Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Nejc Gutnik

Deljeno skladišče dokumentov

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: dr. Andrej Brodnik
Somentor: mag. Robert Sraka

Ljubljana, 2018

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Skladišče dokumentov postaja dandanes običajna infrastruktura, ki jo uporabljajo podjetja. Žal infrastruktura ne sledi vedno potrebam podjetij. V diplomski nalogi preučite in začrtajte arhitekturo, ki bo nudila deljeno skladišče dokumentov večim družbam v okviru istega podjetja. Za avtentikacijo in za tem avtorizacijo za dostop do shranjenih dokumentov uporabite federacijo lokalnih avtentikacijskih storitev posameznih družb. Arhitekturo tudi implementirajte tako, da bo omogočala preprosto povečljivost in razširljivost (vključevanje novih podjetij). Implementacija naj nudi učinkovito iskanje po shranjenih dokumentih ter možnost sledenja sprememb v dokumentih.

Zahvaljujem se vsem, ki so mi dali priložnost in možnost, da postanem najboljši inženir računalništva in informatike. Naj to diplomsko delo velja kot dokaz in zahvala za njihov trud in voljo ter nadaljnjo podporo.

Če bi imel eno uro da rešim svet, bi 55 minut posvetil definiranju problema in 5 minut iskanju rešitve. In šel bi s kolegom na pivo. Hvala, Matic.

The sea is vast. Someday, surely, your nakama will appear. - Jaguar D. Saul.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
AMQP	Advanced Message Queuing Protocol	Napredni protokol za veriženje sporočil
API	Application programming interface	Vmesnik uporabniškega programa
CMS	Content Management System	Sistem za upravljanje vsebin
CSRF	Cross-Site Request Forgery	Napad CSRF
CRUD	Create, Read, Update, Delete	Ustvari, beri, posodobi, izbriši
DI	Dependency Injection	Vstavljanje odvisnosti
ES	Elasticsearch	Iskalni indeks Elasticsearch
EF	Entity Framework	Orodje za objektno-relacijsko preslikavo Entity Framework
EEX	Easy Enterprise.X	Sistem za upravljanje vsebin Easy Enterprise.X
ERP	Enterprise resource planning	Celovita programska rešitev
HTTP	HyperText Transfer Protocol	Protokol za prenos hiperteksta
V/I	Input/output	Vhod/izhod
JSON	JavaScript Object Notation	Zapis za predstavitev Javascript objektov

KC	Keycloak	Orodje za upravljanje dostopa in istovetnosti Keycloak
LDAP	Lightweight Directory Access Protocol	Lahki protokol za dostop do imenikov
LINQ	Language Integrated Query	Poizvedba s pomočjo programskega jezika
MVC	Model-View-Controller	Model-pregled-nadzor
MSSQL	Microsoft Structured Query Language	Microsoftov strukturiran povpraševalni jezik
ORM	Object-Relational Mapping	Objektno-relacijska preslikava
PDF	Portable Document Format	Datotečni format za izmenjavo dokumentov
SD	-	Skladišče dokumentov
RDBMS	Relational Database Management System	Sistem za upravljanje z relacijskimi bazami
REST	Representational State Transfer	Predstavitveni prenos stanja
SAP	Systems, Applications & Products in Data Processing	Sistemi, aplikacije in produkti za procesiranje podatkov (podjetje)
SOA	Service-Oriented Architecture	Storitveno orientirana arhitektura
SOAP	Simple Object Access Protocol	Protokol za preprost dostop do objektov
SUPB	-	Sistem za upravljanje s podatkovnimi bazami
TLS	Transport Layer Security	Varnost transportnega sloja
XSS	Cross-Site Scripting	Večdomensko izvajanje kode

Povzetek

Naslov: Deljeno skladišče dokumentov

V delu je predstavljena arhitektura skladišča dokumentov, ki je bila načrtovana za podporo poslovnih procesov na področju zavarovanja in pozavarovanja. Izdelali smo arhitekturo za sistem, ki je centraliziran in podpira več družb hkrati. Posebno pozornost smo posvetili varnosti, odzivnosti, povečljivosti, razširljivosti in enostavnosti vzdrževanja. Arhitektura sistema temelji na ogrodju *ASP.NET Core*, ki uporablja modul za izdelavo zaledja *ASP.NET Web API*. Dokumenti se shranjujejo arhiv dokumentov *Easy Enterprise.X*. Za upravljanje z metapodatki dokumentov se uporablja SUPB *Microsoft SQL*. Del arhitekture sta tudi strežnik *Keycloak* (avtorizacija, avtentikacija, federacija LDAP) in iskalnik *Elasticsearch*, ki je zadolžen za hitro iskanje po metapodatkih dokumentov.

Ključne besede: ASP.NET, API, REST, JSON, LDAP, Elasticsearch, OAuth, Keycloak, MSSQL, varnost.

Abstract

Title: Shared document repository

In the thesis we present a system architecture for repository of documents that was designed to support business processes in the area of insurance and reinsurance. The architecture supports multiple companies and is centralized. The key system properties we aimed for are security, responsivness, scalability, extensibility and maintainability. The architecture is based on a *ASP.NET Core* framework, which uses a module called *ASP.NET Web API* for creating back-end applications. The documents are saved in a CMS driven archive *Easy Enterprise.X*. For the metadata of documents a *Microsoft SQL* RDBMS is used. Authorization, authentication and LDAP federation are provided via the authorization server *Keycloak*. A search engine *Elasticsearch* is responsible for making quick searches accross the metadata of documents.

Keywords: ASP.NET, API, REST, JSON, LDAP, Elasticsearch, OAuth, Keycloak, MSSQL, security.

Poglavje 1

Uvod

Skupino SavaRe sestavljajo družbe s področja zavarovanja in pozavarovanja. Vsaka družba v skupini ima več aplikacij, preko katerih potekajo poslovni procesi. Aplikacije obsegajo različna področja, na primer, zavarovalništvo, pozavarovalništvo, računovodstvo, finance. Primer take aplikacije, ki se uporablja v pozavarovalnici SavaRe, je lastna pozavarovalna aplikacija REvolve. To zaposlenim omogoča, da vnesejo novo pozavarovalno pogodbo, naredijo obnovo pogodbe iz prejšnjih let, uredijo podatke o pogodbi, dostopajo do poročil ter obdelajo obračune.

Problem v skupini je, da nima sistema, ki bi omogočal skladiščenje in dostop do dokumentov vsem aplikacijam v skupini. Aplikacije v skupini sestavlja širok nabor tipov aplikacij. Mednje sodijo klasične monolitne aplikacije, spletne aplikacije, spletne storitve, mobilne aplikacije in vrsta celovitih programskih rešitev (ERP). Trenutno se v družbah v skupini uporabljajo interne rešitve, ki tega ne omogočajo – so preveč specializirane za določeno družbo in ne podpirajo želenih funkcionalnih zahtev. Na primer, mobilne aplikacije v trenutni arhitekturi ne morejo pridobiti dokumenta iz neke družbe, saj je rešitev omejena na interno okolje družbe.

Težavo predstavljajo tudi primeri dokumentov in skupin dokumentov s področja zavarovanja in pozavarovanja. Ti tipi dokumentov so, na primer, škodni spisi. Škodni spis je skupek pisnih dokumentov, izjav in dokazil o

zavarovalnem primeru, cenitvi in likvidaciji odškodnine. Sestavljen je iz zavarovalne pogodbe oziroma zavarovalne police in podatkov o zavarovalnem primeru – škodnem dogodku [17]. Zaradi obširnosti takih skupin dokumentov ne obstajajo povezave le znotraj skupin, ampak se dokumenti sklicujejo tudi na ostale dokumente znotraj družbe. V trenutni rešitvi ni sistema map, s katerim bi lahko preprosto uskupinjali takšne skupine dokumentov.

Dodatna težava je, da velik del delovnega procesa še vedno temelji na uporabi dokumentov v papirnati obliki. Prostori za shranjevanje so omejeni in ob hitro naraščajočem številu dokumentov upravljanje arhiva dokumentov v papirnati obliki predstavlja velik strošek. Za tako poslovno skupino je ustrezno omogočiti arhitekturo za poenoten pregled nad dokumenti in poslovanjem skupine.

Pripraviti je bilo treba enoten vmesnik, preko katerega se povezujejo aplikacije, in enotno strukturo dokumenta, ki jo lahko te aplikacije uporabljajo. Sistem, ki to omogoča, imenujemo skladišče dokumentov (v nadaljevanju SD).

1.1 Struktura naloge

V uvodu je na kratko predstavljena uvodna motivacija. V naslednjem razdelku so naštete ciljne funkcionalnosti SD (skladišča dokumentov). Nato sledi razdelek, v katerem so predstavljeni razvojna orodja in tehnologije, uporabljene pri razvoju SD. V glavnem poglavju sta predstavljeni stara in nova arhitektura ter podrobneje predstavljene posamezne komponente nove arhitekture. Sledi poglavje, ki ovrednoti lastnosti nove arhitekture in jih primerja s staro rešitvijo. Zadnje poglavje vsebuje sklepne ugotovitve, možne izboljšave v arhitekturi in pomen SD za skupino SavaRe. V dodatku so navodila za nastavitev spletne storitve *.NET Core* in strežnika *Keycloak* za pooblaščanje in overjanje, dodatno slikovno gradivo ter izseke kodi iz nove rešitve.

1.2 Funkcionalne in nefunkcionalne zahteve

Za zagotavljanje podpore poslovnih procesov smo definirali naslednje funkcionalne zahteve v arhitekturi SD:

- **Podpora za več družb hkrati *(ang. multitenancy)***. Arhitektura mora podpirati več družb hkrati. V arhitekturi programske opreme to pomeni, da ena instanca strežnika služi več najemnikom (*ang. tenant*). Najemnik je skupina uporabnikov, ki si delijo skupni dostop z določenimi pravicami do programske instance. Arhitektura je pripravljena na tak način, da vsak najemnik dobi del programa. S tem so mišljeni dostop do podatkov, upravljanje z uporabniki in morebitne posebne zahteve posameznega najemnika [10]. Spletna storitev bo morala, glede na poverilnice pri prijavi, določiti, iz katere družbe prihaja odjemalec in ob vlaganju ta podatek ustrezno dodati v metapodatke dokumenta.

- **Spletna storitev**. Vse aplikacije, ki za svoje delo uporabljajo dokumente, do SD dostopajo preko spletne storitve. Do arhiva nato dostopa le spletna storitev. Ta mora omogočati operacije CRUD in relacije med zbirkami v spletni storitvi. Zahtevamo, da vsebuje metapodatke o dokumentu, metapodatke za klasifikacijski načrt – to je tabela, ki vsebuje podatke o tipu dokumenta in njegovi dobi hranjenja, ter metapodatke o navideznih mapah. Navidezne mape so tip map, ki prikažejo datoteke uporabnikom v drugačnem pogledu. Namesto da uporabijo dejansko fizično strukturo datotečnega sistema, uporabijo metapodatke dokumenta [12].

- **Avtorizacija**. SD mora vsebovati sistem dostopnih pravic po uporabnikih in uporabniških skupinah. Ti so določeni v strežnikih LDAP posameznih družb. Želimo imeti nadzorovan dostop do dostopnih točk spletne storitve in metapodatkov posameznega objekta. Vsaka aplikacija in njeni uporabniki lahko dostopajo do dokumentov, do katerih imajo dovoljenja v okviru svoje družbe.

- **Večkriterijsko iskanje po dokumentih**. Spletna storitev v SD mora vsebovati možnost iskanja po različnih merilih in metapodatkih. Iskalnik mora podpirati iskanje po ključnih besedah, iskanje po vrednostih atributov metapodakov, filtriranje dokumentov glede na nabor izbranih atributov in sestavljanje poizvedb glede na podane logične operatorje. Iskanje mora upoštevati dostopne pravice, ki jih imamo nad dokumenti.

- **Različice dokumentov in beleženje sprememb**. Spletna storitev mora imeti možnost ustvarjanja in prikaza različic dokumentov. Zaželeno je, da je dostop do različic dokumenta preko dostopne točke. Ob vsakem urejanju dokumenta in njegovih metapodatkov se spremembe med različicami zabeležijo. Vsaka aktivnost uporabnikov v SD naj se zapisuje v dnevniško datoteko. Beležiti moramo podatke o tem, kdo je napravil spremembo na dokumentu, kdaj je bila sprememba opravljena, kakšne so spremembe med različicama dokumentov, uspešnost transakcije in čas spremembe. Ob dostopu na to dostopno točko želimo imeti možnost zahteve vsake različice dokumenta posebej. Poleg dokumenta želimo imeti zapisan tudi seznam vseh sprememb, opravljenih na dokumentu.

Ker je arhitektura programske opreme v svoji osnovi zelo odvisna od nefunkcionalnih zahtev, je pomembno, da jih identificiramo in vključimo v načrt. Kot najpomembnejše nefunkcionalne zahteve smo identificirali varnost, razširljivost, vzdržljivost, odzivnost in povečljivost.

1.2.1 Računalniška varnost

S pojmom računalniška varnost označujemo zaščito računalniških sistemov pred krajo ali povzročanjem škode na strojni opremi, programski opremi, informacijah v sistemu in zlorabo njihovih storitev [5].

V SD je varnost ena najpomembnejših zahtev. V SD se bodo shranjevali tako strogo zaupni podatki, kot tudi osebni podatki uporabnikov. To je

razlog, da smo varnosti posvetili posebno pozornost. V naslednjih nekaj odstavkih so predstavljeni tipični napadi in ranljivosti, ki jih želimo preprečiti.

Napad z SQL vrinjenjem

Napad z SQL vrinjenjem (*ang. SQL injection*) je tehnika napada, pri kateri je del uporabnikovega vnosa obravnavan kot stavek SQL [6]. Na primer, da imamo dostopno točko */id/*{*idUser*}, ki vrne podatke o uporabniku. Če parametra *idUser* nebi potrjevali kot podatkovni tip celo število in vnaprej pripravili stavka SQL, bi napadalec ta napad lahko izvedel s klicem */id/'%20or%20'1'='1'*. Tako bi prišlo do izvedbe naslednjega stavka SQL : SELECT * FROM Users WHERE UserID = '' or '1' = '1'. Ta bi vrnil seznam vseh uporabnikov in s tem ogrozil varnost aplikacije.

Večdomensko izvajanje kode

XSS (*ang. cross-site scripting*) je varnostna grožnja, ki poteka tako, da napadalec vnese zlonamerno kodo v izhod aplikacije, ki se nato pošlje v uporabnikov brskalnik. Ta koda se nato izvede v brskalniku [19]. Žrtev ob kliku na zlonamerno povezavo naredi poizvedbo na strežnik, kamor pošlje napadalčevo skripto. Strežnik nato vrne odgovor in izvede napadalčevo zlonamerno kodo. Če napadalcu uspe prestreči odgovor, lahko pridobi zaupne informacije. To napadalcu omogoča, da pride do podatkov, kot so piškotki in sejni žetoni, ali pa brskalnik preusmeri na drugo internetno stran. Napadalec lahko izrabi pravice uporabnika za dostop do spletne storitve preko napada XSS za pridobivanje informacij o spletni storitvi in njeni vsebini.

CSRF

CSRF (*ang. cross-site request forgery*) je napad, ki dovoli napadalcu, da prisili uporabnika, da izvede neželene akcije v spletni aplikaciji, v kateri je trenutno prijavljen, s tem ko uporabi uporabnikove poverilnice in pooblaščenost [9]. CSRF izkorišča zaupanje med spletno aplikacijo in strežnikom. V osnovi

deluje tako, da izkoristi predvidljivost parametrov v zahtevi, da lahko napadalec pripravi zlonamerno zahtevo.

V spletni storitvi za vlaganje v SD želimo uporabiti žeton CSRF za preprečevanje napada CSRF. Pooblaščeni odjemalec naredi zahtevo, ki vključuje spremembo stanja na strani strežnika. Strežnik tvori naključni varnostni žeton in ga pošlje odjemalcu v glavi HTTP odgovora. Ko odjemalec pošlje novo zahtevo, strežnik preveri, ali je žeton v glavi nove zahteve enak tistemu, ki je ga je podal strežnik. Če se vrednosti ne ujemata, strežnik zahtevo zavrne. Zaščita z žetonom CSRF ščiti tudi pred napadom s ponavljanjem, (*ang. replay attack*), saj napadalec po preteku in uporabi žetona ne more ponovno uporabiti prestreženih parametrov za izvedbo zahteve.

