UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

MAGISTRSKO DELO

RAZVOJ SISTEMA ZA SLEDLJIVOST IZDELKOV TER VPELJAVA NA PROIZVODNO LINIJO

JAN BRATINA

2019
MAGISTRSKO DELO
JAN BRATINA

UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

Magistrsko delo

Razvoj sistema za sledljivost izdelkov in vpeljava na proizvodno linijo

(Development of product traceability system and implementation on the production line)

Ime in priimek: Jan Bratina
Študijski program: Računalništvo in informatika, 2. stopnja
Mentor: doc. dr. Peter Rogelj
Delovni somentor: univ. dipl. inž. rač. in inf. Damijan Mihelj

Koper, avgust 2019

Ključna dokumentacijska informacija

Ime in PRIIMEK: Jan BRATINA

Naslov magistrskega dela: Razvoj sistema za sledljivost izdelkov ter vpeljava na proizvodno linijo

Kraj: Koper

Leto: 2019

Število listov: 68 Število slik: 17 Število tabel: 2

Število referenc: 20

Mentor: doc. dr. Peter Rogelj

Delovni somentor: univ. dipl. inž. rač. in inf. Damijan Mihelj

UDK: 004.05(043.2)

Ključne besede: SCADA, programsko inženirstvo, Java, Spring Boot, Angular, sledljivost izdelkov, informacijski sistem

Izvleček:

V magistrskem delu je predstavljen potek razvoja sistema za sledljivost izdelkov in vpeljava tega sistema na določeno proizvodno linijo. Sistemi, ki zagotavljajo sledljivost izdelkov na proizvodni liniji, so v zadnjih letih postali nepogrešljiv del proizvodne linije. Ti sistemi so se razvili zaradi potrebe po višanju kvalitete izdelkov in zaradi zniževanja stroškov v primeru reklamacij. Vse večje so tudi zahteve kupcev po spremljanju izdelkov na proizvodni liniji. Razlog za razvoj sistema za sledljivost proizvodnje je, da bi v podjetju imeli enoten sistem za spremljanje sledljivosti proizvodnje, ki bi bil sposoben podpirati različne zahteve dobaviteljev strojne opreme in kupcev. Razvoj sistema je prikazan s fazami programskega inženirstva, ki so v magistrskem delu podrobneje predstavljene. Strežniški del sistema je razvit v programskem jeziku Java s pomočjo ogrodja Spring Boot. Odjemalni del pa je razvit z pomočjo ogrodja Angular. Povezavo s PLC-krmilniki na proizvodni liniji omogoča OPC-strežnik KEPServerEX, podatki pa se shranjujejo v lokalno podatkovno bazo Microsoft SQL Server. Glavni cilj magistrskega dela je predstaviti razvoj in delovanje sistema za sledljivost izdelkov na določeni proizvodni liniji. Cilji razvoja sistema pa so zmanjšati operativne stroške linije, zmanjšati stroške, ki nastanejo v primeru reklamacij, in nudenje popolne podpore proizvodnji.

Key words documentation

Name and SURNAME: Jan BRATINA

Title of the thesis: Development of product traceability system and implementation on the production line

Place: Koper

Year: 2019

Number of pages: 68 Number of figures: 17 Number of tables: 2

Number of references: 20

Mentor: Assist. Prof. Peter Rogelj, PhD

Co-Mentor: Damijan Mihelj, B.Sc.

UDK: 004.05(043.2)

Keywords: SCADA, software engineering, Java, Spring Boot, Angular, product traceability, information system

Abstract:

This master's thesis presents the development process of the product traceability system and the implementation of this system on the production line. In recent years, systems that ensure product traceability have become an indispensable part of the production line. These systems have been developed to ensure better product quality and cost reduction in case of a complaint. Requirements of the buyers to support product traceability on the production line are constantly increasing. The main reason for the development of product traceability system is that the company would like to have a unified system for production monitoring which would be able to support different requirements of hardware suppliers and customers. The development of the system is presented by software engineering phases. The server part of the system is developed using Java programming language with Spring Boot framework. The client side is developed using Angular framework. Connection to PLC controllers on the production line is enabled by the OPC server KEPServerEx and the data is stored in the local database Microsoft SQL Server. The main goal of the master's thesis is to present the development and operation of the product traceability system on a particular production line. The goals of the development of such system are to reduce operating costs, to reduce the costs in case of complaints and to provide complete support for production.

KAZALO VSEBINE

KAZALO PREGLEDNIC

KAZALO SLIK

SEZNAM KRATIC

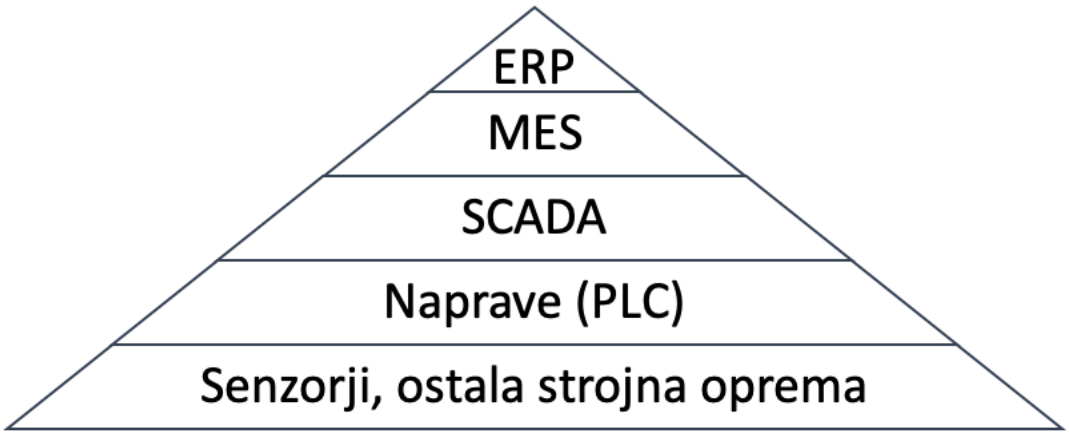
SCADA	Sistem, namenjen nadzorovanju in krmiljenju tehnološkega procesa z računalnikom (angl. Supervisory Control And Data Acquisition)
MES	Sistem za upravljanje proizvodnje (angl. Manufacturing Execution System)
ERP	Sistem za planiranje virov podjetja (angl. Enterprise Resource Planning)
PLC	Programabilni logični krmilnik (angl. Programmable Logic Controller)
OPC	Odprta platforma za komunikacije (angl. Open Platform Communications)
SQL	Strukturiran povpraševalni jezik (angl. Structured Query Language)
CSV	Datoteka, v kateri so vrednosti ločene z vejico (angl. Comma Separated Values)
RAID	Redundantno diskovno polje (angl. Redundant Array of Independent Disks)
SD	Izmenljiva pomnilniška kartica (angl. Secure Digital)
TCP/IP	Standardiziran sklad internetnih protokolov (angl. Transmission Control Protocol, Internet Protocol)
DCOM	Objektni model porazdeljenih komponent (angl. Distributed Component Object Model)
REST	Arhitektura za izmenjavo podatkov med spletnimi storitvami (angl. Representational State Transfer)
JDBC	Javanska knjižnica za dostop do podatkovne baze (angl. Java Database Connectivity)
JSON	Objektni zapis JavaScript (angl. JavaScript Object Notation)
DAO	Objekti za dostop do podatkov (angl. Data Access Object)
HTTP	Protokol za izmenjavo hiperteksta (angl. Hypertext Transfer Protocol)
HTML	Jezik za označevanje hiperteksta (angl. Hypertext Markup Language)
CSS	Prekrivni slogi (angl. Cascading Style Sheet)
CLI	Vmesnik z ukazno vrstico (angl. Command Line Interface)
STS	Razvijalno okolje za Spring (angl. Spring Tool Suite)
UPS	Brezprekinitveno napajanje (angl. Uninterruptible Power Supply)
SVN	Sistem za dokumentiranje sprememb in različic datotek (angl. Subversion)
YAML	Jezik za konfiguracijske datoteke (angl. YAML Ain't Markup Language)
XML	Zapis za izmenjavo strukturiranih dokumentov na spletu (angl. extensible markup language)
JDBC	Standard za baze podatkov v javi (angl. Java Database Connectivity)

ZAHVALA

Zahvaljujem se mentorju doc. dr. Petru Roglju za vso strokovno pomoč in nasvete pri izdelavi magistrskega dela. Zahvalil bi se tudi podjetju MAHLE Electric Drives Slovenia za finančno pomoč tekom študija in za uporabo teme magistrskega dela. Na tem mestu bi se zahvalil še sodelavcem, posebej delovnemu somentorju Damijanu Mihelju za vso strokovno pomoč in izkazano zaupanje pri implementaciji sistema. Zahvaljujem se tudi punci, družini in prijateljem za vso podporo, potrpljenje in spodbujanje med študijem.
Nazadnje bi se rad zahvalil še sošolcem Aneju Marušiču, Blažu Gombaču, Patriku Kocijančiču in Tomažu Grižonu, s katerimi smo se tekom magistrskega študija medsebojno spodbujali in si pomagali.

1 UVOD

Avtomatizacija ter digitalne naprave so dobile svoje mesto na proizvodnih linijah že v tretji industrijski revoluciji med letoma 1950 in 1970. Znana je tudi pod imenom digitalna revolucija in je tesno povezana z razvojem informacijske in komunikacijske tehnologije [9]. V današnjem času so povezane in pametne naprave našle svoje mesto tudi v proizvodnji. Take pametne naprave pa so lahko same sebi namen, če ne obstaja sistem, ki s temi napravami upravlja ter iz njih pridobiva vse potrebne podatke. Tu je največji problem prav upravljanje naprav, zato so se razvili tako imenovani SCADA (angl. Supervisory Control And Data Acquisition) sistemi, ki omogočajo nadzorovanje in upravljanje naprav ter zbiranje podatkov s teh naprav. Sistem SCADA zbrane podatke zabeleži v podatkovno bazo, do katere lahko nato dostopajo ostali informacijski sistemi, kot na primer sistem za upravljanje proizvodnje MES (angl. Manufacturing Execution System). Ta sistem nato skrbi za prikazovanje ter analizo zbranih podatkov. Sistem MES tudi priskrbi vhodne podatke sistemu SCADA, kot na primer podatke o delovnem nalogu, podatke o navodilih za delo itn. Te informacijske sisteme nadzira sistem za planiranje virov podjetja ERP (angl. Enterprise Resource Planning). Ta sistem je tesno vpleten v poslovni svet in tudi tesno povezan z ostalimi informacijskimi sistemi, kot na primer logistični informacijski sistem, sistem za upravljanje življenjskega cikla izdelka itd. Hierarhijo proizvodnih informacijskih sistemov ter pripadajoče strojne opreme na najnižjem nivoju vidimo na Sliki 1.
Tok podatkov med informacijskimi sistemi v podjetjih poteka na naslednji način. Podjetje prejme naročilo stranke, ki se ga nato obdela v ERP-sistemu. V tem sistemu se nato planira proizvodnjo, pripravi zaloge materiala za izdelavo naročenega izdelka ter izdela delovne naloge. Podatki o delovnem nalogu se nato prenesejo v sistem MES, ki nadzoruje proizvodnjo izdelka ter pripravi podatke iz delovnega naloga za prenos v sistem SCADA. Ta sistem nato upravlja z napravami in iz njih pridobiva podatke ter jih nato zabeleži v podatkovno bazo. Sistem MES zbrane podatke obdela, izdela poročila in te podatke (na primer o porabljenem materialu) pošlje nazaj v ERP-sistem.



Slika 1: Hierarhija informacijskih sistemov ter povezane strojne opreme v proizvodnji [2].

Sistem ERP nato knjiži porabljen material ter podatke o končanih izdelkih pripravi za obdelavo v ostalih informacijskih sistemih [4].

V magistrskem delu bo predstavljen razvoj sistema za sledljivost proizvodnje na nivoju SCADA ter implementacijo sistema na določeno novo proizvodno linijo. Obravnavali bomo tudi podatke o tem, kako je sistem umeščen v ostale poslovne informacijske sisteme. Sistem za sledljivost proizvodnje je narejen v sodelovanju s podjetjem, v katerem je tudi ta proizvodna linija postavljena. Proizvodna linija je sestavljena iz petnajstih delovnih mest in dodatnega delovnega mesta za pakiranje izdelkov. Vsako od teh petnajstih delovnih mest je namenjeno izvajanju točno določene operacije. Osnovni potek dela na splošnem delovnem mestu je:

- delavec vzame izdelek, na katerem je dvodimenzionalna črtna koda (angl. data matrix), ki vsebuje serijsko številko izdelka in ga postavi na odložišče,
- PLC-krmilnik sproži optično branje dvodimenzionalne kode izdelka in zabeleži prebrano serijsko številko v spomin,
- PLC-krmilnik začne z izvajanjem sekvence operacije na delovnem mestu in čaka na SCADA-sistem, da sporoči dovoljenje za izvajanje operacije na tem izdelku,
- SCADA-sistem preveri, če je izdelek namenjen na to operacijo, če je v PLC-krmilniku prisoten pravilni recept za delo na izdelku in če je delovno mesto opremljeno s pravilnim materialom,
- SCADA-sistem sporoči PLC-krmilniku dovoljenje za izvajanje operacije na izdelku in ta nadaljuje z izvajanjem sekvence operacije,
- po zaključeni operaciji PLC-krmilnik začne z izvajanjem sekvence za zapis procesnih parametrov v SCADA-sistem,
- SCADA-sistem se odzove na zahtevo, zapiše podatke v podatkovno bazo in sporoči PLC-krmilniku, da so podatki uspešno zapisani,
- operacija na izdelku je tako zaključena in delovno mesto je pripravljeno na prihod novega izdelka.

Potek dela na delovnem mestu pakiranja izdelkov se razlikuje od poteka dela na splošnem delovnem mestu, zato bo tudi v nadaljevanju drugače obravnavan. Ena glavnih razlik je, da to delovno mesto ni povezano s PLC-krmilnikom. Posledično je optični bralnik črtnih kod povezan na strežnik SCADA-sistema in ne na PLC-krmilnik. Potek dela na delovnem mestu pakiranja izdelkov je opisan v nadajevanju:

- delavec preko uporabniškega vmesnika odpre novo pakirno enoto za izdelke,
- delavec vzame izdelek, na katerem je dvodimenzionalna črtna koda in ga želi postaviti v pakirno enoto,
- SCADA-sistem sproži optično branje dvodimenzionalne kode izdelka in preveri, ali se optično prebran izdelek lahko postavi v pakirno enoto,
- delavcu se na uporabniškem vmesniku izpiše status, ali je izdelek uspešno zabeležen v pakirni enoti ali ne,

- v kolikor delavec opazi, da je z že zapakiranim izdelkom nekaj narobe, lahko tak izdelek preko uporabniškega vmesnika izbriše iz pakirne enote.

Potek razvoja sistema za sledljivost proizvodnje bo v magistrskem delu predstavljen s fazami programskega inženirstva. Začeli bomo z definicijo problema in nato nadaljevali s študijo izvedljivosti. Nato bomo analizirali in definirali zahteve. Temu sledi načrtovanje ter izvedba, zaključili pa bomo s fazo testiranja ter z obratovanjem.

Cilj magistrskega dela je predstaviti problem sledenja proizvodnje in se podrobneje spoznati z razvojem takega sistema za določeno novo proizvodno linijo. Razvoj tega sistema je potekal v sodelovanju s podjetjem in ta sistem že deluje na eni izmed proizvodnih linij. Torej smo večino strežniškega dela našega sistema uporabili iz prejšnjega projekta, vendar pa je bilo treba vseeno nekaj funkcij na novo razviti. Večja funkcionalnost sistema na strežniški strani, ki jo je bilo potrebno razviti, je sistem za upravljanje z recepti, saj na prejšnjih proizvodnih linijah ni bilo potrebe po tej funkcionalnosti. Večina našega dela pa je zajemala razvoj odjemalnega dela sistema, zato je tudi temu delu v nadaljevanju magistrskega dela posvečene največ pozornosti. Naše delo je zajemalo tudi namestitev in konfiguracijo vse strojne in programske opreme na strežniku sistema.

2 DEFINICIJA PROBLEMA

Zaradi potrebe po višanju kvalitete izdelkov je treba zagotoviti popolno sledljivost izdelkov na proizvodni liniji ter beležiti vse dogodke, ki se na proizvodni liniji zgodijo. Prav tako so zahteve kupcev po spremljanju izdelkov na proizvodni liniji vse večje. Kupci želijo informacije o tem, na kateri proizvodnji liniji je bil izdelek narejen, na katerih delovnih mestih je bil izdelek prisoten, o zbranih procesnih podatkih, o kvaliteti izdelkov na posameznih delovnih mestih ter sledenju porabljenega materiala. Starejše proizvodne linije ne nudijo možnosti spremljanja proizvodnje po izdelkih, saj izdelki navadno niso označeni s serijskimi številkami. Shranjevanje procesnih podatkov je zaradi starosti strojne opreme praktično nemogoče, zato se lahko zgodi, da v izdelek ni vstavljen ves predvideni material, saj nimamo materialne sledljivosti. Zbiranje vseh teh podatkov pa ni pomembno samo za končnega kupca, temveč je v pomoč tudi proizvajalcu, saj prikaže jasnejšo sliko o uspešnosti proizvodnje ter zagotavlja višjo kakovost izdelkov. Sistem mora zagotavljati sledenje na nivoju serijske številke za končne izdelke ter za vitalne komponente v izdelku. Za drobni material ali standardne komponente pa je dovolj, da zagotavljamo sledljivost na nivoju šarže. V kolikor pride do reklamacije nekega izdelka, lahko proizvajalec enostavno poišče podatke o izdelku ter si na ta način bistveno zmanjša stroške reklamacije oziroma odpoklica izdelka. Če nimamo sledljivosti materiala vsaj na nivoju šarže, so lahko stroški reklamacij oziroma odpoklica izdelkov zelo visoki. Zagotoviti je treba tudi sledljivost pakirnih enot, torej moramo vedeti, v kateri pakirni enoti so prisotni določeni izdelki. Na ta način natanko vemo, kdaj je izdelek zapustil podjetje ter bil predan prevozniku.

V podjetju se kot ERP-sistem uporablja poslovni sistem SAP, ki je že dobro vpeljan v proizvodni proces, torej je smiselno, da sta sistema povezana s stališča pridobivanja osnovnih informacij o procesu. Uporaba podatkov iz sistema SAP je smiselna tudi s stališča upravljanja ter vzdrževanja ključnih podatkov o proizvodnji in o izdelku s strani odgovornih oseb. Na ta način imamo zagotovilo, da bodo vhodni podatki v sistem vedno pravilni. Eden od pomembnejših delov sistema je digitalen prikaz dokumentacije izdelka na zaslonih, saj se je v preteklosti dokumentacijo za vsak izdelek ter za vsako delovno mesto tiskalo na papir. Ta dokumentacija se je lahko zamešala ali pa izgubila, kar se je odražalo tudi na kvaliteti izdelkov. Z uvedbo novega sistema pa moramo imeti zagotovilo, da se za trenutni izdelek prikazuje pravilna dokumentacija. Treba je tudi zagotoviti, da ima strojna oprema na posameznem delovnem mestu pravilna navodila za delo v obliki recepta. Recept je torej dokument v JSON-formatu, v katerem je določeno, s kakšnimi parametri mora strojna oprema delovati. Kratek primer recepta je viden na Sliki 2.

S sledenjem proizvodnji lahko pridobimo tudi informacije o času cikla posameznega delovnega mesta. Na ta način lahko spremljamo, katero delovno mesto na proizvodni liniji je ozko grlo in nato sprožimo ukrepe za optimiziranje delovnega mesta. Z beleženjem pravilnih podatkov o izdelku na posameznih delovnih mestih pridobimo tudi podatek o

kvaliteti izdelka. Potrebno je spremljati, ali se na katerem delovnem mestu izdela veliko slabih izdelkov oziroma izmeta ter se nato sproži ustrezne ukrepe za odpravo morebitne napake na napravi ali na receptu. Sporočanje odgovornim osebam o napakah in dogodkih na proizvodni liniji nato prevzame MES-sistem.

Glavni cilj implementacije sistema za spremljanje proizvodnje je, da bi bil v podjetju prisoten enoten sistem, tesno povezan z obstoječimi informacijskimi sistemi in sposoben podpirati različne zahteve dobaviteljev strojne opreme ter kupcev. Preostali cilji projekta so zmanjšati operativne stroške linije, zmanjšati stroške, ki nastanejo v primeru reklamacij, in nudenje popolne podpore proizvodnji. Rezultat sistema pa je popolna sledljivost izdelkov na liniji, nižji časi preurejanja proizvodne linije ob prihodu novega tipa izdelka, zagotavljanje višje kvalitete izdelkov in beleženje dogodkov na liniji.

```
{
    "name": "DM10-Rec",
    "type": "object",
    "properties": [{
          "name": "ReceptIme",
          "idType": 1,
          "type": "string",
          "value": "primerRecepta"
    }, {
          "name": "Parameteri",
          "type": "object",
          "properties": [{
                "name": "Param1",
                "idType": 2,
                "type": "integer",
                "min": 0,
                "max": 1000,
                "value": "99"
          }, {
                "name": "Par2",
                "idType": 3,
                "type": "decimal",
                "min": 0,
                "max": 1000,
                "value": "99.7"
          }]
    }]
}
```

Slika 2: Primer enostavnega recepta za delovno mesto.

3 ŠTUDIJA IZVEDLJIVOSTI

Namen študije izvedljivosti je, da se čimbolj optimalno odločimo za izvedbo projekta. V tem koraku moramo ugotoviti, ali smo zmožni projekt tehnično izpeljati, ga izpeljati v nekem smiselnem časovnem okviru, kakšni so stroški izvedbe projekta v primerjavi s pričakovanimi koristmi ipd. [15] Obravnavani projekt je obsežen in posledično predstavlja tudi veliko investicijo za podjetje, zato je pomembno projekt temeljito analizirati. Torej če v študiji izvedljivosti ugotovimo, da je projekt smiseln oziroma da prinaša neko dodano vrednost, lahko gremo v razvoj programske rešitve.

