Obrada teksta kod Chatbot sustava vođenog modelom

Bernobić, Nikki

Sveučilište Jurja Dobrile u Puli

Fakultet informatike

NIKKI BERNOBIĆ

OBRADA TEKSTA KOD CHATBOT SUSTAVA VOĐENOG MODELOM

(Text processing in a model-driven Chatbot system)

Završni rad

Pula, rujan, 2020. godine

Sveučilište Jurja Dobrile u Puli

Fakultet informatike

NIKKI BERNOBIĆ

OBRADA TEKSTA KOD CHATBOT SUSTAVA VOĐENOG MODELOM

(Text processing in a model-driven Chatbot system)

Završni rad

JMBAG: 0303075390

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 potpisana Nikki Bernobić, 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.

Studentica

U Puli, rujan, 2020. godine

IZJAVA O KORIŠTENJU AUTORSKOG DIJELA

Ja, Nikki Bernobić, dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom *Obrada teksta kod chatbot sustava vođenog modelom* 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.

Studentica

U Puli, rujan, 2020. godine

SADRŽAJ

SAŽETAK

Chatbot sustavi postoje tek oko pedesetak godina no sve su popularniji zbog njihove široke primjene – danas ih se može vidjeti gotovo u svakom području. U svakodnevnom životu viđamo ih kao Alexa, Google, Siri ili kao jednostavnije chatbotove koji se pojavljuju na web stranicama te služe za odgovaranje na često postavljana pitanja ili za pomoć pri kupnji. Tvrtke koriste ovakve sustave kako bi smanjili izdatke budući da jedan dobro izrađen chatbot nema radno vrijeme, efikasan je i drastično smanjuje upite korisnika. Pitanje koje si postavljamo jest zašto ograničiti korisnost chatbotova samo na odgovaranje upita korisnika?

UVOD

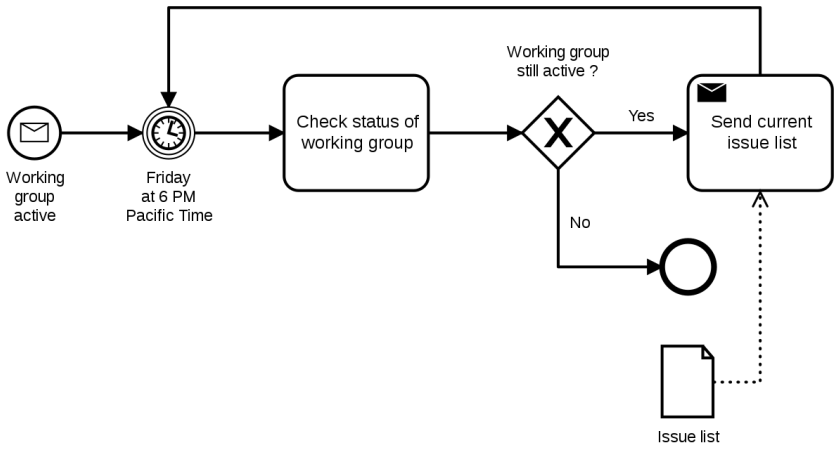
Chatbot sustavi su dijaloški sustavi koji simuliraju konverzaciju s korisnikom, a namijenjeni su za komunikaciju s ljudima korištenjem prirodnog jezika. Chatbot mora biti sposoban za dijalog i razumijevanje korisnika. U današnje vrijeme sve je popularnije korištenje umjetne inteligencije za to, budući da ona može služiti kod razumijevanja primljenog teksta te stvaranju odgovora. No, umjetna inteligencija nije preduvjet za stvaranje dobrog chatbota. Pametno osmišljen deterministički chatbot može biti jednako koristan kao i chatbot koji koristi umjetnu inteligenciju. Kada kažemo deterministički chatbot, mislimo na chatbot koji je unaprijed osmišljen za odgovaranje na upite korištenjem nekih okidača ili unaprijed pripremljenih skripti, korištenjem principa prepoznavanja uzoraka kod unosa. Za stvaranje chatbotova danas se mogu koristiti već gotovi proizvodi kojih je potrebno prilagoditi slučaju uporabe, često i bez potrebe programiranja, na primjer Botsociety[1]. Za veće projekte u mnogo slučajeva koriste se i servisi razumijevanja prirodnog jezika (NLU), kao što su Watson Natural Language Understanding[2] ili Rasa NLU[3]. Ukoliko je potrebna još veća prilagodljivost, moguće je kreirati svoju komponentu razumijevanja.

[1] Botsociety, https://botsociety.io/chatbot
[2] Watson Natural Language Understanding, https://www.ibm.com/cloud/watson-natural-language-understanding
[3] Rasa, https://rasa.com/docs/

Slika 1. Primjer chatbot sustava (https://www.fiuman.hr/andrija-na-whatsappu-ce-vam-pomoci-napraviti-samodijagnozu-za-koronavirus/)

Fibot je chatbot sustav vođen modelom što znači da on funkcionira kao chatbot koji je vođen poslovnim procesom. Poslovni proces je aktivnost ili skup aktivnosti kojim se izvršava specifični cilj organizacije.[4] Pravilnim izvršavanjem poslovnih procesa povećavamo organizaciju, smanjujemo mogućnost pogreške u izvođenju procesa i povećavamo efikasnost samog procesa. Grafički prikaz poslovnog procesa naziva se modelom. Razvijena je i notacija za modeliranje poslovnih procesa pod nazivom BPMN 2.0 koja je danas utvrđena kao standard.

[4] Blaće, Dubravko: Zašto je važno razumjeti poslovne procese i upravljati njima?, 2015. https://www.evision.hr/hr/Novosti/Stranice/zasto-razumjeti-poslovne-procese-upravljati-procesima.aspx

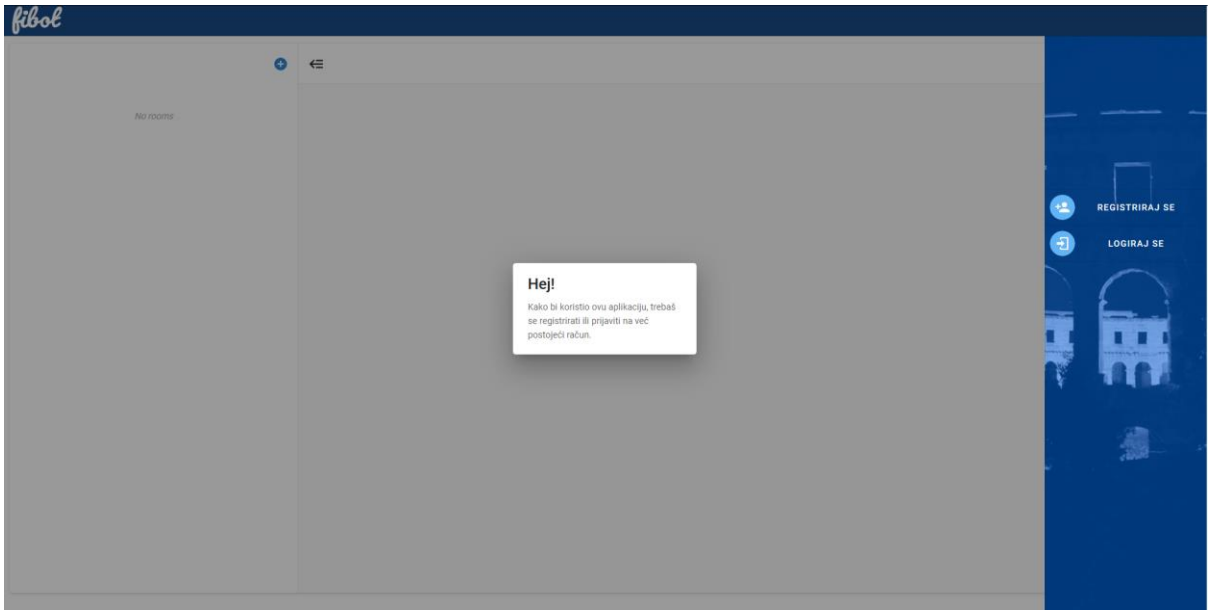


Slika 2. Primjer BPM notacije za proces (https://en.wikipedia.org/wiki/Business_Process_Model_and_Notation)

Ideja je imati chatbot koji će biti sučelje raznim poslovnim procesima te time omogućiti prolaženje kroz kompleksan proces na jednostavan, *user-friendly*, način. Cilj je da to bude što jednostavnije korisnicima, dok menadžerima omogućuje praćenje i analiziranje tijeka procesa. Za sada Fibot nije inteligentan, sam po sebi nema razumijevanja za hrvatski jezik budući da su mogućnosti procesiranja hrvatskog jezika još uvijek tehnološki limitirane. Stoga, koristimo prepoznavanje uzoraka kako bi chatbot prezentirao poslovni proces korisniku i olakšao mu prolaženje kroz njega.

