

# Upravljanje poslovnim procesima (UPP)

**Nositelj:** izv. prof. dr. sc. Darko Etinger

**Asistent:** Luka Blašković, mag. inf.

**Ustanova:** Sveučilište Jurja Dobrile u Puli, Fakultet informatike u Puli



Fakultet informatike u Puli

## (1) Uvod u poslovno modeliranje

#1

UPP

Sva razmatranja o poslovnim procesima temelje se na potrebi da svoje poslove (bez obzira na to radi li se o proizvodnji, trgovini, uslugama, javnoj upravi, zdravstvu, obrazovanju, itd.) obavimo optimalno, odnosno što brže i kvalitetnije te uza što manji utrošak resursa. Modeliranje poslovnih procesa je jedan od ključnih alata za postizanje tog cilja, a ona je prije svega poslovna, a ne informatička disciplina. Na ovom kolegiju ćemo se praktično upoznati s modeliranjem poslovnih procesa pomoću BPMN 2.0 notacije i istražiti kako se ta znanja primjenjuju u razvoju softverskih rješenja.

 Posljednje ažurirano: 7.11.2024.

## Sadržaj

- [Upravljanje poslovnim procesima \(UPP\)](#)
- [\(1\) Uvod u poslovno modeliranje](#)
  - [Sadržaj](#)
- [1. Uvod](#)
- [2. BPMN standard](#)
  - [2.1 Softver](#)
- [3. Osnove modeliranja procesa](#)
  - [3.1 Osnovni elementi](#)
  - [3.2 Skretnice \(eng. Gateway\)](#)
    - [3.2.1 Ekskluzivna skretnica \(eng. Exclusive Gateway\)](#)
  - [3.3 Tumačenje skretnica](#)
  - [Vježba 1: Izdavanje kredita](#)
  - [Vježba 2: Proces obrade natječaja](#)

- [4. Hijerarhija procesa i potprocesa](#)
  - [4.1 Staze i polja](#)
- [Zadaci za Vježbu 1](#)
  - [1. Proces obrade reklamacije](#)
  - [2. Proces najma vozila](#)
  - [3. Proces automatizirane korisničke podrške](#)

## 1. Uvod

---

Uspješno upravljanje organizacijom, a osobito povećanje njezine učinkovitosti radi postizanja postavljenih ciljeva, moguće je samo ako se izvrsno poznaje njezin unutarnji ustroj i način djelovanja. Organizacija djeluje nizom poslovnih procesa koji su međusobno povezani i ovise jedan o drugom, a svaki od njih usmjeren je ka ostvarivanju određenog cilja.

Pojednostavljeni se može reći da je **poslovni proces** skup povezanih radnih koraka za koje se mogu odrediti trajanje izvođenja i potrebni resursi.

Učinkovitost djelovanja organizacije može se povećati unapređenjem i preustrojem poslovnih procesa. Međutim, važno je da svi dionici razumiju poslovne procese, a to se može postići ako se oni opišu jednoznačno i svima razumljivo. Upravo je to cilj **poslovnog modeliranja**.

Na primjer, opis govornim jezikom: "Kupac naručuje proizvod, prodavač zaprima narudžbu, skladištar priprema proizvod za isporuku, vozač dostavlja proizvod kupcu, a blagajnik izdaje račun.", svakako je jedan od načina opisa poslovnog procesa, ali on može biti nedovoljno precizan te različito tumačen od strane različitih dionika.

Stoga se danas poslovni procesi egzaktно opisuju skupom grafičkih simbola s točno definiranom semantikom i čvrstim pravilima njihova povezivanja, a sve je to određeno međunarodnom normom.

## 2. BPMN standard

---

Da bi se neki proces mogao analizirati i unaprijediti, potrebna je ne samo općeprihvaćena definicija već je jednako tako potrebno jednoznačno opisati sva njegova relevantna svojstva. Prikladan je način opisivanje procesa kroz njegov grafički prikaz, osobito ako je dopunjeno formalnim opisom pojedinih značajki. Poslovni ljudi, menadžeri i projektanti informacijskih sustava već odavna primjenjuju različite načine grafičkog prikazivanja poslovnih procesa.

Najnovija i danas najšire primjenjivana norma naziva se [BPMN \(Business Process Modelling and Notation\)](#). BPMN je standard za modeliranje poslovnih procesa koji pruža grafičku notaciju za modeliranje poslovnih procesa, ali i tehničku notaciju za izvođenje tih modela u informacijskim sustavima. BPMN je razvijen od strane **Object Management Group** (OMG) i prvi put je objavljen 2004. godine. Trenutno je najnovija verzija BPMN 2.0, objavljena 2011. godine. OMG grupa je tijekom godina definirala mnoge standarde u području objektno-orientiranog modeliranja i razvoja softvera, osim BPMN-a, neki od poznatijih su:

- **UML** (Unified Modelling Language) - vjerojatno najpoznatiji OMG-ov standard. To je grafički jezik za vizualizaciju, specifikaciju i dokumentiranje softverskih sustava, od strukture, ponašanja i interakcije između različitih elemenata.

- **MDA** (Model Driven Architecture) - standard za razvoj softvera koji naglašava važnost modela u cijelom životnom ciklusu razvoja softvera.
- **CORBA** (Common Object Request Broker Architecture) - nešto stariji standard koji je definirao arhitekturu za distribuiranu objektno-orientiranu komunikaciju. Standard je imao značajan doprinos u razvoju distribuiranih sustava.
- **SysML** (Systems Modelling Language) - standard za modeliranje složenih sustava, koji je proizašao iz UML-a, ali je prilagođen za potrebe modeliranja složenih inženjerskih sustava, uključujući i hardverske komponente.



Mi ćemo se na ovom kolegiju baviti isključivo BPMN 2.0 notacijom, a u nastavku ćemo se upoznati s osnovnim elementima modeliranja procesa kroz jednostavni primjer procesa obrade narudžbi kupaca i izdavanje naručene robe.

## 2.1 Softver

Za modeliranje poslovnih procesa u BPMN notaciji možete koristiti veliki broj alata, a neki od popularnijih su:

- Camunda Modeler: besplatno, open-source rješenje koje podržava BPMN 2.0 i DMN notaciju.  
Preuzmite Desktop verziju [ovdje](#)
- bpmn.io: open-source rješenje koje se može direktno koristiti u web pregledniku. Isprobajte [ovdje](#)
- Flowable: open-source community rješenje koje nudi podršku za modeliranje u web pregledniku.  
Morate se registrirati da biste koristili alat, a možete ga isprobati [ovdje](#)

Ako ste se odlučili za Camunda Modeler na vašem računalu, morate imati instaliran Java JDK 8 ili noviji. Otvorite terminal i upišite:

```
java -version
```

Ako nemate instaliran Java JDK, možete preuzeti i open-source verziju [Open JDK](#).

## 3. Osnove modeliranja procesa

Definiran je sljedeći opis poslovnog procesa:

Zamislimo prodajni centar kao dio neke proizvodne organizacije s pomoću kojeg ona prodaje svoju robu široke potrošnje, primjerice hladnjake, perilice rublja i sl. Takav se proces izvodi u svakom od područnih veleprodajnih centara, kojim proizvodna organizacija robe široke potrošnje opskrbљuje svoje velike kupce (npr. distributere, hotele ili restorane). Zamislimo da je poslovnim poslovnom politikom prodajnog centra propisano da se naručena roba može izdati kupcu samo ako je već plaćena po predračunu.

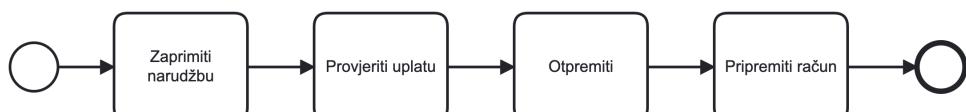
Poslovni proces **PRODATI ROBU** tada se provodi tako da prodajni centar zaprimi narudžbu od kupca, provjeri je li naručena roba plaćena po predračunu, otpremi robu kupcu i pripremi konačni izlazni

račun. Takav slijed poslova ili radnih koraka (pri kojem se upotrebljavaju i podaci o zalihamama, kupcima, narudžbama itd.) nazivamo **poslovnim procesom**.

Uočavamo da ovaj poslovni proces ima svoj **početak** i **kraj**, da se ponavlja svaki put kada neki kupac želi naručiti i preuzeti bilo koju robu te da se sastoji od više povezanih poslova ili radnih koraka koje ćemo općenito nazvati **aktivnostima**.

## 3.1 Osnovni elementi

**Aktivnost (eng. Task)** je osnovni element poslovnog procesa koji predstavlja radni korak koji se izvodi u procesu. Aktivnosti se ne obavljaju proizvoljno, već uvijek u određenom slijedu. Tako opisan proces može se prikazati grafički na sljedeći način:



Slika 1. Poslovni proces **PRODATI ROBU** i njegove aktivnosti

Cijeli je proces na slici 1 prikazan kao niz **aktivnosti**, prikazanih pravokutnicima sa zaobljenim rubovima i povezanih **slijednom vezom**.

**Strelice** povezuju aktivnosti procesa i pokazuju slijed izvođenja aktivnosti.

Početak i kraj procesa su **događaji (eng. events)**, a oni su prikazani krugovima koji su iscrtani kružnicama:

- **početak** koji je iscrtan tankom i
- **kraj** koji je iscrtan debljom crtom.

Dakle upotrebljena su tri simbola koja mora imati svaki model poslovnog procesa prema BPMN normi.