3.1 EKONOMSKA IZVEDLJIVOST

Na trgu že obstaja veliko ponudnikov različnih sistemov za sledljivost proizvodnje, ki podpirajo različne načine komuniciranja z napravami. Največkrat te sisteme ponujajo dobavitelji proizvodne linije oziroma posamezne naprave. V primeru, da imamo enega dobavitelja za celotno linijo, ki skrbi za sledljivost izdelka znotraj linije, ni problema z integracijo v obstoječe poslovne informacijske sisteme. Vendar pa je največkrat proizvodna linija sestavljena iz naprav različnih dobaviteljev, ki podpirajo samo določene tipe komunikacije z napravami. Potem so tu še proizvodne linije, ki so razvite in sestavljene znotraj podjetja. Tudi tu je treba proizvodno linijo podpreti s sistemom sledljivosti. Če bi imeli za vsako proizvodno linijo svoj sistem sledljivosti, bi nastal tudi problem vzdrževanja, saj bi potem morali skrbeti za veliko število sistemov, ki delujejo vsak po svojem principu. V kolikor pa imamo enoten sistem po celotnem podjetju, privarčujemo na vzdrževanju in hitrem reševanju morebitnih težav, saj lahko podporne službe sistem dobro spoznajo in hitreje rešijo morebitne napake.
Zaradi predstavljenih problemov bi bilo smiselno iti v lasten razvoj sistema za sledljivost proizvodnje. Z lastnim razvojem pa nastanejo tudi novi problemi, kot so zagotovitev ter plačilo lastnega kadra, zagotovitev primernih prostorov v podjetju itd.
Največji strošek tako predstavlja začetni razvoj takega sistema. Sistem je smiselno razviti čimbolj modularno, saj ga na ta način lahko hitreje in ceneje prilagodimo potrebam posamezne proizvodne linije. Ključno je prav to, da se sistem lahko čim hitreje implementira na različne proizvodne linije, ki imajo vsaka zase zelo različne potrebe. Pri tem je treba tudi upoštevati stroške uporabljene strojne opreme ter njene obratovalne stroške. Stroške uporabljene strojne opreme ter programske opreme lahko hitro izračunamo. Pri uporabljeni programski opremi je plačljivo samo orodje KEPServerEX, kjer licenca stane približno 500 €. Poleg tega je treba kupiti tudi gonilnike za različne tipe naprav, kjer se cena tudi giblje okrog 500 €. Računalnik, ki ima vlogo lokalnega strežnika, stane približno 1000 € s pripadajočo periferijo. Za odjemalce pa se lahko uporabi Raspberry Pi 3, ki skupaj z napajanjem ter spominsko kartico stane okrog 50 €. Za kliente

nato potrebujemo še zaslon, občutljiv na dotik, ki ga lahko dobimo že za približno 250 €. Za odjemalce bi lahko uporabljali tudi navaden osebni računalnik ali pa računalniške tablice, namenjene proizvodni uporabi, vendar bi hitro prišli na desetkratnik cene predlaganega klienta. Zanesljivost uporabe Raspberry Pi-ja se bo pokazala čez čas, vendar pa je vsaj za fazo testiranja sistema zadostna rešitev.
Prihranki, ki jih lahko tak sistem čez čas prinese, so veliki. Najbolj opazni so prihranki v primeru reklamacij, saj se lahko morebitne odpoklice omeji na majhno število izdelkov. Ne govorimo pa samo o prihranku denarja, ampak tudi o prihranku časa pri planiranju proizvodnje. S spremljanjem časov cikla posameznega delovnega mesta lahko najdlje trajajoče delovno mesto optimiziramo. Vpeljava sistema za sledljivost proizvodnje je tudi strateškega pomena za pridobivanje novih kupcev.

3.2 TEHNIČNA IZVEDLJIVOST

Celoten sistem je treba umestiti v že obstoječe poslovne informacijske sisteme, saj v podjetju že obstajajo sistemi za spremljanje proizvodnje, izmeta, realizacije ipd., vendar se v večini primerov zanašajo na ročni vnos. Pri ročnem vnosu pa obstaja veliko večja verjetnost za vnos napačnih podatkov. Nadomestiti bi bilo treba ročni vnos z avtomatskim beleženjem podatkov, ki jih nato lahko prikažemo v obstoječih aplikacijah. Sistem mora biti sposoben pridobivati podatke iz poslovnega sistema SAP, saj se preko tega sistema razpiše potrebne delovne naloge, dokumentacijo, recepte za delo na izdelku, zagotovi se pravočasno ter zadostno dostavo materiala itd. Na delovnem nalogu so tudi predpisani podatki za upravljanje s proizvodno linijo, kot so uporabljena delovna mesta, zaporedje delovnih mest itd. Torej bi morali sistem sledljivosti proizvodnje obravnavati celovito od upravljanja s proizvodnjo opremo, sledenja komponent in materiala, vključenega v izdelek, ter do prikaza dokumentacije izdelka na posameznem delovnem mestu.
Predlagana sistemska arhitektura je, da imamo na proizvodni liniji lokalni strežnik, na katerem je pognan naš sistem za sledljivost proizvodnje. Na strežniku je tudi postavljena lokalna podatkovna baza ter OPC (angl. Open Platform Communications) strežnik, ki skrbi za komunikacijo s PLC krmilniki. Lokalni strežnik je povezan v poslovno omrežje ter v lokalno omrežje z vsemi napravami ter klienti na proizvodni liniji. Na vsakem delovnem mestu imamo nato klienta, ki je sposoben prikazovati spletno aplikacijo, ki je gostovana na lokalnem strežniku.
Sistem bi bil v celoti razvit v podjetju brez zunanjih izvajalcev. Izbrana lokalna podatkovna baza je Microsoft SQL Server Express. Zaradi uporabe modularnega pristopa pri razvoju sistema je smiselno sistem osnovati na uporabi spletnih tehnologij. S tem pristopom dosežemo tudi odprtost sistema, saj nismo omejeni na en operacijski sistem oziroma na en tip klienta. Edino kar se zahteva od klienta je to, da zna prikazovati spletne strani. Strežniška stran (angl. back-end) bi bila torej razvita s pomočjo javanskega ogrodja Spring Boot, odjemalna stran pa bi bila razvita s pomočjo platforme Angular. Potem je tu

še orodje KEPServerEX podjetja Kepware, ki služi kot OPC-strežnik za povezavo na PLC-krmilnike.

Določiti je treba tudi komunikacijski protokol med našim sistemom ter napravami na proizvodni liniji. Najosnovnejši način komunikacije med našim sistemom ter napravami na proizvodni liniji je preko OPC-protokola. Določiti je treba, kakšne spremenljivke oziroma tako imenovane značke (angl. tags) bomo uporabili za komunikacijo ter kaj vrednosti posamezne značke pomenijo. Nato je treba od dobavitelja naprave zahtevati, da implementira te značke na strani PLC-krmilnika. V osnovi moramo imeti tri skupine značk za komunikacijo, in sicer prejemanje zahteve s strani PLC-krmilnika, odgovor na zahtevo ter procesne podatke, ki jih sistem prebere po koncu operacije na delovnem mestu. Vsi dobavitelji naprav pa ne podpirajo komunikacije s PLC-krmilnikom. Tako moramo pokriti tudi komunikacijo preko CSV-datotek (angl. Comma Separated Values), katerih struktura je določena tudi z naše strani. Te datoteke se nato odlagajo v deljene mape. Nekateri dobavitelji naprav pa ne podpirajo niti komunikacije preko deljenih datotek, zato moramo podpreti tudi komunikacijo z napravo preko podatkovne baze, preko spletnih protokolov itn. S stališča komunikacije z dobavitelji različnih naprav na proizvodni liniji je bolj smiselno, da sistem za sledljivost proizvodnje izvedemo z lastnim razvojem. Tako lahko sami z dobavitelji naprav najbolje uskladimo zahteve, po drugi strani pa se lahko dobavitelju kar se da hitro prilagodimo v primeru, da ne uporablja nobenih standardnih komunikacijskih protokolov.

3.3 OPERATIVNA IZVEDLJIVOST

Sistem mora biti v podporo proizvodnji in posledično ne sme bistveno motiti oziroma spreminjati proizvodnega procesa. Delavci morajo na posameznem delovnem mestu preko sistema prejeti vse bistvene informacije o izdelku, ki ga trenutno izdelujejo. Vsaka večja zakasnitev s strani sistema posledično pomeni izgubo za proizvodnjo.

Za razvoj projekta je treba zagotoviti zadostno število kompetentnih ljudi, ki bodo posvečeni projektu. Projektu je tako namenjena služba proizvodne informatike, ki je že razvila ter skrbi za sistem upravljanja proizvodnje MES. Torej znanje in integracija sistema sledljivosti proizvodnje v obstoječe informacijske sisteme v podjetju ni sporna. Glede kasnejšega vzdrževanja sistema bi bilo potrebno izobraziti ljudi, ki bi bili v podporo službi proizvodne informatike ter bi bili sposobni sami odpraviti manjše napake na sistemu.

3.4 UGOTOVITVE

Glede na izvedeno študijo izvedljivosti se je podjetje odločilo za lasten razvoj sistema za sledljivost proizvodnje tako z ekonomskega kot tudi s tehničnega vidika. Podjetje ima med zaposlenimi potrebno znanje ter potrebno manjšo skupino razvijalcev, ki se bo ukvarjala z razvojem tega produkta. V primeru uspešne izpeljave projekta pa bi bilo treba zaposliti

nove ljudi, ki bi sistem razvijali naprej ter ga prilagajali in implementirali na nove proizvodne linije. Odprto vprašanje je še glede stroškov uporabljene strojne opreme, saj ne vemo, kako se po obnesel odjemalec z Raspberry Pi-jem, ki bi ga v primeru nezadostnih zmogljivosti morali nadomestiti z dražjimi alternativami.

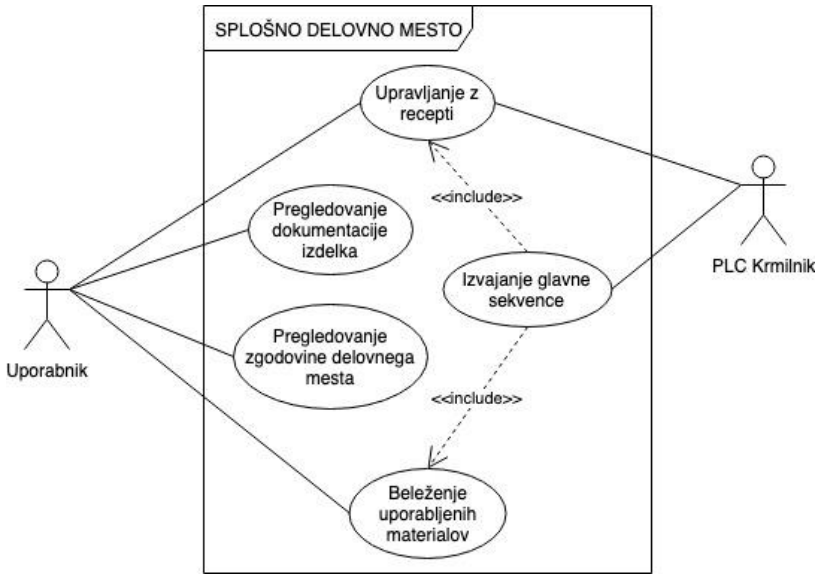
Razvoj sistema za sledljivost izdelkov na proizvodni liniji je potrjen s strani podjetja. Pričakuje se, da bo sistem razvit in testiran v roku šestih mesecev. Sistem bo uspešno opravljen, če bomo uspeli ustreči vsem zahtevam, ki so opisane v naslednjem poglavju.

4 ANALIZA IN DEFINICIJA ZAHTEV

Na osnovi informacij iz prejšnjih faz lahko v tej fazi jasno določimo zahteve sistema. Namen te faze je seznaniti naročnika oziroma uporabnika sistema z možnimi omejitvami oziroma potrebami [10]. Uporabnik mora na podlagi te fazi razumeti problem ter tudi predlagano rešitev. Ta faza je pomembna tudi za razvijalce sistema, saj preko nje dobijo vpogled v uporabnikove zahteve oziroma potrebe. Rezultat te faze je popoln opis delovanja sistema tako s stališča funkcijskih kot tudi nefunkcijskih zahtev.
V poglavjih Definicija problema ter Študija izvedljivosti zajemamo celoten projekt sledljivosti proizvodnje, torej se ne opiramo na specifike za implementacijo sistema na posamezno proizvodno linijo. V naslednjih poglavjih se bomo posvetili specifikam vpeljave sistema za sledljivost izdelkov na točno določeno proizvodno linijo, ki je podrobneje predstavljena v uvodu tega dela. Za predstavitev delovanja sistema na proizvodni liniji menimo, da se ni potrebno ukvarjati s specifikami posameznega delovnega mesta, zato bomo v nadaljevanju uporabljali izraz splošno delovno mesto.

4.1 FUNKCIJSKE ZAHTEVE

Funkcijske zahteve bomo predstavili z diagrami primera uporabe za splošno delovno mesto ter za delovno mesto pakiranja izdelkov. Vsak diagram primera uporabe je tudi podrobno opisan.



Slika 3: Diagram primera uporabe za splošno delovno mesto.

4.1.1 Primer uporabe za splošno delovno mesto

Interakcija med uporabnikom sistema na splošnem delovnem mestu, PLC-krmilnikom in sistemom je predstavljena z diagramom primera uporabe na Sliki 3. Kot vidimo na

diagramu sta na splošnem delovnem mestu prisotna dva akterja, ki upravljata z delovanjem sistema, in sicer uporabnik sistema ter PLC-krmilnik. Podroben opis primerov uporabe sledi.

Pregledovanje dokumentacije izdelka

Uporabniku mora biti omogočeno pregledovanje dokumentacije izdelka na vsakem delovnem mestu. Dokumentacija se prikazuje glede na tip izdelka, ki je trenutno prisoten na delovnem mestu. Skico uporabniškega vmesnika za pregled dokumentacije vidimo na Sliki 6.

Naziv: Pregledovanje dokumentacije izdelka

Akterji: Uporabnik

Zahteve: Dokumentacija izdelka je vezana na tip izdelka, ki ga pridobimo s prisotnostjo izdelka na delovnem mestu.

Prožilec: Uporabnik pritisne na gumb »Dokumentacija« v orodni vrstici.

Osnovni potek:

1. Uporabniku se na zaslonu prikaže seznam razpoložljivih dokumentov za izbrano delovno mesto.
2. Uporabnik s pritiskom na ime dokumenta odpre dokument, ki si ga želi ogledati.
3. Uporabnik si dokument poveča zaradi lažje berljivosti.
4. Uporabnik se s pritiskom na gumb »Dejanja« v orodni vrstici vrne na glavni zaslon aplikacije.

Stanje po zaključku: Aplikacija si zapomni zadnji odprt dokument.

Alternativni poteki:

1a: Uporabnik si ogleda dokumentacijo iz drugega delovnega mesta.

1a1: Uporabnik pritisne na izbrano delovno mesto na spodnjem delu zaslona.

1a2: Primer uporabe se zaključi ali pa se nadaljuje na točki 1 ali 2.

2a: Uporabnik ne želi pregledati nobenega dokumenta.

2a1: Uporabnik pritisne na gumb »Dejanja« v orodni vrstici ter se vrne na glavni zaslon aplikacije.

2a2: Primer uporabe se zaključi.

Pregledovanje zgodovine delovnega mesta

Uporabnik mora imeti možnost vpogleda, kateri izdelki so bili nazadnje izdelani na določenem delovnem mestu. Na uporabniškem vmesniku se prikažeta dve tabeli, v levi tabeli so podatki o zadnjih izdelkih na delovnem mestu, v desni pa so prikazani zabeleženi procesni podatki. Skico uporabniškega vmesnika za pregled zgodovine delovnega mesta vidimo na Sliki 7.

Naziv: Pregledovanje zgodovine delovnega mesta

Akterji: Uporabnik

Zahteve: Vsaj en izdelek je že bil prisoten na delovnem mestu.
Prožilec: Uporabnik pritisne na gumb »Zgodovina delovnega mesta« v orodni vrstici.
Osnovni potek:

1. Uporabniku se na zaslonu prikaže zgodovina zadnjih dvajsetih izdelkov, ki so bili prisotni na delovnem mestu.
2. S pritiskom na izdelek se uporabniku prikažejo vsi shranjeni parametri.
3. Uporabnik se s pritiskom na gumb »Dejanja« v orodni vrstici vrne na glavni zaslon aplikacije.

Stanje po zaključku: Prikazan je glavni zaslon aplikacije.
Alternativni poteki:

1a: Uporabnik ne želi pregledati zgodovine delovnega mesta.

1a1: Uporabnik pritisne na gumb »Dejanja« v orodni vrstici ter se vrne na glavni zaslon aplikacije.

1a2: Primer uporabe se zaključi.

2a: Uporabnik želi pregledati samo kateri izdelki so bili nazadnje na delovnem mestu, ne pa tudi njihovih parametrov.

2a2: Uporabnik ne pritisne na noben izdelek, tako se mu ne pokažejo shranjeni parametri.

2a1: Primer uporabe se zaključi.

Upravljanje z recepti

Recept za delovanje delovnega mesta se v lokalno podatkovno bazo prenese iz MES-sistema. Ta recept imenujemo globalni recept. Vezan je na tip izdelka, ki se trenutno izdeluje na delovnem mestu. Ob prihodu novega tipa izdelka na delovno mesto se recept iz podatkovne baze avtomatsko prenese na PLC-krmilnik. Uporabnik ima nato možnost popraviti recept preko uporabniškega vmesnika na PLC-krmilniku. Ta nato te spremembe sporoči SCADA-sistemu, ki popravljen recept shrani v podatkovno bazo. Ta recept imenujemo lokalni recept. Uporabnik nato preko uporabniškega vmesnika SCADA-sistema pregleda recepte, ki so shranjeni v lokalni podatkovni bazi in izbere recept za prenos na PLC-krmilnik. Pošiljanje receptov iz PLC-krmilnika v SCADA-sistem in obratno se lahko izvaja neodvisno od glavne sekvence. Glavna sekvenca pa se ne izvede, če PLC-krmilnik nima recepta za delo na izdelku.
Naziv: Upravljanje z recepti
Akterji: Uporabnik in PLC-krmilnik
Zahteve: Prisotnost novega tipa izdelka na delovnem mestu.
Prožilec: Nov tip izdelka prispe na delovno mesto.
Osnovni potek:

1. Recept za trenuten tip izdelka se prenese iz podatkovne baze.
2. Preneseni recept se zapiše na PLC-krmilnik.

3. Glavna sekvenca se nadaljuje.

Stanje po zaključku: PLC-krmilnik ima pravilen recept za delovanje.

Alternativni poteki:

1a: V lokalni podatkovni bazi ni recepta za trenuten tip izdelka ali pa je prišlo do napake pri prenosu recepta iz podatkovne baze.

1a1: Uporabnik preko PLC-krmilnika sproži zahtevo za shranjevanje lokalnega recepta.

1a2: Uporabnik nato preko uporabniškega vmesnika SCADA-sistema izbere pravilen lokalni recept in izbiro potrdi z gumbom »Prenesi recept«.

1a3: Recept se zapiše na PLC-krmilnik.

1a4: Primer uporabe se zaključi.

2a: Uporabnik vidi, da je v prenesenem receptu napaka.

2a1: Uporabnik preko PLC-krmilnika popravi recept in sproži zahtevo za shranjevanje lokalnega recepta.

2a2: Uporabnik nato preko uporabniškega vmesnika SCADA-sistema izbere pravilen lokalni recept in izbiro potrdi z gumbom »Prenesi recept«.

2a3: Recept se zapiše na PLC-krmilnik.

2a4: Primer uporabe se zaključi.

Beleženje uporabljenih materialov

Sistem mora za vsak izdelek zabeležiti, kateri material je bil uporabljen na delovnem mestu. Podatke o potreben materialu na delovnem mestu se razpiše na ERP-sistemu in so vezani na tip izdelka. Ti podatki se nato preko MES-sistema prenesejo v lokalno podatkovno bazo sistema SCADA. Fizično je material na delovnem mestu opremljen z logistično nalepko, ki med drugim vsebuje tudi kodo materiala.

Uporabnik ob prihodu novega tipa izdelka z optičnim bralnikom črtnih kod, ki je povezan neposredno na strežnik SCADA-sistema prebere dvodimenzionalno kodo na logistični nalepki. SCADA-sistem nato preveri, ali se prebrana koda materiala ujema s katero izmed kod materialov v lokalni podatkovni bazi za ta tip izdelka. V kolikor se kodi ujemata, se na delovnem mestu ta material aktivira. Dokler na delovnem mestu niso aktivirani vsi razpisani materiali, se glavna sekvenca ne izvede. Sistem zabeleži te materiale v lokalno podatkovno bazo in ne zahteva optičnega branja logističnih nalepk, če so aktivirani vsi materiali na delovnem mestu. Aktivirani materiali so na uporabniškem vmesniku sistema SCADA označeni z zeleno barvo, neaktivirani materiali pa so označeni z rdečo barvo.

Naziv: Beleženje uporabljenih materialov

Akterji: Uporabnik

Zahteve: Prisotnost izdelka na delovnem mestu.

Prožilec: Izdelek prispe na delovno mesto.

Osnovni potek:

1. Sistem preveri aktivne materiale na delovnem mestu.
2. Uporabnik z optičnim bralnikom prebere kodo na logistični nalepki.
3. Optično prebran material se aktivira.

Stanje po zaključku: Delovno mesto je opremljeno s pravilnim materialom.

Alternativni poteki:

1a: Na delovnem mestu so aktivirani vsi razpisani materiali.

1a1: V podatkovno bazo se zabeleži aktiviran material.

1a2: Glavna sekvenca se nadaljuje.

1a3: Primer uporabe se zaključi.

2a: Koda na logistični nalepki ni berljiva.

2a1: Služba logistike mora zahtevati novo logistično nalepko.

2a2: Ko imamo novo nalepko, nadaljujemo s korakom 2.

3a: Optično prebrana koda materiala se ne ujema z nobeno kodo materiala v lokalni podatkovni bazi.

3a1: Na delovnem mestu je prisoten napačen material.

3a2: Služba logistike mora dostaviti pravilen material.

3a3: Ko imamo nov material, nadaljujemo s korakom 3.

Izvajanje glavne sekvence

Glavna sekvenca sistema ne potrebuje nobene interakcije s strani uporabnika, saj komunicira samo s PLC-krmilnikom.

Naziv: Izvajanje glavne sekvence

Akterji: PLC-krmilnik

Zahteve: Prisotnost izdelka na delovnem mestu.

Prožilec: PLC-krmilnik sporoči, da je izdelek prispel na delovno mesto.

Osnovni potek:

1. Aplikacija s poizvedbo v podatkovni bazi na podlagi prejete serijske številke izdelka preveri, če je izdelek namenjen za to delovno mesto.
2. Aplikacija v podatkovno bazo zabeleži, da je izdelek prisoten na delovnem mestu.
3. Aplikacija preveri, ali je na delovnem mestu prisoten pravilen material za delo na trenutnem tipu izdelka.
4. Aplikacija preveri, če se tip trenutnega izdelka razlikuje od tipa prejšnjega izdelka in prenese recept iz podatkovne baze ter ga zapiše na PLC-krmilnik.
5. Po končani operaciji na delovnem mestu PLC-krmilnik sporoči, da je operacija zaključena ter da so procesni parametri pripravljeni.
6. Aplikacija shrani vse procesne podatke v podatkovno bazo ter nato sporoči PLC-krmilniku, da je shranjevanje zaključeno.

