Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Žan Schaffer

Agregacija storitev upravljanja poslovnih procesov

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Mojca Ciglarič

Ljubljana, 2019

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Za razvijalce predstavlja poznavanje podrobnosti delovanja in upravljanja različnih sistemov za upravljanje poslovnih procesov (BPMS) veliko breme. Preučite, kateri BPMS obstajajo na trgu, izmed teh izberite nekaj predstavnikov in jih analizirajte. Zasnujte in razvijte agregacijsko storitev, ki bo razvijalcem omogočala uporabo katerega koli od izbranih sistemov, ne da bi pri tem morali podrobno poznati njihovo delovanje. Storitev ustrezno testirajte ter komentirajte prednosti in slabosti.

Zahvaljujem se vsem, ki so po svojih najboljših močeh prispevali k izdelavi diplomskega dela in me pri tem spodbujali. Posebno se zahvaljujem mentorici izr. prof. dr. Mojci Ciglarič, ki me je po svojih najboljših močeh usmerjala pri izdelavi diplomskega dela. Zahvaljujem se tudi vsem v podjetju SRC sistemske integracije d.o.o. Še posebej bi se zahvalil obema mentorjema, Kristjanu Pucku in Matjažu Terglavu ter vodji oddelka Marjeti Leben.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
BPMS	Business process management suite	Sistem za upravljanje poslovnih procesov
BPM	Business process management	Upravljanje poslovnih procesov
API	Application programming interface	Vmesnik za namensko programiranje
REST	Representational state transfer	Reprezentativni prenos stanja
YAML	YAML Ain't Markup Language	YAML ni označevalni jezik
OAS	OpenAPI Specification	OpenAPI Specifikacija
URI	Uniform Resource Identifier	Enolični kazalnik vira
JSON	JavaScript Object Notation	Objektna notacija JavaScripta
HTTP	HyperText Transfer Protocol	Protokol za prenos hiperteksta

Povzetek

Naslov: Agregacija storitev upravljanja poslovnih procesov

Avtor: Žan Schaffer

To delo predstavi težave v neskladnosti tako pri implementaciji kot tudi uporabi različnih sistemov za upravljanje s poslovnimi procesi. Ker neskladnost negativno vpliva na hitrost razvoja aplikacij in končno ceno izdelka, ponudim rešitev v obliki mikrostoritve, ki deluje kot vmesnik med sistemi za upravljanje s poslovnimi procesi in uporabnikom, ne da bi se mu bilo treba zavedati, kateri sistem teče v ozadju. Mikrostoritev prek vmesnikov REST komunicira s tremi izbranimi sistemi za upravljanje s poslovnimi procesi in agregira komunikacijo v enotne, enostavnejše klice, kar omogoča razvijalcem hitrejši razvoj in lažjo uporabo ter posledično zniža stroške tako za razvijalce kot tudi za stranke.

Ključne besede: BPMS, BPM, agregiranje, poslovni proces, mikrostoritev, vmesnik, Activiti, Camunda, IBM BPM.

Abstract

Title: Aggregation of BPMSs

Author: Žan Schaffer

This thesis presents issues with compatibility of implementation and usage of different Business Process Management Software. Some of those issues are longer developing cycles and an increased cost of the product. To overcome these issues, I developed a microservice, which works as a wrapper around BPMS and hides the specifics of a particular BPMS implementation and allows the user to be agnostic about the implementation. The microservice communicates through REST APIs with BPMS and aggregates their calls into unified calls, making it easier to develop and deliver a more affordable product.

Keywords: BPMS, BPM, aggregate, business process, microservice, API, Activiti, Camunda, IBM BPM.

Poglavje 1

Uvod

Diplomsko delo se spopada s težavo razlik med sistemi za upravljanje poslovnih procesov (t. i. BPMS), in tako kot planinski čevlji niso primerni za na ples, tako tudi sistem za upravljanje poslovnih procesov izberemo glede na zahteve težave.

Pri izbiri se srečamo z vprašanji, kot so: "Kakšna je velikost projekta? Kakšne potrebe ima stranka? Je treba sistem integrirati v druge aplikacije?" Na njih odgovorim v 2. poglavju, v katerem se spustim v izzive, ki jih srečamo ob delu z BPMS in postavim cilje, ki jih želim rešiti v diplomskem delu.

Da pa mi postane bolj jasno, zakaj prihaja do teh izzivov, v 3. poglavju izberem tri sisteme, jih na kratko predstavim ter primerjam prednosti in slabosti.

V 4. poglavju se odločim za rešitev - mikrostoritev, ki združi vmesnike REST posameznih BPMS v bolj splošnega, ki še vedno pokrije osnovne zahteve. V tem poglavju si ogledam tudi sestavne dele storitve ter analiziram pristope in tehnologije, ki so potrebne za izvedbo.

Ker so BPMS brez poslovnega procesa neuporabni, v 5. poglavju analiziram preprost proces za prenos osnovnega sredstva z ene osebe na drugo, ki bo služil kot test rešitve in s tem omejim delo na neko smiselno celoto.

V 6. poglavju se spopadem z osrčjem težave: združevanjem klicev vmesnikov REST BPMS. V tem poglavju identificiram potrebne podatke za izvedbo

procesa in jih združim v neko celoto, skupno vsem obravnavanim BPMS.

V 7. poglavju se osredotočim na dejansko implementacijo storitve. Predstavim izzive, na katere sem naletel, in njihove rešitve ter predlagam izboljšave, ki so primerne za nadaljnje delo. Temu sledi še 8., zaključno poglavje.

Poglavje 2

Opis problematike

Upravljanje poslovnih procesov (angl. BPM, Business Process Management) je disciplina, s katero modeliramo, avtomatiziramo, upravljamo in optimiziramo poslovne procese, kar poveča storilnost in dobiček podjetja. BPM je tudi kategorija programske opreme in tehnologija, ki se uporablja za avtomatizacijo poslovnih procesov. Pet najpomembnejših lastnosti, ki jih želimo od BPM, je [3]:

1. Spreminja papirno zasnovan poslovni proces v elektronsko voden proces, ki nadomešča papirne obrazce, mape, fascikle, dokumente in vsa neučinkovita opravila, povezana z njimi.

2. Vsebuje nadzorne funkcionalnosti, ki zagotavljajo celovitost procesov in odstranjujejo možnost človeških ali sistemskih napak.

3. Zmanjšuje odzivni čas in zmanjšuje t. i. mrtvi čas.

4. Omogoča povratne informacije o statusu procesa v realnem času.

5. Meri porabo časa in stroške, povezane s poslovnim procesom, kar omogoča optimizacijo procesov.

Sistemi za upravljanje poslovnih procesov (angl. BPMS, Business process management Suite oz. Business process management Software) je

orodje za oblikovanje, implementacijo in izboljšavo dejavnosti ali nabora dejavnosti, ki dosežejo določen cilj podjetja.

BPMS pomaga vodstvu podjetja pri neprestanem izboljševanju poslovnih procesov z uporabo orodij za odkrivanje procesov in orodij za modeliranje (angl. process discovery and modeling tools), pogonov poslovnih pravil (angl. business rules engines), pogonov za potek dela (angl. workflow engines) in orodij za simulacijo in testiranje (angl. simulation and testing tools). Tako lahko razvojni inženirji, procesni analitiki, razvijalci in preostalo IT-osebje uporabljajo BPMS za zvišanje produktivnosti, učinkovitosti in dobičkonosnosti.

V diplomskem delu se spopadam s težavo velikega števila BPMS, vsak s svojimi prednostmi in slabostmi, kar povzroča vrsto izzivov tako strankam kot tudi razvijalcem. Trenutno stanje je tako, da ima le redko katero podjetje z BPM podprte poslovne procese, čeprav na daljši rok znižajo stroške in preprečujejo napake zaposlenih [6].

Kot opažam, jih imajo samo večja podjetja, ker so dragi in potrebujejo ekipo za vzdrževanje, vendar jih potrebujejo zaradi zahtev sektorja in večjega nadzora nad poslovnimi procesi.

Težave se pojavljajo tudi v razvojnih ekipah, saj je učna krivulja za člane razvojne ekipe dokaj strma in je za osvojitev treba opraviti tečaje, zato pride do pomanjkanja kadra.

