Univerzitet u Novom Sadu

Ekonomski fakultet u Subotici

**Angular u razvoju poslovnih aplikacija**

DIPLOMSKI RAD

Student:

Marija Matić PIN028/17

Subotica, 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

Subotica, septembar 2021. godine

|  |  |  |
| --- | --- | --- |
| 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 aplikacija |
| 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 |

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. Web 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 web dizajn: HTML5, SCSS, Typescript (Framework Angular)
* back-end web dizajn: .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 NgModules 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 softvera uz pomoć kog 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 se za pojednostavljenu verziju.

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

Za razvoj „prednjeg“, odnosno klijentskog dela aplikacije koristila sam Angular framework. Sam framework 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.

Uvod – istorija razvoja internet apl itd promeni

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.

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.

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 Eskplorera 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 kakav 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[[2]](#footnote-2) - kompajlera koji nadgleda i optimizuje način izvršavanja kompjuterskog koda.

Danas preko 80% veb sajtova koristi neku od biblioteka JavaScripta[[3]](#footnote-3) ili veb-okvir za podršku dinamičnih veb sajtova, veb aplikacija, veb servisa, razvoju veba uopšte.

Angular

Jedan od najpoznatijih framework-a danas, pored Vue.js-a i React-a jeste Angular. Prva verzija ovog framework-a 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[[4]](#footnote-4)“. Kao što je gore već pomenuto, 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.

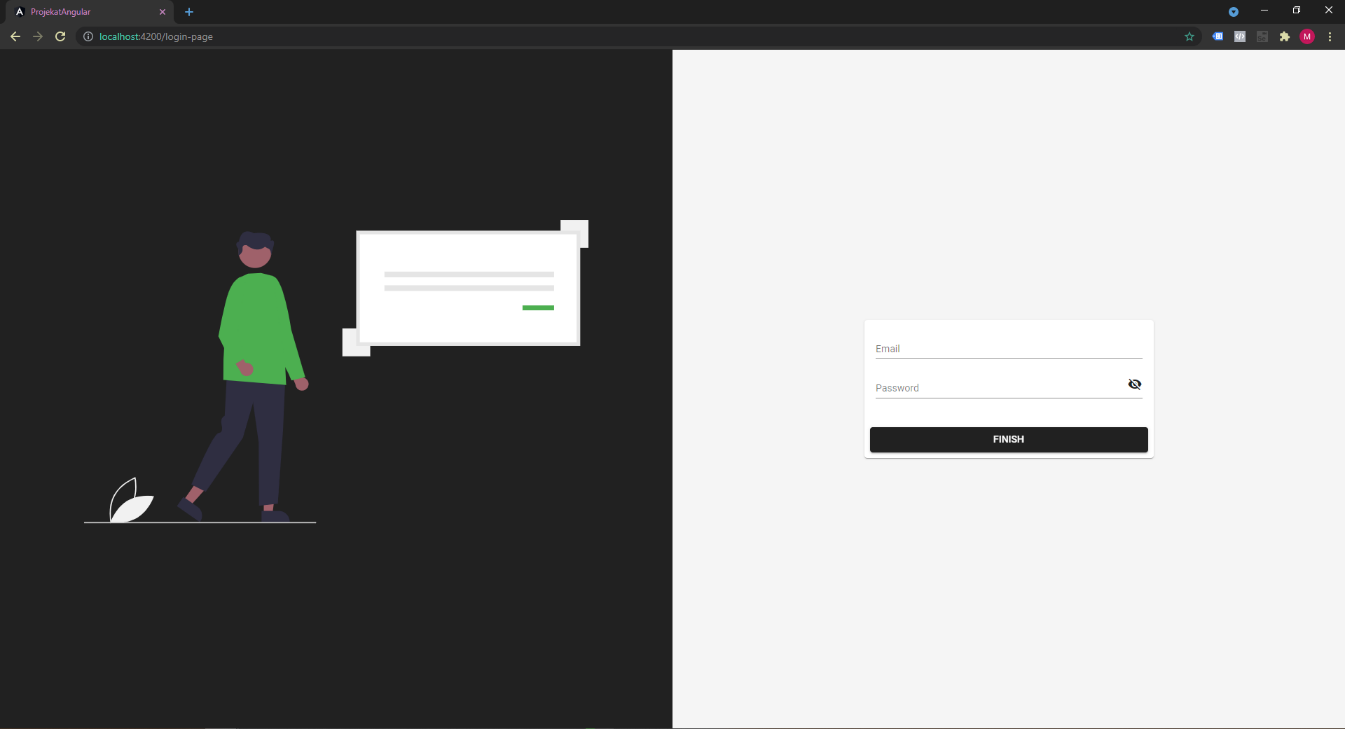
ASP.NET CORE FRAMEWORK

.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, koji predstavlja dobru podlogu za izgradnju skalabilnih RESTful aplikacija baziranih na .NET Framework-u. Kada je reč o