7. Aplikacija posodobi zapis v podatkovni bazi s kvaliteto izdelka ter z naslednjim delovnim mestom.

Stanje po zaključku: Procesni parametri o izdelku so shranjeni v podatkovno bazo in aplikacija je pripravljena na nov izdelek.

Alternativni poteki:

1a: Prejeta serijska številka je napačna ali pa prazna.
- 1a1: Aplikacija se ponastavi in je pripravljena na nov izdelek.
- 1a2: Primer uporabe se zaključi.

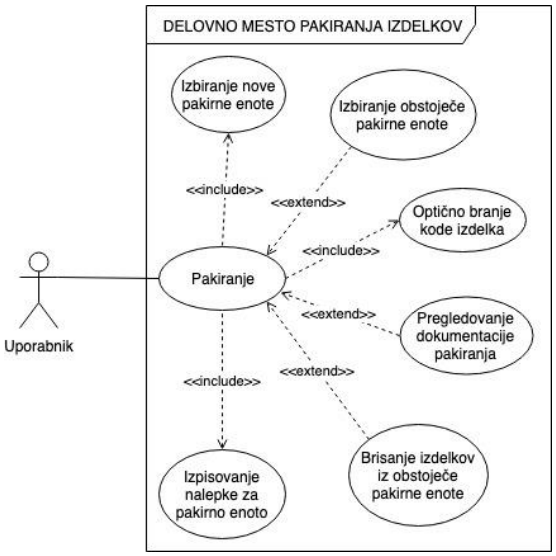
1b: Operacija na tem izdelku ni dovoljena.
- 1b1: Aplikacija se ponastavi in je pripravljena na nov izdelek.
- 1b2: Primer uporabe se zaključi.

5b: Med operacijo na izdelku do prekinitve.
- 5b1: Uporabnik lahko ponovi operacijo na trenutnem izdelku.
- 5b2: Aplikacija se ponastavi.
- 5b3: Primer uporabe se zaključi.

6a: Pri shranjevanju procesnih parametrov je prišlo do napake.
- 6a2: Izdelek se označi kot slab.
- 6a1: Aplikacija nadaljuje na korak 7.

4.1.2 Primer uporabe za delovno mesto pakiranja izdelkov

Interakcija med uporabnikom sistema na delovnem mestu pakiranja izdelkov je predstavljena z diagramom primera uporabe na Sliki 4. Kot vidimo je na delovnem mestu pakiranja prisoten samo en akter, in sicer uporabnik sistema. Podroben opis primera uporabe sledi.



Slika 4: Diagram primera uporabe za delovno mesto pakiranja izdelkov.

Izbiranje nove pakirne enote

Uporabnik mora pred pakiranjem izdelkov ustvariti novo pakirno enoto ali pa izbrati obstoječo pakirno enoto.

Naziv: Izbiranje nove pakirne enote

Akterji: Uporabnik

Zahteve: Razpisani delovni nalogi za določeno proizvodno linijo.

Prožilec: Uporabnik na uporabniškem vmesniku pritisne na gumb »Nova pakirna enota«.

Osnovni potek:

1. Uporabniku se prikaže seznam delovnih nalogov, ki so trenutno na voljo.
2. S pritiskom na izbrani delovni nalog uporabnik potrdi izbiro.
3. Ustvari se nova pakirna enota.

Stanje po zaključku: Aplikacija je pripravljena na pakiranje izdelkov.

Alternativni poteki:

1a: V kolikor ni razpisanega nobenega delovnega naloga, uporabnik ne more odpreti nove pakirne enote.

1a1: Primer uporabe se zaključi.

Izbiranje obstoječe pakirne enote

V kolikor je bila na delovnem mestu pakiranja že ustvarjena kakšna pakirna enota. jo lahko uporabnik ponovno izbere.

Naziv: Izbiranje obstoječe pakirne enote

Akterji: Uporabnik

Zahteve: V podatkovni bazi imamo podatek o vsaj eni ustvarjeni pakirni enoti.

Prožilec: Uporabnik na uporabniškem vmesniku pritisne na gumb »Odpri obstoječo pakirno enoto«.

Osnovni potek:

1. Uporabniku se prikaže seznam obstoječih pakirnih enot.
2. S pritiskom na izbrano pakirno enoto uporabnik potrdi izbiro.
3. Odpre se obstoječa pakirna enota.

Stanje po zaključku: Aplikacija je pripravljena na pakiranje izdelkov.

Alternativni poteki:

2a: Uporabnik ne želi odpreti obstoječe pakirne enote.

2a1: Primer uporabe se zaključi.

Optično branje kode izdelka

Uporabnik mora z optičnim bralnikom prebrati dvodimenzionalno kodo izdelka, ki ga želi zapakirati.

Naziv: Optično branje kode izdelka

Akterji: Uporabnik

Zahteve: Izbrana pravilna pakirna enota ter izdelek na delovnem mestu pakiranja.
Prožilec: Optično prebrana dvodimenzionalna koda izdelka.
Osnovni potek:

1. Optično prebrana koda izdelka se vpiše v pakirno enoto.
2. Izdelek se prikaže na vizualizaciji, ki je v pomoč uporabniku.

Stanje po zaključku: Aplikacija je pripravljena na optično branje kode novega izdelka.
Alternativni poteki:

1a: Optično prebrana koda izdelka ni pravilna za izbrano pakirno enoto.
1a1: Primer uporabe se zaključi.
1b: Optično prebrana vsebina ni prepoznana s strani sistema.
1b1: Primer uporabe se zaključi.

Pregledovanje dokumentacije pakiranja

Uporabniku mora biti omogočeno pregledovanje dokumentacije pakiranja. Dokumentacija pakiranja je vezana na tip izdelka, za katerega je trenutno odprta pakirna enota.
Naziv: Pregledovanje dokumentacije pakiranja
Akterji: Uporabnik
Zahteve: Odprta pakirna enota, preko katere aplikacija pridobi tip izdelka.
Prožilec: Uporabnik pritisne na gumb »Dokumentacija« v orodni vrstici.
Osnovni potek:

1. Uporabnik pregleda dokumentacijo za trenutni izdelek na delovnem mestu.
2. Uporabnik s pritiskom na ime dokumenta odpre dokument, ki si ga želi ogledati.
3. Uporabnik si lahko dokument poveča zaradi lažje berljivosti.
4. Uporabnik se s pritiskom na gumb »Dejanja« v orodni vrstici vrne na glavni zaslon aplikacije za pakiranje izdelkov.

Stanje po zaključku: Aplikacija si zapomni zadnji odprt dokument.
Alternativni poteki:

1a: Uporabnik si ogleda dokumentacijo z drugega delovnega mesta.
1a1: Uporabnik pritisne na izbrano delovno mesto na spodnjem delu zaslona.
1a2: Primer uporabe se zaključi.
2a: Uporabnik ne želi pregledati nobenega dokumenta.
2a1: Uporabnik pritisne na gumb »Dejanja« v orodni vrstici ter se vrne na glavni zaslon aplikacije za pakiranje izdelkov.
2a2: Primer uporabe se zaključi.

Brisanje izdelkov iz obstoječe pakirne enote

Uporabnik mora imeti možnost izbrisa izdelkov iz obstoječe pakirne enote, saj se lahko zgodi, da uporabnik morebitno vizualno napako na izdelku zazna po tem, ko je bil izdelek že zabeležen v pakirni enoti.

Naziv: Brisanje izdelkov iz obstoječe pakirne enote
Akterji: Uporabnik
Zahteve: Izdelek je že zapakiran v eni izmed pakirnih enot.
Prožilec: Uporabnik pritisne na gumb »Brisanje izdelkov«.
Osnovni potek:

1. Odpre se pogled za brisanje izdelkov.
2. Uporabnik z optičnim bralnikom prebere dvodimenzionalno kodo izdelka.
3. Uporabnik potrdi brisanje izdelkov z gumbom »Potrdi izbris izdelkov«.

Stanje po zaključku: Izdelek je izbrisan iz pakirne enote.
Alternativni poteki:

2a: Uporabnik ne želi izbrisati nobenega izdelka.
2a1: Uporabnik pritisne na gumb »Nazaj«.
2a2: Primer uporabe se zaključi.
2b: Aplikacija ne dobi zapisa v podatkovni bazi za optično prebrano kodo izdelka, kar pomeni, da izdelek še ni bil zapakiran v nobeno pakirno enoto.
2b1: Primer uporabe se zaključi.
2c: Uporabnik želi izbrisati več izdelkov naenkrat.
2c1: Uporabnik zaporedno z optičnim bralnikom prebere več izdelkov.
2c2: Aplikacija nadaljuje izvajanje na koraku 3.

Izpisovanje nalepke pakirne enote

Pakirna enota mora biti označena z nalepko, ki vsebuje vse osnovne informacije o pakirni enoti, kot na primer številko pakirne enote, število zapakiranih izdelkov ter tip zapakiranih izdelkov.

Naziv: Izpisovanje nalepke pakirne enote
Akterji: Uporabnik
Zahteve: Ustvarjena vsaj ena pakirna enota z zapakiranimi izdelki.
Prožilec: Uporabnik pritisne na gumb »Natisni nalepko« v primeru predčasne zaključitve pakirne enote ali pa samodejno, ko je pakirna enota polna.
Osnovni potek:

1. Aplikacija pošlje zahtevo za tiskanje nalepke na tiskalniški strežnik.

Stanje po zaključku: Natisnjena nalepka za na pakirno enoto.
Alternativni poteki:

1a: V kolikor pride do napake pri dostopu do tiskalniškega strežnika ali pa se nalepka ne natisne, je potrebna intervencija skrbnika sistema.
1a1: Primer uporabe se zaključi.

Pakiranje

Za pakiranje izdelkov je potrebna interakcija uporabnika s sistemom. Uporabnik je odgovoren za izbiranje pravilne pakirne enote, sistem pa nato avtomatsko preveri, ali je izdelek namenjen za izbrano pakirno enoto ali ne.

Naziv: Pakiranje

Akterji: Uporabnik

Zahteve: Izdelek je prisoten na delovnem mestu pakiranja.

Prožilec: Uporabnik začne z izbiro pakirne enote.

Osnovni potek:

1. Uporabnik izbere novo ali pa že obstoječo pakirno enoto.
2. Uporabnik optično prebere kodo izdelka, ki ga želi zapakirati v izbrano pakirno enoto.
3. Izdelek je uspešno zabeležen v pakirni enoti.
4. Sistem sproži zahtevo za tiskanje nalepke pakirne enote, ko je pakirna enota polna.

Stanje po zaključku: Pakirna enota je zaključena in nalepka pakirne enote natisnjena.

Alternativni poteki:

1a: Uporabnik ne izbere nobene pakirne enote.

1a1: Primer uporabe se zaključi.

2a: Uporabnik z optičnim bralnikom prebere kodo izdelka, ki ni namenjen za to pakirno enoto.

2a1: Izpiše se opozorilo in izdelek se ne zabeleži v pakirno enoto.

2a2: Primer uporabe se zaključi.

2b: Uporabnik želi izbrisati izdelek iz pakirne enote.

2b1: Uporabnik odpre pogled za brisanje izdelkov.

2b2: Uporabnik z optičnim bralnikom prebere kodo izdelka.

2b3: Primer uporabe se zaključi.

4b: Uporabnik želi pakirno enoto predčasno zaključiti in natisniti nalepko.

4b1: Uporabnik sproži zahtevo za tiskanje nalepke.

4b2: Primer uporabe se zaključi.

4.2 NEFUNKCIJSKE ZAHTEVE

Pri načrtovanju sistema ni dovolj, da se posvetimo samo funkcijskim zahtevam ter pravilnemu delovanju sistema, temveč je zelo pomembno, da se določi tudi nefunkcijske zahteve. Ker je sistem del proizvodnje, je zelo pomembno, da deluje stabilno ter zanesljivo. Prav tako je pomembno, da je sistem odziven ter enostavno razširljiv. Opisi posameznih nefunkcijskih zahtev sledijo v naslednjih podpoglavjih.

4.2.1 Odzivnost

Sistem je umeščen v proizvodnjo linijo, ki ima nizek čas cikla, torej komunikacija med sistemom in PLC-krmilnikom ne sme bistveno podaljševati časa cikla. Pričakuje se, da se bo cikel zaradi te komunikacije nekoliko podaljšal, vendar pa ta vrednost ne sme presegati sekunde na delovno mesto. Kar se našega sistema tiče, je to čas od takrat, ko nam PLC-krmilnik poda zahtevo, do takrat, ko sistem krmilniku odgovori na podano zahtevo. Bistveno je, da komunikacijo med PLC-krmilnikom ter našim sistemom zmanjšamo, kolikor je to možno. Preveriti bo treba, kako število delovnih mest vpliva na delovanje sistema oziroma če se zaradi dodajanja novih delovnih mest odzivni čas sistema bistveno poveča. Pomemben bo tudi odzivni čas podatkovne baze, ko bo ta vsebovala veliko podatkov. Potrebno bo narediti analizo najzahtevnejših poizvedb in na podlagi teh rezultatov po potrebi optimizirati podatkovno bazo oziroma same poizvedbe.

4.2.2 Stabilnost

Zagotoviti moramo tako stabilnost programske kot tudi strojne opreme, saj lahko napake v sistemu privedejo do velikih izgub za proizvodnjo. Stabilnost programske opreme lahko do neke mere preverimo z intenzivnim testiranjem pred rednim delovanjem proizvodnje linije. Pričakujemo, da bomo morebitne preostale težave, ki bi lahko vplivale na stabilnost, zaznali med poizkusnim zagonom proizvodne linije, saj bo takrat sistem najbolj realno obremenjen. Stabilnost strojne opreme je težje preveriti, saj to zahteva svoj čas. Je pa pomembno, da imamo pripravljeno rešitev v primeru odpovedi ključne strojne opreme.

4.2.3 Zanesljivost

Podatki, ki se jih beleži na posameznem delovnem mestu, morajo biti zanesljivo shranjeni v bazi podatkov. Podatki o posameznem izdelku so vezani na serijsko številko izdelka, torej moramo zagotoviti, da so pravi podatki vezani na pravo serijsko številko izdelka. Ti podatki morajo biti nato v realnem času varnostno kopirani na drugi strežnik. Tako imamo, tudi v primeru izpada sistema, na proizvodni liniji vedno podatke shranjene še na drugem strežniku. Zagotoviti je treba redno varnostno kopiranje celotnega sistem za primer odpovedi katere od strojnih komponent.

4.2.4 Razširljivost

V primeru, da želimo na proizvodnjo linijo dodati nova delovna mesta, mora sistem omogočati hitro implementacijo teh novih delovnih mest brez velikega vpliva na odzivnost

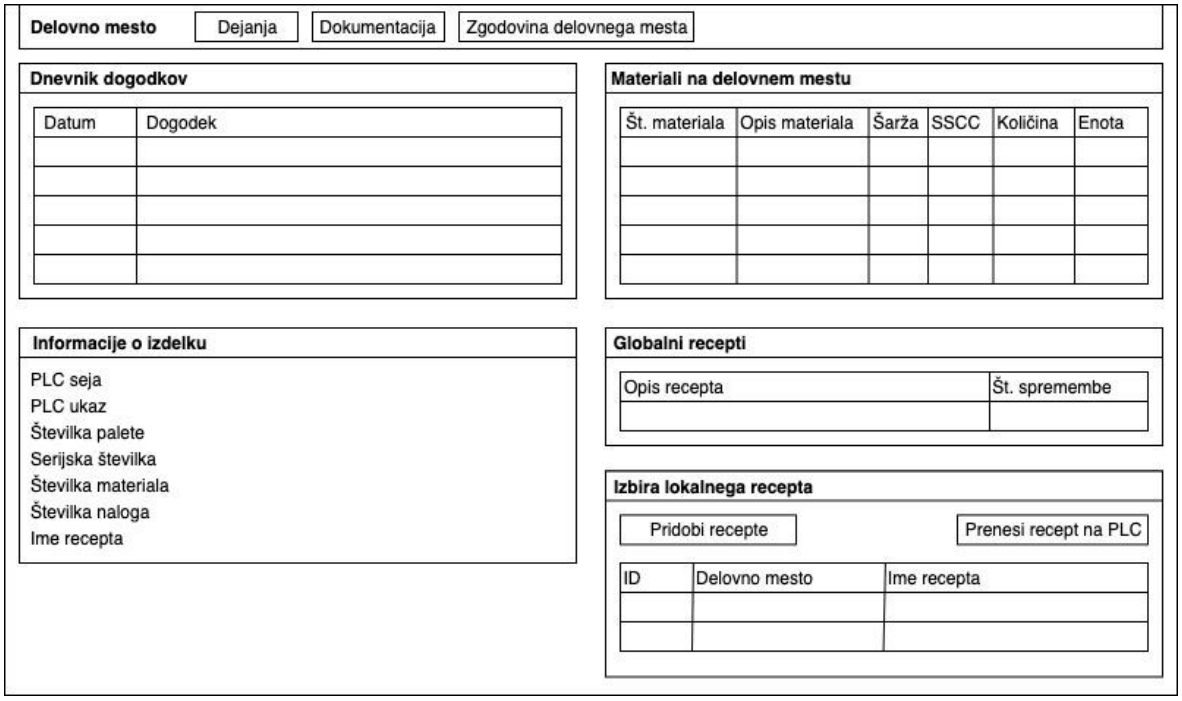
sistema. Če pride do želje po beleženju novih procesnih parametrov iz delovnega mesta, mora sistem omogočiti lahko dodajanje novih podatkov v podatkovno bazo.

4.3 UPORABNIŠKI VMESNIK

Predvidena uporabniška vmesnika našega sistema na splošnem delovnem mestu ter na delovnem mestu pakiranja izdelkov sta predstavljena v naslednjih podpoglavjih. Na splošnem delovnem mestu posebej predstavimo še pogled za pregled dokumentacije izdelka in pogled za pregled zgodovine izdelka. Uporabniški vmesnik mora biti pregleden in intuitiven za uporabo.

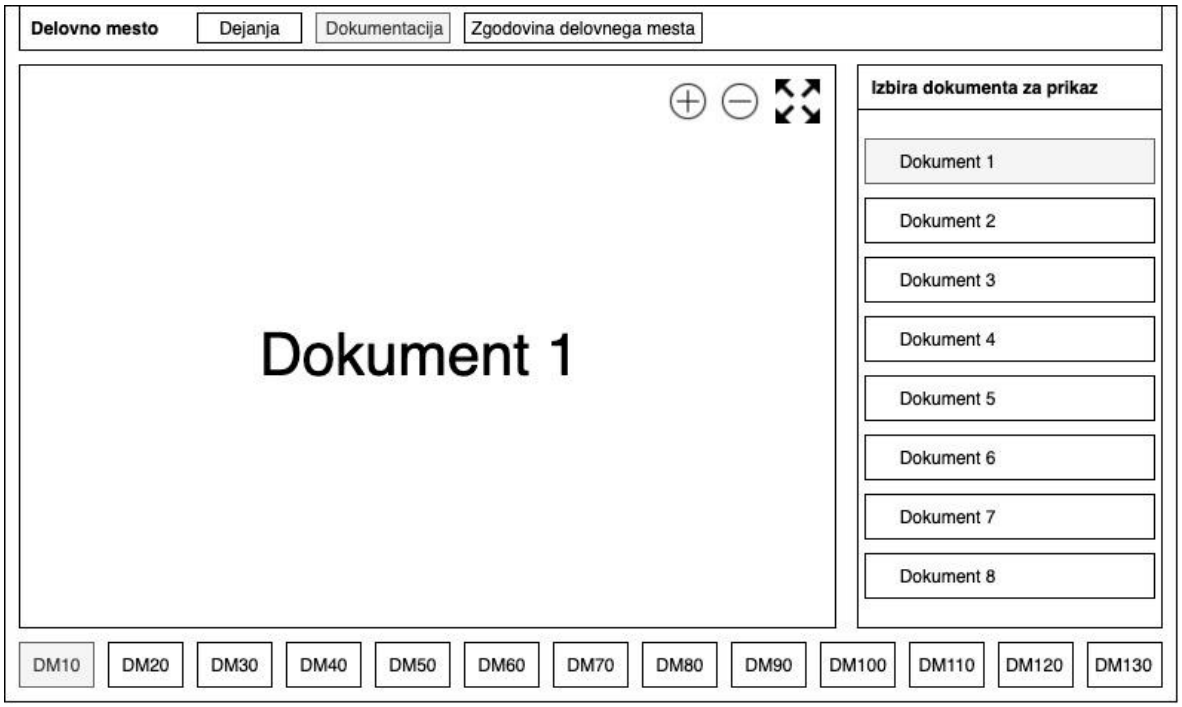
4.3.1 Splošno delovno mesto

Na vrhu zaslona je orodna vrstica z gumbi, ki uporabnika usmerjajo do različnih pogledov aplikacije. Aplikacija ima tri glavne poglede, in sicer »Dejanja«, »Dokumentacija« in »Zgodovina delovnega mesta«. Privzet pogled je pogled »Dejanja«, ki vsebuje informacije o glavni sekvenci programa. Levo zgoraj je dnevnik zadnjih dogodkov, ki jih aplikacija sporoča. Ta dnevnik služi za pridobivanje hitrih informacij o stanju aplikacije in o stanju trenutnega izdelka. Pod dnevnikom dogodkov se nahajajo informacije o stanju na PLC-krmilniku, ki so sočasno tudi informacije o izdelku, ki se trenutno izdeluje na tem delovnem mestu. Desno zgoraj je tabela, ki vsebuje informacije o materialu, ki je uporabljen oziroma zahtevan na trenutnem delovnem mestu. Aktivni material je v tabeli obarvan z zeleno barvo, neaktivni pa z rdečo. Pod to tabelo so prikazane informacije o receptih za trenutno delovno mesto. V nadaljevanju zasledimo informacijo o obstoju globalnega recepta in nato še informacije o receptih, ki so bili pridobljeni lokalno iz PLC-krmilnika. V primeru, da globalni recept ne obstaja, mora uporabnik naložiti lokalni recept, zato je za uporabnika informacija o globalnem receptu potrebna. Nato sta tu prisotna dva gumba, in sicer »Pridobi recepte«, ki preko poizvedbe v lokalni podatkovni bazi pridobi shranjene recepte, ter gumb »Prenesi recept na PLC«, ki prenese izbrani recept na PLC-krmilnik. Del za izbiranje receptov je tudi edini del, ki od uporabnika zahteva interakcijo z aplikacijo. Vse ostalo se zgodi avtomatsko s pomočjo komunikacije s PLC-krmilnikom ter z lokalno podatkovno bazo. Skico uporabniškega vmesnika vidimo na Sliki 5. Na skici je predstavljen glavni zaslon aplikacije, ki uporabniku nudi vse najpomembnejše informacije o delovnem mestu.



