Razvoj web aplikacije Slatka Želja

Milanović, Matej

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

Matej Milanović

RAZVOJ WEB APLIKACIJE SLATKA ŽELJA

Diplomski rad

Pula, rujan 2024.

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

Matej Milanović

RAZVOJ WEB APLIKACIJE SLATKA ŽELJA

Diplomski rad

JMBAG: 0303061692, redovni student

Studijski smjer: Informatika

Predmet: Napredni algoritmi i strukture podataka

Znanstveno područje: Društvene znanosti

Znanstveno polje: Informacijske znanosti

Znanstvena grana: Informacijski sustavi i informatologija

Mentor: izv. prof. dr. sc. Tihomir Orehovački

Pula, rujan 2024.

SAŽETAK

Ovaj diplomski rad opisuje razvoj mrežne aplikacije "Slatka Želja", namijenjene obrtu specijaliziranom za proizvodnju i prodaju slastica. Aplikacija je odgovor na potrebe suvremenog tržišta gdje kupci, nedostatkom vremena, sve više pribjegavaju online kupovini. "Slatka Želja" omogućava narudžbu personaliziranih torti i kolača gdje kupci mogu predati sliku željenog proizvoda, kreirati kombinacije kolača po kilogramima i vidjeti vizualni prikaz s detaljima kao što su cijena i broj kalorija. Dodatno, aplikacija ima funkciju ograničavanja dnevnih narudžbi i određivanja datuma isporuke kako bi se održala visoka kvaliteta usluge. Aplikacija koristi modernu tehnologiju poput Visual Studio Code, MongoDB baze podataka i Vue JavaScript frameworka, što omogućuje fleksibilnost, skalabilnost i interaktivnost sučelja. Cilj aplikacije je unaprijediti poslovanje obrta pružanjem efikasne, jednostavne za korištenje digitalne platforme koja olakšava proces narudžbe i prodaje, i postavlja temelje za budući razvoj i širenje na nova tržišta. Digitalizacijom, tj. uvođenjem "Slatke Želje", obrt povećava svoju konkurentnost te proaktivno sudjeluje u digitalnoj transformaciji, otvarajući nove mogućnosti za rast i adaptaciju u dinamičnom poslovnom okruženju.

Ključne riječi: Backend, Frontend, Visual Studio Code, MongoDB, Node.js, Express.js, Mongoose, Vue, Torte, Kolači.

ABSTRACT

This master's thesis describes the development of the web application "Slatka Želja," designed for a craft business specialized in the production and sale of sweets. The application is a response to the needs of the modern market where time-constrained customers increasingly resort to online shopping. "Slatka Želja" allows for the ordering of personalized cakes and pastries, where customers can submit a picture of the desired product, create combinations of pastries by weight, and see a visual representation with details such as price and calorie count. Additionally, the application features a function to limit daily orders and set delivery dates to maintain high service quality. The application utilizes modern technology such as Visual Studio Code, MongoDB databases, and the Vue JavaScript framework, which provides flexibility, scalability, and interface interactivity. The goal of the application is to improve the craft business's operations by providing an efficient, easy-to-use digital platform that facilitates the ordering and sales process, and lays the groundwork for future development and expansion into new markets. With "Slatka Želja," the craft business not only enhances its competitiveness but also actively participates in digital transformation, opening up new opportunities for growth and adaptation in a dynamic business environment.

Keywords: Backend, Frontend, Visual Studio Code, MongoDB, Node.js, Express.js, Mongoose, Vue, Cakes, Pastries.

SADRŽAJ

1. UVOD

U radu se opisuje izrada mrežne aplikacije pod nazivom „Slatka Želja“, a naziv aplikacije nazvan je po obrtu koji se godinama bavi proizvodnjom i prodajom slastica raznih vrsta. Nalazimo se u vremenu u kojem ljudi zbog nedostatka slobodnog vremena pribjegavaju online trgovinama gdje je željeni proizvod moguće dobiti iz udobnosti svoga doma, što je inicijalna ideja ranije spomenute aplikacije, koja je zamišljena kao virtualna slastičarnica kreirana prema kupcu.

Brzina razvoja tehnologije kao i sam ubrzani način života prisilili su i male obrte na prilagodbu velikim promjenama današnje svakodnevice. Prvenstveni je cilj aplikacije zadovoljstvo samih kupaca, ali i povećanje konkurentnosti na tržištu, što bi u konačnici dovelo do povećanja obima posla samog obrta.

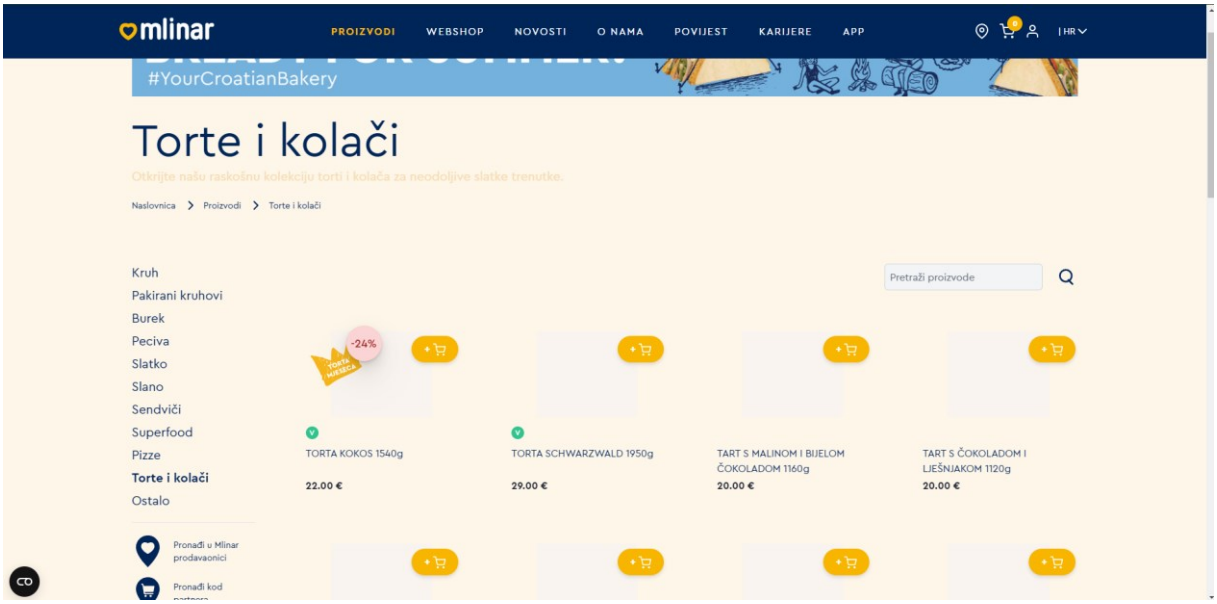
Usluga koju aplikacija pruža; narudžba personalizirane torte; temelji se na pristupačnijoj komunikaciji između kupca i prodavača prilikom koje kupac može priložiti fotografiju željenog izgleda proizvoda pri čemu bi pružatelj usluge dobio jasnu predodžbu kupčeve želje. Isto tako, aplikacija pruža mogućnost narudžbe kolača po pladnjevima, tj. po kilogramima pri čemu kupac samostalno sastavlja željenu kombinaciju proizvoda uz vizualni prikaz, broj kalorija, pojedinačnu cijenu kao i cijenu cijelog pladnja. Osim prodaje, aplikacija sadrži galeriju fotografija gotovih proizvoda; torti i kolača; kako bi kupac imao što jasniju predodžbu ponude, ali i kako bi ga se potaknulo na eventualnu kupnju proizvoda. Također, aplikacija ima ograničenje dnevnog broja narudžbi kako ne bi došlo do zasićenja, ali i kako bi se rasteretio pružatelj usluge zbog čega je prilikom izrade narudžbe potrebno odrediti datum isporuke – sve s ciljem održavanja kvalitete usluge.

Nastavak rada daje detaljno pojašnjene korištene tehnologije za izradu mrežne aplikacije te će se prikazati prototip kao i način na koji teče proces narudžbe te pregled svih narudžbi sa strane administratora i klijenta.

2. PREGLED ALTERNATIVNIH RJEŠENJA

2.1. MLINAR WEBSHOP

MLINAR pekarska industrija d.o.o. najveća je pekarska grupacija u Adria regiji i obuhvaća tržište od 16 milijuna stanovnika. Posluje na preko 300 prodajnih mjesta u Hrvatskoj, Sloveniji te Srbiji, a uz pekarske proizvode proizvodi i prodaje razne torte i kolače. Osim prodavaonica, u kojima je moguće naručiti i kupiti kolače i torte, imaju i svoj webshop za naručivanje proizvoda iz prodavaonice [1]. Ono po čemu se ova mrežna stranica razlikuje od mrežne stranice opisane u ovom diplomskom radu jeu tome da na internetskoj trgovini Mlinara nije moguće personalizirati svoju tortu već isključivo naručiti unaprijed izrađene proizvode koje imaju na stanju. Srodnost dvaju aplikacija jest to da za svaku tortu i kolač piše broj kalorija. slika internetske trgovine Mlinara prikazana je na slici 1.

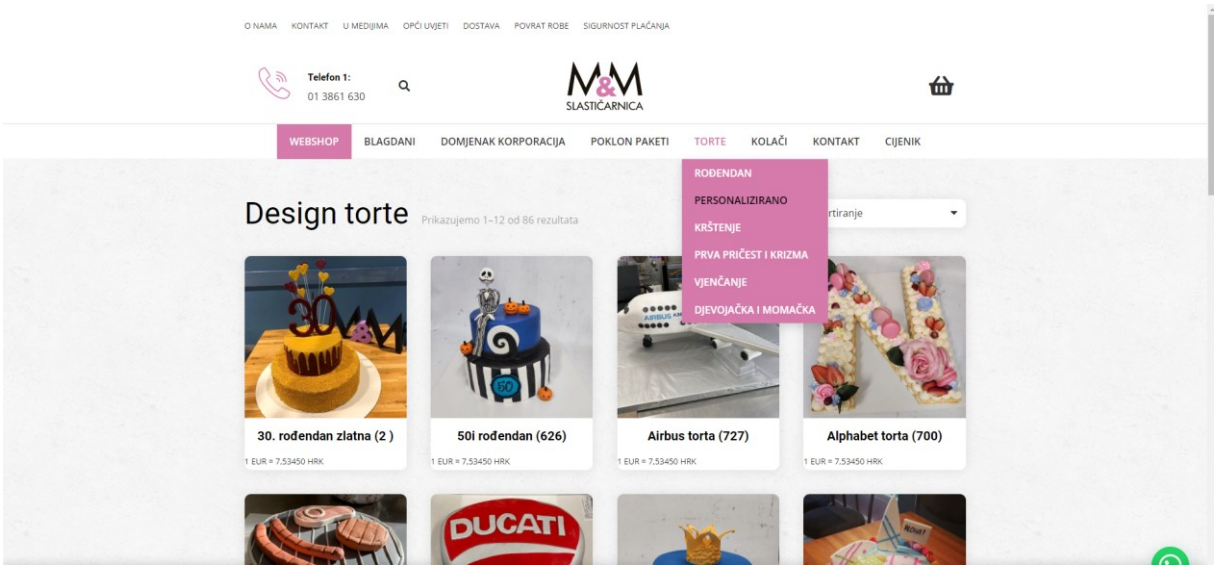


Slika 1. Mlinar webshop [19]

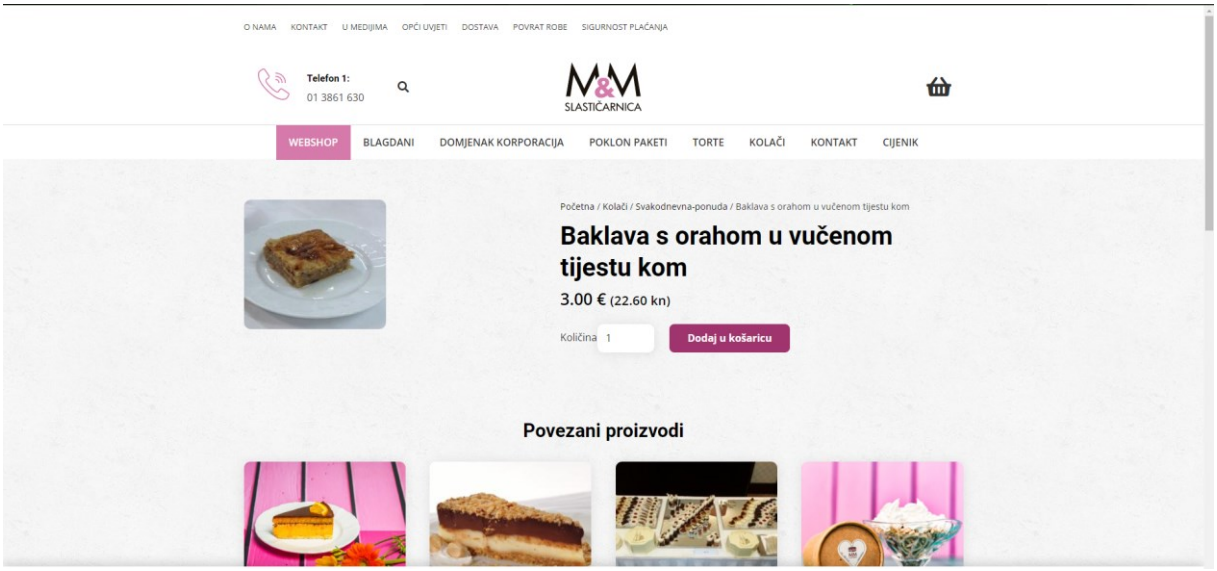
2.2. M&M SLASTIČARNICA WEBSHOP

M&M slastičarnica bavi se proizvodnjom i prodajom torti, slastica i slanih delicija za sve svečane prilike. Primarni je posao izrada i prodaja – torti i kolača pri čemu u ponudi imaju više od 100 artikala, tj. kolača i torti. Slastičarnica ima svoju internetsku trgovinu na kojoj je moguće naručiti torte i kolače [2]. Rade po principu narudžbi torti iz gotove ponude te personaliziranih torti. Razlika u odnosu na „Slatku Želju“ jest što kupac ne može sam personalizirati svoju tortu već postoje unaprijed izrađene torte za različite prigode, primjerice, rođendanske torte s unaprijed upisanim brojem, torta u obliku diplome i sl. kako je prikazano na slici 2. Ukoliko kupac želi sam kreirati tortu utoliko mora kontaktirati slastičarnicu koristeći

njihov blog ili Instagram profil. Slika 3 prikazuje kako prilikom naručivanja kolača iste je moguće naručiti na komad, ali ne postoji opcija slaganja pladnja kao ni uvid kako bi taj pladanj izgledao.



Slika 2. Personalizirane torte M&M slastičarnice [20]

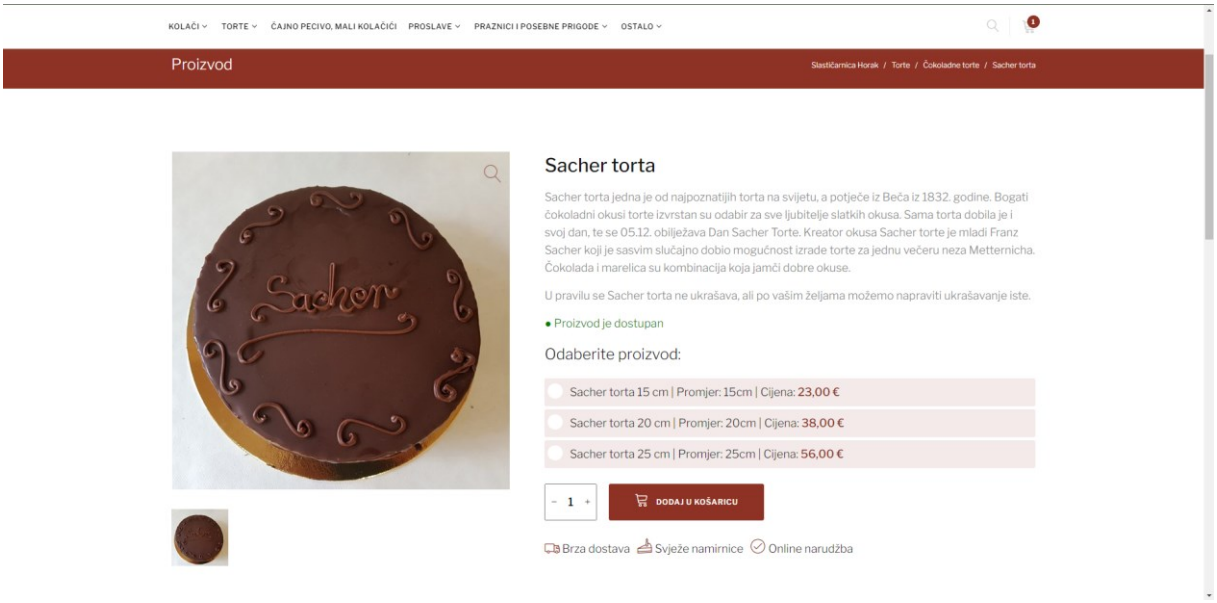


Slika 3. Naručivanje kolača M&M slastičarnice [21]

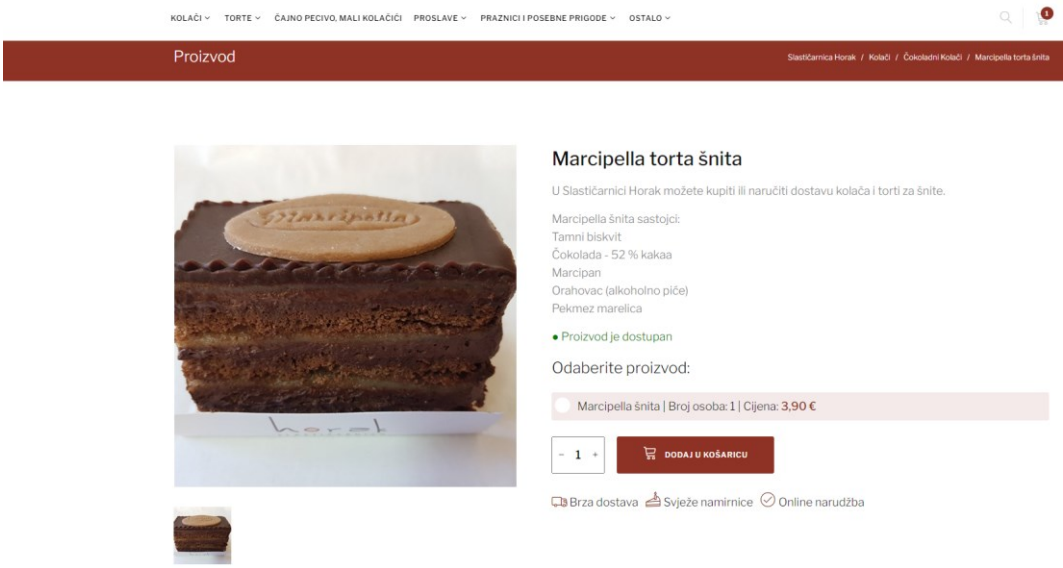
2.3 HORAK SLASTIČARNICA WEBSHOP

Horak slastičarnica bavi se proizvodnjom i prodajom slastica još od davne 1929. godine sa sjedištem u Zagrebu. Slastičarnica nudi raskošnu ponudu torti, kolača i čajnih peciva. Slastičarnica, također, ima svoju internetsku trgovinu pomoću koje je moguće naručivati razne proizvode [3]. Kod naručivanja torti potencijalnom kupcu nudi se već gotova torta te nije moguća personalizacija po vlastitoj želji kao što je slučaj sa „Slatkom Željom“. Prilikom

naručivanja torte dostupan je opis iste te njezina namjena, a korisniku je dana mogućnost izbora promjera torte koji utječe na cijenu što je prikazano na slici 4. Nadalje, kod naručivanja kolača moguće je naručiti kolače na komad te su dostupne informacije o sastojcima od kojih su kolači napravljeni. Međutim, kao što možemo vidjeti na slici 4, u odnosu na „Slatku Želju“ nije dostupna nutritivna vrijednost proizvoda niti je moguće složiti i vidjeti pladanj.



Slika 4. Naručivanje torte Horak slastičarnica [22]



Slika 5. Naručivanje kolača Horak slastičarnica [23]

3. KORIŠTENA RAZVOJNA OKRUŽENJA ZA IZRADU APLIKACIJE

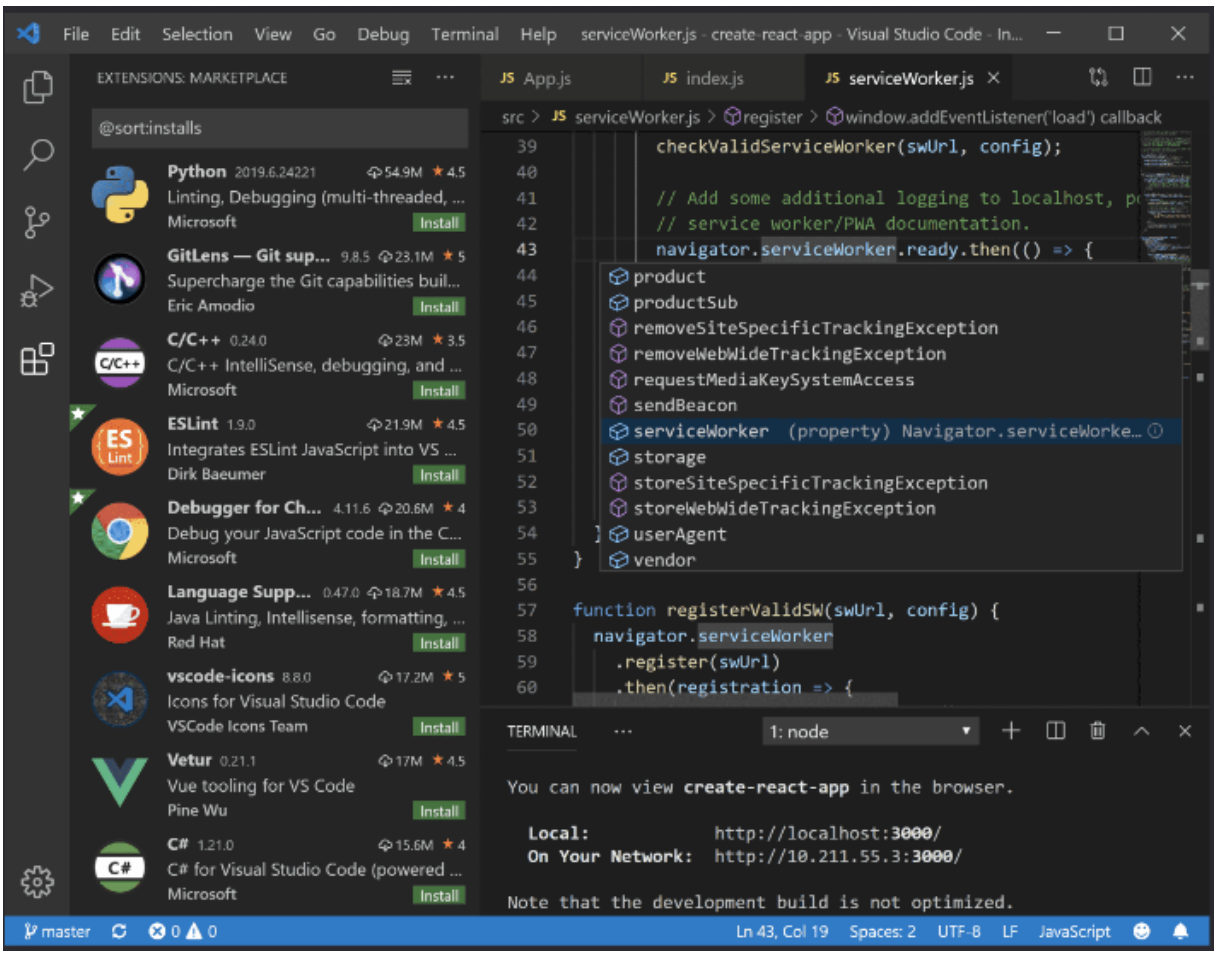
3.1. VISUAL STUDIO CORE

Visual Studio Code (VS Code) napredni je uređivač koda kojeg je razvio i objavio Microsoft 2015. godine, a koji se kroz godine nametnuo kao jedan od vodećih alata među programerima prvenstveno zbog svoje otvorenosti, modularnosti i široke podrške za različite programerske jezike kao što su JavaScript, Python, Java, C++, i dr. [4].