**Kružnica** (za početni i završni događaj)

**Aktivnost** (označava se pravokutnikom)

**Strelica** (za redoslijed izvođenja aktivnosti)

Ovakav temeljni oblik procesa naziva se često i **slijednim dijagramom**.

Međutim, slika 1. prikazuje kako se proces izvodi u idealnom slučaju, odnosno kada je kupac upatio po predračunu točno onaj iznos koji odgovara vrijednosti naručene robe, a tražena roba je dostupna na skladištu te se odmah poslije primitka narudžbe može otpremiti te izraditi račun za kupca.

## 3.2 Skretnice (eng. Gateway)

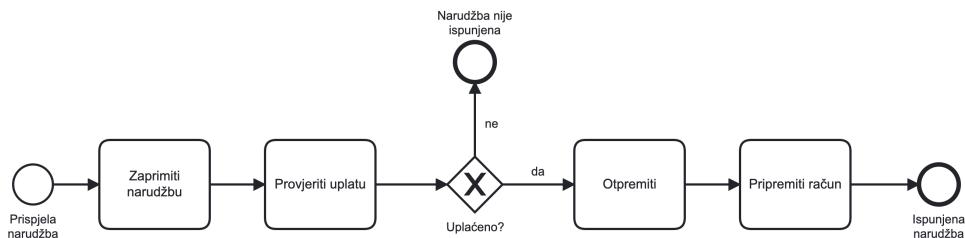
Što ako kupac nije prethodno platio po predračunu ili tražene robe nema na skladištu?

U tom slučaju, posao prodaje robe neće se moći provesti na opisani način. Stoga naš model procesa treba proširiti kako bi se prikazali uvjeti izvedbe prema dopunjrenom scenariju.

U BPMN notaciji za prikaz uvjeta izvedbe koriste se **skretnice** (eng. *Gateway*). Skretnice su elementi koji omogućuju modeliranje uvjeta izvedbe, odnosno odlučivanje o tome koja će se aktivnost izvršiti sljedeća. Skretnice se označavaju **rombom**.

Gdje ćemo dodati **prvu skretnicu** u naš model procesa?

Odgovor je nakon aktivnosti **Provjeriti uplatu** jer je to prvi korak u kojem se može dogoditi odstupanje od idealnog slučaja. Naime, ako kupac nije upatio po predračunu, proces se ne može nastaviti u slijedu definiranom na slici 1.



Slika 2. Proširen model poslovnog procesa **PRODATI ROBU** s prvom skretnicom

Na slici 2. dodana je prva skretnica koja omogućuje modeliranje uvjeta izvedbe. U ovom slučaju, skretnica označava da se proces nastavlja **samo ako je uplata po predračunu primljena**. Ako nije, proces završava u **krajnjem događaju** (eng. *end event*).

Ispod skretnice je uobičajeno pisati uvjet izvedbe, to može biti bilo koja upitna rečenica koja jasno opisuje uvjet. Primjerice:

- *Uplaćeno?*
- *Uplata po predračunu primljena?*
- *Uplata primljena?*
- *Uplata je izvršena?*

Nakon toga skretnica se povezuje s aktivnostima koje slijede, a koje će se izvršiti ovisno o ispunjenosti uvjeta:

- **Da** - ako je uvjet ispunjen
- **Ne** - ako uvjet nije ispunjen

Odgovore na ova pitanja označavamo **strelicama** koje izlaze iz skretnice. U ovom slučaju, događa se sljedeće:

- **Da** - ako je uplata po predračunu primljena, proces se nastavlja s aktivnostima **Otpremiti** i **Pripremiti račun**
- **Ne** - ako uplata po predračunu nije primljena, proces završava u **krajnjem događaju**.

Primijetite da smo dodali **više krajnjih događaja i nazive događajima** kako bi bilo jasno što se događa u svakom koraku procesa. Model u kojem je više početnih i više završnih događaja u skladu je s BPMN normom i teorijski ispravan, ali uvijek treba provjeriti odgovara li izvođenje procesa u stvarnosti zaista nacrtanom modelu.

### 3.2.1 Ekskluzivna skretnica (eng. Exclusive Gateway)

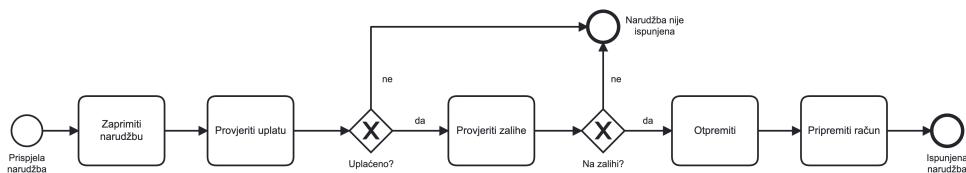
Primijetite još da unutar romba koji opisuje svaku skretnicu, koristimo simbol **X**. Ovaj simbol označava **ekskluzivnu skretnicu (eng. Exclusive Gateway)**. Ekskluzivna skretnica je skretnica koja omogućuje odabir samo jedne od više mogućih putanja. Vrednuje se podatak koji dolazi iz prethodne aktivnosti i na temelju njega **odabire samo jedan mogući slijed** na temelju izračunate vrijednosti ili zadanih uvjeta.

Ova skretnica poznata je i kao **XOR Gateway**.



- Ako je riječ o grananju procesa, onda znači da će se poslije skretnice provoditi aktivnosti **samo na jednom izlaznom slijedu**.
- Ako više uvjeta može biti istinito, ova skretnica odabire samo onaj slijed **koji je prvi zadovoljen**.
- Ako niti jedan uvjet nije zadovoljen, proces vraća grešku. Dobra praksa je osigurati da uvjeti budu **potpuni i iscrpni**.

U sljedećem primjeru, dodat ćemo još **jednu ekskluzivnu skretnicu** u naš model procesa kako bismo modelirali uvjet je li tražena roba dostupna na skladištu.



Slika 3. Proširen model poslovnog procesa **PRODATI ROBU** s dvjema ekskluzivnim skretnicama

Dakle, na slici 3. dodana je **druga ekskluzivna skretnica** koja omogućuje modeliranje uvjeta je li tražena roba dostupna na skladištu. Ako je roba dostupna, proces se nastavlja s aktivnostima **Otpremiti** i **Pripremiti račun**. Ako roba nije dostupna, proces završava u **krajnjem događaju**.

Dodali smo i aktivnost **Provjeriti zalihe** koja prethodi drugoj skretnici. Ova aktivnost odnosi se na samu provjeru zaliha na skladištu. Aktivnost smo dodali budući da nije praksa da se aktivnosti prikazuju kroz skretnice, već da skretnice definiraju uvjete izvedbe aktivnosti.

Dakle slijed je sljedeći: aktivnost -> skretnica -> aktivnost -> skretnica.

1. **Aktivnost:** provjeriti uplatu po predračunu
2. **Skretnica:** je li uplata po predračunu primljena?
3. **Aktivnost:** provjeriti zalihe na skladištu
4. **Skretnica:** je li roba dostupna na skladištu?

Općenito govoreći, svaka skretnica omogućuje stvaranje složenog grafa kojim se, od početne do krajnje točke, može proći putovima, odnosno proces se može ostvariti izvođenjem aktivnosti različitim sljedovima. Svaki od tih sljedova prikazuje pojedinačni i specifični način izvođenja poslovnog slučaja koji pripada istom, generičkom modelu poslovnog procesa.

Svaki od izvedenih sljedova prikazuje jednu **instancu** generičkog procesa, odnosno svaka je instanca jedan od mogućih načina izvođenja procesa s različitim ishodima ili **poslovni slučaj**.

Već na jednostavnom grafu na slici 3 mogu se prepoznati tri različite mogućnosti (ili tri različita slijeda aktivnosti i događaja) izvođenja poslovnog procesa **PRODATI ROBU**. To su:

- | Prispjela narudžba | 'Zaprimiti narudžbu' | 'Provjeriti uplatu' | **Narudžba nije ispunjena** |
- | Prispjela narudžba | 'Zaprimiti narudžbu' | 'Provjeriti uplatu' | Provjeriti zalihe | **Narudžba nije ispunjena** |
- | Prispjela narudžba | 'Zaprimiti narudžbu' | 'Provjeriti uplatu' | Provjeriti zalihe | Otpremi | Pripremiti račun | **Ispunjena narudžba** |

### 3.3 Tumačenje skretnica

Značenje skretnice u danom primjeru treba tumačiti na sljedeći način:

Značenje **prve skretnice** treba tumačiti ovako: nakon što je obavljena aktivnost Provjeriti uplatu znat će se je li kupac uplatio naručenu robu.

Ako potrebni iznos nije uplaćen (ovaj uvjet zapisan je ispod simbola skretnice tekstrom 'Uplaćeno?'), roba neće biti otpremljena kupcu i proces će završiti u krajnjoj točki (događaju) s oznakom 'Narudžba nije ispunjena'.

Ako je potreban iznos uplaćen i roba uspješno otpremljena kupcu, onda se proces nastavlja provjerom može li se otpremi naručenu robu s obzirom na trenutačno stanje zaliha. Taj se uvjet ispituje u **drugoј skretnici** s oznakom 'Na zalihi?' koja imenom podsjeća na uvjet koji se ispituje.

Ako su oba uvjeta ispunjena, poslovni će proces završiti onako kako se očekuje, odnosno poslovni će slučaj završiti događajem koji je nazvan 'Ispunjena narudžba'.

