ovaj način, svaki modul nalazi se u zasebnom fajlu, a skida se i poziva individualno i po potrebi.

Putanje i lenjo očitavanje modula u okviru AdminHomePage panela:

const routes: Routes = [

  { path: '', component: AddressComponent  },

  { path: 'address', component: AddressComponent },

  { path: 'user', component: UserComponent,  },

  { path: 'invoice', component: InvoiceComponent },

  { path: 'product', component: ProductComponent },

  { path: 'customer', component: CustomerComponent },

  { path: 'user-profile', component: UserProfileComponent},

  { path: 'user-profile-edit', component: UserProfileEditComponent},

  { path: 'user', component: UserComponent,

  loadChildren: () => import('../user/user-routing.module').then(m => m.UserRoutingModule)},

  { path: 'address', component: AddressComponent,

  loadChildren: () => import('../address/address-routing.module').then(m => m.AddressRoutingModule)},

  { path: 'customer', component: CustomerComponent,

  loadChildren: () => import('../customer/customer-routing.module').then(m => m.CustomerRoutingModule)},

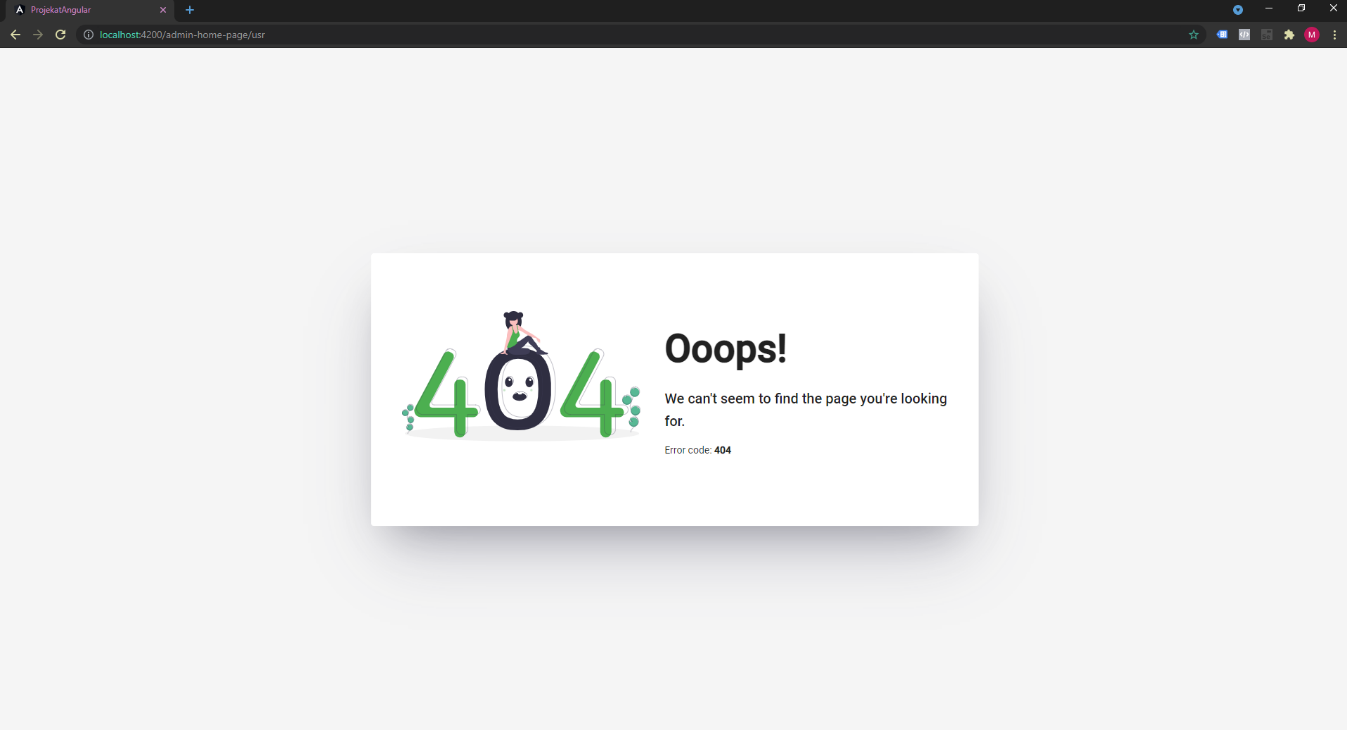
  { path: 'product', component: ProductComponent,

  loadChildren: () => import('../product/product-routing.module').then(m => m.ProductRoutingModule)},

  { path: 'invoice', component: InvoiceComponent,

  loadChildren: () => import('../invoice/invoice-routing.module').then(m => m.InvoiceRoutingModule)}];

Ukoliko korisnik ručno upiše URL adresu u polju za pretragu, koja nije definisana u nekom od modula za rutiranje, biće preusmeren na sledeću pogled komponente Greška 404.



Slika 8. Stranica koja se učitava ruta nije pronađena

## **Servisi**

Komponente ne bi trebalo da direktno pristupaju i upravljaju podacima na serveru. Ukoliko postoje podaci i logika koja se koristi na više mesta u aplikaciji, ona se izmešta u posebnu klasu - servis. Servisi su klase čija je namena da komuniciraju sa bekendom i vraćaju podatke komponentama, koje te podatke dalje prikazuju i koriste.Da bi se metode koje se nalaze u servisima mogle koristile u okviru komponenti, neophodno ih je "ubrizgati" u konstruktor komponente. Na taj način, kreira se instanca klase(servisa) koja se dalje koristi za pozivanje metoda kreiranih u servisu.

Servisi sa serverom komuniciraju putem HttpClient-a (HTTP protokola) kako bi pristupili podacima. Za dobijanje podataka sa servera, koristi se metoda get. Ovom metodom šalje se zahtev za podacima, a metoda vraća rezultat koji se dalje obrađuje. Tip povratne vrednosti get metode, ukoliko ona postoji, definiše se putem observable-a - interfejsa putem kog se upravlja različitim vrstama asinhronih operacija i tipom povratne vrednosti.

Pored get metode, postoje još i PUT, POST i DELETE i PATCH HTTP metoda, koje se koriste u upravljanju podacima. POST zahtev najčešće se šalje prilikom dodavanja novog podatka u bazu. POST metod sličan je GET metodu, prima parametar kojim se može naznačiti tip povratne vrednosti. Pored standardnog URL parametra, može primiti i dva dodatna parametra: "telo" - podatak koji se šalje u telu zahteva i objekat koji se sastoji od ostalih opcija.

Delete metoda koristi se za brisanje podataka, dok se patch i put metode koriste za ažuriranje podataka.

U nastavku su prikazane GET, POST, PUT i DELETE metode u okviru user servisa.

