Rezervacija termina u uslužnim salonima

Škarpa, Leo

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 634

REZERVACIJA TERMINA U USLUŽNIM SALONIMA

Leo Škarpa

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 634

REZERVACIJA TERMINA U USLUŽNIM SALONIMA

Leo Škarpa

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 634

Pristupnik:	**Leo Škarpa (0036522703)**
Studij:	Računarstvo
Profil:	Programsko inženjerstvo i informacijski sustavi
Mentor:	prof. dr. sc. Vedran Mornar

Zadatak: **Rezervacija termina u uslužnim salonima**

Opis zadatka:

Ostvariti sustav za rezervaciju termina u uslužnim salonima. Sustav se sastoji od aplikacije za vlasnike salona i mobilne aplikacije za korisnike. U aplikaciji za vlasnike uređuju se kalendari, izrađuju profili salona te pregledavaju statistički podaci. U mobilnoj aplikaciji korisnici mogu napraviti svoj profil, pretraživati salone, imati uvid u njihove profile, uređivati popis favorita te rezervirati termin. Mobilna aplikacija prati povijest rezervacija i podsjeća na buduće rezervacije putem push notifikacija. Ostvariti i sustav autentikacije i autorizacije. Posebnu pažnju posvetiti ergonomiji i jednostavnosti programskog sučelja. Sustav ostvariti koristeći neki od JavaScript programskih okvira na strani klijenta, Nest.js na strani poslužitelja te bazu podataka PostgreSQL .

Rok za predaju rada: 28. lipnja 2024.

Zahvala

Ovaj diplomski rad posvećujem svojoj obitelji, koja je tijekom cijelog mojeg školovanja (pogotovo zadnjih pet godina) bila tu uz mene i gurala me dalje makar ni sâm ponekad nisam htio; bez Vas ne bi bio danas tu gdje jesam.

Zahvaljujem i kolegici Luciji Lozančić na savršenom dizajnu za aplikaciju, kolegi i dugogodišnjem prijatelju Filipu Bugarinu koji je prolazio sve uspone, padove i neprospavane noći na ovom fakultetu sa mnom, te svima koji su mi pomogli pri izradi ovog rada svojim savjetima, podrškom i preporukama, posebno mom mentoru prof. dr. sc. Vedranu Mornaru.

Sadržaj

Uvod

Tehnologija se danas integrirala u naš cijeli život, od škole i fakulteta sve do poslovnog svijeta. Sve više ljudi koristi tehnologiju, odnosno sustave, kao pomoć pri radu, ali kao i vitalni dio njihovog posla. Međutim, u određenim poslovima se tehnologija sporo integrira. Razloga je za to više: od prekompleksnih radnji koje bi ti sustavi trebali izvršavati, do toga da se ljudi teško mijenjaju te stoga ostaju vjerni dokazanim metodama.

Jedni od takvih su saloni (frizerski, saloni ljepote, itd.). Svi znamo kako rezervacija u salonima funkcionira: nazovemo broj, odaberemo uslugu te rezerviramo termin. Na prvi pogled djeluje jednostavno, tri koraka i gotovi smo. Međutim, ukomponirati te korake u sustav na način da on može u potpunosti zamijeniti ta jednostavna 3 koraka je puno kompleksniji zadatak. To nam govori i činjenica da trenutno na tržištu u Republici Hrvatskoj postoji samo jedan sustav koji to omogućava.

Sada zamislite da ste vlasnik salona ili radnik u istom, koliko bi vam ovakav sustav olakšao posao. Ne biste se trebali javljati na telefon tijekom radnog vremena, niti biste trebali brinuti o tome kada možete nekog „ugurati“ u već prekompleksan kalendar. Idealno, zar ne?

Ovaj sustav, naziva MeNext, koji sam izradio kao dio diplomskog rada upravo se bavi time. Sustav bi trebao pomoći vlasnicima i radnicima u salonima da bezbrižno mogu obavljati svoj posao, te mušterijama da na jednostavniji način mogu odabrati uslugu i termin bez zvanja i čekanja na liniji. Sustav također omogućuje vlasnicima pregled statistika u svojevoljno odabranom intervalu kako bi imali uvid u sve informacije koje se tiču njihovog salona.

1. Korištene tehnologije

1.1. ReactJS

ReactJS[1] je popularna JavaScript biblioteka, razvijena u Facebooku, koja je promijenila način na koji programeri izrađuju korisnička sučelja. Glavni koncept React-a je njezina arhitektura temeljena na komponentama, gdje je korisničko sučelje podijeljeno na komponente koje se mogu ponovno koristiti unutar aplikacije. Ovaj pristup pisanju koda omogućava ponovnu upotrebu, te poboljšava mogućnost održavanja i skalabilnosti velikih aplikacija. Svaka komponenta u React-u može upravljati svojim stanjem (*state*) što olakšava i pomaže u radu sa složenim korisničkim interakcijama i dinamičkim podacima. Još jedna inovacija koja je došla sa React-om je virtualni DOM koja služi za optimiziranje prikazivanja na način da ažurira samo dijelove korisničkog sučelja koji su promijenjeni, umjesto da osvježi cijelo korisničko sučelje.

React je također poseban po tome što podržava jednosmjerni protok podataka, što znači da podaci teku u jednom smjeru te se na taj način lakše razumije kôd i uklanjaju pogreške. To se omogućava korištenjem stanja i svojstava (*props*). Svojstva se koriste za prijenos podataka od nadređene do podređene komponente, dok se stanje koristi za upravljanje lokalnim podacima unutar same komponente. Pored toga, React ima izrazito jak ekosustav sa brojim bibliotekama i alatima, poput React Router za navigaciju te Recoil-a za upravljanje globalnim stanjem. Ovakav ekosustav omogućuje nama, programerima, stvaranje od jednostavnih web stranica do složenih web aplikacija.

React je deklarativne prirode, što znači da se mi programeri brinemo o izgledu korisničkog sučelja u ovisnosti o stanju, a React se brine o ažuriranju korisničkog sučelja kako bi odgovarao tom stanju. Ovaj deklarativni pristup, uz pomoć JSX-a, koji je proširenje React-a za pisanje JavaScripta unutar HTML-a, pomaže programerima pri pisanju i razumijevanju kôda te prilikom čitanja tuđeg.

Također, React-ova popularnost pomaže nama, programerima, u učenju i poboljšavanju vještina zbog velikog broja resursa, tutoriala i biblioteka.

1.2. TypeScript

TypeScript[2], nadskup JavaScripta koji je razvio Microsoft, služi kao dodatak za statičke tipove. Njegov sustav tipova pomaže u otkrivanju pogrešaka tijekom kompiliranja, umjesto tijekom izvođenja, te na taj način može značajno smanjiti broj pogrešaka i poboljšati kvalitetu kôda. Ovo je pogotovo korisno za velike aplikacije, poput ove, gdje složenost i veličina kôda mogu otežati pronalazak i otklanjanje pogrešaka. TypeScript čini kod predvidljivijim i lakšim za refaktoriranje te poboljšava održavanje.

Jedna od glavnih značajki TypeScript-a je njegova kompatibilnost s već postojećim JavaScript kôdom. Programeri mogu polako i postupno uvoditi TypeScript u projekt, s obzirom da je pravilan JavaScript kôd također pravilan TypeScript kôd. Takav inkrementalni model usvajanja omogućuje programerima da polagano uvode TypeScript bez da u potpunosti moraju prepisivati postojeći kôd. Osim toga, TypeScript podržava moderni JavaScript što omogućuje programerima da koriste najnovije ECMAScript standarde čak i ako rade aplikaciju za starije preglednike.

Alati i podrška za integrirana razvojna okuženja (IDE) u TypeScript-u još dodatno poboljšavaju iskustvo programera. Uz TypeScript, programeri mogu koristiti napredne alate za dovršavanje kôda, navigaciju, pretragu i refaktoriranje u okruženjima poput Visual Studio Code-a (VSCode), čime se njihova produktivnost i brzina povećava. Statički tipovi koje nam TypeScript omogućuje također pomažu s dokumentacijom, budući da tipovi sami po sebi služe kao oblik dokumentacije. To dodatno olakšava novim programerima razumijevanje kôda.

TypeScript također ima mogućnost tipizirati sučelja, generičke tipove i zaključivanje tipova što olakšava izradu i rad sa složenijim tipovima i strukturama podataka. Na taj način TypeScript donosi najbolje od oba svijeta te postaje sve popularniji prilikom izrade kompleksnih i skalabilnih web aplikacija.

1.3. Git

Git[3], distribuirani sustav kontrole verzije koji je stvorio Linus Torvalds, trenutno je standard za upravljanje izvornim kôdom u IT industriji. Git omogućuje više programera da surađuju na projektu praćenjem promjena u njegovoj bazi kôda te čuvanjem i održavanjem povijesti svih promjena. Za razliku od centraliziranih sustava za kontrolu verzija, Git-ova distribuiranost omogućava svakom programeru da ima potpunu kopiju repozitorija, uključujući cijelu povijest, direktno na svojem računalu. Ovakva arhitektura poboljšava suradnju unutar timova jer omogućuje programerima da rade izvan mreže i lokalno rade promjene, a zatim kada se vrate na mrežu mogu sinkronizirati svoje promjene s drugim članovima tima.

Najvažnija značajka Git-a su njegove grane (*branch*). Grane omogućuju programerima da se odvoje od glavnog kôda kako bi razvili nove značajke, popravili greške ili eksperimentirali s novim idejama. Jednom kada su gotovi sa radom na grani, mogu ju spojiti (*merge*) natrag u glavnu granu, integrirajući promjene. To znači da glavna grana, i samim time već funkcionalan kôd koji može biti na produkciji, ostaje stabilna dok se novi posao obavlja. Taj model grananja u kombinaciji s mogućnostima spajanja omogućuje većem broju programera da rade istodobno na drugačijim poslovima, čime se izrada aplikacija ubrzava.

Tijekom spajanja grana može doći do konflikata (*conflict*), međutim Git i za to ima rješenje. Git pruža mehanizam za rješavanje takvih konflikata, koji su dosta česti prilikom integracija promjena iz različitih grana. Putem svoje mogućnosti predaje (*commit*) i alata za predaju, Git pomaže programerima identificirati i riješiti konflikte, osiguravajući da će glavna grana ostati dosljedna i funkcionalna. Osim toga, Git-ove mogućnosti praćenja povijesti i zapisivanja olakšavaju praćenje promjena, prepoznavanje kôda i zašto su promjene napravljene. Sve ove funkcionalnosti pomažu programerima u otkrivanju pogrešaka i eventualnih revizija.

Git je trenutno nezamjenjiv dio u cijeloj industriji, te u kombinaciji sa drugim alatima poput GitHub-a, može se naći u svakom razvojnom okruženju.

1.4. GitHub

GitHub[7] je popularna web platforma za hosting i suradnju na Git repozitoriju. On predstavlja središnje mjesto za razvoj softvera, omogućuje programerima da pohranjuju svoj kod, prate promjene i surađuju s drugim članovima tima. Jedna od glavnih značajki su zahtjevi za povlačenjem (*pull request*), koji omogućuju programerima provjeru promjena kôda i integraciju tih promjena u glavnu granu nakon pregleda. Ovaj pristup olakšava pregled kôda, otkrivanje pogrešaka i održavanje kvalitete i konvencija pisanja kôda.

GitHub također nudi napredne alate za upravljanje projektima, kao što su sustavi za praćenje problema, upravljanje zadacima i organizaciju projekta. Ti alati omogućuju timovima praćenje napretka, dodjeljivanje zadataka i bolje upravljanje resursima. GitHub također ima i integriranu platformu za kontinuiranu integraciju i isporuku (CI/CD), GitHub Actions, koja omogućuje automatizaciju tijekom rada, uključujući izgradnju, testiranje i implementaciju aplikacija. Sa korištenjem GitHub Actions-a, timovi mogu biti sigurni da će kôd i promjene biti dosljedne i kvalitetne.

Još jedna značajka GitHub-a je podrška za projekte otvorenog kôda. GitHub pruža besplatni hosting za projekte otvorenog kôda, što programerima olakšava dijeljenje koda. Mnoge popularne biblioteke i okviri (*framework*) otvorenog kôda, poput React-a, Angular-a i Linux kernela, nalaze se na GitHub-u, što olakšava doprinos i suradnju na tim projektima. Osim toga GitHub ima bogatu dokumentaciju, vodiče i zajednicu koja podržava nove korisnike i pomaže im u učenju i razvoju svojih vještina.

1.5. Postman

Postman[4] je jedan od alata za API testiranje i razvoj. Postman pruža korisničko sučelje za kreiranje HTTP zahtjeva, što olakšava programerima testiranje i uklanjanje pogrešaka na početku razvoja. Postman dopušta korisnicima da specificiraju metode zahtjeva, zaglavlja, parametre i tijela (*body*), te na taj način omogućuje sveobuhvatno testiranje RESTful API-ja i drugih web usluga. Postman je također vrlo brz sa slanjem zahtjeva i prikazivanjem odgovora što programerima omogućuje da brzo dijagnosticiraju probleme i potvrde funkcionalnosti API-ja.

Postman podržava i izradu složenih skripti putem JavaScripta, te time omogućuje automatizaciju testiranja i provjeru odgovora. Uz pomoć ugrađenih biblioteka i funkcija, programeri mogu pisati skripte koje potvrđuju odgovore, analiziraju podatke i izvode razne druge operacije. Ova mogućnost automatiziranja testiranja pomaže programerima da budu učinkovitiji tijekom testiranja i da nemaju potrebu za ručnim pregledom i testiranjem. Postman također ima mogućnost kreiranja i dijeljenja zbirki zahtjeva, i samim time olakšava rad između timova i standardiziranje testiranja.

Međutim, Postman-ova ključna značajka je podrška za okruženja. Okruženja omogućuju programerima da definiraju varijable koje se mogu direktno koristiti u zahtjevima. Time omogućuje lakše testiranje API-ja u različitim konfiguracijama, kao što su razvojna (develop), testna (*staging*) i produkcijska (*production*). Koristeći varijable, programeri mogu vrlo lako mijenjati vrijednosti koje su iste unutar istog okruženja, poput URL-a, autentifikacijskog tokena i drugih parametara bez potrebe da mijenjaju cijele zahtjeve.

Postman također dodaje mogućnost automatske autentifikacije putem kolačića (*cookie*), Bearer autentifikacijskog tokena ili nekim drugim načinom. Također omogućuje manipulaciju kolačićima, poput dodavanja, brisanja i mijenjanja vrijednosti kolačića.

1.6. Docker

Docker[8] je kontejnerska platforma koja programerima i administratorima sustava omogućuje izgradnju, testiranje i implementaciju aplikacija u izoliranim okruženjima. Ta okruženja nazivaju se spremnicima (*container*). Spremnici su lagani, prenosivi i sadrže sve potrebne komponente za pokretanje aplikacije poput kôda, biblioteke i konfiguracijske datoteke. Uz pomoć Docker-a programeri mogu osigurati rad aplikacije bez obzira u kojem se okruženju izvodi, što smanjuje probleme povezane sa različitim konfiguracijama sustava.

Jedna od glavnih prednosti Docker-a je njegova sposobnost da pojednostavi tijek rada za razvoj i implementaciju aplikacija. Sve što je potrebno je Dockerfile-a, skripta koja definira kako izgraditi Docker sliku. Docker slike su osnovni blokovi Docker-a i sadrže sve potrebne komponente za pokretanje aplikacije. Korištenjem slika, programeri mogu osigurati da će razvojna, testna i produkcijska okruženja biti ista, i tako smanjiti rizike od pogrešaka u implementaciji.

Docker podržava i orkestraciju spremnika putem alata kao što su Docker Compose i Kubernetes. Docker Compose omogućuje definiranje i pokretanje aplikacija s više spremnika što olakšava upravljanje složenim aplikacijama koje se sastoje od više komponenti i koje to zahtijevaju. Kubernetes je popularni sustav za orkestraciju spremnika koji omogućuje automatsko upravljanje, skaliranje i implementaciju kontejnerskih aplikacija u klasterima (*cluster*). Ako timovi koriste Kubernetes, povećavaju si dostupnost i skalabilnost njihovih aplikacija.

Docker se ne mora koristiti samo za složene aplikacije, već ga se može koristiti i tijekom izrade malih jednostavnih aplikacija. Na taj način, ako ikada aplikacija postane veća i složenija, spremni ste na skaliranje.

1.7. PostgreSQL

PostgreSQL, ili Postgres,[6] jedan je od najraširenijih sustava za upravljanje relacijskim bazama podataka otvorenog kôda. On podržava širok raspon značajki zbog kojih je vrlo često idealan izbor za mnoge aplikacije, od jednostavnih web stranica do velikih i složenih sustava za analizu podataka. Jedna od glavnih njegovih prednosti i značajka je njegova usklađenost sa SQL standardom, što omogućuje programerima laganu migraciju i integraciju s drugim sustavima.

Postgres je također poznat po naprednoj podršci za tipove podataka, uključujući prilagođene tipove, nizove, JSON i hstore. Ovakva fleksibilnost i količina tipova omogućuje pohranu i manipulaciju složenim podacima izravno u bazi, čime smanjuje potrebu za dodatnim slojevima aplikacije. Postgres podržava napredne značajke kao što su transakcije, zaključavanje na razini zapisa, referentni integritet i usklađenost s ACID-om.

