Razvoj programskog sučelja za konverzacijskog agenta vođenog modelom procesa

Starčić, Toni

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

Završni rad

Razvoj programskog sučelja za konverzacijskog agenta vođenog modelom procesa

Toni Starčić

Pula, rujan, 2020. godine

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

Završni rad

Razvoj programskog sučelja za konverzacijskog agenta vođenog modelom procesa

(Development of API for a process model-driven chatbot)

Toni Starčić

JMBAG: 0303055459

Studijski smjer: Informatika

Kolegij: Poslovni informacijski sustavi

Mentor: doc. dr. sc. Darko Etinger

Komentor: doc. dr. sc. Nikola Tanković

Pula, rujan, 2020. godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Toni Starčić, ovime izjavljujem da je ovaj završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio završnog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

U Puli, rujan, 2020. godine

IZJAVA O KORIŠTENJU AUTORSKOG DJELA

Ja, dolje potpisani Toni Starčić dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom „Razvoj programskog sučelja za konverzacijskog agenta vođenog modelom procesa“ koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama. Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

Student

U Puli, rujan, 2020. godine

Sadržaj

Zahvala

Ovim putem se zahvaljujem mentoru doc. dr. sc Darku Etingeru i komentoru doc. dr. sc Nikoli Tankoviću na pruženoj prilici da radim na zanimljivom projektu koji je ujedno i postao moj završni rad. Hvala Vam na pomoći i na svim konzultacijama i smjernicama kako bi ovaj rad uspješno priveli kraju. Nadam se da ćemo ovo ponoviti

Hvala i kolegici Nikki Bernobić na sjajnoj komunikaciji, odličnoj suradnji i zabavnoj radnoj atmosferi.

Hvala od srca obitelji, djevojci i prijateljima na razumijevanju i bezuvjetnoj pomoći. Ne bih mogao bez vas.

Uvod

Sustavi raznih organizacija, kao što su poduzeća ili javne službe, sastoje se od složenih procesa koje nije moguće unaprijediti eksperimentiranjem u realnom vremenu, pa se iz toga razloga razvijaju modeli tih poslovnih procesa kako bi se izvela nova rješenja čiji je ultimativni cilj – automatizacija poslovnih procesa. Disciplina ili sistematska metoda promatranja poslovnih procesa jest Upravljanje poslovnim procesima (eng. Business Process Management) koja se koristi za otkrivanje, modeliranje, analizu, mjerenje, poboljšanje, optimizaciju i automatizaciju poslovnih procesa. „Prema genetičkoj je definiciji poslovni proces … povezani skup aktivnosti i odluka, koji se izvodi na vanjski poticaj radi ostvarenja nekog mjerljivog cilja organizacije, traje određeno vrijeme i troši određene resurse pretvarajući ih u specifične poizvode ili usluge važne za kupca ili korisnika.“[1]

Kao jedan od praktičnih primjera automatizacije generalno, pa tako i procesne automatizacije, su konverzacijski agenti (eng. Chatbots). Konverzacijski agent je softverska aplikacija koja nudi mogućnost konverzacije, a posebno je koristan u situacijama gdje je interakcija naglašena, odnosno dijalog.

Naime, dijalog je najprirodniji način komunikacije kod ljudi, a od računalnih početaka, znanstvenici su nastojali razviti što je moguće prirodniju komunikaciju čovjeka sa strojem. Prvi takav konverzacijski agent, odnosno računalni program napravljen je prije pola stoljeća na MIT-evom laboratoriju za umjetnu inteligenciju naziva „ELIZA“. Danas, pola stoljeća kasnije, konverzacijski agenti su primjenjivi gotovo u svim granama gospodarstva i time imaju visoku komercijalnu vrijednost.

„Fibot“ je web aplikacije koja je razvijena u svrhu završnog rada kao dokaz o konceptu spajanja modela poslovnog procesa i konverzacijskog agenta. Zašto su modeli i

[1] Brumec, Josip i Slaven Brumec. "Modeliranje poslovnih procesa." *Zagreb, Školska knjiga* (2018).

konverzacijski agenti međusobno komplemetarni? Model omogućuje analizu i optimizaciju, a sam chatbot se može promatrati kao sučelje ili kao pogon tog istog modela.

U radu će biti prikazano programsko rješenje i implementacija, s time da će se spomenuti i Camunda platforma koja omogućuje samu implementaciju. Osim Camunde bit će spomenuta arhitektura i struktura aplikacije, a poseban je naglasak na objašnjenju načina rada programskog sučelja.

Programsko rješenje i implementacija

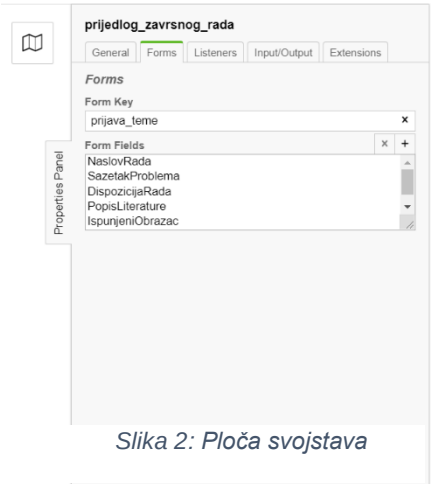
BPMN 2.0

Camunda Platforma

Camunda[3] je platforma otvorenog koda (eng. Open source) i dana je na korištenje pod Apache License 2.0. Camundina platforma je odabrana zbog dokumentiranosti, korisničke podrške i REST API-ja koji nudi povezivanje aplikacije na njihov BPMN engine koji pogoni dijagram poslovnog procesa. Između ostalog tu je i grafički editor imena Camunda Modeler[4] čije je sučelje vidljivo na slici 1. i tri web aplikacije koje dolaze u paketu, a imena su Camunda Tasklist[5], Camunda Admin[6] i Camunda Cockpit[7]. Camunda osim što nudi široki spektar usluga ima i specifičnu terminologiju koja će iz tog razloga biti objašnjena u posebnom ulomku.

Camunda Modeler

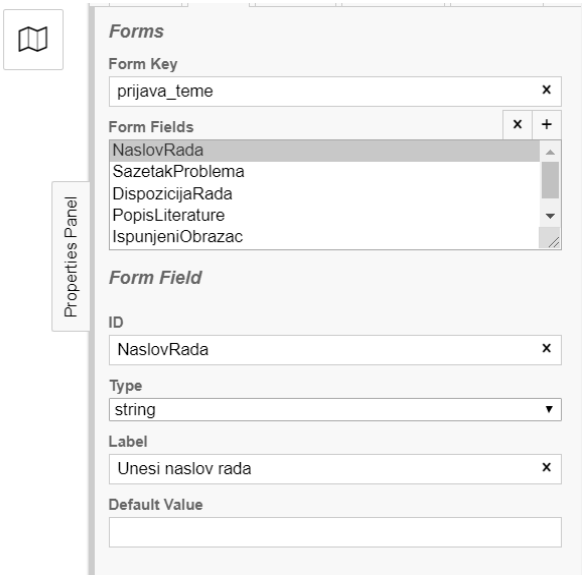
Camunda Modeler je desktop aplikacija koja služi za modeliranje dijagrama poslovnih procesa. Koristi BPMN 2.0 notaciju. Na slici 1. s lijeve strane sučelja vidljivi su simboli koje je kasnije moguće modificirati, odnosno detaljnije specificirati. Primjerice, s funkcionalnošću povuci i ispusti (eng. drag&drop) se prazan zadatak povuče na praznu podlogu koja funkcionira kao radna ploha, pa se zatim ponudi opcija u obliku francuskog



Slika 2: Ploča svojstava

[3] https://camunda.com/
[4] https://camunda.com/products/camunda-bpm/modeler/
[5] https://docs.camunda.org/manual/latest/webapps/tasklist/
[6] https://docs.camunda.org/manual/latest/webapps/admin/
[7] https://docs.camunda.org/manual/latest/webapps/cockpit/

