SVEUČILIŠTE U SPLITU
FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Korištenje jedinstvene prijave u web aplikacijama

Petar Perković

Split, rujan 2021.

S V E U Č I L I Š T E U S P L I T U
FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE

Diplomski studij: **Elektronika i računalno inženjerstvo**

Smjer/Usmjerenje: **Računalno inženjerstvi**

Oznaka programa: 222

Ime i prezime: **Petar Perković**

Broj indeksa: 662-2019

ZADATAK DIPLOMSKOG RADA

Radni naslov: Korištenje jedinstvene prijave u web aplikacijama

Zadatak: Izučiti mogućnosti tehnologija za jedinstvenu prijavu (engl. single sign-on) ponuđenih od različitih platformi (npr. Facebook, Google, Twitter). Istražiti dostupne biblioteke otvorenog koda za implementacaiju jedinstvene prijave. Dizajniati realizirati i testirati web aplikaciju temeljenu na .NET Core i Angular tehnologijama kojom će se demonstrirati mogućnosti i ograničenja navedenih tehnologija.

Prijava rada: 05.10.2016. (početak semestra u kojem se prijavljuje rad)

Rok za predaju rada: 05.01.2017. (deset dana prije završetka semestra u kojem je rad prijavljen)

Rad predan:

Predsjednik
Odbora za diplomski rad:

Mentor:

prof. dr. sc. Ime Prezime

izv. prof. dr. sc. Ime Prezime

IZJAVA

Ovom izjavom potvrđujem da sam diplomski rad s naslovom *Korištenje jedinstvene prijave u web aplikacijama* pod mentorstvom prof. dr. sc. Ljiljane Šerić pisao samostalno, primijenivši znanja i vještine stečene tijekom studiranja na Fakultetu elektrotehnike, strojarstva i brodogradnje, kao i metodologiju znanstveno-istraživačkog rada, te uz korištenje literature koja je navedena u radu. Spoznaje, stavove, zaključke, teorije i zakonitosti drugih autora koje sam izravno ili parafrazirajući naveo/la u diplomskom radu citirao/la sam i povezao/la s korištenim bibliografskim jedinicama.

Student/ica

Petar Perković

SADRŽAJ

1 UVOD

U današnje vrijeme, sve većeg i bržeg tehnološkog razvoja, sve veće brzine života, vrlo je bitno što više obaveza moći obaviti brzo i jednostavno, preko interneta. Bez nepotrebnog zvanja, objašnjavanja i propitkivanja. Sve više ljudi, a osobito mladih, želi moći svoje obaveze za koje je nekada trebalo zvati ili posjetiti neku ustanovu, obaviti iz udobnosti svog doma. Stoga je vrlo bitno za ugostitelje ali i ljude iz drugih poslovnih sfera, ponuditi sve bitne informacije na internetu, a ako je moguće i što više usluga.

Prilikom pružanja usluga preko interneta, vrlo je bitno zaštititi korisnikove podatke i privatnost. Isto tako je bitno zaštititi podatke i osjetljive informacije pružatelja usluge. Stoga je autorizacija i autentifikacija jedan od najosjetljivijih dijelova prilikom izrade web aplikacije. Srećom, u današnje vrijeme mnogo programera je prepoznalo tu potrebu pa na tržištu imamo pregršt gotovih rješenja za jedinstvenu prijavu. Takva riješenja se vrlo jednostavno implementiraju u aplikaciju, te su vrlo pouzdana i sigurna. Uz to što su pouzdana i sigurna nude široku paletu usluga i mogućnosti, te se većina rješenja vrlo lako i jednostavno konfigurira. Naravno sve ovisi o odabranom rješenju, ali svima je jasno da rješenja koja nisu jednostavna za uporabu neće biti prihvaćena kod programera. Isto tako sustav za jedinstvenu prijavu treba biti jednostavan i intuitivan krajnjem korisniku prilikom korištenja aplikacije.

Prije početka izrade gotovog proizvoda bitno je proučiti odabrane tehnologije te razvojne alate. A u prvom dijelu su i predstavljene, te objašnjeni neki bitni principi korišteni u tim tehnologijama. Prikazani su razvojni alati i odabran je sustav za jedinstvenu prijavu. Nakon odabira tehnologija i alata sljedi dizajn rješenja. Vrlo je bitno prije početka pisanja koda sve dobro razraditi i osmisliti, a tek kada je sve definirano i osmišljeno krenuti u realizaciju. Dizajn riješenja je prikazan u drugom dijelu ovog rada. Na posljetku sljedi implementacija dizajniranog rješenja, odnosno pisanje koda. Svi bitniji dijelovi implementacije te izgled bitnijih dijelova korisničkog sučelja su prikazani u trećem dijelu.

S dovršetkom izrade posao na aplikaciji nije gotov, tada tek počinje. Ako vlasnik aplikacije želi imati vrhunski proizvod i biti konkurentan na tržištu, tada neprestano treba ažurirati aplikaciju s novim mogućnostima i željama korisnika. Poželjno je pratiti nove trendove i standarde u razvju web aplikacija, te uvijek paziti da ima intuitivno korisničko sučelje koje se lako i jednostavno koristi.

2 PREGLED KORIŠTENIH TEHNOLOGIJA

Za izradu praktičnog dijela diplomskog rada korišten je ASP.NET Core 3.1 API na server strani te Angular 12.1 za korisničku stranu odnosno za korisničko sučelje. Autentifikacija i autorizacija je implementirana sa Auth0 platformom. Za pisanje koda za server stranu korišten je Visual Studio 19, a korisnička strana pisana je u Visual Studio Code. Auth0 je konfiguriran na službenoj stranici (https://auth0.com/).

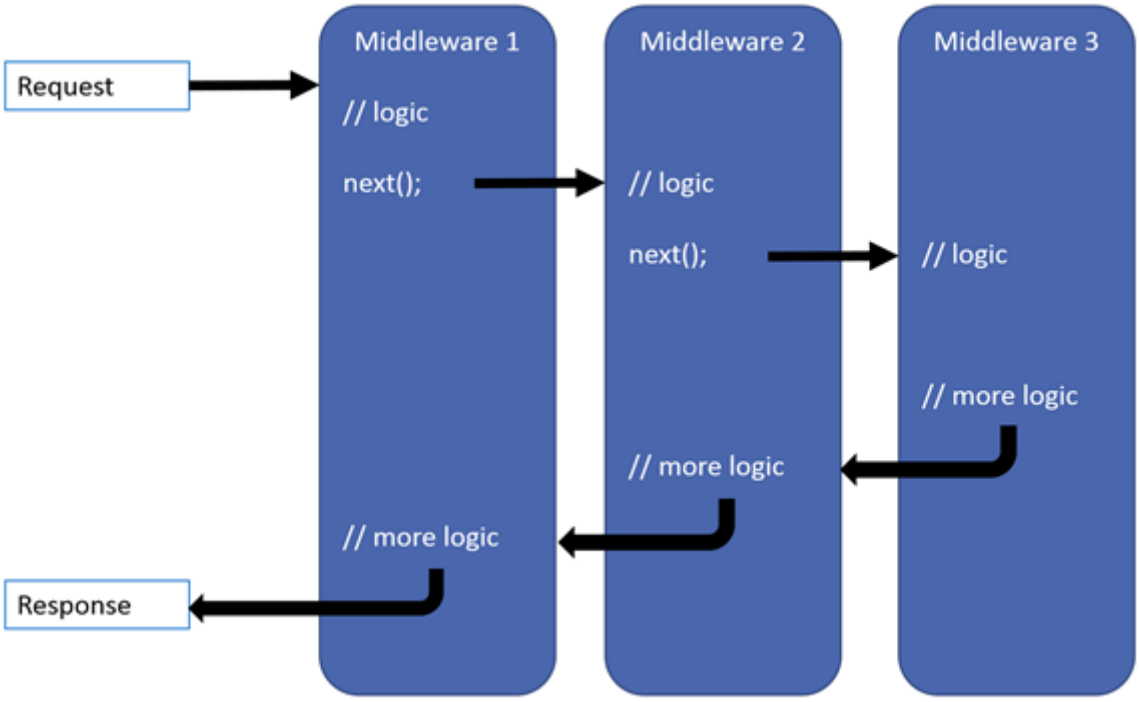
1.1 ASP.NET Core 3.1 API

ASP.NET Core [1] je višeplatformski okvir, visokih performansi, s mogućnošću implementiranja u oblaku, otvorenog koda za izgradnju modernih aplikacija povezanih s internetom. Pomoću ASP.NET Core se može:

- Izraditi web aplikacije i usluge, aplikacije Internet of Things (IoT) i mobilne funkcionalne dijelove (sve ono što nije korisničko sučelje).
- Koristiti razvojne alate u sustavu Windows, macOS i Linux.
- Implementirati u oblak ili lokalno.
- Pokrenuti na .NET Core.

ASP je skraćenica od eng. Active Server Pages što znači aktivne serverske stranice

Svaka ASP.Net Core aplikacija ima fundamentalne dijelove kao što je „Startup“ klasa, „Dependency injection“ servis te „Middleware“. U „Startup“ klasi su konfigurirane usluge koje aplikacija zahtjeva te je definiran cjevovod za upravljanje zahtjevima kao niz „Middleware“ komponenti. ASP.NET Core uključuje ugrađeni okvir za injektirajne ovisnosti koji čini konfigurirane usluge dostupnim u cijeloj aplikaciji. Usluge se obično rješavaju injektiranjem ovisnosti pomoću injektiranja u konstruktor. Injektiranje u konstruktor se implementira deklariranjem parametara konstruktora potrebne vrste ili sučelja. Okvir za injektiranje ovisnosti pruža primjerak usluge toliko dugo ovisno o konfiguraciji usluge u „Startup“ klasi. Cjevovod za obradu zahtjeva sastavljen je kao niz „Middleware“ komponenti (Slika1.1). Svaka komponenta izvodi operacije na „HttpContext“ i/ili poziva sljedeći „Middleware“ u cjevovodu ili završava zahtjev. Konvencijom se „Middleware“ komponente dodaju u cjevovod pozivanjem metode proširenja „Use ...“ u metodi „Startup.Configure“. Na primjer, da bi se omogućilo generiranje statičkih datoteka, pozove se „UseStaticFiles“.



Slika 1.1. Primjer cjevovoda s tri Middleware komponente [2]

request – zahtjev, response – odgovor, logic – logika, next – sljedeće, more - više

ASP.NET Core podržava stvaranje RESTful (eng. Representational State Transfer) usluga, poznatih i kao web API (eng. Application Programming Interface), koristeći C#. A za prijenos podataka koristi se HTTP (eng. Hyper Text Transfer Protocol). HTTP [14] je protokol koji omogućuje dohvaćanje resursa, poput HTML dokumenata. To je temelj svake razmjene podataka na webu i to je klijent-poslužitelj protokol, što znači da zahtjeve pokreće primatelj, obično web preglednik. Cjeloviti dokument rekonstruira se iz različitih poddokumenata koji su dohvaćeni, na primjer tekst, opis izgleda, slike, videozapisi, skripte i drugo. HTTP sadrži pet vrsta zahtjeva:

- GET
- PUSH
- PUT
- PATCH
- DELETE

A sva „moć“ HTTP-a je u zaglavlju [15]. U zaglavlju HTTP zahtjeva ili odgovora se mogu prenjeti razne dodatne informacije, a zaglavlje se s obzirom na sadržaj može podijeliti u četiri skupine:

- Zaglavlja zahtjeva – sadrže više informacija o resursu za dohvaćanje ili o klijentu koji traži resurs.
- Zaglavlja odgovora – sadrže dodatne informacije o odgovoru, poput njegove lokacije ili poslužitelja koji ih pruža.
- Zaglavlja reprezentacije – sadrže informacije o tijelu izvora, poput njegovog MIME tipa ili primjenjenom kodiranju/kompresiji.
- Zaglavlja korisnog tereta – sadrže informacije neovisne o podacima, kao naprimjer duljinu sadržaja i kodiranje koje se koristi za transport.

Za obradu HTTP zahtjeva zaduženi su kontroleri, a kontroleri u web API -ju su klase koje nasljeđuju „BaseController“ klasu. „Namespace Microsoft.AspNetCore.Mvc“ pruža atribute koji se mogu koristiti za konfiguriranje ponašanja web API kontrolora i metoda djelovanja. Na primjer, atribut „Route“ koji definira na kojoj ruti odnosno kako treba izgledati URL za taj kontroler. Također postoji i atribut za definiranje na koji HTTP zahtjev metoda odgovara. Naprimjer „HttpGet“, unutar takvog atributa može se definirati i rutu za taj zahtjev naprimjer „HttpGet('all')“. Za ovakav primjer URL će biti: servera/ruta kontrolera/ruta metode odnosno „all“.

1.2 Angular 12.1

Angular [3] je okvir za razvoj aplikacija i razvojna platforma za stvaranje učinkovitih i sofisticiranih „single-page“ aplikacija. Angular uključuje:

- Okvir temeljen na komponentama za izradu skalabilnih web aplikacija
- Zbirka dobro integriranih knjižnica koje pokrivaju širok spektar značajki, uključujući usmjeravanje, upravljanje obrascima, komunikaciju klijent-poslužitelj i još mnogo toga
- Paket razvojnih alata koji pomažu u razvoju, izgradnji, testiranju i ažuriranju koda

Komponente su gradivni blokovi koji sastavljaju aplikaciju. Komponenta uključuje klasu pisanu u TypeScriptu s „@Component()“ dekoratorom, HTML predloškom i stilovima. Dekorator „@Component()“ navodi sljedeće podatke specifične za Angular:

- CSS selektor koji definira kako se komponenta koristi u HTML predlošku

- HTML element koji odgovara CSS selektoru te postaje instanca komponente, HTML predložak upućuje Angular kako iscrtati komponentu.
- Skup CSS stilova koji definiraju izgled HTML elemenata.

Svaka komponenta ima HTML predložak koji deklarira kako se ta komponenta prikazuje. Ovaj predložak definira se unutar komponente ili putem datoteke. Angular proširuje HTML s dodatnom sintaksom koja omogućuje umetanje dinamičkih vrijednosti iz komponente. Angular automatski ažurira renderirani DOM kad se promijeni stanje komponente. Angular također podržava „property binding“ kako bi pomogao developeru postaviti vrijednosti za svojstva i atribute HTML elemenata i proslijediti vrijednosti u prezentacijsku logiku aplikacije. Također se može dodati „event listener“ da sluša i odgovara na korisničke radnje kao što su pritisci tipki, pokreti miša, klikovi i dodiri. Developeri mogu dodati dodatne funkcionalnosti svojim predlošcima korištenjem direktiva. Najpopularnije direktive u Angular -u su „*ngIf“ i „*ngFor“. Direktive se koriste za izvođenje različitih zadataka, kao što je dinamička izmjena strukture DOM -a. Angular-ovi deklarativni predlošci omogućuju jasno odvajanje logike aplikacije od prezentacije. Predlošci se temelje na standardnom HTML -u radi lakše izgradnje, održavanja i ažuriranja.

Ubrizgavanje ovisnosti omogućuje deklariranje ovisnosti klasa TypeScript-a bez brige o njihovom stvaranju. Umjesto toga, Angular vodi instalaciju. Ovaj uzorak dizajna omogućuje pisanje više provjerljivog i fleksibilnijeg koda.

