IZRADA APLIKACIJE ZA REZERVACIJU USLUGE KOD FRIZERA

Domjanović, Ante

Master's thesis / Specijalistički diplomski stručni

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Split / Sveučilište u Splitu**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:228:572073

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-29**

Repository / Repozitorij:

Repository of University Department of Professional Studies

zir.nsk.hr

UNIVERSITY OF SPLIT

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Specijalistički diplomski stručni studij Informacijske tehnologije

ANTE DOMJANOVIĆ

ZAVRŠNI RAD

IZRADA APLIKACIJE ZA REZERVACIJU USLUGE KOD FRIZERA

Split, srpanj 2020.

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Specijalistički diplomski stručni studij Informacijske tehnologije

Predmet: Napredna uporaba računala

ZAVRŠNI RAD

Kandidat: Ante Domjanović

Naslov rada: Izrada aplikacije za rezervaciju usluge kod frizera

Mentor: mr. sc. Ivica Ružić, viši predavač

Split, srpanj 2020.

SADRŽAJ

SAŽETAK **5**

SUMMARY **6**

1. UVOD **7**

2. BAZA PODATAKA **8**

2.1. Struktura baze podataka praktičnog dijela 9

2.2. MySQL 15

2.3. Doctrine 16

3. TEHNOLOGIJE **18**

3.1. Razvojna okolina 18

3.1.1. Vagrant 18

3.1.2. PuPHPet 20

3.1.3. VirtualBox 21

3.2. PHP & Symfony 22

3.2.1. PHP – PHP:Hypertext Preprocessor 22

3.2.2. Symfony 4 22

3.3. React 25

3.3.1. Koncept React-a 25

3.3.2. React Hooks 29

3.3.3. React Context 30

3.4. Korištene biblioteke, paketi i alati 32

3.4.1. Webpack Encore 32

3.4.2. jQuery 35

3.4.3. Easyadmin bundle 35

3.4.4. Datatables 36

3.4.5. Composer 36

3.4.6. Open react template 36

3.4.7. Admin LTE 36

3.5. Ostali alati 37

3.5.1. Git 37
3.5.2. MySQL Workbench 38
3.5.3. PHPStorm 38
4. OPIS PRAKTIČNOG RADA 39
4.1. Ciljevi aplikacije 39
4.2. Razrada sigurnosne komponete ovisno o razini prava korisnika 40
4.2.1. Uloge korisnika i autentikacija 40
4.2.2. Autorizacija 41
4.3. Upravljanje administracijom 47
4.3.1. Popunjavanje informacija o salonu 47
4.3.2. Definiranje radnih dana i smjena 47
4.3.3. Definiranje usluge 49
4.3.4. Dodavanje zaposlenika 50
4.3.5. Upravljanje kalendarom radnog vremena 51
4.3.6. Upravljanje rezervacijama 55
4.4. Jednostavna rezervacija termina putem klijentske strane 56
5. ZAKLJUČAK 59
6. LITERATURA 60

SAŽETAK

Cilj ovoga završnog rada je izrada web aplikacije koja će omogućiti pružanje administrativne usluge upravljanja rada frizerskog salona vlasnicima, te povezivanja samih salona s klijentima u svrhu rezervacije termina šišanja.

Glavne značajke aplikacije bi bile kompletno administrativno sučelje koje će omogućiti partnerima jednostavno upravljanje radom svog salona, od definiranja usluga i cijena, zaposlenika, praćenja prometa rezervacija i kreiranje novih. Pritom je važno naglasiti da se radi o višekorisničkom adminstrativnom sučelju, što znači da je namijenjeno korištenju više partnera a da pritom oni imaju pristup samo svojim resursima kao što su salon, zaposlenici, rezervacije itd. Korisnici klijenti koji traže termin tj. uslugu friziranja će dobit najjednostavniji način rezerviranja termina u par klikova.

Dijelovi se sastoje od uvoda u kojem je definirana problematika završnog rada, opis korištenih tehnologija, opis praktičnog problema s popratnim dijelovima implementacije te zaključka sa sažetim pregledom i završnom riječi na konačnu aplikaciju.

Ključne riječi: Symfony, React, usluga friziranja, administracijsko sučelje, web aplikacija

SUMMARY

Title: Development of application for booking a service at a hairdresser

The aim of this final work is to create a web application that will provide administrative services for managing the work of the hair salon to the owners and connecting them with clients for the purpose of booking a haircut.

The main features of the application would be a complete administrative interface that will allow the partners to easily manage the work of their salon, from defining services and prices, employees, tracking reservations and creating new ones. It is important to emphasize that this is a multi-user administrative interface, which means that it is intended for the use of multiple partners and that they have access only to their resources such as salon, employees, reservations, etc. Users who are looking for an appointment, i.e. a hairdressing service, will get the simplest way to book an appointment in a few clicks.

Parts of the paper consist of an introduction in which the issue of the final paper is defined, a description of technologies used, a description of practical problems with the accompanying parts of the implementation and a conclusion with a concise overview with final words on the developed application.

Keywords: Symfony, React, hairdressing service, administrative interface, web application

1. UVOD

U današnje vrijeme od svakog obrta se očekuje da ima vlastitu web stranicu. Kod pružatelja usluge friziranja, to još uvijek nije tako, većina njih se služi korištenjem društvenih mreža i oglašavanjem na postojećim da se stvori kontakt s klijentima. To nikako ne može biti trajno rješenje. Samim time otežano je vođenje rada salona sa zaposlenicima, rasporeda radnih dana i izvršavanja rezervacija ako ne postoji mjesto gdje je omogućen pregled navedenih stavki i njihovo upravljanje. Također sami proces rezervacije se izvodi u nekoliko koraka, najčešće uključuje ili osobni dolazak do salona par dana unaprijed, okretanje telefonskog broja i slične načine komunikacije.

Ovaj rad će se temeljiti na izradi web aplikacije koja pruža administrativne usluge upravljanja salona za vlasnike salona i njihove zaposlenike, te pružanja usluge rezervacije termina klijentima putem klijentske aplikacije.

Administrativno sučelje će omogućiti partnerima kompletno upravljanje radom svog salona. Čime se rasterećuje rad njih i samih zaposlenika, što omogućuje potpuni fokus na samog klijenta i pružanja usluge. Korisnici klijenti koji traže termin tj. uslugu friziranja moći će zamijeniti zastarjeli način povezivanja i to obaviti u svega par klikova. Time se automatski pruža ugodno korisničko iskustvo koje bi eliminiralo uvjetne korake koje treba napraviti da se dogovori termin. Cilj ovog rada je napraviti intuitivnu aplikaciju internetske rezervacije termina za šišanje u par klikova i olakšati rad vlasnicima (i radnicima) frizerskih salona.

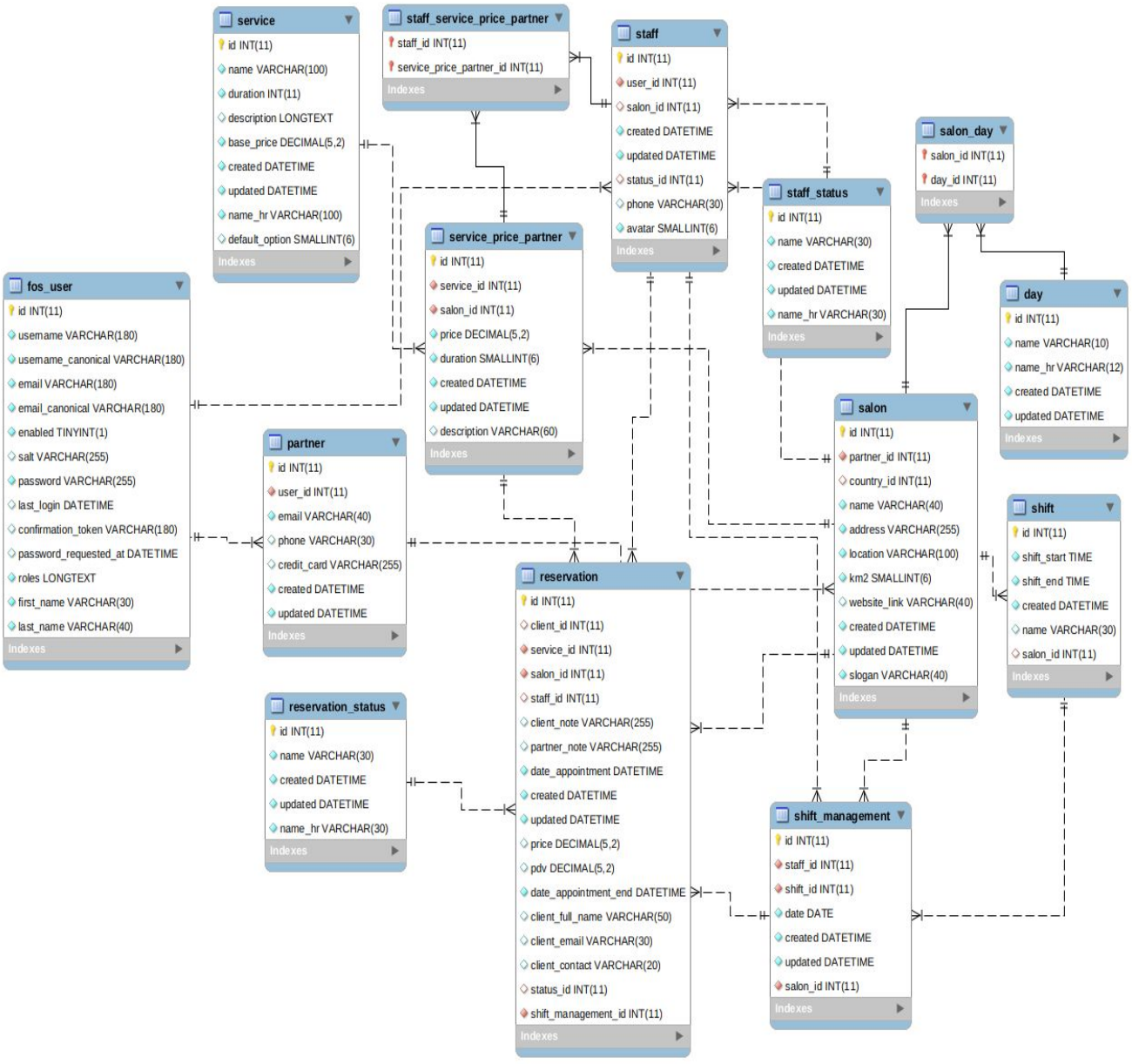
U sljedećem poglavlju će biti prikazana struktura baze podataka i njenih tablica, te opisani najvažniji koncepti. Zatim u nastavnim poglavljima sve korištene tehnologije za izradu aplikacije kao što su Vagrant za razvojnu okolinu, Symfony predložak za izradu serverske strane aplikacije, te React za klijentsku stranu. U praktičnom dijelu će biti opisane najvažnije funkcionalnosti aplikacije uz popratne primjere koda. Zaključak sadrži završnu riječ kao osvrt na kompletni rad.

2. BAZA PODATAKA

Skup podataka pripremljen tako da se može jednostavno koristiti i raditi razne operacije nad njima bez da se naruši njihov integritet kroz relacijski odnos. Relacijske baze podataka počivaju na transakcijama koju imaju sljedeća svojstva:

- Atomnost
    - transakcija se izvodi u komadu i sadržajna je
- Dosljednost
    - transakcija po završetku mijenja stanje baze iz jednog u drugo ispravno stanje
- Izolacija
    - transakcije su međusobno neovisne
- Postojanost
    - transakcija koja je uspješno završena je trajna

2.1. Struktura baze podataka praktičnog dijela



Slika 1. E-R dijagram praktičnog rada

Tablica 1. Struktura tablice *reservation*

Naziv polja	**Tip**	**Primarni ključ**	**Strani ključ**	***Null* vrijednost**
id	INT(11)	DA		NE
client_id	INT(11)		client.id	DA
service_id	INT(11)		service_price_partner.id	NE
salon_id	INT(11)		salon.id	NE
staff_id	INT(11)		staff.id	DA
client_note	VARCHAR(255)			DA
partner_note	VARCHAR(255)			DA
created	DATETIME			NE
updated	DATETIME			NE
price	DECIMAL(5,2)			DA
pdv_decimal	DECIMAL(5,2)			DA
date_appointment	DATETIME			NE
date_appointment_end	DATETIME			NE
client_full_name	VARCHAR(50)			DA
client_email	VARCHAR(30)			DA
client_contact	VARCHAR(20)			DA
status_id	INT(11)		status.id	DA
shift_management_id	INT(11)		shift_management.id	NE

Tablica 2. Struktura tablice *salon_day*

Naziv polja	**Tip**	**Primarni ključ**	**Strani ključ**	***Null* vrijednost**
salon_id	INT(11)	DA	salon.id	NE
day_id	INT(11)	DA	user.id	NE

Tablica 3. Struktura tablice *service_price_partner*

Naziv polja	Tip	Primarni ključ	Strani ključ	*Null* vrijednost
id	INT(11)	DA		NE
service_id	INT(11)		service.id	NE
salon_id	INT(11)		salon.id	NE
price	DECIMAL(5,2)			NE
duration	SMALLINT(6)			NE
created	DATETIME			NE
updated	DATETIME			NE
description	VARCHAR(60)			DA

Tablica 4. Struktura tablice *shift_management*

Naziv polja	Tip	Primarni ključ	Strani ključ	*Null* vrijednost
id	INT(11)	DA		NE
staff_id	INT(11)		staff.id	NE
shift_id	INT(11)		shift.id	NE
salon_id	INT(11)		salon.id	NE
date	DATE			NE
created	DATETIME			NE
updated	DATETIME			NE

Tablica 5. Struktura tablice *reservation_status*

Naziv polja	Tip	Primarni ključ	Strani ključ	*Null* vrijednost
id	INT(11)	DA		NE
name	VARCHAR(30)			NE
name_hr	VARCHAR(30)			NE
created	DATETIME			NE
updated	DATETIME			NE

Tablica 6. Struktura tablice *salon*

Naziv polja	**Tip**	**Primarni ključ**	**Strani ključ**	***Null* vrijednost**
id	INT(11)	DA		NE
partner_id	INT(11)		partner.id	NE
country_id	INT(11)		country.id	DA
name	VARCHAR(40)			NE
address	VARCHAR(255)			NE
location	VARCHAR(100)			NE
km2	SMALLINT(6)			NE
website_link	VARCHAR(40)			DA
slogan	VARCHAR(40)			NE
created	DATETIME			NE
updated	DATETIME			NE

Tablica 7. Struktura tablice *partner*

Naziv polja	**Tip**	**Primarni ključ**	**Strani ključ**	***Null* vrijednost**
id	INT(11)	DA		NE
user_id	INT(11)		user.id	NE
email	VARCHAR(40)			NE
phone	VARCHAR(30)			DA
credit_card	VARCHAR(255)			DA
created	DATETIME			NE
updated	DATETIME			NE

Tablica 8. Struktura tablice *staff_service_price_partner*

Naziv polja	**Tip**	**Primarni ključ**	**Strani ključ**	***Null* vrijednost**
staff_id	INT(11)	DA	staff.id	NE
service_price_partner_id	INT(11)	DA	service_price_partner.id	NE