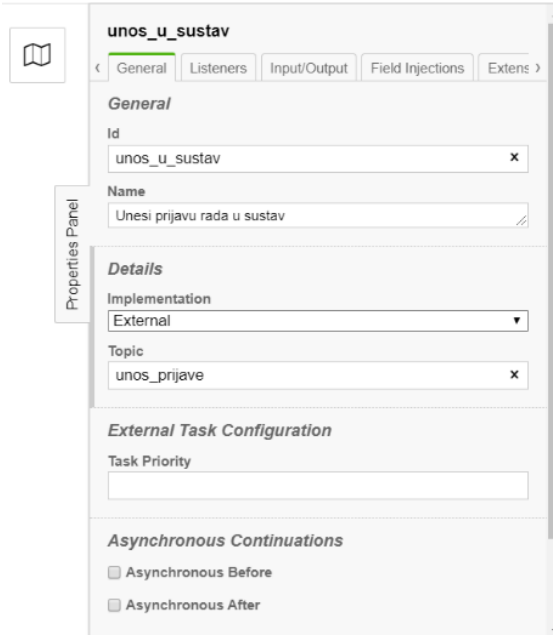
ključa kojem je moguće praznu aktivnost pretvoriti u razne specijalizacije, kao što je korisnički ili servisni zadatak. S desne strane vidljiva je ploča svojstava (eng. Properties Panel) koja se koristi za specifikaciju svakog korištenog simbola dijagrama. U gornjem dijelu je alatna traka koja nudi razne mogućnosti, a ističe se gumb koji omogućuje podizanje (eng. Deployment) procesa na server što zapravo čini dijagram procesa interaktivnim.



Slika 3: Definiranje varijabli u ploči svojstava

Važno je napomenuti da je moguće definirati takozvana polja forme korisničke aktivnosti koja generiraju varijable i kojima se može pristupiti na razne načine. Na slici 2. je vidljiv „Ključ forme“ (eng. Form Key) koji imenuje formu radi postizanja cilja različitosti jer se može dogoditi da XML dijagrama procesa, tj. dijagram u grafičkom obliku sadrži više od jednog polja formi. Ispod „Ključa forme“ vidljiva su „Polja forme“ (eng. Form Fields) koja specificiraju varijable kao što je vidljivo na slici 3. gdje se navode ID, tip i oznaka varijabli. Tim varijablama u aplikaciji „Fibot“ pristupa se i koristi na dva načina: pomoću modula Vue-form-generator[8] koji na temelju dobivenih varijabli generira polja formi i pomoću modula Vue-advanced-chat[9] koji njima pristupa i koristi preko chat input forme. No, Vue-form-generator bit će objašnjen kasnije.

[8] https://github.com/vue-generators/vue-form-generator
[9] https://github.com/antoine92190/vue-advanced-chat



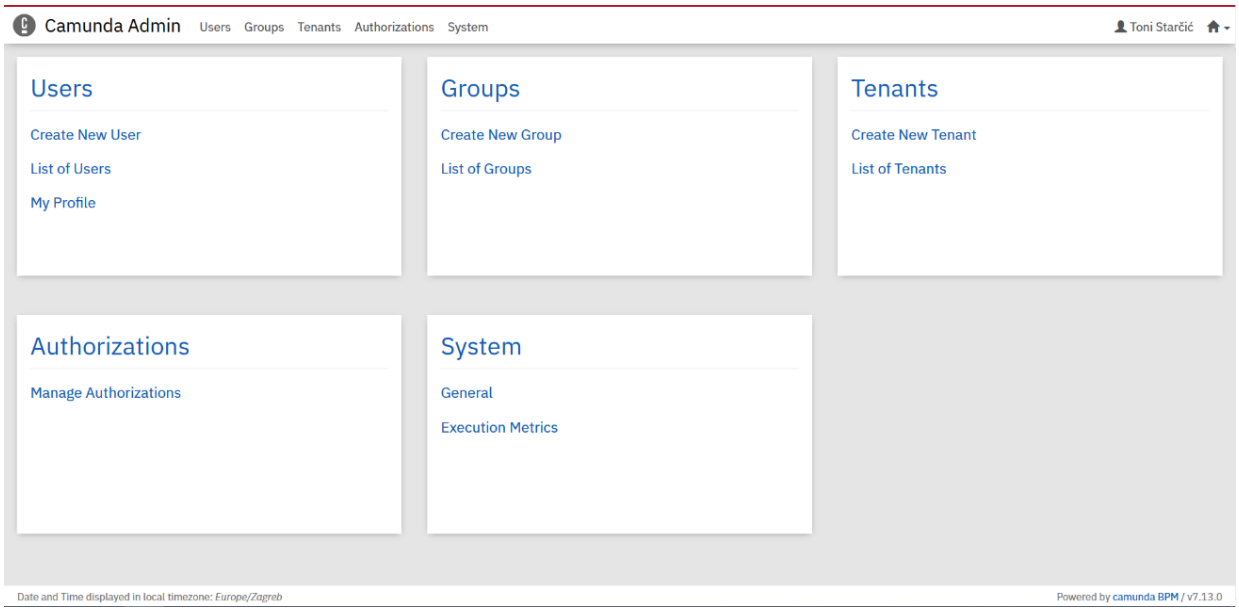
Slika 4: Ploča svojstava vanjskog zadatka

Osim polja u korisničkom zadatku, nužno je specificirati i svojstva servisnog zadatka. U dijelu detalji navedena je vanjska vrsta implementacija (eng. External) koja navodi da će se izvršiti van procesnog enginea, a u web aplikaciji „Fibot“ koriste se prilagođene Python[10] skripte. U polju ispod unosi se zadatak kojem će se pristupati preko procesnog enginea. U ploči svojstava postoje i mnoge druge opcije, no nisu korištene iz razloga što se koristio programski jezik Python, a opcije nameću implementacijsku logiku u programskom jeziku Java[11].

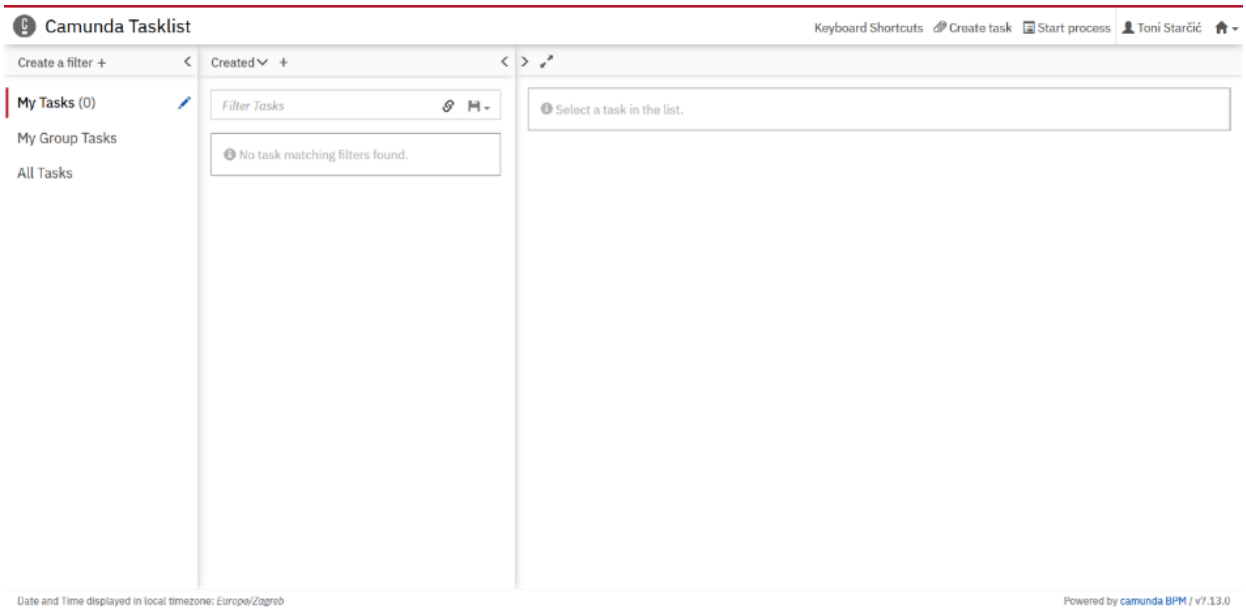
Camunda Web aplikacije

Nakon što se dijagram procesa nacratan u aplikaciji Camunda Modeler „podigne“ na server, moguće ga je koristiti u interakciji s tri Camundine aplikacije. Camunda Admin koja nudi mogućnost definiranja grupa i korisnika radi identifikacije aktera, tj. sudionika u procesu. Camunda Tasklist je web aplikacija koja pruža mogućnost korisniku, odnosno akteru, da odradi zadatak koji mu je dodijeljen u dijagramu procesa. Camunda Cockpit je web aplikacija koja služi za praćenje stanja pokrenutnih instanci procesa i koristila češće od ostalih aplikacija za razvijanje aplikacijskog sučelja. Sučelja triju aplikacija su vidljiva na slikama 5., 6. i 7.