2.1 Izzivi za razvijalca in stranko

2.1.1 Strma učna krivulja

Člani razvojne ekipe morajo osvojiti osnove delovanja BPMS. Za nekatere člane to pomeni opraviti tečaj, za druge pa spoznati vsaj osnove, kot so:

- delovanje BPMS,
- načrtovanje BPMN 2.0 diagramov [7],
- upravljanje poslovnih procesov.

2.1.2 Pomanjkanje kadra

Zaradi narave BPMS imajo razvijalci le redko predhodno znanje BPM, ki je izven nabora "klasičnih znanj" razvijalcev. Tako je skoraj vedno treba izobraziti nov kader, kar zahteva dodatno financiranje.

2.1.3 Visoka cena lastništva (TCO)

Nekateri BPMS so bolj zahtevni z vidika namestitve in vzdrževanja, kar zviša ceno končnemu izdelku, po drugi strani pa so nekateri odprto-kodni in primerni za stranke, ki ne potrebujejo vseh funkcionalnosti dražjih BPMS.

2.1.4 Zaprta arhitektura

Zaprta arhitektura nekaterih BPMS povzroča težjo integracijo v druge sisteme, kar pomeni, da je izbira BPMS odvisna od izziva, ki ga željo rešiti stranke.

2.1.5 Majhen tržni delež

Posledično je na zgoraj omenjene težave trg, ki ga trenutno pokrivajo rešitve z uporabo BPMS, relativno majhen.

2.2 Želje podjetja

2.2.1 Znižanje cen razvoja

Znižanje cene, tako za stranke kot tudi za podjetje. To bi omogočalo, da bi pokrili večji trg, saj bi o uvedbi BPMS lahko začele razmišljati tudi manjše stranke, ki si ga sedaj ne morejo privoščiti.

2.2.2 Znižanje zahtevnosti uporabe

Znižanje zahtevnosti do te mere, da bi bil razvijalec zmožen sodelovati na projektu le po nekaj urah uvoda in brez znanja o specifičnem BPMS, kar bi zmanjšalo zahteve po kadru in izobraževanju.

2.2.3 Preprosta integracija z drugimi aplikacijami

Preprosta integracija z aplikacijami do take mere, da se uporabniku ne bi bilo treba zavedati, kateri BPMS je v ozadju.

2.2.4 Podprte osnovne funkcionalnosti na različnih BPMS

Podprte osnovne funkcionalnosti, tako da je mogoče izvesti preprost poslovni proces, narejen z uporabno BPMS od začetka do konca brez izkoriščanja uporabniških vmesnikov.

Poglavje 3

Predstavitev BPMS

Za namen tega diplomskega dela vam bom predstavil sisteme Activiti, Camunda in IBM BPM. Izbral sem jih zaradi zahtev podjetja, saj so eni izmed bolj priljubljenih BPMS, poleg tega odlično prikazujejo kontrast med staro genaracijo BPMS (IBM BPM) in novo generacijo (Camunda, Activiti). Prav tako pa predstavljajo dovolj velik nabor, da lahko pričakujemo, da je mogoče podpreti tudi druge brez večjih zapletov. Primeri drugih BPMS so:

- ActiveVOS,
- Bizagi BPM Suite,
- Bonita BPM,
- Flowable,
- Imixs-Workflow,
- jBPM,
- Orchestra,
- Sydle SEED.

Med seboj so si zelo podobni, vendar se v nekaterih delih razlikujejo, zato se bomo v nadaljevanju seznanili z vsakim posebej in jih na koncu primerjali.

3.1 Activiti

Activiti je odprto-kodni sistem za upravljanje poteka dela, napisan v Javi, ki podpira poslovne procese, opisane v, BPMN 2.0. Sistem je osnova za Alfrescov Alfresco Process Services (APS), ki je tudi glavni sponzor Activitija. Od različice 7.0 naprej teče tudi v oblaku, vendar je to izven obsega tega diplomskega dela, saj v času pisanja programske kode še ni bil na voljo in lahko povzroča težave v prihodnosti.

Kot že prej omenjeno, je to skupek aplikacij (angl. suite), ki so med seboj povezane in delujejo kot celota:

- Modeler je spletni grafični vmesnik za nadzor poteka dela.
- Designer je Eclipseov dodatek za razvoj poteka dela.
- Engine je osrednji izvajalec procesov.
- Explorer je spletno orodje za namestitev definicij procesov, ustvarjanje novih instanc procesov in upravljanje poteka dela.

3.2 Camunda

Camunda je tako kot Activiti tudi odprto-kodni sistem za upravljanje poteka dela, napisan v Javi, ki poganja poslovne procese, opisane v, BPMN 2.0 notaciji. Na začetku je bila poleg Alfresca ena izmed večjih podpornikov Activitija, vendar so se 18. 3. 2013 ločili od Activitija in zaradi drugačnih pogledov začeli z delom na Camunda BPM.

Sestavni deli Camunde so:

- Camunda Modeler je odprtokodno orodje za modeliranje BPMN 2.0 in specifičnih podrobnosti Camunde. Na voljo je samostojno ali pa kot dodatek k Eclipsu.
- Cockpit je spletna aplikacija za administracijo in nadzor delovnega toka

v produkciji.

- Process Engine je osrednji del Camunde, ki poganja procese BPMN.
- Model Repository je shramba, v kateri so shranjeni vsi modeli, definirani v procesih.
- Camunda Tasklist je spletna aplikacija za upravljanje poteka dela in uporabniških opravil.

3.3 IBM BPM

IBM Business Process Manager je IBM-ova platforma za BPM. Ponuja orodja za nadzor, testiranje, nameščanje poslovnih procesov ter vpogled in nadzor nad njimi. Sestavljen je iz dveh delov:

Process Center

- Process Designer je grafični vmesnik za ustvarjanje in testiranje procesov.
- Process Center Console je vmesnik za upravljanje procesov in orodij.
- Integration Designer gradi aplikacije.

Process Server

- Process Admin Console, ki upravlja procese v izvajalnem okolju.
- Process Portal je grafični vmesnik, ki omogoča uporabnikom sodelovanje v procesih.
- WebSphere Application Server Admin Console je vmesnik za upravljanje strežniškega izvajalnega okolja.

3.4 Primerjave izbranih BPMS

Activiti in Camunda

Activiti in Camunda sta si v osnovi zelo podobna, saj se je Camunda ločila od Activitija. Največja razlika je v filozofiji razvoja poslovnih procesov, saj je pri Camundi treba poznati tudi programske jezike - ročno moramo implementirati določena uporabniška opravila, medtem ko Activiti verjame, da za razvoj poslovnih procesov ne potrebujemo programerskega znanja, zato ponuja Designer, v katerem preko grafičnega vmesnika razvijemo poslovni proces. Še ena ključna razlika je, da je Activiti od različice 7.0 naprej v oblaku, medtem ko Camunde še nekaj časa ne bo, saj imajo težave pri razvoju [9].

IBM BPM in Camunda/Activiti

Večje razlike se pokažejo, ko primerjamo staro (IBM BPM) in novo generacijo (Camundo ali Activiti). Nekaj ključnih razlik in težav, ki jih prinaša starejša generacija, je naslednjih:

1. Starejše sisteme je težje namestiti in vzdrževati.
2. Zaprta arhitektura pomeni težjo integracijo v druge sisteme.
3. Uporabljati je treba njihov lastniški pristop k razvoju aplikacij.
4. Težja pridobitev kvalificirane in cenovno dostopne uporabniške podpore.
5. Višja cena lastništva (TCO) zaradi višjih stroškov vzdrževanja.

Poglavje 4

Analiza in načrt rešitve

4.1 Postavitev procesa v vseh izbranih BPMS

Najprej potrebujem preprost proces, ki pokrije večino primerov uporabe BPMS v praksi. Po pregledu poslovnih procesov, ki jih uporabljam na delovnem mestu, so glavni gradniki:

- **logična vrata**, ki usmerjajo tok poslovnega procesa,
- **uporabniška opravila**, za katere je potreben človekov vnos,
- **sistemska opravila**, za katera poskrbi BPMS.

Primer takega procesa je "PrenosOS", ki vodi poslovni proces prenosa osnovnega sredstva med zaposlenimi. Ta proces je primeren tudi zato, ker je interni projekt, za katerega sem osebno zadolžen.

