Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Štular Janez

Oblačno tiskanje

magistrsko delo

magistrski program druge stopnje
računalništvo in informatika

Mentor: prof. dr. Denis Trček

Ljubljana, 2019

Zahvala

Za pomoč, prejete nasvete in pregled dela se lepo zahvaljujem mentorju prof. dr. Denisu Trčku. Za korektno opravljeno lektoriranje pa se zahvaljujem univ. dipl. prev. Davorinu Lavriču.

Štular Janez, 2019

"The only reason for time is so that everything doesn't happen at once."

— Albert Einstein

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
OSHA	Occupational Safety and Health Administration	Agencija za varnost in zdravje pri delu
ERP	Enterprise resource planning	načrtovanje virov podjetja
SaaS	Software as a Service	programska oprema kot storitev
IoT	Internet of Things	internet stvari
API	Application programming interface	programski vmesnik
QR	Quick Response Code	hitro odzivna koda
TTM	Time To Market	čas do prihoda na tržišče
TCP	Transmission Control Protocol	nadzorni prenosni protokol
IP	Internet Protocol	internetni protokol
HTTP	Hypertext Transfer Protocol	protokol za prenos hiperteksta
HTTPS	Hypertext Transfer Protocol Secure	protokol za varni prenos hiperteksta
SOAP	Simple Object Access Protocol	protokol za dostop do enostavnih objektov
XML	Extensive Markup Language	razširljiv označevalni jezik
WSDL	Web Service Description Language	opisni jezik spletnih storitev
AAD	Azure Active Directory	aktivni imenik Azure

IETF	Internet Engineering Task Force	organizacija IETF
SPA	Single Page Application	aplikacija na eni strani
TLS	Transport Layer Security	varnost transportne plasti
WCF	Windows Communication Foundation	komunikacijski temelji Windows
DoS	Denial of Service	zavrnitev storitve
WS	Web Socket	spletni vtičnik
WSS	Web Socket Secure	varni spletni vtičnik
JSON	JavaScript Object Notation	opis objekta JavaScript
UUID	Universally Unique Identifier	univerzalni enolični identifikator
MVC	Model-View-Controller	model-pogled-nadzornik
MVVM	Model-View-ViewModel	model-pogled-pogled modela
MVP	Model-View-Presenter	model-pogled-predstavnik
HTML	Hyper Text Markup Language	jezik za označevanje hiperbesedila
WPF	Windows Presentation Foundation	predstavitveni temelji Windows
XAML	Extensible Application Markup Language	razširljiv aplikacijski označevalni jezik

Povzetek

Naslov: Oblačno tiskanje

Trendi sodobne industrije stremijo k hitremu razvoju in izpopolnjevanju izdelkov. Vse večkrat sta zahtevana njihova sledljivost in kontrola kakovosti. Etiketiranje podjetjem omogoča povezavo fizičnega sveta izdelkov z njihovimi informacijami, zapisanimi v digitalni obliki. Največkrat ne prinaša dodane vrednosti samih izdelkov, temveč služi le optimizaciji in podpori ostalih procesov.

Glavni cilj magistrske naloge je izboljšati proces tiskanja etiket obstoječega sistema z implementacijo oblačnih razširitev, ki omogočajo integracijo z drugimi sistemi ter možnost tiskanja neodvisno od uporabnikove lokacije. Obstoječi sistem za upravljanje z etiketami z implementacijo razširitev pridobi dodano vrednost, saj podjetjem omogoča integracijo procesa tiskanja z drugimi procesi in s tem njihovo optimizacijo.

Ključne besede

tiskanje, upravljanje, etikete, sistem, podjetja, uporabniki, nadzor dostopa, oblačne storitve, razširitve, integracije

Abstract

Title: Cloud Printing

Trends in modern industry strive to the rapid product development and their improvement. They often require traceability and quality assurance. Labelling supports the connection of physical products with the information saved in digital form. In most cases it does not bring additional value to products themselves, but it serves only for optimization and support to other existing processes.

The main goal of this master thesis is an improvement of the existing printing process with the implementation of two cloud extensions. First, it supports integration of the labelling system with other systems and, second, it supports location independent label printing. The existing label management system with the implementation of those extensions provides added value, because it supports the integration of the printing process with other processes and supports their optimization.

Keywords

printing, management, labels, system, companies, users, access control, cloud services, extensions, integrations

Poglavje 1

Uvod

Trendi sodobne industrije stremijo k hitremu razvoju in izpopolnjevanju izdelkov. Zahtevajo njihovo sledljivost in zadovoljivo kakovost izdelave, s čimer se pojavi potreba po njihovem etiketiranju. Etiketiranje podjetjem omogoča povezavo fizičnega sveta izdelkov in njihovih sestavnih delov z njihovimi digitalnimi informacijami. Največkrat ne prinaša dodane vrednosti samih izdelkov, temveč služi optimizaciji in podpori obstoječih procesov, zaradi česar mora biti časovno in finančno učinkovito, kot opisuje delo [1].

Etiketiranje izdelkov se nahaja v skoraj vseh korakih proizvodnje, predelave in prodaje izdelkov. Njegova implementacija lahko podjetjem v primeru učinkovitega delovanja prinese velike prihranke. Začne se lahko že z nadzorom dobavne verige izdelkov, ki z ustreznim načrtovanjem njihove dobave zmanjša stroške prevoza in skladiščenja. Proizvodnji in predelovalni industriji je načrtovanje omogočeno z nadzorom učinkovitosti proizvodnega oziroma predelovalnega procesa, prodajnim podjetjem pa z analizo prodaje in zahtev trga.

Prednosti učinkovite implementacije se s tem še ne končajo in segajo vse do načrtovanja prodajnih etiket. Zaradi raznolike lokalne regulative in neustreznosti obstoječih etiket je včasih potrebna ponovna označitev izdelkov. Predpisan izgled etiket pa je lahko zelo različen. Zahteva se lahko le opis izdelka v lokalnem jeziku ali pa je določen celoten načrt etiket. Slednji je

običajno zahtevan v strožje reguliranih proizvodnih panogah, katerih primer je regulativa Evropske agencije za varnost in zdravje pri delu OSHA (angl. Occupational Safety and Health Administration), ki predpisuje načrt etikete nevarnih snovi, opisan v dokumentu [2].

Cilj magistrske naloge je podjetjem omogočiti enostaven in učinkovit proces načrtovanja in tiskanja etiket z razširitvijo oblačnih storitev obstoječega sistema za upravljanje z etiketami.

1.1 Opis problema in rešitve

Težava večjih in razpršenih podjetij je uporaba različnih in zastarelih orodij za načrtovanje etiket. Izbrana orodja se včasih razlikujejo že med oddelki, zato enotnost med različnimi lokacijami podjetja ni pričakovana. Poleg tega se ustvarjene etikete največkrat hranijo lokalno na vsaki lokaciji. Posledično imajo podjetja težave z nadzorovanjem različnih različic ter vzdrževanjem in spreminjanjem etiket, kot je opisano v delu [3].

Uporaba sistema za upravljanje z etiketami jim omogoča enotnost, nadzor in hitro prilagodljivost etiket, shranjenih na enem mestu. Podjetjem je na voljo v obliki oblačnih storitev in vključuje celoten sklop programske opreme za načrtovanje in tiskanje etiket. Načrtovalcu omogoča shranjevanje etiket v skupno shrambo, kjer gredo le-te skozi postopek odobritve. Samo ustrezne in odobrene etikete nato postanejo na voljo za tiskanje, ki ga je z uporabo pripadajočega odjemalca mogoče izvajati iz oddaljenih lokacij.

1.2 Prispevki magistrskega dela

Zgoraj opisani sistem za upravljanje z etiketami že izpolnjuje večino zahtev večjih in razpršenih podjetij. Končni cilj magistrske naloge je razumevanje delovanja in arhitekture obstoječega sistema ter razširitev njegove povezljivosti.

Prvi cilj dela je analiza nadzora dostopa uporabnikov in dodelitve njiho-

vih pravic. Obstoječi sistem za preverjanje pristnosti uporabnikov uporablja protokol OAuth 2.0, ki bo razdelan in opisan. Osnovna avtorizacija uporabnikov uporablja pravice pripadajočih vlog dostopa. Poleg osnovne avtorizacije, ki je namenjena samemu dostopu do sistema, njuna kombinacija omogoča tudi dodelitev ločenih pravic dostopa do nekaterih vsebinskih virov.

Drugi cilj predstavlja razširitev oblačnih storitev obstoječega sistema z možnostjo povezave z drugimi tovrstnimi storitvami na enostaven, varen in nadzorovan način. S tem je velikim podjetjem omogočena integracija z različnimi zunanjimi sistemi, kot je ERP (angl. Enterprise Resource Planning) ponudnikov SAP in Oracle. Poleg tega je omogočena tudi integracija z drugimi storitvami in podpora za aktivacijo procesa tiskanja s poljubne aplikacije.

Tretji cilj magistrskega dela je namenjen neposrednemu povezovanju tiskalnikov z oblačnimi storitvami sistema, s čimer je uporabnikom omogočeno tiskanje etiket brez potrebe po namestitvi tiskalniških gonilnikov (angl. driverless printing). S tem ugodimo nekaterim potrebam razpršenih podjetij, ker jim omogočimo neposredno povezavo tiskalnikov z njihovih oddaljenih lokacij.

1.3 Struktura magistrskega dela

V naslednjem poglavju je predstavljena arhitektura sistema v štirih nivojih. Vsebuje opis osnovnih uporabniških zahtev in funkcionalnosti pripadajočih programov na vsakem nivoju.

Sledi ji kontrola dostopa uporabnikov. Opisana sta dva možna načina dodajanja uporabnikov in dva možna načina dodeljevanja njihovih pravic. Preverjanje pristnosti uporabnikov je izvedeno z uporabo protokola OAuth 2.0, katerega delovanje je prav tako opisano.

Sledi oblačni prožilec, namenjen integracijam obstoječega sistema z drugimi tovrstnimi storitvami. Opisani so njegova arhitektura s pripadajočimi podatkovnimi tokovi, implementacija ter način delovanja.

Magistrsko delo se nato nadaljuje z opisom oblačnega tiskalniškega pove-

zovalca, namenjenega neposredni povezavi tiskalnikov z oblačnimi storitvami. Njegova vsebina je razdeljena enako kot pri oblačnem prožilcu. Poleg tega je predstavljena tudi ideja produkcijske zasnove ter dejanska implementacija prototipa.

Pri implementaciji razširitev sistema za upravljanje z etiketami sta bila uporabljena dva arhitekturna in trije načrtovalski vzorci, opisani v poglavju z naslovom Programerski vzorci. Uporabljena arhitekturna vzorca sta namenjena predvsem ločitvi implementacije domenske logike in uporabniškega vmesnika, načrtovalski vzorci pa so namenjeni večkratni uporabi programske kode. S tem je magistrsko delo vsebinsko zaključeno in vsebuje le še pregled sklepnih ugotovitev.

Poglavje 2

Pregled sorodnih del

Trendi sodobne družbe od proizvodnih in predelovalnih podjetij vse pogosteje zahtevajo označevanje in sledljivost izdelkov, ki sega vse od njihovega nastanka do uničenja. Takšnim zahtevam podjetja zaradi njihove razpršenosti najlažje sledijo z označevanjem in etiketiranjem vseh izdelkov ter hranjenjem ustreznih informacij. Različni načini povezovanja fizičnega in digitalnega sveta ter nekatere njihove karakteristike so predstavljeni v delu [1].

Večja sodobna podjetja imajo pri etiketiranju največkrat težave z uporabo raznolike programske opreme za načrtovanje in tiskanje etiket, kar privede do njihovega različnega izgleda in težavnega vzdrževanja. Potreba po enotnosti in nadzorovanem upravljanju različic etiket je opisana v delu [3]. Enoten in dober načrt etiket podjetjem omogoča enostavno in hitro prilagajanje z ustreznim nadzorom kakovosti. Poleg tega večina podjetij deluje na mednarodnih trgih, ki od njih zahtevajo opise izdelkov v lokalnih jezikih. V takšnem primeru lahko ena etiketa s spremenljivim besedilom in ustreznim načrtom nadomesti več etiket s fiksno vsebino.

Sodobna podjetja za dostop in delitev svojih podatkov od ponudnikov programske opreme zahtevajo celostno storitev, ki povezuje njihove naprave in združuje različne podatkovne vire. Odmikajo se od nakupa programske opreme in se vse večkrat odločajo za najem le-te v obliki oblačnih storitev. Podobno je s ponudniki programske opreme, ki za delovanje svojih produktov

uporabljajo različne oblačne storitve, opisane v delu [4].

Ponudba programske opreme, kot so storitve SaaS (angl. Software as a Service), največkrat vključuje uporabo iste aplikacije s strani večjega števila najemnikov storitve. S tem sta ponudniku storitev zagotovljena lažji nadzor nad uporabljenimi različicami aplikacije in boljša izkoriščenost potrebne infrastrukture. Uporaba iste aplikacije ne predstavlja uporabe istih podatkov, ampak zahteva izolacijo vsakega najemnika. Izolacijo in ločitev najemnikov lahko dosežemo z ustrezno uporabo skupnih sistemskih in ločenih vsebinskih podatkovnih baz, kot opisuje delo [5].

Koncept oblačnih storitev se je v zadnjem času zelo razvil. Ponujanje storitev kjerkoli in kadarkoli je danes postalo samoumevno. Delo [6] opisuje združitev oblačnih storitev s svetom interneta stvari IoT (angl. Internet of Things) in ponudbo najrazličnejših stvari v obliki storitev. Z njuno združitvijo sta napravam omogočena zajem in posredovanje podatkov oblačnim storitvam, ki jih lahko primerno obdelajo in predstavijo uporabniku. Oblačna storitev lahko uporabniku omogoči tudi posredovanje podatkov povezanim napravam in s tem nadzor nad njihovim delovanjem.

Povezava naprav IoT in oblačnih storitev nam ponuja možnost vzpostavitve nadzora dostopa. Naprave so lahko nastavljene tako, da zaupajo le oblačni storitvi ali prehodu, ki je nadzorovan z njihove strani. Različni načini takšne povezave so podrobneje opisani v delu [7]. Najenostavnejši in najprimernejši način za namen oblačnega tiskanja je vzpostavitev neposredne povezave tiskalnika z oblačno storitvijo.

Oblačna storitev mora za izvajanje nadzora dostopa do naprav najprej preveriti pristnost uporabnikov in jih nato avtorizirati. Za ta namen je bil predlagan avtorizacijski okvir OAuth 2.0, opisan v določilu RFC 6749 (angl. Request for Comments) v dokumentu [8]. Začne se z opisom sodelujočih vlog in njihovih nalog ter nadaljuje s potekom preverjanja pristnosti uporabnikov in podrobnejšim opisom posameznih sestavnih delov. Predstavljeni so različni načini preverjanja pristnosti in njihove lastnosti. Z uporabo avtorizacijskega okvira OAuth 2.0 nam je omogočena izbira najustreznejšega

načina preverjanja uporabnikov.

Implementacija razširitev, ki so potrebne za podporo oblačnemu tiskanju, mora biti razumljiva in enostavna za vzdrževanje. To zahteva njihovo ustrezno organizacijo in strukturo, ki temelji na uporabi preverjenih načrtovalskih vzorcev, opisanih v knjigi [9]. Predstavljeni so obstoječi primeri uporabe, njihov namen, struktura ter prednosti in omejitve. Njihovo poznavanje nam omogoča izbiro in prilagoditev opisanih vzorcev ter s tem izbiro ustreznejše rešitve obravnavanega problema. Poleg tega nas privede do hitrejše implementacije in lažjega razumevanja delovanja programov.

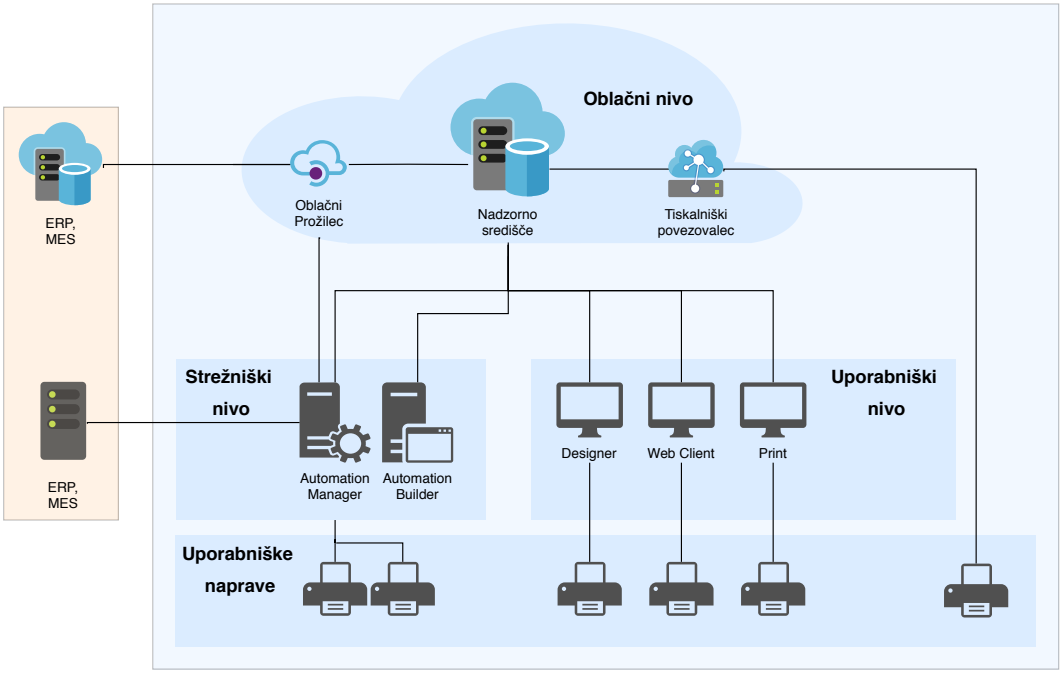
Poglavje 3

Arhitektura sistema

Sistem za upravljanje z etiketami je sestavljen iz 4 nivojev, ki jih prikazuje slika 3.1. Njegova delitev temelji na nalogah posameznih aplikacij in naprav ter na okolju, v katerem se le-te nahajajo. Z nadzornega vidika si nivoji sledijo od zgoraj navzdol, saj višji nivoji praviloma nadzorujejo delovanje nižjih nivojev. Višji nivoji agregirajo nekatere podatke, pridobljene z nižjih nivojev, poleg tega pa so odgovorni tudi za povezovanje in integracije sistema z drugimi potrebnimi sistemi. S tem povečajo njegovo uporabnost in učinkovitost izvajanja procesov podpori, katerih so namenjeni. Omenjeni arhitekturni nivoji so sledeči:

- Oblačni nivo, opisan v podpoglavju 3.4,
- Strežniški nivo, opisan v podpoglavju 3.3,
- Uporabniški nivo, opisan v podpoglavju 3.2,
- Uporabniške naprave, opisane v podpoglavju 3.1.

Uporabniški vidik je v primerjavi z nadzornim ravno nasproten, saj najosnovnejši uporabniki lahko uporabljajo le nižje nivoje za ceno manjšega nadzora in brez naprednejših funkcionalnosti. Nabor osnovnih funkcionalnosti in namen posameznih aplikacij nivojev pa ostajata enaka. V nadaljevanju



Slika 3.1: Arhitektura celotnega sistema.

sledita podrobnejši opis in namen posameznih nivojev, predstavljena z uporabniškega vidika. Utemeljena z zahtevami ter primeri uporabe posameznih aplikacij.

3.1 Uporabniške naprave

Najnižji in najosnovnejši nivo predstavljajo uporabniške naprave, od katerih je odvisen celoten sistem za upravljanje z etiketami. Predstavljajo osnovo za načrtovanje in tiskanje etiket, saj sta oblika in načrt etiket odvisna od njihovih omejitev.

Tiskalniki so glavni predstavnik uporabniških naprav. Njihove lastnosti in omejitve so zelo različne. Razlikujejo se že v vrsti in velikosti medija, na katerega tiskajo. Ta je lahko papir, samolepilni papir, karton, plastika ali celo tekstil. Ob tem pa so mediji lahko tudi različnih velikosti. V odvisnosti od ciljnega medija imajo tiskalniki tudi svoje notranje omejitve, kot

so sposobnost barvnega tiskanja, dvostranskega tiskanja, rezanja medija in podobno.

Z arhitekturnega vidika ločimo predvsem dve vrsti tiskalnikov: klasične in oblačne. Klasični tiskalniki so povezani z uporabniškim ali strežniškim nivojem, oblačni pa z razširitvami oblačnih storitev. Način povezave in uporabe oblačnih tiskalnikov je podrobneje opisan v poglavju 6. Glavna razlika med oblačnimi in klasičnimi tiskalniki je njihov način uporabe. Klasični tiskalniki so namenjeni predvsem uporabi znotraj nekega lokalno izoliranega okolja, medtem ko se oblačni tiskalniki lahko uporabljajo tudi v razpršenih in dislociranih okoljih.

