Internet aplikacija za vođenje stanja skladišta

Jukić, Franjo Josip

Master's thesis / Diplomski rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:419753

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-01**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

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

Sveučilišni studij

INTERNET APLIKACIJA ZA VOĐENJE STANJA SKLADIŠTA

Diplomski rad

Franjo Josip Jukić

Osijek, 2021.

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 06.09.2021.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime studenta:	Franjo Josip Jukić
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1061R, 06.10.2019.
OIB studenta:	28331188654
Mentor:	Doc.dr.sc. Ivan Vidović
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Izv. prof. dr. sc. Tomislav Matić
Član Povjerenstva 1:	Doc.dr.sc. Ivan Vidović
Član Povjerenstva 2:	Izv. prof. dr. sc. Ivan Aleksi
Naslov diplomskog rada:	Internet aplikacija za vođenje stanja skladišta
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	U ovom radu potrebno je istražiti postojeća rješenja za vođenje stanja skladišta. Nakon istraživanja potrebno je razviti vlastitu internet aplikaciju koja će omogućiti vođenje stanja skladišta te će imati napredne mogućnosti kao što su obavještavanje korisnika o neočekivanim odstupanjima od stanja, slanje podsjetnika za naručivanje robe i slično. Tehnologiju za izradu je moguće proizvoljno odabrati, ali je potrebno osigurati responzivnost aplikacije.
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	06.09.2021.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

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

IZJAVA O ORIGINALNOSTI RADA

Osijek, 14.09.2021.

Ime i prezime studenta:	Franjo Josip Jukić
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1061R, 06.10.2019.
Turnitin podudaranje [%]:	1

Ovom izjavom izjavljujem da je rad pod nazivom**: Internet aplikacija za vođenje stanja skladišta**

izrađen pod vodstvom mentora Doc.dr.sc. Ivan Vidović

i sumentora

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

Potpis studenta:

SADRŽAJ

1. UVOD 1

2. PREGLED POSTOJEĆIH RJEŠENJA 2

2.1. Zoho Inventory 2

2.2. InFlow 3

2.3. Katana 3

2.4. Usporedba Zoho Inventoryja, InFlow i Katana aplikacija 4

2.5. Prednosti i nedostaci predloženog rješenja 6

3. IDEJA UPRAVLJANJA SKLADIŠTEM 7

4. KORIŠTENI PROGRAMSKI ALATI I TEHNOLOGIJE 10

4.1. React 10

4.2. NodeJS 10

4.3. MongoDB 11

4.4. Insomnia 11

5. ZAHTJEVI PROJEKTA 12

5.1. Funkcionalni i nefunkcionalni zahtjevi 12

5.2. Arhitektura aplikacije 15

5.3. Shema baze podataka 16

6. PROGRAMSKO RJEŠENJE 18

6.1. Poslužitelj u NodeJS-u s MongoDB bazom 18

6.2. Klijentska strana u ReactJS-u 24

6.2.1. Prijava u sustav 28

6.2.2. Početni zaslon 30

6.2.3. Zasloni za prikaz i rad s podatcima 31

6.2.4. Automatske obavijesti 37

7. TESTIRANJE APLIKACIJE 40

7.1. Testiranje i praćenje rada poslužitelja 40

7.2. Testiranje responzivnosti.....43

8. ZAKLJUČAK.....45

LITERATURA.....46

SAŽETAK.....48

ABSTRACT.....49

ŽIVOTOPIS.....50

PRILOZI.....51

1. UVOD

U ovom radu je obrađeno upravljanje manjim skladištima ili inventarima putem internet aplikacije. Skladište predstavlja objekt u kojem se skladišti roba s namjerom kasnijeg iskorištenja ili opskrbe okolnih područja kojima su potrebni resursi. Nepravilno korištenje skladišta može dovesti do uništenja robe ili drugih gubitaka jer je roba često osjetljiva na uvjete skladištenja. Postoje različite vrste skladišta poput vanjskog, unutarnjeg, ograđenog i neograđenog [1]. Većina skladišta ima osobu zaduženu za brigu i preraspodjelu resursa te dogovaranje transporta do drugih skladišta ili radnih jedinica, no postoje i skladišta koja su u sklopu poslovnog objekta. Problemi lošeg upravljanja skladištem su nepravilno praćenje zaliha, preopterećenje resursima, nemogućnost dijeljenja podataka o zalihama i generiranja izvještaja koji prikazuju poslovanje na mjesečnoj bazi. Navedeni problemi su riješeni u ovome radu implementacijom internet aplikacije za upravljanje skladišta koja pruža dodatne mogućnosti kao što su obavijesti koje omogućuju pravovremeno reagiranje korisnika na neočekivana odstupanja u razini zaliha skladišta te pristup pregledu stanja skladišta putem interneta s bilo kojeg mjesta i u bilo koje vrijeme.

Nakon uvodnog poglavlja, u drugom poglavlju su opisani primjeri postojećih rješenja i njihova usporedba kako bi se korisniku približile razlike pojedinih softvera te prednosti softvera koji je napravljen u ovom radu, u odnosu na postojeća rješenja. Treće poglavlje opisuje ideju upravljanja skladištem, trenutni način upravljanja i općenite prednosti softvera za upravljanje skladištem. Četvrto poglavlje opisuje korištene alate i tehnologije za izradu internet aplikacije. U petom poglavlju su definirani zahtjevi projekta poput funkcionalnih i nefunkcionalnih zahtjeva, arhitekture aplikacije i baze podataka. Šesto poglavlje sadrži programsko rješenje za upravljanje skladištem koje se bazira na klijent-poslužitelj arhitekturi što omogućuje veću fleksibilnost u daljnjem razvoju aplikacije te mogućnost proširenja na druge platforme uz korištenje istog poslužiteljskog dijela. U zadnjem poglavlju je objašnjeno testiranje funkcionalnih i nefunkcionalnih zahtjeva.

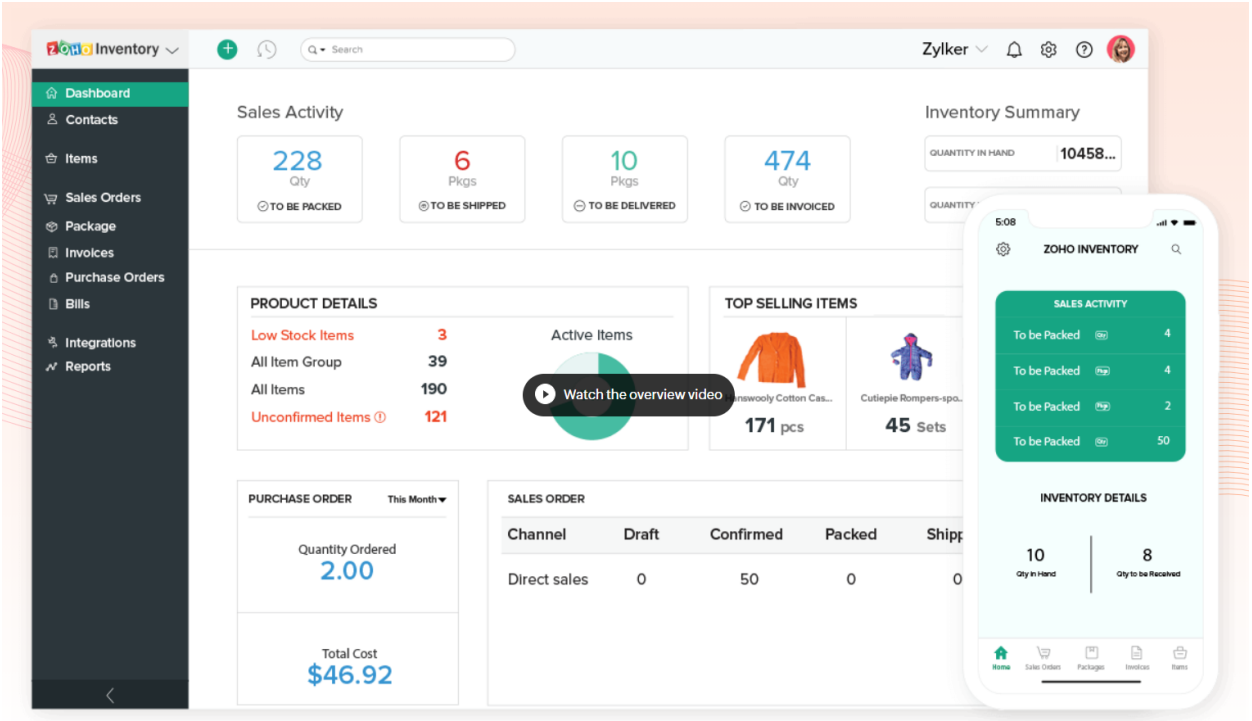
2. PREGLED POSTOJEĆIH RJEŠENJA

Kako bi se odabrao najbolji softver za upravljanje zalihama potrebno je procijeniti poslovanje, analizirati vlastite potrebe i zatim na temelju toga izabrati aplikaciju. U ovom poglavlju su uspoređena tri primjera aplikacijskog rješenja upravljanja skladištem, a to su *Zoho Inventory*, *InFlow* i *Katana*.

2.1. Zoho Inventory

Zoho Inventory je sustav kontrole zaliha zasnovan na oblaku usluga. Omogućava stvaranje strategije prodaje, upravljanje narudžbama za obnavljanje zaliha i cjelovit sustav kontrole zaliha (Slika 2.1.). Glavne značajke su:

1. centralizirani inventar – količine zaliha se automatski ažuriraju na svim povezanim mjestima prilikom izmjene stanja proizvoda,
2. popis proizvoda – lakša organizacija proizvoda i informacija poput detalja, cijene, dostupnosti i skladišta,
3. automatsko ponovno naručivanje – mogućnost postavljanja podsjetnika, dobavljača proizvoda i vremena kada se želi napraviti automatska narudžba potrebnih zaliha.

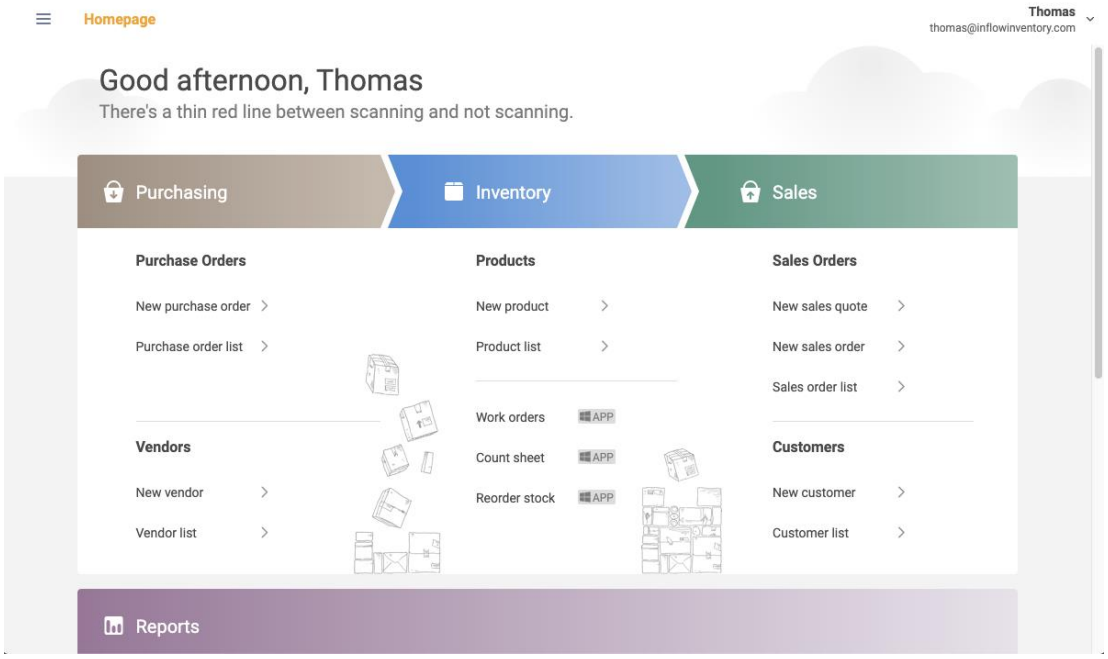


Slika 2.1. *Prikaz Zoho Inventory aplikacije za upravljanje skladištem [5].*

Osim upravljanja zalihama aplikacija dopušta dodavanje više skladišta i kontroliranje protoka između npr. trgovine i skladišta koji su smješteni na različitim mjestima. Pomoću ove aplikacije moguće je voditi evidenciju svih narudžbi, pratiti otpremljene resurse i generirati izvješća koja pomažu u vođenju skladišta [5].

2.2. InFlow

InFlow softver pruža jednostavan način upravljanja skladištem i poslovanjem (Slika 2.2.). Omogućava praćenje zaliha u stvarnom vremenu i nudi rješenje vođenja više skladišta gdje se svakom korisniku mogu odrediti prava pristupa pojedinom skladištu [6].



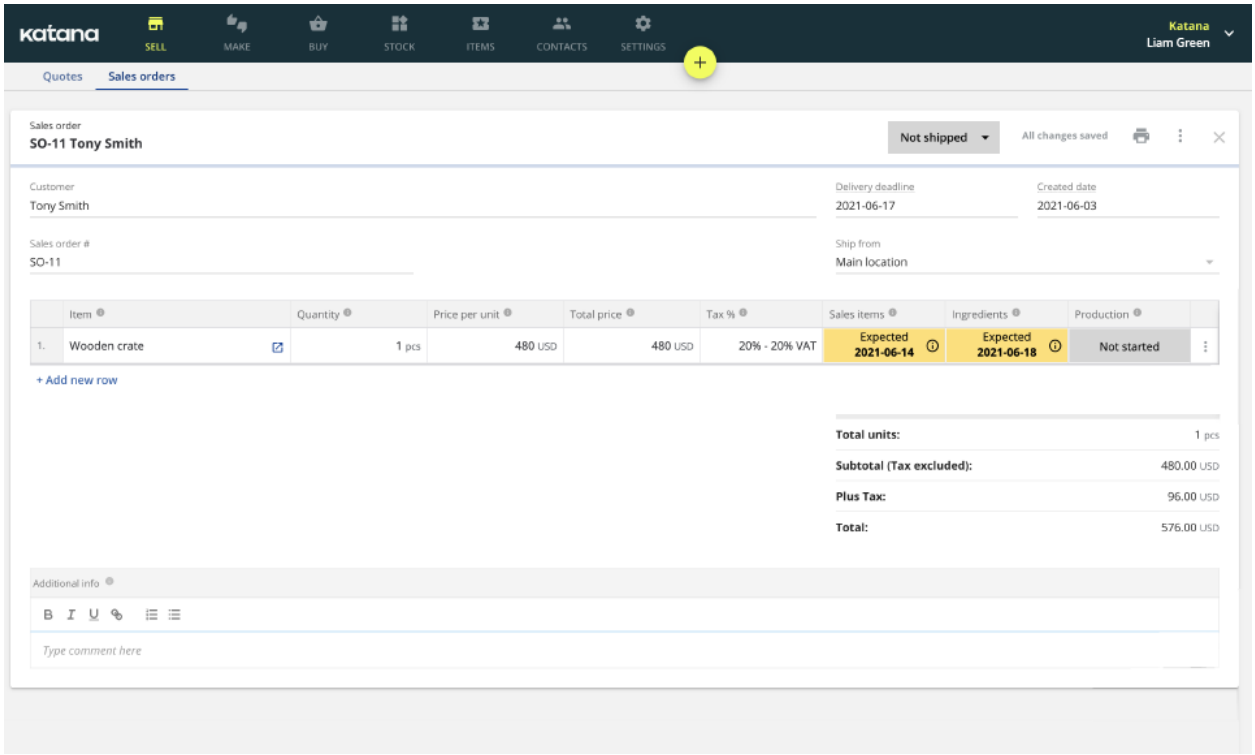
Slika 2.2. *Prikaz InFlow aplikacije za upravljanje zalihama [6].*

Aplikacija nudi generiranje i pregled izvještaja na dnevnoj, mjesečnoj i godišnjoj bazi te više različitih tipova datoteka poput CSV, XSLX i slično.

2.3. Katana

Katana je pametan i vizualni proizvodni softver koji je prilagođen skaliranju skladišta (Slika 2.3.). Olakšava upravljanje proizvodnjom i kontrolu zaliha te je pogodan za tvrtke koje žele ulagati u pravilnu strukturu u svom poslovanju. Dobitnik je nagrade za softver za najjednostavnije upravljanje skladištima. Integracija *Katane* u poslovanje je jednostavna i omogućava evidenciju kretanja zaliha, računa i prodajnih naloga te spajanje i dijeljenje informacija s raznim aplikacijama

poput *QuickBooks Online* i *Xero*. Uz to ima mogućnost automatizacije pojedinih dijelova sustava poput automatskih izvješća za zadane periode [7].



Slika 2.3. *Prikaz Katana početnog zaslona za praćenje prodaje [7].*

2.4. Usporedba Zoho Inventoryja, InFlow i Katana aplikacija

Svaka tvrtka prije odabira aplikacije treba odrediti širinu svog poslovanja i vlastite potrebe. U tablici 2.1. je prikazana usporedba spomenutih aplikacija za upravljanje zalihama u skladištu. Sve ove aplikacije sadrže bazni koncept, a to je da svaki alat sadrži mogućnosti poput upravljanja stanjem, praćenje stanja zaliha, pregled i generiranje izvještaja te pristup putem interneta. Navedene su ostale karakteristike prema kojima se može vidjeti koje značajke neka aplikacija posjeduje, što olakšava odabir.