#### Važna napomena:

Skretnica pri modeliranju procesa i selekcija kao programski konstrukt (odnosno "grananje" programa) nipošto se ne mogu izjednačiti. Skretnica u modeliranju procesa ima mnogo šire značenje od odluke ili grananja u programiranju, odnosno odluka je samo jedna posebna vrsta skretnice. To će biti detaljno objašnjeno u nastavku kolegija.

### Vježba 1: Izdavanje kredita

Na temelju sljedećeg opisa poslovnog procesa i do sada obrađene BPMN notacije, definirajte model poslovnog procesa koji je opisan u sljedećem tekstu. Za vježbu možete koristiti alat za modeliranje po vlastitom izboru.

Banka je ustanova koja pruža razne finansijske usluge svojim klijentima, uključujući i izdavanje kredita. Banka je definirala poslovni proces **IZDATI KREDIT** koji se provodi svaki put kada klijent zatraži kredit. Jednom kada klijent zatraži kredit, banka prvo provjerava je li predani zahtjev kompletan, ako nije, klijenta se ponovo šalje na popunjavanje zahtjeva. Inače banka provjerava kreditnu sposobnost klijenta te prekida proces ako utvrdi da klijent nije kreditno sposoban. Ako je klijent kreditno sposoban, banka potpisuje ugovor s klijentom što u konačnici rezultira isplatom kredita na račun klijenta.

### Vježba 2: Proces obrade natječaja

Na temelju sljedećeg opisa poslovnog procesa i do sada obrađene BPMN notacije, definirajte model poslovnog procesa koji je opisan u sljedećem tekstu. Za vježbu možete koristiti alat za modeliranje po vlastitom izboru.

Tvrtka koja se bavi proizvodnjom i prodajom proizvoda na tržištu odlučila je proširiti svoj tim te je definirala poslovni proces **ODABIR KANDIDATA**. Tvrtka je već provela javni natječaj na koji su se mogli javiti zainteresirani kandidati. Proces započinje jednom kad javni natječaj završava, odnosno kada istekne rok za predaju potrebne dokumentacije. Voditelj odsjeka za upravljanje ljudskim resursima (HR) prikuplja natječaje i provjerava je li barem jedan kandidat dostavio svu potrebnu dokumentaciju. Ako nije, natječaj se poništava. Međutim ako postoji barem jedan kandidat koji je dostavio svu potrebnu dokumentaciju, voditelj HR-a provjerava kvalifikacije kandidata te poništava natječaj ako niti jedan kandidat nema potrebne kvalifikacije. U suprotnom, voditelj HR-a poziva kandidate na razgovor (čak i ako je samo jedan kandidat zadovoljio uvjete natječaja) te na temelju razgovora donosi odluku o zapošljavanju.

## 4. Hijerarhija procesa i potprocesa

Vratimo se na proces **PRODATI ROBU**. Ako se proces detaljnije razmotri s poslovnog gledišta, vidjet će se da je njegov model na slici 3 još uvijek suviše općenit jer ne sadržava sve informacije o mjestu i načinu izvođenja procesa i njegovih aktivnosti te ne govori ništa o tehnologiji kojom se te aktivnosti izvode.

Ako se vratimo na opis poslovnog procesa **PRODATI ROBU**, vidjet će se da u njemu postoji nekoliko **akteri** koji sudjeluju u procesu:

- **Kupac** koji naručuje robu
- **Prodavač** koji zaprima narudžbu
- **Knjigovođa** koji vodi evidenciju o uplatama i izdanim računima
- **Skladištar** koji priprema robu za otpremu

Kako u definiciji, odnosno opisu poslovnog procesa, ovaj proces započinje prispjelom narudžbom, kupca nećemo uvrstiti u granice procesa već ćemo ga smatrati **vanjskim akterom**.

Dakle, okvirno možemo podijeliti aktere u 3, odnosno **organizacijske jedinice** koje sudjeluju u procesu:

1. **Prodaja**
2. **Knjigovodstvo**
3. **Skladište**

### 4.1 Staze i polja

U BPMN notaciji, proces se može podijeliti na **staze (eng. lanes)** i **polja (eng. pools)**. U grubo, staze se koriste za prikazivanje različitih organizacijskih jedinica koje sudjeluju u procesu, dok se polja koriste za prikazivanje različitih poslovnih procesa.

Pojmovi **staza** i **polje** nisu doslovni prijevodi engleskih riječi lane (swim lane) i pool, već izabrane hrvatske riječi koje bolje objašnjavaju značenja u kontekstu modeliranja poslovnih procesa.

Polje	Staza 1
	Staza 2
	Staza 3

U nekim sljedećim poglavljima ćemo detaljno vidjeti koje su dobre prakse modeliranja kroz staze i polja, za sada ćemo **podijeliti naše organizacijske jedinice u staze**, dok će naziv polja biti naziv procesa - **PRODATI ROBU**.

PRODATI ROBU	PRODAJA
	KNJIGOVODSTVO
	SKLADIŠTE

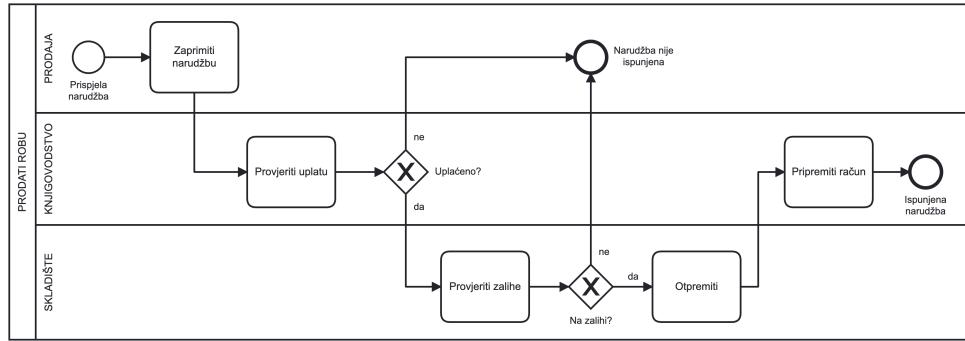
Bez da previše razbijamo glavu kako koristiti staze i polja, možemo se držati sljedećeg pravila:

- **BPMN polja** (pools) opisuju cijele organizacije ili poslovne procese, i sadrže staze
- **BPMN staze** (lanes) opisuju odjeljke organizacije, odnosno tko je odgovoran za koje aktivnosti

Sada ćemo pokazati kako bi izgledao model poslovnog procesa **PRODATI ROBU** s dodanim stazama i poljem.

No prije toga idemo definirati tko obavlja koje aktivnosti u procesu:

- **Prodaja:**
  - Prodavač zaprima narudžbu i obrađuje je
- **Knjigovodstvo:**
  - Knjigovođa provjerava uplatu po predračunu i izdaje račun
- **Skladište:**
  - Skladištar priprema robu za otpremu



Slika 4. Prošireni model poslovnog procesa **PRODATI ROBU** s dodanim stazama i poljem

### Radno mjesto i staza

Sve se aktivnosti ne izvode na istome mjestu: narudžbu od kupca prima **PRODAJA**, robu otprema **SKLADIŠTE**, a **KNJIGOVODSTVO** će provjeriti uplatu i pripremiti račun za kupca.

Radna se mjesta koja sudjeluju u procesu prikazuju izduljenim pravokutnikom koji uz lijevu stranicu ima naziv radnog mjeseta (**staza**), a ucrtavanje simbola za aktivnost unutar simbola za radno mjesto znači, prema konvenciji za BPMN, da se **aktivnost izvodi na onom radnom mjestu na čijoj je površini nacrtana**.

Posao koji je na slici opisan aktivnošću **Otpremiti** nije jednostavan, već se sastoji od više radnih koraka. Takav se posao definira kao **potproces** (eng. subprocess). Svaki potproces ima svoju detaljniju strukturu koju se prikazuje na posebnome modelu. Primjerice, u takav potproces bi se trebala uvrstiti i aktivnost **Provjeriti zalihe** budući da je to radni korak koji se izvodi u skladištu prilikom pripreme robe za otpremu. Dodatno, skretnica **Na zalihi?** također bi trebala postati dio tog potprocesa. O tome ćemo detaljnije govoriti u sljedećim poglavljima.

## Zadaci za Vježbu 1

Temeljem sljedećih opisa poslovnih procesa i do sada obrađene BPMN notacije, **izmodelirajte poslovne procese u alatu po vlastitom izboru**.

Svaki od poslovnih procesa treba sadržavati **nekoliko aktivnosti, ekskluzivne skretnice, polje i više staza**.

Modele exportajte u **png** formatu ili napravite screenshot, zippajte zajedno datoteke (3) i učitajte rješenja na **Merlin**.

Slobodno dodajte napomenu ako želite dobiti povratnu informaciju za vaša rješenja. Komunikacija se odvija putem **Google Chata**.

### 1. Proces obrade reklamacije

Proces započinje kada kupac podnese reklamaciju za proizvod kupljen u trgovini informatičke opreme. Prodavač zaprimlja reklamaciju i provjerava je li priložen račun. Ukoliko račun nije priložen, reklamacija se odbija. Ako je račun priložen, servisni tim procjenjuje opravdanost reklamacije unutar 30 dana od kupnje. U slučaju da reklamacija nije opravdana, ona se odbija.

Ako je reklamacija utemeljena, prodavač proslijeđuje proizvod servisnom timu koji potom procjenjuje je li proizvod moguće popraviti. Ako popravak nije moguć, kupcu se izdaje novi proizvod i proces je završen. Ukoliko je popravak izvediv, servisni tim popravlja proizvod i vraća ga kupcu.