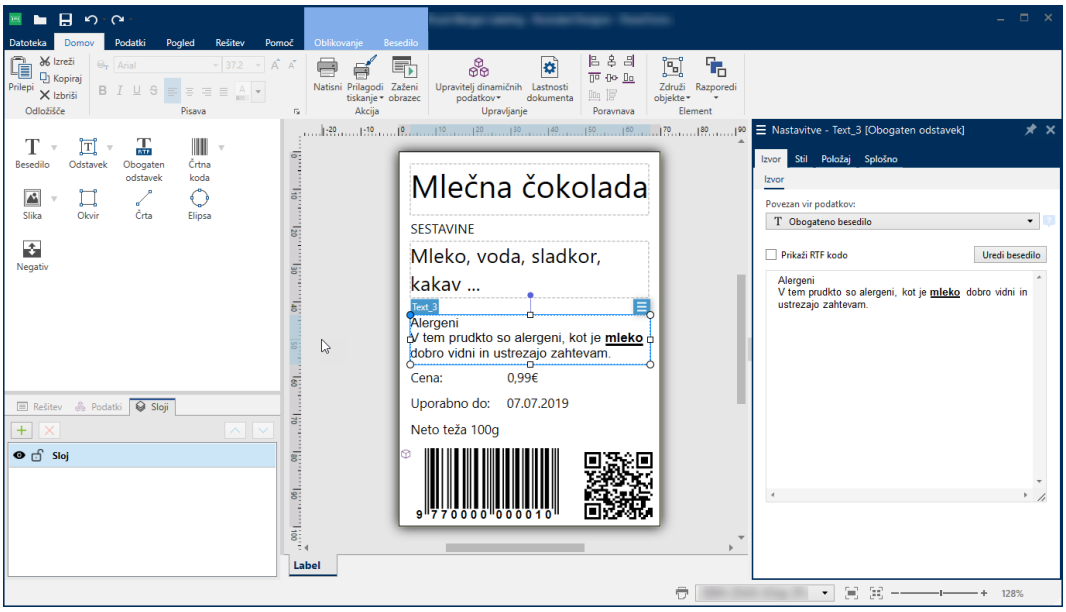
3.2 Uporabniški nivo

Uporabniški nivo predstavlja najnižji nivo programske opreme in je osnova za vse nadaljnje nivoje. Namenjen je končnim uporabnikom sistema za upravljanje z etiketami. Uporablja se ga za načrtovanje in ročno tiskanje etiket, kar v celoti zadošča potrebam osnovnih uporabnikov. Sestavljen je iz treh aplikacij, ki so opisane v podpoglavjih Designer 3.2.1 ter Print in Web Client 3.2.2. Poleg osnovnih funkcionalnosti aplikacij uporabniški nivo omogoča povezavo z višjim nivojem in s tem zadosti potrebam zahtevnejših uporabnikov.

3.2.1 Designer

Z etiketiranjem ali vsaj z uporabo etiket se srečuje večina podjetij, ki rokuje s fizičnimi izdelki. Ta podjetja delujejo na skoraj vseh področjih, od proizvodnje in njene dobavne verige do logistike in trgovine. Na vseh področjih pa se življenjski cikel etiket začne z njihovim načrtovanjem.

Za lažje razumevanje poslovnih potreb si poglejmo primer osnovnega uporabnika. Vzemimo manjšo trgovino, ki se ukvarja s prodajo pohištva. Trgovina pohištvo kupi od dobaviteljev, ga ustrezno uskladišči in označi z etiketami. Potrebne etikete lahko natisnejo v eni izmed pisarn in jih nato prinesejo



Slika 3.2: Prikaz načrta etikete.

v skladišče, kjer jih nalepijo na izdelke. Da se podjetje izogne nepotrebnim napakam, etikete največkrat vsebujejo ime izdelka, njegove osnovne podatke in črtno kodo, ki je potrebna za strojno identifikacijo. Takšne trgovine imajo le dve potrebi: enostavno načrtovanje in enostavno tiskanje etiket.

Posledično je osnovna funkcionalnost celotnega sistema za upravljanje z etiketami njihovo načrtovanje in tiskanje, le-ta pa je podprta s programom Designer, prikazanim na sliki 3.2. Načrtovanje etiket se prične z izbiro ciljnega tiskalnika in velikostjo medija, na katerega bo etiketa natisnjena. Sledi oblikovanje njenega izgleda, za kar je v programu Designer na voljo naslednjih sedem vrst osnovnih gradnikov:

- Besedilo, ki predstavlja človeku razumljiv opis. Njegov vir podatkov je lahko statičen ali dinamičen. Statičen vir predstavlja vnaprej vnešeno besedilo, medtem ko je dinamičen vir lahko vezan na vnos s tipkovnice, trenutni datum, čas ali števec. Poleg različnih virov podatkov lahko za besedilo izberemo tudi različne pisave, oblike in poravnave.

- Odstavek, ki je nadgradnja elementa "Besedilo"in predstavlja eno izmed konkurenčnih prednosti programa Designer, omogoča načrtovanje polja z besedilom, ki ima fiksno širino in sprejme besedilo spremenljive dolžine. Njegova višina je odvisna od izbrane nastavitve uporabnika in je lahko spremenljiva ali fiksna. V primeru spremenljive višine je višina polja odvisna od dolžine besedila, v primeru fiksne višine polja pa se besedilo ustrezno prilagodi. V primeru fiksne višine polja sta uporabniku na voljo prilagodljiva velikost in razteg pisave, katerima mora zaradi boljše berljivosti določiti ustrezno zgornjo in spodnjo mejo. Njegova funkcionalnost ima največjo vrednost v primeru tiskanja etiket spremenljive dolžine, saj lahko s tem prihrani velike količine porabljenega medija.

- Obogaten odstavek, ki podobno kot Odstavek predstavlja konkurenčno prednost. Namenjen je načrtovanju polja obogatenega besedila, ki vključuje uporabo različnih tipov pisav, slogov, barv, postavitev in podobnih lastnosti besedila znotraj enega elementa. Omenjene obogatitve so lahko različne za vsak vsebovan znak besedila. Podobno kot komponenta Odstavek omogoča uporabo fiksne širine. Razlikuje se le v nastavitvah prilagoditve besedila, saj vsebuje zgolj spremenljivo višino polja s prilagodljivo velikostjo pisave. Primer prednosti njegove uporabe v zadnjem času predstavljajo etikete prehrambnih artiklov, ki morajo vsebovati poudarjene in dobro vidne oznake alergenov.

- Črtne kode, ki so namenjene enostavni strojni identifikaciji artiklov. Njihov izgled temelji na izbrani vrsti in funkciji kodiranja podatkov. Na splošno črtne kode delimo v dve družini: enodimenzionalne in dvodimenzionalne. Nam verjetno najbolj znana predstavnica dvodimenzionalnih je koda QR (angl. Quick Response Code) opisana v delu [10].

- Slika iz datoteke, katere namen je lahko lepši izgled etikete ali pa prikaz pomembne informacije. Primer uporabe slike s pomembno informacijo

predstavljajo etikete za označevanje nevarnih snovi, ki morajo prikazovati potrebne simbole.

- Oblike: pravokotnik, črta in elipsa, namenjene boljšemu pregledu in ločevanju posameznih sklopov vsebine etikete.

- Inverz, namenjen barvnemu inverzu elementov etikete.

Etikete načrtovane v programu Designer lahko natisnemo in shranimo na disk ter s tem zadostimo potrebam osnovnega uporabnika.

Predpostavimo, da je lastnik zgoraj omenjene trgovine uspešen poslovnež, ki se je odločil za širitev kapacitet trgovine in njenega skladišča. Postopoma bo začel prodajati večji nabor izdelkov, pri čemer pa za vsak nov izdelek ne želi izdelovati nove etikete in izgubljati dragocenega časa. Za izdelke vsake kategorije želi imeti etikete z enotnim izgledom, ki se bodo razlikovale le v imenu in lastnostih izdelka. Poleg tega želi imeti na njih tudi samodejno generirane črtne kode. Vnos novih izdelkov bi rad zaupal eni osebi, tiskanje etiket pa drugi osebi. Tiskanje se bo izvajalo vedno na istem tiskalniku z istimi nastavitvami. Lastnik takšne trgovine je postal nekoliko zahtevnejši uporabnik, ker potrebuje eno rešitev za nadzorovan vnos podatkov o izdelkih in drugo za prilagojeno tiskanje etiket.

Grajenje aplikacij oziroma rešitev po meri nam omogočajo napredne funkcionalnosti programa Designer. Z njimi lahko brez programiranja prilagodimo obrazec za tiskanje etiket in ustvarimo nove obrazce, potrebne za izdelavo rešitev po meri. Rešitve, zgrajene s programom Designer, se shranijo v obliki posebnih datotek. Zagon zgrajenih rešitev je mogoč z uporabo programa Designer, a je priporočljiv le za namene njihovega razvoja in testiranja, saj uporabnikom rešitev ne smemo omogočiti njihovega urejanja. Podobno je z načrtovanjem in tiskanjem etiket, kjer mora biti načrtovanje etiket omogočeno le načrtovalcem, tiskanje pa le tiskarjem. Za zagon rešitev in tiskanje etiket v produkcijskem okolju je priporočljiva uporaba programov Print in Web Client, opisanih v podpoglavju 3.2.2.

Rešitve po meri so podobno kot etikete zgrajene iz osnovnih gradnikov, ki

so podobni standardnim komponentam uporabniškega vmesnika, in je zato njihov opis izpuščen. Pri načrtovanju etiket in rešitev po meri nam napredne funkcionalnosti programa Designer poleg osnovnih komponent omogočajo tudi uporabo spremenljivk in podatkovnih baz, v primeru sklenjenega naročniškega razmerja in uporabe oblačnih storitev pa tudi neposredno urejanje etiket ter rešitev, shranjenih v shrambi dokumentov, ki je del nadzornega središča, opisanega v podpoglavju 3.4.1.

3.2.2 Print in Web Client

Programa Print in Web CLient sta v produkcijskem okolju zelo pomembna. Predstavljata najenostavnejši aplikaciji uporabniškega nivoja, ker omogočata le tiskanje etiket in zagon rešitev po meri. V osnovi sta si programa zelo podobna, razlikujeta se le v načinu dostopa do dokumentov in nadzoru dostopa. Print je enostavnejši in je namenjen osnovnejšim uporabnikom. Omogoča jim tiskanje etiket in zagon rešitev po meri, shranjenih na lokalno dostopnih datotečnih nosilcih brez vključenega nadzora dostopa uporabnikov. Web Client je za razliko od Printa naprednejši in za svoje delovanje zahteva povezavo z nadzornim središčem oblačne storitve za upravljanje z etiketami. Njegova glavna prednost je možnosti tiskanja etiket in zagon rešitev po meri, ki so shranjene v shrambi dokumentov, s čimer je uporabnikom omogočeno upravljanje različic shranjenih dokumentov in dodeljevanje pravic za njihovo uporabo.

3.3 Strežniški nivo

Strežniški nivo predstavlja naslednji nivo sistema za upravljanje z etiketami. Namenjen je podpori in avtomatizaciji procesov tiskanja etiket ter integracijam z drugimi lokalnimi sistemi. Avtomatizacija procesov lahko podjetjem prinese velike prihranke in zmanjša število nepotrebnih in dragih napak. Poleg zmanjšanja števila napak pa omogoča tudi večjo dinamičnost podjetij in hitrejši odziv na spremembe, kar lahko v današnjem hitro rastočem in dina-

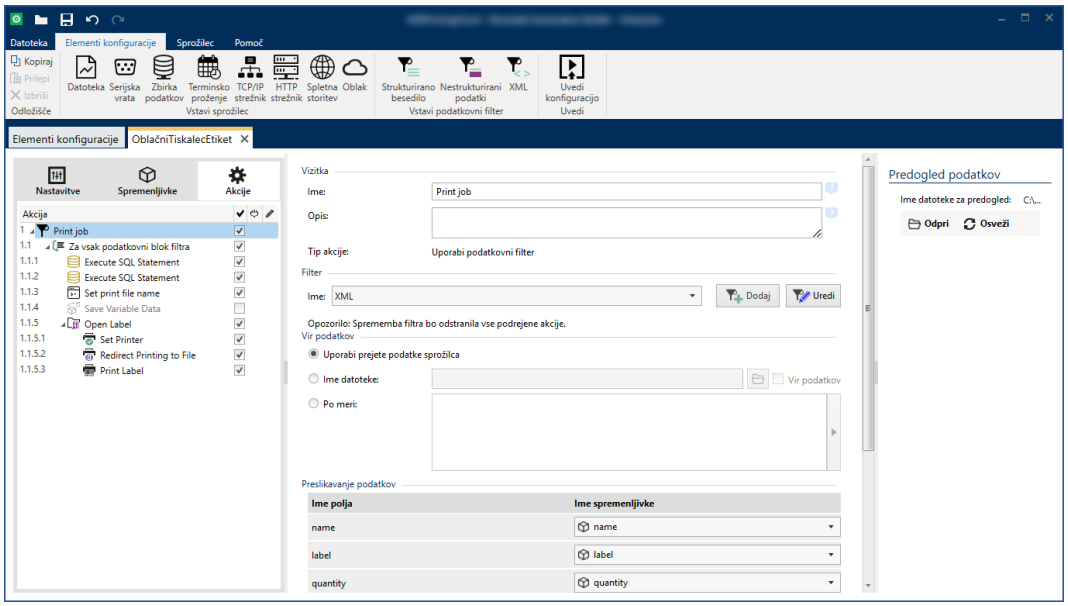
mičnem okolju predstavlja veliko poslovno prednost. Odraža pa se predvsem v krajšem čas prihoda na tržišče TTM (angl. Time To Market).

Za lažje razumevanje potreb uporabnikov nadaljujemo s primerom trgovine s pohištvom, ki ponovno povečuje kapacitete prodanih izdelkov. Tokrat potrebuje učinkovitejšo rešitev, s katero bodo zaposleni lažje in hitreje označevali prispelo blago. Izogniti se želijo ročnemu tiskanju etiket, ki je časovno zamudno in lahko vključuje tudi nepotrebne človeške napake. Želja lastnika trgovine je povečanje obrata izdelkov brez potrebe po povečanju kapacitet skladišča, s čimer želi ohraniti trenutne stroške in povečati zaslužek. Razširiti želi svoje obstoječe etikete in rešitve po meri. Namesto ročnega tiskanja etiket želi imeti samodejno tiskanje, ki se bo prožilo ob vnosu novih izdelkov in količin, shranjenih v podatkovni bazi. Za vnos podatkov želi uporabiti obstoječe rešitve po meri, ki za zahtevano delovanje potrebujejo manjše prilagoditve. Takšnim in podobnim primerom sta namenjena predstavnika strežniškega nivoja, katerih glavna naloga je načrtovanje in izvajanje avtomatizacije procesa tiskanja etiket.

3.3.1 Automation Builder

Avtomatizacija procesa tiskanja je zelo odvisna od potreb uporabnikov, saj temelji na uporabi njihovih etiket in drugih podatkov, potrebnih za izvedbo tiskanja. Temelji na načrtu avtomatizacije procesa tiskanja pripravljenem s pomočjo programa Automation Builder, prikazanem na sliki 3.3. Njegova glavna naloga je definicija različnih prožilcev in njihovih nastavitev. Načrt avtomatizacije programa Automation Builder je lahko zgrajen na osnovi naslednjih osmih prožilcev:

- Datotečni prožilec, ki ga lahko aktiviramo na tri različne načine. Opazuje lahko eno samo datoteka ali celotno mapo z vsebovanimi datotekami. Spremembe opazovanega vira povzročijo njegovo aktivacijo. V primeru opazovane celotne mape aktivacijo prožilca povzroči sprememba katere koli vsebovane datoteke ali sprememba njihovega števila. Tipični primer uporabe datotečnega prožilca predstavljajo programi, ki



Slika 3.3: Prikaz konfiguracije akcij oblačnega prožilca.

na določenem podatkovnem nosilcu in na določeni poti ustvarjajo datoteke.

- Serijski prožilec, ki je namenjen poslušanju prometa serijskih vrat RS232 in ki se aktivira ob prejemu novih podatkov. Njegovo tipično uporabo predstavljajo industrijske tehtnice in podobne naprave. Tehtnice prožilcu največkrat pošljejo le podatek o teži izdelka in njegov enolični identifikator.

- Bazni prožilec, ki je namenjen spremljanju sprememb v podatkovnih bazah. Njegovo proženje povzročijo spremembe zapisov v opazovani tabeli. Tipično se uporablja za tiskanje novih etiket na podlagi novo vnešenih karakteristik in količin izdelkov.

- Terminski prožilec, ki je edini neodvisen od zunanjih vplivov. Njegova aktivacija je odvisna le od nastavljenega časovnega intervala. Namen njegove uporabe predstavlja ponovljivo delo, kot je vsakodnevno brisanje samodejno generiranih dnevniških datotek, starejših od enega

tedna.

- Strežnik TCP/IP (angl. Transmission Control Protocol / Internet Protocol), ki je namenjen poslušanju prometa na lokalnem internetnem naslovu IP in nastavljenih vratih. Njegov način aktivacije nastavi uporabnik. Lahko jo povzroči prekinitev povezave odjemalca, dosežena določena količina prejetih podatkov ali vsebovanost določenih znakov v prejetem podatkovnem toku. Tipično je namenjen neposredni integraciji z drugimi sistemi, ki pošiljajo podatke sistemu za upravljanje z etiketami.

- Strežnik HTTP (angl. Hypertext Transfer Protocol), ki predstavlja naprednejšo različico strežnika TCP/IP. Razlikuje se predvsem v uporabi bolj definiranega protokola HTTP, ki uporabnikom določa obliko sporočil in s tem tudi način komunikacije. Poleg nastavitve internetnega naslova IP in vrat lahko uporabnik strežniku HTTP nastavi pot poslušanja. Določi lahko vsebino odgovora in pripadajoče glave HTTP ter zahteva uporabo varne povezave HTTPS (angl. Hypertext Transfer Protocol Secure). Njegova aktivacija je v primerjavi s strežnikom TCP/IP že vnaprej določena in se zgodi ob prejemu sporočila. Njegov namen uporabe je podoben kot pri strežniku TCP/IP, in to je predvsem prejemanje in posredovanje sporočil drugim sistemom.

- Spletne storitve, katerih delovanje določa protokol SOAP (angl. Service Oriented Access Protocol), ki je podrobneje opisan v delu [11]. Njihova razlika v primerjavi s prej opisanima strežnikoma je določitev zapisa poslanih in prejetih sporočil, ki morajo ustrezati strukturnemu jeziku XML (angl. Extensive Markup Language). Poleg strukture sporočil pa protokol SOAP predpisuje tudi opis delovanja ponujenih storitev, ki mora ustrezati opisnemu jeziku WSDL (angl. Web Service Descripition Language).

- Oblačni prožilec, ki je namenjen povezovanju sistemov za upravljanje z etiketami z drugimi tovrstnimi storitvami. Omogoča vzpostavitev po-

vezave brez potrebe po izpostavljanju lokalnega omrežja in odpiranju vrat požarnih zidov. Njegova implementacija, nastavitev in način uporabe predstavljajo pomemben del magistrske naloge in so podrobneje opisani v poglavju 5. Aktivacija in namen uporabe oblačnega prožilca sta podobna kot pri strežniku HTTP in se razlikujeta le v tem, da je strežnik HTTP namenjen integracijam znotraj lokalno izoliranega omrežja, oblačni prožilec pa integracijam z zunanjimi sistemi.

3.3.2 Automation Manager

Program Automation Manager ima pri avtomatizaciji procesa tiskanja najpomembnejšo vlogo, podobno, kot jo imata programa Print in Web Client pri izvajanju rešitev po meri in ročnemu tiskanju etiket. Namenjen je izvajanju in upravljanju predhodno definiranih načrtov avtomatizacije in vsebovanih prožilcev. Tipično se izvaja na strežniških sistemih. Uporabnikom omogoča dodajanje in odstranjevanje načrtov avtomatizacije, zagon in zaustavitev posameznih prožilcev ter pregled stanja njihovega izvajanja in dnevnika dogodkov.

3.4 Oblačni nivo

Najvišji nivo sistema za upravljanje z etiketami predstavlja oblačni nivo, ki je namenjen najzahtevnejšim uporabnikom in jim je na voljo v obliki programske opreme oziroma kot storitve SaaS, opisane v delu [12]. Njegova glavna naloga je globalni nadzor in administracija celotnega procesa od načrtovanja do tiskanja etiket. Glavna aplikacija tega nivoja je Nadzorno središče, ki je bilo tekom magistrskega dela razširjeno. Poleg tega pa sta bili oblačnim storitvam dodani tudi funkcionalnosti za integracijo z drugimi tovrstnimi sistemi ter možnost neposredne povezave z oblačnimi tiskalniki. Neposredna povezava tiskalnikov uporabnikom omogoča tiskanje etiket brez potrebe po lokalni namestitvi in nastavitvi tiskalniških gonilnikov. Prednosti ponudbe in uporabe oblačnih aplikacij so opisane v delu [4] in so v našem primeru

zgoščene s strani ponudnika Microsoft Azure.

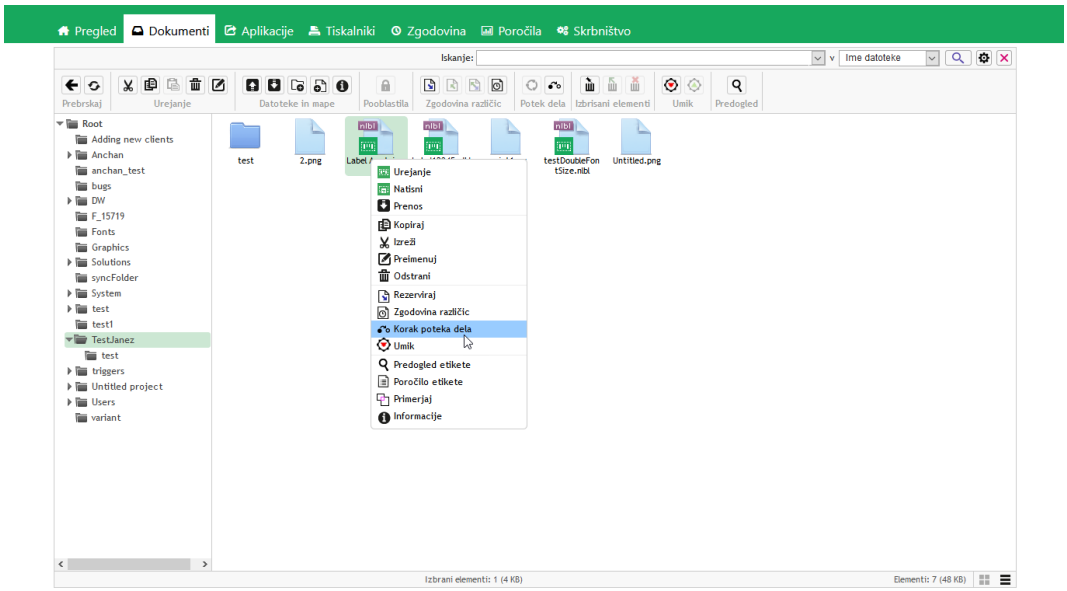
3.4.1 Nadzorno središče

Nadzorno središče je glavna aplikacija oblačnega nivoja. Namenjena je globalnemu nadzoru celotnega sistema za upravljanje z etiketami, ki vključuje vse od hrambe dokumentov in upravljanja njihovih različic do upravljanja uporabniških pravic in beleženja dogodkov. Uporabnikom je na voljo v obliki spletne aplikacije, sestavljene iz šestih glavnih sklopov, ki so opisani v nadaljevanju.