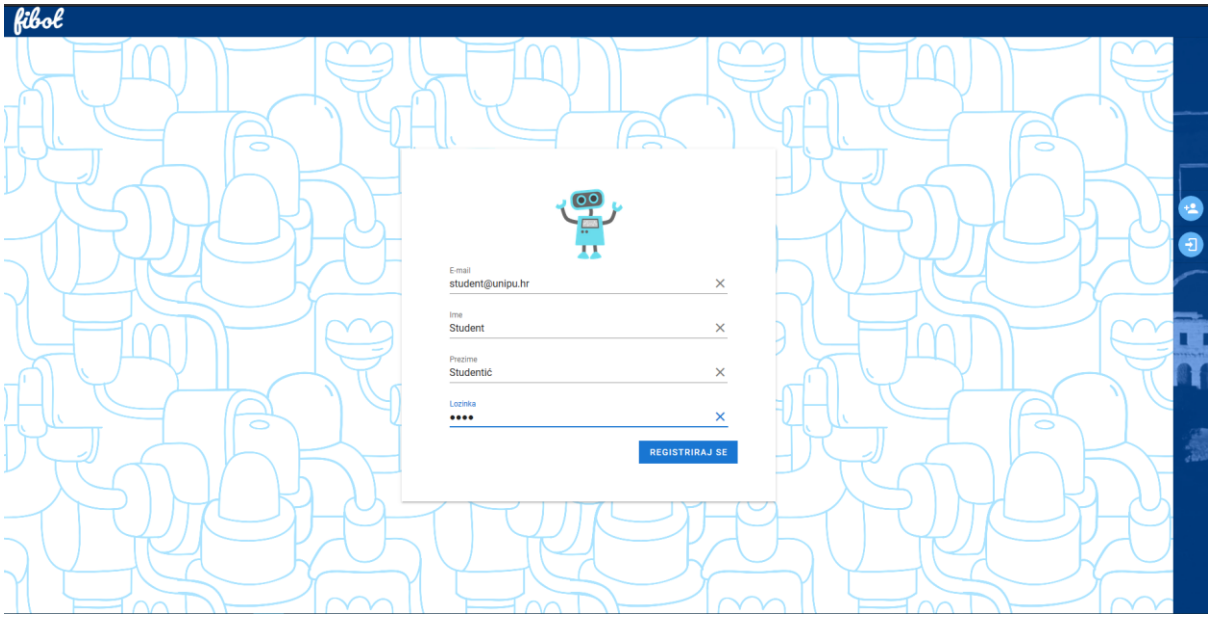
PRIKAZ FUNKCIONALNOSTI

REGISTRACIJA I AUTENTIKACIJA



Slika 3. Prikaz sučelja neregistriranog korisnika

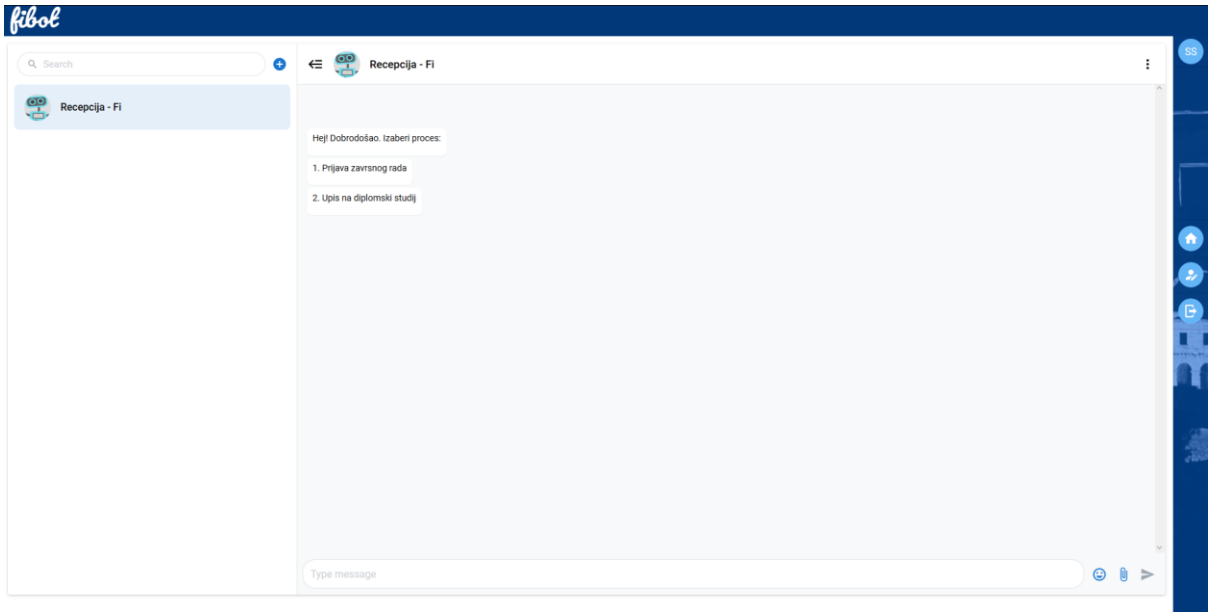
Pri dolasku na početnu stranicu aplikacije, ukoliko korisnik nije prijavljen ili registriran, ne može ju koristiti. Potrebno je registrirati se – trenutno je registracija predviđena za studente.



Slika 4. Stranica registracije

Nakon uspješne registracije, korisnik je prebačen na chat.

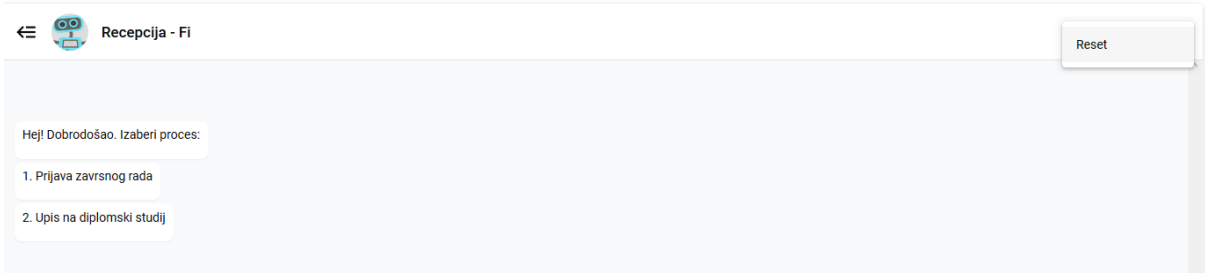
SUČELJE



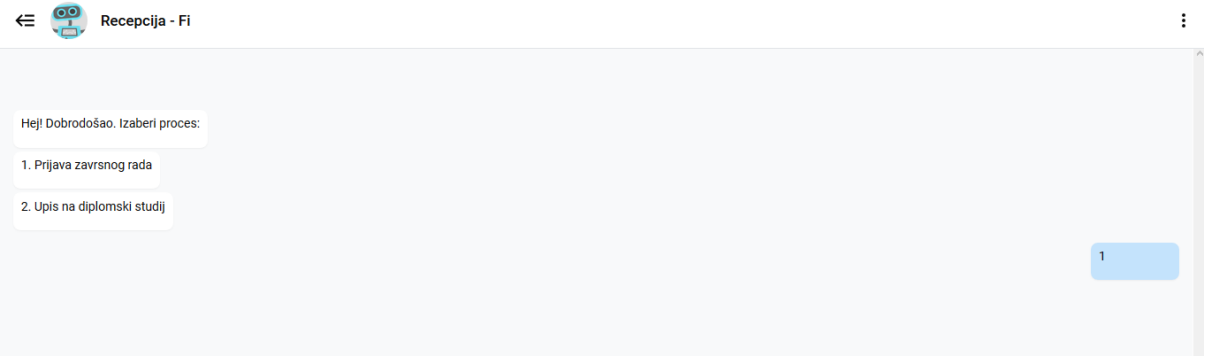
Slika 5. Recepcija

Korisnik se nalazi u sobi Recepcija s Fijem, našim chatbotom. Ovdje korisnik može pokrenuti jedan od navedenih procesa.

U bilo kojem trenutku korisnik može resetirati proces i poruke, ukoliko je to potrebno.

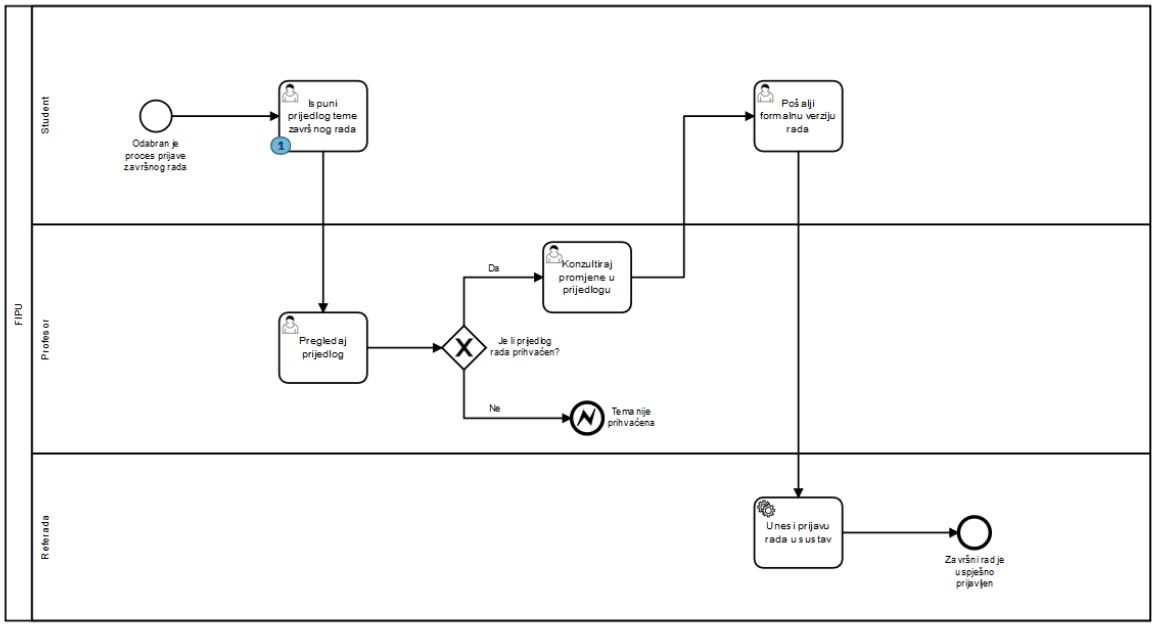


Slika 6. Prikaz ponuđenih procesa



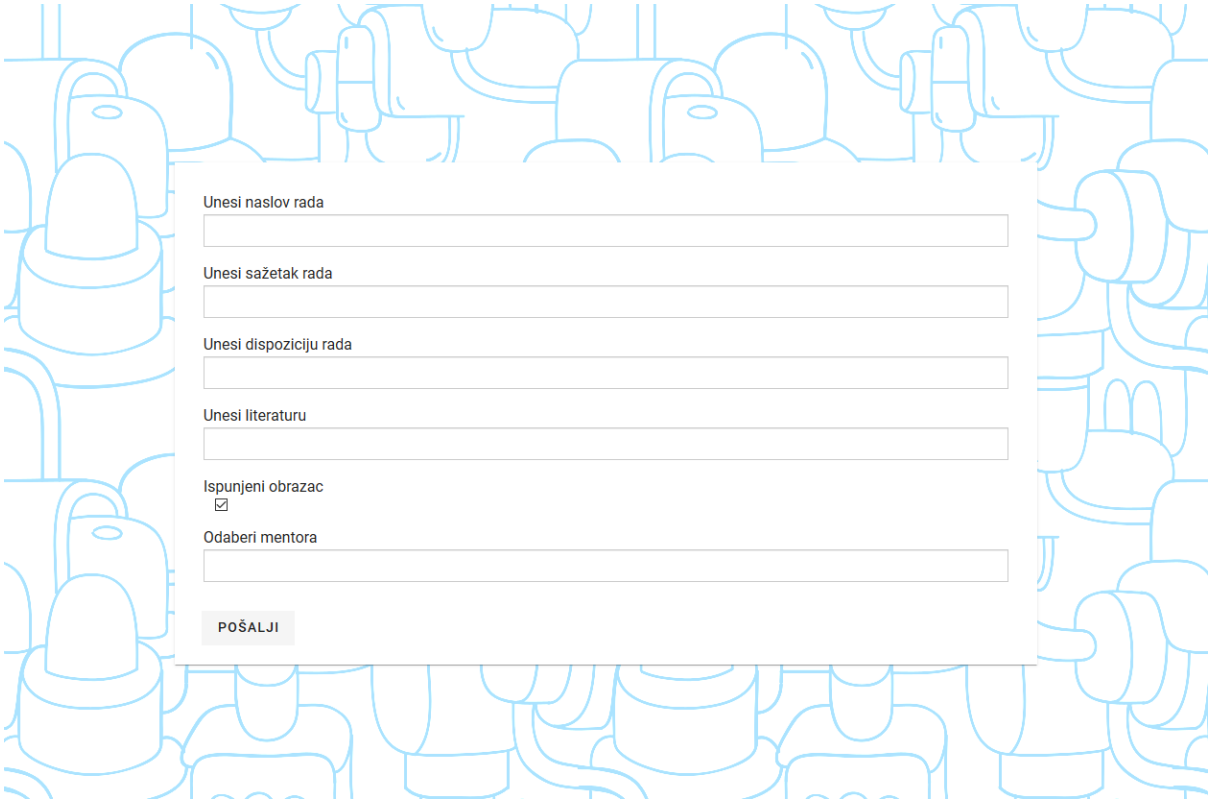
Slika 7. Odabir procesa 1

Korisnik upisuje „1“ i time započinje proces prijave završnoga rada. Time započinje BPMN proces u pozadini te proces traje dok se ne izvrše svi zadaci. Proces izgleda ovako:



