Web sustav za praćenje proizvodnog ciklusa tvrtke za preradu kože

Lučić, Tomislav

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

Sveučilišni studij

Web sustav za praćenje proizvodnog ciklusa tvrtke za preradu kože

Završni rad

Tomislav Lučić

Osijek, 2023.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 01.09.2023.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime Pristupnika:	Tomislav Lučić
Studij, smjer:	Programsko inženjerstvo
Mat. br. Pristupnika, godina upisa:	R4531, 27.07.2020.
OIB Pristupnika:	16015909938
Mentor:	prof. dr. sc. Goran Martinović
Sumentor:	,
Sumentor iz tvrtke:	
Naslov završnog rada:	Web sustav za praćenje proizvodnog ciklusa tvrtke za preradu kože
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rad:	U teorijskom dijelu završnog rada potrebno je proučiti i opisati proizvodni ciklus tvrtke za preradu kože s gledišta otkupa, postupka prerade i prodaje. Također, treba analizirati postojeća slična rješenja i definirati moguće načine poboljšanja navedenih procesa. Nadalje, treba definirati korake postupka stvaranja i praćenja naloga za otkup, preradu i prodaju kože s gledišta vremenskih i prostornih ograničenja i kapacitete tvrtke, odnosno prerade i voznih jedinica. Uz to, koristeći prikladan postupak raspoređivanja i preraspoređivanja treba omogućiti
Prijedlog ocjene završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	01.09.2023.
Datum potvrde ocjene od strane Odbora:	08.09.2023.

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

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

IZJAVA O ORIGINALNOSTI RADA

Osijek, 10.09.2023.

Ime i prezime studenta:	Tomislav Lučić
Studij:	Programsko inženjerstvo
Mat. br. studenta, godina upisa:	R4531, 27.07.2020.
Turnitin podudaranje [%]:	8

Ovom izjavom izjavljujem da je rad pod nazivom**: Web sustav za praćenje proizvodnog ciklusa tvrtke za preradu kože**

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

i sumentora ,

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

Potpis studenta:

SADRŽAJ

1. UVOD 1

1.1. Zadatak završnog zadatka 1

2. PRAĆENJE PROIZVODNOG CIKLUSA TVRTKE, POSTOJEĆA RJEŠENJA I STANJE U PODRUČJU 3

2.1. Praćenje proizvodnog ciklusa tvrtke 3

2.2. Proizvodni ciklus tvrtke za preradu kože 4

2.3. Stanje u području i pregled postojećih rješenja za praćenje proizvodnog ciklusa tvrtke 5

2.3.1. Stanje u području 5

2.3.2. Postojeća slična rješenja 6

3. ZAHTJEVI, MODEL I GRAĐA SUSTAVA ZA PRAĆENJE PROIZVODNOG CIKLUSA TVRTKE ZA PRERADU KOŽE 9

3.1. Funkcionalni zahtjevi na web sustav 9

3.2. Nefunkcionalni zahtjevi na web sustav 9

3.3. Model sustava 10

3.3.1. Parametri bitni za praćenje proizvodnog ciklusa tvrtke za preradu kože 10

3.3.2. Raspoređivanje 11

3.3.3. Preraspoređivanje 12

3.3.4. Primjena raspoređivanja, preraspoređivanja i upozoravanja 13

3.4. Građa sustava, dijagram rada sustava i arhitektura baze podataka 14

3.4.1. Građa web sustava 14

3.4.2. Dijagram tijeka rada sustava 15

3.4.3. Arhitektura baze podataka 17

4. PROGRAMSKO RJEŠENJE WEB SUSTAVA 19

4.1. Korišteni programski jezici, tehnologije i razvojne okoline 19

4.1.1. PostgreSQL 19

4.1.2. C# .NET 19

4.1.3. Postman 19

4.1.4. REST API 20

4.1.5. HTML 20

4.1.6. CSS 20

4.1.7. React.js 21

4.2. Programsko rješenje web sustava 21

4.2.1. Implementacija prijave i registracije korisnika 21

4.2.2. Dodavanje naloga, klijenata i dobavljača 24
4.2.3. Dohvaćanje naloga, klijenata i dobavljača 26
4.2.4. Promjena statusa naloga 28
4.2.5. Obavijesti o promjeni statusa i nepoštivanja roka plaćanja . **Pogreška! Knjižna oznaka nije definirana.**
4.2.6. Odjava korisnika iz sustava 32

5. NAČIN RADA APLIKACIJE, ISPITIVANJE I ANALIZA REZULTATA 33

5.1. Način rada aplikacije 33
5.1.1. Prijava i registracija korisnika 33
5.1.2. Klijentska nadzorna ploča 34
5.1.3. Administratorska nadzorna ploča 36
5.1.4. Prikaz obavijesti 37
5.1.5. Radni nalozi 38
5.1.6. Nalog za proizvodnju 40
5.1.7. Fakture 41
5.1.8. Klijenti i dobavljači 42
5.1.9. Prikaz registriranih korisnika 43

5.2. Ispitivanje web aplikacije 43
5.2.1. Provjera kapaciteta skladišta prilikom stvaranja radnog naloga 43
5.2.2. Provjera obavijesti prilikom promjene statusa naloga 44
5.2.3. Provjera obavijesti prilikom nepoštivanja roka plaćanja 45

5.3. Analize rezultata ispitivanja i testiranja 45
5.3.1. Analiza prvog testnog slučaja 45
5.3.2. Analiza drugog testnog slučaja 46
5.3.3. Analiza trećeg testnog slučaja 46

6. ZAKLJUČAK 47

LITERATURA 48

ŽIVOTOPIS 52

SAŽETAK 50

ABSTRACT 51

PRILOZI 52

1. UVOD

Proizvodnja je ekonomska djelatnost koja ima zadaću pretvarati sirovine i resurse u finalne proizvode spremne za ispunjavanje potreba kupaca. Iz navedenog je jasno da postoji povezanost između proizvodnje i obrade resursa. Obrada resursa obuhvaća različite aktivnosti i procese koji transformiraju određeni resurs u oblik proizvoda tražen od strane klijenata. Sam proces obrade može biti dugotrajan i složen, stoga je od iznimne važnosti pojednostaviti svaku fazu i cjelokupni tijek procesa putem uvođenja parametara koji se koriste za aktivno praćenje procesa obrade.

Cilj ovog završnog zadatka je stvoriti web sustav određen za tvrtku koja se bavi preradom kože kako bi se cijeli proces pojednostavio. Klijentu će biti omogućeno praćenje faktura, a administratoru svih faza procesa, dobavljača, klijenata, korisnika te stanja u skladištu.

U drugom poglavlju će biti detaljno razmatran pojam proizvodnog ciklusa. Prvo će se općenito pojasniti taj pojam, a potom će se posebno istražiti njegova primjena u tvrtki koja se bavi preradom kože. Također će se izvršiti pregled trenutačno dostupnih rješenja u toj sferi. Treće poglavlje će uspostaviti definiciju funkcionalnih i nefunkcionalnih zahtjeva te će pružiti dubinski uvid u model sustava. Ovdje će biti opisani ključni parametri koji su od velikog značaja za praćenje proizvodnog ciklusa. Poglavlje će također obuhvatiti temeljne elemente sustava, uključujući strukturu i dijagram rada sustava te arhitekturu baze podataka. Četvrto poglavlje će se posvetiti tehnologijama programiranja koje su korištene za izgradnju sustava. Ovdje će biti pružen prikaz programskog koda koji omogućuje ostvarivanje svih funkcionalnih zahtjeva sustava. U posljednjem poglavlju prikazuje se način rada sustava te se provodi ispitivanje preciznosti i pouzdanosti sustava kroz analizu tri testna slučaja.

1.1. Zadatak završnog zadatka

U teorijskom dijelu završnog rada potrebno je proučiti i opisati proizvodni ciklus tvrtke za preradu kože s gledišta otkupa, postupka prerade i prodaje. Također, treba analizirati postojeća slična rješenja i definirati moguće načine poboljšanja navedenih procesa. Nadalje, treba definirati korake postupka stvaranja i praćenja naloga za otkup, preradu i prodaju kože s gledišta vremenskih i prostornih ograničenja i kapacitete tvrtke, odnosno prerade i voznih jedinica. Uz to koristeći prikladan postupak raspoređivanja i preraspoređivanja treba omogućiti povećanje učinkovitosti proizvodnje na temelju pravovremenog obavješćivanja odgovorne osobe. Na temelju navedenih specifikacija treba definirati funkcionalne i nefunkcionalne zahtjeve, model, arhitekturu i dizajn web aplikacije na strani korisnika i poslužitelja web sustava koji će omogućiti stvaranje i praćenje naloga, te prikaz stanja procesa na nadzornom sučelju. Opisani web sustav s bazom podataka treba

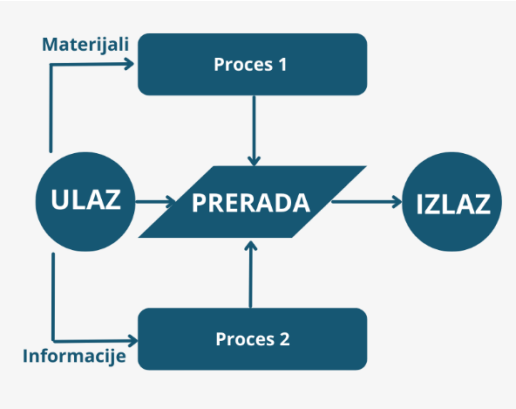
ostvariti koristeći prikladne programske jezike, tehnologije i razvojne okvire, te ga ispitati na odgovarajućim ulaznim podacima i slučajevima korištenja.

2. PRAĆENJE PROIZVODNOG CIKLUSA TVRTKE, POSTOJEĆA RJEŠENJA I STANJE U PODRUČJU

U ovom poglavlju se istražuje opći proizvodni ciklus tvrtke, a nakon toga slijedi pojašnjenje specifičnog proizvodnog ciklusa tvrtke za preradu kože. Također, dan je pregled postojećih sličnih rješenja te uvid u trenutno stanje u području.

2.1. Praćenje proizvodnog ciklusa tvrtke

Prema [14], proizvodnja je prostorno i materijalno određeni proces koji se općenito odnosi na iskorištavanje, preradu te transport proizvoda kako bi se zadovoljile potrebe klijenta te tako se uspostavlja odnos između proizvođača i klijenta. Osnovni čimbenici proizvodnje su uloženi rad, osoblje, roba koja se prerađuje te cilj stvaranja dobara spremnih za transport i prodaju. Proizvodnja se s obzirom na količinu dostupnih jedinica za preradu i osoblja dijeli na: pojedinačnu, masovnu i procesnu proizvodnju. Za pravilnu proizvodnju proizvoda potreban je dobro osmišljen proizvodni ciklus. Proizvodni proces ili ciklus osnova je svake tvrtke a obuhvaća skladištenje ulaznih sredstava te aktivnosti do skladištenja gotovih proizvoda. Kako se proizvodni ciklus sastoji od više procesa kao što su transportni, organizacijski i industrijski, proizvodni proces predstavlja elementarni spoj industrije, ljudskih resursa i ekonomije. Na slici 2.1 nalazi se opći shematski oblik jednostavnog proizvodnog ciklusa s modeliranim ulaznim i izlaznim jedinicama.



Slika 2.1. Shematski prikaz proizvodnog ciklusa tvrtke

Postoji velik broj načina i opcija koje se mogu odabrati kako bi se došlo od ulazne sirovine do konačnog proizvoda. Prije odabira potrebno je detaljno istražiti jesu li dostupni odgovarajući resursi, tehnološki alati, materijali i osoblje. Pri istraživanju, poželjno je uzeti u obzir i dijelove procesa koji ga mogu usporiti, kao što su nedostatak radne snage u određenim trenucima, kašnjenje dostavnog materijala te kvarovi na različitim strojevima. Cilj je izabrati onaj način koji nakon istraživanja pokazuje najveću efikasnost pri dolasku do konačnog proizvoda. Nakon pomno

odabranog proizvodnog ciklusa, potrebno ga je pratiti korištenjem najprikladnijeg sustava za praćenje ovisno o proizvodnoj grani tvrtke. Praćenje proizvodnog ciklusa je u pravilu kontrola procesa s ciljem postizanja sigurnosti jesu li sve aktivnosti regulirane s obzirom na predviđene regulacije. Drugim riječima, to je strategija korištena za analizu i mjerenje kroz proces proizvodnje. Od ulaza do izlaza, proces praćenja omogućuje prikaz performansi svakog koraka procesa u realnom vremenu. Dobro izveden proces praćenja koji pokazuje pozitivne i negativne strane proizvodnog ciklusa može određenoj tvrtki omogućiti veliku prednost nad tržišnom konkurencijom.

2.2. Proizvodni ciklus tvrtke za preradu kože

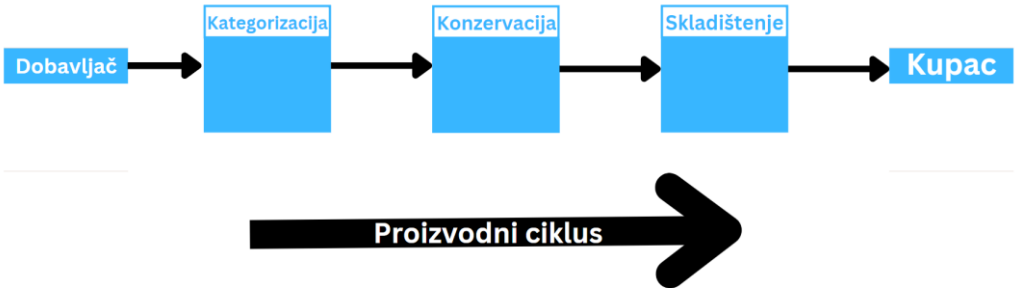
Model proizvodnog procesa tvrtke koja se bavi preradom sirovina, odnosno kože je vrlo sličan procesu shematski prikazanom na slici 2.1 gdje su jedine razlike ulazni čimbenici te razni procesi prerade u konačni proizvod. Sve počinje od ulaznih resursa, dobavljači su mesari i klaonice koje kolju stoku te im ostaje koža koju tvrtke za preradu kože otkupljuju. U tvornicama kože prikupljaju se kože goveda, ovaca, teladi, bikova i krava. Sirovine se sortiraju po veličini i vrsti životinje te slažu u pakiranja ovisno o ulaznim parametrima. Pakirane kože se slažu na palete koje mogu biti između 20 i 30 tona ovisno o postavljenim zahtjevima narudžbe i potom se isporučuju kupcima. Nakon što stignu u tvrtku, sirovine najprije moraju proći kroz određene procese kako bi bile spremne za isporuku. Prvi proces pri dobitku sirovine od dobavljača je kategorizacija. Sirova koža se sortira po vrsti, težinskoj kategoriji i klasi. Prema [1], na slici 2.2 prikazano je kategoriziranje vrsta životinjskih koža s obzirom na klasu kvalitete kojoj pripadaju u obliku tablice.

VRSTA KOŽE	KLASA
Teleća koža	I.-II.
Teleća koža	III.-IV.
Juneća koža	I.-II.
Juneća koža	III.-IV.
Kravlja koža	I.-II.
Kravlja koža	III.-IV

2.2 Primjer kategorizacije koža u klase

Nakon kategorizacije potrebno je orezati sve dijelove kože koje se ne mogu upotrijebiti kožarskoj preradi, a to su većinom rogovi, uši i papci. Kad se ohlade na prikladnu temperaturu okoliša, kože se važu i postaju standardnom ulaznom veličinom koja se dalje obrađuje. Prema [1], dobivene

sirove kože potrebno je konzervirati soljenjem. Postupak konzerviranja koža odvija se procesom koji se naziva salamurenje. Pri ovom procesu sirove kože se 24-48 sati obrađuju salamurom, odnosno zasićenom otopinom kuhinjske soli. Tako kože se potpuno zahvate solju te vrlo dobro isperu. Nakon otkapljivanja slažu se zapakirane na hrpu te nakon najprikladnijeg načina skladištenja ovisno o vrsti kože spremne su za otpremu. Za skladištenje sirovih koža potrebne su tamne, hladne prostorije sa slabim strujanje zraka uz nerijetku potrebu dosoljavanja kože. Na slici 2.3 može se primijetiti kako je proizvodni ciklus tvrtke za preradu kože vrlo sličan, odnosno identičan općem procesu proizvodnje, nepovratno se kreće od dobavljača do kupca.



Slika 2.3. Prikaz proizvodnog ciklusa tvrtke za preradu kože

2.3. Stanje u području i pregled postojećih rješenja za praćenje proizvodnog ciklusa tvrtke

2.3.1. Stanje u području

Proizvodni ciklus u određenim tvrtkama može biti dugotrajan i složen proces s velikim brojem parametara koje treba uzet u obzir pri stvaranju konačnog proizvoda. Prema [16], za poboljšanje performansi i optimizacije cjelokupnog procesa stvaraju se sustavi koji omogućuju detaljan prikaz svih dijelova procesa. Kvalitetan sustav može analizirati i dokumentirati korake, identificirati područja koja zahtijevaju poboljšanja te pratiti stanje skladišta u stvarnom vremenu. Takav sustav omogućuje generiranje analitičkih izvješća s ciljem donošenja prikladnih odluka i planiranja budućih aktivnosti. Prema [17], trenutna evolucija i dostupnost tehnologija interneta stvari (IoT, eng. *Internet of Things*) i računalom potpomognutih sustava (*Cyber-Physical Systems* - CPS) potiču novu industrijsku revoluciju, gdje se tehnologije i procesi proizvodnje koriste uz inteligentnu automatizaciju, razmjenu podataka i sveprisutnu povezanost. U tom kontekstu, unaprjeđenje kontrole i praćenja proizvodnih linija posebno je zanimljivo kao ključni korak upravljanja u sklopu pametne proizvodnje i isporuke proizvoda. Aspekti kao što su prikupljanje detaljnih proizvodnih podataka i mjerenje vremena po fazi duž proizvodnih linija pružaju se za kontroliranje i optimizaciju produktivnosti, izbjegavajući potencijalne pogreške koje mogu proizaći iz metoda praćenja temeljenih na ljudima. Stoga, predloženo rješenje doprinosi boljem

upravljanju proizvodnim resursima, omogućujući smanjenje vremena proizvodnje, brže i informiranije odgovore te unaprijeđenu kontrolu nad proizvodnim procesima općenito. Kroz nekoliko revolucionarnih generacija, industrijska revolucija je doživjela različite prekidače, uključujući parnu mehanizaciju, proizvodnju na temelju električne energije, automatizaciju temeljenu na računalima, te razvoj sustava sa spojem fizičkog i digitalnog svijeta (CPS) i tehnologije Interneta stvari (IoT) za poticanje proizvodnje. Prema [20], podaci i parametri se predaju računalnim komponentama koje ih, u izvornom ili prerađenom obliku, prosljeđuju IoT softverskim platformama. U slučaju postavljanja samostalnih senzora, obavezno je povezati ih sa računalno-komunikacijskim modulima („senzorski čvorovi“) koji izmjerene vrijednosti prosljeđuju do IoT platformi. Prema [18], sljedeća generacija, poznata kao peta generacija, proizvodnju će poticati putem sustava inteligentne automatizacije (IA). To znači da će industrija budućnosti biti zasigurno više automatizirana i inteligentna u načinu provođenja proizvodnih procesa, kao i u donošenju odluka o upravljanju poslovanjem. Ta inteligencija će predviđati različite scenarije proizvodnje i preporučivati najprikladnija rješenja za svaki pojedinačni scenarij kako bi se povećala proizvodnja i/ili udovoljilo potražnji i potrebama klijenata. Istovremeno, inteligencija će optimizirati i povećati učinkovitost proizvodnog procesa pružajući jasne informacije o procesima, opremi, proizvodu i operaterima. Slijedi osvrt na postojeća slična rješenja za praćenje proizvodnog ciklusa tvrtke.