1. **GET** zahtev, getUser metoda – Metoda koja vraća jednog korisnika. Metodi se prosleđuje parametar - id korisnika, a povratna vrednost funkcije je objekat tipa User. Ova metoda poziva može se pozvati u okviru bilo koje komponente, a povratna vrednost smešta se u promenljivu sa kojom se dalje radi.

  getUser(id: any): Observable<User>  {

    return this.http.get<User>(`http://localhost:28846/api/User/${id}`);

  }

2. **POST** zahtev, addUser - Metodi se prosleđuje objekat tipa UserForm - koji se sastoji od podataka o korisniku i adresi, a povratna vrednost jeste objekat tipa User.

  addUser(singupForm: UserForm): Observable<User> {

    return this.http.post<User>(`http://localhost:28846/api/User/singUp`, singupForm);

  }

3. **DELETE** zahtev, deleteUser – Ovom metodom vrši se brisanje korisnika na osnovu prosleđenog id broja korisnika i ova metoda nema povratnu vrednost.

  deleteUser(id: any): Observable<{}>  {

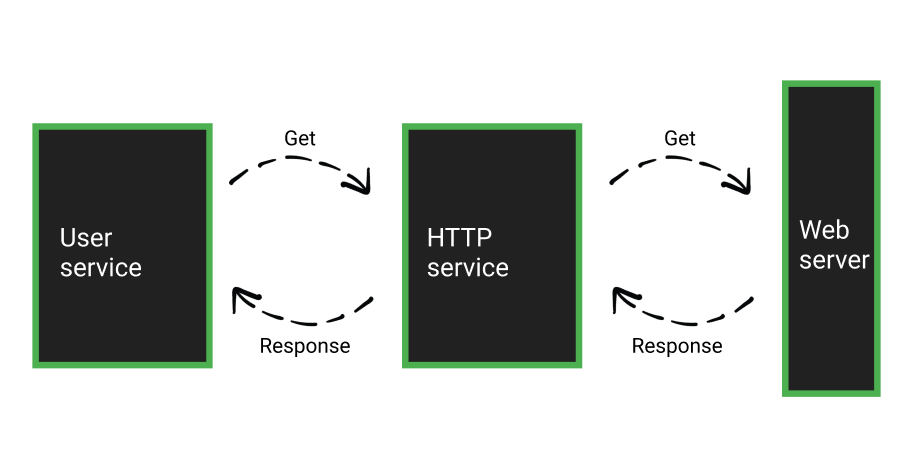
    return this.http.delete(`http://localhost:28846/api/User/softDelete/${id}`);

  }

4. **PUT** zahtev, updateUser - Zahtev kojim se ažuriraju podaci o korisniku. Metodi se prosleđuje objekat tipa Korisnik (koji sadrži promenjene podatke) ali i id korisnika na osnovu kog se korisnik pronalazi i ažurira u bazi podataka. Povratna vrednost zahteva jeste objekat u kom se nalaze novi, izmenjeni podaci korisnika.

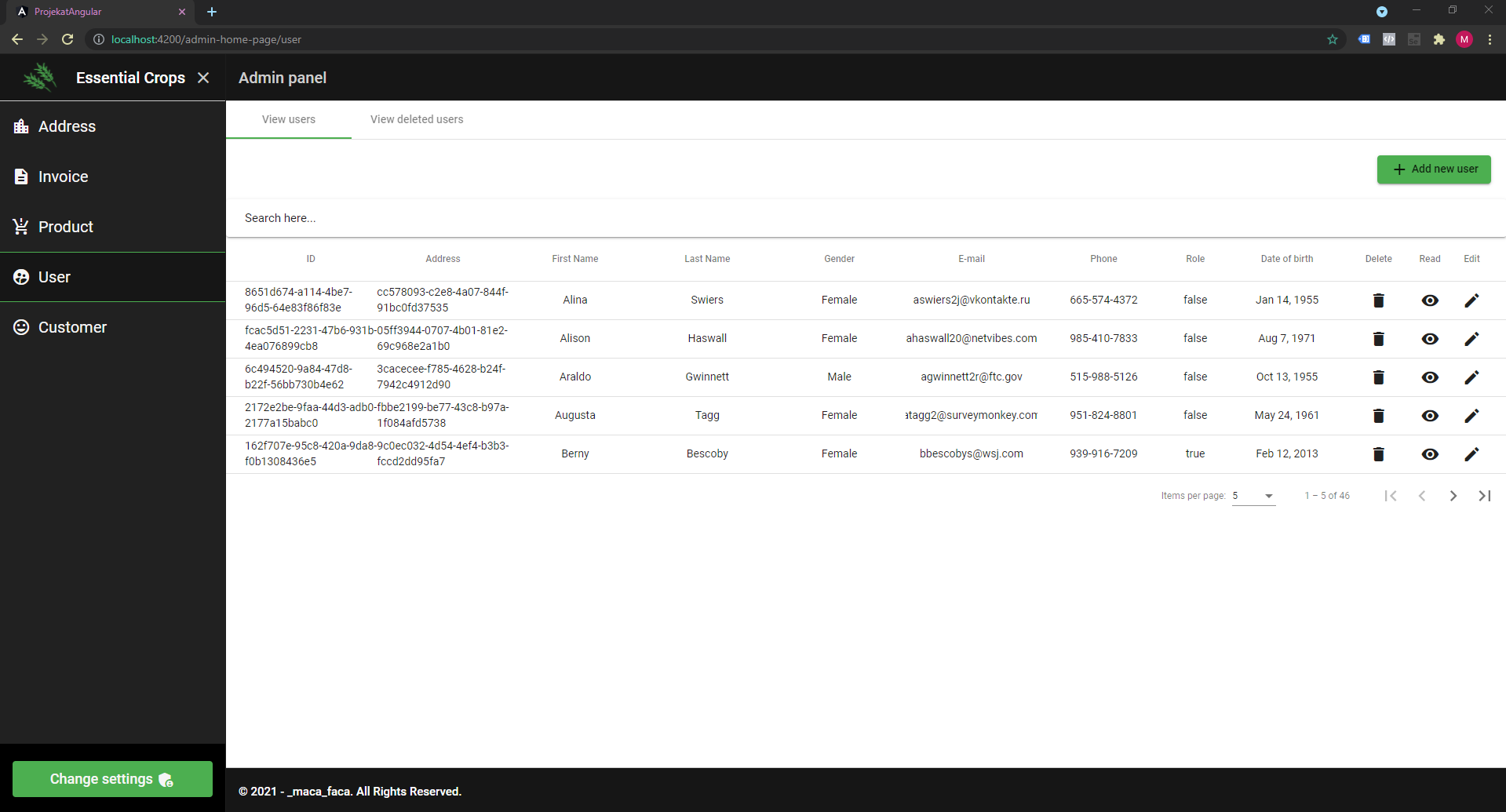
  updateUser(user: User, id: any): Observable<User> {

    return this.http.put<User>(`http://localhost:28846/api/User/${id}`, user);