Tablica 2.1. *Prikaz usporedbe između aplikacija Zoho Inventory, InFlow i Katana [8].*

Aplikacija	Zoho Inventory	InFlow	Katana
Ocjena	4.5/5	4.8/5	4.6/5
Usluge			
Besplatna verzija	✓	✓	×
Osnovni paket	59 dolara mjesečno	71 dolar mjesečno	99 dolara mjesečno

Plus paket	159 dolara mjesečno	179 dolara mjesečno	299 mjesečno
Platforme			
Stolne platforme	×	Windows Virtualni Windows Virtualni Linux	Mac Windows
Mobilne platforme	iOS Android	iOS Android	×
Obuka [✓ / ×]			
Uživo	×	✓	✓
Preko interneta	✓	✓	✓
Videokonferencija	✓	✓	✓
Dokumentacija	✓	✓	✓
Video	✓	✓	✓
Karakteristike [✓ / ×]			
Računovodstvena integracija	×	✓	✓
Obavijesti	✓	✓	✓
Automatski raspored	×	×	✓
Predviđanje	✓	✓	✓
Kategorizacija	✓	✓	×
Barkodiranje / RFID	✓	✓	✓
Više lokacija	✓	✓	✓
Analitika	✓	✓	✓
Upravljanje sigurnošću	×	×	✓

Područja korištenja te konkretne prednosti i nedostaci pojedine aplikacije:

Zoho Inventory – najbolji je za internetska poduzeća i prodavače, maloprodaje te manja i srednja poduzeća koja upravljaju zalihama. Prednost je što sadrži besplatnu verziju, lagan je za korištenje i snalaženje u velikoj količini kategorija, lagan za planiranje te olakšava stvaranje prodajnih naloga

i računa. Nedostaci su to što je ograničen s dubinom razina zaliha do koje se može upravljati, teško je izvoziti velike količine podataka u nekom obliku datoteke poput CSV-a i nedostaju aplikacije s kojima se može integrirati,

InFlow – pogodan je za manja poduzeća, maloprodaje i internetske trgovine koje se bave upravljanjem skladištem. Prednost je lako distribuiranje velikih skladišta s puno proizvoda, lagan način fakturiranja, jednostavno kreiranje izvješća i pretraga. S druge strane, nedostaci su manjak integracije s drugim aplikacijama, neki dijelovi se moraju ručno unositi, samo jedna slika se može unijeti za proizvod na Windows aplikaciji dok na mobilnim uređajima postoji mogućnost unošenja više slika,

Katana – najbolja za integraciju s drugim aplikacijama, internetsku prodaju te upravljanje zalihama manjih i većih poduzeća. Lako se postavljaju prioriteti za poslove koji se trebaju obavljati, automatsko upravljanje i prilagodljiv dizajn te pomoć pri unaprjeđenju poslovanja, dok su nedostaci loša korisnička podrška, često se pogreške ne mogu popraviti i nema sigurnosnih kopija.

2.5. Prednosti i nedostaci predloženog rješenja

Rješenje napravljeno u sklopu ovog diplomskog rada nudi prednosti u odnosu na većinu postojećih aplikacija na tržištu. Glavni potencijal aplikacije je to što je predviđena i prilagođena za hrvatsko tržište na kojem se trenutno rijetko koriste aplikacije ovakvog tipa. Slijedno tome aplikacija pruža sadržaj na hrvatskom jeziku, dok ostale aplikacije većinom nemaju za izbor hrvatski jezik. U slučaju komercijalizacije, cijena aplikacije bi bila prihvatljiva. Jednostavnost aplikacije smanjuje potrebe za složenom dokumentacijom, a pristup internetu omogućava svakodnevni pregled stanja i generiranje novih izvješća. Također, omogućava praćenje svih preuzimanja, dodavanje zaliha, pravljenje inventura, obavještavanje korisnika o neočekivanom odstupanju od stanja te periodično slanje izvješća. Glavni nedostaci predloženog rješenja su to što trenutno nema mogućnost zapisivanja proizvoda putem barkoda i potrebno je određeno vrijeme kako bi se nakon uspješne prilagodbe na tržište nadogradila i omogućila usluga prema korisnicima putem mobilne aplikacije. Također, potrebno je odrediti ciljano tržište za aplikaciju.

3. IDEJA UPRAVLJANJA SKLADIŠTEM

Ideja upravljanja skladištem i zalihama se razlikuje od kontrole zaliha. Iako se ova dva pojma često poistovjećuju, postoji ključna razlika među njima, a to je da se kontrola zaliha odnosi na upravljanje zalihama koje su trenutno na skladištu, dok je upravljanje šire od kontrole i uzima u obzir cijeli proces od dobavljanja, skladištenja i prodaje. Kontrola zaliha uključuje poznavanje zaliha iznutra i izvana, koliko ih je dostupno, gdje i u kakvom su stanju [2]. Upravljanje skladištem osigurava učinkovito skladištenje zaliha, smanjivanje troškova skladištenja i vremena potrebnog za brojanje zaliha. Svako poduzeće prije kontrole zaliha treba uspostaviti sustav upravljanja. Upravljanje zalihama određuje kako će se voditi posao, osigurati zadovoljstvo kupaca i povećati prodaja. Slika 3.1. prikazuje primjer u kojem se kupuje više proizvoda nego je potrebno. Zbog toga dolazi do nepotrebnih troškova i gubitaka pri osiguranju dodatnog prostora skladišta.

Slika 3.1. *Problem skladištenja prevelike količine zaliha [3].*

Tržište upravljanja manjim skladištima i inventarima je veoma razvijeno i kao takvo nudi postojeća rješenja u raznim oblicima od kojih se danas najčešće koriste internetski alati za upravljanje zalihama.

U samim početcima upravljanja zalihama, evidencija stanja skladišta se provodila putem papira što nije bio problem za jednostavne zapise. Širenjem skladišta postalo je teže održavati i zapisivati sve na papir. S vremenom se počeo koristiti *Excel* alat koji uz pomoć *Excel* tablica omogućava prijenos podataka i na druga računala. *Excel* tablica je dobra na osnovnoj razini kada nema puno podataka za upisivanje, a uz to pruža jednostavan način vođenja. Glavni nedostatak *Excela* je mogućnost ljudske pogreške pri upisu u tablicu i pri pojavi kvarova na računalu potencijalni gubitak podataka.

Nakon *Excela* primjenjivao se osnovni softver za upravljanje zalihama. Mnoge aplikacije su zasnovane na radu softvera kao usluga putem oblaka[1]. Unatoč neispunjavanju svih zahtjeva korisnika, pronašao je klijente zbog jednostavnosti korištenja i naprednijeg pristupa od *Excela*. Prednost internetskog alata je to što omogućuje korištenje alata s bilo kojeg mjesta koje ima pristup internetu, bez potrebe za instalacijom ili ažuriranjem u slučaju novije verzije. Nedostatak osnovnih alata je to što su često namijenjeni samo za jedno skladište i kao takvi nisu fleksibilni za prilagodbu korisnikovim potrebama.

Iduća nadogradnja osnovnog internetskog alata je namjenski softver za upravljanje zalihama. Za razliku od prethodnog, ovaj alat nudi mogućnost praćenja i kontrole zaliha. Također, zasnovan je kao alat putem oblaka, no moguće ga je sinkronizirati s drugim aplikacijama poput *Amazon, Shopify* i *Xero* što omogućava dijeljenje i pristup informacijama o praćenju i kontroli zaliha. Lako se integrira u druge sustave što ga čini veoma fleksibilnim i dizajniran je prema zahtjevu korisnika, no zahtjeva pravilno postavljanje jer se u slučaju neispravnog postavljanja ne može iskoristiti puni potencijal alata.

Veća poduzeća koja svoje upravljanje zalihama žele dovesti na višu razinu kao glavni cilj stavljaju planiranje resursa koje pokriva svaki aspekt poslovnog planiranja. Umjesto više postavljenih različitih komponenti u oblaku omogućava se odabir i instaliranje potrebnih modula za upravljanje zalihama. To se postiže jedinstvenim sustavom upravljanja resursa ili ERP (engl. *Enterprise Resource Planning).* Ovakav sustav obuhvaća zalihe, računovodstvo i lanac opskrbe čime je pogodan za veća poduzeća, no zbog toga je skuplji i zahtjeva održavanje i nadogradnju uz dodatne troškove te je potrebna radna snaga i vrijeme za implementaciju [3].

Nakon određenog vremena upravljanja zalihama dolazi se do glavnog pitanja *Kada je vrijeme za nadogradnju?*. Kako bi se odgovorilo na to pitanje potrebno je popratiti znakove i pokazatelje vezane za upravljanje zalihama:

1. proračunske (*Excel*) tablice postaju nefleksibilne za upravljanje zalihama – potrebno je puno više stranica ili zaposlenici ne mogu pristupati podacima u stvarnom vremenu,
2. potrebna veća točnost – postaje teško pratiti koliko ima određenih stvari u bilo kojem trenutku,

[1] Oblak usluga – gotove usluge koje je moguće koristiti putem interneta

3. troškovi zaliha rastu – trošak održavanja raste i koristi se puno više resursa,
4. usporavajući rast – ručno pisanje i ispisivanje izvještaja postaje otežano zbog velike količine podataka.

Ključne prednosti softvera za upravljanje zalihama u oblaku su raznolike, no mogu se svrstati u tri široka područja prikazana na slici 3.2.

Slika 3.2. *Područja prednosti softvera za upravljanje zalihama [3].*

Prvo područje je poboljšanje učinkovitosti. Odnosi se na održavanje zaliha na razini s potražnjom. Potrebno je vidjeti gdje se i u kojoj količini nalaze zalihe kako bi se znalo trenutno stanje u stvarnom vremenu. Zatim je potrebno uskladiti razine potražnje, ravnotežom između rizičnih i prekomjernih zaliha, a to se radi na temelju dosadašnjih podataka i predviđanju budućeg korištenja zaliha [4]. Nakon toga potrebno je obratiti pozornost na odnose s dobavljačima, pa pomoću alata vidjeti tko isporučuje na vrijeme i tko ima najbolje cijene i prema tome odlučiti o daljnjoj suradnji.

Drugo područje se odnosi na rastuću prodaju jer prodaja je okosnica svakog posla. Za maksimiziranje prodaje i zarade potreban je učinkovit sustav zaliha. To se postiže aplikacijom koja ima pristup stanju skladišta u stvarnom vremenu bez potrebe za ručnim unosom podataka ili čestim provjeravanjem stanja skladišta. Kvaliteta dobavljenih proizvoda mora biti na zadovoljavajućoj razini kako bi se ispunile potrebe kupaca koji ove proizvode iz skladišta koriste u bilo kojem obliku i području, od uslužnog prostora poput kafića, restorana do trgovina i trgovačkih lanaca [4].

Treće područje je skaliranje koje predstavlja mogućnost proširenja skladišta na više lokacija. Time raste složenost upravljanja, no na ovaj način moguće je uštedjeti vrijeme i resurse jednostavnim dodavanjem novog mjesta, grada, skladišta ili proizvoda.

4. KORIŠTENI PROGRAMSKI ALATI I TEHNOLOGIJE

4.1. React

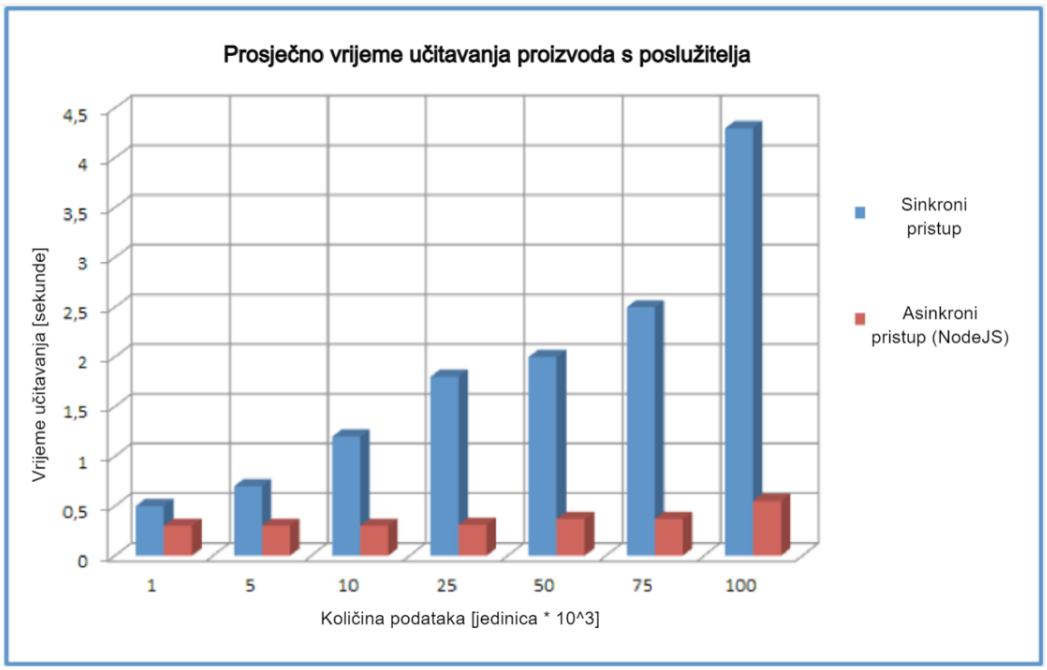
React je jedna od najpopularnijih *JavaScript* biblioteka koja se koristi za kreiranje korisničkih sučelja i koriste ju velike tvrtke poput *Facebooka, Netflixa* ili *Instagrama*. Omogućava kreiranje prilagodljivog i modularnog dizajna što ju čini veoma pogodnom tehnologijom za ovaj rad. Sučelje se može uređivati pomoću poznatih jezika za uređivanje poput *CSS*-a ili napredne verzije *SCSS*-a. Za ikone, kosture tablica, gumbove i slično koriste se gotove komponente iz *MaterialUI* biblioteke [9].

Drugi razlozi za odabir ove tehnologije su to što je u odnosu na *Angular*, *React* lakši za naučiti i jednostavniji za kreiranje, ima bolju učinkovitost te troši manje resursa [10]. *React* sadrži komponente koje se ponašaju kao gradivni elementi i moguće ih je ponovno koristiti uz pravilnu implementaciju. Koristi *JSX* sintaksu što je zapravo napredan JavaScript i omogućava pisanje *HTML* elemenata u obliku koda te direktno upisivanje vrijednosti u njih.

Koristi virtualni objektni model dokumenta, *DOM* (engl. *Document Object Model)* koji je apstrakcija osnovnog *DOM*-a. *DOM* je sučelje koje stvara internet stranica prilikom učitavanja i direktna promjena ovog modela uzima više resursa i vremena za ponovno učitavanje. Virtualni *DOM* uzima vrijednost koja se promijenila i prolazi cijelu svoju strukturu, uspoređujući prethodno stanje s novim objektom. Nakon toga ažurira samo izmijenjeni objekt i prikazuje promjenu na stranici, umjesto ažuriranja i ponovnog prikaza cijele stranice što je slučaj kod *Angulara*. Za praćenje i upravljanje stanjem koristi se *Mobx* biblioteka.

4.2. NodeJS

NodeJS predstavlja *V8 JavaScript* okruženje koje se može koristiti na više platformi i izvodi se u jednom procesu, bez potreba za stvaranjem novih niti za svaki zahtjev. Glavna ideja *NodeJS*-a je to što koristi neblokirajuće procese koji se samostalno izvode u pozadini [11]. Prednost korištenja *NodeJS*-a u ovom radu je povećanje brzine odgovora na zahtjeve, kompatibilnost i jednostavnost postavljanja poslužitelja (engl. *server*) za *MongoDB* bazu.



Slika 4.1. *Prikaz prosječnog učitavanja podataka s poslužitelja [12].*

Na slici 4.1. može se vidjeti koliko *NodeJS* okruženje ubrzava proces učitavanja velike količine podataka s poslužiteljske strane. Glavni potencijal ovog okruženja se postiže tek kada aplikacija radi s velikom količinom podataka i zahtjeva.

4.3. MongoDB

MongoDB je višeplatformska baza podataka i bazirana je na *NoSQL*-u koji nije relacijski i koristi nestrukturirane podatke. To znači da se koristi za rad s podacima u *JSON* obliku što omogućava stvaranje i implementaciju visoko skalabilne baze podataka koja je usmjerena na učinkovitost. *JSON* (engl. *Javascript Object Notation*) oblik dokumenta je lako čitljiv, a u njega se mogu spremiti strukturirane i nestrukturirane informacije. Omogućuje spremanje velike količine podataka i upiti se odrađuju brže od standardnog *SQL*-a [13].

4.4. Insomnia

Insomnia je besplatna višeplatformska aplikacija koja olakšava kreiranje *API* (engl. *Application Programming Interface*) zahtjeva koji se koriste za rad s poslužiteljem. Aplikacija je slična *Postman*-u, no lakše je kreirati i slati zahtjeve. U ovome radu se koristi za dodavanje, brisanje, izmjenu i čitanje podataka iz baze, za popunjavanje baze podataka i provjeru putanja koje se koriste na poslužiteljskoj strani. Time se može provjeriti kako napravljeni poslužitelj reagira na pojedine oblike zahtjeva i što se događa npr. u slučaju krivo neispravnog zahtjeva.

5. ZAHTJEVI PROJEKTA

Jedan od najboljih pristupa prije izrade programskog rješenja projekta je osmisliti i zapisati tražene funkcionalne i nefunkcionalne zahtjeve, korištenu arhitekturu aplikacije i shemu baze. Ovaj pristup izrade aplikacije je najsličniji tradicionalnom pristupu koji se sastoji od faze planiranja, dizajniranja, implementacije, testiranja i održavanja. Ovo poglavlje obuhvaća dio s planiranjem zahtjeva i dizajnom arhitekture.