2.3.2. Postojeća slična rješenja

2.3.2.1. PAUK

Prema [3], PAUK je programski sustav nastao 2018. godine s ciljem da pomaže svim tvrtkama u proizvodnji bez obzira na veličinu. Prema [3], sustav prikazan na slici 2.4 pametno upravlja, planira i prati proizvodnju u realnom vremenu u izravnoj vezi s osobljem i stanjem materijala i sirovina na skladištu. Velika prednost ovog programa je što se može integrirati u proizvodni ciklus s već postojećim rješenjem praćenja. Najprije je potrebno snimiti cijeli proces kako bi se definirala mjesta koja treba poboljšati te nakon toga se stvara ideja implementacije te realan cilj nakon kojeg slijedi izbacivanje prve verzije sustava online. Korištenjem ovog programskog sustava moguće je reagirati na situacije procesu dok se one odvijaju. Brza reakcija je moguća jer su radnici obaviješteni putem SMS-a ili E-maila ako je došlo do komplikacija u procesu, a obavijesti mogu biti i u smislu prikaza stanja proizvodnje te izdavanja radnih naloga u različitim oblicima.

Slika 2.4. Prikaz PAUK MES programskog sustava

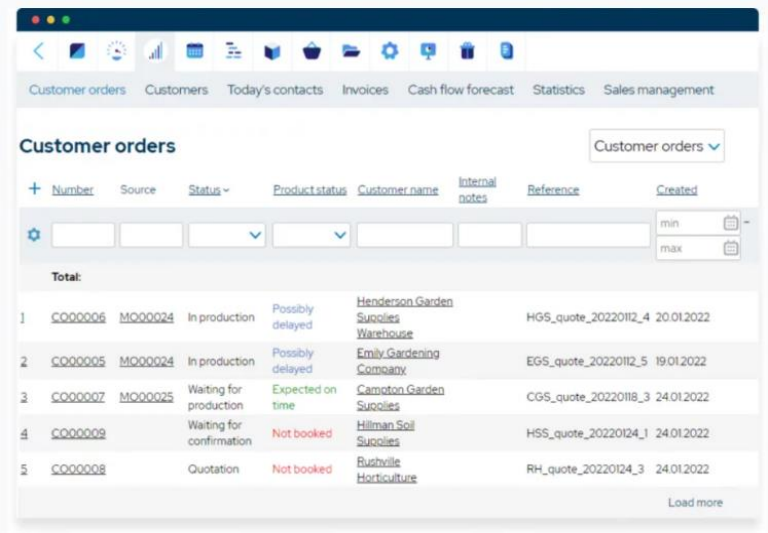
2.3.2.2. PackOS

Prema [2], PackOs je moćan programski sustav za izračunavanje ukupne učinkovitosti opreme koji pruža praćenje performansi procesa u stvarnom vremenu s izračunom OEE (eng. *Overall Equipment Effectiveness*) omjera, dinamičkom analizom uzroka te klasifikacijom prekida rada. Kao što je prikazano na slici 2.5 prema [2], sustav ističe ključne gubitke performansi te pruža korisnicima uvid u to kako ih smanjiti. PackOS je korišten u 21. državi svijeta, podržava praćenje za preko 1800 automatiziranih strojeva te ga koriste i najpopularnije tvrtke kao što su Avon i Carlsberg. Omogućuje i video nadzor nad odabranim strojevima za uvid u potencijalne kvarove koji pristižu upozorenjem u obavještajnom prozoru. Kao inovativno rješenje nudi SLF (eng. *Self Learning Factory*), što predstavlja ekstenziju sustava PackOS koja predviđa stanje strojeva na temelju inteligentnih prediktivnih modela. Stanje strojeva se poboljšava kreiranjem sigurnog okruženja za svo osoblje uključeno u proizvodni ciklus.

Slika 2.5. Prikaz PackOS sustava za praćenje proizvodnog procesa

2.3.2.3. MRPeasy

Prema [15], MRPeasy je sustav jednostavnog korisničkog sučelja koje prikazuje podatke svih područja tvrtke, od prodaje i financija do stanja na skladištu i proizvodnog procesa. Kao što je prikazano slici 2.6 prema [15], aplikacija se koristi tako da se u bazi spremaju unesene narudžbe sa svim potrebnim podacima te se u stvarnom vremenu može pratiti i postavljati stanje narudžbe. MRPeasy ima brojne ugrađene funkcije kao što su CRM (eng. *Customer Relationship Management*) koji omogućava računanje troška proizvodnje proizvoda te najboljeg vremena dostave proizvoda klijentu. Uz to, omogućena je komunikacija s klijentima te timom koji prati zadatke na svoj radnim računalima ili mobilnim uređajima. Aplikacija se vrlo lako koristi i implementira te ju koristi više od 1800 raznih proizvođača svijeta. Može se ugraditi u platforme kao što su Amazon i Spotify te tako korisnici određenih platformi imaju detaljniji pregled cjelokupnog procesa prodaje proizvoda te stvaranja rasporeda za određene timove.



Slike 2.6. Prikaz sučelja MRPeasy sustava za raspored i praćenje narudžbi

3. ZAHTJEVI, MODEL I GRAĐA SUSTAVA ZA PRAĆENJE PROIZVODNOG CIKLUSA TVRTKE ZA PRERADU KOŽE

U ovom poglavlju opisani su zahtjevi na web sustav, konceptualni model sustava te struktura sustava u obliku dijagrama tijeka rada aplikacije.

3.1. Funkcionalni zahtjevi na web sustav

Funkcionalni zahtjevi su opisi specifičnih akcija i operacija za koje korisnik očekuje da sustav može izvršiti kako bi zadovoljio njihove potrebe. Web sustav koji je osmišljen i programski ostvaren u ovom radu ima sljedeće funkcionalne zahtjeve:

- Registracija i prijava korisnika
- Provjera identiteta korisnika s različitim ulogama (klijentska i administratorska)
- Pregled trenutnog stanja skladišta
- Praćenje svakodnevnih aktivnosti putem kalendara
- Stvaranje, pregled i raspoređivanje radnih naloga po skladištima
- Promjena statusa radnih naloga
- Stvaranje, pregled i promjena statusa proizvodnih i naloga za otkup kože
- Pravovremeno automatsko obavještavanje korisnika o promjenama statusa naloga te kašnjenja s plaćanjem
- Pregled korisnika, klijenata, dobavljača
- Dodavanje novih klijenata i dobavljača
- Odjava korisnika iz sustava.

3.2. Nefunkcionalni zahtjevi na web sustav

Nefunkcionalni zahtjevi su kvalitetne karakteristike sustava koje će korisnik iskusiti te opisuju kako bi se sustav trebao ponašati. Nefunkcionalni zahtjevi nisu direktno povezani s osnovnim funkcijama web sustava već ukazuju na druge aspekte koji utječu na uspješnost sustava. Nefunkcionalni zahtjevi su dostupnost, skalabilnost, sigurnost, prilagodljivost, korisničko iskustvo, upotrebljivost te performanse i pouzdanost web sustava koji su najvećim dijelom razvijeni u web sustavu ovog rada. Prema [4], pouzdanost web sustava je mjera mogućnosti neuspjeha web sustava. Kako bi se povećala mjera pouzdanosti, potrebno je ukloniti sve moguće greške koje bi utjecale na sustavne komponente. Performanse opisuju kako se programsko rješenje ponaša u različitim situacijama. Loše performanse mogu dovesti do negativnog korisničkog iskustva i smanjenja sigurnosti sustava.

U ovom radu naglasak je na dobrom korisničkom iskustvu te sigurnosti web sustava. Korisničko iskustvo temelji se na jednostavnom korištenju web sustava, odnosno da su sve komponente logički posložene te lako dostupne korisniku. Kretanje po web sustavu je intuitivno te se pri svakoj akciji dobije povratna informacija koja pomaže pri sljedećem koraku. Sigurnost web sustava odnosi se na sigurnu registraciju, prijavu te odjavu korisnika. Lozinku zna samo korisnik, te pri registraciji nije moguće unijeti već postojeću e-poštu u bazi podataka.

3.3. Model sustava

U ovom potpoglavlju prikazan je i objašnjen model web sustava za tvrtku koja se bavi kupnjom, preradom i prodajom kože, pružajući detaljan uvid u ključne parametre koje on koristi, kao i njegova glavna obilježja.

3.3.1. Parametri bitni za praćenje proizvodnog ciklusa tvrtke za preradu kože

Proizvodnja u industriji kože zahtijeva precizno upravljanje svakim korakom proizvodnog ciklusa kako bi se osigurala visoka kvaliteta proizvoda, maksimalna iskoristivost dostupnih resursa i kapaciteta skladišta te na kraju i zadovoljstvo klijenta. Za olakšano upravljanje takvim procesom postoje parametri koji omogućuju detaljan uvid u svaki korak procesa. Kako je svaki korak predstavljen fazom, svaka od tih faza obuhvaća potrebne parametre kako bi korisnik mogao pratiti proces proizvodnje u realnom vremenu. Kao što je već objašnjeno, razlikuju se faze kupnje resursa, prerađivanje i proizvodnju resursa u gotove proizvode te prodaju i dostavu gotovih proizvoda klijentima. Analizom i dokumentacijom ovih parametara, omogućeno je pravovremeno donošenje odluka, koje olakšavaju poštivanje rokova i osiguravaju prednost nad konkurentnim tvrtkama u industriji prerade kože. U svakoj od faza proizvodnog ciklusa, od kupnje resursa do prodaje gotov prerađenog proizvoda nalazi se niz ključnih parametara koji zajedno čine cjelinu za olakšano praćenje procesa.

Parametri koji su relevantni u radnim nalozima kupnje resursa su:

- Datum izrade naloga
- Rok plaćanja
- Ime resursa
- Kvalitetna klasa resursa
- Cijena resursa po jedinici mase
- Jedinica mase
- Količina resursa
- Broj naloga
- Skladište pohrane resursa

- Status naloga

Parametri koji su relevantni u nalozima za proizvodnju:

- Datum izrade naloga
- Broj naloga
- Broj naloga kupnje resursa
- Materijal korišten u proizvodnji (npr. led)
- Cijena izrade resursa
- Status naloga

Parametri koji su relevantni u nalozima za prodaju gotovog proizvoda:

- Datum izrade naloga
- Podaci o klijentu
- Broj i podaci o nalogu za proizvodnju
- Podaci o proizvođaču gotovog proizvoda
- Ime i prezime stvaratelja naloga
- Status naloga

Parametri su predstavljeni stupcima u bazi podataka što omogućuje trajnu pohranu naloga i njihov prikaz u web sustavu. Neki parametri su vezani za druge kako bi se dobio detaljniji prikaz proizvodnog ciklusa. Prikaz relacija između parametara može se bolje razumjeti kroz relacijski dijagram, koji je prikazan na slici 3.4, a olakšava dublje razumijevanje povezanosti svih aspekata proizvodnog ciklusa.

3.3.2. Raspoređivanje

Prema [5], osnovna svrha postupka raspoređivanja je raspoređivanje obrade posla dodjeljivanjem resursa zadacima i određivanjem njihovih početnih i ako postoje, krajnjih vremena. Prema [15], odluke o raspoređivanju su kompleksne te zahtijevaju značajne kognitivne napore. Ponekad, prilikom procesa raspoređivanja, kognitivni resursi pojedinca mogu biti ograničeni, što znači da može biti nemoguće definirati koja je odluka bolja od druge. Postupkom raspoređivanja se susreće se u svim vrstama sustava, budući da je potrebno distribuirati rad među više entiteta i komponenti. Pri rješavanju problema raspoređivanja, također pojavljuje se i problem dodjele, što se odnosi na određivanje resursa za obradu konkretne operacije. Kako tehnologija i industrije napreduju, problemi raspoređivanja uključuju sve više ograničenja i fleksibilnosti kako bi se modelirali specifični scenariji. Primjer procesa za raspoređivanje u normalnim uvjetima koji bi bio primjenjiv u navedenom web sustavu je FIFO (eng. *First In First Out*). FIFO predstavlja načelo održavanja preciznog slijeda osiguravanjem da prvi dio koji ulazi u proizvodni ciklus ili lokaciju za pohranu

također bude i prvi dio koji izlazi. Međutim, treba uzeti u obzir da u slučaju pogrešaka, smetnji ili promjena uvjeta, nužno dolazi do potrebe za procesom preraspoređivanja. U predloženom rješenju web sustava koristi se raspoređivanje kapaciteta. Prema [19], raspoređivanje po kapacitetu je proces gdje se plan proizvodnje sastoji od sekvenci operacija s ciljem ispunjavanja naloga s obzirom na stanje kapaciteta i resursa. Resursi mogu biti strojevi, kamioni, alati te kao u slučaju ovog web sustava, skladišta.

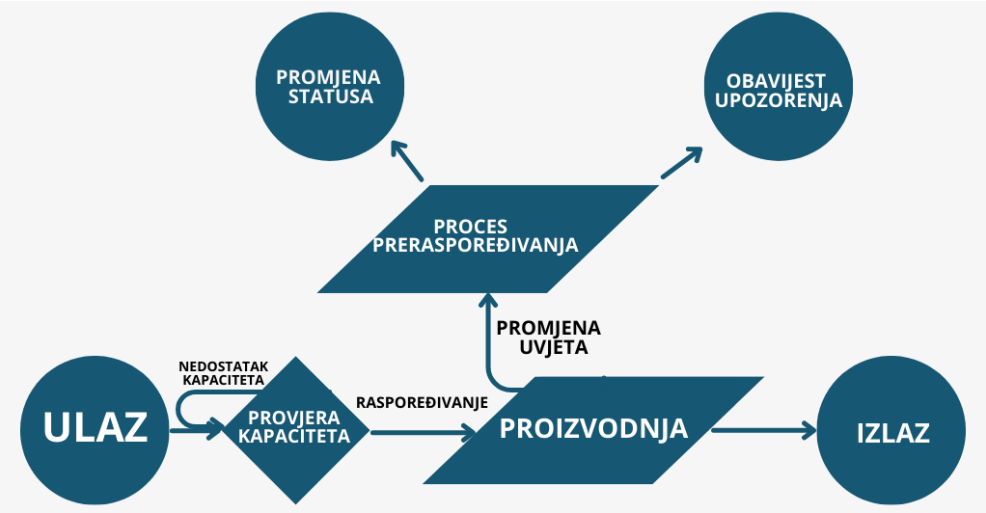
3.3.3. Preraspoređivanje

Prema [6], model preraspoređivanja proučava kako ponovno rasporediti poslove kada se početni plan proizvodnje više ne može nastaviti jer određeni uvjeti ili ograničenja više ne vrijede. Iako se zna dogoditi da je proces raspoređivanja i preraspoređivanja presložen, tehnološki napredak omogućuje razvoj naprednih algoritama koji brzo reagiraju na promjene u proizvodnom okruženju. Preraspoređivanje postaje gotovo obavezno kako bi se minimalizirao utjecaj takvih poremećaja na performanse sustava. Postoji velik broj vrsta poremećaja koji mogu poremetiti plan, uključujući kvarove strojeva, kašnjenja u vremenu obrade, hitne narudžbe, probleme s kvalitetom i nedostupnost materijala. Preraspoređivanje je dinamički pristup koji reagira na prekide, no istovremeno uzima u obzir buduće informacije (stvarajući plan za budućnost). U praksi, preraspoređivanje se periodično izvodi kako bi se planirale aktivnosti za sljedeći vremenski period temeljem stanja sustava. Također se povremeno provodi kao odgovor na značajne prekide. Zbog netočnih vremenskih procjena i neočekivanih događaja, precizno slijediti raspored postaje teže kako vrijeme prolazi. U nekim slučajevima sustav može slijediti sekvencu koju raspored specificira, iako planirana vremena početka i završetka više nisu izvediva. Međutim, s vremenom će biti potreban novi raspored.

U kontekstu web sustava koji se stvara u ovom radu, preraspoređivanje ima ključnu ulogu u osiguravanju kontinuirane učinkovitosti. Prilikom suočavanja s neočekivanim događajima poput kašnjenja u obradi narudžbi, problema s kvalitetom ili nedostupnosti određenih resursa, preraspoređivanje omogućuje reorganizaciju planova kako bi se maksimizirala produktivnost. Ovaj dinamičan pristup omogućuje nam da ne samo reagiramo na poremećaje u stvarnom vremenu, već i da razmišljamo unaprijed stvarajući nove planove za buduće aktivnosti. Time se osigurava da web sustav ostane prilagodljiv i sposoban za optimalno funkcioniranje čak i u uvjetima promjena i neočekivanosti.

3.3.4. Primjena raspoređivanja, preraspoređivanja i upozoravanja

Web sustav opisan u ovome radu bavi se stvaranjem i praćenjem proizvodnog ciklusa tvrtke za preradu kože koji mora zadovoljiti veći broj kriterija, među kojima su kapacitivna raspoloživost skladišta, pravovremeno plaćanje radnih naloga te promjena statusa svih faza proizvodnog ciklusa. Kapacitivna raspoloživost skladišta odnosi se na količinu resursa koje svako od tri skladišta može pohraniti. Problem raspoređivanja u skladišta je nedostatak prostora za pohranu gdje se u tom slučaju postupak preraspoređivanja odvija promjenom statusa radnog naloga u „Odgođeno“ (engl. „*Postponed*“). Ako je željeno skladište nedostupno s obzirom na trenutan dostupan kapacitet, korisniku sustav javlja grešku te je moguće izabrati jedno od preostala dva skladišta ako ispunjavaju količinski uvjet. Pravovremeno plaćanje radnih naloga je vremenski zadatak koji prikazuje krajnji rok dokad neki račun mora biti plaćen dobavljaču. Da bi osigurali pravovremeno plaćanje i poštivanje roka, pomažu postupci upozoravanja. U iščitavanju parametra roka plaćanja, ako je taj rok unutar 7 dana od trenutnog datuma, automatski se šalju upozorenja korisniku s prioritetnom važnošću, koje su dostupne u obavijesnoj komponenti. Obavijest sadrži broj naloga i preostali broj dana do roka otplate kako bi korisniku bio olakšan uvid za njegovo pravovremeno poštivanje. Također, obavijesti upozoravanja vežu se uz postupke preraspoređivanja faza proizvodnog ciklusa koje su ostvarene uz promjenu statusa svakog od tri naloga. Mogući statusi faza su: na čekanju, odgođeno, blokirano, aktivno, rad u skladištu, riješeno i otprema. Primjerice, ako je došlo do kvara na stroju pri procesu prerađivanja naloga koji za status ima „rad u skladištu“, moguće je prerasporediti resurse promjenom statusa u „blokirano“, pri čemu korisniku dolazi obavijest kako je došlo do promjene statusa s informacijama o broju naloga i prethodnog i trenutno postavljenog statusa. Na slici 3.1 prikazana je shema procesa raspoređivanja, postupkom kapacitetskog predraspoređivanja te preraspoređivanja i upozoravanja nakon promjene uvjeta proizvodnog ciklusa.