Za lažje razumevanje uporabniških potreb si ponovno poglejmo primer lastnika trgovine s pohištvom, ki tokrat izjemno širi svoje kapacitete. Zaradi naročanja ogromnih količin izdelkov si njihovega ponovnega označevanja ne more več privoščiti, še vedno pa si želi obdržati svoj izgled etiket in nadzor nad njim. Posledično mora etikete zagotoviti dobaviteljem, ki bodo z njimi označevali njegove izdelke. Prilagojene rešitve za vnos podatkov o izdelkih in tiskanje etiket ima že pripravljene, manjka mu le še njihov varen in nadzorovan način deljenja, kar je glavna naloga nadzornega središča.

Pregled

Prvi najenostavnejši sklop Nadzornega središča predstavlja zavihek za osnovni pregled sistema. Njegov prvi sklop prikazuje informacije o sklenjenem naročniškem razmerju, katerega osnovna kriterija sta izdaja in največje možno število registriranih tiskalnikov. Sledi mu sklop s povezavami za prenos in namestitev programov Designer in Web Client, ki sta namenjena večjemu številu končnih uporabnikov. Omenjenima sklopoma sledita še tabelarični prikaz zgodovine aktivnih delovnih postaj v zadnji uri in seznam nedavnih napak.



Slika 3.4: Prikaz shrambe dokumentov.

Shramba dokumentov

Pregledu sledita vsebinsko zelo pomembna sestavna dela Nadzornega središča. Prvega predstavlja globalna shramba dokumentov, prikazana na sliki 3.4. Namenjena je hranjenju dokumentov in nam poleg osnovnih funkcionalnosti, ki smo jih vajeni iz klasičnih podatkovnih nosilcev, ponuja tudi funkcionalnosti za nadzor nad potekom dela (angl. workflow) in upravljanje različic dokumentov. Omenjeni funkcionalnosti sta zelo pomembni v strožje reguliranih delovnih okoljih. Primera takšnih sta kemijska in farmacevtska industrija saj lahko napake na etiketah zelo hitro vodijo do velike škode. Napačen simbol na jedki kemikaliji lahko vpliva na način rokovanja in posledično povzroči kemične opekline, napačni podatki, natisnjeni na škatlici zdravil, pa lahko povzročijo zastrupitev in celo smrt bolnika. Za preprečitev napak ter podporo poteku dela in upravljanju različic etiket so nam v shrambi dokumentov na voljo naslednje funkcionalnosti:

- rezervacija in sprostitev rezervacije dokumentov v urejanju,

- pregled zgodovine različic posameznih dokumentov,
- prehod na naslednji korak poteka dela,
- umik neprimernih dokumentov in njihov ponovni priklic,
- predogled etiket,
- poročilo o sestavnih delih in lastnostih etiket,
- primerjava etikete iz prejšnjega koraka poteka dela s trenutnim in prikaz njunih vizualnih razlik,
- splošne informacije o dokumentu, pripadajočem poteku dela in naša pooblastila za delo z njim.

Aplikacije

Drugi, vsebinsko zelo pomemben sklop predstavljajo aplikacije, sestavljene iz treh delov: spletnih aplikacij, oblačnih integracij in oblačnih tiskalnikov. Spletne aplikacije so namenjene konfiguraciji dostopa do etiket in rešitev po meri, ki jih lahko izvajamo s programom Web Client. Njihove nastaviteve vsebujejo pomembne vsebinske podatke o uporabljeni rešitvi oziroma etiketi, vključno z avtoriziranimi uporabniki in njihovimi omejitvami, ki so podrobneje opisane v podpoglavju 4.4. Sledi jim nadzor oblačnih integracij, s čimer je skrbnikom sistema omogočeno dodajanje integratorjev in generiranje enoličnih ključev, potrebnih za uporabo oblačnega prožilca, podrobneje opisanega v poglavju 5. Zadnji del pa predstavlja nadzor oblačnih tiskalnikov oziroma njihovih registracij, ki temeljijo na serijski številki in modelu tiskalnika. Registracija in uporaba oblačnega tiskalnika sta podrobneje opisana v podpoglavju 6.2.

Tiskalniki

Sledijo trije sklopi nadzornega središča, namenjeni nadzoru in administraciji. Prvi izmed njih je nadzor povezanih tiskalnikov, ki nam omogoča pregled

in izvedbo nekaterih osnovnih akcij. Možna izvedba akcij in prikazan nivo informacij o tiskalnikih sta odvisna od njihove licenčne rezervacije. Osnovne akcije nam tako omogočajo rezervacijo in sprostitev licenčnih mest za vsakega izmed tiskalnikov. Naprednejše akcije pa nam omogočajo zaustavitev in nadaljevanje procesa tiskanja ter izbris čakalne vrste dokumentov le za licencirane tiskalnike. Omogočeni so nam pregled licenciranih tiskalnikov, pregled vseh tiskalnikov ter pregled tiskalnikov po skupinah. Zadnji privzeto vsebuje štiri skupine, ki temeljijo na stanju tiskalnika. Tiskalnik je lahko v naslednjih stanjih: v napaki, začasno ustavljen, pripravljen na tiskanje ali tiska. Poleg privzetih skupin je uporabniku omogočeno tudi dodajanje novih.

Zgodovina

Naslednji sklop predstavlja zgodovina dejavnosti sistema za upravljanje z etiketami. Zgodovina je ponovno zelo pomembna v reguliranih okoljih, ker nam omogoča pregled preteklih dejavnosti ter analizo in rekonstrukcijo morebitnih neželenih dogodkov. Pregled zgodovine dejavnosti je razdeljen v naslednjih 5 skupin: tiskanje, napake, sistemski dogodki, poslana opozorila in vse dejavnosti. Poudariti velja predvsem sistemske dogodke, ki vsebujejo vse od prijav v sistem, vključno z uporabo aplikacij uporabniškega nivoja, do sprememb, narejenih v nadzornem središču.

Skrbništvo

Zadnji funkcijski sklop nadzornega središča sistema za upravljanje z etiketami predstavlja skrbništvo. Omogoča nam globalni nadzor dostopa, nastavitev upravljanja različic in poteka dela shranjenih dokumentov ter globalnih nastavitev.

Globalni nadzor dostopa je sestavljen iz dveh delov; upravljanja uporabnikov ter upravljanja vlog dostopa in pripadajočih pravic. Nastavitve teh dveh delov so z vidika varnosti celotnega sistema ključnega pomena in so podrobneje opisane v poglavju 4. Naslednja funkcionalnost je namenjena nastavitvi shranjevanja različic in poteka dela. Njena vloga je z vsebinskega

vidika zelo pomembna, saj določa potek odobritve dokumentov in njihove odgovorne osebe.

Sledijo globalne nastavitve. Njihov prvi sklop uporabnikom sistema omogoča zamenjave podatkovnih zbirk in s tem možnost ločitve razvojnega in produkcijskega okolja. Etikete ter rešitve po meri lahko tekom njihovega načrtovanja in razvoja uporabljajo razvojne podatkovne zbirke. Nato gredo skozi postopek poteka dela, ki jih vodi do odobritve. Šele odobrene etikete in rešitve po meri lahko pričnejo z uporabo produkcijskih podatkovnih zbirk. Sledijo globalne spremenljivke, ki so vidne iz programov vseh nivojev sistema za upravljanje z etiketami. Uporabljajo se za deljenje skupnih informacij, kot so število vseh natisnjenih etiket, osnovni podatki za generiranje serijskih številk in podobno.

Nepravilnosti med uporabo sistema za upravljanje z etiketami lahko privedejo do njegovih napak, na opozorila katerih se lahko prijavimo. Ločimo dve vrsti napak. Prva vsebuje napake programov, do katerih je prišlo v produkcijskem okolju: Designer, Print in Automation; druga vrsta pa prikazuje napake, povezane s tiskalniki, in sicer kršitve števila licenciranih tiskalnikov in napake njihovega delovanja.

Opozorilom sledi funkcionalnost za zahtevnejše uporabnike, ki uporabljajo več sistemov za upravljanje z etiketami. Popolnoma lahko ločijo testni in produkcijski sistem ali pa se odločijo za uporabo ločenih sistemov na različnih lokacijah. V primeru uporabe ločenih sistemov lahko med seboj sinhronizirajo vsebino shrambe dokumentov. Sinhronizacija je možna dnevno ob določeni uri ali ob določenem časovnem intervalu. Interval je lahko določen natančno na nekaj minut, ur ali dni.

Zadnji del skrbniškega nivoja predstavljajo podrobnejše informacije o računu. Vključujejo podatke o sklenjenem naročniškem razmerju, porabljenem prostoru shrambe dokumentov, času hranjenja zgodovine podatkov ter uporabniški podatkovni bazi.

3.4.2 Oblačni prožilec

Oblačni prožilec je namenjen povezovanju zunanjih sistemov in oblačnih storitev s sistemom za upravljanje z etiketami. Deluje na dveh nivojih sistema, strežniškem in oblačnem, kot je opisano zgoraj. Na oblačnem nivoju ima oblačni prožilec dve nalogi. Prva naloga je upravljanje programskega vmesnika API (angl. Application Programming Interface) za integracije z drugimi tovrstnimi sistemi in storitvami, druga glavna naloga pa je sam programski vmesnik, katerega implementacija in delovanje sta podrobneje opisana v poglavju 5.

Za lažje razumevanje njegovega namena se ponovno vrnimo k našemu lastniku trgovine, ki zaradi povečanega obsega poslovanja zdaj uporablja oblačne storitve za načrtovanje proizvodnje ERP. Spremljati želi dobavljeno in označeno količino naročenih izdelkov, na podlagi katere bo načrtoval prodajo po posameznih poslovalnicah. Poleg tega bi v oblačni storitvi programa ERP rad videl tudi napake, ki so nastale pri tiskanju etiket. Podatki, potrebni za prikaz zahtevanih informacij, se nahajajo na strežniškem nivoju in so generirani s strani programa Automation Manager. Varna povezava z omenjenim nivojem in pridobitev podatkov sta takšnemu uporabniku omogočena z uporabo razširitve oblačnega prožilca.

3.4.3 Oblačni tiskalniški povezovalec

Oblačni tiskalniški povezovalec je namenjen neposredni povezavi tiskalnikov s sistemom za upravljanje z etiketami. Omogoča uporabo oddaljenih tiskalnikov brez potrebe po nameščanju gonilnikov ter poznavanju njihovih internetnih naslovov IP. Njegova implementacija in način delovanja sta podrobneje opisana v poglavju 6.

Za lažje razumevanje potreb uporabnikov se še zadnjič vrnimo k našemu lastniku trgovine. Zaradi nakupa manjših količin raznolikih izdelkov potrebuje še eno manjše skladišče, ki je geografsko ločeno od prvega. Za tiskanje etiket želi uporabiti obstoječi strežnik z nameščenim programom Automation

Manager. S tem se želi izogniti podvajanju strežnika in stroškom, ki so povezani z vzdrževanjem celotne strežniške infrastrukture. V novem skladišču želi imeti le en računalnik z obstoječo rešitvijo za vnos podatkov o izdelkih in njihovih količinah ter nekaj namenskih tiskalnikov, povezanih z internetnim omrežjem. Vnešni podatki se morajo shraniti v podatkovno bazo in s tem sprožiti avtomatizacijo procesa tiskanja, kar je v primeru ločenih okolij omogočeno z uporaba oblačnega tiskalnika.

Poglavje 4

Kontrola dostopa

Kontrola dostopa sistema za tiskanje etiket je mogoča le z uporabo oblačnega nivoja in njegove glavne aplikacije Nadzornega središča. To središče je odgovorno za preverjanje pristnosti uporabnikov in njihovo avtorizacijo za vse povezane programe, s čimer skrbnikom sistema omogoča globalni nadzor. Za preverjanje pristnosti uporabnikov Nadzorno središče uporablja dva zunanja ponudnika storitev za zagotavljanje identitete (angl. idetity provider), to sta Google in Microsoft. S tem Nadzorno središče omogoča uporabo obstoječih elektronskih računov. Za pridobitev identitete uporabnikov uporablja protokol OAuth2.0, katerega delovanje je podrobneje opisano v podpoglavju 4.2, in povzema določilo RFC 6749, opisano v delu [8]. Poleg samega preverjanja pristnosti uporabnikov Nadzorno središče omogoča tudi njihovo avtorizacijo, ki temelji na uporabi vlog dostopa. Izjema so spletne aplikacije, katerim je zaradi njihovega načina uporabe treba posebej dodatno dodeliti avtorizirane uporabnike.

4.1 Uporabniki

Nadzorno središče omogoča dodajanje dveh vrst uporabnikov: gostujočih in organizacijskih. V skupino gostujočih uporabnikov spadajo vsi posamično povabljeni uporabniki. Skrbnik nadzornega središča mora za povabilo go-

stujočega uporabnika vnesti njegov elektronski naslov in ime, s katerim bo uporabnik zaveden. Poleg tega lahko vabilu doda tudi poljubno sporočilo in si zabeleži kakšno opombo o uporabnikovi vlogi v sistemu. Tako povabljen uporabnik nima še nobenih pravic v sistemu za opravljanje z etiketami. Skrbnik mora uporabniku ob njegovem dodajanju določiti tudi njegovo pripadnost vlogam dostopa ali pa mu omogočiti uporabo katere izmed spletnih aplikacij. Uporabniki po uspešnem dodajanju prejmejo elektronsko pošto z vabilom in povezavo za registracijo. Naloga uporabnikove registracije je pridobitev njegove privolitve (angl. consent), ki se zahteva za uporabo njegovih podatkov o identiteti in se uspešno izvede le z uporabnikovo privolitvijo.

Skupino organizacijskih uporabnikov predstavljajo uporabniki povezanega aktivnega imenika AAD (angl. Azure Active Directory) organizacije, ki je opisan v delu [13]. Za njegovo uspešno povezavo s sistemom za upravljanje z etiketami je potrebna privolitev skrbnika imenika. V primeru njegove privolitve Nadzorno središče v svojo bazo uporabnikov doda identitete vseh uporabnikov imenika AAD. Tako dodane uporabnike je treba kot gostujoče dodeliti še vlogam dostopa, kot je opisano v podpoglavju 4.3.

4.2 OAuth 2.0

Avtorizacijski okvir OAuth 2.0 tretjeosebnim aplikacijam (angl. third party application) omogoča pridobitev omejenega dostopa do različnih virov ponujenih storitev HTTP. Dostop do virov je mogoč v imenu njihovih lastnikov ali v imenu aplikacije. V obeh primerih je od lastnikov zahtevana njihova privolitev. Avtorizacijski okvir je natančno predpisan z določilom RFC 6749, izdanim s strani organizacije IETF (angl. Internet Engineering Task Force). Namenjen je zamenjavi starega protokola OAuth 1.0, predpisanega z določilom RFC 5849, ki je opisan v dokumentu [14]. Poglavje je povzeto po določilu RFC 6749, ki ga opisuje dokument [8].

Zgodovina

V tradicionalnem modelu avtorizacije odjemalca in strežnika sta prisotni le njuni dve entiteti, pri čemer odjemalec od strežnika zahteva vir z omejenim dostopom. Za njegovo uspešno pridobitev potrebuje poverilnice njegovega lastnika. Posledično mora lastnik za uspešno delovanje tretjeosebnih aplikacij (odjemalcev) z njimi deliti tudi svoje poverilnice, kar privede do naslednjih težav in omejitev:

- Tretjeosebne aplikacije morajo za dostop do zavarovanih virov hraniti poverilnice njihovih lastnikov, kar v večini primerov predstavlja hrambo uporabniškega imena in gesla v tekstovni obliki.
- S strani strežnikov je zahtevano preverjanje pristnosti uporabnikov z uporabo gesel kljub njihovim varnostnim pomanjkljivostim.
- Tretjeosebne aplikacije pridobijo neomejen dostop do zavarovanih virov, pri čemer njihovi lastniki nimajo nobene možnosti omejitve pravic, ki so dodeljene aplikacijam, in njihove veljavnosti.
- Lastniki virov posameznim aplikacijam ne morejo odvzeti dodeljenih pravic, lahko jih odvzamejo le vsem. Odvzem pravic lahko lastniki virov izvedejo le z menjavo uporabniškega gesla.
- Varnostne pomanjkljivosti in ranljivosti katerekoli izmed aplikacij se odražajo v ranljivosti gesel lastnikov virov, s čimer so izpostavljeni vsi podatki, ki so z njimi zaščiteni.

Avtorizacijski okvir OAuth 2.0 probleme rešuje z uvedbo dodatnega avtorizacijskega nivoja. S tem loči vlogo tretjeosebnih aplikacij, predstavljenih v vlogi odjemalca, od vloge lastnika vira. Z uporabo okvira OAuth 2.0 je odjemalčeva zahteva za dostop do zavarovanega vira nadzorovana s strani njegovega lastnika, pri čemer so odjemalcu dodeljena ločena pooblastila.

Namesto uporabe lastnikovih pooblastil odjemalec za dostop do zavarovanih virov pridobi dostopni žeton (angl. access token) predstavljen v obliki

niza znakov. Žeton mu določa različne lastnosti dostopa, kot sta nivo uporabe (angl. scope) in čas veljavnosti. Odjemalcem so omenjeni žetoni izdani s strani avtorizacijskega strežnika, ki za njihovo izdajo zahteva potrditev lastnika vira. Odjemalec lahko nato z njihovo uporabo dostopa do zavarovanih virov.

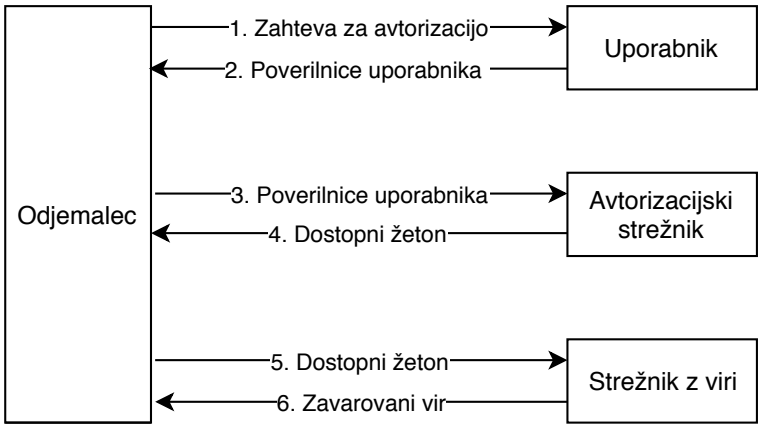
Za primer si lahko ogledamo končnega uporabnika (lastnika vira), ki lahko nadzornemu središču omogoči dostop do njegovih podatkov o imenu, priimku in elektronskem naslovu brez potrebe po deljenju njegovega uporabniškega imena in gesla računa Gmail. Namesto tega končni uporabnik dokaže pristnost strežniku, ki mu gostitelj omenjenih podatkov zaupa. Avtorizacijski strežnik odjemalcu izda dostopni žeton, ki mu omogoča le dostop do zahtevanih in odobrenih virov. V našem primeru sta oba strežnika, tako avtorizacijski kot tudi gostiteljski, pod okriljem podjetja Google.

Avtorizacijski okvir OAuth 2.0 s svojim predhodnikom OAuth 1.0 ni kompatibilen. Zgrajen je na podlagi izkušenj njegove uporabe in je namenjen njegovi zamenjavi. Obe verziji omenjenega protokola lahko sobivata na istem omrežju in nekatere aplikacije lahko podpirajo oba, pri čemer je podpora stare verzije namenjena predvsem preverjanju pristnosti uporabnikov obstoječih aplikacij.

Vloge

Avtorizacijski okvir za svoje delovanje določa naslednje štiri vloge:

- Lastnik zavarovanega vira, ki odjemalcu odobri dostop. V primeru, da je lastnik vira oseba, ga obravnavamo kot končnega uporabnika.
- Strežnik z viri, namenjen sprejemanju prejetih zahtev za dostop do zavarovanih virov, ki morajo vsebovati dostopni žeton.
- Odjemalec oziroma aplikacija, ki zahteva dostop do zavarovanega vira v imenu njegovega lastnika in z njegovo avtorizacijo. Termin odjemalec se ne nanaša na način implementacije aplikacije, ki je lahko kakršnegakoli tipa.



Slika 4.1: Abstraktni podatkovni tok avtorizacijskega okvira OAuth 2.0.

- Avtorizacijski strežnik, ki preveri pristnost lastnika vira in na podlagi njegove avtorizacije izda dostopni žeton za zahtevani vir.

Avtorizacijski strežnik in strežnik virov sta lahko ista (v našem primeru strežnika podjetja Google), lahko pa sta dve ločeni entiteti. Poleg tega lahko več različnih strežnikov z viri za generiranje dostopnih žetonov uporablja isti strežnik za preverjanje pristnosti uporabnikov.