4.2 Izbrane tehnologije

Osnovne zahteve s strani nadrejenih za projekt so, da je aplikacija napisana v **Java 8** in pakirana v mikrostoritev (**Thorntail**) s pomočjo ogrodja **Maven**, kar je industrijski standard za mikrostoritve. To pomeni, da bo aplikacija tekla kot storitev, zavita v t. i. "fat jar"(javanski arhiv, ki vsebuje osnovne komponente strežniškega okolja, potrebnega za delovanje aplikacije). Pred-

nost je, da ne potrebujemo aplikacijskega strežnika, ampak samo Java virtual machine (JVM), kar olajša delo razvijalcem in omogoča storitvi, da deluje v oblaku s pomočjo orodij Docker in Kubernetes.

Izbrane različice so:

- OpenJDK 8,
- Thorntail v2.3.0.Final,
- Maven.

4.3 Integracija storitve v druge aplikacije

Ena izmed bolj pomembnih zahtev za mojo aplikacijo je potreba po integraciji v druge aplikacije, kar pomeni, da potrebujem **vstopno točko** v storitev, s katero lahko komunicirajo druge storitve. V ta namen obstaja vrsta tehnologij, kot so:

- GraphQL,
- REST,
- SOAP.

Med izbirami je **GraphQL** najbolj privlačna izbira, saj poleg poizvedb omogoča tudi obdelavo podatkov, ampak tehnologija še ni dovolj uveljavljena za splošno uporabo.

Ravno obratno je s **Simple Object Access Protocol (SOAP)**, ki je zastarela tehnologija in upočasni razvoj aplikacij v primerjavi z drugimi možnostmi.

Ostane nam **Representational State Transfer (REST)**, ki je najboljša izbira, saj je enostaven in tudi BPMS komunicirajo preko tega protokola. Poglejmo si, kako deluje protokol REST in kako je sestavljen:

HTTP metode, nam predstavljajo operacijo nad določenim virom. Najbolj uporabljene so:

- GET za pridobivanje informacij,
- POST za kreiranje nove entitete,
- PUT za posodobitev entitet in
- DELETE za brisanje entitete.

URI je enolična lokacija vira. To je pot, kamor pošljem klic za upravljanje s storitvijo in BPMS.

Glava vsebuje dodatne podatke o klicu.

Telo vsebuje podatke, ki so potrebni za izvedbo opravil v formatu JSON.

Torej, če sta vstopna in izstopna točka naše storitve preko protokola REST, je naloga moje storitve pretvorba klicev iz BPMS v bolj splošne klice, ki so presek telesa in glave klicev REST do BPMS in so dostopni prek vmesnika na moji storitvi.

Za izvedbo potrebujemo naslednje odvisnosti v projektu:

- Java API for RESTful Web Services (JAX-RS) je vmesnik Java EE, ki navzven izpostavi metode HTTP.
- JSON Processing (JSON-P) je vmesnik Java EE, ki je zadolžen za delo z objekti JSON.

4.4 Znižanje zahtevnosti uporabe

Storitvi lahko na preprost način znižam zahtevnost uporabe z **OpenAPI** 3.0. OpenAPI je del specifikacije MicroProfile od različice 1.4 naprej in se uporablja za dokumentacijo vstopnih točk. Je ključnega pomena za razvijalce,

ker nam nudi dostop do dokumenta *openapi.yaml*, dosegljivega na */openapi*. Naloga *openapi.yaml* je opisati vse metode HTTP, ki jih izpostavlja storitev, in razvijalcem omogoča vpogled v parametre, glavo in telo zahteve ter pošiljanje testnih zahtevkov na vmesnik storitve.

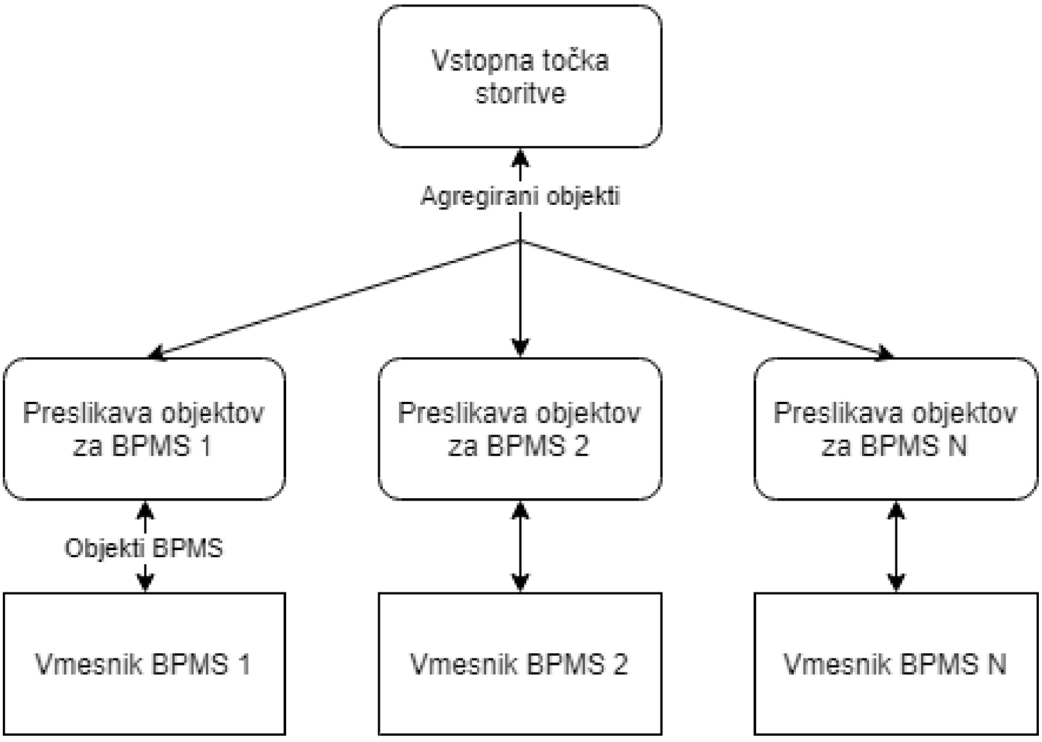
Za izvedbo potrebujemo naslednje odvisnosti v projektu:

- microprofile-config je vmesnik Java EE, ki je zadolžen za upravljanje nastavitev storitve,
- microprofile-openapi je vmesnik Java EE,ki je zadolžen za dokumentacijo vstopnih točk.

4.5 Preslikava objektov in povezava na različne BPMS

Poskrbeti moram tudi za drugo stran storitve, in sicer tisto stran, ki se veže na določen BPMS. Vsi BPMS, ki jih obravnavamo, izpostavljajo svoje vmesnike REST, kar pomeni da je treba poslati klic do BPMS in zajeti njegov odgovor. To je ključni del moje storitve in zahteva, da dobro preučim dokumentacijo Activitija, Camunde in IBM BPM.

Napisati moram obojestransko preslikavo objektov za vsak BPMS, torej pretvorbo klicev moje storitve za izvedbo neke operacije v BPMS v klic, ki ga BPMS razume, in obratno.

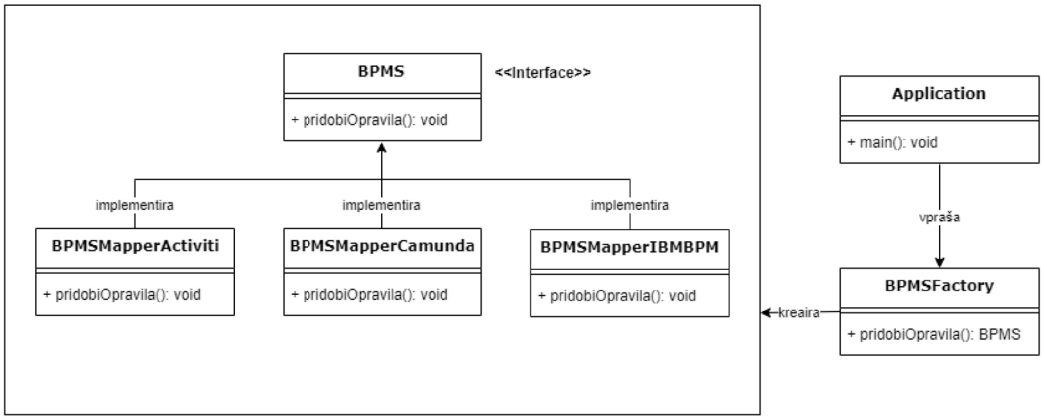


Slika 4.1: Preslikava objektov

Za izvedbo vključim knjižnico Unirest za Javo, ki sicer ni potrebna, ampak poveča produktivnost razvijalcev in omogoča pošiljanje klicev preko protokola REST.