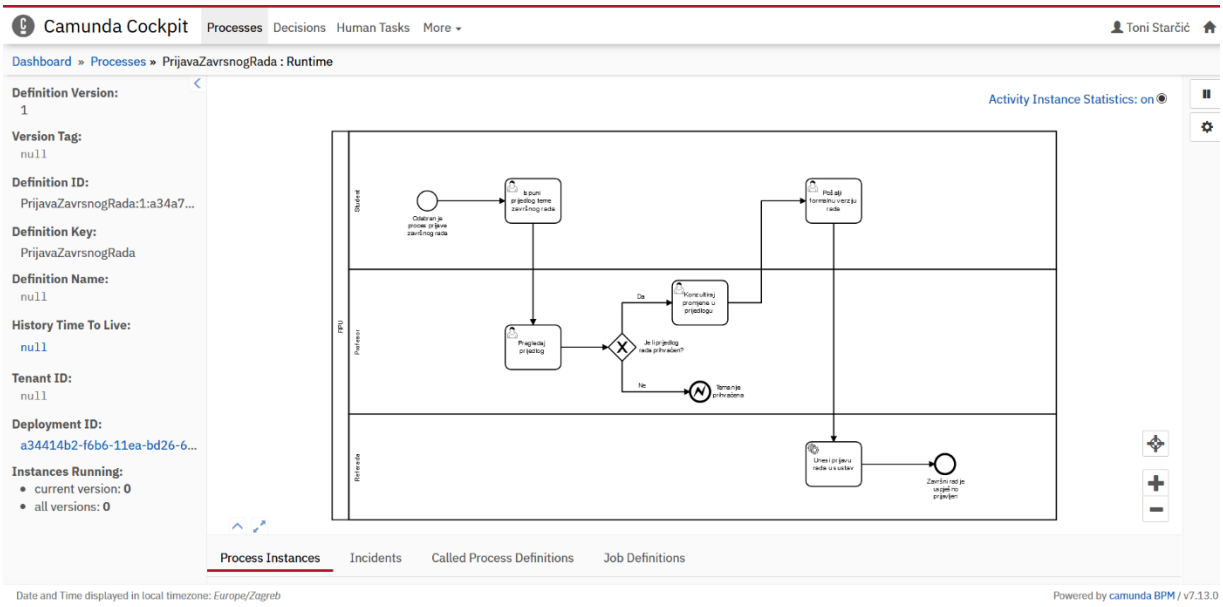
[10] https://www.python.org/
[11] https://www.java.com/en/



Slika 5: Korisničko sučelje Camunda Admin



Slika 6: Korisničko sučelje Camunda Tasklist



Slika 7: Korisničko sučelje Camunda Cockpit

Ključni pojmovi Camundinog procesnog enginea[12]

Definicija procesa (eng. Process Definition[13]) odgovara jednom dijagramu procesa, tj. modelu procesa. Procesna instanca (eng. Process Instance[14]) se nalazi hijerarhijski ispod definicije procesa, ako se koristi analogija objektno-orijentirane paradigme, procesna definicija bi odgovarala klasi, a procesna instanca jednom objektu te klase.

Zadatak (eng. Task) odgovara jednoj jedinici rada koja pripada dijagramu procesa. Ima svoje svoje trajanje i resurse koji su potrebni kako bi se izvršila. Camunda nudi više vrsta zadataka, primjerice manualni taskovi, itd. Korištene su dvije vrste: korisnički zadatak (eng. User Task[15]) kojeg izvodi ljudski akter i Servisni zadatak (eng. External Task[16]) kojeg izvodi neljudski akter, odnosno stroj.

Aktivnost s neljudskim akterom (eng. External Task), odnosno strojem je jedinica posla koju radi neki radnik (eng. Worker), pritom se za radnika misli na udaljeno računalo.

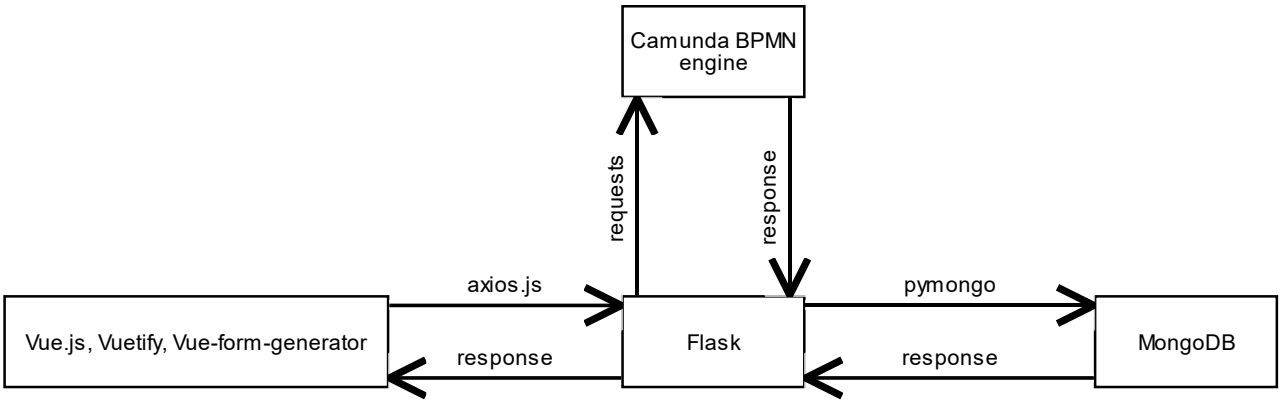
[12] https://docs.camunda.org/manual/latest/user-guide/process-engine/process-engine-concepts/
[13] https://docs.camunda.org/manual/7.13/reference/rest/process-definition/
[14] https://docs.camunda.org/manual/7.13/reference/rest/process-instance/
[15] https://docs.camunda.org/manual/7.13/reference/rest/task/
[16] https://docs.camunda.org/manual/7.13/reference/rest/external-task/

Konceptualno gledajući, aktivnost s neljudskim akterom napravljena je na način da radnik preuzme aktivnost, zaključa ju na određeni vremenski period kako ne bi došlo do pogrešaka u preuzimanju zadataka, budući da jedan zadatak može imati isključivo jednog radnika. Nakon što je zadatak zaključan na neki vremenski period, radnik ima onoliko vremena da odradi taj zadatak ili će u protivnom zadatak ostati neodrađen.

Camunda nudi proširenje u obliku formi korisničkih zadataka (eng. User Task Forms). Radi se o web formama koje služe akteru, odnosno zaduženik (eng. Assignee) da unese potrebne podatke koji su navedeni u Camunda Modeleru kao što je vidljivo na slici 3. Postoji četiri vrste formi: Uklopljene forme (eng. Embedded Task Forms), Generirane forme (eng. Generated Task Forms), Generičke forme (eng. Generic Task Forms) i Vanjske forme (eng. External Task Forms) [17]. Ove potonje su forme koje se izvršavaju van Camunda Tasklista, odnosno u drugoj aplikaciji, kao što je to slučaj s aplikacijom „Fibot“.