USER LOGIN

Slika #. 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 formata email adresa. 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 tokenKey = 'app\_token';

  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 {

  }

  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

export class LoginService {

  constructor(private http: HttpClient) { }

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

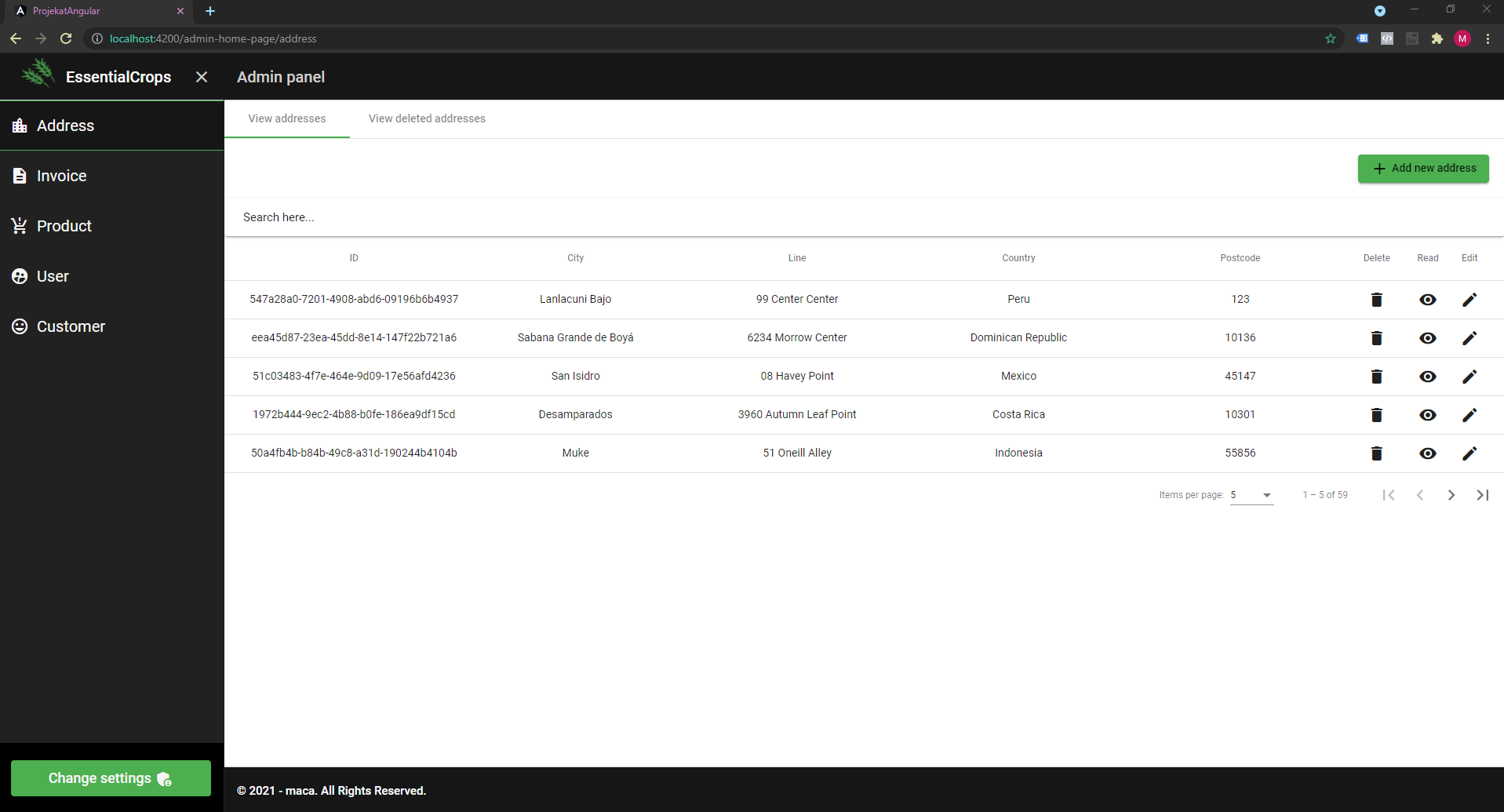
    return this.http.post<User>(`http://localhost:28846/api/User/logIn`, login);

  }

}

Ukoliko je prijava uspešna, korisnik se rutiranjem preusmerava na početnu stranu admin panela.