1.2.2 Razširljivost

Glavno vprašanje pri razširljivosti je, kako zagotoviti dodajanje sprememb in zmanjšati vpliv na delovanje obstoječih sistemskih funkcij [8]. Dobro razširljiv sistem je tak, da se v njega lažje dodaja nove funkcionalnosti, ali pa se to zgodi s preureditvijo trenutne funkcionalnosti. Zahtevamo, da je arhitektura dobro razširljiva. Podpirati mora dodajanje novih najemnikov. Sem spadata tudi dodajanje novih aplikacij in uporabnikov. Pri avtorizaciji želimo imeti možnost dodajanja novih vlog za dostop. Spletna storitev mora biti pripravljena tako, da lahko ob novih funkcionalnih zahtevah dodamo nove dostopne točke. Pri vlaganju dokumenta želimo imeti možnost shranjevanja poljubnih metapodatkov.

1.2.3 Vzdržljivost

Vzdržljivost pomeni načrtovanje sistema na tak način, da dovoljuje dodajanje novih zahtev, ne da bi s tem dodali tveganje za nove napake [8]. Dobra vzdržljivost pomeni večjo verjetnost za lažji nadaljnji razvoj. Vzdržljivost nima določene, enote s katero jo lahko natančno opredelimo. Ocenili jo bomo s pomočjo naslednjih vidikov: Preskušalnost (*ang. testability*), razumljivost

(*ang. understandability*) in spremenljivost kode (ang. modifiability). S preiskušalnostjo bomo preverili, kako dobro je s testi pokrita rešitev. S tem mislimo na pokritost rešitve s testi modulov (*ang. unit test*) in integracijskimi testi (*ang. integration test*). Razumljivost kode nam bo povedala, ali je dobro dokumentirana, in ali sledi dogovorom pisanja v jeziku. S spremenljivostjo kode bomo ugotavljali, ali se jo da preprosto spreminjati in ali so razredi tesno ali šibko sklopljeni.

Ob prehodu rešitve na predprodukcijsko okolje želimo imeti glavni dostopni točki spletne storitve za vlaganje in prevzem dokumenta pokriti z osnovnimi testi modulov. To so testi, ki bi preverjali, da ob pravilnih parametrih za dostop spletna storitev vrne odgovor s statusom 200, ob neobstoječem dokumentu pa vrne odgovor s statusom 404, da se v primeru, da metapodatki dokumenta niso v predpomnilniku, kliče metoda za dodajanje v predpomnilnik in podobno.

Pri razumljivosti kode želimo imeti dostopne točke v spletni storitvi opisane in predstavljene s pomočjo ogrodja za dokumentiranje spletnih storitev *Swagger*. Za kodo želimo, da sledi konvencijam poimenovanja v jeziku *C#*, vendar tu ni nujno, da se jih strogo drži.

Pri spremenljivosti kode želimo, da so razredi v spletni storitvi čim bolj šibko sklopljeni.

1.2.4 Odzivnost

Odzivnost je sposobnost sistema, da opravi nalogo v želenem času [20]. Pri odzivnosti želimo, da je odzivni čas pri odgovoru na zahtevo čim krajši. Odzivni čas je skupen čas, ki je potreben pri spletni storitvi, da odgovori na zahtevo. Odzivni čas je vsota časa za obdelavo zahteve v spletni storitvi, časa čakanja , da zahteva pride na vrsto za obdelavo, in časa za prenos odgovora zahteve nazaj k odjemalcu [21]. Pri SD pričakujemo, da bo odzivni čas precej nihal glede na vrsto zahteve, na primer vlaganje in prevzem dokumenta in pasovno širino na voljo med odjemalcem in strežnikom. Za iskanje po metapodakih dokumenta v spletni storitvi toleriramo odzivni čas 200 ms.

1.2.5 Povečljivost

Povečljivost (*ang. scalability*) je zmožnost povečave sistema v izbrani dimenziji, brez večjih posegov v njegovo arhitekturo [8]. Cilj za to zahtevo je pripraviti arhitekturo na tak način, da bo čim večji del pripravljen na vzporedno (ang. horizontal) povečljivost – v primeru, da v sistem dodamo več virov v obliki strojne opreme, se sistemu poveča njegov celoten iznos. Ker še nimamo pripravljenega okolja, v katerem bi lahko merili iznos, je to za zdaj edina zahteva.

Poglavje 2

Orodja in tehnologije

V skupini SavaRe se za podporo delovnim procesom uporablja večinoma programske rešitve podjetij Microsoft in Oracle. Ustrezna orodja smo iskali tudi v odprtokodnih rešitvah.

2.1 Arhiv dokumentov Easy Enterprise.X

Easy Enterprise.X (v nadaljevanju EEX) je upravitelj vsebin za podjetja (*ang. enterprise content management*) [2]. Vgrajena ima orodja za izdelavo delovnih tokov dokumentov, podporo celovitim programskim rešitvam (SAP, *Navision*), e-sporočanje in arhiviranje dokumentov. EEX vsebuje tudi orodja za elektronski zajem dokumentov, uskupinjanje dokumentov in vezavo dokumentov.

Vsaka družba ima na njem ločen bazen (*ang. pool*), kjer so shranjeni dokumenti. Vsak dokument ima dodeljen ključ. Za podatkovno bazo v arhivu se uporablja SUPB podjetja Oracle v navezavi z datotečnim sistemom.

EEX je trenutno v najemu v enem od zunanjih podjetij, ki ga tudi vzdržuje, do arhiva pa ima družba SavaRe dostop preko spletnih storitev SOAP, ki so jih pripravili v eni od družb. Podrobno delovanje EEX nam zaradi omejenega dostopa ni znano.

Razlog za njegovo uporabo je tem, da morajo dokumentni sistemi za

uporabo v podjetju v Sloveniji dosegati določene standarde. EEX jih dosega in ima potrdilo Republike Slovenije, da ga lahko uporabimo v skupini SavaRe.

Ker je EEX celovita poslovna rešitev, bi lahko, namesto nove arhitekture, v celoti uporabili EEX, saj izpolnjuje vse naše zahteve. Težava nastane pri plačljivih uporabniških licencah, ki močno dvigajo strošek rešitve z uporabo EEX. Nova arhitektura je načrtovana tako, da nadomesti komponente iz EEX in jih zagotovi sama. V arhitekturi SD je zato EEX uporabljen izključno kot arhiv dokumentov.

2.2 Ogrodje spletne storitve

Glavna komponenta, s katero uporabniki v SD upravljajo z dokumenti, je spletna storitev REST. Ta omogoča, da odjemalci preko vnaprej definiranih brezstanjskih operacij dostopajo do spletnih virov, predstavljenih v besedilni obliki. Za njen razvoj smo uporabili ogrodje**ASP.NET Core**. To je brezplačno in odprtokodno ogrodje za razvoj spletnih aplikacij [14]. Pri razvoju s tem ogrodjem smo uporabili vnaprej pripravljene funkcionalnosti, kot so nadzorni razredi za spletno storitev, preprosta možnost uporabe predpomnjenja, predpripravljeni razredi za uporabo standardnih protokolov, ORM *Entity Framework*, ki so nam omogočile hiter in voden razvoj spletne storitve.

Spletna storitev je zgrajena na arhitekturi mikro–storitev (*ang. micro-service architecture*). Mikro storitev je neodvisen proces, ki komunicira s pomočjo sporočil [13]. Modularnost mikro–storitev omogoča, da komponento poljubno urejamo in s tem zmanjšamo verjetnost, da bi vplivali na celoten program. Arhitektura mikro storitev je nasprotna monolitnim aplikacijam, v katerih so moduli med seboj odvisni in sklopljeni. Ogrodje se drži arhitekturnega vzorca, imenovanega model-pregled-nadzor (*ang. model-view-controller).* MVC aplikacijo loči na tri dele. Model skrbi za predstavitev podatkov v aplikaciji, pregled je zadolžen za prikaz podatkov, nadzor pa je zadolžen za procesiranje vhodnih podatkov in primeren odgovor. Ta vzorec med razvojem omogoča, da aplikacija ostaja pregledna, razširljiva in prepro-

sta za vzdrževanje.

Web API je ogrodje za izdelavo spletnih vmesnikov API. Dostopne točke so definirane v nadzornih razredih (*ang. Controller*). V teh razredih je nato mogoče definirati vse operacije na dostopnih točkah. V MVC lahko z atributi za vezavo določimo, iz katerega dela zahteve HTTP naj se vežejo atributi, recimo glava ali telo zahteve HTTP in v kakšnem formatu pričakujemo podatke, na primer JSON ali XML. To omogoča laži in hitrejši razvoj dostopnih točk spletne storitve.

2.3 Orodja in tehnologije za avtentikacijo in avtorizacijo

Gradnja novega sistema za overitev in pooblastitev, ki bi zadovoljila zahtevam, je, časovno in stroškovno neprimerna. To je razlog, da smo se odločili uporabiti že obstoječo rešitev, **Keycloak**, odprtokodni projekt organizacije RedHat. *Keycloak* (v nadaljevanju KC) je orodje za upravljanje dostopa in istovetnosti. Rešitev ponuja enkraten vpis v sistem (*ang. single sign-on*), kar omogoča, da se uporabnik vpiše z enim ID-jem in geslom, ki velja v več medsebojno povezanih neodvisnih sistemih. S KC smo v sistemu zagotovili pooblastitev, overitev in nadzor nad dostopom do virov. Orodje vsebuje nadzorno konzolo za upravljanje z domenami. Za vsako domeno je mogoče nastaviti odjemalce, vloge, uporabnike in skupine.

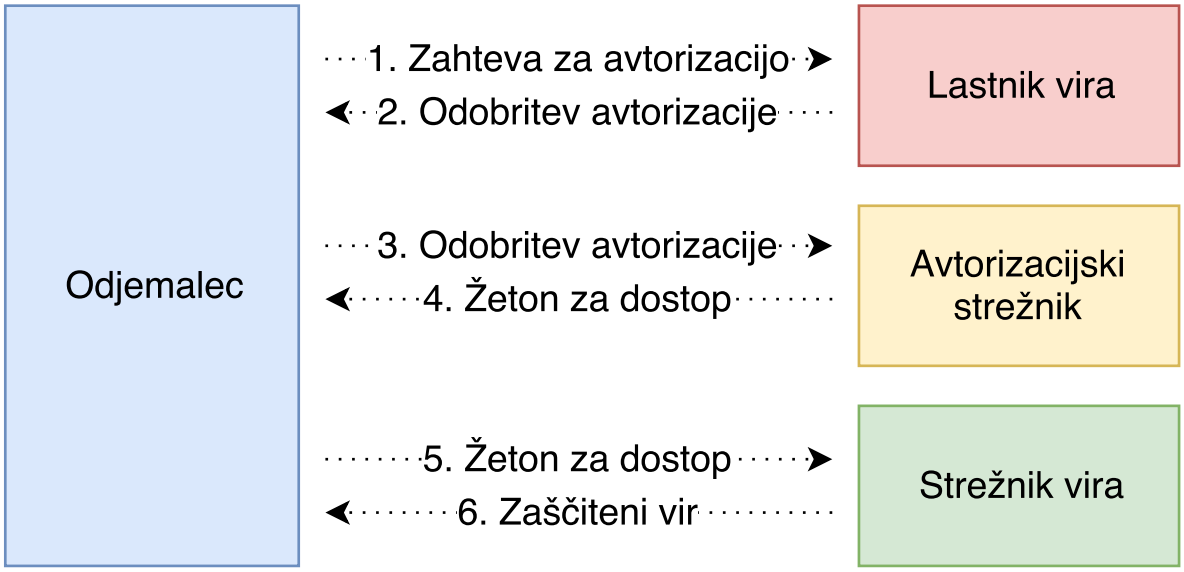
OAuth 2.0 je standard, ki aplikacijam tretjih oseb omogoča možnost omejenega dostopa do storitve HTTP [7]. To stori v imenu lastnika vira, z odobreno komunikacijo med lastnikom vira in storitvijo HTTP, ali pa preko tega, da aplikaciji iz tretje roke dovoli dostop v njenem imenu. OAuth definira naslednje vloge:

- Lastnik vira – je entiteta, ki odobri dostop do zaščitenega vira.
- Strežnik vira – je strežnik, ki nudi zaščitene vire. Strežnik se odziva na zahteve po zaščitenem viru preko žetona za dostop.

- Odjemalec – je aplikacija, ki dostopa do zaščitenega vira v imenu lastnika vira. Ta (lastnik vira) predhodno odobri dostop odjemalcu.

- Avtorizacijski strežnik – je strežnik, ki izdaja žetone odjemalcu za dostop do zašitenih virov na strani strežnika vira.

Na sliki 2.1 je prikazan delovni tok protokola OAuth.



Slika 2.1: Delovni tok protokola OAuth 2.0.

Naslednje oštevilčenje se ujema s tistim na sliki 2.1:

1. Odjemalec pošlje zahtevo za avtorizacijo lastniku vira. Zahteva je lahko narejena neposredno na lastnika vira ali posredno preko avtorizacijskega strežnika.

2. Odjemalec prejme odobritev avtorizacije (*ang. authorization grant*). To so poverilnice, ki prestavljajo avtorizacijo lastnika vira. Tip odobritve avtorizacije je odvisen od metode, ki jo uporabi odjemalec za zahtevo avtorizacije in od tipa, ki ga podpira avtorizacijski strežnik.

3. Odjemalec naredi zahtevo za dostopni žeton s tem, ko se avtenticira pri avtorizacijskem strežniku in mu predstavi pridobljeno odobritev avtorizacije v predhodnem koraku.

4. Avtorizacijski strežnik avtenticira odjemalca in preveri njegovo avtorizacijo. Če je veljavna, strežnik odjemalcu odobri žeton za dostop.

5. Z žetonom za dostop se nato odjemalec avtenticira na strežniku, ki hrani želeni vir.

6. Strežnik preveri veljavnost žetona. To stori tako, da pregleda vsebino žetona. Preveri njegovo časovno veljavnost, v primeru da je podpisan, njegov podpis ter veljavnost navedenega strežnika vira. Če je žeton veljaven, mu strežnik vira odgovori na zahtevo.

Po standardu so definirani štirje tipi poverilnic za pridobitev žetona za dostop: preko avtorizacijske kode, implicitno, z uporabo poverilnic lastnika vira in z uporabo poverilnic odjemalca. V spletni storitvi uporabljamo načina dostopa z uporabo poverilnic lastnika vira in poverilnice odjemalca.

Način z uporabo poverilnic lastnika vira (ponavadi končni uporabnik v aplikaciji) predpostavlja zaupanje med odjemalcem in lastnikom vira. Prednost tega tipa je, da odjemalcu ni treba shranjevati poverilnic lastnika vira, saj lahko uporabi žeton za dostop.

Tip dostopa s poverilnicami odjemalca se uporablja, kadar aplikacije in storitve želijo pridobiti žeton v svojem imenu. Odjemalec je v tem primeru tudi lastnik vira ali pa dostopa do zaščitenega vira, ki je predhodno definiran na avtorizacijskem strežniku.

Ostala dva načina sta uporabljena v primeru, da se spletna storitev razširi s spletno aplikacijo.

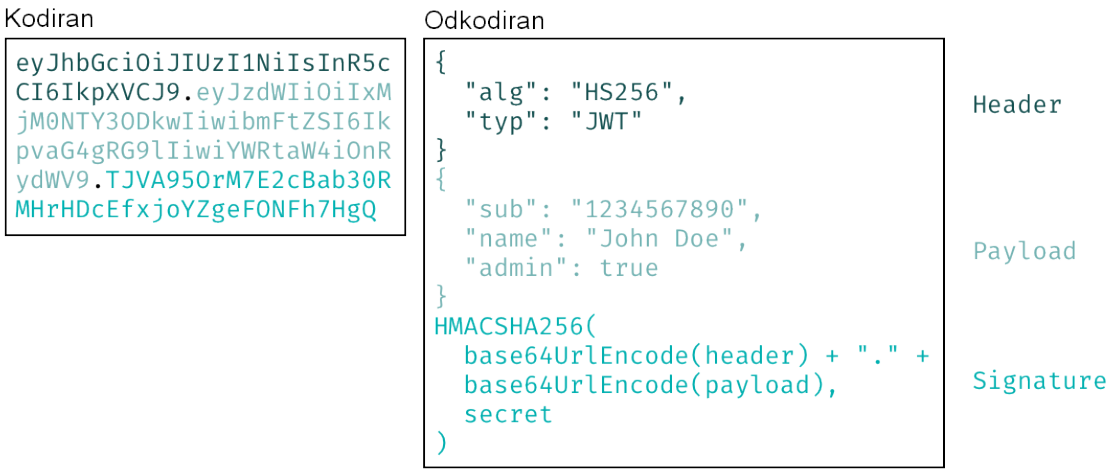
Če gre za strežniško aplikacijo, je uporabljen način dostopa z avtorizacijsko kodo. Namesto, da odjemalec naredi zahtevo po avtorizaciji neposredno do lastnika vira, odjemalec lastnika vira preusmeri na avtorizacijski strežnik. Ta overi odjemalca in pridobi njegovo pooblastitev. Lastnika vira avtorizacijski strežnik preusmeri nazaj k odjemalcu z avtorizacijsko kodo. To kodo

odjemalec uporabi za pridobitev žetona za dostop in ga uporabi za pridobitev želenega vira iz strežnika vira. Prednost tega načina je , da poverilnice lastnika vira niso nikoli deljene z odjemalcem in da ima dostop do žetona za dostop le odjemalec.