Jedna od ključnih značajki VS Code-a njegova je mogućnost proširenja pomoću tisuću dostupnih dodataka koji omogućavaju integraciju s drugim alatima i uslugama, poboljšavaju funkcionalnost uređivača ili dodaju nove jezike i tehnologije. Spomenuti dodaci uključuju alate za pronalaženje pogrešaka, podršku za rad s git repozitorijima, automatizaciju koda pa čak i *cloud* usluge [4].

IntelliSense funkcija je automatskog dovršavanja koda koja koristi kontekstualne tragove iz koda za pružanje relevantnih prijedloga, te je jedna od najkorisnijih funkcija u VS Code-u. Pomaže u smanjenju grešaka tijekom kodiranja, ubrzava razvojni proces, a uređivač podržava pronalaženje pogrešaka direktno iz sučelja što programerima omogućava brzo pronalaženje i otklanjanje grešaka u vlastitom kodu. VS Code prilagođen je za korištenje na različitim platformama uključujući Windows, macOS i Linux što ga čini izuzetno pristupačnim za širok spektar programera. Njegovo je korisničko sučelje jednostavno i čisto s fleksibilnom raspodjelom prozora i mogućnošću prilagođavanja izgleda i ponašanja uređivača prema osobnim preferencijama [5].

Osim tehničkih specifikacija, VS Code promiče kolaborativni rad omogućavajući programerima zajedničko korištenje projekata i koda u realnom vremenu pomoću funkcija kao što je LiveShare i sl. Spomenuta funkcija omogućuje programerima udruživanje te učinkovito rješavanje problema neovisno o fizičkoj udaljenosti [5].



Slika 6. Visual studio code[24]

3.2. NODE.JS

Node.js platforma je za izvođenje JavaScript koda izvan mrežnog preglednika što programerima omogućuje razvoj skalabilnih mrežnih aplikacija. Platforma izvorno predstavljena 2009. godine, od strane Ryana Dahla, brzo je stekla popularnost zahvaljujući svojoj sposobnosti da upravlja asinkronim operacijama i velikim brojem istovremenih veza [6].

Temeljna prednost Node.js-a jest „event-driven“ arhitektura i „non-blocking“ I/O model koji omogućavaju visoku propusnost i odaziv pri čemu je Node.js idealan za razvoj mrežnih aplikacija koje zahtijevaju intenzivnu obradu podataka u realnom vremenu kao što su igre ili chat aplikacije. Node.js koristi V8 JavaScript engine – isti pokreće Google Chrome – što osigurava brzu izvedbu JavaScript koda. Njegova modularna struktura omogućava programerima korištenje i dijeljenje paketa koda putem npm (Node Package Manager), najvećeg ekosustava slobodno dostupnih biblioteka u svijetu [6].

Platforma podržava razvoj na više platformi što programerima omogućuje pokretanje vlastitih aplikacija na Windows, macOS, Linux i drugim operacijskim sustavima. Također,

Node.js postao je omiljen među startupovima i velikim tvrtkama zbog svoje efikasnosti u obradi velikih količina podataka kao i zbog sposobnosti da se lako integrira u različite cloud platforme. Node.js promiče modularni pristup razvoju softvera omogućavajući developerima da izgrade svoje aplikacije koristeći male ponovno upotrebljive komponente [6].

3.3. MONGODB

MongoDB vodeća je NoSQL baza podataka poznata po svojoj fleksibilnosti i visokoj izvedbi pogodna posebno za upravljanje velikim količinama strukturiranih i nestrukturiranih podataka. Razvijena od strane MongoDB Inc. ova je baza podataka prvi put objavljena 2009. godine i od tada je postala popularna među developerima zbog svoje sposobnosti da olakša brz razvoj aplikacija [7].

Ključna značajka MongoDB-a njegova je schema-less struktura koja omogućava pohranu podataka u fleksibilnim JSON-sličnim dokumentima, a to znači kako se struktura podataka može mijenjati lako bez potrebe za izmjenom svih postojećih podataka. Takva struktura omogućava programerima da brže iteriraju i prilagođavaju svoje baze podataka dinamičkim zahtjevima modernih aplikacija [7].

MongoDB podržava indeksiranje svih polja u dokumentim pružajući snažne mogućnosti upita i optimizaciju performansi. Također, podržava agregacijske operacije koje omogućavaju izvođenje složenih transformacija podataka i analitiku izravno unutar baze što je posebno korisno za aplikacije koje zahtijevaju realno-vremensku analizu velikih količina podataka. Platforma pruža visoku dostupnost i skalabilnost kroz svoj model replikacije i mogućnosti širenja. Replikacija osigurava da su podaci dostupni čak i u slučaju hardverskog kvara dok širenje omogućava distribuciju podataka preko više servera što poboljšava performanse čitanja i pisanja [7].

MongoDB može se pokrenuti na različitim platformama uključujući Windows, macOS, i Linux, a integrira se s brojnim programskim jezicima što ga čini izuzetno pristupačnim za raznovrsne razvojne timove. Isto tako, MongoDB Atlas; cloud verzija baze; omogućava timovima da lako postave, upravljaju i skaliraju svoje MongoDB instance u oblaku pružajući automatizirane sigurnosne kopije, nadgledanje i konfiguraciju [7].

3.4. MONGOOSE

Mongoose je ODM (Object Data Modeling) alat za MongoDB i Node.js koji omogućava programerima strukturiranje podataka u bazi putem shema i modela. Biblioteka je posebno

dizajnirana kako bi olakšala rad s MongoDB bazama podataka u Node.js aplikacijama pružajući visoku razinu apstrakcije i udobnosti pri manipulaciji podacima. Koristeći Mongoose programeri mogu definirati shemu za svaki model podataka unutar aplikacije. Shema omogućava programerima jasno specificiranje koje vrste podataka će svako polje sadržavati uključujući tipove podataka kao što su String, Number, Date, Buffer, Boolean, Mixed, ObjectId, Array, Decimal128, Map, i Schema. Kontrola nad strukturom podataka ovakve preciznosti osigurava aplikaciji uporabu konzistentnih i validnih podataka smanjujući mogućnost greške [8].

Osim definiranja sheme Mongoose nudi i bogat set funkcionalnosti za validaciju, upite, transformaciju podataka i izgradnju indeksa. Ključna je značajka validacija jer osigurava da se podaci koji se unose u bazu podataka usklađuju s definiranim pravilima i ograničenjima što je vitalno za održavanje integriteta podataka. Mongoose, također, omogućava programerima izvršavanje složenih upita nad svojim dokumentima koristeći moćne metode kao što su find, findOne, findOneAndUpdate i sl. Navedene metode omogućavaju jednostavno filtriranje i manipulaciju podacima u bazi, a za dodatnu funkcionalnost Mongoose podržava middleware (poznate kao "pre" i "post" hooks) koji omogućavaju izvođenje funkcija prije ili nakon određenih operacija baze podataka, npr. spremanje ili brisanje dokumenta [8].

Prednost Mongoose-a jest i njegova sposobnost rada s asinkronim JavaScriptom koristeći obećanja ili async/await sintaksu što Mongoose integrira savršeno u moderni JavaScript ekosistem olakšavajući programerima razvoj efikasnih, čitljivih i skalabilnih aplikacija.

3.5. EXPRESS.JS

Express.js predstavlja minimalistički i fleksibilan mrežni aplikacijski framework za Node.js dizajniran da olakša izgradnju mrežnih aplikacija i API-ja. Express.js je zbog svoje jednostavnosti i moćnih funkcija, od predstavljanja, postao jedan od najpopularnijih frameworka unutar Node.js zajednice. Ključna je prednost Express.js-a njegova jednostavnost koja omogućava brzo postavljanje mrežnog poslužioca. Express pruža set robusnih značajki za mrežne i mobilne aplikacije kroz jednostavno i pristupačno API sučelje. Express omogućava programerima da izvršavaju različite funkcije kroz njegov middleware model poput provjera autentičnosti, obrada HTTP zahtjeva i upravljanje sesijama na modularan i organiziran način [9].

Middleware funkcije u Expressu mogu pristupiti zahtjevu i odgovoru od HTTP transakcija kao i "next" funkciji u middleware stogu što ishodišno pruža mogućnost izvršavanja kodova u serijama. Funkcionalnost koja je posebno korisna za izgradnju kompleksnih aplikacija jer omogućava lako upravljanje tokom podataka i logikom obrade zahtjeva. Express, također, podržava dinamično usmjeravanje pri čemu programeri mogu konstituirati sofisticirane rute koje se temelje na URL-u i HTTP metodama pružajući fleksibilne opcije za upravljanje različitim zahtjevima na server. Također, Express omogućava lako integriranje s raznim "view" engineima kao što su Pug (ranije zvan Jade), EJS i Mustache dajući programerima mogućnost generiranja HTML-a iz šablona na efikasan način. Express je dizajniran da bude lako proširiv: programeri mogu koristiti brojne dodatke dostupne u npm repozitoriju kako bi dodali dodatne funkcionalnosti ili integrirali Express s drugim bibliotekama i alatima kao što su baze podataka, ORM alati ili autentikacijski servisi [9].

3.6. VUE.JS

Vue.js progresivni je JavaScript framework za izgradnju korisničkih sučelja koji se posebno ističe po svojoj jednostavnosti i fleksibilnosti. Razvijen od strane Evana Youa i prvi put objavljen 2014. godine brzo je stekao popularnost među front-end developerima zbog svoje lakoće učenja i efikasnosti u razvoju modernih mrežnih aplikacija [10].

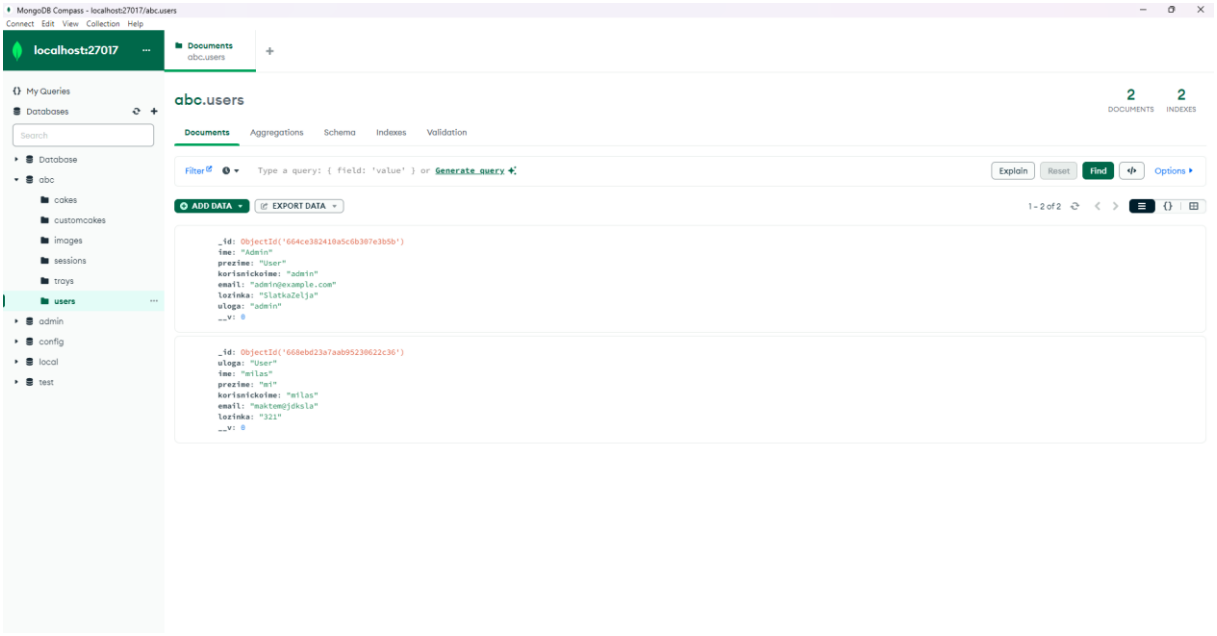
Vue.js temelji se na model-view-viewmodel (MVVM) arhitekturnom obrascu što omogućava razdvajanje logike aplikacije od korisničkog sučelja čime se olakšava upravljanje i održavanje koda. Glavna je prednost Vue.js-a reaktivnost, naime, kada se podaci promijene Vue.js automatski ažurira DOM (Document Object Model) kako bi odražavao te promjene bez potrebe za manualnim intervencijama. Također, podržava komponentni pristup razvoju što znači da se sučelja mogu graditi kao skup nezavisnih, ponovno upotrebljivih, komponenti. Ovako pristupačno, modularno strukturiranje, omogućava programerima da jednostavno razvijaju i testiraju složene aplikacije kao i da dijele i ponovno koriste kod unutar zajednice ili preko projekata. Ključna je funkcija Vue.js-a detaljna dokumentacija koja olakšava programerima da brzo započnu rad s frameworkom. Osim osnovnih funkcionalnosti Vue.js pruža i napredne opcije kao što su Vue Router za izgradnju spa (single-page application) aplikacija i Vuex za upravljanje stanjima aplikacije na centraliziran način [10].

Nadalje, Vue.js poznat je i po svojoj visokoj performansi i brzini što ga čini idealnim za razvoj interaktivnih korisničkih iskustava. Framework je dizajniran da bude progresivan što

znači da se može koristiti u malim dijelovima postojećih projekata ili kao temelj za izgradnju kompletnih front-end aplikacija.

3.7. MONGODB COMPASS

MongoDB Compass grafičko je sučelje dizajnirano kako bi pomoglo u upravljanju bazom podatka pri korištenju MongoDB-a. Koristi se za izvođenje upita, optimizaciju i analizu podatka, a isto tako omogućava analizirati shemu baze podatka kako bi identificirali obrasce i neke iznimke ako ih ima u podacima. Također, pruža metapodatke o kolekcijama koje imamo u bazi kao što su vrijednosti i učestalosti; sučelja za stvaranje i testiranje agregacijskih pipelinova koji su zapravo alati koji se koriste za trasformaciju i analizu podataka; podržava uvoz i izvoz podatka u JSON i CSV formatima, a njegov izgled možemo vidjeti na slici 7 [11].



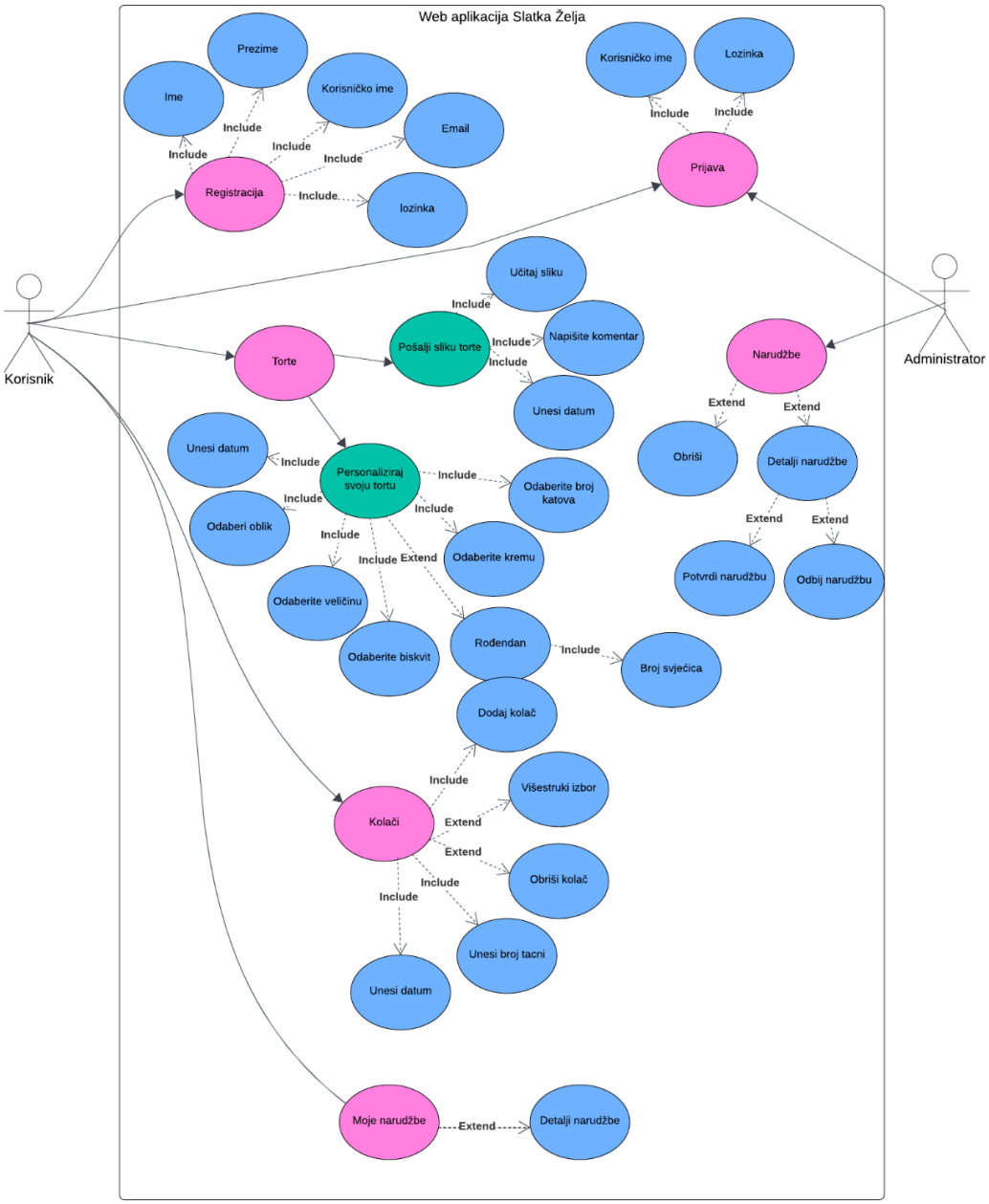
Slika 7. MongoDB Compass prikazano za korisnike mrežne aplikacije „Slatka Želja“

4. DIJAGRAMI

4.1. DIJAGRAM SLUČAJEVA KORIŠTENJA (USE CASE)

Slika 3. prikazuje dijagram slučajeva korištenja, a prikazuje dva korisnika aplikacije – jedan je sami korisnik aplikacije, a drugi je administrator aplikacije. Vidljivo je iz prikaza kako se korisnik mora prijaviti u aplikaciju, ali ako nije prijavljen mora se registrirati. Međutim, administratoru je račun kreiran posebno i postoji samo jedan te nije potrebna registracija već je dovoljno prijaviti se u aplikaciju sa svojim administratorskim podacima. Također, može se vidjeti kako administrator ima samo još jednu moguću opciju, a to je *pregledavanje svih narudžbi*. Isto tako, postoje dvije *Extend* veze koje su povezane s *opcijom brisanja* i *detalji narudžbe*, a opcija *detalji narudžbe* ima još dvije moguće opcije povezane *Extend* vezom gdje može *potvrditi* ili *odbiti narudžbu*.

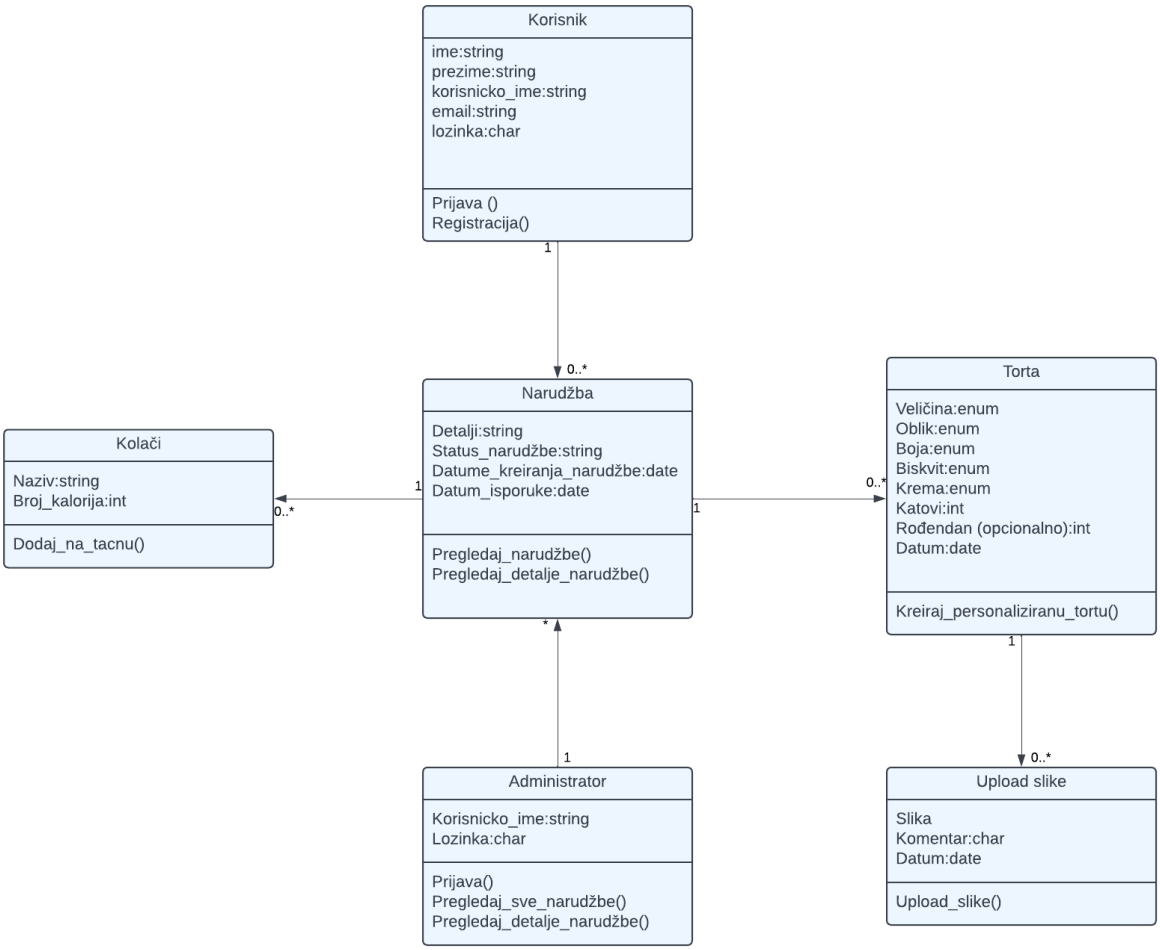
Klijent ima mnogo više mogućih opcija osim *registracije* i *prijave*, a to su: *torte*, *kolači* i *moje narudžbe*. Opcija *kolači* pruža dvije mogućnosti: *pošalji sliku torte* i *personaliziraj svoju tortu*. Opcija *pošalji sliku* torte imamo tri *Include* veze, a to su: *učitaj sliku*, *napišite komentar* i *odaberi datum*. Opcija *personaliziraj tortu* sadrži šest *Include* veza i jednu *Extend* vezu. *Include* veza zahtijeva unos: *oblik torte*, *odabir veličine*, *odabir biskvita*, *odabir kreme*, *odabir broja katova* i *unesi datum*, a kod *Extend* veze korisnik ima mogućnost odabira: ukoliko je torta namijenjena u svrhu rođendana utoliko se otvara još jedna *Include* veza u kojoj korisnik mora odabrati *broj svjećica*. Sljedeća moguća opcija za korisnika su *kolači*, a kod opcije *kolači* vidljive su tri *Include* i dvije *Extend* veze. *Include* veze odnosne se na *dodavanje kolača*, unos *broja pladnjeva* te *odabir datuma*, a *Extend* veza daje mogućnost *višestrukog dodavanja kolača na tacnu* i *brisanje kolača s tacne*. Korisnikova je zadnja i jedina mogućnost *pregledavanje vlastitih narudžbi* [12].