Tablica 9. Struktura tablice *day*

Naziv polja	Tip	Primarni ključ	Strani ključ	*Null* vrijednost
id	INT(11)	DA		NE
name	VARCHAR(10)			NE
name_hr	VARCHAR(12)			NE
created	DATETIME			NE
updated	DATETIME			NE

Tablica 10. Struktura tablice *shift*

Naziv polja	Tip	Primarni ključ	Strani ključ	*Null* vrijednost
id	INT(11)	DA		NE
shift_start	TIME			NE
shift_end	TIME)			NE
salon_id	INT(11)		salon.id	DA
name	VARCHAR(30)			DA
created	DATETIME			NE

Tablica 11. Struktura tablice *staff_status*

Naziv polja	Tip	Primarni ključ	Strani ključ	*Null* vrijednost
id	INT(11)	DA		NE
name	VARCHAR(30)			NE
name_hr	VARCHAR(30)			NE
created	DATETIME			NE
updated	DATETIME			NE

Tablica 12. Struktura tablice *fos_user*

Naziv polja	Tip	Primarni ključ	Strani ključ	*Null* vrijednost
id	INT(11)	DA		NE
username	VARCHAR(180)			NE
username_canonical	VARCHAR(180)			NE
email	VARCHAR(180)			NE
email_canonical	VARCHAR(180)			NE
enabled	TINYINT(1)			NE
salt	VARCHAR(255)			DA
password	VARCHAR(255)			NE
last_login	DATETIME			DA
confirmation_token	VARCHAR(180)			DA
password_requested	DATETIME			DA
roles	LONGTEXT			NE
first_name	VARCHAR(30)			NE
last_name	VARCHAR(40)			NE

Tablica 13. Struktura tablice *staff*

Naziv polja	Tip	Primarni ključ	Strani ključ	*Null* vrijednost
id	INT(11)	DA		NE
user_id	INT(11)		user.id	NE
salon_id	INT(11)		salon.id	DA
status_id	INT(11)		status.id	DA
phone	VARCHAR(30)			DA
avatar	SMALLINT(6)			NE
created	DATETIME			NE
updated	DATETIME			NE

Tablica 14. Struktura tablice *service*

Naziv polja	Tip	Primarni ključ	Strani ključ	*Null* vrijednost
id	INT(11)	DA		NE
name	VARCHAR(100)			NE
name_hr	VARCHAR(40)			NE
duration	INT(11)			NE
default_option	SMALLINT(6)			DA
description	LONGTEXT			DA
base_price	DECIMAL(5,2)			NE
created	DATETIME			NE
updated	DATETIME			NE

2.2. MySQL

Kod relacijskih baza podataka dosljednost je osigurana. Jednom kada je transakcija potvrđena svi naknadni zahtjevi će biti obaviješteni o trenutnoj transakciji. Relacijske baze podataka zasnivaju se na strogim principima pa im je prednost: stabilnost, pouzdanost, otpornost na greške, brzina izvođenja, strukturirani jezik za rad s podacima SQL (eng. *structured query language)*.

Jedan od sustava za upravljanje relacijskim bazama podataka otvorenog koda koji se pokreće na serverskoj strani je MySQL.

MySQL je relacijska baza podataka otvorenog izvornog koda ili skraćeno RDBMS (eng. *relational database management system)*. RDBMS sistemi podatke pohranjuju u tablice koje se sastoje od kolona i redaka. Entitet je element koji se pohranjuje u bazu. Relacije su veze tj. odnosi među entitetima čime se stvara integritet. Primarni ključevi jedinstveno identificiraju podatak u određenoj tablici. Korištena verzija je 5.7.

2.3. Doctrine

Doctrine (*The Doctrine Project*) je set PHP biblioteka primarno fokusiranih na pružanju apstrakcije s radom na bazi podataka. Razina apstrakcije proizlazi kroz objektno relacijsko mapiranje (eng. *ORM object relation mapper*) [8].

Jedan od najčešćih i najizazovnijih zadataka bilo koje aplikacije uključuje rad s podatcima iz baze, čitanje, brisanje, uređivanje podataka u i iz baze. Iako Symfony predložak ne integrira nijednu komponentu za rad s bazama podataka, pruža integraciju s Doctrin-om. Prednost Doctrine za programera je sposobnost usredotočenja na objektno orijentiranu poslovnu logiku kroz rad s entitetima koji su objektno strukturirane tablice baze. Ujedno, jedna od Doctrine značajki je opcija pisanja upita korištenjem DQL (eng. *Doctrine Query Langauge*) objektno orijentirane verzije SQL-a.

Doctrine/ORM koncepti

Entiteti

Entiteti su PHP objekti identificirani s unikatnom vrijednošću ili primarnim ključem. Klase koje ne trebaju nasljeđivati neku apstraktnu ili baznu klasu. Veza s bazom proizlazi kroz *EntityManager* koji je Doctrin-ovo javno sučelje za upravljanje radom baze podataka i poslovne logike aplikacije. On je zadužen za osnovne CRUD (eng. *create-read-update-delete*) operacije, transformaciju podatka s entiteta u tabličnu strukturu.

Bidirekcionalna i unidirekcionalna relacija

Recimo da imamo jedan prema više relaciju: *"klijent ima rezervacije"***,** možemo dodati entitetu Klijent "*jedan naprema više*" svojstvo. Također, tu postoji i "više naprema jedan" relacija "rezervacije imaju klijenta"**,** koje je opcionalno za postaviti na enitetu Rezervacije, to svojstvo će tada relaciju učinit bidirekcionalnom i možemo pristupati podatcima s Klijent objektom preko Rezervacija objekta. U oba slučaja (bila uni. ili bi. relacija) **ORM će održati istu strukturu baze.**

Pravila relacije:

- relacija je bidirekcionalna ako imamo da oba entiteta sadrže referencu na jedno drugoga tj. ako sadrže *owning* i *inverse* stranu
- *owning* strana relacije je ona na kojoj se spremaju promjene na bazi
- ako postavimo referencu na samo jednom entitetu tada je unidirekcionalna relacija
- *inverse* strana bidirekcionalne veze treba se referencirati na *owning* stranu korištenjem *mappedBy* atributa jedne od deklaracija:
    - *OneToOne*
    - *OneToMany*
    - *ManyToMany*
- *owning* i *inverse* strana relacije
    - owning strana je ona koja u sebi sadrži ID druge strane

Doctrine omogućuje reprezentaciju tablica kao entiteta u obliku PHP objekta, samim time su i relacije (eng. *associations*) reference na druge objekte. Sve to olakšava testiranje domenske logike bez da se brine o implementaciji baze, u Doctrinu sve operacije su obuhvaćene u SQL transakcije, što garantira atomičnost [8].

3. TEHNOLOGIJE

3.1. Razvojna okolina

3.1.1. Vagrant

Vagrant je program otvorenog koda za kreiranje i upravljanje prenosivih virtualnih razvojnih softverskih okolina. Napisan je u Ruby programskom jeziku. Inicijalno prva radna stabilna verzija je objavljena u ožujku 2010. godine.

Korištenjem virtualne mašine doprinosi jednostavnijem održavanju, proširenju aplikacije i pritom je uvijek sigurno da će se jednako ponašati na svakom operativnom sustavu. U usporedbi sa Docker-om koji kao upravitelj spremnika je alternativa Vagrant-u, više je orijentiran za izrade mikroservisa [1]. Sa sljedeće tablice je vidljiva usporedba Vagrant-a i Docker-a s obzirom na virtualizaciju:

Tablica 15. Usporedba Vagrant i Docker-a

	Vagrant	**Docker**
Virtualizacija	Virtualna mašina	Linux
Vrsta kernela	Zasebni kernel za svako okruženje	Dijeljeni kernel
Vrijeme pokretanja	Minute	Sekunde
Kreiranje okruženja	10+ minuta	Minute
Veličina	1GB+	100MB+

Jednom kada se postavi virtualni server i pokrene skripta (*provisioning*) na svakoj od mašina, ona će instalirati potrebne programe, pakete, postaviti dozvole, konfigurirati i instalirati sve što je potrebno aplikaciji [2]. Nedostatak Vagranta je što treba pokrenuti cijelu virtualnu mašinu za pokrenuti aplikaciju, zahtijeva dosta memorije i vremena da se sve podesi.

Docker ima drugačiji način rada, koristi se Docker datoteka (30 linija koda) za kreiranje Docker slike. Ona sadrži cijeli projekt (kod), sve potrebne instalacije i konfiguracije. Ne treba nova virtualna mašina za svaki spremnik, samo jedna jer su dizajnirane da rade na vrhu mašine. Tako da se može pokrenuti broj Docker spremnika onoliko koliko ima dozvoljene memorije i snage.

Osnovne naredbe za rad su:

- vagrant init
    - za inicijalizaciju *Vagrantfile* datoteke i direktorija gdje će se nalaziti virtualna mašina, ako se koristi već pripremljena ili generirana okolina ovaj poziv nije potreban
- vagrant up
    - pokreće Vagrant okolinu s definiranom *Vagrantfile* datotekom
- vagrant halt
    - zaustavlja rad virtualne mašine
- vagrant reload
    - radi ponovno pokretanje virtualne mašine sa svim novim izmjenama definiranim u konfiguracijskoj datoteci
- vagrant ssh
    - pozicioniranje unutar mašine

Vagrant radi na principu opskrbitelja (eng. *provisioner*) i pružatelja (eng. *provider*). Opskrbitelji su alati koji služe za konfiguraciju razvojnih okolina. Najučestaliji alati opskrbitelji su Puppet i Chef. Za konfiguraciju virtualne mašine na kojoj je rađen praktični dio poslužio je PuPHPet.

Pružatelji su servisi koje Vagrant koristi za pripremu i kreaciju virtualnih okolina, s prvom verzijom radio je samo s VirtualBox-om a poslije je dobio podršku i za ostale popularne servise kao što su Hyper-V i Docker, dok VMware i Amazon AWS su podržani putem programskih dodataka (eng. *plugins*). Korištena verzija u praktičnom radu je Vagrant 2.0.2 sa Centos 7 operativnim sustavom.

3.1.2. PuPHPet

PuPHPet je web orijentirano grafičko sučelje otvorenog koda koje pomaže za jednostavno kreiranje Vagrant virtualne mašine s učestalim opcijama koje su potrebne za izradu projekta [3]. Postoje razne konfiguracijske opcije od definiranja količine radne memorije za korištenje, definiranje različitih domena ili proširivanja s paketima poput Xdebug alata za lakše praćenje izvršavanja dijelova koda. Isječak iz PuPHPet generirane datoteke za kreiranje Vagrant mašine je vidljiv na sljedećoj slici:

```
vagrantfile:
    target: local
    vm:
        provider:
            local:
                box: bento/centos-7
                box_url: 'false'
                box_version: '0'
                chosen_virtualizer: virtualbox
                virtualizers:
                    virtualbox:
                        modifyvm:
                            natdnshostresolver1: false
                        showgui: 0
                    vmware:
                        numvcpus: 1
                    parallels:
                        linked_clone: 0
                        check_guest_tools: 0
                        update_guest_tools: 0
                machines:
                    machine1:
                        id: osisaj-me.dev.machine
                        hostname: osisajme.master
                        network:
                            private_network: 192.168.56.102
                            forwarded_port:
                                port1:
                                    host: '8396'
                                    guest: '22'
                        memory: '768'
                        cpus: '2'
```

Slika 2. PuPHPet konfiguracijska skripta

3.1.3. VirtualBox

Oracle VM VirutalBox je program koji omogućuje pokretanje drugih operativnih sustava na računalu. Mogu se virtualizirati razne verzije sustava Windowsa, Linux-a, Mac OS X-a. Značajke ovoga programa su mogućnost pokretanja više operativnih sustava istodobno, daljinsko povezivanje, besplatan, dijeljenje mapa, podrška za različite mreže i drugo [4]. Korištena verzija u radu 5.2.8.



Slika 3. VirtualBox + Vagrant = CentOS

3.2. PHP & Symfony

3.2.1. PHP – PHP:Hypertext Preprocessor

"PHP: the accidental language"

- *Robert "Uncle Bob" Martin*

PHP je jedan od najpopularniji skriptni jezika otvorenog koda namijenjen za web razvoj. Sličan je C programskom jeziku po sintaksi i njegova osnovna namjena je za programiranje dinamičnih web stranica.

Jezik se pojavio 1994. godine originalno kreiran od strane Rasmus Lerdorf-a. PHP je serverski skriptni programski jezik. Jedna od značajki jezika je da od početka imao podršku za jednostavno ubacivanje u HTML (*HyperText Markup Language*) kod, prezentacijski jezik za izradu stranica [5]. Uskoro je postao dovoljno moćan da se može koristiti kao predložak za razvoj dinamičkih web stranica. Lerdorf ga je koristio za praćenje posjeta na njegov online životopis. Aktivna verzija za vrijeme pisanja ovog rada je 7.4.

Korištena verzija: PHP 7.2.20 (cli) (built: Jul 2 2019 13:37:16) (NTS) Copyright (c) 1997-2018 The PHP Group Zend Engine v3.2.0, Copyright (c) 1998-2018 Zend Technologies

3.2.2. Symfony 4

Predložak je kolekcija biblioteka koje predstavljaju kostur arhitekture za izgradnju aplikacije. Pritom je fokus na bržem, jednostavnijem i sigurnijem razvoju.

Symfony je PHP predložak koji omogućuje izgradnju web stranica ugodnijom i bržom. Kao predložak nudi sve što bi očekivali od njega, od pružanja potpune kontrole nad konfiguracijom, strukture direktorija, do vanjskih programskih proširenja tj. biblioteka (eng. *plugins/libraries*) uz fokus na ponovnu iskoristivost, brzinu i fleksibilnost [6].

Baziran je na modularnoj arhitekturi. Kod izrade projekta koriste se oni moduli/paketi koji su potrebni za razvoj. Symfony komponente su samostalne i mogu se koristiti u drugim

predlošcima. Korištena verzija je 4.4.9., podizanje na zadnju, *major* verziju praktičnog rada (5.1) trenutno nije moguća zbog ovisnosti nekih paketa (npr. FosUserBundle) koji trenutno nisu podržani na njoj.