Implicitni način se od slednjega razlikuje po tem, da odjemalec od avtorizacijskega strežnika ne pridobi avtorizacijske kode, ampak takoj pridobi žeton za dostop. Ta tip avtorizacije je hitrejši od odobritve z avtorizacijsko kodo in je primeren za spletne aplikacije, ki delujejo na odjemalcih. Ta pristop je tudi nevarnejši, saj avtorizacijski strežnik ne avtenticira odjememalca, kar pomeni, da lahko napadalec prestreže in uporabi žeton za dostop.

OAuth 2 v primerjavi z načinom avtentikacije odjemalec-strežnik, kjer odjemalec zahteva dostop do zaščitenega vira na strežniku s poverilnicami lastnika zasčitenega vira, ponuja boljši nadzor nad poverilnicami lastnikov. Njegova zasnova znižuje verjetnost za možnost prevelike odobritve dostopa do zaščitenih virov in omogoča lažji preklic dostopa do virov aplikacijam iz tretje roke.

Spletni žeton JSON 2.2 (*ang. JSON Web token*) je odprt standard za prenos trditev (*ang. claims*) med dvema strankama.



Slika 2.2: Na levi strani je prikazan spletni žeton JSON, kodiran v formatu Base64, na desni pa je prikazan odkodiran žeton z glavo, koristnim delom in podpisom.

Trditve v žetonu so predstavljene v formatu JSON. Žeton je sestavljen iz glave, koristne vsebine (*ang. payload*) in podpisa. V glavi so podatki o tipu žetona in algoritmu za podpisovanje. V koristni vsebini so trditve – to so izjave o uporabniku in dodatni metapodatki. Glede na naveden algoritem iz glave žetona, so spletni žetoni JSON lahko podpisani s skupno skrivnostjo, ki jo poznata le pooblastitveni strežnik in odjemalska aplikacija, ali pa z uporabo certifikata X.509. S pomočjo podpisa lahko vedno potrdimo, da se integriteta podatkov ni spremenila in da so iz znanega vira.

OpenID Connect je dodatna raven nad protokolom OAuth 2, ki odjemalcem omogoča potrditev identitete končnega uporabnika, glede na pooblastitev avtorizacijskega strežnika [16]. Identiteta je množica atributov, ki so vezani na neko entiteto, na primer človeka, računalnika ali storitev. Protokol omogoča pridobitev informacij o uporabniku preko dostopnih točk protokola REST. Podatki o uporabniku so podani preko identitetnega žetona. Ta vsebuje podatke, kot so ID uporabnika, podatki o strežniku, ki je izdal žeton, seznam strank, ki lahko uporabi ta žeton, čas izdaje in rok trajanja žetona, ter opcijske podatke, kot je, na primer, e-mail. Identitetni žeton je kodiran kot žeton JWT. OpenID Connect omogoča uporabo povezane (*ang. federated*) identitete. To pomeni, da se uporabniki lahko overijo z isto identiteto v več aplikacijah. S tem se znebimo potrebe po razvoju dodatnih sistemov za overitev.

2.4 Orodja za delo s podatki

Elasticsearch (v nadaljevanju ES) je iskalnik, osnovan na odprtokodni knjižnici za iskanje in izbiranje informacij (*ang. information retrieval*) *Apache Lucene* [11] in namenjen iskanju po dokumentih in njihovih metapodatkih [4]. Omogoča iskanje preko protokolov HTTP in REST. Osnovna zgradba ES

temelji na pojmih indeks (*ang. index*), dokument (ang. document), polje (*ang. field*) in izraz (*ang. term*). Indeks je enota, v kateri so vsebovani dokumenti v formatu JSON. Dokument je sestavljen iz polj, medtem ko so polja sestavljena iz zaporedja izrazov, ki so podatkovnega tipa črkovni niz. Shema, zapisana v formatu JSON, v ES definira, kako so dokument in polja shranjeni in indeksirani. S shemo povemo, katera polja naj bodo obravnavana kot številke, datumi, besedila ipd. ES za delovanje uporablja strukturo, imenovano obratni indeks. Ta vrsta indeksa temelji na preslikavi vsebine posameznih dokumentov na ustrezne dokumente, v katerih se besede pojavijo. V tabeli 2.1 je prikazan primer dokumentov z vzorčnim besedilom.

Oznaka dokumenta	Vsebina
Dokument 1	While true na FRI bom pridno studiral
Dokument 2	While true jaz bom priden in programiral

Tabela 2.1: Primer vsebine dokumentov.

ES za izgradnjo obratnega indeksa iz teh dokumentov besedilo razdeli na posamezne besede in ustvari urejen slovar edinstvenih izrazov, ter seznam dokumentov, v katerem so ti izrazi. Primer za ta dva dokumenta je prikazan v tabeli 2.2. V prvem stolpcu so izrazi dokumentov, v drugem in tretjem stolpcu pa je z oznako „x“ zabeležena pojavnost besede v dokumentu.

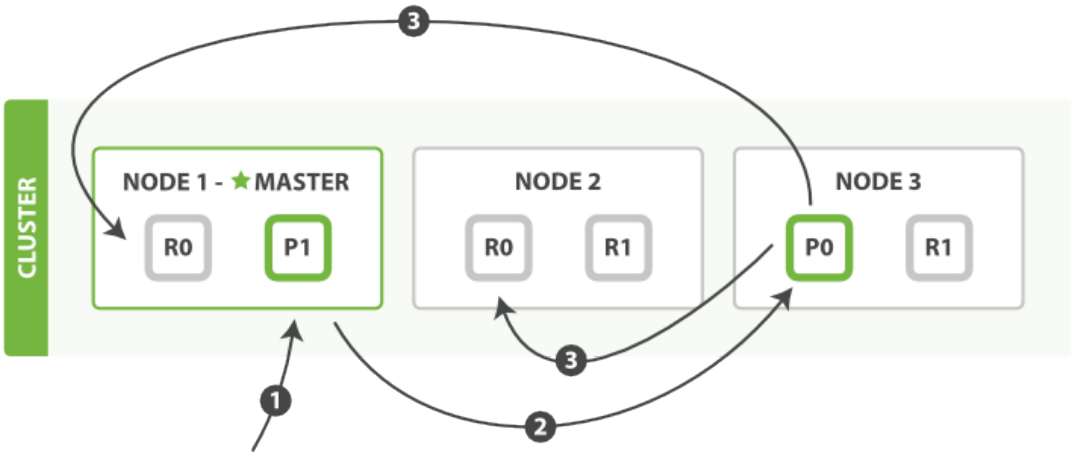
Izraz	Dokument 1	Dokument 2
bom	x	x
fri	x	
in		x
jaz		x
na	x	
priden		x
pridno	x	
programiral	x	
studiral		x
while	x	x
true	x	x

Tabela 2.2: Rezultat po izgradnji besednega vektorja za oba dokumenta

Ob iskalnem nizu z več izrazi, npr. *while jaz*, iskalnik poišče vse izraze v seznamu in kot rezultat prikaže dokumente z najdenimi iskanimi izrazi.

ES ima za potrebe iskanja razvit svoj lasten domenski poizvedovalni jezik (*ang. domain specific language*). Jezik je osnovan na jeziku JSON in omogoča poizvedbe po celotnem besedilu (*ang. full-text search*), kot tudi poizvedbe po posameznih poljih.

ES je dobro povečljiv. En strežnik, na katerem se izvaja iskalnik ES, predstavlja eno vozlišče, zbirka takih vozlišč pa se imenuje gruča. Na sliki 2.3 je prikazan primer gruče treh vozlišč, z vozliščem 1 (node 1) kot matičnim vozliščem in vozliščema 2 (node 2) in 3 (node 3), kot suženjskima vozliščema. Kot primer je v vozliščih en indeks. Indeks v ES sestavljajo manjši indeksi (v nadaljevanju podindeksi, *ang. shards*).



Slika 2.3: Primer izvajanja zahteve za več dokumentov na gručo v ES. Vir: [3]

Vsak podindeks je neodvisen od drugega podindeksa, kar omogoča, da gostimo podindeks na drugem vozlišču. S tem omogočimo vzporedno povečljivost prostora. V primeru je indeks sestavljen iz dveh osnovnih podindeksov, vsak osnovni indeks pa ima dve kopiji (*ang. replicas*). ES skrbi, da na enem vozlišču nimamo nikoli kopije podindeksa iz istega primarnega podindeksa.

Na sliki 2.3 je oštevilčeno zaporedje, ki se izvede v primeru zahteve po ustvarjanju, indeksiranju ali izbrisu več dokumentov naenkrat nad gručo. Ko zahteva prispe na vozlišče 1 (korak 1), se na njem ustvarita dve zahtevi v snopu, ena za vsak osnovni podindeks (korak 2). Na primarnih indeksih se zahteve izvedejo ena za drugo. Če so zahteve uspešno izvedene, se izvede nova zahteva na njihove replike (korak 3) izvede vzporedno. Ko kopije podindeksov javijo uspeh za vse posamezne akcije matičnemu vozlišču, ta vrne odgovor odjemalcu.

Poglavje 3

Arhitektura sistema

Cilj tega poglavja je predstaviti splošne značilnosti dobre arhitekture in staro ter novo arhitekturo. V delu, kjer govorimo splošno o arhitekturi, je opisano, na kaj moramo biti pozorni pri načrtovanju arhitekture in načini predstavitve arhitekture. Pri opisu stare in nove arhitekture je pozornost namenjena predstavitvi komponent in opisu primera transakcije v obeh arhitekturah. Pri novi so predstavljeni tudi opisi zahtevanih funkcionalnih zahtev, ki jih prinaša nova arhitektura.

3.1 Splošno o arhitekturi

Pri načrtovanju programskih rešitev je dobra arhitektura prvi korak. Z arhitekturo določimo glavne komponente sistema in povezave med njimi. Arhitektura programske opreme vpliva na izvajanje, robustnost, distribucijo in vzdrževanje posameznega sistema. Zahteve v sistemu so pogosto povezane z njeno arhitekturo. Z dobro arhitekturo lahko naredimo boljšo specifikacijo zahtev. Pri načrtovanju arhitekture naredimo takšno organizacijo sistema, ki zadovolji vse funkcionalne in nefunkcionalne zahteve.

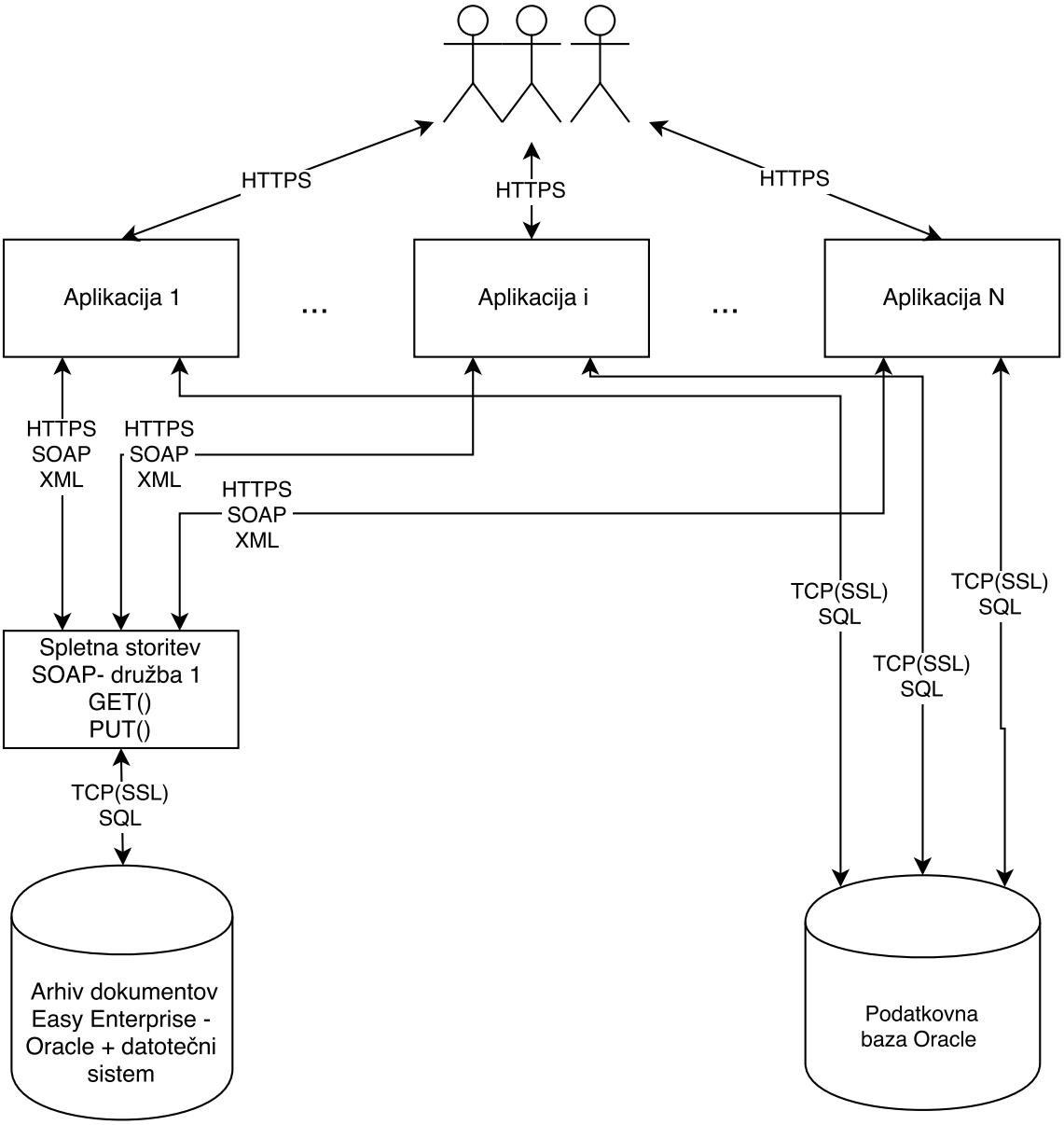
Ločimo dva načina načrtovanja arhitekture. Prvi se nanaša na arhitekturo posameznih programov in na komponente tega programa. Drugi se nanaša na arhitekturo kompleksnih poslovnih sistemov. Ti se povezujejo z ostalimi

sistemi in njihovimi programskimi komponentami.

Osnova za gradnjo storitveno orientirane arhitekture (*ang. service-oriented architecture*) so storitve, ki se izvajajo na porazdeljenih sistemih.

Storitev je šibko sklopljena programska zbirka ene ali več funkcionalnosti, ki je na voljo več odjemalcem za ponovno uporabo [18]. Cilj je razviti storitve, ki so ponovno uporabne v več različnih sistemih. SOA razvijemo tako, da najprej identificiramo želene storitve, načrtamo njihove vmesnike (REST, WSDL) in jih implementiramo. Temu sledita še testiranje pravilnosti delovanja in prenos v produkcijsko okolje.

3.2 Stara arhitektura



Slika 3.1: Postavitev stare arhitekture.

