Primjena i usporedna analiza razvojnih okvira za izradu jednostranične web aplikacije

Majdandžić, Dominik

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

PRIMJENA I USPOREDNA ANALIZA RAZVOJNIH OKVIRA ZA IZRADU JEDNOSTRANIČNE WEB APLIKACIJE

Diplomski rad

Dominik Majdandžić

Osijek, 2023.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 17.09.2023.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Dominik Majdandžić
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1140R, 13.10.2020.
OIB studenta:	79945495999
Mentor:	prof. dr. sc. Goran Martinović
Sumentor:	,
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	prof. dr. sc. Krešimir Nenadić
Član Povjerenstva 1:	prof. dr. sc. Goran Martinović
Član Povjerenstva 2:	izv. prof. dr. sc. Ivan Aleksi
Naslov diplomskog rada:	Primjena i usporedna analiza razvojnih okvira za izradu jednostranične web aplikacije
Znanstvena grana diplomskog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	U teorijskom dijelu diplomskog rada potrebno je objasniti izazove i mogućnosti u razvoju jednostraničnih web aplikacija, te ih analizirati s gledišta aktualnih metodologija razvoja programske podrške, programske arhitekture, programskih alata i jezika. Poseban naglasak treba dati na razvojni okvir Vue.js, predložak programske arhitekture MVVM i biblioteku React koji omogućuju razvoj jednostraničnih web rješenja. Nadalje, treba definirati postavke, načine primjene, te mjerila usporedbe i analize navedenih razvojnih okolina. Uz to, treba odrediti funkcionalne i nefunkcionalne zahtjeve, model i arhitekturu web aplikacije koja na temelju sustava stvaranja
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	17.09.2023.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 27.09.2023.

Ime i prezime studenta:	Dominik Majdandžić
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1140R, 13.10.2020.
Turnitin podudaranje [%]:	13

Ovom izjavom izjavljujem da je rad pod nazivom**: Primjena i usporedna analiza razvojnih okvira za izradu jednostranične web aplikacije**

izrađen pod vodstvom mentora prof. dr. sc. Goran Martinović

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

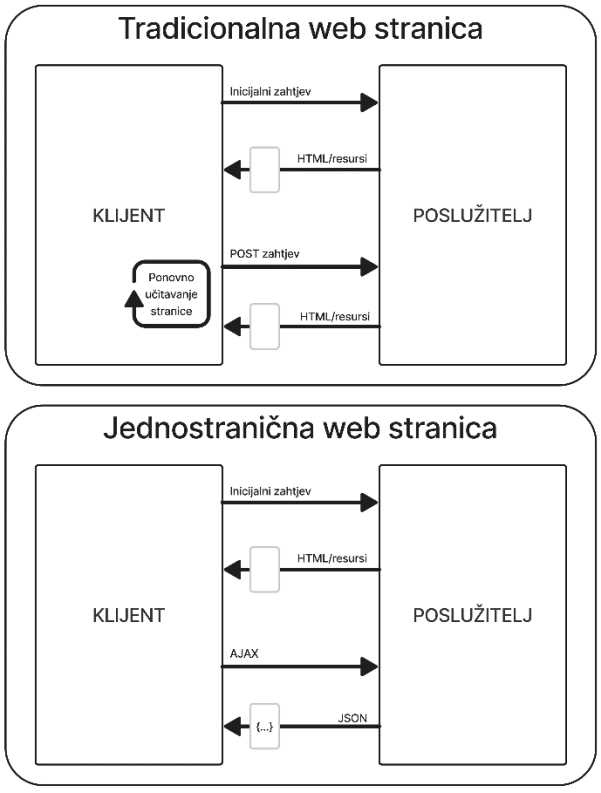
Zadnjih godina, koncept jednostraničnih web aplikacija sve je popularniji. Jednostranična web aplikacija je web aplikacija koja učitava samo jednu stranicu, a zatim dohvaća nove podatke s poslužitelja kojima ažurira sadržaj stranice. Njihova mogućnost dinamičkog mijenjanja elemenata stranice, bez ponovnog učitavanja cijele stranice, čini korisničko iskustvo boljim. Kako bi se razvile takve stranice koriste se JavaScript razvojna okruženja. S obzirom na sve veću popularnost jednostraničnih stranica, stalno se razvijaju takva nova razvojna okruženja. Time se dolazi do pitanja, koje je razvojno okruženje najbolje za razvoj jednostraničnih web aplikacija.

Cilj ovoga rada je usporediti dva razvojna okruženja za izradu jednostraničnih web aplikacija. Odabrana su dva od popularnijih razvojnih okruženja: React i Vue.js. U oba razvojna okruženja će se izraditi identična jednostranična web aplikacija za naručivanje zdrave hrane iz restorana. Izrađena jednostranična web aplikacija će omogućiti korisniku filtriranje jela iz restorana na temelju odabranih opcija. Osim toga, korisnik će moći dodati jela u košaricu te obaviti narudžbu. Također, web aplikacija će imati dio dostupan samo administratoru gdje će administrator moći uređivati dostupna jela. Za izrađene jednostranične web aplikacije će se prikazati i objasniti programski kod kojim su implementirane neke od funkcionalnosti jednostraničnih web aplikacija. Također, usporedit će se i analizirati performanse izrađenih jednostraničnih web aplikacija.

U drugom poglavlju detaljnije je objašnjeno što su jednostranične web aplikacije zajedno s metodologijama i tehnologijama koje se vežu uz njih. Također, pokazani su primjeri rješenja jednostraničnih web aplikacija. U trećem poglavlju navedeni su funkcionalni i nefunkcionalni zahtjevi za jednostraničnu web aplikaciju koja se izrađuje, a pokazat će se model i korisničko sučelje izrađene aplikacije. U četvrtom poglavlju prikazan je i objašnjen React i Vue.js programski kod izrađenih jednostraničnih web aplikacija. Na kraju, u petom poglavlju, obavljena su mjerenja i dana analiza performansi izrađenih inačica jednostraničnih web aplikacija.

2. JEDNOSTRANIČNE WEB APLIKACIJE I OSVRT NA STANJE U PODRUČJU

Jednostranična web aplikacija [1] (engl. *single-page application*) ili skraćeno SPA, je implementacija web aplikacije koja učitava samo jedan web dokument. Zatim ažurira sadržaj tog dokumenta, pomoću JavaScript API-ja (engl. *Application Programming Interface*) kao što su XMLHttpRequest i Fetch, kada je potrebno prikazati neki novi sadržaj. Jednostranična web aplikacija može dohvatiti kompletan HTML (engl. *HyperText Markup Language*), CSS (engl. *Cascading Style Sheets*) i JavaScript aplikacije pri inicijalnom učitavanju ili može dinamički učitavati potrebne resurse kao odgovor na interakciju korisnika ili druge događaje [2]. Nasuprot tome, ostale web aplikacije (MPA, engl. *multi-page application*) predstavljaju korisniku početnu stranicu koja je povezana s ostalim dijelovima aplikacije na zasebnim HTML stranicama, što znači da korisnik mora čekati da se nova stranica učita svaki put kada uputi novi zahtjev. Usporedba tradicionalnih web stranica i jednostraničnih web stranica je prikazana na slici 2.1 po uzoru iz [3].



Sl. 2.1. Životni ciklus MPA i SPA [3]

Jednostranične web aplikacije koriste HTML5 i Ajax (engl. *Asynchronous JavaScript and XML*) kako bi omogućile glatke i dinamične odgovore na zahtjeve korisnika, omogućujući da se sadržaj ažurira odmah nakon što korisnik izvrši radnju. Nakon što se stranica učita, interakcije s

poslužiteljem (engl. *server*) se odvijaju putem Ajax poziva i vraćaju se podatci, većinom u JSON (engl. *JavaScript Object Notation*) formatu, kako bi se stranica ažurirala bez ponovnog učitavanja.

2.1. Prednosti i nedostatci jednostraničnih web aplikacija

Jednostranične web aplikacije su brže od tradicionalnih web aplikacija [4]. Kao što je prije navedeno, većina resursa kao što su HTML, CSS ili JavaScript se učitavaju samo jednom tijekom životnog ciklusa aplikacije. Šalju se jedino podatci između klijenta i poslužitelja. To je izuzetna prednost koja ubrzava učitavanje stranice i smanjuje vrijeme čekanja za korisnike.

Također pružaju bolje cjelokupno korisničko iskustvo [5]. Korisnici ne moraju gledati kako se učitava nova stranica, jer se mijenja samo sadržaj, a ne stranica. Time su jednostranične web aplikacije posebno prikladne za mobilna okruženja. Zbog toga što nema potrebu za osvježavanjem stranice, iskustvo je kontinuirano i navigacija je općenito brža.

Nadalje, jednostranične web aplikacije mogu učinkovito spremati lokalne podatke u priručnu memoriju (engl *cache*). Može jednim zahtjevom dohvatiti i spremiti sve potrebne podatke, nakon čega web aplikacija može funkcionirati bez komunikacije s poslužiteljem ili čak bez konekcije na internet. Ako korisnik ima lošu povezanost na internet, lokalni podatci se mogu sinkronizirati s poslužiteljem kada veza to dopušta.

Naravno, jednostranične web aplikacije imaju svoje nedostatke. Jedan od problema je taj da nema povratka natrag na prijašnju stranicu. Web preglednici spremaju povijest koja se može koristiti za navigaciju naprijed-natrag kroz prijašnje otvorene stranice. Gumb za odlazak natrag šalje korisnika na prijašnju stranicu, a ne u prijašnje stanje stranice. Zbog toga jednostranične web aplikacije gube funkcionalnost povratka natrag. Naravno, taj problem se može riješiti izgradnjom kompleksnije jednostranične web aplikacije (npr. može se koristiti History API).

Kod jednostraničnih web aplikacija, web preglednik odrađuje većinu poslova. Time preglednik koristi mnogo resursa što može biti problem kod mobilnih uređaja. Osim toga, često je potrebno da korisnici imaju što noviju inačicu preglednika, inače se riskira mogućnost da neke funkcionalnosti neće biti podržane.

Korištenjem samo jedne stranice umjesto više stranica može naškoditi optimizaciji stranice (SEO, engl. *Search Engine Optimization*), koji određuje poziciju stranice u rezultatima pretraživanja na pregledniku. Razlog je taj što za indeksiranje stranice potrebno je baviti se JavaScript-om umjesto HTML-om. Trenutno samo Googlebot (Google-ov program za indeksiranje web stranica) može generirati JavaScript stranice, za razliku od Bing-a ili Yahoo-a,

no to ne znači da se ne može postići dobar SEO. Najčešći načini postizanja dobrog SEO-a bi bili: učitavanje na strani poslužitelja, izomorfni JavaScript, predučitavanje (engl. *pre-rendering*) ili otkrivanje značajki (engl. *feature detection*).

Također je potrebno napomenuti da su jednostranične web aplikacije sklonije napadima skriptiranja na više mjesta (engl. *cross-site scripting attack*, skraćeno XSS). Koristeći XSS, hakeri mogu lagano ubaciti skripte na strani klijenta u web aplikaciju. Osim toga, jednostranične web aplikacije imaju veću vjerojatnost da će izložiti osjetljive podatke svim korisnicima.

2.2. MV* obrasci programske arhitekture

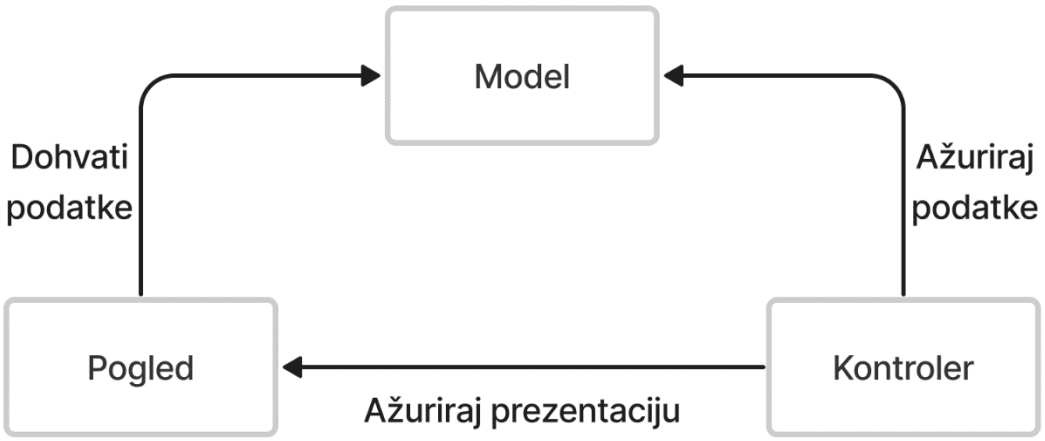
Ideja MV* (engl. *model-view-**) obrazaca je ta da se razdvoji model, pogled i logika koja ih spaja, tj. razdvojiti odgovornosti spremanja podataka, vizualizacije podataka te upravljanja i obrade podataka.. Model je zadužen za spremanje podataka aplikacije, a pogled za prikazivanje tih podataka. Prateći takav princip vodi do jasnog dizajna koji je lagan za održavanje i koji omogućuje efektivno raspoređivanje rada na različitim dijelovima između programera [6]. Postoje tri glavna MV* modela: MVC (engl. *model-view-controller*), MVP (engl. *model-view-presenter*) i MVVM (engl. *model-view-viewmodel*). U daljnjim podnaslovima će se ta tri modela detaljnije pojasniti u kontekstu razvijanja web aplikacija.

2.2.1. MVC

MVC obrazac je osmišljen 1980-ih. Inicijalno se koristio za projektiranje i razvijanje desktop aplikacija s bogatim grafičkim korisničkim sučeljima, ali danas se koristi i pri razvoju web aplikacija. Cilj ovog modela je odvojiti logiku domene (engl. *domain logic*) od logike prezentacije (engl. *presentation logic*) u tri komponente s različitim odgovornostima. Model za pohranu podataka kontroler za obrađivanje korisničkih radnji i pogled za grafičko korisničko sučelje. Vizualni prikaz MVC obrasca je prikazan na slici 2.2 po uzoru iz [7].

MVC se sastoji od tri ključna dijela:

- **Model** – Modeli upravljaju podacima za aplikaciju. Oni se ne bave korisničkim sučeljem niti prezentacijskim slojevima, već umjesto toga predstavljaju jedinstvene oblike podataka koje aplikacija može zahtijevati.
- **Pogled** – Pogledi prikazuju podatke korisniku i omogućuju interakciju s korisnikom.
- **Kontroler** – Kontroler je posrednik između modela i pogleda. On ažurira model kada korisnik manipulira pogledom.