  }

Slika 9. Komunikacija servisa sa serverom posredstvom HTTP servisa

## **Komponenta „Korisnik“**



Slika 10. Prikaz komponente korisnik unutar admin panela

Prikaz komponente korisnika otvara se klikom na link menija, ili ukucavanjem putanje

<http://localhost:4200/admin-home-page/user>. Na stranici korisnika nalaze se podaci o svim korisnicima koji se nalaze u bazi podatka. Podaci su prikazani u vidu tabele, i to podeljeni na 2 jezička. Na prvom jezičku nalaze se podaci o korisnicima, dok se na drugom jezičku nalaze podaci o obrisanim korisnicima. Logika aplikacije je takva, da nijedan korisnik ne može permanentno obrisati nijednu stavku iz baze podataka. Taj problem rešen je korišćenjem metode softDelete, odnosno „mekog“ brisanja, gde se prilikom brisanja bilo koje stavke sa veb aplikacije, toj stavci u bazi menja vrednost svojstva IsDeleted sa 0 na 1. To predstavlja indikator da je torka obrisana, te se ona prilikom ponovnog popunjavanja tabele više ne nalazi u tabeli u kojoj se nalazila, već se premešta na drugi jezičak – u tabelu obrisanih podataka.

Metoda za popunjavanje tabele sa podacima korisnika

  private showUsers(parametars: any): void{

    this.userService.getUsers(this.parametars).subscribe((users) => {

      const metadata = users['metadata'];

      this.pageSize = metadata.pageSize;

      this.currentPage = metadata.currentPage;

      this.currentPage = this.currentPage - 1;

      this.totalSizeOfItems = metadata.totalCount;

      const listOfUsers = users['pagedList'];

      this.dataSource = new MatTableDataSource(listOfUsers);

    });

  }

Ova metoda poziva se prilikom učitavanja komponente tabele korisnika. Na osnovu instance korisničkog servisa, poziva se metoda getUsers, kojoj se prosleđuju odgovarajući parametri a koja kao rezultat vraća niz korisnika. Parametri koji se šalju metodi jeste broj strane od koje se žele prikazivati podaci (pageNumber) i broj stavki po strani (pageSize). Parametri su obavezni zbog postojanja paginacije u okviru tabele korisnika.

parametars: any = {

    pageNumber: 1,

    pageSize: 5

  };

Paginator se prikazuje namom dnu tabele u HTML fajlu i njegova struktura je sledeća:

<mat-paginator

          #paginator

          [pageSize]="pageSize"

          [pageSizeOptions]="[5, 10, 20, totalSizeOfItems]"

          [showFirstLastButtons]="true"

          [length]="totalSizeOfItems"

          [pageIndex]="currentPage"

          (page)="pageEvent = handlePage($event)"

        >

</mat-paginator>

Korisnik na dnu tabele ima opciju promene broja stavki u tabeli, promenu „strana“ tabele: odlazak na prvu, sledeću, prethodnu, ali i na poslednju stranu.

 handlePage(event: any): void {

    this.parametars.pageSize = event.pageSize;

    this.parametars.pageNumber = event.pageIndex + 1;

    this.showUsers(this.parametars);

  }

Bilo kakva promena vezana za paginator, pozvaće metodu handlePage, koja ažurira vrednost parametara i poziva metodu za prikaz podataka shodno novim vrednostima parametara.

Na isti način funkcioniše i popunjavanje tabele obrisanih korisnika:

private showDeletedUsers(parametars: any): void {

    this.userService.getDeletedUsers(this.parametars).subscribe((users) => {

      const metadata = users['metadata'];

      this.pageSize = metadata.pageSize;

      this.currentPage = metadata.currentPage;

      this.currentPage = this.currentPage - 1;

      this.totalSizeOfItems = metadata.totalCount;

      const listOfUsers = users['pagedList'];

      this.dataSource = new MatTableDataSource(listOfUsers);

    });

  }

Svaka torka u tabeli može biti obrisana, otvorena u modu za čitanje ili izmenjena. U zavisnosti od akcije koju izabere korisnik, učitaće se odgovarajući prikaz. Prilikom klika na ikonicu za brisanje stavke, poziva se komponenta koja se prikazuje kao modalni prozor.

  openDeleteModal(id: any): void{

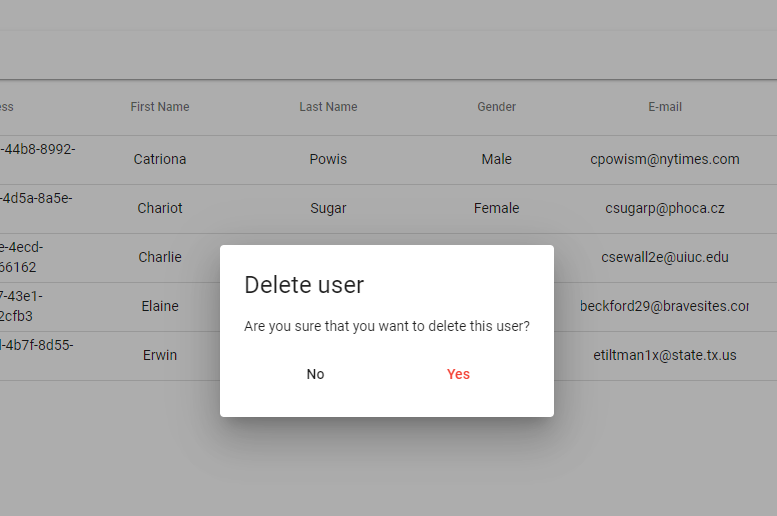
    const dialogRef = this.dialog.open(DeleteModalComponent, {data: {id : id, component: 'user'}});

    dialogRef.afterClosed().subscribe(x => {

      this.showUsers(this.parametars);

    });

   }

Modalnom prozoru šalje se id stavke koju korisnik pokušava obrisati i naziv komponente. Naziv komponente prosleđuje se zbog toga što je modalni prozor za brisanje stavki jedna od zajedničkih komponenti. To je komponenta koja se nalazi u shared folderu, i koju je moguće koristiti u različitim modulima. Na modalnom prozoru nalaze se dva dugmeta: dugme „No“ kojim se zatvara modalni prozor i dugme „Yes“ kojim se nastavlja ka mekom brisanju stavke.