Slika 3.1. Shema dijagrama rada raspoređivanja i preraspoređivanja

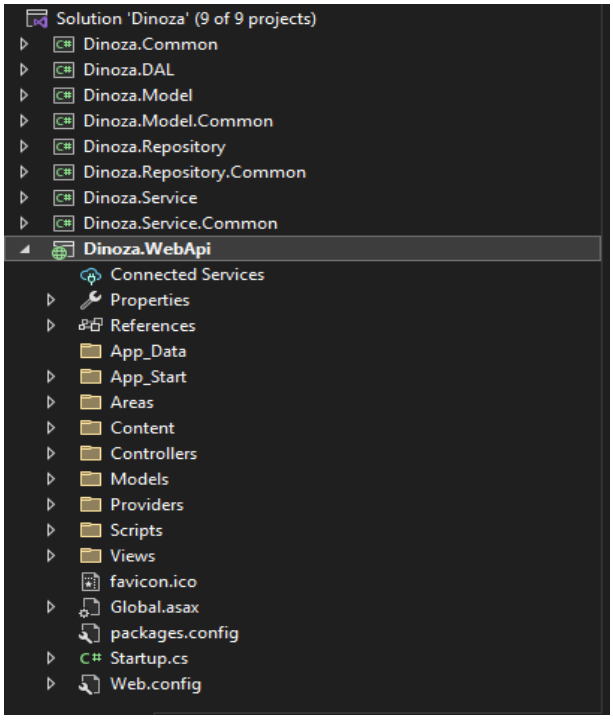
3.4. Građa sustava, dijagram rada sustava i arhitektura baze podataka

3.4.1. Građa web sustava

U okviru ovog projekta, razvijen je web sustav unutar ASP.NET okruženja, koje pruža mogućnost izgradnje HTTP (eng. *Hypertext Transfer Protocol*) usluga pristupačnih s različitih uređaja, uključujući mobilne uređaje i osobna računala. Kao što je opisano u [6], ASP.NET predstavlja tehnološki skup alata temeljenih na Microsoft .NET platformi, omogućavajući visoku razinu sigurnosti prilikom kreiranja web sustava. Arhitektura web sustava temelji se na konceptima MVC (eng. *Model-View-Controller*), iako se prilagođava potrebama web aplikacija i razlikuje od klasične MVC arhitekture primijenjene u aplikacijama s grafičkim korisničkim sučeljem. U ovom slučaju, kao što ilustrira slika 3.3, sustav se strukturira u tri ključna sloja. Prvi sloj, poznat kao sloj interakcije s korisnikom, služi kao ulazna točka za korisničke zahtjeve prema web sustavu. Ovaj sloj prima zahtjeve od korisnika putem web sučelja te prosljeđuje odgovarajuće REST modele prema drugom sloju - sloju poslovne logike. Drugi sloj, sloj poslovne logike, sadrži glavnu logiku sustava. Ovdje se podaci iz modela pripremaju i obrađuju prije nego što budu proslijeđeni dalje prema trećem sloju - sloju pristupa podacima. Treći sloj, sloj pristupa podacima, direktno je povezan s bazom podataka. Kroz ovaj sloj prolazi svaka komunikacija s bazom podataka, bilo da se radi o spremanju novih podataka ili dohvaćanju informacija iz baze. Ova struktura omogućuje jasno odvajanje odgovornosti za pristupanje podacima, čime se pojednostavljuje održavanje i proširivost sustava. U skladu s prikazom na slici 3.2, u ovoj arhitekturi nazivi slojeva su "*controller*", "*service*" i "*repository*". Svaki od tih slojeva ima svoju specifičnu ulogu i zajednički doprinosi cjelokupnoj funkcionalnosti i performansama web sustava. Nadalje, četvrti značajan aspekt ove arhitekture su "modeli". Modeli predstavljaju direktan prikaz entiteta iz baze podataka

i definiraju njihove atribute s odgovarajućim tipovima podataka. Ovi modeli služe kao most između baze podataka i ostatka sustava, olakšavajući manipulaciju podacima i njihovu kvalitetnu obradu.

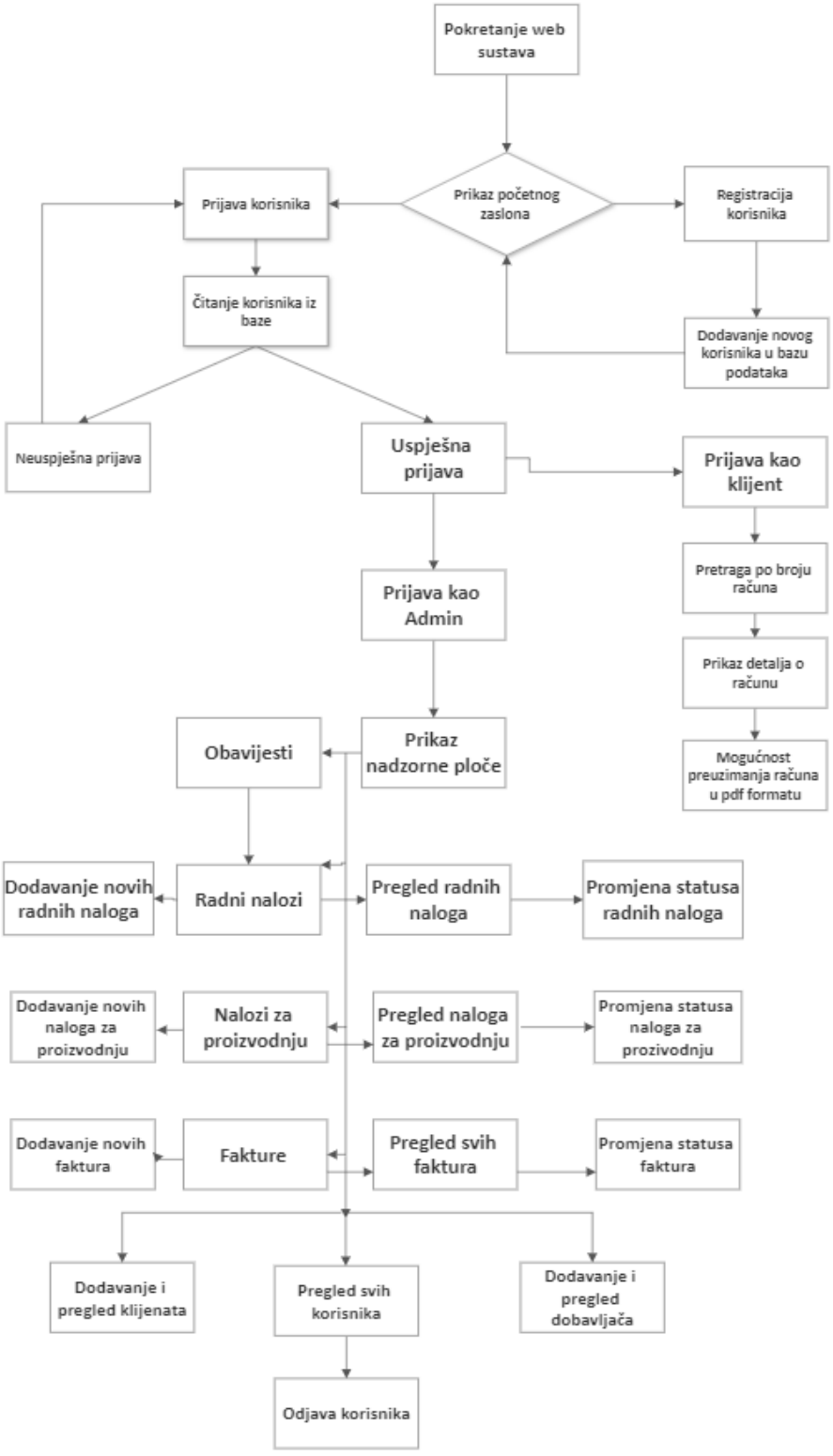
Slika 3.2. Troslojni strukturni model



Slika 3.3. Prikaz arhitekture web sustava

3.4.2. Dijagram tijeka rada sustava

Prema [7], dijagrami rada su važni za razvoj web aplikacija jer pomažu programerima vizualizirati cijeli tijek koda i logičkog prolaska kroz aplikaciju. Stvarajući mapu tijeka programskog koda, programeri lakše identificiraju potencijalne probleme i nesigurnosti. Pomoću dijagrama rada moguće je prepoznati i eliminirati nepotrebne korake kroz tijek aplikacije što može uštedjeti novac i osigurati veću uspješnost cjelokupne operacije. Na slici 3.4 nalazi se dijagram tijeka rada navedenog web sustava. Prilikom otvaranja web sustava, korisniku se prikazuje početni zaslon s osnovnim informacijama o tvrtki za preradu kože. Korisnik ima mogućnosti registriranja i prijava, odnosno stvaranja novog korisnika u bazi podataka ili prijave kao već postojeći korisnik. Ako se korisnik prijavi u ulogu klijenta, ima mogućnost pretraživanja faktura po broju fakture. U slučaju da faktura s tim brojem ne postoji, korisniku se pojavljuje tekst koji ukazuje na nepostojeći broj fakture. S druge strane, ako je broj točan i faktura postoji, korisniku se prikazuju podaci o fakturi te ima mogućnost preuzimanja fakture u pdf obliku koju može ispisati.



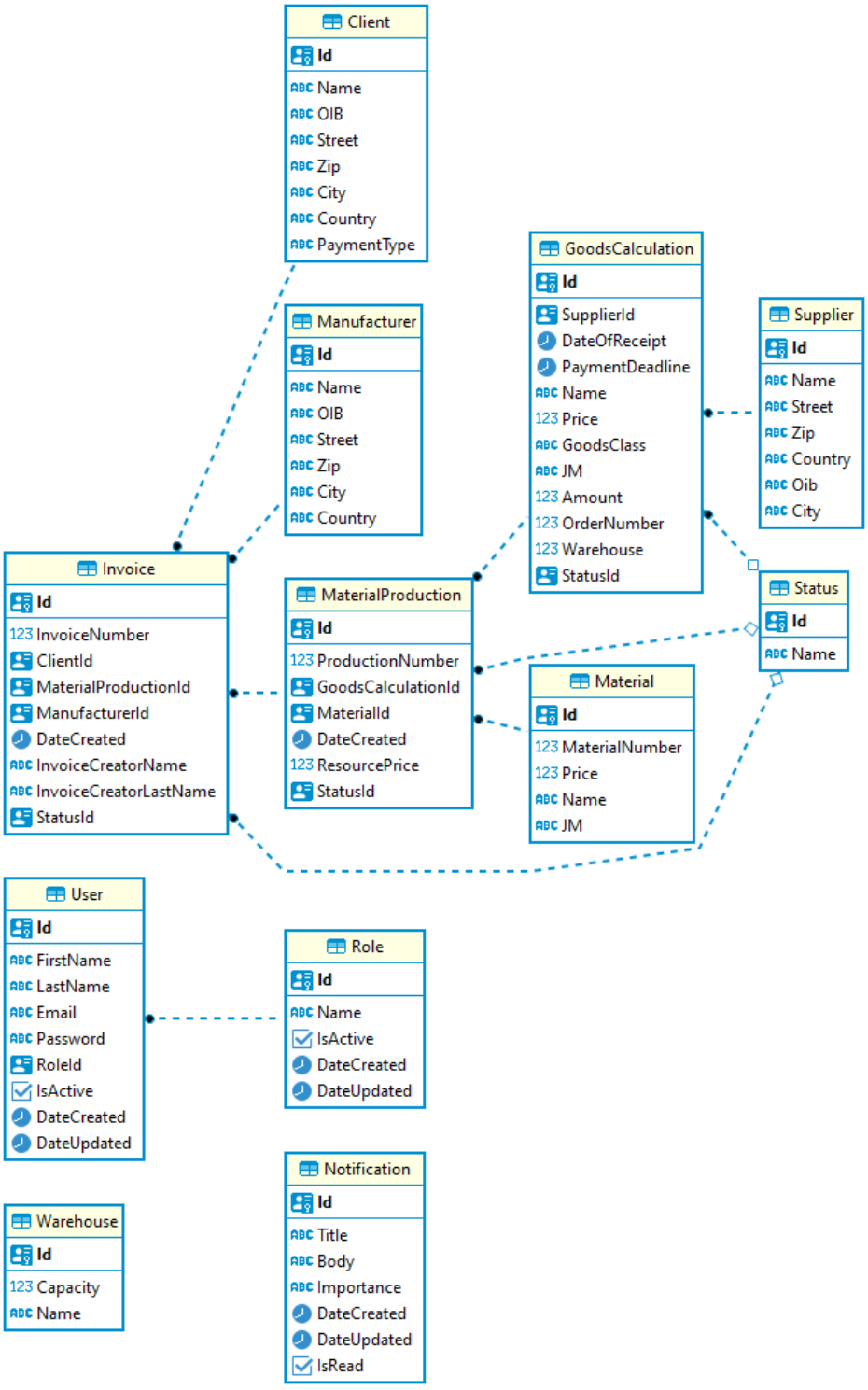
Slika 3.4. Dijagram tijeka rada sustava

U slučaju ako se korisnik prijavi u ulogu administratora, mogućnosti su mu proširene. Korisnika će najprije dočekati radna ploča s podacima o stanju skladišta te kalendar koji služi za upisivanje i praćenje svakodnevnih aktivnosti. Sa strane se nalazi izbornik pomoću kojega korisnik može pristupati ostalim komponentama aplikacije. Korisnik može unositi nove, pregledati sve od tri

vrste naloga, te promijeniti status na svakome od njih pojedinačno. Ako dođe do nekih promjena, korisniku dolaze obavijesti koje se također nalaze na izborniku sa strane. Administratoru su još dostupne i funkcije dodavanja novih dobavljača i klijenata, njihov pregled te pregled svih registriranih korisnika u bazi. Za kraj, korisnik ima mogućnost odjave iz sustava.

3.4.3. Arhitektura baze podataka

Kako bi korisniku bilo omogućeno registriranje, prijava, stvaranje te prikazivanje naloga, dobavljača, klijenata i obavijesti potrebno je sve te podatke spremiti u bazu podataka. U ovome web sustavu korištena je PostgreSQL baza koja je standardna SQL (engl. skraćeno od *Structured Query Language*) gdje se podaci spremaju unutar tablica. Prema slici 3.5, korištena je jedna baza podataka s više tablica, gdje svaka tablica predstavlja jedan model s potrebnim atributima. Dijagram na slici naziva se relacijskim dijagramom koji predstavlja grafički prikaz tablica baze podataka zajedno s njezinim atributima i međusobnim poveznicama s drugim tablicama unutar iste baze. Svaka tablica ima svoj „Id“ stupac koji je njen primarni ključ te je vidljivo odvojen od ostalih atributa tablice. Pored svakog imena atributa nalazi se također i tip podatka, pa se tako na primjer za tekstualni tip podatka pored imena nalazi „ABC“. Vidljivo je da su neke tablice povezane pravcima koji označavaju da te tablice posjeduju atribute koje predstavljaju strani ključ na tablicu s kojom su spojene.



Slika 3.5. Prikaz relacijskog dijagrama baze podataka

4. PROGRAMSKO RJEŠENJE WEB SUSTAVA

U ovom poglavlju objašnjavaju se ključne tehnologije koje su primijenjene u poslužiteljskom i korisničkom dijelu web sustava, kao i tehnologije baze podataka i testiranja zahtjeva. Nakon toga, prikazani su i opisani najbitniji dijelovi programskog koda web sustava.

4.1. Korišteni programski jezici, tehnologije i razvojne okoline

4.1.1. PostgreSQL

Prema [8], PostgreSQL je napredna relacijska baza podataka koja podržava relacijske (SQL) i ne relacijske (JSON) upite. To je vrlo stabilna baza podataka koja se primarno koristi kao skladište podataka za veliki broj web, mobilnih i analitičkih aplikacija. Jedna od najvećih prednosti PostgreSQL-a je što je otvorenog izvora koda (eng. *open source*) što korisnicima omogućuje besplatno korištenje, razvoj i implementaciju u aplikacije.

4.1.2. C# .NET

Prema [9], C# je moderan, objektno orijentiran programski jezik koji korisnicima omogućuje izgradnju velikog broja tipova sigurnih i snažnih aplikacija koje se pokreću u .NET programskom okruženju. C# svoje korijene pronalazi u obitelji C programskih jezika i programerima može biti vrlo sličan Javi i C++ programskim jezicima. Karakteristika objektno orijentiranosti ukazuje da korisnik može stvarati klase, odnosno korisničke tipove podataka te može stvarati i instance tih klasa, odnosno objekte. Glavne novosti kod objektno orijentiranih jezika su koncepti nasljeđivanja i polimorfizma. Nasljeđivanje se odnosi na to da jedna klasa može naslijediti dio ili svu strukturu i ponašanje od druge klase. Klasa koja nasljeđuje naziva se potklasom, dok se klasa koju se nasljeđuje naziva natklasom. S druge strane, polimorfizam se odnosi na to da natklasa sadrži metode zajedničke svim izvedenim klasama u hijerarhiji gdje pojedina klasa može izvesti tu metodu na sebi potreban način. Takve metode u izvedenoj klasi nazivamo virtualnim metodama. Za C# može se reći da je „*strongly typed language*“ što zapravo znači da svaka varijabla i konstanta ima svoj tip i svoju vrijednost. Svaka metoda se deklarira imenom, te povratnim tipom podatka, kao i listom parametara koja može biti prazna.

4.1.3. Postman

Prema [10], Postman je aplikacija koja se koristi za API (eng. *Application Programming Interface*) testiranje. Služi kao HTTP klijent koji testira HTTP zahtjeve, koristi korisničko grafičko sučelje kroz koje dobivamo različite tipove odgovora koji se moraju obraditi. Najkorištenije metode zahtjeva su:

- GET – za dobivanje informacija na korisnički zahtjev
- POST – dodavanje informacija
- PUT - uređivanje informacija
- DELETE – brisanje informacija.

Ove četiri metode zajedno čine CRUD (eng. *Create, Read, Update, Delete*) koje u računalnom programiranju predstavljaju četiri osnovne operacije trajne pohrane informacija.

4.1.4. REST API

Prema [11], RESTful API je programsko sučelje koje se ponaša u skladu s ograničenjima REST arhitektura i omogućuje interakciju sa RESTful web uslugama. API predstavlja skup definicija i protokola za izradu programskog softvera. Drugim riječima, ako se želi međusobno komunicirati s računalom kako bi izvršilo neku akciju, API pomaže u komunikaciji kako bi računalo potpuno razumjelo i ispunilo zahtjev. S druge strane, REST (eng. *Representational State Transfer*) predstavlja skup ograničenja. Kada korisnik šalje zahtjev putem RESTful API-ja, zahtjev koji sadrži neke informacije se prenosi u obliku nekih od formata, od kojih je kao u ovom web sustavu, najkorišteniji JSON, jer je čitljiv i ljudima i strojevima.

4.1.5. HTML

Prema [12], HTML je prezentacijski jezik za stvaranje web stranica. Inicijalno, HTML definira strukturu i prikaz Web dokumenta koristeći raznolike atribute i oznake. Web preglednik može čitati HTML datoteke i od njih sastaviti vidljive web stranice. Preglednik automatski pretvara oznake u sadržaj interpretirajući oznake i atribute za prikaz stranice. Elementi HTML-a čine temeljne blokove svih web stranica. Zahvaljujući HTML-u, moguće je umetati slike i različite objekte te kreirati interaktivne obrasce koji omogućuju slanje podatak natrag na server. Osim toga, pruža sredstvo za stvaranje strukturiranih dokumenata tako da dodjeljuje strukturne značajke tekstualnom sadržaju, kao što su naslovi, paragrafi, liste, poveznice, citati te drugi elementi.