4.6 Logika aplikacije

Ena od zahtev je, da se končnemu uporabniku ni treba zavedati, kateri BPMS teče v ozadju. Možna rešitev je t. i. "Factory Pattern". Ta programski vzorec nam omogoča kreiranje objektov, ne da bi izpostavili logiko kreacije objekta odjemalcu. Na objekt se sklicujemo z uporabo skupnega vmesnika [8].



Slika 4.2: Factory pattern BPMS

Naloga storitve je, da vpraša *BPMSFactory*, ali lahko glede na parametre v okolju aplikacije kreira pravilen objekt, ki vsebuje metode s poslovno logiko komunikacije do specifičnega BPMS, kjer se zgodi pretvorba objektov.

Za izvedbo potrebujemo Contexts and Dependency Injection (CDI), ki nam omogoča upravljanje z življenjskim ciklom komponent s stanjem (t. i. Java Bean) in vstavljanjem odvisnosti.

4.7 Načrt za izvedbo mikrostoritve

1. Postavitev procesa "PronosOS"v vseh BPMS.
2. Preučevanje vmesnikov REST BPMS - zasnova objektov s potrebnimi podatki in izbira klicev, ki jih je treba podpreti.
3. Napisati vstopne točke v storitev in jim dodati dokumentacijo.
4. "Factory pattern" za povezavo med vstopnimi točkami in klici do BPMS.
5. Povezati BPMS z objekti, definiranimi v 2. točki.

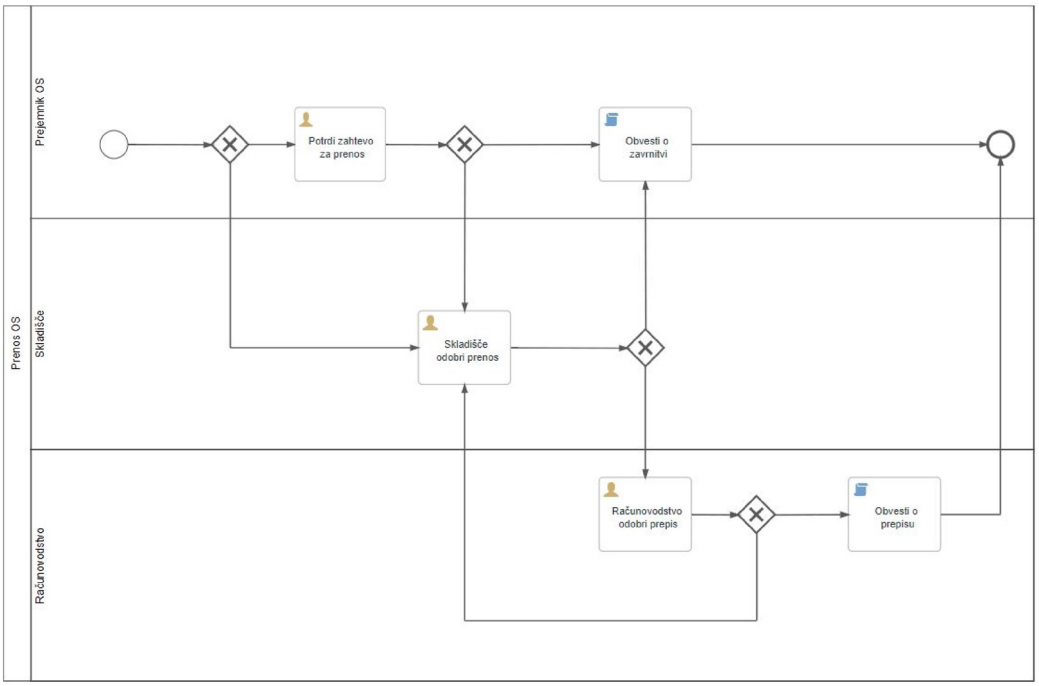
Poglavje 5

Analiza vzorčnega procesa

Kot sem omenil že prej, sem si za poslovni proces, na katerem testiram storitev izbral prenos osnovnih sredstev. Ta proces skrbi za prenos osnovnega sredstva z ene osebe na drugo, ali pa prenos na skladišče v primeru odpisa ali okvare. Na primeru prenosa sredstev so ključni procesi:

- Oddana vloga za prenos sredstva.
- Odobritev prenosa sredstva s strani sedanjega lastnika.
- Knjiženje prenosa v računovodstvu.
- Obvestilo o dokončanem procesu ali o zavrnitvi.

Za pripravo procesa moram najprej modelirati proces z Business Process Modelling Notation (BPMN 2.0.). To je grafična notacija za modeliranje poslovnih procesov in delovnih tokov.



Slika 5.1: Proces prenosa osnovnih sredstev

Najlažje si je predstavljati, da ob zagonu nove instance poslovnega procesa dobim žeton, ki ga nato ob vsakem zaključenem opravilu premaknem po diagramu naprej, glede na odločitve v prejšnjem koraku.

Koraki *Potrdi zahtevo za prenos*, *Skladišče odobri prenos* in *Računovodstvo odobri prenos* t. i. **uporabniška opravila**, ki jih uporabniki ali skupine uporabnikov prejmejo v listo neopravljenih opravil. Takšno opravilo moramo dokončati ročno, po navadi v obliki obrazca, da se žeton premakne naprej.

Opravili *Obvesti o zavrnitvi* in *Obvesti o prepisu* t. i. **skriptni opravili**, kar pomeni, da bo ob prihodu žetona na to opravilo BPMS poskrbel za razrešitev opravila brez sodelovanja uporabnikov.

Drugi pomembni elementi za proces so še **logična ALI vrata** (*kvadrat z križcem v sredini, glej sliko 5.1*), ki usmerijo žeton glede na odločitve iz prejšnih korakov. **Začetno vozlišče** (*krog, glej sliko 5.1*), kjer se proces

začne, in **končno vozlišče** (*odebeljen krog, glej sliko 5.1*), ki predstavlja konec procesa in zaključi proces prenosa osnovnega sredstva.

Začetek procesa

Za začetek procesa mora uporabnik izpolniti vlogo. Na vlogi je treba izbrati sredstva, ki jih želimo prenesti in novega uporabnika (v primeru prenosa na uporabnika), poleg tega ima možnost dodati opombo za lažjo sledljivost procesa. Za ta korak moramo v BPMS sprožiti nov proces in s tem kreirati instanco, ki poskrbi, da bo proces določene vloge potekal po diagramu. Ta dogodek bomo od sedaj naprej imenovali **startProcess**.

Prva vrata

Če smo v prvem koraku izbrali prenos na drugo osebo, je od izbrane osebe zahtevana potrditev ali zavrnitev vloge. To z BPMS storimo tako, da najprej pridobimo vsa opravila s klicem na vmesnik BPMS, ki ga bomo imenovali **getTasks**.

Med svojimi opravili uporabnik, sedaj vidi nedokončano opravilo - *Potrdi zahtevo za prenos*, na katero lahko odgovori s potrdi ali z zavrni. Če oseba zavrne zahtevo za prenos, se vlagatelju vloge pošlje sporočilo o zavrnitvi, v nasprotnem primeru pa se potrjena vloga pošlje v skladišče na obravnavo. Pošiljanje izvede s klicem na vstopno točko **completeTask**, ki zaključi trenutno opravilo in premakne žeton na naslednjo stopnjo.

Zavedati se moramo, da neko opravilo ni nujno dodeljeno osebi, ampak lahko tudi skupini. V takem primeru se kliče vstopna točka **claimTask** in se uporabniku, ki je del skupine, dodeli izbrano opravilo, in onemogoči upravljanje drugim uporabnikom.

