OBLIKOVANJE LOGIKE PROGRAMA I PROGRAMSKO RJEŠENJE ZA JEDNOSTAVNO SKLADIŠNO POSLOVANJE

Pek, Filip

Sveučilište Josipa Jurja Strossmayera u Osijeku
Ekonomski fakultet u Osijeku
Preddiplomski studij (*Poslovna informatika*)

Filip Pek

OBLIKOVANJE LOGIKE PROGRAMA I PROGRAMSKO RJEŠENJE ZA JEDNOSTAVNO SKLADIŠNO POSLOVANJE

Završni rad

Osijek, 2021.

Sveučilište Josipa Jurja Strossmayera u Osijeku
Ekonomski fakultet u Osijeku
Preddiplomski studij (*Poslovna informatika*)

Filip Pek

OBLIKOVANJE LOGIKE PROGRAMA I PROGRAMSKO RJEŠENJE ZA JEDNOSTAVNO SKLADIŠNO POSLOVANJE

Završni rad

Kolegij: Oblikovanje i implementacija informacijskih sustava

JMBAG: 0149219852
e-mail: fpek@efos.hr

Mentor: prof. dr. sc. Josip Mesarić

Osijek, 2021.

Josip Juraj Strossmayer University of Osijek
Faculty of Economics in Osijek
Undergraduate Study (Business informatics)

Filip Pek

SOFTWARE LOGIC DESIGN AND SOFTWARE SOLUTION FOR SIMPLE STORAGE BUSINESS

Završni rad

Osijek, 2021.

IZJAVA

O AKADEMSKOJ ČESTITOSTI, PRAVU PRIJENOSA INTELEKTUALNOG VLASNIŠTVA, SUGLASNOSTI ZA OBJAVU U INSTITUCIJSKIM REPOZITORIJIMA I ISTOVJETNOSTI DIGITALNE I TISKANE VERZIJE RADA

1. Kojom izjavljujem i svojim potpisom potvrđujem da je završni rad isključivo rezultat osobnog rada koji se temelji na mojim istraživanjima i oslanja na objavljenu literaturu. Potvrđujem poštivanje nepovredivosti autorstva te točno citiranje radova drugih autora i referiranje na njih.
2. Kojom izjavljujem da je Ekonomski fakultet u Osijeku, bez naknade u vremenski i teritorijalno neograničenom opsegu, nositelj svih prava intelektualnoga vlasništva u odnosu na navedeni rad pod licencom *Creative Commons Imenovanje – Nekomercijalno – Dijeli pod istim uvjetima 3.0 Hrvatska.*
3. Kojom izjavljujem da sam suglasan da se trajno pohrani i objavi moj rad u institucijskom digitalnom repozitoriju Ekonomskoga fakulteta u Osijeku, repozitoriju Sveučilišta Josipa Jurja Strossmayera u Osijeku te javno dostupnom repozitoriju Nacionalne i sveučilišne knjižnice u Zagrebu (u skladu s odredbama Zakona o znanstvenoj djelatnosti i visokom obrazovanju, NN br. 123/03, 198/03, 105/04, 174/04, 02/07, 46/07, 45/09, 63/11, 94/13, 139/13, 101/14, 60/15).
4. Izjavljujem da sam autor predanog rada i da je sadržaj predane elektroničke datoteke u potpunosti istovjetan sa dovršenom tiskanom verzijom rada predanom u svrhu obrane istog.

Ime i prezime studenta: Filip Pek

JMBAG: 0149219852

OIB: 91468116853

e-mail za kontakt: fpek@efos.hr

Naziv studija: Poslovna informatika

Naslov rada: Oblikovanje logike programa i programsko rješenje za jednostavno skladišno poslovanje

Mentor: prof. dr. sc. Josip Mesarić

U Osijeku, 2021. godine

Potpis: 

Oblikovanje logike programa i programsko rješenje za jednostavno skladišno poslovanje

SAŽETAK

U radu je izgrađen jednostavan sustav za skladišno poslovanje koji će omogućiti jednostavan unos i pregled podataka o robama koje se kodiraju bar kodom. Kontekstualni okvir: skladište trgovačke tvrtke iz kojeg se roba nadopunjuje u maloprodaji i može se prodavati u veleprodaji. Roba dolazi u transportnim jedinicama (kutija/ paleta) koje su označene bar kodom i ima oznaku broja maloprodajnih jedinica. Na temelju prijemnih dokumenata (tovarnog lista s brojem paleta, računa-otpremnice) i kontrolom roba se uvodi u skladišnu bazu. Barkodirana je transportna jedinica, kutija i pojedinačni proizvod u maloprodaji. Maloprodaja se zadužuje u prodajnim jedinicama a veleprodaja ide u kartonima ili paletama. Program omogućava unos i promjenu stanja količine svakog artikla i generiranje interne međuskladišnice i otpremnice u veleprodaji.

Sustav omogućava uvid u stanje pojedinih roba koje mogu imati status raspoloživo ili naručeno. Sustav mora osigurati uvid u stanje skladišta. Aplikacija je izrađena kao web aplikacija korištenjem HTML-a, CSS-a i javascripta.

Ključne riječi: skladišno poslovanje, logika programa, HTML, CSS, javascript

Software logic design and software solution for simple storage business

ABSTRACT

A simple system for storage business which will allow simple data entry about goods that are coded via barcode was built for this final paper. Context: merchandise warehouse from which the goods are supplemented in retail and can be sold in wholesale. Goods come in transport units (box/pallet) which are marked with a barcode and have a tag of retail units quantity. Based on admission documents (waybill with the number of pallets, invoice-shipping bill) and control goods are introduced into the warehouse database. Transport unit, box and individual product are barcoded in retail. Retail is debited in sales units and wholesale in carboard boxes or pallets. Software enables entry and editing of the quantity for each product and it generates internal delivery note in wholesale.

System enables access to the state of individual goods which can have a state of available or ordered. System enables insight into the state of the warehouse. Web application was developed by using HTML, CSS and javascript.

Key words: storage business, software logic, HTML, CSS, javascript

SADRŽAJ

1. Uvod

Informacijski sustavi su postali neizbježan dio svakog uspješnog poslovanja. Briga o toku podataka i informacija, njihovom čuvanju, obradi i raspodjeli je dodijeljena računalima koja ubrzanom obradom pružaju mogućnost kvalitetnijeg donošenja odluka. Svaka aplikacija mora slijediti logiku poslovnog procesa kojeg podupire kao i podatke, njihove transformacije, reprezentaciju, pohranu i distribuciju. K tomu, aplikacija mora osigurati visoku produktivnost, pouzdanost i sigurnost i povezivost s drugim programskim rješenjima unutar informacijskog sustava. Da bi udovoljila navedenim kriterijima, aplikacija se mora planirati, konstruirati pomoću različitih dijagrama i konačno kodirati u izabranom programskom jeziku, testirati i implementirati. Iako su navedeno osnovni zahtjevi, pristup izradi programskih aplikacija se može donekle razlikovati. U radu će se prikazati takozvani Agilan način pristupa razvoju softvera koji je danas najčešće korišten način razvoja softvera, te će u radu biti opisano ekstremno programiranje kao jedna od metoda agilnog razvoja softvera.

Cilj rada je pokazati proces izrade jednostavne aplikacije i staviti naglasak na pomno planiranje i postavljanje uvjeta koje aplikacija treba ispuniti, a potom i izrade logike programskog rješenja.