## **2. Proces najma vozila**

Proces najma vozila započinje kada klijent putem web stranice rent-a-car agencije zatraži uslugu. Ispunjavajući online formu za najam, klijent unosi potrebne podatke i šalje zahtjev na obradu.

Nakon toga, administrator pregledava pristigli zahtjev i provjerava točnost unesenih podataka. Ako podaci nisu ispravni, zahtjev se odbija, čime proces završava. No, ukoliko su svi podaci ispravni, administrator proslijeđuje zahtjev timu zaduženom za upravljanje voznim parkom ("Fleet Management"). Oni provjeravaju dostupnost traženog vozila za navedeni datum.

Ako željeno vozilo nije dostupno, zahtjev se odbija, no u slučaju da vozilo jest na raspolaganju, tim šalje potvrdu administratoru. Administrator potom finalizira rezervaciju i šalje klijentu predračun. Po primitku uplate predračuna, rezervacija postaje službena te klijent prima potvrdu o najmu, čime se proces uspješno završava.

## **3. Proces automatizirane korisničke podrške**

Kompanija koja pruža SaaS usluge odlučila je unaprijediti korisničku podršku automatizacijom pomoću AI chatbota. Proces započinje kada klijent podnese novi zahtjev za podršku.

AI chatbot prima zahtjev, a zatim analizira bazu znanja koristeći napredne AI algoritme kako bi pronašao relevantan odgovor. Ukoliko chatbot pronađe prikladan odgovor, šalje ga klijentu, koji potom procjenjuje je li odgovor bio koristan i rješio njegov problem. Ako nije, chatbot nudi klijentu opciju da razgovara s pravim agentom.

Ako klijent odbije tu opciju, proces se završava. U slučaju da prihvati, razgovor s agentom započinje, a agent pruža dodatnu pomoć u rješavanju problema. Proces završava na jedan od dva načina: problem je uspješno riješen ili klijent i dalje nije zadovoljan ponuđenim rješenjem.

---

# Upravljanje poslovnim procesima (UPP)

**Nositelj:** izv. prof. dr. sc. Darko Etinger

**Asistent:** Luka Blašković, mag. inf.

**Ustanova:** Sveučilište Jurja Dobrile u Puli, Fakultet informatike u Puli



Fakultet informatike u Puli

## (2) Dopunski atributi elemenata modela procesa

#2

UPP

Elementi modela poslovnog procesa imaju svoje prepoznatljive simbole koji se vide na grafičkom prikazu. Simbol svojim oblikom određuje o kakvu je objektu riječ i što je njegova temeljna funkcija. Osim toga, svakom elementu mogu se pridružiti tekstualni, brojčani ili logički podaci koji pobliže opisuju ponašanje određenog objekta u složenome modelu procesa. U ovoj skripti proširit ćemo osnovne objekte koje smo naučili zadnji put i vidjeti koje dodatne informacije možemo definirati unutar njih.

Posljednje ažurirano: 7.11.2024.

### Sadržaj

- [Upravljanje poslovnim procesima \(UPP\)](#)
- [\(2\) Dopunski atributi elemenata modela procesa](#)
  - [Sadržaj](#)
- [1. Uvod](#)
- [2. Nadogradnja poslovnog procesa PRODATI ROBU](#)
- [2.1 Osnovne vrste događaja](#)
- [2.2 Osnove vrste aktivnosti](#)
  - [2.2.1 Radni korak \(eng. Task\)](#)
  - [2.2.2. Potproces \(eng. Subprocess\)](#)
  - [2.2.3 Kako ispravno koristiti send i receive aktivnosti?](#)
- [Zadaci za Vježbu 2](#)
  - [1. Potvrda narudžbe u web trgovini](#)

- [2. Implementacija softvera](#)

# 1. Uvod

Modeliranje poslovnih procesa širok je pojam koji obuhvaća izradu različitih vrsta modela za različite namjene. U literaturi ima više klasifikacija modela poslovnih procesa po različitim kriterijima. U pravilu možemo definirati 3 kriterija za klasifikaciju modela poslovnih procesa:

- Razina detaljnosti:** model procesa može biti opisni, analitički ili izvršivi (eng. *executable*)
- Faze razvoja:** model se može odnositi na postojeći proces (As is) ili na budući optimizirani proces (To Be) koji tek treba uspostaviti
- Pretežiti korisnici:** model može biti namijenjen poslovnim stručnjacima (među kojima je i menadžment), informatičarima ili korisnicima koji su izvan struke

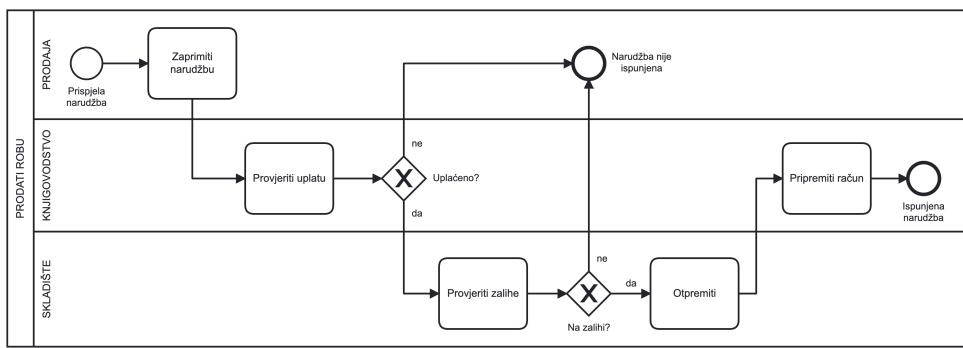
Rekli smo da su objekti toka podataka (eng. *data flow objects*) osnovni elementi modela poslovnog procesa i u pravilu ih možemo podijeliti na **događaje, aktivnosti i skretnice**. To su osnovni elementi svakog modela poslovnog procesa. Ako neki model nema ta tri elementa, onda ga i ne možemo smatrati modelom procesa u smislu definicije dane u prošloj skripti.

Prema normi BPMN 2.0, formalno je moguće da model procesa nema nijedne skretnice, ali to onda znači da je više aktivnosti (ako ih ima) povezano slijednim tokom, odnosno da aktivnosti slijede, jedna iza druge bez ikakvih uvjeta i zastoja. Tada model nije ništa drugo nego popis poslova koje treba obaviti, redom kako su navedeni u popisu.

Također, moguće je da model procesa nema nacrtan nijedan događaj, ali se tada podrazumijeva da postoji jedan početni i jedan završni događaj.

## 2. Nadogradnja poslovnog procesa **PRODATI ROBU**

Prisjetimo se poslovnog procesa **PRODATI ROBU** koji smo modelirali u prošloj skripti.



Slika 1. Poslovni proces **PRODATI ROBU** koji smo definirali u prošloj skripti

Kroz ovaj proces naučili smo definirati spomenute osnovne elemente. Model smo nadogradili i dodali smo **polje** i dodatne **staze** kojima smo definirali aktore koji sudjeluju u procesu.

Za početak, idemo nadograditi model različitim vrstama **događaja**.

## 2.1 Osnovne vrste događaja

---

Rekli smo da događaje definiramo **kružnicama** koje su povezane s tokom. Događaji su važni jer označavaju početak ili kraj neke aktivnosti ili procesa. Definirali smo dva osnovna tipa događaja:

- **Početni događaj** (eng. *Start Event*)
- **Završni događaj** (eng. *End Event*)

**Kružnica** (za početni i završni događaj)

Početni događaj je **istovjetan početku prve aktivnosti**, dok završni događaj **nastaje kada završi zadnja aktivnost**.

U interpretaciji nekog dijagrama poslovnog procesa, **nastupanje nekog događaja naziva se okidanje** jer se pojmom događaja omogućuje izvođenje aktivnosti koja slijedi poslije njega.

Pri određivanju nekog događaja uvijek treba razmotri što je bio **uzrok** njegova nastanka i gdje je nastao te kakva je **posljedica** njegova pojavljivanja i gdje se ona vidi.

Evo nekoliko različitih primjera događaja:

- referent prodaje primio je e-mail sa zahtjevom za ponudu. Događaj je ovdje **trenutak primitka zahtjeva**
- dostignuta je neka vremenska točka u kojoj se mora obaviti aktivnost, npr. **istek roka za dostavu ponude**. Ovdje je događaj **istek roka**
- odzvonilo je 6:30 ujutro, a to je **trenutak početka radnog vremena**. Ovdje je događaj **početak radnog vremena**
- isteklo je neko vrijeme koje smo čekali, npr. postavili smo *timer* na 60 minuta za pečenje kolača. Ovdje je događaj **istek vremena pečenja**
- **roba** je stigla na skladište. Ovdje je događaj **dostava robe**
- ostvaren je neki uvjet koji smo postavili, npr. **kupac je potvrdio narudžbu ili kupac je platio račun**. Ovdje su događaji **potvrda narudžbe i plaćanje računa**
- pojavila se pogreška tijekom provedbe neke aktivnosti. Na primjer, otkazao je stroj u proizvodnoj liniji treba se **pojaviti događaj kvara stroja**
- itd.