5.1. Funkcionalni i nefunkcionalni zahtjevi

Prva faza prije implementacije je planiranje funkcionalnih i nefunkcionalnih zahtjeva. Pri definiranju funkcionalnih zahtjeva određuju se pojedine funkcionalnosti koje su usmjerene na zahtjeve korisnika, dok se nefunkcionalni zahtjevi odnose na izvođenje aplikacije i evaluaciju ispunjenja korisnikovih očekivanja [14].

Ova aplikacija ima dvije uloge korisnika, a to su *Korisnik* i *Administrator*. *Administrator* ima sve razine pristupa u aplikaciji. Uloga *Korisnika* je praćenje stanja, inventura, preuzimanja i ostale glavne mogućnosti pri radu sa skladištem, no nema mogućnosti upravljanja korisnicima aplikacije i drugim višim funkcionalnostima kao *Administrator*. U tablici 5.1. se mogu vidjeti sve funkcionalnosti aplikacije i prava pristupa *Korisnika*.

Tablica 5.1. *Prikaz svih funkcionalnosti aplikacije i ograničenja uloge Korisnik.*

Redni broj	Naziv funkcionalnosti	Prava pristupa uloge *Korisnik* [✓ / × / *posebno ograničenje*]
1	Prijava u sustav	✓
2	Pregled korisnika, dodavanje korisnika, izmjena korisnika, brisanje računa, promjena uloge	×
3	Pregled kategorija	✓
4	Dodavanje, izmjena ili brisanje kategorije	×
5	Pregled potkategorija	✓
6	Dodavanje, izmjena ili brisanje potkategorije	×
7	Pregled ambalaža	✓
8	Dodavanje, izmjena ili brisanje ambalaža	×
9	Pregled proizvoda	✓

10	Dodavanje, izmjena ili brisanje proizvoda	×
11	Pregled stanja skladišta	*Korisnik* može vidjeti samo stanje dodijeljenih skladišta
12	Dodavanje, izmjena ili brisanje stanja skladišta	×
13	Pregled unosa u skladište	*Korisnik* može vidjeti samo vlastite unose u tekućem mjesecu
14	Potvrda unosa u skladište	*Korisnik* može potvrditi samo vlastite unose
15	Pregled skladišta	*Korisnik* može vidjeti samo dodijeljena skladišta
16	Dodavanje, izmjena ili brisanje skladišta	×
17	Pregled gradova	*Korisnik* može vidjeti samo gradove u kojima se nalaze dodijeljena skladišta
18	Dodavanje, izmjena ili brisanje gradova	×
19	Pregled lokacija	*Korisnik* može vidjeti samo lokacije koje su povezane s gradom u kojem se nalaze dodijeljena skladišta
20	Dodavanje, izmjena ili brisanje lokacija	×
21	Pregled preuzimanja	*Korisnik* može vidjeti samo vlastita preuzimanja u tekućem mjesecu
22	Potvrda preuzimanja	*Korisnik* može potvrditi samo vlastita preuzimanja
23	Pregled inventura	*Korisnik* može vidjeti samo vlastite inventure u tekućem mjesecu
24	Potvrda inventura	*Korisnik* može potvrditi samo vlastite inventure

25	Pregled ili brisanje obavijesti iz dnevnika obavijesti	×
26	Pregled, dodavanje, izmjena ili brisanje postavki automatske obavijesti	×
27	Osvježenje automatskih obavijesti	×
28	Generiranje izvješća o unosima za zadani period	×
29	Generiranje izvješća o preuzimanjima za zadani period	×
30	Generiranje izvješća o inventurama za zadani period	×
31	Zahtjev za resetiranje lozinke	✓

Nakon zadanih funkcionalnih zahtjeva potrebno je definirati nefunkcionalne koji su, također veoma važni za korisnikovo zadovoljstvo kao i daljnji razvoj te proširenje aplikacije.

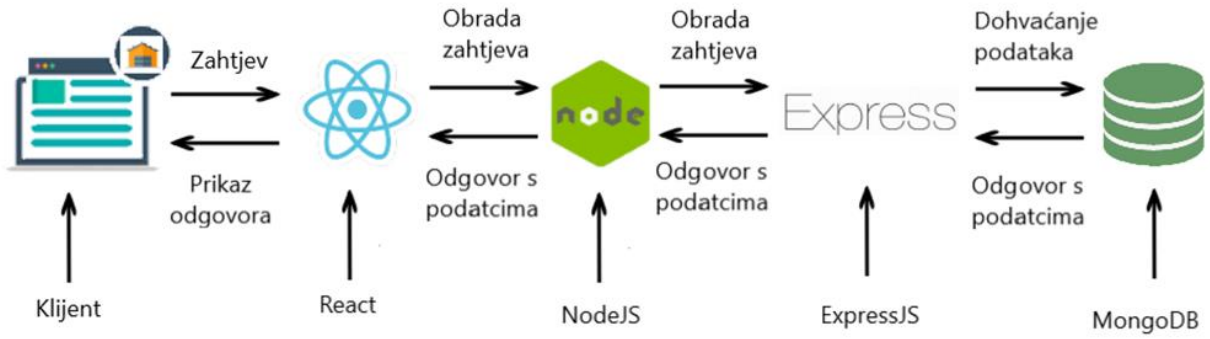
Nefunkcionalni zahtjevi koje aplikacija treba zadovoljiti:

- provjerava se ispravnost unesenih podataka u polja,
- provjeravaju se polja koja su obavezna za unos pri dodavanju ili izmjeni,
- izvještaj se može generirati u pdf obliku,
- u slučaju pogreške ili uspješne akcije s poslužiteljem ili na klijentskoj strani, prikazuje se odgovarajuća poruka,
- aplikacija se može proširiti u slučaju povećanja broja korisnika,
- stranica se mora učitavati brzo (učitavanje zaslona treba biti do četiri sekunde ovisno o količini podataka) [15],
- manje akcije u aplikaciji koje nisu resursno zahtjevne se moraju odrađivati brzo,
- aplikacija mora imati dodatan oblik sigurnosti poput pristupnog tokena i tokena za osvježenje,
- rute kojima se pristupa u aplikaciji moraju biti osigurane od neovlaštenog pristupa,
- dizajn mora biti intuitivan,
- za korištenje aplikacije potrebno je imati pristup internetu i internetskom pregledniku,
- aplikacija se može pokretati na bilo kojem računalu i mobitelu neovisno o specifikacijama.

Svi navedeni nefunkcionalni zahtjevi spadaju pod neku od glavnih kategorija poput izvedbe, skalabilnosti, prenosivosti, kompatibilnosti, pouzdanosti, dostupnosti, održivosti, sigurnosti, upotrebljivosti, upravljivosti, održavanja, integriteta podataka i kapaciteta aplikacije [16].

5.2. Arhitektura aplikacije

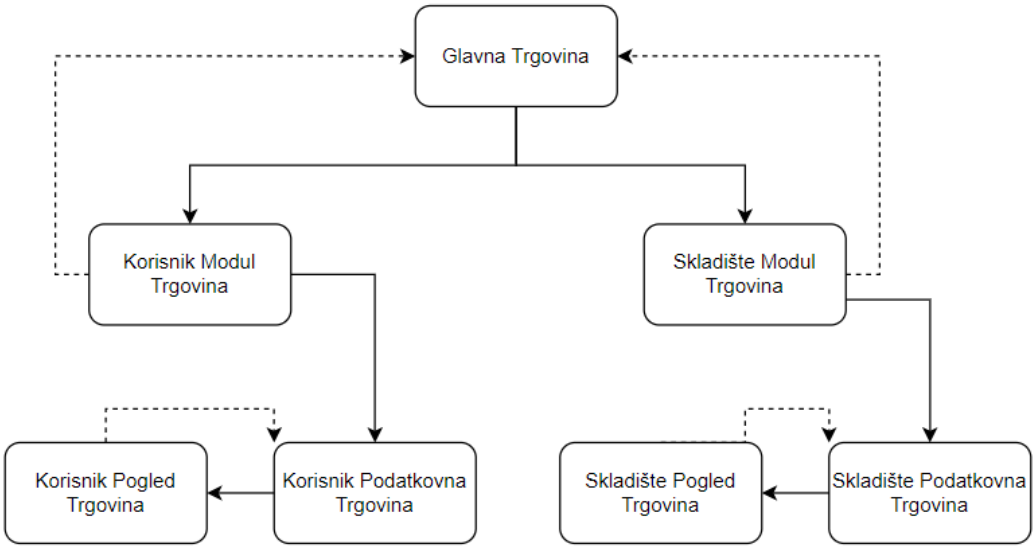
Arhitektura aplikacije opisuje glavne komponente, njihove odnose i interakciju. Pomaže pri određivanju poslovne strategije, atributa kvalitete, dinamike i dizajna okruženja aplikacije. Na slici 5.1. prikazan je tok rada aplikacije koji zapravo opisuje *MERN* arhitekturu. *MERN* predstavlja *MongoDB, ExpressJS, React* i *NodeJS*, što je zapravo troslojna arhitektura koja se sastoji od klijentskog dijela (*React*), poslužitelja (*NodeJS* s *ExpressJS*-om) i baze podataka (*MongoDB*) [17].



Slika 5.1. *Prikaz MERN arhitekture i toka aplikacije preko klijenta i poslužitelja [17].*

Ovaj oblik arhitekture je pogodan za dinamična korisnička sučelja koja koriste *JSON* tip podataka s poslužitelja.

Osim glavne arhitekture aplikacije, slikom 5.2. opisana je unutrašnja arhitektura *React*-a. Ovaj dio bazira se na korištenju *Mobx* biblioteke koja odvaja logiku i stanje komponenti, pomaže pri lakšem testiranju pojedinih *trgovina* (engl. *store*) te dodaje posrednike na akcije s UI dijelom. Primarna svrha ove biblioteke je razdvajanje na više *trgovina* gdje postoji *glavna trgovina* (engl. *Root Store*) koja se u obliku stabla dijeli na ostale *trgovine*. Ovom arhitekturom se jednostavno pristupa svim *trgovinama* iz *glavne trgovine,* koristi se ista instanca na svim mjestima i lakša je komunikacija među njima [18]. U ovoj aplikaciji se može pobliže objasniti na primjeru *Korisnik modul trgovine* koja prima instancu *glavne trgovine* i kreira instancu *Korisnik podatkovne trgovine*. *Korisnik podatkovna trgovina* ima pristup metodama za rad s poslužiteljem, dok *Korisnik pogled trgovina* kroz ubrizgavanje ovisnosti (engl. *Dependency Injection, DI*) prima instancu na podatkovnu trgovinu te s nje prima podatke za prikaz i šalje akcije koje korisnik napravi.



Slika 5.2. *Unutrašnja arhitektura React klijentskog dijela s Mobx bibliotekom.*

U prikazanom primjeru sa slike 5.2. se osim za *Korisnik Modul Trgovinu* prikazuje i arhitektura za *Skladište Modul Trgovinu* (engl. *Warehouse Store*). Ovaj princip se koristi i za sve ostale stranice u ovoj aplikaciji poput prikaza i rada s lokacijama, gradom, skladištima, stanjima, proizvodima, kategorijama, potkategorijama, ambalažama, preuzimanjima, obračunom i obavijestima. Ovim potpoglavljem je obuhvaćena druga faza razvoja prije implementacije aplikacije.

5.3. Shema baze podataka

Treću fazu prije implementacije obuhvaća dizajn baze podataka koji je vrlo važan kako bi se olakšalo spremanje i dohvaćanje podataka. Upravo iz tog razloga je osmišljena i na slici 5.3. prikazana shema baze podataka.

Slika 5.3. *Shema baze podataka aplikacije za vođenje skladišta.*

6. PROGRAMSKO RJEŠENJE

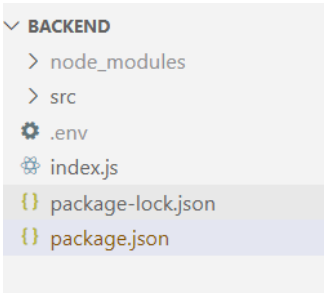
Nakon definiranja zahtjeva, arhitekture i sheme baze podataka prelazi se na implementaciju programskog rješenja. U programskom rješenju najprije je kreiran poslužitelj u *NodeJS*-u i povezan s *MongoDB* bazom. Nakon uspješne implementacije, postavljen je na *Heroku* platformu te je dobiven *url* preko kojeg se pristupa poslužitelju. Drugi dio programskog rješenja obuhvaća kreiranje *ReactJS* aplikacije koja je postavljena na *Vercel* platformu što omogućava pristup aplikaciji putem internet preglednika.

6.1. Poslužitelj u NodeJS-u s MongoDB bazom

Prvi korak je postavljanje *NodeJS* okruženja. Kreirana je mapa *backend* u kojoj se pomoću *PowerShell* naredbene linije, putem naredbe *npm init* postavlja projekt. Ovom naredbom se u mapi stvara datoteka *package.json* koja sadrži poveznice na sve dodatne pakete koji pomažu pri implementaciji poslužitelja. Nakon toga je instaliran *Express* poslužiteljski okvir koji pruža potrebne značajke za razvoj aplikacije i preko njega se izvode upiti na poslužitelj.

Na slici 6.1. prikazana je struktura glavne mape poslužitelja i osim navedene dvije datoteke ona još sadrži:

- *node_modules* – mapa u kojoj se spremaju paketi i gotove funkcionalnosti potrebne za aplikaciju,
- *src* – mapa koja se sastoji od dodatnih podmapa koje se ponašaju kao putanje na poslužitelju,
- *.env* – datoteka koja sadrži stvari koje ne bi trebale biti vidljive korisniku već se iz pozadine učitavaju na poslužitelj i potrebne su za pravilan rad aplikacije, tu se ubrajaju znakovni niz (engl. *string*) za tokene, glavna administratorska lozinka, email, lozinka za email poslužitelj i slično,
- *index.js* – datoteka u kojoj se nalazi skripta za pokretanje poslužitelja.



Slika 6.1. *Struktura glavne mape poslužitelja.*

Slika 6.2. prikazuje glavnu skriptu za pokretanje poslužitelja u kojoj se na vrhu nalaze dohvaćene vanjske datoteke spremljene u varijable. Najvažnija od njih je *app* varijabla jer ona sadrži kreiranu instancu za pokretanje *Express* aplikacije.

Drugi dio skripte obuhvaća:

- *cors* – politika koja omogućava nesmetano dohvaćanje podataka bez provjere izvora,
- *express.json()* – metoda koja omogućava slanje i primanje *JSON* oblika podataka,
- *mongoose.connect()* – metoda za povezivanje s bazom podataka.

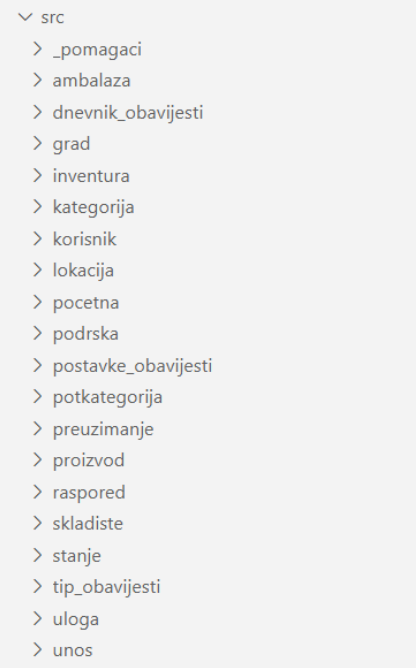
Nakon što se poslužitelj uspješno poveže na *MongoDB* bazu, učitavaju se moduli pomoću metode *ucitajModule()* i rute metodom *ucitajRute()* kojoj se kao parametar predaje *Express* instanca. Aplikacija se pokreće na ulazu (engl. *port*) koji se dohvaća iz *.env* datoteke.

index.js

```
1   const express = require("express");
2   const app = express();
3   const cors = require("cors");
4   const mongoose = require('mongoose');
5   const logger = require("morgan");
6   const { ucitajModule, ucitajRute } = require('./src/podrska/index');
7
8   app.use(logger("dev"));
9   app.use(cors());
10  app.use(express.json());
11
12  if (process.env.NODE_ENV !== 'produkcija') {
13    require('dotenv').config();
14  }
15
16  const mongoDBurl = process.env.MONGO_DB_URL;
17  mongoose.connect(mongoDBurl, { useNewUrlParser: true, useFindAndModify: false, useUnifiedTopology: true }).then(() => {
18    ucitajModule();
19    ucitajRute(app);
20
21    app.get('/', (req, res) => {
22      res.send('Upravljanje Skladištima API..');
23    });
24  });
25
26  const port = process.env.PORT;
27  app.listen(port);
28
29  module.exports = app;
```

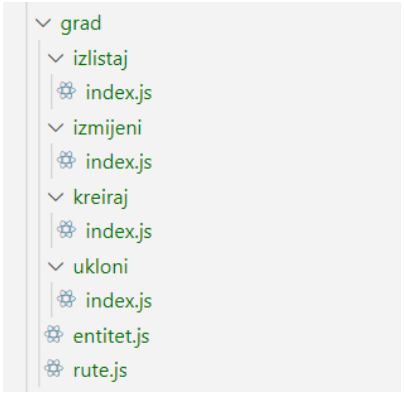
Slika 6.2. *Glavna skripta za pokretanje poslužitelja.*

Na slici 6.3. prikazane su mape čiji naziv ujedno predstavlja i pojedinu rutu poslužitelja osim mape *_pomagaci* koja sadrži pomoćne datoteke poslužitelja. Svaka od ovih ruta je napisana prema sličnoj strukturi koja je dodatno pojašnjena na primjeru rute *Grad*.