Slika 8. Dijagram tijeka

4.2. KLASNI DIJAGRAM

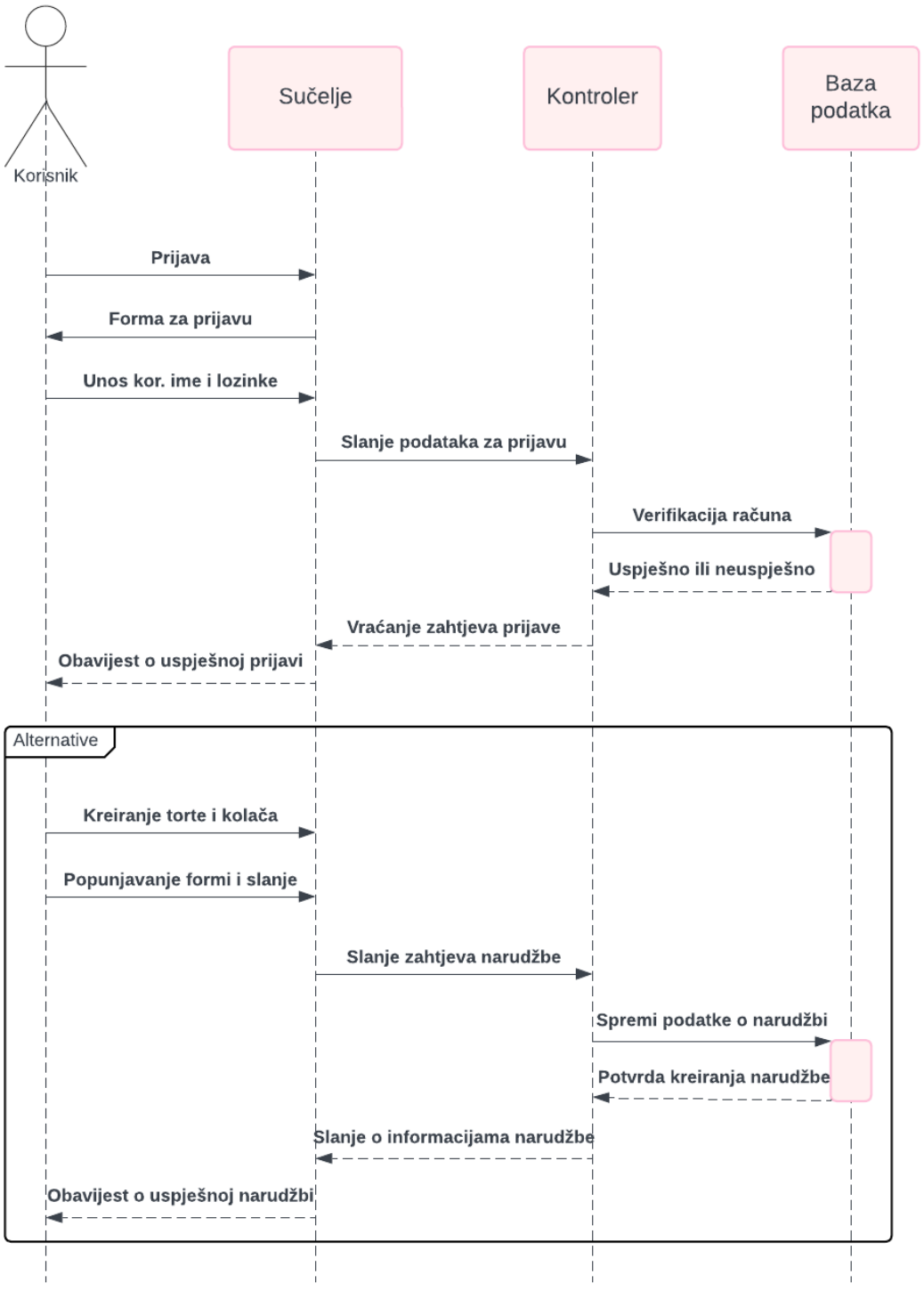
Klasni dijagram prikazuje ključne klase koje uključuju **Korisnik**, **Torta**, **Kolač**, **Narudžba**, **UploadSlike** i **Administrator**. Odnos je između **Korisnika** i **Narudžbe** označen kao "ima" što znači da jedan korisnik može imati nula ili više **Narudžbi**. Svaka **Narudžba** može sadržavati jednu ili više **Torti** i/ili **Kolača** pri čemu svaka **Torta** ili **Kolač** nužno pripada nekoj **Narudžbi**. **Torta** može biti povezana s **UploadSlike** omogućavajući korisnicima prilaganje slike svojih personaliziranih **Torti** dok svaka slika mora biti povezana s jednom **Tortom**. **Administrator** upravlja svim **Narudžbama** unutar sustava što uključuje mogućnost pregleda i ažuriranja statusa **Narudžbi** [13].



Slika 9. Klasni dijagram

4.3. SEKVENCIJALNI DIJAGRAM

Sekvencijski dijagram ilustrira niz koraka koje korisnik mora slijediti kako bi uspješno izvršio narudžbu. Proces započinje kada **Korisnik** posjeti početnu stranicu i odabere opciju za prijavu. **Web sučelje** mu zatim prikazuje obrazac za prijavu u koji **Korisnik** unosi svoje korisničko ime i lozinku. Nakon što su podaci upisani **Web sučelje** ih šalje **Kontroleru** koji podatke prosljeđuje **Bazi podataka** radi provjere autentičnosti. **Baza podataka** povratno šalje rezultat provjere **Kontroleru** koji potom informira **Web sučelje** o ishodu prijave. Ukoliko je prijava uspješna utoliko **Korisnik** dobiva pristup formularu za kreiranje personalizirane torte gdje upisuje detalje poput veličine, oblika i boje torte. Nakon ispunjavanja formulara **Web sučelje** šalje zahtjev za kreiranje torte **Kontroleru** koji informacije prosljeđuje **Bazi podataka**. Nakon što **Baza podataka** potvrdi spremanje torte **Kontroler** šalje potvrdu narudžbe **Web sučelju** koje zatim obavještava **Korisnika** o uspješnoj narudžbi [14].



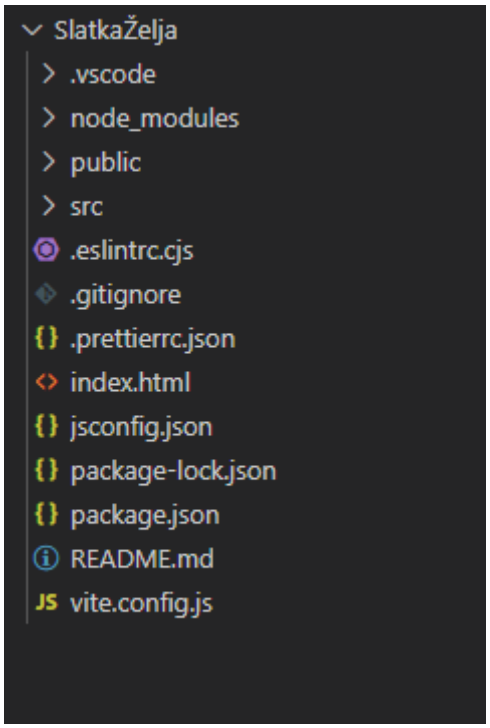
Slika 10. Sekvencijalni dijagram

5. IMPLEMENTACIJA

Kao što je ranije spomenuto aplikacija je izrađena u programskom okviru VUE. Nakon instalacije samoga programa Visual Studio Code potrebno je u terminalu upisati *npm install -g @vue/cli kako* bi uspješno instalirali Vue CLI globalno na računalu te omogućili pristup naredbama iz bilo kojeg direktorija u terminalu. Potom je potrebno navigirati do direktorija gdje se želi kreirati projekt te unijeti u terminal: *vue create ime-projekta* čime je kreiran projekt sa željenim imenom. Ukoliko se želi pokrenuti Vue program utoliko je potrebno u terminal upisati: *npm run serve* pri čemu se pokreće razvojni server koji se nalazi na adresi: *http://localhost:8080.* Također, postoji nekoliko dodataka za Visual Studio Code kao što su: Vetur, ESlint, Prettier i dr.

5.1. IMPLEMENTACIJA FRONTENDA

U ovom će se poglavlju pokazati arhitektura frontenda aplikacije, a svaki će dio biti pojedinačno pojašnjen. Sami izgled fronendskog dijela prikazuje slika 11.



Slika 11. Arhitektura frontenda

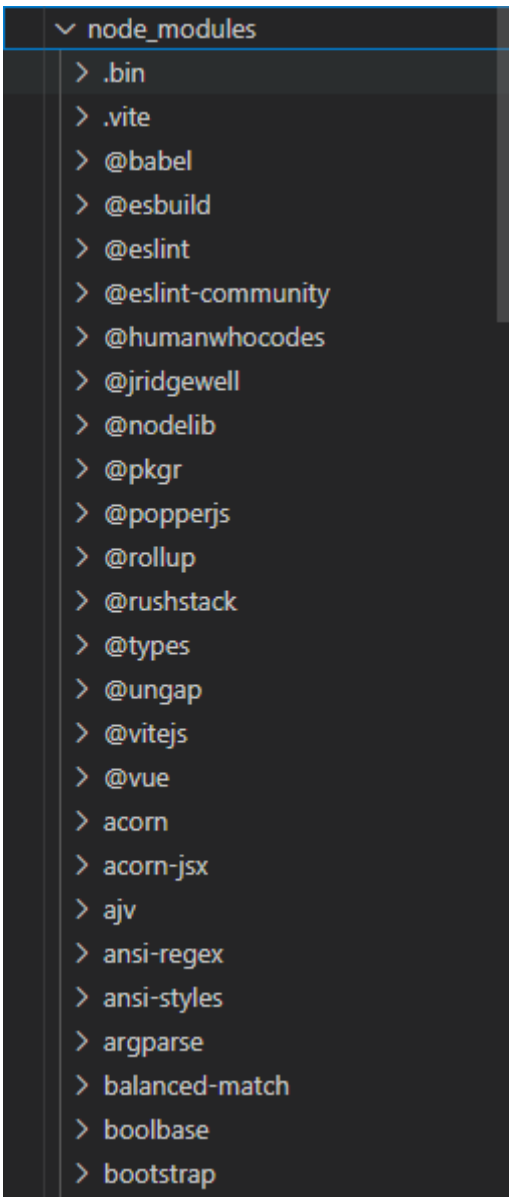
5.1.1.VSCODE

Unutar ove datoteke nalaze se dvije *json* datoteke: *extensions.json* i *settings.json*. Navedene dvije datoteke dolaze iz konfiguracijskih datoteka koje Visual Studio Code koristi za optimizaciju i prilagodbu razvojnog okruženja za specifičan projekt. Unutar *extensions.json*

daoteke nalazi se *Recommendations* koji specificira preporuke za dodatke koji bi trebali biti instalirani kako bi se održao optimalan rad. Koristi se, inače, u slučaju kada više ljudi radi na jednom projektu. Isto tako, unutar *Recomendationsa* nalazi se *Vue.volar*, *dbeumer.vscode.eslint*, *esbenp.prettier-vscode* koji služe za instaliranje dodataka poput *IntelliSense*, *refactoring* i dr. Također, u *setting.json* datoteci nalazi se *editor.codeActionsOnsave*, *editor.formatOnSave* i *editor.defaultFormatter*. Ukratko, navedene opcije konfiguriraju akcije kod spremanja datoteke te specificiranje koji će se *formatter* koristiti za formatiranje koda, a takve postavke pomažu u samom održavanju koda i njegovoj boljoj funkciji i optimizaciji smnjujući šansu za greške ili probleme oko samog formata koda.

5.1.2. NODE_MODULES

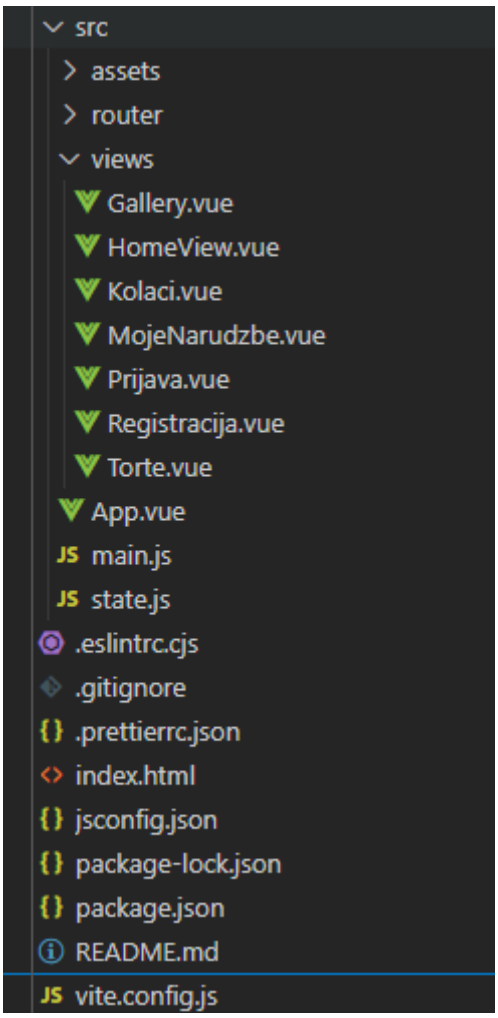
Unutar mape NODE_MODULES nalaze se ključni paketi svakog Node.js projekta koji koristi npm (Node Package Manager). Prilikom svakog izvršenja naredbe *npm install* paketa isti se paketi preuzimaju i spremaju u ovu mapu. Unutar mape NODE_MODULES svaki paket ima svoju zasebnu gdje se nalaze potrebne datoteke za taj paket, a zbog same kompleksnosti nije preporučljivo praviti ručne izmjene unutar mape NODE_MODULES. Pojedine mape, unutar NODE_MODULES (vidi sliku 12), zbog velikog broja datoteka unutar iste nije moguće prikazati na slici već samo određene.



Slika 12. Node_modules

5.1.3. SRC

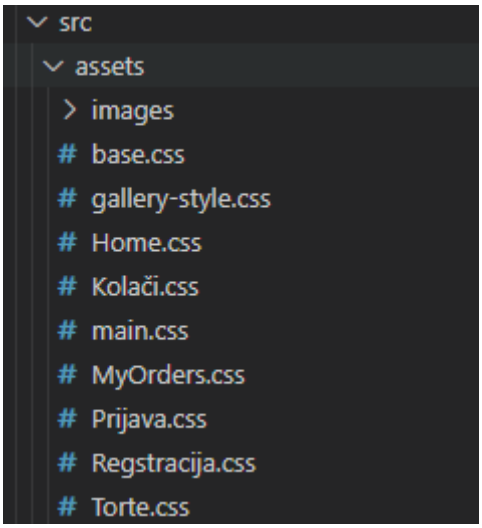
Unutar mape naziva „SRC“, u samom frontendu, nalaze se četiri mape: *assets*, *components*, *router* i *views* te *App.vue*, *main.js*, *state.js* što je vidljivo iz prikaza slike 13.



Slika 13. Arhitektura mape SRC

5.1.3.1. ASSETS

Unutar mape ASSETS nalazi se mapa pod nazivom *images* u kojoj se nalaze sve slike prikazane u navedenoj mrežnoj aplikaciji. Također, slika 14 prikazuje kako se osim mape *image* nalaze i *css* datoteke koje se vezivaju za određenu stranicu.



Slika 14. Arhitektura mape assets

5.1.3.2. ROUTER

Unutar mape ROUTER nalazi se datoteka *index.js* koja služi za konfiguriranje Vue Router-a za samu aplikaciju, a unutar datoteke se definiraju rute i povezane komponente za svaku stranicu u aplikaciji. CreateWebHistory koristi se za upravljanje navigacijom bez ponovnog učitavanja stranice što omogućava bolju korisničku interakciju. Svaka od ruta je definirana određenom putanjom, imenom i komponentom koja će se prikazati kada korisnik posjeti odabranu putanju. Trenutno se nalaze putanje za početnu stranicu, stranicu o tortama, stranicu o kolačima, registraciji, prijavi i ostalo. Organizacija na ovaj način olakšava upravljanje navigacijom unutar same aplikacije, a slika 15 prikazuje kod *index.js* datoteke.

```
import { createRouter, createWebHistory } from 'vue-router'
import HomeView from '../views/HomeView.vue'
import Torte from '../views/Torte.vue'
import Kolaci from '../views/Kolaci.vue'
import Registracija from '../views/Registracija.vue'
import Prijava from '../views/Prijava.vue'
import MojeNarudzbe from '../views/MojeNarudzbe.vue'
import Gallery from '../views/Gallery.vue'

const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes: [
    {
      path: '/',
      name: 'home',
      component: HomeView
    },
    {
      path: '/about',
      name: 'about',
      component: () => import('../views/AboutView.vue')
    },
    {
      path: '/torte',
      name: 'torte',
      component: Torte
    },
    {
      path: '/kolaci',
      name: 'kolaci',
      component: Kolaci
    },
    {
      path: '/registracija',
      name: 'registracija',
      component: Registracija
    },
    {
      path: '/prijava',
      name: 'prijava',
      component: Prijava
    },
    {
      path: '/moje_narudzbe',
      name: 'moje_narudzbe',
      component: MojeNarudzbe
    },
    {
      path: '/galerija',
      name: 'galerija',
      component: Gallery
    }
  ]
})

export default router
```

Slika 15. index.js

5.1.3.4. VIEWS

Unutar mape VIEWS nalazi se sedam *Vue* datoteka, a to su: *Gallery.vue, HomeView.vue, Kolaci.vue, MojeNarudzbe.vue, Prijava.vue, Registracija.vue, Torte.vue*. Svaka od navedenih *Vue* datoteka prikazuje određenu stranicu ili obrađuje određenu stavku na samoj web aplikaciji. Datoteka *Registracija.vue*, primjerice, koristi se za registriranje korisnika kako bi se mogli prijaviti u aplikaciju. Slika 16. prikazuje kod koji obrađuje samu registraciju korisnika, a sastoji se od polja za unos korisničkih podatka kao što su: *ime*, *prezime*, *korisničko ime*, *e-mail* i *lozinka*. Prilikom slanja obrasca metoda *submitForm* provjerava ispravnost e-mail-a i usklađenost lozinki prije slanja podatka na server pomoću HTTP POST zahtjeva. Ukoliko je registracija uspješna utoliko se korisničko ime spremna, a za prikaz poruke je li registracija uspješna ili nije koristi se metoda *showMessage*.

```
methods: {
  showMessage(message, success = true) {
    if (success) {
      this.successMessage = message;
      setTimeout(() => {
        this.successMessage = '';
        this.clearInputFields();
        if (success) {
          this.$router.push('/');
        }
      }, 3000);
    } else {
      this.errorMessage = message;
      setTimeout(() => {
        this.errorMessage = '';
      }, 3000);
    }
  },
  clearInputFields() {
    this.formData.ime = '';
    this.formData.prezime = '';
    this.formData.korisnickoime = '';
    this.formData.email = '';
    this.formData.lozinka = '';
    this.formData.lozinkaPotvrda = '';
  },
  submitForm() {
    const { email, lozinka, lozinkaPotvrda } = this.formData;

    if (!email.includes('@')) {
      this.showMessage('Molimo unesite ispravnu email adresu!', false);
      return;
    }
    if (lozinka !== lozinkaPotvrda) {
      this.showMessage('Lozinke se ne podudaraju. Molimo unesite iste lozinke.', false);
      return;
    }

    console.log('korisnicki podaci: ' , this.formData);
    fetch('http://localhost:3000/slatkaZelja/create', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/x-www-form-urlencoded',
      },
      body: new URLSearchParams(this.formData)
    })
    .then(response => {
      if (!response.ok) {
        throw new Error('Network response was not ok');
      }
      return response.json();
    })
    .then(data => {
      localStorage.setItem('korisnik', this.formData.korisnickoime);
      console.log(`User ${this.formData.korisnickoime} logged in.`);
      window.location.replace('/');
      this.showMessage(data.message, data.status);
    })
    .catch(error => {
      this.showMessage('Došlo je do greške prilikom registracije. Molimo pokušajte ponovno.', false);
```