Angular CLI najbrži je, jednostavan i preporučen način za razvoj Angular aplikacija. Angular CLI izvršava brojne zadatke bez problema. Evo nekoliko primjera:

- ng build - prevodi aplikaciju u izlazni direktorij
- ng serve - sastavlja i služi aplikaciju, obnavljajući promjene datoteka
- ng generation - generira ili mijenja datoteke na temelju sheme
- ng test - pokreće jedinične testove za dati projekt
- ng e2e - izrađuje i opslužuje Angular aplikaciju, a zatim izvodi end-to-end testove

1.2.1 Vezivanje svojstava i interpolacija

Prilikom izrade aplikacija pomoću programa Angular dolazi se u dodir s nekoliko načina za prikazivanje podataka u HTML prikazu. Vezivanje svojstava i interpolacija [5] su vrste vezivanja podataka u Angular -u, koje se koriste za premještanje podataka iz komponente u predložak. Vezivanje podataka vrlo je važan i moćan aspekt razvoja softvera. Uključuje koncept

definiranja komunikacije između komponente i njezinih stavova. Vezanje podataka omogućuje dinamiku i interaktivnost u aplikacijama. U Angular -u postoje četiri vrste povezivanja podataka:

- Vezivanje događaja
- Dvosmjerno vezivanje podataka
- Interpolacija
- Vezivanje svojstava

Vezivanje događaja, povezivanje podataka je kada informacije teku iz HTLM prikaza u komponentu kada se događaj pokrene. Prikaz šalje podatke iz događaja poput klika na gumb koji će se koristiti za ažuriranje komponente. To je upravo suprotno od vezivanja svojstava, gdje podaci idu od komponente do pogleda. Primjer ove vrste povezivanja podataka može se opisati primjerom u nastavku:

```
<p>My name is {{name}}</p>
<button (click)="updateName()">Update button</button>

// component.ts

updateName() {

this.name = 'John'
```

Dvosmjerno povezivanje je mehanizam u kojem podaci teku u oba smjera od komponente do prikaza i natrag. Komponenta i prikaz uvijek su sinkronizirani, a promjene napravljene na oba kraja odmah se ažuriraju u oba smjera. Dvosmjerno vezivanje obično se koristi pri radu s obrascima gdje se korisnički unos koristi za ažuriranje stanja komponente i obrnuto. Angular-ova dvosmjerna sintaksa povezivanja kombinacija je uglatih zagrada i zagrada, [()]. Sintaksa [()] kombinira zagrade vezivanja svojstava, [], sa zagradama vezivanja događaja, (), kako slijedi:

```
<app-sizer [(size)]="fontSizePx"></app-sizer>
```

Interpolacija je mehanizam vezivanja podataka kojim prikazujemo dinamički tekst definiran unutar komponente na HTMl prikazu. U tehnici, tekst koji predstavlja varijable u komponentama postavlja se između dvostrukih uvrnutih zagrada u predlošku. Angular pronalazi varijablu koja odgovara tekstu u komponenti i zamjenjuje tekst vrijednošću dodijeljenom varijabli. Brojevi, datumi itd. Mogu se koristiti izravno između uvrnutih zagrada.

Primjer:

```
<p>{{ name }}</p>

// component.ts name = 'Peter';
```

Vezanje svojstava je jednosmjerni mehanizam koji omogućuje postavljanje svojstva elementa prikaza. Uključuje ažuriranje vrijednosti svojstva u komponenti i njegovo vezanje za element u predlošku prikaza. Vezivanje svojstava koristi sintaksu [] za povezivanje podataka. Primjer je postavljanje onemogućenog stanja gumba:

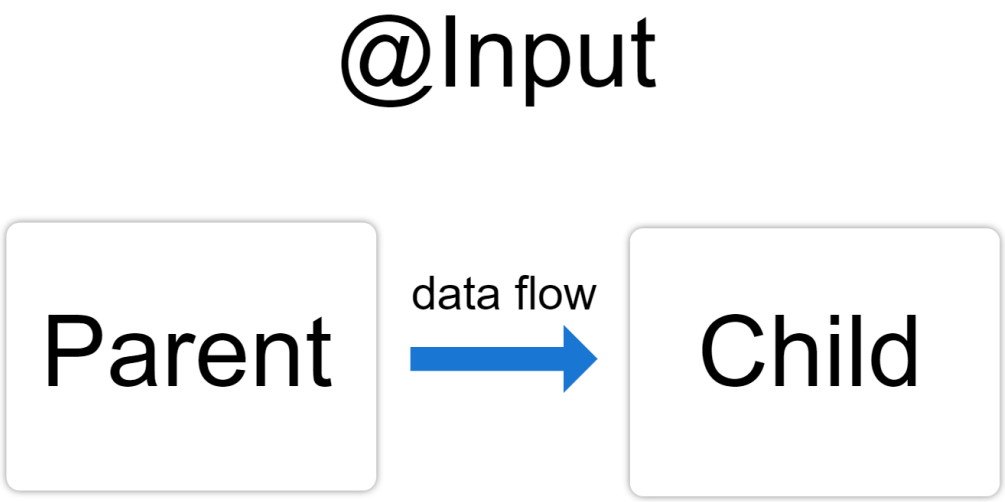
```
<button [disabled]="buttonDisabled"></button>

// component.ts buttonDisabled = true;
```

1.2.2 Dijeljenje podataka između podređenih i roditeljskih direktiva i komponenti

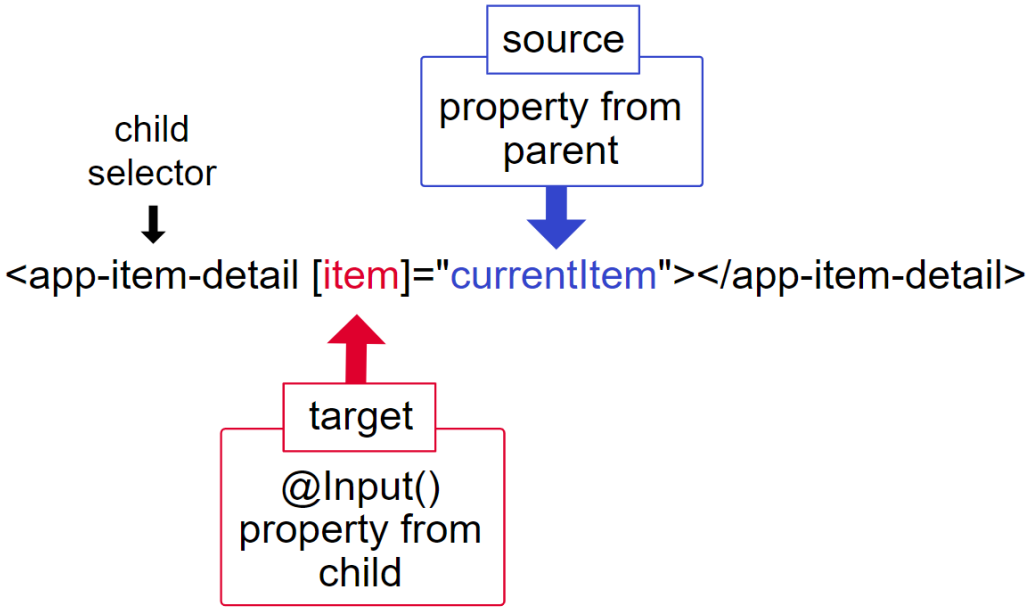
Uobičajeni uzorak u Angular -u je dijeljenje podataka između roditeljske komponente i jedne ili više podređenih komponenti [4]. Za implementaciju ovog uzorka koriste se dekoratori „@Input()“ i „@Output()“. Koji daju podređenoj komponenti način komunikacije s nadređenom komponentom. „@Input()“ dopušta roditeljskoj komponenti ažuriranje podataka u podređenoj komponenti. Nasuprot tome, „@Output()“ dopušta djetetu da šalje podatke roditeljskoj komponenti.

„@Input()“ dekorator u podređenoj komponenti ili direktivi označava da svojstvo može primiti svoju vrijednost od svoje nadređene komponente.



Slika 1.2. Dijagram @Input dekoratora [4]

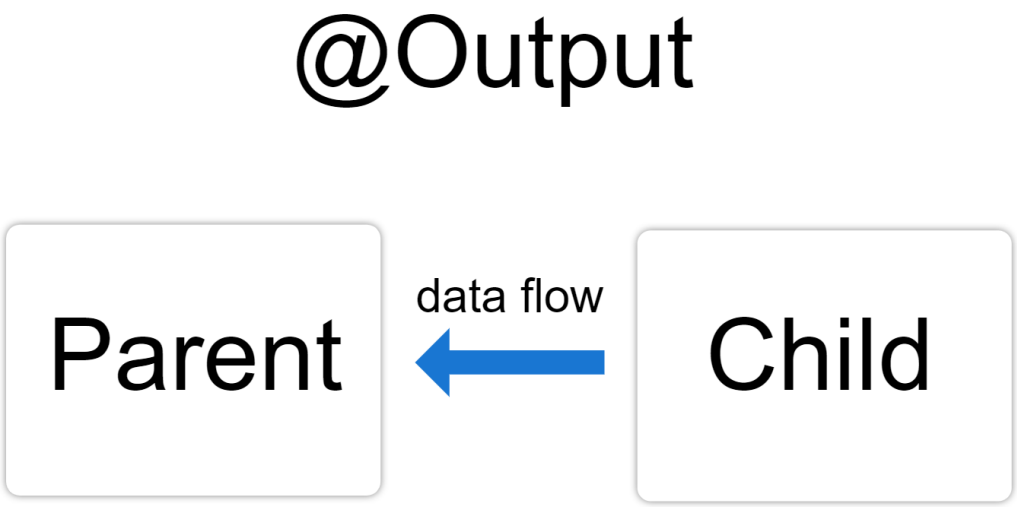
parent – roditelj, *child* – dijete, *data flow* – tok podataka



Slika 1.3. Primjer kako realizirati djeljenje podataka nadređene komponente s podređenom komponentom[4]

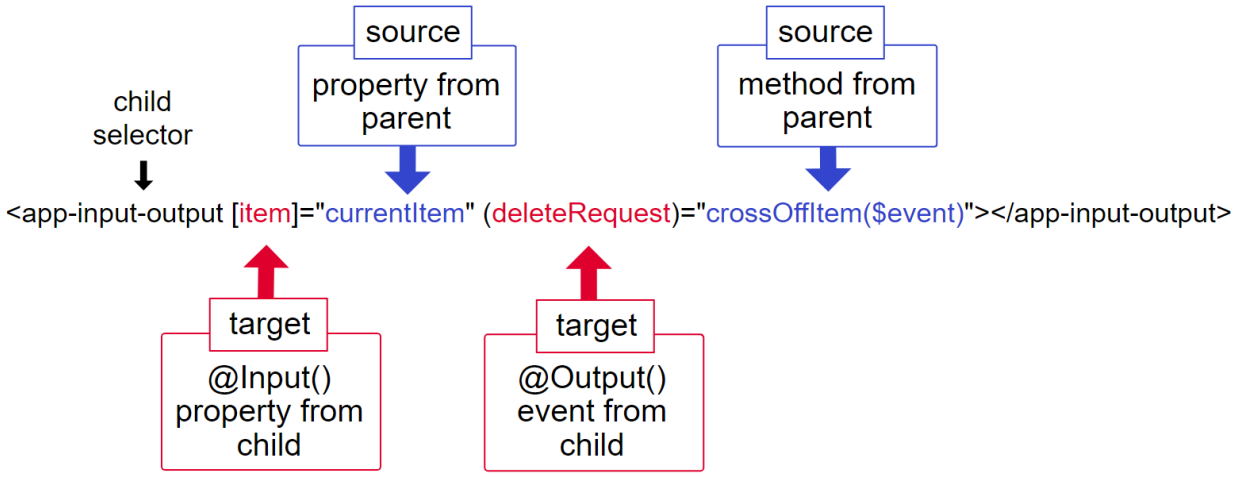
source – izvor, *property from parent* – roditeljsko svojstvo, *target* – cilj, *property from child* – dječije svojstvo, *child selector* – dječiji selektor

„Target“ u uglatim zagradama, je svojstvo koje ima „@Input()“ u podređenoj komponenti. Izvor vezanja, dio desno od znaka jednakosti, su podaci koje nadređena komponenta prosljeđuje ugniježđenoj komponenti.



Slika 1.4. Dijagram @Output dekoratora [4]

data flow – tok podataka, *parent* – roditelj, *child* – dijete



Slika 1.5. Primjer kako implementirati djeljenje podataka među komponentama[4]

child selector – dječiji selektor, *source* – izvor, *property from parent* – roditeljsko svojstvo, *method from parent* – roditeljska metoda, *target* – cilj, *property from child* – dječije svojstvo, *event from child* – dječiji događaj

Podređeni selektor je <app-input-output> sa „item“ i „deleteRequest“ svojstvima @Input () i @Output () u klasi podređene komponente. Svojstvo „currentItem“ i metoda „crossOffItem ()“ su u klasi nadređene komponente.

1.3 Softver za upravljanje identitetom i pristupom korisnika

Softver za upravljanje identitetom i pristupom korisnika (eng. Customer Identity and Access Management - CIAM) [10] široko se koristi i mnogi ljudi traže softverska rješenja koja se lako administriraju i štede vrijeme uz mogućnost društvene prijave te upravljanje pristankom i preferencijama. Drugi važni čimbenici koje treba uzeti u obzir pri odabiru softvera uključuju upravljanje korisnicima i sigurnost. U nastavku (Tablica 1.1) predstavljam deset najpopularnijih softvera. A za implementaciju je odabran Auth0 softver.

Tablica 1.1. Deset softvera za upravljanje identitetom i pristupom korisnika [10]

Softver	Opis
Auth0	Auth0 pruža platformu za provjeru autentičnosti, autorizaciju i siguran pristup aplikacijama, uređajima i korisnicima. Razvojni timovi oslanjaju se na Auth0 -ovu jednostavnost, proširivost i stručnost kako bi platforma bila prikladna za sve.

OneLogin	Moguće proširenje sigurnosti i usklađenosti aplikacije na sve javne i privatne aplikacije u oblaku sa sigurnom jedinstvenom prijavom, višefaktorskom autentifikacijom i opskrbom korisnika.
Okta	Okta je usluga upravljanja identitetom i pristupom na zahtjev za web aplikacije, u oblaku i iza vatrozida.
Ping Indentity	Ping Identity nudi inteligentne mogućnosti identiteta, kao što su jednokratna prijava i višefaktorska autentifikacija za radnu snagu, korisnike i partnere.
CyberArk Identity	Današnja nova stvarnost zahtijeva novu vrstu pristupne platforme. Izgrađen na nultom povjerenju, CyberArk Identity stvara novu eru - siguran pristup posvuda, koja jedinstveno kombinira vodeće tehnologije za besprijekornu integraciju.
WSO2 Identity Server	Poslužitelj identiteta djeluje kao sabirnica identiteta u poduzeću, središnja okosnica za povezivanje i upravljanje više identiteta bez obzira na standarde na kojima se temelje.
Jump Cloud	JumpCloud je potpuna platforma za identitet, pristup i upravljanje uređajima.
LastPass	Poslovna rješenja LastPass pomažu timovima i tvrtkama da preuzmu kontrolu nad upravljanjem svojim identitetom pomoću upravljanja lozinkom, jedinstvene prijave i prilagodljive višefaktorske provjere autentičnosti.
Microsoft Azure Active Directory	Azure Active Directory sveobuhvatno je rješenje u oblaku za upravljanje identitetom i pristupom koje pruža robustan skup mogućnosti za upravljanje korisnicima i grupama i pomaže u sigurnom pristupu aplikacijama, uključujući Microsoftove mrežne usluge poput Office 365 i svijet aplikacija koje ne pripadaju Microsoftu.
SecureAuth Identity Platform	SecureAuth je tvrtka za zaštitu identiteta koja zaposlenicima, partnerima i klijentima omogućuje najsigurnije i fleksibilnije iskustvo autentifikacije. Isporučuje se kao usluga i primjenjuje u oblačnim, hibridnim i lokalnim okruženjima, SecureAuth upravlja i štiti pristup aplikacijama, sustavima i podacima u velikom opsegu, bilo gdje u svijetu.