Još jedna značajka koja čini Postgres nezamjenjivim je njegova proširivost. Putem sustava dodataka korisnici mogu proširiti funkcionalnosti baze dodavanjem novih tipova podataka, operatora, funkcija i indeksa. PostGIS, jedan od popularnijih dodataka, omogućuje prostornu analizu i rad s geolokacijskim podacima, i samim time čini Postgres jakim alatom za geografske informacijske sustave (GIS). Uz to, s obzirom da je Postgres otvorenog kôda, ima jaku zajednicu koja kontinuirano doprinosi njegovom razvoju i pruža podršku kroz brojne resurse i dokumentaciju.

1.8. NestJS

NestJS[5] je napredni okvir za izgradnju učinkovitih, pouzdanih i skalabilnih aplikacija na strani poslužitelja. On koristi TypeScript kao svoj primarni jezik, što omogućava statične tipove i poboljšanu čitljivost kôda. NestJS je inspiriran Angular-om te je tako usvojio modularnu arhitekturu koja olakšava organizaciju i održavanje velikih aplikacija. Jedna od ključnih značajki NestJS-a je upravo ta modularnost. Ideja je da se aplikacija podijeli u module (*module*) koju grupiraju povezane komponente poput kontrolera (*controller*), servisa (*service*) i pružatelja usluga. Na taj način se osigurava ponovna upotreba kôda i poboljšava mogućnost održavanja, dodavanja novih značajki ili ažuriranja postojećih.

Kako bi NestJS omogućio modularnost, koriste se dekoratori, kao što su *@Module()*, *@Controller()* i *@Injectable()*, i tako programeri mogu jednostavno definirati module i njihove komponente. NestJS se temelji na Express.js, ali se može koristiti i Fastify za poboljšane performanse. Ovo također ide u korist programerima jer je Express dobro poznat i naširoko korišten, a Fastify daje bolje performanse u određenim slučajevima. NestJS također integrira podršku za razne ORM-ove, poput TypeORM, Sequelize ili Mongoose, što olakšava rad sa raznim bazama podataka, uključujući SQL i NoSQL baze podataka.

Osim toga, NestJS ima ugrađenu podršku za provjeru valjanosti podataka, autentifikaciju i autorizaciju, kao i kreaciju vlastitih dekoratora za olakšan i ubrzan razvoj. NestJS se također ističe i podrškom za mikro servise. Arhitektura mikro servisa omogućuje „razbijanje“ aplikacije na manje neovisne usluge koje povremeno međusobno komuniciraju putem protokola kao što su HTTP, TCP ili gRPC. On olakšava stvaranje i upravljanje mikro servisima tako što pruža jednostavne API-je za komunikaciju i orkestraciju usluga.

NestJS ima i bogat ekosustav i snažnu zajednicu koja kontinuirano daje brojne dodatke i alate koji još unaprjeđuju i proširuju NestJS. Neki od popularnih dodataka su @nestjs/swagger za generiranje Swagger/OpenAPI dokumenata, @nestjs/graphql za integraciju sa GraphQL-om i @nestjs/passport za jednostavniju i bržu implementaciju autentifikacije i autorizacije. Ovakvi dodaci olakšavaju integraciju s drugim tehnologijama i alatima, te programerima daju veću fleksibilnost i funkcionalnosti koje su im potrebne.

1.9. TypeORM

TypeORM[9] je jedan od najpoznatijih i najkorištenijih objektno-relacijskih mapera (ORM) za TypeScript i JavaScript. Nalazi se u ekosustavu Node.js-a te omogućuje razvoj aplikacija relacijske baze podataka na način koji je intuitivan, jednostavan i usklađen s objektno orijentiranim programiranjem. TypeORM podržava razne baze podataka, uključujući MySQL, PostgreSQL, MariaDB, SQLite, Microsoft SQL Server i druge, zbog čega je najčešći izbor za razne projekte.

Jedna od ključnih prednosti TypeORM-a je njegov deklarativni pristup definiranju entiteta i njihovih odnosa. Programeri koriste dekoratore za definiranje entiteta, stupaca i odnosa, što čini izrazito cisti i pregledan kôd. Na primjer, korištenjem dekoratora *@Entity()*, *@Column()* i *@OneToMany()* programeri mogu vrlo brzo i jasno definirati strukturu baze podataka i odnose između tablica izravno unutar njihovih TypeScript klasa.

TypeORM također daje napredne funkcije kao što su migracije, alati za izradu upita i transakcije. Migracije omogućuju programerima praćenje i upravljanje promjenama baze podataka tijekom vremena. Uz pomoć alata za migracije, programeri mogu primijeniti ili povući promjene u strukturi baze podataka bez da naruše dosljednost spremljenim podacima. Alati za izradu upita pružaju programerima jednostavan način izrade i izvršavanja SQL upita nalik objektno-orijentiranom programiranju, bez da moraju ručno pisati SQL. Transakcije osiguravaju da se skup operacija izvodi kao cjelina, kako ne bi došlo do narušavanja integriteta podataka.

Još jedna od značajki zbog kojih je TypeORM toliko popularan, te zašto sam ga ja odabrao, je njegova integracija s popularnim okvirima kao što su NestJS i Express. Korištenjem TypeORM-a s NestJS-om programeri mogu lako i jednostavno iskoristiti sve funkcionalnosti koje nude oba alata. TypeORM se lako integrira s NestJS-om putem modula i pružatelja usluga te na taj način omogućava lakše upravljanje entitetima i povezivanje s bazom podataka. Zbog svega toga, TypeORM ima jaku zajednicu koja stalno radi na novim resursima za učenje, dodatcima i popravcima grešaka.

2. Arhitektura

2.1. Klijentska strana

Klijentska strana aplikacije je izrađena uz pomoć ReactJS-a kao okvira. Razlog odabira React-a naspram ostalih okvira je njegova jednostavnost i popularnost, te olakšan način izrade korisničkog sučelja uz pomoć komponenata.

Uz React, korištene su i druge biblioteke koje su olakšale i pojednostavile određene dijelove logike i koda.

2.1.1. Recoil

Recoil[10] je biblioteka za upravljanje stanjem u React aplikacijama koja također dolazi od Facebook tima. Ona nam omogućuje da na jednostavan i fleksibilan način upravljamo globalnim stanjem, što smanjuje potrebe za *prop drillingom* iz roditeljskih (*parent*) u dječje (*children*) komponente. Recoil zbog toga ima posebne koncepte kao što su atomi i selektori. Atomi su jedinice stanja koje se mogu dijeliti između različitih komponenata, a selektori omogućuju čitanje i izvođenje stanja iz atoma.

Jedna od glavnih prednosti Recoil-a je njegova sposobnost minimiziranja ponovnog renderiranja komponenti. Zbog atoma, promjene stanja utječu samo na one komponente koje te atome i koriste što može poboljšati rad cijele aplikacije. Recoil također pruža mogućnost implementacije asinkronih operacija putem selektora, što može biti bitno ukoliko radimo s podacima dohvaćenim s poslužitelja.

U MeNext aplikaciji se koristi više atoma, od kojih su *user* i *business* atom najbitniji. Postavljajući njih u atome, možemo ih koristiti kroz cijelu aplikaciju bez da brinemo o drugim komponentama. On nam također olakšava postavljanje i resetiranje atoma na *default* vrijednosti, što nam je bitno prilikom promjena tih entiteta ili odjave korisnika.

2.1.2. React Query

React Query[11] je biblioteka za upravljanje dohvaćanjem podataka u React aplikacijama. Njegova glavna funkcija je olakšati rad sa asinkronim podacima, te na taj način omogućiti jednostavno dohvaćanje, sinkronizaciju i ažuriranje podatka bez da se mi moramo brinuti o tome. React Query automatizira puno uobičajenih i čestih zadataka koje programer mora raditi prilikom dohvata podataka sa poslužitelja.

Također, React Query ima mogućnost pred memoriranja podataka i optimiziranja osvježavanja istih, što poboljšava performanse i izradu korisničkog sučelja. On održava lokalnu pred memoriju podataka i automatski ažurira podatke kada se promijene na poslužitelju.

React Query je vrlo popularna biblioteka i koristi se u skoro svakoj aplikaciji, od jednostavnih web aplikacija do onih složenih. U korist mu također ide to što je implementacija napravljena uz pomoć *hook*-a, što je u modernom React-u postao standard. Osim toga, React Query omogućuje rad sa raznim HTTP bibliotekama, vrlo je fleksibilan i otvorenog koda, zbog čega sam ga i odabrao za pomoć u izradi boljeg korisničkog sučelja.

2.1.3. Axios

Axios[12] je popularna biblioteka HTTP klijenta za JavaScript (i TypeScript) koja olakšava slanje zahtjeva poslužiteljima iz preglednika. Vrlo je jednostavan za korištenje, pruža API za slanje GET, POST, PUT, DELETE i drugih vrsta zahtjeva. Jedna od glavnih značajki Axios-a je njegova jednostavnost prilikom rada sa asinkronim operacijama, za koje on koristi *Promise* objekte.

Axios je u potpunosti konfigurabilan, od zaglavlja, vremenskih ograničenja i autorizacije do svih drugih opcija koje bi velike aplikacije imale potrebu tražiti od jednog HTTP klijenta. Osim toga, jedan od osobno bitnijih značajki koje Axios pruža su presretači zahtjeva i odgovora. Sa njima, on nam pruža mogućnost da se zahtjev, odnosno odgovor, mijenja prije nego što bude poslan ili obrađen. Ovo postaje izuzetno važno i korisno ukoliko postoji potreba za dodavanje tokena za provjeru autentičnosti, globalno rukovanje pogreškama ili obradu odgovora.

Zbog svih ovih razloga, Axios je postao neizbježna biblioteka koju koristim u svakoj aplikaciji na kojoj radim, pa tako i na ovoj.

2.1.4. Atomic design system

Atomic design system[14] je metodologija za izgradnju sustava dizajna i korisničkih sučelja. Razvio ju je Brad Frost i od tada je postaje sve popularnija. Pristup je inspiriran kemijom i načinom na koji se atomi spajaju u molekule, pa zatim u organizme. Njegov cilj je pružiti programerima i dizajnerima modularan i skalabilan način razvoja komponenata.

Atomic design system se sastoji od pet razina, koje se spajanjem mogu strukturirati i organizirati u komponente:

- Atomi
- Molekule
- Organizmi
- Predlošci
- Stranice

2.1.4.1 Atomi

Atomi su osnovni elementi svakog dizajn sustava. U kontekstu Atomic dizajn sustava, atomi su najmanji i najosnovniji dijelovi koji se ne mogu dalje smanjiti. Oni uključuju osnovne HTML elemente poput tipki, naslova, paragrafa, polja za unos, ikona i drugih. Oni predstavljaju pojedinačne komponente koje se mogu ponovno koristiti kroz cijelu aplikaciju i izvršavaju samo jednu dužnost.

U MeNext aplikaciji primjer atoma bi bila „Button“ komponenta, koja se koristi kroz cijelu aplikaciju. Također, atomi su i naslovi (h1, h2, h3…) i paragrafi koji kroz cijelu aplikaciju imaju isti izgled i funkcionalnost.

Zbog njihove jednostavnosti, atomi su početni gradivni elementi za složenije sustave i komponente unutar istih. Najčešće se atomi izrađuju iz dizajna i to na početku izrade sustava, te se kasnije samo ponovno koriste. Na taj način se ostvaruje dosljednost i ujednačenost osnovnih stilova kroz cijelu aplikaciju. Također, ako dođe do ikakve promjene u dizajnu, lagano se promjena primijeni kroz jedan atom.

2.1.4.2 Molekule

Molekule su kombinacija dva ili više atoma. One zajedno čine osnovne UI komponente. Molekule su jednostavne, ali funkcionalne jedinice korisničkog sučelja koje imaju specifičnu zadaću i funkciju. Na primjer, jedna molekula može biti kombinacija polja za unos i tipke za pretragu, na taj način stvarajući mali formular za pretragu.

Molekule su jedan korak iznad atoma i prema složenijim komponentama jer kombiniraju prije navedene atome u funkcionalne dijelove korisničkog sučelja. U aplikaciji MeNext molekule su sva polja za formu. S obzirom da se koriste na više mjesta unutar aplikacije, imalo je smisla napraviti određena polja kao molekule, te ih tako ujednačiti kroz cijelu aplikaciju. Unutar običnog tekstualnog polja za unos se nalaze atomi za polje za unos, dvije ikone, naziv polja za unos te poruke koja se može prikazivati ispod polja.

Takvo kreiranje molekula omogućava programerima da ih ponovno koriste kroz aplikaciju, što ubrzava vrijeme razvoja aplikacije te olakšava primjenjivanje promjena u dizajnu.

2.1.4.3 Organizmi

Organizmi su složene UI komponente koje se sastoje od više molekula i/ili atoma. Organizmi su veće funkcionalne jedinice koje se cesto koriste kao dijelovi stranica ili aplikacije. Na primjer, zaglavlje stranice može biti organizam koji sadrži logo, navigaciju te *dropdown* za korisničke opcije.

Organizmi su tu da predstavljaju veće dijelove korisničkog sučelja koji obavljaju složenije funkcije. U MeNext jedan organizam je cijeli *sidebar*. On služi za navigaciju unutar aplikacije, a sastoji se od više molekula i atoma. Molekule unutar *sidebar*-a su linkovi koji vrše navigaciju, a atom je logo MeNext na vrhu *sidebar*-a.

Organizmi omogućavaju razvoj modularnih i skalabilnih dijelova sučelja. Njihovo kreiranje pomaže u organizaciji i strukturi sučelja, ne ponavljaju dijelova koda, te u konzistentnosti kroz cijelu aplikaciju jer se mogu lagano prilagoditi i ponovno koristiti na drugačijim dijelovima aplikacije.

2.1.4.4 Predlošci

Predlošci su strukture koje definiraju raspored organizama na stranicama. Oni pružaju kontekst komponentama te pokazuju kako bi se organizmi trebali formirati da bi sučelje imalo smisla. Predlošci se više koriste u dizajnu nego u implementaciji korisničkog sučelja, međutim svejedno ih se treba držati jer nam pokazuju raspored i hijerarhiju komponenti.

Oni većinom služe kao vodiči za izgradnju aplikacije, pokazuju nam kako bi komponente trebale zajedno raditi. Na primjer, predložak bi nam mogao pokazati raspored između zaglavlja, glavnog sadržaja, *sidebar*-a i podnožja. Ovaj raspored nam daje strukturu koju možemo pratiti kroz cijelu aplikaciju te na taj način lakše rasporediti kôd.

2.1.4.5 Stranice

Stranice su specifične instance predložaka, ona imaju stvarne podatke i sadržaj. One se također koriste u dizajnu više nego u programiranju, te su stranice ono što mi programeri zovemo „gotov dizajn“.

One su ključne za validaciju korisničkih zahtjeva te krajnjeg dizajna, jer je lakše promijeniti dizajn prije nego što je implementiran u kôd nego kasnije. Ovo također omogućuje testiranje sa klijentima i korisnicima, te ukoliko je stranica validna, može se započeti sa izradom aplikacije u kôdu.

Makar ponekad zna izgledati bespotrebno, kreiranje stranica omogućava uvid u cjelokupno korisničko sučelje i daje nam priliku da identificiramo i ispravimo probleme prije nego što se aplikacija implementira.

2.2. Poslužitelj

Na poslužiteljskoj strani aplikacije se koristio NestJS sa TypeScript-om, zajedno sa TypeORM-om i Express.js-om. Baza koja se koristila je PostgreSQL te se napravio REST API.

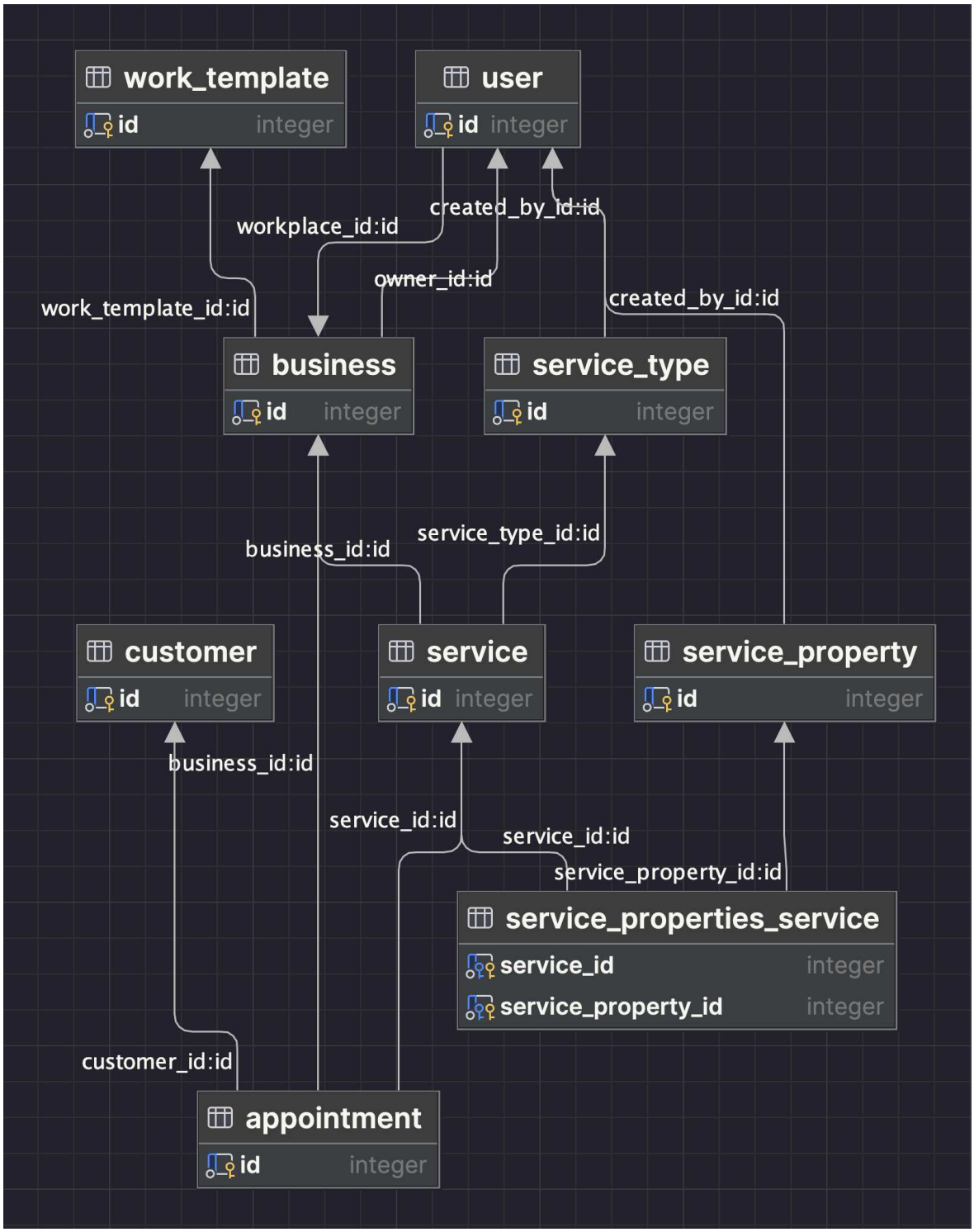
2.2.1. REST API

REST API[13] je jedan od stilova za dizajn distribuiranih sustava koji omogućava komunikaciju između klijenta i poslužitelja preko HTTP protokola. On koristi standardne HTTP metode kao što su GET, POST, PUT i DELETE, to omogućuje jednostavno upravljanje resursima putem URL-ova.