Slika 16. Views

Datoteka *Prijava.vue* sadrži kod koji se koristi za prijavu u aplikaciju dok sama forma sadrži dva ulazna polja za *unos korisničkog imena* i *lozinke* te sami gumb za slanje forme. Korištenjem *v-modela* omogućava se automatsko ažuriranje podataka u *Vue.js* modelu podataka. Slika 17 prikazuje metodu *submitForm* koja se aktivira nakon što korisnik pošalje samu formu. Dakle, metoda šalje *POST* zahtjev na server s korisničkim podacima u *JSON* formatu. Ukoliko je prijava uspješna utoliko se korisničko ime sprema u lokalnu pohranu, a

korisnik se automatski preusmjerava na naslovnu stranicu mrežne aplikacije, ali ako prijava nije uspješna tada se pokazuje poruka o grešci. Stilovi su uvezeni iz odvojene CSS datoteke.

```vue
<template>
  <form id="login-form" @submit.prevent="submitForm">
    <div class="container">
      <p><b>Upišite korisničko ime i lozinku kako bi ste se prijavili</b></p>
      <hr>
      <label for="korisnickoime"><b>Korisničko ime</b></label>
      <input type="text" placeholder="Unesite korisničko ime" v-model="formData.korisnickoime" id="korisnickoime" required>

      <label for="lozinka"><b>Lozinka</b></label>
      <input type="password" placeholder="Unesite lozinku" v-model="formData.lozinka" id="lozinka" required>

      <button type="submit">Prijava</button>
    </div>
  </form>
</template>

<script>
export default {
  data() {
    return {
      formData: {
        korisnickoime: '',
        lozinka: ''
      }
    };
  },
  methods: {
    submitForm() {
      fetch('http://localhost:3000/slatkaZelja/login', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify(this.formData)
      })
      .then(response => {
        if (!response.ok) {
          throw new Error('Network response was not ok');
        }
        localStorage.setItem('korisnik', this.formData.korisnickoime);
        console.log('local storage: ', localStorage)
        console.log(`User ${this.formData.korisnickoime} logged in.`);
        window.location.replace('/');
        return response.json();
      })
      .catch(error => {
        alert('An error occurred: ' + error.message);
      });
    }
  }
};
</script>

<style>
@import '.././assets/Prijava.css';
</style>
```

Slika 17. Prijava.vue

Datoteka *Torte.vue* sadrži kod za dva skočna prozora od kojih je jedan za personalizirano kreiranje torte, a drugi skočni prozor je za učitavanje i slanje slike i komentara. Skočni prozor za personalizaciju torte koristi *v-show* za uvjetno prikazivanje određenih elemenata na temelju stanja aplikacije što je vidljivo na slici 18. Metode *updateCakeShape, updateCakeSize, updateCakeColor* i ostale koriste se kako bi se na osnovu korisnikovih odabira mijenjala sama cijena, a metoda *calculatePrice* računa cijenu na osnovu odabranih opcija. Dakle, cijena se prilagođava samim odabirom korisnika što je vidljivo na slici 19. Također,

funkcija *mounted* koja se pokreće automatski kada se komponenta učita, koristi se za provjeru statusa sesije kao i za dohvat podatka o trenutno prijavljenom korisniku.

```
          <label for="kat">Odaberite broj katova torte:</label>
          <select id="kat" v-model="cakeLayers" @change="updateCakeLayers">
            <option value="" disabled selected>Odaberite broj katova</option>
            <option value="JedanKat">1 kat</option>
            <option value="DvaKata">2 kata</option>
            <option value="TriKata">3 kata</option>
          </select>

          <label for="birthday">Torta za rođendan:</label>
          <input type="checkbox" id="birthday" v-model="isBirthdayCake" @change="toggleBirthdayOptions">
          <div v-show="isBirthdayCake">
            <label for="age">Broj svjećica:</label>
            <input type="number" id="age" v-model="birthdayCandles" min="1" @change="updateBirthdayCandles">
            <button type="button" @click="addCandles">Dodaj</button>
          </div>

          <p id="totalPrice">Ukupna cijena: {{ totalPrice }} EUR</p>
          <label for="pickupDate">Datum preuzimanja:</label>
          <input type="date" id="pickupDate" v-model="pickupDate" >

          <button type="submit">Potvrdite odabir</button>
          <button type="button" class="btn btn-secondary" @click="closeModal">Odustani</button>
        </div>
      </form>
    </div>
  </div>
</div>
```

Slika 18. Personalizacija torte

```
updateCakeColor() {
  console.log("Cake color updated to:", this.cakeColor);
},
updateCakeCream() {
console.log("Cake cream updated to:", this.cakeCream);
this.calculatePrice();
},
updateCakeBiscuit() {
console.log("Cake cream updated to:", this.cakeBiscuit);
this.calculatePrice();
},
updateCakeLayers() {
console.log("Cake cream updated to:", this.cakeLayers);
this.calculatePrice();
},
updateBirthdayCandles() {
console.log("Cake cream updated to:", this.birthdayCandles);
this.calculatePrice();
},

toggleBirthdayOptions() {
  console.log("Birthday options toggled. Is birthday cake:", this.isBirthdayCake);
  if (!this.isBirthdayCake) {
    this.birthdayCandles = 0;
  }
},
addCandles() {
  console.log(this.birthdayCandles, "candles added.");
},
calculatePrice() {
let basePrice = 20;

if (this.cakeShape === "Pravokutna") {
  basePrice += 5;
}

if (this.cakeSize === "Srednja") {
  basePrice += 10;
} else if (this.cakeSize === "Velika") {
  basePrice += 20;
}

if (this.cakeBiscuit === "Badem") {
  basePrice += 5;
} else if (this.cakeBiscuit === "Orah") {
  basePrice += 7;
```

Slika 19. Kreiranje cijene torte

Kod učitavanja slika korisnik odabire sliku, nakon čega se metoda *handleFileChange* pokreće i sprema datoteku u *Vue model*. Nakon što se forma pošalje metoda *submitImageForm* se pokreće te priprema *FormData* objekt koji dodaje sliku i druge relevantne informacije prilikom čega se šalje na server koristeći metodu *fetch* što je vidljivo na slici 20.

```
    handleFileChange(event) {
    this.imageFile = event.target.files[0];
  },
      submitImageForm() {
        let imageComment = this.imageComment;
  let commenterName = 'Ivan';
  let imagePickupDate = this.imagePickupDate;
  let imageFile = this.imageFile;

  console.log('Image File:', imageFile);
  console.log('Image Comment:', imageComment);
  console.log('Pickup Date:', imagePickupDate);

  const formData = new FormData();
  if (imageFile) {
      formData.append('image', imageFile);
  } else {
      console.error('No file selected');
  }
  console.log("before form data user id", this.userId)
  formData.append('userId', this.userId);
  formData.append('comment', imageComment);
  formData.append('pickupDate', imagePickupDate);

  for (let pair of formData.entries()) {
      console.log(pair[0]+ ', ' + pair[1]);
  }

      console.log(formData);
      fetch('http://localhost:3000/upload/', {
          method: 'POST',
          body: formData
      })
      .then(response => {
          if (response.ok) {
          alert('Slika uspješno uploadana!');
          } else {
          throw new Error('Došlo je do greške pri uploadu slike.');
          }
      })
      .catch(error => alert(error.message))
      .finally(() => {
          this.closeImageModal();
          this.resetImageForm();
      });
      },
      resetImageForm() {
      this.imageFile = null;
```

Slika 20. Učitavanje slike i komentara

Datoteka *kolaci.vue* prikazuje kod na fronend stranici koji obrađuje naručivanje kolača po pladnjevima. Svaki *slot* označava jedno mjesto na pladnju za pojedini kolač te je po tome kreirana metoda gdje se *slotovi* dinamički kreiraju u *div* elementu referenciranom kao *tray*. Slika 21 prikazuje kako svaki *slot* može biti odabran kako bi se dodao određeni kolač.

```
  setupTraySlots() {
    const tray = this.$refs.tray;
    if (!tray) {
      console.error('Element with ref "tray" not found.');
      return;
    }

    for (let i = 0; i < 24; i++) {
      const slot = document.createElement('div');
      slot.className = 'slot';
      slot.id = `slot-${i + 1}`;
      tray.appendChild(slot);

      slot.addEventListener('click', () => {
        if (this.multiSelectMode) {
          slot.classList.toggle('selected');
        } else {
          document.querySelectorAll('.slot.selected').forEach(s => s.classList.remove('selected'));
          slot.classList.add('selected');
        }
        this.popupVisible = true;
        this.$refs.currentSlot.value = slot.id;
        this.updateCaloriesInfo();
      });
    }
  }
},
computed: {
  selectedSlot() {
    return document.querySelector('.slot.selected');
  }
},
```

Slika 21. Postavljanje slotova

Datoteka *HomeView.vue* sadrži početnu stranicu mrežne aplikacije, a sastoji se od *slideshowa* koji se nakon dolaska na početnu stranicu pokreće automatski te se svaka slika mijenja nakon četiri sekunde. Korištena metoda naziva se *currentAutoSlide* – prikazano na slici 17 – kada indeks dosegne kraj niza slike, resetira se na nulu i tako ciklus mijenjanja slika pokreće ispočetka. Metoda *changeSlides* koristi se za ručno mijenjanje trenutnog slide-a kada korisnik odabere navigacijske strelice. Dakle, korisnik sam može mijenjati slike u *slideshowu*. Slika 22 nudi prikaz gdje početna stranica sadrži galeriju slika prijašnjih radova gdje se prilikom odabira slike iste može pregledavati pomoću metode *plusSlides* koja omogućava navigaciju slika unutar skočnog prozora. Metoda prima broj (n) koji označava koliko pozicija se treba pomaknuti naprijed ili nazad.

```
plusSlides(n) {
  this.currentSlide += n;
  if (this.currentSlide >= this.modalSlides.length) {
    this.currentSlide = 0;
  } else if (this.currentSlide < 0) {
    this.currentSlide = this.modalSlides.length - 1;
  }
},
startAutoSlides() {
  setInterval(() => {
    this.currentAutoSlide++;
    if (this.currentAutoSlide >= this.slides.length) {
      this.currentAutoSlide = 0;
    }
  }, 4000);
},
changeSlides(n) {
  this.currentAutoSlide += n;
  if (this.currentAutoSlide >= this.slides.length) {
    this.currentAutoSlide = 0;
  } else if (this.currentAutoSlide < 0) {
    this.currentAutoSlide = this.slides.length - 1;
  }
},
```

Slika 22. Automatski slideshow

Datoteka *MojeNarudzbe.vue* sadrži sve narudžbe za koje je korisnik izvršio narudžbu. Sama se stranica sastoji od tablice na kojoj se nalaze informacije kao što su broj narudžbe, datum kreiranja, datum preuzimanja, status i gumb prikaza detalja narudžbe. Korištena metoda je *fetchOrders* koja dohvaća listu narudžbi korisnika sa servera i filtrira ih prema ID-korisnika. Metoda *showOrderDetails* postavlja sadržaj za skočni prozor koji se prikazuje nakon klika na gumb detalji narudžbe. Metoda *FetchCurrentUserDetails* dohvaća detalje o trenutno prijavljenom korisniku i učitava detalje sa servera. Također, korišten je *Mounted* koji automatski poziva *fetchOrders* i *fetchCurrentUserDetails*, a cijeli je kod vidljiv na slici 23.

```
methods: {

  async fetchOrders() {
    try {
      const response = await fetch('http://localhost:3000/my-orders', { credentials: 'include' });
      if (!response.ok) {
        throw new Error('Failed to fetch orders');
      }
      const data = await response.json();
      console.log(data, this.userId);
      console.log("response",data)
      const userOrders = data.filter(order => order.userId._id == this.userId);
      if (userOrders.length > 0) {
        this.orders = userOrders;
      } else {
        console.log("No orders found for user:", this.userId);
      }
    } catch (error) {
      console.error('Error loading orders:', error);
    }
  },
  showOrderDetails(order) {
    this.orderDetailsContent = this.prepareOrderDetails(order);
    this.$refs.detailsModal.style.display = 'block';
  },
```

Slika 23. Moje narudžbe

Dokument po nazivom *Gallery.vue* sadrži popis svih narudžbi, a pristup ima isključivo administrator koji ima uvid u sve narudžbe. Sama tablica prikazana je na sličan način kao u dokumentu *MojeNarudzbe.vue* gdje se podaci učitavaju iz baze o naručenim kolačima i tortama, a u samome kodu se koristi *v-for* koji se upotrebljava za iteriranje kroz sortiranu listu narudžbi. Metoda *updateOrderStatus* koristi se kako bi administrator mogao promjeniti stanje narudžbe budući da su sva stanja narudžbi postavljena kao *na čekanju* te nakon što administrator odluči želi li prihvatiti ili odbiti narudžbu stanje se narudžbe mijenja. Funkcija radi na način da se postavlja ID narudžbe koji će se ažurirati, nakon čega administrator odabire status u padajućem izborniku, nakon čega se šalje ID i novi status narudžbe. Nadalje, pokazuje se poruka koja govori je li promjena bila uspješna ili ne te ponovo učitavanje podatka o narudžbama, a cijeli je proces prikazan na slici 24.

```
async updateOrderStatus(orderId) {
  orderId = this.orderId;

  const status = this.selectedOrderStatus;

  console.log(JSON.stringify({ orderId, status }));
  fetch('http://localhost:3000/update-order-status', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({ orderId, status })
  })
    .then(response => {
      if (!response.ok) {
        throw new Error('Failed to update order status');
      }
      return response.json();
    })
    .then(data => {
      alert(data.message);
      this.closeModal();
      this.fetchAndDisplayOrders();
    })
    .catch(err => {
      console.error('Error updating order status:', err);
      alert('Error updating order status: ' + err.message);
    });
},
closeModal() {
  this.detailsModalVisible = false;
},
```

Slika 24. Sve narudžbe

Brisanje same narudžbe ne odvija se tako da se narudžba briše iz baze podatka nego na način da se „sakrije“ administratoru u slučaju kada bi imalo prevelik broj narudžbi koje su već obrađene te ne želi povijest naručivanja pri čemu je korištena metoda *deleteOrder* kojom se šalje POST zahtjev na endpoint koji označava brisanje, nakon čega slijedi ažuriranje, a narudžba ne bi trebala biti više vidljiva (vidi slika 25).

```
deleteOrder(id) {
  fetch(`http://localhost:3000/mark-order-as-deleted/${id}`, { method: 'POST' })
    .then(response => {
      if (!response.ok) {
        throw new Error('Failed to mark the order as deleted');
      }
      return response.json();
    })
    .then(data => {
      alert(data.message);
      this.fetchAndDisplayOrders();
    })
    .catch(err => {
      console.error('Error marking order as deleted:', err);
      alert('Error marking order as deleted: ' + err.message);
    });
},
```

Slika 25. Brisanje narudžbe

5.1.4. APP.VUE

Slika 26 opisuje datoteku *App.vue* koja sadrži kreirani skočni prozor pomoću kojega je moguće vidjeti podatke korisnika koji je prijavljen te ima mogućnost mijenjanja lozinke nakon unosa stare. Promjena stare lozine koristi metodu *updateUserDetails* prilikom čega se šalje zahtjev na server s ciljem ažuriranja korisničke lozinke. Koristi se *fetch API* za slanje POST zahtjeva koji uključuje staru i trenutnu lozinku kao tijelo zahtjeva, a u samoj se pripremi podatka nalazi indetifikator korisnika što je userID te stara i nova lozinka.

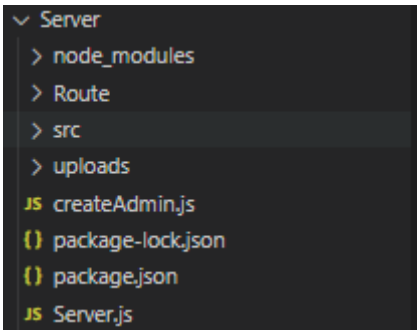
Također, u datoteci se prikazuju stranice na navigacijskoj traci ovisno o „ulozi“ prijavljene osobe , odnosno radi li se o korisniku ili o administratoru. Korišten je *v-if* kako bi se provjerilo radi li se o administratoru ili korisniku. Ukoliko se radi o korisniku utoliko se prikazuje stranica *moje narudžbe*, a ako se administrator prijavi tada se prikazuje stranica *gallery.vue* točnije *sve narudžbe*.

```
async updateUserDetails() {
  try {
    const response = await fetch('http://localhost:3000/update-user-password', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        userId: this.userId,
        oldPassword: this.oldPassword,
        newPassword: this.newPassword
      })
    });

    const data = await response.json();

    if (response.ok) {
      console.log('Password updated successfully:', data.message);
    } else {
      console.log('Failed to update password:', data.message);
    }
  } catch (error) {
    console.error('Error updating password:', error);
  }
  this.closeModal();
},
```

Slika 26. Podaci o korisniku i mjenjanje lozinke

5.2. BACKEND

Slika 27 prikazuje izgled backenda u Visual Studio Codu, a podrobnija će analiza arhitekture backenda biti opisana u narednim podpoglavljima.



Slika 27. Arhitektura backenda

5.2.1. NODE MODULES

Mapa koja se nalazi i u frontendu nalazi se i u backendu, a sadrži *Node.js* pakete koji su potrebni za izvršavanje koda sa server strane. Navedeni slučaj koristi *Express.js framework* koji omogućuje definiranje ruta koje aplikacija koristi na različite klijentske zahtjeve na određenim HTTPS metodama ili URL-ovima. Također, od ostalih paketa u mrežnoj aplikaciji koristi se *cors paket te.end* datoteke, a budući da se radi o *backend* aplikacijama održavanje sigurnosti paketa je od najveće važnosti te je bitno imati pakete koji su *up to date*.

5.2.2. ROUTE

Unutar *Route* mape nalaze se dvije datoteke, a to su *routes.js* i *uploadroutes.js*. Datoteka *route.js* sadrži funkciju *isAutenticated* koja provjerava je li korisnik autentificiran prije nego dozvoli pristup određenim rutama. Ukoliko je korisnik spremljen u sesiji *req.session.korisnik* utoliko funkcija poziva *next()* da nastavi s izvršavanjem sljedećeg *handlera* u *stogu*. Međutim, ako korisnik nije autentificiran šalje se odgovor *Unauthorized*. Isto tako, datoteka sadrži uključivanje kontrolera koji sadrže logiku operacija koje se izvode u mrežnoj aplikaciji kao i definiranje ruta na način da se kreira novi *Router* objekt koji omogućuje definiranje ruta unutar aplikacije. Svaka se ruta povezuje određenim kontrolerom – *slatkaZelja/login* i *slatkaZelja/create* rute su za korisnike funkcije kao što su *prijava* i *registracija*. *Custom-cake/create* je ruta za kreiranje personalizirane torte dok se funkcija *isAutenticated* koristi za rute *create-Tray* i *check.date.availability* kako bi se omogućilo isključivo prijavljenim korisnicima izvršavanje ove akcije, a kod je opisan na slici 28.

```
var express = require('express');

function isAuthenticated(req, res, next) {
    if (req.session && req.session.korisnik) {
        next();
    } else {
        res.status(401).send('Morate biti prijavljeni da biste izvršili ovu akciju.');
    }
}
var userController = require('../src/Controller/userController.js');
const customCakeController = require('../src/Controller/customCakeController.js');
const cake2Controller = require('../src/Controller/customCake2Controller.js');

const router = express.Router();

router.route('/slatkaZelja/login').post(userController.loginUserControllerFn);
router.route('/slatkaZelja/create').post(userController.createUserControllerFn);

router.post('/custom-cake/create', customCakeController.createCustomCake);
router.post('/create-Tray',isAuthenticated, cake2Controller.createCustomCake2);
router.get('/check-date-availability',isAuthenticated, customCakeController.checkDateAvailability);

module.exports = router;
```

Slika 28. Route

Datoteka *uploadRoute.js* predstavlja *Express.js router* koji omogućuje prijenos slika na server koristeći *multer* koji se koristi za obradu datoteka. Dakle, kao što je vidljivo iz prikaza na slici 29, za pohranu slika koristi se *diskStorage* što znači da se datoteke fizički spremaju na server. *Destination* je funkcija koja određuje putanju gdje će se datoteke pohraniti što znači da se u ovoj aplikaciji sve slike pohranjuju u direktorij *./uploads. Filename* je funkcija koja

određuje ime datoteke pod kojim će se ta slika pohraniti, ime se sastoji od imena polja formulara, trenutnog timestampa[1] i ekstenzije originalne datoteke.

```
const storage = multer.diskStorage({
    destination: function(req, file, cb) {
        cb(null, './uploads')
    },
    filename: function(req, file, cb) {
        cb(null, file.fieldname + '-' + Date.now() + path.extname(file.originalname))
    }
});

const upload = multer({ storage: storage });

router.post('/', upload.single('image'), async (req, res) => {
    try {
      if (!req.file) {
        return res.status(400).send('No file uploaded.');
      }

      const { comment, pickupDate, userId } = req.body;

      if (!comment) {
        return res.status(400).send('Comment is required.');
      }

      if (!pickupDate) {
        return res.status(400).send('Pickup date is required.');
      }

      const pickupDateObj = new Date(pickupDate);
      if (isNaN(pickupDateObj.getTime())) {
        return res.status(400).send('Invalid pickup date.');
      }

      const user = await User.findById(userId);

      if (!user) {
        return res.status(404).send('User not found.');
      }

      const imageUrl = `${req.protocol}://${req.get('host')}/uploads/${req.file.filename}`;

      const newImage = new Image({
        imagePath: imageUrl,
        userId: userId,
        comments: [{ commenterName: user.korisnickoime, commentText: comment }],
        pickupDate: pickupDateObj
      });

      await newImage.save();
      res.status(201).json({ message: 'Slika je uspješno poslana!', imageUrl });
    } catch (err) {
      console.error(err);
      res.status(500).send('Server Error');
    }
  });

module.exports = router;
```

Slika 29. uploadRoute

5.2.3. SRC

5.2.3.1. CONTROLLER

Unutar mape *Controller* nalaze se tri datoteke: *customCakeController.js*, *customCake2Controller.js* i *userController.js*. Datoteka *customCakeController.js* sadrži dvije funkcije izrađene pomoću *Express.js-a* koje su dio API-ja za upravljanje personaliziranim

[1] Timestamp je oznaka vremena koja precizno prikazuje kada se neki događaj dogodio. Izvor: (https://en.wikipedia.org/wiki/Timestamp)

tortama. Funkcije su definirane unutar *Node.js* modula koji koristi *Mongoose* model za interakciju s bazom. Funkcija *createCustomCake* prvotno stvara novu instancu *CustomCake* modela pomoću podatka iz tijela zahtjeva uključujući *userID*. Ukoliko su podaci ispravni utoliko se instanca sprema u bazu podatka koristeći metodu *save()*. Funkcija *checkDateAvailability* prima datum te ga pretvara u JavaScript *date* objekt i postavlja vremenske granice za taj dan, točnije 12 po jedom danu. Nakon toga pretražuje bazu podatka za sve *CustomCake* instance koje imaju *pickupDate*, ali isključivo za onaj koji je unutar tih granica. Ukoliko ne postoji nijedna torta zakazana za taj dan utoliko funkcija vraća *datum je dostupan* inače nije dostupan. Sav kod s funkcijama opisan je na slici 30.

```
const CustomCake = require('../Model/customCakeModel');
const User = require('../Model/userModel');

exports.createCustomCake = async (req, res) => {
    try {
        const customCake = new CustomCake({
            ...req.body,
            userId: req.body.userId
        });

        await customCake.save();
        res.status(201).json({ message: 'Torta je uspješno personalizirana!' });
    } catch (error) {
        console.error('Error creating custom cake:', error);
        res.status(500).json({ message: 'Došlo je do greške prilikom spremanja torte.', error: error.toString() });
    }
};

exports.checkDateAvailability = async (req, res) => {
    try {
        const chosenDate = new Date(req.query.date);
        const startOfDay = new Date(chosenDate.setHours(0, 0, 0, 0));
        const endOfDay = new Date(chosenDate.setHours(23, 59, 59, 999));

        const cakes = await CustomCake.find({
            pickupDate: {
                $gte: startOfDay,
                $lt: endOfDay
            }
        });

        res.json({ isAvailable: cakes.length === 0 });
    } catch (error) {
        console.error('Error checking date availability:', error);
        res.status(500).json({ error: error.toString() });
    }
};
```

Slika 30. customCakeController.js

Unutar datoteke *customCake2Controlle.js* nalazi se funkcija za *createCustomCake2* koja je dio API-a koji je zaslužan za kreiranje novih pladnjeva u aplikaciji koristeći *Node.js* i *Mongoose* za interakciju s bazom. Na početku se uvoze *Tray* i *User* modeli. *Tray* koji predstavlja strukturu podatka za pladanj te *User* model koji predstavlja strukturu podatka za korisnika. Sama funkcija radi na principu korištenja konstruktora modela *Tray* za stvaranje nove instance pladnja. ..*Req.body* prenosi sve podatke iz tijela zahtjeva direktno u novi objekt *Tray* uključujući *userID* i pickupDate. *userID* se dodaje iz tijela zahtjeva što označava korisnika koji stvara pladanj, a *pickupDate* se pretvara iz string u JavaScript Date objekt što osigurava da se datum može pravilno spremiti i obraditi u bazi podatka. Nakon kreiranja instance koristi se *save()* metoda za spremanje pladnja u bazu, a kod opisan na slici 31.

```
const Tray = require('../Model/TrayModel');
const User= require('../Model/userModel');


exports.createCustomCake2 = async (req, res) => {
    try {
        const newTray = new Tray({
            ...req.body,
            userId: req.body.userId,
            pickupDate: new Date(pickupDate)
        });

        console.log("New Tray being saved:", newTray);
        await newTray.save();
        res.status(201).json({ message: 'Pladanj je uspješno spremljen!' });
    } catch (error) {
        console.error('Error saving tray:', error);
        res.status(500).json({ message: 'Došlo je do greške prilikom spremanja pladnja.', error: error.toString() });
    }
};
```

Slika 31 customCake2Controller.js

Unutar datoteke *UserController.js* nalaze se dvije funkcije za upravljanje korisnicima. *createUserControllerFn* za registraciju novih korisnika i *loginUserControllerFn* za prijavu korisnika. Funkcija *createUserControllerFn* na početku logira primljeni zahtjev i tip lozinkE. a potom koristi *userService* kojim se provjerava postoji li već korisnik s upisanim korisničkim imenom s obzirom na to da ne mogu dva korisnika posjedovati isto korisničko ime. Ukoliko je korisničko ime slobodno utoliko funkcija poziva *createUserDBService* iz datoteke *userService* te na kraju daje *JSON* odgovor s rezultatom.