Podatkovni tok

Podatkovni tok okvira OAuth 2.0 je prikazan na sliki 4.1. Prikazuje vrstni red in tip sporočil, prenešenih med prej opisanimi štirimi vlogami. Sestavljen je iz naslednjih korakov:

1. Odjemalčeve zahteve avtorizacije oziroma pridobitve pooblastil (angl. authorization grant) s strani lastnika zavarovanega vira. Zahteva je posredovana neposredno lastniku, kot je prikazano na sliki, ali pa, kot je bolj priporočeno, posredno preko avtorizacijskega strežnika.

2. Lastnik vira s potrditvijo zahteve odjemalcu vrne privolitev za uporabo zahtevanih pooblastil v obliki poverilnice. Tip dodeljene poverilnice je odvisen od metode za zahtevo avtorizacije in podprtih tipov poverilnic avtorizacijskega strežnika.

V tem koraku se najprej izvede preverjanje pristnosti lastnika vira in šele potem njegova avtorizacija. Njun potek je popolnoma odvisen od tipa vrnjenih poverilnic. Opis njunega poteka je zaradi različnih podatkovnih tokov izpuščen.

3. Odjemalec avtorizacijskemu strežniku pošlje zahtevo za pridobitev dostopnega žetona, ki ji kot dokaz uporabnikove avtorizacije doda prejete poverilnice.

4. Avtorizacijski strežnik preveri pristnost odjemalca in veljavnost poverilnic. V primeru njihove ustreznosti in veljavnosti strežnik odjemalcu vrne dostopni žeton.

5. Odjemalec pošlje zahtevo strežniku virov, s katero zahteva dostop do prej omenjenega zavarovanega vira in mu dokaže pristnost s priloženim dostopnim žetonom.

6. Strežnik z viri preveri ustreznost in veljavnost dostopnega žetona ter v primeru pozitivnega rezultata odjemalcu odobri dostop.

Poverilnice

Pooblastila so dodeljena v obliki poverilnic, ki so rezultat uspešne avtorizacije lastnika zavarovanega vira in odjemalcu omogočajo dostop. Dodeljene poverilnice so potrebne za odjemalčevo pridobitev dostopnega žetona. Avtorizacijski okvir v svojih določilih RFC določa naslednje 4 tipe poverilnic:

- Avtorizacijska koda, ki je pridobljena z odjemalčevim posredovanjem zahteve avtorizacijskemu strežniku namesto zahtevanja avtorizacije neposredno od lastnika vira. Odjemalec v tem primeru lastnika vira preusmeri na avtorizacijski strežnik, ki preveri pristnost lastnika in od njega pridobi avtorizacijo prejete zahteve. Po uspešnem preverjanju pristnosti in avtorizaciji uporabnika avtorizacijski strežnik odjemalcu dodeli avtorizacijsko kodo, ki je potrebna za pridobitev dostopnega žetona.

Prednost uporabe takšnega pristopa je neposredno preverjanje pristnosti lastnika vira le s strani avtorizacijskega strežnika, s čimer se izognemo potrebi po shranjevanju njegovega uporabniškega imena in gesla. Poleg tega pa omogoča tudi nekaj varnostnih izboljšav, kot je preverjanje pristnosti odjemalca in neposredno posredovanje dostopnega žetona odjemalcu brez njegove izpostavitve drugim aplikacijam. Takšen tip poverilnic je največkrat uporabljen v kombinaciji s strežniškimi spletnimi aplikacijami.

- Implicitni tip poverilnic poenostavi postopek preverjanja pristnosti in je namenjen odjemalcem, ki so implementirani za izvajanje v spletnem brskalniku ter napisani v skriptnih jezikih. Primer takšnega odjemalca predstavljajo spletne aplikacije z eno stranjo SPA (angl. Single Page Application).

  V implicitnem podatkovnem toku se odjemalcu kot rezultat uspešne preverbe pristnosti in avtorizacije lastnika vira namesto dostopne kode vrne dostopni žeton. Ob njegovi izdaji avtorizacijski strežnik ne more preveriti pristnosti odjemalca. Največ, kar lahko naredi, je, da preveri preusmeritveni naslov, na katerega bo vrnil žeton. Zavedati se moramo tudi morebitne izpostavitve dostopnega žetona spletnemu brskalniku, v katerem se izvaja odjemalec. Implicitni tip nam s svojo poenostavitvijo podatkovnega toka omogoči pohitritev na račun varnostnih tveganj.

- Poverilnice lastnika vira v obliki njegovega uporabniškega imena in gesla so lahko uporabljene za neposredno pridobitev dostopnega žetona. Njihova uporaba je priporočljiva le v primeru zelo visoke ravni zaupanja med odjemalcem in lastnikom vira. Primer takšne uporabe je odjemalec, ki je vgrajen v operacijski sistem naprave.

  Tudi z izpostavitvijo uporabniškega imena in gesla sta le-ta uporabljena le enkrat za pridobitev dostopnega žetona. Takšen način odjemalcu omogoča njuno uporabo brez potrebe po njunem shranjevanju. Za nadaljnjo avtorizacijo lahko odjemalec uporablja dostopni žeton z

nastavljenim ustreznim časom veljavnosti ali z uporabo osvežitvenega žetona.

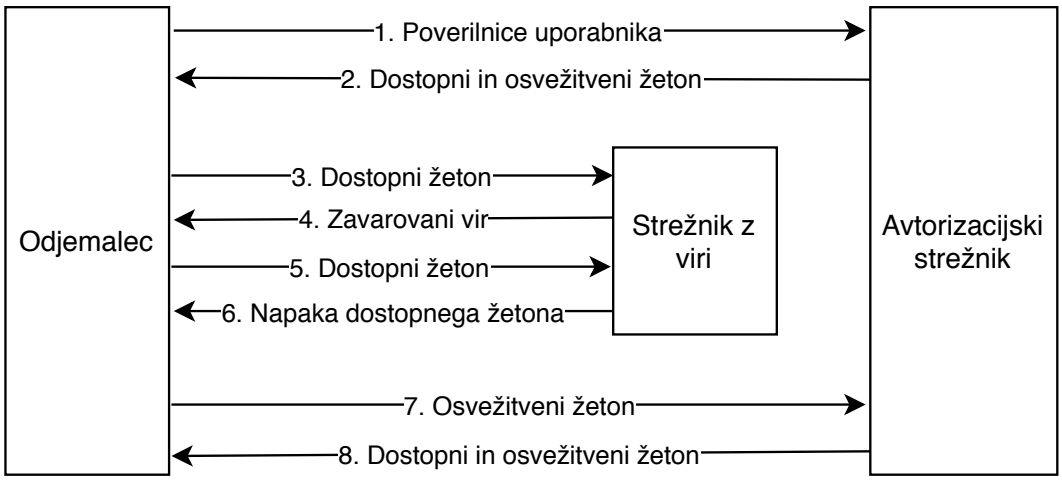
- Poverilnice odjemalca, uporabljene za dostop do virov v njegovem imenu. Tipični primer takšne uporabe poverilnic je, kadar je lastnik zahtevanega zavarovanega vira odjemalec sam ali kadar je odjemalec že vnaprej avtoriziran s strani avtorizacijskega strežnika.

Dostopni žeton

Dostopni žeton je poverilnica potrebna za dostop do zavarovanega vira. Predstavljen je kot niz znakov, ki predstavlja avtorizacijo lastnika vira posredovano odjemalcu. Njegova vsebina je odjemalcu običajno nejasna. Žeton vsebuje določen obseg pooblastil in njihov čas veljavnosti. Omenjena pooblastila so odobrena s strani lastnika vira in so upoštevana s strani avtorizacijskega strežnika ter strežnika z viri.

Dostopni žeton lahko vsebuje identifikator potreben za pridobitev podatkov o avtorizaciji ali pa že vsebuje informacije v preverljivi obliki. Sam žeton zagotavlja nov nivo abstrakcije in nadomešča različne avtorizacijske konstrukte, katerih predstavnik je uporaba uporabniškega imena in gesla. Konstrukte zamenjuje z žetonom razumljivim avtorizacijskemu strežniku. Takšen nivo abstrakcije omogoča izdajo žetonov z omejenimi pooblastili in od strežnikov z zavarovanimi viri ne zahteva poznavanja širokega nabora avtorizacijskih metod.

Žetoni so lahko zapisani v različnih formatih in strukturah. Za svojo uporabo lahko zahtevajo poznavanje različnih kriptografskih funkcij zahtevanih s strani strežnika z viri. Njihova najpogostejša oblika zapisa je tako imenovani "Bearer Token", katerega uporaba je podrobneje opisana v določilih RFC 6750 opisanih v dokumentu [15].



Slika 4.2: Podatkovni tok z vpeljavo osvežitvenega žetona okvira OAuth 2.0.

Osvežitveni žeton

Osvežitveni žeton (angl. refresh token) je poverilnica, potrebna za pridobitev dostopnega žetona. Osvežitveni žeton odjemalcu izda avtorizacijski strežnik. Namenjen je odjemalčevi pridobitvi novih dostopnih žetonov, ko obstoječi postanejo neveljavni. Z njimi lahko odjemalec pridobi tudi nove dodatne dostopne žetone z manjšim ali enakim naborom dodeljenih pooblastil za dostop do istega zavarovanega vira. Izdajanje osvežitvenih žetonov je opcijsko in neobvezno, če pa se avtorizacijski strežnik zanje odloči, so odjemalcu dodeljeni ob dodelitvi dostopnih žetonov.

Osvežitveni žeton je sestavljen enako kot dostopni žeton. Odjemalcu je predstavljen v obliki nejasnega niza znakov. Poleg tega vsebuje podobne informacije in se razlikuje le v prejemniku, saj je vedno poslan avtorizacijskemu strežniku in nikoli strežniku z viri. Postopek njegove pridobitve je podoben prej opisanemu abstraktnemu podatkovnemu toku in vsebuje popolnoma enako interakcijo z lastnikom vira, ki je zato izpuščena. Razlikuje se v načinu pridobitve dostopnega žetona, ki je prikazan na sliki 4.2 in je sestavljen iz naslednjih korakov:

1. Odjemalec avtorizacijskemu strežniku pošlje zahtevo za pridobitev do-

stopnega žetona, overjeno z odobrenimi pooblastili lastnika vira v obliki poverilnic.

2. Avtorizacijski strežnik nato preveri pristnost odjemalca ter ustreznost in veljavnost prejetih poverilnic. V primeru pozitivnega rezultata odjemalcu vrne odgovor z vsebovanim dostopnim in osvežitvenim žetonom.

3. Odjemalec uporabi dostopni žeton, ki ga pošlje strežniku z viri, skupaj z zahtevo za dostop do zavarovanega vira.

4. Strežnik z viri prejme zahtevo ter preveri ustreznost in veljavnost prejetega dostopnega žetona. V primeru pozitivnega rezultata odjemalcu odobri dostop do zahtevanega vira.

5. Koraka 3 in 4 se lahko ponavljata, dokler se dostopnemu žetonu ne izteče čas veljavnosti. Če odjemalec ve, da njegov dostopni žeton ni več veljaven, lahko nadaljuje s korakom številka 7. V nasprotnem primeru mora ponovno poslati zahtevo za dostop do zavorovanega vira.

6. Če strežnik z viri prejme neveljaven dostopni žeton, odjemalcu zavrne dostop in mu vrne sporočilo o napaki, do katere je prišlo zaradi neveljavnega žetona.

7. Odjemalec ponovno pošlje zahtevo avtorizacijskemu strežniku, ki jo overi s priloženim osvežitvenim žetonom. Zahtevana je lahko dodatna preverba pristnosti odjemalca, ki pa je odvisna od njegovega tipa in pravil avtorizacijskega strežnika.

8. Avtorizacijski strežnik preveri pristnost odjemalca ter ustreznost in veljavnost osvežitvenega žetona. V primeru pozitivnega rezultata odjemalcu izda dostopni in opcijsko tudi nov osvežitveni žeton.

TLS

Za povezavo odjemalca s strežnikom se največkrat uporablja varni protokol HTTPS, ki za svojo zaščito uporablja varni transportni nivo TLS (angl.

Transport Layer Security). Varnostne ranljivosti avtorizacijskega okvirja OAuth so zato povezane tudi z uporabljeno različico nivoja TLS in njegovimi ranljivostmi.

Preusmeritve HTTP

Avtorizacijski okvir za svoje delovanje uporablja preusmeritve HTTP, s katerimi odjemalec in avtorizacijski strežnik vodita lastnika vira. Uporabljeni spletni brskalniki preusmeritve največkrat izvedejo na podlagi statusne kode HTTP 302.

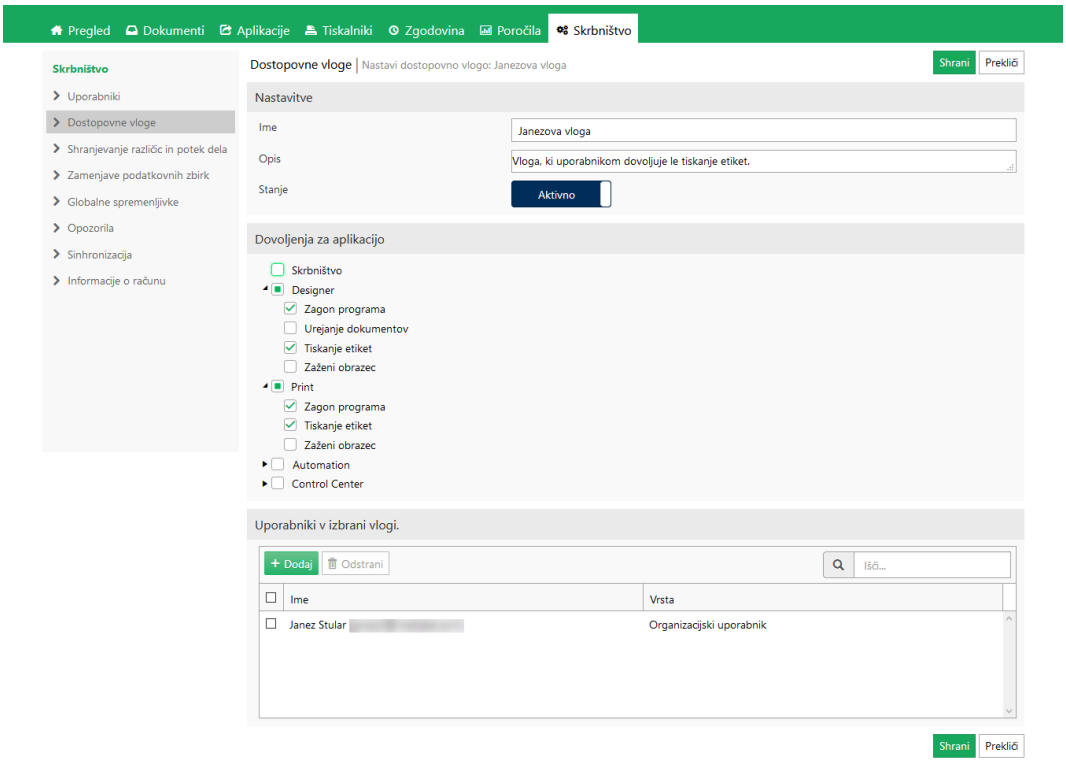
4.3 Vloge dostopa

Nadzorno središče skrbnikom omogoča nadzor in urejanje vlog dostopa, namenjenih dodelitvi uporabniških pravic, kot je opisano v delu [16]. Vsaka vloga dostopa ima v Nadzornem središču svoje enolično ime in je lahko aktivna ali onemogočena. V primeru uporabnikove pripadnosti onemogočeni vlogi mu v njej dodeljene pravice niso upoštevane. Vloge dostopa so sestavljene iz dveh glavnih delov.

Prvi del predstavljajo pravice za uporabo posameznih funkcionalnosti programov, s čimer so njenim pripadnikom omogočene le minimalne funkcionalnosti sistema, ki so zahtevane za opravljanje njihovega dela. S tem so zmanjšana tveganja za izvajanje nepooblaščenih akcij ter preprečene nekatere morebitne napake delovanja in zlorabe sistema.

Drugi pomemben del vlog dostopa predstavljajo pripadajoči uporabniki, ki so lahko gostujoči ali pa so člani povezanega aktivnega imenika AAD. Primer vloge dostopa z uporabniki in njihovimi pravicami je prikazan na sliki 4.3.

Ustvarjene aktivne vloge dostopa uporabljajo vsi programi, povezani z Nadzornim središčem. Takšna povezava od uporabnikov ob uporabi programov zahteva njihovo prijavo v Nadzorno središče. Povezani programi od Nadzornega središča najprej pridobijo potrebne podatke o identiteti uporab-



Slika 4.3: Prikaz vloge dostopa.

nika nato pa še nabor njegovih pravic. Na podlagi pridobljenih pravic se programi zaženejo z ustreznim naborom omogočenih funkcionalnosti.

4.4 Spletne aplikacije

Spletne aplikacije so namenjene deljenju etiket in rešitev po meri, shranjenih v shrambi dokumentov. Deljene so lahko znotraj sistema najemnika Nadzornega središča ter z njegovimi zunanjimi sodelavci in izvajalci storitev. Uporaba spletnih aplikacij je popolnoma vsebinskega pomena in zato vsebuje dodaten nadzor dostopa.

Nastavitve spletne aplikacije so sestavljene iz treh glavnih delov: nastavitev same aplikacije, nabora avtoriziranih uporabnikov in omejitev uporabe. Nastavitve aplikacije zahtevajo vnos njenega enoličnega imena in izbor upo-

rabljene etikete ali rešitve po meri. Poleg omenjenih dveh polj pa omogočajo tudi hrambo poljubnega opisa. Drugi del predstavlja nabor avtoriziranih uporabnikov. Zadnji oziroma tretji del pa predstavljajo naslednje omejitve uporabe:

- Aplikacija je lahko aktivna in dostopna ali pa je njeno izvajanje uporabnikom onemogočeno.
- Hranjenje zgodovine tiskanja je lahko vključeno ali izključeno, kar omogoča uporabo in testiranje aplikacij v razvoju.
- Uporaba aplikacije je lahko omejena na podlagi internetnih naslovov IP, s čimer je omogočen dostop le izbranim napravam posameznikov ali vsem napravam iz omrežja nekega podjetja.
- Omogočena je vključitev omejitve tiskalnikov, s čimer se podjetja v primeru delitve spletne aplikacije z zunanjimi izvajalci lahko izognejo njihovi uporabi vseh licenc tiskalnikov.
- Omogočena je nastavitev vrednosti spremenljivk, uporabljenih na izbrani etiketi ali rešitvi.
- Omogočena je lahko uporaba neobjavljenih datotek iz shrambe dokumentov, kar je priporočljivo le za aplikacije v razvoju. V produkcijskem okolju je priporočljivo uporabljati le objavljene in potrjene različice dokumentov.

Poglavje 5

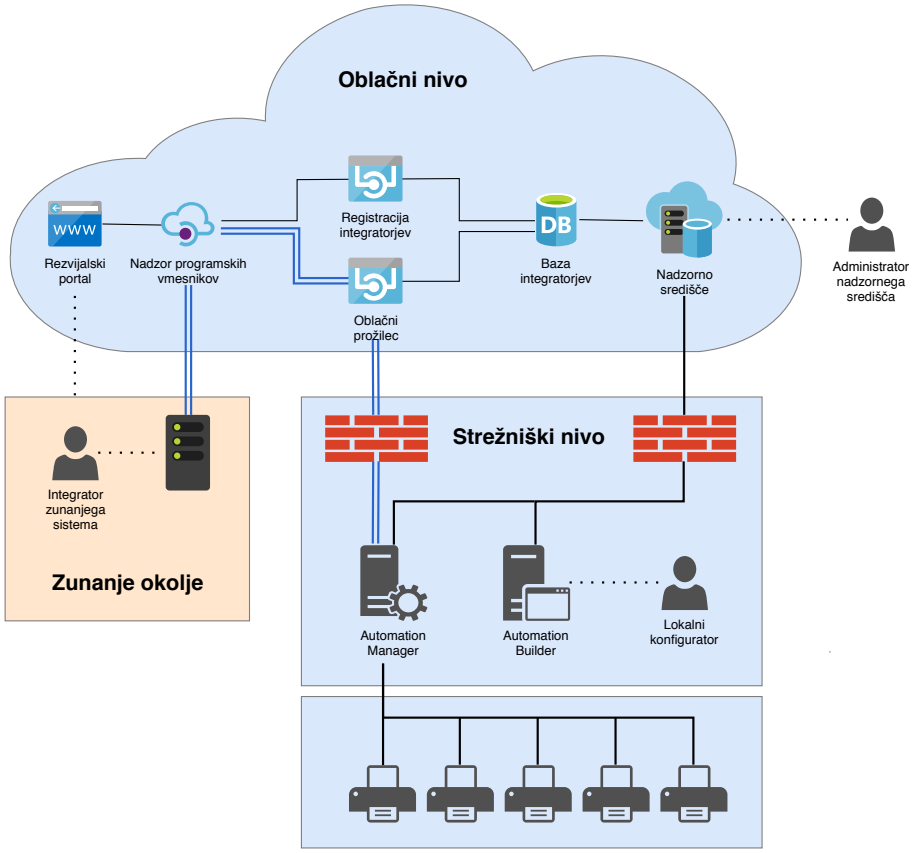
Oblačni prožilec

Oblačni prožilec je namenjen neposrednemu povezovanju in integraciji drugih tovrstnih sistemov in storitev s sistemom za upravljanje z etiketami. Zunanjim integratorjem omogoča enostavno in varno povezavo, poleg tega pa skrbnikom sistema za upravljanje z etiketami omogoča nadzor nad njimi.