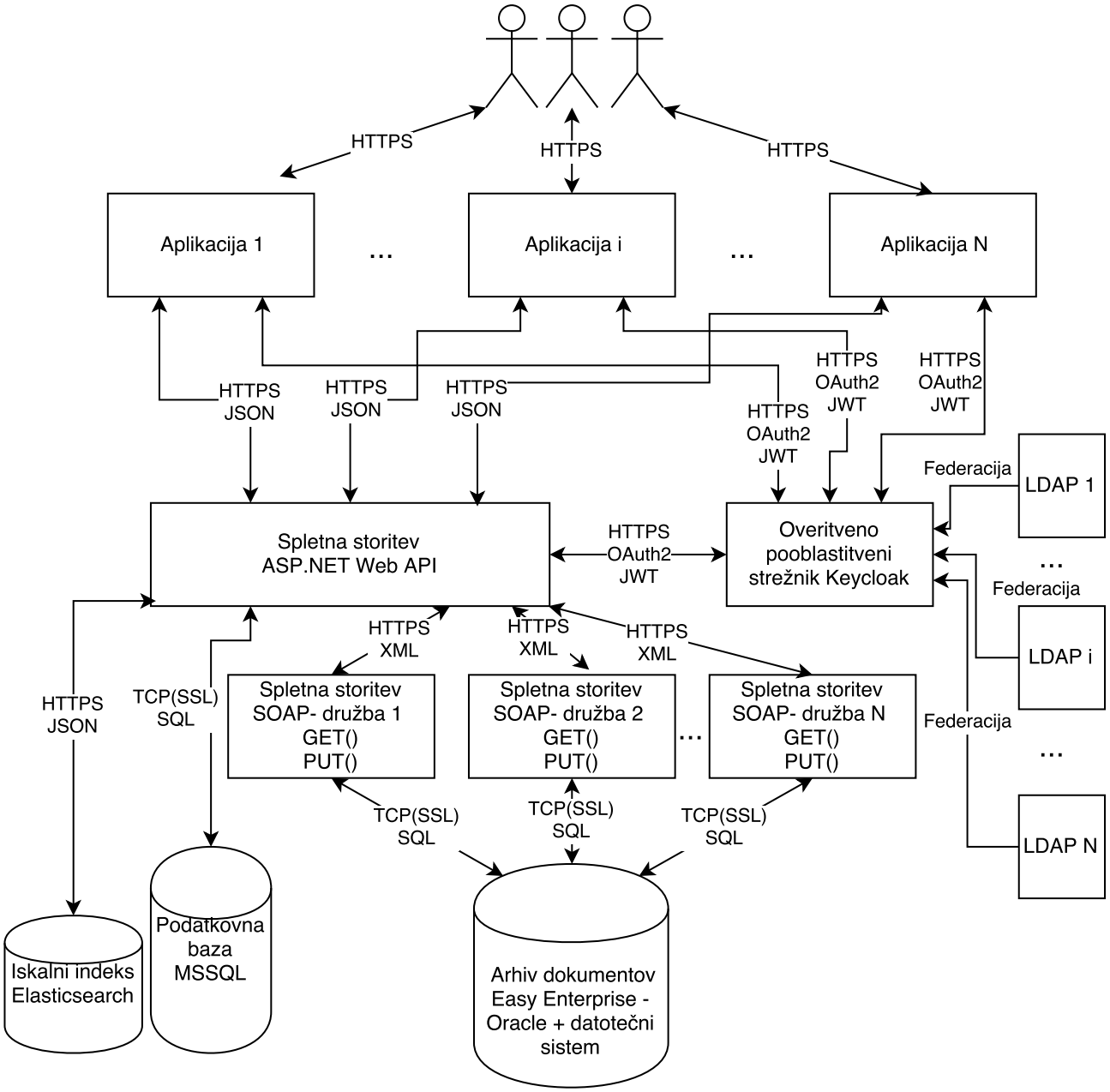
Na sliki 3.1 je prikazana postavitev stare arhitekture v eni izmed družb. Komponente arhitekutre so arhiv EEX, odjemalske aplikacije, tabela v podatkovni bazi Oracle, spletna storitev SOAP za vlaganje in prevzemanje

dokumentov iz arhiva EEX [2].

Vlaganje dokumenta v arhiv poteka na naslednji način. Komunikacija med aplikacijo in arhivom poteka s pomočjo protokola SOAP in podatkovnega formata XML. Pri vlaganju dokumenta moramo, glede na shemo WSDL v dodatku B.1, pripraviti sporočilo SOAP. Ena operacija je namenjena vlaganju dokumenta, na shemi B.1 je označena s *putSava* (B.1, 13–20). Ko naredimo zahtevo HTTP na to dostopno točko s sporočilom, ki vsebuje metapodatke o imenu, tipu (npr. PDF), področju (npr. pozavarovanje), lastniku in klasifikaciji dokumenta (npr. obračun), ter njegovi vsebini v formatu base64, kot odgovor (B.1, 21–25) prejmemo sporočilo o uspešnosti transakcije in ključ dokumenta. Ključ in ostale metapodatke aplikacija shrani v Oraclovo bazo.

Ko želi aplikacija dostopati do dokumenta v arhivu EEX, uporabi ključ za dostop do želenega dokumenta v arhivu dokumentov. Za dostop do dokumenta uporabi dostopno točko v spletni storitvi za pridobivanje dokumenta, na B.1 je zapisana kot *getSava* (B.1, 27–29). Pri zahtevi preko protokola HTTP aplikacija doda metapodatek s ključem dokumenta v sporočilo zahteve. Spletna storitev vrne odgovor (B.1, 30–39) z vsebino dokumenta in vsemi pripadajočimi metapodatki podanimi ob vlaganju dokumenta.

3.3 Nova arhitektura



Slika 3.2: Postavitev nove arhitekture.

Na sliki 3.2 je prikazana postavitev nove arhitekture. Puščice predstavljajo prenos podatkov med komponentami. Komponenta, ki smo jo v celoti ohranili iz stare arhitekture, je arhiv dokumentov EEX. V novi arhitekturi so dodane komponente spletna storitev *Web API*, avtorizacijski strežnik KC, strežniki LDAP, obstoječe spletne storitve SOAP posameznih družb (te so med seboj enake, le dostopajo do drugega bazena) in iskalni indeks ES.

Iz slike lahko razberemo ključne značilnosti nove arhitekture. Uporabniki za dostop do arhiva dokumentov komunicirajo izključno s spletno storitvijo *Web API* in KC. Jedrni del arhitekture predstavlja spletna storitev *Web API*, ki je zadolžena za shranjevanje podatkov o dokumentih v podatkovno bazo (MSSQL in ES) in v arhiv ter njihov prevzem. *Web API* za komunikacijo z arhivom glede na pripadnost uporabnika določeni družbi (podatek za to je v KC oziroma v žetonu za dostop) izbere ustrezno spletno storitev SOAP. Da preverimo, ali je uporabnik upravičen do uporabe spletne storitve *Web API*, v arhitekturi uporabimo KC. Spletna storitev preveri uporabnikov žeton pri avtorizacijskem strežniku in glede na uporabniške pravice, definirane v nadzorni konzoli KC, odobri ali zavrne dostop.

Arhitektura je razširljiva v treh dimenzijah: v aplikacijah, uporabnikih in družbah. Aplikacije lahko poljubno dodajamo ali odvzemamo s pomočjo nadzorne konzole v KC. KC je zadolžen tudi za prenos podatkov o uporabnikih in uporabniških skupinah iz strežnikov LDAP. Če želimo dodati novo družbo, dodamo že obstoječo spletno storitev in popravimo nastavitve za vlaganje v ustrezen bazen v arhivu.

Na prvi pogled se zdi, da imamo v tej arhitekturi redundantne komponente – spletna storitev *Web API* bi lahko, namesto spletnih storitev SOAP, neposredno dostopala do arhiva EEX. V tem delu smo se skupaj z nadrejenimi odločili, da je zaradi kompleksnega dostopa do arhiva EEX, najpreprosteje uporabiti že obstoječe spletne storitve SOAP.

V primeru, da gre za vlaganje dokumenta, imamo tri točke, kamor zapišemo podatke. Prva je arhiv dokumentov EEX. Sem se zapiše celoten dokument in osnovni metapodatki, na enak način, kot pri vlaganju dokumenta v stari arhi-

tekturi. Druga je iskalni indeks ES. Vanj zapišemo metapodatke dokumenta brez dokumenta v formatu base64. Tretja točka, kamor shranimo metapodatke, tudi tokrat brez dokumenta base64, je podatkovna baza MSSQL. Služi za beleženje relacij med zbirkami v spletni storitvi *Web API*.

V naslednji sekciji so predstavljene zbirke v spletni storitvi *Web API*, v novi arhitekturi.

3.4 Zbirke v spletni storitvi *Web API*

Uporabniki do dokumentov v SD dostopajo preko spletne storitve *Web API*. Vmesnik spletne storitve ima dostopne točke, s katerimi je mogoče izvajati operacije nad dokumenti. Celoten seznam trenutno implementiranih dostopnih točk je prikazan na sliki E.1. Poleg metapodakov za dokumente (na sliki E.1 označeno *document*) spletna storitev omogoča upravljanje z metapodatki in relacijami za klasifikacijske načrte (*classification plan*), družbe (*company*) in navidezne mape (*folder*).

Dostopne točke trenutno omogočajo akcije CRUD za vsako vrsto zbirke, iskanje po metapodatkih dokumenta s pomočjo poizvedovalnega jezika za ES, povezovanje med dokumenti, dodajanje dokumenta v mapo, izpis različic dokumentov in dodajanje relacije dokumenta in mape na klasifikacijski načrt. Zbirke v spletni storitvi so predstavljene z razredi *C#*, na tak način, da podpirajo delo z ogrodjem *Entity Framework*.

Klasifikacijski načrt, izsek kode C.1, je načrt družbe o času hranjenja dokumenta. Z njim definiramo skupine dokumentov, stopnjo zaupnosti dokumentov in privzeti dostop do dokumentov. Vsebuje podatke o tipu dokumenta (C.1, 10), obvezni dobi hranjenja (C.1, 11) in dobi, po kateri je treba ta tip dokumenta obvezno uničiti (C.1, 12). Klasifikacijski načrt upravlja skrbniška aplikacija (C.1, 13–14). Ta upravlja s klasifikacijskim načrtom in vanj zapisuje seznam vlog (C.1, 18), ki lahko dostopajo do zbirk dokumentov. Vsak klasifikacijski načrt je, ob ustreznih pravicah za dostop, mogoče povezati z dokumenti (C.1, 16) in navideznimi mapami (C.1, 17), družba pa

se določi samodejno glede na vloge v žetonu za dostop.

Vsaka **družba** C.2 znotraj skupine SavaRe ima svoj bazen v arhivu EEX, kamor se vlagajo dokumenti. Z relacijo na družbo definiramo, v kateri bazen (C.2, 12) družbe (C.2, 11) naj se vloži dokument, katere klasifikacijske načrte ima družba v lasti (C.2, 14) in kateri je identifikator družbe v KC (C.2, 13) za avtomatsko povezovanje klasifikacijskega načrta (s tem tudi dokumentov in map) na ustrezno družbo.

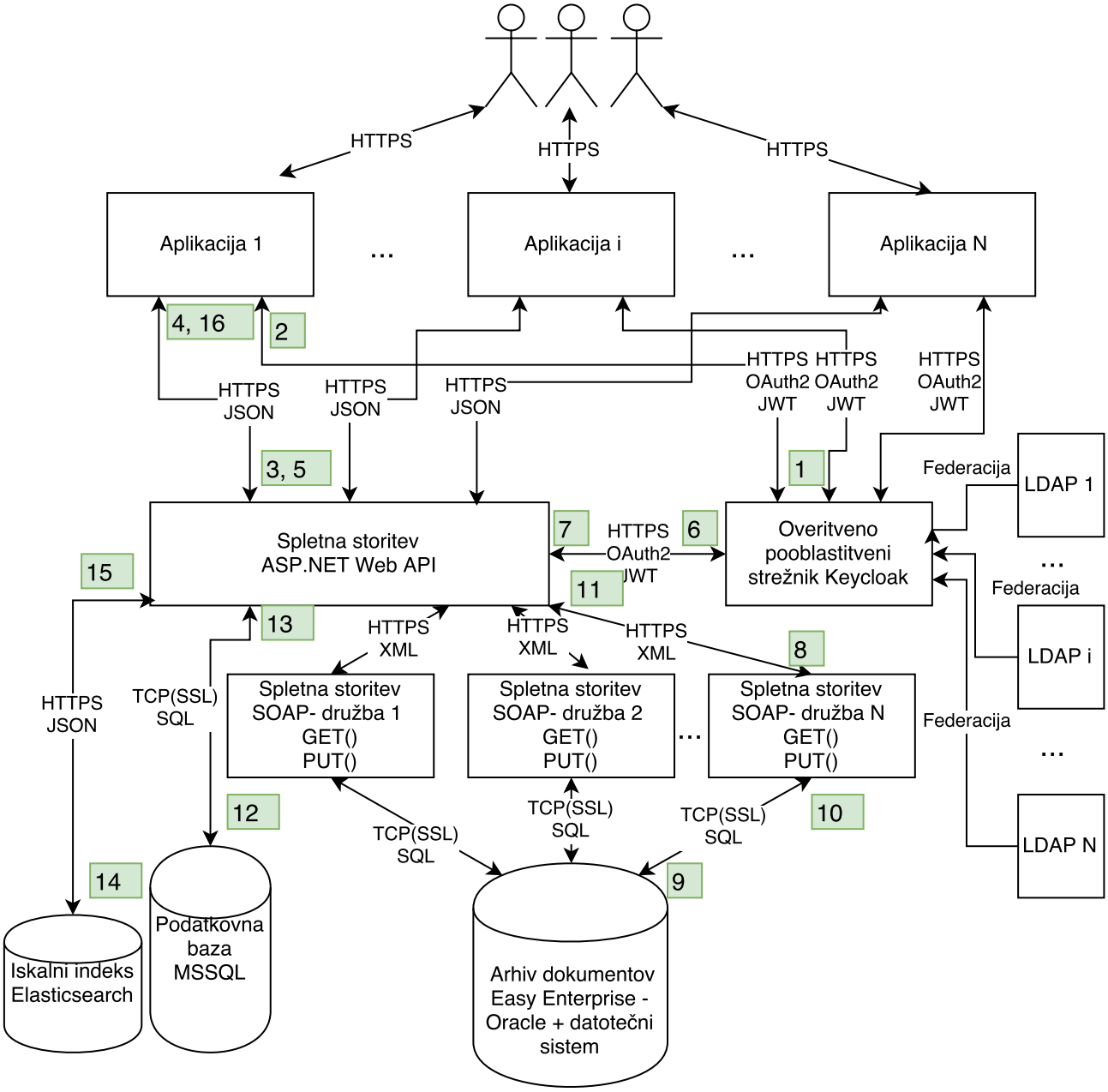
Zbirka **dokument** C.3 predstavlja metapodatke dokumenta v SD. V zbirko shranjujemo podatke, ki so obvezni za zapis v arhiv dokumentov (glej opis vlaganja dokumenta v stari arhitekturi 3.1), dodatno pa hranimo podatke o črtni kodi dokumenta (C.3, 13), dodatnih poljubnih metapodatkih (C.3, 16), ključu v arhivu (C.3, 17), različici (C.3, 18), statusu elektronskega podpisa (C.3, 19), datumu ustvarjanja in spremembe (C.3, 20–21), uporabniku (C.3, 22–23) in seznamu vlog za izjemni dostop do dokumenta (C.3, 24). Zbirka je v razredu *C#* definirana tako, da podpira ustvarjanje relacij otrok–starš za zbirko dokument (C.3, 28–29), relacije za vlaganje dokumentov v navidezne mape (C.3, 30), relacije na klasifikacijski načrt (C.3, 31) in družbo (C.3, 32) ter možnost ustvarjanja različic dokumentov z zapisom spremembe dokumenta v podatkovno bazo (C.3, 33).

S konceptom **navideznih map** C.4 upravljamo s skupinami dokumentov. Vsaka taka mapa vsebuje podatke o imenu mape (C.4, 11), času ustvarjanja in spremembi mape (C.4, 12–13), poljubne metapodatke o mapi (C.4, 14), podatke o uporabniku in seznamu vlog za izjemen dostop do mape (C.4, 15–16), starševskih in otroških mapah (C.4, 17–18) ter povezanih dokumentih (C.4, 19). Podobno, kot dokumente, mape povežemo s klasifikacijskim načrtom (C.4, 20).