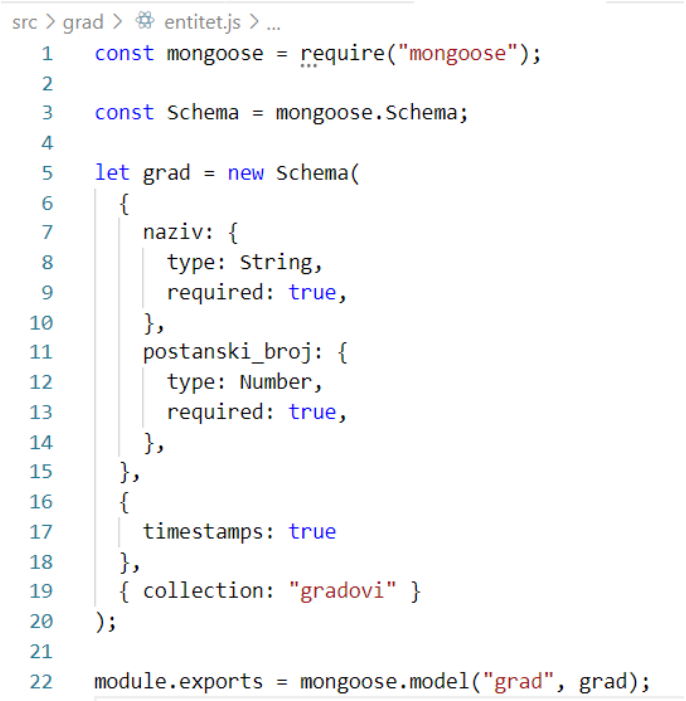
Slika 6.3. *Popis mapa (ruta) na poslužitelju.*

Na slici 6.4. jer prikazan popis mapa koje se ponašaju kao rute. Prema ovoj strukturi se dobiva kostur *url-a* rute *glavni_url/naziv_mape/naziv_podmape* pomoću kojeg se pristupa rutama poslužitelja. Korištenjem *https://localhost:3000/grad/izlistaj* primjera *url-a* dohvaća se lista gradova iz baze. Datoteka *rute.js* sadrži popis svih ruta s kojima se može raditi u ovoj mapi.



Slika 6.4. *Primjer strukture mapa za rutu Grad.*

Na slici 6.5. prikazan je sadržaj datoteke *entitet.js* koji predstavlja shemu entiteta *Grad*. Prema ovom primjeru su napravljeni i ostali entiteti.



Slika 6.5. *Prikaz entiteta Grad.*

Entitet *Grad* se sastoji od četiri atributa:

- *naziv* – naziv grada (znakovni niz, obavezan),
- *postanski_broj* – poštanski broj grada (broj, obavezan),
- *timestamps* – automatsko kreiranje i ažuriranje datuma kreiranja i izmjene,
- *collection* – predstavlja naziv kolekcije koja je vidljiva u bazi nakon prvog unosa i kojoj se pristupa za dohvaćanje gradova.

Na slici 6.6. se nalazi kod u kojem se pozivaju datoteke *autentifikacijaJWT*, *autentifikacijaAdmin* i *provjeraID* koje se ponašaju kao posrednici (engl. *middleware*). Posrednici služe za filtriranje i odbijanje zahtjeva u slučaju da nisu ispunjeni potrebni uvjeti. Prva datoteka *autentifikacijaJWT* služi kao posrednik za provjeru ispravnosti poslanog tokena s klijentske strane. Taj token se naziva *JSON Web Token* i on predstavlja standard za šifriranje pristupa ruti, što znači da joj može pristupiti samo prethodno autorizirani korisnik kojemu se pri prijavi u aplikaciju generirao token. Druga datoteka je *autentifikacijaAdmin* koja sadrži pretragu korisnika prema predanom *ID*-ju. Ako korisnik nije pronađen ili njegova uloga nije *Administrator*, zahtjev se odbija, a u suprotnom nastavlja na idućeg posrednika. Datoteka *provjeraID* služi za provjeru *ID*-ja koji se predaje kao parametar idućoj ruti. Kako bi *ID* bio ispravan, mora biti znakovni niz duljine dvadeset i četiri znaka. Ako su svi prethodni koraci uspješni, zahtjev dolazi do datoteke *index.js* koja se dohvaća pomoću *require* i predaje kao varijabla (*izlistaj*, *dodaj*, *ukloni*, *izmijeni*).

```
const express = require("express");
const router = express.Router();

const autentifikacijaJWT = require("../_pomagaci/autentifikacijaJWT");
const autentifikacijaAdmin = require("../_pomagaci/autentifikacijaAdmin");
const provjeraID = require("../_pomagaci/provjeraID");

const izlistaj = require("./izlistaj/index");
const dodaj = require("./dodaj/index");
const ukloni = require("./ukloni/index");
const izmijeni = require("./izmijeni/index");

router.post("/", autentifikacijaJWT, izlistaj);
router.post("/dodaj", autentifikacijaJWT, autentifikacijaAdmin, dodaj);
router.delete("/ukloni/:id", provjeraID, autentifikacijaJWT, autentifikacijaAdmin, ukloni);
router.patch("/:id", provjeraID, autentifikacijaJWT, autentifikacijaAdmin, izmijeni);

module.exports = router;
```

Slika 6.6. *Izgled datoteke koja sadrži rute i pristupne posrednike.*

Primjer funkcije za dohvaćanje svih gradova prikazana je na slici 6.7. Najprije se dohvaća shema *Grada* i sprema u istoimenu varijablu *Grad* te se metodom *find()* dohvaća lista gradova. Gradove je moguće filtrirati predajom atributa koji se želi filtrirati unutar vitičastih zagrada. Za sortiranje podataka iz liste koristi se metoda *sort()* koja kao parametar prima naziv atributa po kojem se sortiraju podaci. Nakon što su podaci sortirani iz svakog grada s liste se izvlače samo one informacije koje su potrebne na klijentskoj strani pomoću *JavaScript* metode *map* koja prolazi kroz svaki element liste i pretvara ga u novi podatak koji se vrati izjavom *return*. Nakon uspješnog mapiranja podataka, odgovor se šalje pomoću linije *res.status(200).json({gradovi})*. Metoda *status(200)* vraća statusni kod koji opisuje status poziva na poslužitelj. *Status(200)* predstavlja uspješan zahtjev, a uz njega postoje još drugi kodovi poput *status(500)* koji označava pogrešku u radu s bazom. Na kraju odgovora se nalazi *JSON* podatak koji predstavlja listu gradova. Kako bi se osigurao pravilan rad bez rušenja poslužitelja, kod unutar tijela funkcije je omotan u *pokušaj uhvati* (engl. *try catch*) blok koji služi za hvatanje iznimki. Iznimke su pogreške u radu koje mogu dovesti do nepravilnog i nesigurnog izvođenja programa i zbog toga je s njima potrebno pravilno rukovati. U slučaju pogreške vraća se poruka *Dogodila se pogreška, molimo kontaktirajte administratora!*.

Slika 6.7. *Primjer funkcije za dohvaćanje liste svih gradova.*

Osim osnovnih radnji za pregled, dodavanje, izmjenu, potvrdu i brisanje podataka, poslužitelj omogućava praćenje i rad s automatskim obavijestima putem kojih se korisniku šalju izvješća i podsjetnici o stanju proizvoda na skladištima. Automatske obavijesti se izvode pomoću gotovog paketa za raspoređivanje zadataka *node-cron*. Ovaj raspoređivač omogućava kreiranje poslova koji se odrađuju u zadanom vremenu, npr. svaki dan u deset sati, a primjer automatskog zadatka koji se obavlja svaku minutu bez prestanka prikazan je na slici 6.8. Prvi parametar označava vrijeme obavljanja zadatka, a drugi parametar zadatak koji se obavlja. Zvjezdice označavaju stalno vrijeme izvođenja, a moguće ih je zamijeniti brojevima. Također je moguće vidjeti da je predano pet zvjezdica gdje zvjezdice redom označavaju: minuta, sat, dan u mjesecu, mjesec, dan u tjednu. Prema toj strukturi implementirano je kreiranje automatske obavijesti ovisno o postavkama.

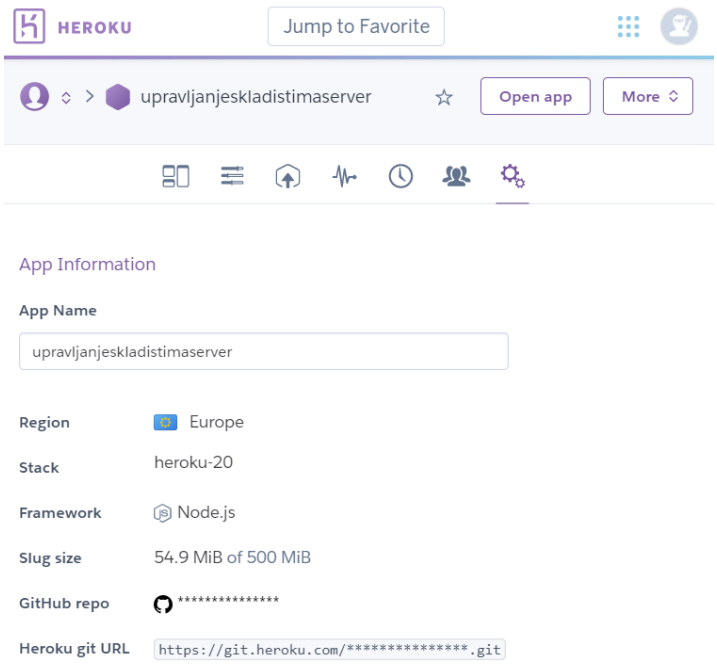
```
var cron = require('node-cron');

cron.schedule('* * * * *', () => {
  console.log("Izvrsavanje zadatka svaku minutu");
})
```

Slika 6.8. *Primjer automatskog zadatka pomoću node-cron paketa.*

Prilikom kreiranja i slanja podsjetnika koristi se paket *nodemailer* za slanje email-a i *jsPDF* paket za kreiranje *PDF* datoteke. Za slanje email-a *nodemailer-om,* potrebno je definirati transporter (poslužitelj), ulaz i podatke za slanje, a zatim ostale informacije o email-u poput poštanske adrese pošiljatelja, primatelja, naslova, podataka i priloga [19]. Ako postoje proizvodi koji nedostaju na skladištu, pri slanju poruke se automatski dodaje *PDF* prilog s podatcima o proizvodima koji nedostaju. Ovaj prilog se kreira pomoću *jsPDF* paketa koji je detaljnije opisan u klijentskom dijelu.

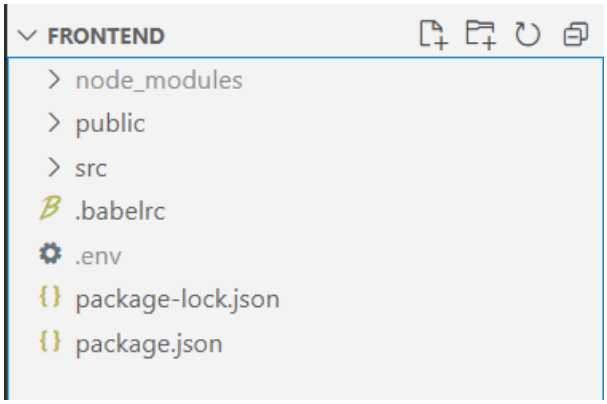
Poslužitelj je postavljen (engl. *deploy*) na *Heroku* platformu povezivanjem s *GitHub* računom i moguće mu je pristupiti s bilo kojeg mjesta u bilo koje vrijeme (Slika 6.9.). *Heroku* platforma pruža usluge besplatne izgradnje, pokretanja i upravljanja aplikacijama u oblaku.



Slika 6.9. *Prikaz postavki poslužitelja na Heroku platformi.*

6.2. Klijentska strana u ReactJS-u

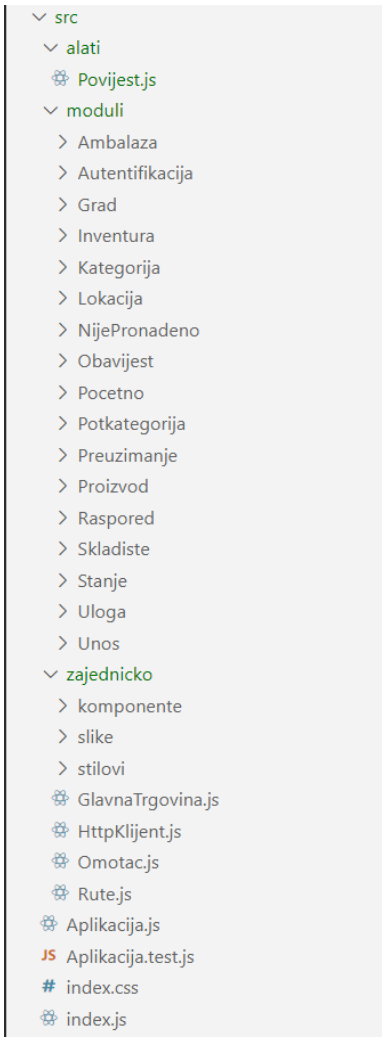
Nakon postavljanja poslužitelja potrebno je implementirati klijentsku stranu koja obuhvaća kreiranje aplikacije u *ReactJS*-u i povezivanje s poslužiteljem. Postupak postavljanja je sličan postavljanju poslužitelja pomoću *PowerShell* naredbene linije, unosom naredbe *npx create-react-app frontend* kreira se *React* aplikacija u mapi *frontend*. Struktura mape *frontend* je prikazana na slici 6.10. Za razliku od poslužitelja, *React* aplikacija sadrži *public* mapu s datotekama za pokretanje početne stranice. Dodana je *.babelrc* datoteka koja sadrži postavke potrebne za rad s *Mobx-om* te *.env* datoteka koja sadrži glavnu administratorsku lozinku i *url* na poslužitelj.



Slika 6.10. *Struktura mapa na klijentskoj strani.*

Glavna struktura mapa prikazana je na slici 6.11. *Src* mapa sadrži sve potrebne komponente za rad ove aplikacije, a to su mape:

- *alati* – sadrži datoteku *Povijest* koja služi za pamćenje korisnikovih kretnji po rutama kako bi se u svakom trenutku mogao vratiti na prethodnu rutu,
- *moduli* – sadrži mape koje predstavljaju pojedinu rutu zapisanu u datoteci *Omotac.js,*
- *zajednicko* – sadrži dijeljene komponente, slike i stilove koji se koriste kroz čitavu aplikaciju kako bi se smanjila potreba za redundancijom prateći pravilo *React*-a, a to je *recikliranje komponenti*. Ova mapa sadrži još klasu *GlavnaTrgovina* opisanu u potpoglavlju 5.2., datoteku *HttpKlijent* koja služi za rad s poslužiteljem, datoteku *Omotac* u kojoj se definiraju nazivi *ruta* i *trgovina* te datoteka *Rute* u kojoj se nalazi popis svih ruta kojima se može pristupiti.

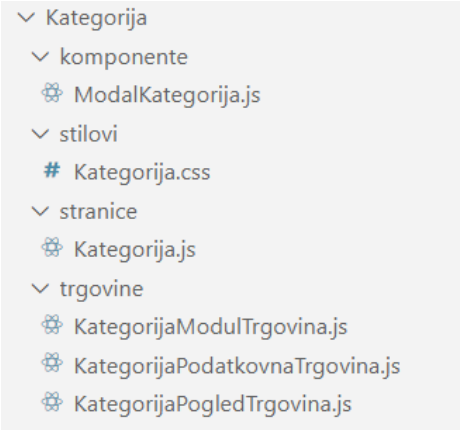


Slika 6.11. *Struktura mapa i datoteka na klijentskoj strani.*

U mapi *src* se još nalaze datoteke *Aplikacija.js* i *index.js* koje su potrebne za pokretanje i rad aplikacije te dodatna datoteka za stiliziranje aplikacije. Na primjeru modula *Kategorija* i strukture mape prikazane na slici 6.12. objašnjena je struktura koja se koristi u svim mapama koje se nalaze u mapi *moduli* jer općenito svaki od ovih modula predstavlja jednu stranicu u aplikaciji osim mape *Autentifikacija* koja se sastoji od stranice za prijavu, korisnike i resetiranje lozinke.

Stuktura mape *Kategorija* je sljedeća:

- *komponente* – sadrži dodatne komponente koje se koriste na stranici *Kategorija* pa se tako *ModalKategorija* može ponovno iskoristiti,
- *stilovi* – sadrži korištene stilove na stranici,
- *stranice* – ova mapa najčešće ima jednu klasu istog imena kao i njena roditeljska mapa,
- *trgovine* – mapa koja u sebi sadržava osnovnu *Mobx* strukturu koja se koristi za *MVC* arhitekturu pojedine stranice, a koja je detaljno opisana u potpoglavlju 5.2.