U prvom poglavlju pobliže će se istražiti poslovno područje – u slučaju ovog rada to je skladišno poslovanje. Identificirati će logičke aktivnosti koje se provode u okviru ove poslovne funkcije, problemi s kojima se suočavaju nositelji aktivnosti i pretpostaviti na što treba biti orijentiran informacijski sustav za skladišno poslovanje. Prikazati će se metode i alati za izradu aplikacije, počevši od dizajniranja logike i izgleda web aplikacije, odabira programskog jezika i kreiranja programskog koda. Nadalje, pobliže će biti istraženi takozvani „stupovi“ weba: HTML (HyperText Markup Language), CSS (Cascading Style Sheets) i JavaScript koji će se koristiti u oblikovanju programskog koda.

Kako bi logiku programskog rješenja, zahtjeve programskog rješenja te sudionike i veze unutar sustava bilo lakše prenijeti timu programera koji će izraditi aplikaciju koristi se UML (Unified Modeling Language) koji prikazuje sve od navedenog kroz standardizirani grafički prikaz.

2. Teorijska podloga i prethodna istraživanja

2.1. Kratki opis skladišnog poslovanja

Skladišno poslovanje je važan dio svakog poslovnog sustava koji je povezan s materijalnim proizvodima u cjelokupnom opskrbnom lancu materijalnih dobara. Prema Hruškar i Šiljeg *„Skladište izravnava neujednačenost ponude i potražnje. Kad ponuda premašuje potražnju, skladište pohranjuje proizvod u iščekivanju zahtjeva kupaca. Kad potražnja premašuje ponudu skladište može ubrzati kretanje proizvoda do kupaca osiguravajući dodatne usluge, kao primjerice označivanje cijena, pakiranje proizvoda, ili montažni sklop" (Hruškar, Šiljeg, 1985).* Skladišno poslovanje omogućava lakši i efikasniji logistički tok. Zadatak logistike i skladišnog poslovanja je imati pravi proizvod na pravom mjestu, u pravo vrijeme i poželjno je po najnižim cijenama. „*Odabir načina skladištenja, struktura zaliha, lokacija skladišta, način i ustroj prijevoza, mogućnost brze i fleksibilne reakcije na upite i zahtjeve kupaca zasigurno povećavaju konkurentnost društva na tržištu" (Krpan, Maršanić, Jedvaj, 2014).* Isti autori ističu da je *„skladište točka u logističkoj mreži na kojoj se predmet skladištenja prihvaća ili prosljeđuje u nekom drugom smjeru unutar mreže. Skladište je prostor u kojemu se roba preuzima i otprema te čuva od raznih fizičkih, kemijskih i atmosferskih utjecaja i, naravno, krađe" (Krpan, Maršanić, Jedvaj, 2014).*

Kako je skladište samo po sebi, a tako i zalihe i proizvodi koji se čuvaju unutar skladišta, značajan trošak, važno je cijeli proces skladištenja realizirati na način koji će donijeti što manje troškove skladištenja.

Raspoloživost zaliha materijala, poluproizvoda ili u konačnici gotovih proizvoda će proizvodnom društvu osigurati kontinuitet proizvodnje dok će trgovačkom društvu osigurati prodajnu spremnost. Zbog sveprisutnosti neizvjesnosti u proizvodnji i poslovanju upravljanje zalihama je nužno za opstanak poduzeća. Ukoliko poduzeće raspolaže viškom zaliha potrebna su im veća skladišta i blokirana obrtna sredstva što će dovesti do povećanja troškova. U suprotnom, manjkom zaliha, dovodi se do prekida proizvodnje što opet vodi do povećanja troškova. Dok su oba slučaja negativna za poduzeće, povećane zalihe imaju i svoje prednosti. Poduzeće ostvaruje količinski popust na veće narudžbe, a višak zaliha, za razliku od manjka koji djeluje samo negativno na poduzeće, može učiniti poduzeće fleksibilnijim na tržištu. „*Nedostatne zalihe ne omogućuje redovan tijek poslovanja i uzrokuju zastoje u proizvodnji i plasmanu proizvoda. Uzrok je povećavanje troškova u svim razinama lanca nabave, a imaju i izravan utjecaj na imidž poduzeća koji se mjeri izgubljenom prodajom - u nabavi se povećavaju*

troškovi zbog dodatnog naručivanja, u proizvodnji dolazi do zastoja, a u prodaji nastaju štete (može doći do raskida ugovora od strane kupca i slično)" (Krpan, Maršanić, Jedvaj, 2014).

2.2. Informacijski sustav skladišnog poslovanja

Skladištenje se mora promatrati kao dio poslovnog sustava. Sustavni ili sistemski pristup osigurava razumijevanje istraživanog fenomena. Sustav je skup komponenti koje su zajedno u mogućnosti proizvesti rezultat koji ne može postići komponenta sama. Strukturu samog sustava čine njegove komponente, njihov položaj u odnosu na druge te veze među njima. Veze mogu biti uspostavljene neposredno ili posredno preko drugih komponenti.
Funkciju sustava je moguće poistovjetiti sa svrhom postojanja sustava odnosno to je uloga koju sustav ima u svojoj okolini i način na koji tu svrhu ostvaruje.
Arhitektura sustava predstavlja strukturu, dinamiku, funkcije i načine njihove izgradnje i realizacije. Obuhvatiti će sve razine promatranja (kontekst, koncept, logiku, fiziku i potrebne razine detaljiziranosti), a istovremeno odgovoriti na pitanja: tko, što, zašto, kako, kada i gdje?

„Informacijski sustav (IS) tvrtke obuhvaća sve ono što je vezano za prikupljanje, čuvanje, obradu i raspodjelu podataka i informacija" (Sekso, 2011).
„Ustroj skladišnog poslovanja uvjetovan je vrstom gospodarske djelatnosti i različit je kod proizvodnih društava, trgovine i uslužnih djelatnosti (distribucije i transporta)" (Sekso, 2011) iz čega je moguće zaključiti da ne postoji jedinstveno programsko rješenje odnosno informacijski sustav koji može univerzalno riješiti ustroj skladišnog poslovanja.

Skladišno poslovanje ne može uspješno, profitabilno i učinkovito poslovati bez adekvatno modeliranog informacijskog sustava. Informacijski sustav omogućuje upravljanje poslovima, ljudskim potencijalima, financijskom imovinom, kupcima i dobavljačima na osnovu kvalitetnih podataka i informacija. Najvažnije informacije u skladišnom poslovanju su: o skladištenju robe, težini, količini, vrsti, cijeni, smještaju, o kupcima i dobavljačima, o zalihama, o čuvanju, održavanju, troškovima skladišnog poslovanja i sl.

Korištenje informacijskog sustava i računalne podrške rezultirati će unapređenjem naručivanja, olakšat će pronalazak traženih proizvoda, praćenje neizvršenih i vraćenih narudžbi te će osigurati praćenje, održavanje i u konačnici poboljšanje standarda rada u skladištu.

2.3. Pristup stvaranju programskih rješenja

„Klasične metode projektiranja informacijskih sustava, kao npr. vodopadni pristup, spiralni pristup i drugi koje su vremenski zahtjevne i orijentirane prema opsežnom dokumentiranju, u današnje vrijeme čestih promjena pokazale su se nedovoljno učinkovitima. Kao novi način upravljanja projektima razvoja softvera pojavljuju se agilne metode“ (Zekić-Sušac, 2013).
Najpopularnije agilne metode su: Scrum, Extreme Programming (XP), Crystal, Dynamic Systems Development Method. Lean Development i Feature-Driven Development.