Slika 11. Modalni prozor za brisanje stavke

softDelete(id: any): void{

    if (this.data.component === 'user') {

      this.userService.softDeleteUser(id).subscribe();

    } else if (this.data.component === 'address') {

      this.addressService.softDeleteAddress(id).subscribe();

    }else if (this.data.component === 'invoice') {

      this.invoiceService.softDeleteInvoice(id).subscribe();

    }else if (this.data.component === 'product') {

      this.productService.softDeleteProduct(id).subscribe();

    }else if (this.data.component === 'customer') {

      this.customerService.softDeleteCustomer(id).subscribe();

    }

    this.dialogRef.close();

  }

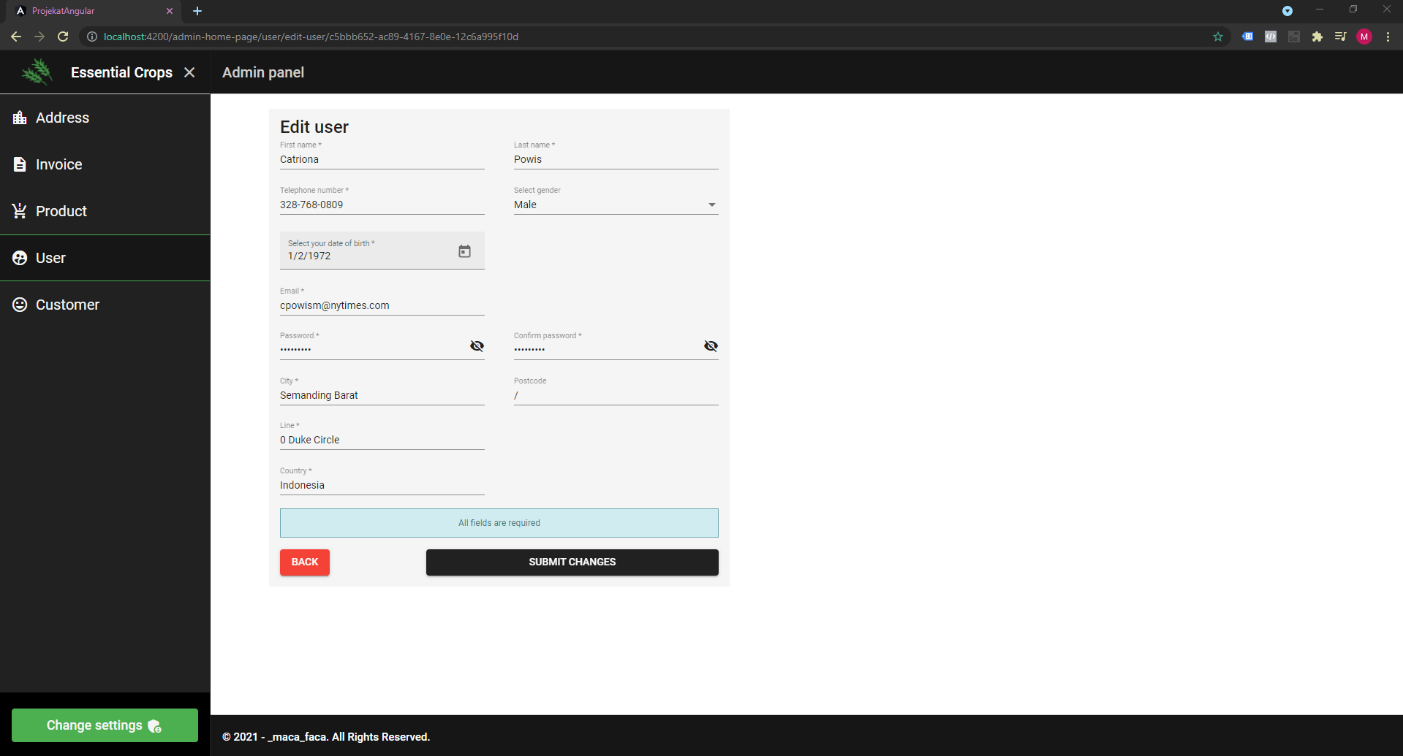
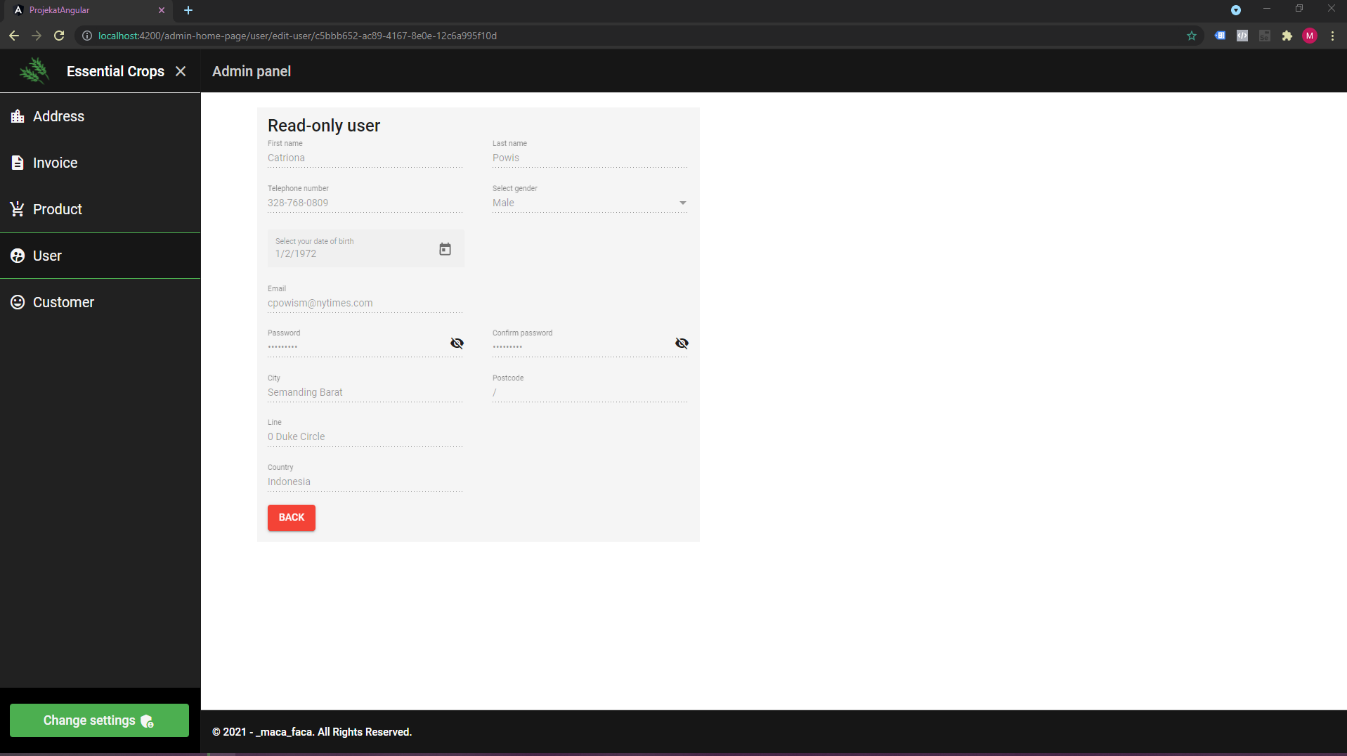
  closeDialog(): void {

    this.dialogRef.close();

  }

U zavisnosti od toga koja komponenta stigne kao parametar prilikom otvaranja modalnog prozora, putem instance odgovarajućeg servisa, poziva se metoda za brisanje stavke u slučaju da korisnik na modalnom prozoru potvrdi brisanje stavke.