5.1 Athitektura in podatkovni tokovi

Oblačne storitve so zaradi nudenja programske opreme kot storitve SaaS in njihovega načina delovanja implementirane po principu večnajemniške arhitekture (angl. multi tenant arhitecture), ki je podrobneje opisana v delu [5].

Večnajemniška arhitektura predstavlja nekaj prednosti predvsem ponudnikom storitev. Omogoča nam centralni nadzor ter boljšo izkoriščenost in učinkovitost ponujenih storitev. V primeru nizkega števila zahtev za neko storitev je v izvajanju lahko le ena njena instanca, namenjena vsem uporabnikom. Izvajanje ločenih instanc za vsakega uporabnika ni potrebno. Poleg tega lahko v času povečanja števila zahtev po tej storitvi enostavno povečamo število njenih instanc v izvajanju, ki jih lahko ob zmanjšanju zahtev ponovno ukinemo. Prednost ponudbe istih instanc večjemu številu najemnikom v primerjavi s ponudbo ločenih instanc vsakemu izmed najemnikov je



Slika 5.1: Arhitektura oblačnega prožilca.

v njihovi enakomerni obremenitvi. Z enakomerno obremenitvijo je dosežena večja učinkovitost posameznih instanc in krajši odzivni čas ponujene storitve.

Implementacija oblačnih storitev sistema za upravljanje z etiketami temelji na zgoraj opisani večnajemniški arhitekturi. V našem primeru predstavlja enega naročnika storitev kot enega najemnika. Celotna arhitektura oblačnega prožilca zajema dva nivoja sistema za upravljanje z etiketami ter njegovo povezavo s drugimi tovrstnimi sistemi, kot prikazuje slika 5.1. Sestavlja jo naslednjih sedem delov:

- Program Automation Builder, ki je namenjen urejanju nastavitev oblačnega prožilca. Njegovi glavni parametri so namenjeni nastavitvi načina komunikacije in vključujejo enolični identifikator prožilca. Nastavitve

poleg omenjenih parametrov vsebujejo tudi nabor akcij, ki jih bo prožilec izvedel ob aktivaciji.

- Automation Manager, povezan z Nadzornim središčem, ki mu nudi informacije o najemniku. Naloga Automation Managerja je izvajanje predhodno urejenih nastavitev oblačnega prožilca. Izvajanje prožilca vključuje komunikacijo programa Automation Manager z oblačnim programskim vmesnikom API. Način njune komunikacije temelji na uporabi komunikacijskih storitev WCF (angl. Windows Communication Foundation), opisanih v delu [17]. Za uspešno izmenjavo njunih sporočil storitev WCF zahteva uporabo pogodbe, ki predpisuje njihovo obliko in vsebino. Pogodbo za povezavo programa Automation Manager in oblačnega programskega vmesnika API prikazuje programska koda 5.1.

```
[ServiceContract(Name = "ICloudTriggerService")]
public interface ICloudTriggerService
{
    [OperationContract]
    HttpTriggerResponseData CallTrigger(HttpTriggerRequestData request);
}
```

Izsek kode 5.1: Uporabljena pogodba prožilca.

Izvajanje in aktivacija oblačnega prožilca na strani programa Avtomation Manager temeljita na uporabi te pogodbe. Programskemu vmesniku API je na voljo v naslovnem prostoru najemnika in se nahaja na naslovu, ki ga določa njegov enolični identifikator. Funkcijo za zagon in izpostavitev oblačnega prožilca s strani programa Automation Manager prikazuje izsek programske kode 5.2.

- Nadzorno središče, povezano s programom Automation Manager in podatkovno bazo integratorjev. Njegova naloga v primeru oblačnega prožilca je skrb za registracijo integratorjev in nadzor njihovega dostopa. Aktivacijo oblačnega prožilca lahko izvede le integrator, ki je dodan v Nadzorno središče in je uspešno izvedel postopek registracije. Postopek dodajanja integratorja v Nadzorno središče generira enolični ključ, ki

```
public override void Start()
{
    // Pridobitev potrebnih podatkov za izpostavitev prozilca.
    string customerUrl = this.GetCustomerUrl();
    CloudTriggerServiceBusData sbData;
    try
    {
        sbData = this.GetServiceBusData().Result;
    }
    catch (AggregateException ae)
    {
        // Posredovanje izvirne napake pridobivanja podatkov.
        throw ae.InnerException;
    }

    // Dolocitev ustreznih nastavitev vodila in dolocitev naslova prozilca.
    ServiceBusEnvironment.SystemConnectivity.Mode = ConnectivityMode.Tcp;
    Uri sbUri = ServiceBusEnvironment.CreateServiceUri("sb", sbData.Namespace,
        $"{sbData.TriggerPath}/{customerUrl}/{this.CloudTrigger.TriggerId}");

    // Kreiranje storitve prozilca in nastavitev funkcije za obdelavo zahteve.
    CloudTriggerService cloudTriggerService = new CloudTriggerService(req =>
        this.ProcessRequest(req, this.cloudTrigger.WaitForTriggerExecution));
    this.cloudTriggerServiceHost = new ServiceHost(cloudTriggerService, sbUri);

    // Kreiranje ustreznih nastavitev izpostavitve.
    IEndpointBehavior serviceRegistrySettings = new
        ServiceRegistrySettings(DiscoveryType.Public);
    foreach (ServiceEndpoint endpoint in
        this.cloudTriggerServiceHost.Description.Endpoints)
    {
        endpoint.Behaviors.Add(serviceRegistrySettings);
        endpoint.Behaviors.Add(sbData.sasCredential);
    }

    // Izpostavitev prozilca.
    this.cloudTriggerServiceHost.Open();
}
```

Izsek kode 5.2: Zagon in izpostavitev oblačnega prožilca.

ga skupaj s potrebnimi podatki o najemniku shrani v podatkovno bazo integratorjev.

- Podatkovna baza integratorjev, ki je namenjena hranjenju podatkov za povezavo integratorjev in najemnikov Nadzornega središča.

- Nadzor programskih vmesnikov Azure API Management. Namenjen je nadzorovanju in posredovanju zahtev zunanjih sistemov oblačnim programskim vmesnikom API. Podrobneje je opisan v delu [18]. Ciljna skupina njegovih uporabnikov so integratorji zunanjih sistemov, ki jim je za delo s programskimi vmesniki API na voljo razvojni portal. Portal je namenjen upravljanju z naročninami (angl. subscription) integratorjev in preizkusu produktov z vsebovanimi programskimi vmesniki

API. Uporaba naročnine s strani integratorjev je obvezna, saj vključuje ključe, potrebne za aktivacijo programskih vmesnikov API. Azure API Management z zahtevano uporabo ključev preprečuje neposredne napade z zavračanjem storitev DoS (angl. Denial of Service). Zaradi njegove vloge posrednika zahtev programskim vmesnikom API omogoča tudi omejitev števila njihovih aktivacij in s tem nadzor nad uporabo virov.

- Programski vmesnik za registracijo integratorja. Njegova naloga je shranitev informacij o naročnini integratorja, sklenjene na razvijalskem portalu Azure API Management, poleg enoličnega ključa integratorja, generiranega v Nadzornem središču. Klic programskega vmesnika se zgodi preko Azure API Managementa, ki je nastavljen tako, da poleg podatkov, zagotovljenih s strani uporabnika, posreduje tudi njegovo naročniško številko. Integrator mora zato ob klicu obravnavanega programskega vmesnika API zagotoviti le njegov enolični ključ z Nadzornega središča. Vmesnik preveri obstoj prejetega ključa v podatkovni bazi integratorjev in poleg njega zapiše naročniško številko integratorja. S tem je pridobljena enolična povezava integratorja in najemnika Nadzornega središča.

- Programski vmesnik oblačnega prožilca, namenjen izpostavitvi prožilca zunanjemu svetu. Njegov klic je ponovno mogoče izvesti preko Azure API Managementa, od katerega poleg ostalih podatkov prejme tudi naročniško številko integratorja. Integrator mora za uspešno aktivacijo prožilca posredovati njegov enolični identifikator in podatke za obdelavo. Programski vmesnik preveri vsebovanost identifikatorja in v podatkovni bazi integratorjev na podlagi naročniške številke poišče podatke o naslovnem prostoru najemnika Nadzornega središča. Na podlagi naslovnega prostora in enoličnega identifikatorja prožilca nato izvede njegovo aktivacijo na strežniškem nivoju, kot prikazuje izsek programske kode 5.3

```
private static async Task<HttpResponseMessage> ProcessTrigger(HttpRequestMessage
    request, string customerUrl, string triggerId, TraceWriter log)
{
    // Pridobitev podatkov vodila in dolocitev naslova prozilca.
    CloudTriggerServiceBusData sbData = this.GetServiceBusData();
    Uri sbUri = ServiceBusEnvironment.CreateServiceUri("sb", sbData.Namespace,
        $"{sbData.TriggerPath}/{customerUrl}/{triggerId}");

    try
    {
        // Nastavitev tipa komunikacije in najvecje velikosti sporocila.
        NetTcpRelayBinding tcpBinding = new NetTcpRelayBinding();
        tcpBinding.MaxReceivedMessageSize = int.MaxValue;
        tcpBinding.MaxBufferSize = int.MaxValue;

        // Kreiranje komunikacijskega kanala za povezavo s strezniskim nivojem
        ChannelFactory<ICloudTriggerChannel> channelFactory = new
            ChannelFactory<ICloudTriggerChannel>(tcpBinding , new
            EndpointAddress(sbUri));
        channelFactory.Endpoint.Behaviors.Add(sbData.sasCredential);
        ICloudTriggerChannel channel = channelFactory.CreateChannel();

        log.Info("Zacetek poslusanja na vodilu.");
        channel.Open();

        try
        {
            try
            {
                // Transformacija prejete zahteve in aktivacija prozilca
                HttpTriggerRequestData requestData = await
                    this.GetDataFromRequest(request);
                HttpTriggerResponseData responseData = null;
                await Task.Run(() =>
                {
                    responseData = channel.CallTrigger(requestData);
                });

                // Generiranje odgovora na podlagi rezultata prozilca
                return this.CreateResponse(responseData);
            }
            catch (Exception ex)
            {
                // Napaka pri aktivaciji prozilca
                log.Error("Error while calling trigger service.", ex);
                return request.CreateErrorResponse(
                    HttpStatusCode.InternalServerError, "Could not connect to
                    trigger.");
            }
        }
        finally
        {
            // Zapiranje komunikacijskega kanala
            channel.Close();
            channelFactory.Close();
        }
    }
    catch (Exception ex)
    {
        // Napaka pri vzpostavitvi povezave s strezniskim nivojem
        log.Error($"Error while setting up trigger '{customerUrl/triggerId}'.",
            ex);
        return request.CreateErrorResponse(HttpStatusCode.InternalServerError,
            $"Could not connect to trigger '{triggerId}'.");
    }
}
```

Izsek kode 5.3: Klic in aktivacija oblačnega prožilca.

5.2 Uporaba

Za celoten postopek integracije zunanjega sistema in vzpostavitev delovanja oblačnega prožilca morajo sodelovati osebe s tremi različnimi vlogami: lokalni konfigurator avtomatizacije procesa tiskanja, administrator Nadzornega središča in integrator zunanjega sistema. Postopek vključuje registracijo integratorja ter konfiguracijo in uporabo oblačnega prožilca.

5.2.1 Registracija integratorja

Prvi del predstavlja registracija integratorja in je v primeru uporabe večjega števila oblačnih prožilcev enega najemnika Nadzornega središča potrebna le ena. V primeru uporabe prožilcev različnih najemnikov pa je postopek registracije treba ponoviti za vsakega najemnika. Opis ter potek posameznih korakov je naslednji:

1. Administrator Nadzornega središča mora najprej integratorju zunanjega sistema odobriti dostop, kar stori z dodajanjem nove integracije v Nadzorno središče. Postopek dodajanja generira in vrne enolični ključ, namenjen povezavi novega integratorja.

2. Integrator zunanjega sistema se mora nato prijaviti v razvojni portal. V primeru, da je to njegova prva integracija in še nima kreiranega svojega uporabniškega računa, mora iti skozi postopek registracije.

3. Na razvojnem portalu prijavljen integrator lahko vidi produkte, ki so mu na voljo. Trenutno je izpostavljen le oblačni prožilec, ki ga izbere. Za izdelavo nove integracije mora najprej ustvariti naročnino (angl. subscription), ki vsebuje dva ključa, potrebna za klic programskih vmesnikov API.

4. Po uspešno ustvarjeni naročnini se integrator vrne med produkte, ponovno izbere oblačni prožilec in nato Cloud API v1. Razvojni portal mu prikazuje primere klicev izpostavljenih programskih vmesnikov

API. Za uspešen zaključek postopka registracije in povezave integratorja z Nadzornim središčem mora integrator izvesti klic programskega vmesnika za registracijo. Klic mora vsebovati poizvedovalni niz HTTP (angl. HTTP query string), ki vključuje polje "integratorKeyž vrednostjo ključa Nadzornega središča in glavi HTTP: "Api-Versionž vrednostjo "v1"ter "Ocp-Apim-Subscription-Keyž vrednostjo enega izmed naročniških ključev. Uspešna aktivacija programskega vmesnika za registracijo integratorja je razvidna iz stanja prejetega odgovora HTTP 200.

5. Administrator nadzornega središča lahko preveri stanje registracije novega integratorja.

5.2.2 Nastavitev in aktivacija oblačnega prožilca

Po uspešni registraciji integratorja mora konfigurator avtomatizacije procesa tiskanja pripraviti ustrezno nastavljen oblačni prožilec, ki ga lahko integrator nato aktivira. Ta postopek je sestavljen iz naslednjih dveh korakov:

1. Konfigurator avtomatizacije procesa tiskanja mora z uporabo programa Automation Builder posodobiti obstoječo nastavitev prožilcev ali ustvariti novo. Dodati ji mora oblačni prožilec, ki vsebuje ime in opis. Za njegovo osnovno delovanje so potrebne nastavitve komunikacije, ki zahtevajo določitev enoličnega identifikatorja, na katerega se sklicuje integrator ob svoji aktivaciji. Nastavitve komunikacije omogočajo enosmerno in dvosmerno izmenjavo sporočil. V primeru enosmerne izmenjave sporočil prožilec le čaka in se izvede ob njegovi aktivaciji. V primeru dvosmerne izmenjave sporočil pa oblačni prožilec po uspešni izvedbi akcij integratorju vrne tudi odgovor s potrditveno kodo HTTP 200. Poslanemu odgovoru lahko konfigurator prožilca nastavi glave HTTP in vsebino.

   Poleg nastavitev komunikacije mora konfigurator prožilcu nastaviti še akcije za izvedbo, ki so tipično namenjene obdelavi prejetih podatkov

in tiskanju etiket. Konfiguracijo avtomatizacije mora nato naložiti v program Automation Manager in tam zagnati prožilec.

2. Predhodno registriran integrator zunanjega sistema lahko zagnan oblačni prožilec aktivira s posredovanjem ustrezne zahteve HTTP. Zahteva mora vsebovati glavo HTTP z naročniškim ključem in mora biti poslana na ustrezen naslov, kot prikazuje naslednji primer:

```
POST https://company.com/api/CloudTrigger/prozilec42
Host: company.com
Api-Version: v1
Ocp-Apim-Subscription-Key: ●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●
```

Poglavje 6

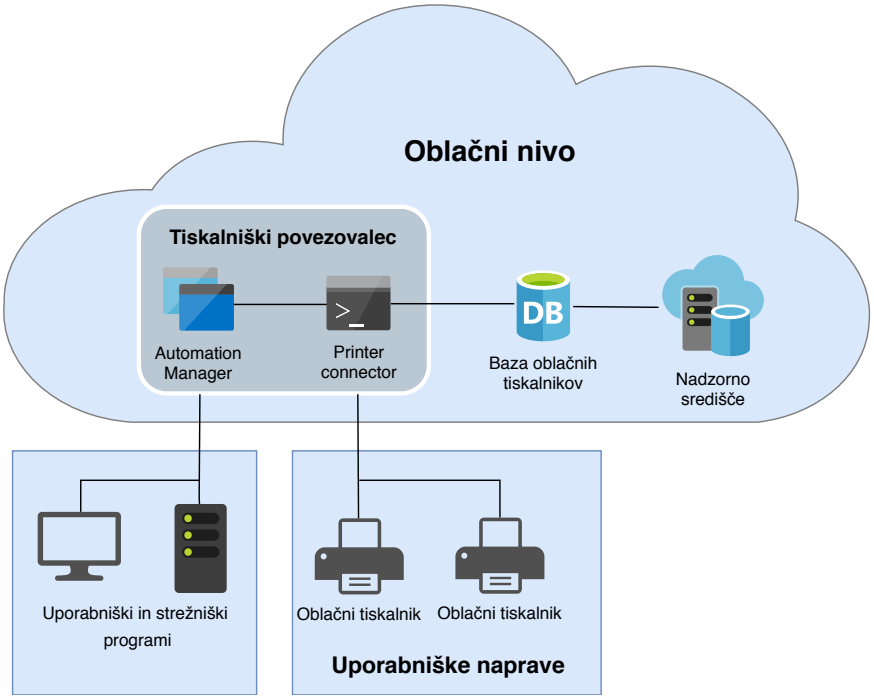
Oblačni tiskalniški povezovalec

Oblačni tiskalniški povezovalec je namenjen neposrednemu povezovanju tiskalnikov z oblačno storitvijo za tiskanje etiket, s čimer sledi sodobnim trendom interneta stvari IoT. Cilj trenda je povezava naprav z internetnim omrežjem ter možnost njihove enolične identifikacije. Napravam sta s tem omogočena predvsem medsebojno povezovanje ter izpostavitev različnim oblačnim storitvam. Prednosti povezave naprav z oblačnimi storitvami so podrobneje opisane v delu [6].

6.1 Arhitektura in podatkovni tokovi

Naloga oblačnega tiskalniškega povezovalca je vzpostavitev in vzdrževanje varne povezava tiskalnika s sistemom za upravljanje z etiketami. Uporabnikom omogoča tiskanje etiket brez potrebe po nameščanju tiskalniških gonilnikov. Zaradi svoje vloge je oblačni tiskalniški povezovalec prisoten na nivoju oblačnih storitev in uporabniških naprav, kot je prikazano na sliki 6.1. Njegovo arhitekturo sestavljajo naslednje štiri komponente:

- Nadzorno središča, ki je namenjeno registraciji tiskalnika. Registracija je pomembna predvsem zaradi zagotavljanja kontrole dostopa, kot je opisano v delu [7], in določitve tipa tiskalnika. Pri registraciji mora uporabnik izbrati model tiskalnika in vnesti njegov enolični identifika-



Slika 6.1: Arhitektura oblačnega tiskalniškega povezovalca.

tor, ki je določen s strani proizvajalca. Omenjena podatka se tekom registracije zapišeta v podatkovno bazo oblačnih tiskalnikov.

- Podatkovna baza oblačnih tiskalnikov, ki je namenjena hranjenju podatkov o tiskalnikih.

- Povezovalec, ki je namenjen povezavi tiskalnika z drugimi tovrstnimi storitvami in ki omogoča tiskanja etiket brez uporabe tiskalniških gonilnikov. Tiskalniški povezovalec mora biti zaradi svoje vloge povezan z dvema entitetama internetnega omrežja: oblačnim tiskalnikom in oblačnimi storitvami. Poleg njune povezave mora zagotavljati tudi nadzor dostopa uporabnikov ter tiskanje etiket. Za omogočanje navedenih storitev mora biti povezovalec povezan s podatkovno bazo oblačnih tiskalnikov. Podatki, zapisani v bazi, oblačnemu povezovalcu omogočajo zavrnitev zahtev neregistriranih tiskalnikov in generiranje toka tiskanja

(angl. print stream).

- Oblačni tiskalnik, povezan z internetnim omrežjem, katerega naloga je le varna komunikacija s sistemom za upravljanje z etiketami. Povezava mu omogoča tiskanje zahtevanih etiket.

Tiskalniški povezovalec je bil razvit kot inovacijski projekt. Njegov razvoj se je zaradi trenutnega pomanjkanja zanimanja končal pri dobro načrtovanem in implementiranem prototipu. Glavni del prototipa je implementiran z uporabo odprtokodnega programskega okvira ASP.NET Core. Omogoča nam izvajanje napisanih programov, neodvisno od uporabljene platforme gostitelja (angl. cross platform software). Ciljno okolje izvajanja prototipa predstavljajo vsebovalniki programa Docker.

Implementacija prototipa tiskalniškega povezovalca je sestavljena iz dveh glavnih delov: programa za povezavo tiskalnika PrinterConnector in programa Automation Manager, ki izvaja ustrezno konfiguracijo oblačnega prožilca.

Program PrinterConnector

Naloga programa PrinterConnector je vzpostavitev, vzdrževanje in nadzor povezave s tiskalnikom ter obdelava zahtev, prejetih s strani oblačnega prožilca programa Automation Manager. Programska koda programa PrinterConnector je, abstraktno gledano, sestavljena iz petih med seboj povezanih enot.

Osnovo programa predstavlja spletni strežnik, ki gosti spletni vtič, na katerega se povezuje tiskalnik. Njegov način delovanja je predpisan z določilom RFC 6455, ki ga je izdala organizacija IETF in je podrobneje opisan v dokumentu [19]. Spletni vtič programa PrinterConnector posluša na vratih 30443 in za svojo identifikacijo uporablja certifikat TLS. Vtič ves čas čaka na prejem zahteve tiskalnika. Ob prejemu zahteve najprej preveri njen tip in ustreznost zahtevanega protokola. Če je zahteva ustrezna, jo sprejme in izvede proces rokovanja s tiskalnikom, ki nadgradi klasično povezavo WS (angl.