Slika 8. Model procesa prijave završnog

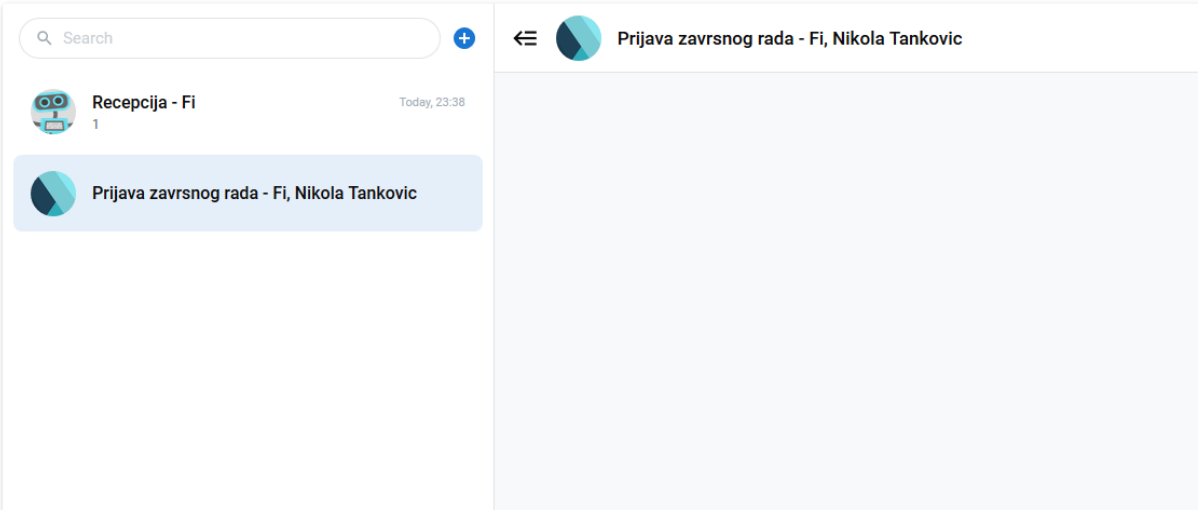
Web aplikacija nas tada traži da ispunimo podatke o prijavi teme završnoga rada.



Slika 9. Generirana forma za upis podataka

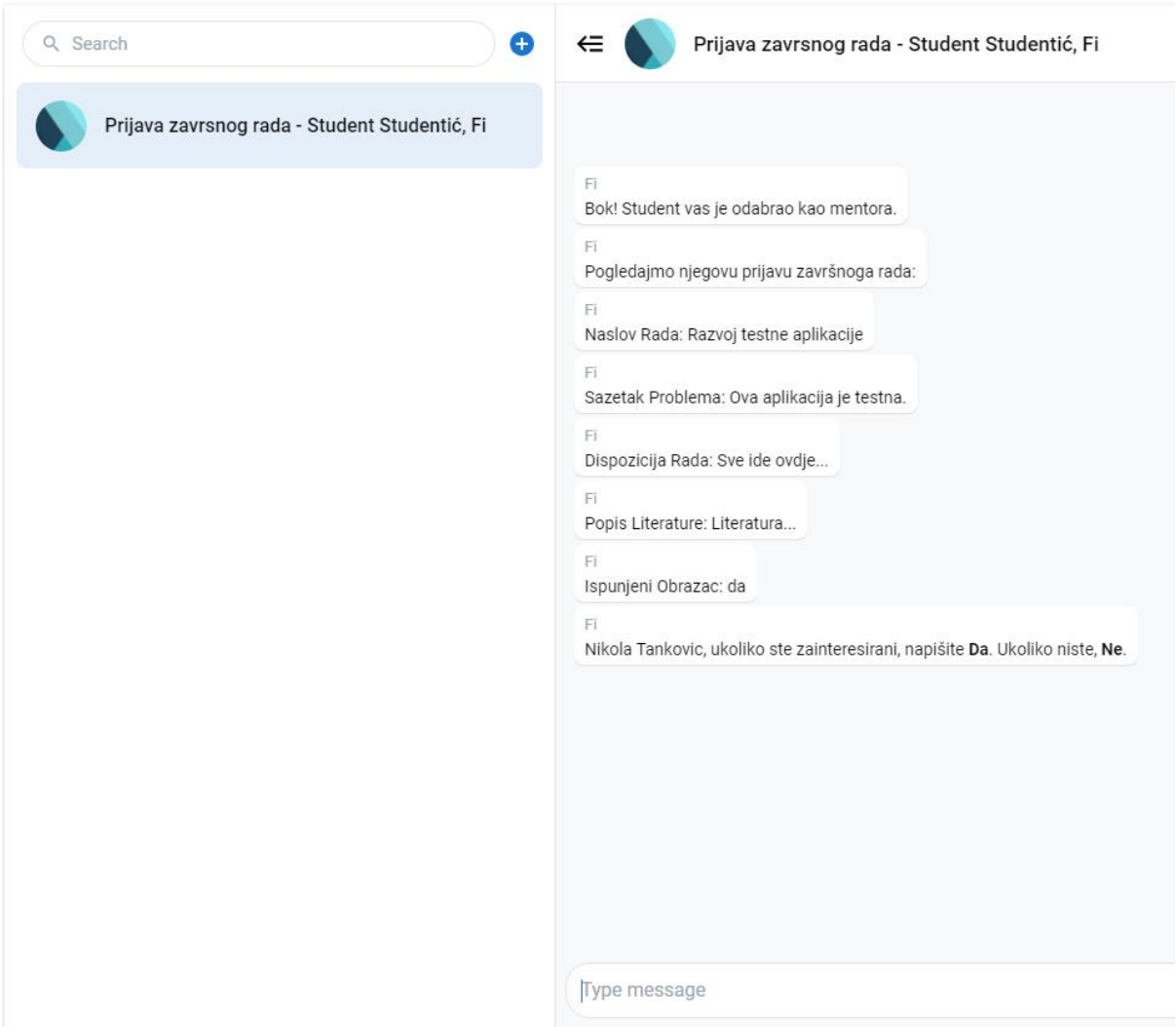
Slika 10. Upisani podaci u formu

Nakon što upišemo podatke, otvara se soba s odabranim mentorom. Ne vidimo sadržaj sobe sve dok se mentor ne prijavi u aplikaciju i pogleda.



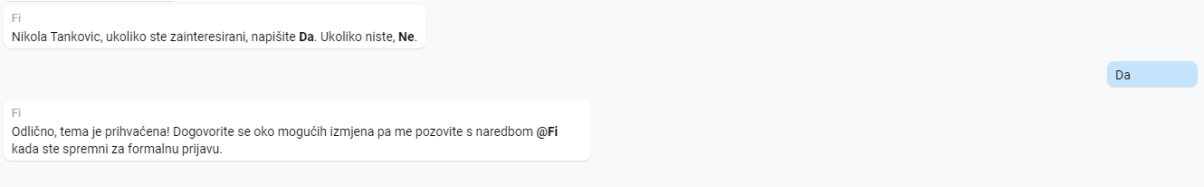
Slika 11. Prikaz procesne sobe korisniku

Kada mentor otvori sobu, prikazani su podaci prijave i njemu i studentu te mora odlučiti da li je zainteresiran ili ne.

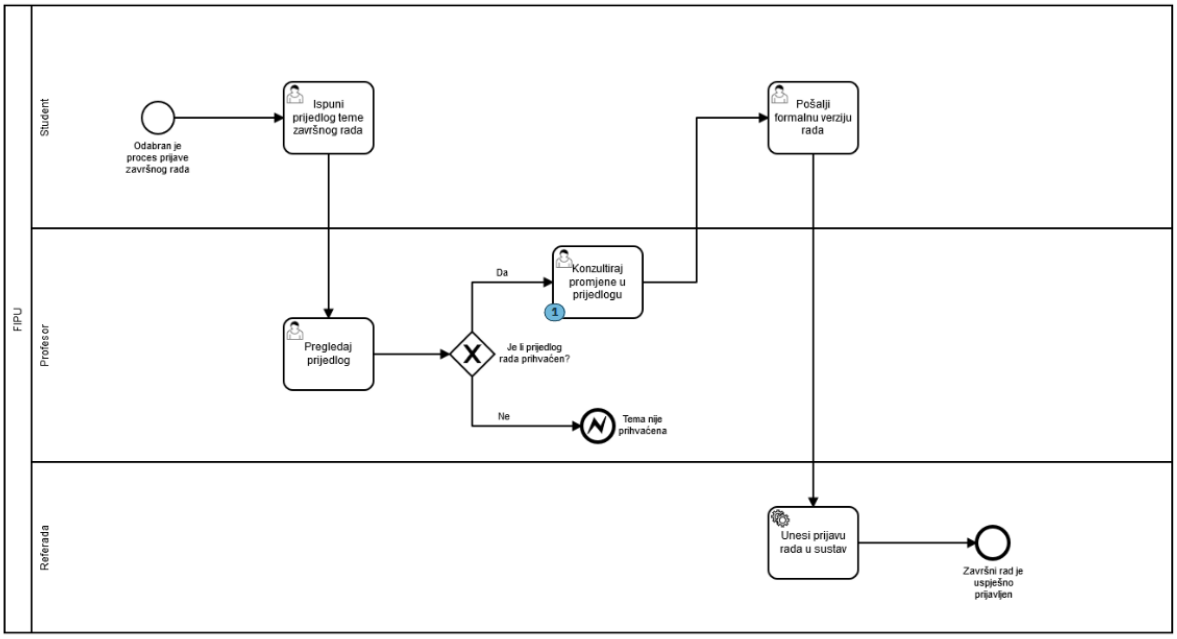


Slika 12. Prikaz procesne sobe mentoru

Proces završava ukoliko mentor nije zainteresiran. Ukoliko je, proces ide dalje.