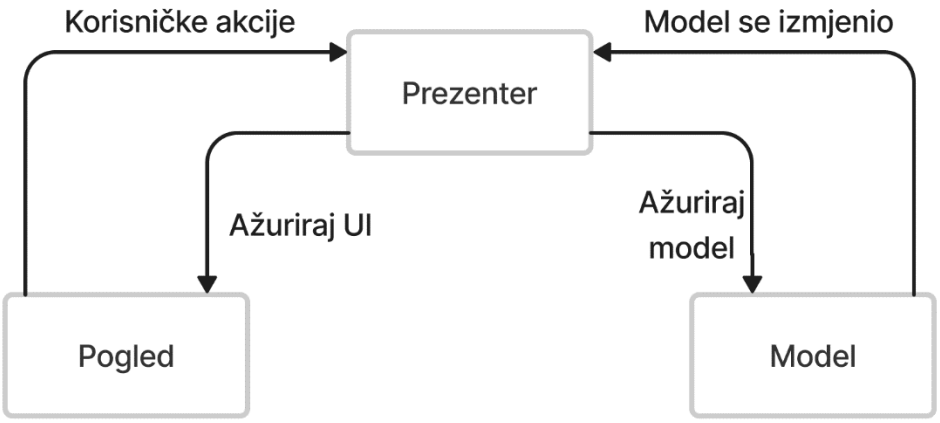


Sl. 2.2. Obrazac MVC [7]

MVC obrazac ubrzava paralelni razvoj [8]. To znači da će tijekom razvijanja web aplikacije jedan programer moći raditi na pogledu dok drugi radi na kontroleru. Zbog toga što je poslovna logika odvojena od podataka, moguće je izraditi više pogleda za jedan model bez dupliciranja koda. MVC vraća podatke bez primjene formatiranja, što znači da se iste komponente mogu koristiti i pozivati koristeći bilo koje sučelje.

2.2.2. MVP

MVP obrazac je izveden iz MVC-a i prikazan je na slici 2.3 po uzoru iz [7]. On se usredotočuje na poboljšanje prezentacijske logike. Model i pogled funkcioniraju isto kao i kod MVC-a, ali umjesto kontrolera koristi se prezenter. Prezenter preuzima potpunu odgovornost za rješavanje svih događaja korisničkog sučelja u ime pogleda. Pogled predaje unos korisnika prezenteru koji obrađuje dobivene podatke uz pomoć modela te prosljeđuje rezultate natrag pogledu.

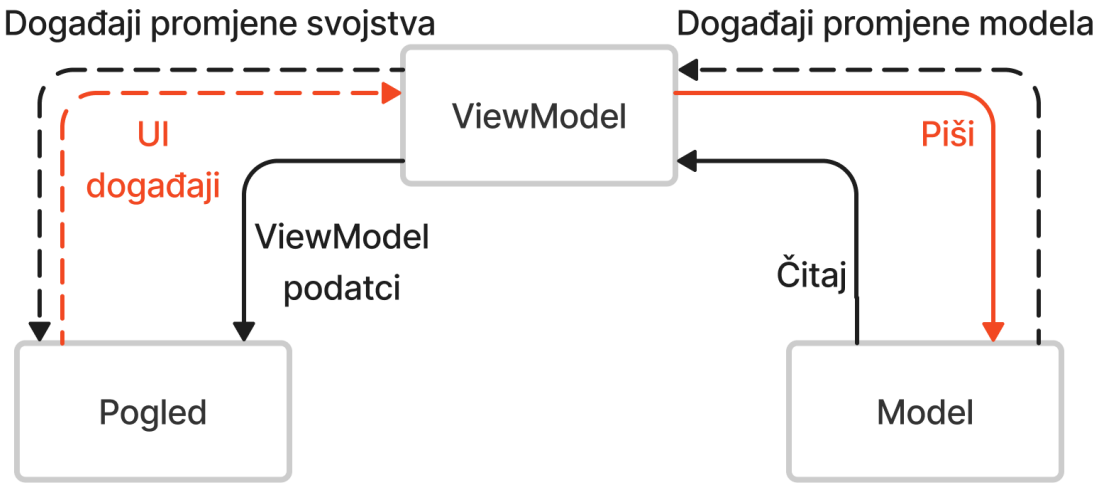


Sl. 2.3. Obrazac MVP [7]

MVP se općenito najčešće koristi u aplikacijama na razini poduzeća gdje je potrebno ponovno iskoristiti što više prezentacijske logike. Aplikacijama s vrlo složenim pogledima i velikom količinom interakcije s korisnicima MVC ne odgovara u potpunosti jer rješavanje tog problema može značiti oslanjanje na više kontrolera. Kod MVP-a se sva ta složena logika može inkapsulirati u prezenter, što može znatno pojednostaviti održavanje. Kako su MVP pogledi definirani kroz sučelje, a sučelje je jedina točka kontakta između sustava i pogleda (osim prezentera), ovaj obrazac također omogućuje programerima da pišu prezentacijsku logiku bez potrebe za čekanjem dizajnera da proizvede izgled i grafike za aplikaciju. Ovisno o implementaciji, MVP može biti lakši za automatizirano jedinično testiranje nego MVC. Česti razlog tome je taj što se prezenter može koristiti za potpunu simulaciju korisničkog sučelja i stoga se može jedinično testirati neovisno o drugim komponentama.

2.2.3. MVVM

MVVM, prikazan na slici 2.4 po uzoru iz [7], je obrazac temeljen na MVC-u i MVP-u koji pokušava jasnije odvojiti razvoj korisničkog sučelja od razvoja poslovne logike i ponašanja aplikacije. U tu svrhu, mnoge implementacije MVVM-a koriste deklarativne veze podataka (engl. *data binding*) kako bi omogućile odvajanje rada na pogledima od drugih slojeva.



Sl. 2.4. Obrazac MVVM [7]

ViewModel se može smatrati specijaliziranim kontrolerom koji djeluje kao pretvarač podataka. Mijenja informacije modela u informacije pogleda, prosljeđujući naredbe iz pogleda u model. Na primjer, ako model sadrži atribut datuma u *unix* formatu. Umjesto da modeli budu svjesni korisničkog pogleda na datum (ljudima čitljiv format, npr. dan/mjesec/godina) kako bi pretvorili atribut u njegov format prikaza, model jednostavno sadrži sirovi podatak. Pogled sadrži

formatirani datum, a *ViewModel* djeluje kao posrednik između njih. Ukratko *ViewModel* se nalazi iza sloja korisničkog sučelja. On izlaže podatke iz modela koji su potrebni pogledu i može se gledati kao izvor koji pogledi koriste za podatke i radnje.

2.3. Metode učitavanja sadržaja web stranica

Učitavanje sadržaja na webu se može izvesti na mnogo načina. Odluka o tome kako i gdje dohvatiti i prikazati sadržaj je ključna za izvedbu aplikacije. Dostupni razvojni okviri i biblioteke se mogu koristiti za implementaciju različitih metoda učitavanja, kao što su učitavanje na strani klijenta (engl. *client-side rendering*) i učitavanje na strani poslužitelja (engl. *server-side rendering*).

2.3.1. Učitavanje na strani klijenta

Učitavanje na strani klijenta je postalo popularno nakon uvođenja JavaScript razvojnih okvira koji su uključivali taj stil razvoja (npr. React, Angular, Vue.js itd.). Kod učitavanja na strani klijenta [9] poslužitelj prikazuje samo kostur HTML spremnika za stranicu. Logikom, dohvaćanjem podataka, izradom predložaka i usmjeravanjem potrebnim za prikaz sadržaja na stranici upravlja JavaScript programski kod koji se izvršava na pregledniku, tj. klijentu.

Proces učitavanja na strani klijenta:

- Korisnik šalje zahtjev za pristup web stranici na pregledniku koristeći URL (engl. *uniform resource locators*).
- Poslužitelj poslužuje statičke datoteke (HTML i CSS) pregledniku klijenta na njegov prvi zahtjev za web stranicom.
- Preglednik će prvo preuzeti HTML sadržaj, a zatim JavaScript. Ovaj proces učitavanja se uglavnom odvija kada korisnik vidi simbole za učitavanje koje je definira web programer. To znači da stranica još uvijek nije vidljiva korisniku.
- Nakon što preglednik preuzme JavaScript, sadržaj se dinamički generira na njemu.
- Web sadržaj postaje vidljiv te se klijent može kretati i komunicirati s web stranicom.

Navedenim postupkom, većina logike aplikacije se izvršava na klijentu i komunicira s poslužiteljem putem API poziva za dohvaćanje ili spremanje podataka. Tako se gotovo cijelo korisničko sučelje generira na strani klijenta. Cijela web aplikacija se učitava na prvi zahtjev. Kako se korisnik kreće klikom na poveznice, poslužitelju se više ne šalje nikakvi zahtjev za učitavanje stranica. Programski kod se izvodi na klijentu kako bi se promijenio pogled.

Učitavanje na strani klijenta omogućuje jednostranične web aplikacije koje podržavaju navigaciju bez osvježavanja stranice i pruža izvrsno korisničko iskustvo (engl. *user experience*). Budući da su podatci koji se obrađuju za promjenu pogleda ograničeni, usmjeravanje između stranica je općenito brže, zbog čega se aplikacija čini osjetljivijom (engl. *responsive*). Učitavanje na strani klijenta također omogućuje programerima da postignu jasnu razliku između klijentskog i poslužiteljskog programskog koda.

2.3.2. Učitavanje na strani poslužitelja

Učitavanje na strani poslužitelja je jedna od najstarijih metoda učitavanja web sadržaja. Operacijama povezivanja i dohvaćanja upravlja poslužitelj. HTML potreban za formatiranje sadržaja također se generira na poslužitelju. Kod učitavanja na strani poslužitelja, svaki zahtjev se tretira nezavisno i poslužitelj će ga obraditi kao novi zahtjev. Čak i ako izlaz dva uzastopna zahtjeva nije jako različit, poslužitelj će ga obraditi i generirati od nule. Budući da jedan poslužitelj koristi više korisnika, mogućnost obrade dijele svi aktivni korisnici u određenom trenutku.

Kod učitavanja na strani poslužitelja, korisnik šalje zahtjev poslužitelju. Poslužitelj obrađuje HTML, CSS i JavaScript na zahtjev i isporučuje potpuno popunjenu stranicu korisnikovom pregledniku. Za razliku od učitavanja na strani klijenta, proces učitavanja se ponavlja svaki put kada korisnik odradi radnju da posjeti drugu stranicu. Poslužitelj će svaki put poslužiti stranicu na zahtjev. Preglednik neprestano postavlja zahtjeve poslužitelju za svaku novu stranicu i zahtjev.

Nedostatak ovog pristupa je to što zahtjeva puno resursa i odgađa isporuku sadržaja korisniku te povećava vrijeme učitavanja stranice u usporedbi s jednostraničnim web aplikacijama. To je zato što poslužitelj mora više puta učitati dinamički sadržaj, dok je kod učitavanja na strani klijenta sadržaj statičan i prikazuje se gotovo trenutno pri ponovnom učitavanju stranice. Veliki promet korisnika može uzrokovati greške u vezi. To se događa kada previše korisnika pokušava pristupiti resursima, a poslužitelj nema dovoljno računalne snage za obradu svih zahtjeva.

2.4. Primjeri rješenja jednostraničnih web aplikacija

Postoje brojna rješenja jednostraničnih web aplikacija, ali će se u ovome poglavlju fokusirati na rješenja koja imaju funkcionalnosti filtriranja kroz skup podataka i njihov prikaz.

2.4.1. Wolt

Wolt [10] je aplikacija za dostavu hrane i drugih proizvoda. Postoje mobilna i web inačica. Obje inačice omogućuju naručivanje hrane iz dostupnih restorana i naručivanje različitih

proizvoda iz dostupnih trgovina. Iako se stranica bavi dostavom hrane, nema previše opcija filtriranja. Kroz jela se uglavnom filtrira preko odabira kategorije jela, npr. deserti.

2.4.2. Glovo

Glovo [11] je, kao i Wolt, aplikacija za dostavu hrane i ostalih proizvoda. Također postoje mobilna i web inačica. Filtriranje proizvoda je isto ograničeno uglavnom na kategorije jela ili proizvoda.

2.4.3. Yummly

Yummly [12] je web stranica za pretraživanje i preporuku recepata. Iako nema mogućnosti naručivanja hrane, ima veće mogućnosti filtriranja. Osim što se može pretraživati recepte po kategorijama, može se i pretraživati po sastojcima i različitim dijetama. To omogućuje korisnicima da lakše pronađu hranu koja odgovara njihovim sklonostima i potrebama.

2.4.4. Airbnb

Airbnb [13] je web stranica za iznajmljivanje nekretnina. Iako ova stranica nije vezana uz prehranu, izrađena je kao jednostranična web aplikacija te ima mogućnosti filtriranja nekretnina u ovisnosti o zahtjevima korisnika kao što su: raspon cijene, broj soba, broj kreveta, vrsta nekretnine, klima, internet, itd.

2.5. Raspoloživi razvojni okviri za izradu jednostraničnih web aplikacija

2.5.1. Vue.js

Vue.js [14] je progresivni JavaScript razvojni okvir koji se koristi za razvoj interaktivnih web sučelja. On se usredotočuje na sloj pogleda. Lako se može integrirati u velike projekte za *frontend* razvoj. Instalacija Vue.js-a je prilično jednostavna i početnici ga mogu lako razumjeti i početi graditi vlastita korisnička sučelja. Vue.js [15] je okvir i ekosustav koji pokriva većinu zajedničkih značajki potrebnih za razvoj *frontend*-a, ali web je iznimno raznolik i stvari koje se grade na njemu se mogu drastično razlikovati u obliku i veličini. Imajući to na umu, Vue.js je dizajniran da bude fleksibilan i inkrementalno prilagodljiv.

Ovisno o slučaju korištenja, može se koristiti na različite načine, kao što su:

- Poboljšanje statičnog HTML-a bez koraka izrade
- Ugradnja (engl. *embedding*) kao web komponente na bilo koju stranicu
- Jednostranične web aplikacije

- *Fullstack*, učitavanje na strani poslužitelja
- *Jamstack*, generiranje statične stranice
- Ciljanje stolnih računala, mobilnih uređaja, WebGL-a (engl. *web graphics library*) pa čak i terminala

2.5.2. React

React [16] je JavaScript biblioteka za razvoj korisničkog sučelja web aplikacije. React je razvio i objavio Facebook. Facebook kontinuirano radi na React biblioteci i poboljšava je ispravljanjem grešaka (engl. *bugs*) i uvođenjem novih značajki.

React pruža minimalan i solidan skup značajki za pokretanje web aplikacije. React zajednica nadopunjuje React biblioteku pružanjem velikog skupa gotovih komponenti za razvoj web aplikacije. Zajednica također nudi napredne koncepte poput upravljanja stanjem (engl. *state management*), usmjeravanje, itd. React omogućuje sastavljanje kompleksnih korisničkih sučelja koristeći male i izolirane dijelove programskog koda koji se nazivaju komponentama (engl. *components*).

Zahvaljujući arhitekturi koja se temelji na komponentama, održavanje jednostraničnih web aplikacija izrađenih pomoću React-a je relativno jednostavno. Stranica temeljena na React-u uključuje virtualni DOM. On omogućuje razvojnom timu da prati i ažurira izmjene bez utjecaja na ostale dijelove stabla, čime se povećava fleksibilnost aplikacije. React je fleksibilniji od ostalih razvojnih okvira zbog svojih samostalnih biblioteka koje omogućuju dobro vrijeme odziva i čine ga odličnim za razvoj jednostraničnih web aplikacija. Također omogućuje distribuciju opterećenja s poslužitelja na klijenta jer obje strane koriste React.