4.1.6. CSS

Prema [12], web preglednici također koriste CSS (eng. *Cascade Styling Sheets*) kako bi definirali izgled i raspored HTML stranice. CSS se može smatrati kao umjetnikom razvoja web stranica, čineći izmjene poput promjene boja elemenata, veličine teksta i rasporeda. CSS je primarno dizajniran kako bi razdvojio komponente i elemente koje grade web stranicu od atributa koje utječu na njihov izgled i prikaz. Ovakva struktura poboljšava pristupačnost sadržaju, pruža fleksibilnost i kontrolu nad specifičnim dijelovima web stranice.

4.1.7. React.js

Prema [13], React je biblioteka JavaScripta za izradu korisničkog prikaza. Iako je to više biblioteka nego jezik te je relativno nova, korištenost React-a rapidno raste među programerima svijeta. Jedan od glavnih razloga zašto koristiti React je mogućnost ponovnog korištenja stvorenih komponenti. Komponente predstavljaju blokove koji izgrađuju React aplikaciju, a aplikacija se sastoji od većeg broja komponenti. Komponente imaju svoju logiku i kontrolu, mogu se koristiti kroz cijelu aplikaciju i tako smanjiti ponavljanje koda. Kako se React najviše sastoji od kombinacije osnovnog HTML-a i koncepta JavaScripta relativno je lak za naučiti, a uz to može se koristiti za razvoj i web i mobilnih aplikacija pomoću React Native okruženja.

4.2. Programsko rješenje web sustava

4.2.1. Implementacija prijave i registracije korisnika

Funkcionalnost registracije korisnika ostvarena je tako da korisnik upisuje podatke o korisničkom računu te taj POST zahtjev najprije ulazi u kontroler, nakon kojeg kroz sloja za logičku jedinicu odlazi u repozitorij koji komunicira s bazom podataka. Kao što je prikazano na slici 4.1, u bazu podataka se spremaju uneseni podaci o korisniku te mu je dodijeljena uloga klijenta. Administratorski korisnici su trajno i manualno pohranjeni u bazu podataka te nije omogućena registracija u ulogu administratora. Logika funkcionalnosti također provjerava jesu li uneseni email i lozinka jer su ti podaci nužni za unos u bazu podataka.

```
int rowsAffected;
try
{
    StringBuilder query = new StringBuilder("INSERT INTO \"User\" (\"Id\", \"FirstName\", \"LastName\", \"Email\",
"VALUES (@Id, @FirstName, @LastName, @Email, @Password, @RoleId, @IsActive, @DateCreated, @DateUpdated)");

    using (var connection = new NpgsqlConnection(Helper.connectionString))
    {
        connection.Open();
        using (NpgsqlCommand command = new NpgsqlCommand(query.ToString(), connection))
        {

            command.Parameters.AddWithValue("@Id", user.Id);
            command.Parameters.AddWithValue("@FirstName", user.FirstName);
            command.Parameters.AddWithValue("@LastName", user.LastName);
            command.Parameters.AddWithValue("@Email", user.Email);
            command.Parameters.AddWithValue("@Password", user.Password);
            command.Parameters.AddWithValue("@RoleId", user.RoleId);
            command.Parameters.AddWithValue("@IsActive", user.IsActive);
            command.Parameters.AddWithValue("@DateCreated", user.DateCreated);
            command.Parameters.AddWithValue("@DateUpdated", user.DateUpdated);

            rowsAffected = await command.ExecuteNonQueryAsync();
            return "User registered successfully.";
        }
    }
}
catch (Exception ex)
{
    Trace.WriteLine(ex.Message.ToString());
    throw;
}
```

Slika 4.1. Funkcionalnost registracije na strani poslužitelja

Na korisničkom sučelju, kao što je prikazano na slikama 4.2 i 4.3, postavljen je „*event*“ na svako polje te se tako prati svaki unos u polja potrebna za registraciju, te atributom „*required*“ svako polje je nužno ispuniti kako bi se korisnik mogao registrirati. Atribut na poljima „*onChange*“ pri svakoj promjeni stanja nekog polja unosa, postavlja na trenutno stanje kako bi se u svakom trenutku primijenila promjena na objekt korisnika.

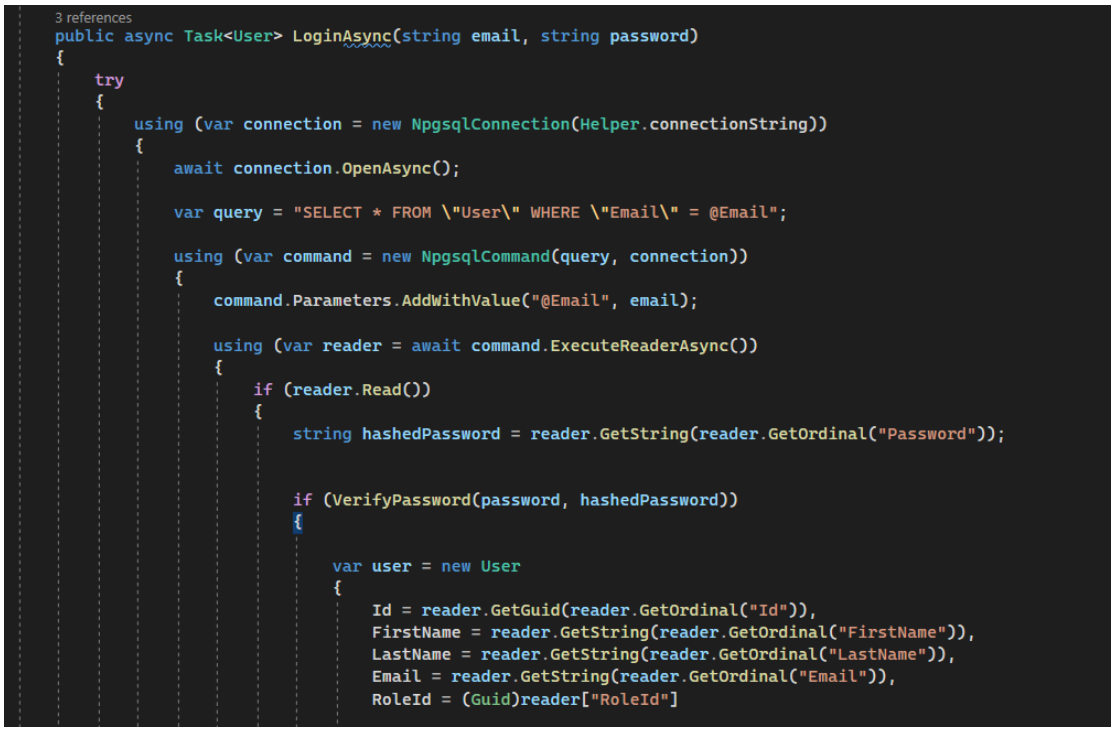
```
const navigate = useNavigate();
const handleEmailChange = (event) => {
  setUser({ ...user, email: event.target.value });
};

const handlePasswordChange = (event) => {
  setUser({ ...user, password: event.target.value });
};

const handleFirstNameChange = (event) => {
  setUser({ ...user, firstName: event.target.value });
};

const handleLastNameChange = (event) => {
  setUser({ ...user, lastName: event.target.value });
};

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

  try {
```

Slika 4.2. Aktivno praćenje svakog unosa u polje

```
return (
  <div className="page-image1">
    <div className="centered-container">
  <div className="register-container">
    <form className="login-div" onSubmit={handleRegister}>
      <label htmlFor="firstName">First Name:</label>
      <input
        required
        type="text"
        id="firstName"
        value={user.firstName}
        onChange={handleFirstNameChange}
      />
      <label htmlFor="lastName">Last Name:</label>
      <input
        type="text"
        required
        id="lastName"
        value={user.lastName}
        onChange={handleLastNameChange}
      />
```

Slika 4.3. Programski kod za polja pri registraciji

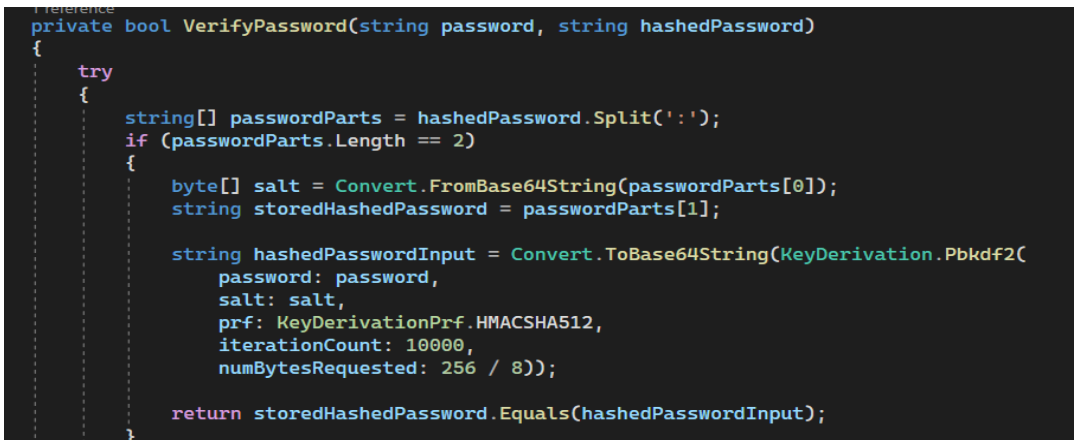
Kao što je prikazano na slici 4.4, prijava korisnika je proces koji najprije dohvaća korisnika po upisanom emailu te se lozinke uspoređuju u kriptiranom obliku i ako se poklapaju, vraća se objekt korisnika do kontrolera. Sve metode web sustava pri deklaraciji imaju ključnu riječ „*async*“ što znači da su te metode asinkrone, odnosno korištenjem „*awaita*“ pri pozivu metoda, te metode će biti izvršene iako korisnik pošalje novi zahtjev dok trenutni nije izvršen.



```
3 references
public async Task<User> LoginAsync(string email, string password)
{
    try
    {
        using (var connection = new NpgsqlConnection(Helper.connectionString))
        {
            await connection.OpenAsync();

            var query = "SELECT * FROM \"User\" WHERE \"Email\" = @Email";

            using (var command = new NpgsqlCommand(query, connection))
            {
                command.Parameters.AddWithValue("@Email", email);

                using (var reader = await command.ExecuteReaderAsync())
                {
                    if (reader.Read())
                    {
                        string hashedPassword = reader.GetString(reader.GetOrdinal("Password"));


                        if (VerifyPassword(password, hashedPassword))
                        {

                            var user = new User
                            {
                                Id = reader.GetGuid(reader.GetOrdinal("Id")),
                                FirstName = reader.GetString(reader.GetOrdinal("FirstName")),
                                LastName = reader.GetString(reader.GetOrdinal("LastName")),
                                Email = reader.GetString(reader.GetOrdinal("Email")),
                                RoleId = (Guid)reader["RoleId"]
```

Slika 4.4. Funkcionalnost prijave korisnika u repozitoriju

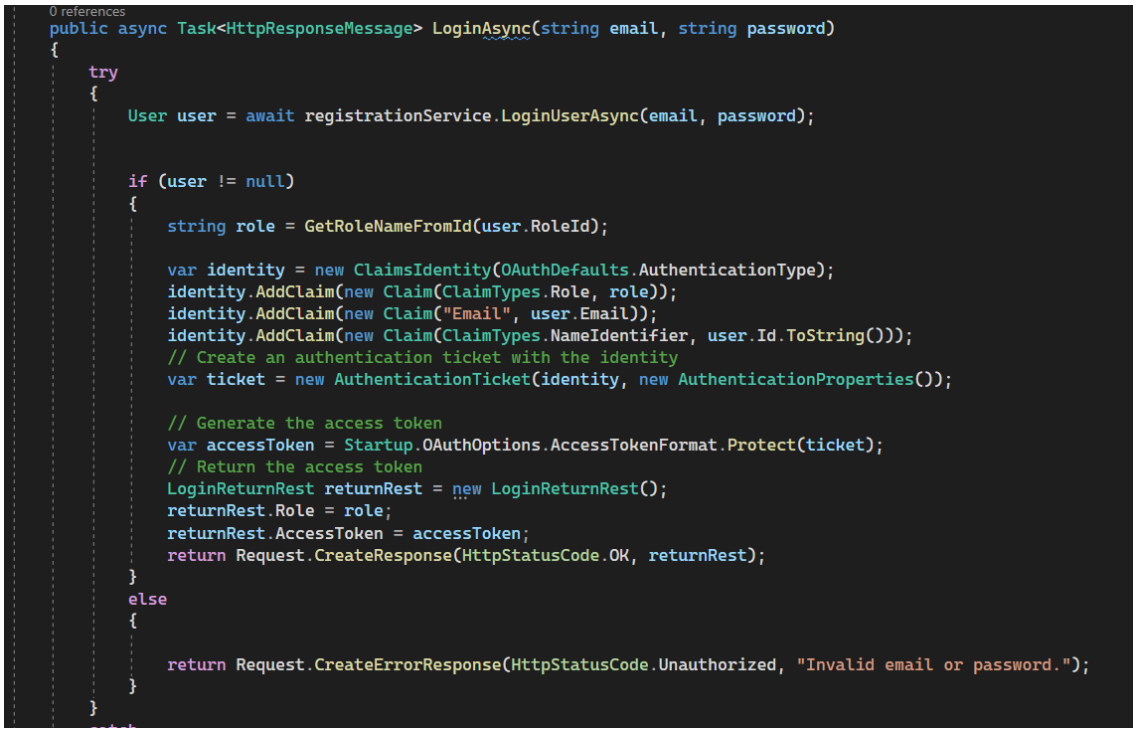
Na slici 4.5 je prikazana metoda za provjeravanje lozinki u kriptiranom obliku. Ako postoji korisnik s poslanim emailom te se lozinke poklapaju, objekt korisnika se šalje u kontroler gdje se stvara novi identitet korisnika. Korisnik dobiva svoj „token“ koji služi za verificiranje korisnika te se pohranjuje u lokalnoj memoriji. Token traje 24 sata i u tom razdoblju iako se npr. ponovno pokrene računalo, a korisnik se ne odjavi, ostat će prijavljen u web sustavu.



```
1 reference
private bool VerifyPassword(string password, string hashedPassword)
{
    try
    {
        string[] passwordParts = hashedPassword.Split(':');
        if (passwordParts.Length == 2)
        {
            byte[] salt = Convert.FromBase64String(passwordParts[0]);
            string storedHashedPassword = passwordParts[1];

            string hashedPasswordInput = Convert.ToBase64String(KeyDerivation.Pbkdf2(
                password: password,
                salt: salt,
                prf: KeyDerivationPrf.HMACSHA512,
                iterationCount: 10000,
                numBytesRequested: 256 / 8));

            return storedHashedPassword.Equals(hashedPasswordInput);
        }
```

Slika 4.5. Programski kod metode za provjeru kriptirane lozinke

Na slici 4.6 prikazana je metoda za stvaranje identiteta i dodjeljivanje tokena korisniku. Prilikom autorizacije korišteno je autorizirajuće okruženje OAuth (eng. *Open Authorization)* koja služi za dodjeljivanje pristupa informacijama na web stranicama ali bez davanja lozinke.



```
0 references
public async Task<HttpResponseMessage> LoginAsync(string email, string password)
{
    try
    {
        User user = await registrationService.LoginUserAsync(email, password);


        if (user != null)
        {
            string role = GetRoleNameFromId(user.RoleId);

            var identity = new ClaimsIdentity(OAuthDefaults.AuthenticationType);
            identity.AddClaim(new Claim(ClaimTypes.Role, role));
            identity.AddClaim(new Claim("Email", user.Email));
            identity.AddClaim(new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()));
            // Create an authentication ticket with the identity
            var ticket = new AuthenticationTicket(identity, new AuthenticationProperties());

            // Generate the access token
            var accessToken = Startup.OAuthOptions.AccessTokenFormat.Protect(ticket);
            // Return the access token
            LoginReturnRest returnRest = new LoginReturnRest();
            returnRest.Role = role;
            returnRest.AccessToken = accessToken;
            return Request.CreateResponse(HttpStatusCode.OK, returnRest);
        }
        else
        {

            return Request.CreateErrorResponse(HttpStatusCode.Unauthorized, "Invalid email or password.");
        }
    }
```

Slika 4.6. Metoda za stvaranje identiteta i prijavu korisnika

Kao što je prikazano na slici 4.7 na korisničkom sučelju, prilikom upisivanja adrese e-pošte i lozinke u polja, sustav putem „*Axiosa*“ šalje zahtjev te ako je prijava valjana, token i uloga korisnika se pohranjuju u lokalnu pohranu. Dohvaćanjem uloge iz lokalne pohrane omogućeno je prikazivanje komponenata ovisno o tome koja je uloga u pitanju. S druge strane, ako prijava nije valjana, u konzolu se sprema obavijest greške, te na korisničkom sučelju se također pojavljuje obavijest o greški prilikom prijave korisnika.

```
try {
  const response = await RegistrationService.postLogin(email, password);

  console.log('Login successful:', response.data);
  localStorage.setItem('token', response.data.accessToken);
  localStorage.setItem('role', response.data.role);
  console.log(localStorage.getItem('token'));

  setEmail('');
  setPassword('');

  if (localStorage.getItem("token") != null){
    navigate("/home");
  }
} catch (error) {
  console.error('Login failed:', error);
  setLoginError(true);
}
};
```

Slika 4.7. Programski kod prijave korisnika na korisničkom sučelju

4.2.2. Dodavanje naloga, klijenata i dobavljača

Proces dodavanja naloga, klijenata i dobavljača prati identičan koncept kao za svaki tip naloga, uz varijacije samo u unosu različitih informacija u bazu podataka. Na primjeru dodavanja faktura,

objašnjen je proces od korisničkog sučelja do komunikacije s bazom podataka. Prvi korak je prikazan na slici 4.8 ilustrirajući unos podataka s korisničkog sučelja koji šalje zahtjev kontroleru, dok je na slici 4.9. ilustriran POST zahtjev za dodavanje faktura u bazu podataka.

```
return (
  <div className="add-data-form">
    <Sidebar />
    <h2>ADD INVOICE</h2>
    <form onSubmit={handleSubmit}>
      <FormControl>
        <InputLabel>Client</InputLabel>
        <Select
          label="Client"
          name="clientId"
          value={formData.clientId}
          onChange={handleChange}
          required
        >
          {clients.map(client => (
            <MenuItem key={client.id} value={client.id}>
              {client.name}
            </MenuItem>
          ))}
        </Select>
      </FormControl>
```

Slika 4.8. POST zahtjev za dodavanje nove fakture

```
const handleChange = (e) => {
  const { name, value } = e.target;
  setFormData((prevData) => ({
    ...prevData,
    [name]: value,
  }));
};

const handleSubmit = (e) => {
  axios.post('https://localhost:44368/api/Invoice', formData)
    .then((response) => {
      console.log('Data added successfully', response.data);
      setFormData({
        clientId: '',
        invoiceNumber: '',
        manufacturerId: 'fba571ce-3753-482f-bf85-6550b38f7b98',
        materialProductionId: '',
        invoiceCreatorName: '',
        invoiceCreatorLastName: '',
        statusId: '',
      });
    })
    .catch((error) => {
      console.error('Error adding data', error);
    });
};
```