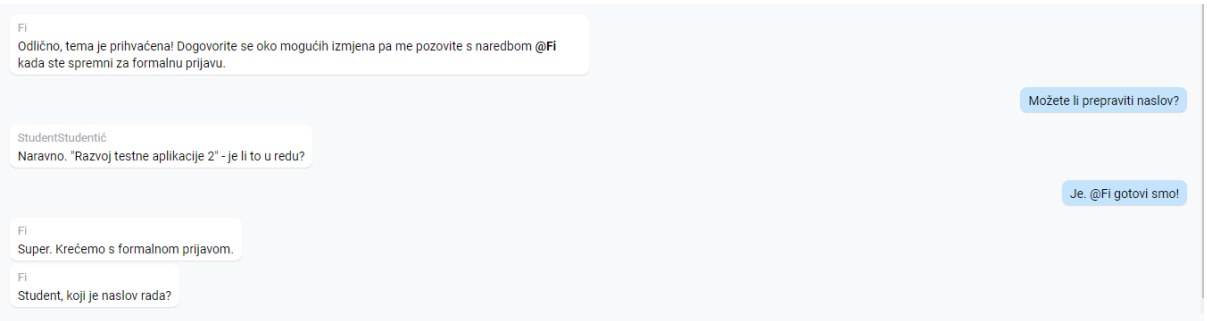


Slika 13. Mentor odlučuje da je zainteresiran



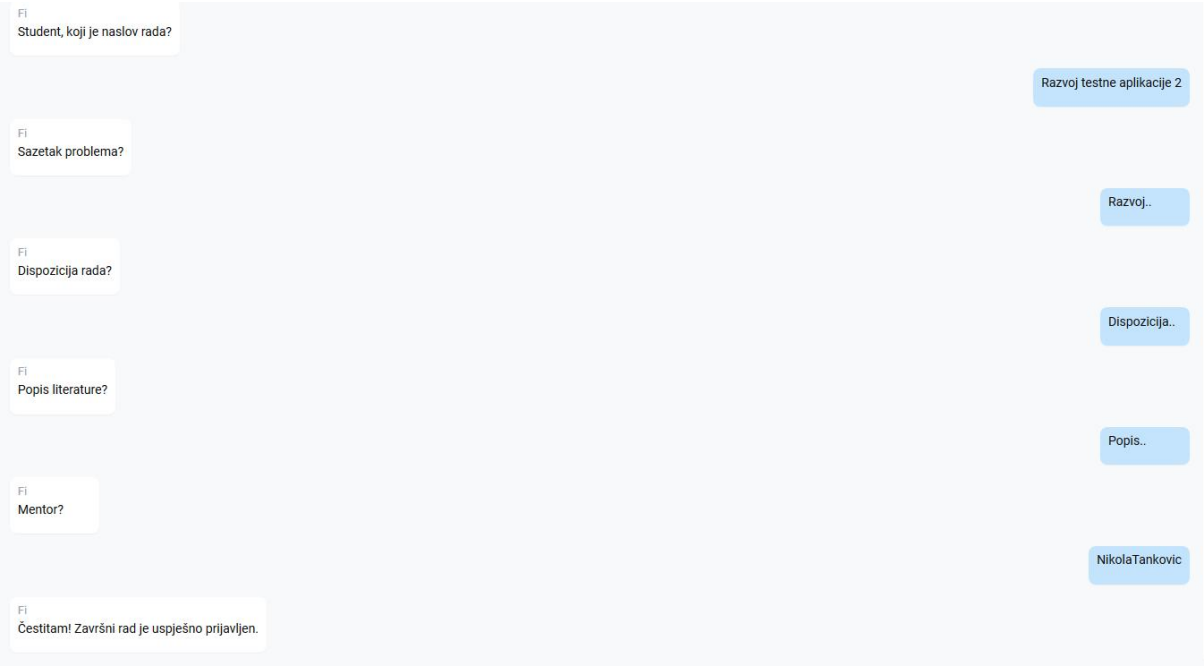
Slika 14. Model procesa trenutno

Sada se mentor i student trebaju dogovoriti oko eventualnih promjena u prijavi i kada su gotovi pozvati Fija.



Slika 15. Dogovor oko promjena

Tada student uz pomoć Fija unosi izmijenjene podatke.

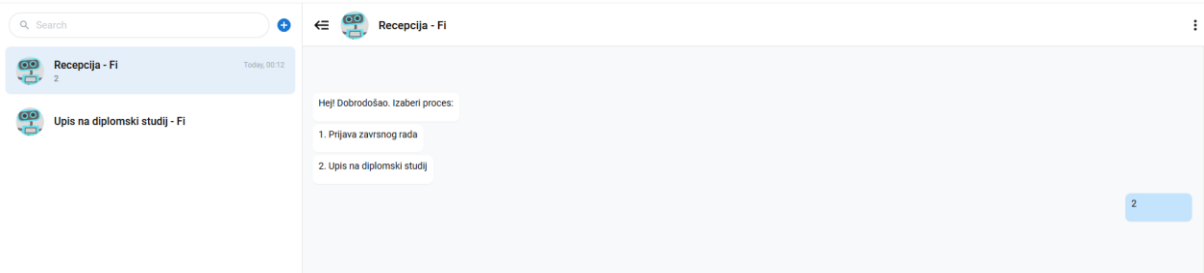


Slika 16. Unos formalne prijave

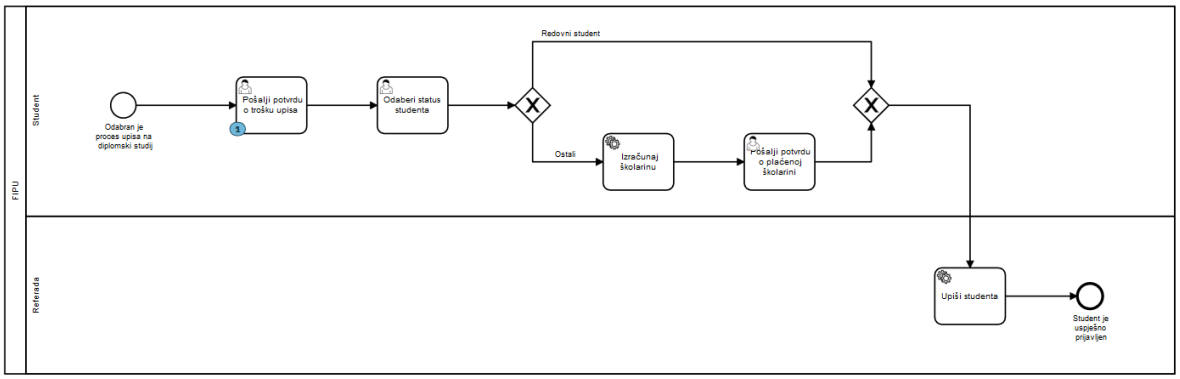
Student je unio konačne podatke prijave te proces uspješno završava i njegova prijava se sprema u bazu podataka.

PROCES UPISA NA DIPLOMSKI STUDIJ VOĐEN CHATBOTOM

Drugi proces je upis na diplomski. Njega pokrećemo upisom „2“ u Recepciju. Otvara se nova soba s chatbotom i pokreće se BPMN instanca procesa.

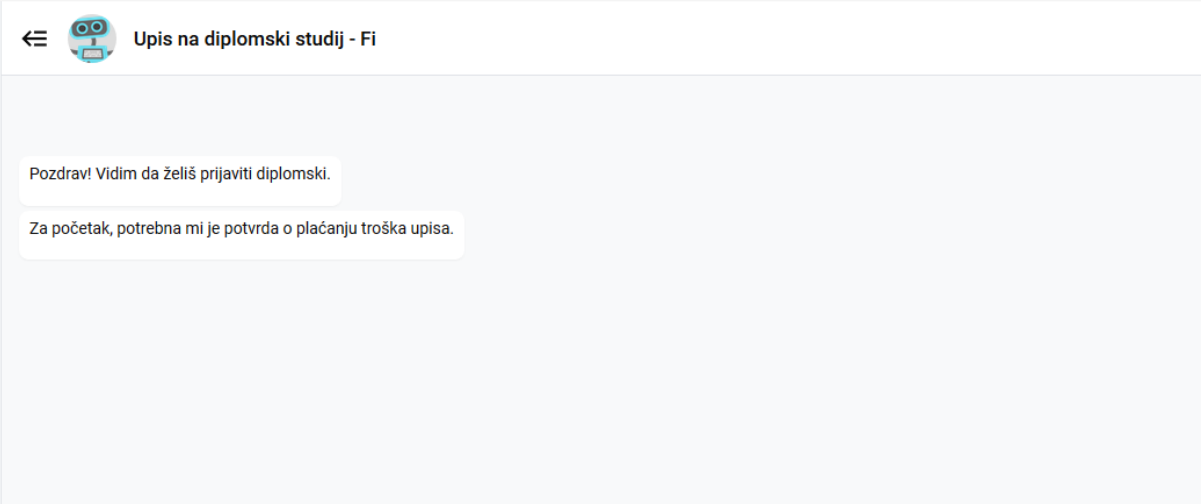


Slika 17. Pokretanje procesa 2



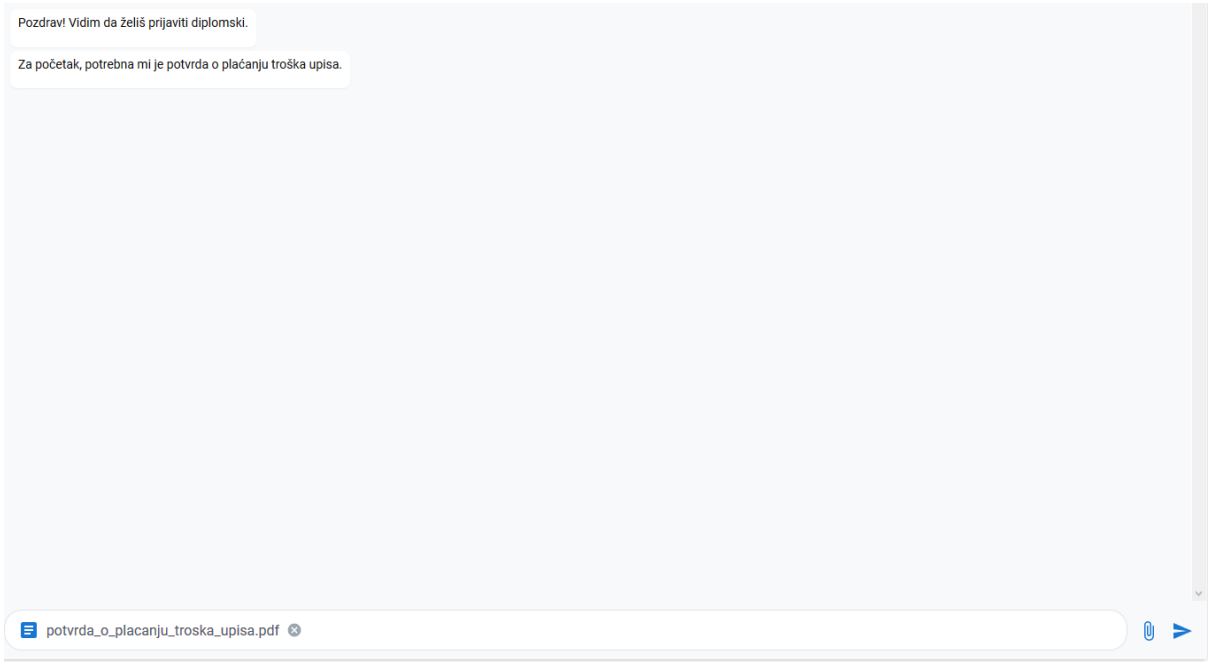
Slika 18. Model procesa upisa diplomskog studija

Fi od nas želi potvrdu plaćanja troška upisa, a to je prvi zadatak u našem procesu.