2.5.3. Angular

Angular [17] je platforma i razvojni okvir za izgradnju jednostraničnih web aplikacija koristeći HTML i TypeScript. Angular je napisan u TypeScript-u. Implementira osnovne i opcionalne funkcionalnosti kao skup TypeScript biblioteka koje se uvoze (engl. *import*) u aplikacije.

Arhitektura Angular aplikacije se oslanja na određene temeljne koncepte. Osnovni dijelovi za izgradnju su Angular komponente koje su organizirane u NgModul-e. NgModul-i prikupljaju povezani programski kod u funkcionalne skupove. Angular aplikacija je definirana skupom NgModul-a. Aplikacija uvijek ima barem korijenski modul koji omogućuje podizanje sustava. Komponente definiraju poglede koji su skupovi elemenata zaslona, koje Angular može birati i

mijenjati prema logici i podatcima programa. Komponente koriste usluge koje pružaju specifične funkcije, koje nisu izravno povezane s pogledima. Moduli, komponente i usluge su klase koje koriste dekoratere (engl. *decorators*). Ovi dekorateri označavaju svoj tip i pružaju meta-podatke koji govore Angular-u kako ih koristiti. Meta-podatci za klasu komponente pridružuju se predlošku koji definira pogled. Predložak kombinira obični HTML s Angular direktivama i obvezujućim oznakama koje Angular-u omogućuju izmjenu HTML-a prije nego što ga pogled prikaže. Meta-podatci za klasu usluge pružaju informacije koje Angular treba da bi ih učinio dostupnima komponentama pomoću ubrizgavanja ovisnosti (engl. *dependency injection*). Komponente aplikacije obično definiraju mnogo pogleda raspoređenih hijerarhijski. Angular pruža uslugu usmjerivača (engl. *router*) koji pomaže definirati navigacijske putove između pogleda.

Angular-ova značajka povezivanja podataka eliminira veći dio programskog koda koji bi programer inače trebao napisati. Dakle, izrada jednostranične web aplikacije pomoću Angular-a zahtijeva manje redova programskog koda. Aplikacije izgrađene koristeći Angular imaju brzo vrijeme učitavanja. To čini mogućim komponenta usmjerivača, isporukom automatskog dijeljenja koda. Omogućuje korisnicima učitavanje samo programskog koda koji se zahtjeva za pogled.

2.5.4. Meteor

Meteor [18] je *fullstack* JavaScript platforma za razvoj modernih web i mobilnih aplikacija. Meteor uključuje ključni skup tehnologija za izgradnju reaktivnih aplikacija povezanih s klijentom, alat za izgradnju i odabrani skup paketa iz zajednice Node.js i općenito JavaScript.

Meteor omogućuje razvoj koristeći JavaScript u svim okruženjima: poslužitelj aplikacija, web preglednik i mobilni uređaj. Meteor koristi podatke na žici (engl. *data on the wire*), što znači da poslužitelj šalje podatke, ne HTML, a klijent ih generira.

Omogućuje savršenu sinkronizaciju, što znači da se svi podatci sinkroniziraju u stvarnom vremenu, čime suradnja na projektu postaje lakša. Ima značajku LiveReload koja omogućuje pregled i analizu svih učinjenih promjena bez osvježavanja stranice. Meteor je integriran s Galaxy-em, što je PaaS (engl. *platform as a service*) rješenje za posluživanje aplikacija s Meteor. Stoga, sve što je potrebno za implementaciju aplikacije je prijava na Galaxy i dodavanje varijabli okruženja.

2.5.5. Backbone.js

Backbone.js [19] je MVC razvojni okvir za web aplikacije koji pruža strukturu za aplikacije pune JavaScript-om. To se postiže opskrbom modela s prilagođenim događajima i vezanjem ključ-vrijednost (engl. *key-value binding*), pogledima koji koriste deklarativno rukovanje događajima i zbirkama s bogatim API-em. Sve ove značajke su povezane s prevladavajućom aplikacijom pomoću RESTful (engl. *representational state transfer*) JSON sučelja.

Backbone.js se može definirati kao iznimno lagana biblioteka koja omogućuje stvaranja lako održivih korisničkih sučelja. Vrlo dobro funkcionira s postojećim JavaScript bibliotekama. On je iznimno koristan za pojednostavljenje postupka za razvoj interaktivnih i složenih aplikacija vođenih podatcima (engl. *data driven*). Backbone.js nudi zgodno rješenje za odvajanje podataka od prezentacije, strukturiranjem programskog koda i cijepanjem u semantički smislene JavaScript datoteke.

Backbone.js uključuje četiri glavne klase:

- **Model** – Modeli su temeljni dio svih JavaScript aplikacija. Modeli sadrže interaktivne podatke uz elemente logike koji okružuju podatke, kao što su provjere valjanosti, konverzije, kontrola pristupa i računalna svojstva. Backbone.js model se može proširiti metodama specifičnim za domenu i nudi standardni skup funkcionalnosti za upravljanje promjenama.
- **Zbirka** – Zbirke u Backbone.js su uglavnom nizovi modela.
- **Pogled** – Pogled osluškuje događaje koje bacaju DOM, zbirke i modeli.
- **Kontroler** – Kontroleri se koriste za stvaranje aplikacija s podatcima o stanju i oznakama uz pomoć *hashbang*-ova.

2.5.6. Ember.js

Ember.js [20] je besplatni JavaScript razvojni okvir na strani klijenta koji se koristi za razvoj web aplikacija. Omogućuje izgradnju JavaScript aplikacija na strani klijenta, pružajući cjelovito rješenje koje sadrži upravljanje podatcima i tijek aplikacije. Koristi MVC obrazac. U Ember.js-u, putanja se koristi kao model, predložak upravljača kao pogled i kontroler manipulira podatcima u modelu.

Ember.js sadrži sljedeće temeljne koncepte:

- **Usmjerivač i rukovatelj putanjama** – Ember.js koristi usmjerivač za preslikavanje URL-a na rukovatelj putanje. Usmjerivač uspoređuje postojeći URL

s putanjom koja se zatim koristi za učitavanje podataka, prikaz predložaka i postavljanje stanja aplikacije. Rukovatelj putanjama (engl. *route handler*) izvodi sljedeće radnje: pruža predložak, definira model koji će biti dostupan predlošku i ako korisnik nema dopuštenje da posjeti određeni dio aplikacije, usmjerivač će ga preusmjeriti na novu putanju.

- **Predlošci** – Predlošci su moćno korisničko sučelje za krajnje korisnike. Ember.js predložak pruža izgled korisničkog sučelja aplikacije koja koristi sintaksu *Handlebars* predložaka. Gradi *frontend* aplikaciju, koja je kao obični HTML.
- **Model** – Rukovatelji putanje prikazuju model koji sadrži informacije na web poslužitelju. Manipulira podacima pohranjenim u bazi podataka. Model je jednostavna klasa koja proširuje funkcionalnost Ember.js podataka.
- **Komponente** – Komponenta kontrolira ponašanje korisničkog sučelja koje uključuje dva dijela: predložak napisan u JavaScript-u i izvornu datoteku koja je napisana u JavaScript-u koja osigurava ponašanje komponenti.

3. MODEL I IZGLED PREDLOŽENE JEDNOSTRANIČNE WEB APLIKACIJE ZA NARUČIVANJE HRANE

Danas više od 17 milijuna Europljana ima alergiju ne neku vrstu hrane. Prema EAACI (engl. *European Academy of Allergy and Clinical Immunology*) broj bolničkih prijema zbog teških alergijskih reakcija na hranu kod djece porastao je sedam puta u posljednjem desetljeću [21]. Osim povećanja broja alergijskih reakcija, pretilost se nastavlja širiti zbog neuravnoteženih prehrana. Kao odgovor tome, aplikacije za dijetu postaju sve češće [22]. Iz tih razloga je odabrana tema za naručivanje hrane pomoću filtriranja s obzirom na korisnikova prehrambena ograničenja.

3.1. Funkcionalni i nefunkcionalni zahtjevi za predloženu jednostraničnu web aplikaciju

Glavni funkcionalni zahtjevi za predloženu jednostraničnu web aplikaciju su:

- Registracija i prijava korisnika
- Usmjeravanje na različite rute
- Administratoru omogućeno dodavanje, brisanje i uređivanje jela, kategorija i sastojaka
- Običnom korisniku omogućeno filtriranje jela
- Običnom korisniku omogućeno sortiranje jela
- Običnom korisniku omogućeno naručivanje jela
- Običnom korisniku omogućeno označavanje jela koja mu se sviđaju

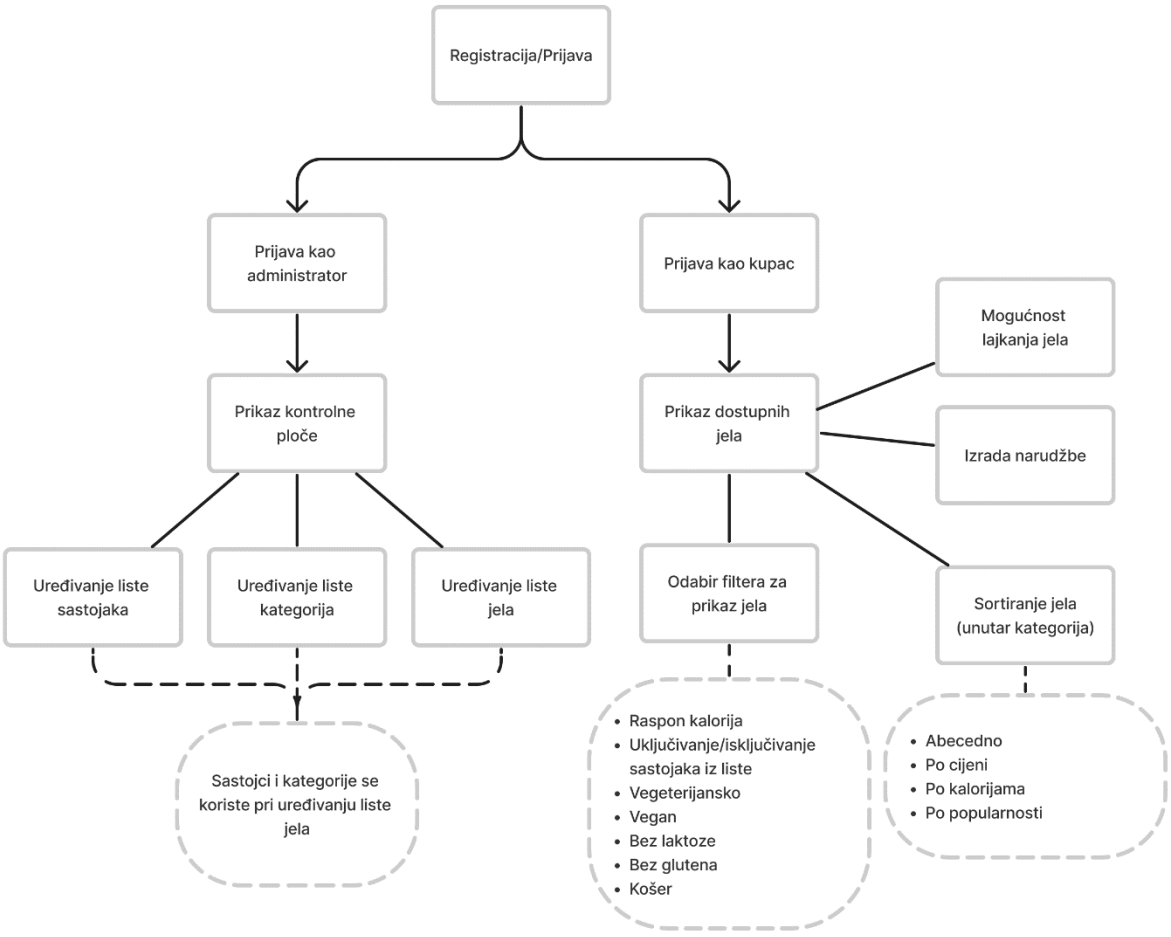
Najbitniji nefunkcionalni zahtjevi su:

- Zaštićene rute
- Rad na operacijskom sustavu Windows 10
- Rad na web preglednicima Chrome, Opera i Firefox
- Ponovno dohvaćanje liste jela pri ponovnom učitavanju stranice

3.2. Model jednostranične web aplikacije

Jednostranična web aplikacija koja će se izraditi u ovome radu će omogućavati restoranu upravljanje jelima, a korisniku filtriranje i naručivanje jela. Model stranice je prikazan na slici 3.1. Pri dolasku na stranicu korisnik će imati opciju registrirati se kao novi korisnik ili se prijaviti kao postojeći. Ako je prijavljeni korisnik označen kao administrator, bit će preusmjeren na kontrolnu ploču. Na stranici kontrolne ploče, administrator će imati mogućnost uređivanja liste dostupnih

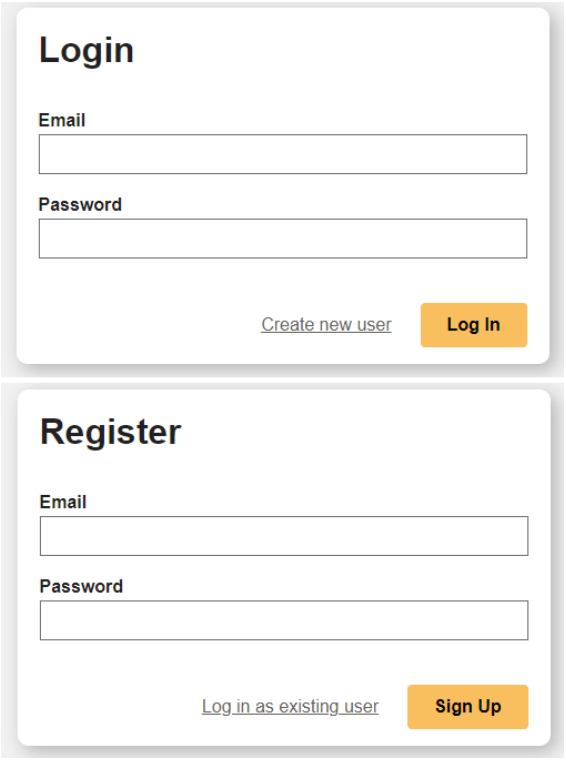
sastojaka, kategorija i jela. Ako prijavljeni korisnik nije administrator, bit će preusmjeren na stranicu koja prikazuje dostupna jela. Na stranici dostupnih jela, obični korisnik će imati mogućnost filtriranja prikazanih jela u odnosu na odabrane opcije, koje su: raspon kalorija, uključivanje i isključivanje određenih sastojaka, te odabir je li jelo odgovara različitim dijetama. Također će imati mogućnost sortirati filtrirana jela unutar kategorija abecedno, po cijeni, po kalorijama i po popularnosti. To znači da se jela unutar svake kategorije sortiraju po odabranoj opciji neovisno o drugim kategorijama, npr. sortirana jela nikada neće biti redoslijedom desert, glavno jelo, desert. Osim toga korisnik će moći označiti da mu se jelo sviđa i dodavati jela u košaricu, a i time ih naručiti.