1.3.1 Auth0

Auth0 [6] fleksibilno je riješenje za dodavanje usluga provjere autentičnosti i autorizacije aplikacijama. Tim ili organizacija mogu izbjeći troškove, vrijeme i rizik koji proizlaze iz izgradnje vlastitog rješenja za provjeru autentičnosti i autorizaciju korisnika. Nekoliko slučajeva upotrebe Auth0:

- Korisnici se trebaju moći prijaviti s korisničkim imenom/lozinkom ili sa svojim društvenim računima (poput Facebooka ili Twittera). Trebaju moći dohvatiti svoj korisnički profil nakon prijave kako bi mogli prilagoditi korisničko sučelje i primijeniti pravila autorizacije.
- Zaštita API-ja
- Zaštita više od jedne aplikacije jedinstvenom prijavom (SSO).
- Siguran pristup API-ju, JavaScript „front-end“ aplikaciji i mobilnoj aplikaciji.
- Autentifikacija korisnika pomoću jezika za označavanje sigurnosnih potvrda (SAML).
- U slučaju neispravne lozinke korisnici se mogu prijaviti pomoću jednokratnih kodova dostavljenih e-poštom ili SMS-om.
- Ako je jedna od adresa e-pošte korisnika ugrožena zbog povrede javnih podataka neke web lokacije, vlasnik i korisnik će biti obaviješten ili će onemogućiti korisniku prijavu u aplikaciju dok ne poništi lozinku.
- Proaktivno blokiranje sumnjive IP adrese ako uzastopno pokušava neuspješnu prijavu, kako bi se izbjegli DDoS napadi.
- Zaposlenicima velikih organizacija omogućuje prijavu u različite interne aplikacije i aplikacije trećih strana koristeći svoje postojeće vjerodajnice za poduzeće.
- Poništavanje lozinke, stvaranje, omogućavanje, blokiranje i brisanje korisnika te korisničko sučelje za upravljanje svim tim.
- Višefaktorska provjera autentičnosti (MFA) kada korisnici žele pristupiti osjetljivim podacima.
- Identitet koji je na vrhu stalno rastućih zahtjeva usklađenosti SOC2, GDPR, PCI DSS, HIPAA i drugih.
- Praćenje korisnika na web lokaciji ili u aplikaciji. Korištenje ovih podatke za stvaranje tokova, mjerenje zadržavanja korisnika i poboljšanje tijeka prijave

Implementacija Auth0-a je podjeljena na dio za SPA i dio za API. Implementacija unutar angulara odnosno za SPA se vrši instaliranjem paketa „auth0-angular“ te korištenjem servisa tog paketa. Servis omogućuje prijavu i odjavu, dohvat korisničkih podataka te još mnogo toga. Sva ostala konfiguracija, dodavanje korisničkih rola, dodavanje korisnika, uređivanje prozora za prijavu i mnogo dugih svari se vrši preko Auth0 stranice vrlo lako i jednostavno. Zaštita API-ja se vrši dodavanjem koda za JSON web tokene (JWT), naime Auth0 koristi JWT za slanje korisničkih podataka na server pa na strani servera nije potrebno ništa osim logike za čitaje podataka iz JWT-ova.

JSON Web Token (JWT) [7] je otvoreni standard (RFC 7519) koji definira kompaktan i samostalan način za siguran prijenos informacija između strana kao JSON objekt. Ti se podaci mogu provjeriti i može im se vjerovati jer su digitalno potpisani. JWT -ovi se mogu potpisati pomoću tajnog ključa (s HMAC algoritmom) ili para javnih/privatnih ključeva koristeći RSA ili ECDSA.

1.4 Visual studio 19

Microsoft Visual Studio [8] je IDE (integrirano razvojno okruženje) koji je napravio Microsoft i koristi se za različite vrste razvoja softvera, poput računalnih programa, web stranica, web aplikacija, web usluga i mobilnih aplikacija. Sadrži alate za dovršavanje, programske prevoditelje i druge značajke koje olakšavaju proces razvoja softvera.

Visual Studio IDE softverski je program za programere koji omogućava pisanje i uređivanje koda. Njegovo korisničko sučelje koristi se za razvoj softvera odnosno za uređivanje, ispravljanje pogrešaka i izradu koda. Visual Studio uključuje uređivač koda koji podržava IntelliSense (komponentu dovršetka koda), kao i preradu koda. Integrirani alat za ispravljanje pogrešaka radi kao ispravljač pogrešaka na izvornoj razini i na računalu. Ostali ugrađeni alati uključuju kodiranje profila, dizajner za izradu GUI aplikacija, web dizajner, dizajner klase i dizajner sheme baze podataka.

1.5 Visual studio code

Visual Studio Code [9] lagani je, ali moćan uređivač izvornog koda koji radi na računalu i dostupan je za Windows, macOS i Linux. Dolazi s ugrađenom podrškom za JavaScript, TypeScript i Node.js te ima bogati sustav proširenja za druge jezike (kao što su C++, C#, Java, Python, PHP, Go) i vrijeme izvođenja (kao što su .NET i Unity).

Visual Studio Code sadrži uređivač izvornog koda, savršen za svakodnevnu upotrebu. Uz podršku za stotine jezika, VS Code pomaže korisniku da bude produktivan. S isticanjem

sintakse, podudaranjem zagrada, automatskim uvlačenjem, odabirom okvira, isječcima i još mnogo toga. Intuitivni prečaci na tipkovnici, jednostavno prilagođavanje i mapiranje prečaca na tipkovnici omogućavaju korisniku da s lakoćom upravlja svojim kodom.

2 DIZAJN RJEŠENJA

Aplikacija je sačinjena od četiri glavna dijela: baza podataka, serverski dio aplikacije, klijentski dio aplikacije te softver za upravljanje identitetom i pristupom korisnika. Prije razrade glavnih djelova bitno je predstaviti zahtjeve (Tablica 2.1.). Za dizajniranje rješenja najprije je potrebno imati zahtjeve koje je potrebno ispuniti tim dizajnom.

Tablica 2.1. Zahtjevi web aplikacije

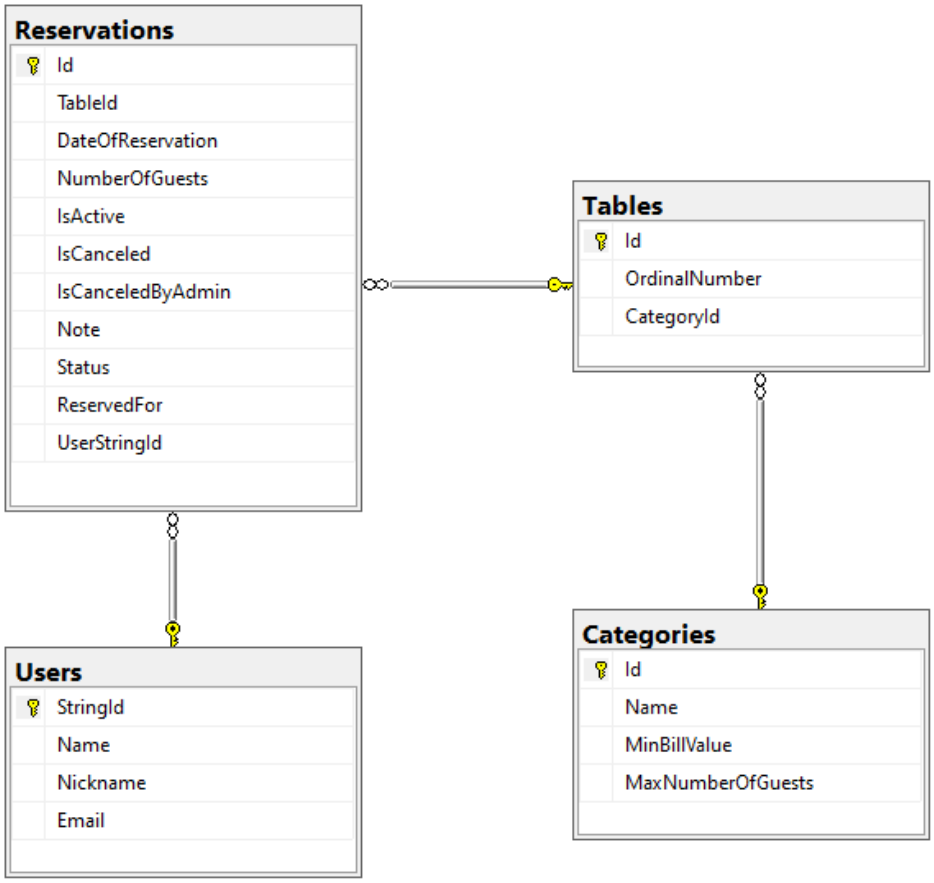
Redni broj zahtjeva	Opis zahtjeva
1.	Trebaju postojati dvije uloge, uloga korisnika, svi korisnici s korisničkim računom i uloga administratora (samo jedan račun koji pripada vlasniku noćnog kluba)
2.	Korisnik treba moći izraditi račun, a za izradu treba upisati puno ime, email adresu te lozinku.
3.	Korisnik treba imati mogućnost prijave u sustav ali za prijavu treba imati kreiran korisnički račun ili se može prijaviti preko neke od platformi (Google ili Facebook).
4.	Navigacijska traka u ljevom kutu treba imati linkove „NightClub“, „Reserve“ i „News“ a u desnom „Login“ link.
5.	Kada se korisnik uspješno prijavi u desnom kutu navigacijske trake se link „Login“ treba pretvoriti u „Logout“ te ljevo od tog linka se treba pojavi link s imenom korisnika koji vodi na stranicu s osobnim podatcima.
6.	Ako se u sustav prijavi administrator onda se umjesto imena treba pojaviti link „Admin center“
7.	Korisnik treba imati mogućnost odjave iz sustava.
8.	Kada se korisnik odjavi iz sustava, aplikacija bi trebala preći na početnu stranicu te treba nestati link s imenom (ili „Admin center“ link) s navigacijske trake te se treba ponovno prikazati „Login“ link.
9.	Svi korisnici bez obzira jesu li prijavljeni ili neprijavljeni u sustav trebaju moći vidjeti početnu stranicu („NightClub“ link).
10.	Početna stranica treba sadržavati „Reseve“ link, tri umanjena članka (naslov, slika i datum događaja) te dio o klubu sa slikom.

11.	Samo prijavljeni korisnik može pristupiti stranici za rezervaciju („Reserve“ link). Ako korisnik koji nije prijavljen pokuša pristupiti stanici za rezervaciju u novom prozoru mu se trebala prikazati stranica za prijavu u sustav.
12.	Svi korisnici bez obzira jesu li prijavljeni ili neprijavljeni u sustav trebaju moći vidjeti stranicu s novostima („News“ link).
13.	Na stranici s novostima se trebaju nalaziti članci koji sadrže naslov, sliku, sadržaj, datum događaja te datum posljednjeg mijenjanja članka.
14.	Na stranici za rezervaciju se treba nalaziti tlocrt noćnog kluba s ucrtanim stolovim, te svaki stol treba imati svoj redni broj.
15.	Korisnik prije odabira stola treba odabrati datum rezervacije da sustav može filtrirati slobodne stolove za taj datum, a dozvoljeni datumi su od toga dana pa unaprijed period koji postavlja administrator. Nakon odabira datuma nedostupni stolovi biti će onemogućeni za klikanje
16.	Korisnik klikom na stol na tlocrtu odabire stol koji će rezervirati, zatim treba odabrati broj gostiju koji će doći sa njim te po želji može napisati napomenu. Klikom na tipku spremi, stol je rezerviran.
17.	Jedan korisnik može rezervirati jedan stol po danu. Iznimno administrator može rezervirati više stolova za jedan datum s tim da treba upisati puno ime onoga kome je rezervacija namjenjena.
18.	Svaki korisnik treba moći vidjeti svoje puno ime, email adresu te rezervacije na stranici s osobnim podatcima.
19.	Rezervacije trebaju imati tri statusa: aktivna, prošla, otkazana. Korisnik treba moći otkazati rezervaciju koja je u statusu aktivna.
20.	Administrator treba moći vidjeti sve aktivne rezervacije po datumima, odabirom datuma u padajućem izborniku trebaju se prikazati rezervacije za odabrani datum.
21.	Administrator treba moći otkazati sve rezervacije za odabrani datum s tim da je obavezan napisati obrazloženje.
22.	Administrator treba moći dodati ili ukloniti članke sa stranice s novostima.
23.	Pri dodavanju novog članka potrebno je unijeti naslov, sadržaj, datum događaja te naslovnu sliku događaja.

24.	Članci na stranici s novostima koji imaju datum događaja u budućnosti trebaju imati tipku rezerviraj, klikom na tu tipku aplikacija treba preći na stranicu za rezerviranje stola i postaviti datum rezervacije na datum tog događaja.
25.	Administrator treba moći mijenjati vrijednost minimalnog račun te vrijednost maksimalnog broja gostiju za svaku kategoriju stola (standardni, zidni te VIP).

2.1 Baza podataka

Za bazu podataka je izabrana MSSQL baza podataka. Za dizajniranje baze podataka je korišten „Code first“ princip. Definirani su modeli te mapiranje, a „Entity Framework-om“ odrađene migracije modela u tablice. „Entity Framework“ je programski okvir koji vrši migracije modela iz C# programskog jezika u tablice SQL baze podataka. Također služi za dohvaćanje podataka iz tablica i još mnogo toga ali to nije tema ovog rada.



Slika 2.1. Dijagram baze podataka za tablice vezane uz rezervacije

Na Slici 2.1. vidimo dijagram tablica vezanih uz rezervacije, a prikazane su četiri tablice te relacije među tim tablicama. Tablica „Reservations“ sadrži „TableId“ odnosno jedna rezervacija ima jedan stol, što znači da jedan stol može biti povezan s više rezervacija (odnos 1:N). Također jedna rezervacija ima jedan „UserStringId“ pa to znači da ima jednog korisnika, odnosno jedan korisnik može biti povezan s više rezervacija (odnos 1:N). Također svaki stol ima „CategoryId“ što znači da svaki stol ima jednu kategoriju, odnosno svaka kategorija može biti povezana sa više stolova (odnos 1:N).

Slika 2.2 Dijagram preostalih tablica u bazi

Na Slici 2.2. je prikazan dijagram preostalih tablica u bazi. Tablica „AdminConfigs“ i tablica „Articles“ nisu relacijski povezane. U bazi još imamo pogled „vwAdminConfigs“ (Slika 2.3.) koji je shematski potpuno isti kao i tablica ali on još dohvaća podatke iz tablice „Categories“ te za svaku kategoriju doda dva retka, jedan za „MinBillValue“ a drugi za „MaxNumberOfGusets“. Također kreirana je i jedna spremljena procedura „stpUpdateAdminConfig“ (Slika 2.4.) koja služi za mijenajane podataka koje prikazuje „vwAdminConfigs“, ta je procedura potrebna jer ovaj pogled ne zadovoljava uvjete da bi se preko njega mogle raditi promjene podataka.

```
CREATE VIEW [dbo].[vwAdminConfigs]
AS
SELECT AC.[Key],
       AC.[Value],
       AC.[TypeName]
FROM dbo.AdminConfigs AC
UNION ALL
SELECT C.[Name] + 'MaxNumberOfGuests_TableCategory' AS [Key],
       CAST( C.[MaxNumberOfGuests] as varchar) AS [Value],
       'int32' AS TypeName
FROM dbo.Categories C
UNION ALL
SELECT C.[Name] + 'MinBillValue_TableCategory' AS [Key],
       CAST( C.[MinBillValue] as varchar) AS [Value],
       'decimal' As TypeName
FROM dbo.Categories C
```

Slika 2.3. View „vwAdminConfigs“

```
CREATE PROCEDURE [dbo].[stpUpdateAdminConfig] @Key varchar(255), @Value varchar(255), @TypeName varchar(255)
AS

DECLARE @TableCategory varchar(255) = '%TableCategory'
DECLARE @VIP varchar(255) = 'VIP%'
DECLARE @Standard varchar(255) = 'Standard%'
DECLARE @MinBillV varchar(255) = '%MinBillValue%'
DECLARE @SQL nvarchar(max) = N'UPDATE '
DECLARE @params nvarchar(400) = '@Key varchar(255), @Value varchar(255), @TypeName varchar(255)'

IF @Key LIKE @TableCategory
BEGIN
    SELECT @SQL = @SQL + 'dbo.Categories SET '
    IF @Key LIKE @MinBillV
        SELECT @SQL = @SQL + 'MinBillValue = @Value '
    ELSE
        SELECT @SQL = @SQL + 'MaxNumberOfGuests = @Value '

    IF @Key LIKE @VIP
        SELECT @SQL = @SQL + 'WHERE Name = ''VIP'''
    ELSE IF @Key LIKE @Standard
        SELECT @SQL = @SQL + 'WHERE Name = ''Standard'''
    ELSE
        SELECT @SQL = @SQL + 'WHERE Name = ''WallTable'''
END

ELSE
        SELECT @SQL = @SQL + 'dbo.AdminConfigs SET [Value] = @Value, TypeName = @TypeName WHERE [Key] = @Key'

EXEC sp_executesql @SQL, @params, @Key, @Value, @TypeName;
```

Slika 2.4. Stored-procedure „stpUpdateAdminConfig“

2.2 Serverska strana aplikacije

Serverska strana aplikacije je dio aplikacije koji se izvršava na udaljenom serveru, te je dizajnirana u tri razine:

- API – (eng. Application Programming Interface) ovaj dio aplikacije služi za posluživanje podataka klijentskom dijelu aplikacije.
- Domain – u ovom dijelu su definirani modeli i sučelja te implementirani servisi.
- Infrastructure – služi za rad sa bazom, definirani su „DbContext“, mapiranje, migracije te implementirani repozitoriji.