Web Socket) v varno povezavo WSS (angl. Web Socket Secure) ob uporabi protokola TLS, ki je opisan v delu [20]. Rokovanje in nadgradnja povezave sta izvedena z uporabo zunanje knjižnice. Naslednja naloga spletnega strežnika je vzdrževanje pravkar vzpostavljene povezave in posredovanje prejetih zahtev delu programa, ki je namenjen njihovi obdelavi.

Drugi del programa PrinterConnector predstavlja preslikovalec kanalov, ki je namenjen vodenju nabora odprtih kanalov za vsak povezan tiskalnik. Njegova prisotnost je potrebna zaradi načina komunikacije tiskalnika s programom PrinterConnector in vsebuje 3 različne vrste kanalov. Prvi je tako imenovan glavni kanal, namenjen predstavitvi tiskalnika, prejemanju obvestil in odpiranju dodatnih dveh vrst kanalov. Prva vrsta so tako imenovani surovi kanali (angl. raw channels), ki so namenjeni pošiljanju ukazov tiskalniku, napisanih v njegovem programskem jeziku. Drugi vrsta pa so konfiguracijski kanali (angl. configuration channels), namenjeni pošiljanju konfiguracijskih ukazov, ki so zapisani v konfiguracijskem jeziku tiskalnika. Vsebina poslanih zahtev po vseh omenjenih kanalih je predstavljena v obliki zapisa JSON (angl. JavaScript Object Notation).

Izvajanje programa se nadaljuje z njegovim tretjim delom – obdelovalcem zahtev tiskalnika. Tiskalnik po uspešno izvedeni nadgradnji povezave pošlje prvi vsebinski paket po glavnem kanalu in se z njim predstavi (angl. discovery packet). Njegova vsebina je zapisana v formatu Base64 in vsebuje osnovne podatke o tiskalniku, ki vključujejo njegov enolični identifikator in model. Primer predstavitvenega sporočila je naslednji:

```
{
    "discovery_b64" : "OiwuBAQBAAFaQlIAAEQwSjE3NDgwMzQyNAA..."
}
```

Vsebina predstavitvenega paketa je pomembna za nadaljnje vodenje preslikav odprtih komunikacijskih kanalov. V slovar preslikav kanalov se glavni kanal tiskalnika shrani pod ključem, ki ga določa dekodirana vsebina predstavitvenega paketa. Ob prejemu morebitnih podvojenih predstavitvenih

paketov in poskusu ponovne vzpostavitve povezave program PrinterConnector zapre vse odprte komunikacijske kanale in prekine obstoječe povezave. V primeru prejema le enega predstavitvenega paketa pa program PrinterConnector v tretjem sestavnem delu po glavnem kanalu pošlje zahtevo za odprtje surovega kanala. Obdelava predstavitvenega paketa je prikazana v naslednjem izseku kode 6.1.

```
private async Task ProcessInitialRequest(string receivedData)
{
    PrinterHandler.logger.LogInformationWithTimestamp($"Process request from
        printer: {receivedData}");
    Dictionary<string, string> responseJson =
        JsonConvert.DeserializeObject<Dictionary<string, string>>(receivedData);
    string discovery64Content = string.Empty;

    // Preverjanje obstoja predstavitvenega sporocila.
    if (responseJson.TryGetValue("discovery_b64", out discovery64Content))
    {
        PrinterHandler.logger.LogInformationWithTimestamp($"Initial MAIN channel
            request");
        this.channelType = ChannelType.Main;

        // Dekodiranje vsebine sporocila in izlocitev kontrolnih znakov.
        string encoded64 = discovery64Content.Split(':')[0];
        byte[] data = Convert.FromBase64String(encoded64);
        string receivedDiscoveryContent = Encoding.UTF8.GetString(data);
        this.DiscoveryContent = new string(receivedDiscoveryContent.Where(c =>
            !char.IsControl(c)).ToArray());

        // Preverjanje obstoja obstojecih povezave.
        PrinterHandler existingMainChannel;
        if (ChannelMappingService.Instance.MainChannels.TryGetValue(
            this.DiscoveryContent, out existingMainChannel))
        {
            string message = $"Main channel for printer {this.DiscoveryContent}
                is already opened. Existing channels will be closed and new will
                be opened.";
            await this.CloseSocketWithInvalidPayloadData(message);
            return;
        }

        ChannelMappingService.Instance.MainChannels.Add(this.DiscoveryContent,
            this);

        // Odpiranje surovega kanala.
        OpenChannelPacket rawChannel = new
            OpenChannelPacket(ChannelName.RawChannelName);
        string command = JsonConvert.SerializeObject(rawChannel);
        await this.SendResponseAsync(command);
    }
    else
    {
        // Zavrnitev neveljavnega predstavitvenega paketa.
        string message = $"Unexpected main channel request for printer
            {this.DiscoveryContent}";
        await this.CloseSocketWithInvalidPayloadData(message);
        return;
    }
}
```

Izsek kode 6.1: Obdelava predstavitvenega paketa tiskalnika.

Tiskalnik ob prejeti zahtevi za odpiranje surovega kanala ponovno sproži celoten postopek vzpostavitve povezave s spletnim strežnikom in ponovno pošlje prvi vsebinski paket. Tokrat paket ne vsebuje predstavitvenega sporočila, temveč ime kanala, njegovo identifikacijsko številko in enolični identifikator tiskalnika. Primer prvega vsebinskega paketa surovega kanala je naslednji:

```
{
    "unique_id" : "●●●●●●●●●●●●",
    "channel_name" : "v1.raw.company.com",
    "channel_id" : "31"
}
```

Ob prejemu omenjenega paketa program PrinterConnector preveri enolični identifikator tiskalnika v slovarju preslikav kanalov. Slovar mora vsebovati natanko en glavni kanal, shranjen pod ključem, ki vsebuje prejeti enolični identifikator tiskalnika. Del programa, odgovoren za preslikavo kanalov, nato združi glavni in surovi kanal. Njuno preslikavo shrani pod ključem, ki ga predstavlja enolični identifikator tiskalnika. Pridobitev preslikave in združitev kanalov sta prikazana v izseku programske kode 6.2. Tiskalniku nato tretji del programa preko surovega kanala pošlje ukaz za prijavo na obvestila. Poleg tega pa že začne z izvajanjem četrtega dela strežnika tiskanja, namenjenega komunikaciji s programom Automation Manager.

Strežnik tiskanja za svoje delovanje uporablja strežnik HTTP, zaradi česar potrebuje peti del programa, imenovan preslikovalec strežnikov tiskanja. Preslikovalec je namenjen vodenju zagnanih strežnikov HTTP in zagotavljanju izvajanja največ enega strežnika HTTP za vsak povezan tiskalnik. Goščeni naslovi omenjenih strežnikov temeljijo na enoličnem identifikatorju tiskalnika.

Strežnik tiskanja pred zagonom novega strežnika HTTP pri preslikovalcu strežnikov vedno preveri zasedenost zahtevanega naslova. V primeru njegove uporabe strežnik tiskanja zaustavi obstoječi strežnik HTTP in na istem naslovu zažene novega. Naloga zagnanih strežnikov HTTP je zelo preprosta in zahteva le sprejemanje sporočil s strani programa Automation Manager in posredovanje njihove vsebine tiskalniku po surovem kanalu.

```
public static ChannelMapping GetChannelMapping(string serialNumber)
{
    // Iskanje obstojece preslikavo kanalov.
    ChannelMapping channelMapping;
    if (ChannelMappingService.Instance.channelMapping.TryGetValue( serialNumber,
        out channelMapping))
    {
        return channelMapping;
    }

    // Iskanje vsebino predstavitvenega paketa z vsebovano serijsko stevilko.
    string discoveryContent =
        ChannelMappingService.Instance.mainChannels.Keys.Where(x =>
        x.Contains(serialNumber)).FirstOrDefault();

    // Preverjanje obstoja vsebine paketa.
    if (!string.IsNullOrEmpty(discoveryContent))
    {
        string[] decodedParts = discoveryContent.Split(' ');
        string printerName = decodedParts[1];

        // Pridobitev ustreznega glavnega kanala.
        PrinterHandler mainChannel =
            ChannelMappingService.Instance.mainChannels[discoveryContent];
        mainChannel.SerialNumber = serialNumber;
        mainChannel.DiscoveryContent = null;

        // Ustvarjanje nove preslikave kanalov, ki temelji na enolicnem
            identifikatorju tiskalnika.
        channelMapping = new ChannelMapping(mainChannel, serialNumber,
            printerName);
        ChannelMappingService.Instance.mainChannels.Remove(discoveryContent);
        ChannelMappingService.Instance.channelMapping.Add(serialNumber,
            channelMapping);

        return channelMapping;
    }

    return null;
}
```

Izsek kode 6.2: Pridobitev in združitev preslikav tiskalniških kanalov.

Med vsem tem časom se izvaja tudi tretji del programa, zadolžen za obdelavo prejetih zahtev tiskalnikov, ki ves čas prejema obvestila s tiskalnika. Prejeta obvestila zapisuje v podatkovno bazo tiskalnikov za namen kasnejše analize.

Program tekom svojega delovanja ves čas uporablja tudi svoj šesti del – zapisovalec dnevnika, ki beleži ključne korake izvajanja. Dnevnik je namenjen odkrivanju in odpravljanju napak programa, do katerih lahko pride med njegovim izvajanjem. Spremljanje ključnih dogodkov nam omogoča dva načina njihovega izpisa. Prvi je namenjen razvijanju programa in dnevnik izpisuje v okno konzole, drugi pa je namenjen produkcijskemu delovanju programa in dnevnik zapisuje v dve ločeni datoteki. Ločitev dnevniških datotek

nam omogoča nadzor nad njihovo velikostjo, saj jim lahko omejimo število zapisov na datoteko. Zapisovalec dnevnika ob doseženi omejitvi vrstic prve datoteke začne s prepisovanjem druge datoteke in s tem zagotavlja konstanten obstoj vsaj polovice zapisov. Prekop dnevniških datotek prikazuje izsek programske kode 6.3.

```
public bool LogFileSwitched()
{
    lock (this.activeFileName)
    {
        // Zapis vsebine v datoteko in preverjanje stevila zapisov.
        this.logStreamWriter.Flush();
        if (this.lineCounter++ > this.maxLines)
        {
            // Zamenjava dnevniske datoteke.
            if (this.activeFileName == this.fileName0)
            {
                this.SetLogFile(this.fileName1);
            }
            else
            {
                this.SetLogFile(this.fileName0);
            }

            return true;
        }

        return false;
    }
}

private void SetLogFile(string fileName)
{
    // Zapiranje obstojece datoteke in podatkovnih tokov.
    this.CloseStreams();

    // Kreiranje nove datoteke in podatkovnih tokov.
    this.activeFileName = fileName;
    this.logFileStream = new FileStream(fileName, FileMode.Create);
    this.logStreamWriter = new StreamWriter(this.logFileStream);

    lineCounter = 0;
}
```

Izsek kode 6.3: Menjava dnevniških datotek tiskalniškega povezovalca.

Konfiguracija prožilcev

Oblačni tiskalniški povezovalec začne opravljati svojo vlogo šele z zagotavljanjem možnosti oddaljenega tiskanja etiket, zaradi česar mora biti izpostavljen na internetnem omrežju. Za njegovo varno izpostavitev je uporabljen program Automation Manager, povezan z Nadzornim središčem najemnika.

Programa PrinterConnector in Automation Manager se za namene prototipa izvajata na istem virtualnem računalniku ponudnika oblačnih storitev Microsoft Azure. Razlog za njuno združitev je možnost komunikacije znotraj lokalnega omrežja računalnika, s čimer se izognemo potrebi po neposredni izpostavitvi programa PrinterConnector. Proces tiskanja etiket je omogočen s klicem oblačnega prožilca, ki ga izvaja Automation Manager. Za njegovo uspešno aktivacijo je treba vsebino sporočila napisati v strukturnem jeziku XML. Sporočilo mora vsebovati nabor zahtev tiskanja. Vsaka izmed zahtev pa mora vsebovati naslednje podatke: ime, pod katerim je registriran oblačni tiskalnik, pot do etikete za tiskanje, število natisnjenih kopij in vrednosti spremenljivk, ki se uporabljajo na etiketi.

Naloga programa Automation Manager je izvajanje dveh prožilcev: oblačnega in datotečnega. Naloga prvega je izpostavitev oblačnega tiskalniškega povezovalca internetnemu omrežju ter generiranje toka tiskanja (angl. print stream), ki ga shrani v datoteko. Naloga drugega prožilca pa je branje datoteke in posredovanje njene vsebine ustreznemu strežniku tiskanaja, ki je del programa PrinterConnector. Aktivacija oblačnega prožilca povzroči izvedbo naslednjih glavnih akcij:

1. Branje prejetega sporočila XML in shranjevanje prejetih vrednosti v spremenljivke prožilca.

2. Pregled vseh prejetih zahtev tiskanja, za katerimi sledi izvedba nadaljnih akcij.

3. Povezava s podatkovno bazo in izvedba skripte za pridobitev enoličnega identifikatorja in modela tiskalnika.

4. Izvedba skripte za generiranje imena datoteke za hranjenje toka tiskanja. Ime je odvisno od enoličnega identifikatorja tiskalnika in enoličnega globalnega identifikatorja UUID (angl. Universally Unique Identifier).

5. Odpiranje ustrezne etikete za tiskanje.

6. Nastavitev tiskalnika in preusmeritev toka tiskanja v datoteko s prej določenim imenom.

7. Generiranje in shranjevanje toka tiskanja obdelovanih etiket.

Izvedba zgoraj opisanih akcij ustvari datoteko na določeni lokaciji, ki jo spremlja datotečni prožilec. Ob njenem nastanku se posledično aktivira datotečni prožilec. Ločitev prožilcev nam omogoča lažje razhroščevanje, saj lahko vsakega izmed prožilcev ločeno zaustavimo. S tem povzročimo aktivacijo le enega prožilca in preverimo njegove rezultate. Nastavitev datotečnega prožilca je sestavljena iz naslednjih glavnih akcij:

1. Branje globalnih spremenljivk, ki hranijo pot datoteke proženja.

2. Izvedba skripte, ki iz poti datoteke proženja pridobi enolični identifikator tiskalnika in ga shrani v spremenljivko prožilca.

3. Izvedba skripte, ki prebere vsebino datoteke proženja, jo zakodira v format Base64 in shrani v spremenljivko prožilca.

4. Povezava s podatkovno bazo in izvedba skripte za pridobitev naslovne poti strežnikov HTTP, ki so del programa PrinterConnector.

5. Pošiljanje zahteve HTTP ustreznemu strežniku programa PrinterConnector. Njegov naslov prožilec določi na podlagi pridobljene naslovne poti in enoličnega identifikatorja tiskalnika. Vsebino poslane zahteve pa prebere iz enkodirane spremenljivke prožilca.

S pošiljanjem zahteve strežniku HTTP datotečni prožilec programu PrinterConnector pošlje generiran enkodiran tok tiskanja, ki je zapisan v programskem jeziku povezanega tiskalnika. Program PrinterConnector prejeti tok nato dekodira in posreduje tiskalniku, ki ga izvede.

6.2 Uporaba

Uporaba tiskalniškega povezovalca je sestavljena iz dveh delov: registracije oblačnega tiskalnika in izvedbe tiskanja. Nastavitev prožilcev ter skrb za njihovo izvajanje je najemniku storitve zagotovljena s strani ponudnika povezovalca. Zagotovljen mu je enolični identifikator oblačnega prožilca ter spletni naslov strežnika, na katerega mora povezati oblačne tiskalnike. Najemniku je za tiskanje etiket zagotovljena rešitev po meri. Najemnik oziroma njegov uporabnik le registrira oblačni tiskalnik in prične s tiskanjem.

6.2.1 Registracija tiskalnika

Oblačni tiskalnik je treba najprej registrirati v Nadzornem središču. Registracija zahteva le izbiro modela tiskalnika, vnos njegovega enoličnega identifikatorja in določitev poljubnega imena. Potreben enolični identifikator in model tiskalnika lahko uporabnik pridobi z izpisom njegovih nastavitev. Po uspešni registraciji mora uporabnik tiskalniku nastaviti še spletni naslov oblačnega tiskalniškega povezovalca, na katerega se tiskalnik povezuje. Nekateri proizvajalci za aktivacijo povezovanja zahtevajo še ponovni zagon tiskalnika in s tem je njegova registracija zaključena.

6.2.2 Izvedba tiskanja

Za izvedbo tiskanja je najemniku oblačnega tiskalniškega povezovalca zagotovljena aplikacija v obliki rešitve po meri. Njen izgled in funkcionalnost sta zelo podobna klasičnemu oknu za tiskanje v programu Designer. Rešitev se razlikuje le v tem, da poleg izbire tiskalnika zahteva tudi izbiro etikete, ki mora biti shranjena v shrambi dokumentov. Lokacija etiket je pomembna zaradi njihove dostopnosti oblačnemu tiskalniškemu povezovalcu. Poleg tega pa uporaba shrambe dokumentov že vključuje tudi nadzorovan dostop do tiskanih etiket.

Poglavje 7

Uporabljeni programerski vzorci

Uporaba preverjenih programerskih vzorcev nam omogoča implementacijo preglednejše in ponovno uporabljive programske kode. Olajša nam njeno vzdrževanje ter omogoča hitrejši in cenejši odziv na zahtevane spremembe in prilagoditve.

Pri implementaciji oblačnih razširitev, ki predstavljajo glavni del magistrske naloge, sta bila uporabljena dva arhitekturna in trije načrtovalski vzorci. Največja razlika med omenjenima vrstama arhitekturnih vzorcev je njuna vloga in pogostost njune izbire. Arhitekturne vzorce največkrat izberemo le enkrat pred začetkom implementacije aplikacije, na izbiro načrtovalskih vzorcev pa moramo biti pozorni ves čas njene implementacije.

7.1 Arhitekturni vzorci

Vsak program, ki vključuje interakcijo s končnim uporabnikom, potrebuje uporabniški vmesnik. Njegova združitev z delovanjem domenske logike predstavlja večni inženirski problem, ki je lahko ublažen z uporabo izbranih arhitekturnih vzorcev, povzetih po delu [21].

Arhitekturni vzorci nam omogočajo implementacijo generičnih rešitev za pojavljajoče se arhitekturne težave. Njihov razvoj se je začel že v osemdesetih letih s prvo implementacijo vzorca model-pogled-nadzornik MVC (angl.

Model-View-Controller), narejeno v jeziku Smalltalk. Zaradi njegove pomanjkljivosti pri implementaciji namiznih aplikacij so se tekom časa razvili tudi novi vzorci podobnega tipa. Njihova največja predstavnika sta model-pogled-pogled modela MVVM (angl. Model-View-ViewModel) in model-pogled-predstavitelj MVP (angl. Model-View-Presenter). Arhitekturni vzorci se z razvojem novih tehnologij spreminjajo in s tem nastajajo celotne družine njihovih predstavnikov z majhnimi vendar pomembnimi razlikami.

Glavna ideja arhitekturnih vzorcev omenjenega tipa je ločitev nalog in odgovornosti posameznih delov aplikacije. Prva glavna naloga je delitev aplikacije na dva nivoja: načrt uporabniškega vmesnika in domensko logiko aplikacije. S tem je zagotovljena čista struktura, ki je enostavnejša za vzdrževanje, omogoča pa nam tudi lažjo delitev dela med razvijalci, ki lahko delajo na različnih nivojih aplikacije, ter poenostavi testiranje delovanja domenske logike.

Danes imamo zaradi pestrosti arhitekturnih vzorcev na voljo pestro izbiro. Izbira ustreznega arhitekturnega vzorca je ključnega pomena za razumevanje delovanja in lažje vzdrževanje aplikacije. Uporaba napačnega ali nobenega arhitekturnega vzorca za integracijo uporabniškega vmesnika z implementacijo domenske logike nas lahko vodi do kompleksne programske kode. Drobne spremembe izgleda uporabniškega vmesnika lahko zahtevajo ogromno spremembo kode, potrebne za njegovo sinhronizacijo z implementacijo domenske logike. Izbira ustreznega vzorca pa ni bila del magistrske naloge. Potrebna je bila predvsem njegova pravilna uporaba in razumevanje delovanja. V nadaljevanju sta opisana uporabljena arhitekturna vzorca MVC in MVVM. Vzorec MVC je uporabljen pri implementaciji spletnih aplikacij, vzorec MVVM pa pri implementaciji namiznih aplikacij sistema za upravljanje z etiketami.

7.1.1 Vzorec MVC

Arhitekturni vzorci MVC, namenjeni sinhronizaciji uporabniškega vmesnika s stanjem aplikacije, veljajo za najvplivnejše na svojem področju. V letih okrog 1980, kamor segajo njihovi začetki, so bili vzorci MVC namenjen načrtovanju

in gradnji bogatih uporabniških vmesnikov namiznih aplikacij, kot je opisano v knjigi [9]. Tekom časa ter ob pojavu novih tehnologij in potreb pa so se razvile nove oblike. Namen današnjih oblik vzorcev MVC ni več enak, saj so namenjeni predvsem integracijam uporabniškega vmesnika z implementacijo domenske logike. Prav tako se tudi ne uporabljajo več za razvoj namiznih aplikacij, ampak predvsem za razvoj spletnih in mobilnih aplikacij.