Sl. 3.1. Model i funkcionalnosti web aplikacije za naručivanje zdrave hrane

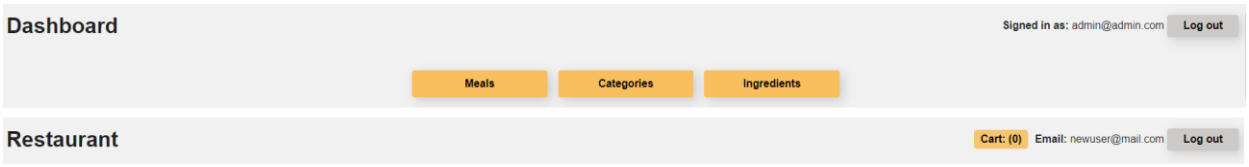
3.3. Korisničko sučelje jednostranične web aplikacije

Pri dolasku na stranicu prikazuje se obrazac za prijavu (slika 3.2). Ako korisnik nema izrađen račun, može se usmjeriti prema obrascu za registraciju (slika 3.2) gdje može izraditi novi račun.



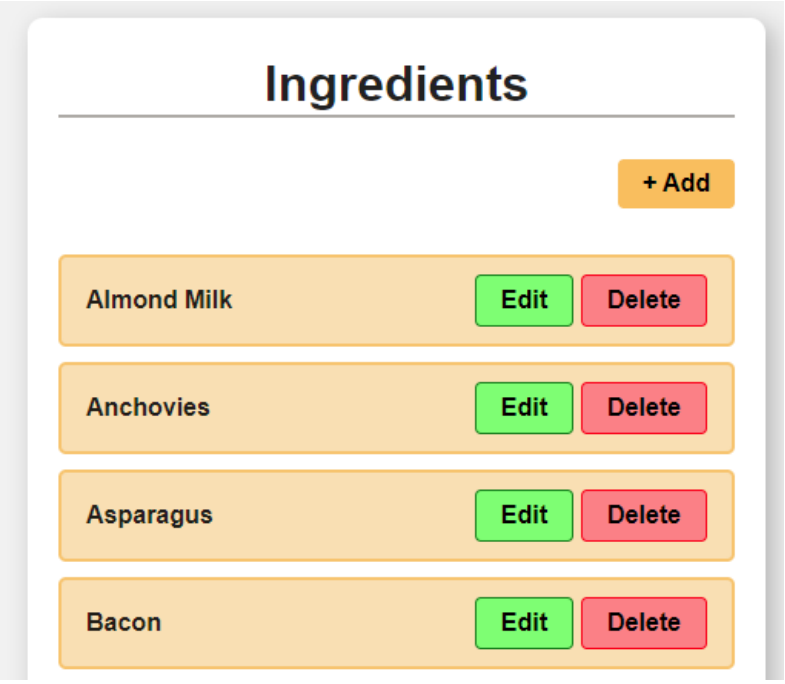
Sl. 3.2. Obrazac za prijavu (gore) i registraciju (dolje)

Nakon prijave, ako je korisnik administrator, bit će preusmjeren na stranicu kontrolne ploče, a ako nije onda će biti preusmjeren na stranicu dostupnih jela. Pri vrhu stranice kontrolne ploče se nalazi zaglavlje (slika 3.3) koje sadrži naslov, informacije o prijavljenom korisniku, gumb za odjavu te gumbove za navigaciju do lista jela, kategorija i sastojaka. Zaglavlje običnog korisnika (slika 3.3) je slično zaglavlju kontrolne ploče i ono također sadrži naslov, informacije o prijavljenom korisniku i gumb za odjavu, ali sadrži i gumb za navigaciju do košarice.



Sl. 3.3. Zaglavlje administratora (gore) i zaglavlje običnog korisnika (dolje)

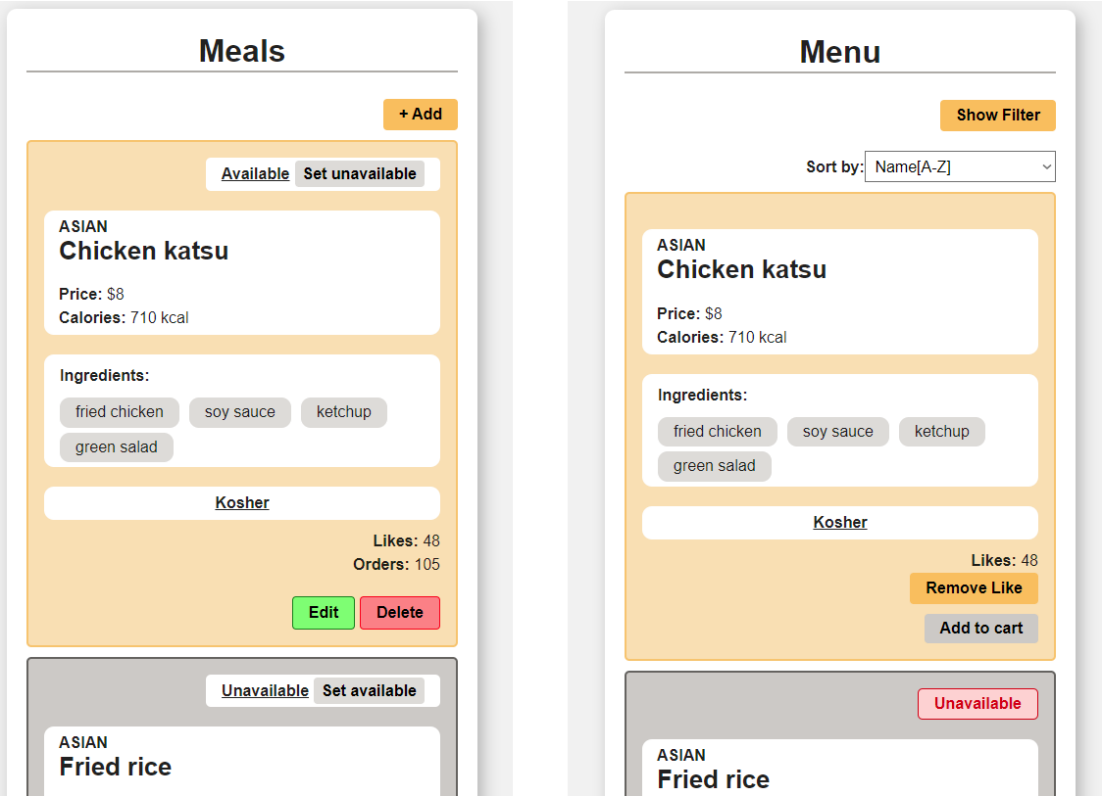
Na slici 3.4 je prikazana lista sastojaka unutar kontrolne ploče. Administrator može uređivati tu listu sastojaka dodavanjem novih sastojaka te brisanjem i uređivanjem postojećih sastojaka, što je prikazano na slici 3.5. Uzimajući u obzir da liste sastojaka i kategorija sadrže iste elemente (samo naziv sastojka/kategorije), lista kategorija ima isti izgled kao i lista sastojaka. Jedina razlika je u samome naslovu liste. Naravno izgled dodavanja i uređivanja kategorije je također isti kao i za sastojke.



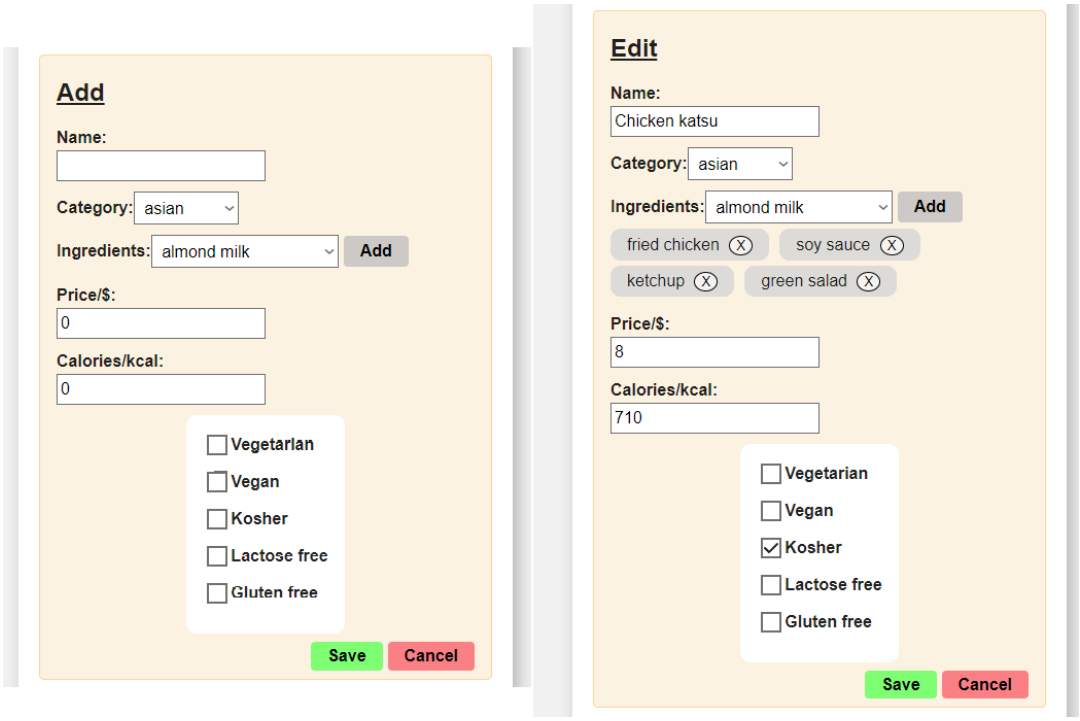
Sl. 3.2. Lista sastojaka

Sl. 3.3. Dodavanje (gore) i uređivanje (dolje) sastojaka

Svako jelo sadrži naziv, kategoriju, cijenu, količinu kalorija, sastojke i dijete kojima jela odgovara (veganska, vegetarijanska, košer, bez laktoze i bez glutena). Lista jela kontrolne ploče je prikazana na slici 3.6. Na listi jela administrator, kao i kod prijašnjih lista, ima mogućnost dodavanja, brisanja i uređivanja jela (slika 3.7). Osim toga, administrator ima opciju izmjenjivanja je li jelo dostupno, čime može onemogućiti naručivanje određenih jela.



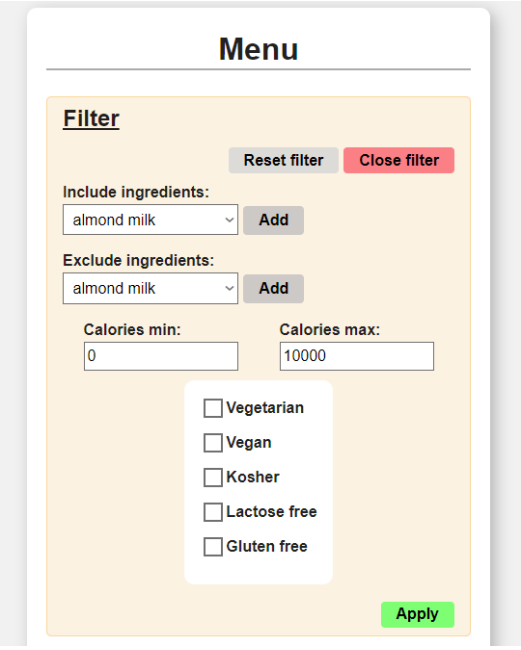
Sl. 3.4. Lista jela kontrolne ploče (lijevo) i list jela vidljiva korisniku (desno)



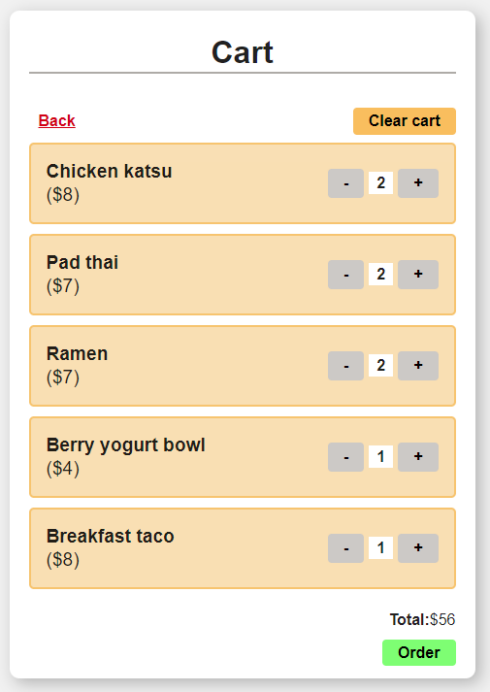
Sl. 3.5. Dodavanje (lijevo) i uređivanje (desno) jela

Također na slici 3.7 je prikazan izgled liste jela na strani običnog korisnika. Naravno tamo korisnik nema opciju uređivanja jela, ali ima opciju sortiranja i filtriranja jela. Osim toga, korisnik može i dodati jelo u košaricu i označiti jelo da mu se sviđa.

Filter za jela je prikazan na slici 3.8. U filteru korisnik može odabrati koje sastojke jela moraju sadržavati i sastojke koje ne smiju sadržavati. Nadalje, korisnik može definirati raspon kalorija i vrste dijeta kojima jelo treba odgovarati. Košarica je prikazana na slici 3.9 gdje korisnik može vidjeti sva jela koja je odabrao za narudžbu zajedno s ukupnom cijenom, a zatim i obaviti tu narudžbu.



Sl. 3.6. Filter jela



Sl. 3.7. Košarica

4. PROGRAMSKO RJEŠENJE FUNKCIONALNIH KOMPONENTI JEDNOSTRANIČNE WEB APLIKACIJE ZA NARUČIVANJE HRANE

U ovome poglavlju će se navesti korišteni alati i tehnologije te objasniti programski kod jednostranične web stranice koja je prikazana u poglavlju 3.3. Prvo će biti objašnjen programski kod Vue.js inačice, a zatim React inačice. S obzirom na to da obje inačice sadrže poveliku količinu programskog koda, fokusirat će se isključivo na nekoliko glavnih funkcionalnosti i problema s kojima se suočava pri izradi jednostranične web aplikacije. Pri objašnjavanju programskog koda se pretpostavlja da je čitatelj upoznat s osnovnom HTML i JavaScript sintaksom. Prvo će se objasniti osnovna sintaksa Vue.js i React komponenti kako bi se ostatak programskog koda lakše razumio. Nakon toga će se objasniti iduća tri problema: usmjeravanje (engl. *routing*), košarica (tj. rad s globalnim stanjem) i autentikacija korisnika (zaštićene rute).

4.1. Korišteni alati i tehnologije

4.1.1. HTML

HTML [23] je opisni jezik koji definira strukturu sadržaja. HTML se sastoji od niza elemenata koji se koriste za omotavanje različitih dijelova sadržaja kako bi izgledao na određeni način ili djelovao na određeni način.