[17] https://docs.camunda.org/manual/latest/user-guide/task-forms/

Arhitektura



Slika 8: Dijagram arhitekture aplikacije

Model za strukturu sustava web aplikacije „Fibot“ je troredni klijent-poslužitelj. Prvi sloj je prezentacijski sloj, drugi je sloj obrade i treći je sloj upravljanja podacima distribuiranog sustava. Prezentacijski sloj, odnosno (eng. Frontend) implementiran je pomoću Vue.js[18] progresivnog aplikacijskog okvira u programskom jeziku JavaScript[19], a nadograđen je s Vuetify[20] UI/UX komponentama koje ovise o Bootstrapu[21] i koje su razvijene kako bi bile potpuno kompatibilne s Vue.js aplikacijskim okvirom. Osim VueJs-a i Vuetify modula, koristi se Axios.js[22] paket (eng. Package) za povezivanje klijenta s poslužiteljem u obliku HTTP poziva i Vue-form-generator kao izdvojeni slučaj uporabe, gdje se nastoji prikazati mogućnost korištenja i tog dijela Camundinog API-ja, tj. BPMN engine-a, nauštrb konverzacijskog agenta imena „Fi“. Sloj obrade, tj. (eng. Backend) implementiran je u Flasku[23] – web aplikacijskom okviru napisanom u Pythonu. Naime, Flask nudi rute prema klijentu, tj. korisničkom sučelju, a ujedno je i Flask sučelje prema Camundinom engineu te prema bazi podataka, ali programsko sučelje. Flask sučelje napisano za Camundu je implementirano pomoću modula Requests[24]. „Requests je elegantna i jednostavna HTTP

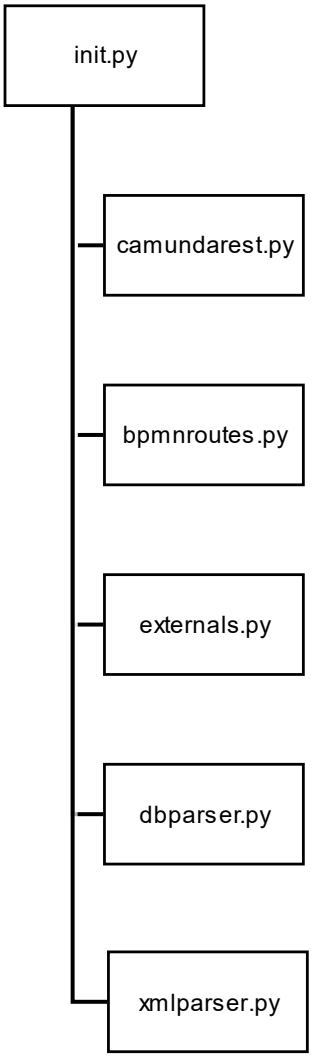
[18] https://vuejs.org/
[19] https://developer.oracle.com/javascript/
[20] https://vuetifyjs.com/en/
[21] https://getbootstrap.com/
[22] https://github.com/axios/axios
[23] https://flask.palletsprojects.com/en/1.1.x/
[24] https://requests.readthedocs.io/en/master/

biblioteka za Python, napravljena za ljudska bića.“[24] Za sloj upravljanja podacima korišten je MongoDB[25] koji je klasificiran kao NoSQL baza podataka bazirana na kolekcijama dokumenata. Kolekcije (eng. Collection) su veće cjeline koje čine bazu podataka, a analogne su relacijskim tablicama u relacijskim bazama podataka. MongoDB sprema podatke u obliku BSON dokumenata (eng. Documents). BSON je binarna reprezentacija JSON podataka. Strukturu dokumenata čine polje-vrijednost parovi, gdje je polje ime podataka, a vrijednost para je vrijednost tog podatka. Od šest kolekcija koje sadrži baza podataka, gledano sa stajališta poslužitelja i Camundinog web servise, istaknute su „chatRooms“, „users“ i „submittedApplications“. „chatRooms“ je jednaka procesnoj instanci jedne procesne definicije, a s gledišta frontenda je jedna soba za komuniciranje između aktera procesa. Sadrži gotovo sve podatke koji su potrebni kako bi se proces izvršio od početka do kraja. Kolekcija „users“ je kolekcija korisnika, kao što samo ime govori, a koristi se za mnoge API pozive u kombinaciji s kolekcijom „chatRooms“ i zadnja kolekcija je „submittedApplications“, u koju se spremaju procesi koji su ispravno i u potpunosti završeni.

[25] https://www.mongodb.com/

Struktura

Poslužitelja čini pet datoteka, a to su: „camundarest.py“, „bpmnroutes.py“, externals.py“, „dbparser.py“ i „xmlparser.py“. Svaka datoteka ima svoju namjenu i zastupljenost.



Slika 9: Datotečna hijerarhija

Datoteka „dbparser.py“ implementirana je kako bi se varijable spremljene u bazi podataka, pripremile za prikaz na klijentu. Uglavnom se koristi kod strukturiranja poruka koje konverzacijski agent ispisuje u chat sobama.

Kao substitut konverzacijskom agentu, smišljene su forme koje se generiraju na temelju prethodno definiranih varijabli kao što je već objašnjeno u sekciji Camunda modeler. Za generaciju formi korišten je Vue-form-generator koji kreira forme na temelju dva objekta modela i scheme. Model je objekt koji sadrži vrijednosti varijabli, a schema je objekt koji opisuje varijablu. Primjerice, nalaže hoće li varijabla biti prikazana kao polje za tekstualni upis (eng. Textarea) ili polje za označavanje (eng. Checkbox). Kako Camundin REST API za dohvat varijabli ne vraća oznaku (eng. Label) navedenu u Modeleru, odlučeno je da će se oznake povlačiti parsiranjem XML-a[26] dijagrama. Datoteka u kojoj se parsiranje radi jest „xmlparser.py“. Modul koji se koristi za parsiranje je ElementTree[27] koji sadrži x-path podršku. Parametar nužan za identifikaciju dedicirane forme je „form_key“ koji je vidljiv na slici 4. Nakon što se forma identificira, slijedi prolazak kroz polja formi koja se zatim pretvaraju u prethodno opisane model, odnosno schema.

[26] https://www.w3.org/XML/
[27] https://docs.python.org/3.9/library/xml.etree.elementtree.html

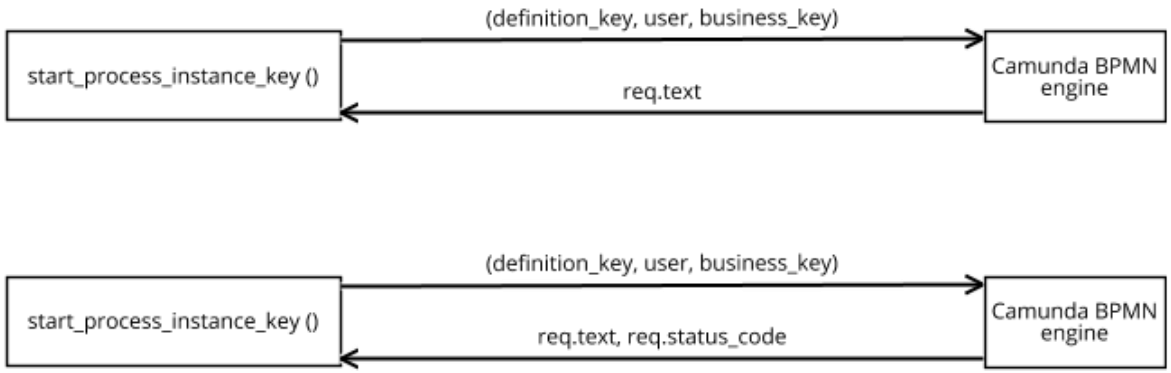
```
<bpmn:userTask id="prijedlog_zavrsnog_rada" name="Ispuni prijedlog teme završnog rada" camunda:formKey="prijava_teme" camunda:assignee="${initiator}">
  <bpmn:extensionElements>
    <camunda:formData>
      <camunda:formField id="NaslovRada" label="Unesi naslov rada" type="string" />
      <camunda:formField id="SazetakProblema" label="Unesi sažetak rada" type="string" />
      <camunda:formField id="DispozicijaRada" label="Unesi dispoziciju rada" type="string" />
      <camunda:formField id="PopisLiterature" label="Unesi literaturu" type="string" />
      <camunda:formField id="IspunjeniObrazac" label="Ispunjeni obrazac" type="boolean" />
      <camunda:formField id="Mentor" label="Odaberi mentora" type="string" />
    </camunda:formData>
  </bpmn:extensionElements>
```

Slika 10: Isječak XML-a dijagrama

Datoteku „externals.py“ čine Python skripte koje oponašaju servise, a napisani su da budu potpuno komplementarni modelima. Koristi se i PyMongo[28] za pristup bazi podataka te modul „Time“ za sinkronizaciju poziva.