2.2.1 API razina

U API razini su definirani kontroleri, Dtos (eng. Data transfer objects), autorizacija i autentifikacija, konfiguracija, „Startup“ klasa te „appsetings.json“ datoteka. „Startup“ klasa je predstavljena u uvodu te je nije potrebno dodatno objašnjavati. „Appsetings.json“ je datoteka u kojoj definiramo neke konstantne podatke, u JSON formatu, koje možemo mijenjati a da pri tom ne moramo ponovno „Build-ati“ aplikaciju. U ovoj datoteci je definiran konekcijski string za bazu te podatci za zaštitu API-a s Auth0.

Kontroleri služe za definiranje krajnjih točaka koje će pozivati klijentska strana aplikacije. Aplikacija ima sedam kontrolera, odnosno jedan apstraktni „MainController“ kojeg nasljeđuju svi ostali redom:

- AdminConfigsController – služi za dohvaćanje i mijenjane podataka iz vwAdminConfigs pogleda, a krajnja točka mu je „/api/adminconfigs“.
- ArticlesController – služi za dodavanje, mijenjanje, dohvaćanje te brisanje artikala, a krajnja točka je „/api/articles“.
- CategoriesController – služi za dohvat svih kategorija artikla, a kranja točka je „/api/categories“.
- ReservationsController – ovo je najsloženiji kontroler u aplikaciji i služi za dohvat svih rezervacija za određeni datum, dohvat svih rezervacija za trenutnog korisnika, dohvat svih rezerviranih datuma za trenutnog korisnika, dohvat svih rezerviranih datuma, dodavanje rezervacije, otkazivanje rezervacije te otkazivanje svih rezervacija za određeni datum. Krajnja točka je „/api/reservations“
- TablesController – služi za dohvat svih stolova, a krajnja točka je „/api/tables“
- UsersController – služi za dodavanje novog korisnika. Krajnja točka je „/api/users“.

Autentifikacija se sastoji od „IsAdminAuthorizationHandler“ klase i „IsAdminAuthorizationRequirement“ klase. Ova autentifikacija služi za provjeru da li prijavljeni korisnik ima „admi“ dopuštenje. U „AuthorizationHandler-u“ je implementirana logika provjere a u „AutorizationRequirement-u“ postavljena je dozvoljena rola.

Konfiguracija se sastoji od četiri različite konfiguracije te su njome konfigurirani servisi i mapiranje modela u Dtos, a to su redom:

- AuthtenticationConfig – konfiguracija servisa za autentifikaciju.
- AuthorizationConfig – konfiguracija servisa za autorizaciju.
- AutomapperConfig – konfiguracija mapiranja modela u objekte za prijenos (Dtos eng. data transfer objects)
- DependencyInjectionConfig – konfiguracija injektiranja ovisnosti

Objekti za prijenos podataka (Dtos eng. data transfer objects) su objekti klase koji sadrže dio svojstava modela ovisno o namjeni objekta za prijenos podataka. Objekt za prijenos može biti namijenjen za dodavanje, uređivanje ili za prikaz rezultata. Objekti za prijenos se koriste da se smanji prijenos podataka, odnosno da se ne prenose nepotrebni podatci koje model sadrži

ali nisu potrebni za tu vrstu zahtjeva. Također se koriste za skrivanje podataka od korisnika. Neke podatke koji su sadržani u modelu korisnik ne smije izravno vidjeti već samo služe za rad aplikacije.

2.2.2 Domenska razina

U ovoj razini imat ćemo sučelja i servise te modele. Koristit ćemo sučelja za rad s injektiranjem ovisnosti, to je slovo D SOLID [13] principa, to je načelo inverzije ovisnosti (poznato i kao IoC eng. inversion of control). A SOLID znači:

- S - Single-responsiblity Principle, klasa bi trebala imati jedan i samo jedan razlog za promjenu, što znači da bi klasa trebala imati samo jedan posao.
- O - Open-closed Principle, objekti ili entiteti trebaju biti otvoreni za proširenje, ali zatvoreni za izmjene.
- L - Liskov Substitution Principle, neka je q(x) svojstvo dokazivo za objekte x tipa T, tada bi q(y) trebalo biti dokazivo za objekte y tipa S gdje je S podtip T.
- I - Interface Segregation Principle, klijenta nikada ne treba prisiljavati da implementira sučelje koje ne koristi, niti ih treba prisiljavati da ovise o metodama koje ne koriste.
- D - Dependency Inversion Principle, entiteti moraju ovisiti o apstrakcijama, a ne o konkretnim klasama, te modul na visokoj razini ne smije ovisiti o modulu niske razine.

Zbog injektiranja ovisnosti ako se nešto promijeni u klasi, to neće utjecati na ostale klase jer to ovisi o apstrakciji. U klase servisa se dodaju poslovna pravila aplikacije. Te klase su između kontrolera i repozitorija. Aplikacijski sloj ne bi trebao imati pristup infrastrukturnom sloju, pa se klase servisa stavljaju u sloju domene koji se koristiti kao posrednik između kontrolera u aplikacijskom sloju i repozitorija u infrastrukturnom sloju. Također, u kontroler se ne dodaju poslovna pravila, pa će kontrolori pozvati servise u koje će biti pravila poslovanja, a servisi će pozvati repozitorije. Modeli služe za migracije, odnosno kreiranje tablica „Code first“ principom, te za sve akcije nad bazom. Podatci se mapiraju iz modela odnosno u modele.

Aplikacija sadrži osam modela od kojih je jedan apstraktni te ga nasljeđuju svi drugi modeli:

- AdminConfig
- AdminCustomConfig
- Article
- Category

- Reservation
- Table
- User

Također sadrži trinaest sučelja od kojih je jedan apstraktni i nasljeđuju ga sva „repository“ sučelja:

- IAdminConfigRepository
- IAdminConfigService
- IArticleRepository
- IArticleService
- ICategoryRepository
- ICategoryService
- IRepository
- IReservationRepository
- IReservationService
- ITableRepository
- ITableService
- IUserRepository
- IuserService

Te sadrži šest servisa koji implementiraju servis sučelja:

- AdminConfigService
- ArticleService
- CategoryService
- ReservationService
- TableService
- UserService

2.2.3 Infrastrukturna razina

Ova razina sadrži „context“, mapiranje, migracije te repozitorije. Migracije sadrže klase s svim generiranim migracijama pomoću kojih je izrađena baza. U „contextu“ se definiraju modeli koji će se koristiti u radu s bazom. Konfiguriraju opće postavke mapiranja te se dodaje mapiranje objekata definirano u datotekama za mapiranje sadržanim u mapi „Mappings“. A ta mapa sadrži sedam datoteka za mapiranje:

- AdminConfigMapping
- AdminCustomConfigMapping
- ArticleMapping
- CategoryMapping
- ReservationMapping
- TableMapping
- UserMapping

Pomoću repozitorija [11] se odvaja logika koja dohvaća podatke i preslikava ih u model entiteta od poslovne logike koja djeluje na model. Poslovna logika trebala bi biti agnostička prema vrsti podataka koja sadrži razina izvora podataka. Na primjer, razina izvora podataka može biti baza podataka, excel popis ili web usluga. Repozitorij posreduje između razine izvora podataka i poslovnih razina aplikacije. On traži izvor podataka, preslikava podatke iz izvora podataka u poslovni model. Repozitorij odvaja poslovnu logiku od interakcija s temeljnim izvorom podataka ili web uslugom. Razdvajanje između podatkovnih i poslovnih razina ima tri prednosti:

- Centralizira logiku podataka ili logiku pristupa web uslugama.
- Pruža zamjensku točku za testove.
- Pruža fleksibilnu arhitekturu koja se može prilagoditi s razvojem cjelokupnog dizajna aplikacije.

Definiran je generički repozitorij „Repository“, a to je apstraktna klasa, što znači da se ta klasa ne može stvoriti, već samo naslijediti. Sve posebne klase repozitorija nasljeđuju od ove glavne klase. U ovoj klasi su implementirane metode iz sučelja „IRepository“. Postoje neke virtualne metode, a razlog za to je dopustiti da se nadjača u drugoj specifičnoj klasi repozitorija ako je to potrebno. Također je implementirana metoda „Dispose“ jer se koristi za oslobađanje memorije u aplikaciji. Repozitoriji koji nasljeđuju generički repozitorij:

- AdminConfigRepository
- ArticleRepository
- CategoryRepository
- ReservationRepository
- TableRepository
- UserRepository

2.3 Klijentska strana aplikacije

Klijentska strana aplikacije je dio aplikacije koji se izvršava u pregledniku. Kao što je već spomenuto korišten je Angular okvir za izradu klijentskog dijela aplikacije. Angular aplikacija se sastoji od:

- modula
- komponenti
- servisa
- modela
- globalnih klasa
- presretača

2.3.1 Modul

Modul [12] je klasa označena dekoratorom „@NgModule“. Modul uzima objekt metapodataka koji opisuje kako sastaviti predložak komponente i kako stvoriti injektor u vrijeme izvođenja. On identificira vlastite komponente modula, direktive i cijevi, čineći neke od njih javnim, putem svojstva „export“, tako da ih vanjske komponente mogu koristiti. U aplikaciji su dva modula:

- app.module – deklariranje komponenti, uključivanje modula te dodavanje servisa
- app-routing.module – definiranje usmjeravanja

2.3.2 Komponenta

Komponenta je klasa označena dekoratorom „@Component“ te je u dekoratoru potrebno definirati selektor, html predložak ili putanju do dokumenta s html predloškom te putanju do dokumenta s css stilom za tu komponentu. Komponenta se dodaje za prikaz na više načina: dodavanjem komponente u usmjeravanju, dodavanjem selektora komponente u glavnu komponentu ili u neku drugu komponentu. Aplikacija sadrži dvanaest komponenti od kojih je „app.component“ glavna komponenta koja dolazi s postavljanjem angular-a:

- article – dodavanje i izmjena članka
- article-list – prikaz liste članaka
- user-profile – prikaz korisničkih podataka ili administratorskog centra
- AuthButton – prikaz linkova za prijavu i odjavu iz sustava
- confirmation-dialog – prikaz prozora za potvrdu ili odbijanje prilikom kritičnih radnji
- datepicker – prikaz prozora za odabir datuma

- footer – prikaz podnožja
- home – prikaz početne stranice
- nav – prikaz navigacijske trake
- reserv-table – prikaz stranice za rezervaciju stola
- tables – prikaz stolova

2.3.3 Servisi

Servisi su klase koje su označene dekoratorom „@Injectable“ što znači da se mogu injektirati u druge klase. Koriste se za definiranje koda koji se koristi u više komponenti pa se ne treba pisati više puta već se samo injektira u konstruktoru. Aplikacija sadrži osam servisa:

- admin-config
- article
- confirmation-dialog
- identity (autorizacija i autentifikacija)
- identity-guard (blokiranje nedozvoljenog usmjeravanja za prijavljenog korisnika)
- reservation
- table
- user

2.3.4 Modeli

Modeli su klase koje su definirane slično kao i modeli na server strani te služe za izradu aplikacije s definiranim tipovima podataka. Koriste se prilikom slanja zahtjeva na server stranu, za odgovore sa server strane te za interpolaciju (dinamičko generiranje podataka u html-u). Definirano je sedam modela:

- AdminConfig
- Article
- Category
- DecodedToken (za dekodiranje JWT-a)
- Reservation
- Table
- User

2.3.5 Globalna klasa

Globalna klasa je klasa koja je dostupna u cijeloj aplikaciji te sadrži podatke i/ili metode koje se koriste na različitim mjestima (u različitim komponentama) unutar aplikacije. Definirane su dvije globalne klase:

- ConfigData – definirane konstante te nizove konstanti za konfiguracijske podatke administratorskog centra
- GlobalApp – sve ostale konstante, podatci i metode potrebne na globalnoj razini aplikacije

2.3.6 Presretač

Presretač je jedinstvena vrsta servisa koju možemo implementirati. Presretači nam omogućuju presretanje dolaznih ili odlaznih HTTP zahtjeva pomoću „HttpClient-a“. Presretanjem HTTP zahtjeva možemo izmijeniti ili promijeniti vrijednost zahtjeva. Definiran je presretač za dodavanje Auth0 JWT-a u zaglavlje HTTP zahtjeva.

3 IMPLEMENTACIJA

Implementacija je provođenje dizajna u stvarnu aplikaciju. U ovom ploglavlju su prikazani svi bitniji dijelovi koda, te je objašnjen princip i način rada. Također su prikazani dijelovi korisničkog sučelja i sučelja za prijavu korisnika. Za razliku od poglavlja dizajn, u kojem je razrađen dizajn serverske i klijentske strane te baze podataka, u implementaciji je prikazan i softver za upravljanje identitetom i pristupom korisnika odnosno Auth0. Auth0 nije prikazan u dizajnu jer je to gotov program kojeg se konfigurira za željenu aplikaciju.

3.1 Baza podataka

Baza podataka nije implementirana pisanjem skripti već pomoću EntityFramework-a (EF) koji izgenerira migracijsku datoteku prema definiranom modelu i konfiguraciji mapiranja. U nastavku je prikazan primjer konfiguracije za „Category“ model:

```
class ArticleMapping : IEntityTypeConfiguration<Article>
    {
        public void Configure(EntityTypeBuilder<Article> builder)
        {
            builder.HasKey(a => a.Id);

            builder.Property(a => a.Title)
                .IsRequired()
                .HasColumnType("varchar(255)");

            builder.Property(a => a.Content)
                .IsRequired()
                .HasColumnType("varchar(max)");

            builder.Property(a => a.PublishingDate)
               .IsRequired();

            builder.Property(a => a.EventDate)
               .IsRequired();

            builder.ToTable("Articles");
        }
    }
```

Iz priloženog koda je poprilično jasno kako postaviti primarni ključ, strani ključ te odnos 1:N među tablicama. Također je vidljivo kako postaviti da li je neki stupac obavezan te kako postaviti tip podataka za odabrani stupac. Prije generiranja migracije potrebno je model koji će se mapirati u tablicu dodati u „DbContex“. To je potrebno da bi EF znao koje modele treba mapirati te da bi se mogao vršit dohvat, spremanje, uređivanje i brisanje podataka iz baze. Način implementiranja „DbContext-a“ za ovu aplikaciju vidljiv je u priloženom kodu:

```
public class NightClubDbContext : DbContext
    {
        public NightClubDbContext(DbContextOptions options) : base(options) { }

        public DbSet<Category> Categories { get; set; }
        public DbSet<Table> Tables { get; set; }
        public DbSet<Article> Articles { get; set; }
        public DbSet<Reservation> Reservations { get; set; }
        public DbSet<User> Users { get; set; }
        public DbSet<AdminConfig> AdminConfigs { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            foreach (var property in modelBuilder.Model.GetEntityTypes()
                .SelectMany(e => e.GetProperties()
                    .Where(p => p.ClrType == typeof(string))))
                property.SetColumnType("varchar(150)");

            modelBuilder.ApplyConfigurationsFromAssembly(typeof(NightClubDbContext
                ).Assembly);

            foreach (var relationship in modelBuilder.Model.GetEntityTypes()
                .SelectMany(e => e.GetForeignKeys())) relationship.DeleteBehavior
                = DeleteBehavior.ClientSetNull;

            base.OnModelCreating(modelBuilder);
        }
    }
```

U metodi „OnNodelCreating“ pozvana je metoda „ApplyConfigurationsFromAssembly“ koja primjeni konfiguraciju mapiranja definiranu u konfiguracijskim klasama. U ovoj metodi moguće je vršiti i razne druge konfiguracije.