U BPMN notaciji događaja ima puno, a možemo ih podijeliti na više načina. Mi smo do sad spomenuli samo početne i završne događaje, ali postoje i **međudogađaji** (eng. *Intermediate events*) koji se javljaju tijekom izvođenja aktivnosti. Međudogađaji su važni jer omogućuju da se aktivnost prekine i nastavi kasnije, ili da se aktivnost ponovi, ili pak da se aktivnost prekine i prebací na neku drugu aktivnost.

**Međudogađaje** (eng. *Intermediate events*) definiramo kružnicom s unutarnjom koncentričnom kružnicom.



Početne i završne događaje koje smo do sad definirali bili su **neoznačeni**, u literaturi ih još nazivamo i **none** ili **blank** events.

U BPMN notaciji postoji više vrsta početnih i završnih događaja ovisno o njihovoj ulozi u procesu.

Camunda 8 podržava sljedeće vrste događaja:

- **Neoznačeni događaji** (eng. *None Event*)
- **Obavijest** (eng. *Message Event*)
- **Mjerač vremena** (eng. *Timer Event*)
- **Greška** (eng. *Error Event*)
- **Eskalacija** (eng. *Escalation Event*)
- **Ukidanje ili opoziv** (eng. *Terminate Event*)
- **Kompenzacija** (eng. *Compensation Event*)
- **Signal** (eng. *Signal Event*)
- **Priključna točka** (eng. *Link Event*)

Međutim, u BPMN notaciji [još vrsta događaja](#).

Dva koja ćemo često koristiti su **obavijest** (eng. *Message event*) i **mjerač vremena** (eng. *Timer event*).



Dakle, odabiremo početni događaj i dodajemo mu oznaku (npr. `početak prodaje`) te dodajemo vrstu događaja. *Message* event je označen pismom unutar kružnice, dok je *Timer* event označen satom unutar kružnice.

### Početni Message i Timer događaji

Završni događaj također možemo označiti **Message eventom** koji onda označava da je proces završio i moramo nekoga obavijestiti o tome, primjerice šaljemo e-mail s potvrdom ili računom. Varijanta završnog **Message eventa** izgleda ovako:



## Završni Message događaj

Kako interpretirati ove događaje?

Početne događaje još nazivamo **prijamnim** (eng. *catch event*), budući da reagiraju na neki vanjski događaj koji se dogodio.

- *primjer:* **primanje e-maila** s narudžbom

Završne događaje još nazivamo **predajnim** (eng. *throw event*) budući da oni indiciraju da se nešto dogodilo i da je potrebno nešto poduzeti.

- *primjer:* **slanje e-maila** s potvrdom

Postoje i **međudogađaji obavijesti** (eng. *Intermediate Message Events*) koji onda mogu biti **catch** ili **throw** ovisno o tome reagiraju li na neki vanjski događaj ili pak generiraju neki vanjski događaj.

Varijanta prijamnog međudogađaja obavijesti izgleda ovako, npr. **primanje e-maila s obavijesti usred procesa:**



### Prijamni Međudogađaj obavijesti (eng. *Intermediate catch message event*)

Postoji i varijanta **predajnog** međudogađaja, npr. **slanje poruke usred procesa, najčešće u svrhu pokretanja nekog drugog procesa:**

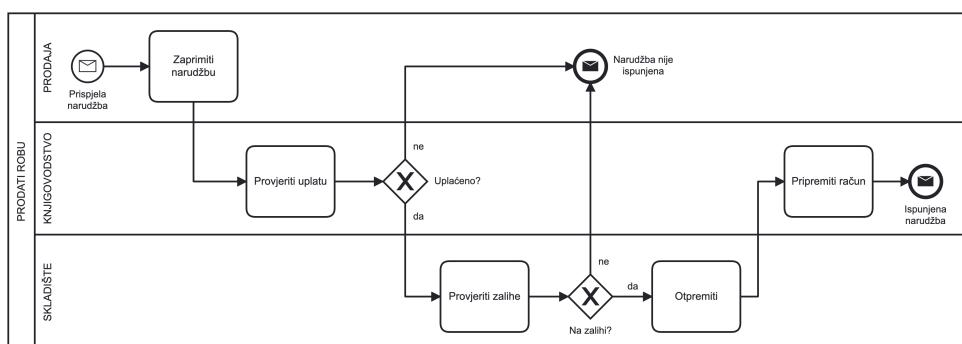


### Predajni Međudogađaj obavijesti (eng. *Intermediate throw message event*)

Idemo nadograditi naš proces PRODATI ROBU ažuriranjem događaja.

Kako imamo događaj **Prispjela narudžba**, možemo zaključiti da je vjerojatno riječ o **Message eventu** budući da je prispjela ili putem e-maila ili putem nekog drugog sustava koji šalje poruke na određeni komunikacijski kanal (ne mora biti e-mail).

Također, imamo 2 završna događaja, jedan za uspješno **ispunjenu narudžbu**, drugi za **neuspješno ispunjenu narudžbu**. Oba završna događaja možemo označiti **Message eventima** jer će se u oba slučaja poslati e-mail ili obavijestiti klijent na neki drugi način.

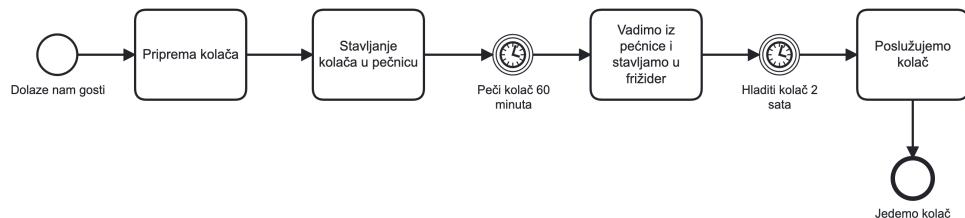


Slika 2. Poslovni proces PRODATI ROBU s dodanim Message eventima

Prije nego se prebacimo na vrste aktivnosti, pogledajmo još jedan primjer korištenja **Timer eventa**.

**Timer eventi** mogu biti korisni za definiranje vremenskih ograničenja ili rokova, zamislimo jednostavni proces **Priprema kolača**. Zamislimo da radimo kolač koji se peče 60 minuta. Nakon tog vremena, moramo ga izvaditi iz pećnice i pustiti da se ohladi u frižideru barem 2 sata. Nakon toga, kolač možemo poslužiti gostima.

Vidimo da u definiciji procesa imamo vremenska ograničenja, a budući da se ona nalaze za vrijeme izvođenja aktivnosti, možemo koristiti **Intermediate Timer evente**.



Slika 3. Proces PRIPREMA KOLAČA s dodanim Message eventima

## 2.2 Osnove vrste aktivnosti

Osim različitih vrsta događaja, imamo i različite vrste aktivnosti.

**Aktivnost** je prema normi BPMN opći (generički) pojam koji se koristi za opis svakog rada izvedenog u poslovnom procesu. Osnovni je simbol pravokutnik sa zaobljenim uglovima, unutar kojeg je opisan **naziv posla koji treba obaviti**.

Za izvođenje aktivnosti potrebni su neki **resursi i vrijeme**. Resursi se uzimaju iz organizacije ili iz okruženja, a vrijeme se odnosi na izvođenje ili trajanje aktivnosti.

Dakle, svaka **aktivnost** označava neku elementarnu radnju koja se u razmatranom dijagramu poslovnih procesa **ne raščlanjuje** i naziva se često i **radni korak**.

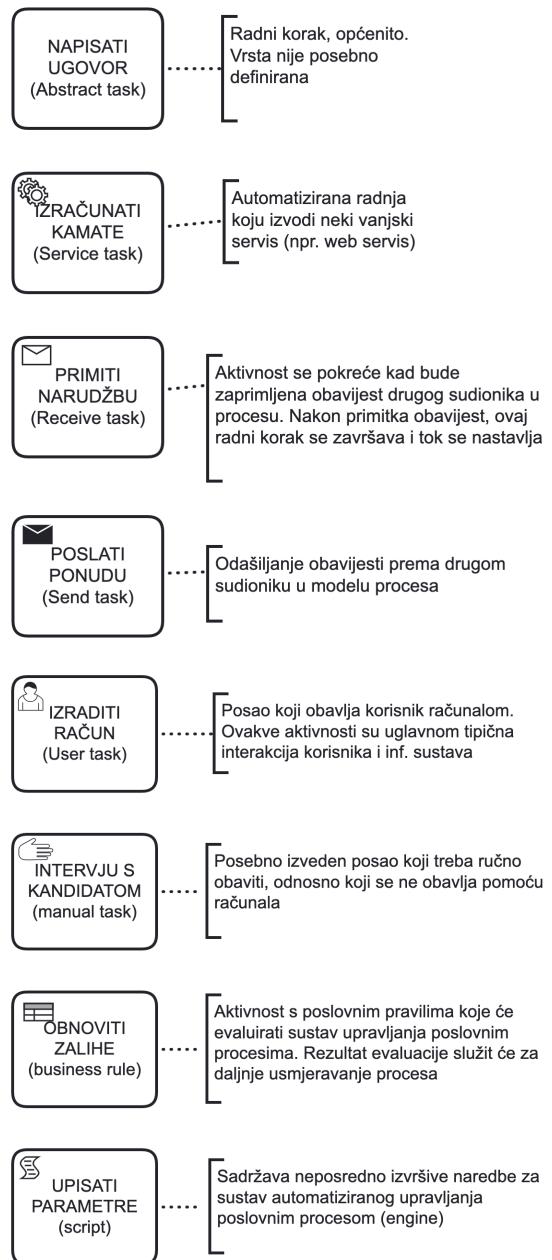
Složene aktivnosti u BPMN notaciji često definiramo kao **potprocese** (eng. *Subprocess*).