Značajke Symfony MVC predložka:

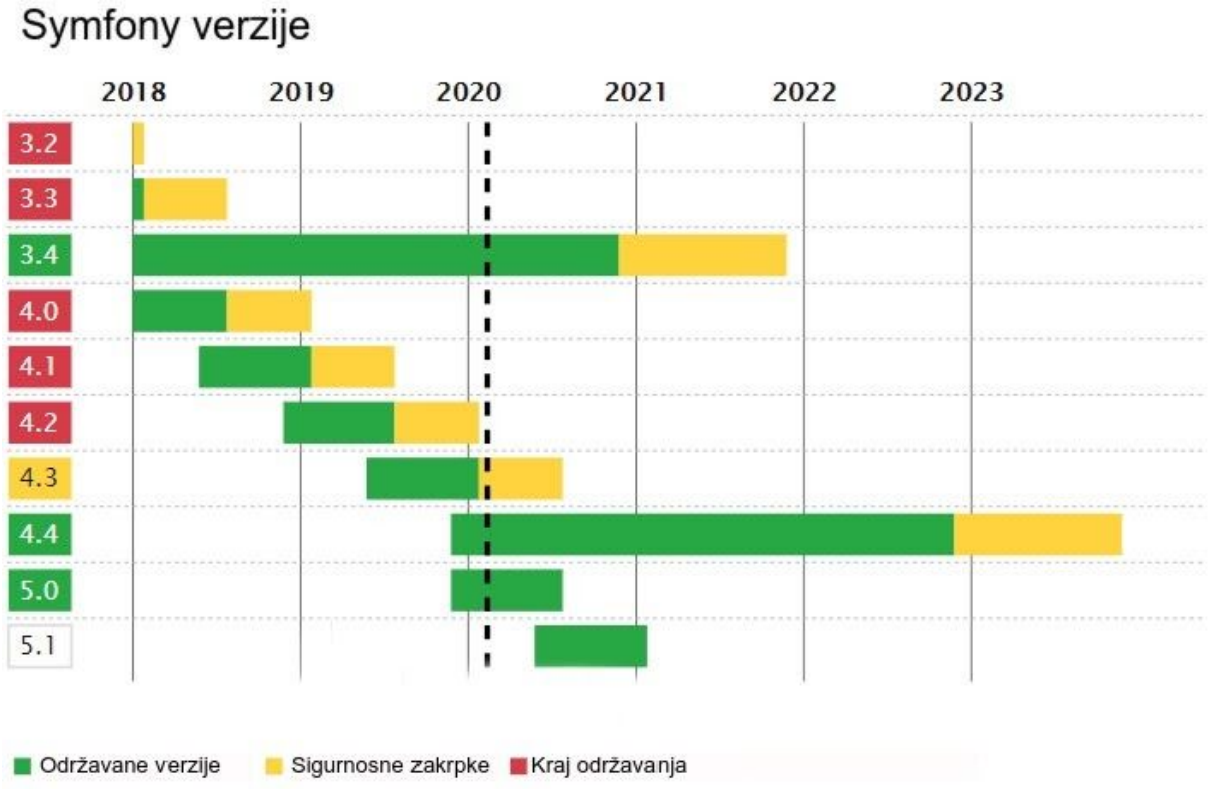
- Putanje (eng. *route*)
- Servisi
- Sigurnost
- Forme
- API
- ORM
- Događaji
- Testiranje
- Translacije

Značajke Symfony-a:

- apstrakcija baze (PDO s Doctrin-om)
- PHPUnit predložak za testiranje aplikacije
- Twig predložak za prikaze
- MVC (eng. *model* - *view* - *controller*) struktura
- sigurnosna komponenta
- komponenta za rad s događajima (eng. *events*)
- odlična dokumentacija
- velika zajednica (2000+ aktivnih korisnika na glavnom repozitoriju)

Značajke Symfony 4.4.9:

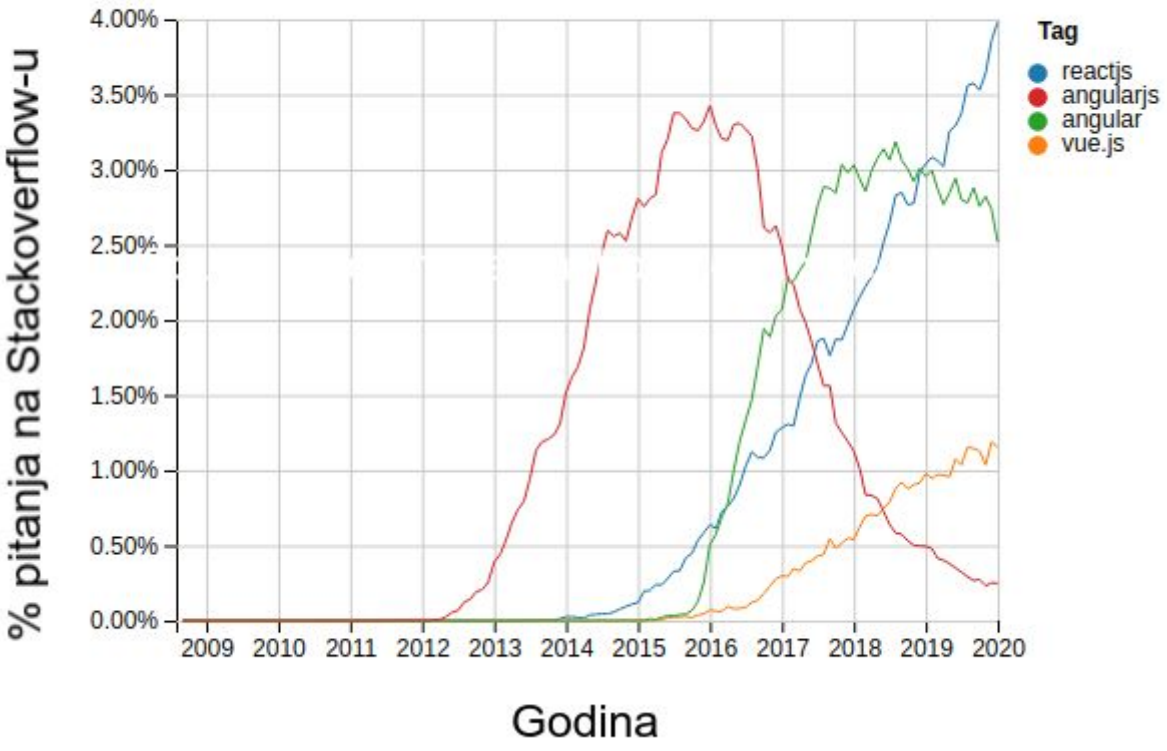
- 70% manji od verzije 3, po osnovi ima samo one pakete koji su potrebni za rad samog predloška, bez osnovnog ORM-a (za bazu), predloška za prikaze (Twig), sigurnosne komponente, komponente za izgradnju formi i ostalih. Zbog toga Symfony 4 je mikro predložak (eng. *microframework*)
- dodan je Flex, alat odnosno dodatak composer-u za automatsko namještanje Symfony aplikacije, svaki paket prati "recept" definiran u Symfony Flex repozitoriju, on će napraviti registraciju paketa (eng. *bundle*) ovisno o razvojnoj okolini (prod/test/dev), kopirat će datoteke s repozitorija u konfiguracijski direktorij aplikacije
- verzija 4.4. je dobila status LTS (eng. *long term support*) [7], što znači do 3 godine od izdavanja pružanja aktivne podrške u obliku proširenja i sigurnosnih zakrpa (Slika 4.)
- unaprijeđeni alat za debugiranje koji prati cijeli proces od *Request*-a do *Response*-a



Slika 4. Symfony aktivne verzije

3.3. React

Za izgradnju klijentske strane aplikacije tj. prednjeg sučelja (eng. *frontend*) koristio se React. Pritom se služilo najnovijim značajkama Reacta, a to su *Hooks*, "zamjena" za trenutni način definiranja komponenti (*class-based*) koji je bio zastupljen kao primarni način sve do verzije 16.8, odnosno pojave Hooks [9].



Slika 5. Usporedba popularnosti *frontened* biblioteka na Stackoverflow platformi

3.3.1. Koncept React-a

React je JavaScript biblioteka za kreiranje korisničkih sučelja. Prva stabilna verzija je izdana 29. svibnja 2013. godine a trenutna stabilna verzija je 16.13.1 (19.06.2020.). Kreator je Jordan Walke, softverski programer iz Facebook-a. Glavna značajka je prikazivanje obrađenih podataka u DOM-u kroz komponente tj. kreiranje moćnih korisničkih sučelja. Rad

s React-om često zahtijeva proširenje s dodatnim bibliotekama popust Redux-a za upravljanjem stanjem ili komunikacije sa serverom pomoću Axios biblioteke.

Virutal DOM

Predstavlja virtualnu reprezentaciju korisničkog sučelja koje je trenutno u memoriji i potpuno je sinkronizirano sa "pravim" DOM-om. To omogućuje da kažemo React-u što da napravi s nekim elementom kada se dogodi neka promjena umjesto da radimo direktnu izmjenu na DOM-u. S time imamo apstrakciju. Princip rada bi bio:

1. render() se pozove
2. prije njega se usporedi stari virtualni DOM (koristi se jer je brži od pravog DOM preglednika i *re-rendera* virtualni DOM i to je interna optimizacija React-a)
3. ako ima razlike onda se *render*-a DOM preglednika

JSX

React koristi JSX (JavaScript XML, ekstenzija od ECMA Script) sintaksu, koja je opcionalna ali olakšava povezivanje HTML koda s JS-om.

Komponente

Komponente se mogu smatrati HTML elementima, dok su konceptualno definirane popust JavaScript funkcija, primaju *props* i vraćaju React element koji opisuje što će se prikazati na ekranu. Korištenjem React-a kreiramo komponente koje su ponovno iskoristive. Cilj je da imamo mnogo manjih komponenti gdje grupiranjem dobijemo cjelinu. Postoje dvije vrste komponenti, koje se kreiraju instanciranjem baze klase (eng. *class based*) ili funkcijske komponente.

Klasne komponente:

- proširuju baznu klasu Component
- imaju pristup *state-u*
- imaju pristup React životnom ciklusu (eng. *life cycle*) komponente

Funkcijske komponente:

- prezentacijske komponente
- nemaju pristup stanju (nakon verzije 16.8. imaju korištenjem React Hooks)
- nemaju pristup životnom ciklusu komponente

Problem klasnih komponenti

Klasne komponente su poznate po tome da se pisanjem njih gomila količina koda koja se možda uopće neće koristiti ili ne bi smjela izvršiti u određenim trenutcima, pa se treba pisati i provjera za to s obzirom na to da li se kod treba izvršiti kod svake promjene (npr. API pozivi za dohvaćanje). Drugi razlog bi bio da komuniciranje između klasnih komponenti postaje nepraktično ovisno kako aplikacija raste, dosta ovisi o načinu prosljeđivanja podataka (eng. *props*) unutarnjim komponentama i njihovom prikazu.

Bolji način pisanja komponeti koje trebaju komunicirati s ciklusnim metodama komponente i imati pristup stanju je s novom značajkom React-a, a to su Hooks. Detaljnije o funkcijskim komponentama slijedi u nastavku kroz React Hooks poglavlje.

Single page aplikacija

React je baziran je za izradu *single-page* aplikacija (SPA). SPA imaju samo jedno učitavanje na klijentskoj strani, dok *multi page* aplikacije sav sadržaj učitava zasebno ovisno koja je stranica otvorena u pregledniku. Sa sljedeće tablice su vidljive prednosti i mane kod odabira da raditi jednostraničnu ili višestraničnu aplikaciju.

Tablica 16. SPA vs MPA

	SPA	**MPA**
UX	Brži prelaz po stranicama, bogatiji doživljaj stranice	Učitavanje za svaku stranicu, bolja opcija za veće aplikacije
SEO	Oslanjaju se na API pozive što otežava indeksiranje za bolje rangiranje	Definiranje ključnih riječi po stranicama (prednost)
Sigurnost	Ukoliko je cijela aplikacija klijentska sigurnost može biti upitna	Zahtijevi se obrađuju na serverskoj strani
Razvoj	Teži početak (zahtijeva dosta apstrakcije), poslije ugodnija nadogradnja, iskoristivo i kod mobilnih uređaja	Klijentska i serverska strana su usko vezane što nije dobro
Brzina	Samo jedno inicijalno učitavanje svih stranica	Za svaki prelaz sa stranice zahtijeva komunikaciju sa serverom
Gdje koristiti	Socijalne mreže, email klijentske aplikacije	Poslovne aplikacije s web prodajom, blog, usluge s oglašivanjem, stranice s mnogo linkova

3.3.2. React Hooks

React *Hooks* su ugrađene funkcije koje omogućuju korištenje React stanja i životnog ciklusa komponenti bez da se trebaju pisati klasne komponente. Zbog toga su izrazito moćna značajka React-a jer omogućuju iskoristivost koda među komponentama bez da se naruši postojeća hijerarhija, te time postaju jednostavnija za korištenje (čitanje i proširivanje) [10].

React Hooks su eksperimentalno predstavljene u verziji 16.7. a od 16.8. su stabilne i sigurne za korištenje u produkciji. Za korištenje s React *Hooks* treba se pridržavati pravila:

- ne smiju se koristiti unutar uvjeta, petlji ili funkcija
- svi *hooks* se trebaju pozivati unutar React funkcije, a ne iz JavasScript funkcije
- ne mogu se koristiti unutar klasni komponenti

Značajke:

- poboljšavaju iskoristivost koda
- omogućuju kompoziciju koda
- fleksibilnost kod proširivanja
- mogu se koristiti samo unutar drugih *hooks* ili funkcijskih komponenti

***Hooks* funkcije:**

- useEffect
    - ovaj *hook* vraća funkciju koja sadrži kod koji će se izvršiti za svaki *render*, poziva se kada bi komponenta bila odmontirana (eng. *unmounted*), poželjno je u njemu raditi API pozive s dohvaćanjem ili preplaćivanje na neki od događaja
    - može se kontrolirati kada će se pozivati prosljeđujući mu niz podataka (s komponente) za koje će se pozivati
    - može se imati više useEffect() poziva, što je zgodno za rasporediti povezane dijelove koda
    - zamjena za ciklusne metode klasne komponente (*componentDidMount*, *componentDidUpdate*, *componentWillUnmount*)

    - izvršava se prvi put kada je komponenta završena s učitavanjem te zatim svaki put kada se njeno stanje promijeni
- useState
    - za imati pristup stanju prije se trebala kreirati klasna komponenta, pomoću ovog hooka dobije se pristup stanju komponente. Pozivom useState funkcije vratit će stanje (inicijalno) i funkciju za promjenu tog stanja
- useContext
    - ovaj hook prima context objekt i vraća njegovu trenutnu vrijednost baziran na Context API funkcionalnosti React-a, predstavlja alternativu Redux-u

3.3.3. React Context

Jedan od najznačajniji dodataka React-u jer daje mogućnost korištenja "globalnog" stanja bez da se treba prosljeđivati kroz svaku komponentu dok ne dođe do one kojoj treba pristup tom stanju [11]. Primjer toga bi bio kada imamo npr. trenutno prijavljenog korisnika, stanje košarice kod kupovine, teme ili odabrane lokalizacije od strane korisnika. Da bi se mogao koristiti Context API treba se prvo kreirati što je vidljivo sa sljedeće slike 6. [12]:

```
import React, {createContext, useContext, useReducer} from 'react';

export const StateContext = createContext();

export const StateProvider = ({reducer, initialState, children}) =>(
    <StateContext.Provider value={useReducer(reducer, initialState)}>
        {children}
    </StateContext.Provider>
);

export const useStateValue = () => useContext(StateContext);
```

Slika 6. Definiranje Context-a

Sa *createContext* definira se objekt koji sadrži *Provider* komponentu koja omogućuje preplaćivanje drugim komponentama koje trebaju znati o promjeni stanja, te *Consumer*

komponentu pomoću koje se vrši pristupanje globalnom stanju. *useStateValue* funkcija vraća niz sa stanjem i akcijom za promjenu tog stanja koje se može koristit u bilo kojoj komponenti, i uvijek ćemo dobiti ono stanje koje smo proslijedili kroz *Provider*-a (Slika 7.).

```
const [{ user }, dispatch] = useStateValue();
```

Slika 7. Korištenje Context-a

Nakon definiranja globalnog stanja, korištenje se može vidjeti na donjoj slici (8.) gdje se poziva *dispatch* metoda nakon što se korisnik uspješno prijavi. Korištenjem *dispatch* metode ovisno o tipu i akciji mijenja se globani stanja.