Prema Zekić-Sušac (2013) svaka metoda ima svoje specifičnosti ali sve se vode određenim principima kao što su:

- *zadovoljstvo naručitelja ranom i neprekinutom isporukom programskog rješenja*
- *mogućnost promjene zahtjeva, čak i u kasnoj fazi razvoja*
- *isporuka upotrebljivog softvera u razmacima od nekoliko tjedana do nekoliko mjeseci, nastojeći da razmak bude što kraći*
- *suradnja poslovnih ljudi i razvojnih inženjera tijekom cjelokupnog trajanja projekta*
- *projekte treba graditi oslanjajući se na motivirane pojedince*
- *najučinkovitija razmjena informacija je licem u lice*
- *upotrebljiv softver je osnovno mjerilo napretka*
- *poticanje i podržavanje održivog razvoja, trebalo bi biti moguće neograničeno dugo zadržati jednak tempo rada*
- *fokusiranost na tehničku izvrsnost i dobar dizajn*
- *naglasak na jednostavnosti*
- *najbolje arhitekture, projektne zahtjeve i dizajn stvaraju samo-organizirajući timovi*
- *timovi u redovitim intervalima trebaju razmatrati načine kako postati učinkovitiji, a zatim se uskladiti i prilagoditi*

Agilno modeliranje obuhvaća slijedeće faze:

- *Počinjanje*
    - *Identificira se inicijalni obujam projekta, potencijalna arhitektura sustava i prihvaćanje od zainteresiranih strana*
- *Elaboracija*
    - *Pokazuje se arhitektura sustava*
- *Konstrukcija*

    - *Razvija se softver na postepenoj osnovi polazeći od najviših prioriteta*
- *Prijelaz*
    - *Vrednuje se i provjerava softver na operativnom okruženju*

Ekstremno programiranje (XP) je najpopularnija metoda agilnog razvoja softvera. Razvio ju je Kent Beck sredinom 80-tih godina prošlog stoljeća. Jedna je od mlađih metoda i prvi put je uporabljena 6. ožujka 1996. godine. Temelji se na pet vrijednosnih grupa: komunikativnost, jednostavnost, povratna veza – testiranje, povjerenje i poštovanje. *„Favorizira se jednostavan dizajn, zajedničke metafore, kolaboracija i verbalna komunikacija. Članovi tima imaju visoko međusobno povjerenje i ne zahtjeva se provjera ispravnosti jer se vjeruje u njihovu visoku profesionalnost (rad u parovima; kolektivno vlasništvo nad softverom). Podrazumijeva se visoka lojalnost timu" (Zekić-Sušac, 2013).*
„XP je prikladno za korištenje kod: prototipskog pristupa gdje se zahtjevi mijenjaju često i gdje nema provjerenih rješenja, u istraživačkim projektima gdje razvoj softvera nije glavni cilj već razvoj domenskog znanja, u manjim projektima gdje je poželjno neformalno vođenje projekta, gdje se može okupiti motivirani visoko profesionalni tim" (Zekić-Sušac, 2013).

„Modeliranje se definira kao čin predstavljanja nečega obično sa manje detalja ili u manjem obujmu. Model je pojednostavljena slika stvarnosti u kojoj se ističu najvažnija svojstva te stvarnosti.
Modeli aspekata s kojih se promatra informacijski sustav određuju:

- *Model podataka*
- *Model funkcija i procesa*
- *Model događaja*
- *Model resursa*
- *Model programa" (Zekić-Sušac, 2013).*

3. Metodologija rada

Projekt koji je predmet ovog rada je po obujmu mali projekt ali se i na njega može primijeniti opisani pristup. Razvoj neće imati sve faze i karakteristike agilnog pristupa ali će se svakako prepoznavati osnovne postavke i filozofija pristupa. Sustav će se najprije modelirati upotrebom UML te će se na temelju razvijenih modela oblikovati programsko rješenje upotrebom skriptnih jezika

3.1. UML – Unified Modeling Language

UML (Unified Modeling Language) ili prevedeno objedinjeni jezik za modeliranje je opći jezik za modeliranje namijenjen pružanju standardnog načina vizualizacije dizajna sustava u području softverskog inženjerstva. Stvaranje UML-a motivirano je željom za standardizacijom notacijskih sustava i pristupa dizajnu softvera. Modeliranje podrazumijeva oblikovanje softverskih rješenja prije kodiranja. *„Model omogućava lakše i brže razumijevanje, evaluaciju i kritičko promatranje za razliku od stvarnog sustava. Prednosti UML-a su formalnost (svaki element ima strogo definirano značenje), sažetost (jednostavne notacije), sveobuhvaćenost (opisuje sve važne aspekte sustava), skalabilnost (upotrebljiv za male ali i velike sustave) i standardiziranost (kontroliran prema standardima). Modeliranje pomoću koda je previše detaljizirano, a modeliranje neformalnim jezicima preopširno, zbunjujuće i dvosmisleno)" (Mesarić, Šebalj 2016).*

Prema Miles, Hamilton (2006) Kruchtenov 4+1 model „razbija" model u skup pogleda, gdje svaki pogled prikazuje određeni aspekt sustava:

- *Logički: apostraktni opisi dijelova sustava, prikazuje od čega je sustav načinjen i kako dijelovi sustava djeluju međusobno*
- *Procesni: opisuje procese unutar sustava*
- *Razvojni: opisuje kako su dijelovi sustava organizirani u module i komponente*
- *Fizički: opisuje kako dizajn sustava, opisan u 3 prethodna pogleda, poprima entitete iz stvarnog svijeta, dijagrami u ovom pogledu pokazuju kako se apstraktni dijelovi preslikavaju u konačno razvijeni sustav*
- *Slučaj korištenja: opisuje funkcionalnost sustava promatranog „izvana", služi za prikaz što sustav treba raditi, svi drugi pogledi se oslanjaju na ovaj*

Dijagram slučajeva korištenja prikazuje samo ono što sustav treba raditi (funkcionalni zahtjevi). Funkcionalne zahtjeve se prikazuje na početku projekta (ušteda vremena i novca kasnije). Prikazuje ponašanje sustava iz korisnikove perspektive. Glavna pitanja na koja odgovara su: „Što se opisuje? Tko ima interakciju sa sustavom? Što sudionici mogu učiniti?“. Sudionik je vanjski entitet povezan sa sustavom koji nije dio sustava.

„Dijagram klasa opisuje sustav pomoću klasa i relacija među klasama. Služi za modeliranje statičke strukture sustava. Elementi i relacije između sustava se ne mijenjaju tijekom vremena. Dijagram klasa je ujedno i najčešće korišteni UML dijagram. Primjenjuje se u raznim fazama procesa razvoja softvera. Razina detalja ili apstrakcije je različita u svakoj fazi“ (Mesarić, Šebalj 2016).

„Dijagram objekata prikazuje kako objekti nekog sustava djeluju između sebe. Generira se iz dijagrama klasa i prikaz može biti djelomičan ili cjelovit. Objekti sustava imaju različite vrijednosti atributa. Prikazuju se samo atributi po kojima se pojedinci unutar dijagrama međusobno razlikuju. Ne mogu postojati dva pojedinca s istim svim atributima“ (Mesarić, Šebalj 2016).

„Sekvencijski dijagram opisuje kako će i kojim redoslijedom sustav zapravo napraviti neke zadatke. Naglasak je na vremenskom redoslijedu kojim se odvija interakcija među dijelovima sustava“ (Mesarić, Šebalj 2016).

„Dijagram aktivnosti prikazuje međusobno povezane aktivnosti visoke razine u određenom procesu. Osobito su korisni za modeliranje poslovnih procesa. Naglasak je na jednostavnosti i poslovnim operacijama koje se odvijaju slijedno, jedna za drugom“ (Mesarić, Šebalj 2016).

3.2. Front end development – HTML, CSS i JS

Front end development je praksa konvertiranja podataka u grafičko sučelje, korištenjem HTML-a (Hypertext Markup Language), CSS-a (Cascading Style Sheets) i JavaScripta, na taj način koji će omogućiti pregled i interakciju s podacima.