### 2.2.1 Radni korak (eng. Task)

Radnih koraka u BPMN notaciji ima više vrsta, mi smo do sada koristili samo opći (eng. *abstract*) radni korak. U BPMN notaciji postoji više vrsta radnih koraka:

- **Opći radni korak** (eng. *abstract*)
- **Servisni radni korak ili servis** (eng. *service*)
- **Prijamni radni korak** (eng. *receive*)
- **Otpremni radni korak** (eng. *send*)
- **Korisnički radni korak** (eng. *user*)
- **Ručni radni korak** (eng. *manual*)
- **Poslovno pravilo ili pravilo** (eng. *business rule*)
- **Naputak** (eng. *script*)

U pravilu ćemo koristiti sve radne korake osim naputka, jer je naputak vrlo specifičan i koristi se samo u određenim situacijama. U nastavku ćemo objasniti svaku vrstu radnog koraka:



Ono što vas za sad može zbunjivati je razlika između **događaja obavijesti** (*eng. message events*) i prijamnih i otpremnih radnih koraka (*eng. send and receive tasks*). Razlike mogu biti zbunjujuće i zato ćemo ih izvježbati u sljedećoj skripti detaljno, međutim sada trebate zapamtiti da su događaji tzv. **okidači** koji pokreću neki proces ili vanjski potproces, ili pak završavaju određeni proces. Međudogađaji se često koriste kao pasivni elementi koji čekaju na neki događaj da se dogodi, dok radni koraci definiraju slijed aktivnosti koji se izvršava i često su vidljivi (*eng. trackable*) korisniku ili sustavu.

Vrsta radnje obavezno se navodi u analitičkom i izvršivom modelu procesa radi točnog opisa ponašanja aktivnosti u procesnim aplikacijama, gdje će se za svaku vrstu radnoga koraka generirati specifična programska procedura.

Idemo vidjeti kako možemo nadograditi naš proces PRODATI ROBU s različitim vrstama radnih koraka.

Obzirom da poslovni proces PRODATI ROBU započinje **message eventom Prispjela narudžba** koji predstavlja okidač našeg procesa, možemo preimenovati aktivnost koja slijedi nakon toga iz **Zaprimiti narudžbu** u **Obrada narudžbe** budući da je to radnja koja nam pobliže opisuje što se događa u toj aktivnosti.

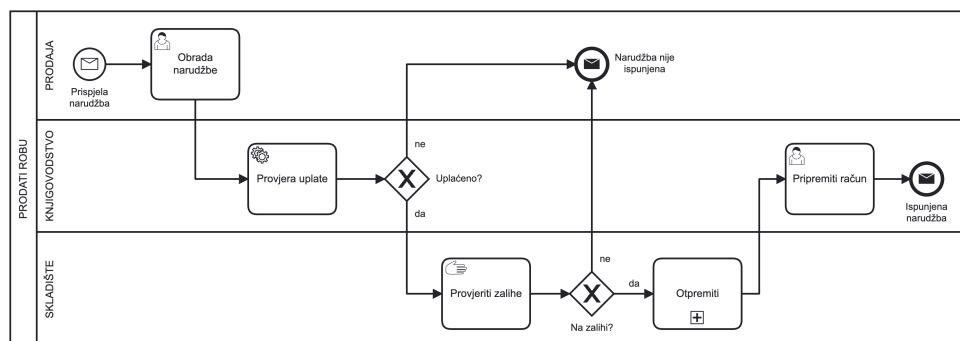
Aktivnost **Obrada narudžbe** možemo definirati kao **korisnički radni korak** budući da je to radnja koju obavlja djelatnik poslovnice ili referent prodaje **na računalu kroz neki informacijski sustav**

Aktivnost **Provjeriti uplatu** možemo također definirati kao **korisnički radni korak** jer se radi o radnji koju obavlja djelatnik poslovnice ili referent prodaje **na računalu kroz neki informacijski sustav**, ali ako je provjera automatska, odnosno ako se uplata provjerava automatski u računovodstvu nakon obrade narudžbe, onda možemo koristiti **servisni radni korak**.

Provjeriti zalihe možemo definirati kao **servisni radni korak** ako se provjera zaliha radi automatski, međutim kako je to neuobičajeno i manje vjerojatno za naš zadatak, možemo ga definirati kao **korisnički radni korak** ako korisnik provjerava zalihe kroz neki informacijski sustav, ili kao **ručni radni korak** ako korisnik provjerava zalihe fizički, npr. odlaskom u skladište.

Otpremiti robu možemo definirati također kao **ručni radni korak** jer se radi o fizičkoj radnji, odnosno odlasku u skladište i pakiranju robe, međutim uskoro ćete vidjeti kako preciznije definirati ovu aktivnost **kroz potproces**.

Pripremiti račun možemo definirati kao **korisnički radni korak** jer se radi o radnji koju obavlja djelatnik iz računovodstva kroz neki IS.



Slika 4. Poslovni proces **PRODATI ROBU** s različitim vrstama radnih koraka

## 2.2.2. Potproces (eng. Subprocess)

Potproces je posebna vrsta složene aktivnosti koju **ne možemo nazvati radnim korakom** budući da on nije elementarna radnja, već je **složenija radnja koja se sastoji od više manjih radnji**. Potproces se koristi **za grupiranje i organizaciju** aktivnosti koje se **ponavljaju** u procesu ili su **složene i ne mogu se opisati jednostavno**.

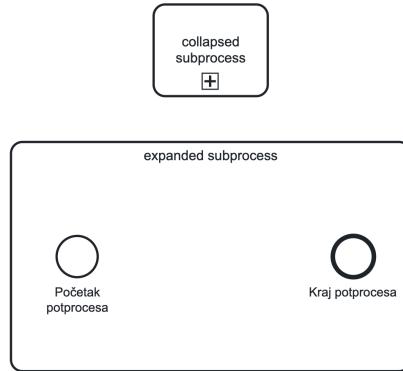
Primjer potprocesa iz našeg poslovnog procesa PRODATI ROBU je **Otpremiti**.

Ako usporedimo tu aktivnost, s npr. aktivnošću **Provjeriti uplatu** ili **Pripremiti račun**, vidimo da je **Otpremiti robu** složenija radnja koju trebamo malo preciznije definirati. Za to koristimo **potproces**.

Potproces je doslovno **proces unutar procesa** te može biti **sklopljeni** (eng. collapsed) ili **prošireni** (eng. expanded).

- **Sklopljeni potproces** je onaj koji je skriven, odnosno ne vidimo unutrašnjost potprocesa, dok je

- **Prošireni potproces** onaj koji je vidljiv i unutar kojeg možemo vidjeti sve aktivnosti (stvarni proces).



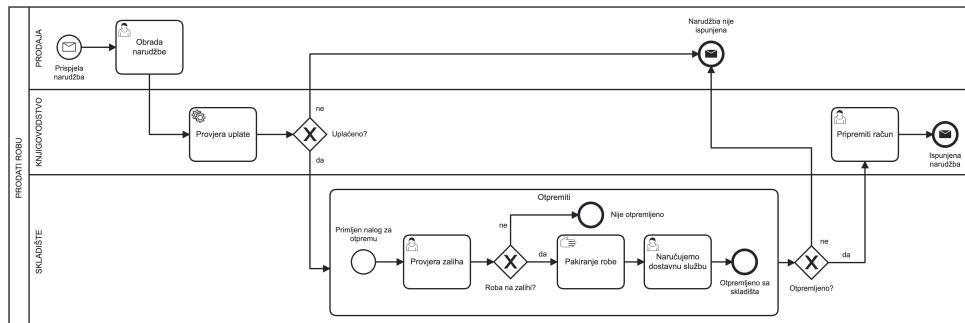
Prošireni potproces mora imati **početni i završni događaj** koji označavaju početak i kraj potprocesa!

Kako ćemo definirati potproces **Otpremiti**?

Proces započinje **primitkom nalogu za otpremu** u skladište, nakon čega je **potrebno provjeriti zalihe**.

Ako je roba na zalihamu, možemo spakirati robu i dogovoriti prijevoz s dostavnom službom. Nakon toga, šaljemo robu i proces završava otpremom robe sa skladišta. Ako roba nije na zalihi, proces završava bez otpreme.

Nakon potprocesa, vraćamo se na glavni proces kroz XOR skretnicu "Otpremljeno?". Ako je otpremljeno, pripremamo račun i završavamo s procesom. Ako nije, narudžba nije ispunjena.



Slika 5. Poslovni proces PRODATI ROBU s dodanim potprocesom "Otpremiti"

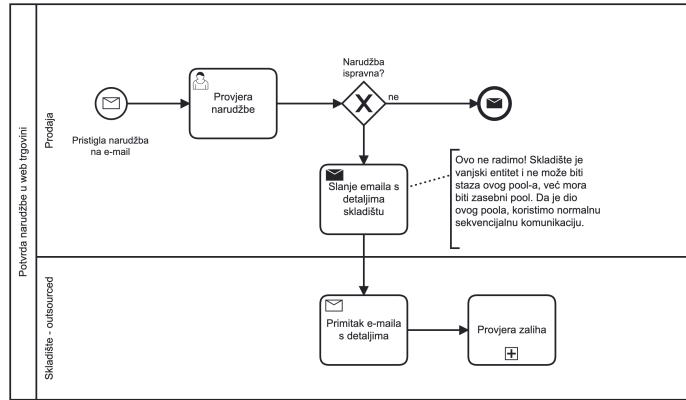
## 2.2.3 Kako ispravno koristiti **send** i **receive** aktivnosti?