Funkcijom *loginUserControllerFn* ističu se korisničko ime i lozinka iz tijela zahtjeva, a pomoću *userModela* pronalazi se korisnik s odgovarajućim korisničkim imenom. Postoji li korisnik i lozinke se podudaraju – korisnik je autentificiran, a ako je korisnik autentificiran informacije o njemu pohranjuju se u sesiju. Podaci u sesiji bitni su za ostale elemente koje koristi mrežna aplikacija što navedenu funkciju čini izrazito bitnom. Ovisno o ulozi korisnika šalje se odgovor s URL-om na koji korisnik treba biti preusmjeren, a ako podatak nije ispravan korisnik prima poruku greške čiji kod je opisan na slici 32.

```
var userService = require('../userService.js');
var userModel = require('../Model/userModel.js');

var createUserControllerFn = async (req, res) => {
  try {
    console.log("Zahtjev za registracijom primljen:", req.body);
    console.log('Tip lozinke:', typeof req.body.lozinka);

    const existingUser = await userService.checkUsernameExists(req.body.korisnickoime);
    if (existingUser) {
      return res.json({ status: false, message: 'Korisničko ime već postoji. Molimo odaberite drugo.' });
    }

    var result = await userService.createUserDBService(req.body);
    console.log(result);

    res.json(result);
  } catch (err) {
    console.log(err);
    res.json({ status: false, message: 'Greška prilikom registracije. Molimo pokušajte ponovno.' });
  }
}

var loginUserControllerFn = async (req, res) => {
  try {
      const { korisnickoime, lozinka } = req.body;
      const user = await userModel.findOne({ korisnickoime: korisnickoime });

if (user && user.lozinka === lozinka) {
  req.session.korisnik = {
      _id: user._id,
      korisnickoime: user.korisnickoime,
      uloga: user.uloga
  };

  let redirectUrl = user.uloga === 'admin' ? '/gallery.html' : '/';
  return res.json({ status: true, redirect: redirectUrl });
} else {
  return res.status(401).json({ status: false, message: 'Neuspješna prijava. Provjerite korisničko ime i lozinku.' });
}

  } catch (err) {
      console.log(err);
      return res.status(500).json({ status: false, message: 'Greška prilikom prijave. Molimo pokušajte ponovno.' });
  }
}

module.exports = { createUserControllerFn, loginUserControllerFn };
```

Slika 32. userController.js

5.2.3.2. MODEL

Unutar mape *Model* nalaze se modeli koji se koriste u mrežnoj aplikaciji: *cakeModel.js*, *customCakeModel.js*, *imageModel.js*, *TrayModel.js*, *userModel.js*. Unutar datoteke *cakeModel.js* nalazi se *trayShema* koja je definirana u *Mongoose* biblioteci koja organizira strukturu podatka za pladnjeve kolača. Svaki pladanj povezan je s korisnikom putem *userID*-a, a *trayCount* označava broj pladnjeva u narudžbi. Unutar svakog pladnja moguće je imati više vrsta kolača gdje svaki sadrži neke informacije poput: tip kolača, broj komada, cijena po komadu i ukupne kalorije. Shema prati i ukupnu cijenu i kalorije svih kolača na pladnju, a datum preuzimanja – *pickupDate* – obavezan je podatak vezan za svaku narudžbu. Status narudžbe može biti: *na čekanju*, *prihvaćeno* i *odbijeno*. Svaki dokument uključuje informacije o vremenu kreiranja i posljednjoj izmjeni pomoću funkcije *timestamps*, a cijeli kod je opisan na slici 33.

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const traySchema = new mongoose.Schema({
    userId: { type: Schema.Types.ObjectId, ref: 'user' },
    trayCount: Number,
    cakes: [{
        cakeType: String,
        count: Number,
        pricePerPiece: Number,
        totalCalories: Number
    }],
    totalPrice: Number,
    totalCalories: Number,
    pickupDate: { type: Date, required: true },
    status: { type: String, enum: ['na čekanju', 'prihvaćeno', 'odbijeno'], default: 'na čekanju' },
    isDeleted: { type: Boolean, default: false }
},{ timestamps: true });

module.exports = mongoose.model('Tray', traySchema);
```

Slika 33. cakeModel.js

Unutar datoteke *customCakeModel.js* nalazi se shema za perosnaliziranje torte koja definira strukturu i tipove podatka koje će svaka instanca *CustomCake* sadržavati. Podaci poput oblika, veličine, boje, biskvita, kreme, broj katova, ukupna cijena, datum preuzimanja, rođendan i status narudžbe obavezna su polja koja imaju različite opcije ovisno o tome što korisnik odabere. Primjerice, broj svjećica nije obavezno polje jer torta može, ali i ne mora biti namijenjena za tu prigodu. Funkcija *timestamps: true* u shemi omogućava *Mongooseu* da automatski dodaje i ažurira polja *createdAt* i *updateAt* za svaki dokument koji se kreira ili mijenja, a kod je opisan na slici 34.

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const customCakeSchema = new mongoose.Schema({
    shape: { type: String, required: true },
    size: { type: String, required: true },
    color: { type: String, required: true },
    biscuit: { type: String, required: true },
    cream: { type: String, required: true },
    kat: { type: String, required: true },
    birthday: { type: Boolean, required: true },
    age: { type: Number },
    totalPrice: { type: Number, required: true },
    pickupDateTime: { type: Date, required: true },
    userId: { type: Schema.Types.ObjectId, ref: 'user' },
    status: { type: String, enum: ['na čekanju', 'prihvaćeno', 'odbijeno'], default: 'na čekanju' },
    isDeleted: { type: Boolean, default: false }
}, { timestamps: true });

module.exports = mongoose.model('CustomCake', customCakeSchema);
```

Slika 34. customCakeModel

Datoteka *imageModel.js* sadržava shemu za spremanje slika i komentara u bazu. Shema sadržava *imagePath* polje koje sadržava putanju do slike što je obavezno polje koje nužno označeno *string*.*Comments* jer je niz objekata koji sadrži informacije o komentaru koje je korisnik napisao. *userID* se koristi kako bi identificirali korisnika koji je poslao sliku. *Ref: user* povezuje polje s *User* modelom omogućavajući da se dohvate korisnički podaci koji su povezani sa slikom. *PickupDate* datum je kada bi torta sa slike trebala biti preuzeta, polje je obvezno i *Date* je tipa. Status označava status narudžbe, a polje je obavezno. Shema sadrži i polje *isDeleted* koje označava je li slika izbrisana, a s obzirom na to da je postavljeno na *false* to znači da slika trenutno nije izbrisana što se vidi iz prikaza na slici 35.

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const imageSchema = new mongoose.Schema({
    imagePath: {
        type: String,
        required: true
    },
    comments: [{
        commenterName: String,
        commentText: String
    }],
    userId: { type: Schema.Types.ObjectId, ref: 'user' } ,
    pickupDate: { type: Date, required: true },
    status: { type: String, enum: ['na čekanju', 'prihvaćeno', 'odbijeno'], default: 'na čekanju' },
    isDeleted: { type: Boolean, default: false }
}, { timestamps: true });

module.exports = mongoose.model('Image', imageSchema);
```

Slika 35. imageModel.js

Datoteka *TrayModel.js* sadržava shemu o podacima koji se nalaze kod sastavljanja pladnja s kolačima. *Traycount* polje predstavlja broj pladnjeva i obavezno je polje te se sprema kao numerička vrijednost. Polje *Cakes* niz je objekata koji detaljno opisuje svaku vrstu kolača na pladnju i to na način da se prvo nalazi tekstualni opis tipa kolača, broj komada, cijena po komadu, ukupan broj kalorija, ukupna cijena te *userID*, a sami kod je opisan na slici 36.

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const traySchema = new mongoose.Schema({
    trayCount: Number,
    cakes: [{
        cakeType: String,
        count: Number,
        pricePerPiece: Number,
        totalCalories: Number
    }],
    totalPrice: Number,
    totalCalories: Number,
    userId: { type: Schema.Types.ObjectId, ref: 'user' }
});

const Tray = mongoose.models.Tray || mongoose.model('Tray', traySchema);

module.exports = Tray;
```

Slika 36. trayModel.js

Slika 37, na kojoj je vidljiv kod, prikazuje datoteku *userModel*.js u kojoj se nalazi shema koja sadrži polja s podacima o korisniku. Model sadrži polja: ime, prezime, korisničko ime, e-mail, lozinka i uloga, a sva su polja obvezna.

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var userSchema = new Schema({

    ime: {
        type: String,
        required: true
    },
    prezime: {
        type: String,
        required: true
    },
    korisnickoime: {
        type: String,
        required: true,
        unique: true
    },
    email: {
        type: String,
        required: true
    },
    lozinka: {
        type: String,
        required: true
    },
    uloga: {
        type: String,
        default: 'User'
    }
});

module.exports = mongoose.model('user', userSchema);
```

Slika 37. userModel.js

5.2.4. USERSERVICE.JS

Datoteka *userService.js* sadrži tri funkcije *checkUsernameExist*, *createUserDBService* i *loginUserDBService*. *checkUsernameExist* funkcija provjerava postojanje korisničkog imena u bazi podataka na način da prvo prima korisničko ime kao argument i koristi metodu *findOne* na modelu *userModel* za traženje korisnika pod tim korisničkim imenom. Kao rezultat vraća se *true* u slučaju da korisnik postoji, a u slučaju greške vraća se *false*. Funkcija *createUserDBService* kreira novog korisnika u bazi podatka samo u slučaju ako korisničko ime nije zauzeto. Postojanje imena pomoću funkcije *checkUsernameExist* provjerava se na početku, a ako ime postoji povratno se upućuje poruka o nedostupnosti korisničkog imena. Ukoliko je korisničko ime slobodno utoliko se za korištenje kreira novi zapis u bazi koristeći model *userModel* s detaljima koje funkcija kao *userDetails*. Podaci se u bazu spremaju pomoću metode *save(),* a opisani kod za obje funkcije prikazan je na slici 38.

```
module.exports.checkUsernameExists = async (korisnickoime) => {
  try {
    const existingUser = await userModel.findOne({ korisnickoime: korisnickoime });
    return existingUser !== null;
  } catch (error) {
    console.error('Error checking username:', error);
    return false;
  }
};

module.exports.createUserDBService = async (userDetails) => {
  console.log('Received request:', userDetails);
  console.log('Form data:', userDetails);

  const usernameExists = await module.exports.checkUsernameExists(userDetails.korisnickoime);

  if (usernameExists) {
    return { status: false, message: 'Korisničko ime već postoji. Molimo odaberite drugo korisničko ime.' };
  }

  try {
    console.log('Inside createUserDBService');

    var userModelData = new userModel();

    userModelData.ime = userDetails.ime;
    userModelData.prezime = userDetails.prezime;
    userModelData.korisnickoime = userDetails.korisnickoime;
    userModelData.email = userDetails.email;
    userModelData.lozinka = userDetails.lozinka;

    console.log('Usermodeldata: ' + userModelData.ime);
    const result = await userModelData.save();
    console.log('User saved successfully:', result);
    return { status: true, message: 'Uspješno ste se registrirali!' };
  } catch (error) {
    console.error('Error saving to database:', error);
    return { status: false, message: 'Registracija nije uspjela. Molimo pokušajte ponovno.' };
  }
};
```

Slika 38. checkUserNameExist i createUserDBService

Funkcija *loginUserDBService* koristi se kako bi se identificirao korisnik na temelju korisničkog imena i lozinke, gdje se na početku traži korisnik s odgovarajućim korisničkim imenom koristeći *findOne*. Ukoliko se pronađe korisnik te su lozinke istovjetne utoliko se korisnički podaci spremaju u sesiju pomoću *express-session*, a sama se sesija pohranjuje koristeći *session.save*. Nakon toga, vraća se status o uspješnoj ili neuspješnoj prijavi, tj. vidimo je li korisnik prijavljen ili nije, a sami kod funkcije prikazan je na slici 39.

```
module.exports.loginUserDBService = async (loginDetails, req) => {
  try {
    console.log('Received login request:', loginDetails);

    const korisnickoime = loginDetails.korisnickoime;
    const lozinka = loginDetails.lozinka;

    const user = await userModel.findOne({ korisnickoime: korisnickoime });
    console.log('Tražim usera u bazi za korisnickoime:', korisnickoime);
    console.log('Unesena lozinka:', lozinka);

    if (user && user.lozinka === lozinka) {
      console.log("user??" + user)
      req.session.korisnik = user;
      console.log('req sessuibn jorn' + req.session.korisnik)
      req.session.save((err) => {
        if (err) {
          console.error('Session save error:', err);
          throw err;
        }
        req.session.korisnik = user;
        console.log('Uspješna prijava!', req.session.korisnik);
      });
      return { status: true, message: 'Uspješna prijava!' };
    } else {
      console.log('Neuspješna prijava. Provjerite korisničko ime i lozinku.');
      return { status: false, message: 'Neuspješna prijava. Provjerite korisničko ime i lozinku.' };
    }
  } catch (error) {
    console.error('Error during login:', error);
    return { status: false, message: 'Greška prilikom prijave. Molimo pokušajte ponovno.' };
  }
};
```

Slika 39. loginUserDBService

5.3. CREATEADMIN.JS

Datoteka *createAdmin.js* sadrži skriptu za stvaranje administratorskog računa jer se administrator ne može registrirati kao i ostali korisnici već je potrebno prilikom izrade mrežne aplikacije izraditi račun administratoru ostavljajući mogućnost kreiranja dodatnih administratorskih računa ako za to bude potrebe. Unutar se skripte koristi *mongoose* koji omogućuje definiranje shema te *userModel* koji predstavlja shemu za korisničke podatke. *Mongoose.connect* funkcija uspostavlja vezu s bazom podataka na lokalnom serveru. Opcije *useNewUrlParser* i *useUnifiedTopology* postavljenje su na *true* kako bi se omogućilo korištenje najnovijeg URL parsera i upravljača topologije što je preporučljivo za stabilnije i sigurnije povezivanje s bazom. Funkcija *createAdmin* kreira novu instancu *userModel*, ali samo za administratora što uključuje dodavanje osnovnih podatka kao što su ime, prezime, e-mail, lozinka te najbitniji uloga. Metoda *adminUser.save*, prikazana na slici 40, sprema novostvoreni dokument u bazu, a *try* i *catch* blokovi se koriste za rukovanje mogućim greškama dok funkcija *mongoose.disconnect* prekida vezu s bazom podataka.

```
const mongoose = require('mongoose');
const userModel = require('./src/slatkaZelja/userModel');

mongoose.connect('mongodb://localhost:27017/abc', {
    useNewUrlParser: true,
    useUnifiedTopology: true
});

const createAdmin = async () => {
    const adminUser = new userModel({
        ime: 'Admin',
        prezime: 'User',
        korisnickoime: 'admin',
        email: 'admin@example.com',
        lozinka: [redacted]
        uloga: 'admin'
    });

    try {
        await adminUser.save();
        console.log('Admin account has been successfully created!');
    } catch (error) {
        console.error('Error creating admin account:', error);
    }

    mongoose.disconnect();
};

createAdmin();
```

Slika 40. Kreiranje admina

5.4. SERVER.JS

Datoteka *server.js* pokreće mrežnu aplikaciju koristeći *Express framework* i *MongoDB* za pohranu podataka, a na početku datoteke potrebno je uvesti module kako bi se moglo upravljati serverom i rutama zahtjeva. Uvozi se *Express-modul* za upravljanje serverom, *Express-session* za upravljanje sesijama korisnika, *Cors* za sigurno omogućavanje zahtjeva između različitih domena, *path modul* koji pomaže u manipulaciji putanja datoteka, *mongoose* je ODM (Object Dana Modeling) biblioteka za *MongoDB* za lakše upravljanje vezama i modelima baze podataka, *connect-mongo* koji se koristi za povezivanje sesija s *MongoDb* bazom. Definiramo modele podatka *image*, *customCake*, *Cake*, *Tray* te *userService* koji sadrži logiku za upravljanje korisničkim podacima. Prikazano na slici 41.

```
const express = require('express');
const session = require('express-session');
const cors = require('cors');
const path = require('path');
const mongoose = require('mongoose');
const MongoStore = require('connect-mongo');
const app = express();

const Image = require('./src/Model/imageModel');
const CustomCake = require('./src/Model/customCakeModel');
const Cake = require('./src/Model/cakeModel');
const User = require('./src/Model/userModel');
const Tray = require('./src/Model/TrayModel');
const userService = require('./src/userService');
```

Slika 41. Uvedeni moduli

Server.js sadrži funkciju koja se aktivira prilikom slanja korisnika POST zahtjeva na URL */slatkaZelja/login* gdje je na početku funkcije ekstrakcija podatka koji se nalaze u *req.body*. Zatim se poziva funkcija *userService.loginUserDBService(loginDetails, req)* koja je zadužena za provjeru autentičnosti korisnika te ako je *user.status* istinit korisnički se objekt pohranjuje u sesiji *req.session.korisnik* što omogućava korisniku prijavu na bilo kojoj stranici neovisno o lokaciji. Potom se sesija eksplicitno sprema pomoću *req.session.save()*, a kod je prikazan na slici 42.

```
app.post('/slatkaZelja/login', async (req, res) => {
    const loginDetails = req.body;
    try {
        const user = await userService.loginUserDBService(loginDetails, req);
        if (user.status) {
            req.session.korisnik = user.user;
            req.session.save((err) => {
                if (err) {
                    console.error('Session save error:', err);
                    res.status(500).json({ success: false, message: 'Server error while logging in.' });
                } else {
                    res.status(200).json({ success: true, message: 'Uspješna prijava!' });
                }
            });
        } else {
            res.status(401).json({ success: false, message: 'Netočno korisničko ime ili lozinka' });
        }
    } catch (error) {
        console.error('Error logging in:', error);
        res.status(500).json({ success: false, message: 'Server error while logging in.' });
    }
});
```

Slika 42. API login

Funkcija koja se nalazi u *server.js* aktivira se prilikom korisnikova slanja GET zahtjev na URL */session*-status, a glavna je svrha provjeriti status sesije te informirati korisnika o prijavi. Provjera sesije odrađuje se na način da se utvrdi postoji li *req.session.korisnik*, a

potvrdom postojanja daje se znak da je korisnik prijavljen. Isto tako, sesija sadrži *objekt korisnik* koji je prethodno pohranjen tijekom procesa prijave pri čemu se funkcija vraća *JSON* odgovorom u statusu *LoggedLn: true* i korisničkim imenom. Nepostojanje *req.session.korisnika* govori kako korisnik nije prijavljen, a kod je prikazan na slici 43.

```
app.get('/session-status', (req, res) => {
    console.log('req session', req.session);
    if (req.session.korisnik) {
        res.json({ loggedIn: true, korisnickoime: req.session.korisnik.korisnickoime });
    } else {
        res.json({ loggedIn: false });
    }
});
```

Slika 43. API session/status

Datoteka *server.js* prikazuje kod (vidi slika 39) koji se odnosi na *API endpoint /gallery* (podaci o svim narudžbama). Nakon što se „posjeti“ ovaj *endpoint* server dohvaća podatke iz tri različite lokacije u bazi, a to su *Image*, *CustomCake* i *Tray*. Funkcije *Image.find, CustomCake.find i Tray.find* filtriraju podatke na način da ne uključuju one koje su označene kao izbrisane *isDeleted: false*. Funkcija *.populate(userID)* koristi se kod podataka o tortama te omogućuje pridruživanje detalja o korisniku koji je kreirao navedene objekte. Podaci se mapiraju u niz *galleryData* koji uključuje podatke kao što je tip, putanja i ime korisnika koji je izvršio naručivanje, datum preuzimanja i status svake narudžbe. Prikaz koda nalazi se na slici 44.