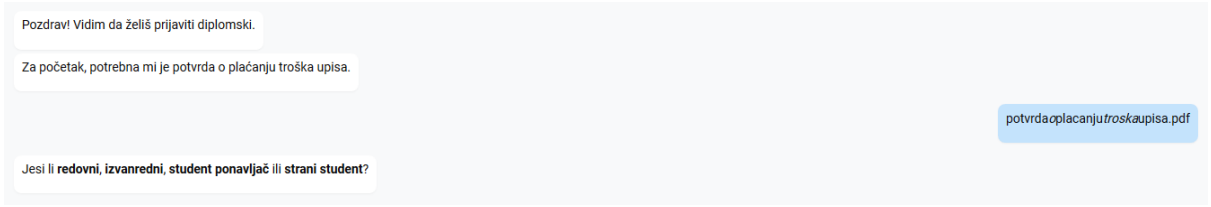
Slika 19. Početak procesa

Kada mu to pošaljemo, proces može ići dalje.

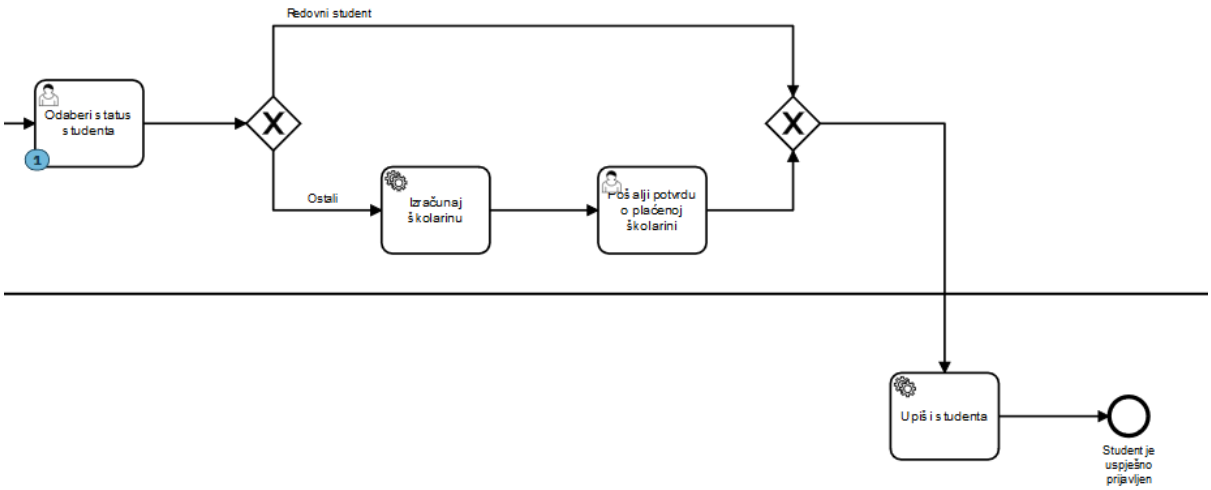


Slika 20. Slanje datoteke

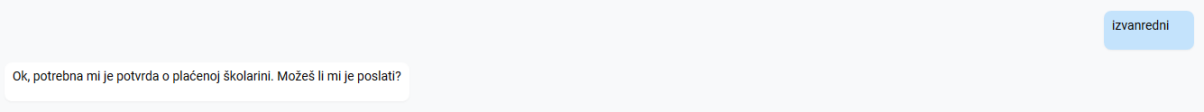
Tada trebamo reći chatbotu kakav status studenta imamo. Redovni studenti ne plaćaju školarinu, dok izvanredni, studenti ponavljači i strani studenti plaćaju.



Slika 21. Upit o statusu studenta

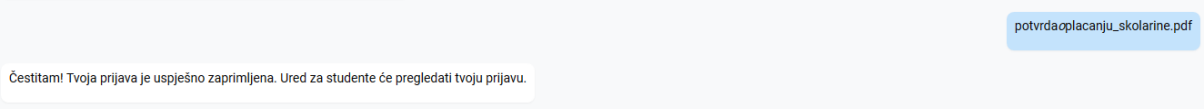


Slika 22. Tijek procesa



Slika 23. Potrebna potvrda o plaćenoj školarini

Budući da svi osim redovnih studenata trebaju platiti školarinu, trebamo poslati potvrdu da smo platili školarinu.



Slika 24. Uspješno odvijen proces

Proces je uspješno završio te su podaci spremljeni u bazu jer smo obavili sve korake koji su bili potrebni.

PROGRAMSKO RJEŠENJE I IMPLEMENTACIJA

KORIŠTENE TEHNOLOGIJE

Aplikaciju dijelimo na dva dijela, *front end* i *back end*. Front end čine Vue.js s Vuetifyom, Vuex, Axios i komponentama vue-advanced-chat[5] i vue-form-generator[6]. Back end čini Flask s Flask-CORS, Flask-JWT-Extended, Flask-Bcyrpt i PyMongom za MongoDB koju koristimo za spremanje podataka. Uz front end i back end, koristimo i Camundin Process Engine.

Vue.js je open-source *model-view-viewmodel* JavaScript *framework* koji omogućuje izradu korisničkih sučelja i SPA. Vuetify koristimo s Vue.js kako bismo imali konzistentan izgled sučelja i komponente kao što su navbar, draweri i slično za njega. Vuex nam služi za upravljanje stanjem, a funkcionira kao centralizirano spremište za sve komponente u aplikaciji. Pomoću Axiosa radimo pozive na back end, budući da je on HTTP klijent. Vue-advanced-chat je komponenta koja nam omogućuje lijep i konzistentan prikaz chat soba, poruka, korisnika i daje nam prostora za našu logiku i potrebne modifikacije za naš slučaj uporabe.

[5] Vue-advanced-chat, https://github.com/antoine92190/vue-advanced-chat
[6] Vue-form-generator, https://github.com/vue-generators/vue-form-generator

Flask je *web framework*, tj. Python modul koji omogućuje jednostavno kreiranje web aplikacija. Omogućuje pokretanje Python koda na serveru jer je *WSGI (Web Server Gateway Service)*[7]. Flask koristimo kao sponu između process engine-a, baze, back end logike i front enda. Flask-CORS koristimo kako bismo neometano slali pozive s front enda na back end. JWT i Bcrypt nam služe za sigurnu autentifikaciju korisnika.

MongoDB je NoSQL open-source baza podataka, što znači da umjesto običnih redaka i stupaca koristi dokumente i kolekcije za organizaciju podataka. Osnovna jedinica podataka sastoji se od skupa key-valuea. Komuniciramo s bazom pomoću modula PyMongo koji omogućuje jednostavno slanje upita na bazu.

U back endu je razvijen API s kojim komuniciramo s process engineom kako bismo pratili tijek procesne instance za pojedinog korisnika. Ovu temu u cijelosti obrađuje kolega Starčić u svom radu *Razvoj programskog sučelja za konverzacijskog agenta vođenog modelom procesa*[8].

ARHITEKTURA

INICIJALNI SETUP

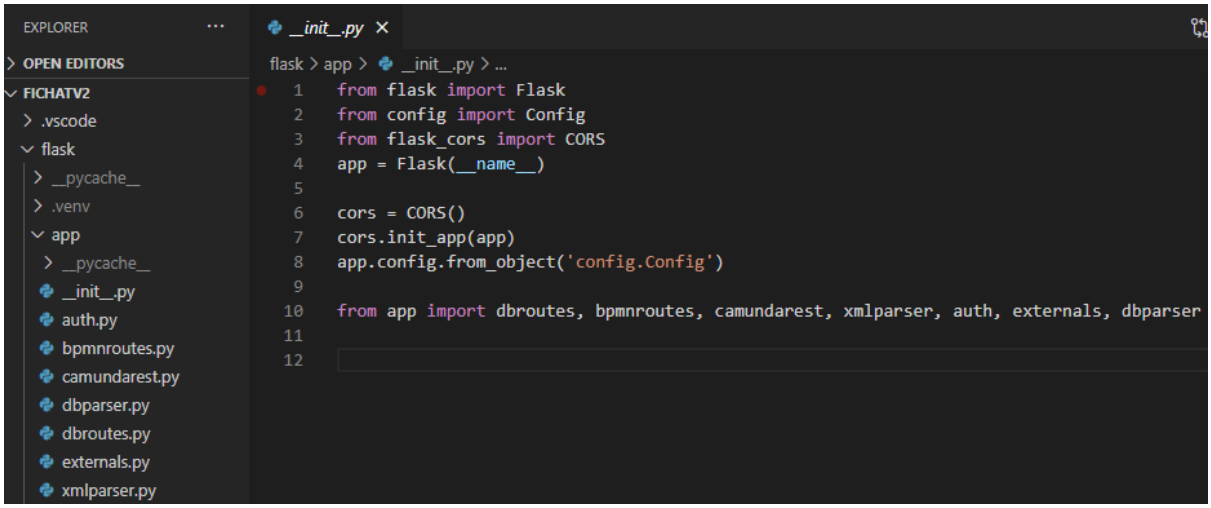
Postoje razni načini konfiguracije projekta, mi smo koristili strukturu jednog repozitorija gdje su front end i back end odvojeni budući da nismo htjeli koristiti Flaskove templateove za sučelje već Vue.js, gdje je funkcionalnost dinamičke web aplikacije u potpunosti sačuvana.

Flask radi na portu 5000, a front end na 8081 stoga za njega definiramo proxy u *package.json* kako bi dvije cjeline bile povezane. U scripts dodali smo *start-flask* naredbu kako bi jednostavno pokretali flask.

[7] What is WSGI?, https://wsgi.readthedocs.io/en/latest/what.html
[8] Starčić, Toni: Razvoj programskog sučelja za konverzacijskog agenta vođenog modelom procesa, 2020.

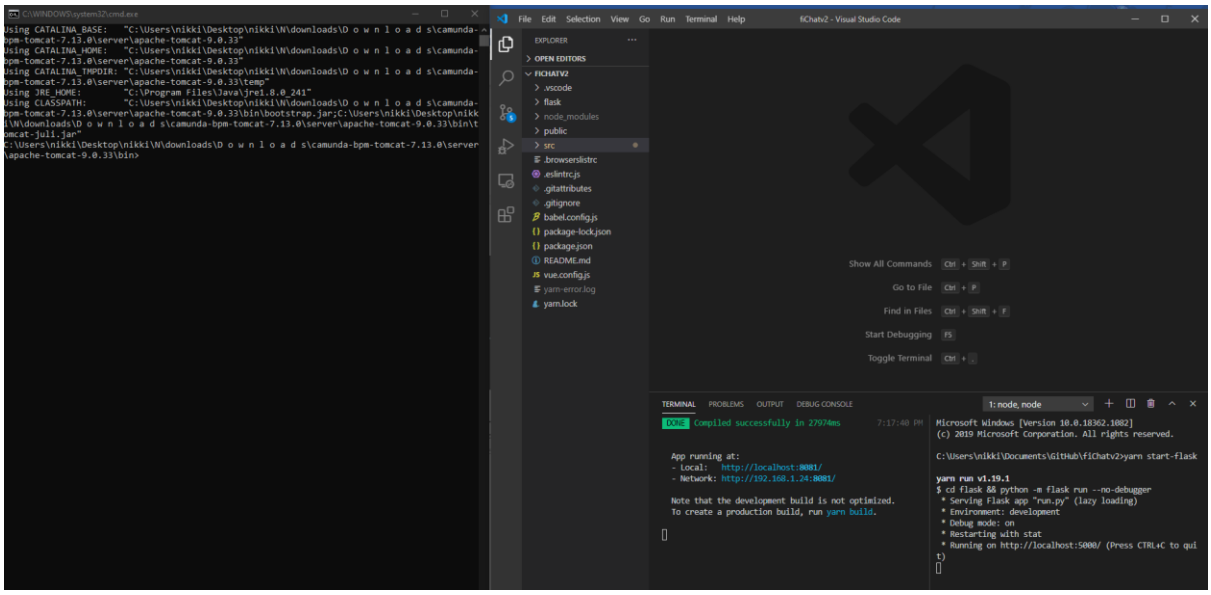
Slika 25. Sadržaj package.json

Flask se sastoji od *run.py, config.py, .flaskenv, .env* i *app* direktorija. App direktorij u sebi sadrži svu logiku back enda, a *__init__.py* datoteka poziva konfiguraciju *config.py* te ostale datoteke koje su dio tog direktorija. Osjetljive varijable stavljamo u *.env*, a *.flaskenv* varijable koristi flask prilikom pokretanja. *Run.py* sve to poziva.