REST API je zasnovan na resursima, tako da svaki resurs ima jedinstveni identifikator, najčešće u obliku URL-a. Ovisno koja metoda se koristi prilikom slanja zahtjeva ta operacija će se izvršiti nad resursom, što čini REST API dosljedan i predvidljiv.

Jedna od glavnih prednosti REST API-ja je njegova skalabilnost i fleksibilnost. To je moguće zbog jednostavnosti HTTP-a i standardiziranog pristupa. Oni se također mogu integrirati s različitim platformama i tehnologijama, što ih čini najpopularnijim način izrade API-ja.

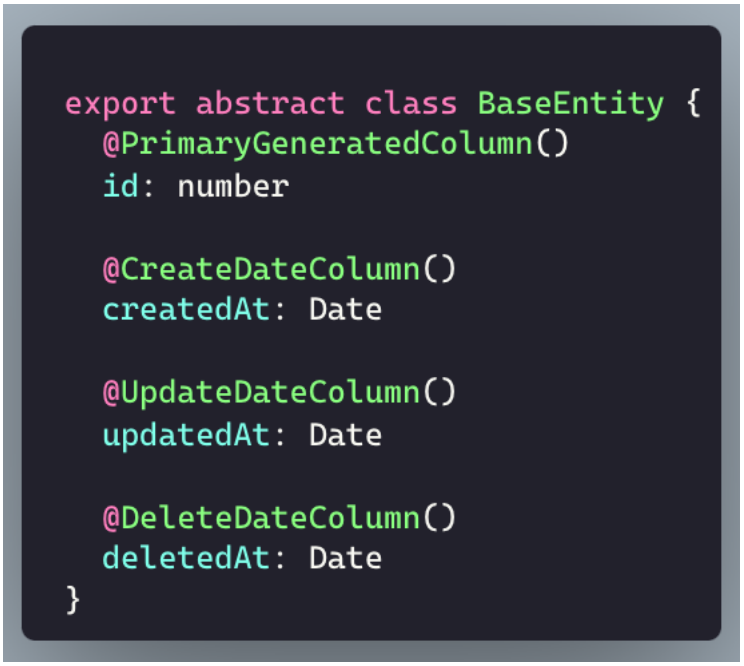
2.2.2. Dijagram baze podataka



Sl. 2.1 Dijagram baze podataka

2.2.3. Opis tablica baze podataka (entiteta)

Posto se na poslužitelju koristi NestJS sa TypeORM-om, radi lakše implementacije i jednostavnosti svaka klasa entiteta je napravljena od Base entiteta, koji sadrži `id` kao primarni ključ, `createdAt`, `updatedAt` i `deletedAt` kao *timestamp*-ove, te zasebnih kolumni koje svaki entitet ima.



Sl. 2.2 Base entitet

2.2.3.1 User

User tablica je tablica u koju se spremaju vlasnici salona te radnici u istim. Prilikom kreacije user entiteta se koristio STI, „*single table inheritance*“, što znači da se u bazi pokazuje samo jedna user tablica, dok se u kodu nalaze 3 entiteta. Jedan entitet, User, je glavni entitet kojeg preostala dva nasljeduju sa svojim posebnim kolumnama.

Ta preostala dva entiteta su Owner i Employee, koji su razdvojeni zbog povezanosti sa preostalim tablicama.

```ts
@Entity()
@TableInheritance({
  pattern: 'STI',
  column: {
    name: 'role',
    type: 'enum',
    enum: UserRoleEnum,
    enumName: 'user_role_enum',
    default: UserRoleEnum.OWNER,
  },
})
export class User extends BaseEntity {
  @Column()
  firstName: string

  @Column()
  lastName: string

  @Column({ unique: true })
  email: string

  @Column({ select: false })
  password: string

  @Column()
  phone: string

  @Column({
    name: 'role',
    type: 'enum',
    enum: UserRoleEnum,
    enumName: 'user_role_enum',
    default: UserRoleEnum.OWNER,
  })
  role: UserRoleEnum

  @OptionalColumn({ type: 'varchar' })
  avatar: Nullable<string>

  @OptionalColumn({ type: 'timestamp with time zone' })
  birthday: Nullable<Date>

  @OptionalColumn({ type: 'enum', enum: GenderEnum, enumName: 'gender_enum' })
  gender: Nullable<GenderEnum>
}
```

Sl. 2.3 User entitet

```
@ChildEntity(UserRoleEnum.OWNER)
export class Owner extends User {
  @OneToMany(() => Business, business => business.owner)
  ownedBusinesses: Business[]

  @OneToMany(() => ServiceType, serviceType => serviceType.createdBy)
  serviceTypes: ServiceType[]

  @OneToMany(() => ServiceProperty, serviceProperty => serviceProperty.createdBy)
  serviceProperties: ServiceProperty[]
}
```

Sl. 2.4 Owner entitet

```
@ChildEntity(UserRoleEnum.EMPLOYEE)
export class Employee extends User {
  @Column({ type: 'decimal' })
  salary: number

  @ManyToOne(() => Business, business => business.employees)
  workplace: Business
}
```

Sl. 2.5 Employee entitet

2.2.3.2 Appointment tablica

Appointment tablica je tablica u koju se spremaju sve rezervacije. Ona je povezana sa Service-om, Customer-om i Business-om, te sadrži `reservedFrom` i `reservedTo` *timestamp* kolumne koje određuju vrijeme rezerviranog termina. Također, kolumne `customerFirstName` i `customerLastName` se popunjavaju u slučaju da je rezervaciju napravio vlasnik ili radnik salona, u protivnom se rezervacija povezuje sa korisnikom (Customer).

```
@Entity()
export class Appointment extends BaseEntity {
  @Column({ type: 'timestamp with time zone' })
  reservedFrom: Date

  @Column({ type: 'timestamp with time zone' })
  reservedTo: Date

  @OptionalColumn({ type: 'varchar' })
  customerFirstName: Nullable<string>

  @OptionalColumn({ type: 'varchar' })
  customerLastName: Nullable<string>

  @OptionalColumn({ type: 'text' })
  notes: string

  @ManyToOne(() => Customer, customer => customer.appointments)
  customer: Customer

  @ManyToOne(() => Business, business => business.appointments)
  business: Business

  @ManyToOne(() => Service, service => service.appointments)
  service: Service
}
```

Sl. 2.6 Appointment entitet

2.2.3.3 Business tablica

U Business tablicu se spremaju svi podaci potrebni za opis određenog salona. S obzirom da vlasnik može imati više salona, kreirana je veza prema User-u. Također salon može imati više zaposlenika i Service-a pa su i te veze dodane. Business entitet je povezan još i sa WorkTemplate-om te Appointment-ima.

```
@Entity()
export class Business extends BaseEntity {
  @Column()
  name: string

  @Column()
  description: string

  @Column()
  address: string

  @OptionalColumn({ type: 'varchar' })
  cover: Nullable<string>

  @Column({ type: 'varchar', array: true, default: [] })
  gallery: string[]

  @ManyToOne(() => Owner, owner => owner.ownedBusinesses)
  owner: Owner

  @OneToMany(() => Employee, employee => employee.workplace)
  employees: Employee[]

  @OneToMany(() => Service, service => service.business, { eager: true, cascade: ['soft-remove'] })
  serviceMenu: Service[]

  @OneToOne(() => WorkTemplate, workTemplate => workTemplate.business, {
    eager: true,
    cascade: ['soft-remove'],
  })
  @JoinColumn()
  workTemplate: WorkTemplate

  @OneToMany(() => Appointment, appointment => appointment.business)
  appointments: Appointment[]
}
```

Sl. 2.7 Business entitet

2.2.3.4 Customer tablica

S obzirom da je User tablica napravljena za vlasnike i radnike salona, Customer tablica služi za korisnike druge strane aplikacije, oni koji rezerviraju termine. Entitet je otprilike isti kao i User entitet, uz dodatak povezanosti sa Appointment-ima kako bi korisnik na svom *dashboard*-u mogao vidjeti sve prošle i buduće rezervacije.

```
@Entity()
export class Customer extends BaseEntity {
  @Column()
  firstName: string

  @Column()
  lastName: string

  @Column({ unique: true })
  email: string

  @Column({ select: false })
  password: string

  @Column()
  phone: string

  @OptionalColumn({ type: 'varchar' })
  avatar: Nullable<string>

  @OneToMany(() => Appointment, appointment => appointment.customer)
  appointments: Appointment[]
}
```

Sl. 2.8 Customer entitet

2.2.3.5 Service tablica

Service tablica je tablica u koju se spremaju usluge. Svaka usluga je povezana sa ServiceProperty-ima i ServiceType-om. Service entitet je glavni entitet aplikacije, on povezuje većinu entiteta te sadrži najveću logiku s obzirom da cijeli proces rezervacije ide preko njega. Također, Service entitet ima mogućnost povezivanja na više ServiceProperty-a, s obzirom da svaka usluga u salonu može biti kompleksna i sadržavati više dodataka.

```
@Entity()
export class Service extends BaseEntity {
  @Column({ type: 'interval' })
  duration: IPostgresInterval

  @Column({ type: 'numeric', precision: 5, scale: 2, transformer: new ColumnNumericTransformer() })
  price: number

  @Column({ default: '€' })
  currency: string

  @OptionalColumn({ type: 'enum', enum: GenderEnum, enumName: 'gender_enum' })
  gender: Nullable<GenderEnum>

  @ManyToOne(() => Business, business => business.serviceMenu)
  business: Business

  @ManyToOne(() => ServiceType, type => type.services, {
    eager: true,
    cascade: ['update'],
  })
  serviceType: ServiceType

  @ManyToMany(() => ServiceProperty, properties => properties.services, {
    eager: true,
    cascade: ['update'],
  })
  @JoinTable({ name: 'service_properties_service' })
  serviceProperties: ServiceProperty[]

  @OneToMany(() => Appointment, appointment => appointment.service)
  appointments: Appointment
}
```

Sl. 2.9 Service entitet

2.2.3.6 ServiceProperty tablica

ServiceProperty entitet služi kao dodatak na Service entitet. On je vrlo jednostavan te pomoću njega vlasnici mogu dodati dodatne uvjete ili dodatke na uslugu. Iz toga razloga Service entitet ih može imati više, ali i ni jednog. Primjer ServiceProperty-a bi bili kratka kosa, duga kosa, plava kosa, itd. Oni pobliže definiraju uslugu na koju se spajaju te tako korisnik koji radi rezervaciju može lakše odlučiti koju uslugu želi.

```
@Entity()
export class ServiceProperty extends BaseEntity {
  @Column()
  name: string

  @OptionalColumn({ type: 'text' })
  description: Nullable<string>

  @ManyToMany(() => Service, service => service.serviceProperties)
  services: Service

  @ManyToOne(() => Owner, owner => owner.serviceProperties)
  createdBy: Owner
}
```

Sl. 2.10 ServiceProperty entitet

2.2.3.7 ServiceType tablica

ServiceType entitet je glavni dio usluge. On je taj koji se koristi na više usluga te označava glavnu uslugu bez dodataka. Primjer jednog ServiceType-a bi bilo šišanje. Nije nam bitno je li muško, žensko, kratka kosa ili duga. On označava samo najbitniji dio usluge te je stoga povezan sa Service entitetom.

```
@Entity()
export class ServiceType extends BaseEntity {
  @Column()
  name: string

  @OptionalColumn({ type: 'text' })
  description: Nullable<string>

  @OneToMany(() => Service, services => services.serviceType)
  services: Service[]

  @ManyToOne(() => Owner, owner => owner.serviceTypes)
  createdBy: Owner
}
```

Sl. 2.11 ServiceType entitet

2.2.3.8 WorkTemplate tablica

WorkTemplate tablica definira radno vrijeme salona (Business). Ona je također rastavljena na dvije vrste: DailyWorkTemplate i WeeklyWorkTemplate.

DailyWorkTemplate definira na svaki dan isto radno vrijeme, od ponedjeljka do nedjelje, ukoliko se nedjelja označi kao radnom.

WeeklyWorkTemplate definira na svaki tjedan isto radno vrijeme, što znači da vlasnik može definirati svaki dan u tjednu određeno radno vrijeme, a ono se ponavlja svaki budući tjedan.

```
@Entity()
@TableInheritance({
  pattern: 'STI',
  column: {
    name: 'type',
    type: 'enum',
    enum: WorkTemplateTypeEnum,
    enumName: 'work_template_type_enum',
  },
})
export class WorkTemplate extends BaseEntity {
  @OneToOne(() => Business, business => business.workTemplate)
  business: Business

  @Column()
  sundayIncluded: boolean

  @Column({ type: 'enum', enum: WorkTemplateTypeEnum, enumName: 'work_template_type_enum' })
  type: WorkTemplateTypeEnum
}
```

Sl. 2.12 WorkTemplate entitet

```
@ChildEntity(WorkTemplateTypeEnum.DAILY)
export class DailyWorkTemplate extends WorkTemplate {
  @Column({ type: 'time' })
  dailyOpenHoursFrom: string

  @Column({ type: 'time' })
  dailyOpenHoursTo: string
}
```

Sl. 2.13 DailyWorkTemplate entitet

```
@ChildEntity(WorkTemplateTypeEnum.WEEKLY)
export class WeeklyWorkTemplate extends WorkTemplate {
  @Column({ type: 'time' })
  mondayOpenHoursFrom: string

  @Column({ type: 'time' })
  mondayOpenHoursTo: string

  @Column({ type: 'time' })
  tuesdayOpenHoursFrom: string

  @Column({ type: 'time' })
  tuesdayOpenHoursTo: string

  @Column({ type: 'time' })
  wednesdayOpenHoursFrom: string

  @Column({ type: 'time' })
  wednesdayOpenHoursTo: string

  @Column({ type: 'time' })
  thursdayOpenHoursFrom: string

  @Column({ type: 'time' })
  thursdayOpenHoursTo: string

  @Column({ type: 'time' })
  fridayOpenHoursFrom: string

  @Column({ type: 'time' })
  fridayOpenHoursTo: string

  @Column({ type: 'time' })
  saturdayOpenHoursFrom: string

  @Column({ type: 'time' })
  saturdayOpenHoursTo: string

  @OptionalColumn({ type: 'time' })
  sundayOpenHoursFrom: Nullable<string>

  @OptionalColumn({ type: 'time' })
  sundayOpenHoursTo: Nullable<string>
}
```

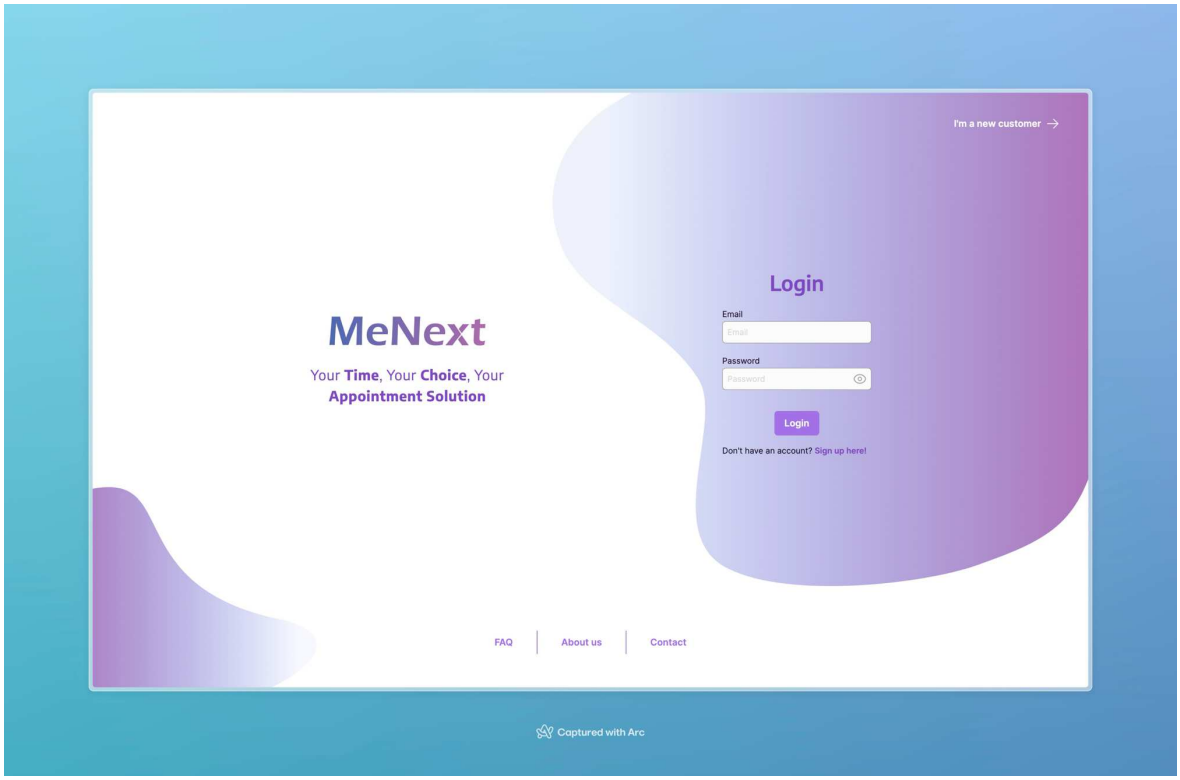
Sl. 2.14 WeeklyWorkTemplate entitet

3. Implementacija

3.1. Prijava

Stranica za prijavu korisnika je ista za vlasnika, zaposlenika i korisnika (Customer). Radi takve implementacije pojednostavljena je navigacija same aplikacije.