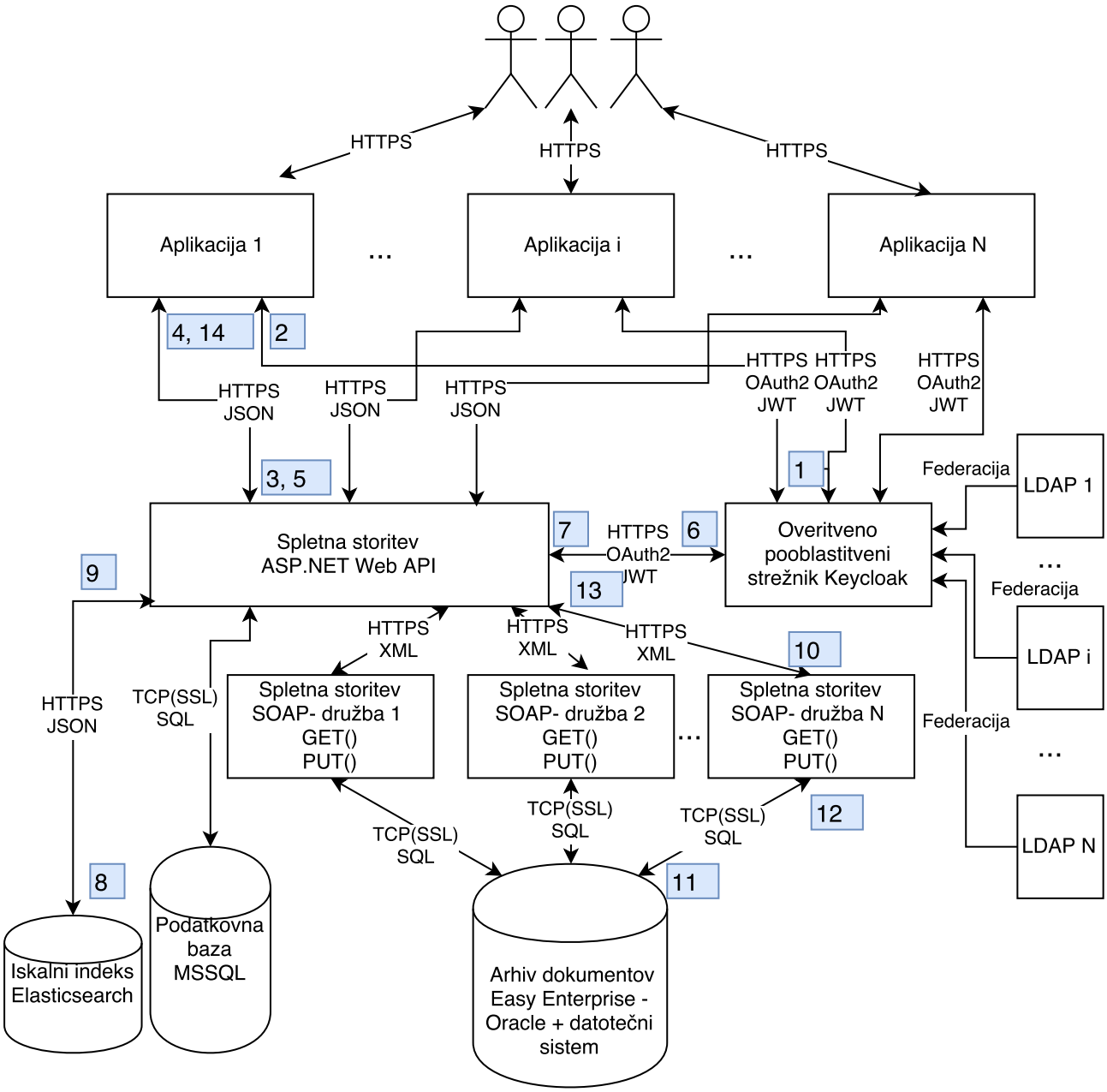
3.5 Potek transakcij v novi arhitekturi

Za predstavo delovanja celotne arhitekture, zadošča vpogled v delovanje dostopnih točk spletne storitve *Web API* za vlaganje in prevzem dokumenta v

arhiv dokumentov. V pomoč, pri opisu njunega delovanja, bosta sliki nove arhitekture (3.3, 3.4), na katerih sta oštevilčena vrstna reda poteka obeh transakcij, na katera se bom skliceval v besedilu.



Slika 3.3: Vrstni red pri vlaganju dokumenta v novi arhitekturi.



Slika 3.4: Vrstni red pri prevzemu dokumenta v novi arhitekturi.

3.5.1 Vlaganje dokumenta

Primer poteka vlaganja dokumenta v arhiv dokumentov je naslednji. Uporabnik pošlje zahtevo za dostopni žeton na dostopno točko avtorizacijskega strežnika za pridobitev žetona (3.3, 1). Za tip odobritve avtorizacije uporabimo odobritev s poverilnicami odjemalca. V glavi zahteve se pošljeta ID odjemalca (*ang. client id*) in skrivnost odjemalca (*ang. client secret*). Avtorizacijski strežnik vrne žeton za dostop skupaj s podatki o času veljavnosti, žetonom za osvežitev (*ang. refresh token*) in tipom žetona (*ang. token type*) (3.3, 2). Drugi pomemben žeton, ki ga je treba pridobiti za vlaganje dokumenta, je **žeton CSRF**. Tega prejmemo v glavi odgovora spletne storitve *Web API* (3.3, 4), ko ji pošljemo žeton za dostop (3.3, 3).

S pridobljenima žetonoma imamo pripravljene vse parametre, da lahko začnemo vlaganje dokumenta v SD. V telo zahteve odjemalec poda metapodatke poslanega dokumenta v formatu JSON in pošlje zahtevo na dostopno točko storitve za vlaganje dokumenta (3.3, 5).

Zahteva se najprej preusmeri na KC (3.3, 6), kjer se, na osnovi žetona za dostop, preveri, ali ima odjemalec za želeni vir (dostopno točko) ustrezne dostopne pravice (za nastavitev glej dodatek A.1). Po odobrenem dostopu KC preusmeri zahtevo nazaj na spletno storitev *Web API* (3.3, 7), kjer se v ogrodju preveri še ujemanje žetona CSRF.

Ko zahteva prispe do spletne storitve, se v metodi, zadolženi za njeno procesiranje, pregledajo metapodatki v telesu zahteve. Spletna storitev ima dodano potrjevanje vhodnih metapodatkov dokumenta, ki jih prejmemo od odjemalca. Vsak ključ, v objektu JSON, ima nato še dodatno potrjevanje, kjer se podrobno definirajo dovoljene vrednosti. V primeru D.2 potrjevanja za zbirko dokument, smo uporabili metode *NotEmpty()* (lastnost je obvezna), *WithMessage()* (privzeto sporočilo ob neveljavnem metapodatku) in *Must()* (lastnost mora ustrezati pogoju). Z njimi smo definirali pravila za ustreznost prejetih metapodatkov dokumenta in javljanje morebitnih nepravilnosti odjemalcu.

V primeru, da je potrjevanje metapodatkov uspešno, metoda v spletni

storitvi preveri, ali metapodatki za ustvarjanje relacij za vložen dokument, ustrezajo njegovim dostopnim pravicam. Tu se preverja, na primer, ali lahko odjemalec doda dokument v mapo (relacija dokumenta na mapo), ki jo je navedel v poslanih metapodatkih. To se preverja tako, da se pregledata seznam vlog v povezanih klasifikacijskih načrtih in seznam vlog na zbirkah.

Sledi zahteva HTTP na spletno storitev za vlaganje v arhiv dokumentov (3.3, 8). Na osnovi vloge odjemalca, določene v Keycloak, spletna storitev *Web API* ve, katero spletno storitev SOAP mora uporabiti, ta pa zapiše vsebino dokumenta v ustrezen bazen v arhivu (3.3, 9). Spletna storitev vrne odgovor spletni storitvi *Web API* o uspešnosti vlaganja s ključem dokumenta – enako kot v stari arhitekturi (3.3, 10–11).

V primeru, da je vlaganje dokumenta v arhiv uspešno, se metapodatki zapišejo v podatkovno bazo MSSQL (3.3, 12). Pri zapisu metapodatkov dokumenta v bazo pridobimo ID dokumenta (3.3, 13). S tem ID-jem dostopamo do spletne storitve ES, kamor pošljemo metapodatke o vloženem dokumentu (3.3, 14).

Spletna storitev *WEB API* se na spletno storitev REST iskalnika ES povezuje s pomočjo protokola HTTP, format sporočila v telesu zahteve pa je JSON.

V odjemalcu HTTP za povezavo na spletno storitev ES smo opredelili metode CRUD za komunikacijo s spletno storitvijo ES. V izseku kode D.1 je primer metode, ki je zadolžena za vlaganje in posodabljanje dokumenta v iskalni indeks ES. Metoda kot argument (D.1, 1) prejme objekt z metapodatki dokumenta, ki jih je odjemalec poslal na spletno storitev *Web API*. Nato se kliče asinhrona metoda za vlaganje dokumenta v ES (D.1, 3). Kot argument sprejme podatek o ID-ju dokumenta in prejete metapodatke. Sledi odgovor HTTP spletne storitve ES s podatki o uspešnosti vlaganja dokumenta v ES (3.3, 15).

V primeru, da je bilo tudi vlaganje v ES uspešno, spletna storitev *Web API* ID dokumenta vrne odjemalcu (3.3, 16).

3.5.2 Prevzem dokumenta

Odjemalec želeni dokument prevzema s pomočjo ID-ja, ki ga je pridobil pri vlaganju dokumenta. ID doda kot argument v naslov dostopne točke spletne storitve *Web API* za prevzem dokumenta (glej dostopno točko */api/v1/fullDocumentid* na sliki E.1). V glavo zahteve HTTP doda veljaven žeton za dostop in žeton CSRF ter pošlje zahtevo na spletno storitev *Web API* (postopek pridobitve žetonov in njihovega preverjanja je enak, kot pri vlaganju dokumenta).

Spletna storitev ima na dostopnih točkah za prevzem dokumenta dodano **predpomnjenje**. Namesto pridobivanja metapodatkov dokumenta preko povezave na bazo in dostopa do diska, jih shranimo v predpomnilnik spletne storitve. Pri obdelavi zahteve se najprej preveri, ali je objekt shranjen v predpomnilniku. Šele v primeru zgrešitve predpomnilnika program poišče podatek v bazi. V predpomnilnik se podatki shranijo v obliki ključ–vrednost. V spletni storitvi *Web API* se predpomnjenje doda kot mikro storitev, do predpomnilnika pa se nato dostopa s pomočjo razreda *IMemoryCache*.

V dodatku je prikazan izsek kode D.3, kjer objekt v predpomnilnik dodajamo s pomočjo omenjenega razreda. Z metodo *TryGetValue* (D.3, 2) preverimo, ali je objekt že v predpomnilniku. V primeru, da ne (D.3, 4), se izvede poizvedba na iskalnik ES (3.4, P8). V primeru, da poizvedba vrne zadetke (D.3, 9), s pomočjo metode *Set* (D.3, 15) nastavimo ključ in vrednost objekta v predpomnilniku.

Objekt ostane v predpomnilniku do zapolnitve spomina (to je privzeta nastavitev – za to skrbi ogrodje *ASP.NET Core*) ali do njegovega preklica. Preklic predpomnilnika se izvede ob vsakem klicu na dostopne točke spletne storitve, ki sprožijo spremembo stanja metapodatkov v spletni storitvi. Za zamenjavo objekta uporabimo metodo *Set*, za izbris pa *Remove*. Da lahko izvedemo preklic predpomnilnika na želenem objektu, pri klicu obeh metod, kot argument, podamo ID dokumenta v spletni storitvi *Web API*,

V izseku kode D.4 je metoda za tvorbo poizvedbe na spletno storitev ES, ki kot parameter dobi ID dokumenta, vrne pa metapodatke dokumenta

iz iskalnika ES. V poizvedbo se dodajo ustrezni filtri, ki upoštevajo vloge odjemalca. Te se preberejo iz žetona za dostop v metodi *getAclTerms* (D.4, 3).

Za filtriranje dokumentov, glede na vloge odjemalca, smo uporabili poizvedovalni jezik ES. Uporabili smo poizvedbo tipa *bool*. Poizvedba tega tipa omogoča sestavljanje poizvedb z logičnim operatorjem ekvivalentnimi izrazi (in-must, ali-should, negacija-must_not). Če indeks vsebuje dokument s podanimi vlogami, bo spletna storitev ES vrnila ustrezne dokumente s pripadajočimi metapodatki (3.4, 9).

Iz metapodatkov dobimo ključ dokumenta v arhivu EEX in podatek o povezani družbi, ki ji pripada dokument. Podatek o družbi služi za izbor spletne storitev SOAP za prevzem dokumenta iz pripadajočega bazena v družbi. Prevzem poteka na enak način, kot v stari arhitekturi (3.4, 10–13).

Vsebina dokumenta iz arhiva se doda k obstoječim metapodatkom ES, ki se kot odgovor HTTP z vsebino v formatu JSON vrnejo odjemalcu.

3.6 Večkriterijsko iskanje po dokumentih

Spletna storitev *Web API* vsebuje dostopno točko, preko katere lahko izvajamo večkriterijsko iskanje po dokumentih. Na sliki E.1 je zapisana kot */api/v1/_search*. Poizvedba se izvede na enak način, kot če bi delali poizvedbo neposredno na spletno storitev ES. Uporabnik mora, tako kot pri ostalih zahtevah na spletno storitev, v glavi zahteve HTTP podati žeton za dostop in žeton CSRF, telo zahteve pa mora vsebovati veljavno poizvedbo v poizvedovalnem jeziku ES. Dostopna točka podpira vse načine iskanja, ki so podprti v ES. To so, na primer, iskanje po celotnem besedilu, iskanje po enem ali več metapodatkih, iskanje po razponu metapodatka, iskanje po predponi in možnost kombiniranja iskalnih načinov.

Poizvedba mora biti v obliki *bool*, da lahko v metodi spletne storitve dodamo ustrezne filtre za avtorizacijo. Po dodatku filtrov se, preko protokola HTTP, posreduje poizvedbo na spletno storitev ES, iz katere *Web API* dobi

odgovor z rezultatom iskalne poizvedbe in ga vrne odjemalcu.

3.7 Nadzor dostopa do virov

KC skupaj s strežniki LDAP tvori federacijo (*ang. federation*). To pomeni, da se iz strežnikov LDAP obstoječi podatki o uporabnikih prenesejo v lokalno podatkovno bazo KC. Avtorizacijski strežnik ima možnost periodične sinhronizacije uporabnikov iz strežnikov LDAP.

KC ima več načinov upravljanja s preklicem predpomnilnika. Privzeto je izbrana možnost dnevnega preklica predpomnilnika, ob točno določeni uri. KC podpira še tedenski preklic predpomnilnika, možnost določitve največje življenjske dobe za uporabnika v predpomnilniku in izklop predpomnilnika.

Ker so podatki shranjeni v lokalni KC podatkovni bazi, se zmanjša obremenitev strežnikov LDAP. Ti so odgovorni le za avtentikacijo uporabnikov. Pri tem strežnik KC podpira preprosto avtentikacijo (*ang. basic authentication*), pri čemer strežniku LDAP pošlje uporabnikovo razločevalno ime in geslo. Ker geslo ni zakrito, je uporaba protokola TLS pri tej komunikaciji obvezna.

Uporabnikom se v KC določijo družbe in vloge (za podrobnosti nastavitev glej dodatek A). Vloge in družbe so nato vidne v žetonu za dostop, kar spletna storitev *Web API* uporabi za avtorizacijo dostopa do dokumentov.

Vloge so v spletni storitvi *Web API* zapisane na dveh mestih. Obvezno so zapisane v klasifikacijskem načrtu, opcijsko pa tudi na dokumentu. Vloge na klasifikacijskem načrtu služijo kot privzete za skupino dokumentov, ki so povezani s tem klasifikacijskim načrtom. To pomeni odobritev dostopa, v primeru, da je vloga odjemalca za dostop na žetonu za dostop in na klasifikacijskem načrtu, ki je povezan z dokumentom, do katerega želimo dostopati.

Na primer, za dokument tipa „račun“ imamo v KN seznam vlog za dostop „tehnika“ in „računovodstvo”. Če ima odjemalec dodeljeno eno od vlog „tehnika“ ali „računovodstvo”, mu je dostop odobren.

Recimo, da imamo primer, ko želimo dostop do posameznega dokumenta

„račun“ dodeliti nekomu, ki nima vloge *tehnika* ali *računovodstvo*. Gre torej za poseben primer, ko želimo omogoči izreden dostop do dokumenta tudi za drugo vlogo, ki ni privzeto v klasifikacijskem načrtu. Primer uporabe je, če želimo izdati dokument tipa „račun“ točno določeni osebi.

Za reševanje takih primerov imajo skrbniške aplikacije možnost zapisa vloge, kot metapodatek, na dokument. Poleg pregleda vlog iz žetona na KN, se v spletni storitvi vedno opravi tudi pregled vlog, podanih v metapodatkih dokumenta.

Na enak način je dostop omejen tudi za navidezne mape.

3.8 Pregled vpogledov v dokumente in njihovih sprememb

V SD se vodi evidenca vpogledov v dokumente. Ta se opravlja na dva načina. Prvi je z **zapisovanjem vseh aktivnosti uporabnikov v dnevniško datoteko**. Za urejanje formata zapisov v tej datoteki uporabljamo knjižnico *Nlog*.

Knjižnica omogoča, da na osnovi zaporedja dogovorjenih ključev za želene vrednosti pripravimo format zapisa v dnevniško datoteko. V spletni storitvi zapisujemo podatke točen čas vpogleda na dokument, metodo, ki se je izvedla v nadzornem razredu, IP naslov odjemalca, uporabniško ime, rezultat zahteve na dostopno točko in komentar. Knjižnica omogoča tudi povezavo z atributom *HttpContext.TraceIdentifier*, ki predstavlja ID zahteve v spletni storitvi *Web API*. Tako lahko lažje sledimo poteku posamezne zahteve in pripravimo analizo delovanja spletne storitve.

Drugi način je, z **zapisom sprememb metapodatkov pri posodabljanju dokumenta**. Z uporabo knjižnice *JsonDiffPatch* iz metapodakov dveh različic dokumentov, v obliki JSON, pridobimo seznam razlik v metapodakih. Seznam sprememb je zapisan v obliki popravek JSON (*ang. JSON Patch*). Popravek JSON (primer je v izseku D.6) je dokumentna struktura za ponazoritev operacij, ki jih izvedemo nad dokumentom v obliki JSON

[1]. Možne akcije so dodaj, odstrani, zamenjaj, premakni, kopiraj in testiraj (za enakost). Seznam sprememb in metapodakte starega dokumenta spletna storitev Web API zapiše v podatkovno bazo.