Slika 4.9. Primjer polja za odabir klijenta

Nakon što zahtjev stigne do kontrolera, kako je prikazano na slici 4.10, koristi se metoda za konfiguraciju modela fakture iz REST modela koji dolazi s korisničkog sučelja. Određena polja, poput ID-a i datuma stvaranja fakture, ne unose se ručno u bazu podataka, već se popunjavaju programski. Korištenjem metode NewGuid(), svakoj fakturi se u stupac „Id“ dodjeljuje jedinstveni ključ koji se sastoji od kombinacije 32 znaka.

```
1 reference
private Invoice SetModelFromRest(InvoiceRest invoiceRest)
{
    Invoice invoice = new Invoice();
    invoice.Id = Guid.NewGuid();
    invoice.InvoiceNumber = invoiceRest.InvoiceNumber;
    invoice.ClientId = invoiceRest.ClientId;
    invoice.ManufacturerId = invoiceRest.ManufacturerId;
    invoice.InvoiceCreatorLastName = invoiceRest.InvoiceCreatorLastName;
    invoice.InvoiceCreatorName = invoiceRest.InvoiceCreatorName;
    invoice.DateCreated = DateTime.Now;
    invoice.MaterialProductionId = invoiceRest.MaterialProductionId;
    invoice.StatusId = invoiceRest.StatusId;

    return invoice;
}
```

Slika 4.10. Metoda za pretvaranje REST modela u mode fakture

Kontroler kroz uslugu prosljeđuje postavljeni model fakture do repozitorija koji, kako je prikazano na slici 4.11 postavlja fakturu u bazu podataka koristeći sintaksu SQL-a. Metoda za dodavanje

fakture kao parametar prima objekt fakture, pomoću kojeg unosi vrijednosti iz objekta u bazu podataka.

```
public async Task<bool> AddInvoiceAsync(Invoice invoice)
{
    using (NpgsqlConnection conn = new NpgsqlConnection(Helper.connectionString))
    {
        NpgsqlCommand cmd = new NpgsqlCommand("INSERT INTO \"Invoice\" (\"Id\", \"InvoiceNumber\", \"ClientId\",
                                              "(@Id, @InvoiceNumber, @ClientId, @MaterialProductionId, @Manufact

        cmd.Parameters.AddWithValue("@Id", invoice.Id);
        cmd.Parameters.AddWithValue("@InvoiceNumber", invoice.InvoiceNumber);
        cmd.Parameters.AddWithValue("@ClientId", invoice.ClientId);
        cmd.Parameters.AddWithValue("@MaterialProductionId", invoice.MaterialProductionId);
        cmd.Parameters.AddWithValue("@ManufacturerId", invoice.ManufacturerId);
        cmd.Parameters.AddWithValue("@DateCreated", invoice.DateCreated);
        cmd.Parameters.AddWithValue("@InvoiceCreatorName", invoice.InvoiceCreatorName);
        cmd.Parameters.AddWithValue("@InvoiceCreatorLastName", invoice.InvoiceCreatorLastName);
        cmd.Parameters.AddWithValue("@StatusId", invoice.StatusId);

        conn.Open();

        int numberOfAffectedRows = await cmd.ExecuteNonQueryAsync();

        if (numberOfAffectedRows > 0)
        {
            return true;
        }
    }
    return false;
}
```

Slika 4.11. Metoda za dodavanje fakture u bazu podataka

4.2.3. Dohvaćanje naloga, klijenata i dobavljača

Kao i kod dodavanja naloga, koncept dohvaćanja i prikazivanja naloga slijedi jednak model za svaku vrstu naloga, s varijacijom u prikazu različitih informacija o svakoj pojedinoj vrsti. Proces, ilustriran kroz primjer faktura, odvija se od korisničkog zahtjeva sve do komunikacije s bazom podataka, koja potom vraća podatke o nalogu na korisničko sučelje. Na slici 4.12 prikazana je metoda GET zahtjeva za dohvaćanjem svih faktura pohranjenih u bazi podataka. Korištenjem *console.log* metode u konzoli se uz navedeni tekst navodi i tekst greške prilikom neuspješnog zahtjeva za dohvaćanjem faktura.

```
useEffect(() => {

  axios.get('https://localhost:44368/api/Invoice')
    .then(response => {
      setData(response.data);
      console.log(response.data);
    })
    .catch(error => {
      console.error('Error fetching data:', error);
    });
}, []);
const filteredData = data.filter(invoice =>
  invoice.invoiceNumber.toString().includes(searchQuery)
);
```

Slika 4.12. Metoda GET za dohvaćanje faktura

Repozitorij uspostavlja komunikaciju s bazom podataka te dohvaća sve fakture. Zatim, putem liste koja sadrži sve objekte faktura, preko usluge ih šalje u kontroler. Na slici 4.13 prikazana je metoda za dohvaćanje svih faktura iz baze podataka. Za prikaz faktura koriste se informacije koje se ne nalaze u tablici faktura, no s njom su spojene korištenjem stranog ključa na „Id“ te tablice. U tom slučaju, za dohvaćanje tih informacija potrebno je u SQL upitu koristiti ključnu riječ *Join* koja putem stranog ključa dodjeljuje vrijednosti prilikom dohvata.

```
2 references
public async Task<List<GetInvoice>> GetAllInvoicesAsnyc()
{
    List<GetInvoice> invoices = new List<GetInvoice>();
    try
    {
        using (NpgsqlConnection connection = new NpgsqlConnection(Helper.connectionString))
        {
            connection.Open();
            using (NpgsqlCommand cmd = new NpgsqlCommand(
                "SELECT I.\"Id\" AS InvoiceId, " +
                       "I.\"InvoiceNumber\", " +
                       "I.\"DateCreated\", " +
                       "I.\"InvoiceCreatorLastName\", " +
                       "I.\"InvoiceCreatorName\", " +
                       "C.\"Name\" AS ClientName, " +
                       "M.\"Name\" AS ManufacturerName, " +
                       "MP.\"ResourcePrice\", " +
                       "ST.\"Name\" AS StatusName " +
                "FROM \"Invoice\" I " +
                "JOIN \"Client\" C ON I.\"ClientId\" = C.\"Id\" " +
                "JOIN \"Manufacturer\" M ON I.\"ManufacturerId\" = M.\"Id\" " +
                "JOIN \"MaterialProduction\" MP ON I.\"MaterialProductionId\" = MP.\"Id\" " +
                "JOIN \"Status\" ST on I.\"StatusId\" = ST.\"Id\"", connection))
            {
                using (NpgsqlDataReader reader = await cmd.ExecuteReaderAsync())
                {
                    while (await reader.ReadAsync())
                    {
                        GetInvoice invoice = new GetInvoice();
                        invoice.Id = (Guid)reader["InvoiceId"];
                        invoice.InvoiceNumber = (int)reader["InvoiceNumber"];
                        invoice.DateCreated = (DateTime)reader["DateCreated"];
                        invoice.InvoiceCreatorLastName = (string)reader["InvoiceCreatorLastName"];
                        invoice.InvoiceCreatorName = (string)reader["InvoiceCreatorName"];
                        invoice.ClientName = (string)reader["ClientName"];
                        invoice.ManufacturerName = (string)reader["ManufacturerName"];
                        invoice.Price = (double)reader["ResourcePrice"];
                        invoice.StatusName = (string)reader["StatusName"];

                        invoices.Add(invoice);
                    }
```

Slika 4.13. Metoda za dohvaćanje svih faktura iz baze podataka

U modelu faktura, sadrže se informacije poput identifikacije fakture u obliku Guid (eng. *globally unique identifier*) tipa podataka. Takve informacije ne bi trebale biti vidljive korisniku na korisničkom sučelju. Slika 4.14 prikazuje metodu za postavljanje modela u REST model koji će biti prikazan korisniku.

```
private List<GetInvoiceRest> SetModelToRest(List<GetInvoice> invoices)
{
    List<GetInvoiceRest> restInvoices = new List<GetInvoiceRest>();

    foreach (GetInvoice invoice in invoices)
    {

        GetInvoiceRest invoicesRest = new GetInvoiceRest();
        invoicesRest.Id = invoice.Id;
        invoicesRest.InvoiceNumber = invoice.InvoiceNumber;
        invoicesRest.Price = invoice.Price;
        invoicesRest.DateCreated = invoice.DateCreated;
        invoicesRest.InvoiceCreatorLastName = invoice.InvoiceCreatorLastName;
        invoicesRest.InvoiceCreatorName = invoice.InvoiceCreatorName;
        invoicesRest.ManufacturerName = invoice.ManufacturerName;
        invoicesRest.ClientName = invoice.ClientName;
        invoicesRest.StatusName = invoice.StatusName;
        restInvoices.Add(invoicesRest);
    }
    return restInvoices;
```

Slika 4.14. Metoda za postavljanje modela fakture u REST model

Informacije se korisniku prikazuju u obliku tablice radi lakšeg razumijevanja vrijednosti svakog atributa. Slika 4.15 prikazuje primjer programskog koda za postavljanje tablice i prikazivanje vrijednosti za svaki nalog. Svaki broj fakture vodi na novu stranicu koja služi kao prikaz detalja o pojedinačnoj fakturi s mogućnošću promjene statusa fakture. U tijelo tablice predaje se lista faktura, gdje se po *id-u* prolazi lista i prikazuju detalji o pojedinačnoj fakturi.

```
<table className="table">
  <thead>
    <tr>
      <th>Invoice Number</th>
      <th>Client Name</th>
      <th>Manufacturer Name</th>
      <th>Resource price</th>
      <th>Invoice date</th>
      <th>Invoice creator</th>
      <th>Invoice status</th>
    </tr>
  </thead>
  <tbody>
    {filteredData.map(invoice => (
      <tr key={invoice.id}>
        <td>
          <Link to={`/invoice/view/${invoice.id}`} className="no-decoration-link">
            #{invoice.invoiceNumber}
          </Link>
        </td>
        <td>{invoice.clientName}</td>
        <td>{invoice.manufacturerName}</td>
        <td>{invoice.price}€</td>
        <td>{new Date(invoice.dateCreated).toLocaleDateString()}</td>
        <td>{invoice.invoiceCreatorName} {invoice.invoiceCreatorLastName}</td>
        <td>{invoice.statusName}</td>
      </tr>
```

Slika 4.15. Prikazivanje naloga u obliku tablice

4.2.4. Promjena statusa naloga

Za aktivno praćenje i raspoređivanje naloga i resursa unutar proizvodnog ciklusa, postoji funkcionalnost za promjenu statusa za svaku vrstu naloga. Ova funkcionalnost će biti detaljno objašnjena kroz primjer ažuriranja statusa fakture. Korisnik ima mogućnost na korisničkom

sučelju pojedinačno izmijeniti status odabranog naloga. Nakon tog koraka, zahtjev se šalje kontroleru, koji putem usluge prenosi novi status repozitoriju. Repozitorij zatim uspostavlja komunikaciju s bazom podataka kako bi ažurirao točno odabrano polje za taj nalog. Na slici 4.16 prikazan je programski kod POST zahtjeva koji šalje ID fakture te novi status za ažuriranje polja statusa u bazi podataka.

```
axios.put(`https://localhost:44368/api/Invoice/${id}`, {
  statusId: selectedStatus.id,
})
  .then(response => {
    setInvoiceDetails({
      ...invoiceDetails,
      statusName: newStatus
    });
    setEditableStatus(false);
  })
  .catch(error => {
    console.error('Error updating status:', error);
  });
};
```

Slika 4.16. Programski kod zahtjeva za ažuriranje statusa fakture

Metoda za ažuriranje statusa prima dva parametra: ID fakture i ID novog statusa koji će biti postavljen u stupac "StatusId" u bazi podataka. Stupac "StatusId" predstavlja strani ključ koji se povezuje s tablicom "Status", omogućavajući dobivanje imena statusa pri dohvaćanju fakture. Na slici 4.17 prikazana je metoda za ažuriranje statusa koja vraća broj pogođenih redaka u tablici "Invoice" nakon ove promjene.

4.2.1. Obavijesti o promjeni statusa i nepoštivanja roka plaćanja

Funkcionalnost automatskog slanja obavijesti administratoru predstavlja ključni dio web sustava koji omogućava brzo i adekvatno reagiranje na promjene statusa naloga ili nepoštivanje rokova plaćanja. Kroz primjer promjene statusa fakture, objasnit ću kako se ostvaruje automatsko slanje obavijesti administratoru. Kao što je prikazano na slici 4.18 kada dođe do ažuriranja statusa, prvo se u usluzi dohvaća trenutno stanje statusa. Zatim se u bazu podataka dodaje nova obavijest koja za svoje informacije koristi broj fakture, trenutni status fakture i novi status fakture.

Kao što je prikazano na slici 4.18, usluga poziva metodu za dodavanje obavijesti u bazu podataka. Za dodavanje obavijesti potrebno je navesti naslov obavijesti, sadržaj poruke, važnost obavijesti, datum dodavanja i ažuriranja obavijesti. Na slici 4.19 nalazi se programski kod dodavanja obavijesti u repozitorij. Može se primijetiti da postoji i polje „IsRead“ koje predstavlja jedno od stanja obavijesti odnosno stanje pročitane i nepročitane obavijesti.

```
try
{
    using (NpgsqlConnection connection = new NpgsqlConnection(Helper.connectionString))
    {
        await connection.OpenAsync();

        using (NpgsqlCommand command = new NpgsqlCommand())
        {
            StringBuilder updateQuery = new StringBuilder("UPDATE \"Invoice\" SET ");

            if (updateGoods.StatusId.HasValue)
            {
                updateQuery.Append("\"StatusId\" = @StatusId, ");
                command.Parameters.AddWithValue("@StatusId", updateGoods.StatusId.Value);
            }

            if (updateQuery.Length > 0)
            {
                updateQuery.Length -= 2;
            }

            if (updateQuery.Length > 0)
            {
                updateQuery.Append(" WHERE \"Id\" = @Id");
                command.Parameters.AddWithValue("@Id", id);

                string query = updateQuery.ToString();
                command.CommandText = query;
                command.Connection = connection;

                rowsAffected = await command.ExecuteNonQueryAsync();
            }
        }
    }
}
```

Slika 4.18. Programski kod ažuriranja statusa fakture u bazi podataka

```
[HttpPut]
2 references
public async Task<int> UpdateInvoiceStatusAsync(Guid id, [FromBody] UpdateGoodsCalculationModel updateGoods)
{
    try
    {
        var result = await InvoiceRepository.UpdateInvoiceStatusAsync(id, updateGoods);
        if (result != null)
        {
            var status = await StatusRepository.GetAllStatusesAsync();
            var statusName = status.Find(x => x.Id == updateGoods.StatusId).Name;
            Notification notification = new Notification();
            notification.Id = Guid.NewGuid();
            notification.Title = "Invoice Status Change";
            notification.Body = $"Invoice status has changed to {statusName}";
            notification.Importance = "Normal";
            notification.DateCreated = DateTime.Now;
            notification.DateUpdated = DateTime.Now;
            await NotificationRepository.AddNotificationAsync(notification);
        }
        return result;
    }
    catch (Exception ex)
    {
        Trace.WriteLine(ex.Message.ToString());
        throw;
    }
}
```

Slika 4.18. Programski kod za dodavanje obavijesti u bazu podataka

```
5 references
public async Task<bool> AddNotificationAsync(Notification notification)
{
    using (NpgsqlConnection conn = new NpgsqlConnection(Helper.connectionString))
    {
        NpgsqlCommand cmd = new NpgsqlCommand("INSERT INTO \"Notification\" (\"Id\", \"Title\", \"Body\", \"Importance\", \"DateCreated\
                                              "(@Id, @Title, @Body, @Importance, @DateCreated, @DateUpdated, @IsRead)", conn);

        cmd.Parameters.AddWithValue("@Id", notification.Id);
        cmd.Parameters.AddWithValue("@Title", notification.Title);
        cmd.Parameters.AddWithValue("@Body", notification.Body);
        cmd.Parameters.AddWithValue("@Importance", notification.Importance);
        cmd.Parameters.AddWithValue("@DateCreated", notification.DateCreated);
        cmd.Parameters.AddWithValue("@DateUpdated", notification.DateUpdated);
        cmd.Parameters.AddWithValue("@IsRead", notification.IsRead);

        conn.Open();

        int numberOfAffectedRows = await cmd.ExecuteNonQueryAsync();
```

Slika 4.19. Programski kod dodavanja obavijesti u bazu podataka

Prilikom dohvaćanja radnog naloga, sustav provjerava je li došlo do nepoštivanja roka plaćanja. Ako se takav slučaj dogodi, sustav automatski obavještava administratora. Slika 4.20 prikazuje programski kod koji poziva metodu za dodavanje nove obavijesti u bazu podataka.

```
try
{
    goodsCalculation = await goodsCalculationService.GetSpecificGoodsCalculationAsync(id);
    List<GoodsCalculation> goodsCalculations = new List<GoodsCalculation>{
        goodsCalculation
    };
    List<GoodsCalculationRest> goodsCalculationsRest = new List<GoodsCalculationRest>();

    goodsCalculationsRest = SetModelToRest(goodsCalculations);
    if (goodsCalculationsRest != null)
    {
        TimeSpan timeUntilDeadline = goodsCalculation.PaymentDeadline - DateTime.Now;

        if (timeUntilDeadline.TotalDays < 0)
        {
            Notification notification = new Notification();
            notification.Id = Guid.NewGuid();
            notification.Title = "Payment Deadline Is Overdue!";
            notification.Body = $"Payment Deadline of order #{goodsCalculation.OrderNumber} was due " +
                $"{Math.Abs(((int)timeUntilDeadline.TotalDays))} days ago";
            notification.Importance = "Urgent";
            notification.DateCreated = DateTime.Now;
            notification.DateUpdated = DateTime.Now;

            await notificationService.AddNotificationAsync(notification);
        }
    }
    return Request.CreateResponse(HttpStatusCode.OK, goodsCalculationsRest);
```

Slika 4.20. Metoda za obavijesti ako nije poštivan rok plaćanja

Nakon što je obavijest dodana u bazu podataka, automatski se ažurira i tablica korisničkog sučelja za prikazivanje svih obavijesti. Programski kod dohvaćanja obavijesti prati identičan koncept kao pri dohvaćanju jednog od tipova naloga no s različitim informacijama. Kao što je spomenuto,

sustav razdvaja pročitane i nepročitane obavijesti putem stupca „IsRead“ te kao što je prikazano na slici 4.21, drugačije su prikazani redovi pročitanih i nepročitanih obavijesti.

```
useEffect(() => {
  // Fetch data using Axios
  axios.get('https://localhost:44368/api/Notification')
    .then(response => {
      setData(response.data);
      const unreadNotifications = response.data.filter(notification => !notification.isRead);
      setUnreadNotifications(unreadNotifications.length);
    })
    .catch(error => {
      console.error('Error fetching data:', error);
    });
}, [clickedRow]);
```