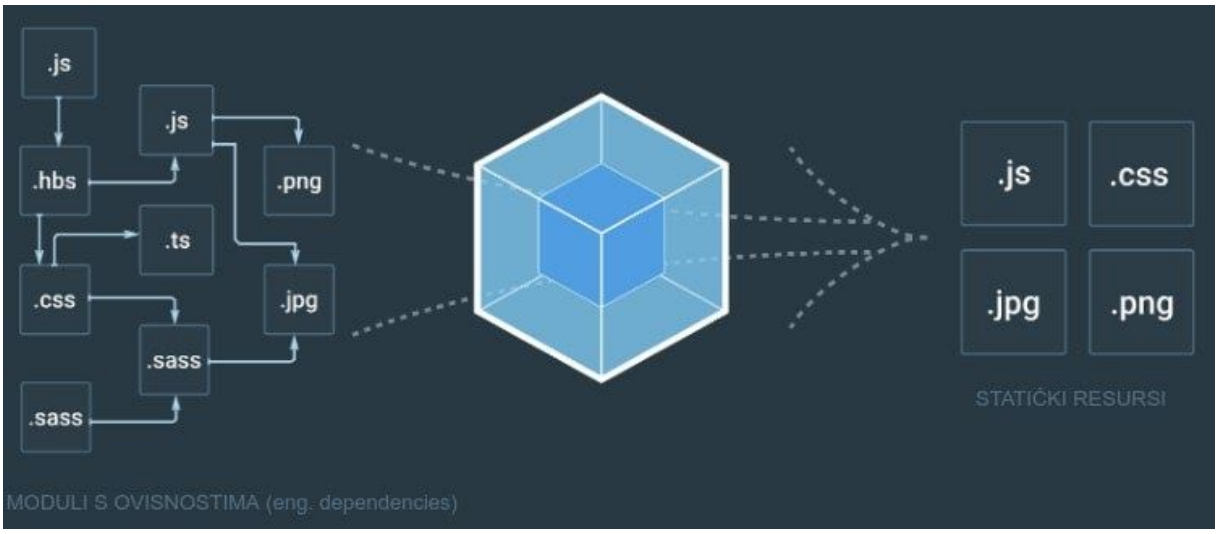
```
dispatch({
  type: 'setUser',
  updateUser: {
    id: user.id,
    firstName: user.firstName,
    lastName: user.lastName,
    email: user.email,
    checkedIfLogged: true,
  }
})
```

Slika 8. Primjer korištenja *dispatch* metode za postavljanje globalnog state-a

3.4. Korištene biblioteke, paketi i alati

3.4.1. Webpack Encore

Webpack je alat koji uzima sve module aplikacije s proširenjima (eng. *dependencies*), slikama i slično koji su na neki način povezani i "zapakira" ih u standardni JavaScript, CSS te neki od formata za slike [13]. Moduli u Symfony-ju se upravljaju s bibliotekom Assetic koja se temelji na PHP-u, a Encore je dodatak orijentiran za korištenje kod Symfony aplikacija (može se koristiti i samostalno) koji će uzeti sve te module s proširenjima i od njih napraviti grupaciju modula (Slika 9.).



Slika 9. Prikaz princip rada Webpack-a

Pomoću Encore-a se dosta lakše definiraju ulazne točke sa svim modulima, jednostavnije razdvajanje konfiguracija za dijelove aplikacije pa se mogu nezavisno pozivati za grupiranje i minimizaciju (manji finalni paket što znači da preglednik brže dohvaća sa servera) datoteka. Primjer dijela konfiguracije za klijentsku stranu je prikazan na slici 10.

```
Encore
    .setOutputPath('public/build/frontend')
    .setPublicPath('/build/frontend')

    .addEntry( name: 'css/main-frontend', src: './assets/css/main-frontend.scss')

    .addEntry( name: 'js/main-frontend', src: './assets/js/main-frontend.js')
```

Slika 10. Ulazne točke za grupiranje modula (primjer klijentske konfiguracije)

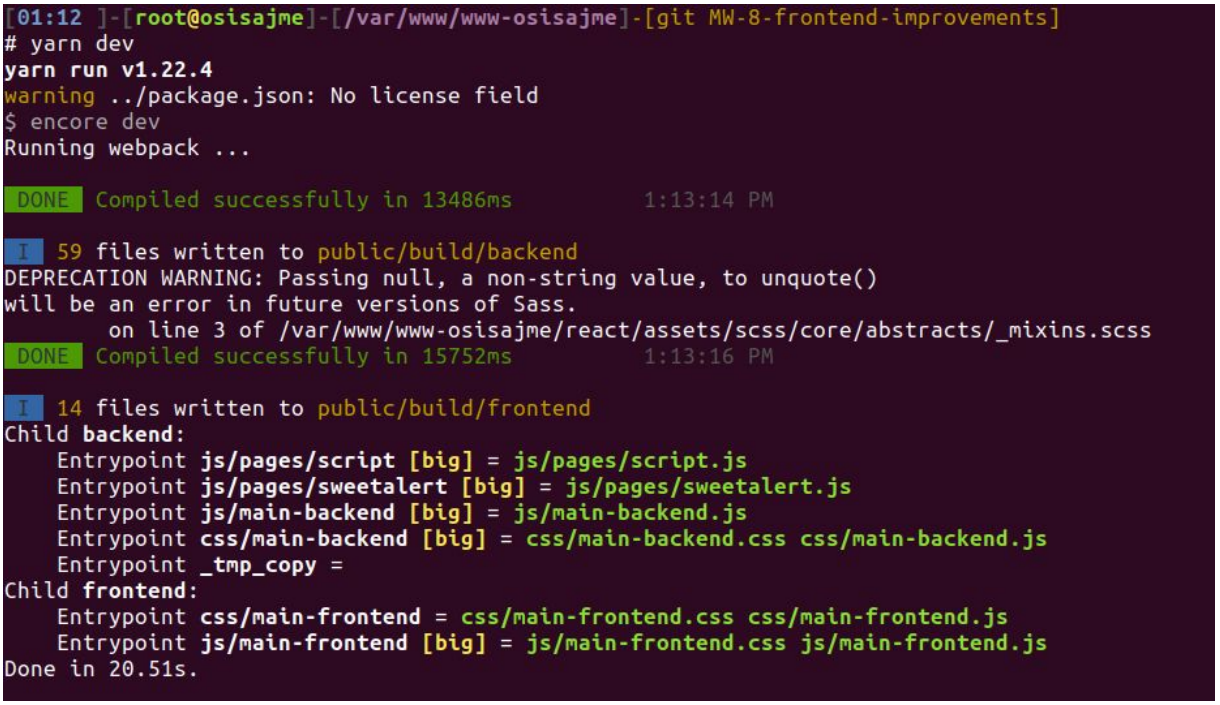
Za instalaciju i daljno korištenje kod modulacije svih paketa, bitno je imati jedan od upravitelja paketima (eng. *package manager*), NPM i Yarn su dva najpopularnija. U radu se služilo s Yarn-om. Instalacija Encore-a se može napraviti na serverskoj (pomoću composer-a) ili klijentskoj (pomoću Yarn/NPM). Sa slika 11. i 12. je vidljiv dio package.json datoteke s definiranim skriptama za pozivanje Webpack Encore-a, te samog *build* procesa.

Yarn je upravitelj paketima napravljen od strane Facebook-a, alternativa NPM-u, radi instalaciju iz registra (package.json), u usporedbi s NPM:

- NPM je više zastupljeniji
- iste funkcionalnosti
- instalacija paketa je nešto brža kod Yarn-a i manje striktnija

```
{
    "devDependencies": {},
    "license": "UNLICENSED",
    "private": true,
    "scripts": {
        "dev-server": "encore dev-server",
        "dev": "encore dev",
        "watch": "encore dev --watch",
        "build": "encore production --progress",
        "dev-backend": "encore dev --progress --config-name backend",
        "dev-frontend": "encore dev --progress --config-name frontend",
        "build-backend": "encore production --progress --config-name backend",
        "build-frontend": "encore production --progress --config-name frontend"
    },
```

Slika 11. Package.json kofiguracijska skripta



Slika 12. Yarn build za dev okolinu

3.4.2. jQuery

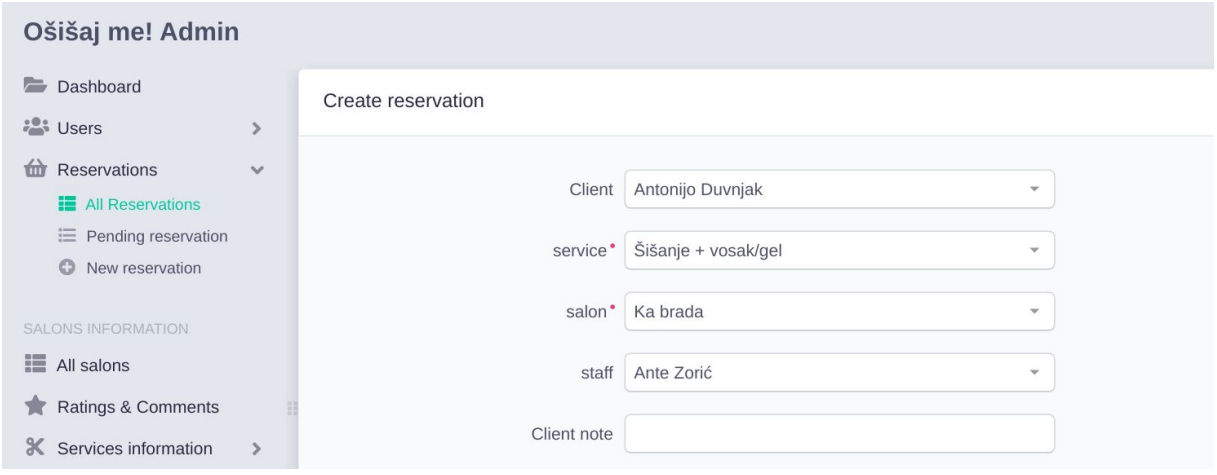
jQuery je JavaScript biblioteka dizajnirana za jednostavnije pristupanje i upravljanje elementima na HTML DOM stablu, kao i rukovanje događajima, CSS animacijama i Ajax pozivima. Besplatni softver otvorenog koda s MIT licencom [14].

Od svibnja 2019. jQuery koristi 73% od 10 milijuna najpopularnijih web stranica.

Web analiza pokazuje da je to najrasprostranjenija JavaScript biblioteka koja ima 3 do 4 puta veću upotrebu od bilo koje druge JavaScript biblioteke. Modularni pristup jQuery-u omogućava stvaranje moćnih dinamičnih web stranica i web aplikacija. Danas, u vrijeme kada je postoji iznimno mnogo JS biblioteka, jQuery i dalje je jedan od najpopularnijih ponajviše jer je olakšao upravljanje DOM-om raznim metodama i API-jevima. Za razliku njemu, React koji u zadnje vrijeme dobiva sve više na popularnosti, je fokusiran kako jednostavno upravljati stanjima na stranicama i njihovom prikazu. To može uključivati umetanje dinamičnih podataka, kao što su korisnikov profil i slika, te učinkovito ažuriranje stranice kada se podaci promijene.

3.4.3. Easyadmin bundle

EasyAdmin je Symfony paket koji pomaže kod kreiranja administracijskog sučelja bez potrebe za pisanja koda, odnoso sve se može kreirati kroz samu konfiguraciju. Podržava Symfony 4.1+, automatsko generiranje CRUD (eng. *create, read, update, delete*) operacija, translacije, puna podrška s radom na bazi kroz ORM. Korišten je prilikom kreiranja prve verzije administrativnog sučelja (Slika 13.) [15].



Slika 13. Administrativno sučelja korištenjem EasyAdmin paketa

3.4.4. Datatables

Iznimno popularan dodatak jQuery-ju za kreiranje tabličnog prikaz podataka, paginacije, automatske pretrage i filtracije podataka, sortiranje na više stupaca, jednostavno proširenje dodatnim funkcionalnostima, optimiziran za rad na mobilnim uređajima (responzivnost), omogućuje razne načine učitavanja podataka u tablicu.

3.4.5. Composer

Composer je upravitelj paketima (eng. *dependency manager*) za PHP. Jednostavan alat za integraciju vanjskih paketa ili biblioteka u PHP. Svi paketi se nalaze na globalnom repozitoriju Packagist [https://packagist.com/], te nakon instalacije jednog od paketa, se generira automatski učitavač (eng. *autoloader*) s obzirom na strukturu direktorija i putanja. Pomoću njega se može jako brzo napraviti struktura aplikacije s osnovnim nužnim paketima za rad aplikacije. Ključne dvije datoteke su composer.json gdje su definirani paketi te composer.lock koji služi kao provjera prilikom pokretanja instalacija ili nadogradnje postojećih a da se pritom ne naruše zaključane verzije postojećih paketa, inače ne bi bili kompatibilni.

3.4.6. Open react template

Besplatni predložak otvorenog koda za React naslovnu stranicu za brzu izradu projekta. Korišten za definiranje strukture i postizanje responzivnosti stranice ovisno na kojem se uređaju koristi [16].

3.4.7. Admin LTE

Besplatni predložak otvorenog koda za kreaciju responzivnog administrativnog sučelja. Baziran je na Bootstrap predlošku i JS/jQuery biblioteci. Jednostavan je za proširenje funkcionalnosti [17].