ADMIN-HOME-PAGE



Slika #. 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.

export class AdminHomePageComponent implements OnInit, AfterViewInit {

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

  @ViewChild('isLoggedInTemplate', { read: TemplateRef }) template: TemplateRef<any>;

  selectedLang = true;

  row = '';

  selectedRow = 'Admin';

  showFiller = false;

  status = false;

  constructor(private router: Router, private resolver: ComponentFactoryResolver,

              private translate: TranslationService) {

              }

  ngOnInit(): void {

  }

  ngAfterViewInit(): void {

    const componentFactory = this.resolver.resolveComponentFactory(AddressComponent);

    this.viewContainer.createComponent(componentFactory);

  }

  openComponent(component): void {

    this.viewContainer.clear();

    let clickedComponent = component + 'Component';

    if (clickedComponent === 'CustomerComponent') {

      const componentFactory = this.resolver.resolveComponentFactory(CustomerComponent);

      this.viewContainer.createComponent(componentFactory);

    } else if (clickedComponent === 'UserComponent') {

      const componentFactory = this.resolver.resolveComponentFactory(UserComponent);

      this.viewContainer.createComponent(componentFactory);

    } else if (clickedComponent === 'ProductComponent') {

      const componentFactory = this.resolver.resolveComponentFactory(ProductComponent);

      this.viewContainer.createComponent(componentFactory);

    } else if (clickedComponent === 'InvoiceComponent') {

      const componentFactory = this.resolver.resolveComponentFactory(InvoiceComponent);

      this.viewContainer.createComponent(componentFactory);

    } else if (clickedComponent === 'UserProfileComponent') {

      const componentFactory = this.resolver.resolveComponentFactory(UserProfileComponent);

      this.viewContainer.createComponent(componentFactory);

    } else if (clickedComponent === 'UserProfileEditComponent') {

      const componentFactory = this.resolver.resolveComponentFactory(UserProfileComponent);

      this.viewContainer.createComponent(componentFactory);

    } else {

      const componentFactory = this.resolver.resolveComponentFactory(AddressComponent);

      this.viewContainer.createComponent(componentFactory);

    }

  }

  logOut(): void {

    localStorage.removeItem('isLoggedIn');

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

  }

  switchLanguage(language: string): void {

    this.translate.switchLanguage(language);

  }

}

Svaki put kada korisnik uđe na admin panel, komponenta koja će se naći učitana u okviru centralnog dela jeste Adresa. Centralni deo admin panela menja se upotrebom ViewContainer. To je kontejner čiji se sadržaj (HTML blok koda) može dinamički menjati. Kada korisnik klikne na bilo koji od ostalih linkova, inicijalna vrednost (HTML kod komponente adresa), briše se iz kontejnera, a metodi openComponent šalje se ime komponente na koju je korisnik kliknuo. Uz pomoć resolveComponentFactory-a, kreira se instanca te prosleđene kompoente i ona se smešta u pomenuti kontejner.

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

  ngAfterViewInit(): void {

    const componentFactory = this.resolver.resolveComponentFactory(AddressComponent);

    this.viewContainer.createComponent(componentFactory);

  }

  openComponent(component): void {

    this.viewContainer.clear();

    let clickedComponent = component + 'Component';

    if (clickedComponent === 'CustomerComponent') {

      const componentFactory = this.resolver.resolveComponentFactory(CustomerComponent);

      this.viewContainer.createComponent(componentFactory);

    } else if (clickedComponent === 'UserComponent') {

      const componentFactory = this.resolver.resolveComponentFactory(UserComponent);

      this.viewContainer.createComponent(componentFactory);

    } else if (clickedComponent === 'ProductComponent') {

      const componentFactory = this.resolver.resolveComponentFactory(ProductComponent);

      this.viewContainer.createComponent(componentFactory);