Prijamni (*eng. receive*) i otpremni (*eng. send*) radni koraci koriste se za komunikaciju između više procesa, procesa i vanjskog sustava ili procesa i vanjske jedinice. **Česta greška** je koristiti ove aktivnosti unutar jednog procesa između dviju staza (npr. dvaju odjela u tvrtki). U tom slučaju komunikacijske veze potrebno je definirati kroz slijedne tokove.

Recimo da imamo primjer procesa potvrde web trgovine koja nema svoje skladište, već koristi vanjsko skladište za otpremu robe. Recimo da skladište posluje s mnogo trgovina i komunikacija se odvija prvenstveno putem e-maila. U tom slučaju, koristimo **send** i **receive** između ta entiteta.

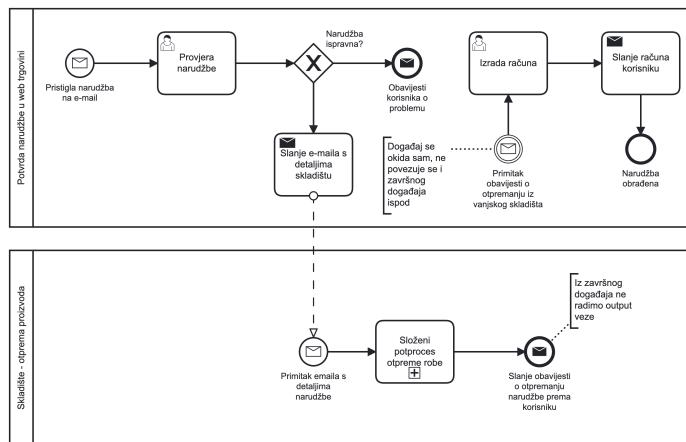
Međutim, kako se radi o udaljenim entitetima, ne želimo ih definirati unutar istog polja.

Dakle, sljedeće je pogrešno:



Da se radi o nekoj malo trgovini koja ima svoje malo interno skladište, onda bi to mogli ovako prikazati međutim umjesto **send** i **receive** koristili bismo **obične sekvensijalne veze**.

S druge strane, ako se radi o **vanjskom skladištu**, onda je to ispravno **pokazati na sljedeći način**:



### Zapamti!

- u pravilu želimo koristiti polje i staze za komunikaciju između entiteta unutar jednog procesa (npr. odjeli unutar iste tvrtke)
- dva ili više polja koristimo kada naši entiteti nisu dio iste cjeline, već su udaljeni i komuniciraju prvenstveno kroz komunikacijske kanale (npr. e-mail) ili nemaju povezani interni informacijski sustav. Npr. tvrtka koja koristi vanjsko skladište za otpremu robe.
- ako definiramo **end message event** u potprocesu isopd, ne smijemo iz njega definirati nikakve vanjske veze. Kako bismo prikazali čekanje na poruku u entitetu web trgovine, koristimo **intermediate message event** koji se okida kada se pošalje poruka iz vanjskog skladišta.

Ove detalje ćemo detaljno obraditi na budućim vježbama.

## Zadaci za Vježbu 2

Temeljem sljedećih opisa poslovnih procesa i do sada obrađene BPMN notacije, **izmodelirajte poslovne procese u alatu po vlastitom izboru**.

Modele exportajte u **png** formatu ili napravite screenshot, zippajte zajedno datoteke (2) i učitajte rješenja na **Merlin**.

Slobodno dodajte napomenu ako želite dobiti povratnu informaciju za vaša rješenja. Komunikacija se odvija putem **Google Chata**.

# **1. Potvrda narudžbe u web trgovini**

---

Proces započinje primitkom automatiziranog e-maila s podacima o naručenoj robi. Narudžbu preuzima referent prodaje koji prvo provjeri u sustavu je li narudžba ispravna. Nakon provjere, zaposlenik u malom skladištu tvrtke provjerava zalihe robe. Ako roba nije na zalihi, referent prodaje šalje e-mail vanjskom skladištu s kojim posluju i traži dostavu robe. Ako roba je na zalihi, zaposlenik u skladištu pakira robu, naručuje pickup dostavne službe, a referent prodaje priprema račun i šalje ga kupcu. Tu proces završava.

Vanjsko skladište zaprima poruku preko e-maila i započinju složeni proces otpreme robe. Složeni proces završava ako robe nema i o tome se obavještava referent prodaje te web trgovine. Ako roba postoji, skladište ju pakira i putem dostavne službe šalje trgovini u interno skladište. Tada zaposlenik u internom skladištu web trgovine prepakira robu u vlastitu ambalažu, naručuje pickup dostavne službe, a referent prodaje priprema račun i šalje ga kupcu. Tu proces završava.

# **2. Implementacija softvera**

---

Tvrtka UPPTech odlučila je implementirati novi softver za upravljanje poslovnim procesima. Tvrtka se sastoji prvenstveno od Management tima koji donosi poslovne odluke i malog internog IT tima koji održava informacijski sustav tvrtke. Proces započinje kad management tim ustanovi potrebu za implementacijom novog softvera. IT tim provjerava dostupne softvere na tržištu i donosi odluku postoji li softver koji zadovoljava potrebe.

Ako postoji, management tim dogovara kupnju softvera, a IT tim instalira softver na server tvrtke i tu proces završava.

Ako ne postoji, IT tim odlučuje outsourcati razvoj novog softvera i kontaktira vanjsku tvrtku CrazyTech. CrazyTech zaprima ponudu putem emaila i procjenjuje troškove razvoja. O tome obavještavaju management tim UPPTecha, a ako je ponuda prihvaćena, ovi im odgovaraju i tu započinje složeni proces razvoja softverskog rješenja. Ako je ponuda odbijena, tu proces završava.

Složeni proces razvoja softverskog rješenja započinje s definiranjem zahtjeva, dizajnom i razvojem softvera. Nakon razvoja, softver se testira i ako testiranje prođe u redu, softver se deploys na server tvrtke UPPTech i tu završava proces. Ako testiranje ne prođe u redu, vraća se na aktivnost razvoja.

---

# Upravljanje poslovnim procesima (UPP)

**Nositelj:** izv. prof. dr. sc. Darko Etinger

**Asistent:** Luka Blašković, mag. inf.

**Ustanova:** Sveučilište Jurja Dobrile u Puli, Fakultet informatike u Puli



Fakultet informatike u Puli

## (3) Složena grananja

#3

UPP

Skretnice upravljačke slijedom izvođenja aktivnosti u procesu. Do sad smo vidjeli kako možemo koristiti jednostavne ekskluzivne skretnice za odabir između dvije ili više opcija prilikom izvođenja poslovnog procesa. Ključna stvar je što se uvijek odabire samo jedna opcija i to samo ona koja je zadovoljena danim uvjetom. U ovom poglavlju, upoznat ćemo se s drugim oblicima skretnicama te vidjeti kako definirati "čekanje" na rezultat izvođenja aktivnosti uvjetovanih kroz više skretnica.

**Posljednje ažurirano:** 20.11.2024.

### Sadržaj

- [Upravljanje poslovnim procesima \(UPP\)](#)
- [\(3\) Složena grananja](#)
  - [Sadržaj](#)
- [1. Ekskluzivna \(eng. Exclusive\) skretnica](#)
  - [1.1. XOR skretnica spajanja \(eng. XOR merge/join\)](#)
- [2. Paralelna \(eng. Parallel\) skretnica](#)
  - [2.1 AND skretnica spajanja \(eng. AND merge/join\)](#)
- [3. Inkluzivna \(eng. Inclusive\) skretnica](#)
  - [3.1 OR skretnica spajanja \(eng. OR merge/join\)](#)
- [4. Ukratko, kada koristiti koju skretnicu?](#)
- [Zadaci za Vježbu 3](#)
  - [1. Wolt - dostava hrane](#)

# 1. Ekskluzivna (eng. Exclusive) skretnica

**Ekskluzivnu (XOR) skretnicu** (eng. *Exclusive gateway*) već ste upoznali kroz prethodne primjere. Ona se koristi za odabir jedne između više opcija, gdje se uvijek odabire samo maksimalno jedna opcija. Ukoliko je zadovoljen predikat (rezultat poslovne aktivnosti i/ili poslovna odluka) definiran na skretnici, izvršava se **samo jedan sljedni tok** dok se ostali tokovi zanemaruju.



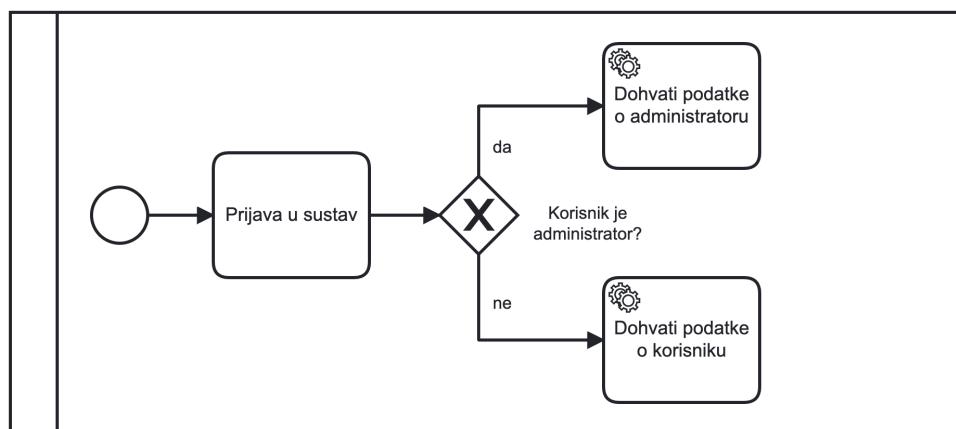
Ekskluzivna skretnica se definira rombom, s oznakom X

*Primjer ekskluzivne skretnice:* Nakon što se korisnik prijavi na sustav, želimo provjeriti je li korisnik administrator ili običan korisnik. Ukoliko je korisnik administrator, želimo mu prikazati dodatne opcije koje običan korisnik nema. Ukoliko je običan korisnik, želimo mu prikazati opcije koje su mu dostupne.