Slika 4.21. Programski kod zahtjeva za dohvaćanje svih obavijesti

4.2.2. Odjava korisnika iz sustava

Funkcionalnost odjave korisnika iz sustava je komponenta implementirana na korisničkom sučelju tako da se iz lokalne pohrane brišu token i uloga korisnika. Tako, bez tokena iz lokalne memorije neprijavljenom korisniku prikazivat će se samo one komponente u trenutku kada u lokalnoj pohrani ne postoji token. Na slici 4.22 prikazan je programski kod gumba za odjavu korisnika iz sustava koji uklanja token iz lokalne memorije.

```
function LogoutForm() {
  const navigate = useNavigate();

  const handleLogout = async () => {
    try {
      localStorage.removeItem('token');
      localStorage.removeItem('role');
      navigate('/');
    } catch (error) {

      console.error('Logout error:', error);
    }
  };

  return (
    <form className="log-out-button" onSubmit={handleLogout}>
      <button className="log-out-button" type="submit">Logout</button>
    </form>
  );
}

export default LogoutForm;
```

Slika 4.22. Prikaz komponente za odjavu korisnika iz sustava

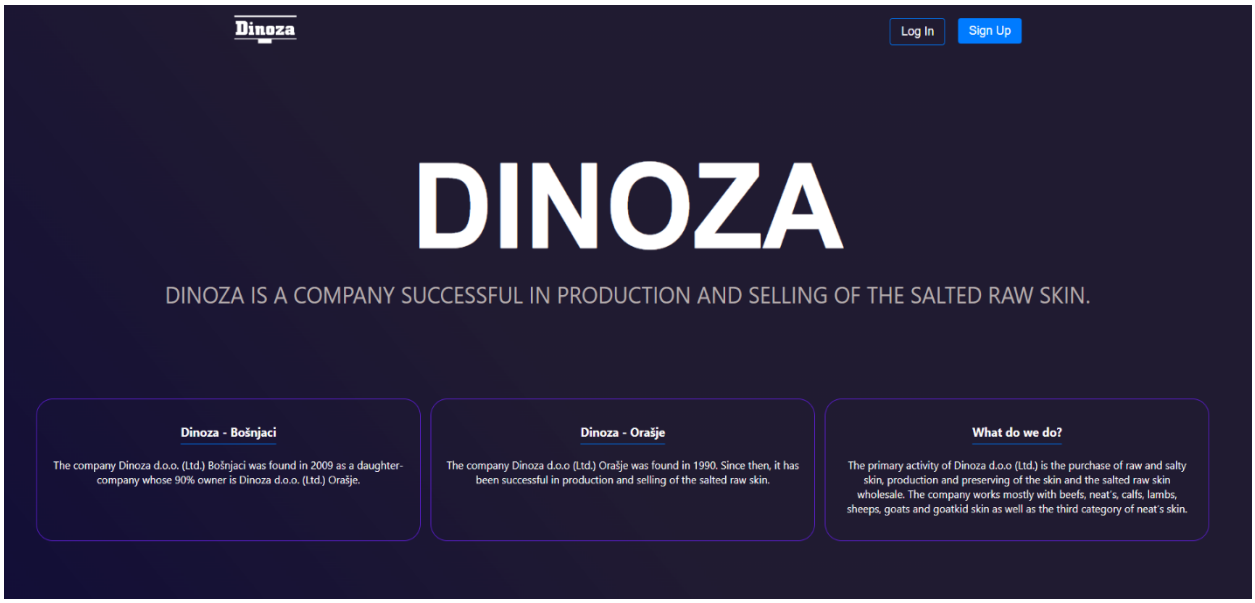
5. NAČIN RADA APLIKACIJE, ISPITIVANJE I ANALIZA REZULTATA

U ovom poglavlju prikazano je korisničko sučelje web sustava i opisan način korištenja. Također, kroz tri testna slučaja provedeno je ispitivanje web sustava i načinjena analiza ispunjavanja funkcionalnosti.

5.1. Način rada aplikacije

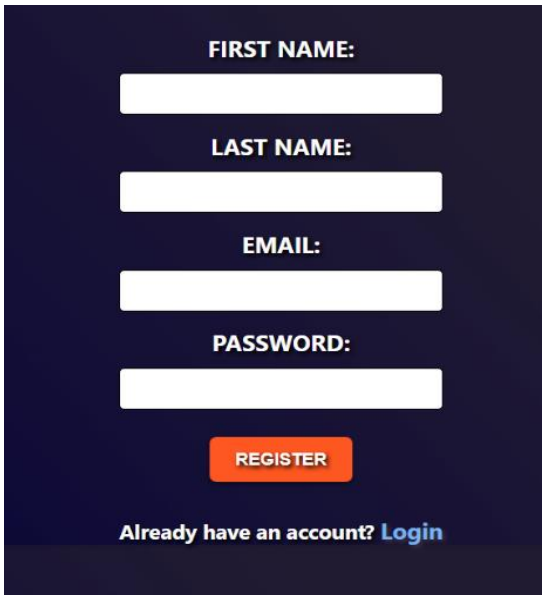
5.1.1. Prijava i registracija korisnika

Otvaranjem web aplikacije korisniku se prikazuje odredišna stranica s osnovnim informacijama, galerijom i kontakt podacima. Kao što je prikazano na slici 5.1, u gornjem desnom kutu se nalaze gumbi za prijavu ili registraciju korisnika.

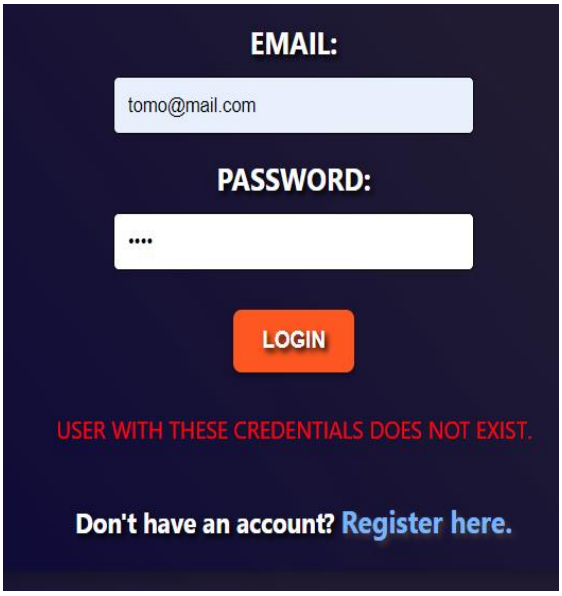


5.1. Prikaz početne stranice web sustava

Prema slici 5.2, za registraciju su potrebni ime, prezime, email i lozinka. E-mail je u bazi podataka jedinstven, odnosno ne mogu se stvoriti dva ista korisnika s istom e-mail adresom. Prilikom stvaranja korisnika, lozinka će u bazu podataka biti spremljena u obliku kriptirane lozinke, odnosno upravitelj baze podataka web sustava neće imati pristup lozinkama registriranih klijenata. Na slici 5.3 prikazan je obrazac za prijavu, gdje se uz netočan navod e-maila ili lozinke javlja greška da korisnik s tim navedenim podacima za prijavu ne postoji u bazi podataka. Nakon što je korisnik uspješno registriran, moguće je prijavljivanje u aplikaciju u jednu od dvije uloge, ulogu administratora ili ulogu klijenta.



Slika 5.2. Obrazac za registraciju korisnika



Slika 5.3. Obrazac za prijavu korisnika

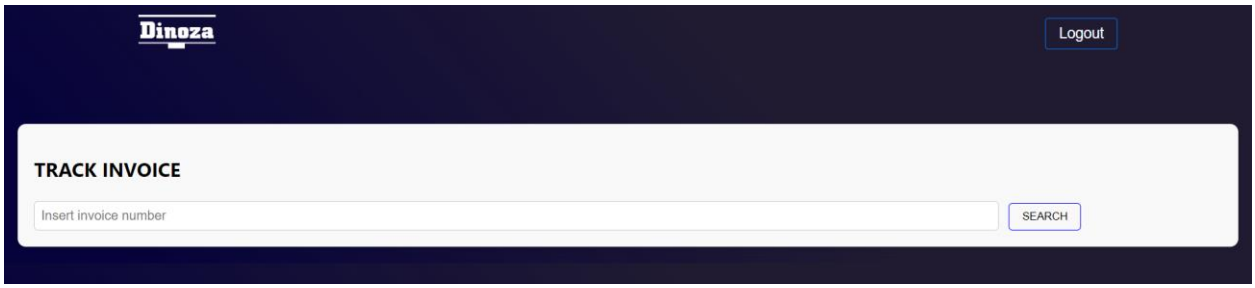
Na slici 5.4. ilustriran je prikaz spremanja korisnika u bazu podataka.

Id	FirstName	LastName	Email	Password	RoleId
ecdfb622-2f12-48ec-8e7c-cc70c6d94512	Tomislav	Lucic	tomo@mail.com	HWX9CyDZbOSO9cyi14HSxQ==:jMNrpfb1YtVFmRmyIeksArpyU7OZt	c72d4c65-4d08-49ab-84e1-6cb3341f8bb6

Slika 5.4. Prikaz spremanja korisnika u bazu podataka

5.1.2. Klijentska nadzorna ploča

Ako se korisnik prijavi u web sustav u ulogu klijenta, prikazat će mu se klijentska nadzorna ploča gdje je ostvaren zahtjev za pretraživanjem i preuzimanjem faktura koje se nalaze u bazi podataka. Slika 5.5 prikazuje klijentsku nadzornu ploču, gdje korisnik u traku za pretraživanje unosi broj fakture, te ako faktura s tim brojem postoji prikazat će se detalji o fakturi.

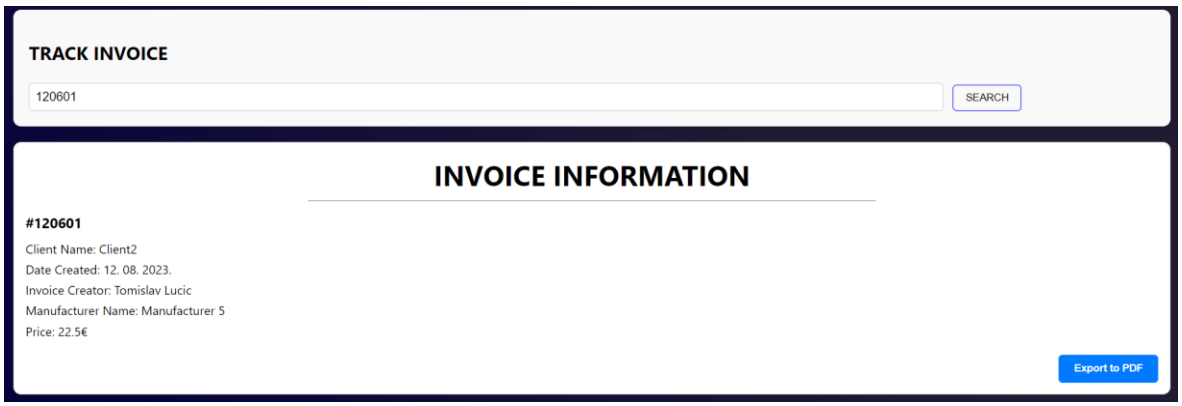


Slika 5.5. Klijentska nadzorna ploča

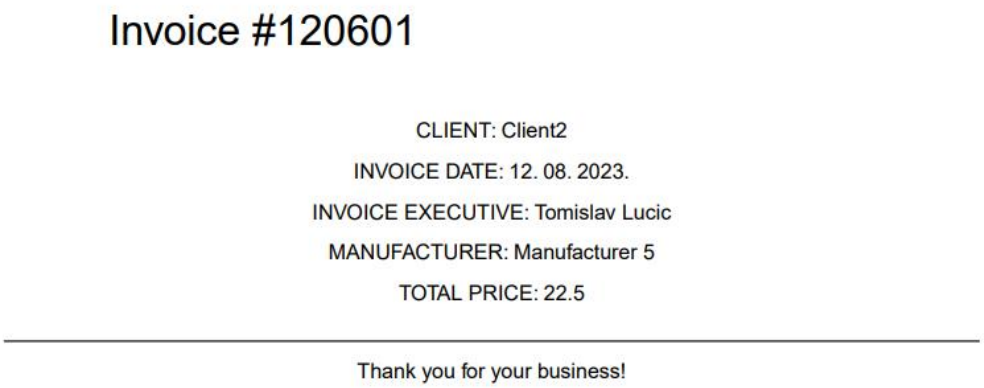
Prilikom upisivanja broja fakture, u bazu podataka se šalje zahtjev za pretragom u tablici faktura gdje postoji faktura s tim brojem te na sljedećim slikama (slika 5.6 i 5.7) može se vidjeti ishod u oba slučaja.

Slika 5.6. Neuspješan pokušaj pretrage fakture

Prema slici 5.7, u donjem desnom kutu korisniku je dostupan izvoz fakture u pdf formatu zbog potrebe ispisivanja faktura. Kao što je prikazano na slici 5.8, izvoz fakture sadrži identične informacije o fakturi kao na klijentskoj nadzornoj ploči.



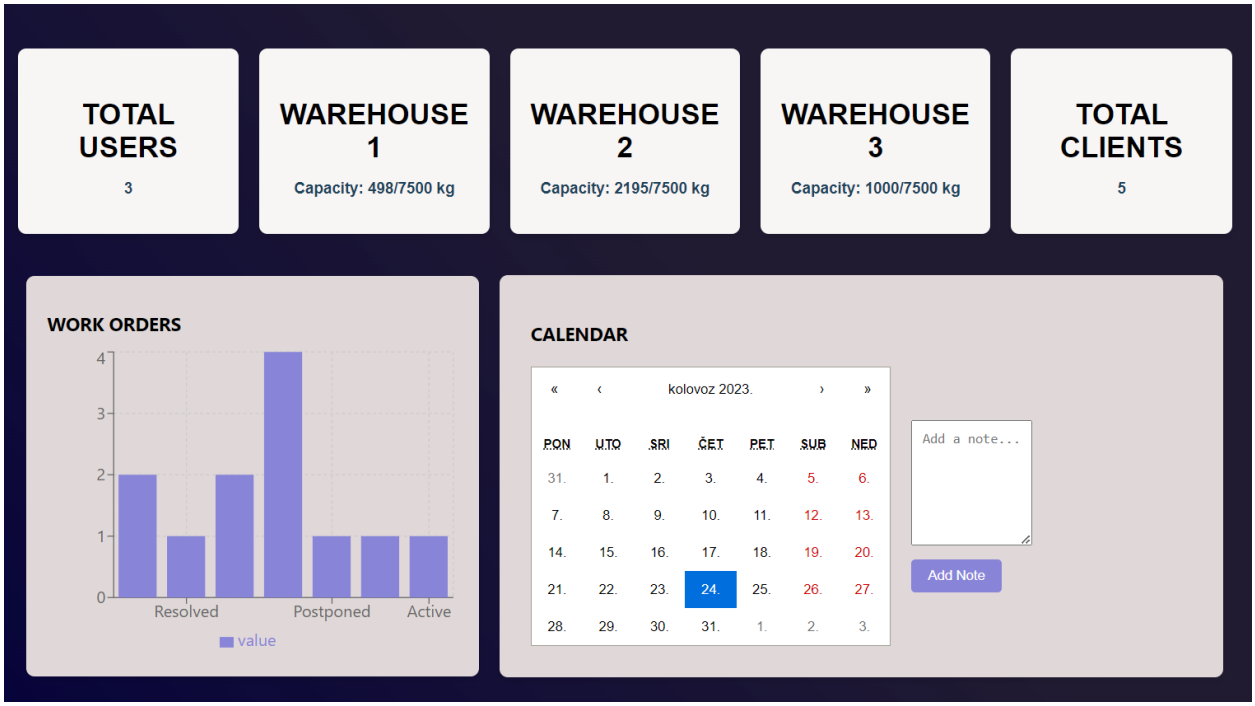
Slika 5.7. Uspješan pokušaj pretrage i prikaz detalja fakture



Slika 5.8. Prikaz sadržaja fakture u obliku pdf formata

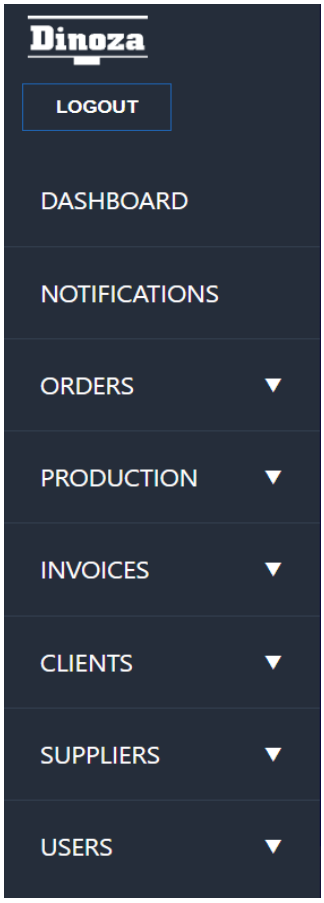
5.1.3. Administratorska nadzorna ploča

Ako je korisnik prijavljen u ulogu administratora, početni prikaz bit će znatno drugačiji te s više mogućih akcija nego što je to u slučaju klijenta. Kao što je prikazano na slici 5.9, administrator ima uvid u stanja skladišta, sveukupan broj registriranih korisnika i postojećih klijenata te kalendar koji služi za aktivno praćenje aktivnosti.

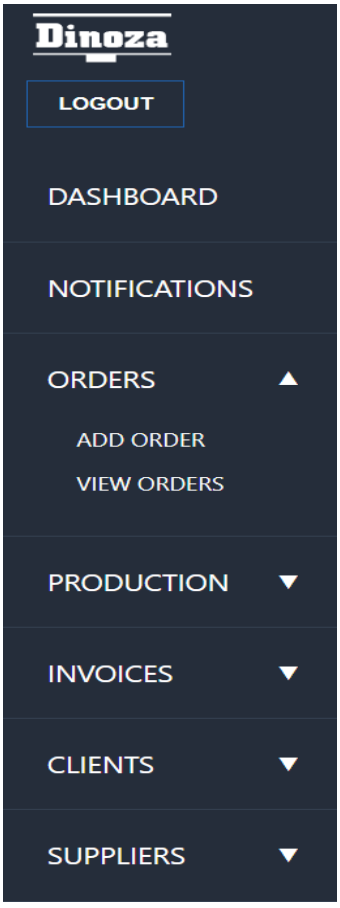


Slika 5.9. Prikaz početne administratorske nadzorne ploče

Na lijevoj strani nadzorne ploče, nalazi se bočna traka (eng. „*sidebar*“), koja se prikazuje na svakoj stranici web sustava, te služi kao pomoć pri korištenju web sustava. Prema slici 5.10, na bočnoj traci osim gumba za odlazak na početnu stranicu ili odjavu iz aplikacije nalaze se i komponente za pregled obavijesti, radnih naloga, naloga za proizvodnju, faktura, dobavljača, klijenata i korisnika. Primjećuje se da neke od tih komponenti, kao što je ilustrirano na primjeru prikazanom na slici 5.11, koriste padajuće izbornike koji omogućavaju brz pristup pregledu ili dodavanju relevantnih podataka. Ovakav dizajn omogućava usklađenost svih ključnih komponenti sustava, čineći ih lako dostupnima i omogućavajući korisnicima glatku tranziciju između različitih funkcionalnosti. Cjelokupna bočna traka doprinosi povezanosti svih dijelova sustava, omogućavajući jednostavno prebacivanje između različitih aspekata aplikacije. Intuitivno pozicioniranje komponenti navigacije dodatno olakšava korisnicima orijentaciju i upotrebu, pridonoseći ukupnom korisničkom iskustvu.