    } else if (clickedComponent === 'InvoiceComponent') {

      const componentFactory = this.resolver.resolveComponentFactory(InvoiceComponent);

      this.viewContainer.createComponent(componentFactory);

    } else if (clickedComponent === 'UserProfileComponent') {

      const componentFactory = this.resolver.resolveComponentFactory(UserProfileComponent);

      this.viewContainer.createComponent(componentFactory);

    } else if (clickedComponent === 'UserProfileEditComponent') {

      const componentFactory = this.resolver.resolveComponentFactory(UserProfileComponent);

      this.viewContainer.createComponent(componentFactory);

    } else {

      const componentFactory = this.resolver.resolveComponentFactory(AddressComponent);

      this.viewContainer.createComponent(componentFactory);

    }

  }

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 (pogled) 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 direktive 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)},

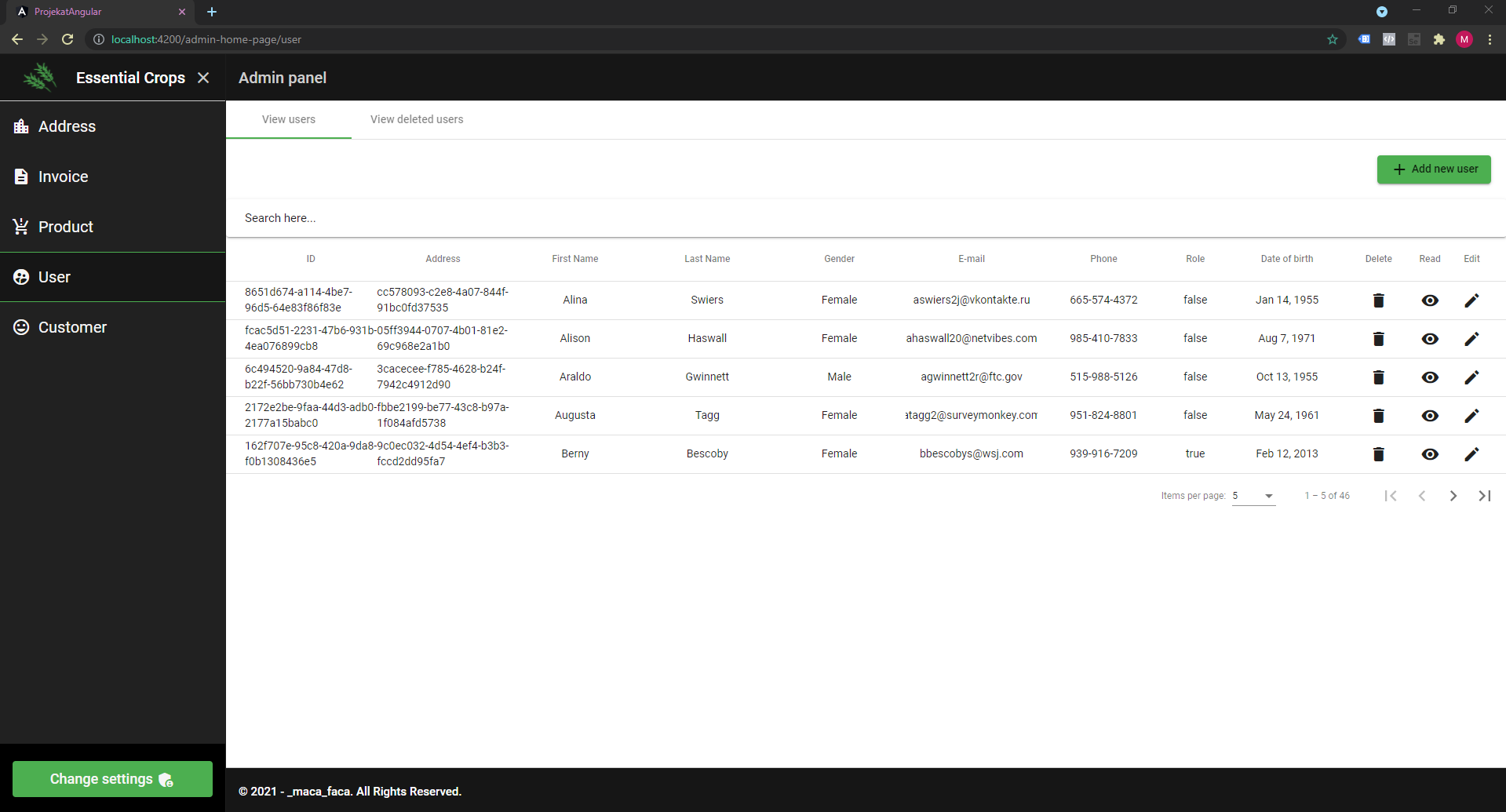
  { path: 'product', component: ProductComponent,