Kada je sve potrebno implementirano EF može generirati migracijsku datoteku pomoću koje će se izvršiti ažuriranje baze. Migracijske datoteke su vidljive programeru te je preporučljivo provjeriti da li je generirana datoteka u skladu s željenom konfiguracijom. U nastavku je prikazan kod migracijske datoteke za dodati „Users“ tablicu:

```
public partial class AddUsers : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Users",
                columns: table => new
                {
                    StringId = table.Column<string>(type: "varchar(150)", nullable
                        : false),
                    Name = table.Column<string>(type: "varchar(255)", nullable
                : false),
                    Nickname = table.Column<string>(type: "varchar(255)", nullable
                : true),

                    Email = table.Column<string>(type: "varchar(255)", nullable
                : false)
```

```
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Users", x => x.StringId);
                });
        }

        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropTable(
                name: "Users");
        }
    }
```

Bitno je napomenuti da se generirane migracijske datoteke mogu izbrisati te da se baza u svakom trenutku može ažurirati na željenu migracijsku datoteku. Zbog toga postoje dvije metode „Up“ i „Down“. Prva služi kada se migracijska datoteka primjenjuje na bazu a druga služi kada se to što je tom migracijskom datotekom promjenjeno na bazi želi poništiti.

3.2 Serverska strana aplikacije

Serverska strana aplikacije je prema definiranom dizajnu podjeljena na API, „Domain“ i „Infrastructure“ razine. „Infrastructure“ razina je vezana uz izvor podataka, odnosno uz bazu podataka pa je velika većina ove razine obješnjena u potpoglavlju baza podataka. A implementacija ostatka te razine i druge dvije razine su objašnjene u daljnjem tekstu.

3.2.1 API razina

API razina je zaslužna za primanje HTTP zahtjeva s klijentske strane aplikacije te vrši odgovre na te zahtjeve. Kontroler je klasa u kojoj je definirano na koje zahtjeve i kako API razina odgovara, a primjer kontrolera za administratorsku konfiguraciju je priložen u nastavku:

```
[Route("api/[controller]")]
    public class AdminConfigsController : MainController
    {
        private readonly IAdminConfigService _adminConfigService;
        private readonly IMapper _mapper;

        public AdminConfigsController(IAdminConfigService adminConfigService,
                                      IMapper mapper)
        {
            _adminConfigService = adminConfigService;
            _mapper = mapper;
        }

        [HttpGet]
        public async Task<IActionResult> GetAll()
        {
            var adminConfigs = await _adminConfigService.GetAll();
            return Ok(_mapper.Map<IEnumerable<AdminConfigResultDto>>(adminConfigs)
                 );
        }
```

```
        [HttpGet("{key}")]
        public async Task<IActionResult> GetByKey(string key)
        {
            var adminConfig = await _adminConfigService.GetByKey(key);

            if (adminConfig == null) return NotFound();

            return Ok(_mapper.Map<AdminConfigResultDto>(adminConfig));
        }

        [Authorize(Policy = ADMIN_POLICY)]
        [HttpPut("{key}")]
        public async Task<IActionResult> Update(string key,
                        [FromBody] AdminConfigUpdateDto adminConfigUpdateDto)
        {
            if (key != adminConfigUpdateDto.Key) return BadRequest();

            if (!ModelState.IsValid) return BadRequest();

            var adminConfig = _mapper.Map<AdminConfig>(adminConfigUpdateDto);

            var adminConfigResult = await _adminConfigService.Update(adminConfig);

            if (adminConfigResult == null) return BadRequest();

            return Ok(_mapper.Map<AdminConfigResultDto>(adminConfigResult));
        }
```

Kontroler za administratorsku konfiguraciju odgovara na „HttpGet“ i „HttpPut“ zahtjeve. Atributima su definirane vrste HTTP zahtjeva te ruta do kranje točke. Atributom je također definirano da se metoda pod „HttpPut“ zahtjevom izvršava samo ako je prijavljeni korisnik administrator. U konstruktor klase su injektirani potrebni servsi. „AdminConfig“ servis vezan uz administratorsku konfiguraciju te „AutoMapper“ koji služi za mapiranje modela u objekte za prijenos i obratno.

Objekti za prijenos su implementirani u mapi Dtos te svaki model ima pripadajuće objekte za prijenos. Primjer objekta za dodati novi članak priložen je u nastavku:

```
public class ArticleAddDto
    {
        [Required(ErrorMessage ="The field is required")]
        [StringLength(255, ErrorMessage ="The field max length is 255 characters")
]
        public string Title { get; set; }

        [Required(ErrorMessage = "The field is required")]
        public string Content { get; set; }

        [Required(ErrorMessage = "The field is required")]
        public string PhotoURL { get; set; }

        [Required(ErrorMessage = "The field is required")]
        public DateTime EventDate { get; set; }
    }
```

Objekt za dodati novi članak sadrži samo svojstva koja su potrebna za tu radnju. Također raznim atributima je konfigurirano koja su svojstva obavezna te kakvog oblika trebaju bit. Time je omogućena provjera dolaznog objekta u kontroleru. A ako objekt ne zadovoljava konfiguraciju kontroler odgovori s odgovorom „Bad Request“ odnosno krivi zahtjev. Prije korištenja objekata za prijenos potrebno je konfigurirati „AutoMapper“. Odnosno postaviti koju klasu će mapirati u koji objekt za prijenos i obratno, u nastavku je prikazan primjer za model stola:

```
public class AutomapperConfig : Profile
    {
        public AutomapperConfig()
        {
            CreateMap<Table, TableResultDto>().ReverseMap();
        }
    }
```

Injektiranje u u konstruktor je moguće tek kada je konfiguriran DI (eng. Dependency Injection) to jest kada je postavljeno koja klasa odgovara odabranom sučelju. Postoje tri metode za konfigurirati DI: „AddTransient“, „AddScoped“ te „AddSingleton“ a razlikuju se u vremenu života injektiranog objekta. S „AddTransient“ metodom objekt je uvijek različit, odnosno svaki kontroler ili servis dobije novu instancu. Za „AddScoped“ objekt je isti za isti zahtjev ali je različit za različite zahtjeve. „AddSingleton“ ima vrijeme života objekta isto kao i sama aplikacija te je objekt isti za sve zahtjeve. U nastavku je prikazan kod za konfiguraciju DI:

```
public static class DependencyInjectionConfig
    {

        public static IServiceCollection ResolveDependencies(
                                        this IServiceCollection services)
        {
            services.AddScoped<NightClubDbContext>();

            services.AddScoped<ICategoryRepository, CategoryRepository>();
            services.AddScoped<ITableRepository, TableRepository>();

            services.AddScoped<ICategoryService, CategoryService>();
            services.AddScoped<ITableService, TableService>();

            services.AddScoped<IArticleService, ArticleService>();
            services.AddScoped<IArticleRepository, ArticleRepository>();

            services.AddScoped<IUserService, UserService>();
            services.AddScoped<IUserRepository, UserRepository>();

            services.AddScoped<IAdminConfigService, AdminConfigService>();
            services.AddScoped<IAdminConfigRepository, AdminConfigRepository>();

            services.AddScoped<IReservationService, ReservationService>();
            services.AddScoped<IReservationRepository, ReservationRepository>();
```

```
            services.AddSingleton<IAuthorizationHandler,
                                  IsAdminAuthorizationHandler>();

            return services;
        }
    }
```

3.2.2 Domain razina

U domenskoj razini je implementirana poslovna logika aplikacije. Implementirani su modeli, sučelja te servisi. Kada se kaže da je poslovna logika implementirana u domenskoj razini misli se na servise koji su implementirani u toj razini. Metode servisa se pozivaju u kontrolerima a servisi pozivaju metode repozitorija. Stoga su servsi posrednici između kontrolera i repozitorija. Primjer servisa u nastavku:

```
public class AdminConfigService : IAdminConfigService
    {
        private readonly IAdminConfigRepository _adminConfigRepository;

        public AdminConfigService(IAdminConfigRepository adminConfigRepository)
        {
            _adminConfigRepository = adminConfigRepository;
        }

        public async Task<IEnumerable<AdminConfig>> GetAll()
        {
            return await _adminConfigRepository.GetAll();
        }

        public async Task<AdminConfig> GetByKey(string key)
        {
            return await _adminConfigRepository.GetByKey(key);
        }

        public async Task<AdminConfig> Update(AdminConfig adminConfig)
        {
            if (_adminConfigRepository.SearchAsync(ac => ac.Key ==

                                            adminConfig.Key).Result.Count() < 1)
                return null;

            await _adminConfigRepository.Update(adminConfig);
            return adminConfig;
        }

        public void Dispose()
        {
            _adminConfigRepository?.Dispose();
        }
    }
```

Servisi i repozitoriji implementiraju sučelja definirana u domenskoj razini, a sučelja su potrebna da bi se moglo konfigurirati injektiranje ovisnosti. Primjer sučelja za „AdminConfig“ servis u nastavku:

```
public interface IAdminConfigService : IDisposable
    {
        Task<IEnumerable<AdminConfig>> GetAll();
        Task<AdminConfig> GetByKey(string key);
        Task<AdminConfig> Update(AdminConfig adminConfig);
    }
```

Sučelje nasljeđuje „IDisposable“ sučelje, a servis implementira metodu „Dispose“ koja služi za čišćenje i otpuštanje memorije te poništavanje neupravljanih resursa, poput rukovanja datotekama i bazom podataka. Modeli su potrebni na svim razinama jer se podatci među razinama prenose pomoću instanci modela. Tek kada se prenose van serverske strane aplikacije mapiraju se u objekte za prijenos podataka. Primjer modela prikazan je u nastavku:

```
public class Category : Entity
    {
        public string Name { get; set; }

        public decimal MinBillValue { get; set; }

        public int MaxNumberOfGuests { get; set; }

        /* EF */
        public IEnumerable<Table> Tables { get; set; }
    }
```

Svaki stol ima svojstvo kategorije, pa tako i svaka kategorija ima više stolova. Zbog toga je potrebno klasi „Category“ dodati svojstvo „Tables“ te u klasi „Table“ svojstvo „Category“.

3.2.3 Infrastructure razina

U infrastrukturnoj razini su implementirani repozitoriji te sve vezano uz EF i bazu podataka (već objašnjeno u potpoglavlju baza podataka). Repozitoriji su implementirani tako da imamo jedan generički apstraktni repozitorij kojeg nasljeđuju svi ostali repozitoriji. U generičnom repozitoriju su implementirane osnovne metode, prikazano u nastavku:

```
public abstract class Repository<TEntity> : IRepository<TEntity> where
                                                    TEntity : Entity
    {
        protected readonly NightClubDbContext Db;
        protected readonly DbSet<TEntity> DbSet;

        protected Repository(NightClubDbContext db)
        {
            Db = db;
            DbSet = db.Set<TEntity>();
        }
```

```
        public virtual async Task<List<TEntity>> GetAll()
        {
            return await DbSet.ToListAsync();
        }

        public virtual async Task<TEntity> GetById(int id)
        {
            return await DbSet.FindAsync(id);
        }

        public virtual async Task Add(TEntity entity)
        {
            DbSet.Add(entity);
            await SaveChangesAsync();
        }

        public virtual async Task Update(TEntity entity)
        {
            DbSet.Update(entity);
            await SaveChangesAsync();
        }

        public virtual async Task Remove(TEntity entity)
        {
            DbSet.Remove(entity);
            await SaveChangesAsync();
        }

        public async Task<IEnumerable<TEntity>> SearchAsync(
                                Expression<Func<TEntity, bool>> predicate)
        {
            return await DbSet.AsNoTracking().Where(predicate).ToListAsync();
        }

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

        public void Dispose()
        {
            Db?.Dispose();
        }
    }
```

Neke metode imaju „virtual“ ključnu riječ da bi se mogle nadjačati u ostalim repozitorijima. Svi repozitoriji nasljeđuju ovaj repozitorij te imaju ovdje definirane metode. U određenom repozitorijiu moguće je dodati dodatne metode te pregaziti postojeće kao što je vidljivo u primjeru u nastavku:

```
public class AdminConfigRepository : Repository<AdminConfig>,
                                        IAdminConfigRepository
    {
        public AdminConfigRepository(NightClubDbContext context) : base(context) {
 }
```

```
        public async Task<AdminConfig> GetByKey(string key)
        {
            return await Db.AdminConfigs.AsNoTracking().FirstOrDefaultAsync(ac =>
                                                                    ac.Key == key);
        }

        public override async Task Update(AdminConfig adminConfig)
        {
            var affectedRows = await Db.Database.ExecuteSqlRawAsync(
                                    "EXEC dbo.stpUpdateAdminConfig {0}, {1}, {2}",
                parameters: new[] { adminConfig.Key, adminConfig.Value,
                                            adminConfig.TypeName });

            if (affectedRows == 0) throw new Exception("Problem with
                                            dbo.stpUpdateAdminConfig procedure!");
        }
    }
```

Ključna riječ „override“ znači da smo tom metodom nadjačali baznu metodu.

3.3 Klijentska strana aplikacije

Klijentska strana aplikacije je implenemtirana s Angular 12.1. po zadanom dizajnu. Na klijentskoj strani je implementirano i korisničko sučelje pa će u ovom poglavlju biti prikazan izgled i dizajn korisničkog sučelja. Također je implementirano korisničko sučelje za jedinstvenu prijavu pomoću Auth0 aplikacije te autorizacija i prikaz osobnih podataka prijavljenog korisnika. Dio o presretaču će biti objašnjen i prikazan u dijelu o Auth0 jer je to direktno vezano uz autorizaciju i autentifikaciju.

3.3.1 Modul

Implementirana su dva modula „app.module“ i „app-routing“ koji dolaze s postavljanjem Angular-a. U nastavku je prikazan kod „app“ modula:

```
@NgModule({
  declarations: [
    AppComponent,
    ArticleComponent,
    ArticleListComponent,
    HomeComponent,
    NavComponent,
    ConfirmationDialogComponent,
    NgbdDatepickerPopup,
    FooterComponent,
    AuthButtonComponent,
    UserProfileComponent,
    ReservTableComponent,
    TablesComponent
  ],
```

```
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule,
    FormsModule,
    BrowserAnimationsModule,
    NgbModule,
    ToastrModule.forRoot(),
    AuthModule.forRoot({
      domain: environment.auth.domain,
      clientId: environment.auth.clientId,
      audience: environment.auth.audience
    }),
  ],
  providers: [
    TableService,
    ConfirmationDialogService,
    GlobalApp,
    ReservData,
    ArticleService,
    IdentityGuardService,
    ReservationService,
    IdentityService,
    { provide: HTTP_INTERCEPTORS, useClass: Auth0Interceptor, multi: true }
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

U prvom dijelu „declarations“ su deklarirane sve komponente, koje je potrebno deklarirati da bi se mogle koristiti u aplikaciji. Zatim sljedi dio „imports“ u kojem se deklariraju vanjski moduli koji su potrebni u aplikaciji, obično su to moduli preuzeti s npm (eng. Node Package Manager). Posljednji dio je „providers“ u kojem deklariramo servise iz aplikacije. A servise je potrebno deklarirati da bi se mogli injektirati u željenu komponentu ili drugi servis. „app-routing.module“ služi za definiranje usmjeravanja, odnosno koja komponenta odgovara kojem URL-u (eng. Uniform Resource Locator). Kod modula u nastavku:

```
const routes: Routes = [
  { path: 'home', component: HomeComponent },
  { path: 'article/new', component: ArticleComponent,
                        canActivate: [IdentityGuardService] },
  { path: 'article/edit/:id', component: ArticleComponent,
                        canActivate: [IdentityGuardService] },
  { path: 'news', component: ArticleListComponent },
  { path: 'user', component: UserProfileComponent,
                        canActivate: [IdentityGuardService] },
```

```
  { path: 'reserv-table', component: ReservTableComponent,
                          canActivate: [IdentityGuardService] },
  { path: '**', redirectTo: 'home', pathMatch: 'full' }
];

@NgModule({
  imports: [RouterModule.forRoot(routes, { relativeLinkResolution: 'legacy' })
],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

„canActivate“ svojstvo služi za postavljanje ograničenja pristupa pojedinim komponentama. Neke komponente su dostupne samo administratoru, neke su dostupne svim prijavljenim korisnicima,dok su ostale dostupne svim posjetiteljima stranice.