Klikom na ili ikonicu neke od stavki tabele, otvara se komponenta za modifikovanje korisnika. Reč je o jednoj komponenti, user-edit, koja se otvara u edit ili read-only prikazu, u zavisnosti od toga šta korisnik odabere. Ako se otvori u prikazu za čitanje, forma sa podacima korisnika je zaključana, i tada postoji samo dugme koje omogućava vraćanje na prethodnu stranu. U slučaju da se korisnik odluči za modifikaciju, otvara se forma za promenu podataka izabrane stavke iz tabele.



Slika 12. Komponenta za modifikovanje i čitanje podataka korisnika

U oba slučaja, prilikom učitavnja komponente, popunjavaju se dve forme. Prva forma sastoji se od podataka o korisniku: ime, prezime, broj telefona, pol, datum rođenja ali i kredencijali za prijavu na sistem. Pored ličnih podataka korisnika, na drugoj formi, nalaze se podaci o adresi stanovanja korisnika – grad, ulica i broj, poštanski broj i država. Ovi podaci nalaze se smešteni u poljima na formi i popunjavaju se na osnovu id-a korisnika, koji se prosleđuje klikom na dugmad koja se nalaze u redu tabele.

 ngOnInit(): void {

    if (localStorage.getItem('lockedItem') === null) {

      this.pageHeader = 'Read-only user';

      this.isButtonVisible = false;

      this.detailForm.disable();

      this.addressForm.disable();

    }

    this.id = this.getUrlParams();

    this.userService.getUser(this.id).subscribe((user) => {

      this.user.addressId = user.addressId;

      this.user.role = user.role;

      this.user.id = user.id;

      this.address.id = user.addressId;

      this.selected = user.gender;

      this.userAddressId = user.addressId;

      this.detailForm.patchValue(user);

      this.detailForm.patchValue({

        date: user.dateOfBirth,

        confirmPassword: user.password

      });

      this.patchAddressForm();

      this.initalValues = this.detailForm.value;

    });

  }

  patchAddressForm(): void {

    this.userService.getUserAddress(this.userAddressId).subscribe((address) => {

      this.addressForm.patchValue(address[0]);

      this.initalValuesAddress = this.addressForm.value;

    });

  }

Prilikom izmene podataka, neophodno je da svako polje na obe forme bude popunjeno. Ukoliko to nije slučaj, na dnu forme pojaviće se indikator da je potrebno popuniti sva polja.

U slučaju da korisnik klikne na dugme „Back“ za povratak na prethodnu stranu, ukoliko je modifikovao bilo šta na formama, otvoriće se modalni prozor koji obaveštava korisnika o odustajanju od promena. Klikom na dugme „Submit changes“ vrši se ažuriranje podataka o korisniku i povratak na stranicu sa tabelom korisnika.

 submit(): void {

    this.fillOutForm();

    if (this.detailForm.valid && this.addressForm.valid) {

      this.userService.updateUser(this.user, this.id).subscribe(

        (response) => {

          this.updateAddress(this.userAddressId);

          this.formFilled = true;

          localStorage.removeItem('lockedItem');

          this.unlockItem(this.id);

          this.openSnackBar();

          this.router.navigate(['admin-home-page/user']);

        },

        (error) => {

          this.formFilled = false;

          console.log(error.error);

        }

      );

    } else {

      this.formFilled = false;

    }

  }

Takođe, nakon uspešne izmene podataka, otvara se snackBar (u vidu poruke na dnu ekrana), koji korisnika obaveštava o uspešnoj izmeni podataka.

  openSnackBar(): void {

    this.snackBar.open('User successfully edited!', 'Close', {

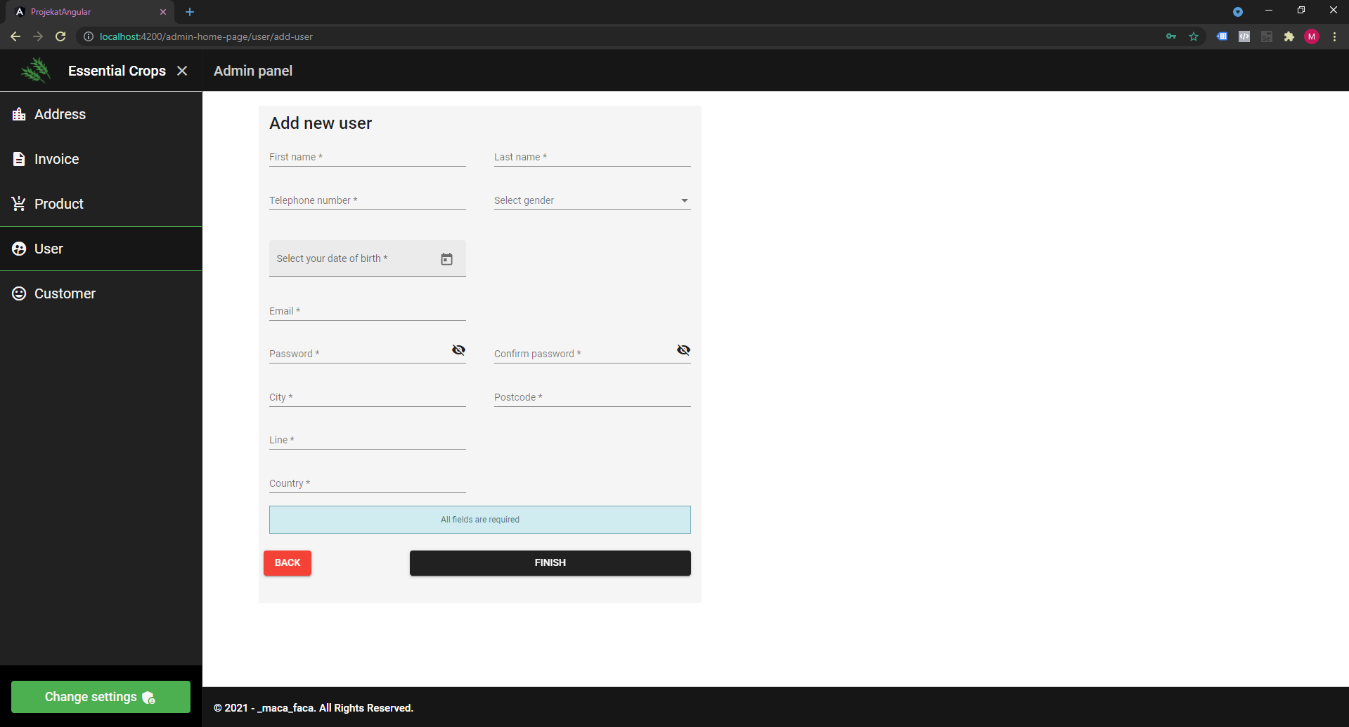
      duration: 2000,

      panelClass: ['snackbar']

    });

  }

Pored opcije za ažuriranje i brisanje korisnika, postoji i mogućnost dodavanja novog korisnika. Klikom na dugme „Add new user“, koje se nalazi u desnom uglu iznad tabele, otvara se prikaz za dodavanje novog korisnika.