Za prikaz različic dokumentov moramo narediti klic na dostopno točko /api/v1/{id}/version/{version}, kjer kot argument podamo ID dokumenta in številski atribut za različico dokumenta. Število za različico pomeni odmik od trenutne različice. Če želimo dostopati do različice dokumenta za dve spremembi nazaj, je število „2“ parameter, ki ga pošljemo v naslovu zahteve REST. V metapodatkih imamo zapisano tudi referenco za pridobitev dokumenta iz arhiva dokumentov, s katero lahko spletna storitev vrne ustrezno različico dokumenta iz arhiva in ustrezne metapodatke iz podatkovne baze spletne storitve.

3.9 Vzdržljivost spletne storitve *Web API*

Spletna storitev *Web API* je preverjena s testi modulov in integracijskimi testi. V času pisanja besedila so s testi modulov preverjene dostopne točke CRUD za zbirko dokument. Testi modulov so pripravljeni s pomočjo objektov za oponašanje (*ang. mock*). Ti objekti oponašajo delovanje dejanskih objektov in s tem omogočajo od njih neodvisno testiranje. Glede na želeni test, jim določimo privzete vrednosti za vsako lastnost objekta.

Primer nastavitve takega objekta je v izseku D.5, kjer nastavimo oponašanje za razred *IMemoryCache*, ki smo ga spoznali v poglavju 3.5.1. V izseku kode pripravimo novo instanco objekta za oponašanje (D.5, 1). Sledi nastavitev odgovorov za oponašanje metod za dodajanje (D.5, 3–6) in pridobivanje (D.5, 7–10) objekta v predpomnilnik.

Za testiranje delovanja spletne storitve uporabljamo orodje za testiranje API-jev imenovano Postman. Trenutno imamo za vsako dostopno točko v spletni storitvi predpripravljene primere zahtev HTTP. Z njimi lahko hitro testiramo delovanje storitve.

Spletna storitev je razvita v programski tehniki imenovani vstavljanje

odvisnosti. Vstavljanje odvisnosti je tehnika za doseganje šibkega sklapljanja med objekti in njihovimi odvisnostmi. Šibko sklapljanje v sistemu pomeni, da se komponente v sistemu malo oziroma nič ne zavedajo ostalih komponent sistema. Vstavljanje odvisnosti sledi principu obratne odvisnosti. To pomeni, da moduli, ki so v hierarhiji višje, ne bi temeljili na modulih, ki so nižje v hierarhiji. Oboji naj bi temeljili na abstrakcijah.

Prednost vstavljanja odvisnosti je, da se namesto novih instanc objektov, ali statičnih reference uporablja drug način. Objekt se v prvotni objekt, v katerega vstavljamo odvisnost, vstavi preko konstruktorja, spremenljivke, ali preko metode in s tem se odgovornost za iskanje reference na ta objekt preseli izven tega objekta. Komponente v takem sistemu so zamenljive in so manj omejene na parametre pri razvoju, kot so platforma, programski jezik in operacijski sistem.

Spletna storitev je podrobno opisana s pomočjo orodja za dokumentiranje spletnih storitev *Swagger* (slika E.1 prikazuje avtomatsko pripravljeno spletna stran orodja *Swagger* za dokumentacijo spletne storitve *Web API*). V dokumentaciji so predstavljene vse dostopne točke, tip posamezne zahteve (GET, POST, PUT, DELETE), metapodatki za zbirke, vhodni parametri in pričakovani odgovori spletne storitve.

Poglavje 4

Ovrednotenje rešitve

Novo arhitekturo smo ovrednotili s pregledom izpolnjevanja zahtev iz uvodnega poglavja in s primerjavo med staro in novo arhitekturo. Za vsako zahtevo je opisano njeno izpolnjevanje v stari (*ležeč zapis*) in novi arhitekturi.

4.1 Funkcionalne zahteve

- **Podpora za več družb hkrati**

  Podatkovna baza za shranjevanje metapodatkov dokumentov se nahaja lokalno v eni izmed družb in ni dostopna ostalim družbam. Metapodatke lahko zapisujejo le uporabniki, ki so v domeni te družbe. Arhitektura ne vsebuje komponente, ki bi omogočala hranjenje uporabnikov in uporabniških skupin na enem mestu.

  V novi arhitekturi so najemniki družbe v skupini, skupno programsko instanco pa predstavlja spletna storitev v tehnologiji REST za upravljanje z dokumenti v SD. Dostop do dokumentov je mogoč le preko spletne storitve *Web API*, ki do arhiva dostopa preko spletnih storitev SOAP. Uporabniki in uporabniške skupine iz strežnikov LDAP posameznih družb so v KC, kar poenoti njihovo upravljanje.

Ocena: Nova arhitektura odpravi pomanjkljivost stare in izpolnjuje zahtevo. S to arhitekturo lahko brez težav nudimo podporo več družbam.

- **Spletna storitev**

  Arhitektura ne vsebuje spletne storitve za dostop do metapodatkov dokumenta. Na voljo je spletna storitev SOAP za vlaganje in prevzem dokumenta iz arhiva. Stara arhitektura podpira samo osnovno shranjevanje metapodatkov o dokumentu v podatkovno bazo. Arhitektura tudi ne podpira funkcionalnosti navideznih map za uskupinjanje dokumentov.

  Arhitektura vsebuje spletno storitev *Web API*. Preko nje je mogoče izvajati operacije CRUD nad zbirkami v SD. Poleg metapodatkov o dokumentu v novi arhitekturi hranimo podatke o zbirkah navideznih map, klasifikacijskih načrtov dokumentov in podjetij. Metapodatki se shranjujejo v podatkovno bazo MSSQL in iskalni indeks ES. Navidezne mape v SD so implementirane s pomočjo zapisov metapodatkov v podatkovno bazo MSSQL. Povezave med dokumenti in mapami so predstavljene s pomočjo relacij v podatkovni bazi.

  Ocena: Podobnost med arhitekturama opazimo v načinu shranjevanja metapodatkov. Obe rešitvi za shranjevanje metapodatkov uporabljata relacijsko podatkovno bazo. Obe arhitekturi vsebino dokumenta vlagata v arhiv EEX. To počneta na enak način, z uporabo spletne storitve SOAP za vlaganje v arhiv dokumentov. Izboljšave v novi arhitekturi so v naprednem beleženju metapodatkov in v spletni storitvi REST, ki omogoča njihovo upravljanje.

- **Avtorizacija**

  Arhitektura ne vsebuje sistema avtorizacije, razen v primeru dostopa do podatkov v podatkovni bazi.

  V SD se dostop do posameznih objektov vodi s pomočjo vlog, dodeljenih v KC, in z vlogami, zapisanimi v klasifikacijskem načrtu ter na

posameznih metapodatkih v zbirkah. Za vsako relacijo med objekti se s pomočjo nadzornih razredov preveri, ali ima uporabnik dodeljene vloge, in na osnovi te preverbe odobri ali zavrne dostop.

Ocena: Arhitektura ustreza zahtevi. Kombinacija vlog v KC in na metapodatkih omogoča zelo podrobno avtorizacijo na ravni posameznega dokumenta. Vloge za dostop so zelo prilagodljive in jih lahko določamo za uporabnika, skupino ali na ravni družbe. Z zapisom vlog uporabnikov pri vlaganju dokumenta v SD in pravilno pripravo poizvedbe na ES omogočimo nadzorovano avtorizacijo za dostop do dokumenta.

V nadaljevanju bi bilo mogoče novo avtorizacijo izboljšati za natančnejšimi vlogami za dostop v metapodatkih dokumenta. Poleg podatka o vlogi, bi zapisali dodaten parameter za branje ali posodabljanje dokumenta za posamezno vlogo. Za ta način bi morali preveriti, ali poizvedovalni jezik ES podpira tako natančne filtre, s katerimi bi lahko pravilno filtrirali poizvedbe.

- **Večkriterijsko iskanje po dokumentih**

  Trentno na nekaterih internih aplikacijah v posameznih družbah za polno iskanje po besedilu uporabljamo iskalnik, ki je že vgrajen v obstoječo podatkovno bazo; v našem primeru je MSSQL. Zaradi slabih izkušenj z vgrajenimi isklanikom v podatkovni bazi MSSQL se v stari arhitekturi ne uporablja iskalnik, vgrajen v Oraclovo podatkovno bazo. Pretekle izkušnje z uporabo vgrajenega iskalnika so bile, kljub relativno majhni velikosti tabel, precej slabe. Hitrost poizvedb in splošno delovanje sta bila počasna. Poleg tega je prišlo tudi do težav pri uporabi iskanja zaradi hroščev oziroma slabe implementacije podpore v MSSQL.

  SD preko dostopne točke v spletni storitvi Web API omogoča funkcionalnost večkriterijskega iskanja po dokumentih s pomočjo poizvedovalnega jezika ES. S tem podpira vse želene načine iskanja, naštete v zahtevah.

  Ocena: Nova arhitektura, v nasprotju s staro, doda možnost večkriterijskega

iskanja po dokumentih in izpolnjuje zahtevo. Zelo dobro podpira napredne iskalne poizvedbe, saj uporablja poizvedovalni jezik ES.

- **Različice dokumentov in beleženje sprememb**

  V stari arhitekturi je nov dokument nov zapis v tabelo v podatkovni bazi. Iz zapisov ni mogoče pridobiti različic dokumentov. Metapodatki se beležijo v podatkovno bazo Oracle, vendar ta postopek ni obvezen, kar pomeni, da arhitektura slabo podpira beleženje sprememb.

  Spletna storitev v novi arhitekturi podpira ustvarjanje različic dokumentov. Pri spremembah na dokumentih se metapodatki o spremembi in stanje objekta zapišejo v podatkovno bazo. Spletna storitev *Web API* ima na voljo dostopno točko, s katero omogočimo izpis poljubne različice dokumenta.

  Vpogledi se v dnevniško datoteko zapisujejo s pomočjo knjižnice *NLog*. Spremljamo lahko, kdo je dostopal do spletne storitve, katero dostopno točko je uporabil, kakšen je izvor izvedbe, kakšna poizvedba na podatkovno bazo se je tvorila, kakšen je rezultat poizvedbe in podobno. Vsako posodabljanje dokumenta v bazi tvori nov zapis v tabelo s spremembami na dokumentu, kamor se zapiše seznam sprememb med staro in novo različico metapodatkov s pomočjo zapisa popravek JSON.

  Ocena: Arhitektura odpravi pomanjkljivost stare in zadovoljuje zahtevo. Dodatno spremljanje sprememb bi lahko omogočili tako, da bi dnevniške zapise poslali na indeks v ES. Na tak način bi lahko s pomočjo orodja Kibana, ki je del sklada Elastic, pripravili pametne preglede in nadzirali delovanje spletne storitve. Pri tem bi morali podrobneje definirati dnevniški zapis in pripraviti ustrezen indeks na ES.

4.2 Nefunkcionalne zahteve

- **Varnost**

  Za dosego čim boljše varnosti v sistemu smo občutljive vire zaščitili z večplastno varnostno arhitekturo. Ta vsebuje potrjevanje vhodnih podatkov ter avtorizacijo in avtentikacijo. Arhitektura ščiti pred pogostimi varnostnimi grožnjami: napad z vrinjenjem stavka SQL se preprečuje z uporabo predpripravljenih poizvedb LINQ in ogrodja *Entity Framework*. S tem urejanje stavka SQL in napad z vrinjenjem nista mogoča. Napad z večdomenskim izvajanjem kode se preprečuje s potrjevanjem vhodnih podatkov, napad CSRF in napad s ponavljanjem pa se preprečujeta z uporabo naključnega varnostnega žetona CSRF.

  Ocena: Z novo arhitekturo smo storitev zaščitili pred najpogostejšimi napadi in zagotovili varno komunikacijo med komponentami.

- **Razširljivost**

  Razširljivost v stari arhitekturi je zaradi preprostosti arhitekture precej dobra. Kot primer, komponente stare arhitekture so ohranjene v novi arhitekturi, zamenjane so le nekatere tehnologije in orodja.

  Lastnost razširljivosti nove arhitekture je upoštevana na več komponentah. V KC lahko brez težav dodajamo nove družbe s federacijo LDAP. Za nov primer uporabe v spletni storitvi dodamo novo dostopno točko. Spletna storitev je razširljiva zaradi arhitekture mikro storitev in uporabe EF za upravljanje z metapodatki v storitvi in podatkovni bazi. Arhitektura mikro storitev nam omogoča hitro dodajanje obstoječih ali novih mikro storitev, EF pa nam dovoljuje hitro dodajanje novih lastnosti za zbirke za preslikavo v podatkovno bazo.

  Zaledje spletne storitve v *ASP.NET Core* je zasnovano tako, da poleg formata JSON podpira tudi XML. To pomeni lažjo integracijo s starejšimi, že obstoječimi spletnimi storitvami in možnost podpore za aplikacije, ki za prenos podatkov uporabljajo ta format.

Ker spletna storitev uporablja programski model MVC, je mogoča kasnejša dograditev spletne storitve s spletno aplikacijo saj *ASP.NET Core* podpira izgradnjo spletnih strani. Spletna storitev v *ASP.NET Core* je na voljo za več računalniških okolij. Lahko jo uporabimo na sistemih *Windows*, *MacOS* in *Linux*. To prinaša manjše stroške, v primeru, da želimo poganjati spletno storitev v kakšnem drugem okolju, kot *Windows*.

Ocena: Nova arhitektura zadovoljuje zahtevo po dobri razširljivosti.

- **Vzdržljivost**

  Stara arhitektura je srednje dobro vzdržljiva, saj, razen arhiva EEX, ne vsebuje testov modulov in integracijskih testov ter pripravljene podrobne dokumentacije. Spremenljivosti kode ne morem oceniti, saj nimamo dostopa do izvorne kode spletnih storitev SOAP.

  Vse komponente, razen spletnih storitev SOAP, so preverjene s testi in imajo dobro dokumentirano izvorno kodo. Ta je berljiva (glej izseke kode iz dodatka) in sledi standardom. Spletna storitev *Web API* izpolnjuje zahtevo po dokumentaciji s pomočjo orodja *Swagger*. Storitev je implementirana s pomočjo tehnike vstavljanja odvisnosti.

  Ocena: Nova arhitektura zelo dobro izpolnjuje to zahtevo. Za spletno storitev *Web API* se bodo sproti dodajali novi testi, ki bodo tako še povečali vzdržljivost.

- **Odzivnost**

  Stara arhitektura ni imela težav z odzivnostjo. Funkciji zapisovanja v podatkovno bazo in spletna storitev za vlaganje dokumenta v arhiv pri interni rabi nista imeli težav z odzivnostjo.

  Odzivnost nove arhitekture povečujemo z uporabo predpomnjenja v spletni storitvi in uporabo ES. Prednost nove arhitekture je, da kot glavni format za prenos sporočil uporablja JSON. Ta format ima, v

primerjavi s formatom XML, precej manj režije. To pomeni, da ob enakem prometu lahko pričakujemo manjšo obremenjenost mreže.

Dolgotrajnejše operacije v spletni storitvi *Web API* so pripravljene tako, da se izvedejo **asinhrono**. Med tem, ko proces čaka, da se V/I operacija zaključi, se nit sprosti za procesiranje ostalih zahtev. Tako ima spletni strežnik, ob morebitni povečani obremenitvi, na voljo več niti in večji izkoristek procesorskega časa. Asinhroni klici so nastavljeni tudi pri izvajanju zahtev HTTP na ostale komponente arhitekture.

V razvojnem okolju smo pripravili dva kratka testa, s katerima smo merili odzivni čas. V obe arhitekturi smo vložili enak dokument. Prvi test je petkrat zaporedoma zahteval njegove metapodatke. Družba SavaRe do baze z metapodatki v stari arhitekturi nima dostopa, zato smo test opravili le na novi. Drugi test je petkrat zaporedoma zahteval celoten dokument iz arhiva. Pri drugem testu smo uspeli pridobiti odzivne čase tudi za staro arhitekturo. Rezultati so prikazani v tabelah 4.1 (metapodatki) in 4.2 (celoten dokument), odzivni čas pa je podan v milisekundah.

Zaporedje zahteve	Nova arhitektura (čas v ms)	Stara arhitektura (čas v ms)
1	118,00	/
2	100,00	/
3	111,00	/
4	76,00	/
5	116,00	/
Povprečje	104,20	/

Tabela 4.1: Odzivni časi poizvedb po metapodatkih dokumenta v stari in novi arhitekturi.

Zaporedje zahteve	Nova arhitektura (čas v ms)	Stara arhitektura (čas v ms)
1	3120,00	1440,00
2	2780,00	1530,00
3	2850,00	1220,00
4	2880,00	1230,00
5	2710,00	1350,00
Povprečje	2870,00	1350,00

Tabela 4.2: Odzivni časi poizvedb po celotnem dokumentu v stari in novi arhitekturi.