3.2.1. HTML/CSS

HTML je kratica za HyperText Markup Language. Besplatan je, jednostavan za uporabu i lako se uči zbog čega je opće prihvaćen i popularan na cijelom webu. Hipertekst dokument se stvara, a njegov sadržaj se stvara i oblikuje, pomoću HTML-a Hipertekst dokument se prikazuje u web pregledniku. Zadaća HTML-a jest uputiti web preglednik kako prikazati hipertekst dokument. HTML nije programski jezik. Služi samo za opis hipertekstualnih datoteka, a u njemu se ne mogu izvršiti čak ni najjednostavnije operacije zbrajanja ili oduzimanja dvaju cijelih brojeva. Osnovni građevni elementi svake stranice su tagovi koji opisuju kako će se nešto prikazati u web pregledniku.

CSS je kratica za Cascading Style Sheets. CSS je stilski jezik koji se rabi za opis prezentacije dokumenta napisanog pomoću HTML-a. Kako se web razvijao, prvotno su u HTML ubacivani elementi za definiciju prezentacije (npr. tag <font>). Dovoljno brzo je uočena potreba za stilskim jezikom koji će HTML osloboditi potrebe prikazivanja sadržaja (što je prvenstvena namjena HTML-a) i njegovog oblikovanja (čemu danas služi CSS). Drugim riječima, stil definira kako prikazati HTML elemente. CSS-om se uređuje sam izgled i raspored stranice.

JavaScript je skriptni programski jezik koji se izvršava u web pregledniku na strani korisnika. Uz HTML i CSS, JavaScript čini takozvane temelje weba. Javascript daje korisniku mogučnost interakcije sa web stranicom i njenim elementima, također je esencijalni dio web aplikacija. Kad god web stranica prikazuje više od samo statičnog sadržaja JavaScript je uključen u izvođenje dinamičkih aktivnosti. JavaScript omogućuje i implementaciju takozvanih Application Programing Interfaces (APIs). APIs su setovi koda spremni koji omogućuju implementaciju kompleksnih programa na web stranicu. DOM (Document Object Model) API dozvoljava manipuliranje HTML-a i CSS-a, dodavanje, brisanje i mijenjane HTML-a te dinamičnu dodjelu stilova stranici. Svaki put kada se na stranici pojavi novi element ili pop-up to je odrađeno pomoću DOM-a.

3.3. Web aplikacije

Web aplikacija umjesto izvršne datoteke (.exe) generira skup dokumenata koji se mogu vidjeti s pomoću web preglednika (npr. Google Chrome, Mozilla Firefox i dr.) koji podržavaju html/xhtml/xml. Web aplikaciju sukladno navedenom nije potrebno instalirati na korisničko računalo. Web aplikacija dinamički generira niz html dokumenata koji se prikazuju u web

pregledniku. Ako se žele dodati neki dinamički elementi u korisničko sučelje koriste se skriptni jezici koji se izvode na klijentskoj strani (npr. JavaScript). Aplikacija se može kreirati i korištenjem skriptnih jezika koji se izvode na poslužitelju s pomoću PHP ili ASP skriptnih jezika. Korisnik može podatke unositi s pomoću web formi koje su uključene u stranice.

Prednosti web aplikacija su što rade bez obzira na operativni sustav koji je instaliran na korisnikovom računalu (programer ne mora raditi posebno za windows, MacOS, Unix ili dr. operativni sustav). Nedostatci su vezani za brziinu rada aplikacije odnosno ovisnost o brzini mrežne povezanosti sa poslužiteljem na kojem se nalazi aplikacija (brzina interneta ili intraneta), također se javljaju sigurnosni problemi (zaštita protiv upada, virusa i dr.).

3.4. Primjena JavaScripta

Do Web 2.0 inačice web stranice su bile statične i nije bila moguća interaktivnost ili dodavanje sadržaja. S uvođenjem Web 2.0 stranice su dobile dinamičnost i otvorila se mogućnost za izradu web aplikacija. JavaScript ima veliku ulogu u izradi web aplikacija. JS je ono što daje web aplikaciji interaktivnost i dinamičnost. Pomoću već spomenutog DOM (Document Object Model) API-a omogućeno je manipuliranje HTML-a i CSS-a, dodavanje, brisanje i mijenjane HTML-a te dinamična dodjela stilova stranici.

DOM je moguće iskoristiti na jednostavan način za dohvaćanje elemenata iz HTML-a korištenjem getElementById() ili getElementsByClassName() funkcija. Potrebno je prvo HTML tagu dodati svojstvo id ili class s proizvoljnim nazivom. HTML dokument će sadržavati sljedeću liniju koda:

```
<div id="idName"></div>
```

U JavaScript datoteci dohvaćanje div elementa sa svojstvom id=“idName“ će se odraditi na sljedeći način:

```
document.getElementById('idName');
```

Spremanjem dohvaćenog elementa u varijablu omogućiti će lakše korištenje istoga na više mjesta u JavaScript datoteci:

```
var varijabla = document.getElementById('idName');
```

Korištenjem varijable iz primjera („varijabla“) moći će se dohvatiti div element s tagom „idName“ puno jednostavnije i na više mjesta.

Poslovni informacijski sustavi će zasigurno zahtijevati neku vrstu kalkulacije, npr. skladišno poslovanje vodi brigu o zalihama robe, dodaje se iz zaliha, oduzima, mijenja stanje. JavaScript podržava korištenje jednostavnih ali i kompleksnih matematičkih funkcija koje će olakšati rad sa vođenjem evidencije zaliha.

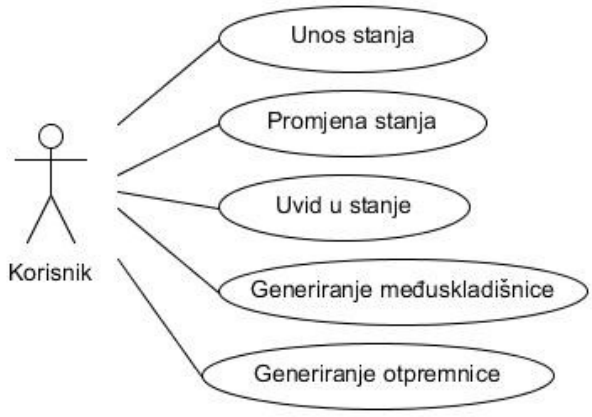
4. Opis istraživanja i rezultati istraživanja

4.1. Modeliranje i izrada web aplikacije za jednostavno skladišno poslovanje

Potrebno je izgraditi jednostavan sustav za skladišno poslovanje koji će omogućiti jednostavan unos podataka o robama koje se kodiraju bar kodom. Kontekstualni okvir: skladište trgovačke tvrtke iz kojeg se roba nadopunjuje u maloprodaji i može se prodavati u veleprodaji. Roba dolazi u transportnim jedinicama (kutija/ paleta) koje su označene bar kodom i ima oznaku broja maloprodajnih jedinica. Na temelju prijemnih dokumenata (tovarnog lista s brojem paleta, računa-otpremnice) i kontrolom roba se uvodi u skladišnu bazu. Barkodirana je transportna jedinica, kutija i pojedinačni proizvod u maloprodaji. Maloprodaja se zadužuje u prodajnim jedinicama a veleprodaja ide u kartonima ili paletama. Program treba omogućiti unos i promjenu stanja količine svakog artikla i generiranje interne međuskladišnice i otpremnice u veleprodaji. Sustav mora osigurati uvid u stanje skladišta. U radu će biti prikazani primjeri UML dijagrama za navedeni problem. Nadalje će biti prikazan primjer JavaScript koda potrebnog za izvođenje unosa artikla i pregled unesenih artikala.

4.1.1. Dijagram slučajeva korištenja

Slika 1. opisuje sudionika aplikacije za skladišno poslovanje te navodi sudionikove mogućnosti u interakciji s aplikacijom. Omogućen mu je unos stanja količine artikla, promjena stanja količine artikla, uvid u stanje pojedinog artikla te generiranje međuskladišnice i otpremnice.