Skladišče odobri prenos

V tem koraku skladišče lahko sprejme ali zavrne vlogo glede na podatke iz prejšnjih korakov. V primeru okvare ali odpisa pa moramo procesu dodati še servisni zapisnik. Za izvedbo koraka se kliče dve vstopni točki, **claimTask** za prevzem opravila (saj je skladišče skupina uporabnikov) in **completeTask** za dokončanje opravila.

V primeru zavrnitve se podobno kot pri procesu *Potrdi zahtevo za prenos* proces konča z obvestilom o zavrnitvi vlagatelju vloge, v primeru sprejema pa se vloga pošlje računovodstvu, kar sproži premik žetona korak naprej.

Računovodstvo odobri prenos

Ta korak je podoben koraku *Skladišče odobri prenos*, kjer računovodstvo poknjiži prenos in ga nato sprejme ali zavrne prenos. Za ta korak prav tako kličemo vstopni točki **claimTask** in **completeTask**.

V primeru sprejema vloge in tip prenosa "prenos na novo osebo" se vlagatelju pošlje sporočilo, da je bila vloga sprejeta in je postal novi lastnik osnovnega sredstva.

Če je vloga zavrnjena (npr. pomanjkanje podatkov), se proces premakne nazaj, vlogo pa se pošlje nazaj v skladišče za dopolnitev.

Poglavje 6

Združevanje vmesnikov BPMS

V dokumentaciji Activitija [1], Camunde [2] in IBM BPM [4] moram preučiti naslednje klice:

- začni proces (startProcess),
- pridobi opravila (getTasks),
- prevzemi opravilo (claimTask),
- zaključi opravilo (completeTask).

Pri pregledovanju dokumentacije in orodij IBM BPM za upravljanje vmesnikov REST sem naletel na logično razdelitev klicev:

- klici, povezani s procesi (startProcess) na */processes*,
- klici, povezani z opravili (getTasks, claimTask,completeTask) na */tasks*,
- splošni klici, povezani z upravljanjem BPMS na */search*.

To razdelitev bom uporabil tudi v končni rešitvi storitve, ker naredi dokumentacijo vmesnika bolj pregledno.

6.1 Začni proces (startProcess)

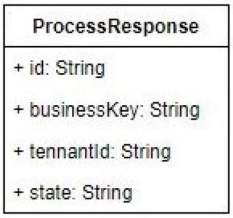
Za klic startProcess, ki naredi novo instanco definicije procesa, potrebujemo naslednje klice:

Activiti POST /runtime/process-instances
s ključem poslovnega procesa (PrenosOS) in spremenljivkami (npr. tip prenosa) v telesu.

Camunda POST /engine-rest/process-definition/key/{key}/start,
kjer je {key} ključ poslovenga procesa, spremenljivke pa so v telesu.

IBM BPM POST /bpm/dev/rest/bpm/wle/v1/process?action=start&bpdId={BPD_ID},
kjer je {BPD_ID} ID procesa, spremenljivke pa so v telesu.

Po identifikaciji polj, potrebnih za delovanje aplikacije prenosa osnovnih sredstev v odgovorih zgornjih klicev, nam ostane objekt **ProcessResponse**.



Slika 6.1: Objekt odgovora startProcess

Kjer sta *id* in *businessKey* kazalnika opravila, na katera se lahko pozneje sklicujemo. *TennantId* je kazalnik skupine, ki je odgovorna za opravilo, *state* pa opisuje stanje, v katerem je proces.

6.2 Pridobi opravila (getTasks)

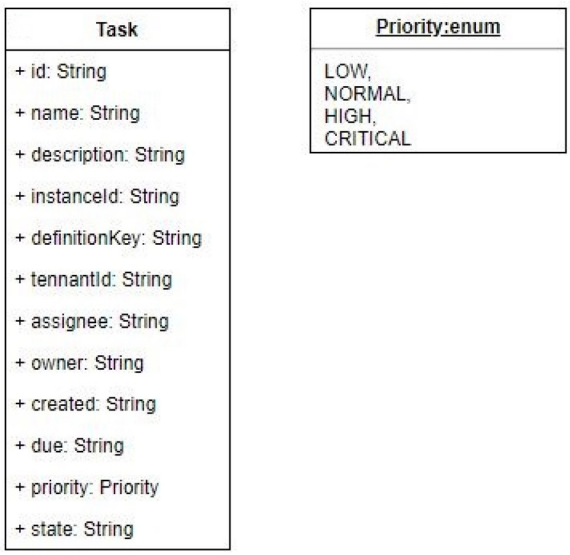
Naloga klica getTask je prikazati listo opravil za določenega uporabnika. Identiteto uporabnika določijo glede na podatek *Authorization* glave zahtevka.

Activiti GET /runtime/tasks

Camunda GET /engine-rest/task

IBM BPM GET /rest/bpm/wle/v1/search/query?
organization=byInstance&run=true&shared=false&filterByCurrentUser=true, kjer moramo nastaviti iskalne parametre, da dosežemo enako delovanje.

Ko združimo odgovore BPMS v obliko, primerno za svojo storitev dobimo **Task**.



Slika 6.2: Objekt odgovora getTasks

6.3 Prevzemi opravilo (claimTask)

ClaimTask prevzame opravilo iz skupine na uporabnika, kar onemogoči drugim uporabnikom upravljanje z opravilom. Podobno kot pri getTasks je tu želja, da bi opravilo prevzel trenutni uporabnik s podatki iz glave zahtevka *Authorization*. To sicer ni osnovno delovanje klica, saj z ustreznimi pravicami prevzamemo opravilo na katerega koli uporabnika, vendar ne nujno nase, ampak v primeru prenosa osnovnega sredstva tega ne potrebujemo.

Activiti POST /runtime/tasks/{id}
Tukaj je {id} kazalnik opravila, ki ga želimo prevzeti, v telesu pa podamo kazalnik uporabnika, ki ga prevzame, podati pa moramo tudi vrednost claim polja action.

Camunda /engine-rest/task/{id}/claim
Tukaj je {id} kazalnik opravila, ki ga želimo prevzeti, v telesu pa podamo kazalnik uporabnika, ki ga prevzame.

IBM BPM PUT /rest/bpm/wle/v1/task/{id}?action=start
Tukaj je {id} id opravila.

V primerjavi s prejšnjimi klici, pri claimTask ni telesa odgovora, ampak nam vrne samo status odgovora. Če je status med vrednostma 200 in 299, vemo, da je bil prevzem uspešen.

6.4 Zaključi opravilo (completeTask)

Naloga completeTask je zaključiti opravilo. Klicu moram dodati tudi spremenljivke, ki so potrebne za dokončanje procesa, v nasprotnem primeru mi BPMS vrne napako.

Activiti POST /runtime/tasks/{id}
Tukaj je {id} kazalnik opravila. V telesu podamo kazalnik uporabnika, podati pa moramo tudi action z vrednostjo complete in spremenljivke, ki so potrebne za izvedbo opravila.

Camunda /engine-rest/task/{id}/complete
Tukaj je {id} kazalnik opravila. V telesu podamo kazalnik uporabnika, podati pa moramo tudi potrebne, ki so potrebne za izvedbo opravila.

IBM BPM PUT /rest/bpm/wle/v1/task/{id}?action=finish
Tukaj je {id} kazalnik opravila, spremenljivke pa so v telesu zahtevka.