```
app.get('/gallery', async (req, res) => {
    try {
        const images = await Image.find({ isDeleted: false });
        const customCakes = await CustomCake.find({ isDeleted: false }).populate('userId');
        const trays = await Tray.find({ isDeleted: false }).populate('userId');
        const galleryData = [
            ...images.map(img => ({
                _id: img._id,
                tip: 'Slike',
                imagePath: img.imagePath,
                orderedByUsername: img.comments[0]?.commenterName,
                comments: img.comments.map(c => `${c.commenterName}: ${c.commentText}`).join(', '),
                datumPreuzimanja: img.pickupDate.toLocaleDateString("hr-HR"),
                status: img.status,
            })),
            ...customCakes.map(cake => ({
                _id: cake._id,
                tip: 'Torte',
                detalji: `Oblik: ${cake.shape}, Veličina: ${cake.size}, Boja: ${cake.color}, Biskvit: ${cake.biscuit}, Krema: ${cake.cream}, Katovi: ${cake.kat}, Rođendan: ${cake.bir
                orderedByUsername: cake.userId.korisnickoime,
                datumPreuzimanja: cake.pickupDateTime.toLocaleDateString("hr-HR"),
                status: cake.status,
            })),
            ...trays.map(tray => ({
                _id: tray._id,
                tip: 'Kolači',
                brojTacni: tray.trayCount,
                kolači: tray.cakes.map(cake => `Vrsta kolača: ${cake.cakeType}, Broj: ${cake.count}, Cijena po komadu: €${cake.pricePerPiece.toFixed(2)}`),
                ukupnaCijena: `€${tray.totalPrice.toFixed(2)}`,
                orderedByUsername: tray.userId.korisnickoime,
                datumPreuzimanja: tray.pickupDate.toLocaleDateString("hr-HR"),
                status: tray.status,
            }))
        ];

        console.log(galleryData);
        res.json(galleryData);
    } catch (err) {
        console.error(err);
        res.status(500).json({ error: 'Server Error' });
    }
});
```

Slika 44. API Gallery

Kod za spremanje novog pladnja s kolačima u bazu podataka predstavlja novi API *endpoint* koji prima POST zahtjev i koristi se za spremanje novog pladnja u bazu podataka. Nakon što se *endpoint* aktivira podaci potrebni za kreiranje novog pladnja uzimanju se iz tijela *req.body*, a uključuju broj pladnjeva *trayCount*, popis kolača *cakes*, ukupnu cijenu *totalPrice*, ukupan broj kalorija *totalCalories*, datum preuzimanja *pickupDate* i identifikaciju korisnika *userID*. Nakon uzimanja podataka koristi se konstruktor *Tray* za kreiranje novog objekta pladnja sa specificiranim podacima dok se datum posebno obrađuje kako bi mogao biti u odgovarajućem formatu koristeći *newDate*. Nakon toga se podaci spremaju u bazu pozivanjem metode *save()* na instanci *newTray*, a kod je prikazan na slici 45.

```
app.post('/save-tray', async (req, res) => {
    const { trayCount, cakes, totalPrice, totalCalories, pickupDate, userId } = req.body;
    try {
        const newTray = new Tray({
            userId,
            trayCount,
            cakes,
            totalPrice,
            totalCalories,
            pickupDate: new Date(pickupDate)
        });
        await newTray.save();
        console.log(newTray);
        res.status(201).json({ message: 'Tray saved successfully!' });
    } catch (error) {
        console.error('Error saving tray:', error);
        res.status(500).json({ error: 'Failed to save tray', message: error.toString() });
    }
});
```

Slika 45. API save-tray

Označavanje narudžbi kao da su izbrisane koristi se *endpoint /mark-order-as-deleted/:id.*, a spomenuti *endpoint* omogućava administratoru prikrivanje narudžbe kako bi smanjio broj ukoliko ne želi imati previše završenih narudžbi u tablici sve narudžbe. Narudžbe se označavaju kao izbrisane putem POST zahtjeva, dakle, koristi se dinamički parametar u URL-u *:id* za identifikaciju specifične narudžbe. Koristeći funkciju *Image.findByIdAndUpdate*, na samom početku, pokušava se ažurirati objekt *Image* u bazi podatka gdje se *id* izvlači iz zahtjeva *req.params.id*, a polje *isDeleted* postavlja se na *true*. Međutim, ako se prvi pokušaj pokaže neuspješnim proces se ponavlja za *CustomCake*, a ako je i tada neuspješan ponavlja se i za *Tray*. Nakon što se objekt pronađe i ažurira dolazi poruka o uspješnosti akcije, a kod je opisan na slici 46.

```
app.post('/mark-order-as-deleted/:id', async (req, res) => {
    try {
        let result = await Image.findByIdAndUpdate(req.params.id, { isDeleted: true });
        if (!result) {
            result = await CustomCake.findByIdAndUpdate(req.params.id, { isDeleted: true });
        }
        if (!result) {
            result = await Tray.findByIdAndUpdate(req.params.id, { isDeleted: true });
        }

        if (result) {
            res.json({ message: 'Narudžba je obrisana!' });
        } else {
            res.status(404).json({ message: 'Narudžba nije pronađena!' });
        }
    } catch (error) {
        console.error(error);
        res.status(500).json({ message: 'Došlo je do greške pri označavanju narudžbe kao obrisane.' });
    }
});
```

Slika 46. API mark-order-as-deleted

Ažuriranje narudžbe koristi funkciju API *endpoint /update-order-status*. Pozivanje *endpointa* odvija se putem POST metode gdje se na početku kod prima *orderId* i status iz tijela *req.body.* Prikupljene se informacije koriste za identifikaciju narudžbi kojoj se želi promijeniti status narudžbe. Nakon toga proces ažuriranja započinje ažuriranjem dokumenata u kolekciji *CustomCake* pomoću metode *findByIdAndUpdate* koja traži narudžbu po *orderID*-u i postavlja status narudžbe na novu vrijednost uz opciju *{ new: true }* koja osigurava da metoda vrati dokument koji je ažuriran. Ukoliko ažuriranje u *CustomCake* nije uspješno utoliko se proces nastavlja s kolekcijom *Tray*, a ako i tada nije uspješan provjerava se kolekcija *Image*. Nakon što je ažuriranje uspješno obavljeno dobiva se poruka o uspješnosti akcije, a kod je opisan na slici 47.

```
app.post('/update-order-status', async (req, res) => {
    const { orderId, status } = req.body;

    console.log('Order status: ',req.body)

    try {
        let updatedOrder = await CustomCake.findByIdAndUpdate(orderId, { status }, { new: true });
        if (!updatedOrder) {
            updatedOrder = await Tray.findByIdAndUpdate(orderId, { status }, { new: true });
        }
        if (!updatedOrder) {
            updatedOrder = await Image.findByIdAndUpdate(orderId, { status }, { new: true });
        }

        if (updatedOrder) {
            res.json({ message: 'Status narudžbe ažuriran.' });
        } else {
            res.status(404).send('Narudžba nije pronađena.');
        }
    } catch (error) {
        console.error('Error updating order status:', error);
        res.status(500).send('Greška pri ažuriranju statusa narudžbe.');
    }
});
```

Slika 47. API update-order-status

Detalje o korisniku dobijaju se putem endpointa */get-user-details/:userName* koji koristi GET metodu. Korisničko se ime preuzima iz URL-a putem parametra *userName* što omogućava direktno dohvaćanje korisničkog imena iz zahtjeva koristeći *req.params.userName. Try* blok koristi metodu *User.findOne* koja pretražuje kolekciju korisnika u bazi i dohvaća jednog korisnika gdje je *korisnickoime* istovjetno s korisničkim imenom, a kod je opisan na slici 48.

```
app.get('/get-user-details/:userName', async (req, res) => {
    const userName = req.params.userName;

    try {
        const user = await User.findOne({ korisnickoime: userName });
        console.log(user)
        if (!user) {
            return res.status(404).send('Korisnik nije pronađen.');
        }
        res.json({ user });
    } catch (error) {
        console.error('Server Error:', error);
        res.status(500).send('Došlo je do server greške.');
    }
});
```

Slika 48. API get-user-details

Prilikom promjene lozinke korisnika koristi se API *endpoint /update-user.password* na uz primjenu POST metode gdje se na početku izdvaja *oldPassword* što je trenutna lozinka korisnika i *newPassword* što označava novu lozinku iz *req.body-a*. *req.body* koja mora

sadržavati *userID* korisnika čija se lozinka želi ažurirati. Nakon toga se koristi funkcija *User.FindById* kako bi se pronašao točan korisnik, a ako je pronađen provjerava se je li *user.lozinka* ista kao i *oldPassword.* Međutim, ako lozinke nisu iste nova se lozinka ažurira na *newPassword* i promjene se spremaju u bazu pomoću *user.save()* metode. Navedeni je kod prikazan na slici 49.