Glavni dijelovi HTML su:

- **Oznaka otvaranja** – Sastoji se od naziva elementa, omotanog u izlomljenim zagradama. Ovo navodi gdje element počinje ili počinje djelovati.
- **Završna oznaka** – Ista je kao i početna oznaka, osim što uključuje kosu crtu ispred naziva elementa. Ovo navodi gdje element završava.
- **Sadržaj** – Sadržaj elementa koji se nalazi između oznaka otvaranja i zatvaranja. Sadržaj je većinom tekst ili neki drugi element.
- **Element** – Element čine početna oznaka, završna oznaka i sadržaj. Elementi također mogu sadržavati atribute koji sadrže dodatne informacije o elementu.

4.1.2. CSS

CSS [24] je jezik za opisivanje prezentacije dokumenta napisanog u HTML-u ili XML-u (engl. *Extensible Markup Language*). CSS opisuje kako se elementi trebaju prikazati na ekranu, na papiru, u govoru ili drugim medijima. CSS je jedan od temeljnih jezika otvorenog weba i

standardiziran je u svim web preglednicima prema W3C (engl. *World Wide Web Consortium*) specifikacijama.

Dizajniran je da omogući odvajanje prezentacije od sadržaja. Ovo razdvajanje može poboljšati dostupnost sadržaja, pružiti veću fleksibilnost i kontrolu u specifikaciji karakteristika prezentacije, omogućiti višestrukim web stranicama dijeljenje formatiranja navođenjem CSS-a u zasebnoj datoteci.

4.1.3. JavaScript

JavaScript [25] je moćan programski jezik koji web stranici dodaje interaktivnost. Sam JavaScript je relativno kompaktan, ali vrlo fleksibilan. Programeri su napisali razne alate povrh osnovnog JavaScript jezika, omogućujući golemu količinu funkcionalnosti uz minimalan napor.

JavaScript omogućuje spremanje korisnih vrijednosti unutar varijabli. Moguće je izvoditi operacije na dijelovima teksta. Također je moguće izvođenje programskog koda kao odgovor na određene događaje koji se odvijaju na web stranici. Međutim, funkcionalnost koja se najviše ističe je API. Oni su gotovi skupovi blokova programskog koda koji programeru omogućuju implementaciju programa kojeg bi inače bilo teško ili nemoguće implementirati.

4.1.4. Vue

Vue aplikacije se sastavljaju pomoću komponenta koje se pišu u datoteke s ekstenzijom „.vue“. Sav HTML, CSS i JavaScript kod relevantan za komponentu se piše unutar te datoteke. Vue komponente će biti detaljnije opisane u nadolazećem poglavlju o programskom rješenju, a više općenitih informacija o Vue-u se nalaze u prijašnjem poglavlju 2.4.1.

4.1.5. React

React aplikacije se također sastavljaju koristeći komponente. Koristi se JSX (engl. *JavaScript XML*) sintaksa koja omogućuje da se HTML elementi koriste direktno unutar JavaScript koda. To će biti detaljnije opisano u idućem poglavlju o programskom rješenju, a više općenitih informacija o React-u se nalaze u prijašnjem poglavlju 2.4.2.

4.1.6. Firebase

Firebase [26] je platforma za pomoć pri razvoju aplikacija. U ovome radu će se koristiti za *backend* kao baza podataka na kojoj će se čuvati podatci o svim jelima. Također će se koristiti za autentikaciju korisnika.

4.1.7. Figma

Figma [27] je web bazirana aplikacija za uređivanje grafike i dizajn korisničkog sučelja. Može se koristiti za obavljanje svih vrsta grafičkog dizajna kao što su dizajniranje sučelja mobilnih aplikacija, dizajna prototipa, izrada objava na društvenim mrežama, itd.

Figma se razlikuje od ostalih alata za uređivanje grafike po tome što radi izravno na pregledniku. To znači da se može pristupiti projektima i započeti projektiranje s bilo kojeg računala ili platforme bez kupnje više licenci ili instalacije programa. To omogućuje da više programera može raditi u isto vrijeme na istom projektu. Figma također omogućuje stvaranja biblioteka komponenti za višekratnu upotrebu kojima ima pristup cijeli tim. To znači da kada se ažurira komponenta u biblioteci, promjene će se izvesti na svim dizanima koji koriste tu komponentu. Moguće je izraditi prototip stvaranjem veza na dizajnu kako bi se simulirala korisnikova interakcija sa sučeljem.

4.1.8. Visual Studio Code

Visual Studio Code [28] je lagan, ali moćan uređivač izvornog koda dostupan na Windows, macOS i Linux sustavima. Dolazi s ugrađenom podrškom za JavaScript, TypeScript i Node.js te ima bogat ekosustav proširenja za druge jezike.

4.2. Programsko rješenje koristeći Vue.js

4.2.1. Vue komponente

Osnovnu sintaksu i dijelove Vue komponente će se objasniti pomoću dvije jednostavne komponente. Svaka komponenta se sastoji od dva dijela, *template* (predložak) i *script* (skripta). Unutar predloška se piše HTML kod, a unutar skripte JavaScript kod. Također postoji i treći dio, *style* (stil), gdje se piše CSS kod, ali za ovaj primjer nije potreban. Komponenta „App.vue“ (programski kod 4.1) unutar predloška sadrži polje za unos, komponentu „AnimalInfo.vue“ (programski kod 4.2) i gumb. Unutar skripte se nalazi nekoliko opcija. *Components* sadrži sve komponente koje se koriste unutar predloška. *Data* sadrži varijable koje komponenta koristi, a *methods* funkcije. Također u komponenti „AnimalInfo.vue“ postoji opcija *props* koja definira svojstva koje joj prosljeđuje roditeljska komponenta. Naravno to su samo neke od osnovnih opcija, a ostale će biti objašnjene kada se pojave. Unutar predloška (programski kod 4.1) postoje neke ključne riječi koje se koriste na atributima HTML elemenata ili na atributima Vue komponenti. V-model se koristi isključivo na *input* elementima i on povezuje vrijednost *input* elementa s podatkom iz *data* opcije, u ovom slučaju je to „animalName“. Time svaki puta kada se izmjeni

vrijednost unutar polja za unos, promijenit će se i vrijednost povezane varijable. V-if se koristi za uvjetno prikazivanje komponenti (prihvaća *boolean* vrijednosti). V-bind se koristi za povezivanje podataka (engl. *data binding*). Pomoću njega možemo atributu postaviti vrijednosti koje nisu tekst. U našem slučaju vežemo atribut *name* (koji je također naziv svojstva koje predajemo komponenti „AnimalInfo.vue“) uz vrijednost „animalName“. Time osiguravamo da svaki puta kada se izmjeni vrijednost „animalName“, da će se ta nova vrijednost ponovno proslijediti atributu *name*. Skraćeni način korištenja v-bind je samo dvotočka „:“ i taj način pisanja će se koristiti u daljnjim programskim kodovima. V-on se koristi za pokretanje funkcija na događaje (u ovom slučaju na klik). Skraćeni način pisanja je „@“ znak i on će se dalje koristiti. Konačna funkcionalnost komponente „App.vue“ je ta da svaki puta kada korisnik unese nešto u polje za unos, istog trena će se ta vrijednost predati komponenti „AnimalInfo.vue“ koja će ispisati tu vrijednost ekran. Također kada korisnik klikne na gumb, komponenta „AnimalInfo.vue“ će se naizmjenično obrisati i dodati.

```
<template>
  <input v-model="animalName" />
  <animal-info v-if="isVisible" v-bind:name="animalName"/>
  <button v-on:click="clickHandler">Change visibility</button>
</template>

<script>
import AnimalInfo from './components/AnimalInfo.vue';
export default {
  components: { AnimalInfo },
  data() {
    return {
      animalName: '',
      isVisible: true,
    };
  },
  methods: {
    clickHandler() {
      this.isVisible = !this.isVisible;
    },
  },
};
</script>
```

Programski kod 4.1. Programski kod Vue komponente App.vue

```
<template>
  <p>Animal name: {{ name }}</p>
</template>
<script>
export default {
  props: ['name'],
};
</script>
```

Programski kod 4.2. Programski kod Vue komponente AnimalInfo.vue

4.2.2. Usmjeravanje koristeći Vue.js

Zbog toga što jednostranične web aplikacije samo jednom učitaju stranicu, URL će stalno biti isti. To znači da postoji samo jedna poveznica na stranicu koja uvijek učita početno stanje stranice. Usmjerivač omogućuje definiranje vlastitih ruta, tj. poveznica, pomoću kojih se može direktno otvoriti neko drugo stanje osim početnog. Kod Vue.js će se koristiti vue-router. Usmjerivač je definiran u zasebnoj datoteci „router.js“. Programski kod 4.3 prikazuje definiranje vlastitih ruta.

```
const router = createRouter({
  history: createWebHistory(),
  routes: [
    {
      path: '/',
      redirect: '/login',
    },
    {
      path: '/admin',
      component: AdminDashboard,
      meta: {
        requiresAuth: true,
      },
      children: [
        {
          path: 'meals',
          component: MealsPage,
          meta: {
            requiresAuth: true,
          },
        },
        {
          path: 'categories',
          component: CategoriesPage,
          meta: {
            requiresAuth: true,
          },
        },
        {
          path: 'ingredients',
          component: IngredientsPage,
          meta: {
            requiresAuth: true,
          },
        },
      ],
    },
//...
```

Programski kod 4.1. Dio programskog koda datoteke router.js u Vue projektu

Za svaku rutu se mogu definirati svojstva: putanja (engl. *path*), komponenta (engl. *component*), meta (proizvoljni podatci), djeca i preusmjeravanje (engl. *redirect*). Putanja definira nastavak koji će se pridodati na glavni URL, a komponenta definira Vue komponentu koja će se

prikazati dolaskom na tu rutu. Svaka ruta može imati definirane dječje rute. Putanja dječje rute se nadodaje na putanju roditeljske rute. Također se pri stvaranju usmjerivača definira povijest (engl. *history*) pomoću funkcije „createWebHistory“. To je nužno jer se stranica nikada neće ponovno učitavati te zbog toga se neće zapisati u povijest preglednika. Definiranjem povijesti omogućujemo navigaciju naprijed i nazad pomoću gumbova na pregledniku.

Da bi se definirani usmjerivač mogao koristiti potrebno ga je omogućiti unutar datoteke „main.js“. Prije nego što se aplikacija montira (engl. *mount*), potrebno je navesti da će se koristiti izrađeni usmjerivač (programski kod 4.4). Osim toga, u glavnoj komponenti „App.vue“ je potrebno unutar predloška staviti element „router-view“ (programski kod 4.5) koji će usmjerivaču reći gdje točno da generira komponente koje su definirane za određenu putanju. Za navigaciju kroz rute preko HTML elemenata potrebno je dodati „router-link“ element (programski kod 4.6). On funkcionira kao obični HTML *anchor* (<a>) element za hipervezu. Ima atribut „to“ kojemu se predaje putanja do koje se želi preusmjeriti.

```
app = createApp(App);
app.use(router);
app.use(store);
app.mount('#app');
```

Programski kod 4.4. Dio programskog koda datoteke main.js vezan uz usmjerivač unutar Vue projekta

```
<template>
  <router-view></router-view>
</template>
```

Programski kod 4.5. Dio programskog koda Vue komponente App.vue

```
<template>
<!-- ... -->
   <li>
     <router-link to="/admin/meals"> Meals </router-link>
   </li>
   <li>
     <router-link to="/admin/categories"> Categories </router-link>
   </li>
   <li>
     <router-link to="/admin/ingredients"> Ingredients </router-link>
   </li>
<!-- ... -->
</template>
```

Programski kod 4.6. Dio programskog koda Vue komponente AdminDashboardNav.vue odgovoran za navigaciju

4.2.3. Košarica i globalno stanje koristeći Vue.js

Zbog toga što se koriste isti podatci i funkcije za rad s tim podatcima na dvije različite stranice (slika 3.6 i slika 3.9), potrebno ih je pohraniti u globalno stanje kako bi se izbjeglo dupliciranje programskog koda. U tu svrhu se koristi vuex. U programskom kodu 4.7 je prikazano stvaranje vuex skladišta (engl. *store*). Vuex skladište je sastavljeno od više modula, gdje svaki modul sadrži podatke i funkcije koje su međusobno povezane. Vidljivo je da košarica nije jedino globalno stanje koje se koristi unutar projekta, ali će se koristiti kao primjer. Svaki modul (programski kod 4.8) sadrži stanje (engl. *state*), mutacije (engl. *mutations*), funkcije za dohvaćanje (engl. *getters*) i akcije.

```
const store = createStore({
  modules: {
    ingredients: ingredientsModule,
    categories: categoriesModule,
    meals: mealsModule,
    cart: cartModule,
    auth: authModule,
  },
```

Programski kod 4.7. Dio programskog koda koji definira vuex *store*

```
export default {
  namespaced: true,
  state() {
    return {
      cart: [],
    };
  },
  mutations,
  actions,
  getters,
};
```

Programski kod 4.8. Programski kod modula košarice za vuex

Stanje sadrži sve relevantne podatke. Mutacije se koriste za izmjenu stanja (mogu se gledati kao *setter*-i). *Getter*-i dohvaćaju vrijednost stanja. A akcije pozivaju mutacije. Također je definirano da se koristi imenski prostor (engl. *namespace*) čime će se izbjeći podudaranje imena stanja i funkcija (npr, ako dva modula imaju isti naziv stanja, to više neće stvarati problem). Akcija za dodavanje jela u košaricu je prikazana programskim kodom 4.9. Akcije kao prvi argument primaju kontekst, koji vraća trenutnu *store* instancu (u ovom slučaju je to modul za košaricu). Preko konteksta se mogu koristiti prijašnje definirana stanja, *getter*-i, mutacije i akcije. U ovom slučaju se vidi način na koji se pristupa stanju i *getter*-ima (programski kod 4.9, 2. i 3. linija) te kako pozvati mutaciju pomoću ugrađene *commit* funkcije (programski kod 4.9, zadnja linija).

```
addToCart(context, item) {
    const currentCart = context.state.cart;
    const isItemInCart = context.getters.getItem(item);
    if (isItemInCart) {
      context.commit(
        'setCart',
        currentCart.map((cartItem) =>
          cartItem.id === item.id
            ? { ...cartItem, quantity: cartItem.quantity + 1 }
            : cartItem
        )
      );
    } else {
      context.commit('setCart', [...currentCart, { ...item, quantity: 1 }]);
 }}
```

Programski kod 4.9. Programski kod akcije za dodavanje jela u košaricu, definirana unutar modula košarice