Slika 5: Skica uporabniškega vmesnika za splošno delovno mesto.

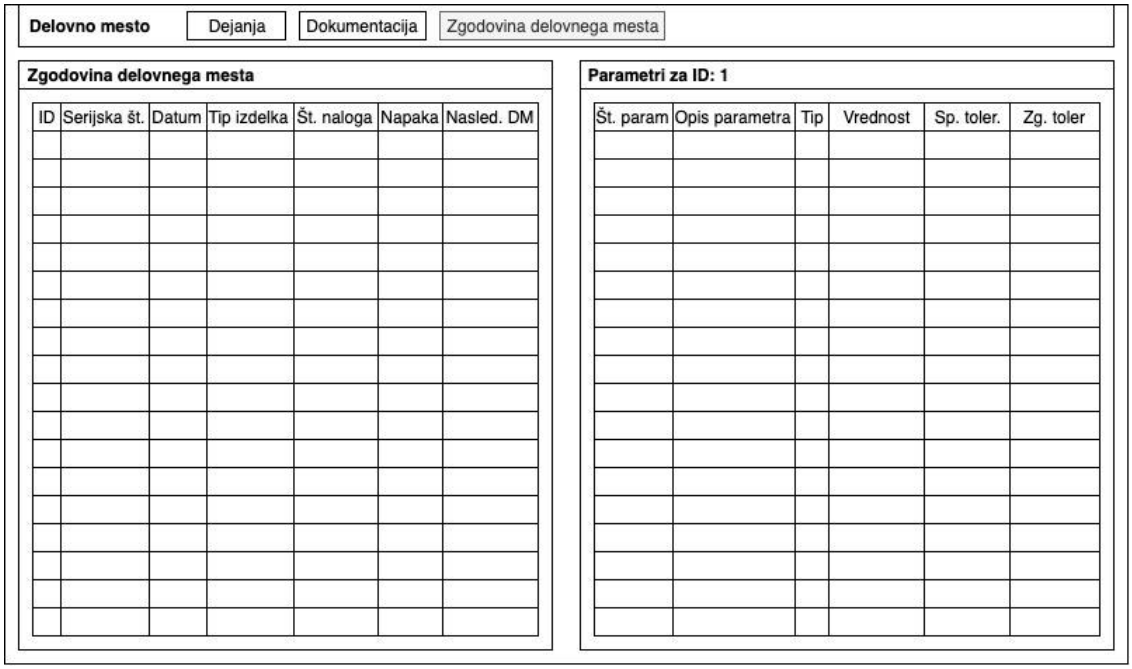
Pogled za pregled dokumentacije je razdeljen. Na levi strani je predogled izbranega dokumenta. V oknu, kjer se prikaže dokumentacija, ima uporabnik možnost prikazan dokument povečati, pomanjšati ali pa odpreti dokument čez celoten zaslon.



Slika 6: Skica uporabniškega vmesnika za pregled dokumentacije.

Na desni strani je možnost izbire dokumenta za pregled. Privzeto se odpre dokumentacija za trenutno delovno mesto, obstaja pa tudi možnost pregleda dokumentacije za vsa ostala

delovna mesta na proizvodni liniji. Prikazana dokumentacija je vedno vezana na tip izdelka, ki se trenutno izdeluje na delovnem mestu. Skico uporabniškega vmesnika za pregled dokumentacije vidimo na Sliki 6.
Pogled za zgodovino delovnega mesta ja razdeljen na dva dela. Na levi strani je tabela, ki vsebuje informacije o zadnjih dvajsetih izdelkih, ki so bili izdelani na tem delovnem mestu. S pritiskom na vrstico izbranega izdelka se na desni strani odpre tabela z vsemi procesnimi parametri, ki jih je sistem zabeležil za izbrani izdelek na trenutnem delovnem mestu. Skico uporabniškega vmesnika za pregled zgodovine delovnega mesta vidimo na Sliki 7.



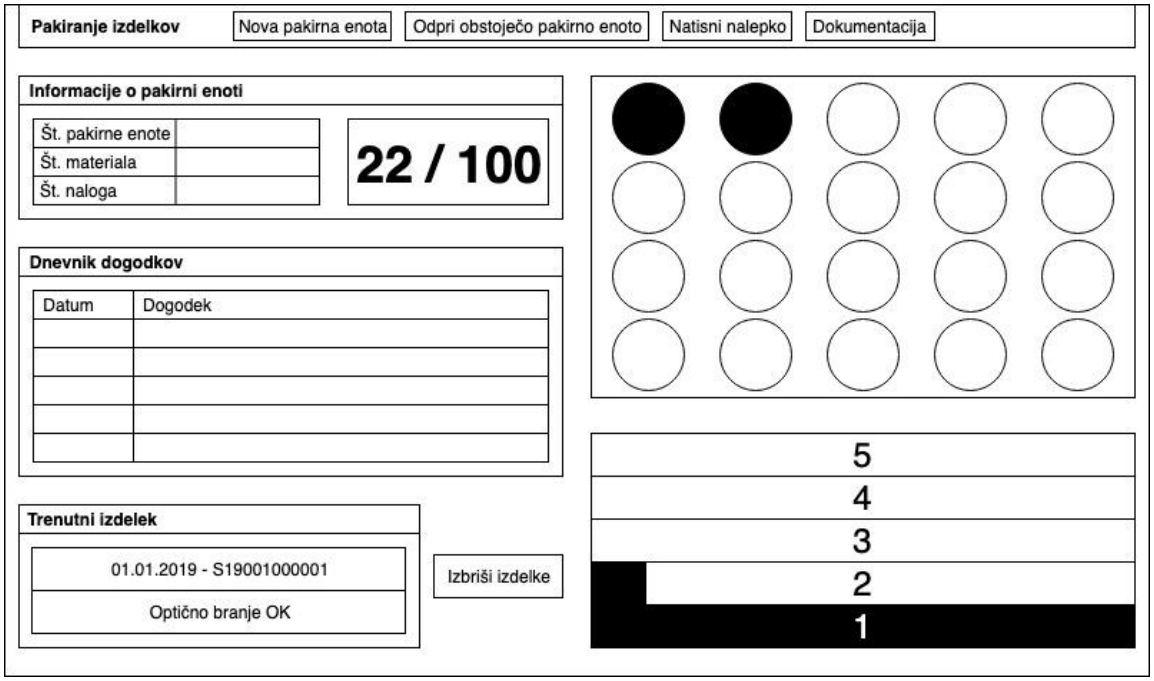
Slika 7: Skica uporabniškega vmesnika za pregled zgodovine delovnega mesta.

Cilj uporabniškega vmesnika je, da ne otežuje dela uporabniku ter da mu nudi vse osnovne informacije o izdelku. Zaradi lažjega pregleda mora biti uporabljena večja in lahko berljiva pisava. Razlika med barvo ozadja in barvo pisave mora biti kontrastna zaradi lažje berljivosti. Gumbi za interakcijo z uporabniškim vmesnikom morajo biti primerno veliki, saj se aplikacija odjemalca izvaja na napravi z zaslonom na dotik. Uporabniški vmesnik od uporabnika ne zahteva interakcije za osnovno delovanje aplikacije. Interakcija je zahtevana samo za lokalno upravljanje z recepti, pregledovanje dokumentacije ter pregledovanje zgodovine delovnega mesta.

4.3.2 Delovno mesto pakiranja izdelkov

Na vrhu zaslona je orodna vrstica z gumbi, ki uporabnika usmerjajo do različnih pogledov aplikacije. V tej orodni vrstici je tudi gumb »Natisni nalepko«, ki sproži zahtevo za tiskanje nalepke z vsemi informacijami o trenutno odprti pakirni enoti.

S pritiskom na gumb »Nova pakirna enota« v orodni vrstici se odpre pogled za izbiro delovnega naloga, s katerim bomo ustvarili pakirno enoto. Prikazani so delovni nalogi, ki so trenutno na voljo na proizvodni liniji. V primeru, da uporabnik delovnega naloga ne vidi, lahko uporabi iskalnik na desni strani zaslona. S klikom na vnosno polje se odpre številčnica za vpis številke delovnega naloga. Če delovnega naloga še vedno ni na seznamu, lahko uporabnik s tipko »Prenesi delovni nalog« sproži zahtevo za prenos delovnega naloga iz globalne baze v lokalno. S pritiskom na izbrani delovni nalog uporabnik potrdi izbiro in odpre se privzeti pogled aplikacije z ustvarjeno novo pakirno enoto.
S pritiskom na gumb »Odpri obstoječo pakirno enoto« v orodni vrstici se odpre tabela z vsemi pakirnimi enotami, ki so bile narejene na proizvodni liniji. S pritiskom na izbrano pakirno enoto se odpre privzeti pogled aplikacije z izbrano obstoječo pakirno enoto.
Privzeti pogled aplikacije je pogled na stanje trenutno odprte pakirne enote. Če nimamo odprte nobene pakirne enote, je vsebina ekrana prazna. V nasprotnem primeru je levo zgoraj tabela z osnovnimi informacijami o trenutno odprti pakirni enoti. V tej tabeli je tudi prikazana količina trenutno zapakiranih izdelkov in količina izdelkov, ki jih lahko zapakiramo v izbrano pakirno enoto. Pod to tabelo je dnevnik zadnjih dogodkov, ki jih je aplikacija sporočila. Dnevnik služi za pridobivanje hitrih informaciji o trenutnem stanju aplikacije. Pod dnevnikom dogodkov se odpre polje za vsako optično prebrano kodo izdelka. Polje vsebuje osnovne informacije o izdelku. Če optično prebrani izdelek ni namenjen za to pakirno enoto, se ozadje tega polja obarva rdeče, v nasproten primeru se ozadje obarva zeleno. Na ta način uporabnik hitro dobi informacijo o tem, ali je izdelek namenjen za to pakirno enoto ali ne. Pod tem poljem se nahaja gumb, ki uporabnika vodi do pogleda za brisanje izdelkov. Ko je aplikacija v tem pogledu, lahko uporabnik optično prebere kode izdelkov, ki jih želi izbrisati. Uporabnik lahko optično prebere več izdelkov, ti izdelki se nato izpišejo v seznamu za brisanje. Uporabnik nato potrdi brisanje izdelkov s pritiskom na gumb »Potrdi izbris izdelkov«. Če si uporabnik premisli, lahko prekine brisanje izdelkov s pritiskom na gumb »Nazaj«. Celotna desna stran zaslona je rezervirana za vizualizacijo trenutne pakirne enote. Zgoraj je vizualizacija dolžine in širine trenutne pakirne enote. Vsak posamezen izdelek je predstavljen s krogom, ki se obarva, ko je koda izdelka optično prebrana ter uspešno preverjena. Če uporabnik optično prebere napačno kodo izdelka, se krog ne obarva. Pod tem pogledom je pogled o višini oziroma nivojih trenutne pakirne enote. Ko je nivo poln, se obarva.
S pritiskom na gumb »Dokumentacija« se odpre dokumentacija za delovno mesto pakiranja. Ta pogled je enak kot na vseh ostalih delovnih mestih.



Slika 8: Skica uporabniškega vmesnika za delovno mesto pakiranja izdelkov.

Uporabniški vmesnik na delovnem mestu pakiranja mora biti v pomoč uporabniku. Uporabnik mora hitro razbrati, ali je optično prebrana koda izdelka namenjena za to pakirno enoto ali pa mora uporabnik izbrati drugo pakirno enoto za ta izdelek. Pomembno vlogo ima tudi vizualizacija pakiranja, saj na ta način uporabnik hitro opazi, ali je optično prebral vse kode izdelkov, ki jih trenutno ima fizično v pakirni enoti. Skico uporabniškega vmesnika na delovnem mestu pakiranja vidimo na Sliki 8. Kot vidimo, vizualizacija pakiranja izdelkov vzame velik del uporabniškega vmesnika, saj je v veliko pomoč uporabnikom. Prav tako mora biti dobro vidno število že zapakiranih izdelkov.

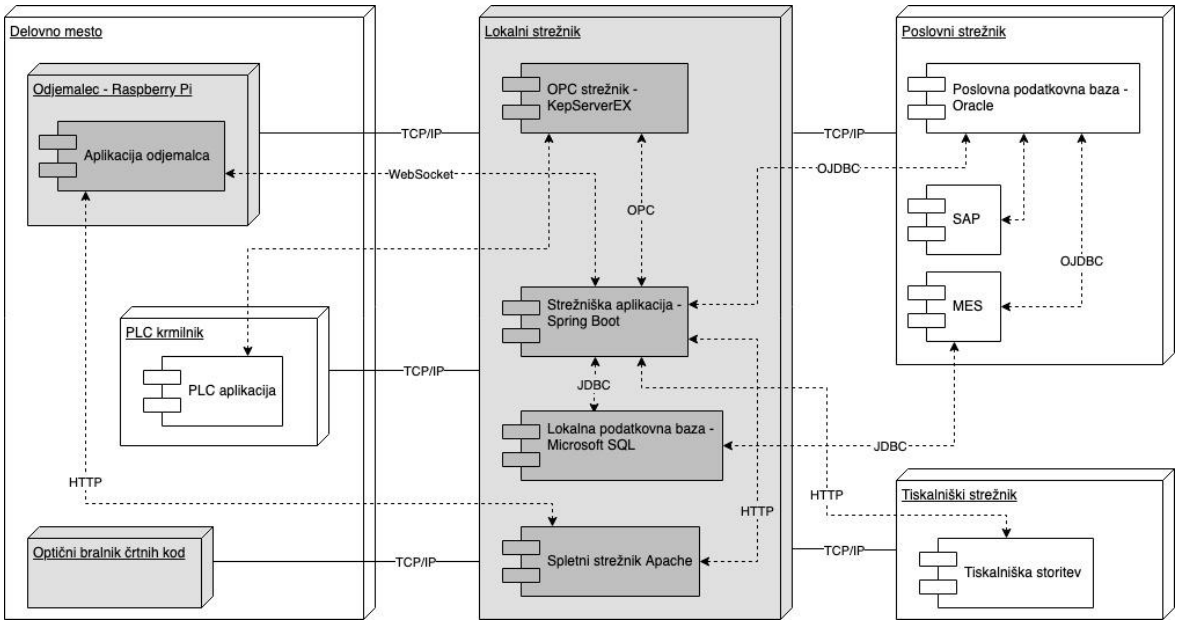
5 NAČRTOVANJE SISTEMA IN KOMPONENT

Po tem ko smo definirali in analizirali zahteve sistema, sledi faza načrtovanja sistema in njegovih komponent. Delitev sistema na komponente je ključna, saj je razvoj več enostavnejših komponent lažji od razvoja sistema kot celote. Z delitvijo tudi lažje razumemo delovanje sistema ter si olajšamo vzdrževanje sistema. S fazo načrtovanja sistema pretvorimo specifikacijo zahtev v načrtovalski model sistema [10].
Naš sistem za sledljivost izdelkov v osnovi deluje po principu odjemalec strežnik. Torej bi v grobem naš sistem lahko razdelili na dve komponenti, in sicer na odjemalni del in na strežniški del. Strežniški del pa je razdeljen na več pomembnejših komponent, ki jih bomo predstavili posebej v poglavju 5.3.

5.1 NAČRTOVANJE SISTEMA

Strežniški del našega sistema se bo izvajal na lokalnem strežniku na proizvodni linij. Vhodni podatki iz poslovnih informacijskih sistemov bodo v lokalni strežnik vstopali preko poslovnega strežnika. Izven proizvodne linije se v poslovnem omrežju nahaja še tiskalniški strežnik, na katerega se bo pošiljalo zahteve za tiskanje nalepk na proizvodni liniji. Nato so tu še odjemalec, PLC-krmilnik in optični bralnik črtnih kod, ki bodo vsi fizično del delovnega mesta na proizvodni liniji. Vsi predstavljeni elementi strojne opreme bodo med seboj povezani s TCP/IP-povezavo. Lokalni strežnik mora zaradi varnosti imeti fizično ločeno povezavo med poslovnim omrežjem in omrežjem, ki se nahaja lokalno na proizvodni liniji. Podrobnosti uporabljene strojne opreme so opisane v nadaljevanju.
Elementi in komponente sistema, ki smo jih v tem magistrskem delu načrtovali, so na Sliki 9 označeni s sivo barvo. Kot vidimo na sliki, smo se ukvarjali z elementi in komponentami, ki so del proizvodne linije, saj se aplikacije in storitve na poslovni strani v podjetju uporabljajo tudi na ostalih proizvodnih linijah. Na delovnem mestu nismo načrtovali aplikacije PLC-krmilnika, saj je za to zadolžena druga služba v podjetju. Je pa vloga PLC-krmilnika v sklopu celotnega sistema za sledljivost izdelkov zelo pomembna, saj preko njega pridobimo vse zbrane procesne podatke na delovnem mestu. Optični bralnik črtnih kod se bo uporabljal za optično branje dvodimenzionalnih kod izdelka na delovnem mestu pakiranja izdelkov in za optično branje kod na logistični nalepki za materialno sledljivost. Njegova vloga je pošiljanje prebrane vsebine črtne kode strežniškemu delu našega sistema. Na delovnem mestu tako ostane še odjemalec našega sistema. Aplikacija odjemalca se bo izvajala v spletnem brskalniku na napravi Raspberry Pi, ki poganja operacijski sistem Linux. Vloga odjemalca bo zbiranje procesnih podatkov o izdelku, pošiljanje dovoljenja za delo na izdelku, zagotavljanje materialne sledljivosti, upravljanje z recepti, pregledovanje dokumentacije in na delovnem mestu pakiranja še zbiranje podatkov o pakiranju izdelkov. Odjemalec bo imel v celotnem sistemu veliko vlogo, saj bo posredno skrbel za delovanje

proizvodne linije. V kolikor bo na odjemalcu prišlo do napake, se bo proizvodna linija ustavila. Aplikacije odjemalcev na posameznih delovnih mestih se bodo po sekvenci delovanja malenkostno razlikovale, vendar pa bodo vse pokrivale prej naštete funkcionalnosti. Strežniška stran našega sistema bo sestavljena iz štirih komponent, in sicer OPC-strežnika, lokalne podatkovne baze, spletnega strežnika in glavne strežniške aplikacije. Na lokalnem strežniku bomo uporabljali operacijski sistem Windows 10. Vloga lokalnega strežnika bo zagotavljanje povezave s PLC-krmilniki, shranjevanje podatkov v podatkovno bazo, gostovanje aplikacije odjemalca, povezava na optične bralnike črtnih kod in povezava s poslovnim omrežjem, saj naprave iz lokalnega omrežja ne morejo neposredno dostopati do poslovnega omrežja.



Slika 9: Diagram postavitve sistema za sledljivost izdelkov.

5.2 KOMUNIKACIJSKI PROTOKOLI

Preden bomo naš sistem razdelili na komponente, bomo predstavili komunikacijske protokole, ki te komponente povezujejo. Predstavili bomo dva manj poznana komunikacijska protokola, in sicer OPC ter WebSocket. Preostala protokola HTTP in JDBC sta v podobnih aplikacijah v uporabi že vrsto let in sta že dobro poznana.

5.2.1 OPC

OPC je standard za varno in zanesljivo izmenjavo podatkov med industrijskimi napravami ter ostalimi aplikacijami. Namen OPC-protokola je ustvariti skupen vmesnik, ki deluje med različnimi ponudniki industrijskih krmilnikov ter ostalo programsko opremo [19]. Za razvoj in vzdrževanje skrbi OPC Foundation, ki je 1996 izdala prvo specifikacijo

protokola, ki jo danes poznamo kot OPC DA (angl. OPC Data Access) [19]. Ta specifikacija temelji na Microsoftovi tehnologiji DCOM, zato posledično deluje samo na Microsoftovem operacijskem sistemu Windows [20]. To se je kasneje izkazalo kot omejitev, zato je bila leta 2008 izdana specifikacija OPC UA (angl. OPC Unified Architecture), ki deluje na vseh operacijskih sistemih [19].
Naš sistem omogoča uporabo tako OPC DA kot tudi OPC UA komunikacijskega protokola, konkretno na opisani liniji pa je uporabljen OPC DA-protokol, saj izbrani PLC-krmilniki podpirajo ta protokol brez dodatne konfiguracije. Prav tako nismo omejeni z operacijskim sistemom, saj je na strežniku na proizvodnji liniji nameščen operacijski sistem Windows.

5.2.2 WebSocket

WebSocket je standardiziran komunikacijski protokol, ki preko ene TCP-povezave omogoča polno dvosmerno (angl. full-duplex) komunikacijo med odjemalcem in strežnikom [17]. V preteklosti se je za vzpostavitev tovrstne komunikacije zlorabljalo HTTP-protokol v smislu kontinuiranega preverjanja (angl. polling) strežnika, če ima kakšno novo informacijo [17]. V tem primeru mora strežnik za vsako zahtevo in za vsak odgovor odpirati novo TCP-povezavo. Poleg tega pa pri tem pristopu prihaja do zahtev, na katere strežnik nima odgovora. Rešitev tega problema je uvedba ene TCP-povezave za dvosmerno komunikacijo, kar poznamo pod imenom WebSocket [17]. Povezava med strežnikom in odjemalcem ostaja vedno odprta, kar pomeni, da lahko tako strežnik kot odjemalec kadarkoli pošiljata in sprejemata zahteve. Vzpostavitev komunikacije preko WebSocket protokola se začne s fazo rokovanja (angl. handshake). V tej fazi odjemalec pošlje zahtevo za nadgraditev HTTP-protokola v WebSocket protokol strežniku [17]. V kolikor strežnik odgovori na zahtevo, to pomeni, da strežnik podpira WebSocket protokol. Faza rokovanja je pomembna, saj v starejših različicah spletnih brskalnikov protokol WebSocket ni podprt.

5.3 NAČRTOVANJE KOMPONENT

V tem poglavju bomo podrobneje opisali načrtovanje posameznih komponent sistema za sledljivost izdelkov, v nadaljevanju pa načrtovanje aplikacije odjemalca, OPC-strežnik, spletni strežnik, lokalno strežniško aplikacijo in lokalno podatkovno bazo. Poleg načrtovanja bodo predstavljene tudi tehnologije, s pomočjo katerih bomo te komponente v naslednjem koraku implementirali.