Povezanost na Camundin engine je ostvaren u datoteci „camundarest.py“. Svi dijagrami prikazuju pozive prema Camundinom engineu, s time da gornja instanca prikazuje uspješne pozive, a doljnja neuspješne. U lijevom pravokutniku se nalazi ime funkcije koja radi poziv, a s desne je naznačen „Camunda BPMN engine“ kao samostalni entitet. Na gornjim strelicama su navedeni parametri, a na doljnjim odgovori Camundinog web servisa, odnosno API-ja.

```
def start_process_instance_key(definition_key, user, business_key)
```



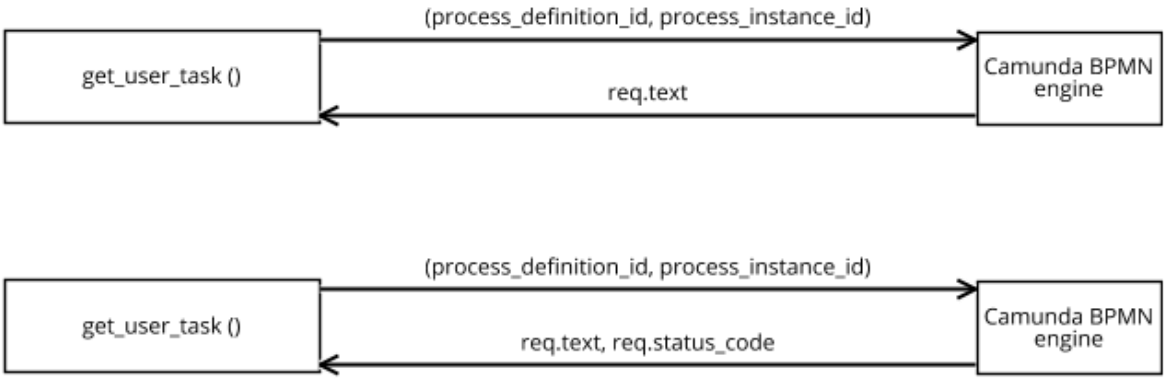
Slika 11: API poziv za pokretanje procesa

Slanje zahtjeva „start_proces_instance_key()“ pokreće procesnu instancu u Camundinom engineu. Parametar „definition_key“ ovisi od procesa do procesa,

[28] https://pymongo.readthedocs.io/en/stable/

uglavnom se radi o ljudski čitljivom stringu koji predstavlja ime procesa. Korisnik, ili parametar koji predstavlja korisnika - „user“ se koristi kao posebna varijabla „initiator“ koji pokretača procesne instance zadužuje na prvi zadatak, a „business_key“ se generira na poslužitelju kao kombinacija ostalih parametara. Parametar „business_key“ je ljudski čitljiva verzija „process_instance_id“, no ne koristi se obilato u aplikaciji „Fibot“.

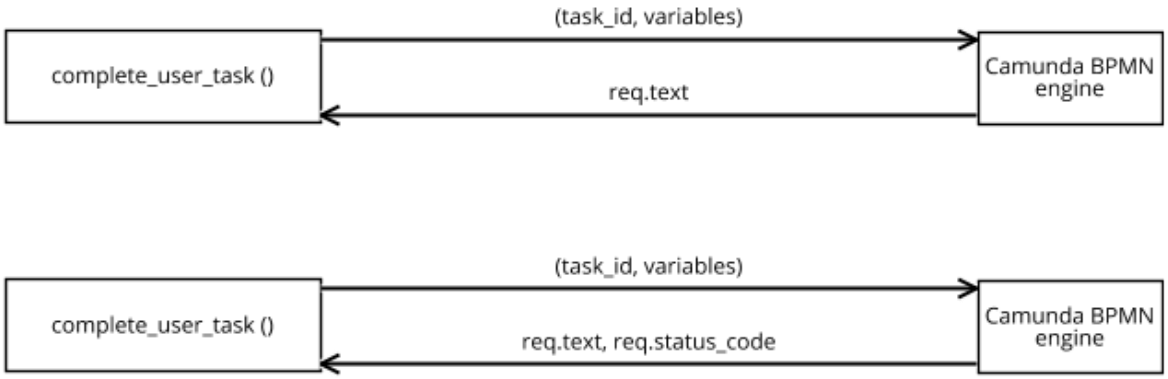
```
def get_user_task(process_definition_id, process_instance_id)
```



Slika 12: API poziv za dohvat ljudskog zadatka

Zahtjev poziva „get_user_task()“ koristi parametre „process_definition_id“ i „process_instance_id“ koji su generirani prilikom pokretanja procesne instance, odnosno odgovor iz BPMN engine je spremljen u kolekciju „chatRooms“ u bazi podataka. Tako da prije samog poziva na servis, pozvana je soba u kojoj se nalaze potrebiti podaci. Poziv se koristi za vraćanje dediciranog zadatka kojeg je potrebno odraditi, a odgovor može i ne mora sadržavati varijable.

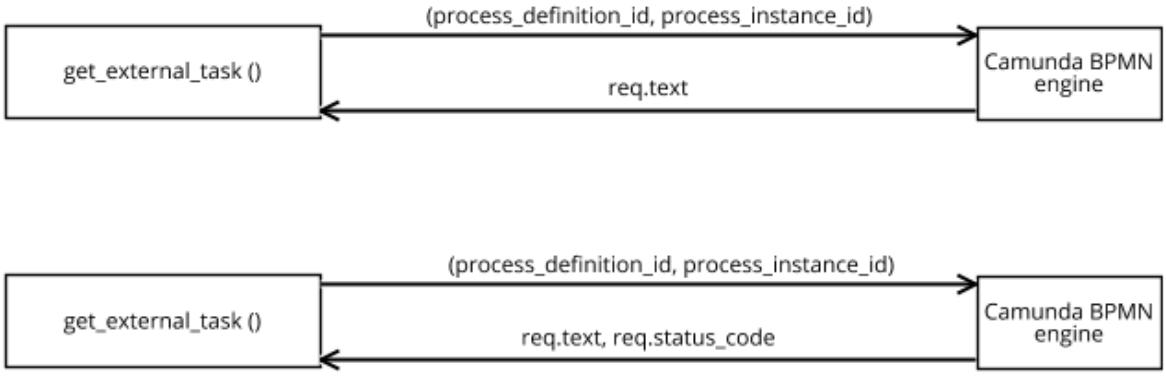
```
def complete_user_task(task_id, variables)
```



Slika 13: API poziv za završetak ljudskog zadatka

Kako bi se dovršio trenutni zadatak ljudskog aktera potrebno je poslati zahtjev na Camundin engine. Parametar „task_id“ se dobavlja prethodno opisanim načinom, a „variables“ može biti i prazan, ali tada se koristi ključna riječ (eng. keyword) „None“ koja je zapravo objekt klase „NoneType“.

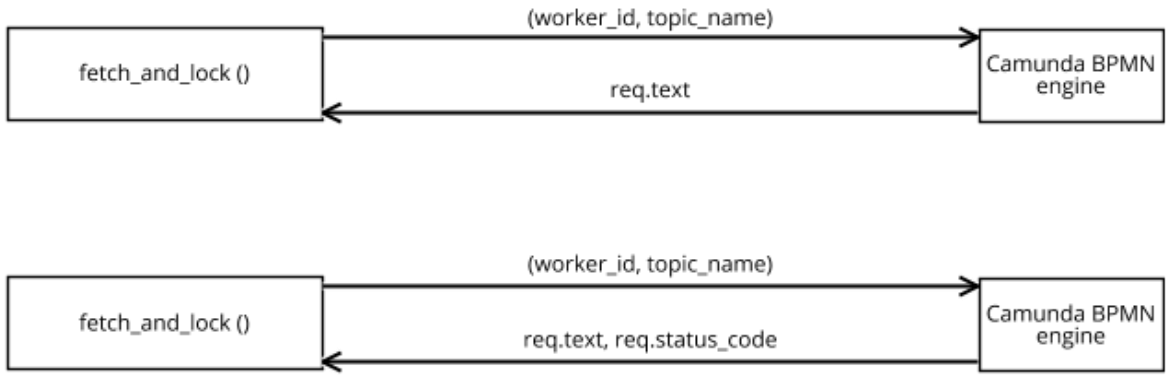
```
def get_external_task(process_definition_id, process_instance_id)
```



Slika 14: API poziv za dohvat servisnog zadatka

„Get_external_task()“ po mnogo čemu sliči na dohvat zadatka s ljudskim akterom, no razlika je u tome što se ovdje dohvaća servisni zadatak.