3.5. Ostali alati

3.5.1. Git

Alat za praćenje izmjena u kodu (eng. *version control system*). Prva službena verzija objavljena 2005. godine, iznimno brzo je dobio na popularnosti. U usporedbi s drugim sličnim alatima, Git je jednostavan za korištenje, besplatan, responzivan, dizajniran da dobro radi i s tekstualnim datotekama i najvažnije, grananje (eng. *branching*) tj. kreiranje nove grane na kojoj se može nezavisno nastaviti raditi od produkcijske verzije, što je odlično za potrebe testiranje, rada na nekoj novo stvari, a uvijek se jednostavno prebaciti na onu granu gdje si stao i želiš nastaviti i povezati (eng. *merge*) trenutni rad s njom. Prikaz git log poruka po završetku praktičnog dijela.

```
commit efe91aa0780009a5e0f094622f4acd0f17262224 (HEAD -> MW-9-symfony)
Author: Ante994 <adomjanovich@hotmail.com>
Date:   Sun Jun 7 23:23:45 2020 +0200

    symfony upgrade to 4.4.9, upgrate to 5 require fosuserbundle removal

commit 5ea0031ccad6a88a6a0114cb806bed56f0297b60 (origin/master, origin/HEAD, master)
Merge: 988ae16 f9d1fe6
Author: Ante994 <adomjanovich@hotmail.com>
Date:   Sun Jun 7 12:47:38 2020 +0200

    Merge branch 'MW-8-frontend-improvements'

commit f9d1fe6cc063c65aa398404092e3e21e8ec8da43 (origin/MW-8-frontend-improvements, MW-8-frontend-improvements)
Author: Ante994 <adomjanovich@hotmail.com>
Date:   Sat Jun 6 01:36:59 2020 +0200

    more improvements on reservation form for v1, datepicker hr localization, style match, price tag added ...

commit bec53203afd34a575a8717839d123846943d7011
Author: Ante994 <adomjanovich@hotmail.com>
Date:   Fri Jun 5 14:39:50 2020 +0200

    fix for scroll reveal

commit 988ae16171ed1186a83e8c59c2950ec65fb331e5
Merge: e4d2b7a 4fbbdf8
Author: Ante994 <adomjanovich@hotmail.com>
Date:   Fri Jun 5 11:01:15 2020 +0200

    Merge branch 'MW-8-frontend-improvements'

commit 4fbbdf8882bce02fa51aaf95905fc421d9a415d7
Author: Ante994 <adomjanovich@hotmail.com>
Date:   Fri Jun 5 10:59:52 2020 +0200

    [react frontend] user login, global state for logged user, checking if is logged (remember me), more functionality an
d improvements

commit caca657585835234db89679dcb2a00959b84e751
Author: Ante994 <adomjanovich@hotmail.com>
Date:   Thu Jun 4 17:03:27 2020 +0200

    adding global context state for user auth

commit 2fb89318fb008f7d7cd682176b2fa8c77ffaef36
Author: Ante994 <adomjanovich@hotmail.com>
Date:   Thu Jun 4 12:08:49 2020 +0200

    not found page added

commit 5afe8369f10684d7976083c4778314de11daff54
Author: Ante994 <adomjanovich@hotmail.com>
Date:   Thu Jun 4 11:51:56 2020 +0200

    sending mail with reservation data + implemented style
```

Slika 14. Git log

GitHub je za razliku od Git-a predstavlja mjesto na kojem se nalaze repozitoriji napravljeni korištenjem Git-a. Pa se može reći Git je alat dok je GitHub servis za projekte koji koriste Git.

3.5.2. MySQL Workbench

Samostalni besplatni alat za rad s bazom. Pruža jednostavno grafičko sučelje, puna podrška za rad sa SQL bazom podataka, mogućnosti koje nudi: kreiranje i pregled baza/tablica, kreiranje/optimiziranje/izvršavanje upita (eng. *query*), konfiguracija servera, razdvajanje razine prava po korisniku, pohrana i oporavak, pregled statusa servera i druge [18].

3.5.3. PHPStorm

PHPStorm je komercijalni (koristila se besplatna licenca koje je omogućena studentima) IDE (eng. *integrated development environment*) za programski jezik PHP napravljen od JetBrains kompanije. Napisan je u Javi. Izrazito moćan alat koji uz integrirane značajke nudi mnoštvo dodatnih opcija za proširenje kroz razne programske dodatke [19].

Glavne značajke bi mu bile: kompletna podrška za PHP verzije >=5.3, isticanje sintakse, automatsko kompletiranje koda, PHPDoc, opcije za brzo refaktoriranje, podrška za popularne PHP predloške (Symfony, Laravel, itd.), razni načini za testiranje rada aplikacije, povezivanje s bazom i još mnoge.

4. OPIS PRAKTIČNOG RADA

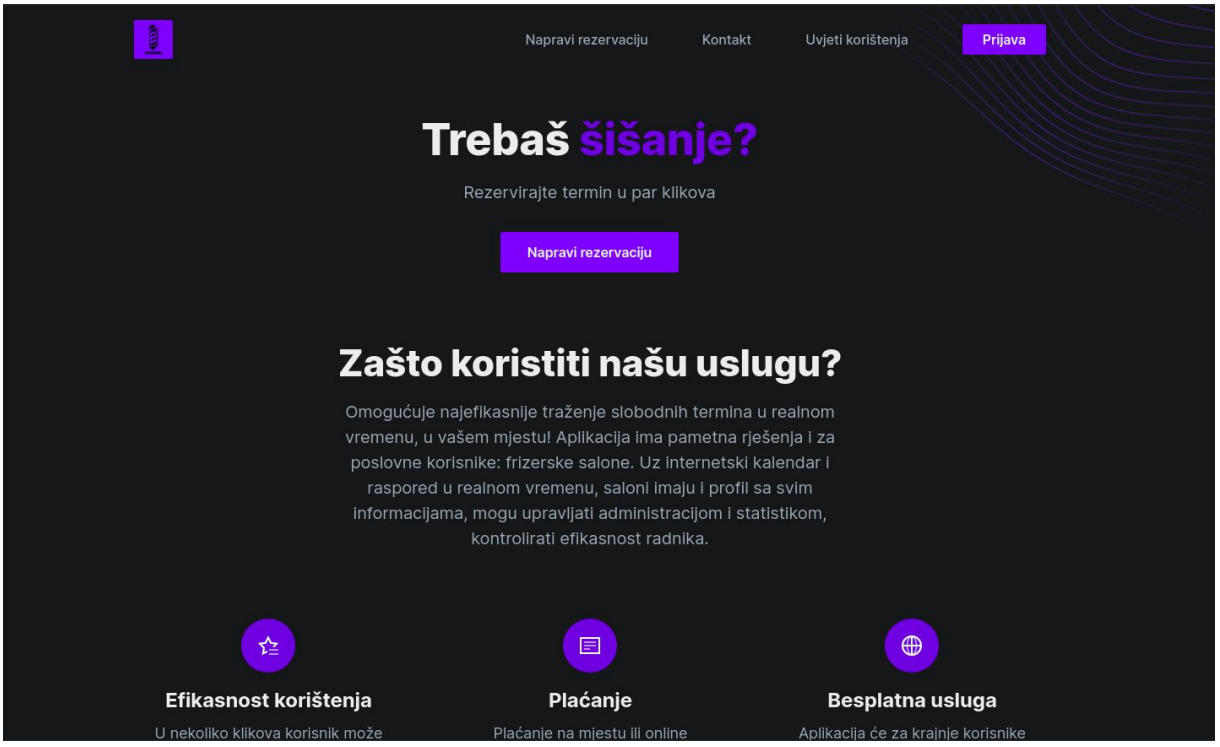
4.1. Ciljevi aplikacije

Ova aplikacija je zamišljena kao proizvod koja će olakšati svima onima koji traže uslugu šišanja. Kao glavna funkcionalnost je rezervacija termina u svega par klikova.

S druge strane, upravljanje pristiglim rezervacijama, odnosno upravljanje radom cijeloga salona, od definiranja različitih usluga i cijena pa do zaposlenika i njihovih smjena rada, partneri će dobiti bolju organizaciju poslovanja i veću efikasnost rada.

Aplikacija riješava aktualne probleme rezervacije usluge šišanja i nudi administrativno sučelje za vlasnike salona. Glavne problematike ovoga rada biti će opisane kroz nastavak uz popratne primjere koda, sami zahtjevi bili su:

- razrada sigurnosne komponente ovisno o razini prava korisnika i samom korištenju na višeklijentskom admnistrativnom sučelju
- implementacija administracijskog sučelja koje će olakšati upravljanje radom salona, zaposlenika, usluga i rezervacija samome vlasniku
- napraviti intuitivnu klijentsku stranicu s jednostavnom rezervacijom termina uz par klikova



Slika 15. Naslovna stranica (klijentska strana)

4.2. Razrada sigurnosne komponete ovisno o razini prava korisnika

4.2.1. Uloge korisnika i autentikacija

Izrada administrativnog sučelja je bazirana na korištenju više korisnika koji imaju pristup samo onim dijelovima za koji su autorizirani. Što znači, kada se osoba (vlasnik salona) jednom registrira kao partner dobiva puni pristup sučelju gdje može voditi upravljanje radom svog salona. Ta osoba ima svoju rolu, ROLE_PARTNER. Uloge u Symfony-u pomažu da se stvori hijerarhija prava korisnika. ROLE_SUPER_ADMIN je najjača rola jer je njoj omogućen pristup svim dijelovima aplikacije kao i pregled svih podataka. U usporedbi s njome, ROLE_ADMIN je zapravo identična po funkcionalnosti. Admin sučelju pristup imaju

uz admina, ROLE_PARTNER i ROLE_STAFF. Umjesto dodjeljivanja više uloga korisniku bolje se oslanjati na dobro definiranu hijerarhiju tj. struktura nasljeđivanja (Slika 15.).

```
role_hierarchy:
    ROLE_STAFF: ROLE_USER
    ROLE_PARTNER: ROLE_STAFF
    ROLE_ADMIN: ROLE_PARTNER
    ROLE_SUPER_ADMIN: ROLE_ADMIN
```

Slika 15. Hijerarhija uloga

4.2.2. Autorizacija

S definiranjem uloga i hijerarhije, su se stvorile razine prava. Sljedeći korak je definiranje kontrolnih točaka kojima se može pristupati ovisno o korisnikovoj ulozi. Nakon što se korisnik registrirao ili prijavio na stranicu, kreira se sesija koja u sebi sadrži i vezanu rolu za njegov korisnički račun. U svakome trenutku, odnosno kod svakog novog napravljenog HTTP poziva radi se provjera da li ima pristup određenoj stranici. Taj dio pripada Symfony sigurnosnoj komponenti.

Na sljedećoj slici (Slika 16.) se vide definirane točke pristupa koje su zaštićene ovisno o roli i hostu. Pomoću toga se jednostavno može ograničiti pristup cijeloj administraciji, ne prijavljenim korisnicima (autentikacija) ili običnim korisnicima (klijenti) koji nemaju pristup administraciji (autorizacija). Za potrebe rada odvojene su domene te se koristi osisajme.master za prikaz klijentskog dijela stranice, dok je prikaz administracije odvojen na admin.osisajme.master domenu. Taj dio se rješava kroz konifguraciju PuPHPet skripte i definiranje IP adresa u host mašini. Za to je potrebno napraviti vagrant reload ako je mašina već kreirana. Definirani host je podatak iz parametara.

```yaml
access_control:
    - { path: ^/login$, role: IS_AUTHENTICATED_ANONYMOUSLY }
    - { path: ^/register, role: IS_AUTHENTICATED_ANONYMOUSLY }
    - { path: ^/resetting, role: IS_AUTHENTICATED_ANONYMOUSLY }
    - { path: ^/$, role: ROLE_STAFF, host: '%osisajme_backend.base%' } # dev env
    - { path: ^/dashboard$, role: ROLE_PARTNER, host: '%osisajme_backend.base%' }
    - { path: ^/superadmin/$, role: ROLE_ADMIN, host: '%osisajme_backend.base%' }
```

Slika 16. Kontrola pristupa

Pošto se radi o administraciji koju koriste više korisnika s različitim pravima, potrebno je definirati za svaki resurs kojemu se pristupa da li pripada tome korisniku, odnosno da li ga može pregledavati, mijenjati ili brisati. Za to služe glasači (eng. *voter*). Cilj je da imamo neku vrstu sigurnosne provjere za svaku akciju koju korisnik napravi na nekome resursu. Ako ima pravo raditi dozvolit ćemo mu izvršavanje akcije, inače vraćamo poruku s kojom ga obavještavamo da nije autoriziran za izvršavanje zatražene akcije. Pošto želimo za svaki resurs (entitet) imati zasebnog *Voter*-a, može se definirati apstraktna klasa AbstractVoter (Slika 17.).

```php
/**
 * Perform a single access check operation on a given attribute, subject and token.
 * It is safe to assume that $attribute and $subject already passed the "supports()" method check.
 *
 * @param string $attribute
 * @param mixed $subject
 * @param TokenInterface $token
 *
 * @return bool
 */
protected function voteOnAttribute($attribute, $subject, TokenInterface $token)
{
    if ($this->decisionManager->decide($token, ['ROLE_ADMIN'])) {
        return true;
    }

    $user = $token->getUser();

    if (!$user instanceof User) {
        return false;
    }

    switch ($attribute) {
        case self::SHOW:
            return $this->canShow($token, $subject);
        case self::DELETE:
            return $this->canDelete($token, $subject);
        case self::CREATE:
            return $this->canCreate($token, $subject);
        case self::EDIT:
            return $this->canEdit($token, $subject);
    }

    throw new \LogicException( message: 'This code should not be reached!');
}
```

Slika 17. AbstractVoter klasa

Svaki novi *Voter* će ju implementirati s metodama koje određuju prava izvršavanja određene akcije koje korisnik zatraži. Akcije su najčešće standardne CRUD operacije. Sljedeći primjer je metoda **canEdit**, apstraktna metoda AbstractVoter klase koju je potrebno implementirati. Na slici (Slika 18.) je vidljivo da prima **token** iz kojeg se može dohvatiti rola korisnika. Drugi parametar je entitet za kojeg se testira da li ima pristup korištenja. Argument tom parametru je **salonReviw** entitet. Metoda vraća **true** ako korisnik ima pravo pristupa.

```php
/**
 * @param TokenInterface $token
 * @param SalonReview $salonReview
 * @return bool
 */
protected function canEdit(TokenInterface $token, $salonReview): bool
{
    if ($this->decisionManager->decide($token, array('ROLE_STAFF'))) {
        if ($token->getUser()->getSalon() == $salonReview->getSalon()) {
            return true;
        }
    }

    return false;
}
```

Slika 18. Implementacija apstraktne metode canEdit na SalonReviewVoter klasi

Pošto pristup administraciji uz ROLE_ADMIN (koji ima pristup svemu) imaju i korisnici s ulogom ROLE_PARTNER ili ROLE_STAFF, njihov pristup je baziran kojem salonu pripadaju. Tako je definirana pomoćna metoda **getSalon** na **User** entitetu gdje se dohvaća salon s **Partner** ili **Staff** entiteta (oni nasljeđuju User entitet s dodatnim poljima koja su prikladna tom entitetu) što je vidljivo na sljedećoj slici:

```php
/**
 * @return Salon|null
 */
public function getSalon()
{
    if ($this->getPartner() && $this->getPartner()->getSalon()) {
        return $this->getPartner()->getSalon();
    } elseif ($this->getStaff() && $this->getStaff()->getSalon()) {
        return $this->getStaff()->getSalon();
    }

    return null;
}
```

Slika 19. getSalon metoda na User entitetu

Sada dolazimo do dijela koji povezuje cijeli princip rada administracije koji uključuje obradu primljenog zahtjeva, izvršavanje sigurnosne komponente, pripreme podataka za prikaz na stranici i sami prikaz stranice. **AbstractAdminController** je apstraktni kontroler koji implementira bazne metode koje se koriste u svim drugim kontrolerima koji nasljeđuju bazni. Time se olakšava kreacija i proširenje aplikacije jer odmah imamo implementirane dijelove koje se trebaju izvršiti. Kod je time čitkiji jer nema "*copy paste"* dijelova. Sve što treba napraviti je implementirati apstraktne metode (Slika 20.):

```php
abstract protected function getEntityFormType(): string;

abstract protected function getIndexRoute(): string;

abstract protected function getShowRoute(): string;

abstract protected function getEntityClass(): string;

abstract protected function getTemplateEntityName(): string;

abstract public function datatable(Request $request): JsonResponse;
```