3.3.2 Komponenta

Prema definiranom dizajnu implementirane su komponente. Komponente su implementirane pomoću angular CLI-a pa svaka komponenta ima tri datoteke: TypeScript datoteku, HTML datoteku te CSS datoteku. HTML datoteka sadrži predložak izgleda komponente na korisničkom sučelju, u CSS datoteci je definiran stil predloška, a TypeScript datoteka sadrži logiku odnosno kod komponente. Kod komponente za „article“ prikazan je u nastavku:

```
Component({
  selector: 'app-article',
  templateUrl: './article.component.html',
  styleUrls: ['./article.component.css']
})
export class ArticleComponent implements OnInit {
  article: Article = {
    id: 0,
    title: '',
    content: '',
    publishingDate: null,
    photoFilePath: '',
    photoURL: "#",
    eventDate: null
  }
  fileName: string = 'Choose image';
  fileBtn: string = this.fileName;
  isImageInputInvalid: boolean = true;
  isImageInputClicked: boolean = false;
  header: string = 'New';
  isDateValid = false;
  isDateClicked = false;
```

```
@ViewChild('fileInput', { static: false }) fileInput: ElementRef;

constructor(private _articleService: ArticleService,
  private _router: Router,
  private _route: ActivatedRoute,
  private _toastr: ToastrService) {

  this._route.params.subscribe(p => {
    this.article.id = +p['id'] || 0;
  });
  if (this.article.id != 0) {
    this.header = 'Edit';
    this.fileBtn = 'Replace image';
    this.fileName = 'Replace image';
    this.isImageInputInvalid = false;
  }
}

ngOnInit(): void {
  if (this.article.id != 0) {
    this._articleService.getArticleById(this.article.id)
      .subscribe(a => {
        this.article = a;
      },
        error => {
          this._toastr.error(GlobalApp.ServerError);
        });
  }
}

uploadPhoto() {
  var files: HTMLInputElement = this.fileInput.nativeElement.files;
  var reader = new FileReader();
  this.fileName = files[0].name;
  this.fileBtn = 'Replace image'
  this.isImageInputInvalid = false;
  reader.readAsDataURL(files[0]);
  reader.onload = (_event) => {
    this.article.photoURL = reader.result;
  }
}

uploadClick() {
  this.isImageInputClicked = true;
}

save() {
  if (this.article.id == 0) {
    this._articleService.addArticle(this.article)
```

```
        .subscribe(() => {
          this._toastr.success('The article has been added.');
          this._router.navigate(['/news/']);
        },
          error => {
            this._toastr.error('Failed to add the article.');
          })
    }
    else {
      this._articleService.updateArticle(this.article.id, this.article)
        .subscribe(() => {
          this._toastr.success('The article has been updated.');
          this._router.navigate(['/news/']);
        },
          error => {
            this._toastr.error('Failed to update the article.');
          })
    }
  }
  setDate(model: NgModel) {
    if (!model.valid) {
      return;
    }
    this.isDateValid = true;
    this.article.eventDate = new Date(model.value.year, model.value.month,
                                      model.value.day);
  }

  dateClick() {
    this.isDateClicked = true;
  }
}
```

Prikazan je TypeScript kod komponente za dodavanje te mijenjanje članka u novostima. U konstruktor su injektirani potrebni servisi kao naprimjer „ArticleServis“. Ova komponenta implementira „OnInit“ sučelje pa ima metodu „ngOnInit“ u koju se dodaje kod koji će se izvršit prilikom dovršenja stvaranja komponente. Komponenta je tako napisana da prema parametrima iz URL-a prepozna da li se dodaje novi ili se ažurira postojeći članak. Dalje sljedi HTML datoteka koja nije prikazana u cijelosti već samo dio (ostatak je vrlo sličan):

```
<div class="container py-2 pb-5 heder">
    <h5 class="text-uppercase font-weight-light text-center
      m-5">{{header}} article</h5>
    <div class="row justify-content-center">
        <div class="col-8">
            <form #f="ngForm">
                <div class="form-group">
```

```
                    <label for="title">Title</label>
                    <input [(ngModel)]="article.title"
                           #title="ngModel"
                           required
                           name="title"
                           type="text"
                           class="form-control"
                           id="title"
                           placeholder="Enter title">
                </div>
                <div class="alert alert-danger"
                        *ngIf="title.touched && !title.valid">
                        Title is required.</div>
                <div class="form-group">
                    <label for="date">Event Date</label>
                    <ngbd-datepicker-popup  id="date"
                                            placeholder="{{article.eventDate ?
                                        (article.eventDate | date:'yyyy-MM-dd')
                                              : 'YYYY-MM-DD'}}"
                                            (dateChanged)="setDate($event)"
                                            (click)="dateClick()">
                    </ngbd-datepicker-popup>
                </div>
                <div class="alert alert-danger"
                        *ngIf="!isDateValid && isDateClicked">
                        Event date is required.</div>
```

U ovoj komponenti je vidljivo vezivanje svojstava i interpolacija. Također je korištena druga komponeta (datepicker - ngbd-datepicker-popup selektor). Komponenta se dodaje unutar druge komponente pomoću selektora komponente. Na posljetku sljedi CSS datoteka koja sadrži standardnu CSS sintaksu pa je prikazan samo dio:

```
#upload-label {
  position: absolute;
  top: 50%;
  left: 1rem;
  transform: translateY(-50%);
}

.image-area {
  border: 2px dashed rgba(255, 255, 255, 0.7);
  padding: 1rem;
  position: relative;
}
.image-area img {
  z-index: 2;
  position: relative;
}
```

```
.uploader {
  background-color: #757f9a;
  background-image: linear-gradient(147deg, #757f9a 0%, #d7dde8 100%);
}
```

3.3.3 Servisi

Implementirani su razni servisi a pretežno služe za pozivanje HTTP zahtjeva. Metode za pozivanje HTTP zahtjeva se deklariraju unutar servisa jer je u velikom dijelu slučajeva poziv istog zahtjeva potreban u više komponenti pa se servisom izbjegne dupliciranje koda. Servis također može sadržavati metode koje ne pozivaju HTTP zahtjev ali sadrže logiku koja se koristi u više komponenti pa ih definiramo u servsu kojeg vrlo lako injektiramo u željenu komponentu. Vrlo slična servisu je i globalna klasa koja pretežito sadrži konstante potrebne unutar aplikacije. Primjer servisa:

```
@Injectable({
  providedIn: 'root'
})
export class UserService {
  private _baseUrl: string = environment.baseUrl + 'api/users/';

  constructor(private _http: HttpClient) { }

  public addUser(user: User) {
    return this._http.post(this._baseUrl, user);
  }

  public getAll(): Observable<User[]> {
    return this._http.get<User[]>(this._baseUrl);
  }

  public getUserById(id: string): Observable<User> {
    return this._http.get<User>(this._baseUrl + id);
  }
}
```

U ovom primjeru je vidljiva primjena modela spomenuta ranije u tekstu. Kada metoda „getAll“ ne bi imala definiran povratni tip, prilikom korištenja tog vraćenog objekta nebi mogli direktno pristupiti svojstvima objekta bez da nam programski prevoditelj prikaže grešku prilikom prevođenja. Stoga je iduće potpoglavlje o modelima.

3.3.4 Modeli

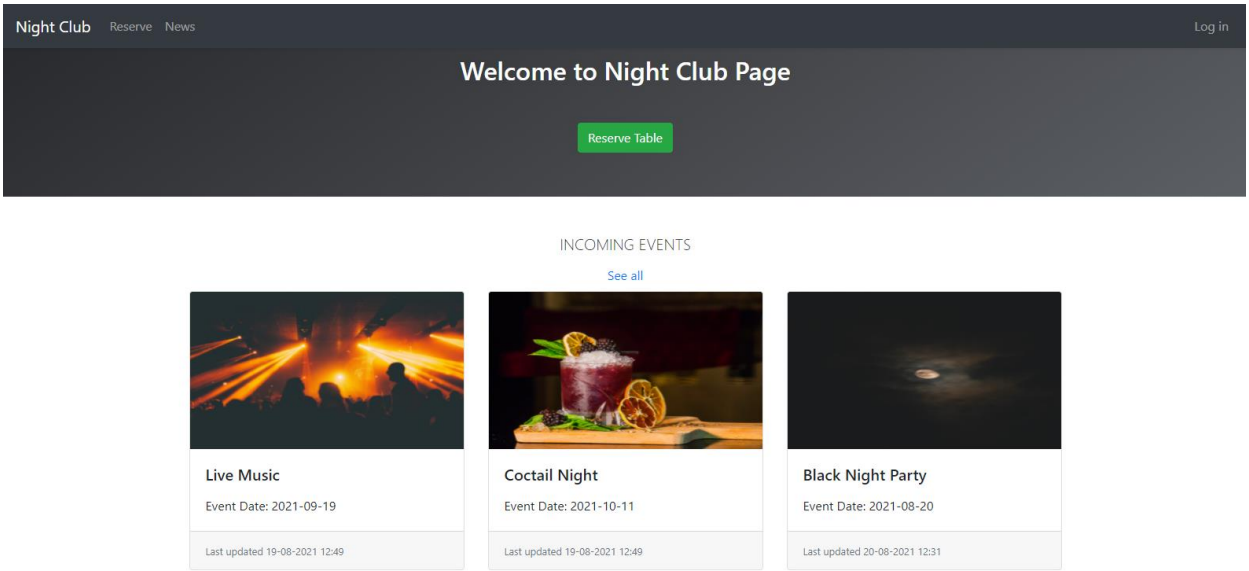
Modeli nam služe da bi mogli imati striktno definirane tipove podataka unutar aplikacija. Pa su definirani svi modeli kao i na serverskoj strani uz neka različita svojstva. Definirani su i još neki dodatni modeli potrebni na klijentskoj strani. Kod u nastavku:

```
export class DecodedToken {
    aud: string[];
    azp: string;
    exp: number;
    iat: number;
    iss: string;
    permissions: string[];
    scope: string;
    sub: string;
}
```

Ovaj model je definiran da bi prilikom dekodiranja JWT-a bilo moguće odmah koristiti svojstvo tog objekta, inače bih programski prevoditelj javljao grešku.

3.3.5 Korisničko sučelje

Korisničko sučelje ili UI (eng. User Interface) je grafički prikaz komponente (HTML predloška uz definirani stil) na ekranu. U nastavku su prikazane snimke zaslona nekih važnijih dijelova aplikacije:

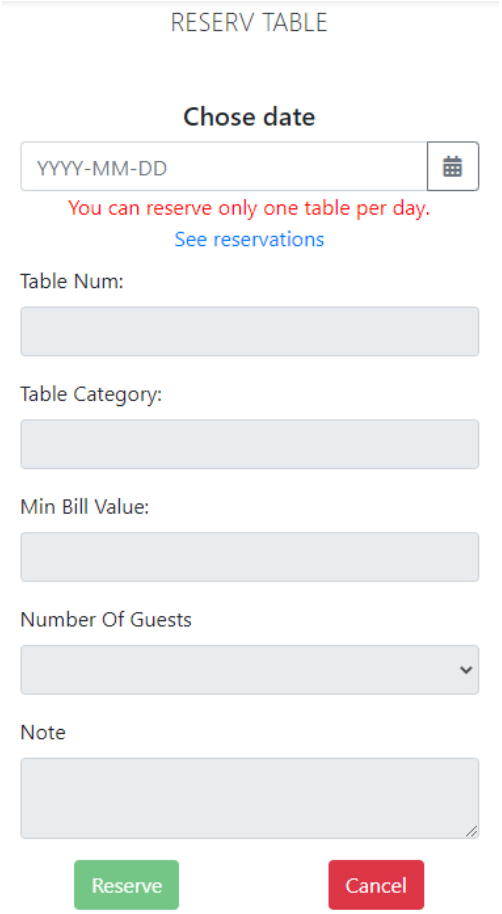


Slika 3.1. Slika navigacijske trake i dio početne stranice

Slika 3.2. Slika navigacijske trake kada je korisnik prijavljen

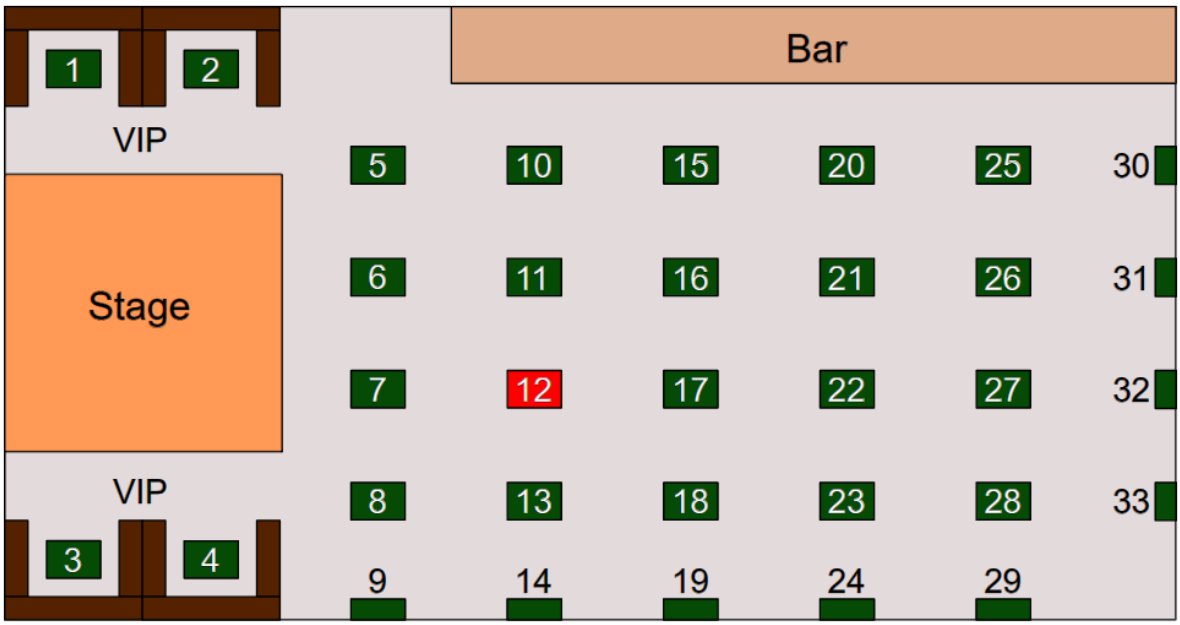
Slika 3.3. Slika navigacijske trake kada je administrator prijavljen

Iz prethodne tri slike (Slika 3.1. / 3.2. / 3.3.) vidimo da su zadovoljeni zahtjevi vezani uz linkove na navigacijskoj traci te na početnoj stranici. Na stranici za rezervacije prvo je potrebno odabrati datum kao što je vidljivo na Slici 3.4.