Ocena: Arhitektura izpolnjuje zahtevo po dobri odzivnosti pri prevzemu metapodatkov iz nove arhitektura, saj je povprečni odzivni čas pod 200 ms. Zahteva, pri kateri se nova arhitektura obnese slabše, kot stara, je zahteva po celotnem dokumentu. Daljši odzivni čas lahko pripišemo dodatni komponenti (spletna storitev Web API) med odjemalcem in arhivom. Podrobne teste bomo izvedli v testnem in predprodukcijskem okolju.

- **Povečljivost**

  V stari arhitekturi ni bilo večjega poudarka na povečljivosti, saj je mišljena za interno rabo.

  Povečljivost je zagotovljena z uporabo predpomnjenja. Poleg predpomnjenja, uporabljenega na enem strežniku, ogrodje *.NET Core* podpira tudi možnost razpršenega predpomnjenja. V tem načinu je predpomnilnik skupen vsem strežniškim aplikacijam. S tem se ohrani integriteta podatkov, poveča se njihova razpoložljivost in omogoči neodvisno dodajanje in odstranjevanje strežnikov.

  K povečjljivosti pripomore tudi uporaba iskalnika ES. Iskalnik omogoča poizvedbe v snopu. To pomeni, da se z enim klicem na spletno storitev

ES izvede več operacij naenkrat, na primer več indeksiranj dokumentov v indeks. S tem zmanjšamo število klicev na spletno storitev ES in zmanjšamo zasedenost pasovne širine. Strežnik ES lahko izvajamo na več gručah, kar omogoča dobro povečljivost.

Ocena: Arhitektura je pripravljena, da podpira vzporedno povečljivost in izpolnjuje zahtevo. Arhitektura podpira vzporedno povečljivost za komponente ES, KC in spletno storitev *Web API*, vendar v tem trenutku še nimamo pripravljene fizične postavitve in ne moremo natančno oceniti prispevka dodatne strojne opreme.

Poglavje 5

Zaključek

V okviru diplomskega dela sta bili izdelani nova arhitektura in razvojna različica spletne storitve *Web API*. Podporne komponente so pooblastitveno-overitveni strežnik KC, podatkovni strežnik MSSQL, arhiv dokumentov EEX. in iskalnik ES. Arhitektura uspešno zadovoljuje funkcionalne in nefunkcionalne zahteve. Trenutno je spletna storitev v fazi razvoja, ostale komponente pa so večinoma pripravljene za povezovanje, ko bo spletna storitev končana. V diplomskem delu smo dobro raziskali trenutno stanje arhitekture in pregledali, katere zahteve še niso izpolnjene. Arhitektura, ki smo jo načrtali, je uporabna v podobnih sistemih. Z dobrim načrtom arhitekture programske opreme smo pridobili pomembne prednosti. Ob podobnem projektu bomo ujemajoče se komponente ponovno uporabili in s tem zmanjšali stroške izdelave.

Arhitekturo bi kasneje lahko izboljšali z uporabo sistema za veriženje sporočil (*ang. message queue*). Primer takega sistema je *RabbitMQ* [15]. *RabbitMQ* je posrednik sporočil (*ang. message broker*), ki implementira protokol AMQP. Protokol AMQP omogoča usklajenost med skladnimi odjemalci in posrednikiškimi strežniki za veriženje sporočil. Protokol vsebuje tri glavne komponente, to so komponenta za izmenjavo (*ang. exchange*) sporočil, vrsta sporočil (*ang. message queue*) in komponenta za vezavo (*ang. bind*). Komponenta za izmenjavo prejme sporočilo od aplikacij za objavljanje sporočil

(*ang. publisher*) in ga postavi v vrsto. Sporočila so shranjena v vrsti, dokler jih ne procesirajo naročniške aplikacije (*ang. consumer*). Z vezavo definiramo relacijo med sporočilno vrsto in izmenjavo. Vezava definira parametre, na osnovi katerih izmenjava pridobi informacijo za usmerjanje sporočil v ustrezno vrsto. Razlogi za uvedbo takšnih sistemov so naslednji.

Ker se spletne storitve lahko ustavijo, želimo, da v takih primerih ne vplivajo na delovanje celotnega sistema. V takih primerih, je smiselno uporabiti enoten sporočilni sistem. Storitve bi se tako lahko posvetile svoji dejanski vlogi, pri spletni storitvi v SD je to, na primer, shranjevanje metapodatkov o zbirkah, za prenašanje sporočil o uspešnosti transakcij pa bi skrbel sporočilni sistem. Ta rešitev bi izboljšala razširljivost in vzdržljivost arhitekture, saj bi lahko dodajali mikro–storitve, neodvisno od že obstoječih storitev, kar bi zmanjšalo tveganje za nastanek novih napak.

Če arhitektura tega ne podpira, se podatki ob neuspešno izvedenem procesu izgubijo. Sistem omogoča ohranjanje sporočil, dokler niso v celoti obdelana.

Prednost veriženja sporočil je tudi asinhrono procesiranje. Namesto da odjemalec čaka na odgovor strežnika, lahko odloži sporočilo v vrsto in nadaljuje z delom.

Tehnologija veriženja sporočil služi tudi kot medpomnilnik za sporočila. To pomeni, da lahko operacije, kot je polnjenje baze, združimo v eno zahtevo in s tem zmanjšamo promet na mreži.

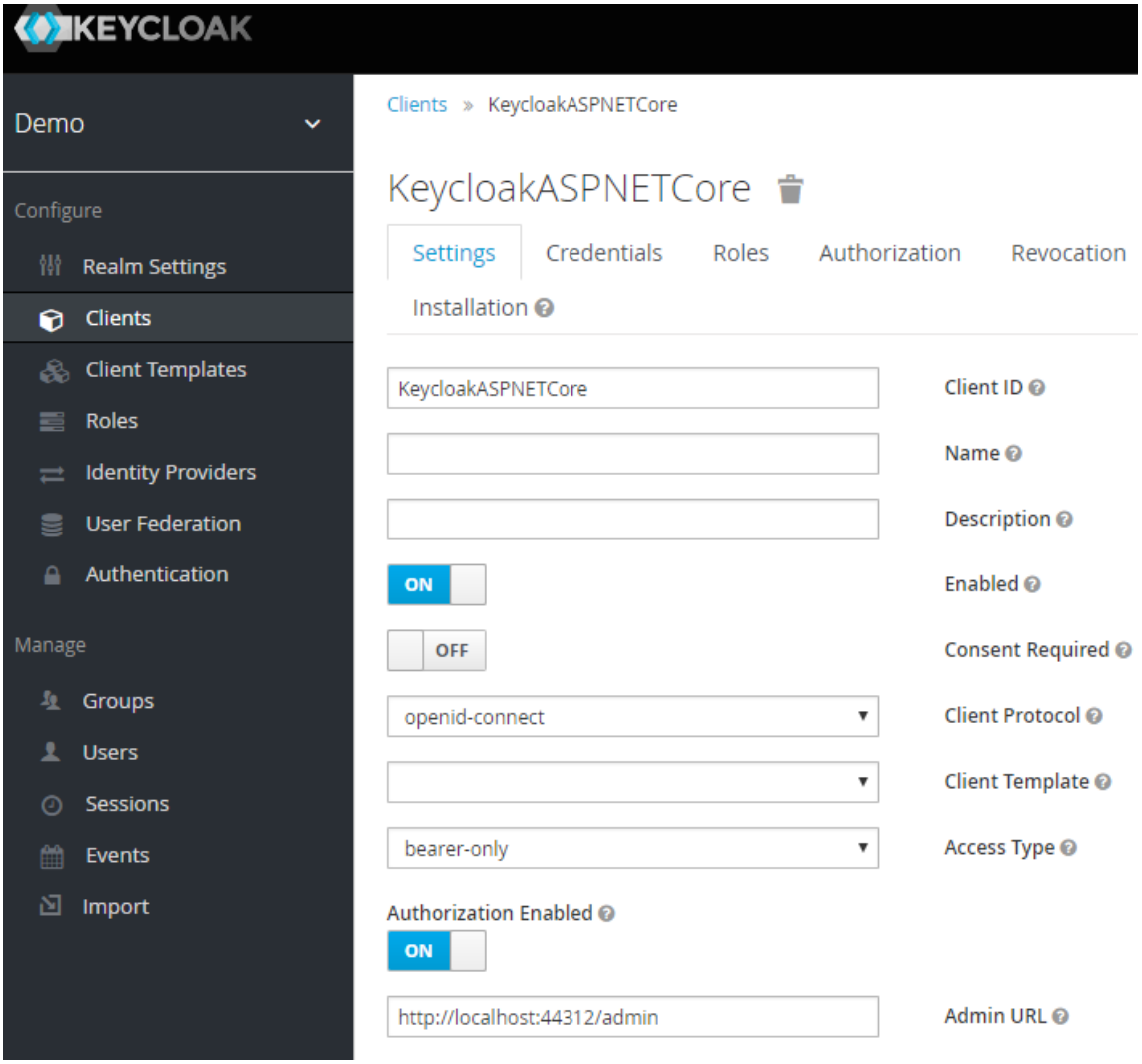
Trenutno tak sistem zaradi neobstoja enotnega sporočilnega sistema, ki bi ga uporabljale vse storitve v družbi, še ni mogoč. Podrobnosti vključitve sporočilnega sistema v novo arhitekturo so del interne strategije družbe SavaRe.

Za skupino SavaRe arhitektura SD pomeni pomemben korak k uvedbi brezpapirnega poslovanja. Pripomogla bo k hitrejšemu izvajanju poslovnih procesov in uvedla sodoben način poslovanja z dokumenti.

Dodatek A

Nastavitev avtorizacije v spletni storitvi

V konzoli KC smo spletno storitev dodali *Web API* med odjemalce. Na sliki A.1 je prikazan uporabniški vmesnik nadzorne konzole v KC. V zavihku z nastavitvami odjemalca smo nastavili dostopni način (*ang. access type*), protokol na odjemalcu (*ang. Client Protocol*) in vklopili avtorizacijo. V zavihku z vlogami (*ang. Roles*) smo nastavili vloge odjemalcev. V zavihku za nastavitev pooblastitve (slika A.1) smo določili podrobnosti glede avtorizacije. V skladu s procesom avtorizacije v KC smo v zavihku viri (*ang. Resources*) definirali dostopne točke spletne storitve, dovoljene tipe virov in dovoljene akcije. Vsako dovoljenje smo asociirali z želenim področjem in politiko dostopa.



Slika A.1: Nastavitev spletne storitve v nadzorni konzoli KC.

Odjemalsko aplikacijo smo registrirali kot odjemalca v KC. V nastavitvah odjemalca smo dostopni način nastavili na zaupno (*ang. confedential*). Tu smo vklopili nastavitev za vklop storitvenega računa (*ang. service account*) za odjemalsko aplikacijo in ji dodelili vlogo za dostop do spletne storitve.

V naslednjem koraku smo nastavili avtorizacijo v spletni storitvi *ASP.NET Core* A.1.

```
1
2           app.UseJwtBearerAuthentication(new
                JwtBearerOptions
3           {
4               AuthenticationScheme = "bearer",
5               AutomaticAuthenticate = true,
6               AutomaticChallenge = true,
7               Authority = Configuration["Authentication:
8               KeycloakAuthentication:
9               ServerAddress"] + "/auth/realms/" +
                    Configuration["Authentication:
10              KeycloakAuthentication:Realm"],
11              TokenValidationParameters = new
                    Microsoft.IdentityModel.
12              Tokens.TokenValidationParameters
13              {
14                  ValidAudiences = new string[] { "curl",
                        "financeApplication",
                        "accountingApplication", "swagger",
                        "Odjemalec_B", "Odjemalec_A" }
15              },
16              RequireHttpsMetadata = false,
17              SaveToken = true
18
19          });
```

Listing A.1: Nastavitev avtentikacije s pomočjo nosilnega žetona v spletni storitvi *Web API*.

V razredu *Startup.cs* smo za overitev z nosilnim žetonom uporabili vmesno opremo *JwtBearerAuthentcation*, ki je del ogrodja *.NET Core*. V konfiguraciji smo nastavili naslov pooblastivenega strežnika in registrirali veljavne odjemalske aplikacije s parameterom *ValidAudiences*. Za odjemalske aplika-

cije smo uporabili ime, enako tistemu, ki je registrirano za ime odjemalske aplikacije v KC. Dodatno smo nastavili parameter *SaveToken* na *vrednost True*, ki shrani žeton za dostop.

Za vsako dostopno točko smo nastavili pooblastitvene zahteve (*ang. authorization requirement*), ki morajo biti izpolnjene za uspešno avtorizacijo. Pooblastitvene zahteve smo dodali kot značke nad metodami v kontrolnih razredih. Ko odjemalska aplikacija dostopa do dostopne točke, se izvede preverba v pooblastitvenem nadzornem razredu (*ang. authorization handler*) z ustrezno avtorizacijsko zahtevo. Nadzorni razred nato uporabi dostopni žeton odjemalske aplikacije, da preko KC dostopne točke, imenovane *Entitelment API*, pridobi seznam vlog za trenutno odjemalsko aplikacijo. Spletna storitev kot parametre v glavi zahteve HTTP poda dostopni žeton odjemalske aplikacije in ID spletne storitve *Web API*. Kot odgovor dobimo žeton za zahtevo zabave (*ang. requesting party token*). Vsebino žetona dekodiramo s pomočjo skrivnosti spletne storitve. Dovoljenja se nato dodajo v slovar *HttpContext.Items*, kjer ostanejo med življenjskim obdobjem zahteve. S pomočjo teh dovoljenj in z uporabo pooblastitvene storitve (*ang. authorization service*), ki jo dodamo v kontrolne razrede s pomočjo DI, lahko izvedemo pooblastitev na ravni objekta. Vsaka zbirka v spletni storitvi ima v metapodatkih obvezen atribut z dodeljenimi vlogami za dostop, s pomočjo katerih odobrimo ali zavrnemo dostop do objekta.

Dodatek B

Shema WSDL v spletni storitvi stare arhitekture

```
1  <definitions name="Easy" targetNamespace="urn:Easy"
       xmlns:tns="urn:Easy"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
       xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
       xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
       xmlns="http://schemas.xmlsoap.org/wsdl/">
2  <types>
3  <xsd:schema targetNamespace="urn:Hello"
       xmlns="http://www.w3.org/2001/XMLSchema">
4  <xsd:element name="fileName" type="xsd:string"/>
5  <xsd:element name="fileContent" type="xsd:string"/>
6  <xsd:element name="fileSize" type="xsd:int"/>
7  <xsd:element name="easyDocRef" type="xsd:string"/>
8  <xsd:element name="message" type="xsd:string"/>
9  <xsd:element name="result" type="xsd:boolean"/>
10 </xsd:schema>
11 </types>
12 <!-- putSava stuff -->
```

```
13 <message name="putSava">
14 <part name="fileName" type="xsd:string"/>
15 <part name="fileContentBase64Encoded" type="xsd:string"/>
16 <part name="fileType" type="xsd:string"/>
17 <part name="fileArea" type="xsd:string"/>
18 <part name="fileOwner" type="xsd:string"/>
19 <part name="fileClassification" type="xsd:string"/>
20 </message>
21 <message name="putSavaResponse">
22 <part name="result" type="xsd:boolean"/>
23 <part name="message" type="xsd:string"/>
24 <part name="easyDocref" type="xsd:string"/>
25 </message>
26 <!-- get stuff -->
27 <message name="getSava">
28 <part name="easyDocRef" type="xsd:string"/>
29 </message>
30 <message name="getSavaResponse">
31 <part name="result" type="xsd:boolean"/>
32 <part name="message" type="xsd:string"/>
33 <part name="fileContentBase64Encoded" type="xsd:string"/>
34 <part name="fileName" type="xsd:string"/>
35 <part name="fileType" type="xsd:string"/>
36 <part name="fileArea" type="xsd:string"/>
37 <part name="fileOwner" type="xsd:string"/>
38 <part name="fileClassification" type="xsd:string"/>
39 </message>
40 <portType name="EasyPort">
41 <operation name="putSava">
42 <input message="tns:putSava"/>
43 <output message="tns:putSavaResponse"/>
44 </operation>
45 <operation name="getSava">
```

```
46 <input message="tns:getSava"/>
47 <output message="tns:getSavaResponse"/>
48 </operation>
49 </portType>
50 <binding name="EasyBinding" type="tns:EasyPort">
51 <soap:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http"/>
52 <operation name="putSava">
53 <soap:operation soapAction="urn:putSavaAction"/>
54 <input>
55 <soap:body use="encoded" namespace="urn:Easy"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
56 </input>
57 <output>
58 <soap:body use="encoded" namespace="urn:Easy"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
59 </output>
60 </operation>
61 <operation name="getSava">
62 <soap:operation soapAction="urn:getSavaAction"/>
63 <input>
64 <soap:body use="encoded" namespace="urn:Easy"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
65 </input>
66 <output>
67 <soap:body use="encoded" namespace="urn:Easy"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
68 </output>
69 </operation>
70 </binding>
71 <service name="EasyService">
72 <port name="EasyPort" binding="tns:EasyBinding">
73 </port>
```

```
74 </service>
75 </definitions>
```

Listing B.1: Shema WSDL v spletni storitvi stare arhitekture.