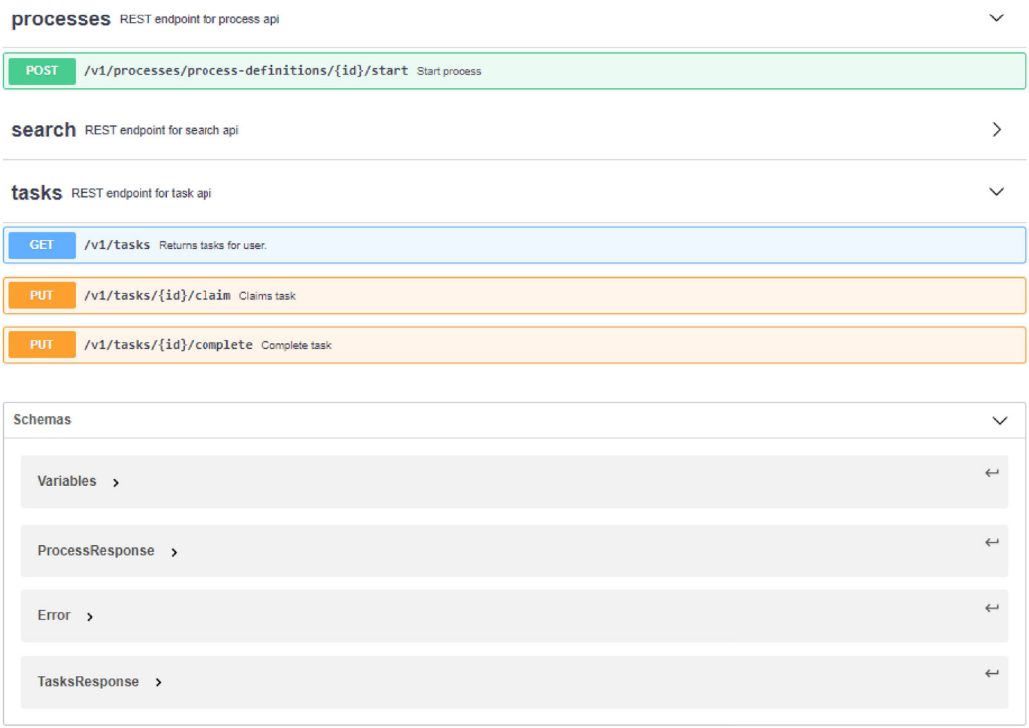
Enako kot pri claimTask tu ni telesa odgovora, ampak nas zanima samo status odgovora.

Poglavje 7

Test in ovrednotenje storitve

7.1 Vstopna točka

Pomemben del vstopne točke je dokumentacija OpenAPI. Storitev izpostavi dokument *openapi.yaml*, ki ga lahko odpremo v poljubnem urejevalniku.



Slika 7.1: Dokumentacija vstopnih točk

Pri pisanju programske kode za vmesnik sem naletel na veliko težav. Testiranje vstopnih točk ni delovalo, ne glede na to da so bili tako testi kot vstopne točke pravilni. Izkazalo se je, da integracija v druge aplikacije zahteva Cross-Origin Resource Sharing (CORS) [5] in je bilo treba dodati t. i. CorsFilter, ki v glavo klicev doda potrebne elemente za klice iz drugih domen.

Za klica *startProcess* in *completeTask* je bilo treba sprejeti poljubne spremenljivke poljubnega tipa, kar pomeni, da odvisnost JSON-P ni znala pretvoriti zahtev v Java objekte. To sem rešil tako, da sem za vrednost sprejel generičen objekt Java in od uporabnika zahteval tip spremenljivke v polju type. To prikazuje naslednji izsek kode:

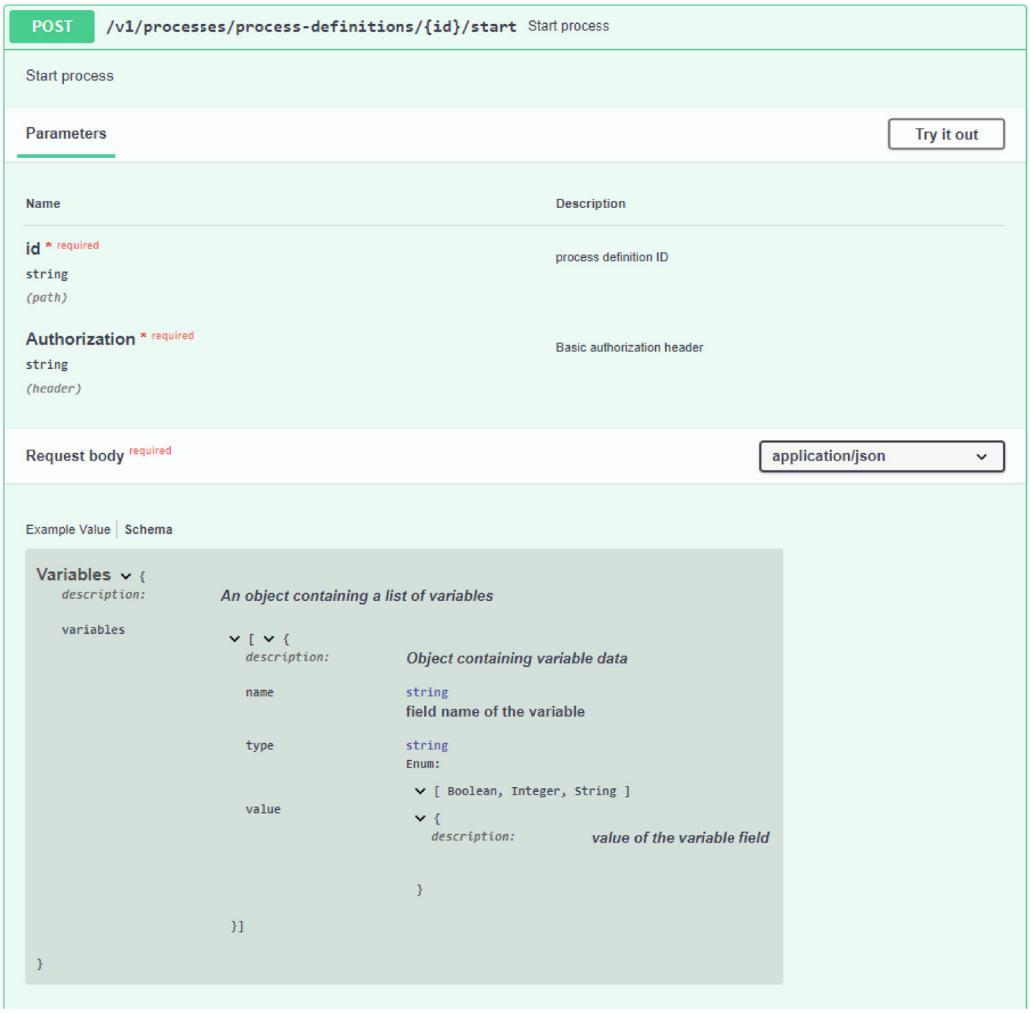
```
@Data
@Schema(description = "Object containing variable data")
public class Variable {
    @NotNull
    @Schema(description = "field name of the variable")
    private String name;
    @NotNull
    @Schema(description = "value of the variable field")
    private Object value;
    private Type type;
}
```

Naslednji izziv je bil, kako hraniti žeton za avtorizacijo in ga prenesti iz vstopne točke do klicev BPMS. Pošiljanje preko parametrov ni najboljša rešitev zaradi uporabe "factory pattern", kar bi pomenilo veliko podvajanja nepomembnih parametrov. Težavo sem rešil z uporabo zrna Java, ki živi za čas zahteve in iz katerega lahko berem žeton na drugih ravneh storitve (t. i. RequestScope Bean).

Uporabnikom sem izpostavil naslednje vstopne točke, ki so bolj podrobno opisane v dokumentu *openapi.yaml*:

- POST /v1/processes/process-definitions/{id}/start za startProcess,
- GET /v1/tasks za getTasks,
- PUT /v1/tasks/id/claim za claimTask,
- PUT /v1/tasks/id/complete za completeTask.

Vsaka izmed njih je opisana kot startProcess na tej sliki:



Slika 7.2: Dokumentacija klica startProcess

7.2 Razvejanje logike

Pri povezavi vstopnih točk s klici do BPMS sem uporabil "factory pattern", opisan v poglavju 4.6. Podobno kot za vstopne točke sem logiko razbil na search, process in task, da bi zmanjšal možnost napake in povečal berljivost. Primer "factory pattern" za startProcess:

```
//po kreaciji BPMFactorya
@PostConstruct
private void init() {
//preberi konfiguracijo
String enabledBpm =
        ReadConfig.getInstance()
        .getConfigByName("bpm-provider");

//uporabi vmesnik z logiko izbranega BPMS
switch (enabledBpm) {
    case "IBM":
        processInterface = processInterfaceIBM;
        break;
    case "Camunda":
        processInterface = processInterfaceCamunda;
        break;
    case "Activiti":
        processInterface = processInterfaceActiviti;
        break;
    default:
        throw new InternalServerErrorException();
    }
}
```

```
//klic startProcess na vmesnik izbranega BPMS
public ProcessResponse startProcess(String id,
Variables variables) throws CustomResponseException {
    return processInterface.startProcess(id, variables);
}
```

Pri implementaciji "factory pattern" sem imel težavo s propagiranjem napak, ki se zgodijo v BPMS do uporabnika moje storitve. Rešil sem jo s pomočjo preslikovalnika napak, ki skrbi za prenos napak čez ravni. Slabost takega pristopa je, da so sporočila uporabniku storitve enaka tistim iz BPMS in ne mojim sporočilom, ki so neodvisna od BPMS. To delno krši zahtevo po tem, da se uporabniku ne bi bilo treba zavedati implementacije BPMS, ampak bi v primeru pošiljanja lastnih napak izgubili preveč informacij o napaki, ali pa bi za implementacijo porabil preveč časa.