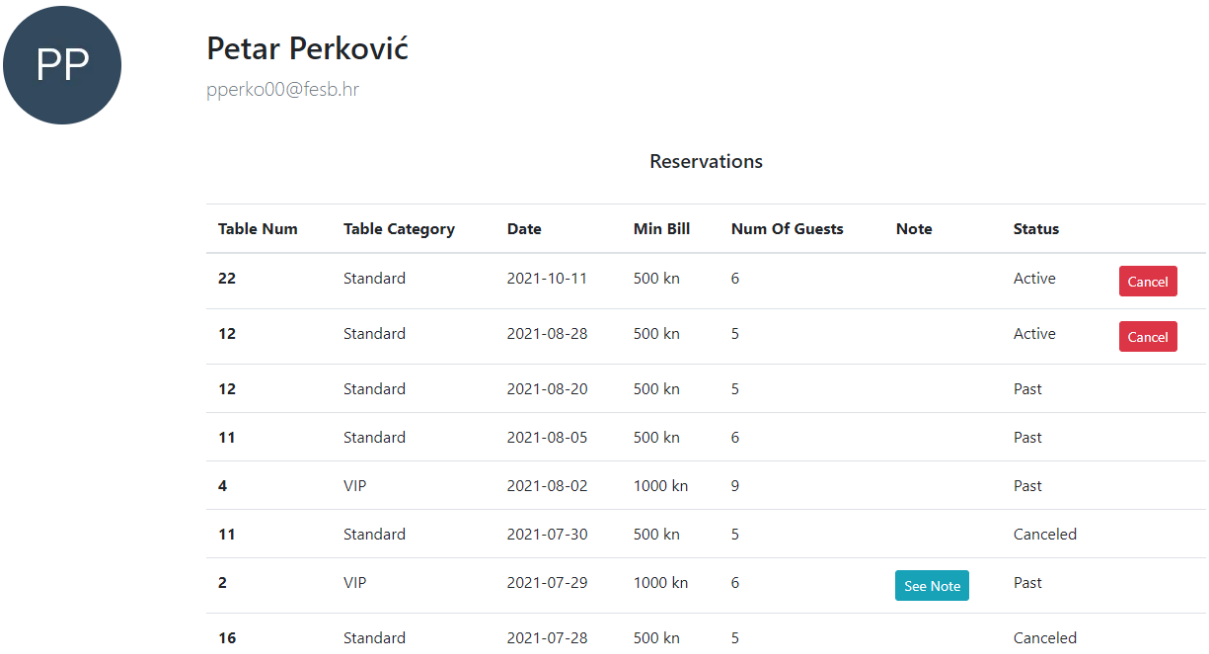
Slika 3.4. Forma za odabir datuma rezervacije i podataka o stolu

Forma za upis podataka o stolu je onemogućena dok se ne odabere stol, a stol je nemoguće odabrati dok se ne odabere datum. Na slici 3.5. je vidljiv grafički prikaz odabira stola.

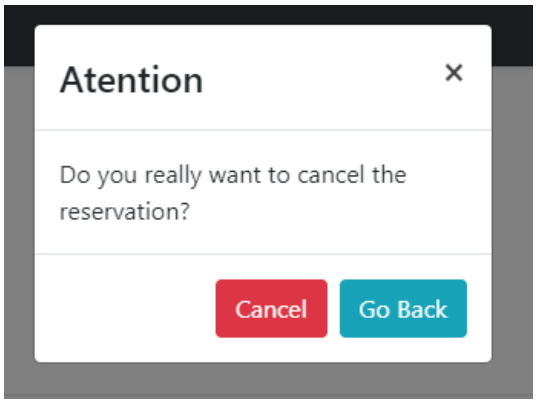


Slika 3.5. Grafički prikaz stolova

Zeleni stolovi su dostupni za rezervaciju a crveni su već rezervirani odnosno zauzeti. Vidljive su tri vrste stolova. VIP stolovi koji imaju sjedeća mjesta, standardni, te zidni kojima je jedna strana naslonjena na zid. Kada se stol rezervira upali se stranica s osobnim podatcima te rezervacijama. Na slici 3.6. vidimo da aktivne rezervacije možemo otkazati pritiskom na tipku „Cancel“ a prije otkazivanja se upali prozor za potvrdu (Slika 3.7.).

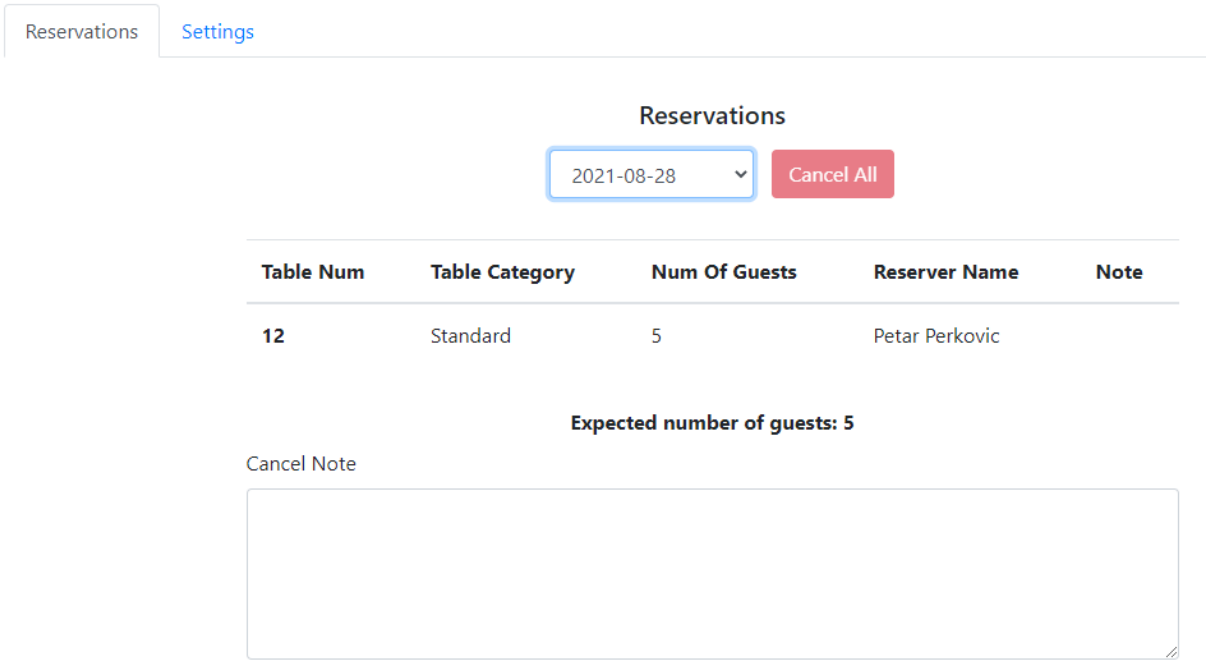


Slika 3.6. Stranica sa osobnim podatcima

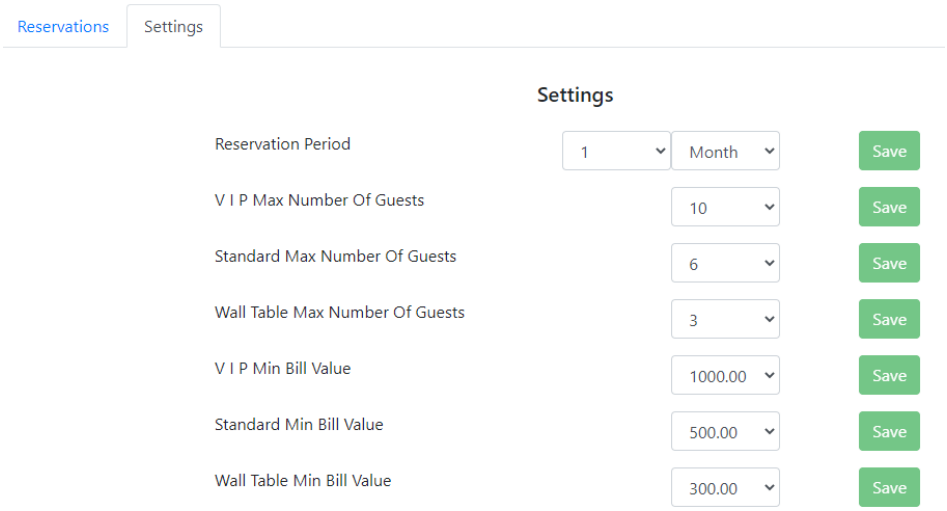


Slika 3.7. Prozor za potvrdu

Administrator na kartici „Reservations“ može otkazati sve rezervacije za odabrani datum ali je obavezan napisati razlog (Slika 3.8.). Također postavlja najveći dopušteni broj gostiju te minimalni iznos računa za svaku kategoriju stola i definira period rezervacije na tabu „Settings“ (Slika 3.9.). Kartice se nalaze na stranici „Admin center“.



Slika 3.8. Kartica s rezervacijama



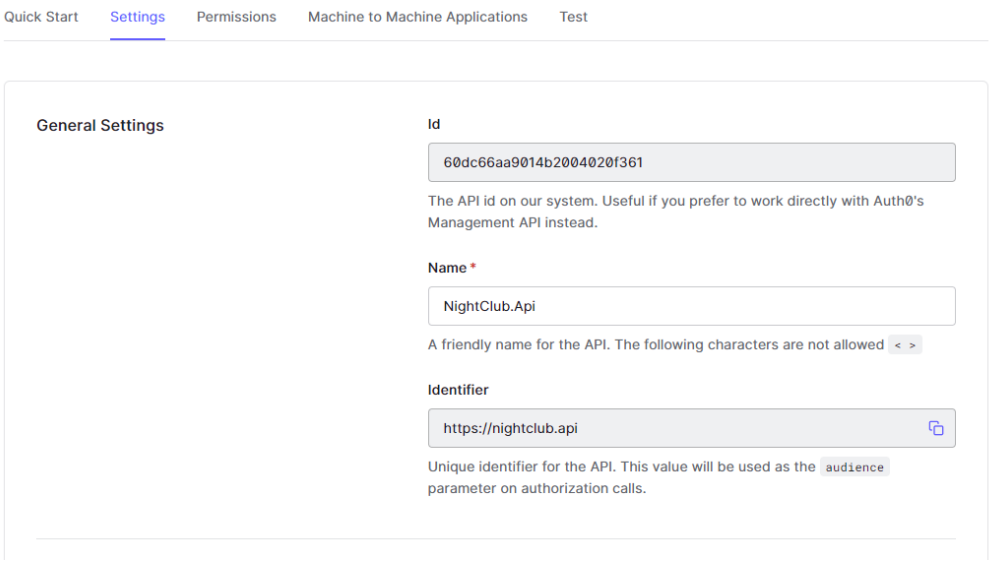
Slika 3.9. Kartica s postavkam

3.4 Implementacija Auth0-a

Za implementirati Auth0 potrebno se prijaviti na službenu stranicu te odraditi konfiguraciju na službenoj stranici. Potrebno je izvršiti konfiguraciju na API razini te uključiti npm modul za Auth0 u Angular aplikaciju.

3.4.1 Konfiguracija na službenoj stranici

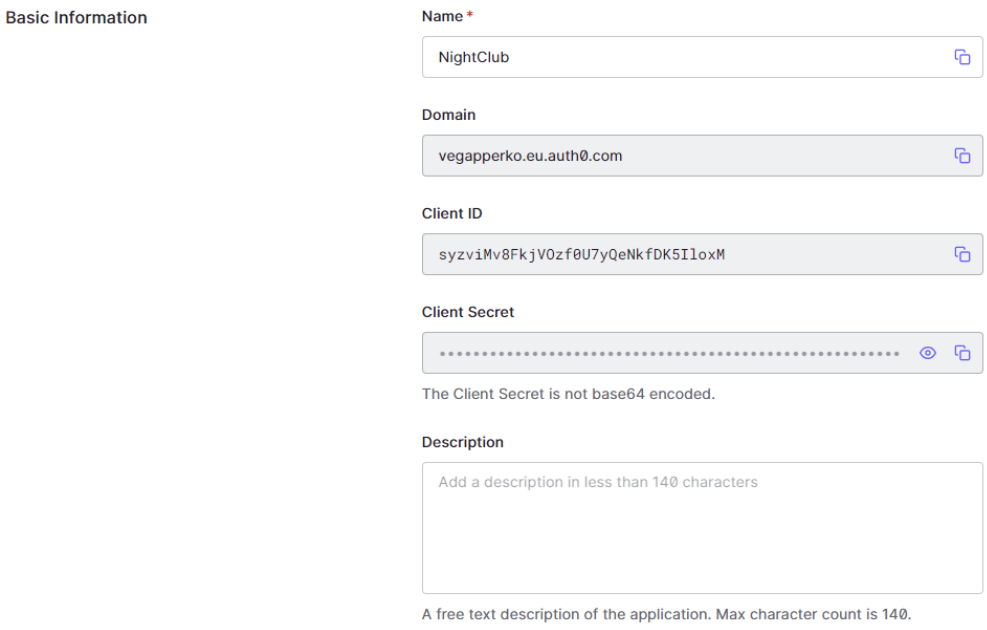
Posebno se konfigurira dio za API razinu a posebno za Angular aplikaciju. Prvo se kreira novi API te se postave opće postavke (Slika 3.10.).



Slika 3.10. Opće postavke za API u Auth0-u

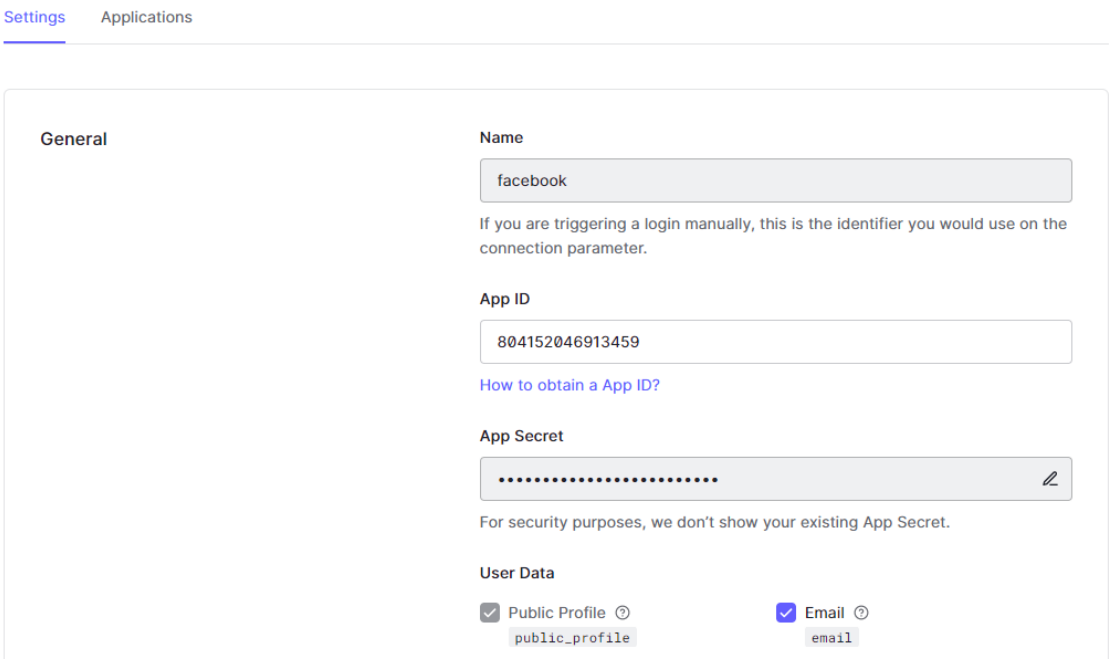
Nakon općih postavki potrebno je još odobriti „Enable RBAC“ postavku te „Add Permissions in the Access Token“ u dijelu „RBAC Settings“. Također je u dijelu „permissions“ potrebno

dodati admin odobrenje. A za korištenje Auth0 unutar Angular-a potrebno je dodati novu aplikaciju te prilikom dodavanja odabrati Angular. Zatim je potrebno postaviti osnovne postavke (Slika 3.11.) te postaviti „Application URIs“ na URI serverskog dijela aplikacije.