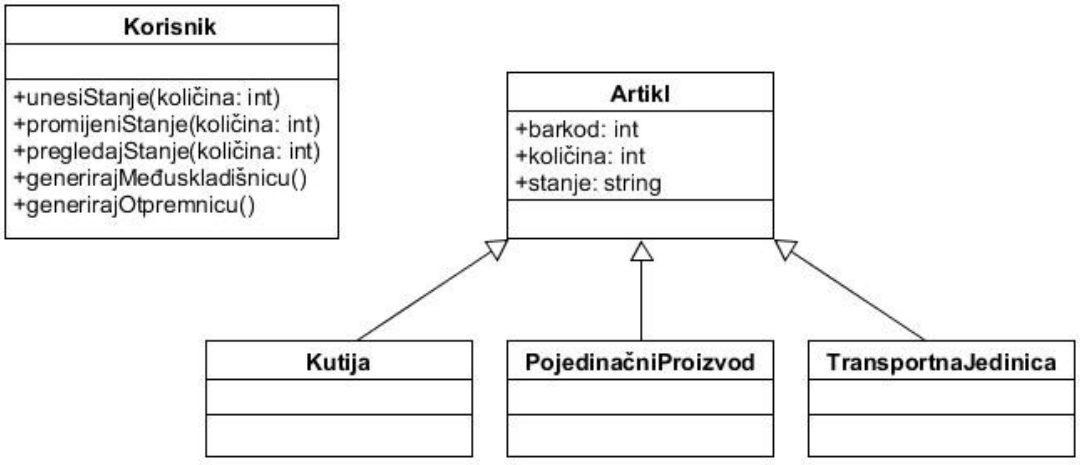


Slika 1. Dijagram slučajeva korištenja

Izvor: izrada autora

4.1.2. Dijagram klasa

Slika 2. prikazuje klase u aplikaciji. Klasa „Korisnik“ nema atribute ali ima mogućnost izvršavanja funkcija „unesiStanje()“, „promijeniStanje()“, „pregledajStanje()“, „generirajMeđuskladišnicu()“ i „generirajOtpremnicu()“. Prve tri funkcije će utjecati na promjenu ili pregled atributa klase „Artikl“. Klasa „Artikl“ ima tri atributa: barkod koji je jedinstven za svaki proizvod, količina i stanje (raspoloživo/naručeno). Klase „Kutija“, „PojedinačniProizvod“ i „TransportnaJedinica“ nasljeđuju sva svojstva klase „Artikl“.

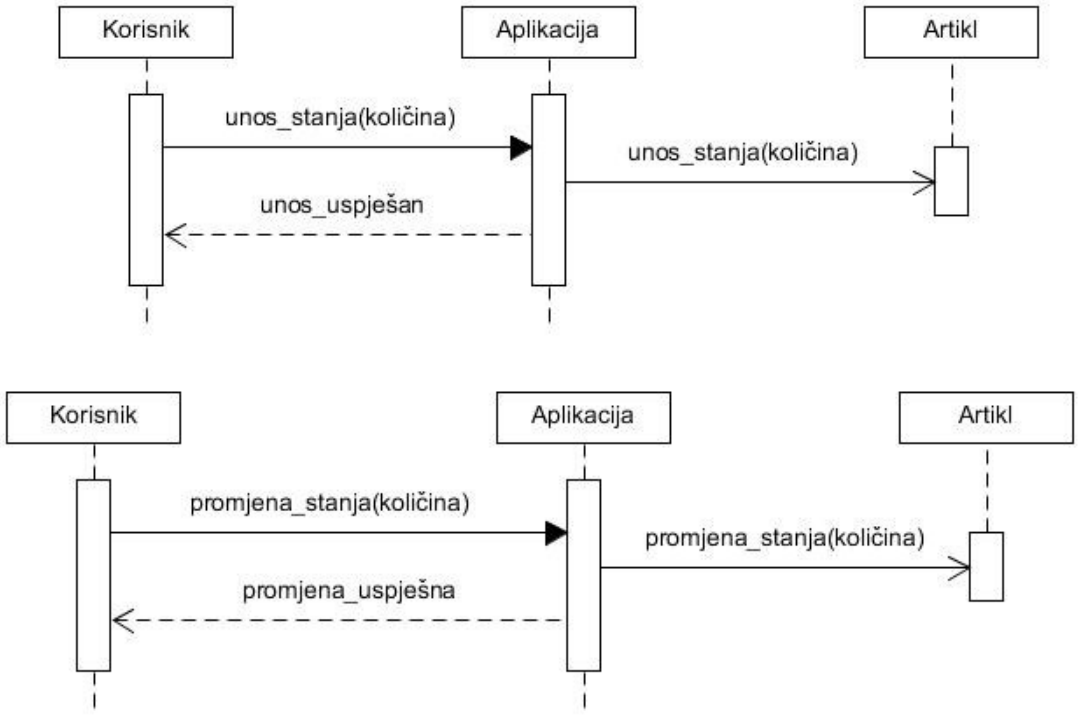


Slika 2. Dijagram klasa

Izvor: izrada autora

4.1.3. Sekvencijski dijagram

U sekvencijskom dijagramu prikazanom slikom 3. može se vidjeti kako i kojim redoslijedom sustav odrađuje pojedini zadatak, u ovom slučaju unos i promjenu stanja robe. Korisnik inicira unos ili promjenu u aplikaciji, nakon toga aplikacija sprema unesene podatke u klasu artikl te vraća korisniku potvrdu da je unos uspješan.



Slika 3. Sekvencijski dijagram

Izvor: izrada autora

4.1.4. Izrada web aplikacije

Polazeći od prethodnih modela u radu će se izraditi jednostavna aplikacija korištenjem Node.js-a, odnosno njegovog frameworka pod nazivom Express. Express je minimalistički i fleksibilni Node.js framework za izradu web aplikacija koji pruža robustan set značajki za web ali i mobilne aplikacije. Za generiranje dinamičkog HTML-a i implementaciju samog JavaScript koda unutar istog koristit će se EJS (Embedded JavaScript templating) koji, kako mu i ime navodi, omogučuje ugrađivanje i izradu JavaScript predložaka u HTML kod. Kako bi se uneseni podaci o proizvodima pohranili poslužit će MongoDB baza podataka u kombinaciji sa mongoose npm-om (node package module) za modeliranje baze i generiranje upita prema bazi podataka kao što su pretraživanje, uređivanje postojećih unosa te u konačnici brisanje podataka. Za postavljanje i pokretanje te testiranje aplikacije u ovom primjeru će se koristiti GitBash konzola.

Postavljanje Express aplikacije započinje navigacijom u novo kreiranu mapu za projekt te upisivanjem sljedeće komande:

```
$ npm install express –save
```

Navedena komanda kreira potrebne temelje Express aplikacije u mapi projekta na kojima kreće izrada web aplikacije. Idući korak je kreiranje primarne datoteke koja će pokretati aplikaciju. U ovom primjeru datoteka nosi naziv „app.js“. Na početku app.js datoteke se navode svi zahtjevi aplikacije, odnosno npm-ovi koji će se koristiti.

```
const express = require('express');
const path = require('path');
const mongoose = require('mongoose');
const ejsMate = require('ejs-mate');
const session = require('express-session');
const flash = require('connect-flash');
const ExpressError = require('./utils/ExpressError');
const methodOverride = require('method-override');
```

Osim Express i Mongoose npm-ova ostali su proizvoljni i u ovom slućaju instalirani kako bi olakšali izradu ili pružili dodatne funkcionalnosti aplikaciji. Nakon instalacije npm-ova potrebno je napraviti konekciju s MongoDB bazom podataka.

```
mongoose.connect('mongodb://localhost:27017/sklado', {
    useNewUrlParser: true,
    useCreateIndex: true,
    useUnifiedTopology: true,
    useFindAndModify: false
});

const db = mongoose.connection;
db.on('error', console.error.bind(console, 'connection error:'));
db.once('open', () => {
    console.log('Database connected');
});
```

console.log('Database connected'); ispisuje pri pokretanju aplikacije u konzolu „Database connected“ što služi kao potvrda da je baza podataka uspješno povezana.