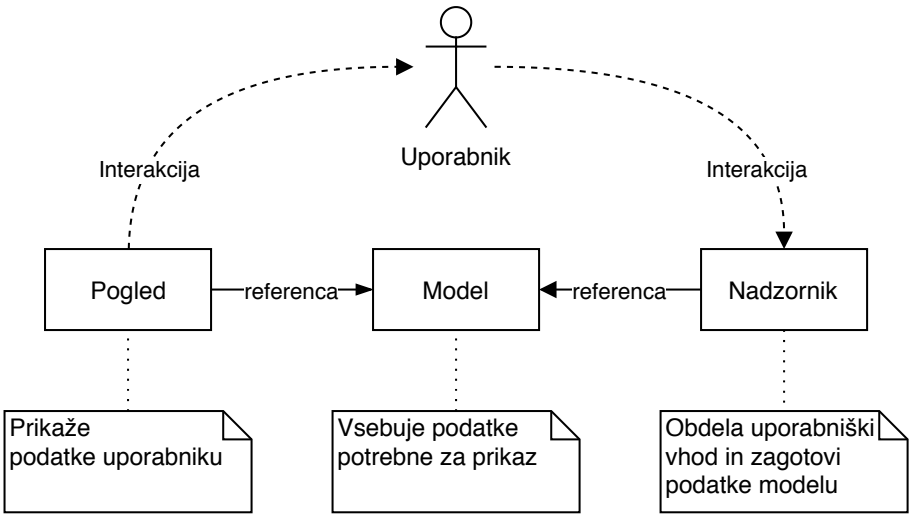
Pri razvoju Nadzornega središča sistema za upravljanje z etiketami je bila uporabljena spletna različica arhitekturnega vzorca ASP.NET MVC, ki je opisana v nadaljevanju. Nanjo se navezujejo tudi vse nadaljnje omembe vzorca MVC v tem poglavju.

Okolje izvajanja spletnih aplikacij je ločeno na dva dela: prikaz spletne strani na strani odjemalca ter obdelavo zahtev na strani strežnika. Zaradi ločitve okolja spletne aplikacije že v zasnovi sovpadajo z idejo vzorca MVC, tj. ločitev implementacije domenske logike od uporabniškega vmesnika. Ločitev je dosežena z delitvijo spletne aplikacije na tri sestavne dele s predpisanimi nalogami. Prvi je model, namenjen hranjenju domenskih podatkov; drugi je nadzornik, namenjen obdelavi odjemalčevih zahtev; tretji pa je pogled, namenjen generiranju uporabniškega vmesnika. Okolje izvajanja spletnih aplikacij samo po sebi že podpira ločitev odgovornosti pogleda in nadzornika, saj se podatki prikazujejo v obliki spletne strani na strani odjemalca, obdelava zahtev pa se izvaja na strani strežnika. Spletne strani uporabniku največkrat prikazujejo vsebino, ki je rezultat obdelave podatkov, narejene s strani strežnika.

Struktura

Struktura arhitekturnega vzorca MVC je prikazana na sliki 7.1 in je sestavljena iz naslednjih delov in njihovih nalog:

- Model, namenjen hranjenju podatkov, ki so potrebni za prikaz. Vsebuje lahko podatke domenskih entitet in podatke, potrebne za ustrezen izris posameznih komponent pogleda.



Slika 7.1: Struktura arhitekturnega vzorca MVC.

- Pogled, namenjen prikazu uporabniškega vmesnika. Temelji na podlagi pripravljenih podatkov, shranjenih v modelu. Uporabniku je prikazan v obliki spletnih strani, napisanih v jeziku HTML (angl. Hyper Text Markup Language) oziroma njegovi Microsoftovi različici spletnih strani Razor, napisanih v jeziku CSHTML.

- Nadzornik, namenjen obdelavi uporabnikovih zahtev. Odgovoren je za klic in izvedbo ustrezne aplikacijske logike, komunikacijo s podatkovnimi bazami ter pripravo ustreznih podatkov, potrebnih za kreiranje modela.

Uporabnik z vnosom naslova spletne aplikacije in njegove potrditve aktivira ustreznega nadzornika. Nadzornik na podlagi prejetih podatkov izvede zahtevane akcije in zgradi model, ki ga posreduje pogledu. Pogled vzame model in zgradi uporabniški vmesnik, ki je prikazan uporabniku.

Primer

Za lažjo predstavo si poglejmo primer uporabnika, ki išče informacije o novo izdanem albumu svoje najljubše glasbene skupine. Za pridobitev teh informa-

cij uporabnik v spletnem brskalniku obišče spletno stran Glasbena knjižnica. V njen iskalnik izvajalcev vnese ime glasbene skupine in klikne na gumb išči. S tem strežniku pošlje zahtevo z vnešenimi podatki. Strežnik prejme zahtevo in na podlagi naslova aktivira ustrezno akcijo nadzornika. Nadzornikova akcija na podlagi imena glasbene skupine pridobi seznam njenih albumov, ustvari nov model in vanj shrani pridobljen seznam. Nadzornik ta model nato posreduje pogledu. Pogled prejme model in na podlagi vsebovanih podatkov v modelu zgradi uporabniški vmesnik ter ga prikaže uporabniku v obliki spletne strani.

Posledice uporabe

Uporaba arhitekturnega vzorca MVC prinaša naslednje prednosti, opisane v krovnem članku [21] in na spletnem mestu [22]:

- Vzorec nam predpisuje logično delitev posameznih komponent aplikacije. Definicija uporabniškega vmesnika pripada pogledu, obdelava odjemalčevih zahtev in izvedba potrebne logike pripada nadzorniku, predstavitev in nadzor domenskih podatkov pa pripada modelu.

- Z ločitvijo uporabniškega vmesnika od ostalih komponent spletne aplikacije nam arhitekturni vzorec omogoča lažje testiranje posameznih delov.

- Zaradi opuščenega neposrednega stanja aplikacije nam omogoča večji nadzor nad njenim delovanjem, saj lahko na podlagi prejete zahteve predvidimo potek njene obdelave.

7.1.2 Vzorec MVVM

Arhitekturni vzorci tipa MVVM so se razvili zaradi pomanjkljivosti izvirnih arhitekturnih vzorcev MVC. Namenjeni so bili implementaciji bogatih uporabniških vmesnikov namiznih aplikacij. Njihova največja pomanjkljivost je slabo upravljanje stanja aplikacije in nezmožnost interakcije zgolj na podlagi

uporabniškega vnosa. Primer takšne interakcije predstavlja vnosno polje, namenjeno finančnim poročilom. Njegova vrednost mora biti v primeru vnosa negativnega števila obarvana rdeče v nasprotnem primeru pa črno. Posledično so se razvili arhitekturni vzorci MVVM, ki so danes v veliki večini namenjeni implementaciji namiznih in mobilnih aplikacij.

Za razvoj uporabniških programov sistema za upravljanje z etiketami je bil uporabljen arhitekturni vzorec Microsoft MVVM v kombinaciji z grafičnim prikazovalcem WPF (angl. Windows Presentation Foundation). Vzorec Microsoft MVVM je podrobneje opisan v nadaljevanju. Nanj se navezujejo tudi vse nadaljne omembe vzorca MVVM v tem poglavju.

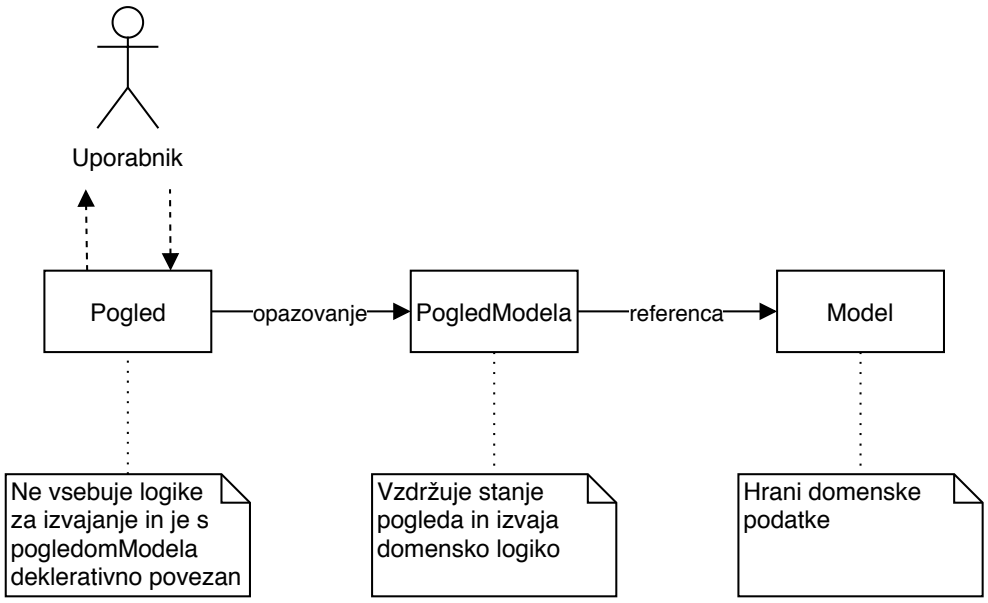
Namizne aplikacije se od spletnih razlikujejo že v okolju izvajanja. Izvajajo se na strani odjemalca in so odgovorne za vse dele od prikaza uporabniškega vmesnika do obdelave uporabniških in domenskih podatkov. Osnovna ideja vzorca MVVM je enaka kot pri vzorcu MVC: ločitev uporabniškega vmesnika od implementacije domenske logike. Aplikacijo prav tako razdeli na tri glavne dele. Prvi je model, namenjen hranjenju domenskih podatkov; drugi je pogled modela, namenjen upravljanju stanja in uporabniški interakciji; tretji pa pogled namenjen prikazu uporabniškega vmesnika.

Poleg nalog in odgovornosti vzorec MVVM vsakemu pogledu omogoča povezovanje le s pripadajočim pogledomModela, medtem ko vsakemu pogleduModela omogoča povezovanje z več pogledi. S tem podpira možnost prikaza enakih podatkov na več različnih načinov hkrati.

Struktura

Arhitektura vzorca MVVM je popolnoma linearna, kot prikazuje slika 7.2. Sestavljena je iz naslednjih delov in njihovih nalog:

- Model, namenjen hranjenju domenskih podatkov s popolnim nezavedanjem o obstoju pogledaModela.
- Pogled modela, namenjen upravljanju stanja pogleda in uporabniški interakciji. Njegova naloga je tudi spremljanje in obdelava domenskih



Slika 7.2: Struktura arhitekturnega vzorca MVVM.

podatkov, shranjenih v modelu, in izvedba ustrezne aplikacijske logike. Podobno kot se model ne zaveda obstoja pogleda modela, se tudi pogled modela ne zaveda obstoja pogleda.

- Pogled, namenjen prikazu uporabniškega vmesnika, implementiranega v jeziku XAML (angl. Extensible Application Markup Language). Njegova naloga poleg prikaza je vzdrževanje enosmernega sklica na pogled modela, ki služi njegovemu opazovanju, spreminjanju lastnosti ter aktivaciji njegovih akcij.

O vsaki spremembi modela je obveščen pogled modela in o vsaki spremembi pogleda modela je obveščen pogled. Ob uporabniški interakciji s pogledom je največkrat neposredno obveščen tudi pogled modela, ki na podlagi njegovih dejanj izvede ustrezne akcije in po potrebi spremeni model.

Primer

Za lažje razumevanje delovanja arhitekturnega vzorca MVVM si poglejmo primer aplikacije Finančno poročilo. Njena naloga je prikaz dveh različnih pogledov: tabelaričnega in grafičnega.

Ob vsaki spremembi opazovanih finančnih podatkov se spremeni model. Njegovo spremembo zazna pogled modela, ki od modela pridobi seznam podatkov o posameznih imetnikih računov in njihovih stanjih. Pridobljene podatke pogledaModela shrani v svojo lastnost, imenovano "Računi". Pogled modela opazujeta dva pogleda: prvi je namenjen izrisu tabele drugi pa izrisu grafa. Oba pogleda na podlagi spremembe lastnosti "Računi"na svoj način spremenita svoj izgled. Prvi izriše tabelo s podatki o imetnikih računov ter njihovih stanjih, drugi pa na podlagi stanj izriše graf s pripadajočo legendo.

Posledice uporabe

Uporaba arhitekturnega vzorca MVVM prinaša naslednje prednosti, opisane v krovnem članku [21] in na spletnem mestu [22]:

- Ločitev pogleda od pogledaModela in modela omogoča razširjeno testiranje enot (angl. unit testing) pogledaModela in modela brez uporabe pogleda.

- Linearna arhitektura vzorcu MVVM omogoča gradnjo različnih pogledov, ki temeljijo na enakem naboru podatkov, in njihovo sočasno sinhronizacijo.

- Možnost implementacije pogleda izključno v jeziku XAML omogoča enostavno spreminjanje uporabniškega vmesnika brez potrebe po spreminjanju druge programske kode.

- Dvosmerna vezava podatkov pogleda in pogledaModela. Omogočena je s pogledovim opazovanjem pogledaModela in neposrednim spreminjanjem njegovih lastnosti.

- Zasnova arhitekturnega vzorca je pripravljena za delo s tehnologijami WPF in Silverlight na platformah .NET ter uporabi njihovih programskih jezikov.

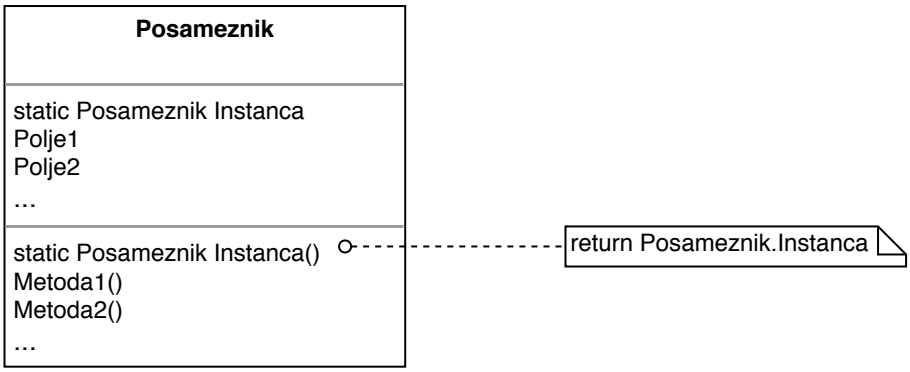
Ob uporabi načrtovalskega vzorca MVVM se moramo zavedati tudi nekaterih njegovih slabosti. Največkrat izpostavljeni sta njegova odvisnost od uporabljene tehnologije ter oteženo razhroščevanje kompleksnejših vezav med pogledom in pogledomModela.

7.2 Načrtovalski vzorci

Dobro načrtovanje programske kode je že samo po sebi zahtevno delo, ki postane še zahtevnejše z namenom njene večkratne uporabe. Za lažjo izvedbo načrtovanja si lahko pomagamo z uporabo različnih načrtovalskih vzorcev, katerih poglavje magistrske naloge je povzeto po knjigi [9].

Za dobro načrtovanje večkrat uporabljene programske kode je treba zaznati njene ključne entitete in jih predstaviti v obliki razredov. Treba jim je določiti vmesnike in medsebojne relacije. Načrt programske kode mora biti po eni strani dovolj specifičen za obravnavani problem, po drugi strani pa dovolj splošen, da omogoča njeno ponovno uporabo. Izogibati se mora potrebi po ponovnemu načrtovanju in prepisovanju programske kode ob uvajanju predvidenih sprememb. Implementacija takšnega načrta kode je zelo zahtevna in največkrat vključuje nekaj poskusov uporabe in popravkov implementacije.

Načrtovalski vzorci omogočajo ponovno uporabo uspešno implementiranih načrtov in arhitektur programske kode. Poznavanje dobrih praks njihove uporabe pa nam omogoča lažje in boljše načrtovanje aplikacij. Pomagajo nam pri izbiri primernih vzorcev, ki omogočajo ponovno uporabo kode, in pri izogibanju neprimernim, ki nas pri tem omejujejo. Uporaba primernih načrtovalskih vzorcev z opisom implementiranih razredov in njihovih interakcij izboljša tudi razumljivost dokumentacije ter olajša vzdrževanje. Poenostavljeno nam izbira načrtovalskih vzorcev omogoča lažjo in hitrejšo pridobitev



Slika 7.3: Struktura načrtovalskega vzorca posameznik.

ustreznega načrta.

7.2.1 Posameznik

Pri nekaterih razredih je pomembno, da imajo ustvarjeno le eno instanco. Podobno je že pri samih operacijskih sistemih, kjer sistem teče na enem datotečnem sistemu in uporablja enega upravljalca za prikaza oken.

Zagotoviti je treba, da ima takšen razred ustvarjeno le eno enostavno dostopno instanco. Globalne spremenljivke nam omogočajo dostopnost, ampak ne preprečujejo kreiranja večjega števila instanc. Zato je najbolje, da je razred sam odgovoren za nadzor števila svojih instanc. Le-ta lahko zagotavlja kreiranje le ene instance in omogoča enostaven način dostopa.

Uporaba

Načrtovalski vzorec posameznik (angl. singleton) je priporočljivo uporabiti v primeru, ko potrebujemo natanko eno instanco nekega razreda, ki mora biti dostopna več delom projekta. Poleg tega nam omogoča tudi njeno razširitev z dedovanjem brez potrebe po spreminjanju osnove programske kode.

Struktura

Struktura omenjenega načrtovalskega vzorca je zelo preprosta in je prikazana na sliki 7.3. Vsebuje le en razred z lastnostjo Instanca, ki njenim klicateljem omogoča dostop do edine ustvarjene instance razreda. Omenjena lastnost je poleg izpostavitve lahko odgovorna tudi za kreiranje instance.

Klicatelj lahko do instance razreda tipa posameznik posledično dostopa izključno z uporabo te lastnosti.

Posledice uporabe

Uporaba načrtovalskega vzorca posameznik nam ponuja naslednje prednosti:

- Nadzorovan dostop do edine instance razreda. Razred ima sam nadzor nad njo, s čimer lahko drugim klicočim razredom omogoči ali prepreči dostop.
- Omogoča izboljšavo svojih operacij in svoje predstavitve. Razred tipa posameznik je lahko razširjen z dedovanjem. Podedovanemu razredu lahko enostavno nastavimo dodatne lastnosti, ki jih lahko spreminjamo tudi med delovanjem programa.
- Omogoča enostavno spremembo števila ustvarjenih instanc. Če si tekom razvoja premislimo in ugotovimo, da bi namesto ene potrebovali natanko N instanc razreda, lahko to enostavno storimo.

  Poleg tega lahko uporabimo tudi enak pristop za nadzor števila uporabljenih instanc. V tem primeru moramo spremeniti le metodo, ki dovoljuje dostop do njih.
- Predstavitev je mogoče implementirati tudi z uporabo statične funkcije Instanca.

Implementacija

Primer uporabe načrtovalskega vzorca posameznik predstavlja preslikovalec tiskalniških kanalov, ki je del programa PrinterConnector. Njegova glavna

naloga je vodenje referenc odprtih tiskalniških kanalov, potrebnih za njihovo sinhronizacijo, odpiranje in zapiranje. Implementacija preslikovalca kanalov je prikazana v izseku programske kode 7.1.

```
class ChannelMappingService
{
    private static ILogger logger =
        LoggingService.GetLogger<ChannelMappingService>();

    private static ChannelMappingService instance = new ChannelMappingService();

    private Dictionary<string, PrinterHandler> mainChannels = new
        Dictionary<string, PrinterHandler>();

    private Dictionary<string, ChannelMapping> channelMapping = new
        Dictionary<string, ChannelMapping>();

    // Izpostavitev edine instance razreda
    public static ChannelMappingService Instance =>
        ChannelMappingService.instance;

    public Dictionary<string, PrinterHandler> MainChannels => this.mainChannels;

    public Dictionary<string, ChannelMapping> ChannelMapping =>
        this.channelMapping;

    public static async Task<bool> CanOpenChildChannel(WebSocket webSocket,
        CancellationToken token, string serialNumber, Func<string, bool>
        isPrinterSerialRegistred) {...}

    public static ChannelMapping GetChannelMapping(string serialNumber) {...}
}
```

Izsek kode 7.1: Preslikovalec tiskalniških kanalov.

7.2.2 Prilagojevalnik

Razredi raznovrstnih knjižnic so običajno namenjeni večkratni uporabi in razširitvam, kar žal ni vedno enostavno, ker njihovi vmesniki (angl. interface) največkrat niso združljivi z domensko specifičnimi aplikacijami.

Za lažje razumevanje potreb si poglejmo primer orodja za risanje. Omogoča nam urejanje osnovnih elementov, kot so črta, poligon in besedilo, ki so potrebni za načrtovanje slik in diagramov. Urejevalnikova abstrakcija osnovnih elementov je grafični objekt, ki lahko prilagodi obliko in se zna izrisati. Osnovni elementi izhajajo iz vmesnika imenovanega Oblika, ki ga dedujejo. Za lažje razumevanje so po njem tudi poimenovani: OblikaČrta, OblikaPoligon, OblikaBesedilo itd.