Slika 3.11. Osnovne postavke za angular u Auth0-u

Platforme za prijavu se dodaju u dijelu „Authentication“ pod „Socila“. Kreira se konekcija, odabere se željena platforma te se postave opće postavke. U Auth0-u „Google“ platforma je dodana inicijalno a primjer za „Facebook“ je na Slici 3.12.



Slika 3.11. Opće postavke za prijavu preko Facebook-a

3.4.2 Konfiguracija na serverskoj strani

Što se tiče serverske strane dvoljno je dodati logiku za JWT. Naime Auth0 koristi JWT za autentifikaciju API-a. Zbog korištenja korisničkih uloga još je potrebno dodati logiku za autorizaciju. Autorizacija se postavlja dodavanjem servisa za autorizaciju koji čita „permissions“ dio JWT-a. Sve to prikazano je u nastavku:

```
services.AddAuthentication(options =>
            {
                options.DefaultAuthenticateScheme =
                                        JwtBearerDefaults.AuthenticationScheme;
                options.DefaultChallengeScheme =
                                        JwtBearerDefaults.AuthenticationScheme;
            })
              .AddJwtBearer(options =>
              {
                  options.Authority = configuration["Auth0:Authority"];
                  options.Audience = configuration["Auth0:Audience"];
              });


services.AddAuthorization(options =>
            {
                options.AddPolicy(ADMIN_POLICY, policy => policy.Requirements
                                .Add(new IsAdminAuthorizationRequirement()));
            });


public class IsAdminAuthorizationHandler :
                        AuthorizationHandler<IsAdminAuthorizationRequirement>
    {
        protected override Task HandleRequirementAsync(
                                AuthorizationHandlerContext context,
                                IsAdminAuthorizationRequirement requirement)
        {
            var permission = context.User?.Claims?.
                                FirstOrDefault(x => x.Type == "permissions" &&
                                x.Value == requirement.ValidPermission);
            if (permission != null)
                context.Succeed(requirement);

            return Task.CompletedTask;
        }
    }
```

3.4.3 Konfiguracija unutar Angular-a

Implementacija Auth0 unutar angulara započinje instaliranjem „Auth“ modula te uključivanjem modula u importima. „Auth“ servis iz „Auth“ modula se koristi u drugim servisima te u komponentama. Pomoću „Auth“ servisa je implementiran „identity“ servis te

sadrži metode za prijavu i odjavu. Metoda za prijavu se također poziva i unutar „identity-guard“ servisa koji se koristi za zaštitu stranica od nedozvoljenog pristupa, kod prikazan u nastavku:

```
@Injectable()
export class IdentityGuardService implements CanActivate {

  constructor(private _app: GlobalApp,
    private _router: Router,
    private _toastr: ToastrService,
    private _identity: IdentityService,) {}

  canActivate(route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): boolean {
    if (this._app.isAuthenticated() == false)
      {
      this._identity.loginWithAuthentication(state.url);
      return false;
      }
    if ((route.routeConfig.path == "article/new" ||
                              route.routeConfig.path == "article/edit/:id")
                              && this._app.getRole() != GlobalApp.Admin)
      {


      this._toastr.warning('You are not allowed to view this page.
                              You are redirected to news Page');
      this._router.navigate(['/news/']);
      return false;
    }
    return true;
  }
}
```

Ako korisnik nije prijavljen u sustav a želi pristupiti nekoj od zaštićenih stranica, otvori se prozor za prijavu. Ako prijavljeni korisnik nije administrator i pokuša pristupiti starnicama koje mu nisu dozvoljene prikazati će se poruka te će se preusmjeriti na stranicu s novostima.

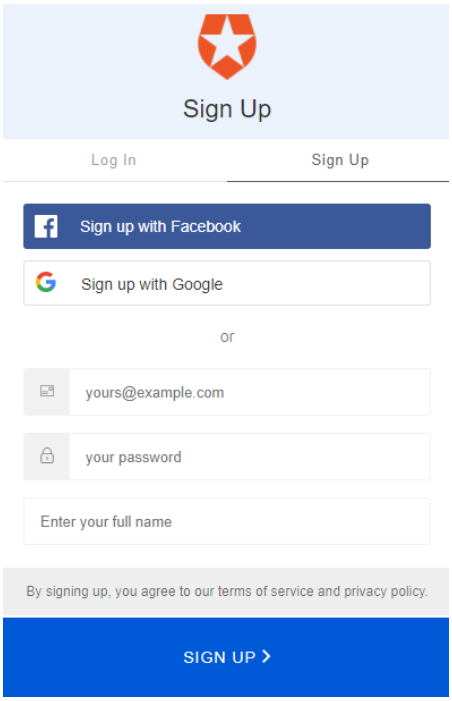
Prilikom slanja zahtjeva na serversku stranu potrebno je poslati i JWT, koji se šalje u zaglavlju zahtjeva. Pošto je JWT potrebno dodati na svaki zahtjev to je implementirano presretačem, posebna vrsta servisa, koji „presretne“ svaki zahtjev te ako je korisnik prijavljen u zaglavlje doda JWT. S tim načinom implementacije je rješeno kopiranje istog koda u sve metode koje šalju zahtjev. Kod je prikazan u nastavku:

```
@Injectable()
export class Auth0Interceptor implements HttpInterceptor {
    constructor(private _auth: AuthService) {}
```

```
    intercept(
        req: HttpRequest<any>,
        next: HttpHandler): Observable<HttpEvent<any>> {
        return this._auth.getAccessTokenSilently().pipe(
            mergeMap(token => {
                const tokenReq = req.clone({
                    setHeaders: { Authorization: `Bearer ${token}` }});
                return next.handle(tokenReq);}),
            catchError((error) => {
                return next.handle(req);
            }));
}}
```

Na svaki zahtjev će se pokušati dodati token, a ako nema tokena, odnosno korisnik nije prijavljen doći će do greške i izvršiti će se dio „catchError“, koji također vraća „next.handle(req)“. „req“ je zahtjev kojem nije dodan token jer token ne postoji kada korisnik nije prijavljen.

Auth0 sadrži inicijalni prozor za prijavu korisnika te za registraciju novog korisnika. Inicijani prozor prilikom registracije zahtjeva samo email adresu i lozinku. Zbog potrebe za imenom korisnika prilikom rezervacije stola na inicijalnom prozoru u tabu za registraciju dodano je i polje za unos imena (Slika 3.12.).



Slika 3.12. Prozor za registraciju novog korisnika

Na slici se vidi da je moguća prijava s „Facebook“ i „Google“ platformom te se vide potrebni podatci za registraciju. Kartica za prijavu je vrlo slična samo ne sadrži polje za unos imena.

4 ZAKLJUČAK

Svakodnevno sve veći broj ljudi koristi internet u razne svrhe. Razvojem računala, pametnih telefona te internetske mreže, danas je gotovo nemoguće zamisliti život bez interneta. Stoga veliki broj poslovnih ljudi, firmi pa čak i malih obrtnika traži svoje buduće klijente na internetu. Nude im razne usluge i proizvode. Stoga ovaj diplomski rad obuhvaća vrlo zanimljivu temu. Mogućnost vrlo jednostavne rezervacije, uz mogućnost odabira pozicije, iz udobnosti doma. Smatram da to znatno povećava broj rezervacija osobito kod mlađe populacije, koja je naviknuta na „čari“ interneta.

Prilikom izrade aplikacije vrlo je bitna i sigurnost podataka i osobnih informacija korisnika ali i vlasnika. Stoga jer u ovom radu korišten jedan od najpopularnijih softvera Auth0. Auth0 pruža razne mogućnosti i modifikacije, te je jako pouzdan i jednostavan za implementiranje u željenu aplikaciju. Također za što veći broj korisnika vrlo je bitno omogućiti prijavu preko družtvenih medija. Mnogi ljudi ne voli izrađivati korisnički račun, a pomoću ove mogućnsti mogu koristit račun neke od društvenih mreža. Pa tako jednim klikom imaju pristup željenoj aplikaciji bez potrebe za kreiranjem korisničkog računa.

Aplikacija je izrađena u nekim od najpopularnijih tehnologija. Na server strani je korišten ASP.NET Core a na klijentskoj strani Angular, a baza je implemntirana u MSSQL-u. S ovim tehnologijama te bibliotekama koje se mogu koristiti unutar njih, izrada aplikacije je znatno olakšana. A tome pridonosi i činjenica da su te tehnologije svjetski rasprostranjene i popuralne pa se rješenje za skoro svaki problem vrlo lako i brzo pronađe na internetu. Nakon same izrade, potreno je stalno pratiti broj korisnika te njihove zahtjeve, te u skladu s tim ažurirati aplikaciju. Također je potrebno pratiti nove trendove u svijetu web aplikacija i dizajna korisničkog sučelja, te uvijek biti u korak s ostatkom svijeta.

LITERATURA

[1] Anderson, R. i Smith, S.: „ASP.NET Core fundamentals“, s Interneta, https://docs.microsoft.com/en-us/aspnet/core/fundamentals/?view=aspnetcore-3.1&tabs=linux, 30.03.2020.

[2] Anderson, R. i Smith, S.: „ASP.NET Core Middleware“, s Interneta, https://docs.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-3.1, 15.07.2020.

[3] Angular Tim: „What is Angular?“, s Interneta, https://angular.io/guide/what-is-angular, 08.03.2021.

[4] Angular Tim: „Sharing data between child and parent directives and components“, s Interneta, https://angular.io/guide/what-is-angular, 08.03.2021.

[5] Nwamba, C: „Understanding Angular Property Binding and Interpolation“, s Interneta, https://www.telerik.com/blogs/understanding-angular-property-binding-and-interpolation, 03.06.2019.

[6] Auth0 Tim: „Understand How You Can Use Auth0“, s Interneta, https://auth0.com/docs/get-started/auth0-overview, 05.02.2020.

[7] JWT Tim: „Introduction to JSON Web Tokens“ , s Interneta, https://jwt.io/introduction, 20.05.2019.

[8] Nepoznat Autor, „Visual Studio“, s Interneta, https://www.incredibuild.com/integrations/visual-studio, pristup 17.08.2021.

[9] VS Code Tim, „Why did we build Visual Studio Code?“, s Interneta, https://code.visualstudio.com/docs/editor/whyvscode, pristup 17.08.2021.

[10] Nepoznat Autor, „Auth0 Alternatives & Competitors“, s Interneta, https://www.g2.com/products/auth0/competitors/alternatives, pristup 24.08.2021.

[11] Microsoft Tim, „The Repository Pattern“, s Interneta, https://docs.microsoft.com/pt-br/previous-versions/msp-n-p/ff649690(v=pandp.10), 27.04.2010.

[12] Angular Tim: „NgModules“, s Interneta, https://angular.io/guide/ngmodules, pristup 25.08.2021.

[13] Oloruntoba S.: „SOLID: The First 5 Principles of Object Oriented Design“, s Interneta, https://www.digitalocean.com/community/conceptual_articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design, 17.12.2020.

[14] MDN suradnici: „An overview of HTTP“, s Interneta, https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview, 14.08.2021.

[15] MDN suradnici: „HTTP headers“, s Interneta, https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers, 19.08.2021.

POPIS OZNAKA I KRATICA

API	Application Programming Interface
ASP	Active Server Page
CIAM	Customer Identity and Access Management
CLI	Command Line Interface
CSS	Cascading Style Sheets
ECDSA	Elliptic Curve Digital Signature Algorithm
e2e	End To End
EF	Entity Tramework
DDoS	Distributed Denial Of Service
DI	Dependency Injection
DOM	Document Object Model
Dtos	Data Transfer Objects
IDE	Integrated Development Environment
IoC	Inversion of Control
IoT	Internet of Things
IP	Internet Protocol
JSON	JavaScript Object Notation
JWT	JSON Web Token
GDPR	General Data Protection Regulation
GUI	Graphical User Interface
HIPAA	Health Insurance Portability and Accountability Act
HMAC	Hash Based Message Authentication Code
HTML	Hyper Text Markup Language
HTTP	Hypertext Transfer Protocol

MIME	Multipurpose Internet Mail Extensions
MFA	Multi Factor Authentication
MSSQL	Microsoft Structured Query Language
Mvc	Model View Controler
npm	Node Package Manager
REST	Representational State Transfer
RSA	Rivest Shamir Adleman
PCI DSS	Payment Card Industry Data Security Standard
PHP	Hypertext Preprocessor
SAML	Security Assertion Markup Language
SMS	Short Message Service
SOC2	Systems and Organizations Controls 2
SPA	Single Page Application
SSO	Single Sign On
SQL	Structured Query Language
UI	User Interfaceo
URL	Uniform Resource Locator
VIP	Very Important Person
VS	Visual Studio

SAŽETAK

Ovim diplomskim radom je prikazan proces pregleda tehnologija, dizajniranja rješenja, te same implementacije. Također je i razjašnjena smisao i potreba za ovakvom aplikacojim. Izbor korištenih tehnologija ovisi o velikom broju čimbenika. U ovom radu su izabrane ASP.NET Core, Angular, MSSQL te Auth0. Spomenute tehnologije su vrlo populare te imaju širok spektar biblioteka što znatno olakašava posao.

Aplikacija je podijeljena u nekoliko dijelova. Prva u nizu je baza podataka koja je implementirana u MSSQL uz „code-first“ princip. Baza podataka je spremište podataka u koje se podatci spremaju, iz kojeg se podatci dohvaćaju, te u kojem je omogućena izmjena spremljenih podataka. Zatim sljedi infrastrukturni dio serverske strane aplikacije koji je odgovoran za rad sa spremištem podataka odnosno bazom podataka. On sadrži konfiguraciju baze te konfiguraciju mapiranja objekata u tablice. Također sadrži repozitorije preko kojih se razmjenjuju podatci sa bazom. Idući je domenski dio koji sadrži domenske modele, sučelja te servise. Unutar servisa je implemtirana poslovna logika. Servsi su također posrednici između aplikacijske te infrastrukturne razine. Aplikacijska razina je razina koju konzumira klijentska strana aplikacije odnosno Angular.

Angular je vrlo popularan okvir za izradu korisničkog dijela aplikacije. Angular se gradi korištenjem komponenti. A komponente za svoju logiku konzumiraju servise injektirane kroz konstruktor. Na korisničkom sučelju je prikazan grafički dizajn definiran unutar HTTML predloška komopnente. Angular konzumira aplikacijsku razinu tako da šalje HTTP zahtjeve te tako razmjenjuje podatke sa serverskom stranom aplikacije. A na posljetku je prikazana implementacija Auth0 softvera za jedinstvenu prijavu.

Ključne riječi: ASP.NET Core, Angular, Auth0, jedinstvena prijava, rezervacije

Title: Using a single login in web applications

SUMMARY

This thesis presents the process of reviewing technologies, designing solutions, and implementation them. The meaning and need for such an application have also been clarified. The choice of technologies used depends on a large number of factors. ASP.NET Core, Angular, MSSQL, and Auth0 are selected in this thesis. The mentioned technologies are very popular and have a wide range of libraries, which makes the job much easier.

The application is divided into few parts. The first in the series is a database that is implemented in MSSQL with the "code-first" principle. A database is a data warehouse in which data is stored, from which data is retrieved, and in which it is possible to modify saved data. This is followed by the infrastructure part of the back-end part of the application, which is responsible for working with the data warehouse (database). It contains the configuration of the database and the configuration of mapping objects to tables. It also contains repositories through which data is exchanged with the database. The next is the domain part which contains domain models, interfaces, and services. Business logic is implemented within the service. Services are also intermediaries between the application and infrastructure levels. Application-level is the level consumed by the front-end part of the application (Angular).

Angular is a very popular framework for creating the front-end part of an application. Angular is built using components. And the components for their logic consume services injected through the constructor. The user interface shows the graphic design defined within the HTTML component template. Angular consumes the application layer by sending HTTP requests and in that way exchanging data with the back-end part of the application. And finally, the implementation of Auth0 single sign-on software is shown.

Keywords: ASP.NET Core, Angular, Auth0, Unique Login, Reservations