5.3.1 OPC-strežnik

OPC-strežnik bo v našem sistemu omogočen s produktom KEPServerEX. KEPServerEX zagotavlja vmesnik med našim sistemom ter PLC-krmilniki. Razvilo ga je podjetje Kepware, ki se ukvarja z razvojem programskih rešitev za povezovanje različnih naprav v svetu industrije. Produkt KEPServerEX, ki je v vlogi OPC-strežnika, preko svojega uporabniškega vmesnika omogoča povezovanje, spremljanje in nadzorovanje povezanih naprav. Povezava med OPC-strežnikom in PLC-krmilniki je omogočena preko različnih gonilnikov [8]. Gonilnike za povezovanje na naprave različnih proizvajalcev podjetje Kepware prodaja posebej, torej moramo za vsak tip naprave dokupiti gonilnik. Podatki iz PLC-krmilnika so predstavljeni s tako imenovanimi značkami, pri čemer ima vsaka značka določen podatkovni tip ter naslov, v katerem podatkovnem bloku PLC-krmilnika se podatek nahaja. Strežnik KEPServerEX omogoča, da značke konstantno beremo z določenim časovnim zamikom, lahko pa jih preberemo na zahtevo.
Izmenjevanje podatkov med našim sistemom ter PLC-krmilniki bo torej potekalo preko komunikacijskega protokola OPC, ki je podrobneje opisan v podpoglavju 5.2.1. Na strežniški aplikaciji bo treba konfigurirati OPC-odjemalce. Zaradi zmogljivosti ter neodvisnosti je priporočljivo, da za vsak PLC-krmilnik konfiguriramo lastnega klienta. Kako konfiguriramo kliente na strežniški aplikaciji, je podrobneje predstavljeno v poglavju 6. OPC-strežnik za komunikacijo s PLC-krmilniki uporablja Siemens TCP/IP-gonilnik, saj se bodo v primeru opisane proizvodne linije uporabljali PLC-krmilniki podjetja Siemens.

5.3.2 Lokalna strežniška aplikacija

Lokalna strežniška aplikacija bo implementirana z uporabo programskega jezika java in ogrodjem Spring Boot. To ogrodje nudi podporo pri izgradnji spletnih javanskih aplikacij [12]. Je del širšega aplikacijskega ogrodja Spring. Namen ogrodja Spring Boot pa je, da čim hitreje ter z malo konfiguracije vzpostavimo Spring aplikacijo oziroma spletni servis [12]. To je bil tudi eden glavnih razlogov za uporabo tega ogrodja. Vsebuje že konfigurirane, najpogosteje uporabljene knjižnice, ki pa jih lahko po želji tudi spremenimo oziroma zamenjamo [12]. Za upravljanje z odvisnostmi, knjižnicami ter za prevajanje kode bomo v našem projektu uporabljali orodje Maven. Vgrajen strežnik Tomcat pa bo poskrbel za zagon spletne aplikacije ter spletnih servisov. Za potrebe našega projekta bomo morali v projekt vključiti tako knjižnice, ki jih ponuja ogrodje Spring, kot tudi nekaj knjižnic izven tega ogrodja. Knjižnice, ki jih planiramo uporabiti v projektu, so naslednje:

- *spring-boot-starter-web* – knjižnica je uporabljena za izdelavo spletnih aplikacij ter RESTful aplikacij [13],

- *spring-boot-starter-batch* – knjižnica pomaga pri implementaciji funkcij, ki se izvajajo v nekem zaporedju, naj bo to časovno zaporedje ali pa vezano na nek prožilec,
- *spring-boot-starter-test* – knjižnica, ki je uporabljena za testiranje Spring aplikacij. Med drugim vsebuje tudi ogrodje za testiranje JUnit [13],
- *spring-boot-starter-jdbc* – knjižnica, ki omogoča pisanje in klicanje SQL-poizvedb znotraj javanske kode,
- *spring-boot-starter-integration* – ta knjižnica omogoča integracijo zunanjih sistemov v našo aplikacijo. V našem primeru to pomeni konfiguracijo OPC-strežnika, povezavo na optične bralnike kod itd. [14],
- *spring-boot-starter-websocket* – ta knjižnica omogoča vzpostavitev WebSocket protokola med našo aplikacijo ter ciljno napravo [13],
- *jtds* – je odprto kodna knjižnica, ki prinaša v našo aplikacijo JDBC-gonilnik za Microsoft SQL Server, ki je lokalna baza v tem sistemu [16],
- *ojdbc* – ta knjižnica prinaša v našo aplikacijo JDBC-gonilnik za Oracle podatkovno bazo, ki je poslovna podatkovna baza,
- *openscada* – knjižnica, ki omogoča povezavo in pisanje podatkov na OPC-strežnik ter branje podatkov iz OPC-strežnika [1],
- *jackson* – knjižnica, ki prinaša podporo razčlenjevanju JSON-datotek v javanski kodi. V našem projektu je uporabljena pri upravljanju z recepti, saj so recepti za delovna mesta napisani v JSON-formatu,
- *jpedal* – ta knjižnica omogoča pretvarjanje PDF-dokumentov v slikovne dokumente, ki jih nato lažje prikažemo na delovnem mestu,
- nazadnje je tu še lastna, v podjetju razvita javanska knjižnica, ki se jo uporablja tudi v drugih projektih znotraj podjetja. Ta knjižnica vsebuje vse potrebne objekte, servise ter objekte za dostop do podatkov (angl. Data Access Object − DAO), ki so vmesnik med aplikacijo ter podatkovno bazo.

Spring Boot aplikacija bo imela v našem sistemu vlogo zalednega (angl. backend) sistema. Aplikacija bo morala omogočiti dostop do funkcij preko unikatnega URL-naslova, npr. ”/api/imeFunkcije”. V spletni aplikaciji bomo nato uporabili klice standardnih HTTP-metod, in sicer GET, POST, PUT in DELETE. To funkcionalnost bomo omogočili z uporabo arhitekture za izmenjavo podatkov med spletnimi storitvami (angl. representational state transfer – REST). Strežniško aplikacijo bomo razvijali v razvojnem okolju Spring Tool Suite, v katerem bomo lahko za potrebe testiranja aplikacijo tudi zagnali. Za zaganjanje v produkcijskem načinu bomo zgradili .jar datoteko.

Aplikacija bo povezana na OPC-strežnik, optične bralnike črtnih kod, lokalno podatkovno bazo, spletni strežnik, aplikacijo odjemalca, tiskalniški strežnik in na poslovno podatkovno bazo. Za dostop do vseh naštetih naprav in storitev bo morala aplikacija podpirati različne komunikacijske protokole, kot so OPC, WebSocket, HTTP in JDBC. Vse naštete storitve

in povezave so predstavljene na Sliki 8. Strežniška aplikacija bo gostovala na naslovu »http://192.168.10.20:8090«. Komunikacijski kanal med odjemalcem in strežnikom bo omogočen s protokolom WebSocket na naslovu »http://192.168.10.20:8090/topic/wcdispay«. Na ta naslov bodo s strani strežnika posredovane vse informacije, ki so namenjene na aplikacijo odjemalca. Na ta naslov bomo na primer pošiljali vsebino optično prebrane črtne kode ali pa vrednost določene PLC-značke. Komunikacija med optičnim bralnikom črtnih kod in strežniško aplikacijo bo potekala skozi vrata 5310, med OPC-strežnikom in strežniško aplikacijo skozi vrata 135, med lokalno podatkovno bazo in strežniško aplikacijo skozi vrata 1433 ter med poslovno podatkovno bazo in strežniško aplikacijo skozi vrata 1521. Ti podatki so zelo pomembni, saj moramo konfigurirati požarni zid sistema, da dovoli promet skozi omenjena vrata.

5.3.3 Aplikacija odjemalca

Aplikacija odjemalca bo razvita z orodjem Angular. To je platforma in ogrodje za izdelovanje spletnih aplikacij. Aplikacije se piše v programskem jeziku TypeScript, ki je nadgradnja JavaScripta [3]. Programski jezik TypeScript omogoča uporabo tipov, vmesnikov, razredov … Te lastnosti pomagajo pri hitrejšem ter lažjem programiranju. Napisana TypeScript koda se nato prevede v običajno JavaScript kodo. Ta korak je potreben zato, ker spletni brskalniki ne znajo prikazati aplikacije, napisane v TypeScriptu [6]. Angular temelji na modularnem pristopu, ki se ga poskušamo držati tudi pri razvoju lastne aplikacije. Za nadzor ter dodajanje novih modulov je v našem projektu uporabljen npm (angl. node package manager). Angular aplikacije so razdeljene na komponente (angl. components), ki so sestavljene iz aplikacijske kode v TypeScriptu, ter kode za prikaz, ki je sestavljena iz HTML-ja in CSS-ja. Angular ponuja kar nekaj značk znotraj HTML-kode, ki pospešijo in olajšajo razvoj. Težimo k temu, da komponente pišemo čim bolj splošno, saj lahko na ta način komponento večkrat uporabimo in tako ne ponavljamo programske kode po nepotrebnem. Komponente, do katerih želimo dostopati preko URL-naslova, moramo dodati v poseben modul za preusmerjanje (angl. routing module), kjer jim tudi enolično določimo URL-naslov. Pomemben del Angular aplikacije so tudi tako imenovane storitve (angl. services), ki so specializirani razredi, namenjeni opravljanju točno določene funkcije [7]. Komponente in storitve so v Angularju ločene z namenom doseganja modularnosti ter ponovne uporabnosti kode [3]. Komponente navadno uporabljajo storitve za klic funkcij strežniške aplikacije. V našem primeru tako preko storitev in z uporabo standardnih HTTP-metod kličemo pripravljene funkcije iz naše Spring Boot aplikacije. Na ta način pridobimo vse potrebne podatke iz strežniške aplikacije.

Pri razvoju Angular aplikacije nam zelo pomaga orodje Angular CLI, s pomočjo katerega lahko generiramo nove komponente ter storitve, končno prevedemo ter zgradimo projekt, itd. [5] Z ukazom »ng new imeAplikacije« postavimo začetno ogrodje projekta. Nato pa z

ukazom »ng generate component imeKomponente« naredimo ogrodje za novo Angular komponento, z ukazom »ng generate service imeStoritve« naredimo ogrodje za novo storitev. To orodje omogoča tudi hitro gostovanje Angular aplikacij v fazi razvoja in testiranja. Gostovanje Angular aplikacije sprožimo z ukazom »ng serve«. Tako gostovanje aplikacije pospeši predvsem fazo testiranja, saj ni potrebno zgraditi aplikacije za vsako spremembo, temveč se ob vsaki spremembi aplikacija avtomatsko osveži. V fazi produkcije bomo Angular aplikacijo prevedli in zgradili z orodjem Angular CLI ter jo nato gostovali na spletnem strežniku Apache, ki bo predstavljen v nadaljevanju. S tem ko Angular aplikacijo zgradimo za produkcijski način, pridobimo na hitrosti nalaganja spletne aplikacije ter tudi na njeni odzivnosti. Tako gostovanje v fazi testiranja aplikacije ni praktično, saj je treba aplikacijo vedno znova zgraditi.
Aplikacija odjemalca bo neposredno povezana s spletnim strežnikom preko HTTP-povezave skozi vrata 8080. Bo pa aplikacija odjemalca tudi neposredno povezana s strežniško aplikacijo preko protokola WebSocket. Ta komunikacijski kanal bo rezerviran za časovno kritične podatke, kot so podatki iz PLC-krmilnika, ki jih moramo čim hitreje obdelati.

5.3.4 Spletni strežnik

Za produkcijsko gostovanje aplikacije odjemalca uporabljamo spletni strežnik Apache. Spletni strežnik je sistem, ki obdeluje in posreduje zahteve z uporabo HTTP-protokola [11]. Glavna vloga spletnega strežnika je obdelava in gostovanje spletnih strani vsem povezanim odjemalcem. Komunikacija med spletnim strežnikom in odjemalcem se začne s HTTP-zahtevo, ki jo pošlje spletni brskalnik na odjemalcu. Strežnik nato vrne odgovor na to zahtevo v obliki zahtevane vsebine ali pa napake [11]. Apache je trenutno najpopularnejši spletni strežnik, saj ga uporablja 44,6 % vseh znanih spletnih strani [18].
Spletna aplikacija odjemalca gostuje na standardnem naslovu »http://192.168.10.20:8080«. Spletni strežnik je potrebno konfigurirati, da vse zahteve, ki pridejo s strani odjemalca in se začnejo na »/api«, posreduje na naslov »http:// 192.168.10.20:8090/api«.

5.3.5 Lokalna podatkovna baza

Za relacijsko podatkovno bazo smo izbrali Microsoft SQL Server Express Edition. To relacijsko podatkovno bazo se je v podjetju uporabljalo že pri prejšnjih projektih, kar je bil tudi eden od razlogov za izbiro tega sistema. Ena od omejitev te zastonjske verzije relacijske podatkovne baze je, da lahko shrani do 10 GB podatkov. Ker se varnostno kopiranje izvaja v realnem času na poslovno podatkovno bazo Oracle, omejitev Microsoft SQL Express Serverja ne predstavlja velikega problema, saj lahko starejše podatke brišemo iz lokalne podatkovne baze. Za uporabniški vmesnik baze je uporabljen Microsoft SQL

Server Management Studio. Preko uporabniškega vmesnika je enostavno kreirati nove tabele. To orodje pa nudi še naprednejša orodja, kot sta analiza najzahtevnejših poizvedb v realnem času ter analiza porabljenega prostora.
Nabor tabel relacijske podatkovne baze bo v nadaljevanju predstavljena v treh delih. Prvi del je viden na Sliki 10, drugi del na Sliki 11 in tretji del na Sliki 12.

Tabela FileItem

Tabela »FileItem« vsebuje podatke o parametrih, ki jih preberemo iz datotek. Identifikator *ID_FILE_ITEM* je hkrati tudi primarni ključ tabele. Ime parametra, ki ga bomo prebrali iz datoteke, je navedeno v atributu *ITEM_NAME*. Atribut *ID_ITEM_TYPE* pove tip parametra, kjer število 1 prestavlja znakovni niz, 2 cela števila, 3 realna števila, 4 datume in 5 Boolovo vrednost. Atributi, ki sledijo, so opcijski. *ITEM_TYPE* lahko vsebuje ime podatkovnega tipe, *ITEM_DESC* lahko vsebuje opis podatkovnega tipa, *ITEM_FORMAT* pa v kakšnem formatu se zapiše realno število (koliko decimalnih mest upoštevamo), *GRP_SEPAR* lahko določa ločilo tisočic, *DEC_SEPAR* pa decimalno ločilo, ki je uporabljeno. Potem sta tu še obvezna atributa *WORK_CENTER* in *PLANT*, ki določata na naziv delovnega mesta in obrata, na katerem bomo prebrali ta parameter iz datoteke.

Tabela BarcodeScanner

Tabela »BarcodeScanner« je namenjena hranjenju osnovnih podatkov o optičnih bralnikih črtnih kod. Vsak zapis ima identifikator *ID_BC_SCANNER*, ki je hkrati tudi primarni ključ tabele. Atribut *WORK_CENTER* vsebuje naziv delovnega mesta, atribut *PLANT* pa naziv obrata v podjetju, kjer se optični bralnik kod nahaja. Atribut *NAME* vsebuje ime optičnega bralnika kod. Nato sledita še atributa *IP_ADDRESS* in *PORT*, ki služita za identifikacijo optičnega bralnik kod v omrežju.

Tabela MachineCommReq

Tabela »MachineCommReq« vsebuje dnevnik poslanih ukazov oziroma podatkov na PLC-krmilnik. Služi predvsem kot pomoč pri odpravljanju napak. Identifikator *ID_MACH_COMM_REQ* je hkrati tudi primarni ključ tabele. Atribut *ID_SESSION* pove, v kateri seji so bili poslani podatki na PLC-krmilnik. Za informacije o času, kdaj smo poslali podatke na PLC-krmilnik, služi atribut *CREAT_TS*. Atribut *COMMAND* pove, kateri ukaz smo poslali na PLC-krmilnik. V atributu *REQUEST* imamo zabeleženo celotno zahtevo, ki smo jo poslali na PLC-krmilnik. Atributa *WORK_CENTER* in *PLANT* pa sporočata, na katero delovno mesto in obrat smo poslali zahtevo.

Tabela MachineRecipeLog

Tabela »MachineRecipeLog« je namenjena shranjevanju lokalnih receptov, ki jih preberemo s PLC-krmilnika. Identifikator *ID_MACH_REC_LOG*, ki je hkrati tudi primarni

ključ tabele. Atribut *CREAT_TS,* poda informacijo, kdaj je bil recept shranjen. Atribut *PROD_MATERIAL_NR* vsebuje kodo izdelka, za katero je ta recept namenjen. Potem sta tu neobvezna atributa *ORDER_NR* in *OPERATION_NR*, ki sporočata, na kateri delovni nalog ter na katero operacijo je vezan shranjen recept. Atributa *WORK_CENTER* in *PLANT* določata, na katerem delovnem mestu in obratu je shranjen recept. Atribut *RECIPE_DESC* vsebuje ime, atribut *RECIPE_CONT* pa vsebino shranjenega recepta. Zadnji atribut je *DATA_BACKUP*, ki sporoča, ali je vsebina zapisa že varnostno kopirana v poslovno podatkovno bazo.

Tabela MachineRecipeMetaData

Tabela »MachineRecipeMetaData« vsebuje osnovno obliko recepta oziroma metapodatke o receptu. Služi nam kot pomoč pri izdelavi recepta. Vsako delovno mesto ima en zapis meta podatkov o receptu. Identifikator *ID_MACH_REC_MD* je hkrati tudi primarni ključ tabele. Atribut *CREAT_TS* pove, kdaj smo naredili zapis o meta podatkih. Atributa *WORK_CENTER* in *PLANT* določata, na katero delovno mesto in obrat je vezan ta meta recept. Sledita še atributa *RECIPE_DESC*, ki vsebuje ime meta recepta in *RECIPE_CONT*, ki vsebuje vsebino meta recepta.

Tabela PackagingUnit

Tabela »PackagingUnit« vsebuje informacije o že ustvarjenih pakirnih enotah. Identifikator *ID_PACK_UNIT* je hkrati tudi primarni ključ tabele. Atributa *CREAT_TS* in *CHANG_TS* sporočata, kdaj je bila pakirna enota ustvarjena ter kdaj je bil nazadnje dodan kakšen izdelek v pakirno enoto. Atribut *MATERIAL_NR* pove, katera številka materiala je zapakirana, *QUANT_PACK* pa koliko izdelkov je že zapakiranih v tej pakirni enoti. Atribut *UNIT* sporoča, v kakšni enoti so izdelki zapakirani, navadno je to kos. Potem sledi atribut *SSCC_PACK*, ki vsebuje enolično oznako pakirne enote. Sledita atributa *ORDER_NR*, ki sporoča, na kateri delovni nalog je vezana pakirna enota, ter atribut *CONFIRM_NR*, ki vsebuje številko potrditve. V atributu *PRINT_LABEL* je shranjena informacija o tem, ali je bila natisnjena nalepka za pakirno enoto. Atribut *USER* sporoča, kateri uporabnik je ustvaril pakirno enoto. Atributa *WORK_CENTER* in *PLANT* pa določata, na katerem delovnem mestu in obratu je bila pakirna enota ustvarjena. Potem sta tu še atributa *REC_STAT*, ki sporoča, ali je pakirna enota že zaključena, in atribut *DATA_BACKUP*, ki sporoča, ali je vsebina zapisa že varnostno kopirana v poslovno podatkovno bazo.

Tabela PackagingUnitList

Tabela »PackagingUnitList« vsebuje informacije o tem, katere številke materialov dovolimo pakirati na nekem delovnem mestu. Identifikator *ID_PU_LIST* je hkrati primarni ključ tabele. Atributa *WORK_CENTER* in *PLANT* določata delovno mesto in obrat, na katerem dovolimo pakirati izdelek, atribut *MATERIAL_NR* pa vsebuje naziv tega izdelka.

Potem imamo še atribute, ki nam pomagajo pri izrisu vizualizacije pakirne enote. To so atribut *MAX_QUANT_PACK*, ki vsebuje količino, atribut *NUM_LEVELS*, ki pove, koliko nivojev je v pakirni enoti, atribut *NUM_X_AXIS*, ki sporoča, koliko izdelkov je v dolžini pakirne enote, in atribut *NUM_Y_AXIS*, ki poda informacijo o količini izdelkov v širini pakirne enote.

Tabela PackagingUnitMaterial

Tabela »PackagingUnitMaterial« vsebuje informacije o tem, kateri izdelki so zapakirani na pakirni enoti. Identifikator *ID_PU_MAT* je hkrati tudi primarni ključ tabele. Identifikator *ID_PACK_UNIT* pa je tuji ključ tabele in je vezan na tabelo »PackagingUnit«. Atribut *CREAT_TS* pove, kdaj je bil izdelek zapakiran. Serijska številka zapakiranega izdelka je zabeležena v atributu *SERIAL_NR*. Potem imamo šest opcijskih atributov, v katerih lahko zabeležimo dodatne informacije o zapakiranem izdelku. Sledita atributa *ORDER_NR* in *CONFIRM_NR*, ki sporočata, na kateri delovni nalog je vezan zapakiran izdelek ter pod katero številko potrditve. Atributa *WORK_CENTER* in *PLANT* sporočata, na katerem delovnem mestu in obratu je bil izdelek zapakiran. Zadnji atribut *REC_STAT* pa poda informacijo o stanju zapakiranega izdelka.

Tabela PackagingUnitScript

Tabela »PackagingUnitScript« se uporablja za določitev skripte za preverjanje vsebine optično prebrane kode izdelka. Identifikator *ID_PU_SCRIPT* je hkrati tudi primarni ključ tabele. Atributa *WORK_CENTER* in *PLANT* določata, na katerem delovnem mestu in obratu se ta skripta uporablja. Atribut *MATERIAL_NR* vsebuje številko izdelka, za katerega je namenjena ta skripta. Sledi atribut *SCRIPT_TYPE*, ki vsebuje ime skripte. Identifikator *ID_SCRIPT_LIST* je hkrati tudi tuji ključ tabele in je vezan na tabelo »ScriptList«.

Tabela ScriptList

Tabela »ScriptList« vsebuje dejansko skripto za preverjanje vsebine optično prebrane kode izdelka. Identifikator *ID_SCRIPT_LIST* je hkrati tudi primarni ključ tabele. Atribut *NAME* določa ime skripte. Sledi atribut *SCRIPT*, v katerem je shranjena vsebina skripte za preverjanje. V atributu *PARAMS* imamo določene parametre, ki so potrebni za izvedbo skripte. Nazadnje imamo še atribut *DESCRIPTION*, ki vsebuje kratek opis skripte.