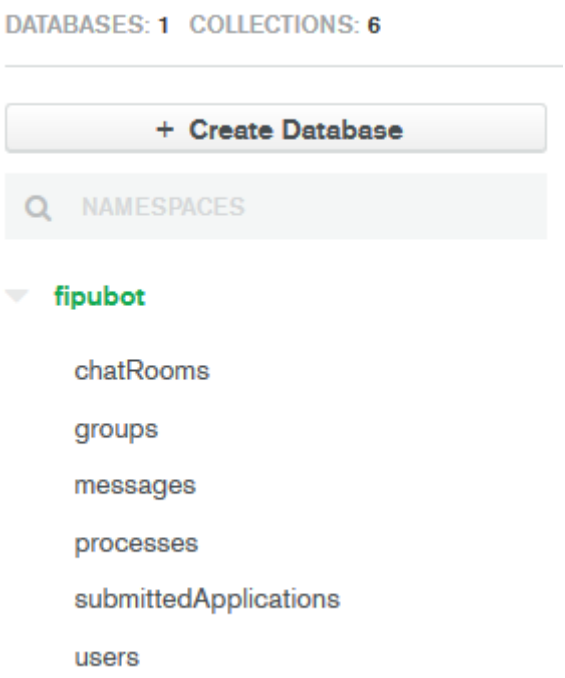


Slika 26. Sadržaj __init__.py

Vue.js se podešava koristeći osnovne postavke, s time da koristimo *vue-router* za olakšanu navigaciju. Za pokretanje aplikacije potrebno je prvo pokrenuti *camunda-bpm-tomcat* server, zatim u jednom terminalu pokrenuti *yarn-serve*, a u drugom *yarn start-flask*.

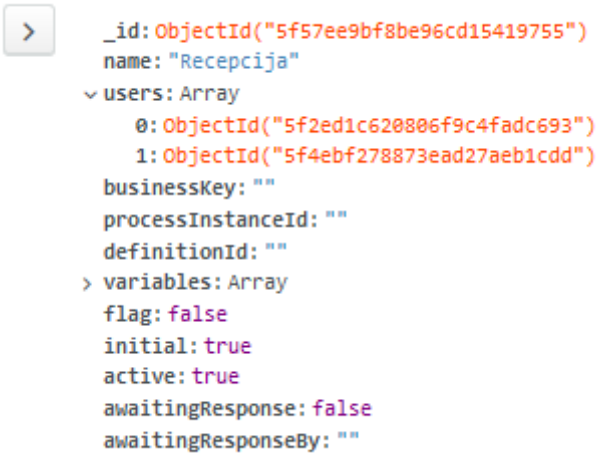


Slika 27. Prikaz pokrenute aplikacije



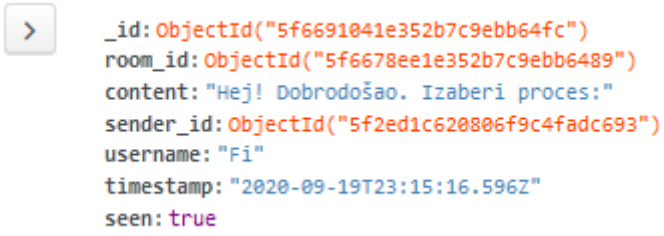
Slika 28. Kolekcije u MongoDB

Baza se sastoji od šest kolekcija. *chatRooms* sadrži recepcije i instance procesa. Recepcije su namijenjene studentima kako bi pokretali procese s Fijem, a sobe za instance procesa se kreiraju ovisno o procesu kojeg student odabere.



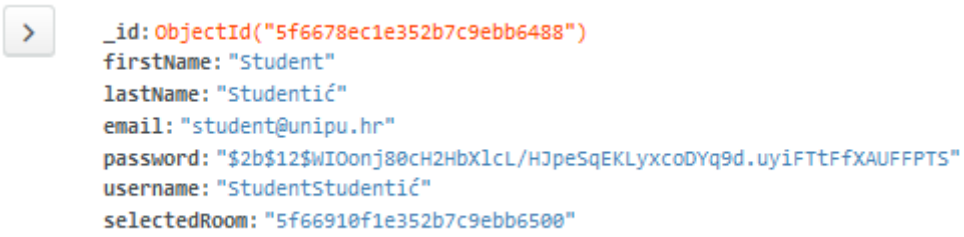
Slika 29. Primjer sobe u bazi

groups su korištene od strane vue-form-generatora za generiranje forme, tj. polja za odabiranje mentora. U *messages* spremamo sve poslane poruke.



Slika 30. Primjer poruke u bazi

processes služe za spremanje procesa, a u *submittedApplications* spremamo podatke završenih procesa. U *users* spremamo podatke o korisnicima.



Slika 31. Primjer korisnika u bazi

API ZA BAZU

Na Flasku u *dbroutes.py* definirane su rute za dohvat, ažuriranje, slanje i brisanje podataka s baze. Za pravilno parsiranje JSON-a potrebno je koristiti *json.dumps* i *json_util* kako bi dobili pravilan odgovor kojeg možemo koristiti na frontu:

```
# sve sobe
@app.route('/api/rooms', methods=['GET'])
def getRooms():
    try:
        docs_list = list(mongo.db.chatRooms.find())
        return json.dumps(docs_list, default=json_util.default)
    except Exception as e:
        return json.dumps({'error': str(e)})
```

Slika 32. Ruta za dohvaćanje soba

Prilikom ažuriranja, slanja ili brisanja podataka s front enda, potrebno je koristiti *@cross_origin* iz Flask-CORS modula jer inače zahtjev neće proći do baze. Također, potrebno je paziti na to da id-eve šaljemo u pravilnom formatu, kao *ObjectId*.

```python
# ažuriraj field poruke
@app.route('/api/messages', methods=['PUT'])
@cross_origin()
def updateMessageField():
    try:
        data = request.get_json()
        room_id = ObjectId(data["room"])
        values = {'$set': {
            data["field"]: data["value"]
        }}
        query = {"room_id": room_id}
        mongo.db.messages.update_many(query, values)
        return "ok"
    except Exception as e:
        return json.dumps({'error': str(e)})
```

Slika 33. Ruta za ažuriranje polja u bazi

```python
# dodaj poruku
@app.route('/api/messages', methods=['POST'])
@cross_origin()
def addMessage():
    try:
        data = request.get_json()
        data["sender_id"] = ObjectId(data["sender_id"])
        data["room_id"] = ObjectId(data["room_id"])
        mongo.db.messages.insert_one(data)
        return "ok"
    except Exception as e:
        return json.dumps({'error': str(e)})
```

Slika 34. Ruta za dodavanje nove poruke

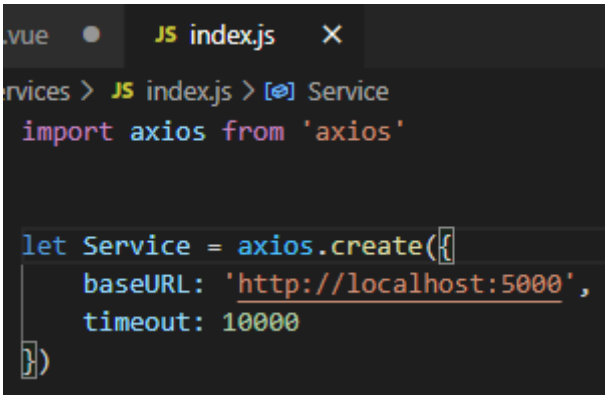
```python
# izbrisi određenu sobu
@app.route('/api/rooms/<room_id>', methods=['DELETE'])
@cross_origin()
def deleteRoom(room_id):
    try:
        mongo.db.chatRooms.remove({"_id": ObjectId(room_id)})
        return "ok"
    except Exception as e:
        return json.dumps({'error': str(e)})
```

Slika 35. Ruta za brisanje sobe

FRONT END

AXIOS

Potrebno je namjestiti Axios kako bi mogli slati zahtjeve na back end ili proslijeđivati zahtjeve s back enda. U ovom slučaju namješten je na port 5000 zbog toga što nam se tamo nalazi Flask i endpointovi s kojima želimo raditi.



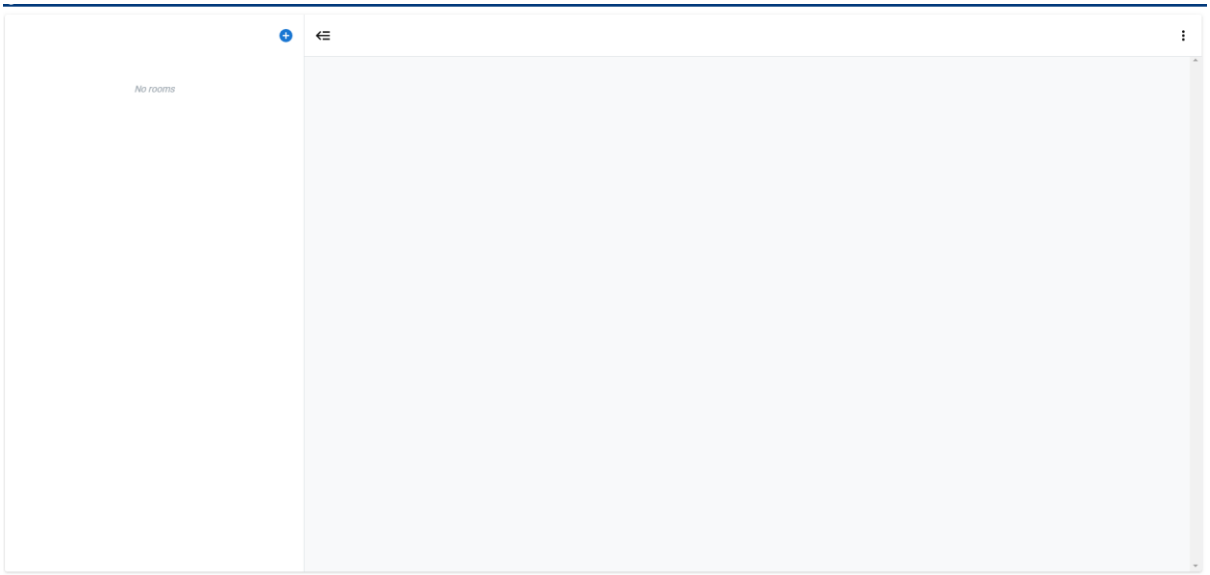
Slika 36. Postavke Axiosa

Kada radimo funkcije za Axios, koristimo *async-await* kako bi sačekali zahtjev da odradi što je potrebno, i tek onda išli dalje. Problematično je ako pokušavamo raditi s podacima, a zahtjev nije odrađen do kraja – tu nam uvelike pomaže *async-await* koji će osigurati da su podaci vraćeni ili poslani, ovisno o tome što nam je potrebno.

```
async getAll() {
    let response = await Service.get('/api/rooms')
    return response.data.map(doc => {
        return {
            roomId: doc._id,
            roomName: doc.name,
            users: doc.users,
            businessKey: doc.businessKey,
            processInstanceId: doc.processInstanceId,
            definitionId: doc.definitionId,
            variables: doc.variables,
            flag: doc.flag,
            initial: doc.initial,
            active: doc.active,
            awaitingResponse: doc.awaitingResponse,
            awaitingResponseBy: doc.awaitingResponseBy
        };
    });
},
```