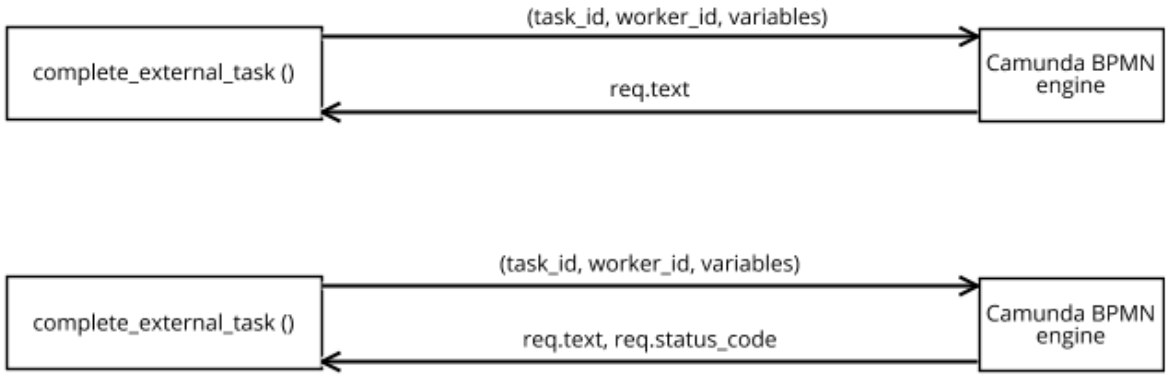
```
def fetch_and_lock(worker_id, topic_name)
```



Slika 15: API poziv za dohvat i zaključavanje servisnog zadatka

„Fetch_and_lock()“ sadrži dva parametra u slanju zahtjeva, a to su „worker_id“ koji identificira radnika, pritom se misli na udaljeno računalo i „topic_name“ koji je vidljiv na slici 4., a označava ime neljudskog zadatka. Važno je napomenuti da se servisni zadatak ne može dovršiti ukoliko nije „dohvaćen i zaključan“ od istog radnika.

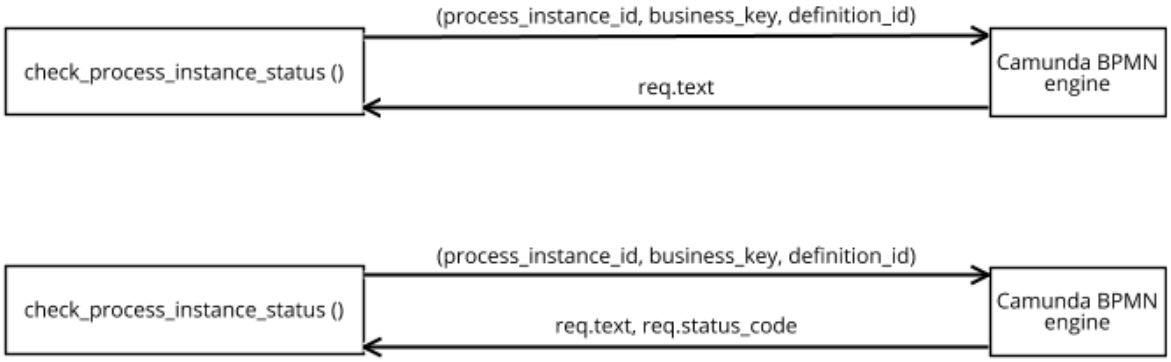
```
def complete_external_task(task_id, worker_id, variables)
```



Slika 16: API poziv za završetak servisnog zadatka

Poziv „complete_external_task()“ sadrži tri parametra od kojih je jedino parametar „variables“ može biti prazan. „Worker_id“ se generira na poslužitelju, a „task_id“ se dohvaća pozivom „get_external_task()“ čiji odgovor sadrži taj parametar.

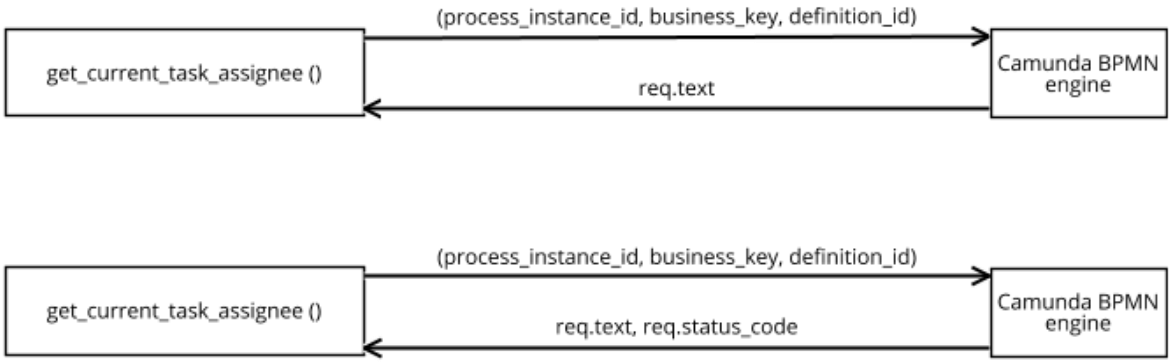
```
def check_process_instance_status(process_instance_id, business_key, definition_id)
```



Slika 17: API poziv za provjeru statusa procesne instance

Poziv „check_process_instance_status()“ provjerava je li procesna instanca završila, budući da je nemoguće znati unaprijed je li trenutni task ujedno i zadnji. Stoga se taj poziv koristi na početku funkcije u dohvatu varijabli u datoteci „bpmnroutes.py“.

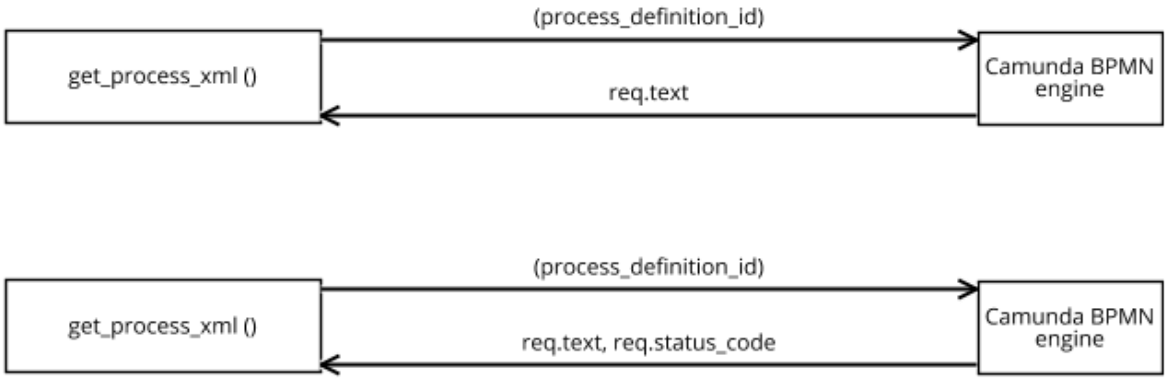
```
def get_current_task_assignee(process_instance_id, business_key, process_definition_id)
```



Slika 18: API poziv za dohvat trenutnog zaduženika

Poziv „get_current_task_assignee ()“ dohvaća ljudskog aktera s trenutnog zadatka. Koristi se na frontendu zbog sinkronizacije u slanju poruka.

```
def get_process_xml(process_definition_id)
```