```
app.post('/update-user-password', async (req, res) => {
    const { oldPassword, newPassword } = req.body;
    try {
        const user = await User.findById(req.body.userId);
        if (!user) {
            return res.status(404).send('Korisnik nije pronađen.');
        }
        if (user.lozinka === oldPassword) {
            user.lozinka = newPassword;
            await user.save();
            res.send({ success: true, message: 'Lozinka je ažurirana.' });
        } else {
            res.status(400).send({ success: false, message: 'Stara lozinka nije ispravna.' });
        }
    } catch (error) {
        console.error('Server Error:', error);
        res.status(500).send({ message: 'Došlo je do server greške.' });
    }
});
```

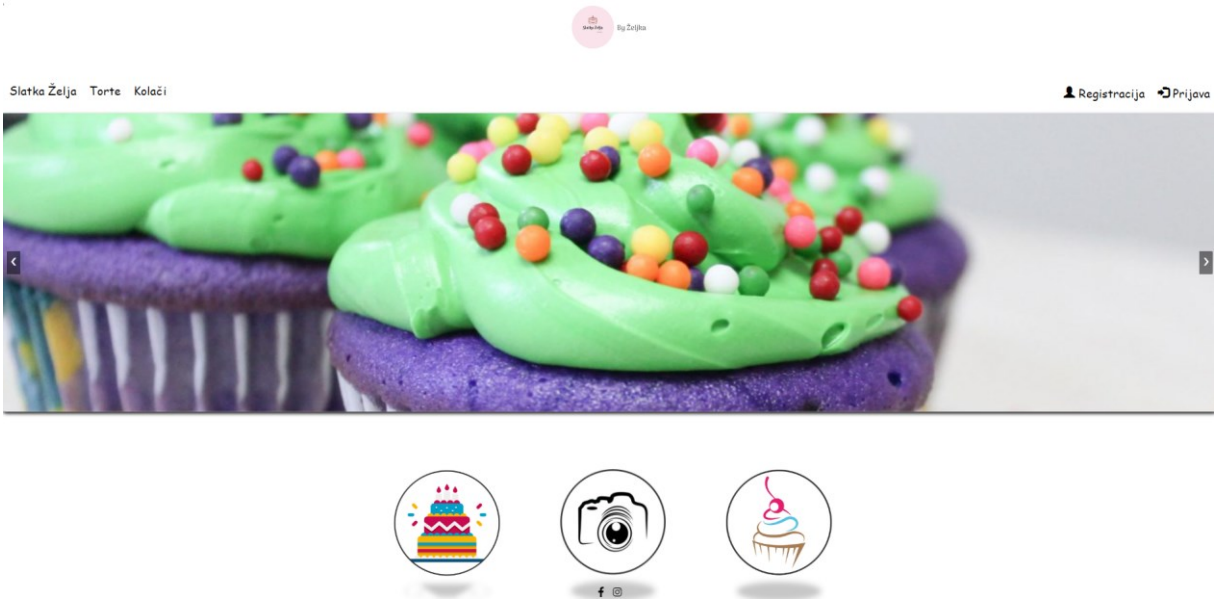
Slika 49. API update-user-password

6. KORISNIČKE UPUTE

Prikaz za korisnike bez prijave, prikaz korisnika koji je prijavljen te administratorski prikaz su tri prikaza putem kojih će se pojasniti korisničke upute mrežne aplikacije „Slatka Želja“.

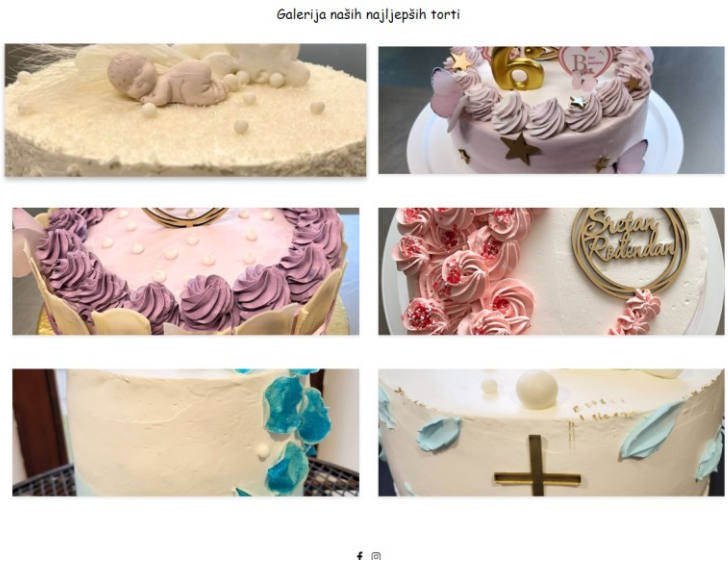
6.1. POČETNA STRANICA

Prilikom pokretanja mrežne aplikacije korisniku se učitava početna stranica na kojoj se nalazi automatska dijaprojekcija gotovih proizvoda samog obrta, a podno dijaprojekcije nalaze se tri ikone. Prikazane ikone ujedno su i linkovi koji vode korisnika na stranicu za slanje slike, personalizaciju torte ili naručivanje kolača što je prikazano na slici 50.

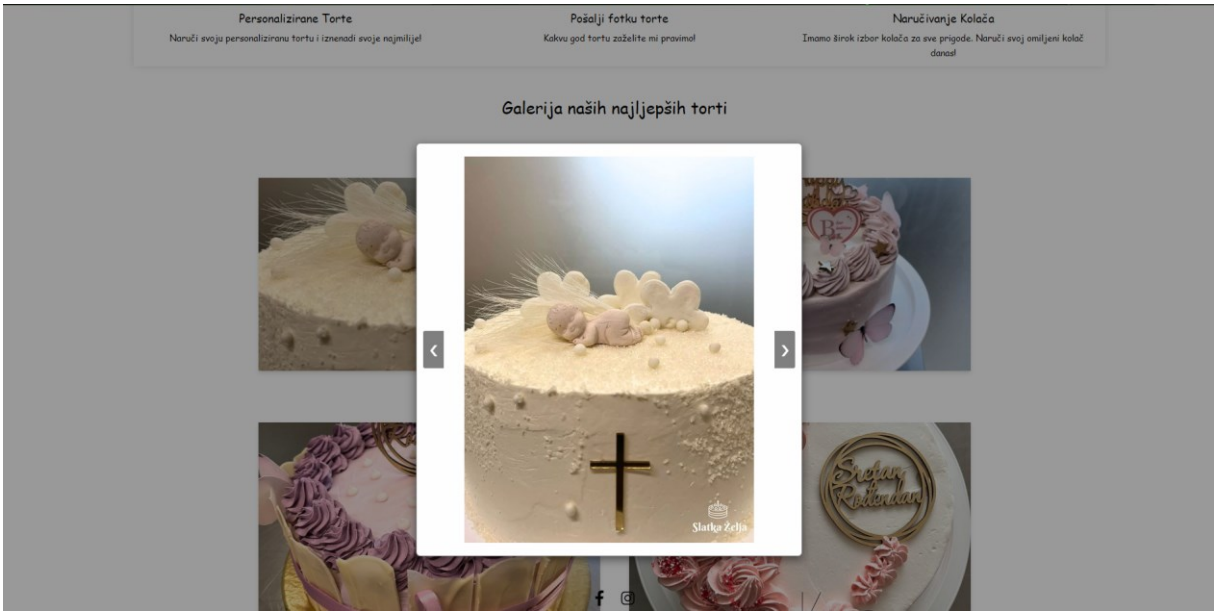


Slika 50. Naslovna stranica

Slike 51 i 52 prikazuju ostatak početne stranice na kojoj se nalazi galerija slika gdje neprijavljeni korisnici mogu dobiti uvid u gotove torte kao svojevrstan marketing.



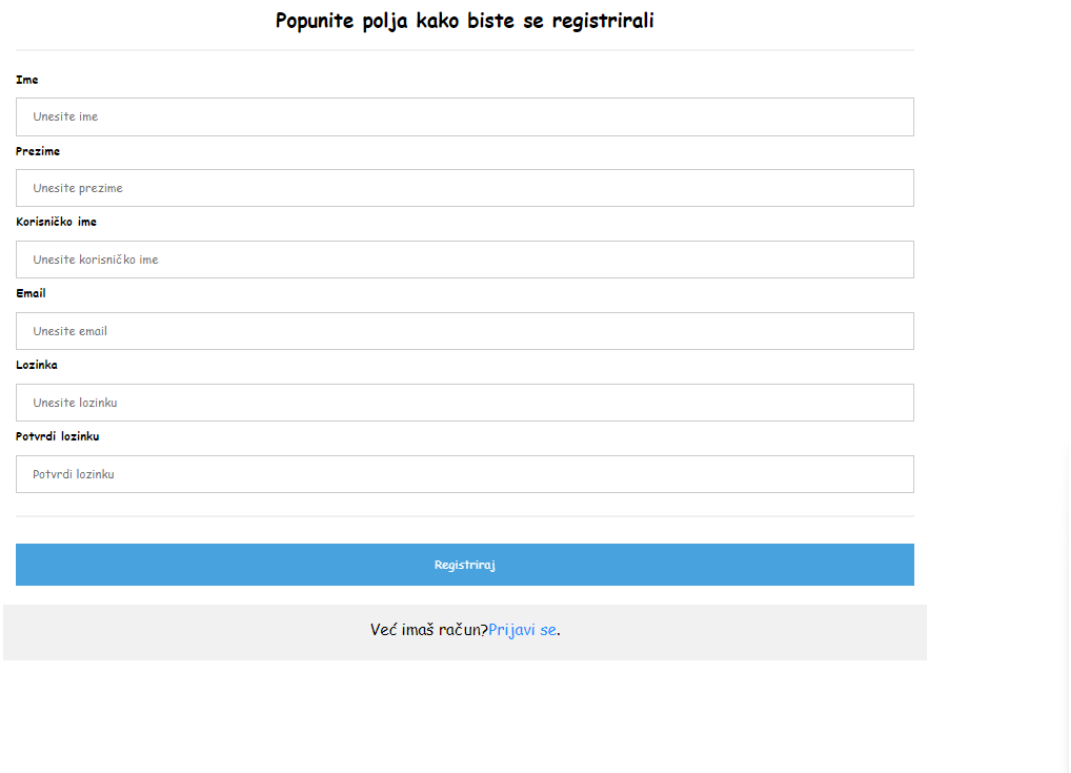
Slika 51. Galerija slika



Slika 52 Slideshow

6.2. REGISTRACIJA

Slika 53 prikazuje formu za ispunjavanje uspješne registracije novog korisnika, a sama se forma sastoji od polja: *ime*, *prezime*, *korisničko ime*, *e-mail*, *lozinka* te *potvrda lozinke*. Uspješna registracija moguća je samo ako su sva navedena polja ipravno ispunjena. Također, ispod gumba za registraciju nalazi se link za već registrirane korisnike.



Slika 53. Stranica za registraciju

Slika 54 projekcija je potencijalne situacije koja može nastati prilikom registracije u slučaju neistovjetnosti lozinke prilikom čega korisniku dolazi automatska obavijest kako je potrebno izmijeniti jednu od lozinka.

Slika 54. Greška pri registraciji

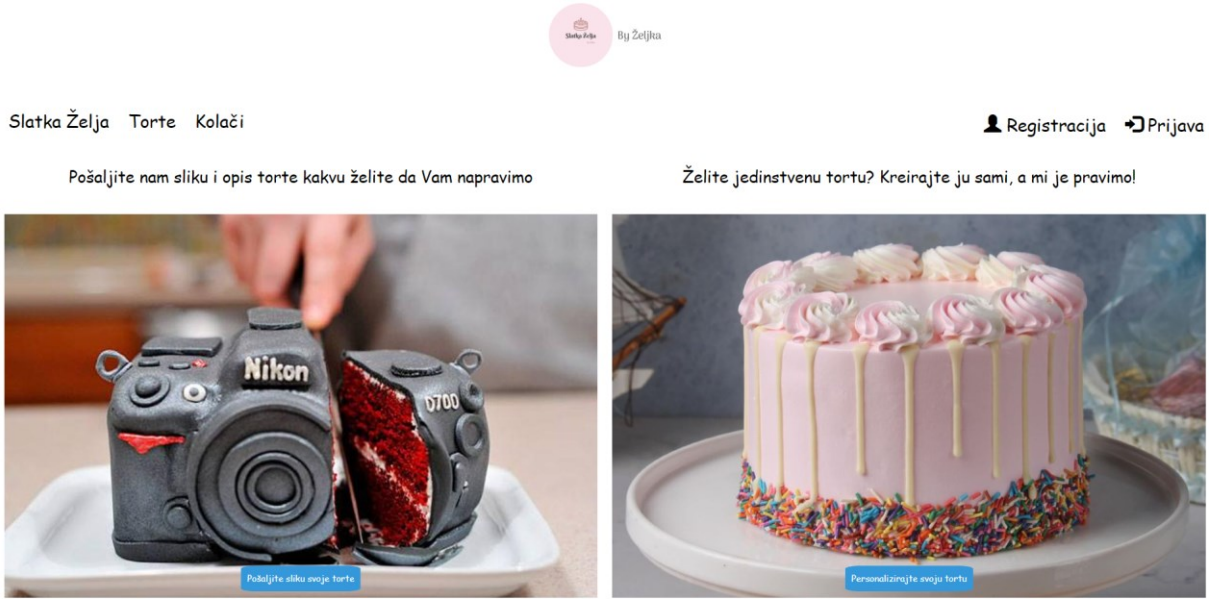
6.3. PRIJAVA

Slika 55 prikazuje formu za prijavu korisnika koja se sastoji od dva polja za unos podataka od kojih je jedan korisničko ime, a drugi lozinka pri čemu je za uspješnu prijavu nužan točan unos korisničkog imena i lozinke nakon čega se korisnik preusmjerava na početnu stranicu.

Slika 55. Stranica za prijavu

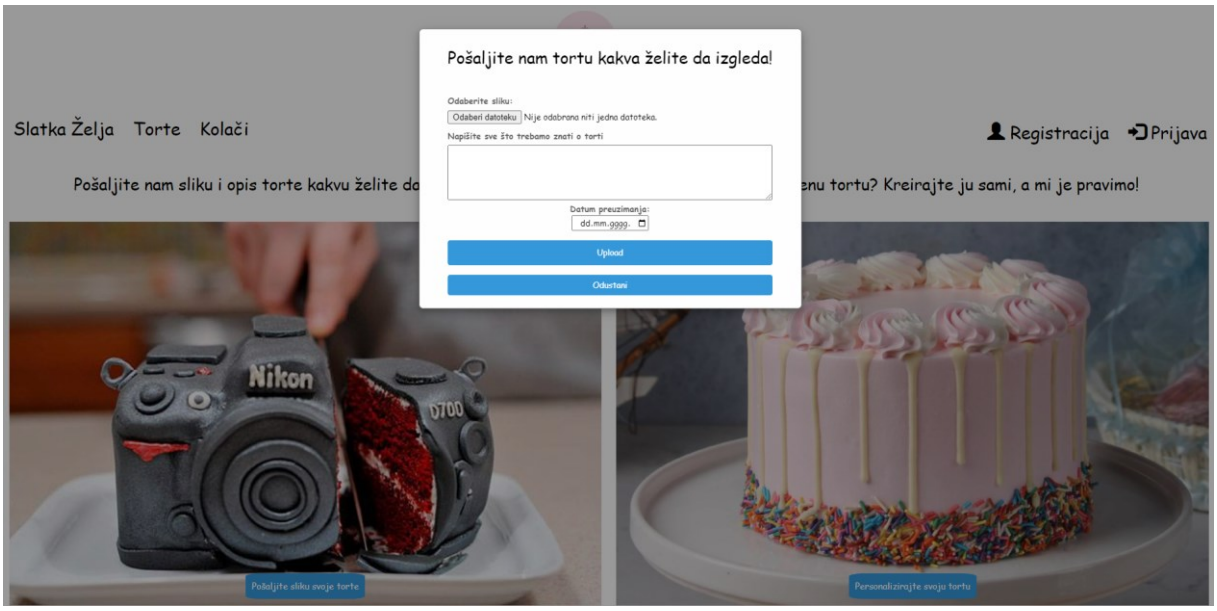
6.4. TORTE

Slika 56 prikazuje dvije slike na kojima se nalazi gumb pomoću kojeg korisnik odabire jednu od dvije opcije: pošalji sliku svoje torte ili personaliziraj svoju tortu.



Slika 56. Stranica za naručivanje torti

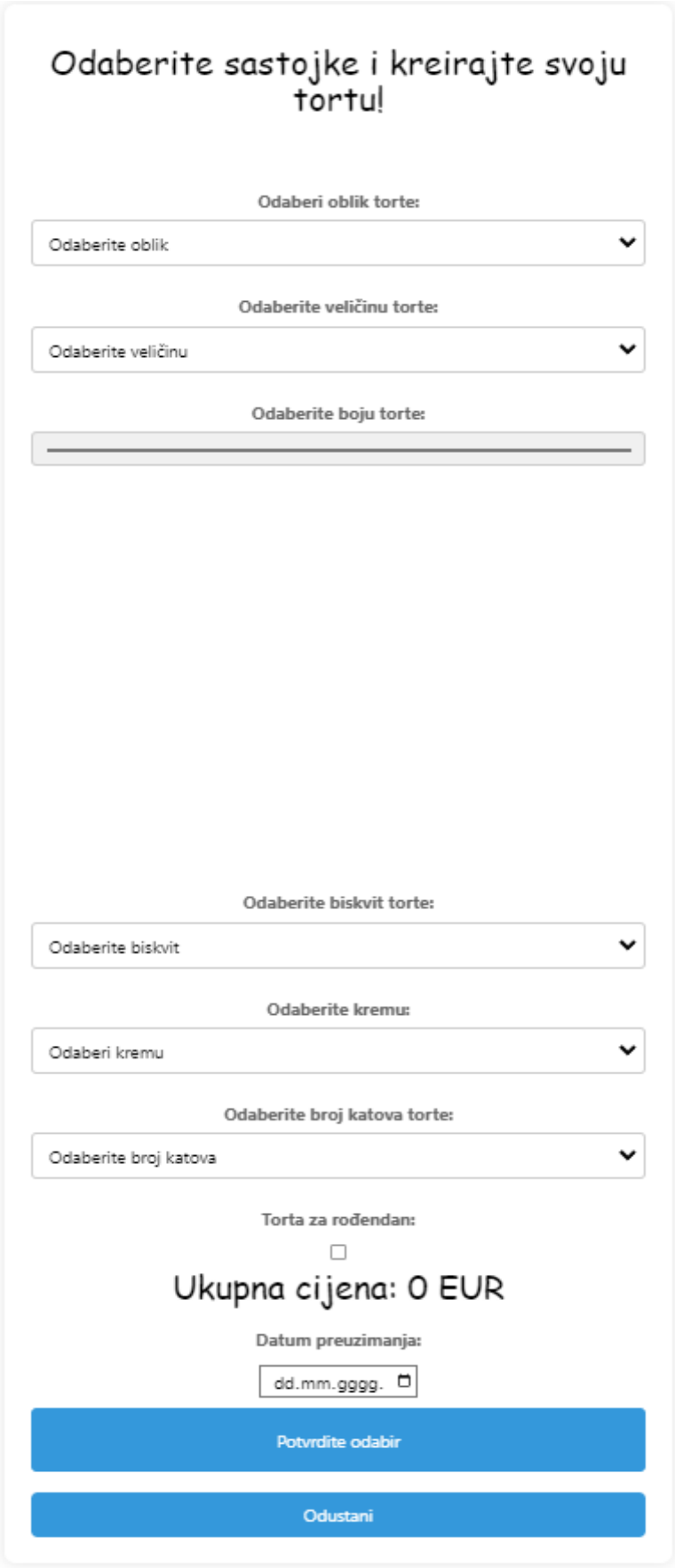
Slika 57 prikazuje skočni prozor za slanje slike torte gdje korisnik prvobitno odabire sliku koju želi poslati te dodaje komentar i datum te na kraju klikom na gumb *upload* završava proces. Također, ako kupac u bilo kojem trenutnu želi odustati to može učiniti klikom na gumb *odustani*.



Slika 57. Skočni prozor za slanje slika

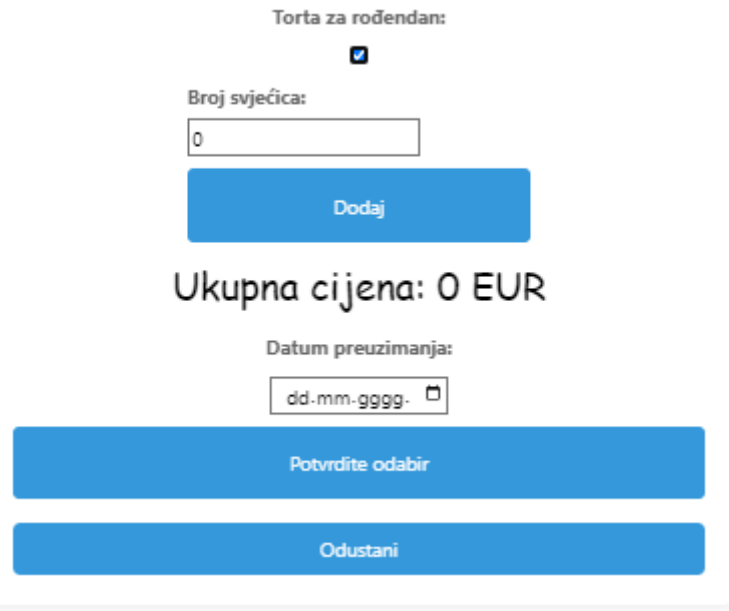
Slika 58 prikazuje skočni prozor za personalizirano naručivanje torti gdje korisnik mora odabrati tražene podatke izuzev kvadratić za rođendan koji je opcionalan. Nakon što korisnik odabere sastojke potrebno je odabrati datum za koji želi naručiti tortu te klikom na gumb potvrđuje odabir.

Isto tako, korisnik ima opciju i odustati što može klikom na gumb *odustani*. Početna je cijena svake torte 10 €, a svakim se novim odabirom polja mijenja i finalna cijena torte.



Slika 58. Skočni prozor za personalizirano naručivanje torti

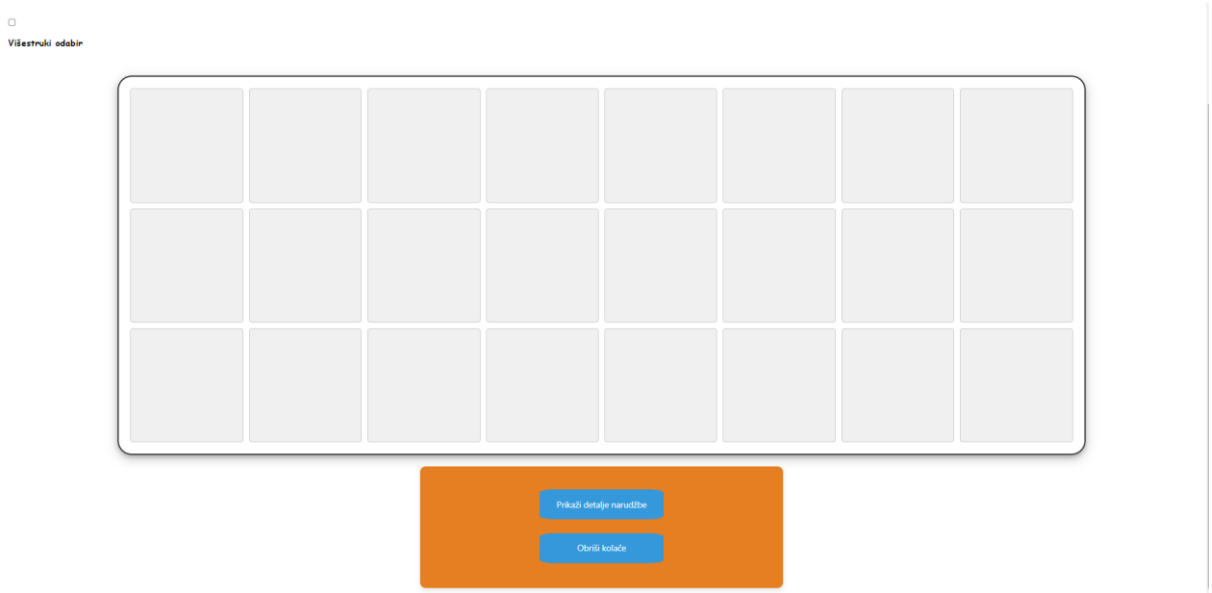
Slika 59 prikazuje kako korisnik, ako je torta namijenjena za rođendan, treba kliknuti na kvadratić *torta za rođendan* te odabrati broj svjećica na torti i kliknuti na gumb dodaj kako bih dodao svjećice na tortu prilikom čega se povećava i cijena same torte.



Slika 59. Dodavanje svijećica na tortu

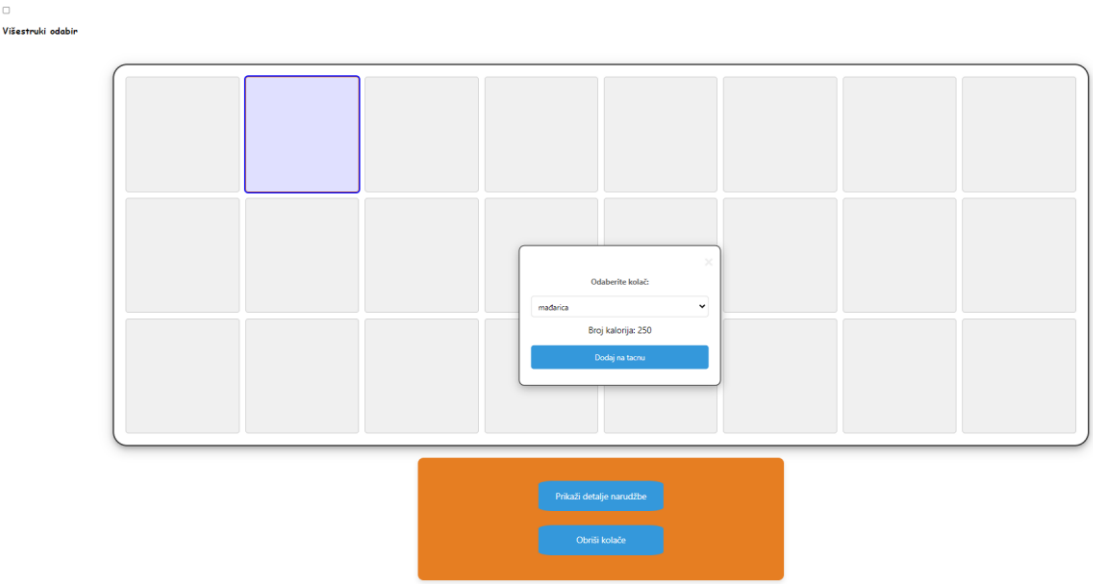
6.5. KOLAČI

Slika 60 prikazuje stranicu za naručivanje kolača gdje je prikaz stiliziranog pladnja koji se sastoji od 24 slota koji prikazuju jedan kolač na pladnju. Korisnik klikom u padajućem izborniku odabire jedan od kolača, a svaki je imenovan te sadrži prikaz broja kalorija te se na dnu nalazi gumb koji prikazuje detalje narudžbe.

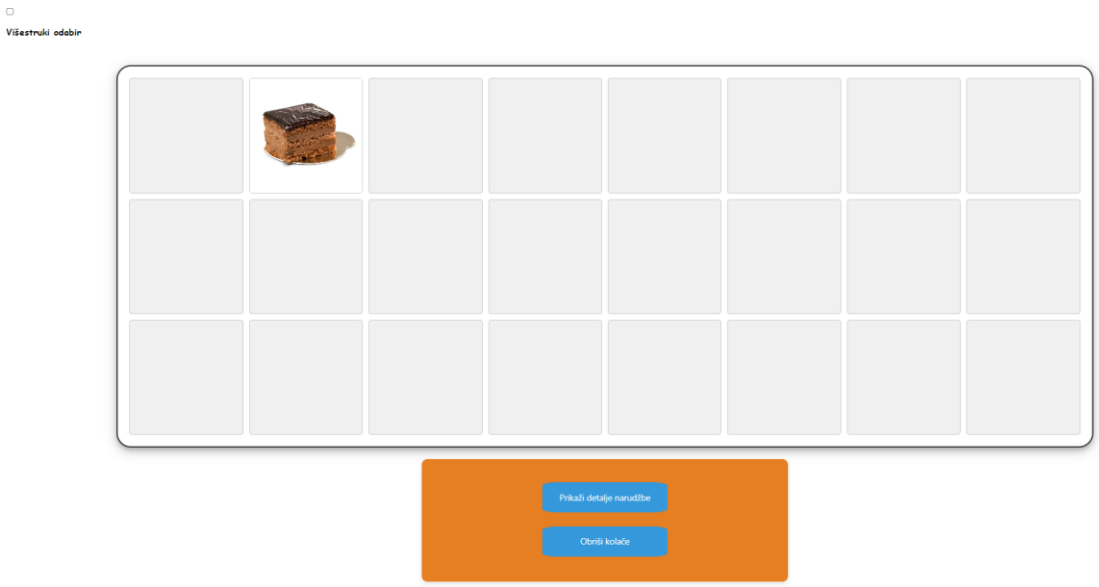


Slika 60. Stranica za naručivanje kolača

Slika 61 prikazuje proces dodavanja kolača na stilizirani pladanj; prethodno pojašnjeno; gdje se nakon dodavanja proizvoda slika istoga pojavljuje na pladnju što je prikazano na slici 62.

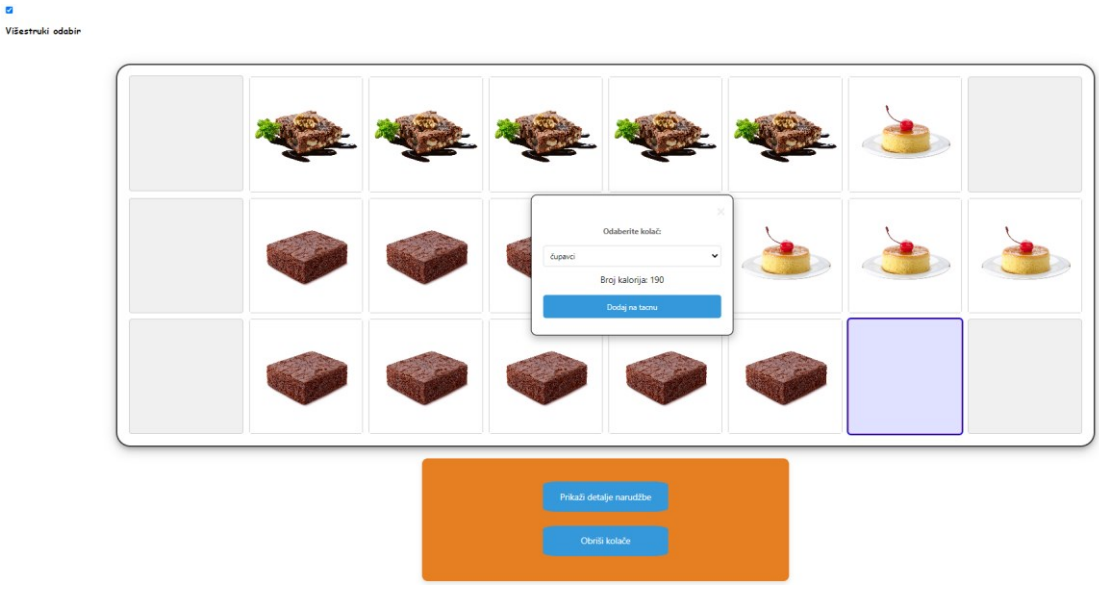


Slika 61. Dodavanje kolača na pladanj



Slika 62. Prikazivanje kolača na pladnju

Promjena dodanog proizvoda na pladnju moguća je na način da se klikom označi željeni slot, a potom se pojavi skočni prozor na kojemu odaberemo kolač po izboru. Ukoliko korisnik želi obrisati kolače sa pladnja potrebno je kliknuti na gumb *obriši kolače* te će svi kolači sa pladnja biti obrisani. Kako bi se izbjegao pojedinačan unos istog proizvoda u slotove korisnik treba odabrati kvadratić *višestruki izbor* te označiti željene slotove na koje želi staviti odabrani proizvod što je prikazano na slici 63.

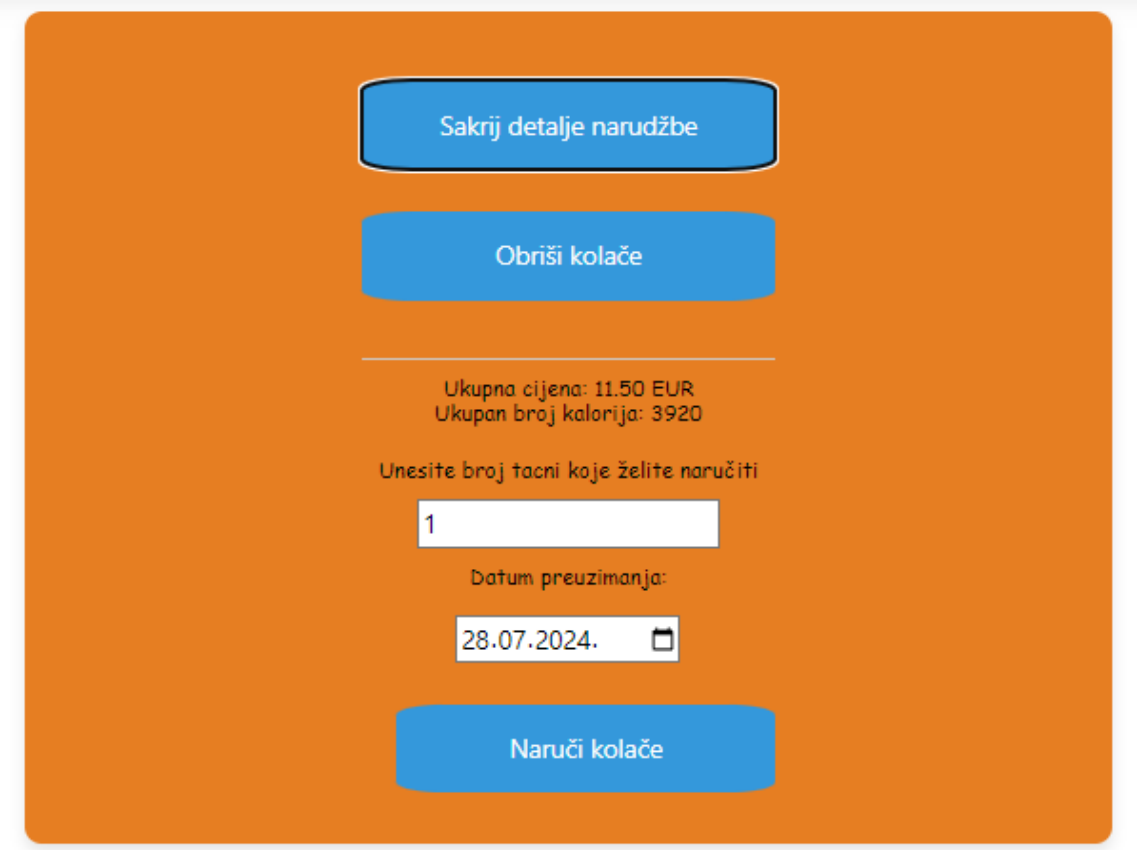


Slika 63. Višestruki izbor i zamjena kolača

Kako bi narudžba bila uspješna potrebno je postaviti minimalno 21 proizvod na stilizirani pladanj, jer u protivnome pladanj nije isplativ poslodavcu, zatim se klikom na gumb *prikaži detalje narudžbe* korisniku kreira cijena i ukupan broj kalorija. Ukoliko korisniku

odgovaraju cijena i broj kalorija utoliko je potrebno kliknuti na gumb *naruči kolače* ili korisnik može kliknuti na gumb *sakrij detalje narudžbe* kako bi podaci o cijeni i kalorijama bili skriveni.

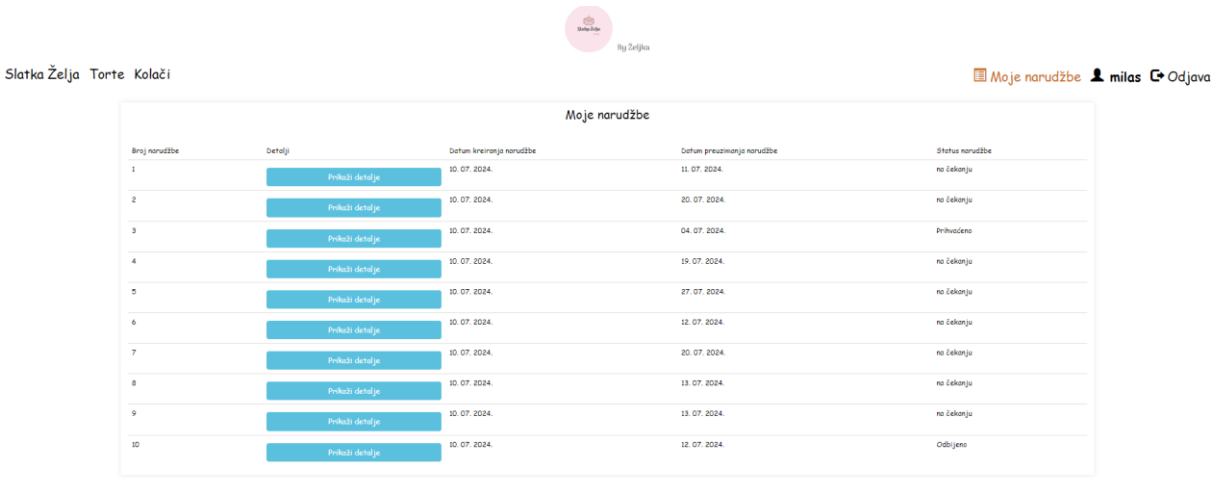
Kada korisnik odluči da je cijena proizvoda odgovarajuća unosi broj pladnjeva koje želi; minimalan broj je jedan i unese datum za kada želi naručiti kolače potrebno je kliknuti na gumb *naruči kolače* pri čemu se formira poruka o uspješnosti narudžbe, a sve navedeno prikazano je na slici 64.



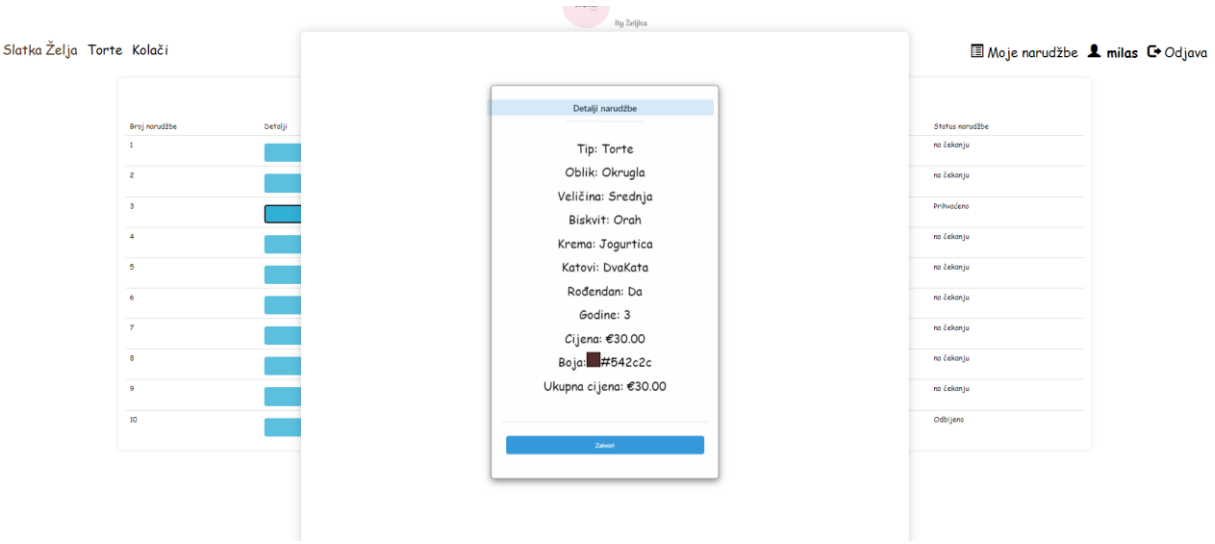
Slika 64. Detalji narudžbe

6.6. MOJE NARUDŽBE

Stranica *Moje narudžbe* prikazuje se na stranici samo ako je korisnik prijavljen, a sadrži tablični prikaz svih izvršenih narudžbi neovisno o prihvaćenosti. Tablica se sastoji od: rednog broja narudžbe, detalja te gumba na koji je moguće kliknuti kako bi se prikazali detalji narudžbe što je prikazano na slici 65. zatim datuma kreiranja narudžbe, datuma preuzimanja narudžbe i statusa same narudžbe. Stranica *Moje narudžbe* prikazana je na slici 66.



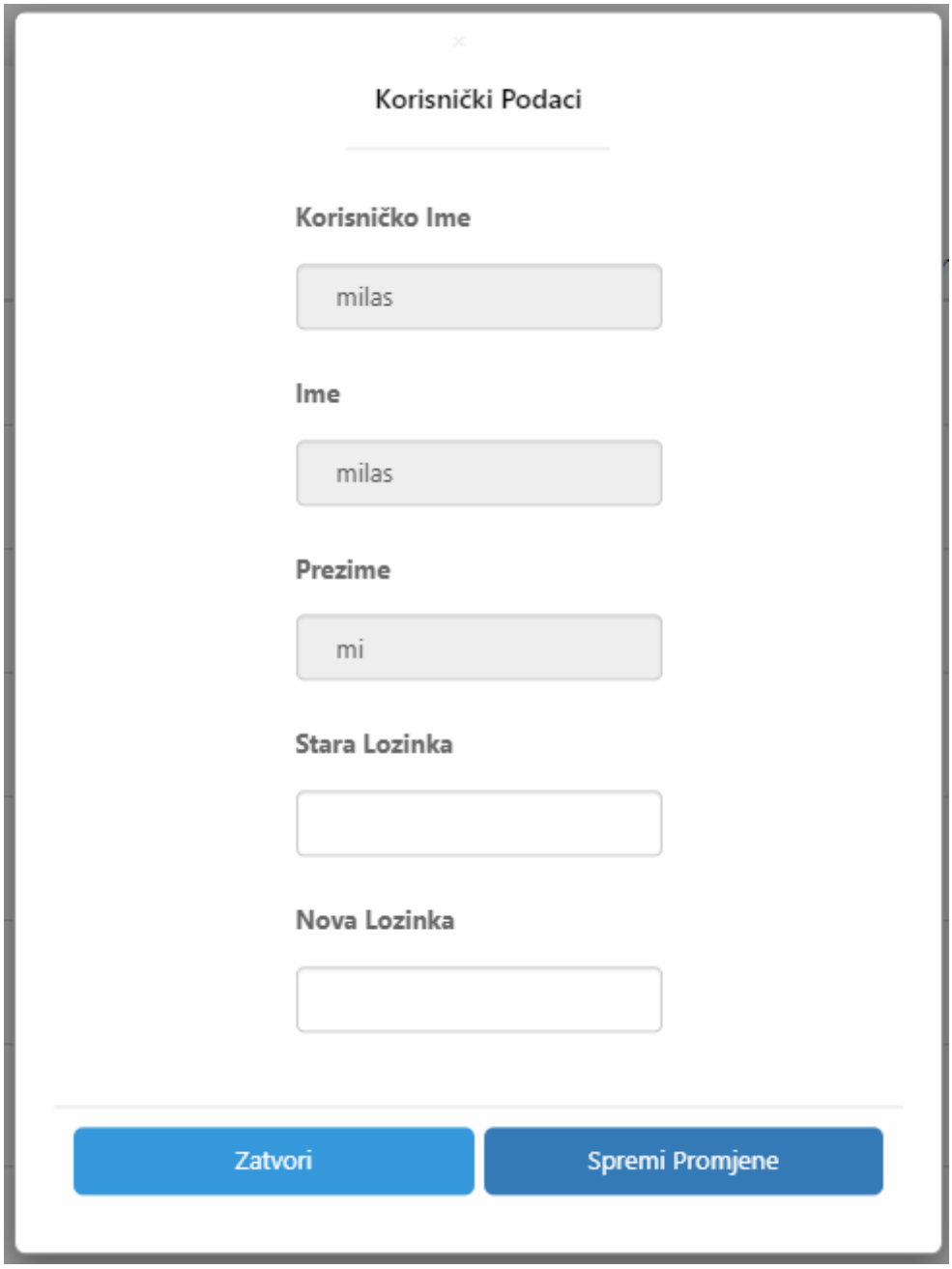
Slika 65. Moje narudžbe



Slika 66. Detalji narudžbe torte

Nakon što je korisnik uspješno prijavljen može pregledati svoje podatke klikom na ikonu za korisničke podatke prikazanu na slici 67 gdje se pojavljuje skočni prozor koji prikazuje podatke: korisničko ime, ime, prezime te je potrebno unijeti ispravnu staru lozinku kako bi bilo moguće promijeniti trenutnu lozinku na računu. Prikazano na slici 68.

Slika 67. Gumb za prikazivanje detalja i mijenjanje lozinke

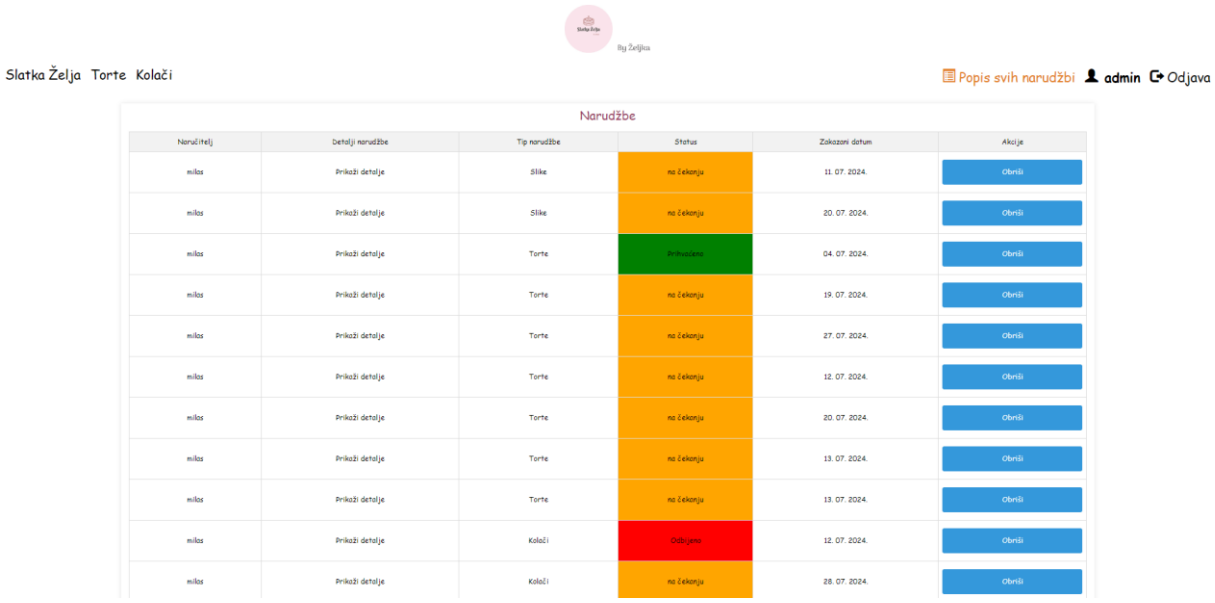