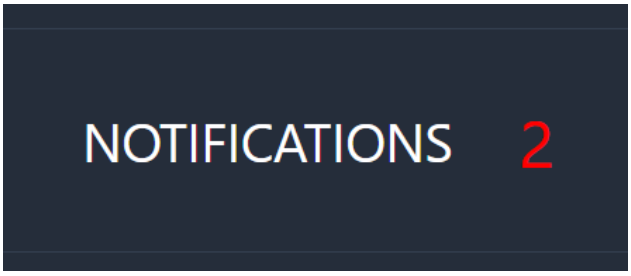
Slika 5.10. Prikaz bočne trake



Slika 5.11. Prikaz padajućeg izbornika bočne trake

5.1.4. Prikaz obavijesti

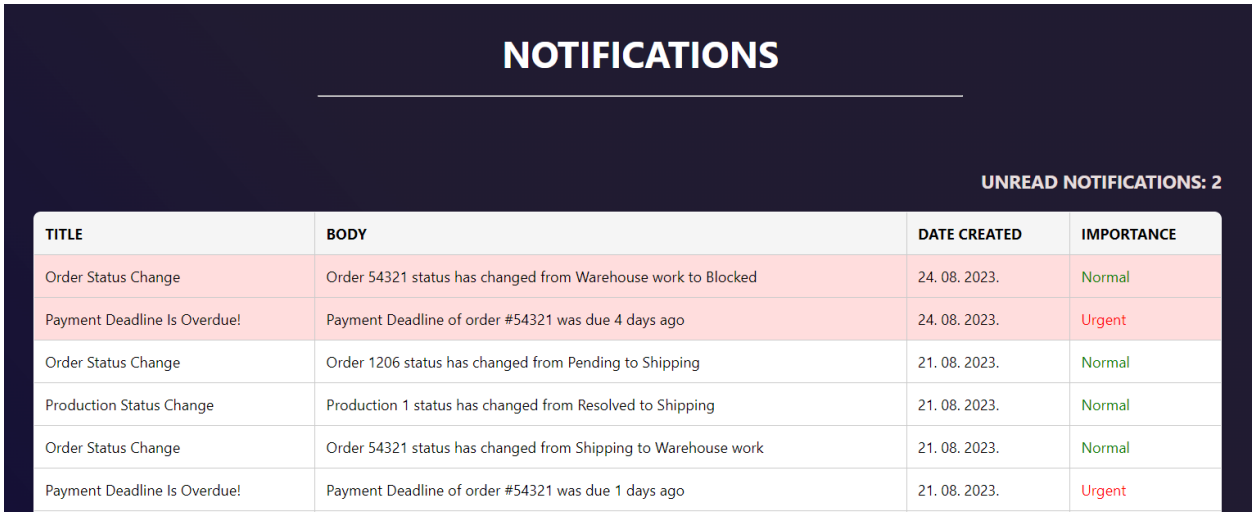
Prva komponenta koja se nalazi na padajućem izborniku su obavijesti koje administratoru ukazuju na promjenu statusa na nekom od naloga u bazi podataka. Osim promjene statusa, obavijesti dolaze i u slučaju kada dolazi do kašnjenja u plaćanju resursa koje pristižu od dobavljača na stanje u skladište. Na slici 5.12 prikazuje se komponenta obavijesti na bočnoj traci ako postoje nepročitane obavijesti.



Slika 5.13. Prikaz nepročitanih obavijesti na bočnoj traci

Na stranici obavijesti, administratoru se u tabličnom obliku prikazuju sve obavijesti, te kao što je vidljivo na slici 5.14 obavijesti se sastoje od sljedećih informacija: naslov, sadržaj obavijesti, datum nastanka te važnost obavijesti. Iz slike 5.15 zaključuje se da su nepročitane obavijesti

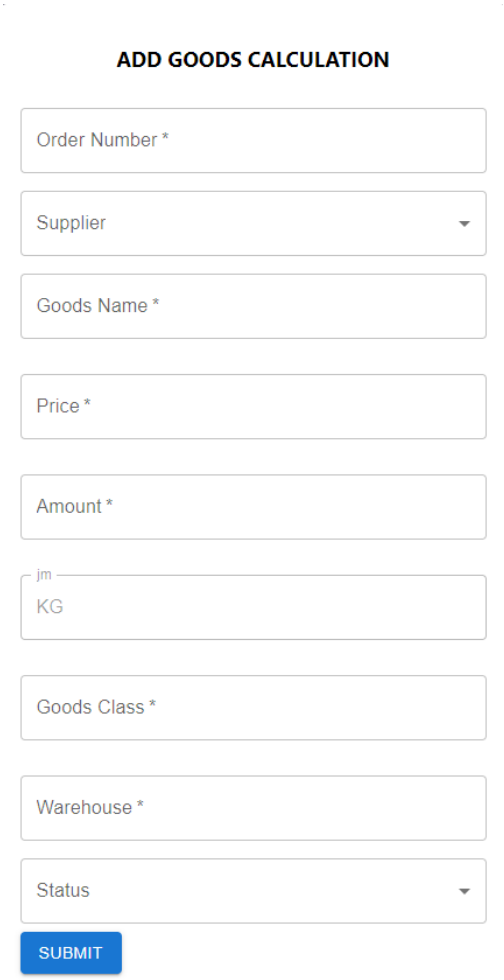
drugačije boje od pročitanih kako bi privukle pozornost administratora, te pritiskom na nepročitanu obavijest, red mijenja boju u neutralnu.



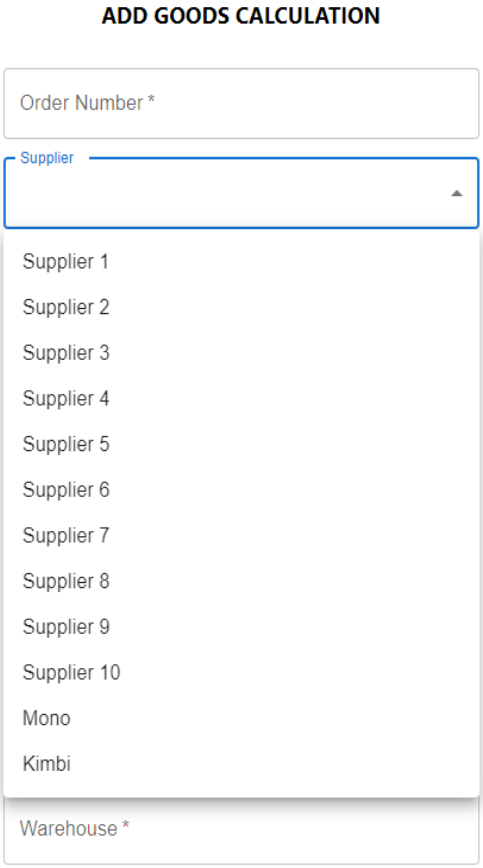
Slika 5.15. Prikaz stranice obavijesti

5.1.5. Radni nalozi

Nakon obavijesti, na bočnoj traci se nalazi komponenta koja se odnosi na stvaranje novih te pregled postojećih radnih naloga u bazi podataka. Radni nalozi služe za praćenje informacija o resursima koje pristižu od dobavljača u jedno od skladišta. Kao što je prikazano na slici 5.16 pri stvaranju novog radnog naloga potrebno je navesti sljedeće podatke: broj radnog naloga, izabrati dobavljača, ime resursa, cijenu resursa, količinu resursa, kvalitativnu klasu resursa, skladište u koje će se pohraniti te trenutni status radnog naloga. Dobavljač se odabire kroz padajući izbornik gdje su ponuđeni svi dobavljači koji se trenutno nalaze u bazi podataka.

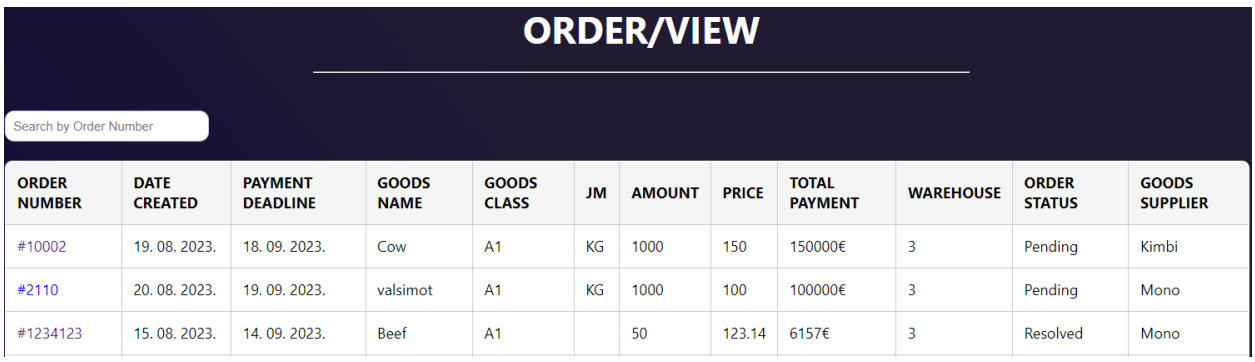


Slika 5.16. Dodavanje novog radnog naloga



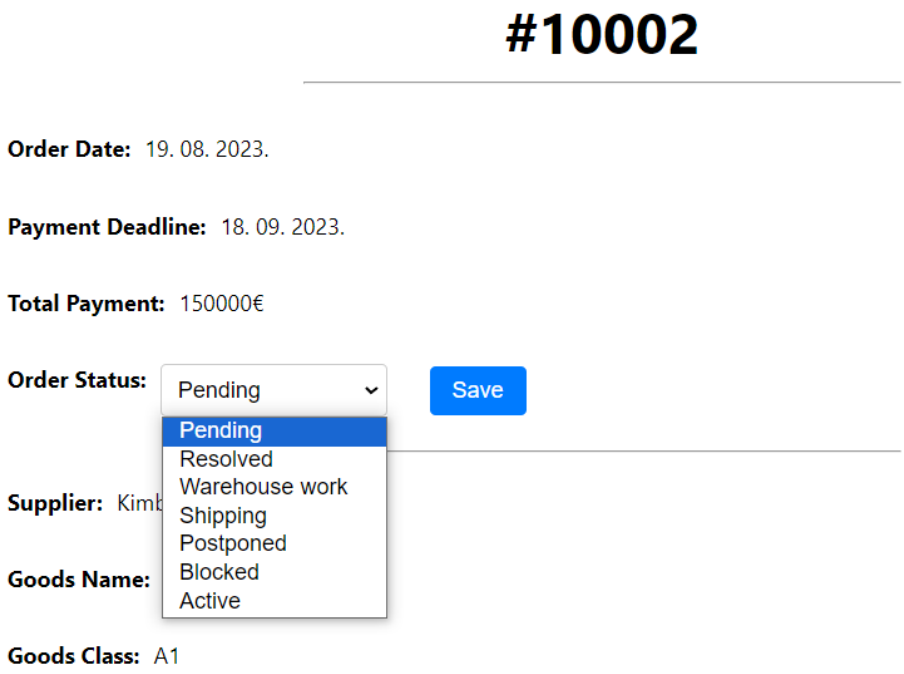
Slika 5.17. Odabir dobavljača

Osim dodavanja novog radnog naloga, administrator ima pristup svim radnim nalozima u bazi podataka te također svakom radnom nalogu pojedinačno. Na slikama 5.18 i slika 5.19 prikazana je tablica radnih naloga te primjer pojedinačnog naloga gdje administrator može promijeniti status naloga.



ORDER NUMBER	DATE CREATED	PAYMENT DEADLINE	GOODS NAME	GOODS CLASS	JM	AMOUNT	PRICE	TOTAL PAYMENT	WAREHOUSE	ORDER STATUS	GOODS SUPPLIER
#10002	19. 08. 2023.	18. 09. 2023.	Cow	A1	KG	1000	150	150000€	3	Pending	Kimbi
#2110	20. 08. 2023.	19. 09. 2023.	valsimot	A1	KG	1000	100	100000€	3	Pending	Mono
#1234123	15. 08. 2023.	14. 09. 2023.	Beef	A1		50	123.14	6157€	3	Resolved	Mono

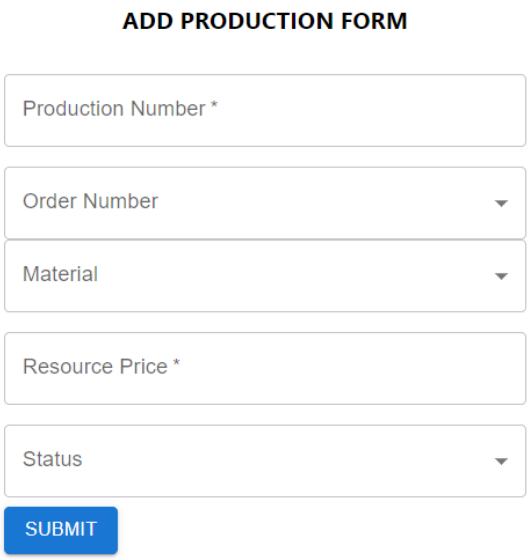
Slika 5.18. Prikaz radnih naloga



Slika 5.19. Prikaz promjene statusa radnog naloga

5.1.6. Nalog za proizvodnju

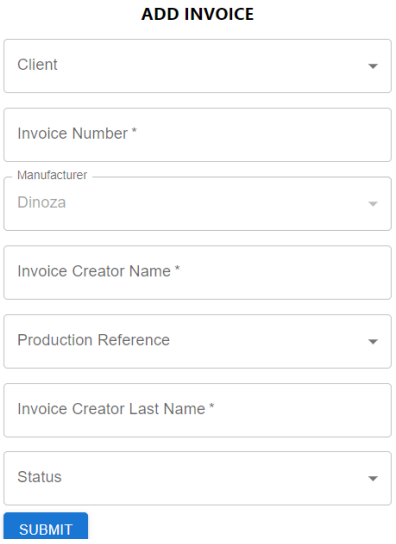
Slično kao i prikaz radnih naloga, tablični prikazi za naloge za proizvodnju i detaljni pregled pojedinačnih naloga dijele slične karakteristike, ali s različitim informacijama prilagođenim njihovoj svrsi. Na primjeru prikaza naloge za proizvodnju, struktura i funkcionalnost podsjećaju na prikaz radnih naloga. Prikaz tablice naloge za proizvodnju omogućava uvid u relevantne podatke i detalje o svakom nalogu. Ovdje, umjesto informacija vezanih uz radne naloge, fokus je na aspektima proizvodnje. U isto vrijeme, pojedinačni pregled naloge za proizvodnju omogućava korisnicima dublji uvid u pojedinosti naručenih proizvoda. Prilikom dodavanja novog naloga za proizvodnju, kao što je ilustrirano na slici 5.20, potrebno je unijeti osnovne informacije koje će pridonijeti preciznoj obradi i praćenju naloga. To uključuje unos broja naloga za proizvodnju, povezivanje s odgovarajućim radnim nalogom, specificiranje materijala koji će biti korišteni, te određivanje cijene gotovog proizvoda spremnog za prodaju. Najbitniji parametar je trenutni status radnog naloga, koji omogućava točno praćenje napretka.



Slika 5.20. Obrazac za dodavanje naloga za proizvodnju

5.1.7. Fakture

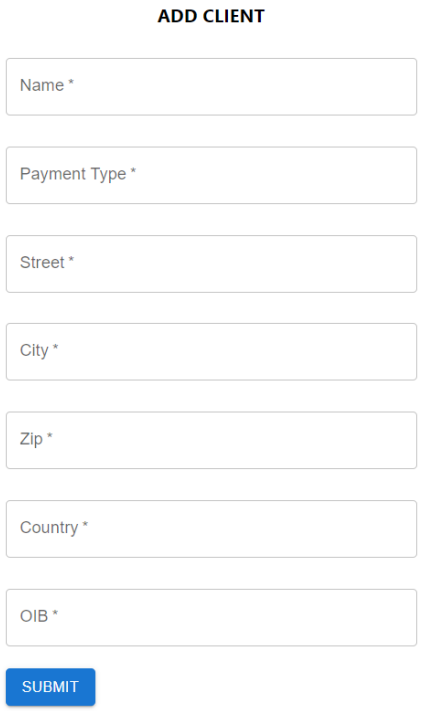
Slično kao i kod prikaza radnih naloga i naloge za proizvodnju, prikaz faktura slijedi konzistentan obrazac. Tablični prikaz faktura pruža pregled ključnih informacija o svakoj fakturi, dok pojedinačni pregled omogućava dublji uvid u detalje svake pojedinačne transakcije. Prilikom dodavanja nove fakture, kao što je prikazano na slici 5.21, korisnicima je omogućeno unositi ključne podatke kako bi se faktura precizno generirala i evidentirala. Unos relevantnih informacija, kao što su odabir klijenta, broj fakture, ime i prezime stvaratelja fakture, referenca na nalog za proizvodnju te trenutni status fakture, osigurava točno i pravilno izdanu fakturu.



Slika 5.21. Obrazac za dodavanje novih faktura

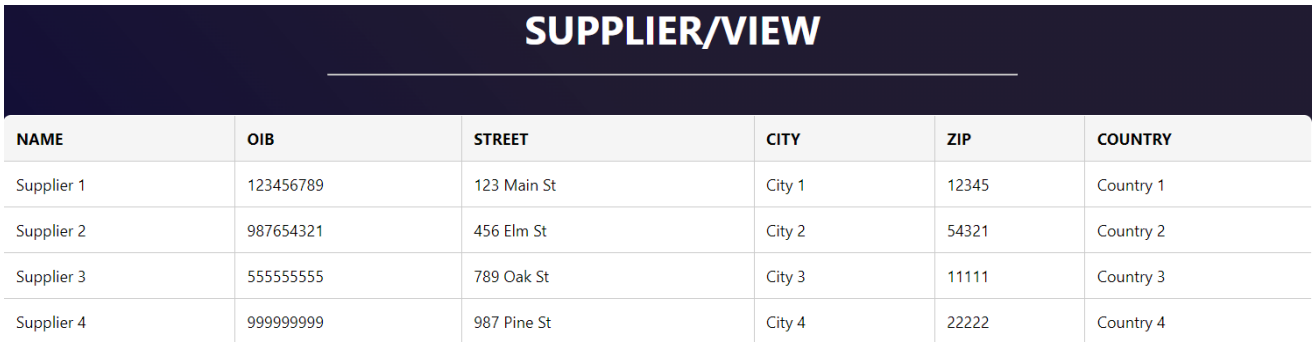
5.1.8. Klijenti i dobavljači

Osim dodavanja, praćenja i promjene statusa naloga svih faza proizvodnog ciklusa, administrator također može pregledati i dodavati nove klijente i dobavljače. Prema slici 5.22, mogu se vidjeti podaci koji su potrebni za dodavanje novog klijenta u bazu, te isti princip obrasca za dodavanje klijenata koristi se i kod dodavanja dobavljača.



Slika 5.22. Obrazac za dodavanje klijenta

Nakon što je dodan novi klijent ili dobavljač, korisnik pri stvaranju nekog od naloga može odabrati samo postojeće klijente i dobavljače iz baze podataka. Kao što je prikazano na slici 5.23, koncept prikazivanja svih dobavljača i klijenata identičan je kao što je u slučaju prikazivanja naloga, no bez mogućnosti otvaranja stranice pojedinačnog dobavljača.