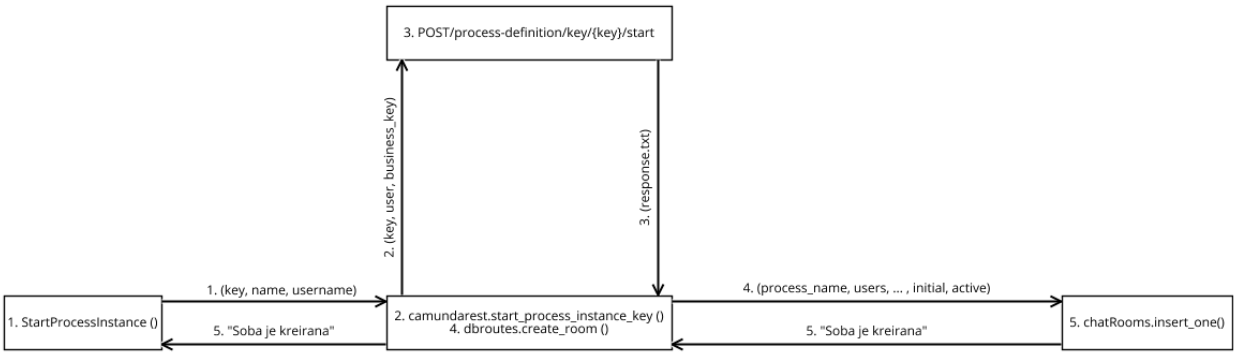
Slika 19: API poziv za dohvat XML-a

Poziv „get_process_xml ()“ dohvaća xml procesa na način i proslijedi ga u „xmlparser.py“ gdje se isti formatira na način koji prigodan za prikazivanje na klijentu.

Datoteka „bpmnroutes.py“ je centralna datoteka u poslužitelju jer poziva sve već navedene datoteke i pruža API za klijenta. Svaki poziv sadrži rutu na koju klijent pristupa, HTTP metodu, „cross_origin()“ funkciju iz modula CORS[29] i naziv same funkcije. Tri funkcije su pokrivene dijagramima: pokretanje procesa, slanje varijabli i na kraju dohvat samih varijabli čiji je dijagram nacrtan pomoću BPMN 2.0 notacije, no moglo ga se i nacrtati usmjerenim grafom.

[29] https://flask-cors.readthedocs.io/en/latest/

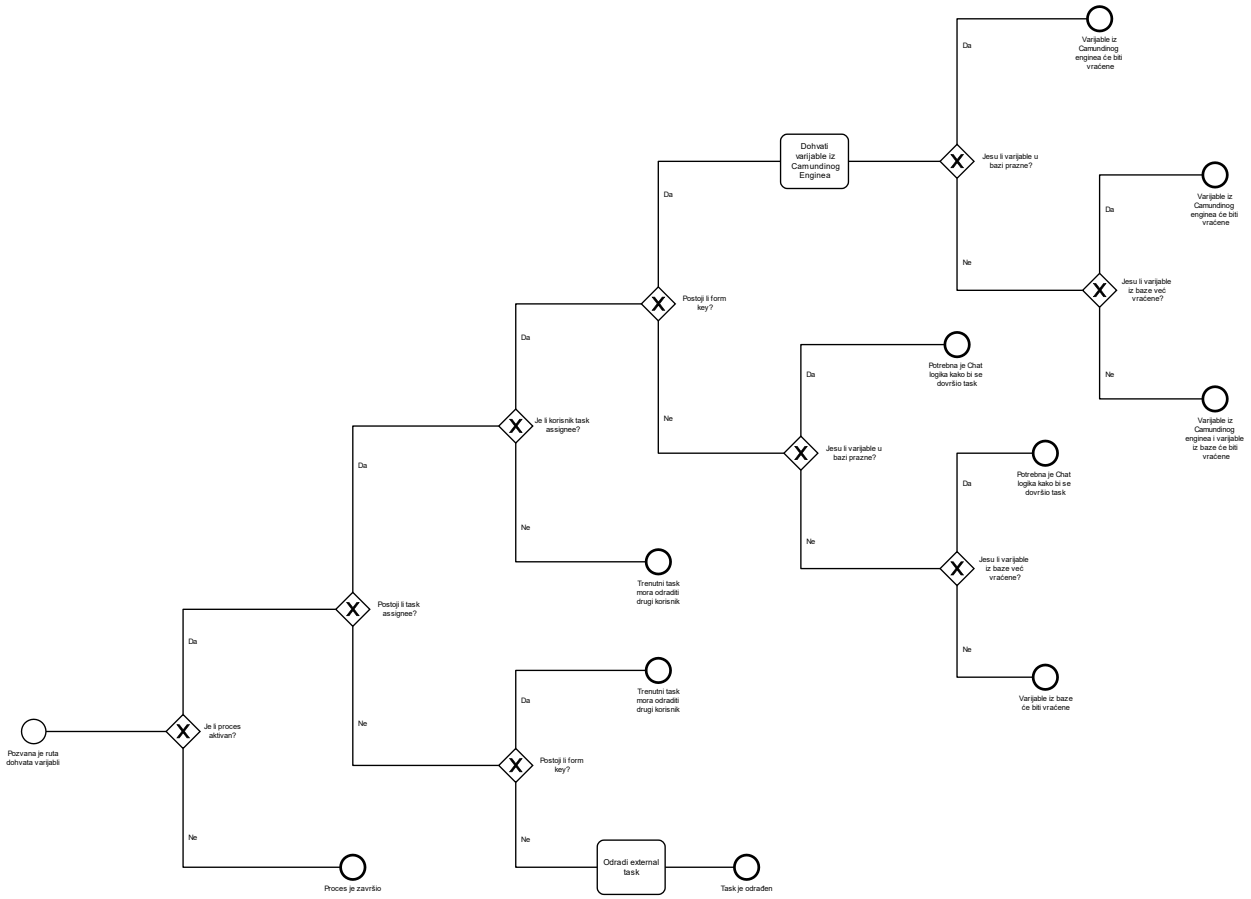
```
@app.route('api/process-instance/<key>, methods= ['POST'])

@cross_origin()

def start_instance(key)
```



Slika 20: Redoslijed izvšavanja poziva i odgovora za pokretanje procesne instance

Proces pokretanja instance kreće s klijenta koji šalje parametre „key“, „name“ i „username“ prosljeđuju se poslužitelju koji zatim generira „business_key“ kako bi se moglo „pozvati“ Camunda BPMN engine. Engine kreira procesnu instancu i vraća podatke na poslužitelja, a poslužitelj te iste podatke prosljeđuje bazi podataka i sprema ih u kolekciju „chatRooms“. Na kraju se vraća odgovor da je soba uspješno kreirana.