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

  { path: 'invoice', component: InvoiceComponent,

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

KOMPONENTA USER



Pogled 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 softDelete-a, 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 pogled. 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'}});

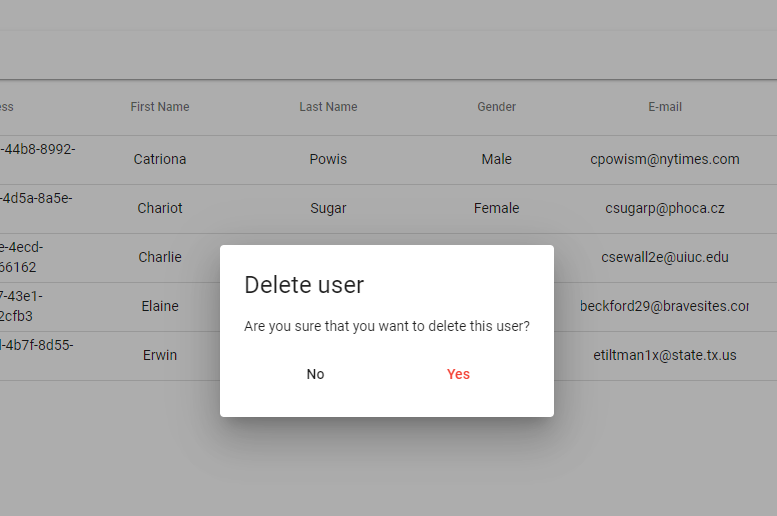
    dialogRef.afterClosed().subscribe(x => {

      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.



  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();