7.3 Povezava z BPMS

V tem koraku je večina programske logike moje storitve, saj je treba za vsak klic za vsak BPMS implementirati preslikavo, torej trikrat za BPMS in štirikrat za klice. Primer uspešega odgovora Camundinega startProcess:

```
{"links": [
    {"method": "GET",
     "href":"http://localhost:8080/
      rest-test/process-instance/anId",
     "rel":"self"}],
  "id":"anId",
  "definitionId":"aProcessDefinitionId",
  "businessKey":"myBusinessKey",
  "tenantId":null,
  "ended":false,
  "suspended":false}
```

se pretvori v:

```
{"id":"anId",
 "businessKey":"myBusinessKey",
 "tenantId":null,
 "state":"not_suspended"}
```

in nam v primeru prenosa osebnih sredstev nudi dovolj informacij za izvedbo procesa. Slabost take implementacije je odvisnost od različice vmesnika BPMS. Lahko se zgodi, da ob novi različici ali podpori dodatnega BPMS pride do neskladnosti mojih objektov, kar zahteva spremembo programske kode in povzroči dodatno delo.

7.4 Testiranje rešitve

Za testiranje moram preveriti vse štiri klice in stanje procesa v BPMS. To naredim z naslednjim zaporedjem klicev:

1. Prenos na skladišče - okvara

```
POST /v1/processes/process-definitions/PRENOSOS/start
Host: localhost:8081
Content-Type: application/json
Authorization: Basic YWRtaW46dGVzdA== (admin:test)
{"variables":[
        {"name":"vrstaPrenosa",
         "value":2,
         "type":"String"}]}

ODGOVOR 200 OK
{"id": "22",
 "businessKey": null,
 "tenantId": null,
 "state": "Not suspended"}
```

2. Pridobi opravila - skladišče

```
GET /v1/tasks?start=1&amp;amp; size=3
Host: localhost:8081
Authorization: Basic YWRtaW46dGVzdA==

ODGOVOR 200 OK
{"tasks": [
        {"id": "31",
    "name": "Skladisce odobri prenos",
    "description": null,
    "instanceId": "22",
    "definitionKey": "PROSNAUPA2",
    "tenantId": "",
    "assignee": null,
    "owner": null,
    "created": "2019-03-10T16:44:12.982+01:00",
    "due": null,
    "priority": "CRITICAL",
    "state": null}]}
```

3. Prevzemi opravilo - admin

```
PUT /v1/tasks/31/claim HTTP/1.1
Host: localhost:8081
Authorization: Basic YWRtaW46dGVzdA==

ODGOVOR 204 NO CONTENT
```

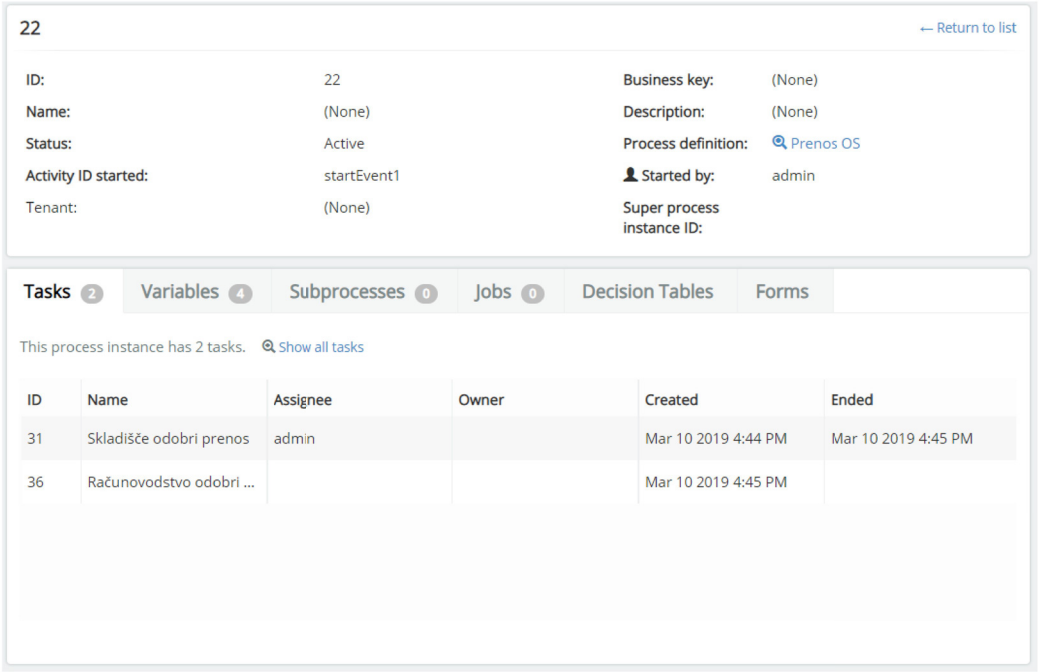
4. Končaj opravilo - admin

```
PUT /v1/tasks/31/complete HTTP/1.1
Host: localhost:8081
Content-Type: application/json
Authorization: Basic YWRtaW46dGVzdA==

{"variables":[
        {"name":"seStrinja",
         "value":true,
         "type":"Boolean"}]}
```

ODGOVOR 204 NO CONTENT

Po izvedbi zgornjih klicev si v BPMS, lahko pogledam stanje procesa. Odločil sem se za Activiti, ker je bolj pregleden, saj vse podatke prikaže na enem mestu.



Slika 7.3: Prikaz stanja procesa v Activiti

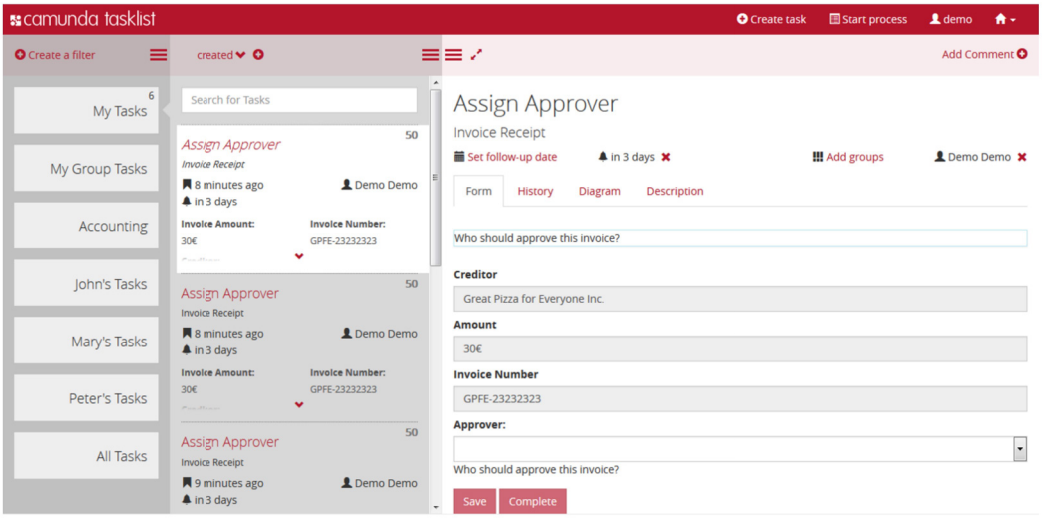
Na zgornji sliki vidimo začetek procesa z ID 22, ki je zaradi *vrsteProsena* 2 premaknil žeton na skladišče in odprl novo opravilo z ID 31 (vidno pri klicu pridobi opravila). Zatem ga je prevzel in dokončal uporabnik z imenom admin, ki je zaključil opravilo z ID 31, kar je prestavilo proces na računovodstvo.