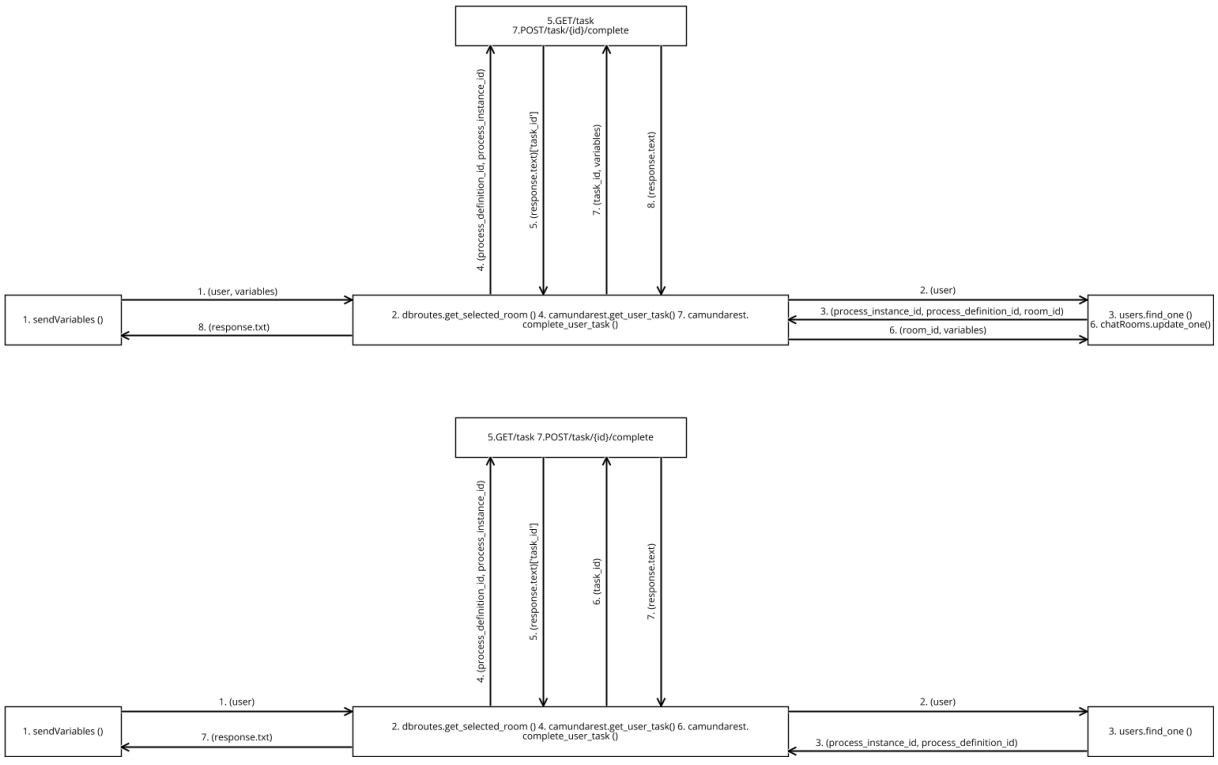
```
@app.route('api/<user>/task/variables, methods= ['GET'])

@cross_origin()

def get_task_variables(user)
```



Slika 21: Dijagram procesa ispitivanja uvjeta za dohvat varijabli na klijent

Proces dohvaćanja varijabli je nimalo trivijalan, budući da ima mnogo rubnih uvjeta. Počinje provjerom je li proces aktivan, ako to nije slučaj, vraća se odgovor da je proces završio. Slijedi pitanje ima li trenutni zadatak zaduženika (eng. Task Assignee), ukoliko nema, sljedeći uvjet je onaj u kojem se pita ima li taj zadatak ključ forme (eng. Form Key). Naime, ukoliko zadatak ne sadrži zadužnika i ključ forme – zadatak je servisnog ili neljudskog karaktera koji se ujedno i kompletira. Ukoliko zadatak sadrži ključ forme u ovom grananju, tada je riječ o slučaju u kojem trenutni zadatak procesa mora odraditi drugi korisnik kako bi proces nastavio sa svojim odvijanjem. Kada postoji zaduženik, slijedi uvjet u kojem se ispituje je li trenutni korisnik – zaduženik, ako nije, zadatak mora odraditi druga osoba. Ukoliko to jest slučaj, slijedi uvjet u kojem se ispituje postoji li ključ forme. Važno je napomenuti da ključ forme signalizira da postoje Camundine varijable koje je potrebno prikazati klijentu. Ako se ustanovi da zadatak ne sadrži ključ forme tada

slijede uvjeti koji ispituju jesu li varijable u bazi prazne, odnosno već vraćene. U navedena dva slučaja varijable se ne parsiraju i šalju na klijenta, u drugom slučaju situacija je analogna. Nego, ako se ustanovi da postoji ključ forme slijedi poziv na Camundin engine kako bi se vratile varijable koje se također parsiraju kako bi bile prigodne za prikazivanje na klijentu. To što je potrebno povući varijable iz Camunde ne znači da je dohvat varijabli gotov. Potrebno je ispitati kakvo je stanje s varijablama u bazi podataka. U slučaju da su varijable iz baze već vraćene ili ne postoje, tada se vraćaju na klijenta isključivo varijable generirane iz engine-a, ako taj uvjet nije zadovoljen – vraćaju se varijable iz baze podataka i iz Camundinog engine-a. Ova ruta je s namjerom glomazna, a namjera je bila omogućiti konverzacijskom agentu jednu i samo jednu rutu na koju se treba pozvati kako bi se varijable njemu i vratile.



Slika 22: Redoslijed izvšavanja poziva i odgovora za završetak zadatka

```
@app.route('api/<user>/task/variables, methods= ['POST'])

@cross_origin()

def send_task_variables(user)
```

Kod slanja varijabli na poslužitelja je također cilj bio napraviti jednu rutu koja je nužna kako bi se zadatak dovršio. Naime, sve kreće sa slanjem varijable „usera“ na poslužitelja koji isti podatak prosljeđuje bazi kako bi dobio odgovor u kojoj se chat sobi korisnik nalazi, budući da se u sobi nalaze svi podaci korisnikovog procesa. Vraćeni podaci se šalju u engine iz razloga što je nužan „task_id“ parametar za uspješni završetak zadatka. Nakon što je dobavljen taj parametar, ovisno o tome postoje li varijable, šalje se poziv na Camundin web servis kako bi se zadatak završio, iako nisu nužne.

```
@app.route('api/<user>/task/assignee, methods= ['GET'])
@cross_origin()
def get_assignee(user)
```

Ova ruta služi za dohvaćanje zaduženika trenutnog zadatka i sadrži velik potencijal u smislu nadogradnje sustava, budući da se tu mogu implementirati takozvane „push“ poruke ili obavijesti koje zaduženika obavještavaju da nastavak procesa ovisi o njemu.

```
@app.route('api/mentors, methods= ['GET'])
@cross_origin()
def get_mentors()
@app.route('/api/<user>/task/form/complete', methods=['GET])
@cross_origin()
def complete_task_form(user)
```

Ove dvije rute su implementirane kako bi se omogućilo dohvaćanje mentora radi odabira mentora u automatski generiranoj formi i sam završetak zadatka nakon što se ispunjena forma pošalje na poslužitelja.

Zaključak

Kako bi ovaj dokaz o konceptu zaživio u stvarnosti, potrebno je implementirati još mnogo funkcionalnosti. Model procesa mora pružati mogućnost podprocesa, pritom se to odnosi da jedna instanca ima više podinstanci, primjerice, u prijavi završnog rada, akter student bi mogao komunicirati s više potencijalnih mentora odjednom. Potrebno je nadograditi programsko sučelje do te razine da se mogu modelirati i složeniji dijagrami. Oponašajuće servisne zadatke zamijeniti onima koji izvode stvarne web servise, kao što su već spomenute „push“ obavijesti. Što se tiče chatbota, treba implementirati sustav u realnom vremenu. U konačnici, cijeli sustav bi se mogao i modularizirati s ciljem da se od gotovih modula, odnosno komponenti može složiti novi sustavi napravljeni spajanjem tih gotovih cjelina.

Literatura

Brumec, Josip i Slaven Brumec. "Modeliranje poslovnih procesa." Zagreb, Školska knjiga (2018).

https://www.omg.org/bpmn/

https://camunda.com/

https://camunda.com/products/camunda-bpm/modeler/

https://docs.camunda.org/manual/latest/webapps/tasklist/

https://docs.camunda.org/manual/latest/webapps/admin/

https://docs.camunda.org/manual/latest/webapps/cockpit/

https://github.com/vue-generators/vue-form-generator

https://github.com/antoine92190/vue-advanced-chat

https://www.python.org/

https://www.java.com/en/

https://docs.camunda.org/manual/latest/user-guide/process-engine/process-engine-concepts/

https://docs.camunda.org/manual/7.13/reference/rest/process-definition/

https://docs.camunda.org/manual/7.13/reference/rest/process-instance/

https://docs.camunda.org/manual/7.13/reference/rest/task/

https://docs.camunda.org/manual/7.13/reference/rest/external-task/

https://docs.camunda.org/manual/latest/user-guide/task-forms/

https://vuejs.org/

https://developer.oracle.com/javascript/

https://vuetifyjs.com/en/

https://getbootstrap.com/

https://github.com/axios/axios

https://flask.palletsprojects.com/en/1.1.x/

https://requests.readthedocs.io/en/master/

https://www.mongodb.com/

https://www.w3.org/XML/

https://docs.python.org/3.9/library/xml.etree.elementtree.html

https://pymongo.readthedocs.io/en/stable/

https://flask-cors.readthedocs.io/en/latest/