Slika 13. Forma za dodavanje novog korisnika

Kao i u slučaju izmene korisnika, da bi se dodao novi korisnik, neophodno je popuniti sva polja na obe forme. Prilikom dodavanja novog korisnika u bazu, takođe se dodaje i nova adresa. Podaci sa obe forme upisuju se u promenljivu tipa UserForm(interfejs koji se sastoji od 2 data transfer object-a – korisnika i adrese), koja se, kao parametar, prosleđuje metodi addUser i kojom se vrši upis novog korisnika i njegove adrese u bazu podataka.

finishSingup(): void {

    if (this.checkPasswords) {

      this.fillOutForm();

      this.userService.addUser(this.singup).subscribe(

        (response) => {

          this.formFilled = true;

          this.openSnackBar();

          this.router.navigate(['admin-home-page/user']);

        },

        (error) => {

          this.formFilled = false;

          console.log(error.error);

        }

      );

    } else {

      this.formFilled = false;

    }

  }

  fillOutForm(): void {

    this.singup.UserDto.firstName = this.singupForm.value.firstName;

    this.singup.UserDto.lastName = this.singupForm.value.lastName;

    this.singup.UserDto.gender = this.singupForm.value.gender;

    this.singup.UserDto.dateOfBirth = this.singupForm.value.date;

    this.singup.UserDto.email = this.singupForm.value.emailSingUp;

    this.singup.UserDto.phone = this.singupForm.value.phoneNumber;

    this.singup.UserDto.password = this.singupForm.value.passwordSingUp;

    this.singup.AddressDto.country = this.singupForm.value.country;

    this.singup.AddressDto.city = this.singupForm.value.city;

    this.singup.AddressDto.postcode = this.singupForm.value.postcode;

    this.singup.AddressDto.line = this.singupForm.value.line;

  }

## **Zaključavanje stavki**

Pošto postoji mogućnost da više korisnika istovremeno koriste veb aplikaciju, može se desiti da više korisnika istovremeno želi modifikovati neku stavku. Nastanak ovog problema sprečava se korišćenjem zaključavanjem stavki prilikom otvaranja stavke u modu za modifikovanje.

U okviru prikaza tabele sa korisnicima kreirana je promenljiva lockedItem – tipa LockedItem, koja se sastoji od dva svojstva: id-a korisnika koji je trenutno ulogovan na sistem i koji pokušava da modifuje podatke, i id-a stavke u tabeli koju pokušava da modifikuje.

  lockedItem: LockItem = {

    itemId: '',

    userId: ''

  };

Prilikom inicijalizacije komponente tabele korisnika, iz lokalnog skladišta, popunjava se promenljiva korisnički id.

  ngOnInit(): void {

    this.user = JSON.parse(localStorage.getItem('userObject'));

    this.lockedItem.userId = this.user.id;

    this.editIndicator = true;

    this.showUsers(this.parametars);

  }

U okviru metode u kojoj se popunjavaju podaci u tabeli, poziva se metoda kojoj se kao parametar prosleđuju svi korisnici, a koja sve korisnike koji se trenutno modifikuju dodaje u niz „zaključanih“ korisnika.

  getLockedUsers(allUsers: any): void {

    allUsers.forEach(user => {

      this.lockService.getIsItemLocked(user.id).subscribe((res) => {

        if (res) {

          this.listOfLockedUsers.push(user.id);

        }

      });

    });

  }

Ukoliko korisnik pokuša da otvori stranu za editovanje neke od stavki tabele koju već neko menja, to mu neće biti dozvoljeno. Kada neki korisnik otvori stranicu za modifikovanje, klik na dugme za edit poziva metodu openUserEditPage, kojoj se prosleđuje id stavke i putem metode getIsItemLocked vrši se provera da li je ta stavka već zaključana. Ukoliko je rezultat funkcije ne, odnosno, ukoliko ta stavka nije zaključana, tada se korisniku dozvoljava izmena željene stavke, on se preusmerava na stranicu za modifikovanje, a ta stavka se zaključava pozivom metode lockItem.

  openUserEditPage(id: any): void{

      this.lockService.getIsItemLocked(id).subscribe((result) => {

        if (result === false) {

              this.router.navigate([`/admin-home-page/user/edit-user/${id}`]);

              this.lockItem(id);

        } else {

          this.openSnackBar();

        }

      });

  }

  lockItem(id: any): void {

    this.lockedItem.itemId = id;

    this.lockService.postLockItem(this.lockedItem).subscribe();

  }

Ukoliko je rezultat potvrdan, odnosno, ukoliko se stavka već nalazi u fazi modifikovanja, korisnik će ostati na istoj strani i dobiće poruku o tome na dnu ekrana.

 openSnackBar(): void {

    this.snackBar.open('User is currently being modified! Try again later.', 'Close', {

      duration: 4000,

      panelClass: ['snackbar']

    });}

Pored poruke o tome da je stavka zaključana, postoji i indikator u redu tabele u kojoj se sama stavka nalazi. Umesto simbola, sve stavke koje su trenutno u fazi modifikovanja biće označene sledećim simbolom: . Na taj način će korisnik imati uvid u sve stavke koje se trenutno modifikuju.

HTML struktura dugmeta:

<mat-icon [color]="editIndicator == true &&

this.listOfLockedUsers.includes(element.id)  ? 'warn' : ''">mode\_edit

</mat-icon>

Stavke se otključavaju u slučaju kada korisnik završi sa modifikovanjem stavke – klikom na dugme „submit changes“, ili klikom na „back“ dugme. Takođe, ukoliko dođe do prinudnog napuštanja stranice za izmenu stavke, to će isto otključati stavku. U svakom od navedenih slučajeva, poziva se metoda unlockItem.