Enako se zgodi tudi pri drugih klicih do zaključka procesa, ki bomo zaradi preglednosti izpustili, ker je dovolj, da prikažemo delovanje štirih osnovnih klicev, ki nam kateri koli enostaven proces pripeljejo do konca. Prav tako bom izpustil prikaz za druge BPMS, saj je edina razlika drugačen uporabniški vmesnik.

7.5 Nadaljnje delo

7.5.1 Uporabniški vmesnik za prikaz opravil

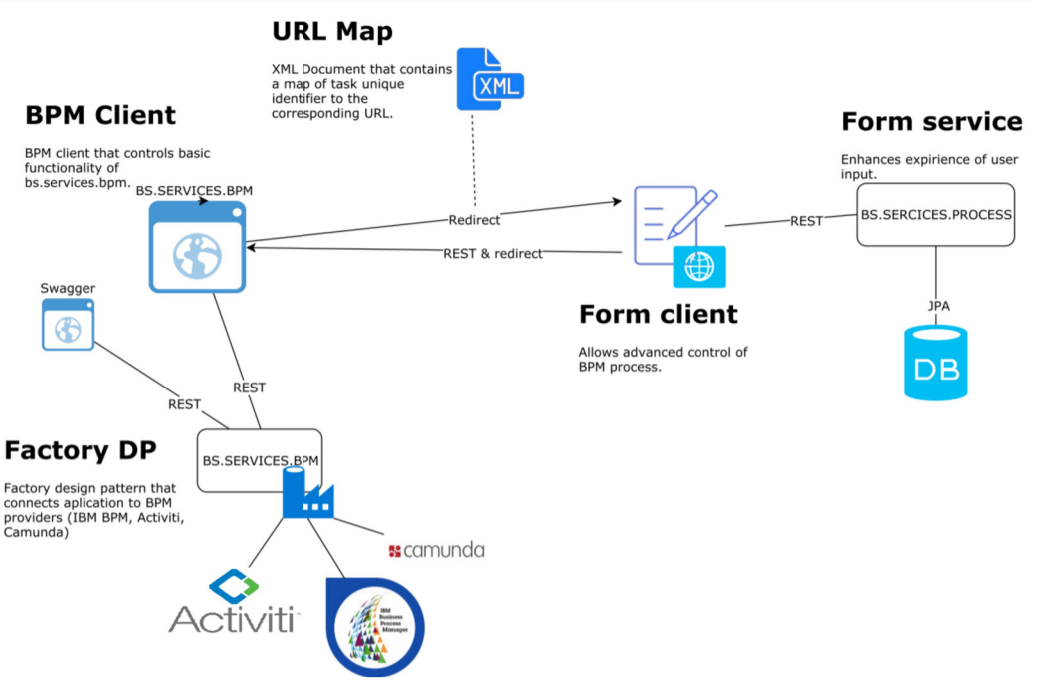
Uporabna izboljšava bi bil uporabniški vmesnik za prikaz opravil. Preko njega bi uporabnik lahko upravljal opravila na enak način, kot sedaj to omogočajo BPMS s svojimi uporabniškimi vmesniki za prikaz opravil. V primerjavi z drugimi uporabniškimi vmesniki BPMS bi na svojega lahko povezal poljuben BPMS ali celo več BPMS in s tem pridobil vpogled v vse sisteme na enem mestu.



Slika 7.4: Uporabniški vmesnik za opravljanje opravil v Camundi

7.5.2 Generiranje obrazcev

Če bi želel, da je uporabniški vmesnik interaktiven, bi potreboval dodatno storitev, ki bi znala dinamično ustvariti obrazec glede na spremenljivke in tip spremenljivk za izvedbo opravila. S tem bi v celoti preko grafičnega vmesnika podprl tematiko tega diplomskega dela in omogočil uporabnikom, ki niso vešči programiranja upravljanja z BPMS, na podoben način, kot to omogoča Activiti.



Slika 7.5: Diagram povezave storitve za generiranje obrazcev z obstoječo arhitekturo

Poglavje 8

Sklepne ugotovitve

Znižanje zahtevnosti uporabe in integracija v druge aplikacije sta zahtevi, ki sta po mojem mnenju najbolj izpolnjeni. OAS je močno orodje, ki s primeri in z možnostjo testiranja klicev razvijalcem olajša delo z vmesniki. To stori do te mere, da se storitev na mojem delovnem mestu uporablja kot predloga projekta za nove razvijalce, ki se učijo Jave. Poleg tega obstajajo orodja, ki generirajo logiko za odjemalca in s tem ponudijo razvijalcem moje storitve začetno točko, iz katere lahko razvijejo svojo aplikacijo.

Uspešno pokrite osnovne funkcionalnosti BPMS je še ena zahteva, ki se mi zdi uspešno pokrita. Vmesniki REST pri vseh obravnavanih BPMS pokrijejo glavne elemente BPMN 2.0 in omogočajo upravljanje enostavnih poslovnih procesov. Slabost je, da ni mogoče podpreti čisto vseh, obstajajo namreč klici, ki so specifični za določene BPMS (določeni klici so v enem BPMS enakovredni večim klicem v drugih BPMS, določeni klici pa morda sploh niso podprti). Lahko se nam zgodi, da ob spremembi različice BPMS pride do napak v komunikaciji med aplikacijami, kar zahteva dodaten vložek z naše strani in nam onemogoči upravljanje osnovnih funkcionalnosti.

Znižanje cene lastništva je zahteva, za katero nisem najbolj prepričan, ker je bilo za analizo, raziskovanje in implementacijo porabljenega kar nekaj časa, delo pa še zdaleč ni končano, saj je storitev treba sproti dopolnjevati

glede na potrebe podjetja in skrbeti za skladnost z BPMS. Vložek za razvoj servisa je večji, kot bi bil vložek za izobrazbo novega kadra. Težavo bi lahko rešili, če bi storitev ponudili kot produkt, kar pa bi zahtevalo velik finančni vložek.

Poleg tega nisem zadovoljen z implementacijo propagiranja napak med BPMS in uporabnikom, ker storitev trenutno pošilja napake iz BPMS neposredno do uporabnika, kar je sicer uporabno za razhroščevanje, ampak uporabnika v primeru napake naredi odvisnega od BPMS, ki delujejo v ozadju, in njihovih orodij za vpogled v sistem. To pomeni, da še vedno potrebujemo nekoga, ki zna upravljati točno določen BPMS, zato je na tej točki lažje, če pri razvoju sodeluje strokovnjak, kot pa da razvijam aplikacijo za interno uporabo, ki je časovno bolj potratna.

Literatura

[1] Activiti. Activiti user guide. Dosegljivo: https://www.activiti.org/userguide/#_rest_api, 2017. [Dostopano: 4. 3. 2018].

[2] Camunda. Rest api reference. Dosegljivo: https://docs.camunda.org/manual/7.10/reference/rest/, 2015. [Dostopano: 4. 3. 2018].

[3] CREA. Kaj je bpm? Dosegljivo: http://www.crea.si/WWW/Pages/Ultimus/BPM.aspx. [Dostopano: 31. 7. 2018].

[4] IBM. Rest interface for bpd-related resources. Dosegljivo: https://www.ibm.com/support/knowledgecenter/SSFTN5_8.0.1/com.ibm.wbpm.ref.doc/rest/bpmrest/rest_bpm_wle.htm. [Dostopano: 4. 3. 2018].

[5] MDN. Cross-origin resource sharing (cors). Dosegljivo: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS, 2019. [Dostopano: 4. 3. 2018].

[6] Jim Rudden. Making the case for bpm: A benefits checklist. *BPTrends*, pages 1–4, 2007.

[7] Bruce Silver. *BPMN Method and Style, with BPMN Implementer's Guide: A structured approach for business process modeling and implementation using BPMN 2.0.* Cody-Cassidy Press Aptos, 2011.

[8] TutorialsPoint. Design pattern - factory pattern. Dosegljivo: https://www.tutorialspoint.com/design_pattern/factory_pattern.htm. [Dostopano: 2. 3. 2018].

[9] zambrovski. camunda-bpm-cloud. Dosegljivo: https://github.com/holunda-io/camunda-bpm-cloud, 2018. [Dostopano: 3. 3. 2018].