Programski kod 4.10. prikazuje korištenje izrađenog skladišta unutar Vue komponente. *Getter*-i se mogu raspakirati unutar *computed* opcije, a akcije unutar metoda. *Computed* opcija se koristi kao i *data* opcija, tj. sprema neki podatak. Razlika između *dana* i *computed* je ta što *computed* vrijednosti izvode neku funkciju kako bi vratili tu vrijednost, u ovome slučaju se pokreći *getter* funkcije koje vraćaju vrijednosti stanja iz *store*-a. Za *getter*-e se koristi funkcija „mapGetters“ unutar kojoj se predaje objekt. Unutar objekta naziv svojstva je proizvoljan, a kao vrijednost svojstva se poziva *getter* iz *store*-a pomoću sintakse „naziv_modula/naziv_gettera“. Tako se ne mora u komponentu postavljati sve funkcije iz *store*-a, nego samo one koje će se koristiti. Isto vrijedi za dohvaćanje akcija pomoću funkcije „mapActions“. Sada su *getter*-i i akcije dostupne komponenti za korištenje kao obične varijable i funkcije. Naravno, slično usmjerivaču, potrebno je omogućiti korištenje *store*-a unutar aplikacije. To se izvodi na isti način kao i kod usmjerivača, pomoću funkcije „use“ unutar datoteke „main.js“ (programski kod 4.4).

```
<script>
import { mapActions, mapGetters } from 'vuex';
export default {
  computed: {
    ...mapGetters({
      cart: 'cart/cart',
      cartTotal: 'cart/cartTotal',
      getItem: 'cart/getItem',
    }),
  },
  methods: {
    ...mapActions({
      addToCart: 'cart/addToCart',
      removeFromCart: 'cart/removeFromCart',
      clearCart: 'cart/clearCart',
    }),},};
</script>
```

Programski kod 4.10. Programski kod za korištenje *getter*-a i akcija unutar Vue komponenti

4.2.4. Autentikacija korisnika koristeći Vue.js

Postoje dva tipa korisnika: administrator i obični korisnik. Potrebno ih je razlikovati te ih preusmjeriti na točnu stranicu. Osim toga potrebno im je blokirati pristup određenim stranicama ovisno o njihovom tipu (npr. obični korisnik ne smije pristupiti stranici kontrolne ploče). Za autentikaciju korisnika koristi se Firebase autentikacija pomoću *mail*-a i šifre. Također se koristi Firebase baza podataka u stvarnom vremenu (engl. *realtime database*) za spremanje informacija o korisniku. U ovome slučaju su to informacija o tome je li korisnik administrator i jela koja se sviđaju korisniku. Kako bi se Firebase mogao koristiti potrebno ga je instalirati u projekt koristeći npm (engl. *node package manager*). Također je potrebno konfigurirati i omogućiti pristup Firebase-u unutar samog projekta. Konfiguracija je napisana u zasebnoj firebase.js datoteci (programski kod 4.11). Unutar firebase.js datoteke se inicijalizira aplikacija koristeći konfiguraciju koja se dobije pri stvaranju Firebase projekta online. Zatim se preko inicijalizirane aplikacije izvode (engl. *export*) zasebno dio za autentikaciju i dio za bazu podataka.

```
import firebase from 'firebase/compat/app';
import 'firebase/compat/auth';
import 'firebase/compat/database';

const firebaseConfig = {
  // ...
};

const app = firebase.initializeApp(firebaseConfig);

export const auth = app.auth();
export const database = app.database();
export default app;
```

Programski kod 4.11. Programski kod konfiguracija Firebase-a unutar datoteke firebase.js

Samo spremanje korisničkih podataka unutar web aplikacije se izvodi pomoću vuex-a. Način korištenja vuex-a je prijašnje objašnjen u poglavlju 4.1.3 na primjeru košarice. Programski kod 4.12 prikazuje način korištenja definiranih Firebase „auth“ i „database“ komponenti unutar vuex modula za autentikaciju. Pri registraciji se koristi ugrađena Firebase funkcija „createUserWithEmailAndPassword“, a pri spremanju podataka u bazu funkcija „set“. Osim tih akcija, postoje i akcije za prijavu korisnika, odjavu korisnika, spremanje korisničkih podataka lokalno, brisanje lokalnih korisničkih podataka, provjera je li prijavljen novi korisnik i ažuriranje liste jela koja se sviđaju korisniku.

```
async signup(context, { email, password }) {
    const res = await auth.createUserWithEmailAndPassword(email, password);
    return res;
  },
async updateOnlineUserData() {
    await database.ref('users/' + auth.currentUser.uid).set({
      email: auth.currentUser.email,
      isAdmin: false,
      likes: ['1'],
    });
  }
```

Programski kod 4.12. Programski kod akcije za registraciju korisnika i akcija za spremanje podataka korisnika u bazu podataka unutar Vue projekta

Iduće je potrebno preusmjeriti korisnika na pravilnu rutu nakon prijave, registracije i odjave. Taj dio programskog koda se nalazi unutar datoteke „main.js“ (programski kod 4.13).

```
auth.onAuthStateChanged(async () => {
  if (auth.currentUser) {
    const isNewUser = await store.dispatch('auth/checkIfNewUser');

    if (isNewUser) {
      await store.dispatch('auth/updateOnlineUserData');
    }
    await store.dispatch('auth/updateLocalUserData');

    if (store.getters['auth/userData'].isAdmin) {
      router.push('/admin');
    } else {
      router.push('/shop');
    }
  } else {
    await store.dispatch('auth/clearUserData');
    router.push('/login');
  }
  if (!app) {
    app = createApp(App);
    app.use(router);
    app.use(store);
    app.mount('#app');
  }});
```

Programski kod 4.13. Programski kod za preusmjeravanje korisnika unutar main.js datoteke u Vue projektu

Preusmjeravanje se odvija unutar Firebase „onAuthStateChanged“ funkcije. Unutar nje se definira funkcija koja će se pokrenuti svaki puta kada se stanje korisnika izmjeni. Ta funkcija sadrži dva glavna *if* bloka. Prvi blok provjerava je li korisnik definiran, a drugi blok provjerava je li web aplikacija definirana. Drugi *if* blok se nalazi unutar „onAuthStateChaged“ funkcije kako bi se osiguralo da se aplikacija ne generira prije dohvaćanja podataka o korisniku. Unutar prvog *if* bloka (ako postoji korisnik) prvo se provjerava je li trenutni korisnik novi korisnik, tj. jesu li njegovi podatci već spremljeni u bazu. Ako je novi korisnik, onda će se prvo njegovi podatci

spremiti na bazu. Nakon toga će se dohvatiti korisnički podatci s baze te spremiti lokalno, neovisno o tome je li novi korisnik ili ne. Navedene radnje se izvode preko prijašnje definiranog vuex modula za autentikaciju. Nakon dohvaćanja korisničkih podataka, provjerava se je li prijavljeni korisnik administrator. Ako je onda će biti preusmjeren na stranicu kontrolne ploče, a ako nije onda će biti preusmjeren na stranicu trgovine/restorana. Preusmjeravanje se izvodi koristeći prijašnje definiran vue-router. Ako trenutno ne postoji prijavljeni korisnik, onda će biti preusmjeren na stranicu za prijavu.

Također je potrebno zaštiti rute kako bi se spriječio odlazak na određene stranice neovlaštenim korisnicima. To se postiže pomoću „beforeEach“ funkcije na usmjerivaču (programski kod 4.14). Ona kao argumente prihvaća *to* (ruta na koju se želi preusmjeriti), *from* (ruta s koje se pokreće preusmjeravanje) i *next* (funkcija za preusmjeravanje).

```
router.beforeEach(function (to, from, next) {
  const currentUser = auth.currentUser;
  const requiresAuth = to.meta.requiresAuth;
  const isAdmin = store.getters['auth/userData'].isAdmin;
  if (
    to.fullPath != '/admin' &&
    to.fullPath != '/admin/meals' &&
    to.fullPath != '/admin/ingredients' &&
    to.fullPath != '/admin/categories' &&
    requiresAuth &&
    currentUser &&
    isAdmin
  ) {
    next('/admin');
  }
// ...
   else {
    next();
  }});
```

Programski kod 4.14. Dio programskog koda za zaštitu ruta unutar datoteke router.js u Vue projektu

Ta funkcija se poziva pri svakome zahtjevu za preusmjeravanje. Unutar programskog koda 4.14 prikazan samo primjer za jednu rutu „admin“. Provjerava se je li je li ruta na koju se želi preusmjeriti različita od rute „auth“ ili bilo koje njezine dječje ruta. Nakon toga se provjerava ako ta ruta zahtjeva da je korisnik prijavljen, što je definirano pomoću *meta* polja „requiresAuth“ unutar definicije rute (programski kod 4.3). Zatim se provjerava je li korisnik prijavljen i je li prijavljen korisnik administrator. Ako su svi ti uvjeti zadovoljeni poziva se *next* funkcija s rutom na koju se želi preusmjeriti, u ovom slučaju je to ruta za kontrolnu ploču. Na sličan način je definirana provjera za sve ostale rute. Na kraju, unutar *else* bloka se poziva prazna *next* funkcija koja odobrava zahtijevano preusmjeravanje.

4.3. Programsko rješenje koristeći React

4.3.1. React komponente

Za objašnjenje osnova React komponenti i sintakse će se koristiti komponente s istim funkcionalnostima kao što su se koristile za Vue inačicu (poglavlje 4.2.1). Programski kod glavne App komponente je prikazan programskim kodom 4.15, a programski koda „AnimalInfo“ komponente je prikazan programskim kodom 4.16.

```
const App = () => {
  const [animalName, setAnimalName] = useState('');
  const [isVisible, setIsVisible] = useState(true);
  const changeVisibility = () => {
    setIsVisible((prevState) => {
      return !prevState;
    });
  };
  return (
    <div>
      <input
        value={animalName}
        onChange={(e) => setAnimalName(e.target.value)}
      />
      {isVisible && <AnimalInfo name={animalName} />}
      <button onClick={changeVisibility}>Change visibility</button>
    </div>
  );
};
```

Programski kod 4.15. Programski kod React komponente App.js

```
const AnimalInfo = (props) => {
  return <p>Animal name: {props.name}</p>;
};
```

Programski kod 4.16. Programski kod React komponente AnimalInfo.js

React komponente se definiraju kao JavaScript funkcije koje vraćaju HTML elemente koji se žele prikazati. To je moguće zbog JSX sintakse, koja omogućuje direktno korištenje HTML elemenata unutar JavaScript koda. React komponente imaju pristup React *hook*-ovima, koji pružaju laki način korištenja značajki Reacta. U ovome primjeru (programski kod 4.15) se koristi *hook* „useState“, koji omogućuje lako upravljanje stanjem. Definira se naziv stanja i naziv funkcije za mijenjanje stanja te početna vrijednost stanja. Kao i kod Vue.js-a, kada se stanje izmjeni, nova vrijednost će se predati svim elementima koji ga koriste. Za reagiranje na neki događaj se koriste React specifični atributi za HTML elemente, kao što su: „onChange“ i „onClick“. Kako bi „AnimalInfo“ komponenta imala pristup podatcima iz roditeljske komponente, potrebno je kao argument komponenti predati „props“. To je objekt koji sadrži sve vrijednosti koje je predala

roditeljska komponenta (u ovome slučaju vrijednost *name*). Uvjetno prikazivanje komponenti se izvodi pomoću logičke operacije I (&&), gdje je jedna od vrijednosti uvjet za prikazivanje elementa, a druga vrijednost je element koji se treba prikazati. Razlog zašto to funkcionira je zbog JSX sintakse, gdje će se element za prikazivanje gledati kao obična vrijednost, tj. vrijednost koja je definirana i različita od nule.

4.3.2. Usmjeravanje koristeći React

Za usmjeravanje React inačice će se koristiti react-router. On se definira unutar glavne komponente App.js (programski kod 4.17). Usmjerivač se stvara pomoću funkcije „createBrowserRouter“. Kao i kod vue-routera, definiraju se rute gdje svaka ruta ima svoju putanju, element koji će se prikazati za tu putanju i dječje rute.

```
const router = createBrowserRouter([
  {
    path: '/',
    element: <MainPage />,
    children: [
      {
        path: 'admin',
        element: <AdminDashboardPage />,
        children: [
          { path: 'meals', element: <MealsPage /> },
          { path: 'categories', element: <CategoriesPage /> },
          {
            path: 'ingredients',
            element: <IngredientsPage />,
          },
        ],
      },
//...
```

Programski kod 4.17. Dio programskog koda za stvaranje usmjerivača unutar React komponente App.js

Za korištenje usmjerivača potrebno je koristiti „ReactRouter“ komponenta, koja je dio react-routera. Koristi se unutar App.js komponente (programski kod 4.18). I njoj se kao atribut predaje izrađeni usmjerivač. Za prikazivanje elementa rute potrebno je definirati gdje će se taj element postaviti. To se izvodi koristeći „Outlet“ komponentu, vidljivo u programskom kodu 4.19. Ovisno o ruti, umjesto *outlet*-a će se generirati element koji je definiran za tu rutu. Unutar AdminDashboardPage komponente se nalazi i komponenta „AdminDashboardNav“, koja će biti generirana za svaku rutu. Prazni HTML elementi su skraćeni način pisanja „Fragment“ komponente koja se koristi kao spremnik bez generiranja dodatnih elemenata (slično „div“ elementu, ali se neće prikazati u DOM-u).

```
function App() {
  return (
    <AuthProvider>
      <CartProvider>
        <RouterProvider router={router} />
      </CartProvider>
    </AuthProvider>
  );
}
```

Programski kod 4.18. Dio programskog koda React komponente App.js

```
const AdminDashboardPage = () => {
  return (
    <>
      <AdminDashboardNav />
      <Outlet />
    </>
  );
};
```

Programski kod 4.19. Dio programskog koda React komponente AdminDashboardPage.js

4.3.3. Košarica i globalno stanje koristeći React

Za korištenje globalnog stanja i funkcija kod React-a će se koristiti *hook* „useContext“ (programski kod 4.20). Prvo se definira kontekst, a zatim pružatelj to konteksta (engl. *provider*). Pružatelj konteksta se definira kao obična React komponenta, unutar koje se definiraju sva stanja i funkcije za upravljanje tim stanjem. Kao element se vraća „CartContext.Provider“ (pružatelj konteksta) gdje su atributi sva stanja i funkcije iz pružatelja koje se trebaju moći koristiti izvan samog pružatelja. Također se definira „props.children“, koji označava sve komponente koje će se nalaziti unutar pružatelja i time omogućiti pristup stanjima i funkcijama pružatelja.

```
export const CartContext = createContext();

export const CartProvider = (props) => {
//definicija stanja i funkcija...
  return (
    <CartContext.Provider
      value={{
        cartItems: cartItems,
        addToCart,
        removeFromCart,
        clearCart,
        getCartTotal,
      }}
    >
      {props.children}
    </CartContext.Provider>
  );
};
```

Programski kod 4.20. Dio programskog koda datoteke cart-context.js u React projektu

U programskom kodu 4.18 je prikazano kako omogućiti korištenje izrađenog pružatelja, u kojemu se unutar komponente „CartProvider“ stavljaju sve komponente kojima se želi pružiti pristup podatcima konteksta. Način pristupanju stanjima i funkcijama iz konteksta je prikazan u programskom kodu 4.21. Nakon što ih se definiram, mogu se koristiti kao i sva ostala stanja i funkcije unutar komponente.

```
const { cartItems, addToCart, removeFromCart, clearCart, getCartTotal } =
    useContext(CartContext);
```

Programski kod 4.21. Dio programskog koda React komponente CartPage.js za pristup kontekstu