Pokretanje Express servera odrađuje funkcija „express();“ koja se nerijetko sprema u proizvoljnu varijablu.

```
const app = express();
```

Aplikacija će pomoću posebne metode ugrađene u Express slušati zahtjeve. Potrebno ju je pozvati unutar app.js datoteke.

```
app.listen(3000, () => {
    console.log('Serving on port 3000');
});
```

3000 označava port na kojemu će se pokretati aplikacija lokalno, a upisivanjem adrese „localhost:3000“ u preglednik moguće je pristupiti aplikaciji. Aplikaciju je prethodno potrebno pokrenuti naredbom „$ node app.js“ u GitBash konzoli.

Moongose omogućuje kreiranje takozvane sheme za bazu podataka. Aplikacija će koristiti jednu shemu za proizvod koja će primati numeričku vrijednost za barkod, string tip podatka za naziv proizvoda i numeričku vrijednost za količinu proizvoda. Za shemu se kreira posebna datoteka i sprema se u „models“ mapu unutar mape projekta. Datoteka product.js sadrži shemu u nastavku i izvozi se na kraju module.exports linijom koda što omogučuje njeno korištenje u ostalim datotekama projekta.

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

const ProductSchema = new Schema({
    barcode: Number,
    name: String,
    quantity: Number
});

module.exports = mongoose.model('Product', ProductSchema);
```

Za uvoz sheme i njeno korištenje u ostalim datotekama potrebno je navesti sljedeći zahtjev.

```
const Product = require('../models/product');
```

Postavljanje ruta koje će preusmjeravati na stranice sa zasebnim funkcionalnostima aplikacije odradit će se kreiranjem „routes“ mape. U ovom primjeru kreirana je datoteka „products.js“ unutar navedene mape i u njoj se nalaze svi zahtjevi za preusmjeravanje. Stranicu za unos novog proizvoda poziva se na sljedeći način:

```
router.get('/new', (req, res) => {
    res.render('new');
});
```

res.render('new'); funkcija poziva „new.ejs“ datoteku iz mape „views“ u kojoj se nalaze sve strukture stranica koje se prikazuju korisniku. U mapi „views“ je mapa „layouts“ u koju je spremljena datoteka „boilerplate.ejs“. Boilerplate datoteka sadrži strukturu svake stranice, uključuje poveznice na pripadajuće stilove i skripte.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" re
l="stylesheet" integrity="sha384-
EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC" crossorigin="anonymous">

    <title>Sklado</title>
</head>

<body class="d-flex flex-column vh-100">
    <%- include('../partials/navbar')  %>
    <main class="container mt-5">
      <%- include('../partials/flash')  %>
      <%- body %>
    </main>
    <%- include('../partials/footer')  %>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.
js" integrity="sha384-
MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM" crossorigin="anonymous"><
/script>

    <script src="/javascripts/validateForms.js"></script>
</body>
</html>
```

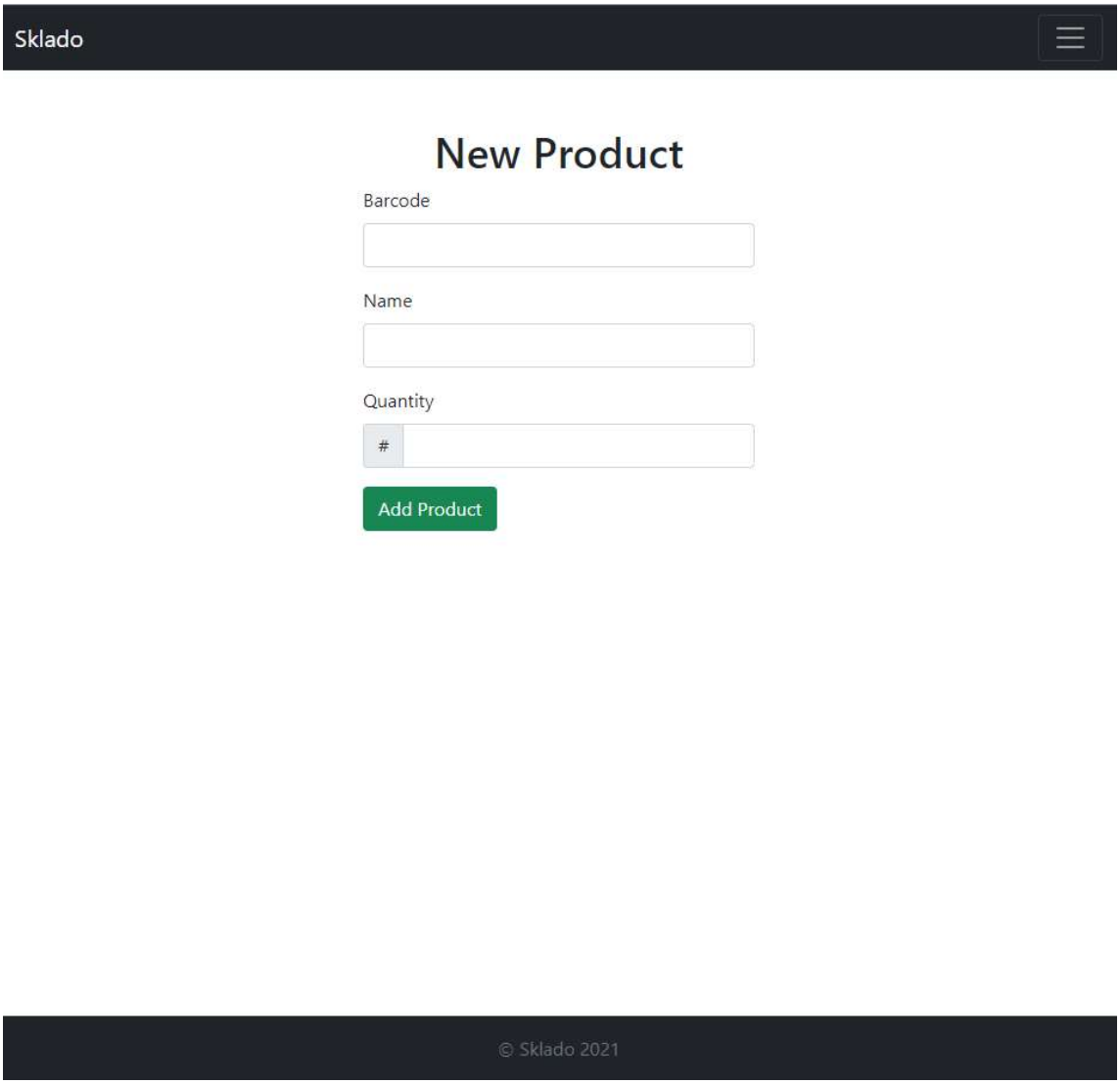
EJS sintaksa omogučava pozivanje navedene strukture na svakoj stranici korištenjem jedne linije koda.

```
<% layout('layouts/boilerplate') %>
```

Bitan dio „new“ stranice je forma koja omogučuje unos novog proizvoda, barkod, naziv i početnu količinu istog.

```
<form action="/products" method="POST" novalidate class="validated-form">
            <div class="mb-3">
                <label class="form-label" for="barcode">Barcode</label>
                <input class="form-
control" type="text" id="barcode" name="product[barcode]" required>
            </div>
            <div class="mb-3">
                <label class="form-label" for="name">Name</label>
                <input class="form-
control" type="text" id="name" name="product[name]" required>
            </div>
            <div class="mb-3">
                <label class="form-label" for="quantity">Quantity</label>
                <div class="input-group mb-3">
                    <span class="input-group-text" id="quantity-label">#</span>
                    <input class="form-
control" type="number" id="quantity" name="product[quantity]" required>
                </div>
            </div>
            <div class="mb-3">
                <button class="btn btn-success" type="submit">Add Product</button>
            </div>
</form>
```

Preglednik će formu prikazati kao što je prikazano slikom 4.