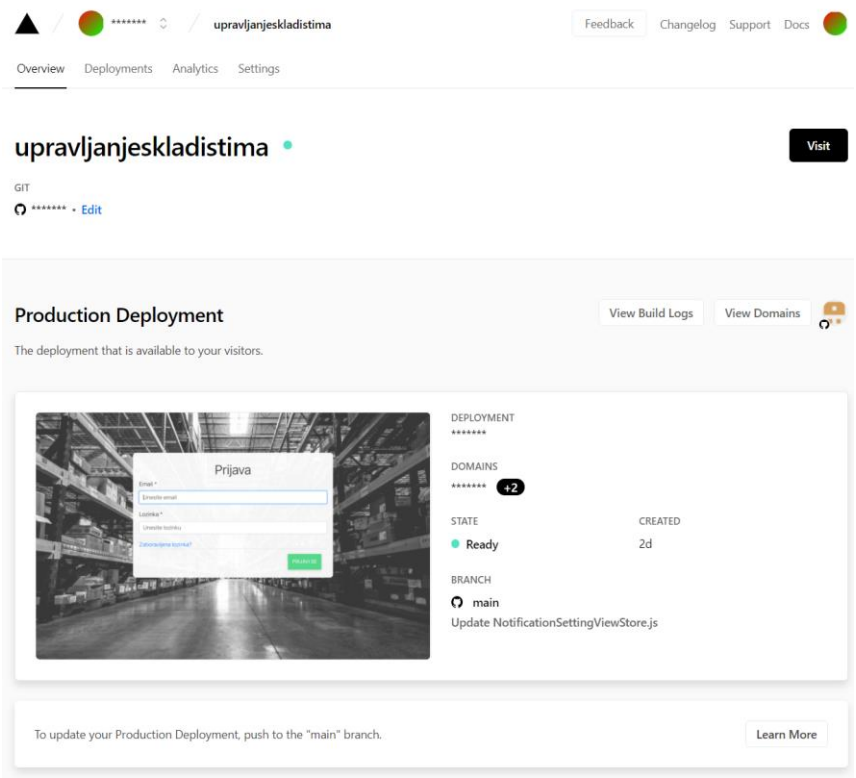
Slika 6.12. *Prikaz strukture mapa unutar mape Kategorija.*

Nakon završetka implementacije stranica potrebno ih je povezati i omogućiti rad s podatcima iz baze. Taj dio je riješen pomoću *HttpKlijent* datoteke koja sadržava pristupnu točku i metode za rad s bazom. Slika 6.13. prikazuje primjer rute za kreiranje novog podatka pri čemu je ruta generalizirana pa se tako pri pozivu ove funkcije predaje tijelo zahtjeva s podatcima koji se žele kreirati i *apiUrl* s rutom na kojoj se želi kreirati ovaj podatak.

```
kreiraj = async (tijeloZahtjeva) => {
    const postavke = {
        method: "POST",
        headers: this.zaglavljeZahtjeva,
        tijeloZahtjeva
    }
    const zahtjev = new Request(this.apiUrl + "/kreiraj", postavke);
    let odgovor = await (fetch(zahtjev));
    let podatci = await odgovor.json();
    return podatci;
}
```

Slika 6.13. *Zahtjev za kreiranje novog podatka u bazi.*

Nakon povezivanja s bazom potrebno je postaviti *React* aplikaciju na neku od dostupnih platformi. Za platformu je odabrana stranica *Vercel* koja pruža besplatnu uslugu pokretanja i izvođenja aplikacije na oblaku. Na slici 6.14. prikazan je glavni zaslon nakon prijave u platformu. Aplikacija se vrlo jednostavno postavlja povezivanjem na *GitHub* račun prilikom čega se sinkroniziraju podatci i nakon toga osvježava dostupna aplikacija.

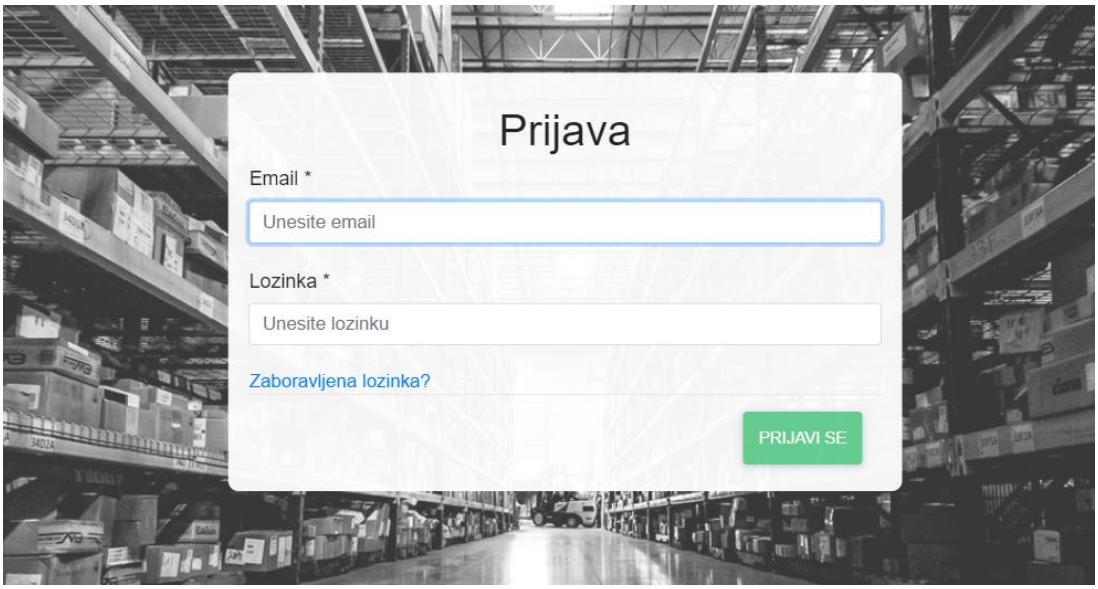


Slika 6.14. *Početni zaslon Vercel platforme za postavljanje aplikacije na oblak.*

Velika prednost ove platforme je automatsko podešavanje postavki aplikacije zbog koje nije potrebno osvježavati promjene već se ta promjena automatski učita s *GitHub-a* i implementira. Osim toga ova stranica nudi besplatnu analizu korištenja stranice te izbor domena putem kojih se može pristupiti aplikaciji.

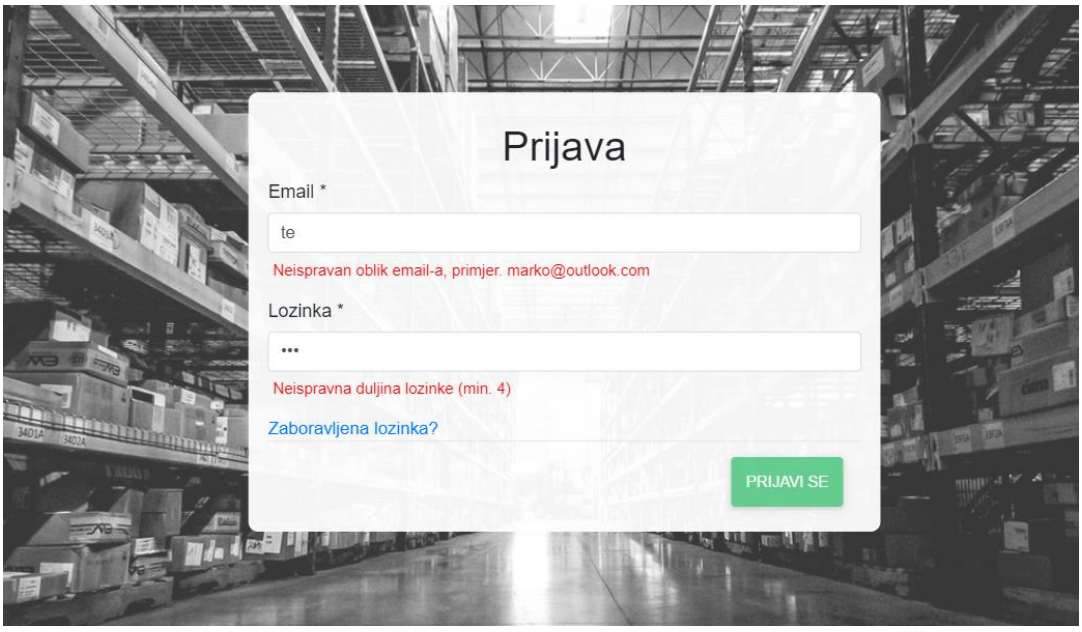
6.2.1. Prijava u sustav

Pri pokretanju aplikacije u internet pregledniku, ako korisnik još nije prijavljen, otvara se početna stranica za prijavu prikazana na slici 6.15. Nije moguće prijaviti se u sustav sve dok korisnik ne unese sve podatke u ispravnom formatu.



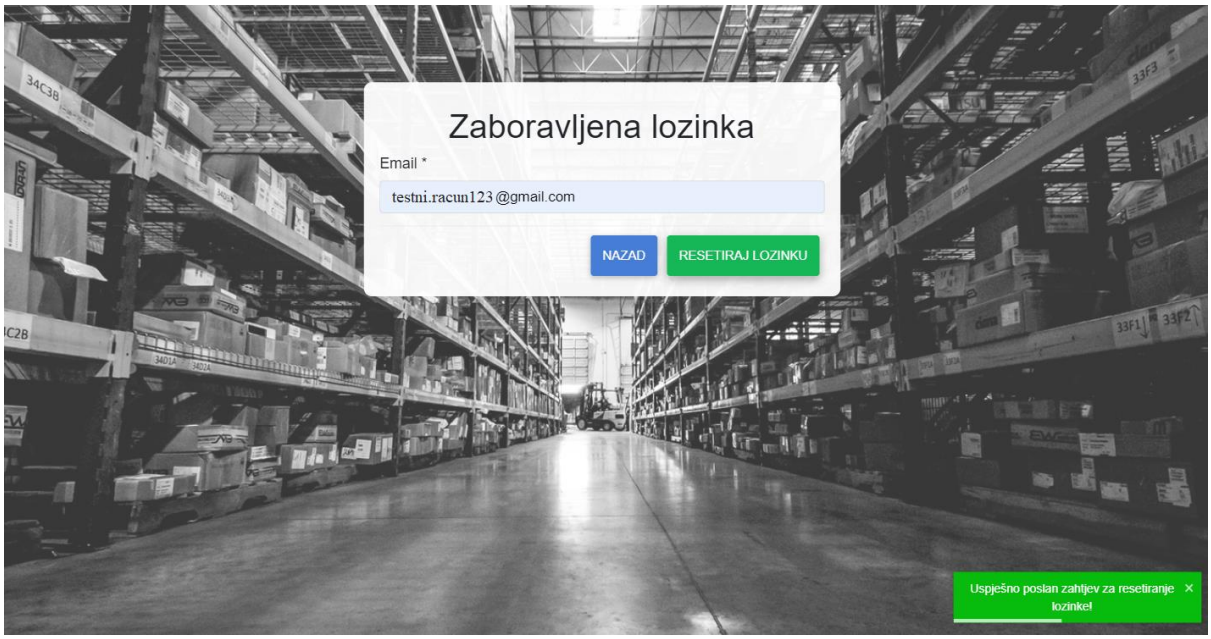
Slika 6.15. *Početni zaslon za prijavu korisnika u sustav.*

Slika 6.16. prikazuje poruke koje se prikazuju korisniku u slučaju neispravnog unosa podataka. U slučaju neispravne prijave prikazuje se poruka *Neispravni korisnički podatci!*.



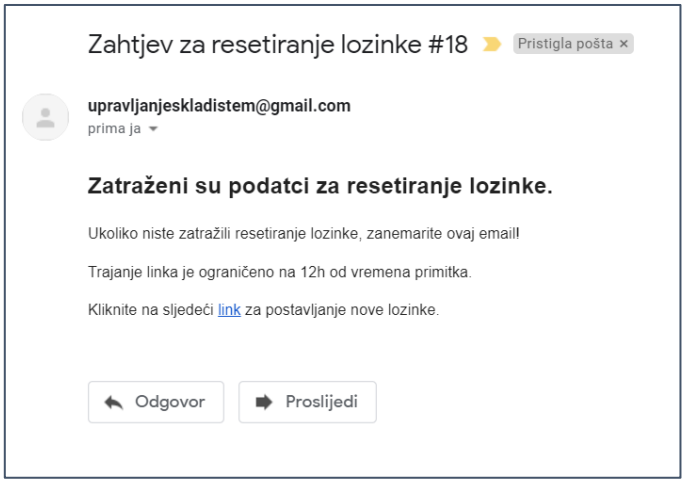
Slika 6.16. *Pomoćne poruke u slučaju neispravnog unosa vrijednosti u formu.*

Na slici 6.17. prikazan je zaslon za resetiranje lozinke. U slučaju da je korisnik zaboravio svoju lozinku, moguće ju je izmijeniti unosom email-a na stranici koja se otvara klikom na gumb *Zaboravljena lozinka?*. Nakon podnošenja zahtjeva ispisuje se povratna informacija s poslužitelja.



Slika 6.17. *Zaslon za resetiranje lozinke i povratna informacija s poslužitelja.*

Na slici 6.18. prikazan je email koji se dobiva nakon uspješno podnesenog zahtjeva za resetiranje lozinke. Klikom na dobiveni link otvara se stranica za resetiranje lozinke u kojoj se upisuje nova lozinka. Link je jednokratan i ograničen na dvanaest sati od vremena primitka. Provjera valjanosti linka se radi svakih pet minuta putem automatskog zadatka koji provjerava sve zahtjeve za resetiranje lozinke.

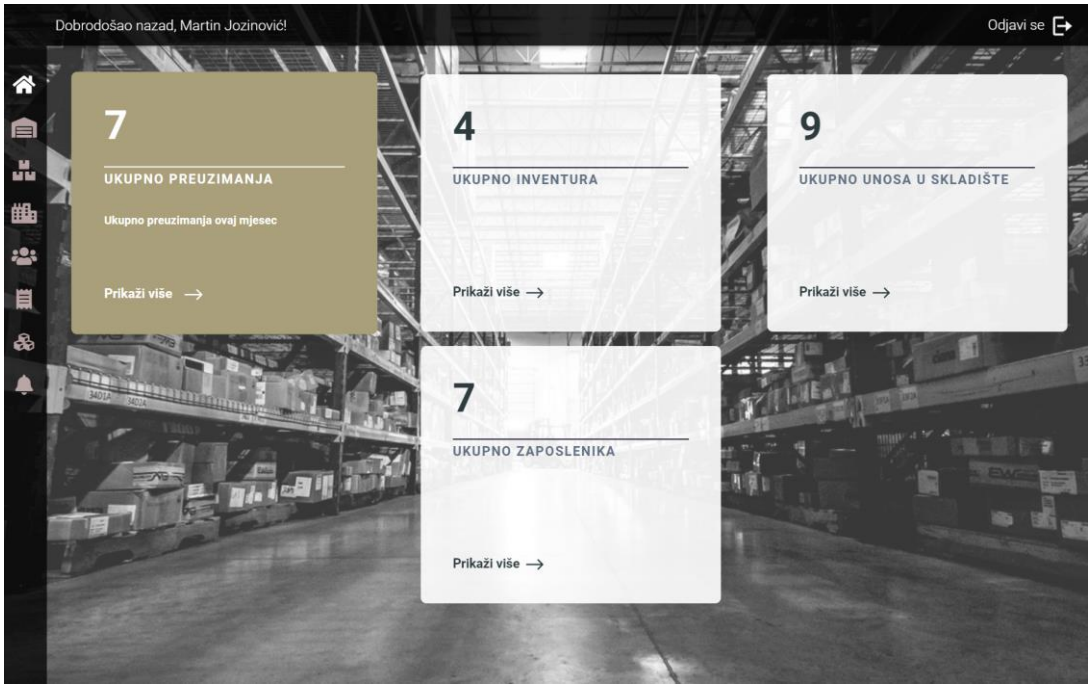


Slika 6.18. *Izgled email-a koji sadrži link za resetiranje lozinke.*

6.2.2. Početni zaslon

Nakon uspješne prijave u sustav otvara se početni zaslon s osnovnim informacijama o skladištu. Ove informacije ovise o ulozi prijavljenog korisnika. *Administratoru* se prikazuju

podatci o ukupnom broju preuzimanja, inventura i unosa u svim skladištima za tekući mjesec te ukupan broj ljudi koji se nalaze u sustavu i koji sudjeluju u vođenju skladišta. *Korisniku* se prikazuje samo ukupan broj njegovih preuzimanja, inventura i unosa u tekućem mjesecu. Na slici 6.19. prikazan je početni zaslon za *Administratora* i navigacijska traka je postavljena vertikalno lijevo.