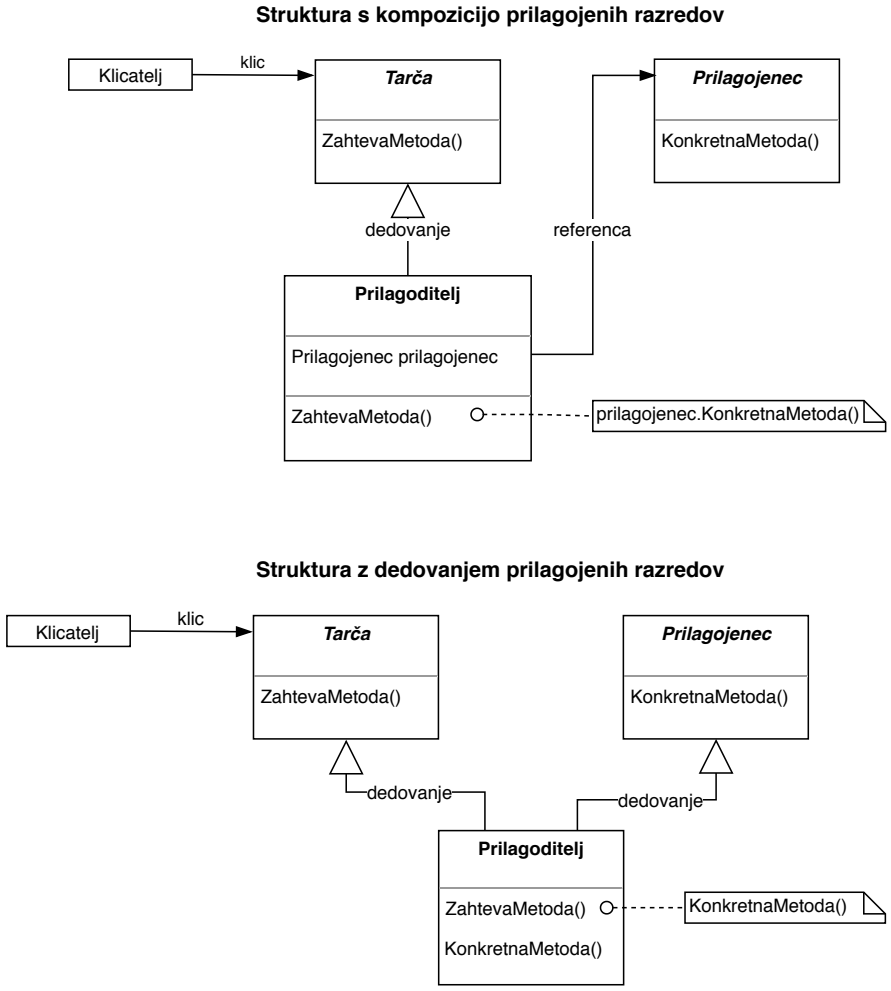
Implementacija razredov OblikaČrta in OblikaPoligon je enostavna, saj sta njuna oblika in izris precej omejena. OblikaBesedilo pa je ravno nasprotje, ker osnovna implementacija že vsebuje kompleksnejše osveževanje in hranjenje vsebine. V času implementacije orodja za risanje lahko obstaja idealna knjižnica, ki vsebuje dovršeno implementacijo razreda za element besedilo imenovano PogledBesedilo. Do težave pride pri njeni uporabi. Razred PogledBesedilo ni bil načrtovan z uporabo vmesnika Oblika in v orodju za risanje ne more biti neposredno uporabljen.

Naloga načrtovalskega vzorca prilagojevalnik (angl. adapter) je združitev razredov z različnimi vmesniki brez spreminjanja njihove implementacije. Lahko bi spremenili razred PogledBesedilo, da bi se prilegal vmesniku Oblika, ampak to ni mogoče, če nimamo njegove izvorne kode. Poleg tega naloga razredov knjižnic ni njihovo prilagajanje domenskim lastnostim.

Namesto tega lahko ustvarimo nov razred, ki prilagodi vmesnik razreda PogledBesedilo tako, da ustreza vmesniku Oblika. To lahko storimo na dva načina: z dedovanjem vmesnika Oblika in razreda PogledBesedilo ali z vključitvijo instance razreda PogledBesedilo v razred OblikaBesedilo. Tako nastali razred OblikaBesedilo imenujemo prilagojevalnik. Pogosto je odgovoren za implementacijo funkcionalnosti, ki jih vključeni razred ne vsebuje. V našem primeru bi to lahko bila funkcionalnost za izris in premik besedila.

Uporaba

Načrtovalski vzorec prilagojevalnik je priporočljivo uporabiti v treh primerih. Prvi je, ko želimo uporabiti obstoječi razred, ki ne ustreza našim potrebam, saj ne vsebuje vseh potrebnih funkcionalnosti; drugi je, ko želimo ustvariti ponovno uporabljiv razred, ki je združljiv z različnimi nepoznanimi vmesniki; in zadnji je, ko moramo uporabiti množico obstoječih razredov z različnimi vmesniki in dedovanje vseh vmesnikov enostavno ni praktično. V tem primeru razred tipa prilagojevalnik deduje le od korenskega vmesnika uporabljenih razredov.



Slika 7.4: Strukturi načrtovalskega vzorca prilagojevalnik.

Struktura

Osnovni strukturi programerskega vzorca prilagojevalnik sta prikazani na sliki 7.4, ki je sestavljena iz naslednjih štirih elementov:

- Tarča, ki določa domensko specifičen vmesnik, ki ga uporablja klicatelj. Razred Oblika v predstavljenem primeru.
- Klicatelj, ki uporablja funkcije razredov, predpisane v vmesniku, ki ga določa tarča. V predstavljenem primeru je to orodje za risanje.

- Prilagojenec, katerega vmesnik je treba uporabiti in prilagoditi. PogledBesedilo v predstavljenem primeru.
- Prilagojevalnik, ki prilagodi vmesnik razreda prilagojenec tako, da ustreza vmesniku razreda tarča. OblikaBesedilo v predstavljenem primeru.

Klicatelj vedno uporablja instanco razreda prilagojevalnik, ki prejete zahteve posreduje razredu prilagojenec. Le-ta nato vrne rezultat, ki ga prilagojevalnik posreduje klicatelju.

Posledice uporabe

Uporaba razredov in instanc načrtovalskega vzorca prilagojevalnik nas privede do naslednjih prednosti in slabosti:

- Razred tipa prilagojevalnik točno prilagodi razred tipa prilagojenec tako, da ustreza vmesniku razreda tipa tarča. Posledično takšen način ne bo deloval, kadar želimo prilagoditi nek razred in vse njegove podrazrede.
- Dovoljuje razredu tipa prilagojevalnik spremembo nekaterih funkcionalnosti razreda tipa prilagojenec.
- Omogoča uporabo le enega razreda tipa prilagojevalnik za dostop do razreda tipa prilagojenec brez potrebe po dodatnih sklicih nanj.
- Omogoča enemu razredu tipa prilagojevalnik delovanje z več razredov tipa prilagojenec, s čimer omogoča sočasno razširitev vseh razredov tipa prilagojenec.
- Otežuje prepis funkcij razreda tipa prilagjenec, saj bi s tem zahteval njegovo dedovanje.

Implementacija

Načrtovalski vzorec prilagojevalnik je pri implementaciji oblačnih razširitev večkrat uporabljen. Lep primer njegove uporabe predstavlja prilagoditev

prejetih in poslanih sporočil oblačnega prožilca in prožilca HTTP v enotno obliko. Potrebna je zaradi uporabe različnih vrst sporočil HTTP ter skupne kode za obdelavo podatkov in izvedbo akcij prožilcev. Izsek programske kode 7.2 prikazuje prilagoditev rezultata oblačnega prožilca v sporočilo HTTP.

```
private static HttpResponseMessage CreateResponse(HttpTriggerResponseData
    responseData)
{
    HttpResponseMessage responseMessage = new HttpResponseMessage();

    foreach (var header in responseData.Headers)
    {
        responseMessage.Headers.Add(header.Key, header.Value);
    }

    if (responseData.Content != null && responseData.Content.Length > 0)
    {
        ByteArrayContent content = new ByteArrayContent(responseData.Content);
        content.Headers.ContentLength = responseData.Content.Length;

        if (!string.IsNullOrEmpty(responseData.ContentType))
        {
            content.Headers.ContentType = new
                MediaTypeHeaderValue(responseData.ContentType);
        }

        responseMessage.Content = content;
    }

    responseMessage.ReasonPhrase = responseData.StatusDescription;

    if (responseData.StatusCode.HasValue)
    {
        responseMessage.StatusCode =
            (HttpStatusCode)responseData.StatusCode.Value;
    }

    return responseMessage;
}
```

Izsek kode 7.2: Prilagoditev rezultata prožilca v sporočilo HTTP.

7.2.3 Vzorčna metoda

Začnimo s primerom programskega okvirja, ki vsebuje razreda Aplikacija in Dokument. Razred Aplikacija je odgovoren za odpiranje obstoječih dokumentov, shranjenih v zunanjih datotekah. Razred Dokument pa je odgovoren za predstavitev informacij odprtega dokumenta.

Aplikacije, zgrajene z uporabo takšnega okvirja, lahko dedujejo predstavljena razreda in ju prilagodijo svojim potrebam. Aplikacija za risanje bo

tako definirala razreda RisanjeAplikacija in RisanjeDokument, aplikacija za delo s tabelami pa razreda TabelaAplikacija in TabelaDokument.

Abstraktni razred Aplikacija mora zato definirati algoritem za odpiranje in branje vsebine dokumentov, imenovan OdpriDokument. Algoritem mora vsebovati vsak korak odpiranja dokumenta. Preveriti mora, ali je datoteko mogoče odpreti, ustvariti instanco razreda Dokument, definirano s strani razreda Aplikacija, jo dodati v svoj seznam dokumentov ter vanjo zapisati vsebino odprte datoteke.

Takšna definicija algoritma temelji na načrtovalskem vzorcu vzorčna metoda (angl. template method). Vzorec definira algoritem, sestavljen iz klicev abstraktnih metod, ki so implementirane s strani dedujočih razredov. Razredi, dedovani z razreda Aplikacija, morajo implementirati dve metodi. Prva je namenjena preverjanju možnosti odpiranja datoteke z dokumentom, druga pa kreiranju in hranjenju instance razreda Dokument. Razredi, dedovani iz razreda Dokument, pa morajo implementirati metodi za odpiranje in branje vsebine dokumenta.

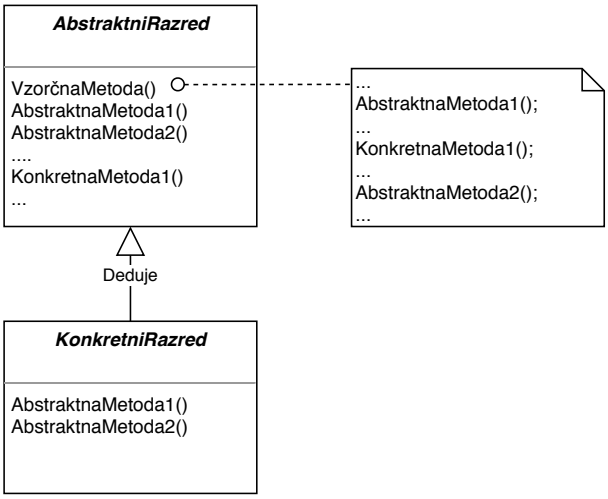
Z definicijo abstraktnih korakov algoritma in definicijo abstraktnih metod načrtovalski vzorec določa zaporedje izvajanja in hkrati dedujočim razredom predpisuje implementacijo teh metod.

Uporaba

Načrtovalski vzorec vzorčna metoda je priporočljivo uporabiti v enem izmed treh primerov. Prvi je, kadar želimo implementirati nespremenljiv del algoritma enkrat, spremenljivi del pa prepustiti njegovim dedujočim razredom.

Drugi primer je, kadar želimo izluščiti skupno kodo dedujočih razredov. V tem primeru moramo najprej identificirati njihove razlike v obstoječi kodi, ki jih nato ločimo v nove metode. Po uspešni ločitvi spremenimo obstoječi algoritem tako, da vsebuje le skupno kodo in klice abstraktnih metod. Implementacijo abstraktnih metod pa prepustimo dedujočim razredom.

Zadnji razlog za uporabo načrtovalskega vzorca vzorčna metoda pa je nadzor nad razširitvami dedujočih razredov. Algoritem lahko definiramo



Slika 7.5: Struktura načrtovalskega vzorca vzorčna metoda.

tako, da izvede klic razširitev le ob izvajanju predpisanih korakov, in s tem omogočimo njihov klic le ob izbranih trenutkih.

Struktura

Struktura načrtovalskega vzorca vzorčna metoda je prikazana na sliki 7.5 in je sestavljena iz naslednjih dveh delov:

- Abstraktni razred z definicijo abstraktnih metod, implementiranih v dedujočih razredih, in implementacijo metode algoritma. Algoritem je sestavljen iz klicev abstraktnih in konkretnih metod. Konkretne metode so implementirane znotraj abstraktnega razreda in vsebujejo skupno kodo, ki je potrebna za delovanje algoritma.

- Dedujoči razredi z implementacijo predpisanih abstraktnih metod.

Dedujoči razredi se za svoje delovanje vedno zanašajo na abstraktnega. Odvisni so od implementacije algoritma in metod s skupno logiko.

Posledice uporabe

Programerski vzorec vzorčna metoda je velikokrat uporabljen pri implementaciji knjižnic. Predstavlja sredstvo za lokalizacijo skupne programske kode in obnašanje posameznih skupin razredov.

Njegova uporaba vodi do zamenjave nadzora nad potekom programa, ki je posledica klicev funkcij dedujočega razreda z abstraktnega razreda. Algoritem načrtovalskega vzorca pri izvajanju kliče naslednje metode:

- Abstraktne metode, implementirane s strani dedujočih razredov.
- Konkretne metode s skupno kodo, implementirane v abstraktnem razredu.
- Metode razširitev s privzeto implementacijo v abstraktnem razredu in privzetim načinom delovanjem. Njihova prava implementacija je neobvezna in prepuščena dedujočim razredom.

Pri implementaciji obravnavanega programerskega vzorca je ločitev abstraktnih metod in metod razširitev zelo pomembna. Implementacija abstraktnih metod je obvezna, implementacija razširitev pa opcijska.

Implementacija

Pri implementaciji oblačnih razširitev je bila uporabljena prilagojena različica načrtovalskega vzorca vzorčna metoda. Glavna ideja o definiciji algoritma je ostala enaka. Vključuje le spremembo vpeljave abstraktnih metod, ki so algoritmu podane preko parametra.

Primer implementacije prilagojene različice načrtovalskega vzorca je prikazan v izseku programske kode 7.3. Koda je namenjena razširitvi zapisovanja dnevniških sporočil. Zapisanemu sporočilu doda časovni žig in omogoča preusmeritev izpisa dnevnika v dve datoteki.

```
// Preusmeritev zapisovanja dnevnika v dve datoteki.
public static void EnableFileLogging(string logFile0, string logFile1, int
    maxLogLines = 5000) {...}

// Preusmeritev zapisovanja dnevnika v konzolo.
public static void DisableFileLogging() {...}

// Zapis napake.
public static void LogErrorWithTimestamp(this ILogger logger, string message,
    object[] args)
{
    LoggingExtensions.LogWithTimestamp(logger.LogError, message, args);
}

// Zapis opozorila.
public static void LogWarningWithTimestamp(this ILogger logger, string message,
    object[] args)
{
    LoggingExtensions.LogWithTimestamp(logger.LogWarning, message, args);
}

// Zapis informacije.
public static void LogInformationWithTimestamp(this ILogger logger, string
    message, object[] args)
{
    LoggingExtensions.LogWithTimestamp(logger.LogInformation, message, args);
}

private static void LogWithTimestamp(Action<string, object[]> loggingFunction,
    string message, object[] args)
{
    // Dodajanje casovnega ziga obstojecemu sporocilu.
    string formattedMessage = $"[{DateTime.Now}] {message}";

    // Klic "abstraktne" funkcije za zapisovanje.
    loggingFunction(formattedMessage, args);

    if (LoggingExtensions.fileLoging)
    {
        // Zamenjava dnevniskih datoteke v primeru dosezenega limita zapisov
        if (LoggingExtensions.fileLogger.LogFileSwitched())
        {
            Console.SetOut(LoggingExtensions.fileLogger.LogStreamWriter);
        }
    }
}
```

Izsek kode 7.3: Razširitve zapisovanja dnevniških sporočil.

Poglavje 8

Sklepne ugotovitve

Pogledali smo si celotno arhitekturo sistema za upravljanje z etiketami. Predstavljeni so bili vsi njegovi deli programske in strojne opreme, vključno z njihovimi ključnimi funkcionalnostmi. Za lažje razumevanje njihovega delovanja in uporabniških zahtev je bil celoten sistem ponazorjen s primerom trgovine s pohištvom in njeno širitvijo. Z opisom arhitekture in omenjenega primera smo spoznali vlogo sistema v podjetjih različnih velikosti.

Spoznali smo, da sta delitev in nadzor etiket ter rešitev po meri ključnega pomena za enoten proces načrtovanja in tiskanja. Zato mora biti zagotovljen ustrezen nadzor dostopa uporabnikov, ki jim omogoča uporabo potrebnih funkcionalnosti sistema. Opisan je postopek njihovega dodajanja in dodeljevanja pravic. Za uspešno uporabo sistema za upravljanje z etiketami morajo biti uporabniki vanj prijavljeni z elektronskim računom Google ali Microsoft. Preverjanje njihove pristnosti temelji na opisanem protokolu OAuth 2.0, ki jim omogoča uporabo obstoječih računov.

Predstavljene so potrebe večjih podjetij, ki za svoje delovanje uporabljajo različne med seboj povezane sisteme in oblačne storitve. Zaznana je bila potreba po možnosti integracije in tiskanja etiket z uporabo tovrstnih sistemov. Tovrstna podpora je bila zagotovljena z implementacijo razširitve oblačnega prožilca, ki omogoča nadzor procesa tiskanja z uporabo obstoječih aplikacij.

Z možnostjo integracije smo pokrili zahteve večjih podjetij, ostala pa

so nam še razpršena podjetja, katerih proces tiskanja skušamo izboljšati z uvedbo oblačnega tiskalniškega povezovalca. Z njim je uporabnikom omogočeno tiskanje etiket neodvisno od lokacije in uporabljenega računalnika, saj namestitev tiskalniških gonilnikov ni potrebna.

Implementirane oblačne razširitve sistema za upravljanje z etiketami temeljijo na uporabljenih programskih vzorcih, predstavljenih v zadnjem poglavju. Raziskana, uporabljena in predstavljena sta bila dva arhitekturna in trije načrtovalski vzorci.

Sistem za upravljanje z etiketami je z implementacijo opisanih oblačnih razširitev pridobil na svoji uporabnosti. Podjetjem omogoča integracijo procesa tiskanja z drugimi procesi in s tem njihovo optimizacijo. Poleg tega pa predvsem manjšim in razpršenim podjetjem omogoča tudi uporabo oblačnih tiskalnikov in možnost tiskanja etiket brez namestitve tiskalniških gonilnikov. S tem se v primeru uporabe le oblačnih tiskalnikov lahko popolnoma izognejo strežnikom za deljenje tiskalnikov in njihovih gonilnikov.

Slike

Izseki kode

Literatura

[1] K. Weigelt, M. Hambsch, G. Karacs, T. Zillger, A. C. Hübler, Labeling the world: Tagging mass products with printing processes, IEEE Pervasive Computing 9 (2) (2010) 59 – 63.

[2] OSHA, A guide to the globally harmonized system of classification and labeling of chemicals (ghs), https://www.osha.gov/dsg/hazcom/ghsguideoct05.pdf, dostopano: 26.02.2019.

[3] Siemens, Siemens standardizes labeling across its global factories to drive new levels of efficiency, https://industrytoday.com/wp-content/uploads/2018/03/Siemens_Labeling_Case_Study.pdf, dostopano: 26.02.2019.

[4] P. Louridas, Up in the air: Moving your applications to the cloud, IEEE Software 27 (4) (2010) 6 – 11.

[5] G. Milener, C. Rabeler, N. Schonning, Dayo, s. Moses, Multi-tenant saas database tenancy patterns, https://docs.microsoft.com/en-us/azure/sql-database/saas-tenancy-app-design-patterns, dostopano: 26.02.2019.

[6] A. R. Biswas, R. Giaffreda, Iot and cloud convergence: Opportunities and challenges, 2014 IEEE World Forum on Internet of Things (WF-IoT) (2014) 375 – 376.

[7] V. Beltran, A. F. Skarmeta, Overview of device access control in the iot and its challenges, IEEE Communications Magazine 57 (1) (2019) 154 – 160.

[8] IETF, The oauth 2.0 authorization framework, https://tools.ietf.org/html/rfc6749, dostopano: 26.02.2019.

[9] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design patterns elements of reusable object-oriented software (2005).

[10] S. Tiwari, An introduction to qr code technology, 2016 International Conference on Information Technology (2016) 39 – 44.

[11] P. Louridas, Soap and web services, IEEE Software 23 (6) (2006) 62 – 67.

[12] R. D. Caytiles, S. Lee, B. Park, Cloud computing: The next computing paradigm, International Journal of Multimedia and Ubiquitous Engineering 7 (2) (2012) 297 – 302.

[13] E. Ross, C. de Guzman, B. Mathers, M. Garg, S. Cai, R. Agiewich, T. Sanders, D. Bahall, B. Kess, What is azure active directory?, https://docs.microsoft.com/en-us/azure/active-directory/fundamentals/active-directory-whatis, dostopano: 26.02.2019.

[14] IETF, The oauth 1.0 protocol, https://tools.ietf.org/html/rfc5849, dostopano: 26.02.2019.

[15] IETF, The oauth 2.0 authorization framework: Bearer token usage, https://tools.ietf.org/html/rfc6750, dostopano: 26.02.2019.

[16] D. R. Kuhn, E. J. Coyne, T. R. Weil, Adding attributes to role-based access control, Computer 43 (6) (2010) 79 – 81.

[17] W. Zhang, G. Cheng, A service-oriented distributed framework, 2009 International Conference on Web Information Systems and Mining (2009) 302 – 305.

[18] S. Danielson, J. Kornich, T. Nevil, K. Whitlatch, JiayeueHu, A. Pasic, S. Rolfe, M. Wnzel, K. Crider, E. Reitan, C. Rollandy, C. Fowler, What is api management?, `https://docs.microsoft.com/en-us/azure/api-management/api-management-key-concepts`, dostopano: 26.02.2019.

[19] IETF, The websocket protocol, `https://tools.ietf.org/html/rfc6455`, dostopano: 26.02.2019.

[20] S. Turner, Transport layer security, IEEE Internet Computing 18 (6) (2014) 60 – 63.

[21] A. Syromiatnikov, D. Weyns, A journey through the land of model-view-design patterns, 2014 IEEE/IFIP Conference on Software Architecture (2014) 21 – 30.

[22] A. Holca, Mvc vs mvvm, `https://assist-software.net/blog/mvc-vs-mvvm`, dostopano: 26.02.2019.