Slika 20. AbstractAdminController apstraktne metode

Apstraktne metode admin kontrolera:

- **getEntityFormType** vraća naziv kreirane forme za entitet
- **getIndexRoute** vraća naziv putanje za naslovnu stranu entiteta (predstavlja prikaz liste podataka zadanog entiteta preko *Datatable-a*)
- **getEntityClass** vraća naziv entiteta
- **getTemplateEntityName** vraća naziv direktorija gdje se nalaze Twig datatotke za definirani entitet
- **datatable** vraća podatke na zahtjev klijenta u JSON (eng. *JavaScript Object Notation*) formatu za popunjavanje *Datatable* tablice

Na sljedećim slikama je prikaz bazne implementacije **delete** metode (Slika 21.) te prikaz stranice s odgovorom nakon uspješnog brisanja (Slika 22.).

```php
/**
 * Delete record for passed ID
 *
 * @param int $id
 *
 * @return JsonResponse
 */
public function delete(int $id)
{
    $entity = $this->entityManager->getRepository($this->getEntityClass())->find($id);

    $this->denyAccessUnlessGranted( attributes: AbstractVoter::DELETE, $entity);

    $this->entityManager->remove($entity);
    $this->entityManager->flush();

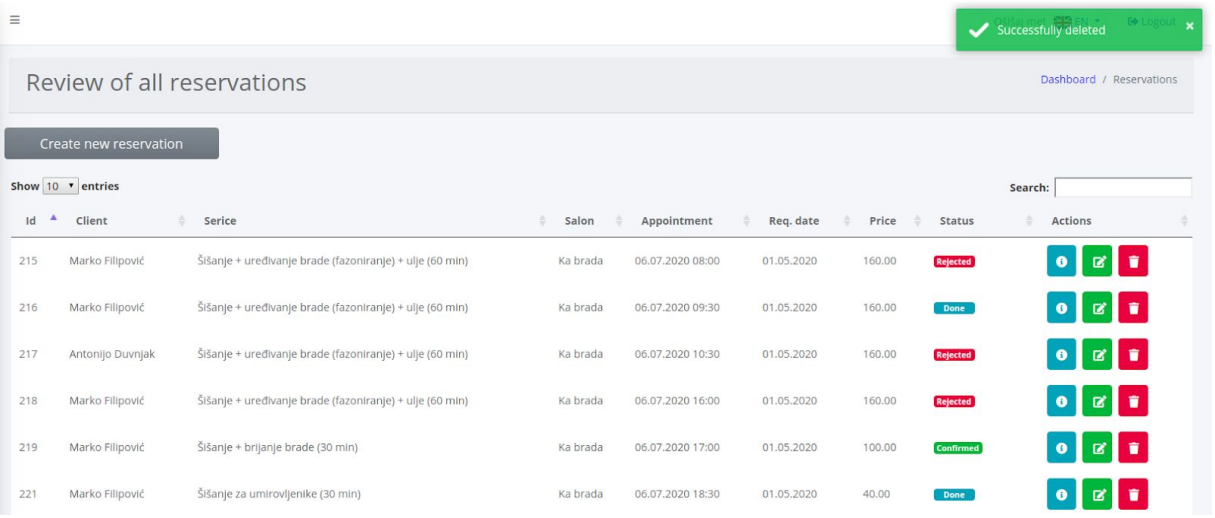
    $entityNameTranslated = $this->translator->trans($this->getEntityClassNameShort());
    $this->addFlash( type: 'success', $this->translator->trans( id: 'backend.partner.base.messages.delete', ['%entity%' => $entityNameTranslated, '%id%' => $id]));

    return new JsonResponse(
        [
            'message'   => 'Successfully deleted!',
            'level'     => 'success',
            'action'    => self::DELETE_STATE
        ],
        status: Response::HTTP_OK
    );
}
```

Slika 21. Prikaz implementacije delete metode

Princip rada bazne *delete* metode:

- prima id objekta zatraženog resursa (primljeni zahtjev)
- vrši provjeru da li postoji u bazi i pravo na brisanje preko *Voter*-a (sigurnosna komponenta)
- nakon toga ide brisanje (obrada nad podatcima)
- vraćanje odgovora (eng. *Response*) s porukom o uspješnom brisanju (prikaz stranice)



Slika 22. Pregled rezervacija na administraciji nakon akcije brisanja

Autentikacija i autorizacija korisnika predstavljaju srž sigurnosne komponente i zbog toga je jako bitno imati dobro definiranu strukturu po kojoj će se pristigli zahtjevi na server moći pozvati i obaviti potrebne provjere prava.

4.3. Upravljanje administracijom

Jednom kada korisnik (vlasnik salona) dobije korisnički račun (kreira ga admin) s razinom prava partnera može pristupati administraciji. Korisnički račun sadrži osnovne informacije o salonu jer partner nema razinu prava za kreiranje novog salona već samo uređivanje. Partner zatim treba napraviti sljedeće korake:

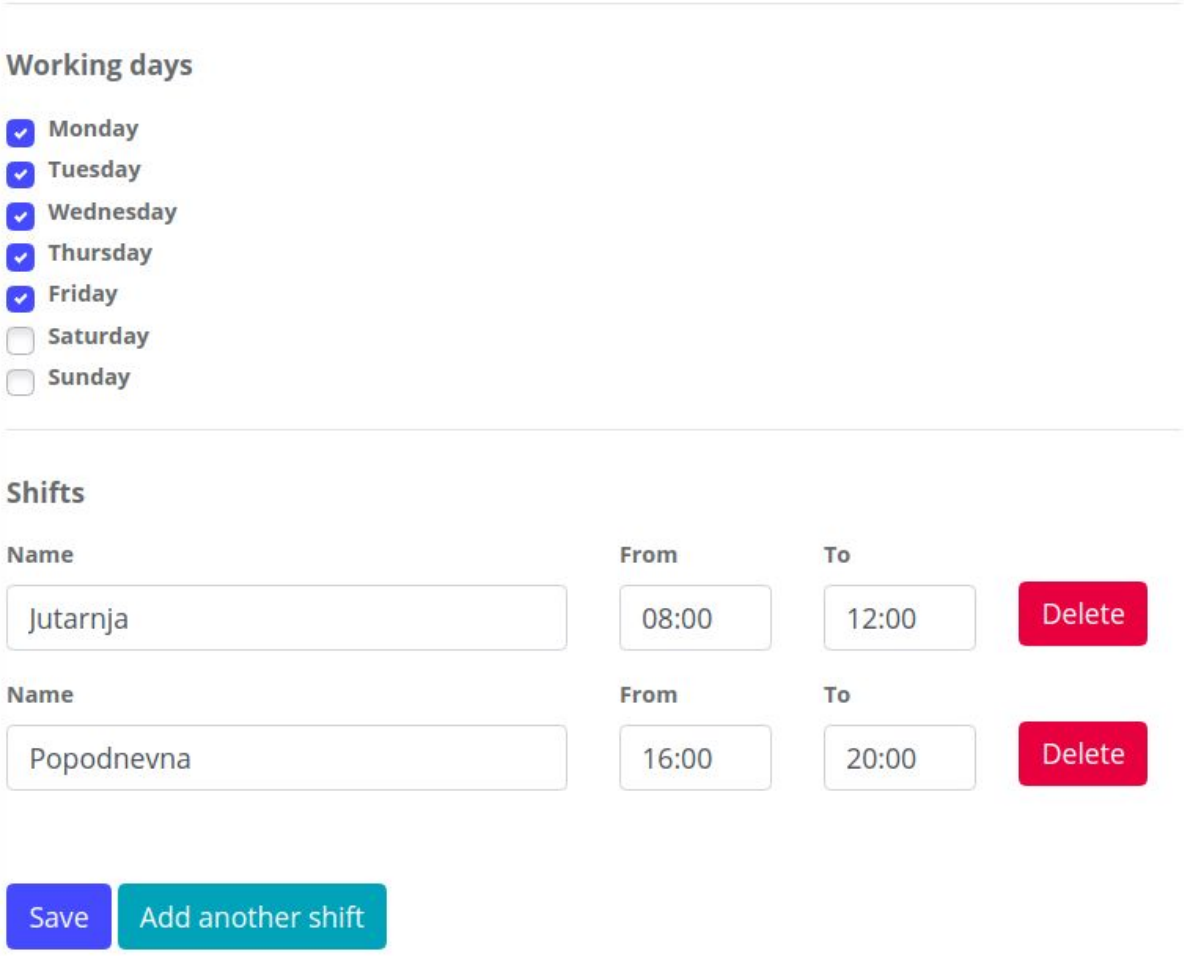
- popunjavanje informacija o salonu
- definiranje radnih dana i smjena
- definiranje usluga koje pruža
- dodavanje zaposlenika
- generiranje kalendara radnog vremena
- upravljanje rezervacijama

4.3.1. Popunjavanje informacija o salonu

Partner unosi informacije poput naziva, lokacije i adrese, kontakta, veličine prostora, web stranice (ako ima) i slično. To su podatci koji služe za okvirni pregled salona a i za potencijalna proširenja u budućnosti. Na primjer, ukoliko vlasnik salona nema vlastitu web stranicu, odnosno mjesto gdje pruža usluge oglašavanja i same rezervacije termina, može koristiti generiranu stranicu na klijentskoj strani sa zasebnim linkom koji predstavlja njegov salon i informacije koje je unio. Cilj je promocija. Tako nešto se može proširiti s blog funkcionalnosti, gdje se pisanjem tekstova koji mogu npr. sadržavati preporuke za korištenje preparata, koje su frizure u trendu, savjete za održavanje kose i slične stvari. Time se može istaknuti među ostalim salonima i lakše pristupiti samim klijentima.

4.3.2. Definiranje radnih dana i smjena

Partner nakon toga definira radne dane i smjene. Na osnovu toga će se poslije generirati kalendar radnog vremena zaposlenika. Bitno je odabrati sve dane kojima se radi, tako i smjene. One se automatski ne generiraju unaprijed, već je potrebno putem upravljanja kalendara generirati vremena koja će biti slobodna za primanje rezervacija. Taj dio se može automatizirati s cron zadatkom koji bi se izvršavao na tjednoj bazi. Na sljedećoj slici se može vidjeti jedan primjer definiranih radnih dana i smjena za salon.



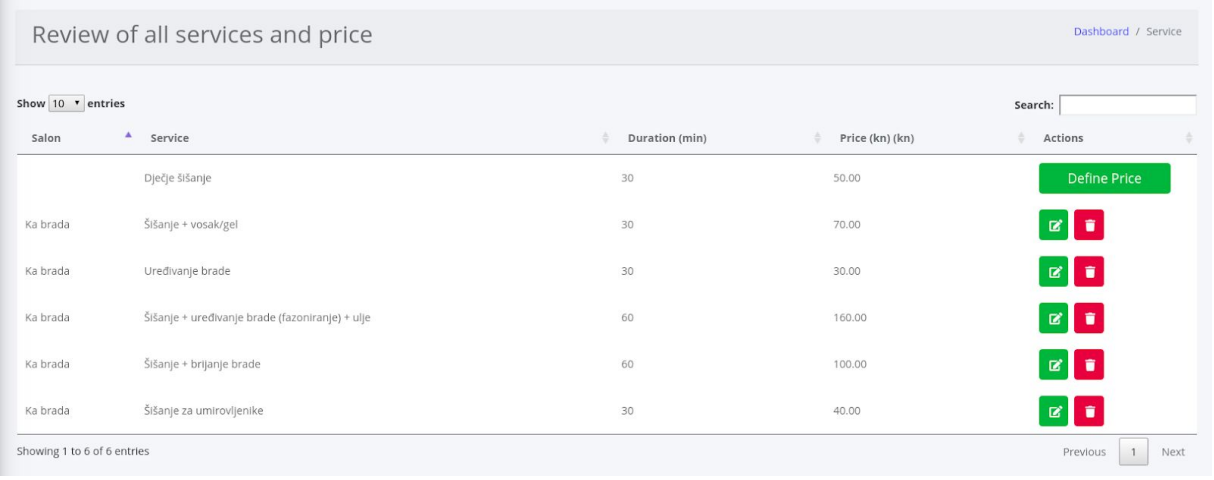
Slika 23. Definiranje radnih dana i smjena

4.3.3. Definiranje usluge

Kako se u nastavku može vidjeti, postoje tri glavne usluge koje klijent putem klijentske strane aplikacije može odabrati s vremenom trajanja od 30 ili 60 minuta:

- šišanje
- uređivanje brade
- šišanje + uređivanje brade

Usluge su zapravo već unaprijed definirane od strane administratora, jer na tome počiva cijela funkcionalnost aplikacije. Ideja aplikacije i njene glavne značajke je da bude jednostavna za korištenje. Razlika u daljnoj specifikaciji usluga je zasnovana na definiranju različitih cijena i detaljnijom opisu usluge (Slika 24.). Tako se može imati šišanje za odrasle s cijenom od 70 kuna, dok šišanje za djecu može biti 40 kuna, te opet šišanje za umirovljenike 50 kuna. Pritom će vrijeme biti uvijek ili 30 minuta ili 60 minuta. Time se ujedno bolje organizira rad zaposlenika jer klijentu se nudi mogućnost rezervacije u punom satu ili u pola sata.



Slika 24. Prikaz usluga

Definiranje cijene i trajanja je nužno da bi ta usluga bila dostupna koju klijent može zatražiti i odabrati slobodni termin. Na sljedećoj slici (Slika 25.) je prikaz forme s poljima koja su unaprijed odabrana (salon i usluga) te odabir za duljinu trajanja s opcijama 30/60 minuta.