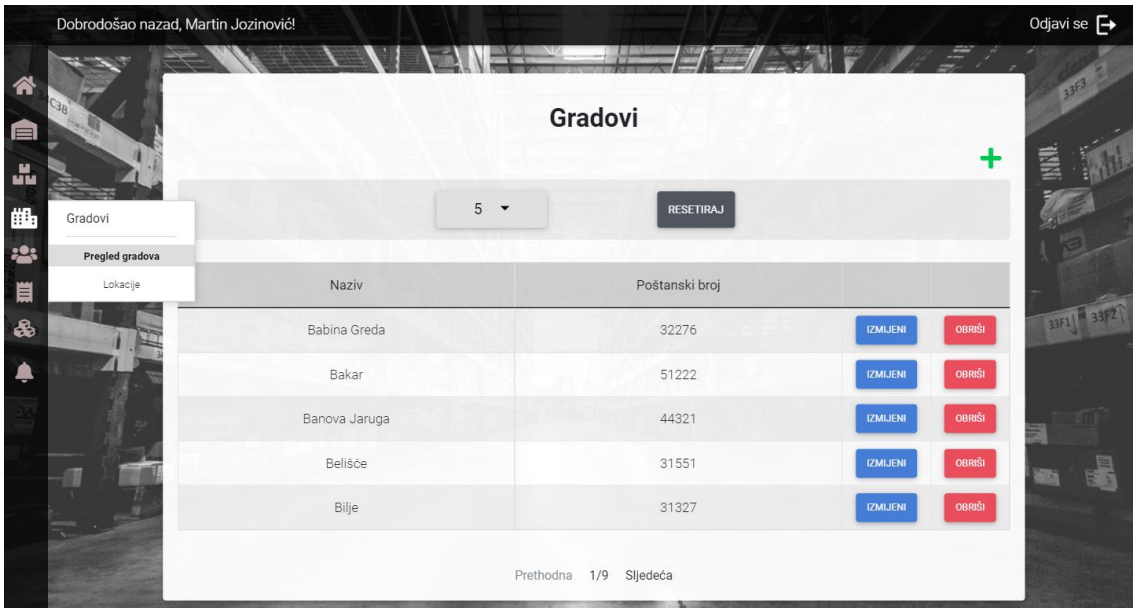


Slika 6.19. *Početni zaslon aplikacije za korisnika s ulogom Administrator.*

6.2.3. Zasloni za prikaz i rad s podatcima

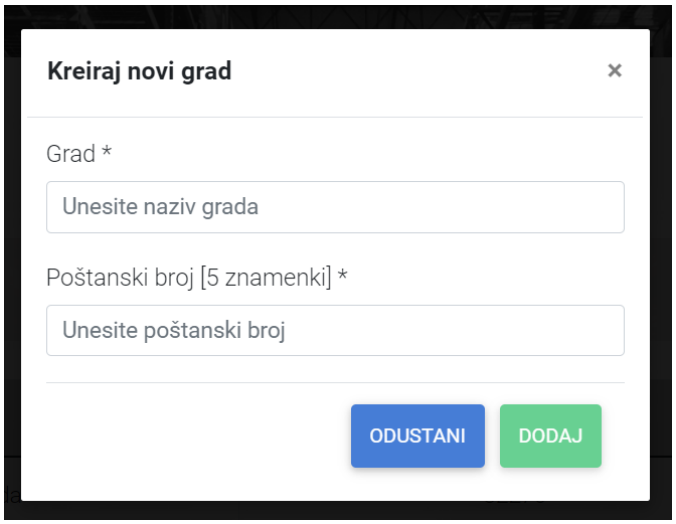
U aplikaciji se razlikuju jednostavne i složene tablice za prikaz podataka. Jednostavne tablice prikazuju podatke koji u sebi sadrže samo jednu razinu složenosti poput podataka iz entiteta *Grad* ili *Lokacija*. Ovi entiteti sadrže osnovne podatke poput naziva, ulice ili stranog ključa na drugu tablicu. Primjer entiteta *Lokacija* sadrži atribute *ulica* i *ID grada* preko kojega se mogu dohvatiti atributi *naziv* i *poštanski broj* iz povezanog entiteta *Grad*. Složeni podatci su oni podatci koji imaju veću dubinu pa se na primjeru entiteta *Preuzimanje* može vidjeti da *Preuzimanje* sadrži atribute *ID skladišta, ID lokacije, datum kreiranja, ID proizvoda, staro stanje, preuzeta količina i ID korisnika.* Preko *ID Lokacije* se može dohvatiti *ulica*, ali i *grad* koji je povezan s tom lokacijom, a pomoću *ID proizvoda,* podatci o proizvodu poput *naziva, kategorije, potkategorije, ambalaže.* Time je dobivena veća složenost dohvaćenih podataka. Na slici 6.20 prikazana je stranica *Gradovi* s jednostavnom tablicom koja prikazuje popis svih gradova. Na navigacijskoj traci prelaskom miša preko ikone *Gradovi* otvara se padajući izbornik za navigaciju na povezane stranice *Pregled*

gradova i *Lokacije*. Iznad tablice se nalazi izbornik za odabir količine podataka po stranici. Izbornik omogućava pregled pet, deset ili petnaest podataka po stranici. Osim toga podatke je moguće dodavati klikom na zeleni plus znak i izmjenjivati ili brisati klikom na gumbe *Izmijeni* i *Obriši*.



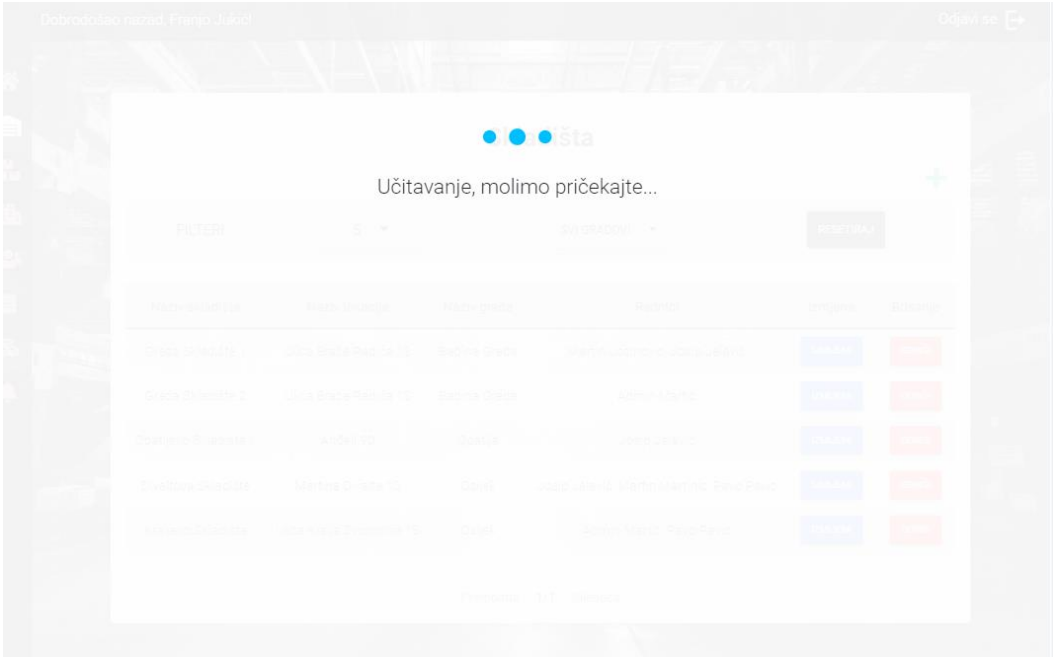
Slika 6.20. *Prikaz zaslona aplikacije za pregled gradova.*

Prilikom klika na plus znak otvara se *modal* s formom za dodavanje novog grada prikazan na slici 6.21. Svaki *Grad* ima svoj *naziv* i *poštanski broj* koji je duljine pet znamenki. Modali u aplikaciji imaju provjere podataka pri upisu kao npr. provjera duljine znakovnog niza, broj znamenki, odabir vrijednosti iz izbornika i slično. Osim poruka i provjera pri upisu, onemogućeno je brisanje podataka koji su povezani s drugim podatcima, npr. nije moguće obrisati grad s kojim su povezane lokacije.



Slika 6.21. *Prikaz modal forme za unos novog grada.*

Kako bi se informiralo korisnika o događanjima u pozadini aplikacije, prikazuje se ekran za učitavanje (Slika 6.22.).

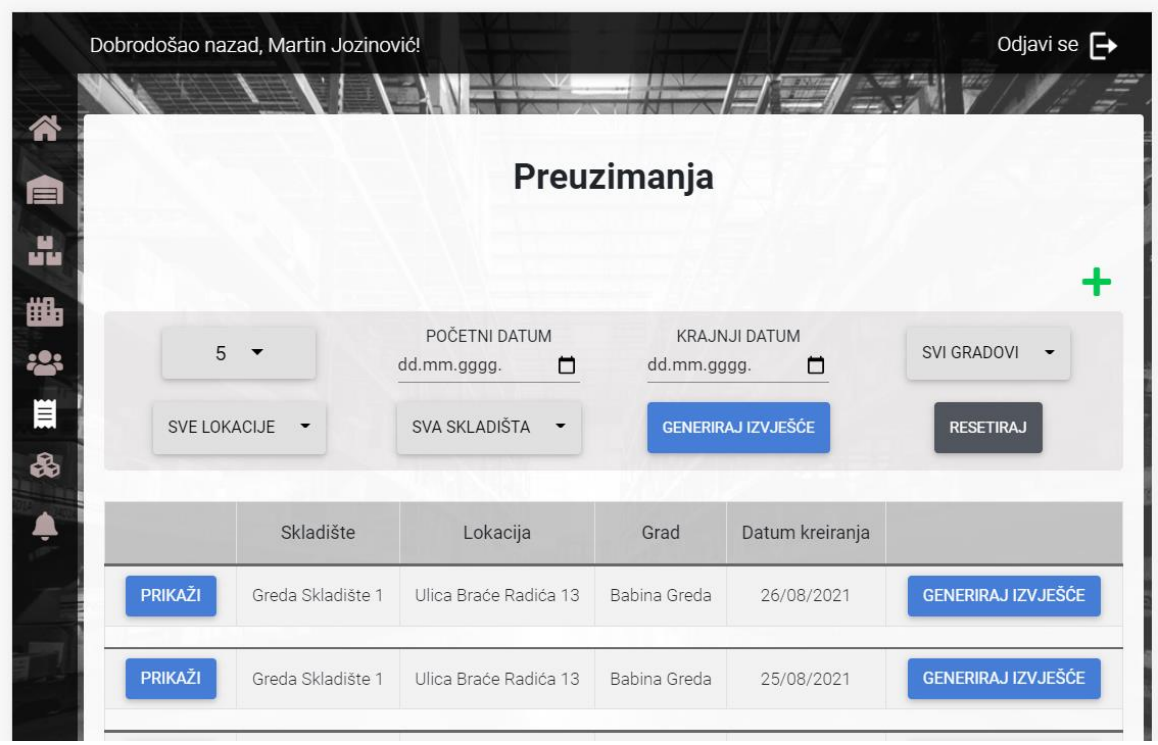


Slika 6.22. *Prikaz zaslona za učitavanje.*

Za kreirane gradove je moguće dodavati lokacije. U svakom gradu može postojati više lokacija, ali je onemogućeno dodavanje lokacije s istim nazivom u istom gradu. Zaslon s lokacijama omogućava i filtriranje lokacija prema odabranom gradu, kako bi se jasnije vidjele lokacije za dodavanje skladišta. Osim tablica *Grad* i *Lokacija*, aplikacija u navigacijskoj traci sadrži pristup drugim jednostavnim tablicama poput:

1. *skladište* – sadrži podatke o nazivu skladišta, lokaciji, gradu i radnicima koji su povezani s tim skladištem. U svako skladište se postavljaju na stanje proizvodi koji se mogu dodavati tijekom rada te radnici koji imaju pristup podatcima iz skladišta. Moguće je dodatno filtrirati podatke u tablici prema odabranom gradu,
2. *proizvod* – sadrži podatke o proizvodu poput kategorije, potkategorije i ambalaže. Također, moguće je filtrirati podatke prema ovim stupcima,
3. *kategorija* – tablica koja se koristi za entitet *Proizvod* i sadrži naziv kategorije proizvoda. Kategorija može biti hrana, piće, promotivni materijal i slično,
4. *potkategorija* – naziv potkategorije proizvoda. Primjeri potkategorija za piće su topli napitci, hladni napitci i pivo,
5. *ambalaža* – oblik ambalaže koji opisuje proizvod. Razlikuju se ambalaže poput boca različitih volumena, majice različitih veličina, vrećice od sokova,
6. *korisnik* – tablica sa svim korisnicima koji su registrirani u ovoj aplikaciji. *Administrator* ne može obrisati samog sebe već može mijenjati sve podatke o drugima osim lozinke. Stranica s ovom tablicom sadrži filtriranje prema ulozi korisnika,
7. *dnevnik obavijesti* i *postavke obavijesti* – sadrži podatke o automatskim obavijestima koje se šalju korisniku i rad stranice je detaljnije objašnjen u potpoglavlju 6.2.4.

Prikaz podataka u složenoj tablici je objašnjen u nastavku na primjeru stranice *Preuzimanje* koja ja prikazana na slici 6.23. Ova stranica služi za preuzimanje i evidenciju preuzetih proizvoda sa skladišta. Preuzimanje se kreira klikom na zeleni plus znak i unosom skladišta, proizvoda i željene količine. Uz željenu količinu se prikazuje i trenutna količina proizvoda na skladištu. Korisniku se pri kreiranju preuzimanja prikazuju samo oni proizvodi koji postoje u skladištu. Zbog sigurnosti je dodana dvostruka potvrda nakon koje se skida željena količina proizvoda sa stanja. Stranica sadrži opciju *Generiraj izvješće* koja korisniku daje uvid u informacije o aktivnostima vezanima za preuzimanja. S ovom opcijom je moguće pregledati podatke za specifičan datum ili raspon datuma te odabrani grad, lokaciju ili skladište.



Slika 6.23. *Prikaz stranice za preuzimanje proizvoda sa stanja.*

Slika 6.24. prikazuje primjer *PDF*-a koji je napravljen za sva skladišta koja sadrže potvrđena preuzimanja u zadanom rasponu datuma. Napravljeni *PDF* se sastoji od:

- *zaglavlje* – sadrži logo skladišta, naslov *PDF*-a, naslov stranice, naziv *url*-a te vremenski raspon za koji su uzeti podatci,
- *tijelo* – sastoji se od sortiranih tablica gdje se svaka tablica odnosi za specifično skladište, lokaciju, grad i proizvode sortirane prema vremenu preuzimanja.

Pri kreiranju *PDF*-a, ako postoji više preuzimanja na isti dan za isti proizvod i skladište, zbrajaju se preuzete količine proizvoda kako bi se što jednostavnije sažele informacije o preuzimanjima.

WAREHOUSE
BUSINESS STORAGE

PREUZIMANJA

Upravljanje skladištima

upravljanjeskladistima.vercel.app

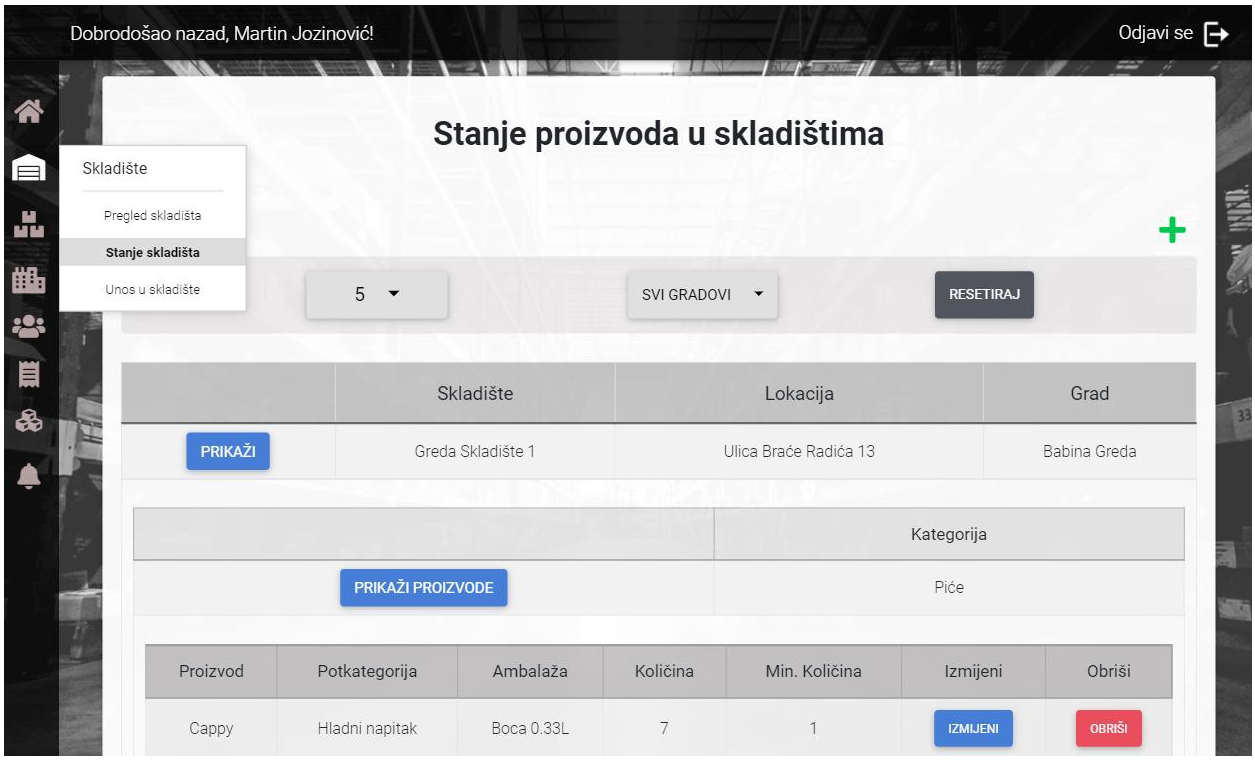
Izvještaj za period 26.07.2021. do 27.08.2021.

Skladiste: Greda Skladiste 1		Lokacija: Ulica Brace Radica 13		Grad: Babina Greda	
Proizvod	Kategorija	Potkategorija	Ambalaza	Kolicina	Datum
Sendvic Kulen	Hrana	Sendvic	Pakiranje	5	24.08.2021.
Heineken	Pice	Alkohol	Boca 0.33L	10	24.08.2021.
Heineken	Pice	Alkohol	Boca 0.33L	7	23.08.2021.
Cappy	Pice	Hladni napitak	Boca 0.33L	1	26.08.2021.
Senzacija	Pice	Hladni napitak	Boca 0.5L	2	23.08.2021.
Capuccino	Pice	Topli Napitak	Vrecica	3	23.08.2021.
Osjecko majica	Promotivni Materijal	Majica	Velicina L	3	16.08.2021.

Slika 6.24. *Prikaz popisa preuzimanja u PDF-u.*

Generiranje izvješća je još dostupno na stranicama *Inventura* i *Unos na stanje*. *Inventura* služi za pregled inventura u skladištima i zapisivanje izbrojane količine proizvoda na skladištu. Kako bi se kreirala inventura potrebno je unijeti skladište, proizvod i izbrojanu količinu. Zbog sigurnosti informacija, prilikom dodavanja inventure u bazu, zapisuje se trenutno stanje skladišta koje se prikazuje u stupcu *Prava količina*. Za lakše potvrđivanje velike količine preuzimanja, inventura ili unosa, na stranici se prikazuje gumb *Potvrdi sve*. Klikom na gumb, potvrđuju se ispravni zahtjevi, dok se neispravni preskoče nakon čega se na zaslonu prikazuje pripadajuća poruka s poslužitelja. Pri radu sa skladištem i proizvodima omogućeno je spremanje posljednje korištenih podataka kako bi se olakšalo kreiranje novog preuzimanja, inventure ili unosa. Za razliku od *Inventure, Unos na stanje* omogućava praćenje tko, kada i koliko proizvoda je stavio na stanje. Nakon uspješne potvrde, zapisana količina proizvoda se dodaje na stanje.