Tabela LogisticUnit

Tabela »LogisticUnit« je namenjena beleženju materiala na delovnih mestih. Identifikator *ID_LOG_UNIT* je hkrati tudi primarni ključ tabele. Sledijo trije atributi *CREAT_TS*, *MODIF_TS* in *LAST_CAPTUR_TS*, ki sporočajo, kdaj je material prispel na delovno mesto, kdaj je bil spremenjen in kdaj je bil nazadnje uporabljen. Sledi atribut *COMP_NR*,

ki predstavlja številko materiala. Naslednji atribut *COMP_BATCH* pove, iz katere šarže je material. Sledi atribut *COMP_SSCC*, ki sporoča, iz katere logistične enote smo pridobili ta material. Naziv optičnega bralnik kod za materialno sledljivost beležimo v atributu *SCANNER_NR*. Nato sledita atributa *QUANITITY* in *CURRENT_QTY*, ki povesta celotno količino materiala, ki je prišla s to pakirno enoto, ter količino materiala, ki je trenutno na delovnem mestu. V atributu *UNIT* beležimo enoto, v kateri nastopa ta material. Sledita še atributa *WORK_CENTER* in *PLANT*, ki določata, na katerem delovnem mestu in obratu je uporabljen ta material. Nazadnje imamo še atribut *REC_STAT*, ki določa, ali je ta material še aktiven ali ne.

Tabela PLCItem

Tabela »PLCItem« vsebuje podatke in vrednosti spremenljivk oziroma značk, ki jih preberemo iz PLC-krmilnika. Identifikator *ID_PLC_ITEM* je hkrati tudi primarni ključ tabele. Atribut *OPC_SERVER* določa ime OPC-strežnika. Atributa *PATH* in *ITEM_NAME* skupaj določata pot do spremenljivke v PLC-krmilniku. Sledi atribut *ID_ITEM_TYPE*, ki določa tip spremenljivke, kot smo spoznali v tabeli »FileItem«. Atributa, ki sledita, sta opcijska. *ITEM_TYPE* lahko vsebuje ime podatkovnega tipe in *ITEM_DESC*, ki lahko vsebuje opis podatkovnega tipa. Atribut *CHANG_MON* pove, ali zabeležimo vsako spremembo vrednosti te spremenljivke. Nato sledijo atributi za spremljanje trenutne in prejšnje vrednosti spremenljivk za vsak podatkovni tip posebej. Sledi atribut *CAPTURED*, ki sporoča, kdaj je bila nazadnje spremenljivka prebrana. Nato sledijo trije atributi, ki so uporabljeni, če želimo PLC-krmilniku avtomatsko odgovarjati. Na ta način hitro opazimo, če povezava med PLC-krmilnikom in strežnikom pade.

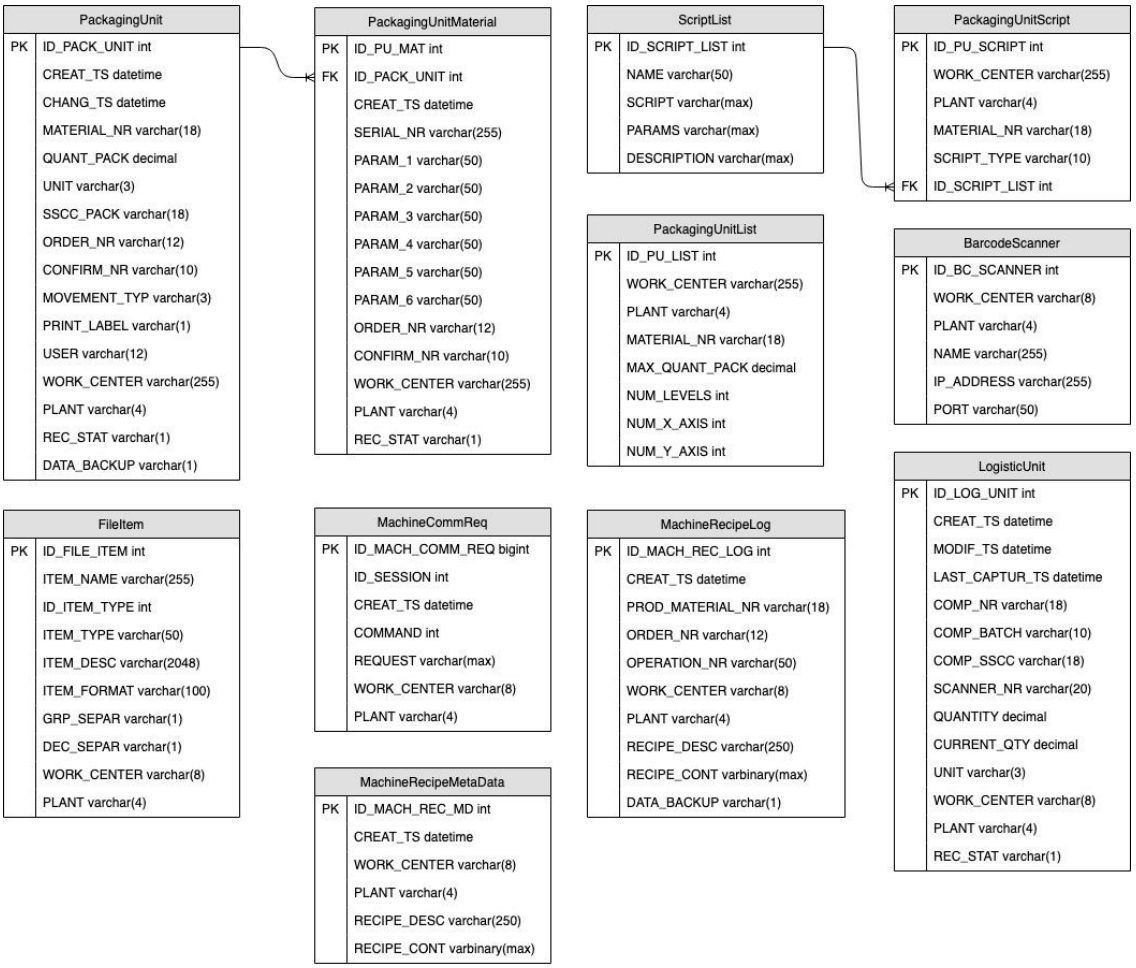
Tabela ProductErrorList

Tabela »ProductErrorList« vsebuje podatke o napakah, ki se jih beleži na proizvodni liniji. Identifikator *ID_PRODUCT_ERROR* je hkrati tudi primarni ključ tabele. Atributa *WORK_CENTER* in *PLANT* določata, na katerem delovnem mestu in obratu se beleži ta napaka. Sledi unikatna številka napake, ki se beleži v atributu *ERROR_NR*. Nazadnje imamo še dva atributa za kratek in dolg opis napake, in sicer *PROD_DESC_SHORT* ter *PROD_DESC_LONG*.

Tabela WorkCenterList

V tabeli »WorkCenterList« hranimo podatke o vseh delovnih mestih na proizvodnji liniji. Identifikator *ID_WORK_CENTER* je hkrati primarni ključ tabele. Atribut *PLANT* določa, v kateri obrat spada to delovno mesto. Sledita atributa *WORK_CENTER*, ki določa številko delovnega mesta, in atribut *NAME*, ki določa ime delovnega mesta. Sledi atribut *IP_ADDRESS*, ki pove IP-naslov prikazovalnika na tem delovnem mestu. Nazadnje imamo

še atribut *ID_PARENT*, ki ga uporabimo v primeru, ko obstaja neko nadrejeno delovno mesto.



Slika 10: Prikaz nabora tabel podatkovne baze – prvi del.

Tabela ProdStat

Tabela »ProdStat« vsebuje podatke o zaporednem števcu izdelkov na delovnem mestu v enem dnevu. Uporablja se recimo za generiranje serijske številke izdelka. Identifikator *ID_PROD_STAT* je hkrati tudi primarni ključ tabele. Atributa *WORK_CENTER* in *PLANT* določata, na katerem delovnem mestu se beleži števec. Sledi atribut *DATE*, ki pove, na kateri datum je beležen števec. V zadnjem atributu *COUNT_PROD* imamo trenutno vrednost števca.

Tabela TraceDataHead

Tabela »TraceDataHead« je glavna tabela, v katero se beleži sledljivost izdelka po posameznih delovnih mestih. Identifikator *ID_TRC_DATA_HEAD* je hkrati tudi primarni ključ tabele. Sledi atribut *CREAT_TS*, ki pove, kdaj je bil izdelek na delovnem mestu. Sledita atributa *PROD_MATERIAL_NR*, ki sporoči kodo izdelka, ter *SERIAL_NR*, ki pove

serijsko številko izdelka. Atribut *ORDER_NR* poda informcijo, pod katerim delovnim nalogom se izdelek izdeluje. Sledi atribut *QUALITY*, ki pove, ali je bil izdelek na delovnem mestu dober ali slab. V primeru, da je bil izdelek slab, se v naslednje dva atributa *ERROR_NR* in *ERROR_DESC* vpiše kodo ter opis napake, zaradi katere je bil izdelek na delovnem mestu slab. Atribut *PLANT* sporoča, v katerem obratu se izdelek izdeluje. Sledita atributa *WORK_CENTER_CURR*, ki pove, katero je trenutno delovno mesto, na katerem je izdelek, in atribut *WORK_CENTER_NEXT*, ki posreduje podatek o naslednjem delovnem mestu, kamor je izdelek namenjen. Zadnji atribut je *DATA_BACKUP*, ki sporoča, ali je bil zapis že varnostno kopiran v poslovno podatkovno bazo. V kakšnem razmerju je tabela »TraceDataHead« povezana z ostalimi tabelami, vidimo na Sliki 10. Tabele na sliki so najpomembnejše s stališča spremljanja sledljivosti izdelka po delovnih mestih, shranjevanja parametrov ter beleženja uporabljenega materiala.

Tabela TraceProductComp

V tabelo »TraceProductComp« se vpisuje, kateri material je bil uporabljen na nekem delovnem mestu. Identifikator *ID_TRC_PRODUCT_COMP* je hkrati tudi primarni ključ tabele. Identifikator *ID_TRC_DATA_HEAD* pa je tuji ključ, vezan na tabelo »TraceDataHead«. Sledijo atributi *COMP_NR*, ki vsebuje številko porabljenega materiala, atribut *COMP_BATCH*, ki posreduje podatek o šarži, iz katere je material, in atribut *COMP_SSCC*, ki pove, v kateri logistični enoti je bil uporabljen material. Če porabljen material vsebuje serijsko številko, se to zapiše v atribut *COMP_SERIAL_NR*. Koliko enot materiala se je porabilo, se vpiše v atribut *QUANTITY*, samo enoto pa v atribut *UNIT*. Zadnji atribut je *DATA_BACKUP*, ki sporoča, ali je bil zapis že varnostno kopiran v poslovno podatkovno bazo.

Tabela TraceProductParam

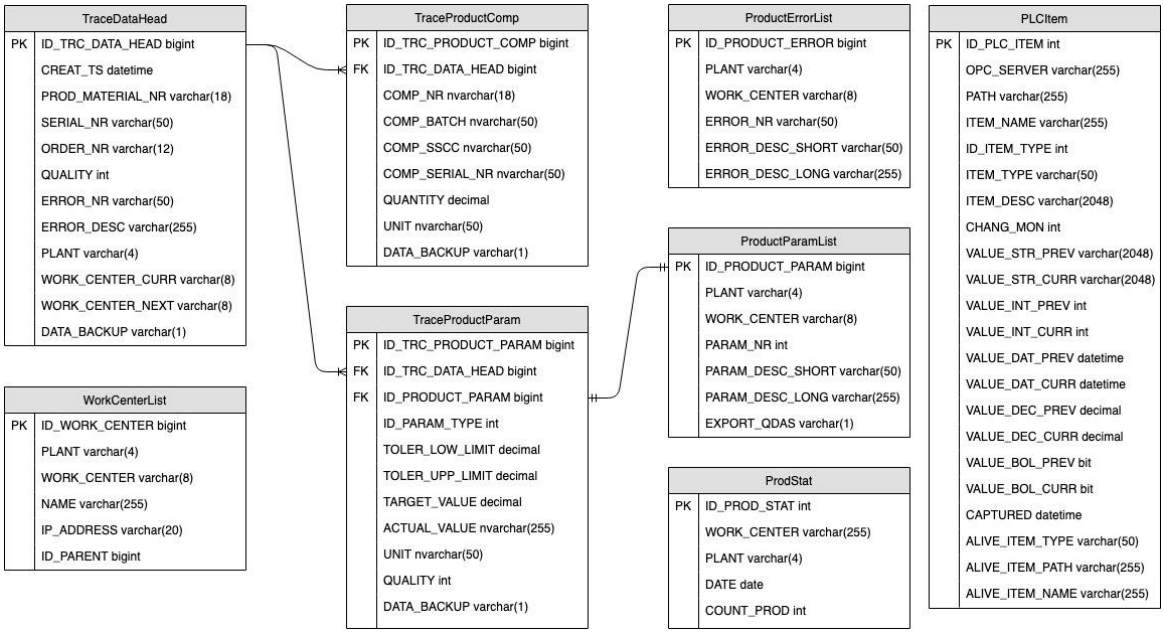
V tabelo »TraceProductParam« se zapišejo vsi procesni parametri, ki jih preberemo s PLC-krmilnika ali pa iz datoteke. Identifikator *ID_TRC_PRODUCT_PARAM* je hkrati primarni ključ tabele. Identifikator *ID_TRC_DATA_HEAD* je tuji ključ, vezan na tabelo »TraceDataHead«. Sledi še en identifikator *ID_PRODUCT_PARAM*, ki je tuji ključ, vezan na tabelo »ProductParamList«. Sledi atribut *ID_PARAM_TYPE*, ki pove podatkovni tip parametra. Nato sledijo trije atributi, ki služijo za hitro kontrolo meritve, in sicer *TOLER_LOW_LIMIT*, *TOLER_UPP_LIMIT* in *TARGET_VALUE*. To so atributi, ki povejo zgornjo in spodnjo toleranco ter pričakovano vrednost. Dejanska vrednot parametra se zapiše v atribut *ACTUAL_VALUE*. Enoto meritve se lahko zabeleži v atribut *UNIT*. Atribut *QUALITY* pa vsebuje kvaliteto meritve. Zadnji atribut je *DATA_BACKUP*, ki sporoča, ali je bil zapis že varnostno kopiran v poslovno podatkovno bazo.

Tabela ProductParamList

Tabela »ProductParamList« vsebuje podatke o vseh parametrih, ki se jih beleži na proizvodni liniji. Identifikator *ID_PRODUCT_PARAM* je hkrati tudi primarni ključ tabele. Sledita atributa *PLANT* in *WORK_CENTER*, ki sporočata, na katerem obratu in delovnem mestu se beleži ta parameter. Sledi številka parametra, ki se beleži v atributu *PARAM_NR*. Sledita še atributa *PARAM_DESC_SHORT* in *PARAM_DESC_LONG*, ki vsebujeta kratek in dolg opis parametra. Nazadnje imamo še atribut *EXPORT_QDAS*, ki sporoča, ali se parameter izvaža v program za statistične obdelave QDAS ali ne.

Tabela OrderHead

V tabeli »OrderHead« se beležijo vsi delovni nalogi, ki so bili delani na proizvodni linij. Ta tabela je obsežna, zato bomo opisali samo atribute, ki jih potrebujemo. Identifikator *ID_ORD_HEAD* je hkrati tudi primarni ključ tabele. Atribut *CREAT_TS* pove, kdaj so bile informacije o nalogu zapisane v tabelo. Sledi atribut *ORDER_NR*, ki vsebuje številko delovnega naloga. V atributu *ORDER_STAT* dobimo podatke o tem, ali je nalog aktiven ali ne. Atribut *PLANT* pove, na katerem obratu je bil nalog razpisan. Sledita še atributa *MATERIAL_NR* in *MATERIAL_DESC*, ki vsebujeta informacije o izdelku, za katerega je nalog razpisan. Pomemben je še atribut *QUANT_TOTAL*, ki sporoča količino izdelkov, predvidenih za izdelavo na tem delovnem nalogu.

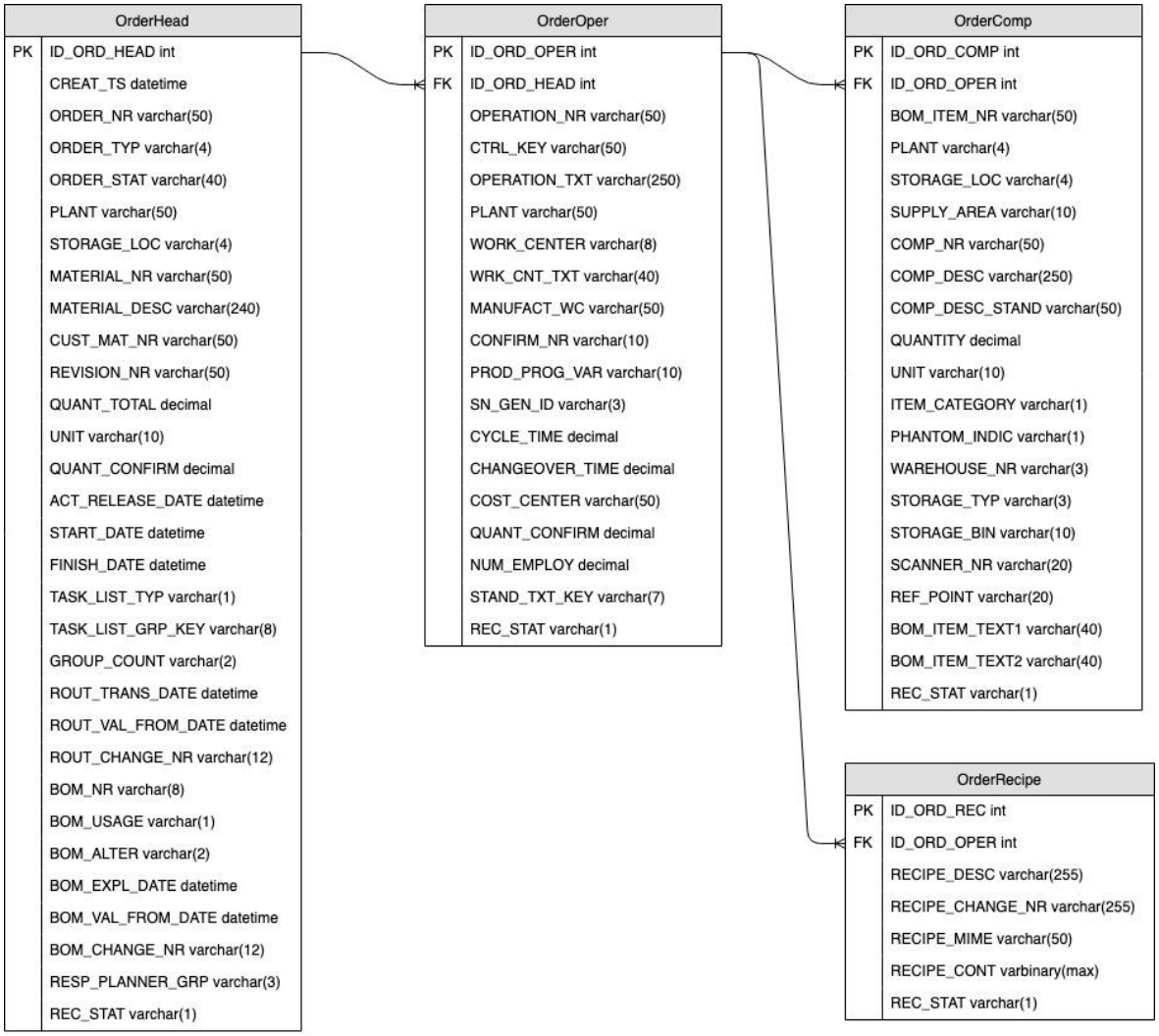


Slika 11: Prikaz nabora tabel podatkovne baze – drugi del.

Tabela OrderOper

Tabela »OrderOper« vsebuje podatke o operacijah, ki so predvidene na nekem delovnem nalogu. Tudi o tej tabeli bomo pisali le najnujnejše atribute. Identifikator *ID_ORD_OPER* je hkrati tudi primarni ključ tabele. Identifikator *ID_ORD_HEAD* je tuji ključ, ki je vezan

na tabelo »OrderHead«. Sledi atribut *OPERATION_NR*, ki je številka operacije, kot je zabeležena na SAP-poslovnem sistemu, atribut *OPERATION_TXT* pa predstavlja tekstovni opis te operacije. Sledijo atributi *PLANT, WORK_CENTER* in *WRK_CNT_TXT*, ki sporočijo delovno mesto, obrat in tekstovni opis delovnega mesta, na katerem se izvaja operacija. Pomemben je še atribut *PROD_PROG_VAR*, iz katerega lahko dobimo informacijo o številki programa, ki jo nato posredujemo na PLC-krmilnik.



Slika 12: Prikaz nabora tabel podatkovne baze – tretji del.

Tabela OrderComp

Tabela »OrderComp« vsebuje podatke o materialih, ki jih potrebujemo na posameznem delovnem mestu. Tudi tu bomo opisali najnujnejše atribute. Identifikator *ID_ORD_COMP* je hkrati tudi primarni ključ tabele. Identifikator ID_ORD_OPER je tuji ključ, vezan na tabelo »OrderOper«. Atribut *COMP_NR* predstavlja številko materiala, atribut *COMP_DESC* pa njegovo tekstovno ime. Sledi atribut *QUANTITY*, ki pove, koliko enot materiala bomo potrebovali na neki operaciji. Pomemben je še atribut *REF_POINT*, ki sporoči, ali je material vključen v materialno sledljivost ali ne.

Tabela OrderRecipe

V tabeli »OrderRecipe« so shranjeni globalni recepti, vezani na posamezno operacijo. Identifikator *ID_ORD_REC* je hkrati tudi primarni ključ tabele. Identifikator *ID_ORD_OPER* je tuji ključ, vezan na tabelo *»OrderOper«*. Sledi atribut *RECIPE_DESC*, ki je opis recepta, in atribut *RECIPE_CHANGE_NR*, ki je številka spremembe recepta. Naslednji atribut *RECIPE_MIME* določa format dokumenta, v katerem je recept shranjen. Sledi atribut *RECIPE_CONT*, v katerem je shranjena vsebina recepta. Nazadnje imamo še atribut *REC_STAT*, ki nam pove, ali je recept aktiven ali ne.