Login forma se sastoji od dva polja za tekstualni unos: email te lozinku, i gumba za slanje forme na poslužitelj. Također, ispod login forme nalazi se link na stranicu za kreaciju profila vlasnika, dok se na vrhu stranice nalazi link za nove korisnike (Customer).



Sl. 3.1 Login stranica

Funkcionalnost logina napravljena je uz pomoć NestJS-ovog okvira te biblioteke Passport.js. Za autentifikaciju se koristi kolačić (cookie), unutar kojeg je spremljen email korisnika te njegov id. Korištenje NestJS-ovog okvira omogućuje nam korištenje dekoratora uz koje možemo lakše zaštititi privatne rute te dohvatiti ulogiranog korisnika (Owner ili Customer).

```
@Injectable()
export class JwtStrategy extends PassportStrategy(Strategy, 'jwt') {
  constructor(
    @InjectRepository(User)
    private userRepository: Repository<User>,
    @InjectRepository(Customer)
    private customerRepository: Repository<Customer>,
  ) {
    super({
      jwtFromRequest: ExtractJwt.fromExtractors([
        JwtStrategy.extractJwtFromCookie,
        ExtractJwt.fromAuthHeaderAsBearerToken(),
      ]),
      secretOrKey: 'jwt-secret',
    })
  }

  private static extractJwtFromCookie(req: Request): string | null {
    if (req.cookies && 'access_token' in req.cookies && req.cookies.access_token.length > 0) {
      return req.cookies.access_token
    }

    return null
  }

  async validate(payload: JwtPayload) {
    const user = await this.userRepository.findOneBy({ id: payload.sub, email: payload.email })
    const customer = await this.customerRepository.findOneBy({
      id: payload.sub,
      email: payload.email,
    })
    return user ?? customer
  }
}
```

Sl. 3.2 Login strategija koja koristi Passport.js

```
@Injectable()
export class JwtGuard extends AuthGuard('jwt') {
  constructor(private reflector: Reflector) {
    super()
  }

  canActivate(context: ExecutionContext): boolean | Promise<boolean> | Observable<boolean> {
    const isPublic = this.reflector.getAllAndOverride(MetadataKeys.IS_PUBLIC, [
      context.getHandler(),
      context.getClass(),
    ])

    if (isPublic) return true

    return super.canActivate(context)
  }
}
```

Sl. 3.3 Dekorator koji provjerava je li osoba ulogirana

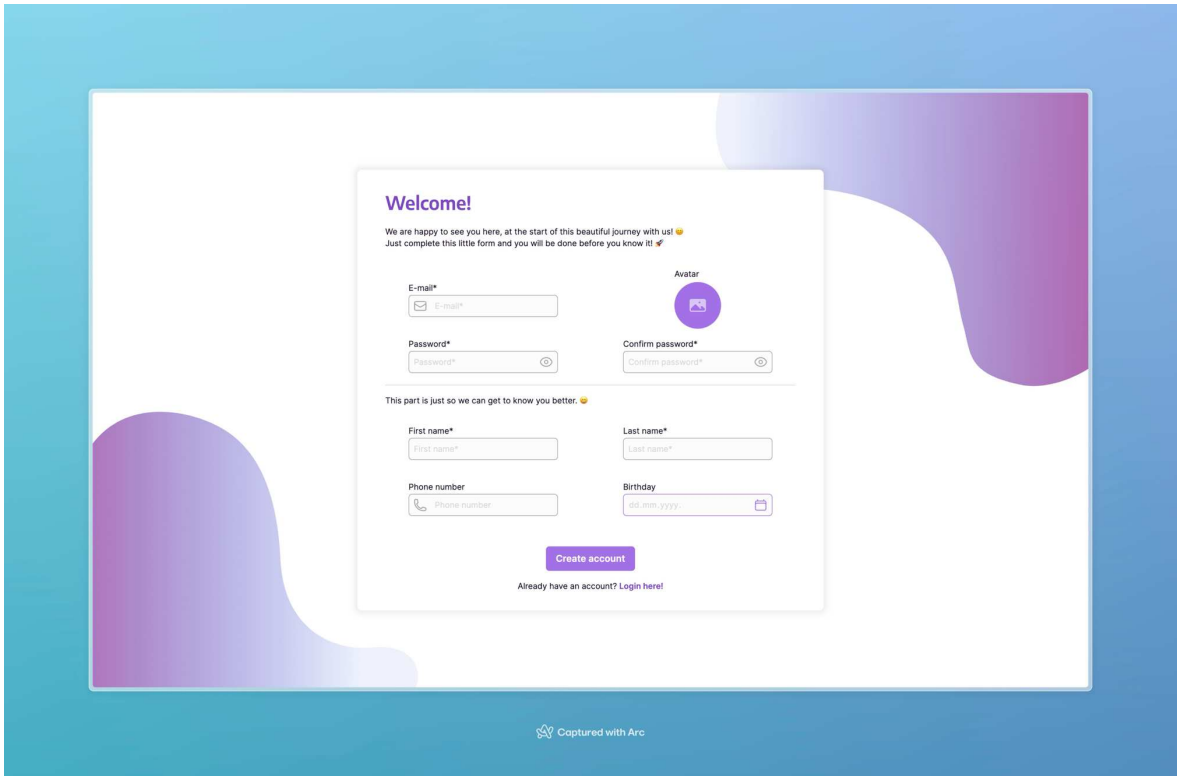
```
export const CurrentUser = createParamDecorator((data: unknown, ctx: ExecutionContext) => {
  const request = ctx.switchToHttp().getRequest<Request>()

  return request.user
})
```

Sl. 3.4 Dekorator za dohvat ulogiranog korisnika

3.2. Registracija

Registracija vlasnika i korisnika su vrlo slične, s obzirom da dolaze od sličnih entiteta. Registracijska forma sastoji se od više tekstualnih polja za unos, te polja za unos slike profila.

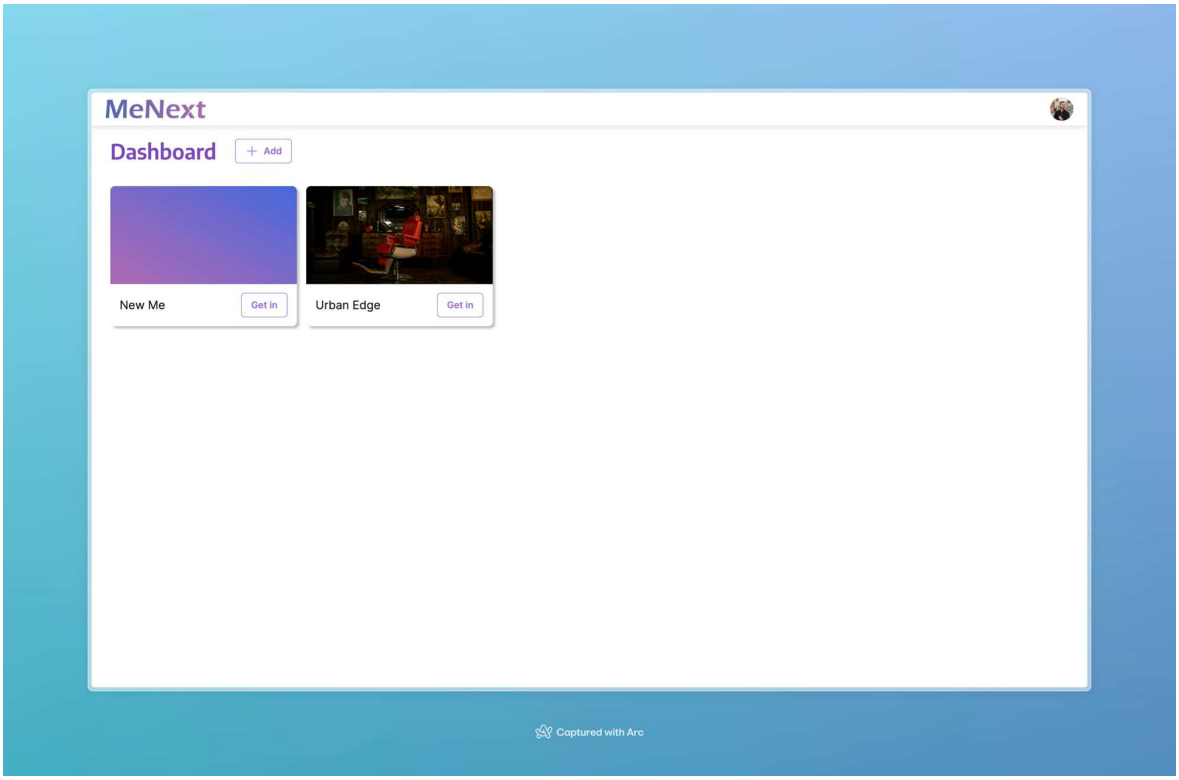


Sl. 3.5 Stranica za registraciju

3.3. Dashboard vlasnika

Na *dashboard* stranici vlasnika salona nalaze se kartice sa svim njegovim prije kreiranim salonima. Odavde vlasnik može odabrati jedan od prije kreiranih salona ili kreirati novi. Ukoliko je vlasnik novi član MeNext aplikacije, te nema kreiranih salona, aplikacija će ga automatski preusmjeriti na stranicu za kreiranje salona.

Također na zaglavlju stranice nalazi se logo aplikacije i *dropdown* sa slikom profila koju je vlasnik odabrao prilikom kreacije profila.



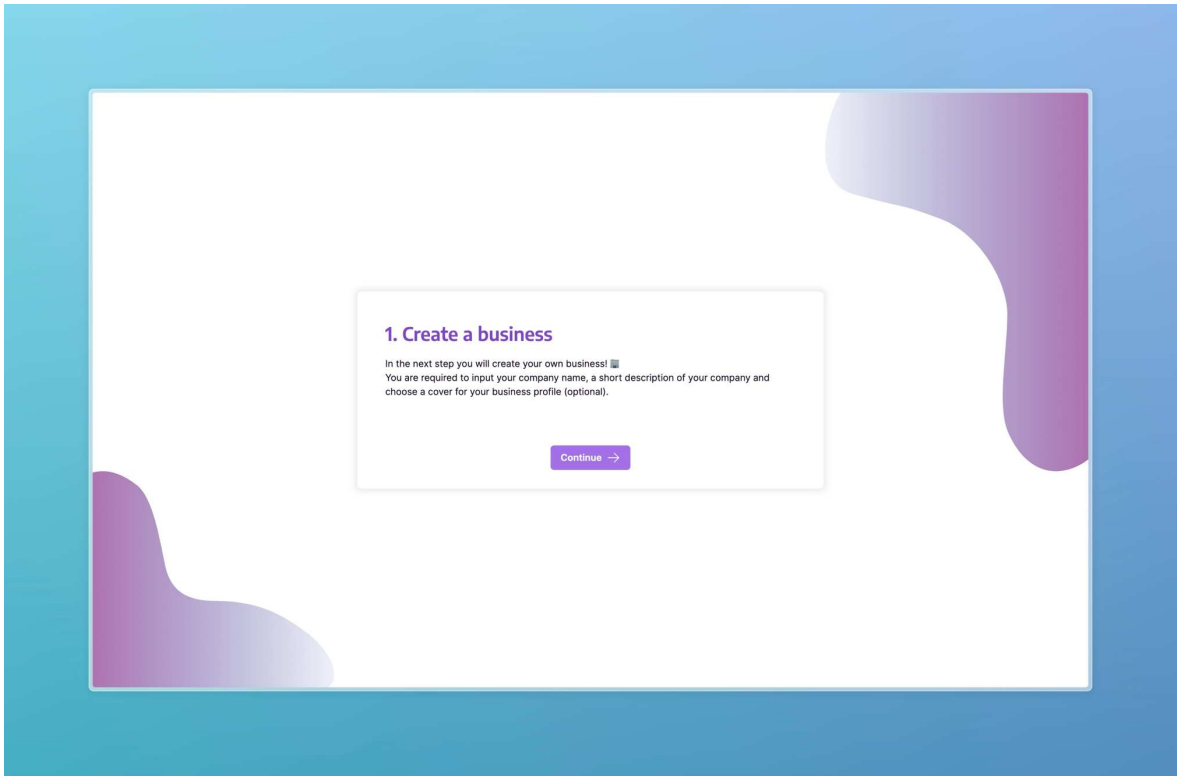
Sl. 3.6 Dashboard stranica vlasnika

3.4. Stranice za kreaciju salona

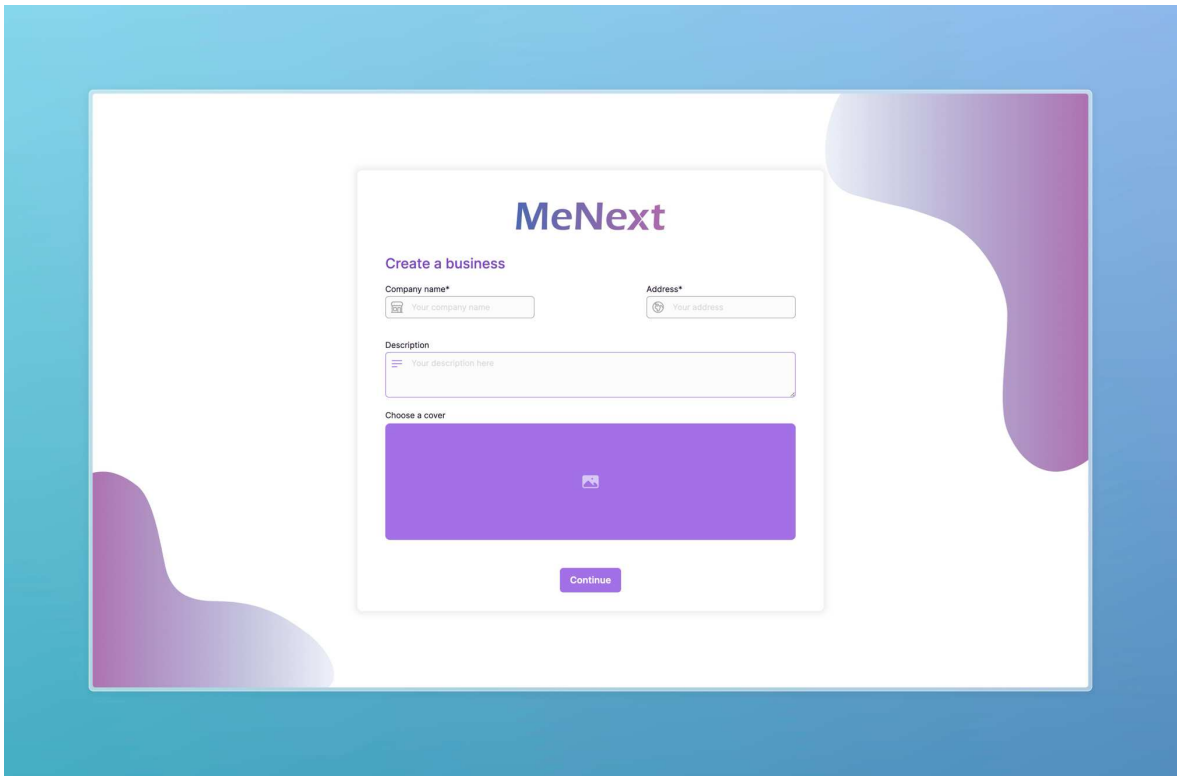
Kreacija salona je podijeljena na 6 zasebnih stranica. One su povezane u parove tako da prva stranica služi kao informacijska stranica koja objašnjava vlasniku što je potrebno ispuniti i na koji način.