Slika 4. Stranica za unos novog proizvoda

Podnošenjem zahtjeva iz forme poziva se spremanje novog proizvoda u bazu.

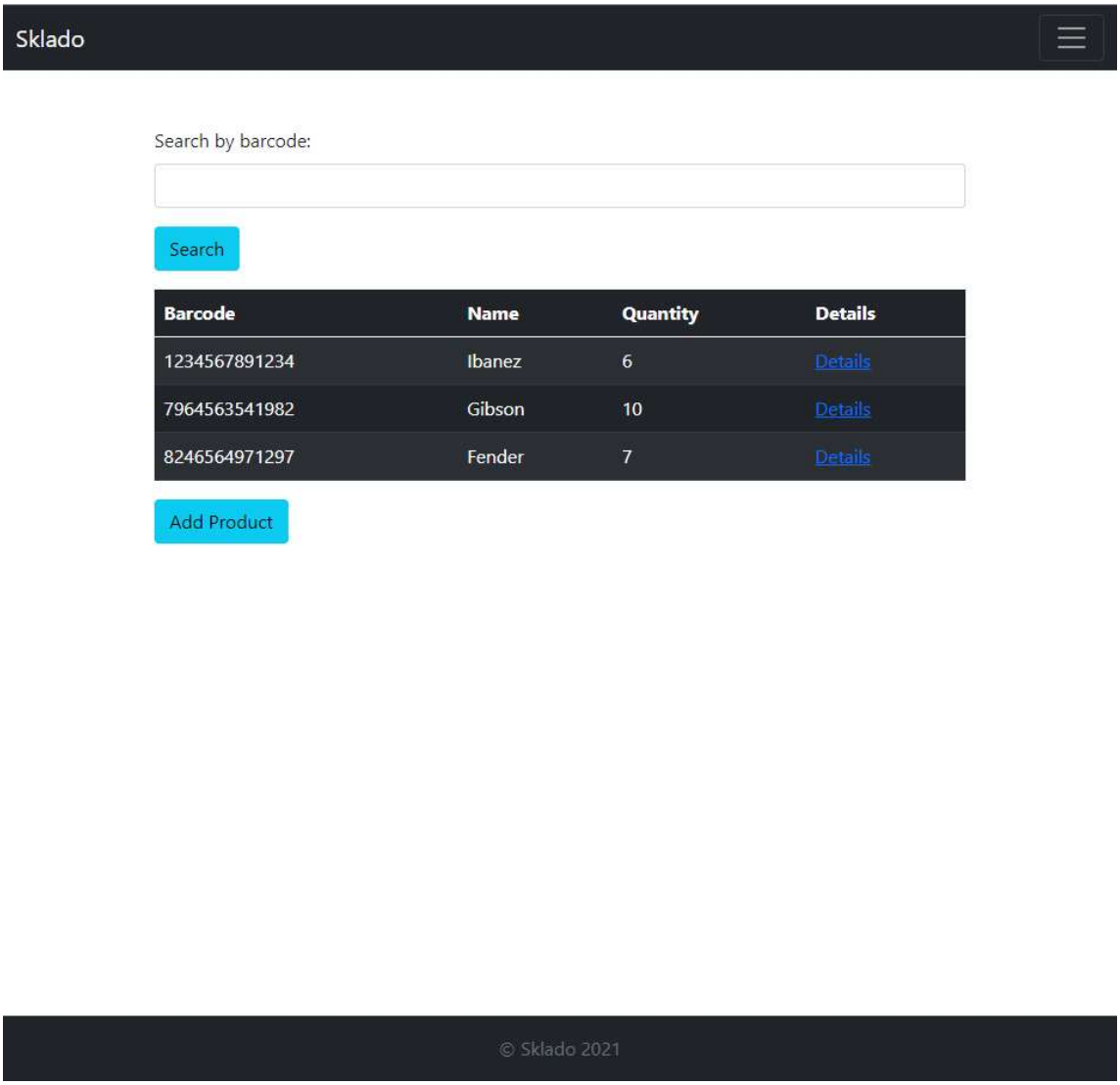
```
router.post(
    '/',
    validateProduct,
    catchAsync(async (req, res, next) => {
        const product = new Product(req.body.product);
        await product.save();
        req.flash('success', 'New product added successfully!');
        res.redirect('products');
    })
);
```

Potrebno je prikazati unesene proizvode iz baze podataka. GET ruta ce pronaći sve proizvode u bazi podataka, a ako je u zahtjevu za prikaz unesen atribut barkod prelistat će bazu podataka i prikazati samo proizvode koji imaju odgovarajući barkod. Korištenjem jednostavne EJS sintakse unutar HTML strukture generirat će se tablica koja će proizvode i prikazati.

```
router.get(
    '/',
    catchAsync(async (req, res, next) => {
        const { barcode } = req.query;
        if (barcode) {
            const products = await Product.find({ barcode });
            res.render('index', { products, barcode });
        } else {
            const products = await Product.find({});
            res.render('index', { products, barcode });
        }
    })
);
```

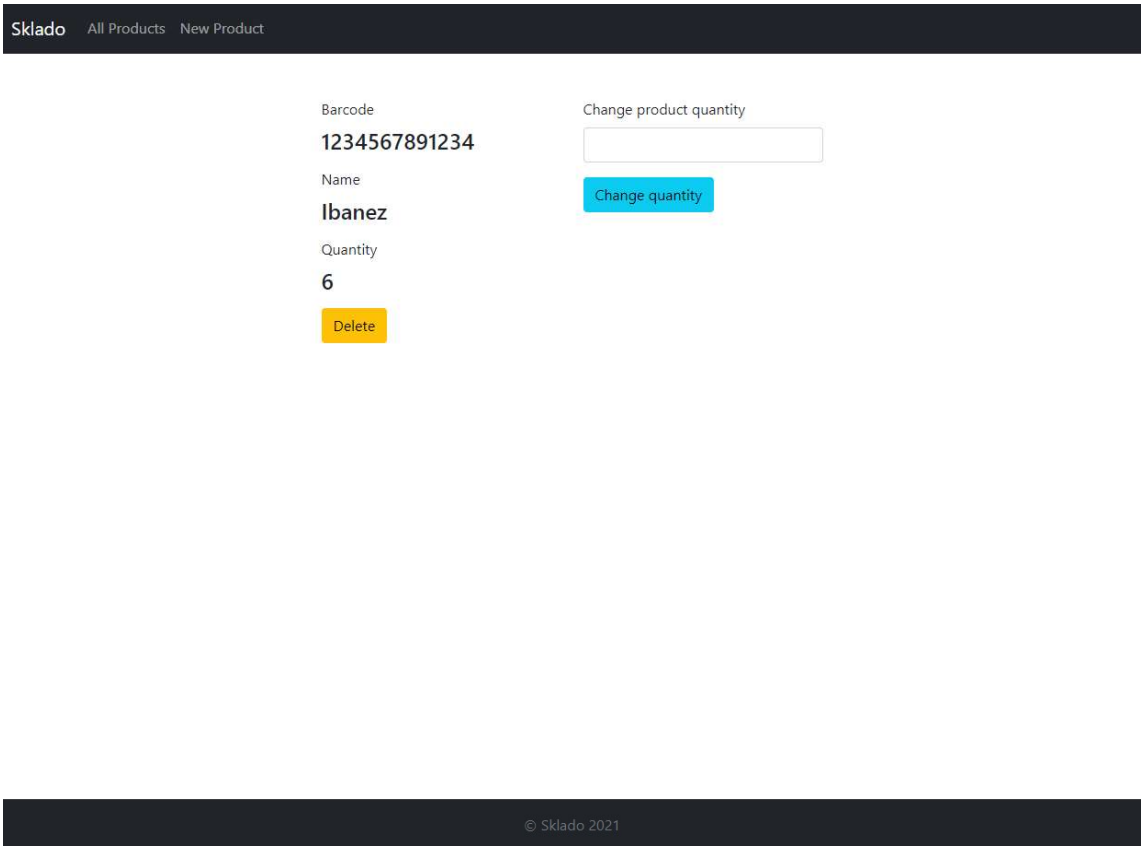
```
<table class="table table-dark table-striped">
    <thead>
        <tr>
            <th scope="col">Barcode</th>
            <th scope="col">Name</th>
            <th scope="col">Quantity</th>
            <th scope="col">Details</th>
        </tr>
    </thead>
    <tbody>
        <% for( let product of products) { %>
            <tr>
                <td><%= product.barcode %></td>
                <td><%= product.name %></td>
                <td><%= product.quantity %></td>
                <td><a href="/products/<%= product._id %>">Details</a></td>
            </tr>
        <% } %>
    </tbody>
</table>
```