4.3.4. Autentikacija korisnika koristeći React

Autentikacija u React inačici se odvija isto kao i u Vue.js inačici, koristeći Firebase. Potrebno instalirati Firebase koristeći npm i napisati konfiguraciju unutar firebase.js datoteke (kod 4.11.). Za spremanje korisničkih podataka i funkcija za autentikaciju se koristi kontekst. Način korištenja je opisan u prijašnjem poglavlju 4.2.3 na primjeru košarice. Kao i za košaricu, potrebno je omogućiti pristup kontekstu cijeloj aplikaciji (programski kod 4.18). Kako bi se odgodilo prikazivanje podataka dok se ne dohvate korisnički podatci, koristi se uvjetno prikazivanje elemenata unutar pružatelja (programski kod 4.22).

```
return (
    <AuthContext.Provider value={value}>
      {!loading && props.children}
    </AuthContext.Provider>
  );
```

Programski kod 4.22. Dio programskog koda datoteke auth-context.js React projektu

Za preusmjeravanje korisnika na pravilnu stranicu prilikom prijave i registracije, koristi se „navigate“ funkcija koja je definirana koristeći „useNavigation“ *hook* (programski kod 4.23). Unutar „submitHandler“ funkcije se prvo provjerava tip prijave, tj. provjerava je li se prijavljuje postojeći korisnik ili se prijavljuje novo registrirani korisnik. Ako se prijavljuje postojeći korisnik, poziva se „login“ funkcija iz konteksta za autentikaciju. Nakon toga se provjerava je li prijavljeni korisnik administrator. Ako je administrator, onda se preusmjerava na stranicu kontrolne ploče, a

ako nije, onda se preusmjerava na stranicu trgovine/restorana. U drugom slučaju, ako se prijavljuje novi korisnik, pokreće se „signup“ funkcija te se preusmjerava na stranicu trgovinu/restorana.

```
const navigate = useNavigate();
const { signup, login, userData } = useContext(AuthContext);

  const submitHandler = async (event) => {
    event.preventDefault();

    if (props.isLogin) {
      try {
        setError('');
        setLoading(true);
        await login(emailRef.current.value, passwordRef.current.value);
        if (userData.isAdmin) {
          navigate('/admin');
        } else {
          navigate('/shop');
        }
      } catch (e) {
        setError(e.message);
      }
    } else {
      try {
        setError('');
        setLoading(true);
        await signup(emailRef.current.value, passwordRef.current.value);
        navigate('/shop');
      } catch (e) {
        setError(e.message);
      }
    }
    setLoading(false);
  };
```

Programski kod 4.23. Dio programskog koda React komponente AuthenticationForm.js

Zaštita ruta se odvija unutar „MainPage“ komponenta. Unutar nje se sve ostale komponente generiraju. Koristi se „useEffect“ *hook* (programski kod 4.24) koji prima dva argumenta. Prvi argument je funkcija koja se treba izvršiti, a drugi argument su vrijednosti na čija se izmjena prati. Kada se izmjeni vrijednost neka od praćenih vrijednosti, definirana funkcija se izvršava. Također, „useEffect“ *hook* se uvijek pokrene jednom, pri samome generiranju komponente. Prvo se provjerava jesu li se dohvatili korisnički podatci. Ako su dohvaćeni podatci prazni, tj. korisnik nije prijavljen, preusmjerava se na stranicu za prijavu. Ako je dohvaćeni korisnik administrator

preusmjerava se na stranicu kontrolne ploče, a ako nije, onda je preusmjeren na stranicu trgovine/restorana.

```
useEffect(() => {
    if (!userDataLoading) {
      if (userData) {
        if (userData.isAdmin) {
          setLoading(false);
          navigate('/admin');
        } else {
          setLoading(false);
          navigate('/shop');
        }
      } else {
        setLoading(false);
        navigate('/login');
      }
    }
  }, [navigate, userData, userDataLoading]);
```

Programski kod 4.24. Dio programskog koda React komponente MainPage.js

5. USPOREDBA PERFORMANSI VUE.JS-a I REACT-a

U ovome poglavlju će se uspoređivati performanse Vue.js-a i React-a. Usporedba na izrađenim stranicama će se izvoditi koristeći Chrome alat za mjerenje performansi (engl. *chrome performance tool*) [29], koji je ugrađen u preglednik. Osim toga će se referencirati na tuđa mjerenja rada s API-em te rada s velikim količinama podataka.

5.1. Mjerenje performansi koristeći Chrome alat

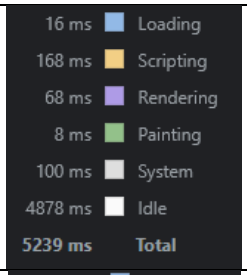
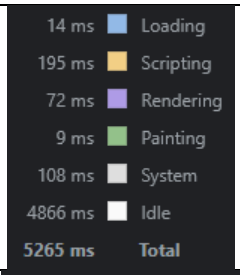
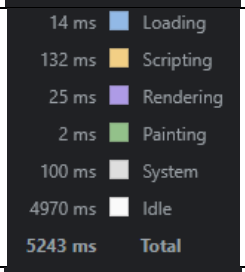
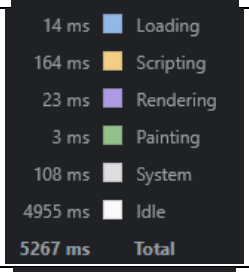
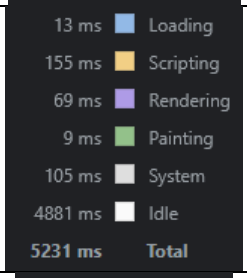
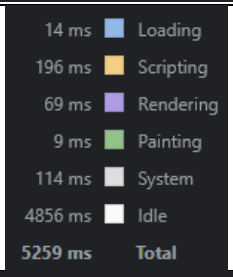
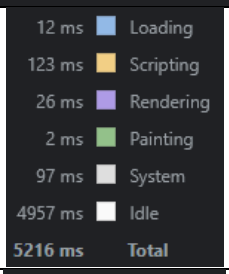
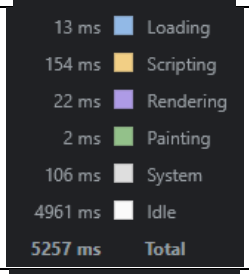
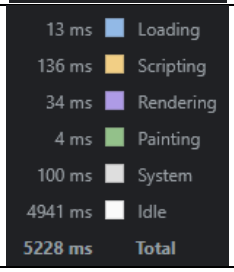
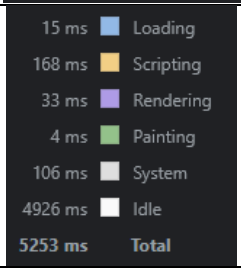
Za početak, potrebno je generirati produkcijske inačice aplikacija (engl. *production build*). Dobivena Vue *build* inačica je veličine 3.48 MB, dok React inačica 7.13 MB. Tu ima prednost Vue, jer je React inačica duplo veća. Kako bi se *build* inačice mogle pravilno pokrenuti, potrebno ih je postaviti na poslužitelj ili ih poslužiti lokalno. U ovome slučaju će se poslužiti lokalno. To će se učiniti koristeći npm serve paket. Nakon što je korištenje stranice omogućeno, započinju se mjerenja. Mjerenja će se izvoditi na pet različitih ruta navedenih u tablici 5.1. Mjerenje će se izvoditi za učitavanje stranice, tj. rute. Znači mjerenje započinje dolaskom na rutu i završava kada se cijela stranica učita i prikaže. Glavna razlika u vremenu je *scripting* vrijeme. Ono je vrijeme izvođenja JavaScript koda. Kod njega je Vue brži po prosijeku za 30.6 ms. Po vremenu generiranja (engl. *render*) i slikanja (engl. *painting*) su podjednaki, gdje se razlikuju za nekoliko milisekundi. Gdje je vrijeme generiranja, primjenjivanje CSS-a, tj. određivanje veličine i pozicioniranje elemenata. A vrijeme slikanja predstavlja slikanje boja, teksta, slika i sjena.

Za dodatnu analizu mjerenja će se koristiti mjerenja izvedena na jednostavnoj aplikaciji za kalkulator [30] gdje je također glavna razlika bila u *scripting* vremenu, no React inačica je bila brža za 57 ms. Dok su vremena generiranja i slikanja ponovno slična te se razlikuju za nekoliko milisekundi.

Pri testiranju stranica u ovome radu je Vue brži, a u drugome radu je React brži. Razlike pri testiranju su veoma male i jako je mala vjerojatnost da bi sami korisnik uopće primijetio tu razliku. Do različitih dobivenih rezultata mjerenja dvije različite aplikacije je moguće zbog samog načina pisanja programskog koda. Za pravilniji test bi bilo potrebno savršeno optimizirati Vue i React inačice aplikacije, što bi bilo veoma teško, a možda i nemoguće, za izvesti.

S obzirom na to da su obje aplikacije male, a i sama razlika u mjerenjima je zanemariva, može se zaključiti da za performanse manjih aplikacija nije presudno koji razvojni okvir se koristi između Vue-a i React-a. Čime se odabir razvojnog okvira za manji projekt svodi na osobne preferencije.

Tablica 5.1. Vue i React usporedba koristeći Chrome alat za mjerenje performansi

Ruta	Vue	React
/store		
/store/cart		
/admin/meals		
/admin/categoires		
/admin/ingredients		

5.2. Usporedba performansi hook-ova

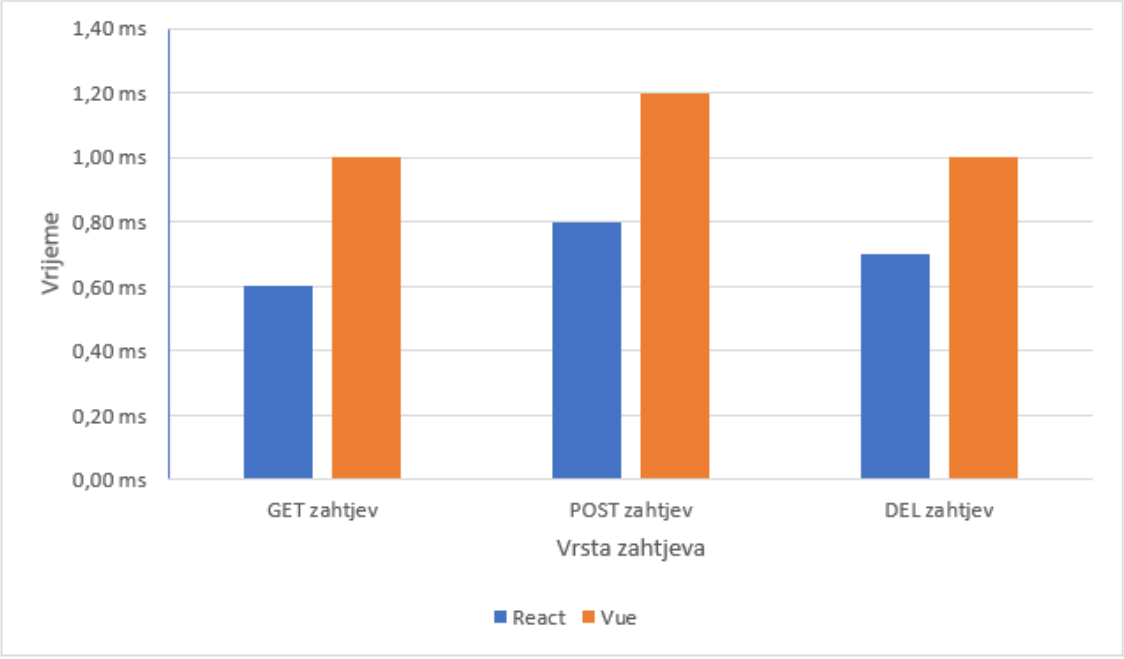
U ovome poglavlju će se analizirati usporedba performansi React *hook*-ova i Vue Composition API *hook*-ova [31]. Za testiranje performansi, istraživači su se fokusirali na dvije vrijednosti. Trajanje generiranja (engl. *render*) i ponovnog generiranja (engl. *re-render*) mjereno u milisekundama te vrijeme potrebno za uspješno dohvaćanje API zahtjeva (isto u milisekundama). Mjerenje se izvršavalo tri puta. Jednom za stvaranje podataka, jednom za čitanje podataka i jednom

za brisanje podataka. Za API se koristio MySQL kao baza podataka. Lista API zahtjeva je prikazana tablicom 5.2.

Tablica 5.2. Lista API zahtjeva

Vrsta API zahtjeva	Opis
GET – dohvati sve radnike	Dohvaća sve podatke radnika. Struktura tablice u bazi podataka sadrži ime, poziciju, vještine i prebivalište.
POST – dodaj ponudu za posao	Dodaje ponudu za posao u bazu podataka. Struktura tablice u bazi podataka sadrži ID radnika koji se povezuje s radnikom iz tablice radnika, naziv posla, opis posla i plaća.
DEL – obriši ponudu za posao	Briše ponudu za posao iz baze podataka.

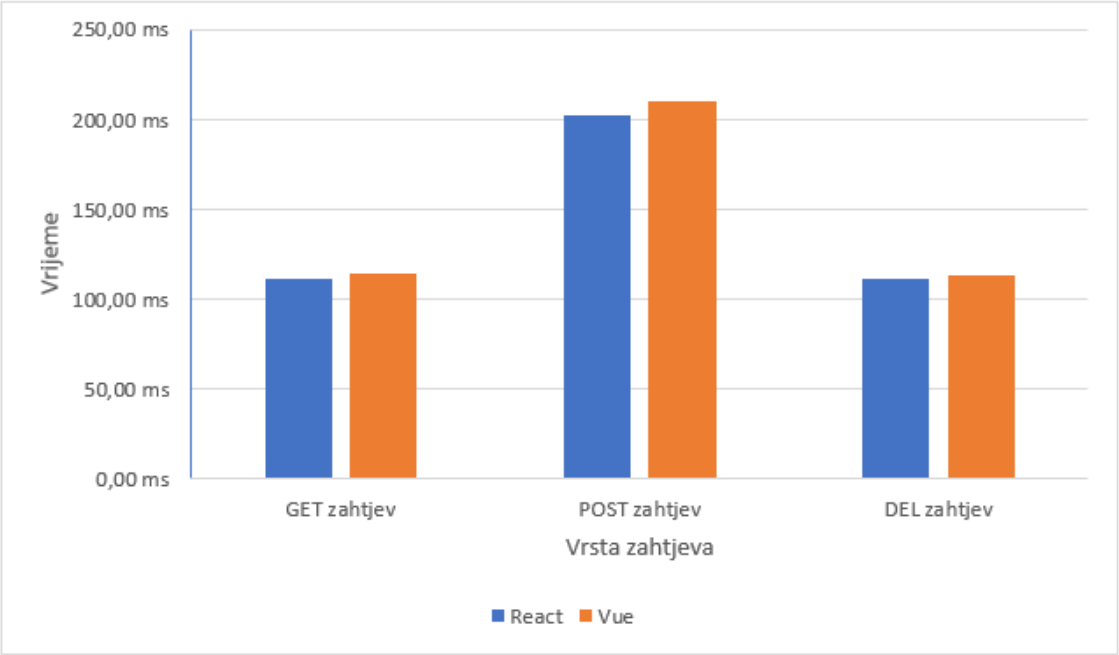
Nakon pokretanja API-ja i prototipa lokalno, oba prototipa (Vue i React) će generirati svoje komponente i u isto vrijeme slati zahtjeve. Zatim se za svaku vrstu zahtjeva mjeriti vrijeme za dohvaćanje i prikazivanje podataka. Na slici 5.1, po uzoru na [31], vidljivo je da su React *hook*-ovi bili brži za prosječno 0.37 ms pri generiranju komponenti koristeći GET zahtjev ili pri ponovnom generiranju komponenti gdje je došlo do izmjene podataka koristeći API zahtjev.