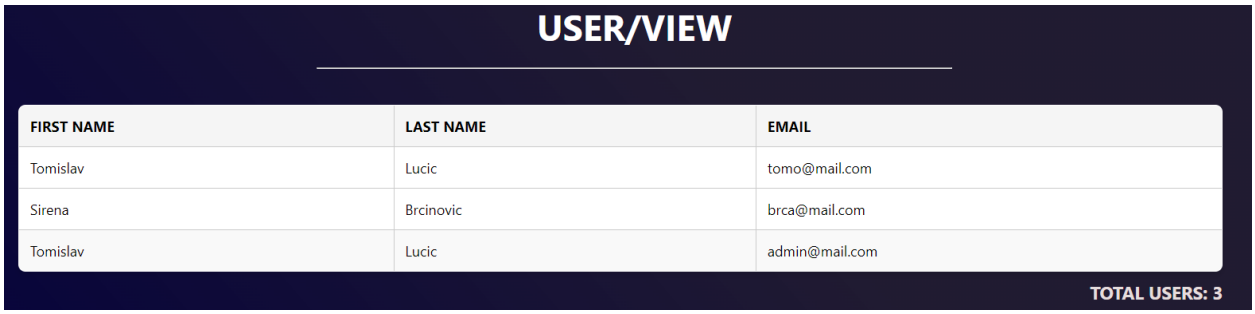


NAME	OIB	STREET	CITY	ZIP	COUNTRY
Supplier 1	123456789	123 Main St	City 1	12345	Country 1
Supplier 2	987654321	456 Elm St	City 2	54321	Country 2
Supplier 3	555555555	789 Oak St	City 3	11111	Country 3
Supplier 4	999999999	987 Pine St	City 4	22222	Country 4

Slika 5.23. Prikaz svih dobavljača

5.1.9. Prikaz registriranih korisnika

Posljednji element u bočnoj traci je posvećen korisnicima, odnosno stranici koja omogućava pregled svih registriranih korisnika. Koncept prikaza korisnika vodi se obrascem koji se već koristi u svim tabličnim prikazima unutar web sustava. Ova stranica daje pregled ključnih informacija o svakom korisniku, uključujući ime, prezime i e-mail adresu. Prikaz je kreiran s ciljem olakšavanja upravljanja korisničkim računima te omogućavanja administratorskom timu da brzo i lako pregleda osnovne podatke svakog korisnika. To je posebno korisno u situacijama gdje je potrebno brzo identificirati korisnike ili provjeriti njihove kontakte. Prikaz ovog koncepta ilustriran je na slici 5.24, gdje možete vidjeti kako su informacije o korisnicima organizirane u tabličnom formatu.



Slika 5.24. Prikaz svih registriranih korisnika

5.2. Ispitivanje web aplikacije

Prikladno i sistematično testiranje web sustava ključno je za utvrđivanje točnosti implementiranih funkcionalnosti. Kroz tri slučaja, provesti će se testiranje web sustava u različitim situacijama.

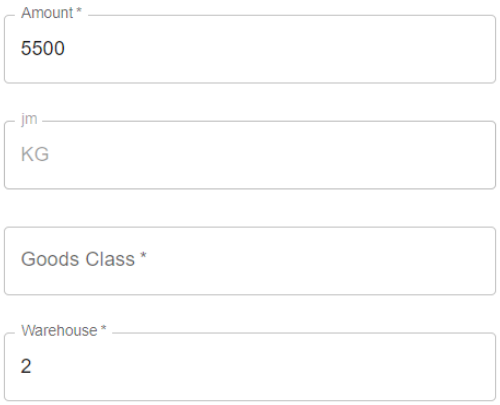
5.2.1. Provjera kapaciteta skladišta prilikom stvaranja radnog naloga

Ovaj testni slučaj provjerava može li se resurs pohraniti u odabrano skladište prilikom unosa u obrazac. Na slici 5.26. ilustrirani su koraci koji su potrebni za provođenje ovog ispitivanja:

- Korisnik se prijavljuje kao administrator
- Unos novog radnog naloga
- Unos količine koja ne može stati u odabrano skladište

Očekivani rezultat web sustava je izbacivanje greške u obliku prozora koja ukazuje da skladište s tim brojem ne može primiti tu količinu resursa.

Slika 5.25. Prikaz stanja skladišta



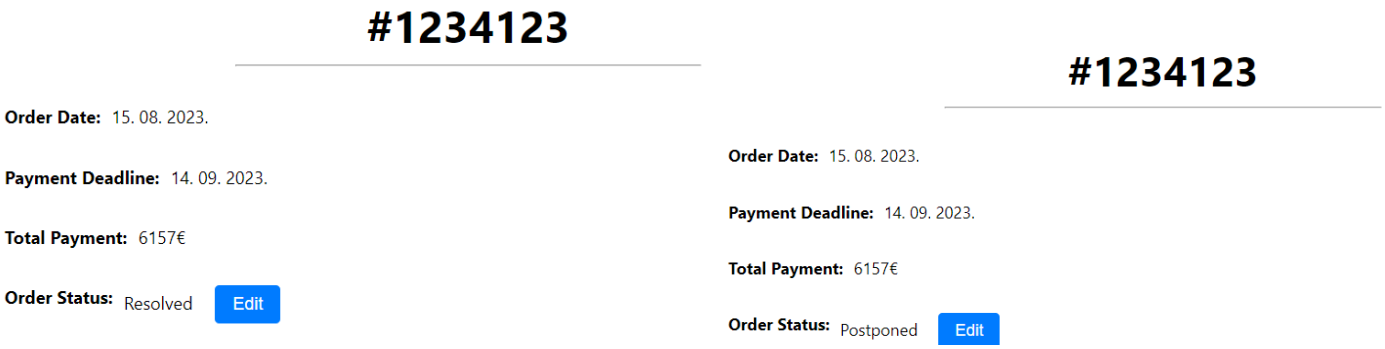
Slika 5.26. Unos količine resursa koje odabrano skladište ne može pohraniti

5.2.2. Provjera obavijesti prilikom promjene statusa naloga

Sljedeći testni slučaj ispituje funkcionalnost obavještavanja korisnika ako je došlo do promjene statusa na jednom od naloga. Na slici 5.27 prikazani su koraci koji su potrebni za provođenje ovog testiranja:

- Korisnik prijavljen kao administrator
- Prikaz detalja o pojedinom nalogu
- Promjena trenutnog statusa u novi

Očekivani rezultat ovog testnog slučaja je automatsko slanje obavijesti gdje će biti prikazani podaci o broju naloga te prethodni i trenutno postavljeni status.



Slika 5.28. Promjena statusa naloga

5.2.3. Provjera obavijesti prilikom nepoštivanja roka plaćanja

Posljednji testni slučaj provjera funkcionalnost obavještavanja korisnika ako se pregleda nalog gdje se nije ispoštovao krajnji rok plaćanja. Na slici 5.29 ilustrirani su koraci prilikom testiranja navedene funkcionalnosti:

- Korisnik prijavljen kao administrator
- Pregled detalja naloga gdje nije ispoštovan rok plaćanja

Očekivani rezultat testiranja je automatsko slanje obavijesti s naznakom hitno u stupcu važnosti obavijesti, koje dodatno upozorava korisnika kako nije na vrijeme otplatio

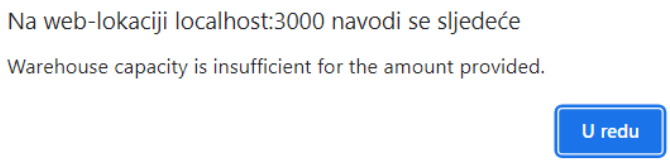
Slika 5.29. Prikaz naloga čiji je rok plaćanja bio prije 4 dana

5.3. Analize rezultata ispitivanja i testiranja

Nakon ispitivanja, analiza rezultata ključna je komponenta u procesu razvoja web sustava jer pruža uvid u to poklapaju li se dobiveni rezultati s očekivanim. Ovim putem utvrđuje se točnost funkcionalnosti web sustava te je moguće identificirati potencijalne nedostatke i pružiti povratne informacije koje će dalje unaprijediti razvoj web sustava.

5.3.1. Analiza prvog testnog slučaja

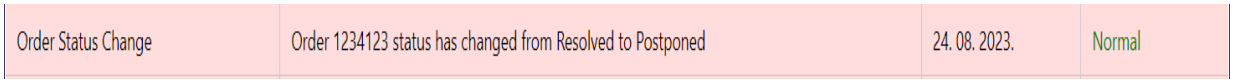
Tijekom testiranja, zaključeno je da sustav uspješno prepoznaje kada korisnik pokuša unijeti određenu količinu resursa u skladište koje nema dovoljnu količinu prostora. Kao što je očekivano i prikazano na slici 5.30, sustav je reagirao prikazujući poruku pogreške u obliku skočnog prozora. Obavijest je ukazala na nedostatak kapaciteta te onemogućila unos naloga u bazu podataka te tako potvrdila točnost funkcionalnosti raspoređivanja resursa u skladište. Raspoređivanje se vrši kroz postupak kapacitetnog raspoređivanja, odnosno, sustav najprije provjerava dostupnost skladišta, nakon toga raspoređuje ulazni resurs u, ako je to moguće, odabrano skladištu. U suprotnom, sustav obavještavanjem vrši preraspoređivanje gdje se resurs raspoređuje u jedno od dostupnih skladišta.



Slika 5.30. Prikaz obavijesti ako nema dovoljno prostora u skladištu

5.3.2. Analiza drugog testnog slučaja

Nakon provođenja testiranja drugog testnog slučaja, zaključeno je da korisnici uspješno mogu promijeniti status naloga te da sustav pouzdano prikazuje obavijest o promjeni statusa. Kao što je prikazano na slici 5.31, obavijesti su se prikazale korisniku kao što je bilo i očekivano, uz detalje o nalogu te prethodnom i trenutno postavljenom stanju. Ova funkcionalnost doprinosu jasnom praćenju promjena statusa i brzoj komunikaciji unutar web sustava.



Slika 5.31. Prikaz obavijest ako je došlo do promjene statusa

5.3.3. Analiza trećeg testnog slučaja

Nakon testiranja, zaključeno je da web sustav uspješno prepoznaje naloge s kašnjenjem u plaćanju te da pravilno prikazuje obavijest korisnicima. Kao što je prikazano na slici 5.32, obavijest prikazuje sve potrebne informacije te označava obavijest kao „hitno“, čime se naglašava važnost informacije. Ovako ostvarena funkcionalnost omogućuje brzu reakciju korisnika na neplaćene naloge, što je ključan faktor za financijsku transparentnost tvrtke.

Slika 5.32. Prikaz obavijesti neplaćenog naloga

Analizom testnih slučaja potvrđene su očekivane vrijednosti. To je pokazatelj točnosti sustava i njegove usklađenosti s planiranim funkcionalnostima. Sustav pravilno interpretira i obrađuje korisničke zahtjeve te ispunjava korisnikova očekivanja. Ova dosljednost između funkcionalnosti i korisničkih zahtjeva značajno doprinosi povećanju povjerenja između klijenta i proizvođača. Kroz dokazanu sposobnost ispunjenja obećanih performansi, korisnici stječu povjerenje u proizvođača da će sustav i dalje ispravno funkcionirati.

6. ZAKLJUČAK

Cilj ovog završnog rada bio je unaprijediti i automatizirati praćenje proizvodnog ciklusa tvrtke specijalizirane za preradu kože u gotove proizvode. Kroz ovu studiju slučaja, postignut je značajan napredak u olakšavanju poslovnog procesa. Za korisnike u ulozi klijenta, omogućena je jednostavna pretraga i preuzimanje faktura u obliku PDF dokumenata. S druge strane, administratorima je pružena mogućnost dodavanja, upravljanja statusima te praćenja radnih naloga, naloga za proizvodnju i faktura. Dodatno, administratori mogu upravljati dobavljačima, klijentima i korisnicima sustava. Razvoj ovog web sustava temeljio se na programskom jeziku C# i React.js biblioteci, uz primjenu HTML-a, CSS-a i Javascripta za oblikovanje korisničkog sučelja. PostgreSQL baza podataka koristila se za pohranu informacija o nalozima, klijentima, dobavljačima i registriranim korisnicima.

Provedeno ispitivanje sustava putem tri testna slučaja potvrdilo je točnost i funkcionalnost aplikacije. Analiza rezultata istraživanja potvrdila je uspješnost implementiranih funkcionalnosti, dokazavši kvalitetu razvijenog rješenja. Mogućnosti daljnjeg unapređenja ovog web sustava su mnogobrojne. Jedna od perspektiva je razvoj mobilne aplikacije koja bi olakšala komunikaciju i praćenje naloga između dobavljača, proizvođača i klijenata. Dodatno, implementacija praćenja dostave i isporuke bi omogućila bolju transparentnost i praćenje resursa i proizvoda u stvarnom vremenu. Time bi se unaprijedila pouzdanost i efikasnost poslovnih procesa te bi tvrtka mogla bolje odgovoriti na zahtjeve tržišta.

LITERATURA

[1] Tehnička enciklopedija, www.tehnika.lzmk.hr, 1980, pp. 7. svezak, kožarstvo.

[2] Ilabo, Rafael, Building a digital factory by deploying PackOS in Avon - *Case Study*, 2022.

[3] TheAppSolutions, "Engineering, Functional vs Non-functional Requirements in Systems, 2023."[Mrežno]. Dostupno na: https://theappsolutions.com/blog/development/functional-vs-non-functional-requirements/. [Pokušaj pristupa: 26.7.2023.]

[4] V.T. i. J.-C. Billaut, Multicriteria Scheduling Theory, Models and Algorithms, Heidelberg: Springer-Verlag Media, Berlin, 2006.

[5] C. Zhuoyi, J. Limin, Z. Weihua, "Research and Development of the Long Distance Coach Management System Based on ASP.net Technology," *2nd International Conference on Consumer Electronics, Communications and Networks (CECNet)*, Yichang, China, 21-23 April 2023, pp. 1992–1995, 2012.

[6] "Why Flowcharts Are Important For Web Development – JojoCms,". Dostupno na: https://www.jojocms.org/why-flowcharts-are-important-for-web-development/, [Pokušaj pristupa: 26.7.2023].

[7] "What is PostgreSQL? – Amazon Web Services" [Mrežno]. Dostupno na: https://aws.amazon.com/rds/postgresql/what-is-postgresql/. [Pokušaj pristupa: 26.7.2023.].

[8] R. Hat, "What is a REST API?, 8 Svibanj 2020." [Mrežno]. Dostupno na: https://www.redhat.com/en/topics/api/what-is-a-rest-api. [Pokušaj pristupa: 26 srpanj 2023.]

[9] B. Carter, "HTML Architecture, a Novel Development System (HANDS): An Approach for Web Development," u 2014 Annual Global Online Conference on Information and Computer Technology, Louisville, KY, USA, 03-05 prosinac 2014, pp. 2-4, 2014.

[10] B. Wagner, "Learn C# Programming - for Beginning Developers, Developers new to C#, and Experienced C# / .NET developers," 2020.

[11] T. Mikac, Planiranje i upravljanje proizvodnjom, Rijeka, 2007.

[12] M.R. Peasy, "Production Tracking Software." [Mrežno]. [Pokušaj pristupa 25 lipanj 2023.]

[13] L.E.R. Daniel, "An IoT Platform for Production Monitoring in the Aerospace Manufacturing Industry," Journal of Cleaner Production, Vol. 368, rujan 2022.

[14] P.K. Burgess K., "Path-Clearing Policies for Flexible Manufacturing Systems," IEEE Transactions on Automatic Control, Vol. 44, Issue 3, pp. 1-4, ožujak 1999.

[15] G.E.L. Vihar, »Internet of Things (IoT): A Literature Review,« *Journal of Computer and Communications*, Vol. 3, Issue 5, p. 164, 2015.

[16] G.R.G. Gerhard, "Production Scheduling, Management and Control," Practical E-Manufacturing and Supply Chain Management, Newnes, pp. 243-269, 2004.

[17] D. Damjanović, "Primena internet of things (IOT) rešenja u upravljanju održavanjem u proizvodnji," Zbornik Međunarodnog kongresa o procesnoj industriji – Procesing, Beograd, Srbija, 1-2 lipanj 2017 , pp. 115-120, 2017.

[18] Vanado, "PAUK," 2019. [Mrežno]. Dostupno na: https://www.hgk.hr/documents/pauk-prezentacija-hr5b9b91fab639a.pdf. [Pokušaj pristupa: 12 lipanj 2023.]

[19] Simplilearn, "The Best Guide to Know What Is React," 7 Veljača 2023. [Mrežno]. Available:https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs#what_is_react. [Pokušaj pristupa 26 srpanj 2023.]

[20] Encora, "Postman Tutorial for Beginners to Perform API Testing," 29 Lipanj 2021. [Mrežno]. Dostupno na:https://www.encora.com/insights/what-is-postman-api-test. [Pokušaj pristupa 26 srpanj 2023.]

SAŽETAK

U ovom završnom radu osmišljen je i razvijen web sustava prilagođenog tvrtki za preradu kože, koja omogućuje učinkovitu potporu i praćenje proizvodnog ciklusa, upravljanje nalozima, klijentima, dobavljačima i korisnicima. Za programsku implementaciju web sustava korišteni su programski jezik C#, biblioteka React.js, a za rukovanje bazom podataka PostgreSQL. Web aplikacija omogućuje prijavu korisnika kao administratora i klijenta, te pristupe različitim funkcionalnostima. Administratoru se prikazuju obavijesti te može upravljati radnim nalozima, nalozima za proizvodnju, fakturama, dobavljačima, klijentima i korisnicima. Klijentima je omogućena pretraga i preuzimanje faktura. Ispitivanje aplikacije kroz tri testna slučaja potvrdilo je uspješnost svih implementiranih funkcionalnosti. Ova web aplikacija pridonosi olakšavanju poslovnih procesa tvrtke za preradu kože, unaprjeđujući učinkovitost i transparentnost u upravljanju proizvodnjom, raspoređivanju resursa u skladišta te preraspoređivanju statusa naloga.

Ključne riječi: proizvodni ciklus, raspoređivanje i preraspoređivanje, tvrtka za preradu kože, upravljanje radnim nalozima, web sustav

ABSTRACT

In this thesis, a web system tailored to a leather processing company has been conceived and developed, providing efficient support and monitoring of the production cycle, order management, customers, suppliers, and users. For the programming implementation of the web system, the C# programming language and the React.js library were used, with PostgreSQL serving as the database management system. The web application enables user login as both an administrator and a client, granting access to various functionalities. Administrators are alerted with notifications and can manage work orders, production orders, invoices, suppliers, customers, and users. Clients have the ability to search for and download invoices. Testing of the application through three test cases confirmed the successful implementation of all intended features. This web application contributes to streamlining the business processes of the leather processing company, enhancing efficiency and transparency in production management, resource allocation to warehouses, and reallocation of order statuses.

Keywords: production cycle, scheduling and rescheduling, leather processing company, work order management, web system

ŽIVOTOPIS

Tomislav Lučić rođen je 12. lipnja 2001. godine u Županji. Nakon pohađanja Osnovne škole Mate Lovraka u Županji, upisuje matematičku gimnaziju u Županji. Nakon završetka srednje škole upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, smjer računarstvo, programski inženjer. Od programskih jezika poznaje C#, C i Python, a od programskih okolina i okvira React, Django i .NET.

PRILOZI

Prilog 1. Završni rad u datoteci docx

Prilog 2. Završni rad u datoteci pdf

Prilog 3. Programski kod projekta web sustava