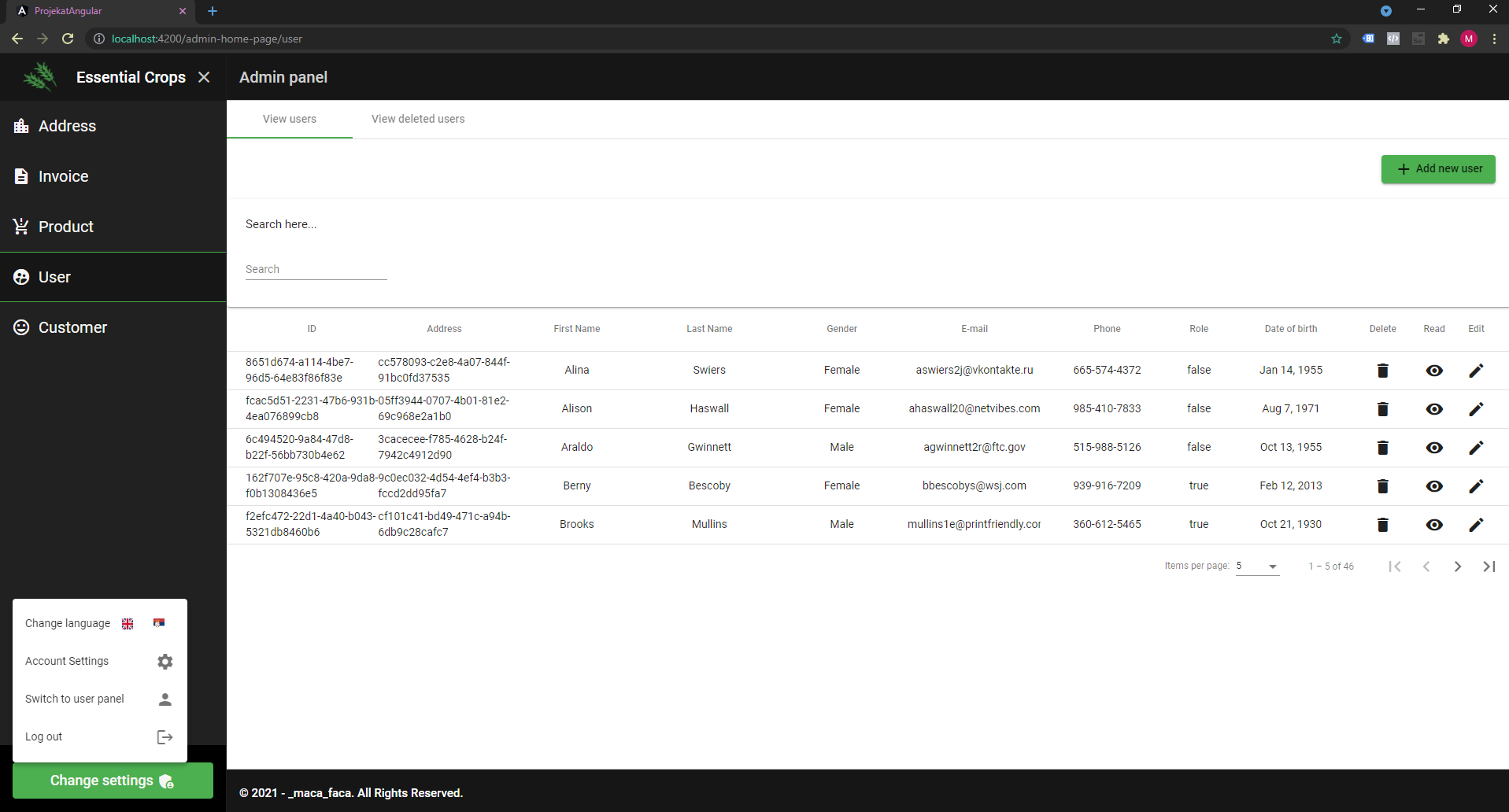
unlockItem(lockedItemId: any): void {

    this.lockService.postUnlockItem(lockedItemId).subscribe();

}

## **Lokalizacija**

Klikom na dugme Settings koje se nalazi na dnu vertikalnog menija, korisnik ima opciju da aplikaciju prevede sa engleskog na srpski jezik i obrnuto.



Slika 14. Veb aplikacija na engleskom jeziku

U servisu za prevođenje kreirana je metoda koja, prilikom klika na dugme za promenu jezika, proverava jednakost labela sa podacima u json fajlu. JSON fajl sadrži niz ključ-vrednost parova, te se provera vrši na način da se vrednost labele upoređuje sa ključem, i u slučaju poklapanja, vrednost labele menja se vrednošću odgovarajućeg ključa.

Metoda prima vrednost koju želimo da prevedemo i mapira je na željeni rezultat:

public response;

  private jsonURL = '../../../assets/i18n/sr.json';

public getJSONfile(value: any): any {

    return this.getJSON().pipe(map((res) => {

        for (let key in res) {

          if (key === value) {

            this.response = res[key];

            return this.response;

          }

        }

      })

    );}

  public getJSON(): Observable<any> {

    return this.http.get(this.jsonURL)}

Deo sr.JSON fajla:

{

  "Address": "Adresa",

  "Product": "Proizvod",

  "User": "Korisnik",

  "Customer": "Kupac",

}

Za transformaciju datuma, brojčanih iznosa, stringova i ostalih podataka koriste se pipe-ovi. Za prevođenje labela u tu svrhu je kreiran pipe I u okviru njega metoda kojoj se kao ulazni parametar prosleđuje vrednost labele a koja kao povratnu vrednost ima vrednost odgovarajućeg ključa:

export class TranslateLabelsPipe implements PipeTransform {

  constructor(private getTranslation: TranslationService){}

  transform(value: string, ...args: any): any {

    return this.getTranslation.getJSONfile(value);