Sl. 5.1. Graf usporedbe *hook*-ova pri generiranju stranice nakon API zahtjeva

Na slici 5.2 je graf koji prikazuje vrijeme potrebno da *hook* dohvati API zahtjev. Ponovno su React *hook*-ovi bili malo brži od Vue *hook*-ova. Unatoč tome, na grafu su prikazani podatci temeljeni na standardnom API upitu. Zbog toga bi se uspoređeni podatci mogli razlikovati ako je

API imao veliko vrijeme odziva. Sveukupno React *hook*-ovi su bili nešto brži od Vue *hook*-ova. Iako se mjerenje uspješno izvršilo, postoje neka ograničenja pri mjerenju. U tome radu [31], istraživači su koristili prototipove umjesto potpuno funkcionalne web aplikacije. Također se mjerenje izvršavalo lokalno umjesto posluživanja stranice preko poslužitelja.



Sl. 5.2. Graf usporedbe *hook*-ova pri dohvaćanju API zahtjeva

5.3. Usporedba vremena manipulacije velikim brojem elemenata

U ovome naslovu će se izvoditi dva mjerenja. Prvo mjerenje se odnosi na manipulaciju velike količine DOM elemenata [32]. Izvodit će se funkcije stvaranja, brisanja i izmjenjivanja na 1000, 10000 i 50000 elemenata. Dobiveni rezultati su prikazani u tablicama 5.3 i 5.4.

Tablica 5.3. Vrijeme za funkcije stvaranja i brisanja

Broj elemenata	Vue	React
1000	8.35 ms	15.3 ms
10000	55.9 ms	275.05 ms
50000	364.9 ms	5255.6 ms

Tablica 5.4. Vrijeme za funkciju izmjenjivanja

Broj elemenata	Vue	React
1000	9.7 ms	7.8 ms
10000	55.6 ms	51.7 ms
50000	267.9 ms	179.4 ms

Pri stvaranju i brisanju elemenata, Vue je bio brži od React-a. Iz podataka je vidljivo da se pri stvaranju 50000 elemenata vrijeme za React drastično povećalo što upućuje na to da je Vue bolji pri generiranju velike količine elemenata. Pri izmjenjivanju elemenata React je bio brži, ali razlika u brzini je znatno manja nego pri stvaranju elemenata. Sveukupno u ovome testu bi prednost imao Vue.

Drugo mjerenje se izvodi generiranjem i manipuliranjem tablice s velikom količinom podataka. Ono se izvodi koristeći GitHub projekt za mjerenje performansi (engl. *benchmark*) JavaScript razvojnih okvira [33]. Test se izvodi stvaranjem niza određene veličine te njegovim popunjavanjem nasumično generiranim nizom od tri tekstualna podatka (engl. *string*). Izvodi se na nizovima s 1000 i 10000 podataka. U tablici 5.5 su prikazani dobiveni rezultati pokrenutog testa.

Tablica 5.5. Vrijeme izvođenja funkcija s velikom količinom podataka

Funkcija	Vue	React
Stvaranje 1000 redova	119.3 ms	126.2 ms ms
Izmjenjivanje svih 1000 redova	113.2 ms	126.7 ms
Izmjena svakog desetog reda od sveukupno 1000 redova	359 ms	399.5 ms
Odabir (označavanje) reda	53.7 ms	105.2 ms
Zamjena 2 reda od 1000	73.6 ms	494.1 ms
Brisanje jednog retka od 1000	28 ms	27.8 ms
Stvaranja 10000 redova	1149.2 ms	1488.1 ms
Pridruživanje tablice od 1000 redova tablici od 10000 redova	268.8 ms	322.5 ms
Brisanje svih 1000 redova	90 ms	101 ms
Geometrijska sredina	1.00 ms	1.46 ms

Vidljivo je da su vremena generiranja, brisanja i zamijene svih redova brža koristeći Vue, ali kao i u prijašnjim mjerenjima razlika je veoma mala. Ipak kod odabira određenog reda i kod zamjene dva reda je Vue bio znatno brži. Sveukupno je Vue bio brži pri radu s većom količinom podataka.

6. ZAKLJUČAK

Na izrađenim jednostraničnim web aplikacijama je objašnjen programski pristup za rješavanje nekih od problema s kojima se one suočavaju koristeći razvojna okruženja React i Vue.js. Problemi koji su objašnjeni su: osnovni pristup izrade komponenti, izrada usmjerivača, upravljanje globalnim stanjima i autentikacija. Objašnjenjem programskog koda mogu se vidjeti razlike, odnosno sličnosti pri rješavanje navedenih problema u različitim razvojnim okruženjima.

Kod izrađenih jednostraničnih web aplikacija mjerene su i uspoređivane performanse koristeći Google alat za mjerenje performansi. Dobiveni rezultati pokazuju da je Vue.js inačica aplikacije ponešto brža. Dobiveni rezultati uspoređeni s rezultatima mjerenja iz literature nisu se poklapali. U mjerenjima iz literature je React ponešto brži. U oba rada su razlike u performansama toliko male da se mogu zanemariti. Moguće je da bi dobiveni rezultati mjerenja bili nešto drugačiji ako bi se programski kod dodatno optimizirao, no razlike bi vjerojatno ponovno bile zanemarive. Izvedenim mjerenjima i dobivenim rezultatima dolazi se do zaključka da je za manje projekte razlika između korištenja Vue.js-a i React-a zanemariva, tj. odabir okvira za korištenje se može odabrati proizvoljno, jer neće znatno utjecati na performanse izrađene jednostranične web aplikacije. Također, mjerene su performanse programskog koda u radu s velikim brojem elemenata. Dobiveni rezultati pokazali su da je pri samom generiranju ili brisanju elemenata razlika u vremenu između razvojnih okruženja zanemariva. Međutim, kod rada s elementima iz velikog skupa elemenata, Vue.js je bio primjetljivo brži. Dobivenim rezultatima mjerenja rada s velikom količinom elemenata se zaključuje da bi za veće projekte Vue.js bio bolji. Također, može se zaključiti da za izradu manjih jednostraničnih web aplikacija neće biti primjetne razlike između korištenja razvojnih okruženja React i Vue.js, ali za veće aplikacije će Vue.js imati prednost.

LITERATURA

[1] „MDN Web Docs“, dostupno na: https://developer.mozilla.org/en-US/docs/Glossary/SPA [datum pristupa, 12.6.2023.]

[2] J. Oh, W.H. Ahn, S. Jeong, J. Lim and T. Kim, "Automated Transformation of Template-Based Web Applications into Single-Page Applications," 2013 IEEE 37th Annual Computer Software and Applications Conference, Kyoto, Japan, 2013, pp. 292-302.

[3] S. Ivanova and G. Georgiev, "Using modern web frameworks when developing an education application: a practical approach," 2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, 2019, pp. 1485-1491.

[4] „ScriptEvolve“, dostupno na: https://www.scriptevolve.com/blog/advantages-and-disadvantages-of-single-page-applications/ [datum pristupa, 12.6.2023.]

[5] „iTechArt“, dostupno na: https://www.itechart.com/blog/pros-cons-of-single-page-applications/ [datum pristupa, 12.6.2023.]

[6] A. Syromiatnikov and D. Weyns, "A Journey through the Land of Model-View-Design Patterns," 2014 IEEE/IFIP Conference on Software Architecture, Sydney, NSW, Australia, 2014, pp. 21-30.

[7] „GeeksforGeeks“, dostupno na: https://www.geeksforgeeks.org/difference-between-mvc-mvp-and-mvvm-architecture-pattern-in-android/?ref=lbp [datum pristupa, 17.6.2023.]

[8] „YourStory“, dostupno na: https://yourstory.com/mystory/mvp-vs-mvc-vs-mvvm/amp [datum pristupa, 17.6.2023.]

[9] „Patterns dev“, dostupno na: https://www.patterns.dev/posts/client-side-rendering/ [datum pristupa, 17.6.2023.]

[10] „Wolt“, dostupno na: https://wolt.com/en/hrv [datum pristupa, 22.6.2023.]

[11] „Glovo“, dostupno na: https://glovoapp.com/hr/hr/osijek/ [datum pristupa, 22.6.2023.]

[12] „Yummly“, dostupno na: https://www.yummly.com [datum pristupa, 22.6.2023.]

[13] „Airbnb“, dostupno na: https://www.airbnb.com [datum pristupa, 22.6.2023.]

[14] „Tutorialspoint“, dostupno na: https://www.tutorialspoint.com/vuejs/vuejs_overview.htm [datum pristupa, 24.6.2023.]

[15] „Vue.js“, dostupno na: https://vuejs.org/guide/introduction.html#the-progressive-framework [datum pristupa, 24.6.2023.]

[16] „Tutorialspoint“, dostupno na: https://www.tutorialspoint.com/reactjs/reactjs_introduction.htm [datum pristupa, 25.6.2023.]

[17] „Angular“, dostupno na: https://angular.io/guide/architecture [datum pristupa, 25.6.2023.]

[18] „Meteor“, dostupno na: https://docs.meteor.com/#what-is-meteor [datum pristupa, 25.6.2023.]

[19] „Techopedia“, dostupno na: https://www.techopedia.com/definition/28258/backbonejs [datum pristupa, 25.6.2023.]

[20] „Tutorialspoint“, dostupno na: https://www.tutorialspoint.com/emberjs/emberjs_overview.htm [datum pristupa, 25.6.2023.]

[21] K. Grifantini, "Knowing What You Eat: Researchers Are Looking for Ways to Help People Cope with Food Allergies," in IEEE Pulse, vol. 7, no. 5, Sept.-Oct. 2016, pp. 31-34.

[22] A. Hsu and Y. -F. Hsu, "Comprehensive Analysis of Dieting Apps: Effectiveness, Design, and Frequency Usage," 2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC), Torino, Italy, 2023, pp. 549-557.

[23] „MDN Web Docs“, dostupno na: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics [datum pristupa, 4.7.2023.]

[24] „MDN Web Docs“, dostupno na: https://developer.mozilla.org/en-US/docs/Web/CSS

[25] „MDN Web Docs“, dostupno na: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript [datum pristupa, 4.7.2023.]

[26] „Firebase“, dosutpno na: https://firebase.google.com [datum pristupa, 4.7.2023.]

[27] „Theme Junkie“, dostupno na: https://www.theme-junkie.com/what-is-figma/ [datum pristupa, 4.7.2023.]

[28] „Visual Studio Code“, dostupno na: https://code.visualstudio.com/docs [datum pristupa, 4.7.2023.]

[29] „Chrome Performance Tool“, dostupno na: https://developer.chrome.com/docs/devtools/performance/ [datum pristupa, 27.7.2023.]

[30] C. M. Novac, O. C. Novac, R. M. Sferle, M. I. Gordan, G. BUJDOSó and C. M. Dindelegan, "Comparative study of some applications made in the Vue.js and React.js frameworks," 2021 16th International Conference on Engineering of Modern Electric Systems (EMES), Oradea, Romania, 2021, pp. 1-4.

[31] J. Sianandar and I.B. Kerthyayana Manuaba, "Performance Analysis of Hooks Functionality in React and Vue Frameworks," 2022 International Conference on Information Management and Technology (ICIMTech), Semarang, Indonesia, 2022, pp. 139-143.

[32] R. N.V. Diniz-Junior et al., "Evaluating the performance of web rendering technologies based on JavaScript: Angular, React, and Vue," 2022 XVLIII Latin American Computer Conference (CLEI), Armenia, Colombia, 2022, pp. 1-9.

[33] „Js framework benchmark“, dotupno na: https://github.com/krausest/js-framework-benchmark [datum pristupa, 27.7.2023.]

SAŽETAK

U ovome radu su izrađene dvije identične inačice jednostranične web aplikacije za naručivanje jela iz restorana. Prva jednostranična web aplikacija je izrađena koristeći Vue.js, a druga koristeći React. Izrađene jednostranične web aplikacije omogućuju korisniku naručivanje jela iz restorana, a administratoru uređivanje tih jela. Korisnik ima mogućnost filtriranja jela po rasponu kalorija, sastojcima koje jelo mora sadržavati, sastojcima koje jelo ne smije sadržavati i po dijetama kojima jelo mora odgovarati. Također, korisnik ima mogućnost dodavanja jela u košaricu te naručivanje odabranih jela. Za obje inačice izrađenih jednostraničnih web aplikacija je prikazan i objašnjen programski kod za izradu komponenti, izradu usmjerivača, korištenje globalnog stanja i implementaciju autentikacije korisnika. Mjerenje performansi izrađenih jednostraničnih web aplikacija izvodilo se koristeći Chrome alat. Dobiveni rezultati mjerenja, kao i mjerenja iz literature analizirana su s ciljem usporedbe i analize korištenih razvojnih okruženja.

Ključne riječi: jednostranična web aplikacija, React, Vue.js, usporedba razvojnih okvira.

ABSTRACT

In this paper, two identical versions of single-page applications for ordering meals from a restaurant were made. The first single-page application was built using Vue.js, and the second using React. The created single-page applications allow a user to order meals from a restaurant and the administrator to edit those meals. The user has the option of filtering the meals by calorie range, ingredients that the meal must contain, ingredients that the meal must not contain, and diets that the meal must correspond to. Also, the user has the option of adding meals to the cart and ordering the selected meals. For the object version of the created single-page applications, the program code for creating components, creating a router, using global state and implementing authentication is shown and explained. The performance measurements of the created single-page applications were performed using the Chrome performance tool. The obtained results of the measurement, as well as the measurement from the literature, were analyzed in order to compare and analyze the used frameworks.

Keywords: single-page application, React, Vue.js, framework comparison.

ŽIVOTOPIS

Dominik Majdandžić rođen je 6. prosinca 1997. godine u Osijeku. Pohađao je OŠ Augusta Šenoe u razdoblju od 2004. do 2012. godine. Godine 2012. upisuje III. Gimnaziju u Osijeku gdje je sudjelovao na županijskom natjecanju iz programiranja. Godine 2016. završava srednju školu i upisuje preddiplomski sveučilišni studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija. Godine 2020. završava preddiplomski studij računarstva te upisuje sveučilišni diplomski studij računarstva.

_____

Potpis autora

PRILOZI

Prilog 1. Diplomski rad u datoteci docx

Prilog 2. Diplomski rad u datoteci pdf

Prilog 3. Programski projekt diplomskog rada