  }

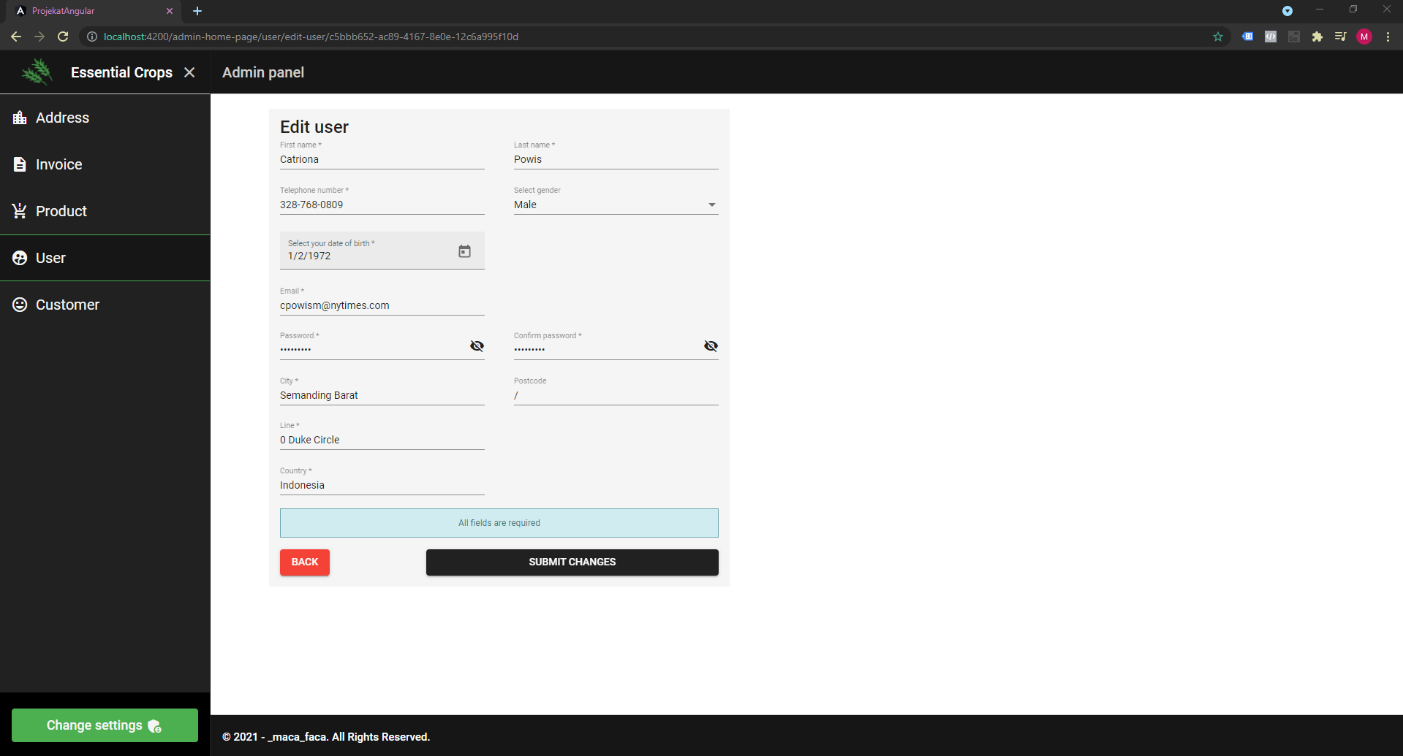
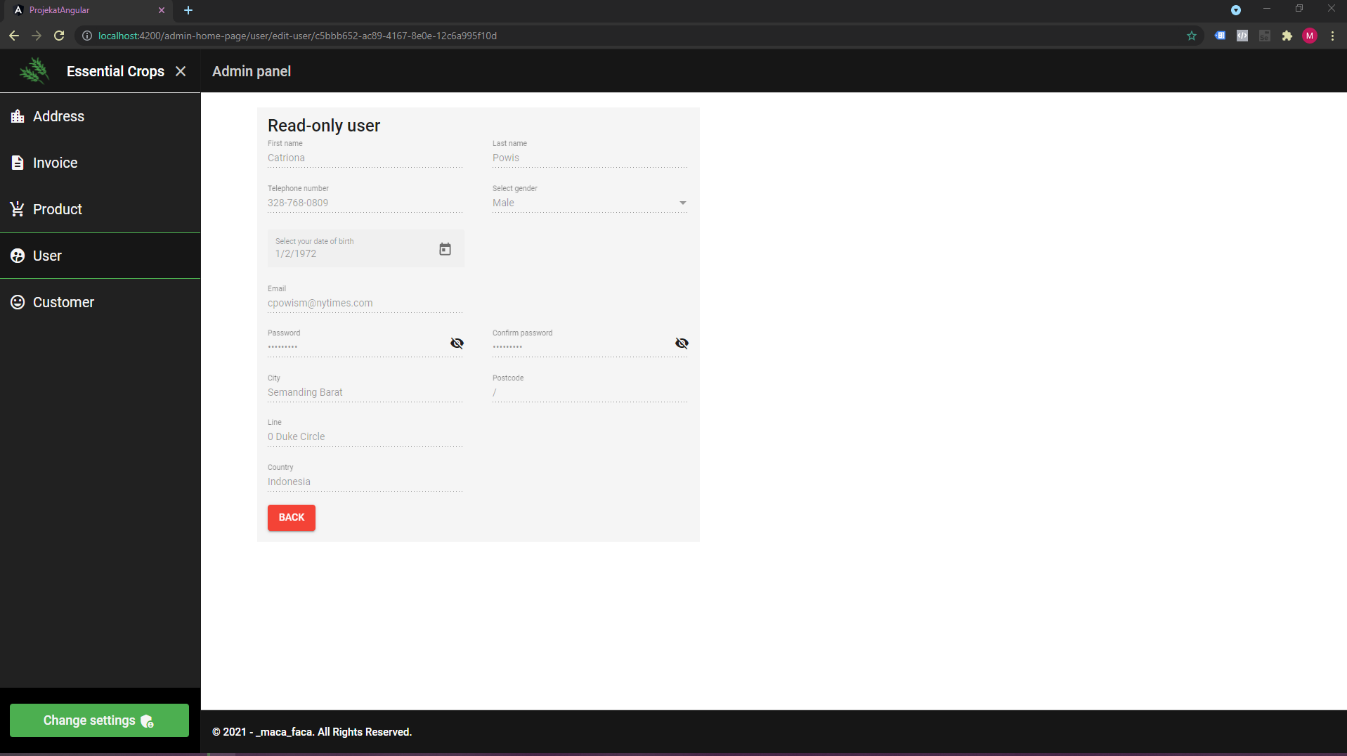
  closeDialog(): void {

    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 modu, u zavisnosti od toga šta korisnik odabere. Ako se otvori u modu 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.