Slika 37. Primjer Axios get-a

Korištenje vue-advanced-chat komponente za chat uvelike nam olakšava sam dio s dizajnom korisničkog sučelja jer je sučelje praktički gotovo, no i dalje je to samo okvir i nedostaju mu funkcionalnosti koje sami moramo implementirati.



Slika 38. Prazan chat okvir

Na chat-window vežemo podosta varijabli i funkcija.

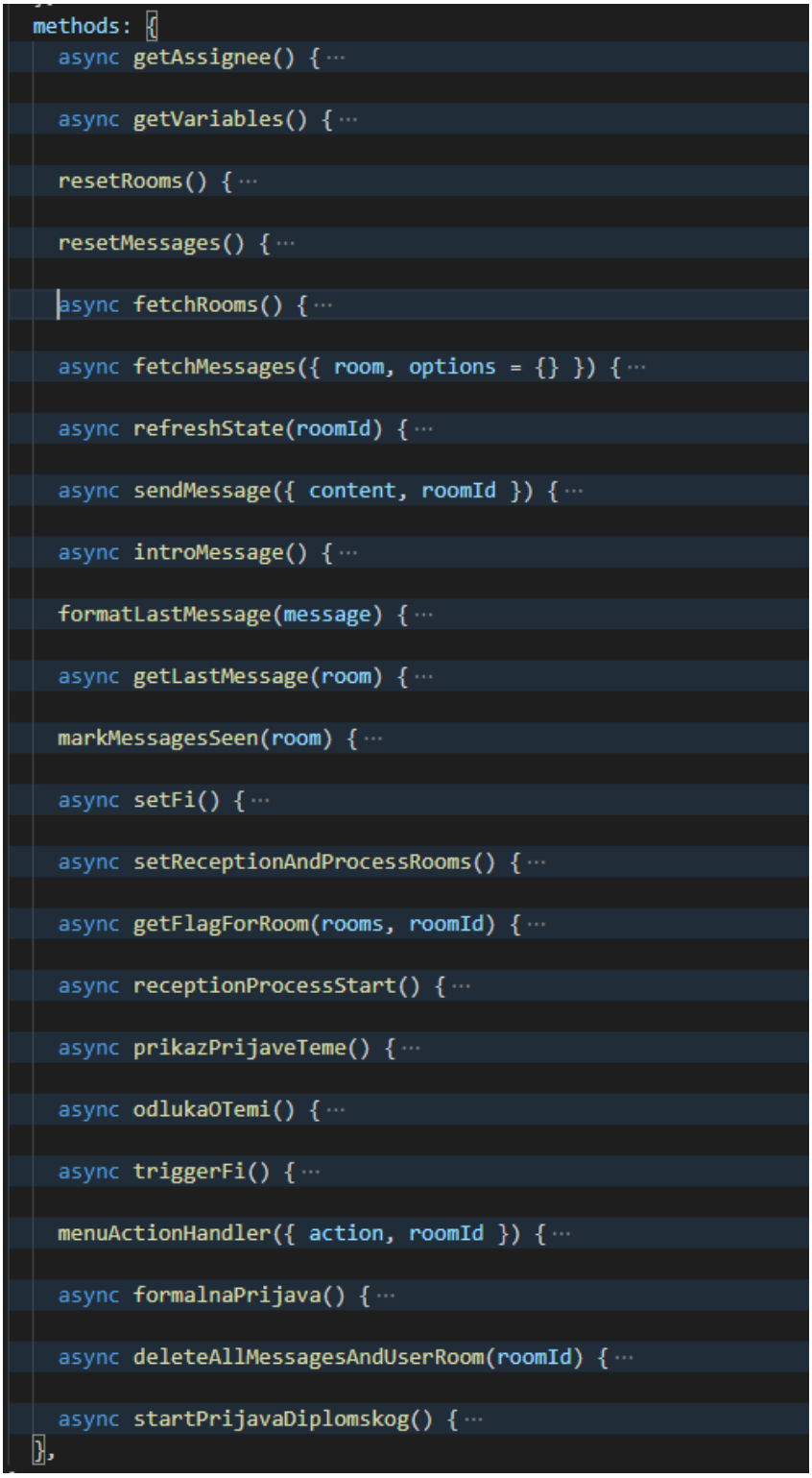
```
<chat-window
  height="calc(100vh - 90px)"
  :theme="theme"
  :currentUserId="currentUserId"
  :rooms="rooms"
  :loadingRooms="loadingRooms"
  :messages="messages"
  :messagesLoaded="messagesLoaded"
  :styles="styles"
  :menuActions="menuActions"
  @menuActionHandler="menuActionHandler
  @sendMessage="sendMessage"
  @fetchMessages="fetchMessages"
/>
```

Slika 39. chat-window komponenta

Varijablu *theme* možemo odabrati *light* ili *dark*, to već dolazi s komponentom. *currentUserId* prati koji je korisnik trenutno prijavljen u chatu, *rooms* sadrži sve sobe tog korisnika, *loadingRooms* služi za *spinner* dok čekamo da se neka soba učita, *messages* sadrži sve poruke

svih soba, *messagesLoaded* služi za označavanje jesu li sve poruke učitane. *styles* omogućuje podešavanje stila komponente, mijenjanje boja i slično. *menuActions* služi za *overflow* meni na kojeg se mogu vezati neke funkcije, npr. dodavanje korisnika, brisanje korisnika, resetiranje i slično. *@sendMessage* se okida prilikom slanja poruke, a *@fetchMessages* se okida prilikom dolaska u sobu ili selektiranja sobe.

Chat ima puno metoda:

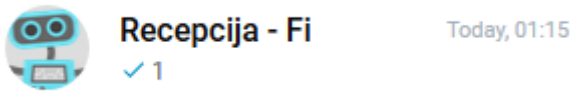


Slika 40. Metode za chat i chatbota

Glavne su *fetchRooms(), fetchMessages({room, options = {}), refreshState(roomId), sendMessage({content, roomId})*.

`fetchRooms()`

Poziva se u *mounted().* Uzima sobe korisnika slanjem upita na bazu za sobe u kojima se nalazi *currentUserId*. Radi listu soba i u te sobe stavlja korisnike iz kolekcije *users*, te uzima zadnje poruke sobe i to postavlja u tu listu kako bi se lijevo u chatu vidjelo kada je zadnja poruka poslana te je li pročitana. U toj funkciji također se postavlja avatar i naziv svake sobe.



Slika 41. Prikaz sobe, zadnje poruke i vremena slanja

`fetchMessages({room, options = {})`

Prima room i options, ti su parametri predefinirani iz komponente što znači da u rooms dobijemo podatke o sobe u kojoj se okida *fetchMessages* funkcija. S njom uzimamo poruke iz sobe koja je trenutno označena pomoću endpointa s back enda. Budući da se ova funkcija okida ulaskom u sobu, ovdje se nalaze i funkcije za chat logiku. U bazu za korisnika se postavlja selectedRoomId polje jer nam je jedna od osnova za uvjete ulaska u pojedine funkcije.

```
await Users.updateUserField(
  this.currentUserId,
  "selectedRoom",
  room.roomId
);
```

Slika 42. Axios upit za ažuriranje polja

Pozivamo *setReceptionAndProcessRooms()*, *introMessage()*, *markMessagesSeen()* i ovisno o tome je li soba stvorena za određeni proces, *prikazPrijaveTeme()* ili *startPrijaveDiplomskog()*. Važno je napomenuti da u ovoj funkciji postavljamo interval koji svakih devet sekundi poziva *refreshState()* zbog toga što još uvijek nemamo *real-time* funkcionalnost u aplikaciji i intervalom rješavamo taj problem.

`refreshState(roomId)`

Ona uzima korisničke sobe, filtrira ih i gleda imaju li flag *awaitingResponseBy* i *awaitingResponse* postavljen. Te varijable koristimo kako bismo vidjeli od koga trenutno očekujemo odgovor i čekamo li uopće odgovor. Zatim uzima poruke određene sobe kako bismo uvijek imali „svježe“ poruke u sobi, u slučaju da netko drugi pošalje nešto. Koristimo varijablu *checkBot* koja postavlja *messagesByBot* na true ili false, ovisno ako u toj sobi ima već prijašnjih poruka chatbota. To nam služi kao uvjet za ulazak u funkciju *startPrijavaDiplomskog().* Na kraju postavlja *messagesLoaded* na true budući da su sve poruke učitane i poziva getFlagForRoom().

`sendMessage({content, roomId})`

Služi za slanje poruka, ali i kao okidač mnogim funkcijama. Budući da je ona direktno vezana za send gumb, nakon senda poziva se *refreshState(roomId)* za instantno prikazivanje poslane poruke, funkcije vezane za proces završnog – *receptionProcessStart()*, *odlukaOTemi()*, *triggerFi()*, *formalnaPrijava()* i zadnja funkcija, *startPrijavaDiplomskog()*, ukoliko soba u kojoj se nalazimo pripada procesu prijave diplomskog.

`introMessage()`

Poziva se samo ukoliko je odabrana soba recepcija i nema prijašnjih poruka chatbota. Kao što naziv kaže, ona je inicijalna poruka koju chatbot šalje korisniku kako bi pokrenuli proces. Ona poziva *getProcesses()* s Axiosa kako bi dobila listu procesa i njih ponudila korisniku. Na kraju, polje *awaitingResponse* postavljamo na true jer očekujemo odgovor od korisnika.

`getAssignee()`

Uzima korisničko ime korisnika od kojeg se očekuje radnja kako bi se zadatak odradio. To je postignuto APIjem prema Camundi.