3.4.1. Općenite informacije

Stranice za općenite informacije su prve u nizu kreacije salona. Prva stranica daje informacije o popunjavanju forme, a druga sadrži jednostavnu formu za brzu kreaciju salona sa općenitim informacijama: nazivom, adresom i opisom salona te slikom.



Sl. 3.7 Stranica sa informacijama o popunjavanju općenitih informacija

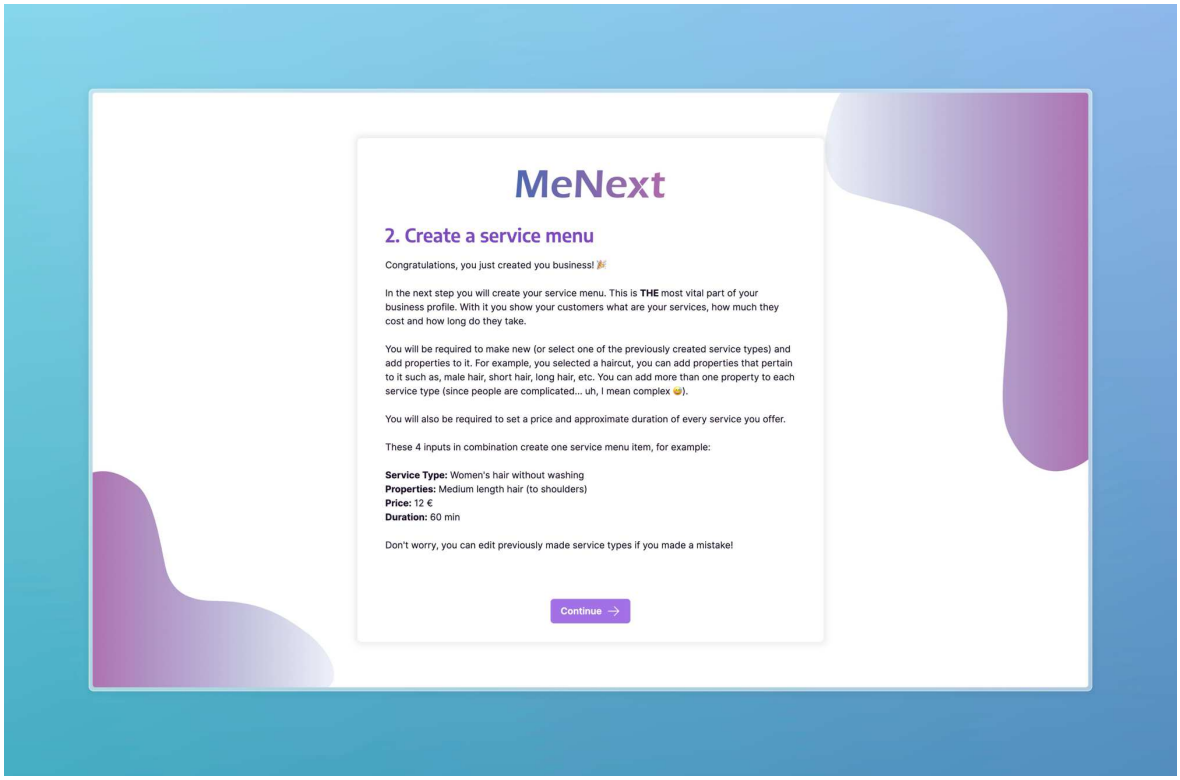


Sl. 3.8 Stranica sa formom za općenite informacije

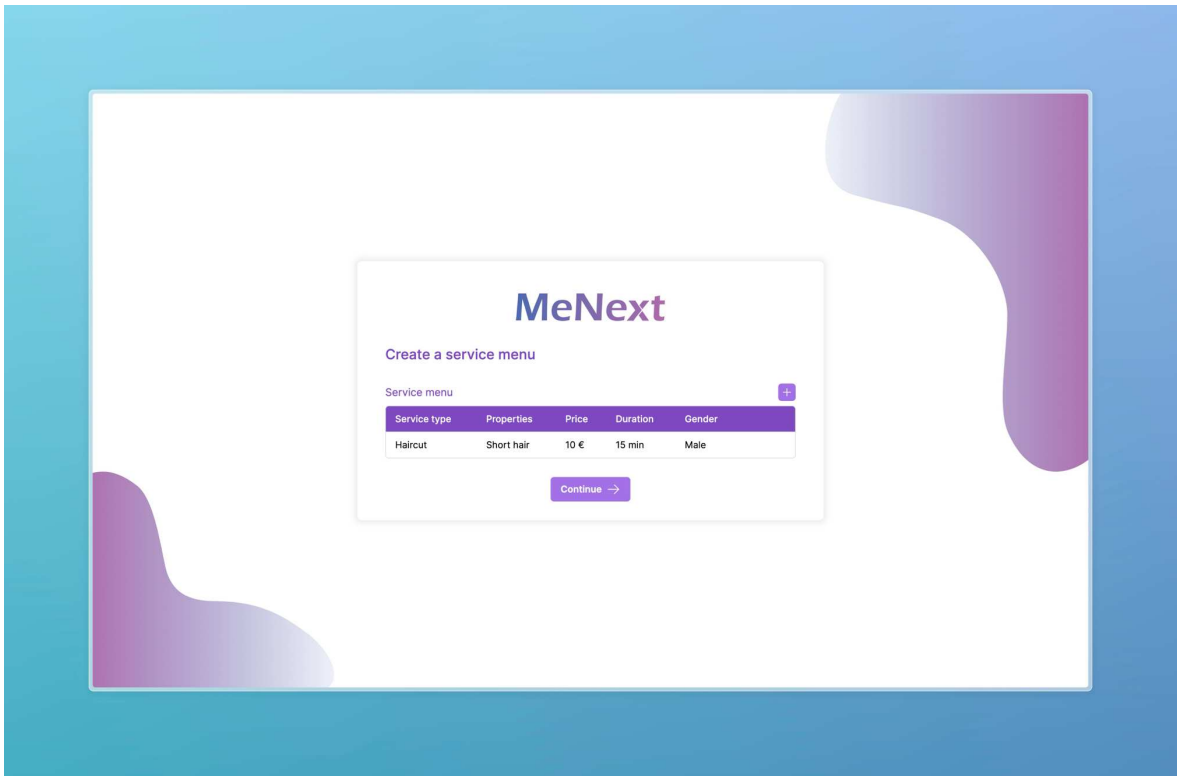
Klikom na gumb „Continue“ za nastavak, na poslužitelju se kreira novi salon te se vlasnika preusmjerava na idući korak kreacije: usluge.

3.4.2. Usluge

Sljedeće su stranice za kreaciju usluga. Prva daje informacije o popunjavanju forme za usluge te zašto su one bitne. Na drugoj stranici se nalazi tablica usluga te gumb za kreiranje novih.

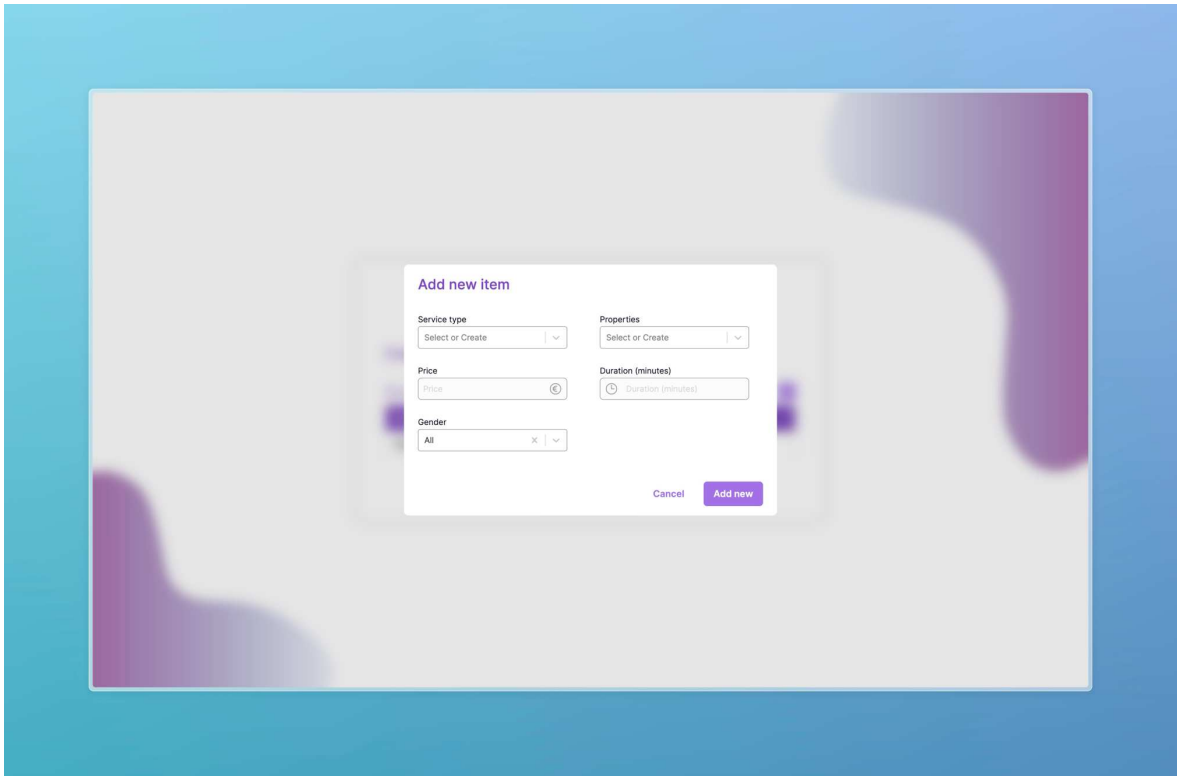


Sl. 3.9 Stranica sa informacijama o kreiranju usluga



Sl. 3.10 Stranica sa tablicom usluga

Nakon što se klikne na gumb za kreiranje nove usluge, otvori se modal u kojem se nalazi forma za kreaciju nove usluge.



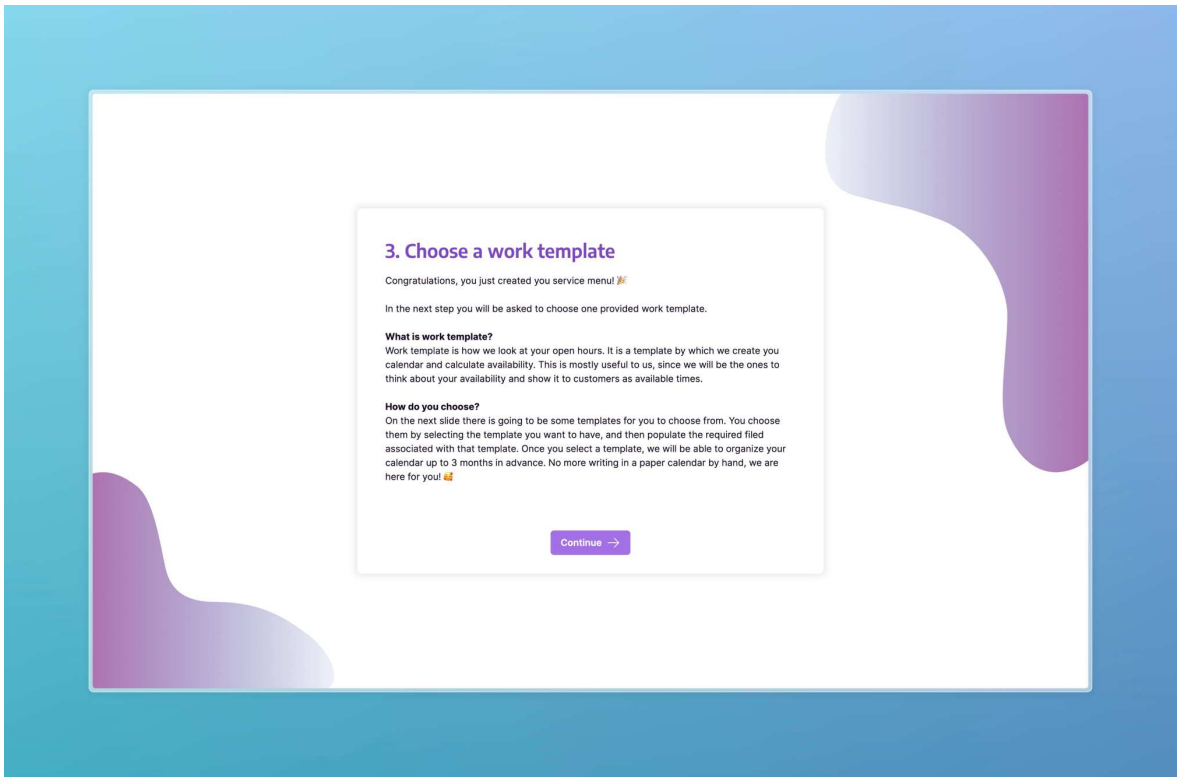
Sl. 3.11 Modal za kreaciju nove usluge

U formi se nalaze dva polja za odabir: ServiceType i ServiceProperty. Kao što sam već naveo prije, ServiceType je glavna usluga te se tako odabire samo jedna, dok su ServiceProperty-i dodatci na tu uslugu, pa se tako može odabrati i njih više. Taj spoj glavne usluge i dodataka omogućava preciznije i lakše određivanje cijene i trajanja usluge, što je bitno korisnicima jednako kao i vlasniku.

Nakon što je vlasnik zadovoljan sa napravljenim uslugama, klikom na gumb „Continue“ nastavlja na idući korak u kreaciji salona: radno vrijeme.

3.4.3. Radno vrijeme

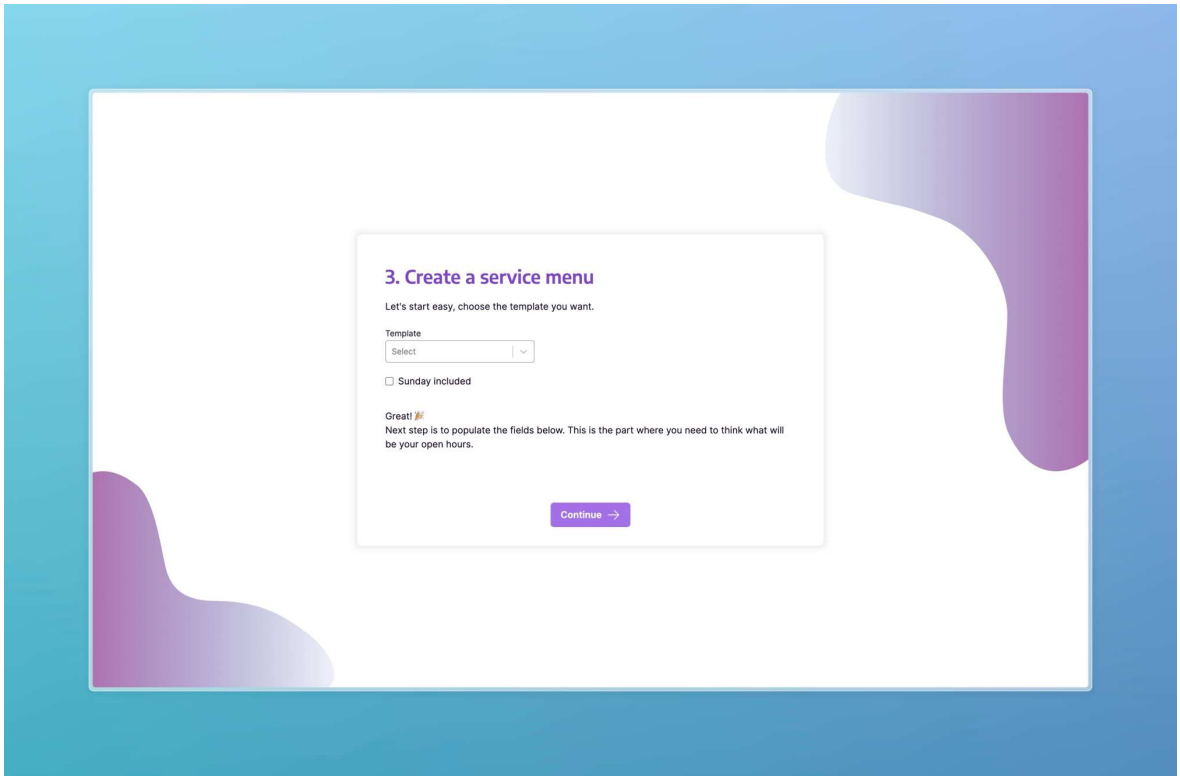
Na ovim stranicama vlasnik odabire svoje preferirano radno vrijeme salona. Na prvoj stranici kao i u prijašnja dva poglavlja nalaze se informacije o kreaciji radnog vremena te zašto je ona bitna i kako će pomoći pri rezervaciji termina.



Sl. 3.12 Stranica sa informacijama o kreaciji radnog vremena salona

Na drugoj stranici se nalaze dvije forme. Jedna je odmah vidljiva, a to je jednostavno polje za odabir gdje vlasnik može odabrati želi li napraviti dnevno (Daily) radno vrijeme ili tjedno (Weekly).