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 odstajanju 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', {

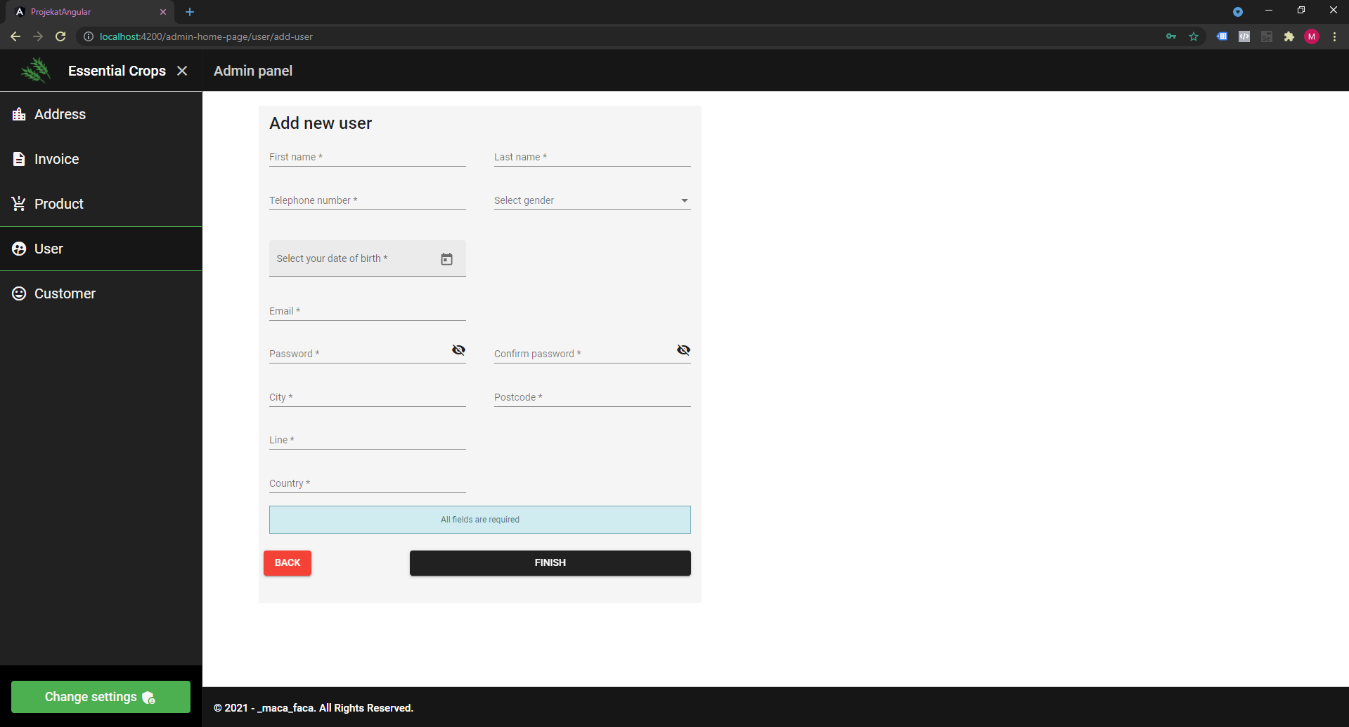
      duration: 2000,

      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 pogled 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 u isto vreme žele da modifikuju neku stavku. Nastanak ovog problema sprečava se korišćenjem zaključavanjem stavki prilikom otvaranja stavke u modu za modifikovanje.

U okviru pogleda 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 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();

}

MODULI I IMPORTI

S obzirom na to da je modul admin panela modul koji će učitavati ostale komponente u aplikaciji, u okviru njega se moraju uvesti ostali zajednički moduli koje će koristiti moduli unutar njega. Na primer, ukoliko želimo da koristimo bilo koju komponentu Angular materijala (**dodaj gore ili bilo gde šta je angular material**), moramo neg de importovati modul te komponente. 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.

1. Document Object Model [↑](#footnote-ref-1)
2. Just-in-time [↑](#footnote-ref-2)
3. https://en.wikipedia.org/wiki/JavaScript [↑](#footnote-ref-3)
4. Single-page application [↑](#footnote-ref-4)