5.4 UPORABLJENA STROJNA OPREMA

Za lokalni strežnik na proizvodni liniji potrebujemo navaden osebni računalnik z operacijskim sistemom Windows z vsaj 8 GB pomnilnika ter dva trda diska, ki ju postavimo v polje RAID 1. Odločili smo se za navaden osebni računalnik, saj je cenejši od industrijskih izvedenk. Navaden osebni računalnik lahko v primeru okvare hitreje zamenjamo, saj imajo industrijski računalniki lahko dobavni rok tudi več mesecev, medtem ko navadni osebni računalnik lahko dobimo v nekaj tednih. Nadomestni osebni računalnik imamo lahko zaradi nižje cene na zalogi in se tako povsem izognemo dobavnim rokom. Z uporabo RAID polja smo nekoliko zaščiteni pred nenadno odpovedjo enega od trdih diskov. Za večjo varnost se sliko trdega diska redno varnostno kopira na oddaljen strežnik. Lokalni strežnik mora imeti tudi tri fizično ločene mrežne kartice. Prva se uporablja za dostop do poslovnega omrežja, druga za dostop do vseh odjemalcev ter optičnih bralnikov črtnih kod ter tretja za dostop do krmilnikov PLC. Priporočljivo je tudi, da imamo lokalni strežnik priključen na UPS v primeru nenadnega izpada električne energije. Na vsakem delovnem mestu se za odjemalca uporablja Raspberry Pi 3, nanj pa je priklopljen zaslon, občutljiv na dotik. Raspberry Pi je uporabljen predvsem zaradi lažjega upravičevanja stroškov strojne opreme, zmogljivostno pa zadošča trenutnim potrebam. Dobra lastnost uporabe Raspberry Pija je v tem, da ga lahko zelo hitro zamenjaš v primeru okvare. Tudi če pride do okvare pomnilnika, v tem primeru SD-kartice, jo lahko enostavno zamenjamo in proizvodnja lahko nemoteno teče naprej. Zaslon, občutljiv na dotik, uporablja optično tehnologijo zaznave dotika. To je pomembno, saj mora zaslon zaznati dotik, tudi če uporabnik uporablja rokavice. Tudi uporaba teh zaslonov na dotik je pogojena z lažjim upravičevanjem stroškov strojne opreme, saj so ti zasloni veliko cenejši od klasičnih zaslonov na dotik. Nazadnje so tu še termični tiskalniki za nalepke ter optični bralniki kod. Optične bralnike kod potrebujemo na delovnem mestu pakiranja ter za preverjanje ustreznega materiala na delovnih mestih.

5.5 POTEK IZVAJANJA APLIKACIJE NA SPLOŠNEM DELOVNEM MESTU

Potek izvajanja aplikacije na delovnem mestu bomo predstavili z diagramom aktivnosti na Sliki 13. Kot smo že omenili ima vsako delovno mesto na proizvodni liniji svoje specifike, vendar pa je zaradi poenostavitve opisan najosnovnejši potek izvajanja, ki je uporabljen na vseh delovnih mestih.

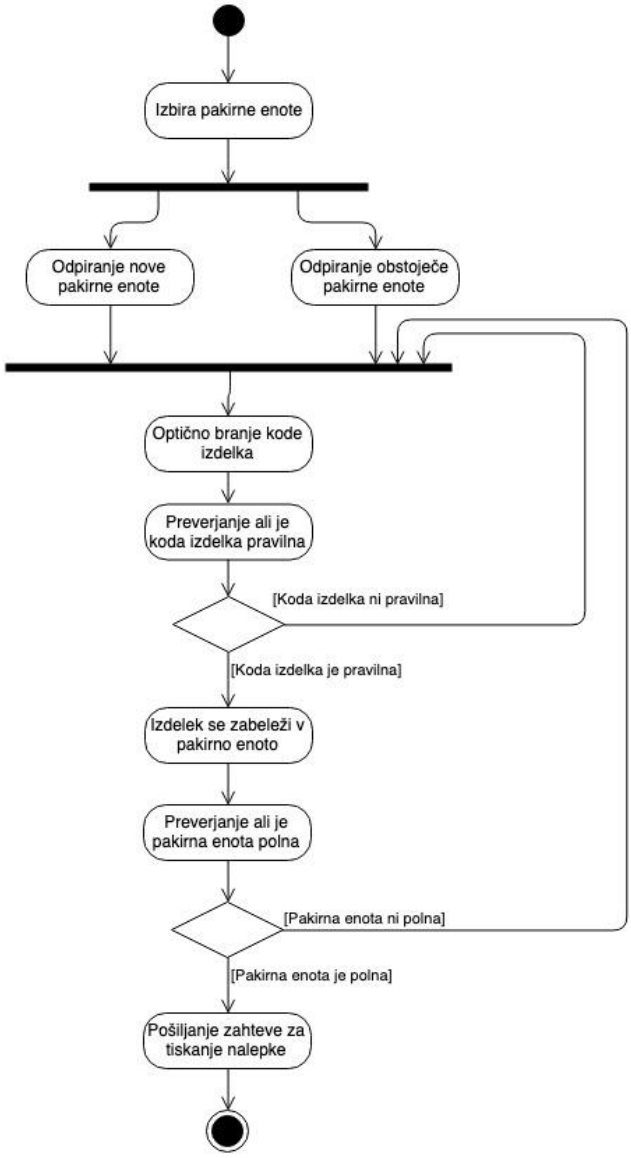
Ko prispe izdelek na delovno mesto, PLC-krmilnik aplikaciji sporoči serijsko številko izdelka, ki jo pridobi preko optičnega bralnika črtnih kod, povezanega neposredno na krmilnik PLC. Ta nato pošlje zahtevo za obdelavo izdelka in aplikacija s poizvedbo v podatkovni bazi preveri, ali je izdelek namenjen za to delovno mesto. Če izdelek ni namenjen za to delovno mesto, aplikacija PLC-krmilniku odgovori na zahtevo in sekvenca se zaključi. Če pa je izdelek namenjen za trenutno delovno mesto, se v naslednjem koraku primerja kodo trenutnega izdelka s kodo izdelka, ki se je prej obdeloval na tem delovnem mestu. Koda izdelka predstavlja tip izdelka, ki se trenutno izdeluje, in je različna od unikatne serijske številke, s katero je opremljen vsak izdelek. Če se ti dve kodi izdelka razlikujeta, aplikacija začne s pridobivanjem globalnega recepta. V kolikor pride do napake v prenosu globalnega recepta ali pa globalni recept ne obstaja, ima uporabnik možnost prenosa lokalnega recepta. Če pa se kodi izdelka ujemata, ni potrebe po ponovnem nalaganju receptov, zato se sekvenca lahko nadaljuje brez prenosa recepta. Pogoj za nadaljevanje na naslednji korak je torej, da ima PLC-krmilnik pravilni recept za delovanje. Nato aplikacija preveri, če je na delovnem mestu prisoten ustrezen material. Če te ustreznosti ni, aplikacija od uporabnika zahteva, naj optično prebere kode pravilnih materialov. Ko je na delovnem mestu ves material aktiven, se sekvenca lahko nadaljuje. Naredi se zapis v tabeli »TraceDataHead«, da je izdelek prispel na delovno mesto.

V naslednjem koraku aplikacija PLC-krmilniku odgovori na zahtevo ter mu pošlje dovoljenje za delo na tem izdelku. Po končani operaciji na izdelku PLC-krmilnik poda aplikaciji zahtevo za shranjevanje procesnih podatkov. Aplikacija nato shrani procesne podatke v tabelo »TraceProductParam« ter posodobi prej narejen zapis v tabeli »TraceDataHead« s kvaliteto izdelka ter naslednjim delovnim mestom. Naslednje delovno mesto se pridobi preko delovnega naloga. Nato aplikacija PLC-krmilniku odgovori na zahtevo in sekvenca se zaključi. Aplikacija je pripravljena na sprejem novega izdelka.

Slika 13: Diagram aktivnosti za splošno delovno mesto.

5.6 POTEK IZVAJANJA APLIKACIJE NA DELOVNEM MESTU PAKIRANJA IZDELKOV

Potek izvajanja aplikacije na delovnem mestu pakiranja bomo prav tako predstavili z diagramom aktivnosti, ki ga vidimo na Sliki 14.



Slika 14: Diagram aktivnosti za delovno mesto pakiranja izdelkov.

Na delovnem mestu pakiranja mora uporabnik najprej odpreti pakirno enoto. Lahko odpre novo pakirno enoto ali pa že obstoječo. Ko je pakirna enota odprta, uporabniku dovolimo optično branje kode novega izdelka. Nato aplikacija preveri prebrano kodo, ali ustreza vsem zahtevam. Če jim ne ustreza, se uporabniku izpiše opozorilo, da prebrana koda ni dovoljena za pakiranje v to pakirno enoto. V nasprotnem primeru pa se koda izdelka zabeleži v trenutno odprto pakirno enoto in se prikaže na vizualizaciji pakirne enote. Uporabnik se lahko tudi odloči za brisanje izdelkov iz pakirne enote. Ko uporabnik odpre pogled za brisanje izdelka, mora optično prebrati kodo izdelka in izdelek je izbrisan iz pakirne enote. Uporabnik lahko sproži zahtevo za tiskanje nalepke pakirne enote kadarkoli želi zaključiti trenutno odprto pakirno enoto. Sekvenca se zaključi, ko uporabnik zaključi obstoječo pakirno enoto in odpre novo.

6 IZVEDBA

Faza izvedbe sledi fazi načrtovanja sistema ter komponent. V tej fazi se na podlagi upoštevanja vseh prejšnjih faz razvije končen sistem. V tej fazi navadno odkrijemo pomanjkljivosti, če bi se nam pojavile v kateri od predhodnih faz. V kolikor so vse prejšnje faze korektno in temeljito izpeljane s samo izvedbo oziroma razvojem sistem ne bi smeli imeti večjih težav. V tem poglavju bomo predstavili potek izvedbe našega sistema.

Izvedbo našega sistema smo začeli s konfiguracijo strežnika sistema. Najprej je treba vzpostaviti podatkovno bazo in pripraviti podatke v določenih tabelah, nato pa konfigurirati OPC-strežnik. V program KEPServerEx moramo uvoziti podatkovne bloke oziroma značke iz PLC krmilnika. Te značke bodo nato na voljo OPC-klientu za branje ali pisanje. Nato sledi konfiguracija Apache strežnika za gostovanje aplikacije našega klienta. Pri nastavitvah Apache strežnika moramo biti posebej pozorni na posredovanje HTTP-klicev, ki se začnejo na ”/api”. To so HTTP-klici, namenjeni na našo strežniško aplikacijo. Po uspešni namestitvi in konfiguraciji naše programske opreme na strežniku začnemo z implementacijo strežniškega dela našega sistema.

Zaradi funkcijskih zahtev opisane proizvodne linije smo morali razviti sistem za upravljanje z recepti, saj tak sistem še ni bil v uporabi na nobeni proizvodni liniji. Vsak recept je prilagojen tipu izdelka, ki se izdeluje na proizvodni liniji ter delovnem mestu. Recepti se v naš sistem prenesejo iz poslovnega informacijskega sistema in se shranijo v lokalno podatkovno bazo v binarnem zapisu. Na poslovni informacijski sistem so recepti naloženi v JSON-formatu, torej ima vsak zapis ključ in vrednost, kjer ključ predstavlja ime značke na PLC-krmilniku, vrednost pa predstavlja predpisano vrednost te značke. Format JSON podpira tudi objekte, torej se moramo po JSON-datoteki sprehoditi rekurzivno, da dosežemo vse vrednosti. Za razčlenitev JSON-datoteke si v javi pomagamo s knjižnico »jackson« in vsako dobljeno vrednost zabeležimo v začasni seznam. Ko rekurzivno obdelamo celotno JSON-datoteko, ta seznam vrednosti z enim klicem funkcije zapišemo na PLC-krmilnik. V kolikor bi vsako posamezno dobljeno vrednost iz JSON-datoteke takoj nanj zapisali, bi za zapis vrednosti rabili več klicev funkcije, kar je pa veliko počasneje. Sistem mora tudi omogočati grajenje recepta v obratni smeri, torej s strani PLC-krmilnika v podatkovno bazo. Ko sistem zazna prožilec za shranitev recepta v podatkovno bazo, se moramo sprehoditi po celotni strukturi podatkovnega bloka v PLC-krmilniku in prebrati vrednosti vseh značk v podatkovnem bloku. Značke se v podatkovnem bloku nahajajo na več nivojih, torej je pomembno, da tako drevesno ureditev obdržimo tudi pri gradnji JSON-datoteke. Za vsako posamezno delovno mesto imamo v podatkovni bazi shranjen tudi metarecept v formatu JSON, ki vsebuje ključe, ki so usklajeni z imeni značk na PLC-krmilniku ter njihove podatkovne tipe. Metarecept v podatkovni bazi tako določa strukturo končnega recepta v JSON-formatu. Slabost uporabe metarecepta je v tem, da moramo v

primeru dodajanja nove podatkovne značke v PLC-krmilnik na novo zgraditi tudi metarecept in ga shraniti v podatkovno bazo, da obdržimo pravilno strukturo recepta.
Po uspešni implementaciji vseh potrebnih funkcij na strežniški strani aplikacije sledi konfiguracija aplikacije s podatki za določeno podatkovno linijo. Eden od ciljev razvoja sistema za sledljivost izdelkov je imeti enotno strežniško stran sistema ne glede na to, na katero proizvodno linijo želimo vpeljati sistem. Edini dve datoteki, ki se razlikujeta po proizvodnih linijah, sta »application.yml« ter »imeProizvodneLinijeInt-context.xml«. Datoteka »application.yml« je konfiguracijska datoteka v YAML-formatu, v katerem določimo povezave na vse podatkovne baze ter določimo profil izvajanja Spring aplikacije. Primer konfiguracijske datoteke vidimo na Sliki 15.

```
spring:
profiles: prod
datasource:
db-local:
            driverClassName: net.sourceforge.jtds.jdbc.Driver
            url: jdbc:jtds:sqlserver://127.0.0.1:1433/imeLokalneBaze
            validationQuery: select 1

      db-global:
            driverClassName: oracle.jdbc.OracleDriver
            url: jdbc:oracle:thin:@//192.168.1.1:1521/imePoslovneBaze
            validationQuery: select 1 from dual
```

Slika 15: Primer konfiguracijske datoteke »application.yml«.

V datoteki »imeProizvodneLinijeInt-context.xml« pa konfiguriramo Spring aplikacijo s pomočjo XML-sheme. V tej datoteki določimo OPC-strežnik, na katerega se strežniška aplikacija poveže, ter določimo povezave na optične bralnike črtnih kod. Primer konfiguracijske datoteke vidimo na Sliki 16. Na sliki je prikaz povezave na optične bralnike črtnih kod, za katere potrebujemo IP-naslov bralnika črtnih kod ter podatek, na kateri dogodek se proži klic funkcije »processBarcode«. V tem primeru konfiguracije je prožilec znak za pomik v novo vrstico (angl. line feed). Za vsako delovno mesto moramo definirati, kje se nahaja OPC-strežnik. Na ta način na strežniški strani odpremo OPC-klienta za vsako posamezno delovno mesto. Vsako delovno mesto mora imeti definiranega lastnega OPC-klienta, saj želimo, da delovna mesta delujejo neodvisno ena od druge. Po uspešni konfiguraciji teh dveh datotek je strežniška aplikacija pripravljena za delovanje. Treba je samo še zgraditi aplikacijo v ».jar« datoteki.
Po uspešno zgrajeni strežniški aplikaciji nadaljujemo z razvojem aplikacije odjemalca. Pri začetni postavitvi projekta delo zelo olajša orodje Angular CLI, ki smo ga podrobneje opisali v poglavju 5.3.3. Po začetni postavitvi ogrodja aplikacije je treba najprej konfigurirati povezavo z našo strežniško aplikacijo. V začetni mapi našega projekta moramo ustvariti datoteko z imenom »proxy.conf.json«, ki je konfiguracijska datoteka v JSON-formatu.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xsi:schemaLocation="http://www.springframework.org/schema/beans">

    <!-- Client - Pack - Barcode readers -->
    <bean   id="lfSerializer"   class="org.springframework.integration.ip.tcp.serializer.
ByteArrayLfSerializer"/>

    <int-ip:tcp-connection-factory id="tcfPackBR"
        type="client"
        host="192.168.1.1"
        port="5310"
        single-use="false"
        deserializer="lfSerializer"/>

    <int:channel id="inChannelPackBR"/>

    <int-ip:tcp-inbound-channel-adapter id="Pakiranje"
        connection-factory="tcfPackBR"
        channel="inChannelPackBR"
        client-mode="true"/>

    <int:object-to-string-transformer
        input-channel="inChannelPackBR"
        output-channel="outChannelPackBarcodeStr"/>

    <bean id="packagingService" class="com.app.service.PackagingService"/>

    <int:service-activator id="processPackBarcode"
        input-channel="outChannelPackBarcodeStr"
        ref="packagingService"
        method="processBarcode"/>

    <!-- OPC servers -->
    <bean id="LinijaOPCServer1" class="com.app.service.OPCService">
        <property name="opcServer" value="LinijaOPCServer1"/>
        <property name="host" value="localhost"/>
        <property name="domain" value=""/>
        <property name="user" value="admin"/>
        <property name="password" value="admin"/>
        <property name="progId" value="Kepware.KEPServerEX.V6"/>
        <property name="clsid" value="7BC0CC8E-482C-47CA-ABDC-0FE7F9C6E729"/>
        <property name="pollPeriod" value="100"/>
    </bean>
</beans>
```

Slika 16: Primer konfiguracijske datoteke »imeProizvodneLinijeInt-context.xml«.

V tej datoteki določimo, da Angular aplikacija posreduje vse HTTP-zahteve, ki se začnejo na »/api« na naslov »http://localhost:8090«, na katerem gostuje naša strežniška aplikacija. To sicer velja samo, če Angular aplikacijo gostujemo s pomočjo orodja Angular CLI. V produkcijskem načinu za posredovanje teh zahtev skrbi spletni strežnik Apache. Vse zahteve, ki jih preko WebSocketa strežniška aplikacija posreduje aplikaciji odjemalca, se začnejo na »/topic/wcdisplay«. V Angular aplikaciji se naročimo (angl. subscribe) na vse zahteve, ki pridejo s tega naslova. Če je na primer PLC-značka na strežniški strani aplikacije določena, da spremljamo vsako spremembo njene vrednosti, bo vsakič poslala

sporočilo na WebSocket, ki ga bo nato Angular aplikacija prestregla in obdelala. Tako je naša Angular aplikacija povezana z našo strežniško aplikacijo. V začetni mapi projekta se nahaja tudi datoteka »package.json«, v kateri se nahajajo vse knjižnice in odvisnosti (angl. dependencies), ki jih Angular aplikacija potrebuje za delovanje. Nato začnemo z razvojem storitev v Angular aplikaciji. Storitve vsebujejo funkcije, ki jih uporabljamo za pošiljanje standardnih HTTP-zahtev (GET, POST) strežniški aplikaciji. S klicem storitve torej sprožimo klic funkcije na strežniški aplikaciji in se sočasno tudi naročimo na morebiten odgovor funkcije. Ker je strežniška aplikacija enaka na vseh proizvodnih linijah, so lahko tudi storitve v Angular aplikaciji enake za vse proizvodne linije, kar nam zelo pospeši razvoj aplikacije na novih proizvodnih linijah. Po uspešni implementaciji storitev nadaljujemo z razvojem komponent Angular aplikacije, ki so podrobneje opisane v poglavju 5.3.3. Vsako delovno mesto na proizvodni liniji ima svojo komponento v Angular aplikaciji. Vse komponente je potrebno vključiti v datoteko »app.module.ts«, ki združuje vse uporabljene komponente, storitve in module. V datoteko »app-routing.module.ts« pa moramo vključiti tiste komponente, do katerih želimo dostopati neposredno prek URL-naslova. V našem primeru so to komponente za vsako posamezno delovno mesto, ki ji določimo enoličen URL-naslov npr. http://localhost/app/wcdisplay10. Komponento v Angular aplikaciji označimo z dekoratorjem (angl. decorator) »@Component«, ki vsebuje metapodatke o komponenti, kot vidimo na spodnjem primeru. Na primeru metapodatkov komponente na Sliki 17 vidimo:

- selector – določa ime, s katerim lahko to komponento vključimo v druge komponente. To storimo tako, da to ime postavimo med HTML-značke npr. <app-wcdisplay10></app-wcdisplay10>,
- templateUrl – določa, v kateri HMTL-datoteki se nahaja uporabniški vmesnik te komponente,
- styleUrls – določa, v kateri CSS-datoteki se nahaja oblikovanje našega uporabniškega vmesnika,
- providers – vsebuje imena razredov vseh storitev, ki jih bomo potrebovali v komponenti.