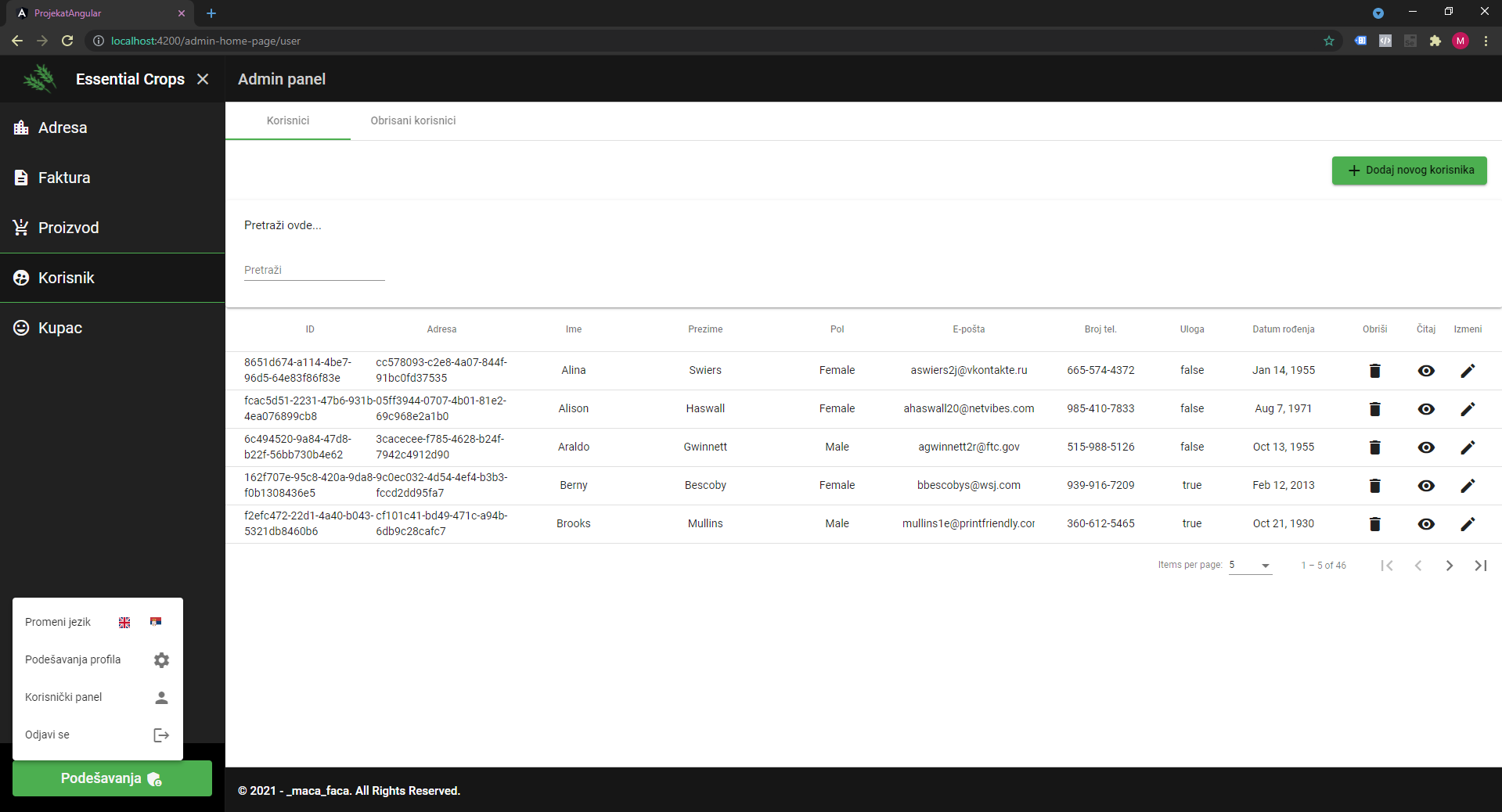
  }

}

Poslednji korak je naznačivanje labela koje želimo da prevedemo u okviru strukture HTML fajlova:

<mat-header-cell \*matHeaderCellDef> {{'Address'  | translate }}</mat-header-cell>

Ključnom rečju translate, poziva se metoda translate kojoj se prosleđuje labela “Address”.



Slika 15. Veb aplikacija na srpskom jeziku

# **Zaključak**

Planiranje i razvoj veb aplikacije za upravljanje fakturama u jednoj ogranizaciji donelo mi je nova iskustva i realan pogled na modernu softversku industriju. Istraživanjem rada organizacije, uvidela sam kako softverske kompanije zapravo istražuju potrebe klijenata, i na koji način kreiraju rešenja.

Koristeći Angular okvir, njegove alate i prednosti, razvila sam uprošćenu verziju aplikacije za praćenje i upravljanje poslovanja jedne manje organizacije – kreiranje i praćenje izdatih faktura, kupaca, proizvoda i isporuka. Veb aplikacija razvijana kao modularna i laka za održavanje. To znači, da ukoliko budem želela da dalje usavršavam aplikaciju i da je obogatim novim mogućnostima, to mogu lako učiniti.

Multiplatformski razvoj, modularnost, responzivnost, kod lak za čitanje i testiranje, puno biblioteka i alata , visok kvalitet i standard izrade, skalabilnost i detaljna dokumentacija samo su neki od brojnih razloga korišćenja Angular-a za razvoj poslovnih veb aplikacija danas.

# **Literatura**

* 1. WILKEN JEREMY, *Angular in Action*, 2018.
  2. ULUCA DOGUHAN, *Angular for Enterprise-Ready Web Applications*, 2020.
  3. SESHADRI SHYAM, Angular Up & Running, Learning Angular, Step by Step, 2018.
  4. Angular Dokumentacija, <https://angular.io/> [pristupljeno 15.08.2021.]
  5. Angular v2 Archive, *Lifecycle hooks,* preuzeto s: [https://v2.angular.io/docs/ts/latest/guide/lifecycle-hooks.html#!#hooks-overview](https://v2.angular.io/docs/ts/latest/guide/lifecycle-hooks.html#!) [pristupljeno 23.08.2021.]
  6. ANASTASIIA SHYBEKO, *8 advantages of angular for businesses and developers,* preuzeto s*:* <https://light-it.net/blog/8-advantages-of-angular-for-businesses-and-developers/> [pristupljeno 24.08.2021.]
  7. Piyush Verma,*Angular 7 – Architecture*, 2020, preuzeto s: <https://www.geeksforgeeks.org/angular-7-architecture/>[pristupljeno 25.08.2021.]
  8. Piyush Verma,*Angular 7 – Angular Data Services using Observable*, 2020, preuzeto s: <https://www.geeksforgeeks.org/angular-7-angular-data-services-using-observable/?ref=rp> [pristupljeno 25.08.2021.]
  9. MANH PHAN, Data binding in Agnular, 2019, preuzeto s: <https://ducmanhphan.github.io/2019-08-03-Data-binding-in-Angular/>[pristupljeno 25.08.2021.]

[10.] Wikipedia, *Angular*, preuzeto s:

<https://en.wikipedia.org/wiki/Angular_(web_framework)>

1. DOM (Document Object Model) [↑](#footnote-ref-1)
2. MDN, *JavaScript*, 2021, https://developer.mozilla.org/en-US/docs/Web/JavaScript [↑](#footnote-ref-2)
3. Jeremy Wilken, *Angular in Action*, (New York, 2018), str. 2 [↑](#footnote-ref-3)
4. Angular CLI [↑](#footnote-ref-4)
5. Wikipedia,  *Relaciona baza podataka,* https://sh.wikipedia.org/wiki/Relacijska\_baza\_podataka [↑](#footnote-ref-5)
6. W3SCHOOLS, *SQL Tutorial, https://www.w3schools.com/sql/default.Asp* [↑](#footnote-ref-6)
7. Angular dokumentacija, *NgModules*, https://angular.io/guide/ngmodules [↑](#footnote-ref-7)
8. Angular dokumentacija, *Angular Components Overview,* *https://angular.io/guide/component-overview* [↑](#footnote-ref-8)
9. Angular dokumentacija, *Angular Components Overview*, https://angular.io/guide/component-overview [↑](#footnote-ref-9)
10. Angular V2 Archive, *Lifecycle Hooks*, https://v2.angular.io/docs/ts/latest/guide/lifecycle-hooks.html#!#hooks-overview [↑](#footnote-ref-10)
11. Angular dokumentacija, *Text Interpolation*, https://angular.io/guide/interpolation [↑](#footnote-ref-11)