Korisnik može pregledati trenutno stanje skladišta klikom na ikonu skladišta i odabirom opcije *Stanje skladišta* (Slika 6.25.).



Slika 6.25. *Prikaz stanja proizvoda u skladištima.*

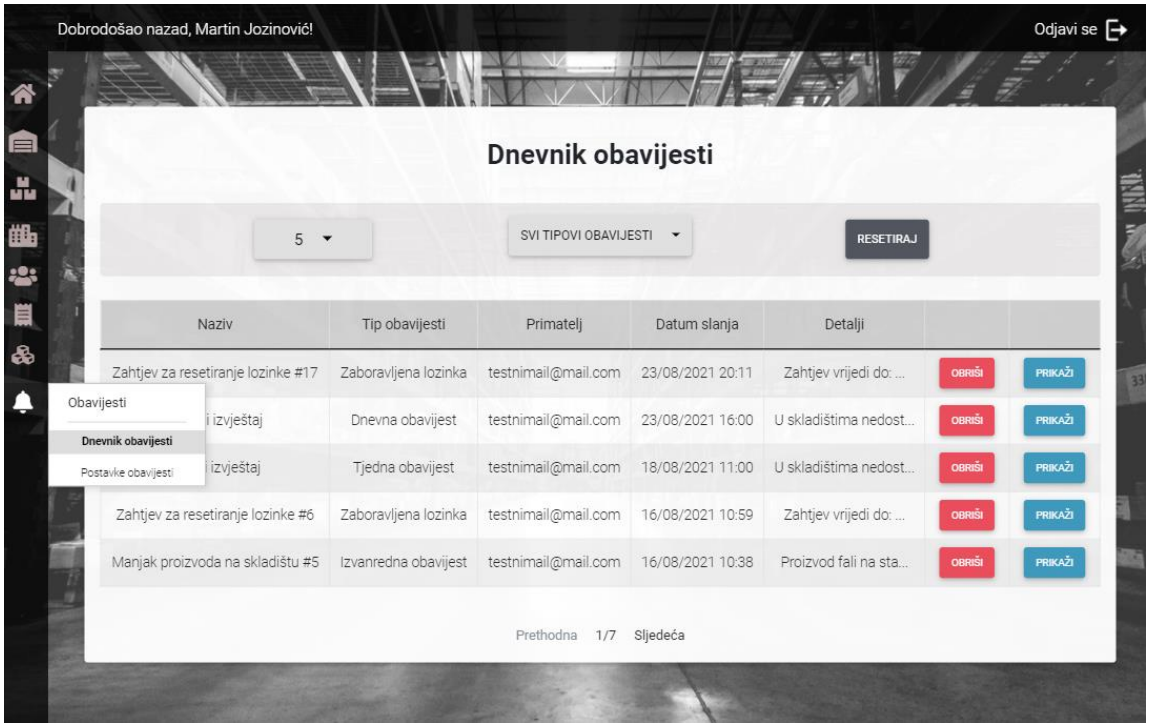
6.2.4. Automatske obavijesti

U aplikaciji su omogućene automatske obavijesti kako bi korisnik na vrijeme dobivao informacije o stanju proizvoda na skladištu te prema tome nadopunio stanje. Razlikuju se pet tipova obavijesti:

1. dnevna obavijest – dnevni podsjetnik u zadano vrijeme,
2. tjedna obavijest – podsjetnik koji se dobiva periodično svaki tjedan u zadano vrijeme,
3. mjesečna obavijest – podsjetnik koji se dobiva prvog dana u mjesecu u zadano vrijeme,
4. izvanredna obavijest –izvanredni podsjetnik koji se šalje *Administratoru* u slučaju da nakon preuzimanja stanje proizvoda padne ispod minimalne granice,
5. zaboravljena lozinka – automatska obavijest prilikom resetiranja lozinke.

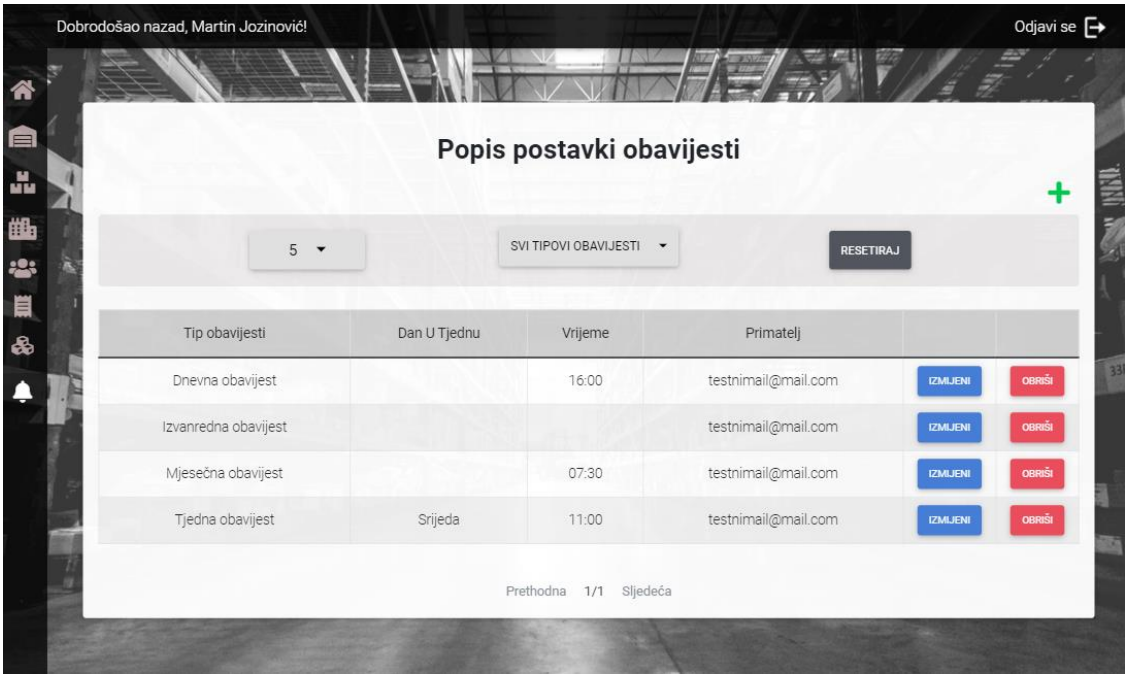
Informacije o poslanim obavijestima i postavkama obavijesti je moguće vidjeti na stranicama *Dnevnik obavijesti* i *Postavke obavijesti*. *Dnevnik obavijesti* sadrži popis poslanih obavijesti i informacija o njima. Klikom na *Prikaži više* za obavijest koja je tipa *Zaboravljena lozinka*, moguće je vidjeti do kada traje poslani link, a ako se radi o dnevnoj, tjednoj ili mjesečnoj obavijesti korisnik može pregledati popis proizvoda i skladišta na kojima su nedostajali proizvodi. Na slici 6.26. moguće je vidjeti popis poslanih automatskih obavijesti. Preko ovog popisa je moguće vidjeti

informacije o poslanoj obavijesti (vrijeme, email primatelja, tip obavijesti i poslane podatke). Stranica omogućava filtriranje podataka prema tipu obavijesti.



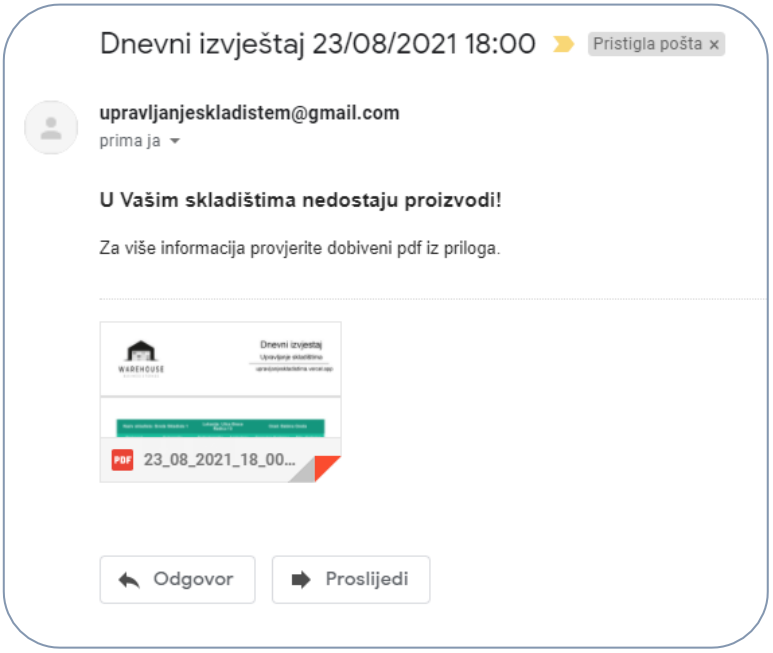
Slika 6.26. *Prikaz dnevnika obavijesti.*

Na stranici *Postavke obavijesti* korisnik dodaje automatske obavijesti koje se šalju ovisno o predanim parametrima (Slika 6.27.).



Slika 6.27. *Prikaz stranice s popisom postavki automatskih obavijesti.*

Kod izvanredne obavijesti, u emailu se šalje informacija o skladištu i proizvodu koji je pao na razinu minimalne količine. Za razliku od ove obavijesti, dnevna, tjedna ili mjesečna automatska obavijest kao prilog šalje napravljeni *PDF* s proizvodima koji nedostaju na skladištima (Slika 6.28.). Rad s *PDF*-ovima omogućava lagan način organiziranja i praćenja stanja skladišta jer se oni generiraju s datumom i vremenom generiranja u nazivu priložene datoteke.



Slika 6.28. *Primjer email-a dnevnog izvještaja.*

7. TESTIRANJE APLIKACIJE

Testiranje predstavlja važnu fazu pri razvoju aplikacije. Razlikuje se funkcionalno i nefunkcionalno testiranje čiji su zahtjevi definirani u potpoglavlju 5.1. Ovisno o odabranom modelu razvoja, testiranje se može izvoditi nakon implementacije pojedine faze i/ili nakon završetka svih faza. Pri izradi ove aplikacije korištena su oba načina testiranja, što znači da se svaka važnija promjena u kodu dodatno testira. Nakon uspješne implementacije aplikacije, napravljeno je testiranje kojim se provjerava ispunjava li softver zadane funkcionalne i nefunkcionalne zahtjeve. Većina ovih zahtjeva je ranije obrađena kroz poglavlja o klijentu i poslužitelju, a odnose se na prikaz i obradu podataka, dodatnu sigurnost s tokenom, rukovanje pri unosu u polja, generiranje izvještaja u *PDF* obliku, prikaz poruke s poslužitelja, zaštita od neovlaštenog pristupa i slično.

7.1. Testiranje i praćenje rada poslužitelja

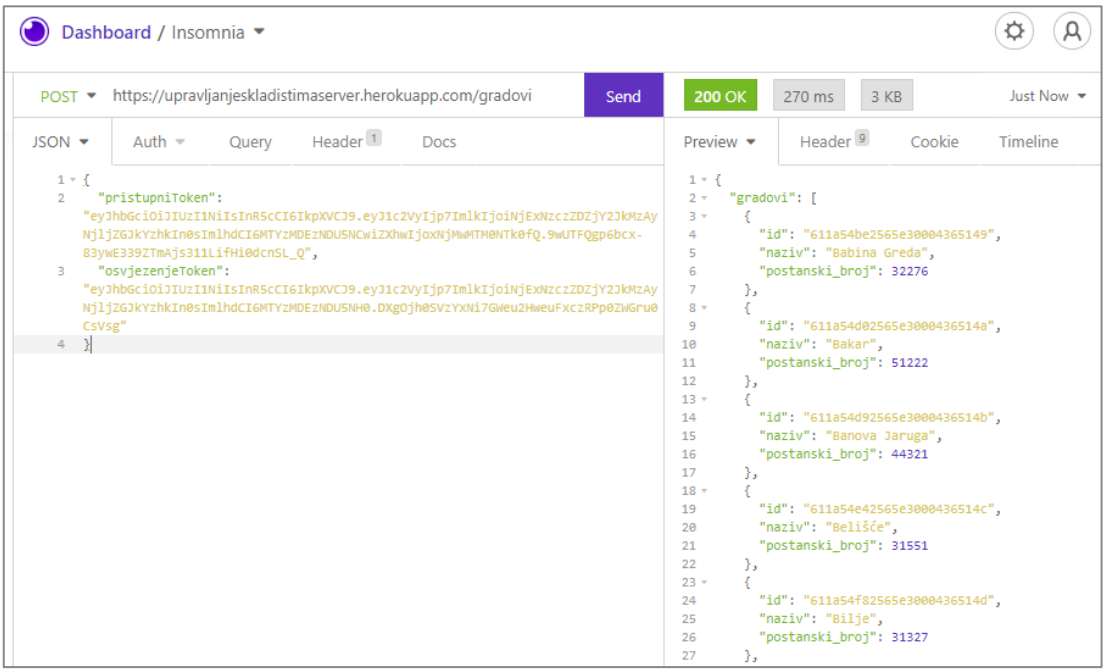
Pri testiranju rada poslužitelja odabran je model *crne kutije* pri čemu se šalju podatci na poslužitelj i promatra slažu li se ti podaci s očekivanima [20]. Za ovaj korak se koristi aplikacija *Insomnia* koja je prethodno opisana u potpoglavlju 4.4. Na slici 7.1. prikazana je struktura mape koja olakšava testiranje ruta te primjer zahtjeva koji se nalaze u mapi *Grad*.

Slika 7.1. *Primjer strukture mapa i zahtjeva za testiranje ruta iz aplikacije Insomnia.*

Većina ruta sa slike ima testiranje zahtjeva za kreiranje, dohvaćanje, izmjenu i brisanje podataka. Mape *Preuzimanje, Inventura* i *Unos na stanje* sadrže još zahtjeve za generiranje izvješća i

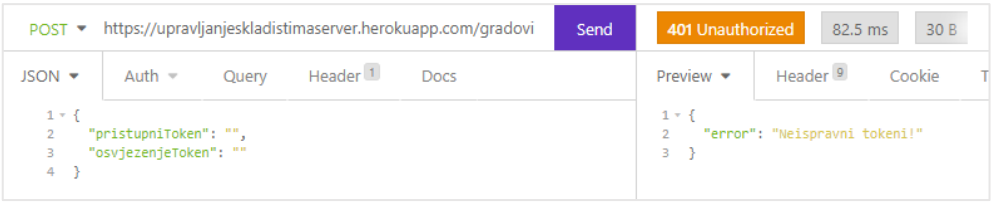
potvrdu, a ruta *Korisnici* sadrži još zahtjeve za prijavu, odjavu, resetiranje lozinke i zahtjev za resetiranje lozinke.

Način testiranja je objašnjen na primjeru dohvaćanja gradova čiji se zahtjev *Dohvati gradove* nalazi u mapi *Grad*. Struktura zahtjeva i odgovor s poslužitelja je prikazan na slici 7.2. Kako bi zahtjev bio ispravan na svim rutama, osim na ruti za prijavu, potrebno je imati tokene koji se dobivaju prilikom prijave u aplikaciju. Taj dio je prethodno napravljen i iskorišteni su dobiveni tokeni koji su zapisani u tijelu zahtjeva u *JSON* obliku (*pristupniToken* i *osvjezenjeToken*).



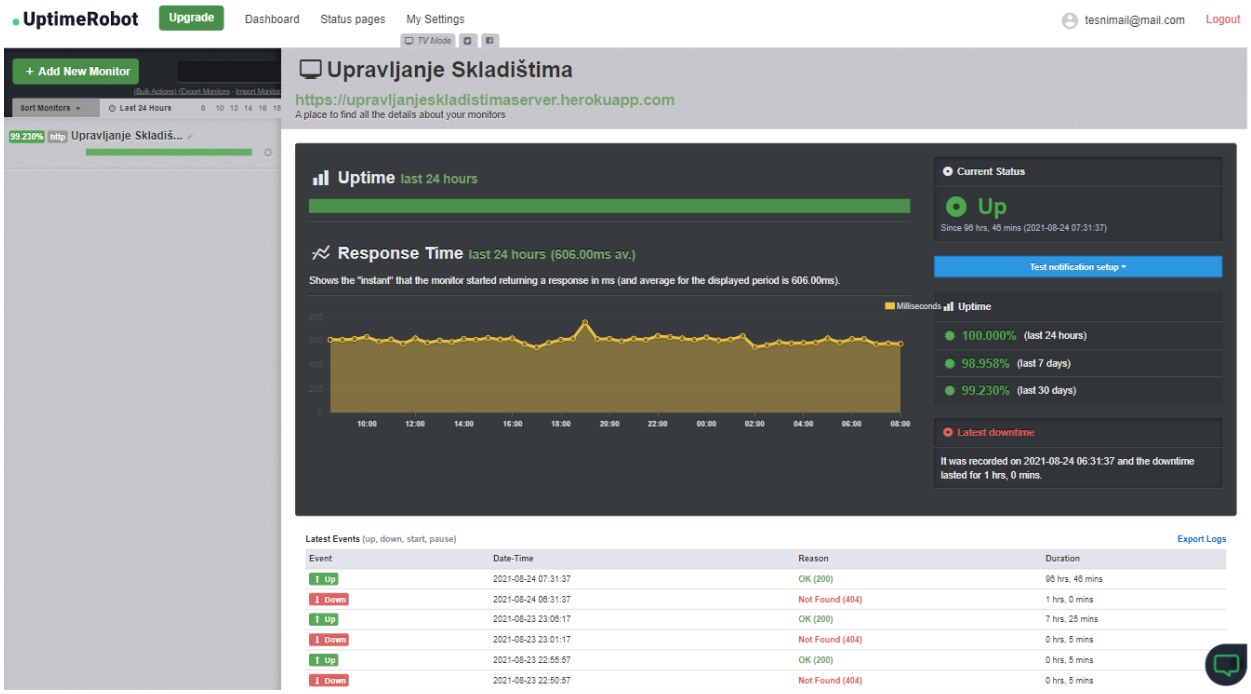
Slika 7.2. *Izgled zahtjeva i odgovora s poslužitelja za rutu Dohvati gradove.*

Glavni *url* zahtjeva je *https://upravljanjeskladistimaserver.herokuapp.com/gradovi*, a tip zahtjeva *POST*. Razlikuju se razni tipovi *API* zahtjeva, a glavni su *POST*, *PATCH*, *PUT*, *GET* i *DELETE*. Uobičajeno je slati zahtjeve za dohvaćanje podataka s tipom *GET* no u ovom slučaju se koriste podaci poslani u tijelu pri čemu je s *ReactJS-a* lakše slanje s tipom *POST*. Ako su svi podaci ispravno napisani, uspješno je poslan zahtjev na poslužitelj. Poslužitelj kao odgovor vraća zatraženu listu gradova. U slučaju da zahtjev nije ispravan zbog neispravnog tokena poslužitelj vraća *status(401)* niste autorizirani za ovu akciju i poruku *Neispravni tokeni* (Slika 7.3.). Ako se pošalje zahtjev s neispravnom ulogom, vraća se odgovor *status(403)* što označava da korisnik nema prava pristupa ovoj ruti. Ovi odgovori poslužitelja se odnose na ispravno rukovanje nefunkcionalnim zahtjevima vezanim za sigurnost od neovlaštenog pristupa i dodatan oblik sigurnosti poput tokena.