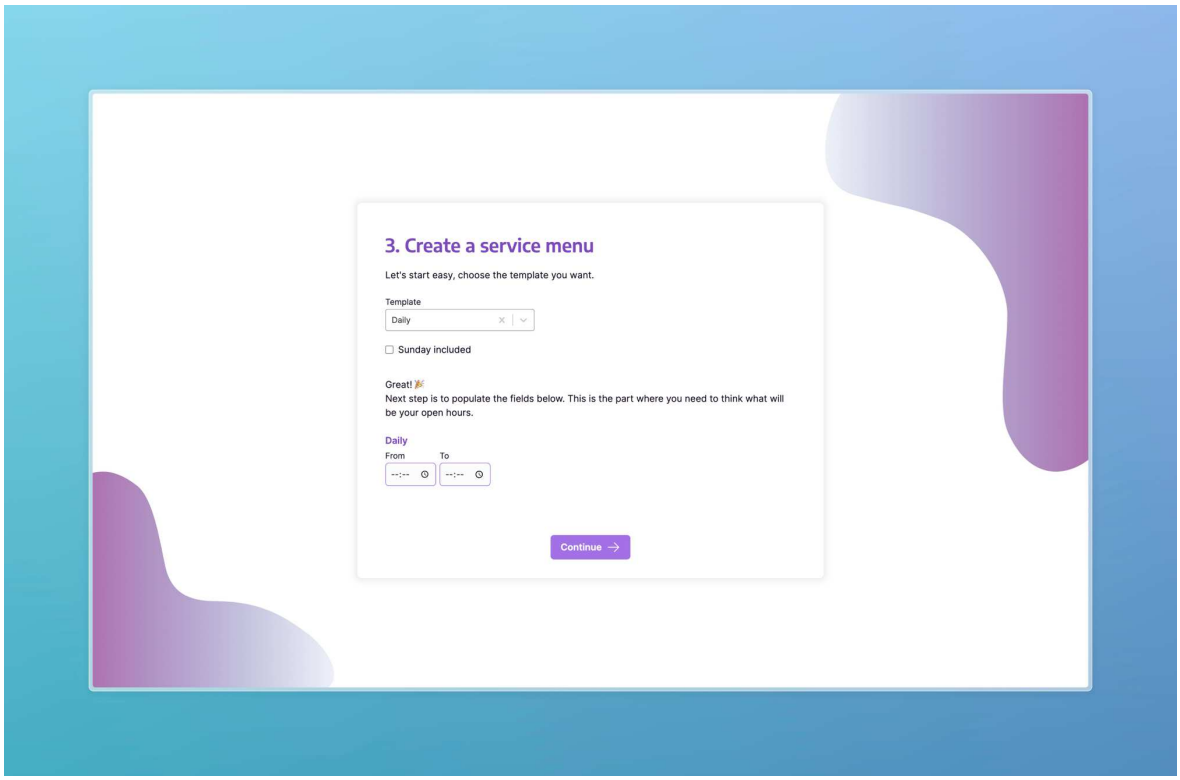
Jednom kad odabere jednu od te dvije opcije pokazuje se forma sukladna njegovom odabiru.



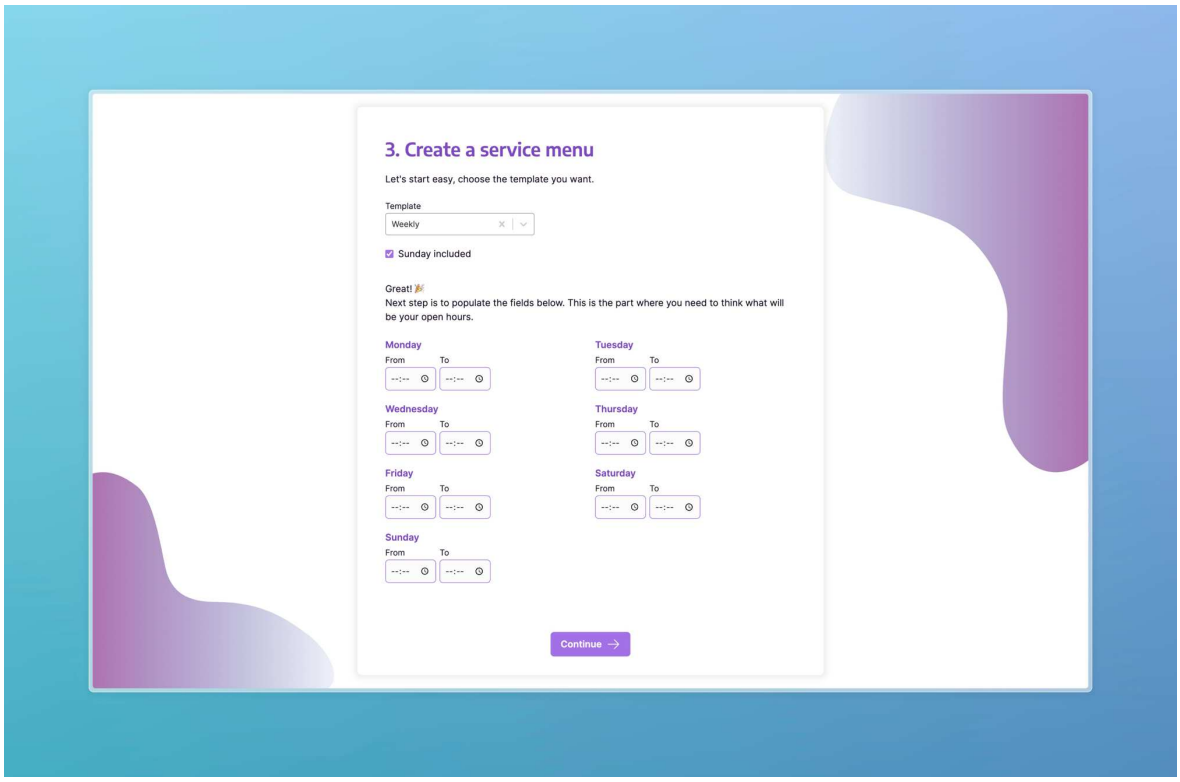
Sl. 3.13 Stranica za odabir radnog vremena

Ukoliko odabere dnevno radno vrijeme, pokazuje mu se forma za unos dnevnog radnog vremena, sa samo dva polja za unos vremena. On će se koristiti kao predložak za svaki dan, uključujući nedjelju ukoliko je vlasnik odabrao tu opciju.

Ukoliko je odabrao tjedno radno vrijeme, pokazat će mu se forma za unos tjednog radnog vremena po danima, od ponedjeljka do nedjelje, ako je vlasnik odabrao da salon radi i nedjeljom. Polja za unos su ista kao i kod dnevnog radnog vremena, samo ih je više. Sa tim će vlasnik napraviti predložak za tjedno radno vrijeme te će se ono koristiti za svaki budući tjedan.

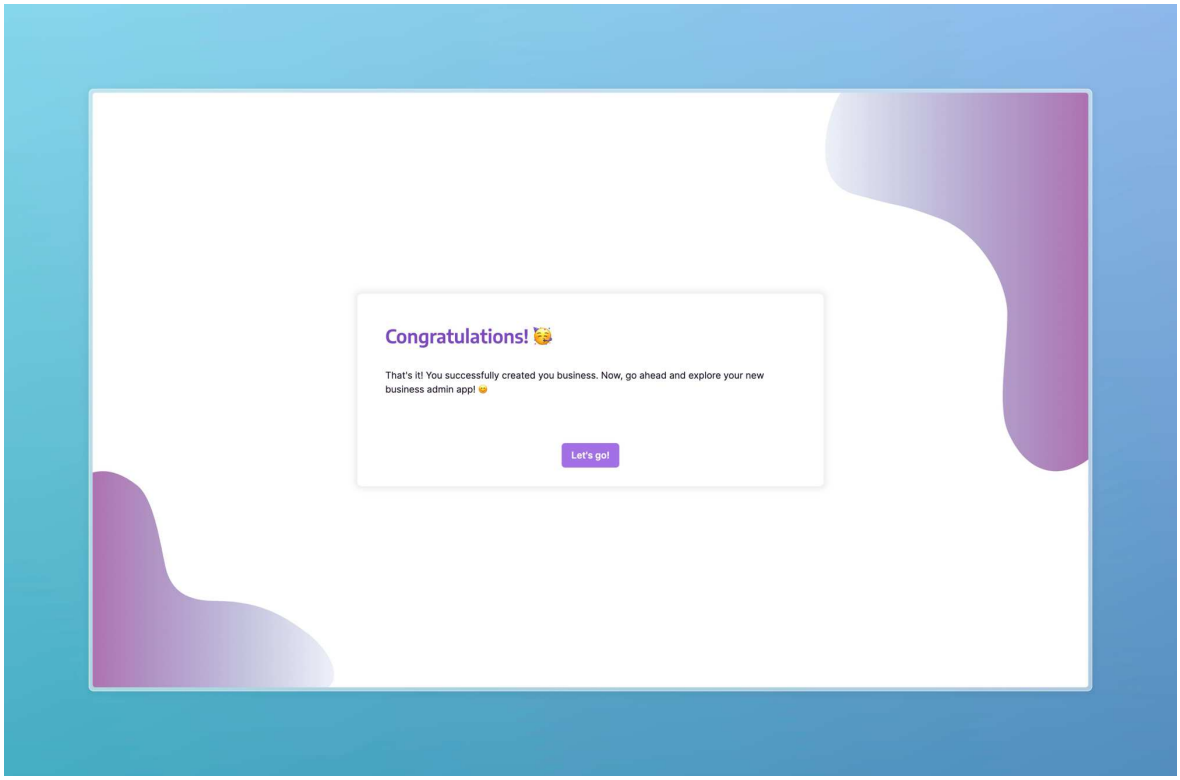


Sl. 3.14 Stranica sa formom za dnevno radno vrijeme



Sl. 3.15 Stranica sa formom za tjedno radno vrijeme

Ukoliko je vlasnik zadovoljan sa svojim radnim vremenom, klik na gumb „Continue“ vodi ga na zadnju stranicu na kojoj se vlasniku prikazuje informacija da je uspješno kreirao salon. Gumb ispod informacija vodi ga nazad na Dashboard gdje mu se pokazuje kartica novo kreiranog salona.

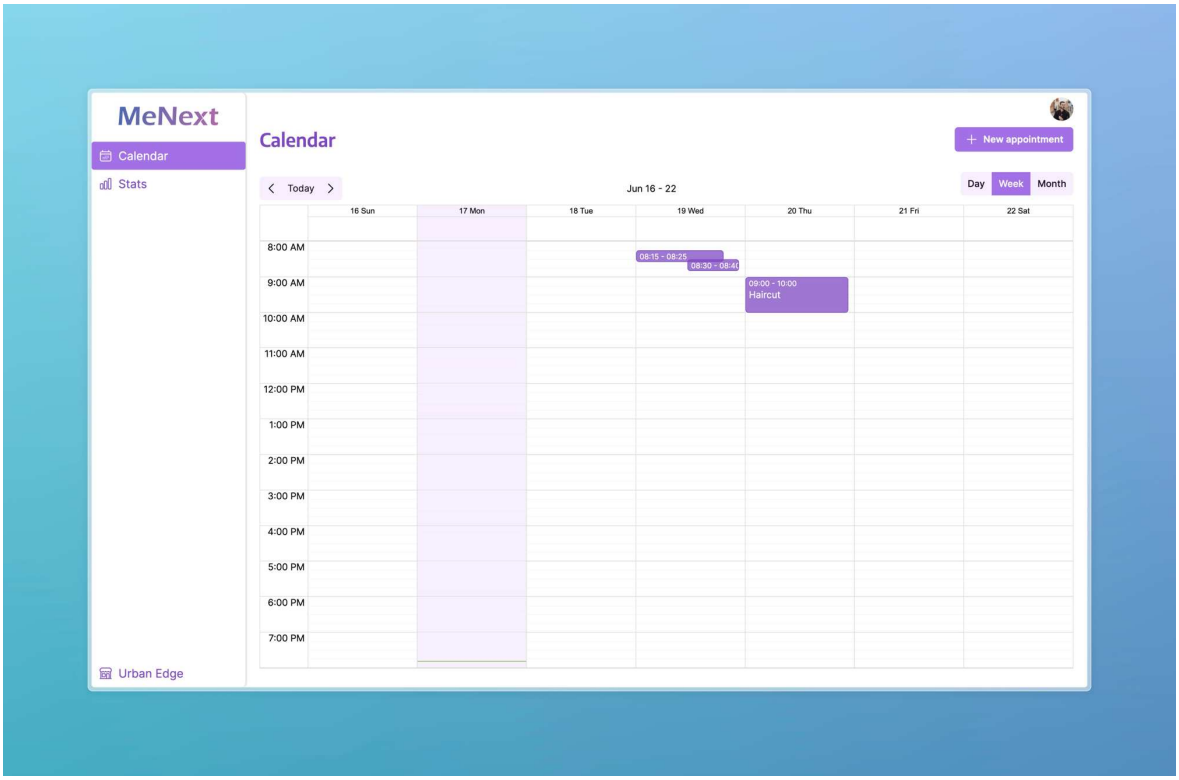


Sl. 3.16 Stranica sa informacijama o uspješnoj kreaciji salona

3.5. Kalendar

Nakon klika na karticu salona, navigacija vlasnika vodi u glavni dio aplikacije: kalendar. Stranica se sastoji od *sidebar*-a s lijeve strane sa linkovima za navigaciju, zaglavlja sa slikom profila iz kojeg izlazi *dropdown* sa dodatnim linkovima za navigaciju, te glavnog prostora u sredini aplikacije.

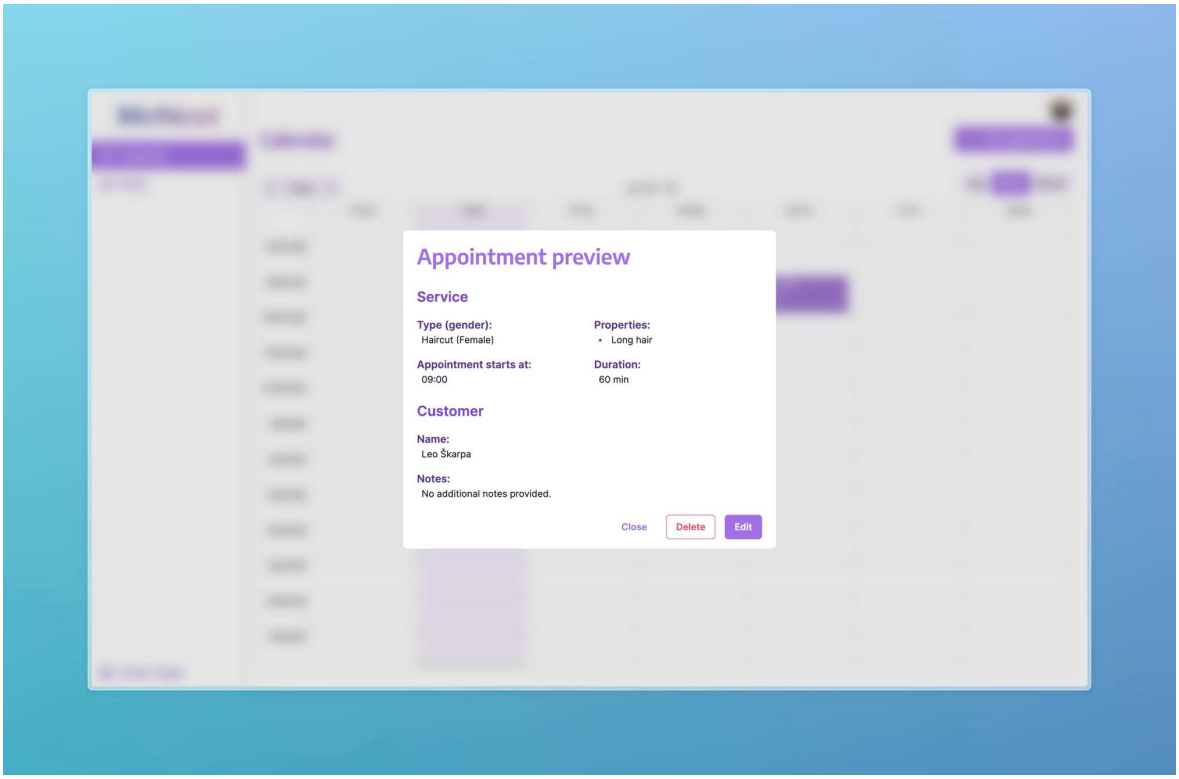
U glavnom prostoru nalazi se kalendar sa napravljenim rezervacijama. Kalendar ima opciju tri prikaza: dnevnog, tjednog i mjesečnog, te pomicanja sa navigacijskim tipkama. Sve rezervacije su vidljive u kalendaru te se nova može kreirati klikom na gumb iznad kalendara.



Sl. 3.17 Kalendar salona

Ukoliko se želi pregledati prijašnja ili buduća rezervacija, moguće je uz dupli klik na rezervaciju unutar kalendara. Tada se otvara novi modal sa svim potrebnim informacijama o rezervaciji: vrsta usluge sa svim svojim dodacima, vrijeme početka rezervacije, trajanje usluge, te informacije o korisniku (Customer) poput imena i dodatnih bilješki.

Ukoliko je tu rezervaciju napravio vlasnik salona, na modalu se prikazuje gumb za uređivanje rezervacije. Inače se prikazuju gumbi za zatvaranje modal-a, te gumb za brisanje rezervacije, ukoliko rezervacija još nije prošla.