`getVariables()`

Uzima varijable koje su potrebne za izvršenje određenog zadatka, također putem APIja prema Camundi. Ukoliko je zadatak eksternalni, pozivanje ove funkcije izvršava taj zadatak stoga treba oprezno pozivati ovu funkciju.

`formatLastMessage(message)`

Ova funkcija je pozvana u *fetchRooms()* i služi za formatiranje prikaza zadnje poruke, pogotovo budući da je u bazi timestamp i taj timestamp treba pretvoriti u čitljivo vrijeme.

`getLastMessage(room)`

Služi za dohvaćanje zadnje poruke i prikaz iste u lijevoj sekciji chata.

`markMessagesSeen(room)`

Pri ulasku u sobu, sve poruke označava pročitanim i mijenja polje seen u bazi na true.

`setFi()`

Funkcija se poziva iz *mounted()* te služi za postavljanje id-a chatbota u *data()*.

`setReceptionAndProcessRooms()`

Uzima korisničke sobe iz baze i procese, filtrira i postavlja varijable u *data()* za sobe koje su zapravo procesi kako bismo mogli imati odvojene funkcije ovisno o tome o kojem se procesu tu radi. Također postavlja *receptionRoom* varijablu.

`getFlagForRooms(rooms, roomId)`

Poziva se ako soba nije recepcija i gleda da li soba ima flag u bazi postavljen na true, ako je flag postavljen na true to znači da su Camundine varijable povučene i prikazane u chatu.

`receptionProcessStart()`

Pokreće se ako korisnik nije Fi, ako je soba recepcija i ako se očekuje odgovor od korisnika. Služi za pokretanje procesa, u ovom slučaju procesa prijave završnog ili procesa upisa na diplomski. Nakon odgovora korisnika, postavlja vrijednosti *awaitingResponse* na false i *awaitingResponseBy* na prazno. Na kraju funkcije poziva se *fetchRooms()* kako bi se osvježio prikaz soba.

`prikazPrijaveTeme()`

Ukoliko varijable nisu prikazane u chatu i ukoliko je soba vezana za proces prijave završnog, ova funkcija se izvršava. Služi za ispis onoga što je korisnik upisao u generiranu formu i postavlja se čekanje odgovora od mentora.

`odlukaOTemi()`

Izvršava se ako dohvaćene varijable iz Camunde imaju objekt model i ako objekt model u sebi ima key „odluka“. Ova funkcija čeka odgovor od mentora želi li on raditi sa studentom na ovoj temi, ili ne. Ovdje se pojavljuje funkcija APIja prema Camundi *sendTaskVariables(user, variables)* koja nam omogućuje slanje varijabla. U ovom slučaju, varijabla koju nam Camunda daje je „odluka“ i mi pomoću *sendTaskVariables* šaljemo da je vrijednost odluke da ili ne, ovisno o odgovoru mentora.

`triggerFi()`

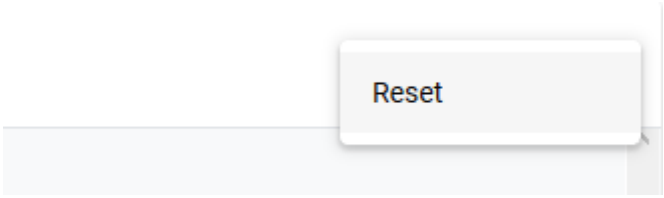
Ova funkcija se okida kada mentor kaže *@Fi* nakon što se on i student dogovore oko prijave rada. Tada Fi započinje proces upisivanja novih podataka za formalnu prijavu rada sa studentom. Ovdje također koristimo *sendTaskVariables* za slanje varijabli.

`formalnaPrijava()`

Pokreće se nakon *triggerFi()* kada student odgovori inicijalnom pitanju Fija. Zatim Fi postavlja pitanja studentu sve dok zajedno ne prođu sve potrebne varijable za prijavu. Funkcija koristi varijablu *stepsCounter* kako bi se svakim odgovorom studenta vraćali u funkciju ali na neki drugi korak sve do kraja. Varijable šaljemo pomoću *sendTaskVariables*. Na kraju funkcije pozivamo *getVariables()* kako bi završili proces budući da više nema varijabli za preuzeti.

`deleteAllMessagesAndUserRoom(roomId)`

Funkcija se nalazi u *overflow* meniju i njome brišemo sve poruke i sobu korisnika, stoga se taj gumb zove reset. Nije moguće brisati recepcije, samo procesne sobe.



Slika 43. Reset gumb u overflow meniju

`startPrijavaDiplomskog()`

Funkcija koja obavlja cijeli proces upisa diplomskog. Započinje porukama chatbota, i ovaj put pomoću *stepsCounter2* dobiva odgovor i odgovara korisniku. Sličan princip prolaska kroz proces kao od *formalnaPrijava()* funkcije. Varijable šaljemo pomoću *sendTaskVariables*.

STANJE I RESPONZIVNOST

U aplikaciji vrlo je važno održavanje stanja, a to činimo s Vuexom, bazom i *data()* varijablama. Prilikom prijave korisnika, trebamo održati njegovo stanje prijave čak i kada on osvježi stranicu. Tako trebamo održavati stanje konverzacija korisnika i chatbotova kako se stvari ne bi duplicirale, kako se korisnik ne bi osjećao izgubljenim i kako bi sve bilo spremljeno. Vuex primarno koristimo za održavanje autentikacije korisnika i za držanje podataka o tome koje su sobe korisnika vezane za proces. Ostalo radimo pomoću *data()* i baze, zato imamo i puno varijabli u bazi kako bismo mogli održavati uvjete za ulazak u pojedine funkcije i kako ne bismo pozivali nepotrebne funkcije.

```
Vue.use(Vuex)

export default new Vuex.Store({
    state: {
        auth: false,
        firstName: "",
        lastName: "",
        username: "",
        id: "",
        initials: "",
        processRoomId1: "",
        processRoomId2: "",
    },
    mutations: {
        setToTrue(state) {
            state.auth = true;
        },
        setToFalse(state) {
            state.auth = false;
        },
        setFirstName(state, value){
            state.firstName = value
        },
        setLastName(state, value){
            state.lastName = value
        },
        setUsername(state, value){
            state.username = value
        },
        setId(state, value){
            state.id = value
        },
        setInitials(state, value){
            state.initials = value
        },
        setProcessRoomId1(state, value){
            state.processRoomId1 = value
        },
        setProcessRoomId2(state, value){
            state.processRoomId2 = value
        },
    },
    actions: {
        setFirstName(context, value){
            context.commit('setFirstName', value)
```

Slika 44. Prikaz Vuex-a

Kod održavanja stanja i responzivnosti ne smijemo zaboraviti na *computed* svojstva koja se mijenjaju ovisno o promjeni nekih varijabli što nam daje brzinu i jednostavnost rada.

```
computed: {
  lastMessage: function () {
    if (!(this.messages === undefined || this.messages.length == 0)) {
      return [
        {
          sender: this.messages[this.messages.length - 1].sender_id,
          content: this.messages[this.messages.length - 1].content,
        },
      ];
    }
    return "None";
  },
  awaitingResponse: function () {
    return this.awaitingResponseDb;
  },
  awaitingResponseBy: function () {
    return this.awaitingResponseByDb;
  },
},
```

Slika 45. Computed svojstva u chatu

lastMessage je jedan od najvažnijih svojstava jer nam omogućava postavljanje okidača na posljednju poruku i to nam puno znači budući da nemamo integriran websocket pomoću kojeg bismo mogli imati okidač vezan na endpoint koji direktno javlja dodavanje poruke u kolekciju *messages* na bazi.

DALJNJI KORACI

Uspjeli smo stvoriti *user-friendly* ljusku na dva poslovna procesa, no koji su koraci dalje? Rješenje vidimo u koraku naprijed prema automatizaciji. Automatizacija s chatbot strane bi značilo pametno razumijevanje upisanog teksta i stvaranje odgovora na temelju izvučenog konteksta. Odgovori mogu biti predefinirani ovisno o modeliranom poslovnom procesu, te onda kada chatbot shvati što korisnik želi, pošalje mu se odgovor i odradi se zadatak koji je trenutno na redu u procesu.

Izvlačenje konteksta jednostavno je raditi na engleskom jeziku budući da postoje razni alati za to, za hrvatski jezik poznat je jedan alat te izgleda obećavajuće. Radi se o ReLDIanno[9], a omogućuje tagiranje, lematizaciju, NER (named entity recognition) i parsiranje ovisnosti.

[9] ReLDIanno – storitev za označevanje slovenskih, hrvaških in srbskih besedil, http://www.clarin.si/info/k-center/spletne-storitve/

Također sadrži i leksikon. Alat je napisan u Pythonu što znači da ako nam je sljedeći korak korištenje njega, sva chat logika ide u back end budući da nam se tamo nalazi Python.

	Surface	Lemma
1.	Gosti	gost
2.	su	biti
3.	stigli	stići
4.	.	.

Slika 46. Lematizacija u ReLDIanno

ZAKLJUČAK

Poslovni procesi vođeni chatbotom definitivno imaju svoje mjesto u budućnosti jer pružaju mnogo prednosti. Možemo vidjeti da su procesi u praksi generalno prekompleksni, teško ih je pratiti i pravilno izvršavati. Danas uz pomoć chatbot sustava poslovne procese možemo znatno olakšati i time pridonijeti internoj organizaciji te zadovoljiti korisnika i poduzeće koje ih koristi.

LITERATURA

Adiwardana, Luong: Towards a Conversational Agent that Can Chat About…Anything, 2020. (https://ai.googleblog.com/2020/01/towards-conversational-agent-that-can.html)

Artificial Solutions: Chatbots: The Definitive Guide, 2020. (https://www.artificial-solutions.com/chatbots)

Blaće, Dubravko: Zašto je važno razumjeti poslovne procese i upravljati njima?, 2015. (https://www.evision.hr/hr/Novosti/Stranice/zasto-razumjeti-poslovne-procese-upravljati-procesima.aspx)

Hunt-Walker, Nicholas: An introduction to the Flask Python web app framework, 2018. (https://opensource.com/article/18/4/flask)

Kollegger, Eric: What is Axios.js and why should I care?, 2018. (https://medium.com/@MinimalGhost/what-is-axios-js-and-why-should-i-care-7eb72b111dc0)

McQuistan, Adam: Single Page Apps with Vue.js and Flask: Setting up Vue.js, 2017. (https://stackabuse.com/single-page-apps-with-vue-js-and-flask-setting-up-vue-js)

Medlin, Jace: Combining Flask and Vue, 2020. (https://testdriven.io/blog/combine-flask-vue/)

MongoDB: What Is MongoDB?, 2020. (https://www.mongodb.com/what-is-mongodb)

PythonBasics: What is Flask Python, 2020. (https://pythonbasics.org/what-is-flask-python/)

Siying, Han: Business process automation through chatbots implementation: A case study of an IT service process at Philips, 2019.

Starčić, Toni: Razvoj programskog sučelja za konverzacijskog agenta vođenog modelom procesa, 2020.

Šnajder, Jan: 2. Basics of Natural Language Processing, 2018.

Vuetify: What's the difference?, 2020. (https://vuetifyjs.com/en/introduction/why-vuetify/)