Slika 7.3. *Izgled odgovora s poslužitelja u slučaju neispravnih tokena.*

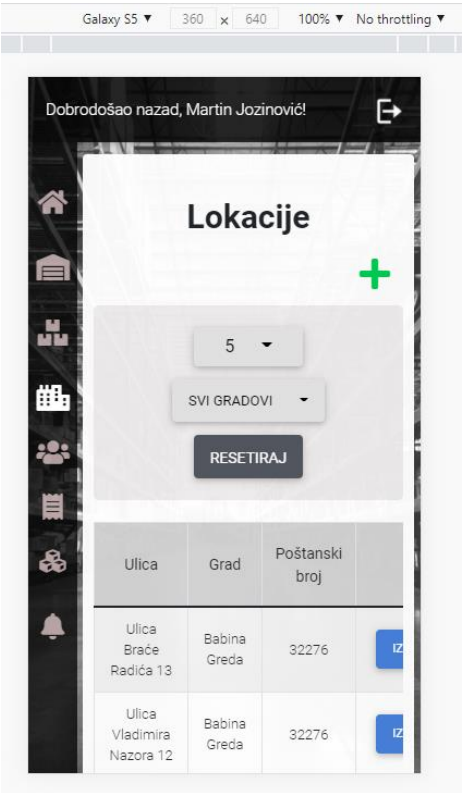
Osim rada ruta vrlo je važno znati vrijeme rada i dostupnost poslužitelja. Navedena metrika označava nefunkcionalan zahtjev vezan za izvođenje aplikacije. Vrijeme rada se odnosi na vrijeme izraženo u godinama, mjesecima, danima, satima, minutama i sekundama, dok je dostupnost postotak vremena u vremenskom intervalu kada se poslužitelj mogao koristiti za predviđenu svrhu [21]. Za navedeno praćenje korištena je aplikacija *UptimeRobot* koja pruža usluge analize i praćenja rada stranice. Na slici 7.4. prikazano je sučelje *UptimeRobot* aplikacije na kojem se može vidjeti koliko vremena je sustav radio u posljednjih dvadeset i četiri sata, sedam dana ili mjesec dana. Uz to prikazani su podaci o dostupnosti sustava u postotku, ukupno vrijeme u kojem poslužitelj nije bio dostupan, poslani zahtjevi na poslužitelj te brzina odgovora sustava čiji je prosjek oko šesto milisekundi. Dostupnost poslužitelja je vrlo važna za ovu aplikaciju zbog slanja automatskih obavijesti s poslužitelja te zbog toga stranica radi analizu na temelju slanja zahtjeva svakih pet minuta.



Slika 7.4. *Prikaz sučelja platforme za analizu rada poslužitelja.*

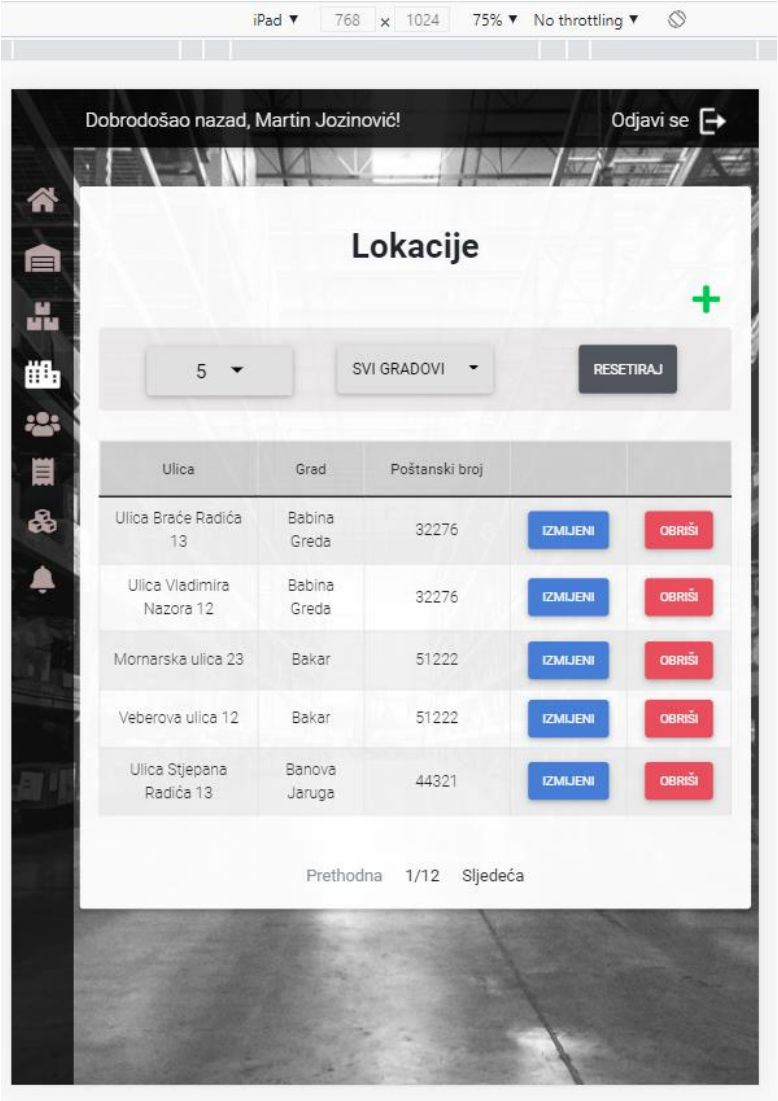
7.2. Testiranje responzivnosti

Testiranje responzivnosti stranice se odnosi na zahtjev za pokretanje aplikacije preko internet preglednika neovisno o veličini zaslona ili platformi na kojoj se izvodi. Za testiranje su odabrane dvije veličine zaslona: *768 x 1024* piksela koji predstavlja prosjek zaslona srednje veličine i tableta, dok se *360 x 640* piksela odnosi na prosječnu veličinu zaslona mobitela. Na slici 7.5. prikazan je izgled zaslona na mobilnim uređajima, gdje se tablica može pomicati lijevo-desno za lakši pregled podataka.



Slika 7.5. *Izgled zaslona na mobilnom uređaju.*

Slika 7.6. prikazuje zaslon srednje veličine koju najčešće imaju *tableti*. Dizajn je napravljen da bude intuitivan korisniku, od ikona do gumbova s nazivom akcije pri čemu se pazi na količinu i raspored informacija kako bi se podiglo korisničko iskustvo.



Slika 7.6. *Izgled zaslona na tabletu i zaslonima srednje veličine.*

8. ZAKLJUČAK

Cilj ovog diplomskog rada je bio istražiti postojeća rješenja na tržištu koja se bave vođenjem stanja skladišta ili inventara putem internet preglednika te na temelju njih razviti vlastitu aplikaciju koja nudi prednosti u odnosu na većinu sličnih aplikacija na tržištu. Glavni potencijal je to što je aplikacija predviđena za hrvatsko tržište na kojem se trenutno rijetko koriste aplikacije ovog tipa. Aplikacija nudi mogućnost obavještavanja korisnika o neočekivanom odstupanju od stanja, periodično slanje dnevnih, tjednih ili mjesečnih izvješća i slanje izvanrednih izvješća u slučaju da prilikom preuzimanja stanje proizvoda padne ispod minimalne granice. Jednostavnost aplikacije smanjuje potrebe za složenom dokumentacijom, a generiranje izvješća omogućava lakše vođenje evidencije stanja proizvoda.

U aplikaciji postoje dvije uloge: korisnik koji može pratiti stanje skladišta, dodavati proizvode na stanje te kreirati preuzimanja i inventure za dodijeljena skladišta i administrator koji ima napredne mogućnosti praćenja svih preuzimanja i inventura, dodavanje zaliha, pristup stanju skladišta i vođenju osoblja koje trenutno radi na skladištima.

Prilikom ulaska u aplikaciju korisnik dobiva osnovne informacije o napravljenim akcijama u tekućem mjesecu poput ukupnog broja preuzimanja, inventura i unosa na stanje. Podaci su prikazani putem tablica, a unos, izmjena ili brisanje podataka se vrši putem dijaloga. Osigurana je dvostruka provjera s potvrdom kako bi se smanjile pogreške pri unosu. Pogreške u inventuri može izbrisati samo administrator. Pri unosu proizvoda na stanje prikazuje se trenutna količina proizvoda na skladištu.

Aplikacija je dostupna preko internet preglednika bez obzira na platformu i osigurana je responzivnost kako bi se mogla koristiti i na manjim uređajima poput mobitela. Nedostaci predloženog rješenja su to što nema mogućnost unosa proizvoda na stanje putem barkoda te je potrebno odrediti ciljano tržište kako bi se aplikacija uspješno prilagodila.

U budućnosti je plan spojiti funkcionalnosti glavnih zaslona pri čemu bi se s jednog zaslona lagano pratilo i pristupalo inventurama, preuzimanjima, stanju i unosu na stanje i time povećala intuitivnost korištenja aplikacije.

LITERATURA

[1] K. Ljudevit, R. Maršanić, i V. Jedvaj, *Upravljanje zalihama materijalnih dobara i skladišno poslovanje u logističkoj industriji*, sv. 8, 3 sv. Koprivnica: Sveučilište Sjever, 2014.

[2] D. Waters, *Inventory Control and Managment, Second Edition*. Sussex, England: Wiley, 2003.

[3] „What is Inventory Management? Techniques for 2021“, *Unleashed Software*. https://www.unleashedsoftware.com/inventory-management-guide (pristupljeno srp. 01, 2021).

[4] „Five simple ways to improve warehouse efficiency“, *Supply Management*. https://www.cips.org/supply-management/opinion/2015/august/five-simple-ways-to-improve-warehouse-efficiency/ (pristupljeno srp. 01, 2021).

[5] „Online Inventory Management Software | Zoho Inventory“. http://www.zoho.com/inventory/ (pristupljeno srp. 01, 2021).

[6] „Inventory Management Software System Made Easy | inFlow“, *inFlow Inventory*. https://www.inflowinventory.com/ (pristupljeno srp. 01, 2021).

[7] „Katana | Manufacturing ERP Software for Modern Manufacturers“, *Katana*. https://katanamrp.com/ (pristupljeno srp. 01, 2021).

[8] „Best Warehouse Management Software 2021 | Reviews of the Most Popular Tools & Systems“. https://www.capterra.com/warehouse-management-software/ (pristupljeno srp. 01, 2021).

[9] „30-days-of-react-ebook-fullstackio.pdf“. Pristupljeno: srp. 01, 2021. [Na internetu]. Dostupno na: https://www.newline.co/fullstack-react/assets/media/sGEMe/MNzue/30-days-of-react-ebook-fullstackio.pdf

[10] „React vs Angular: 10 Most Important Differences You Must Know!“ https://www.guru99.com/react-vs-angular-key-difference.html (pristupljeno srp. 01, 2021).

[11] „Why The Hell Would I Use Node.js? A Case-by-Case Tutorial“, *Toptal Engineering Blog*. https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js (pristupljeno srp. 01, 2021).

[12] „How to Build an eCommerce Web Application using Node.js?“, *Insights on Latest Technologies - Simform Blog*, lis. 09, 2019. https://www.simform.com/build-ecommerce-web-application-node-js/ (pristupljeno srp. 09, 2021).

[13] „The most popular database for modern apps“, *MongoDB*. https://www.mongodb.com (pristupljeno srp. 01, 2021).

[14] K. Wiegers i J. Beatty, „Software requirements“, u *Software requirements*, Third Edition., Microsoft Press, 2013, str. 42/673.

[15] Shaun Anderson |, „How Fast Should A Website Load & How To Speed It Up“, *Hobo*, velj. 11, 2020. https://www.hobo-web.co.uk/your-website-design-should-load-in-4-seconds/ (pristupljeno ruj. 03, 2021).

[16] P. Rome, „What are Non Functional Requirements — With Examples“, *Perforce Software*. https://www.perforce.com/blog/alm/what-are-non-functional-requirements-examples (pristupljeno kol. 30, 2021).

[17] „What is the MERN Stack? Introduction & Examples“, *MongoDB*. https://www.mongodb.com/mern-stack (pristupljeno srp. 09, 2021).

[18] „React, TypeScript & Mobx“, *DEV Community*. https://dev.to/shevchenkonik/react-typescript-mobx-4mei (pristupljeno srp. 09, 2021).

[19] „Nodemailer :: Nodemailer“. https://nodemailer.com/about/ (pristupljeno kol. 30, 2021).

[20] N. Kshirasagar i T. Priyadarshi, „Software testing and quality assurance“, New Jersey: Wiley, 2008, str. 58/648.

[21] „Server Uptime Monitoring for Successful IT Operations“, *Kaseya*. https://www.kaseya.com/blog/2020/10/08/server-uptime-monitoring/ (pristupljeno kol. 28, 2021).

SAŽETAK

Na hrvatskom tržištu se rijetko koriste aplikacije za upravljanje skladištem i zbog toga bi se ovaj tip aplikacije uz pravilnu prilagodbu lako proširio. Proučeno je trenutno stanje tržišta te je na temelju toga razvijena aplikacija za tvrtke koje koriste manja skladišta i inventare. Aplikacija omogućava online vođenje skladišta uz mogućnost prilagodbe ovisno o tipu skladišta kojim se upravlja.

Osim vođenja stanja skladišta, aplikacija pruža mogućnost obavještavanja korisnika o neočekivanom odstupanju proizvoda i periodično slanje izvješća o trenutnom stanju na skladištu. Pomoću ovih informacija korisnik može pravovremeno reagirati. Radnik može vidjeti samo skladišta na koja je dodijeljen i na njima može raditi unos, preuzimanje ili inventuru proizvoda. Administrator ima pristup svim mogućnostima aplikacije poput generiranja izvješća i on upravlja osobljem.

Aplikacija je dostupna preko internet preglednika pri čemu je zasebno postavljen klijent i poslužitelj što pruža veću sigurnost i kontrolu mrežnog prometa uz jednostavan pristup sigurnosnim kopijama baze podataka s jednog mjesta.

Ključne riječi: ReactJS, NodeJS, MongoDB, Mobx, vođenje skladišta

ABSTRACT

Title: Internet Application for Warehouse Status Management

Warehouse management applications are not often used on the Croatian market, and the application would expand with the correct customization. Based on the current market, an application was developed for companies that use smaller warehouses and inventories. It is necessary to study the current market to achieve this and, on that basis, propose a solution to companies that use smaller warehouses or inventories. A solution would be to switch to online warehouse management with easy customization of the application depending on the warehouse type.

Besides managing the state of the warehouse, the application provides the ability to notify users of unexpected deviations and periodically send reports with information about the current state of the warehouse. With this information, the user can respond on time and add products. A user can only see the repositories to which he is assigned and can perform product entry, receipt, or inventory on them. The administrator has access to all application features such as report generation and manages the team working on the repository.

The application is available via an Internet browser with a separate client and server, which provides greater security and control of network traffic with easy access to database backups from one place.

Keywords: ReactJS, NodeJS, MongoDB, Mobx, warehouse management

ŽIVOTOPIS

Franjo Josip Jukić rođen je 28.12.1997. u Vinkovcima. Pohađao je Osnovnu školu Stjepana Cvrkovića u Starim Mikanovcima, nakon toga upisuje Tehničku školu Ruđera Boškovića Vinkovci smjer Tehničar za mehatroniku. Na trećoj godini se prebacuje u Srednju strukovnu školu Antuna Horvata Đakovo smjer Tehničar za mehatroniku. Nakon završetka srednje škole 2016. upisuje preddiplomski studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Preddiplomski sveučilišni studij završava 2019. godine te iste godine upisuje diplomski sveučilišni studij Računarstva smjer Programsko inženjerstvo.

_____

Potpis autora

PRILOZI

Programski kod je priložen na CD-u.