Generirana tablica je prikazana slikom 5.



Slika 5. Prikaz proizvoda iz baze podataka

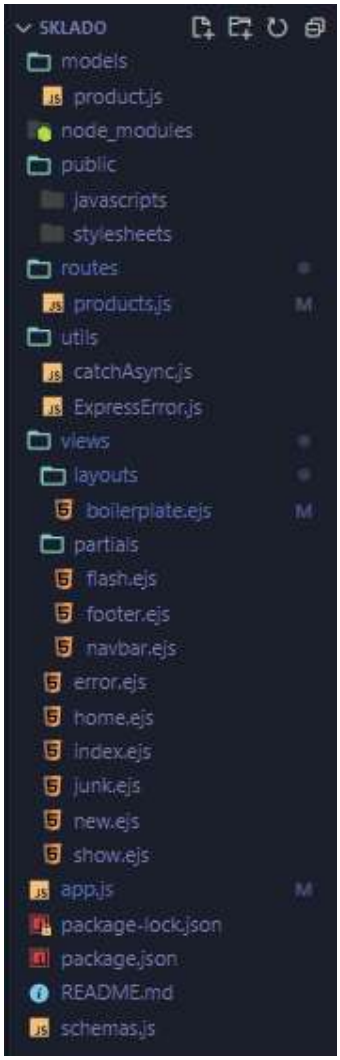
Detaljni prikaz pojedinačnog proizvoda prikazuje barkod, naziv i količinu proizvoda te omogučava promjenu količine i brisanje proizvoda. Promjena količine će dodati unesenu vrijednost iz forme trenutnoj količini proizvoda, rezultat će biti nova količina koja se pohranjuje u bazu podataka.



Slika 6. Detaljni prikaz proizvoda i promjena količine

```
router.put(
    '/:id',
    catchAsync(async (req, res, next) => {
        const { id } = req.params;
        const productQuantity = await Product.findById(req.params.id);
        const formQuantity = await req.body.product;
        const newQuantity = productQuantity.quantity + Number(formQuantity.quantity);
        if (newQuantity >= 0) {
            const product = await Product.findByIdAndUpdate(id, { $inc: { ...req.body.produ
ct } });
            req.flash('success', 'Product updated successfully!');
            res.redirect(`/products/${product._id}`);
        } else {
            req.flash('error', 'Product cannot be found!');
            return res.redirect(`/products/${product._id}`);
        }
    })
);
```

U konačnici struktura mape projekta je prikazana slikom 7. iz koje su vidljive sve glavne mape projekta i pripadajuće datoteke.



Slika 7. Struktura mape projekta

4.1.5. Osvrt na izradu web aplikacije

Izradom jednostavne web aplikacije zadovoljen je uvjet pohranjivanja proizvoda u bazu podataka, pretraživanja po barkodu i izmjene količine odnosno stanja proizvoda na skladištu. Potrebno je navesti kako jednostavna aplikacija ima određena ograničenja. Poželjno pretraživanje proizvoda po barkodu bi se moglo odrađivati skeniranjem barkoda skenerom. Povezivanje takvog sustava sa web aplikacijom iziskuje složenije programsko rješenje nego što je izgrađeno u primjeru ovog završnog rada. Za eventualne nadogradnje aplikacije je predviđeno implementiranje povijesti transakcija. Aplikacija bi trebala bilježiti vrijeme izmjene količine odnosno stanja proizvoda i u detaljnom prikazu proizvoda također prikazati povijest. Kada bi se takav prikaz filtrirao po tjednima, mjesecima ili godinama dobio bi se detaljni prikaz

i podaci potrebni za statističke izračune. To bi omogučilo bolje predviđanje i lakše donošenje odluke u kojem trenu i koliko proizvoda optimalno naručiti.

5. Zaključak

Doba digitalizacije u svijetu poslovanja stavilo je naglasak na prikupljanje, čuvanje, obradu i raspodjelu podataka i informacija. Informacijski sustav je postao neizbježni dio poslovanja i uvelike je ubrzao i olakšao proces donošenja bitnih odluka za poduzeće. Uspješno poslovanje ovisi o upravljanju poslovima, ljudskim potencijalima, financijskom imovinom, kupcima i dobavljačima na osnovu kvalitetnih podataka i informacija.

Web aplikacija je pogodnija za manja poduzeća jer se može koristiti bez obzira na operacijski sustav koji poduzeće koristi i ne iziskuje troškove izrade i prilagodbe za određeni operacijski sustav. Zbog sigurnosnih nedostataka poput mogućnosti upada i virusa ne preporuča se za veća poduzeća. Obzirom na navedeno ekstremno programiranje kao metoda razvoja softvera je pogodna za izradu web aplikacija.

Izradi web aplikacije pristupiti će se prvotno detaljnom obradom zahtjeva i izradom modela koristeći se UML dijagramima. UML dijagramima će se prikazati logika iza aplikacije, sudionici i njihove interakcije unutar sustava. Time će se olakšati samo programiranje i pisanje koda. JavaScript kao skriptni jezik je svestran i pruža gotovo neograničene mogućnosti te stoga uz kvalitetnu razradu zahtjeva i izradu modela moguće je na brz i djelotvoran način osposobiti web aplikaciju za razne upotrebe.

Literatura

[1] Krpan, Lj., Maršanić, R., Jedvaj V. (2014). *Upravljanje zalihama materijalnih dobara i skladišno poslovanje u logističkoj industriji*

[2] Sekso, M. (2011). *Uloga informacijskih sustava u upravljanju materijalima i zalihama.* Zbornik radova Međimurskog veleučilišta u Čakovcu, 2 (1), 125-133.

[3] Zekić-Sušac M. (2013). *Pristup izradi poslovnih aplikacija - Agilne metode razvoja aplikacija.* Dostupno na: http://www.efos.unios.hr/razvoj-poslovnihaplikacija/wp-content/uploads/sites/228/2013/04/RPA_P1_Agilne-metode1.pdf. [Pristupljeno: 03.04.2020.]

[4] Mesarić, J., Šebalj, D. (2016). *Oblikovanje i implementacija informacijskih sustava – nastavni materijali*

[5] *MDN web docs* https://developer.mozilla.org/en-US/ [pristupljeno: 15.07.2020.]

Slike