Slika 25. Prikaz forme za definiranje usluge

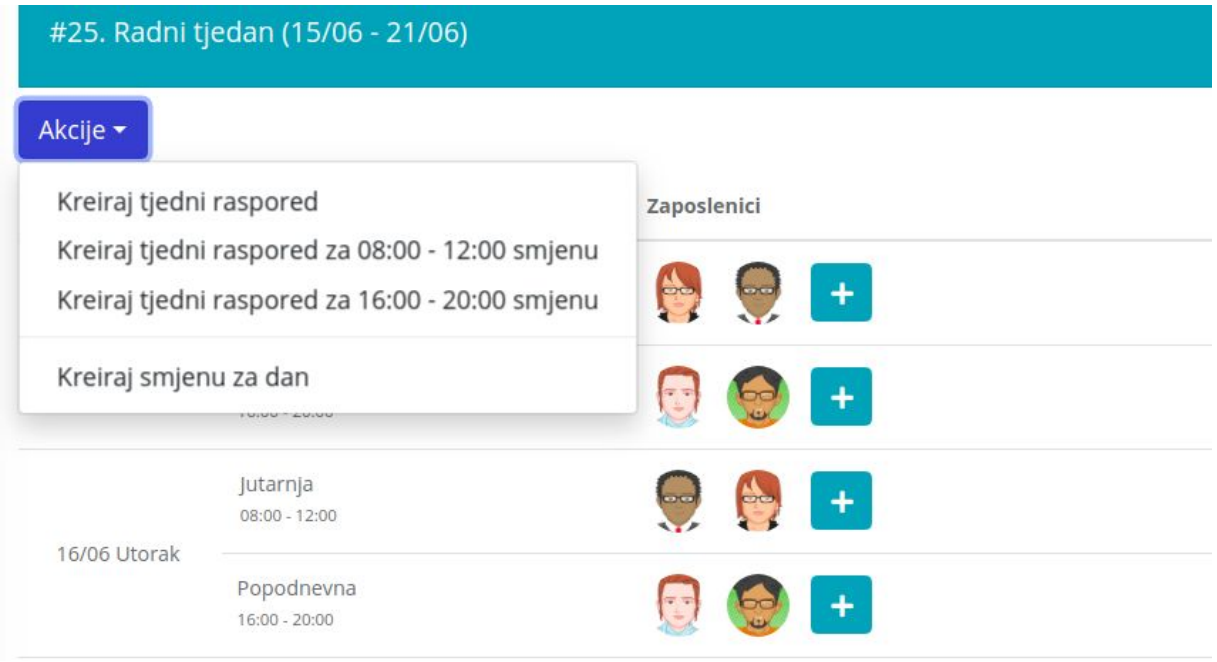
Ovdje je također ostavljeno mjesta za daljnja proširenja a vezano je na prijašnje spomenuti primjer gdje salon može imati zasebnu stranicu (na klijentskoj strani) za rezervaciju. Tu bi se mogla ponuditi cjelokupan asortiman koje salon nudi.

4.3.4. Dodavanje zaposlenika

Kreiranjem novog zasposlenika mu se otvara novi korisnički račun i postavlja uloga zaposlenika. Zaposlenik ima manju razinu prava pa tako npr. ne može pristupati uređivanju podataka o salonu ili nekog od drugih zaposlenika. Dodavanjem uloge korisnika se dodatno proširila funkcionalnost aplikacije. Zaposlenik može kreirati rezervaciju putem administracije što je bio nužan uvjet s obzirom na to kako bi stranica u početku funkcionirala, jer veća je vjerojatnost da će klijenti zatražiti rezervaciju na "stari" način. Također, na zaposlenika se mogu postaviti usluge koje može raditi, promijeniti mu status kada je neaktivan (npr. na godišnjem odmoru) i slično.

4.3.5. Upravljanje kalendarom radnog vremena

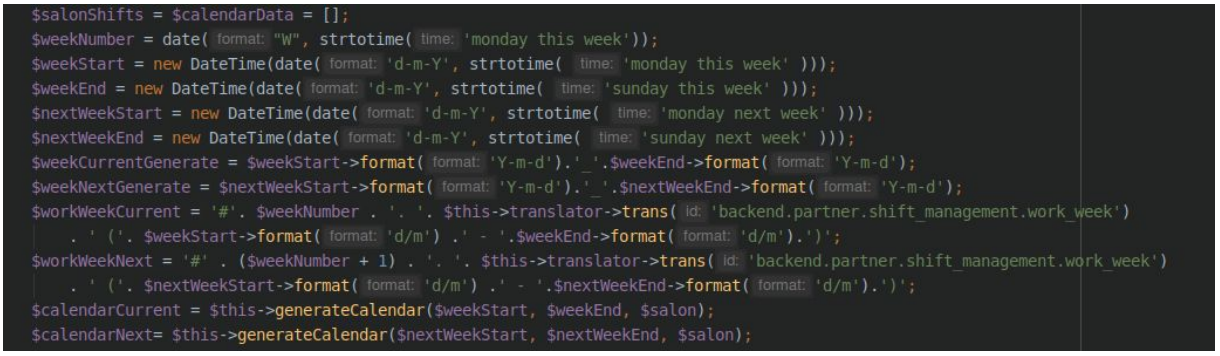
Glavna funkcionalnost administracije je upravljanje radnim vremenom zaposlenika, upravljanje primljenim rezervacijama i pregled statusa zaposlenika (Slika 26.). U nastavku će biti prikazana razrada i dijelovi implementacije, sama po sebi predstavljaju najkompleksniji dio aplikacije jer su sadržani svi prethodno definirani elementi.



Slika 26. Pregled kalendara

Generiranje prikaza kalendara

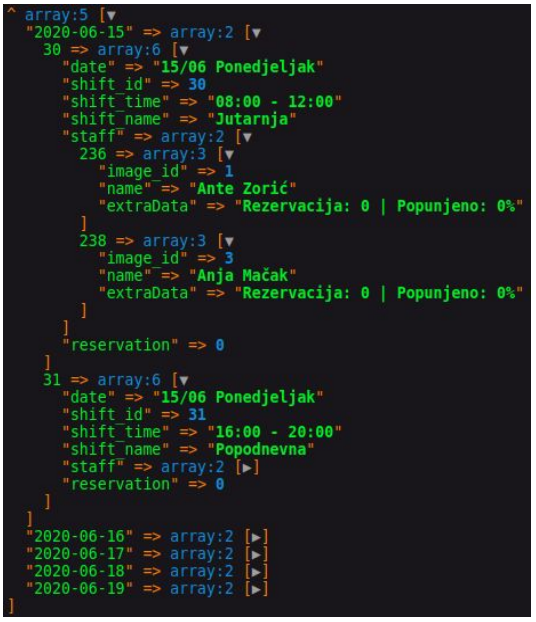
Ako admin pristupa ovom djelu stanice prvo treba odabrati salon za koji želi vidjeti kalendar aktivnosti. S gornje slike se vidi da za svaki radni dan salona postoji generirana smjena i zaposlenici koji rade taj dan. To se može generirati pomoću predočenih akcija čime se olakšava korištenje. Kalendar prikazuje trenutni i sljedeći radni tjedan. Priprema podataka je vidljiva na sljedećoj slici:



```
$salonShifts = $calendarData = [];
$weekNumber = date( format: "W", strtotime( time: 'monday this week'));
$weekStart = new DateTime(date( format: 'd-m-Y', strtotime( time: 'monday this week' )));
$weekEnd = new DateTime(date( format: 'd-m-Y', strtotime( time: 'sunday this week' )));
$nextWeekStart = new DateTime(date( format: 'd-m-Y', strtotime( time: 'monday next week' )));
$nextWeekEnd = new DateTime(date( format: 'd-m-Y', strtotime( time: 'sunday next week' )));
$weekCurrentGenerate = $weekStart->format( format: 'Y-m-d').'_'.$weekEnd->format( format: 'Y-m-d');
$weekNextGenerate = $nextWeekStart->format( format: 'Y-m-d').'_'.$nextWeekEnd->format( format: 'Y-m-d');
$workWeekCurrent = '#'. $weekNumber . '. '. $this->translator->trans( id: 'backend.partner.shift_management.work_week')
    . ' ('. $weekStart->format( format: 'd/m') .' - '.$weekEnd->format( format: 'd/m').')';
$workWeekNext = '#' . ($weekNumber + 1) . '. '. $this->translator->trans( id: 'backend.partner.shift_management.work_week')
    . ' ('. $nextWeekStart->format( format: 'd/m') .' - '.$nextWeekEnd->format( format: 'd/m').')';
$calendarCurrent = $this->generateCalendar($weekStart, $weekEnd, $salon);
$calendarNext= $this->generateCalendar($nextWeekStart, $nextWeekEnd, $salon);
```

Slika 27. Priprema podataka za kalendar

Nakon što se generiraju datumi za aktivni i sljedeći tjedan, glavna metoda koja napravi pripremu podataka (Slika 28.) koja uključuje zaposlenike za datum, rezervacija itd., koji su vidljivi na samoj stranici je generateCalendar. Ona prima datum početka i kraja tjedna, te salon za koji se generira sadržaj. Prvo se pomoću DQL radi upit gdje se dohvate definirane smjene s zaposlenicima po danima. Također se napravi i upit za računanje postotka rezervacija po danu, čime se zatim može lako vidjeti koji od zaposlenika ima više ili manje slobodnih termina, pa čak po potrebi se može napraviti i dodavanje novo zaposlenika za dan.



```
^ array:5 [▼
  "2020-06-15" => array:2 [▼
    30 => array:6 [▼
      "date" => "15/06 Ponedjeljak"
      "shift_id" => 30
      "shift_time" => "08:00 - 12:00"
      "shift_name" => "Jutarnja"
      "staff" => array:2 [▼
        236 => array:3 [▼
          "image_id" => 1
          "name" => "Ante Zorić"
          "extraData" => "Rezervacija: 0 | Popunjeno: 0%"
        ]
        238 => array:3 [▼
          "image_id" => 3
          "name" => "Anja Mačak"
          "extraData" => "Rezervacija: 0 | Popunjeno: 0%"
        ]
      ]
      "reservation" => 0
    ]
    31 => array:6 [▼
      "date" => "15/06 Ponedjeljak"
      "shift_id" => 31
      "shift_time" => "16:00 - 20:00"
      "shift_name" => "Popodnevna"
      "staff" => array:2 [▶]
      "reservation" => 0
    ]
  ]
  "2020-06-16" => array:2 [▶]
  "2020-06-17" => array:2 [▶]
  "2020-06-18" => array:2 [▶]
  "2020-06-19" => array:2 [▶]
]
```

Slika 28. Podatci za popunavanje kalendara u Twig-u

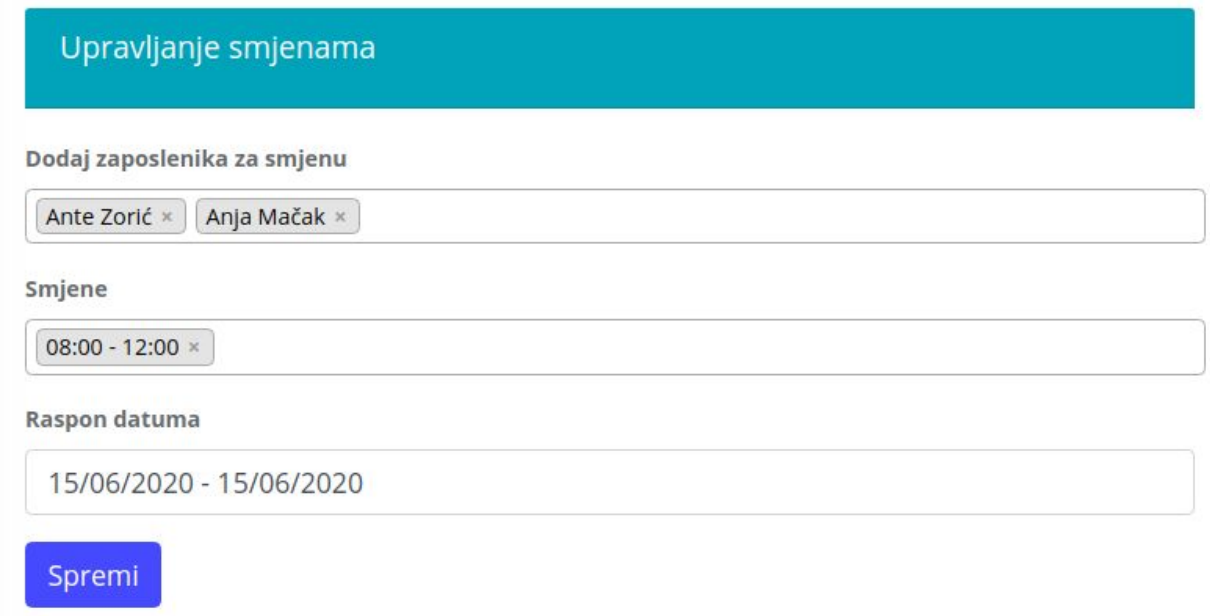
Generiranje smjene za radni dan

Ako se prvi put pristupa kalendaru, treba napraviti radno vrijeme za zaposlenike po datumu. Forma je izuzetno fleksibilna, uz različite načine kreacije:

- generiranje po danu
- generiranje po smjenama
- generiranje za cijeli tjedan

Symfony forma je jedna od najmoćnijih komponenti jer nudi mnoštvo opcija za implementaciju uz popratnu validaciju nad podatcima, radom s kolekcijama, prikazom poruka s greškom, rad s događajima te rad s JavaScript (npr. AJAX pozivi za dohvaćanje podataka). Komponenta daje da se trivijalno prikaže forma koja se može korisiti na više mjesta, validiranje i popunjavanje PHP objekta iz forme s podatcima na siguran način.

Pošto je forma usko vezana za entitet i kako ne postoji entitet koji definira jedan dan (datum), zaposlenika i smjenu, bilo je potreba proširiti funkcionalnost forme s prilagođenim rješenjem. Uključivao je provjeru i pripremu podataka koji bi se zatim poslali u formu. Na slici 30. je vidljiv primjer gdje za jedan datum potrebno napraviti izmjenu smjena zaposlenika, gdje su već postavljeni zaposlenici za taj dan:



Slika 29. Prikaz definiranja radnog vremena zaposlenika za jedan dan

Kada se na formi završi s uređivanjem i podatci se proslijede na server, ako prođe validaciju (najčešće se radi o HTML validaciji i validaciji na poljima entiteta) potrebno je napraviti provjeru nad onim podatcima koji su spremljeni (Slika 30.).

```php
/** For submitted date range generate new shift managements (also check for existing ones for unique constraint or required deletion) */
while ($from <= $to) {
    $day = date( format: "D", strtotime($from->format( format: "Y-m-d")));
    /** Only for working days of salon */
    if (in_array($day, $workingDays)) {
        /** @var Shift $shift */
        foreach ($shifts as $shift) {
            /** @var Staff $staff */
            foreach ($staffs as $staff) {
                if (!array_key_exists( key: $from->format( format: 'dmY').'_'.$shift->getId().'_'.$staff->getId(), $persistedShiftManagementArray)) {
                    $shiftManagement = new ShiftManagement();
                    $shiftManagement->setDate(new DateTime($from->format( format: "Y-m-d")));
                    $shiftManagement->setSalon($salon);
                    $shiftManagement->setShift($shift);
                    $shiftManagement->setStaff($staff);
                    $this->entityManager->persist($shiftManagement);
                } else {
                    $persistedShiftManagementArray[$from->format( format: 'dmY').'_'.$shift->getId().'_'.$staff->getId()]['delete'] = false;
                }
            }

        }
    }
    $from->modify( modify: '+1 day');
}

foreach ($persistedShiftManagementArray as $persistedCheckForDelete) {
    if ($persistedCheckForDelete['delete']) {
        $shiftManagement = $shiftManagementRepository->find($persistedCheckForDelete['id']);
        $this->entityManager->remove($shiftManagement);
    }
}
```

Slika 30. Provjera o postojećim smjenama za dan nakon *submit*-a forme za upravljanje smjenama

Sa slike 31. je vidljiv dio koji će za raspon datuma proslijeđen s forme napraviti provjeru nad **$persistedShiftManagementArray** nizom koji sadrži već postojeće definirane smjene. S time se osiguravamo da ne dođe do dupliciranja podataka, nepotrebnog brisanja i ponovnog kreiranja te fleksibilnost da se uvijek može kreirati za jedan dan, jednu smjenu ili cijeli tjedan bez da utječe na postojeće.