Slika 68. skočni prozor s korisničkim podacima

6.7. PREGLEDAVANJE SVIH NARUDŽBI

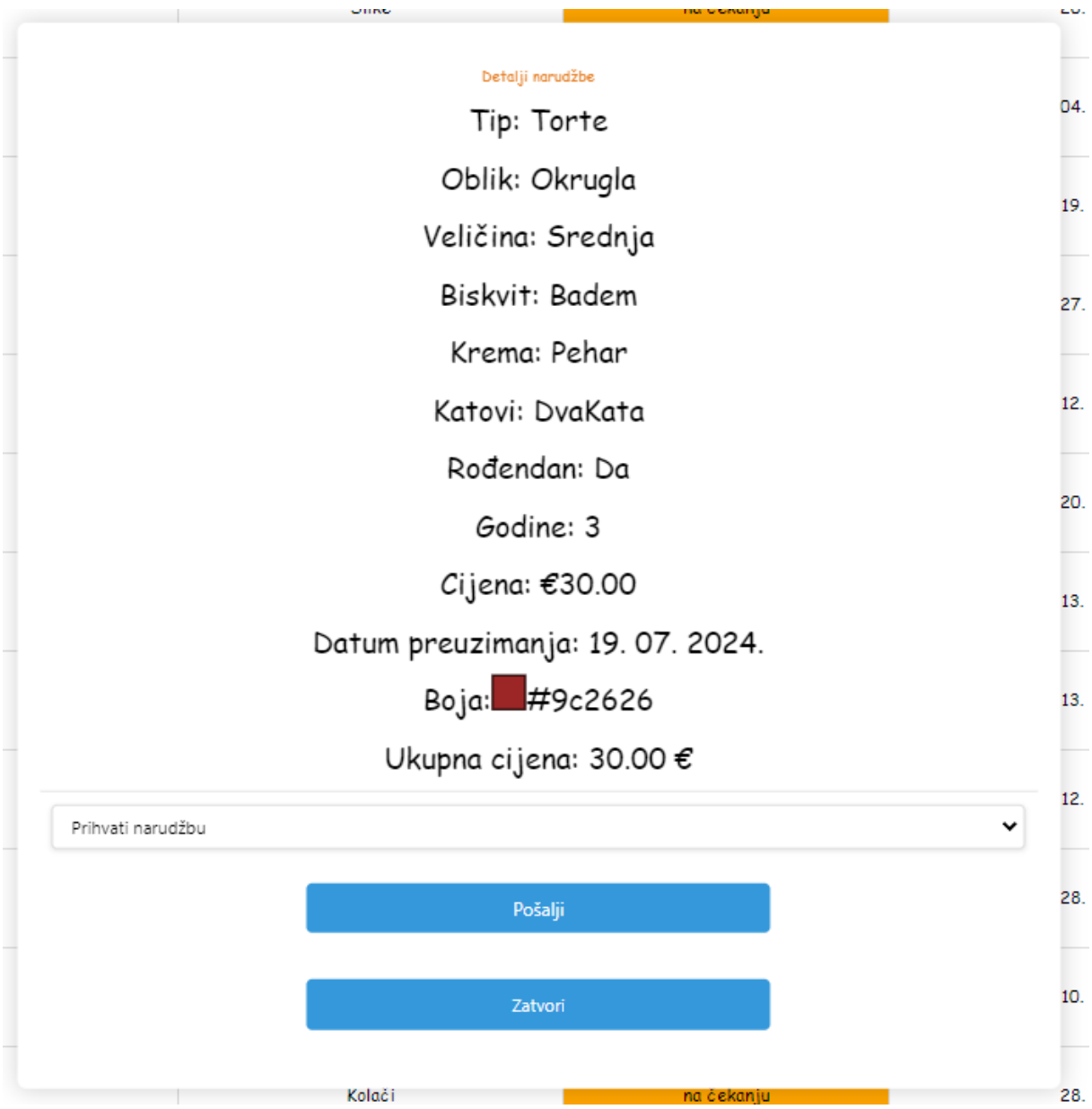
Administrator prilikom prijave jedini može pregledavati sve narudžbe koje su korisnici kreirali pritom se pojavljuje tablica koja se sastoji od naručitelja, tj. korisničkog imena naručitelja, detalja narudžbe, tipa narudžbe, statusa, zakazanog datuma i akcije u kojoj je gumb za brisanje narudžbe, a pri samome dnu tablice nalazi se gumb *odjava*.

Isto tako, statusi narudžbe, tj. ćelije ispunjene su bojama, ovisno o statusu, zbog lakšeg snalaženja u tablici. Primjerice, žuta boja je za narudžbe koje su na čekanju, zelena označava prihvaćene narudžbe, a crvena odbijene. Navedene akcije prikazane su na slici 69.



Slika 69. Sve narudžbe

Nakon što administrator klikne na gumb *Prikaži detalje* pojavljuju se detalji narudžbe slično kao i u situaciji kada korisnik pregledava vlastite narudžbe, ali za razliku od stranice *Moje narudžbe* na klijentskoj strani administrator ima mogućnost potvrđivanja, odnosno odbijanja narudžbe. Skočni prozor s prikazom detalja u padajućem izborniku omogućava administratoru odabir *prihvaćeno* ili *odbijeno*. Nakon odabira jednog potrebno je kliknuti na gumb *Pošalji* pri čemu se status automatski ažurira na stranici *Moje narudžbe* za svakog korisnika za kojeg je obrađen status narudžbe. Ukoliko administrator želi zatvoriti skočni prozor s detaljima narudžbe utoliko je potrebno kliknuti na gumb *Zatvori*, a sve opisano prikazuje slika 70.



Slika 70. Detalji narudžbe i ažuriranje statusa narudžbe

7. ZAKLJUČAK

Cilj je ovog diplomskog rada stvaranje jedinstvene mrežne aplikacije naziva *Slatka Želja* koja na moderan način pristupa slastičarskim uslugama i proizvodima pružajući klijentima mogućnost da iz udobnosti svojih domova naruče personalizirane torte i kolače. Kroz samu aplikaciju korisnici, osim klasične narudžbe, mogu vlastitim vizualom kreirati proizvod po svojoj želji što dodatno pospješuje njihovo zadovoljstvo.

Optimizacija protoka posla, a samim time i smanjenje rizika od preopterećenosti obrta putem precizno, datumski, planirane isporuke narudžbe velika su prednost i vrlo važan čimbenik u zadovoljavanju potreba između korisnika i pružatelja usluge, tj. proizvoda. Razvoj *Slatke Želje* omogućen je korištenjem *Visual Studio Code* koji je pružio jednostavno okruženje za razvoj s podrškom za sve tehnologije uključene u projekt. Upotreba *MongoDB-a* kao baze podataka, zbog svoje fleksibilnosti i skalabilnosti, bila je ključna za upravljanje velikim volumenom podataka koje ova aplikacija generira. Frontend aplikacije izrađen je korištenjem *Vue*, modernog *JavaScript framework-a* koji omogućava izradu interaktivnih i responzivnih korisničkih sučelja. Kompletan kod koji se nalazi u aplikaciji možete pronaći na linku: https://github.com/Milas117/SlatkaZelja.git

Aplikacija *Slatka Želja* predstavlja znatan korak naprijed za obrt pružajući jednostavnu platformu koja ne samo da olakšava kupcima da ostvare svoje želje, već i obrtu da unaprijedi svoje poslovanje. Također, sama je platforma kreirana na način da bude jednostavna i lako upotrebljiva svakoj osobi koja će je koristiti. Uvođenjem digitalne transformacije *Slatka Želja* postavlja temelje budućeg rasta i razvoja otvarajući mogućnosti za proširenje tržišta i pružanje usluga širem spektru klijenata. Budući razvoj aplikacije mogao bi uključivati marketinšku integraciju putem društvenim mrežama za još veću vidljivost i napredak poslovanja te uvođenje novih funkcionalnosti koje bi dodatno poboljšale korisničko iskustvo i efikasnost obrade narudžbi.

Kroz ovaj je rad mrežna aplikacija *Slatka Želja* pokazala na koji način tehnološka inovacija može transformirati tradicionalne poslovne modele i prilagoditi ih potrebama suvremenog društva pritom ističući važnost prilagodljivosti i inovativnosti u digitalnom dobu. Aplikacija je zasigurno odličan primjer i poticaj malom poduzetniku koji želi unaprijediti i povećati obujam posla što je, smatram, cilj svakog obrtnika ili poduzetnika.

8. LITERATURA

[1] F. era d.o.o, „Mlinar | Torte i kolači“. Pristupljeno: 24. srpanj 2024.
Dostupno na: https://mlinar.hr/hr/proizvodi/kategorije/torte-i-kola%C4%8Di

[2] „Slastičarnica M&M, Zagreb“, M&M slastičarnica. https://mim.hr/ pristupljeno: 24. srpnja 2024.

[3] „Slastičarnica Horak Zagreb - torte i kolači po narudžbi | Horak“. Pristupljeno: 24. srpanj 2024.
Dostupno na: https://www.horak.hr/naslovnica/

[4] „What is Visual Studio Code? Microsoft’s extensible code editor“, InfoWorld. Pristupljeno: 03. ožujak 2024
. Dostupno na: https://www.infoworld.com/article/2335960/what-is-visual-studio-code-microsofts-extensible-code-editor.html

[5] „IntelliSense in Visual Studio Code“. Pristupljeno: 03. ožujak 2024.
Dostupno na: https://code.visualstudio.com/docs/editor/intellisense

[6] „Node.js Introduction“. Pristupljeno: 06. ožujak 2024.
Dostupno na: https://www.w3schools.com/nodejs/nodejs_intro.asp

[7] „MongoDB - Working and Features“, GeeksforGeeks. Pristupljeno: 12. ožujak 2024. Dostupno na: https://www.geeksforgeeks.org/what-is-mongodb-working-and-features/

[8] „Introduction to Mongoose for MongoDB“, freeCodeCamp.org. Pristupljeno: 28. ožujak 2024.
Dostupno na: https://www.freecodecamp.org/news/introduction-to-mongoose-for-mongodb-d2a7aa593c57/

[9] „Express JS Tutorial [Understand in 5 Minutes]“, Simplilearn.com. Pristupljeno: 02. travanj 2024.
Dostupno na: https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-express-js

[10] „Vue.js“. Pristupljeno: 05. travanj 2024.
Dostupno na: https://vuejs.org/

[11] TheKnowledgeAcademy, „MongoDB Compass - Explained in detail“. Pristupljeno: 07. travanj 2024.
Dostupno na: https://www.theknowledgeacademy.com/blog/mongodb-compass/

[12] „UML Use Case Diagram Tutorial“, Lucidchart. Pristupljeno: 16. travanj 2024. Dostupno na: https://www.lucidchart.com/pages/uml-use-case-diagram

[13] „UML Class Diagram Tutorial“. Pristupljeno: 16. travanj 2024.

Dostupno na: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/

[14] „Sequence Diagram Tutorial – Complete Guide with Examples | Creately“. Pristupljeno: 16. travanj 2024.

Dostupno na: https://creately.com/guides/sequence-diagram-tutorial/

[15] „How to use an API: Guide + tutorial for beginners | Zapier“. Pristupljeno: 22. travanj 2024.

Dostupno na: https://zapier.com/blog/how-to-use-api/

[16] T. P. Team, „How to Build an API“, Postman Blog. Pristupljeno: 24. travanj 2024.

Dostupno na: https://blog.postman.com/how-to-build-an-api/

[17] „MongoDB Tutorial“. Pristupljeno: 28. travanj 2024.

Dostupno na: https://www.tutorialspoint.com/mongodb/index.htm

[18] „ExpressJS Tutorial“. Pristupljeno: 26. svibanj 2024.

Dostupno na: https://www.tutorialspoint.com/expressjs/index.htm

[19] F. era d.o.o, „Mlinar | Torte i kolači“. Pristupljeno: 24. srpanj 2024.

Dostupno na: https://mlinar.hr/hr/proizvodi/kategorije/torte-i-kola%C4%8Di

[20] Slastičarnica M&M, Zagreb“, M&M slastičarnica. pristupljeno: 24. srpnja 2024.

Dostupno na : https://mim.hr/kategorija-proizvoda/torte/design-torte/]

[21] Slastičarnica M&M, Zagreb“, M&M slastičarnica. pristupljeno: 24. srpnja 2024.

Dostupno na : https://mim.hr/proizvod/br-artikla-314/

[22] „Slastičarnica Horak Zagreb - torte i kolači po narudžbi | Horak“. Pristupljeno: 24. srpanj 2024.

Dostupno na: https://www.horak.hr/proizvod/sacher-torta/

[23] „Slastičarnica Horak Zagreb - torte i kolači po narudžbi | Horak“. Pristupljeno: 24. srpanj 2024.

Dostupno na: https://www.horak.hr/proizvod/marcipella-torta-snita/

[24] "Visual Studio Code" - Code editor. Pristupljeno 03.04.2024.

Dostupno na: https://code.visualstudio.com/

9. POPIS SLIKA