Dodatek C

Razredi za zbirke v spletni storitvi

```
1    public class ClassificationPlan : IEntityBase
2    {
3        public ClassificationPlan()
4        {
5            Documents = new List<Document>();
6            Folders = new List<Folder>();
7        }
8
9        public int Id { get; set; }
10       public string ClassificationLabel { get; set; }
11       public int RetentionPeriod { get; set; }
12       public int DeletionPeriod { get; set; }
13       public string MainGroup { get; set; }
14       public string UserId { get; set; }
15       public Company Company { get; set; }
16       public ICollection<Document> Documents { get; set; }
17       public ICollection<Folder> Folders { get; set; }
18       public string AccessControlList { get; set; }
19
```

```
20     }
```

Listing C.1: Razred *C#* za zbirko klasifikacijski načrt.

```
1     public class Company : IEntityBase
2     {
3
4         public Company()
5         {
6             ClassificationPlans = new
                  List<ClassificationPlan>();
7             Documents = new List<Document>();
8         }
9
10        public int Id { get ; set; }
11        public string Name { get; set; }
12        public string PoolURI { get; set; }
13        public string Scope { get; set; }
14        public ICollection<ClassificationPlan>
              ClassificationPlans { get; set; }
15        public ICollection<Document> Documents { get; set; }
16
17    }
```

Listing C.2: Razred *C#* za zbirko družba.

```
1  public class Document : IEntityBase
2     {
3         public Document()
4         {
5
6             LinkedDocuments = new List<Document>();
7             IncludedInFolders = new List<DocumentFolder>();
8             DocumentDiffs = new List<DocumentDiff>();
9         }
```

```
10
11      public int Id { get; set; }
12      public string Name { get; set; }
13      public string Barcode { get; set; }
14      public string Type { get; set; }
15      public string Area { get; set; }
16      public string Metadata { get; set; }
17      public string ArchiveID { get; set; }
18      public double Version { get; set; }
19      public string SignedStatus { get; set; }
20      public DateTime DateOfCreation { get; set; }
21      public DateTime DateOfChange { get; set; }
22      public string UserId { get; set; }
23      public string Application { get; set; }
24      public string AccessControlList { get; set; }
25      public Document ParentDocument { get; set; }
26      public ICollection<Document> LinkedDocuments { get;
            set; }
27      public ICollection<DocumentFolder> IncludedInFolders
            { get; set; }
28      public ClassificationPlan ClassificationPlan { get;
            set; }
29      public Company Company { get; set; }
30      public ICollection<DocumentDiff> DocumentDiffs {
            get; set; }
31  }
```

Listing C.3: Razred *C#* za zbirko dokument.

```
1   public class Folder : IEntityBase
2   {
3
4       public Folder()
5       {
```

```
6           IncludedDocuments = new List<DocumentFolder>();
7           ChildFolders = new List<Folder>();
8       }
9
10      public int Id { get; set; }
11      public string Name { get; set; }
12      public DateTime DateOfCreation { get; set; }
13      public DateTime DateOfChange { get; set; }
14      public string Metadata { get; set; }
15      public string UserId { get; set; }
16      public string AccessControlList { get; set; }
17      public Folder ParentFolder { get; set; }
18      public ICollection<Folder> ChildFolders { get; set; }
19      public ICollection<DocumentFolder> IncludedDocuments
            { get; set; }
20      public ClassificationPlan ClassificationPlan { get;
            set; }
21  }
```

Listing C.4: Razred *C#* za zbirko mapa.

Dodatek D

Izseki kode iz spletne storitve *Web API*

```
1  public async Task<IUpdateResponse<DocumentViewModel>>
       UpdateDocument(DocumentViewModel document)
2      {
3          var asyncUpdateResponse = await
               _client.UpdateAsync<DocumentViewModel,
               DocumentViewModel>(
4              document.Id,
5              descriptor => descriptor.Doc(document)
6              );
7          return asyncUpdateResponse;
8      }
```

Listing D.1: Metoda za vlaganje in posodabljnje dokumenta v indeks ES.

```
1    public class DocumentPostPutIModelValidator:
2    AbstractValidator<DocumentPostPutModel>
3    {
4        public DocumentPostPutIModelValidator()
5        {
6            RuleFor(document => document.Base64)
```

```
7               .NotEmpty()
8               .WithMessage("'Base64' cannot be empty");
9           RuleFor(document => document.Name)
10              .NotEmpty()
11              .WithMessage("'Name' cannot be empty");
12          RuleFor(document => document.Barcode)
13              .NotEmpty()
14              .WithMessage("'Barcode' cannot be empty");
15          RuleFor(document => document.DocumentType)
16              .NotEmpty()
17              .WithMessage("'DocumentType' cannot be empty");
18          RuleFor(document => document.Version)
19              .NotEmpty()
20              .WithMessage("'Version' cannot be empty");
21          RuleFor(document => document.ClassificationLabel)
22              .NotEmpty()
23              .WithMessage("'ClassificationLabel' cannot be
                    empty");
24          RuleFor(document => document.AccessControlList)
25              .NotEmpty()
26              .WithMessage("'AccesControlList' cannot be
                    empty");
27          RuleFor(document => document.AccessControlList)
28              .Must(acl => acl.Length ==
                    acl.Distinct().Count())
29              .WithMessage("'AccesControlList' must have
                    unique entries");
30
31      }
32
33  }
```

Listing D.2: Razred *C#* za potrjevanje zbirke dokument.

```
1           DocumentViewModel _document;
2           bool cacheExist =
               _memoryCache.TryGetValue(id.ToString(), out
               _document);
3
4           if (!cacheExist || _document == null)
5           {
6              var elasticDocumentsResponse = await
                  _elasticsearch
7              .SearchDocumentsByIDFilteredByRolesAndID(
8                 User,
9                 id);
10             if (elasticDocumentsResponse.Hits != null &&
                  elasticDocumentsResponse.IsValid &&
                  elasticDocumentsResponse.Hits.Any())
11             {
12                _document =
                     elasticDocumentsResponse.Documents.ToList()
13                .FirstOrDefault();
14                DocumentViewModel _documentCopy =
                     _document.GetClone();
15                _documentCopy.Metadata = _document.Metadata;
16                _documentCopy.DocumentDiffs =
                     _document.DocumentDiffs;
17                _memoryCache.Set(id.ToString(),
                     _documentCopy);
18                _documentCopy.DocumentDiffs =
                     _document.DocumentDiffs;
19                _documentCopy.Metadata = _document.Metadata;
20             }
21             else
22             {
23                return NotFound();
```

```
24              }
25          }
```

Listing D.3: Primer uporabe predpomnjenja v spletni storitvi.

```
1       public async
           Task<ISearchResponse<DocumentViewModel>>
           SearchDocumentsFilteredByRoles(ClaimsPrincipal
           user)
2       {
3          var ACLterms = getAclTerms(user);
4
5          QueryContainer aclResource = new TermsQuery() {
6             Field = "accessControlListDocument.keyword",
7             Terms = ACLterms };
8          QueryContainer aclClassificationPlan = new
              TermsQuery() {
9             Field =
                 "accessControlListClassificationPlan.keyword",
10            Terms = ACLterms };
11
12         var asyncSearchResponse = await
              _client.SearchAsync<DocumentViewModel>(
13            s => s.Query(
14               q => q.Bool(
15                  b => b.Filter(
16                        f => f.Bool(
17                           boolFilter => boolFilter
18                           .Should(
19                                 aclResource,
20                                 aclClassificationPlan
21                                 ).MinimumShouldMatch(1)
22                           )
23                  )
```

```
24                 )
25             )
26           );
27           return asyncSearchResponse;
28       }
```

Listing D.4: Metoda za pripravo poizvedbe v poizvedovalnem jeziku ES za prevzem metapodatkov dokumenta spletne storitve ES.

```
1  public static Mock<IMemoryCache> GetMemoryCache(bool
       tryGetvalue, object expectedValue, ICacheEntry
       cacheEntry)
2          {
3              var mockMemoryCache = new Mock<IMemoryCache>();
4              mockMemoryCache
5                  .Setup(x => x.TryGetValue(It.IsAny<object>(),
                       out expectedValue))
6                  .Returns(tryGetvalue);
7              mockMemoryCache
8                  .Setup(repo =>
                       repo.CreateEntry(It.IsAny<object>()))
9                  .Returns(cacheEntry);
10             return mockMemoryCache;
11         }
```

Listing D.5: Metoda, v kateri pripravimo objekt, za oponašanje objekta IMemoryCache

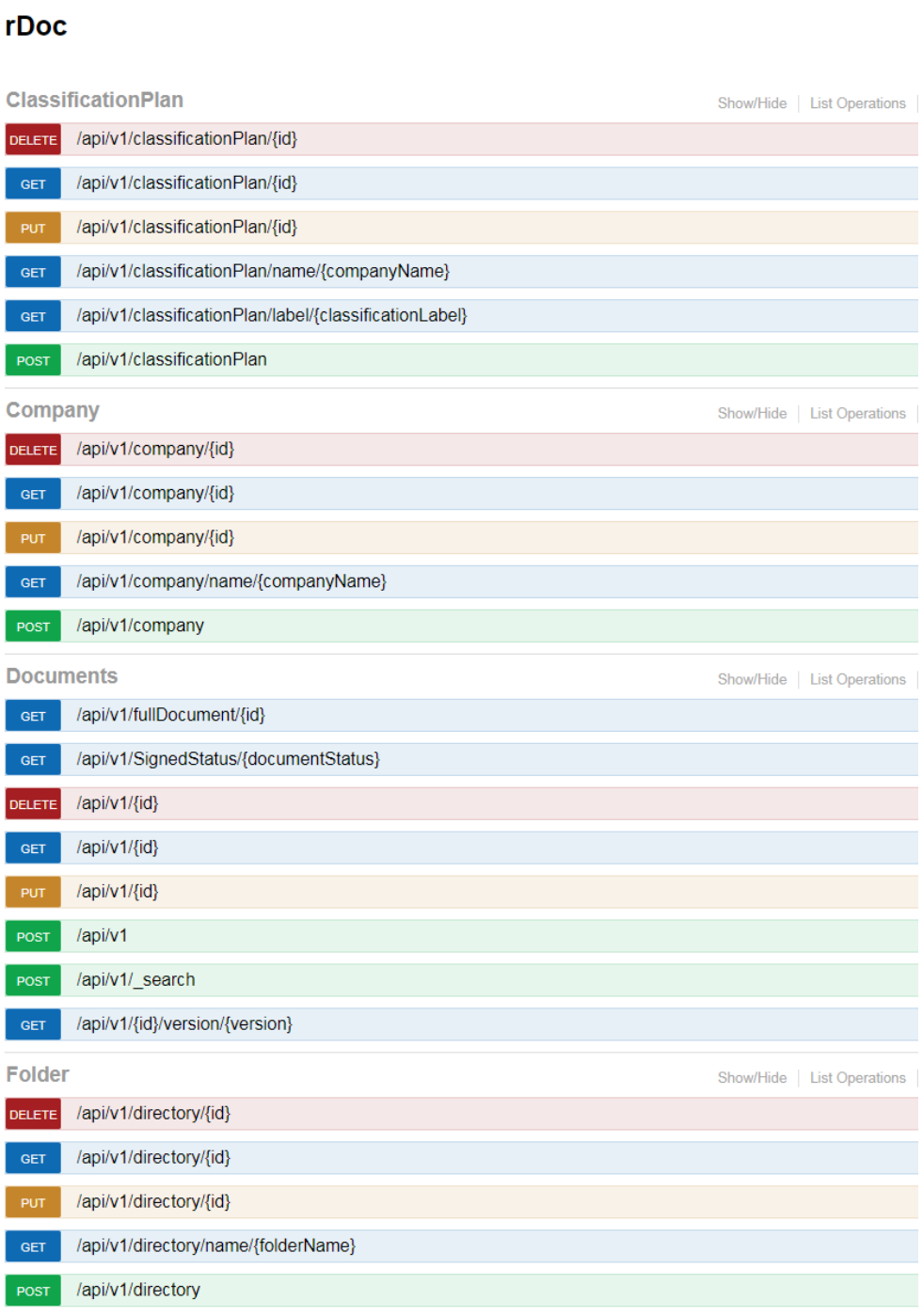
```
1   {
2       "Version": [
3             0.1,
4             0.2
5       ],
6       "DateOfChange": [
7             "2017-12-06T12:15:05.9474634",
```

```
8               "2017-12-06T12:15:21.8034634+01:00"
9         ],
10        "LinkedDocuments": {
11              "0": [
12                    4375
13              ],
14              "_t": "a"
15        },
16        "IncludedInFolders": {
17              "0": [
18                    1173
19              ],
20              "_t": "a"
21        }
22  }
```

Listing D.6: Prikaz primera zapisa spremembe v metapodatkih dokumenta v obliki popravek JSON. Na primeru vidimo, da sta se spremenili različica (D.6, 2–5) in datum spremembe (D.6, 6–9), dodala pa sta se en nov povezan dokument (D.6, 10–15) in povezava na mapo (D.6, 16–21)

Dodatek E

Dostopne točke v spletni storitvi *Web API*



Slika E.1: Seznam dostopnih točk spletne storitve, prikazane s pomočjo ogrodja *Swagger*.

Literatura

[1] Bryan, P., and Mark Nottingham. *JavaScript Object Notation (JSON) Patch.* No. RFC 6902. 2013.

[2] Easy Enterprise.X DMS Server [Online]. Dosegljivo: https://easy.de/Veranstaltung/easy-enterprise-x-dms-server/ [Dostopano 19. 11. 2017].

[3] Elasticsearch: The Definitive Guide [Online]. Dosegljivo: https://www.elastic.co/guide/en/elasticsearch/guide/current/images/elas_0402.png, Clinton Gormley Zachary Tong, pod licenco CC BY-NC-ND 3.0 [Dostopano 23. 12. 2017].

[4] Gormley, Clinton, and Zachary Tong. *Elasticsearch: The Definitive Guide: A Distributed Real-Time Search and Analytics Engine.* O'Reilly Media, Inc, 2015.

[5] Gasser, Morrie. *Building a secure computer system.* Van Nostrand Reinhold, 1988.

[6] Halfond, William G., Jeremy Viegas, and Alessandro Orso. *A classification of SQL-injection attacks and countermeasures. Proceedings of the IEEE International Symposium on Secure Software Engineering. Vol. 1.* IEEE, 2006.

[7] Hardt, Dick. *The OAuth 2.0 authorization framework.* RFC 6749, 2012.

[8] Johansson, Niklas, and Anton Löfgren. *Designing for Extensibility: An action research study of maximizing extensibility by means of design principles.* BS thesis, 2009.

[9] Kafer, Konstantin. *Cross site request forgery.* Technical Report, Hasso-Plattner-Institut, Posdam, 2008.

[10] Krebs, Rouven, Christof Momm, and Samuel Kounev. *Architectural Concerns in Multi-tenant SaaS Applications.* Closer 12, 2012.

[11] McCandless, Michael, Erik Hatcher, and Otis Gospodnetic. *Lucene in Action: Covers Apache Lucene 3.0.* Manning Publications Co., 2010.

[12] Moore, Jason F., et al. *System and method utilizing virtual folders.* U.S. Patent No. 7,925,682. 12 Apr. 2011.

[13] Nadareishvili, Irakli, et al. *Microservice Architecture: Aligning Principles, Practices, and Culture.* O'Reilly Media, Inc., 2016.

[14] Freeman, Adam. *Pro ASP. NET MVC 5 Platform.* Pro ASP. NET MVC 5 Platform. Apress, 2014.

[15] RabbitMQ [Online]. Dosegljivo: http://lucene.apache.org/core/ [Dostopano 14. 11. 2017].

[16] Sakimura, Nat, et al. *OpenID Connect Core 1.0 incorporating errata set 1.* The OpenID Foundation, specification, 2014.

[17] Škodni spis [Online]. Dosegljivo: http://www.adriatic-slovenica.si/o-druzbi/slovarcek/ss [Dostopano 26. 11. 2016].

[18] Sommervile, Ian. *Software engineering.* Pearson, 2011.

[19] Vogt, Philipp, et al. *Cross Site Scripting Prevention with Dynamic Data Tainting and Static Analysis.* NDSS. Vol. 2007. 2007.

[20] Weik, Martin. *Computer scomcience and munications dictionary.* Springer Science & Business Media, 2000.

[21] Wescott, Bob. *The Every Computer Performance Book, Chapter 3: Useful laws.* CreateSpace, 2013.