```
@Component({
  selector: 'app-wcdisplay10',
  templateUrl: './wcdisplay10.component.html',
  styleUrls: ['./wcdisplay10.component.css'],
  providers: [
    MachCommService,
    TraceService,
    RecipeService
  ]
})
```

Slika 17: Primer meta podatkov Angular komponente.

V tem koraku začnemo z razvojem aplikacijske logike, pri čemer tesno sledimo korakom, ki so definirani v fazi načrtovanja sistema. Sočasno z razvojem aplikacijske logike razvijamo tudi uporabniški vmesnik sistema. Ker je sistem za sledljivost izdelkov razvit v sodelovanju s podjetjem, izvorne kode projekta ne smemo deliti.

7 TESTIRANJE

Po fazi razvoja aplikacije oziroma izvedbe sledi faza testiranja. Namen faze testiranja je odkritje napak v aplikaciji, preden aplikacijo predamo končnemu uporabniku. Pomembno je, da so testni primeri povezani z zahtevami, torej se moramo pri izvedbi faze testiranja opirati na fazo specifikacije zahtev. Testiranje lahko opravi sam razvijalec sistema ali pa neodvisni preizkuševalec, ki aplikacije še ne pozna. Glavni interes razvijalca sistema je pravočasna predaja aplikacije v uporabo, glavni interes neodvisnega preizkuševalca pa je zagotavljanje kakovosti aplikacije. Torej je testiranje učinkovitejše, če ga opravijo neodvisni preizkuševalci [10]. V našem primeru smo testiranje aplikacije opravili skupaj s programerji PLC-krmilnikov.

Testiranje našega sistema smo začeli postopno. Najprej smo se osredotočili na strežniški del in komunikacijo z lokalno podatkovno bazo. Testirali smo vse funkcije za branje in pisanje v lokalno podatkovno bazo. Nato smo strežniški del našega sistema povezali v omrežje in testirali povezavo do poslovne podatkovne baze. Testirali smo prepis podatkov iz poslovne v lokalno podatkovno bazo, saj mora naš sistem imeti vse potrebne podatke za delovanje proizvodne linije, shranjene v lokalni podatkovni bazi. Nato smo na strežniški del sistema povezali optične bralnike črtnih kod. Tu smo preverili, ali prebrana vsebina črtne kode prispe do funkcije za obdelavo prebranih črtnih kod. Potem smo v sistem povezali še en PLC-krmilnik in z njim vzpostavili osnovno povezavo za branje ter pisanje značk. Testirali smo, ali se naša strežniška aplikacija poveže z OPC-strežnikom in uspešno zapiše vrednost v značko na PLC-krmilniku. Nato smo testirali še branje te iste značke in primerjali zapisano ter prebrano vrednost. Na tem mestu je bila naša strežniška aplikacija pripravljena, treba je bilo samo še testirati povezavo do odjemalnega dela našega sistema. Na odjemalnem delu smo najprej pripravili testno aplikacijo prav z namenom testiranja povezave s strežniškim delom sistema. Najprej smo iz odjemalnega dela sistema sprožili HTTP-klic funkcije. Aplikacija na strežniškem delu je klic prejela in poslala odgovor nazaj odjemalcu. Potem smo testirali še WebSocket povezavo med strežniškim in odjemalnim delom. Za testiranje te povezave smo konfigurirali optični bralnik črtnih kod, da prebrano vrednost črtne kode pošlje na »/topic/wcdisplay«. V testni aplikaciji odjemalca smo se naročili na vse zahteve, ki pridejo s tega naslova. Nato smo prebrali črtno kodo in prebrana vrednost je bila vidna v aplikaciji na strani odjemalca. Po uspešno vzpostavljeni komunikaciji med strežnikom in odjemalcem našega sistema smo začeli še s testiranjem aplikacije odjemalca. To je podrobneje napisano v nadaljevanju tega poglavja.

Zaradi časovne stiske pri izvedbi tega projekta se testiranje ni izvajalo med točno določenimi testnimi primeri ali pa z avtomatskim testiranjem. Najprej se je testiralo posamezne funkcionalnosti sistema, sistem kot celoto pa smo večinoma testirali med testnim obratovanjem proizvodne linije. Prav tako nismo dokumentirali poteka testiranja.

Kako velik pomen ima temeljito in načrtno testiranje, je bilo predstavljeno vodstvu, vendar pa se zaradi časovne stiske tega vseeno ni izvedlo.

7.1 STRUKTURNO TESTIRANJE

Cilj strukturnega testiranja je zagotoviti, da je vsak pogoj in vsak del izvorne kode uporabljen vsaj enkrat. Strukturno testiranje (angl. white-box testing) temelji na poznavanju notranje strukture programa [10]. Pri tej vrsti testiranja je pomembno poznavanje celotne strukture programa, torej je smiselno, da pri teh testih sodeluje razvijalec programa.
Strukturno testiranje je bilo opravljeno še preden smo vzpostavili komunikacijo med našim sistemom ter PLC-krmilniki, torej smo to komunikacijo morali simulirati. Najprej smo testirali osnovne poti programa, nato pa vse pogoje, ki jih je v naši aplikaciji veliko. Testiranje pogojev smo na strežniškem delu opravili s pomočjo razhroščevalnika, ki je vgrajen v razvijalsko okolje STS (angl. Spring Tool Suite). To okolje temelji na bolj znanem razvijalskem okolju Eclipse. Na strani odjemalca pa smo aplikacijo testirali v razvijalskem načinu spletnega brskalnika Google Chrome. Razvijalski način ima tudi možnost razhroščevanja, ki nam je bila med fazo testiranja zelo v pomoč. Testiranje na strežniški strani se ni bistveno razlikovalo od testiranja na odjemalni strani. V obeh primerih smo na mesta pogojev postavili prekinitvene točke (angl. breakpoints) in nato preverili pogoje. Po testiranju pogojev smo izvedli še testiranje zank. Tu smo se predvsem osredotočili na to, da se vse naše zanke zaključijo. Po uspešno zaključenem strukturnem testiranju nato nadaljujemo z vedenjskim testiranjem, ki je opisano v naslednjem podpoglavju.

7.2 VEDENJSKO TESTIRANJE

Pri vedenjskem testiranju (angl. black-box testing) ne poznamo notranjega delovanja programa. To testiranje torej temelji na zunanjih lastnostih sistema [10]. Pri vedenjskem testiranju se torej ne osredotočamo na kontrolni tok podatkov, temveč na same informacije. Cilj takega testiranja je zagotavljanje pravilnega delovanja sistema, ne glede na to, kakšne vhodne podatke pošljemo v sistem.
Najprej smo testirali izvajanje glavne sekvence našega programa, ki je v veliki meri odvisna od komunikacije s PLC-krmilniki. Zato je bil pogoj za opravljanje tega testiranja vzpostavljena komunikacija s PLC-krmilniki, saj ravno ti skrbijo za vhodne podatke našega sistema. Pri tem testiranju so bili prisotni tudi PLC-programerji, ki so v vlogi neodvisnega preizkuševalca. Ker sta strežniški in odjemalni del medsebojno odvisna, oba dela smatramo kot enoten sistem oziroma kot enotno »črno škatlo«.
Ker je bilo testiranje našega sistema usklajeno s testiranjem sekvence izvajanja programa na PLC-krmilniku, smo najprej izvedli testiranje v laboratoriju z enim PLC-krmilnikom, ki

je bil povezan v naš sistem. Tako smo lahko testirali samo komunikacijo s PLC-krmilnikom ter tudi možnosti različnih vhodnih podatkov v naš sistem. To testiranje je naš sistem uspešno prestal, saj se je odzival samo na pravilne vhodne podatke, napačne pa je ignoriral. Sočasno smo tudi testirali komunikacijo našega sistema s podatkovno bazo. Tudi tu ni bilo problemov, saj je sistem od zahteve za shranjevanje podatkov te podatke uspešno zabeležil v pravilno tabelo v podatkovni bazi. Po uspešno opravljenem laboratorijskem testiranju smo testiranje nadaljevali na enem delovnem mestu na proizvodni liniji.

Na delovnem mestu smo najprej še enkrat testirali komunikacijo našega sistema s PLC-krmilnikom. Po uspešno vzpostavljeni povezavi smo začeli s testiranjem upravljanja z recepti, saj tega dela našega sistema v laboratoriju nismo testirali. Najprej smo testirali pošiljanje receptov s PLC-krmilnika v naš sistem. Recept smo v naš sistem uspešno shranili, vendar pa je izvajanje trajalo občutno preveč časa. Težavo smo kasneje odkrili v načinu branja podatkov s PLC-krmilnika. Po predelavi funkcije za shranjevanje receptov smo čas izvajanja te funkcije zmanjšali s približno 50 sekund na manj kot sekundo. Potem ko smo recepte uspešno shranili v naš sistem, smo lahko preverili delovanju gumbov »Pridobi recepte« in »Prenesi recept na PLC« na našem uporabniškem vmesniku. Ugotovili smo, da gumba delujeta, kot je bilo predvideno. Nato smo testirali pošiljanje receptov v obratni smeri, torej iz našega sistema do PLC-krmilnika. Tu je predvsem pomembno, da se vsebina recepta zapiše na pravilno mesto v PLC-krmilniku. V tej smeri kakršnih koli težav nismo opazili. Nato smo na delovnem mestu ponovili še laboratorijske teste. Po uspešno prestanem testu na enem delovnem mestu smo potem iste teste ponovili še na vsakemu izmed petnajstih delovnih mest, ki sestavljajo proizvodno linijo. Ko smo se prepričali, da naš sistem deluje na vsakem posameznem delovnem mestu, smo v naš sistem skupaj povezali vsa delovna mesta.

Na tem mestu smo lahko začeli s preizkušanjem našega sistema kot celote, saj so bile v sistem povezane vse odjemalne naprave ter vsi PLC-krmilniki. Testiranje se je začelo z izdelavo izdelka na prvem delovnem mestu in nato nadaljevalo po posameznih delovnih mestih do zadnjega, to je delovno mesto pakiranja izdelkov. V primerjavi s testiranjem na posameznih delovnih mestih smo opazili, da so se odzivni časi našega sistema konkretno povečali. Primerjava odzivnih časov je vidna v Tabeli 1. Ker je odzivnost sistema ena izmed glavnih nefunkcijskih zahtev, smo ji morali na podlagi rezultatov iz testiranj posvetiti veliko časa. Kako je podrobneje potekala optimizacija sistema, je opisano v poglavju 7.3.2. Z vedenjskim testiranjem smo torej opazili, da sistem deluje pravilno in zanesljivo, vendar pa so potrebne optimizacije glede odzivnosti sistema.

7.3 SISTEMSKO TESTIRANJE

Namen sistemskega testiranja je ugotavljanje obnašanja programskega produkta v prisotnosti drugih sistemskih elementov, kot so strojna oprema in uporabniki sistema [10]. Sistemsko testiranje se je v nekaterih primerih prekrivalo z vedenjskim testiranjem, saj je

delovanje našega sistema odvisno od delovanja PLC-krmilnikov. Smo pa v tem koraku testirali oživljanje ter zmogljivost našega sistema ob polni obremenitvi proizvodne linije.

7.3.1 Testiranje oživljanja

Zagotoviti moramo, da naš sistem po prisotnosti napake normalno obratuje naprej. Pri testiranju oživljanja smo se osredotočili na ponovno vzpostavitev sistema v primeru izpada električne energije. Strežnik našega sistema je opremljen z UPS-zaščito (angl. Uninterruptible Power Supply), kar omogoča varen izklop sistema v primeru izpada električne energije. Sistem se po povratku električne energije samodejno vzpostavi, saj se vsa potrebna programska oprema samodejno zažene zagonu operacijskega sistema.

7.3.2 Testiranje zmogljivosti

V nefunkcijskih zahtevah imamo kot zahtevo navedeno tudi odzivnost našega sistema. Ta točka je poleg zanesljivosti najpomembnejša točka med nefunkcijskimi zahtevami, saj ima proizvodna linija kratek čas cikla in je za komunikacijo sistema s PLC-krmilniki rezervirana samo ena sekunda. Med vedenjskim testiranjem smo opazili, da se je odzivnost našega sistema poslabšala potem, ko smo v naš sistem povezali vse PLC-krmilnike. Torej je bilo treba izvesti temeljito testiranje zmogljivosti našega sistema. Odzivnost našega sistema smo merili za začetku operacije. Z merjenjem smo začeli, ko je PLC-krmilnik podal zahtevo za obdelovanje izdelka, končali pa, ko je naš sistem odgovoril krmilniku z dovoljenjem za delo. Najprej smo ta čas izmerili na enem delovnem mestu, ne da bi bili v sistem povezani ostali PLC-krmilniki. Ta čas nam bo služil za referenco. Nato smo v sistem povezali vse PLC-krmilnike in izmerili odzivni čas našega sistema na enem delovnem mestu. Meritve smo izvajali na strežniškem delu. Rezultate meritev so predstavljeni v Tabeli 1. Kot vidimo, se je odzivni čas našega sistema v povprečju povečal za več kot desetkrat, kar pa seveda ni sprejemljivo. Izmerjeni časi več kot enkrat večji od časa, ki je predviden in rezerviran za komunikacijo sistema s PLC-krmilniki.

Tabela 1: Meritve odzivnega časa sistema ob prihodu novega izdelka.

	1. meritev	**2. meritev**	**3.meritev**	**Povprečje**
Eno delovno mesto	193 ms	189 ms	196 ms	192,67 ms
Vsa delovna mesta	2031 ms	2043 ms	2038 ms	2037,33 ms

Nato smo začeli z analizo poteka našega programa. Posamezno smo izmerili čase, ki jih potrebujemo za dostop do podatkovne baze, ter čase, ki jih potrebujemo za komunikacijo s PLC-krmilniki. Opazili smo, da je dostop do podatkovne baze časovno zanemarljiv v primerjavi s komunikacijo s PLC-krmilniki. Težavo smo nato odkrili v konfiguraciji

našega OPC-strežnika KEPServerEx. Z uporabo privzetih nastavitev OPC-strežnik kontinuirano na 100 ms preverja, ali se je spremenila vrednost vsem značkam s PLC-krmilnikov. Vseh značk s PLC krmilnikov na delovnih mestih pa je 7922. To ustvari na mrežni povezavi zelo veliko prometa in zato je prišlo do bistvenega povečanja odzivnega časa. Nato smo spremenili konfiguracijo OPC-strežnika, da kontinuirano preverja vrednost samo ene značke na posameznem PLC-krmilniku, vrednosti vseh ostalih značk pa se preberejo na zahtevo OPC-klienta. Vrednost v tej znački nam pove na primer, da je izdelek prispel na delovno mesto in je pripravljen za obdelovanje. Po tej spremembi konfiguracije OPC-strežnika smo še enkrat izvedli meritve. Tokrat smo meritve izvedli samo, ko so v sistem priključeni vsi PLC-krmilniki. Rezultati meritev so vidni v Tabeli 2.

Tabela 2: Meritve odzivnega časa sistema po optimizaciji.

	1. meritev	**2. meritev**	**3.meritev**	**Povprečje**
Vsa delovna mesta	264 ms	272 ms	269 ms	268,33 ms

Na podlagi rezultatov lahko razberemo, da smo odzivni čas našega sistema bistveno zmanjšali. Vidimo tudi, da se odzivni čas našega sistema ni bistveno povečal glede na kontrolno meritev z enim delovnim mestom. Z doseženimi časi smo tudi ugodili nefunkcijski zahtevi glede odzivnosti našega sistema.

8 OBRATOVANJE

Po obsežnem testiranju sledi predaja sistema v obratovanje. Vseh napak med testiranjem ne moremo zajeti, zato smo pred rednim obratovanjem proizvodne linije zagnali poskusno obratovanje. Med poskusnim obratovanjem smo delovanje našega sistema pozorno spremljali ter manjše napake sproti popravili. Pomembno je poudariti, da vse popravke sproti beležimo s pomočjo sistema za dokumentiranje sprememb SVN (angl. subverison). Tako lahko naš sistem v primeru slabega popravka hitro vrnemo na prejšnjo verzijo in sistem deluje naprej. V fazi poskusnega obratovanja smo tudi izvedli izobraževanje končnih uporabnikov sistema, ki je zajemalo normalno delovanje sistema, kaj storiti v primeru napake na sistemu in kako ponovno vzpostaviti sistem v primeru izpada električne energije. Izobraževanje končnih uporabnikov sistema je v tej fazi ključnega pomena, saj se mora uporabnik spoznati z vsemi funkcijami sistema. Pri tem nam pomaga tudi dokumentacija sistema, ki je trenutno še v fazi izdelave. Med poskusnim obratovanjem smo lahko tudi prvič preizkusili integracijo našega sistema v sistem za spremljanje proizvodnje ter v poslovni informacijski sistem. Za izvajanje proizvodnje izdelka mora biti razpisan delovni nalog, na katerega je vezana tudi dobava materialov na proizvodnjo linijo, zato smo integracijo z drugimi informacijskimi sistemi lahko testirali šele v tej fazi. Po nekaj tednih poskusnega obratovanja smo bili z delovanjem sistema zadovoljni in tako smo lahko pričeli z rednim obratovanjem proizvodne linije.

Omenjena proizvodna linija, na katero smo implementirali sistem za spremljanje proizvodnje, redno obratuje od septembra 2018. V tem času smo na proizvodno linijo implementirali še nekatere dodatne zahteve ter popravili manjše programske napake, ki so se pojavile med rednim obratovanjem. S strojno opremo našega sistema zaenkrat še ni bilo težav. Po petih mesecih rednega obratovanja pa se je pojavila večja težava, in sicer podaljšani časi komunikacije med našim sistemom ter podatkovno bazo. Ta težava je potem tudi podaljševala cikel proizvodne linije, kar pa ni sprejemljivo. Nato smo s pomočjo orodja SQL Server Management Studio naredili analizo vseh poizvedb v podatkovni bazi in ugotovili, da se dve poizvedbi v tabelo »TraceProductParam« izvajata počasneje zaradi velike količine podatkov v tej tabeli. Težavo smo nato odpravili s postavitvijo indeksov na določene stolpce v tabeli, ki se uporabljajo v dveh kritičnih poizvedbah.

9 ZAKLJUČEK

V magistrskem delu smo predstavili razvoj sistema za sledljivost izdelkov ter njegovo vpeljavo na določeno proizvodno linijo. Opisan je celoten proces razvoja izdelka od definicije problema do obratovanja, saj sistem deluje v rednem obratovanju proizvodne linije od septembra 2018. Med pisanjem tega dela smo spoznali pomembnost vsake od faz programskega inženirstva, saj le s temeljito izvedbo vsake faze lahko dostavimo dober sistem. Pomembno je, da imamo pred samim začetkom razvoja jasno definirane zahteve in cilje sistema, saj lahko dodatne, slabo definirane zahteve v naslednjih fazah ogrozijo razvoj celotnega sistema. Eden glavnih razlogov za razvoj tega sistema je bil, da bi imeli v podjetju prisoten enoten sistem za sledljivost izdelkov. Lahko potrdimo, da nam je to zaenkrat dobro uspelo, saj je opisani sistem za sledljivost izdelkov trenutno prisoten na osmih novih proizvodnih linijah. Vsaka proizvodna linija ima sicer svoje specifike, vendar pa strežniška stran sistema ostaja enaka na vseh proizvodnih linijah. Za podporo večjemu številu proizvodnih linij pa je prišlo do kadrovske podhranjenosti, zato se je v podjetju začelo iskati alternative, ki bi omogočale lažjo in hitrejšo implementacijo SCADA-sistema. Kot ena od alternativ se je pojavilo orodje Ignition, ne izključuje pa se niti zunanji dobavitelj SCADA-sistema. Kljub temu smo z razvojem tega sistema dosegli zastavljene cilje in tudi vse funkcijske kot tudi nefunkcijske zahteve.

Vseeno pa sistem ni popoln in ima še nekaj možnosti za izboljšave. Ena izmed izboljšav je optimiziranje pisanja podatkov v lokalno podatkovno bazo, saj je podatkovna baza v nekaterih primerih ozko grlo. Tu ne pričakujemo drastičnega izboljšanja odzivnosti sistema, vendar pa je pri SCADA-sistemu vsaka pridobljena stotinka sekunde dobrodošla, saj na ta način posledično skrajša čas cikla proizvodne linije. Naslednja izboljšava pa je bolj drastična, saj bi morali spremeniti arhitekturo celotnega sistema. Ideja je, da bi vso aplikacijsko logiko odjemalca preselili na strežniško stran, odjemalna stran pa bi bila namenjena prikazu dogodkov na delovnem mestu, prikazu dokumentacije in pregledu zgodovine delovnega mesta. Na ta način bi bil sistem neodvisen od delovanja odjemalca in bi proizvodna linija lahko nemoteno delovala tudi v primeru okvare odjemalne naprave. Pričakujemo tudi, da bi pridobili na zmogljivosti in odzivnosti sistema, saj bi na zahteve PLC-krmilnika odgovorili neposredno iz strežniške aplikacije, brez posredovanja odjemalcu.

Z razvojem tega sistema se je zagotovilo popolno sledenje izdelkom na proizvodni liniji in zvišanje kakovosti izdelkov. To za podjetje pomeni konkurenčno prednost in s tem lažje pridobivanje novih kupcev. S stališča razvoja sistema smo tudi zadovoljni, saj smo dosegli zastavljene cilje. Je pa prihodnost na tem področju zanimiva, saj se področje industrijskega interneta stvari zelo hitro razvija.

10 LITERATURA IN VIRI

[1] ”About OpenSCADA”. [Na spletu]. Dostopno na: http://oscada.org/wiki/About. [Datum ogleda: 21.4.2019]

[2] Advantech, ”The Advantech automation system pyramid”. [Na spletu]. Dostopno na: http://processengineering.co.uk/article/2017704/the-automation-syste. [Datum ogleda: 2.4.2019]

[3] ”Angular Architecture Overview”. [Na spletu]. Dostopno na: https://angular.io/guide/architecture. [Datum ogleda: 24.4.2019]

[4] C. Albert in C. Fuchs, Durchblick im Begriffsdschungel der Business-Software, Universität Würzburg, 2007.

[5] ”CLI Command Reference”. [Na spletu]. Dostopno na: https://angular.io/cli. [Datum ogleda: 27.4.2019]

[6] ”Difference between TypeScript and JavaScript”. [Na spletu]. Dostopno na: https://www.geeksforgeeks.org/difference-between-typescript-and-javascript/. [Datum ogleda: 27.4.2019]

[7] ”Introduction to services and dependency injection”. [Na spletu]. Dostopno na: https://angular.io/guide/ architecture-services. [Datum ogleda: 27.4.2019]

[8] ”Kepware KEPServerEX”. [Na spletu]. Dostopno na: https://www.kepware.com/en-us/products/kepserverex/. [Datum ogleda: 17.4.2019]

[9] N. Herakovič, Izzivi industrije 4.0, *Ventil* (2016), 10−16.

[10] P. Rogelj, Skripta za predmet programsko inženirstvo, Koper, 2014.

[11] ”Spletni strežniki”. [Na spletu]. Dostopno na: https://nsa-splet.si/splet/uvod/splet-uvod-03-server.php. [Datum ogleda: 30.6.2019]

[12] ”Spring Boot”. [Na spletu]. Dostopno na: https://spring.io/projects/spring-boot. [Datum ogleda: 18.4.2019]

[13] ”Spring Boot Starters”. [Na spletu]. Dostopno na: https://www.javatpoint.com/spring-boot-starters. [Datum ogleda: 18.4.2019]

[14] ”Spring Integration”. [Na spletu]. Dostopno na: https://spring.io/projects/spring-integration. [Datum ogleda: 18.4.2019]

[15] ”Študija izvedljivosti, analiza stroškov in koristi, poslovna tveganja”. [Na spletu]. Dostopno na: https://projektni-management.si/2011/01/22/studija-izvedljivosti-analiza-stroskov-in-koristi-poslovna-tveganja/. [Datum ogleda: 22.3.2019]

[16] ”The JTDS project”. [Na spletu]. Dostopno na: http://jtds.sourceforge.net/. [Datum ogleda: 20.4.2019]

[17] ”The WebSocket Protocol”. [Na spletu]. Dostopno na: http://www.rfc-editor.org/rfc/pdfrfc/rfc6455.txt.pdf. [Datum ogleda: 30.6.2019]

[18] ”Usage of web servers broken down by ranking”. [Na spletu]. Dostopno na: https://w3techs.com/technologies/cross/web_server/ranking. [Datum ogleda: 30.6.2019]

[19] ”What is OPC?”. [Na spletu]. Dostopno na: https://opcfoundation.org/about/what-is-opc/. [Datum ogleda: 17.4.2019]

[20] X. Hong in W. Jianhua, Using standard components in automation industry: A study on OPC Specification, Institute of Electrical Engineering, Xi'an JiaoTong University, 2005.