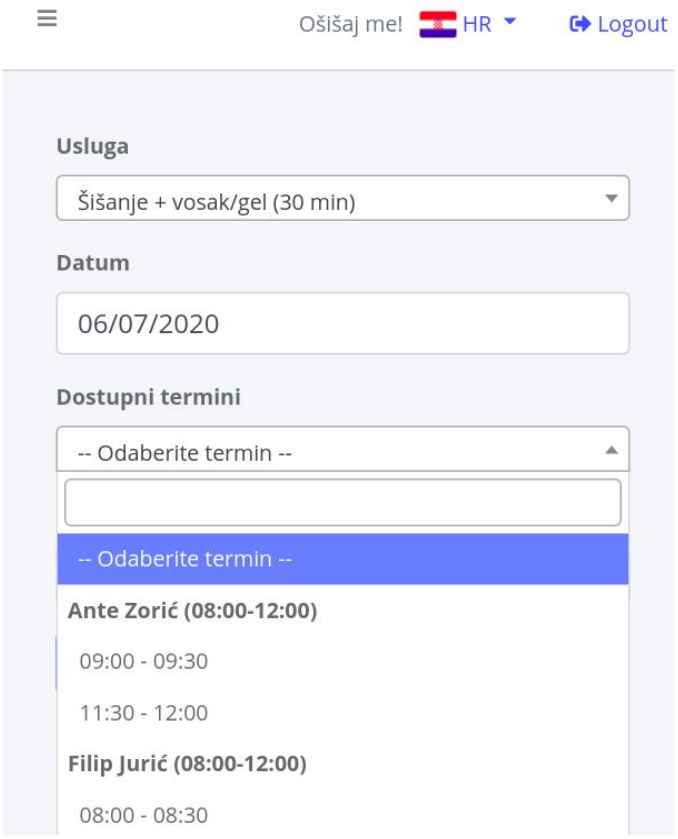
```php
/** Fetch all already exist shift management for date range from submitted form */
$persistedShiftManagementArray = [];
$persistedShiftManagement = $shiftManagementRepository->fetchSalonDateShiftStaffScheduledForDateRange($salon, $from, $to, $shifts);

foreach ($persistedShiftManagement as $persisted) {
    $persistedShiftManagementArray[$persisted['date']->format("dmY").'_'.$persisted['id'].'_'.$persisted['staffId']] = [
        'id' => $persisted['shiftManagementId'],
        'delete' => true,
    ];
}
```

Slika 31. Niz s postojećim smjenama zaposlenika po datumu

4.3.6. Upravljanje rezervacijama

Vlasnik salona ili zaposlenik ima pristup pregledu rezervacije i njenom uređivanju. S donje slike (Slika 32.) se vide za odabranu uslugu i datum, koji su slobodni termini i kod kojeg zaposlenika. Ako se uređuje postojeća, pod dostupne termine je uključen i termin same rezervacije. Logika izvođenja je dijelom bazirana kao i kod generiranja smjena, jer se kreira kalendar ali samo za odabrani datum. No ovaj put slobodni termini su grupirani po zaposlenicima. Na ovaj način (grupirani termin + zaposlenik) eliminirana je potreba za odvojenim poljima gdje bi se prvo birao zaposlenik pa bi se tek onda vidjelo kada postoji slobodni termin.



Slika 32. Uređivanje rezervacije na mobilnom uređaju

Polja s forma koja su ovisna o promjeni se popunjavaju pomoću AJAX poziva kod svake promjene. Tako npr. ako admin ima potrebu napraviti rezervaciju uvijek će imati one podatke koji su od salona.

4.4. Jednostavna rezervacija termina putem klijentske strane

Glavna motivacija za izradu klijentskog dijela aplikacije je jednostavnost korištenja. Omogućiti klijentu u što kraćem vremenu odabrati uslugu i termin za šišanje. Glavne značajke su:

- ugodno korisničko iskustvo (eng. UX - *User experience*)
- omogućiti odabir opcija umjesto pretrage i korištenja tipkovnice
- responzivnost stranice s obzirom na kojem se uređaju koristi
- minimalistički dizajn

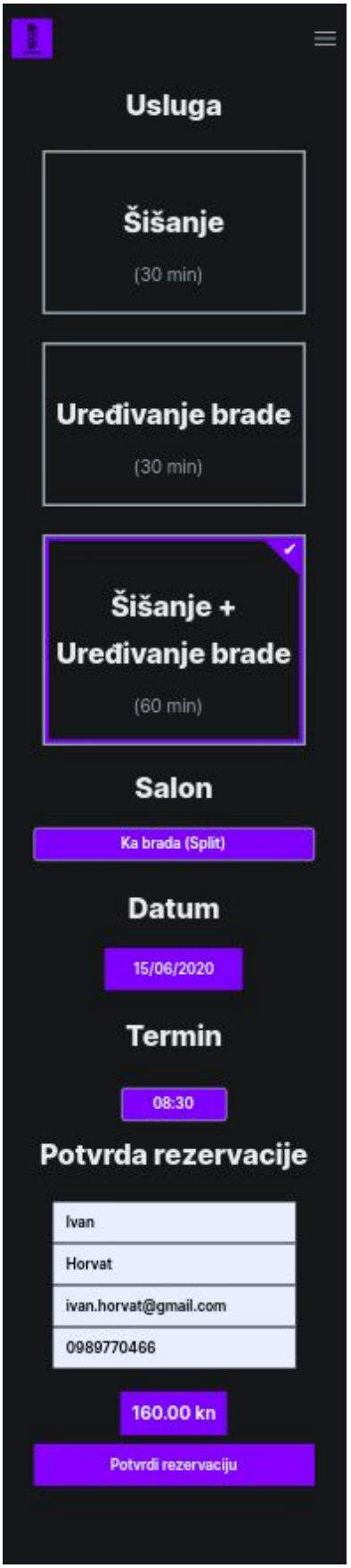
Korištenje segmentirani botuna umjesto padajućih izbornika za odabir usluge, salona, datuma i termina su dobar UX način kada korisnik treba unijeti podatke. S time se dodatno naglašavaju opcije koje se biraju. Odabir usluge radi na principu radio botuna koji je dizajniran kao kartica. Odmah se može jednostavno usporediti koje su razlike dok slučaj kod korištenja padajućeg izbornik bi bio otežan, a u ostalom zahtijeva minimalno dva klika, te pred izabrana opcija može zbuniti klijenta kao da je jedina.

Unos podataka funkcionira na vertikalnom načinu popunjavanja forme. Gdje se unosom ili odabirom jedne sekcije prebacuje na sljedeću, s time se može stvoriti savršeni *flow* unosa.

Radio botuni nisu uvijek savršen odabir, da se prikazuju jedan do drugog. Slučaj kada ima više ponuđenih opcija za odabir (preko 7, tada je bolje koristiti padajuće izbornike). Konkretan primjer ja odabir termina.

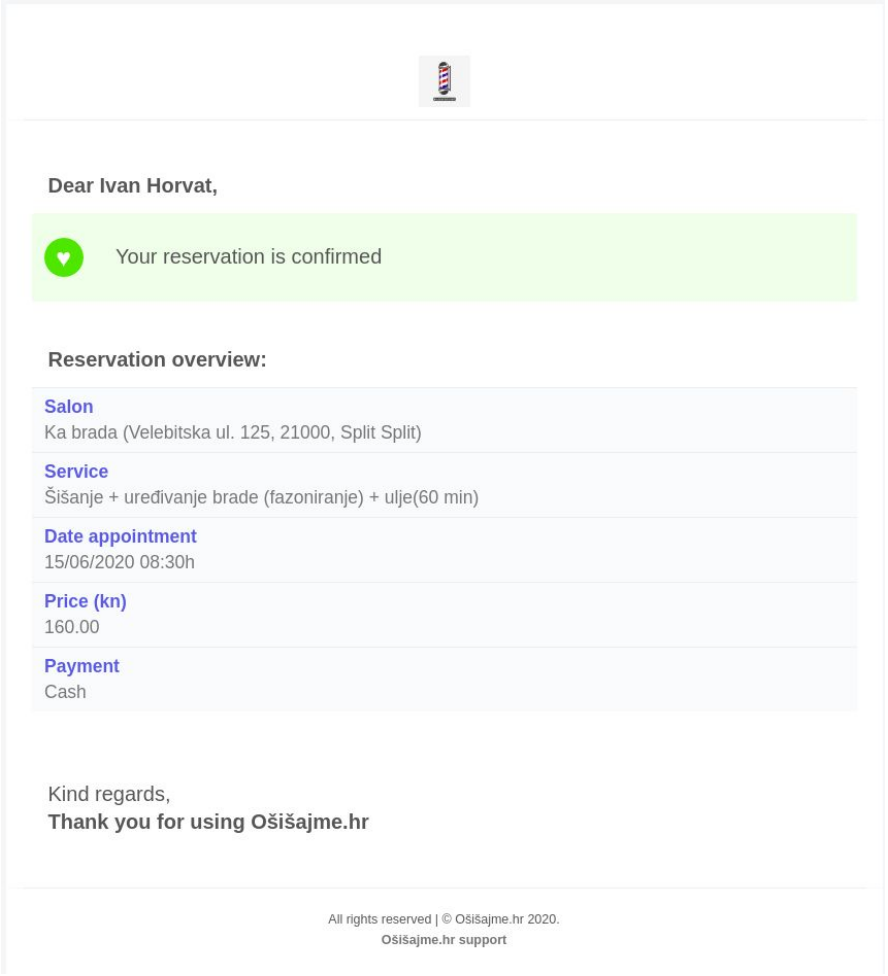
Korisnik može tako napraviti rezervaciju bez da je trebao koristiti tipkovnicu. Jedini slučaj korištenja tipkovnice bi bio kada sami korisnik nije prvotno registriran odnosno prijavljen. Tada na posljednem koraku, kod unosa vlastitih informacija potrebno je unijeti na tekstualna polja ime, prezime, email i kontakt broj. Ako je prijavljen na stranicu tada će se korisniku automatski popuniti polja s obzirom koja su unesena tijekom registracije.

Na sljedećoj slici (Slika 33.) je prikazan responzivni mobilni dizajn s cijelim sadržajem rezervacijske forme i unesenim podatcima korak prije same potvrde.



Slika 33. Mobilni prikaz klijentske forme za rezervaciju sa svim poljima

Klijent nakon što potvrdi rezervaciju, šalje mu se mail sa podatcima o napravljenoj rezervaciji (Slika 34.).



Slika 34. Email s potvrdom rezervacije

5. ZAKLJUČAK

Korištenjem različitih tehnologija i pridržavanja standarda pisanja programskog koda izrađena je aplikacija za korisnike koja nudi pretragu i rezervaciju termina za šišanje. Aplikacija sadrži i kompletnu administraciju za poslovne partnere tj. vlasnike salona i njihove zaposlenike. Administracija sadrži funkcionalnosti kao što su upravljanje radom zaposlenika i njihovih smjena, definiranje usluga i praćenja rezervacija odnosno sučelje za upravljanje rada salona.

Izgradnja ovakvog tipa aplikacije, koja je imala fokus na informatizaciji dijela poslovanja s kojim se mijenja način povezivanja klijenata sa salonom je rezultirala izgradnjom klijentske strane aplikacije. Korisnik može obaviti rezervaciju termina u najkraćem roku na najjednostavniji način te se time eliminirala potreba za bilo kojim drugačijim načinom dogovaranja rezervacije. Time se rasterećuje rad zaposlenika. Izgradnja administracije u konačnici će olakšati upravljanje radom salona i praćenje rezervacija.

Na osnovu priloženoga rada, može se smatrati da su uspješno implementirani postavljeni zahtjevi. Rad kao takav podoban je i za daljnja proširenja, kao npr. nove mogućnosti kod administrativne ili klijentske strane bile bi:

- omogućiti partneru kroz interaktivnu formu (princip rada “čarobnjaka”) popunjavanje informacije o salonu, dodavanje usluga, cijena, zaposlenika i generiranja kalendara
- prikaz salona po gradu povezanih pomoću linkova (klikom na jednog otvaraju se svi saloni) ili s obzirom na lokaciju korisnika pronaći najbliže salone što dodatno poboljšava UX

6. LITERATURA

[1] Vagrant https://en.wikipedia.org/wiki/Vagrant_(software) (23.06.2020.)

[2] Vagrant vs Docker https://dzone.com/articles/vagrant-vs-docker-which-is-better-for-software-dev (10.06.2020.)

[3] PuPHPet https://github.com/puphpet/puphpet (10.06.2020.)

[4] VirtualBox https://www.freecodecamp.org/news/what-is-a-virtual-machine-and-how-to-setup-a-vm-on-windows-linux-and-mac/ (10.06.2020.)

[5] PHP https://hr.wikipedia.org/wiki/PHP (10.06.2020.)

[6] Symfony https://symfony.com/what-is-symfony (24.06.2020.)

[7] Symfony versions https://symfony.com/releases (10.06.2020.)

[8] Doctrine https://www.doctrine-project.org/projects/doctrine-orm/en/2.7/reference/association-mapping.html (10.06.2020.)

[9] React (wiki) https://en.wikipedia.org/wiki/React_(web_framework) (14.06.2020.)

[10] React Hooks https://reactjs.org/docs/hooks-intro.html (14.06.2020.)

[11] React Context https://reactjs.org/docs/context.html (14.06.2020.)

[12] React state management with hooks and context https://medium.com/simply/state-management-with-react-hooks-and-context-api-at-10-lines-of-code-baf6be8302c

[13] Webpack Encore https://symfony.com/doc/current/frontend.html (14.06.2020.)

[14] jQuery https://en.wikipedia.org/wiki/JQuery (14.06.2020.)

[15] Easyadmin bundle https://github.com/EasyCorp/EasyAdminBundle (14.06.2020.)

[16] Open react template https://github.com/cruip/open-react-template (14.06.2020.)

[17] Admin LTE https://github.com/ColorlibHQ/AdminLTE (14.06.2020.)

[18] MySQL Workbench https://www.mysql.com/products/workbench/ (10.06.2020.)

[19] PHPStorm https://www.jetbrains.com/phpstorm (14.06.2020.)

[20] Symfony tutorials https://symfonycasts.com (10.06.2020.)

[21] Symfony Security component https://symfony.com/doc/current/security.htm (10.06.2020.)

[22] Symfony Security https://symfony.com/doc/current/security.htm (10.06.2020.)

[23] Datepicker design https://uxplanet.org/how-to-design-a-perfect-date-picker-control-7f47d1290c3a (10.06.2020.)

[24] Axios React https://alligator.io/react/axios-react/ (10.06.2020.)

[25] Making sense of React Hooks https://medium.com/@dan_abramov/making-sense-of-react-hooks-fdbde8803889 (10.06.2020.)

[27] React Handbook https://www.freecodecamp.org/news/the-react-handbook-b71c27b0a795/#single-page-applications (10.06.2020.)