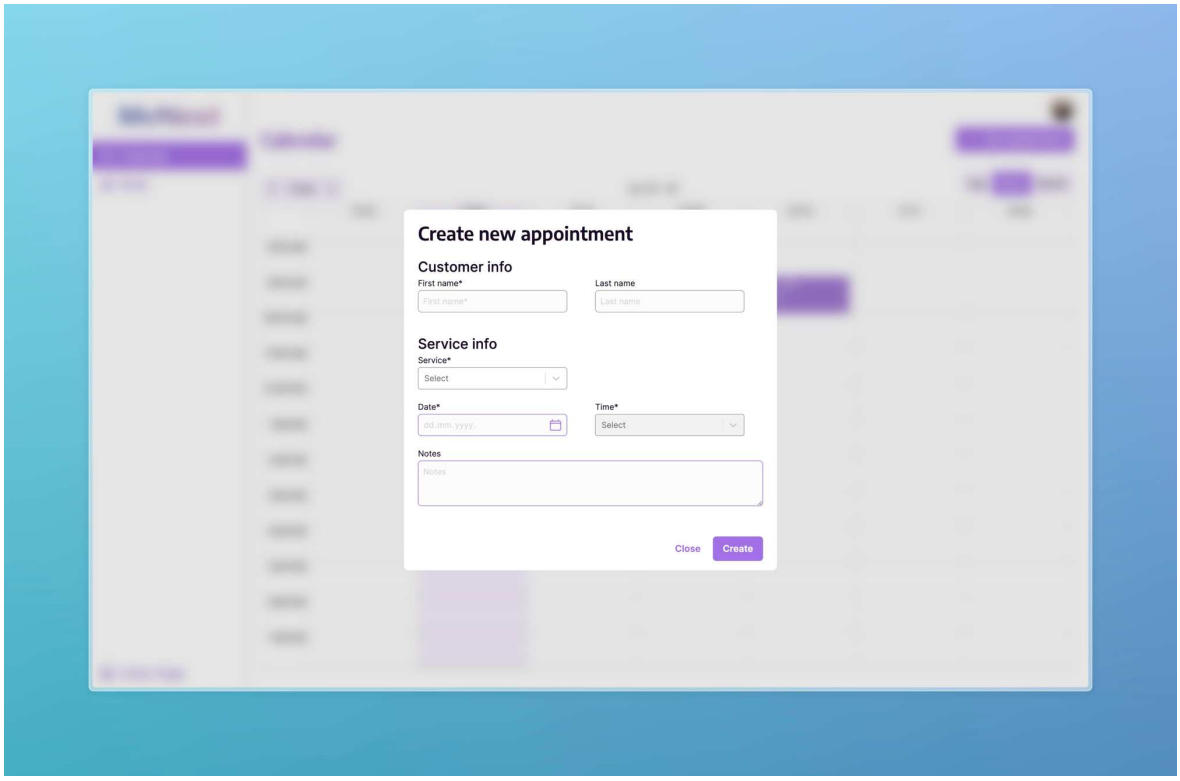
Sl. 3.18 Modal sa informacijama o rezervaciji

Ukoliko vlasnik želi kreirati novu rezervaciju, može to učiniti klikom na gumb iznad kalendara. Tada se otvara modal sa formom za kreaciju rezervacije.

U tu formu vlasnik mora upisati općenite informacije o korisniku poput imena i prezimena, te koju uslugu želi rezervirati tom korisniku. Ovdje mu se ponuđuju sve usluge prije kreirane, zajedno sa svim svojim dodacima, cijenom i trajanjem.

Konačno, vlasnik mora odabrati datum i vrijeme rezervacije. Ovdje se koristi WorkTemplate entitet koji je povezan sa salonom kako bi se odredili slobodni termini. Vremena su limitirana na svakih petnaest minuta, te se ne mogu odabrati ona koja su u bilokakvom presjeku sa već prije napravljenim rezervacijama. Ta limitacija na svakih petnaest minuta je napravljena radi lakšeg odabira termina, te se lagano može povećati ili smanjiti.

Na kraju vlasnik može upisati bilješke vezane za ovu rezervaciju ili korisnika kako bi detaljnije opisao rezervaciju.



Sl. 3.19 Modal za kreaciju nove rezervacije

Nakon kreacije, modal se zatvara te se u kalendaru prikazuje novi termin zajedno sa svim već prije napravljenim.

3.6. Statistika

Do statistike vlasnik može doci klikom na link „Stats“ u sidebar-u. Tamo se nalaze dvije statistike određene sa dva polja za unos datuma. Gornja statistika prikazuje broj rezervacija između odabranih datuma, a donja prikazuje zaradu po danu.

Sl. 3.20 Stranica sa statistikom salona

Statistika je nešto što ORM-ovi poput TypeORM-a ne mogu sami napraviti, te sam stoga morao napisati čisti SQL, i predati ga TypeORM-u da ga izvrši.

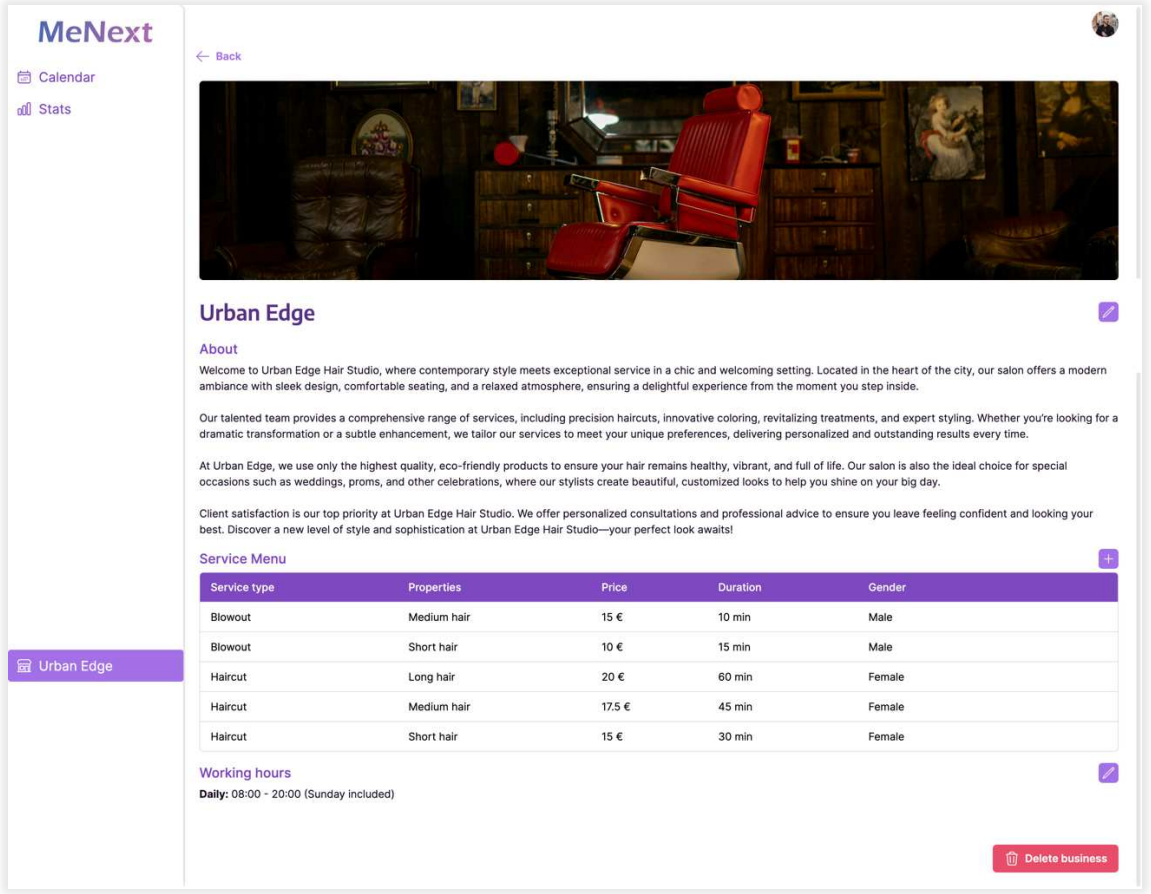
```
getNumOfAppointmentsStats(businessId: number, fromDate: Date, toDate: Date) {
    const manager = this.dataSource.manager
    return manager.query(
      `
      select date.d as date, count(app.id) as value
      from (select d as d
            from generate_series($2::timestamptz, $3::timestamptz,
                                 '1 day'::interval) as d) as date
              left outer join appointment app on app.reserved_from::date between date.d and (date.d + '1 day'::interval) and
                                                 app.business_id = $1 and
                                                 app.deleted_at is null
      group by date.d
      order by date;
    `,
      [businessId, fromDate, toDate],
    )
  }
```

Sl. 3.21 Kôd za statistiku broja rezervacija

3.7. Stranica salona

Zadnji link na sidebar-u vodi na stranicu salona. Na njoj vlasnik može pregledati sve informacije prije napravljene o salonu, od općenitih informacija i usluga, do radnog vremena.

Također na ovoj stranici vlasnik može urediti sve informacije koje želi klikom na gumb pored sekcije koju želi urediti.

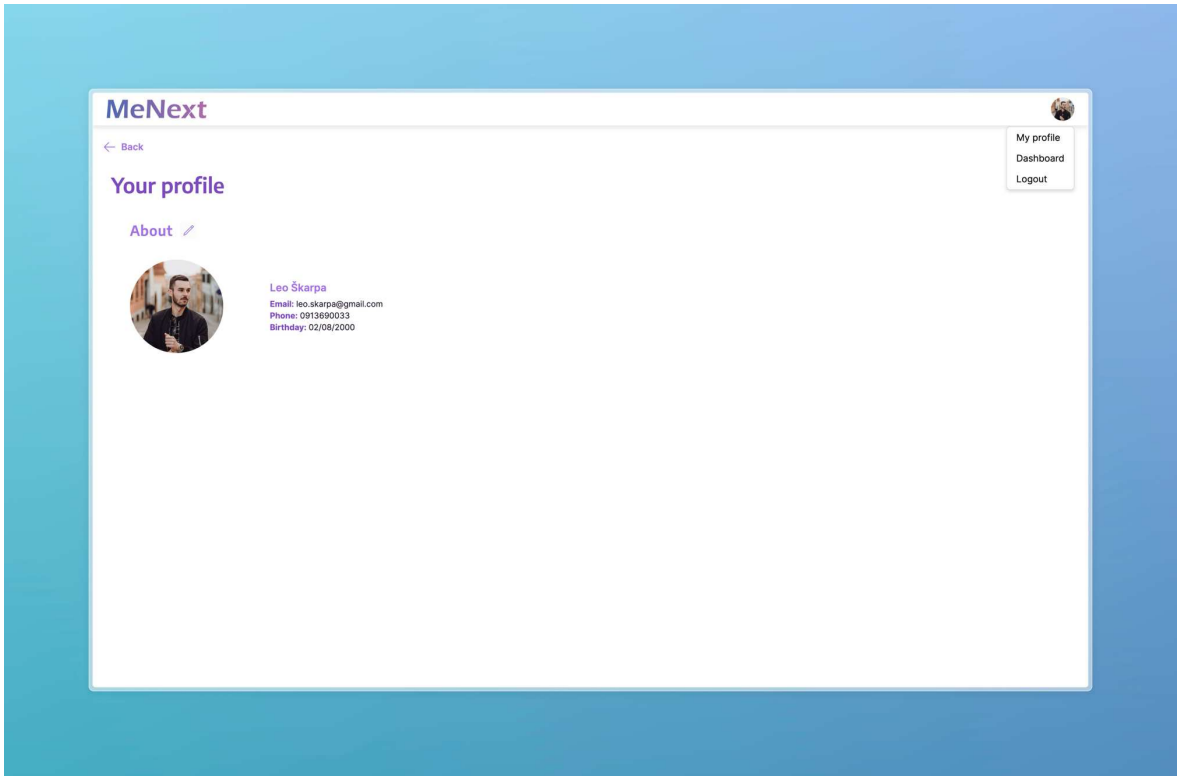


Sl. 3.22 Stranica salona

3.8. Stranica profila

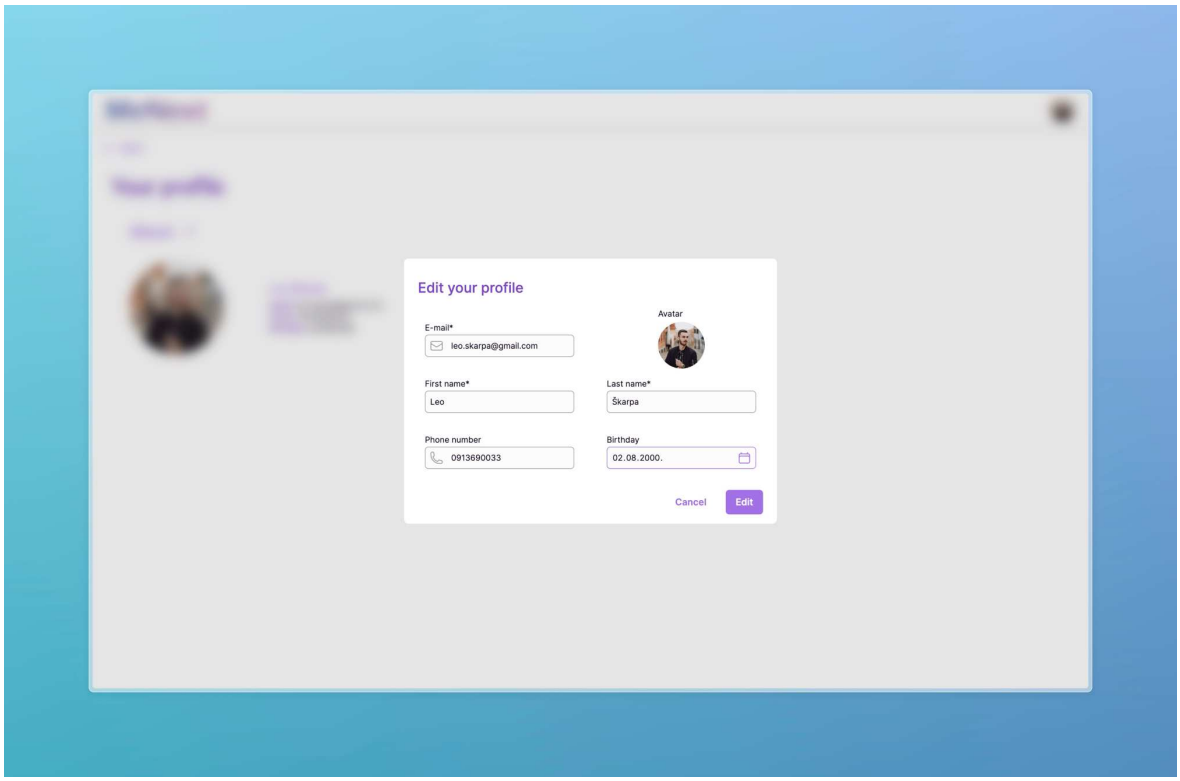
Do stranice profila dolazi se klikom na sliku profila u zaglavlju stranice, te odabirom „My profile“ linka unutar *dropdown*-a.

Stranica je vrlo jednostavna, sastoji se od par informacija o vlasniku te povećane slike profila. Klikom na ikonu olovke pored naslova vlasnik može otvoriti modal za uređivanje informacija svojeg profila.



Sl. 3.23 Stranica profila sa otvorenim dropdown-om ispod slike profila

Modal za uređivanje profila sastoji se od forme sa višestrukim poljima za unos teksta te jednog polja za unos slike profila.



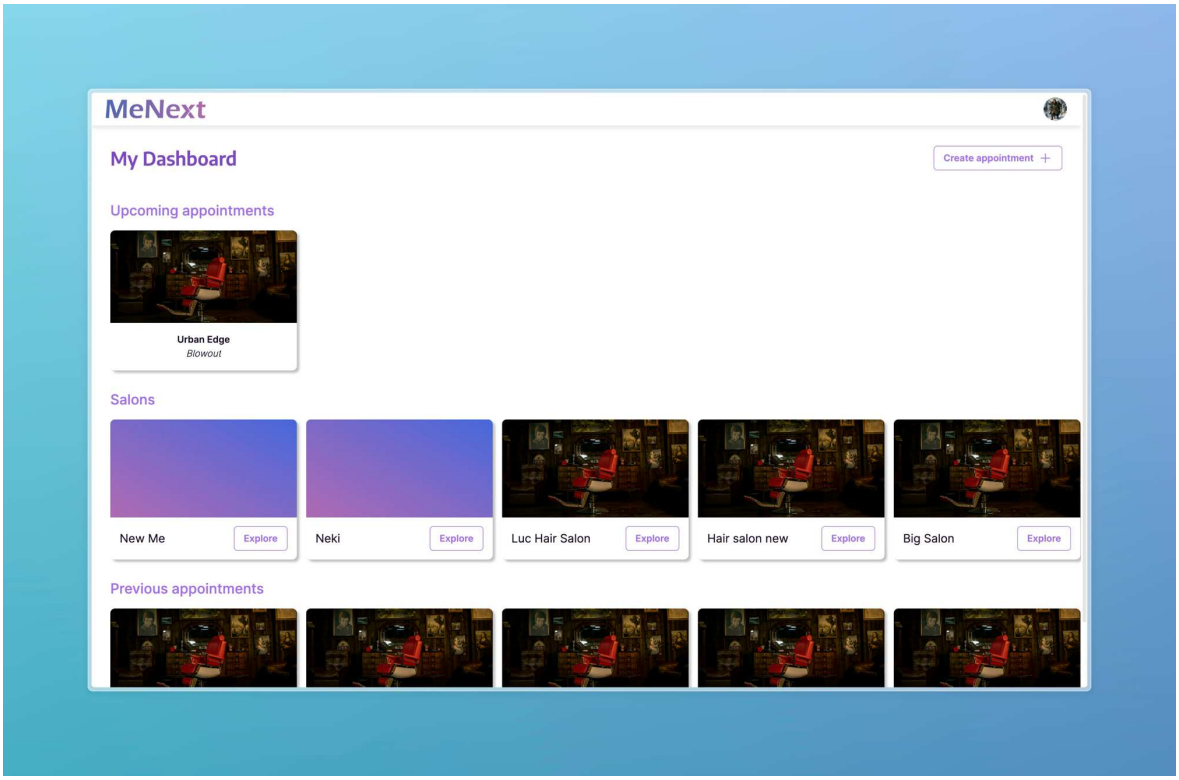
Sl. 3.24 Modal za uređivanje profila

Preostala dva linka u *dropdown*-u ispod slike profila vode na dashboard stranicu te na login stranicu nakon što se korisnik odjavi.

3.9. Dashboard korisnika

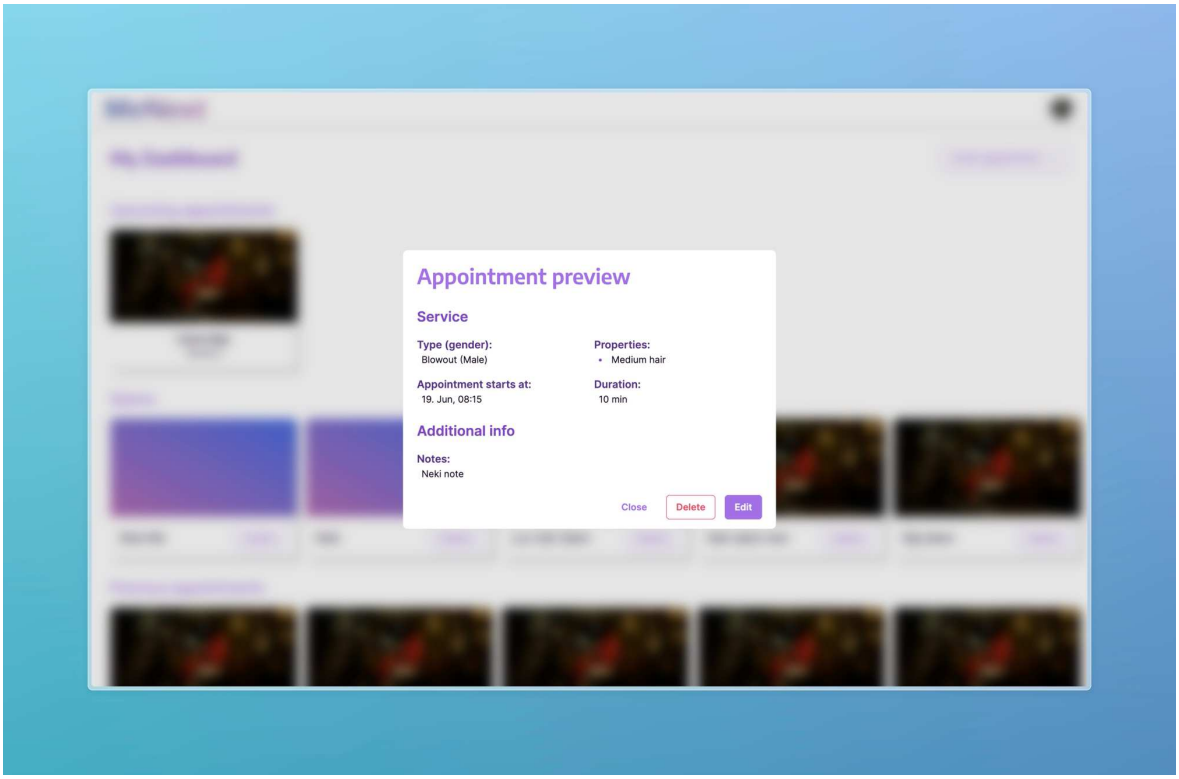
Nakon prijave, korisnik je preusmjeren na njegovu verziju *dashboard*-a. Na njemu se nalaze tri sekcije kartica.

Prva sekcija su buduće rezervacije. Klikom na jednu od njih otvara se modal sa dodatnim informacijama o rezervaciji.



Sl. 3.25 Dashboard stranica korisnika

U modalu o rezervaciji se nalaze informacije o odabranoj usluzi, te dodatne bilješke. Ukoliko je rezervacija u budućnosti na dnu se prikazuju gumbi za brisanje rezervacije i uređivanje iste. Klikom na gumb za uređivanje otvara se novi modal sa formom za uređivanje rezervacije.

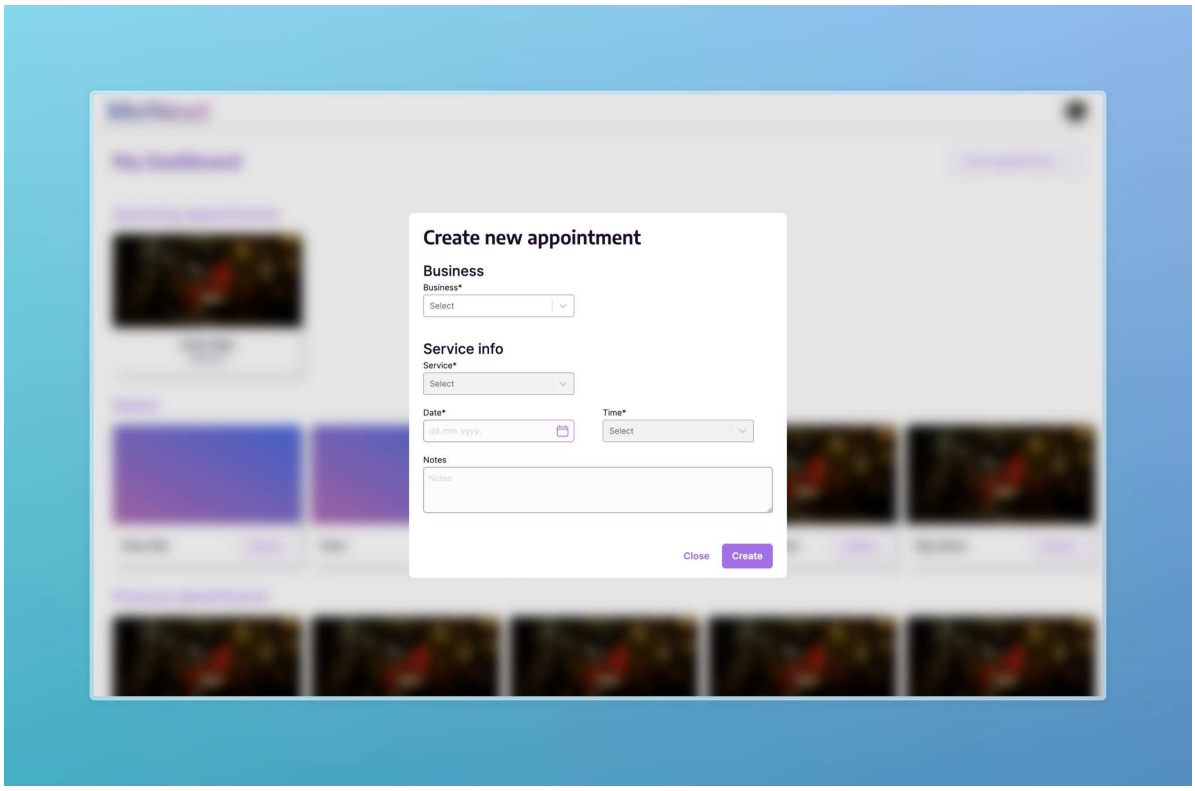


Sl. 3.26 Modal za prikaz informacija buduće rezervacije

Iznad svih sekcija se nalazi gumb koji otvara novi modal za kreiranje nove rezervacije. U njemu korisnik mora prvo odabrati salon za koji želi napraviti rezervaciju, te zatim odabrati datum i slobodno vrijeme. Na kraju može zapisati bilješku koja će se prikazivati i vlasniku salona kada otvori ovu istu rezervaciju u svom kalendaru. Jednom kada korisnik kreira novu rezervaciju, ona će se prikazati kao kartica na dashboard-u.

Druga sekcija su kartice svih dostupnih salona. Klikom na gumb u njima korisnika se vodi na stranicu o salonu, gdje može vidjeti sve potrebne informacije o salonu koje je vlasnik salona napravio.

Zadnja sekcija na stranici su prošle rezervacije. One imaju iste kartice kao i buduće, međutim klikom na njih otvara se modal sličan onom za buduće, ali bez mogućnosti brisanja i uredovanja rezervacije.

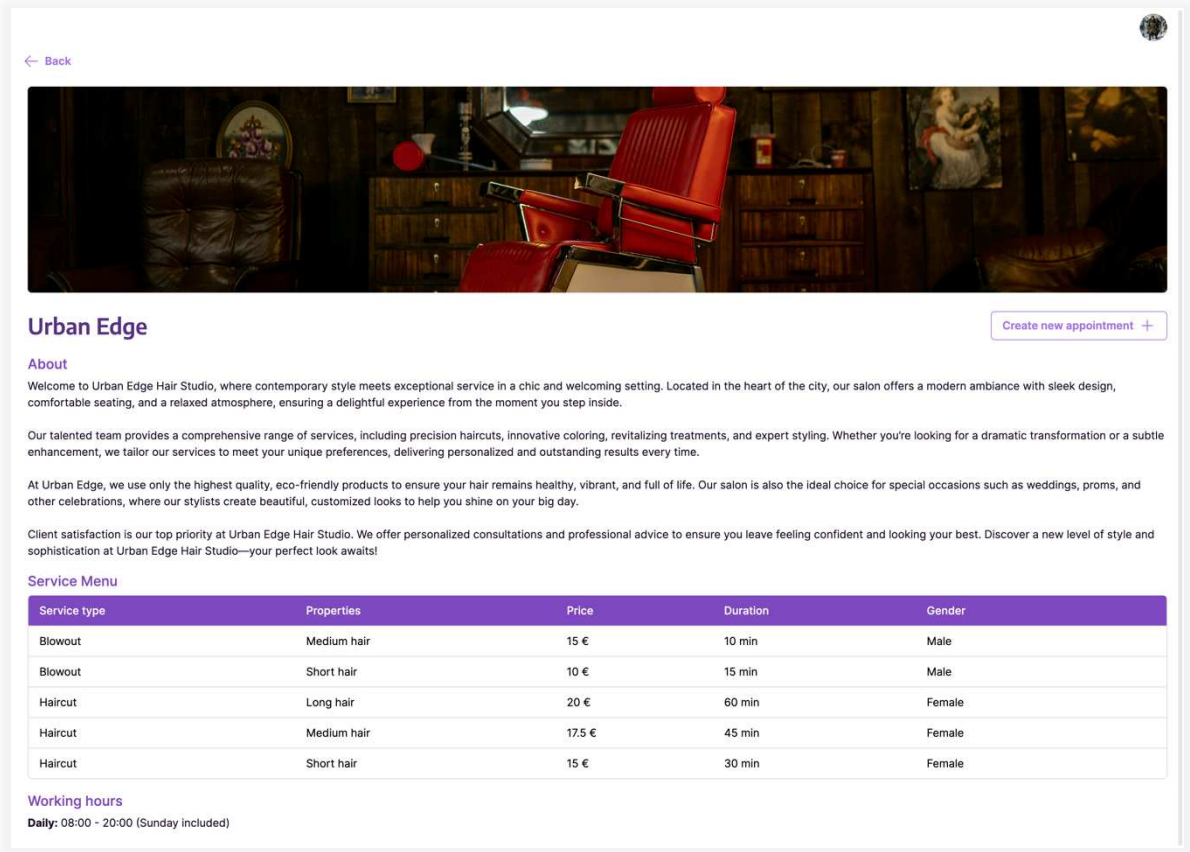


Sl. 3.27 Modal za kreaciju novog termina

3.10. Korisnička stranica salona

Ova stranica otvara se na klik gumba unutar jedne od kartica salona. Stranica je slična kao i stranica salona za vlasnika, ali bez mogućnosti uredovanja salona.

Ispod slike salona nalazi se gumb za kreaciju nove rezervacije. U ovom slučaju se otvara isti modal za kreaciju kao na *dashboard*-u, ali sa već odabranim salonom i bez mogućnosti promjene istog. Ova značajka omogućava brže kreiranje rezervacije direktno sa stranice salona.



Sl. 3.28 Korisnička stranica salona

3.11. Korisnička stranica profila

Korisnička stranica profila je identična vlasnikovoj stranici profila. Sastoji se od jednostavnog prikaza podataka o profilu, te je klikom na ikonu olovke moguće otvoriti modal za uređivanje profila.

Klikom na sliku profila otvara se *dropdown* identičan onom na vlasnikovom dijelu aplikacije te ima iste funkcionalnosti.

Zaključak

Cilj ovog rada bio je napraviti aplikaciju koja bi pomogla vlasnicima i radnicima salona te korisnicima usluga istih salona.

Ova aplikacija služi vlasnicima da naprave svoje salone, dodaju sve potrebne informacije te imaju pregled u cjelokupni sustav rezerviranja termina. Uz pomoć svih kreiranih entiteta, korisnicima je lagano napraviti novu rezervaciju jednostavnim odabirom salona, usluge i termina.

Ideja je bila da se prvenstveno pomogne vlasnicima u vođenju salona, te da se mogu fokusirati na svoj rad, a ne na konstantno javljanje na telefon zbog rezervacija. Uz to, vlasnicima je omogućen prikaz statistike broja rezervacija te prihoda unutar određenog intervala.

U sklopu ovog rada napravljen je i klijentski i poslužiteljski dio, uz bazu, tako da je spremna za *deploy* na produkciju. Ukoliko bi se na aplikaciji nastavilo raditi, postoje određene značajke koje bi se dodale i tako uveliko doprinijele cijeloj aplikaciji i ideji kojoj služi.

Neke od potencijalnih budućih značajki bi bilo dodavanje dijela sustava za radnika te njegovo korisničko sučelje (koje bi se razlikovalo od vlasničkog), omogućavanje dodavanja i brisanja radnika iz salona za vlasnika, dodatne statistike za radnike. Samim tim dodavanjem radnika u aplikaciju, dodao bi se i odabir korisnika kod kojeg radnika želi rezervirati uslugu. Također, korisnički dio bi se trebao povećati, kao dodavanje favorita, ostavljanje recenzija, te bi se *dashboard* stranica trebala redizajnirati da se slaže sa novim značajkama.

Aplikacija MeNext je uspješno napravljena te zadovoljava sve početne korisničke zahtjeve. Uz dodatnu nadogradnju novih značajki te dobar marketing, smatram da bi ova aplikacija mogla biti konkurent na tržištu.

Literatura

[1] React Official Documentation, https://react.dev/, pristupljeno 16. lipnja 2024.

[2] TypeScript Official Documentation, https://www.typescriptlang.org/, pristupljeno 16. lipnja 2024.

[3] Git Official Documentation, https://git-scm.com/doc, pristupljeno 16. lipnja 2024.

[4] Postman Learning Center, https://learning.postman.com/, pristupljeno 16. lipnja 2024.

[5] NestJS Official Documentation, https://docs.nestjs.com/, pristupljeno 16. lipnja 2024.

[6] PostgreSQL Official Documentation, https://www.postgresql.org/docs/current/index.html, pristupljeno 16. lipnja 2024.

[7] GitHub Guides, https://guides.github.com/, pristupljeno 16. lipnja 2024.

[8] Docker Official Documentation, https://docs.docker.com/, pristupljeno 16. lipnja 2024.

[9] TypeORM Official Documentation, https://typeorm.io/, pristupljeno 16. lipnja 2024.

[10] Recoil Official Documentation, https://recoiljs.org/, pristupljeno 16. lipnja 2024.

[11] React Query Official Documentation, https://tanstack.com/query/latest, pristupljeno 17. lipnja 2024.

[12] Axios GitHub Repository, https://axios-http.com/, pristupljeno 17. lipnja 2024.

[13] RESTful API Tutorial, https://restfulapi.net/, pristupljeno 17. lipnja 2024.

[14] Atomic Design System, https://atomicdesign.bradfrost.com/chapter-2/, pristupljeno 18. lipnja 2024.

Sažetak

Naslov:

Rezervacija termina u uslužnim djelatnostima

Sažetak:

Tehnologija se danas proširila u skoro sve djelatnosti. Ova aplikacija, naziva MeNext, se širi u salone. Ideja njezine implementacije je da pomogne vlasnicima lakše manipuliranje rezervacijama, te korisnicima lakše rezerviranje termina.

Aplikacija se sastoji od klijentske strane napisane u ReactJS-u te poslužiteljske strane napisane u NestJS-u. Vlasnici imaju pregled kalendara i statistika svojeg salona, te mogućnost kreacije, uređivanja i brisanja rezervacija direktno sa kalendara. Korisnici imaju olakšanu mogućnost kreacije rezervacija, te pregled svih budućih i prošlih rezervacija i dostupnih salona.

Ključne riječi:

ReactJS, NestJS, rezervacija, salon, kalendar, vlasnik, korisnik

Summary

Title:

Reservation of appointments in service salons

Summary:

Technology has expanded into almost every field today. This application, named MeNext, is expanding into salons. The idea behind its implementation is to help owners manage reservations more easily and to facilitate customers in booking appointments.

The application consists of a client side written in ReactJS and a server side written in NestJS. Owners have an overview of the calendar and statistics of their salon, as well as the ability to create, edit, and delete reservations directly from the calendar. Customers have an easy way to create reservations and can view all future and past reservations and available salons.

Keywords:

ReactJS, NestJS, appointment, business, calendar, owner, customer