Samim time, na XOR skretnicu ćemo upisati uvjet `je administrator?` ili `korisnik je administrator?`. Iz skretnice definiramo **dva ili više toka** koristeći sekvencijalne tokove (linije) prema aktivnostima koje želimo izvršiti. **XOR skretnica je u pravilu skretnica uvjetovana podacima koje sa sobom donosi instanca procesa** (npr. korisnička uloga, tip korisnika, itd.).

Ovaj način korištenja skretnica zovemo **grananje** (eng. *splitting/branching*).

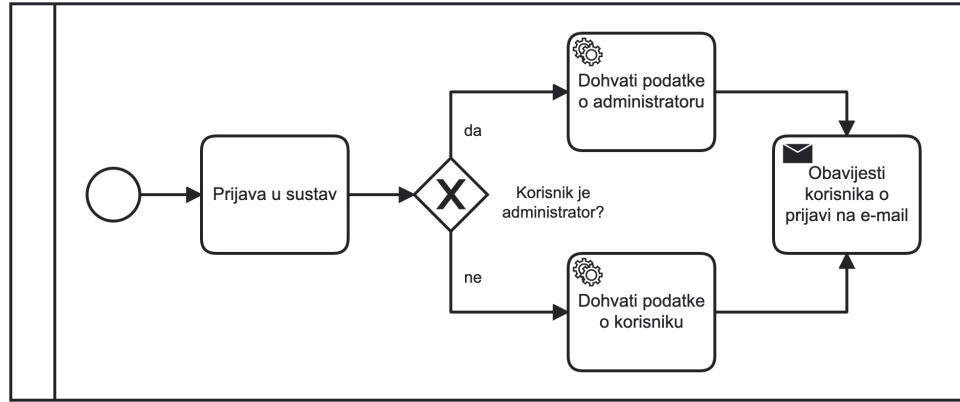
**Skretnica grananja** (ili samo grananje) ima jedan ulazni sljedni tok ili ulazni put (obično usmjeren u lijevi ili gornji vrh romba), a više izlaznih sljedova (iz desnog ili donjeg vrha romba).



Slika 1: Primjer korištenja XOR skretnice za odabir između vrste korisnika

Međutim, što ako je sljedeća aktivnost koja slijedi nakon odabira jednaka za oba korisnika? U tom slučaju, ne želimo ponavljati istu aktivnost za svaku opciju, već želimo samo strelice toka preusmjeriti u ponavljajuću aktivnost.

*Primjer:* Nakon prijave i dohvata podataka o korisniku, želimo poslati korisniku obavijest o uspješnoj prijavi na e-mail.



Slika 2: Nakon odabране opcije, jednostavno proslijedujemo tok prema sljedećoj aktivnosti

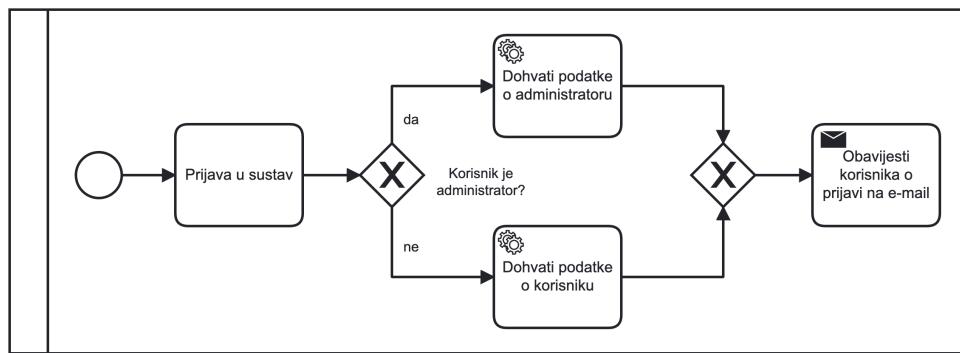
## 1.1. XOR skretnica spajanja (eng. XOR merge/join)

Osim grananja, koje smo do sad koristili, skretnice je moguće koristiti i za **spajanje** (eng. *merge/join*). Spajanje se koristi kada se više tokova treba spojiti u jedan, tj. kada se više tokova vraća u jedan tok. Drugim riječima, **skretnica spajanja ima više ulaznih slijednih tokova** (lijevo ili gore) i **jedan izlazni slijedni tok** ili izlazni put (desno ili dolje).

Iako nije potrebno, u svrhu čišćeg i preciznijeg prikaza moguće je koristiti XOR skretnicu spajanja **kako bi naglasili da se izlazni tokovi spajaju u jedan tok**, koji se nastavlja čim je zadovoljen uvjet.

Kod XOR skretnice, čak i kad je moguće da je više uvjeta istinito, **ona aktivnost koja prva završi će nastaviti tok** (budući da je XOR skretnica ekskluzivna - uvjeti se isključuju). Dakle, iako je moguće definirati inkluzivne aktivnosti pred ovu skretnicu, to nije poželjno.

Ako se vratimo na primjer iznad, XOR merge skretnicu jednostavno postavljamo prije same aktivnosti **i ne dodajemo joj naziv**.



Slika 3: Dodajemo XOR merge skretnicu u koju se "spajaju" svi izlazni tokovi XOR split skretnice

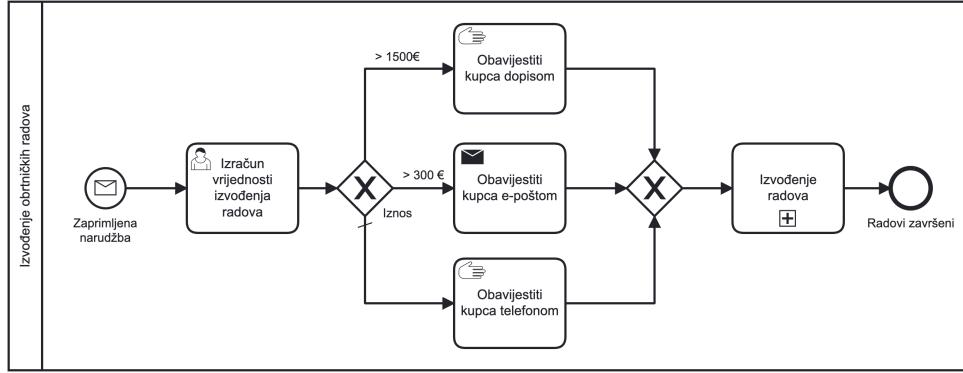
**XOR skretnicu spajanja** interpretirajte kao: "pričekaj ulaz barem jednog toka, a zatim nastavi dalje". U gornjem slučaju, pričekaj dohvati podataka o korisniku (bilo da se radi o administratoru ili korisniku) i nastavi dalje slanjem e-maila.

Rekli smo da se **XOR skretnicom može definirati i više od dva izlazna toka** (recimo kad predikat nije boolean oblika, odnosno rezultat nije Točno/Netočno). Takva sintaksa je dozvoljena i onda je **poželjno koristi ekvivalentnu XOR skretnicu spajanja**.

- za takve ekskluzivne skretnice (s 2 ili više uvjeta) **kažemo da su uvjetovane podacima**

*Primjer:* Vlasnik tvrtke za izvođenje obrtničkih radova dobiva narudžbu. Obrtnik će izračunati vrijednost radova i obavijestiti naručitelja **telefonom, mailom ili dopisom**, ovisno o vrijednosti, a tek onda izvesti naručeno.

- **XOR skretnica** za odabir načina obavještavanja (**telefon, mail, dopis**) ovisno o podacima (**vrijednost izvođenja radova**)
- **XOR merge skretnica** za spajanje toka nakon obavještavanja naručitelja i nastavak prema potprocesu izvođenja radova



Slika 4: XOR skretnica za odabir načina obavještavanja ovisno o vrijednosti radova

Definirali smo 3 moguća toka ovisno o iznosu:

- **Slanje dopisa:** za iznose veće od 1500 eura
- **Slanje e-maila:** za iznose veće od 300 eura
- **Obavještavanje telefonom:** za iznose manje od 300 eura

Logičke izraze (`> 1500 eur` i `> 300 eur`) smo zapisali na strelicama, a oznakom (`/`) smo definirali *defaultni* tok (za iznose manje od 300 eura).

Nakon obavještavanja, ovisno o uvjetu odabrat će se samo jedna aktivnost, a skretnica spajanja će **pričekati na ulazni tok jedne od aktivnosti i nastaviti dalje**.

## 2. Paralelna (eng. Parallel) skretnica

**Paralelna** (AND) skretnica (eng. *Parallel gateway*) koristi se za modeliranje situacija u kojoj se **tok procesa grananja dešava paralelno**, odnosno kada želimo definirati više aktivnosti ili tokova koji se izvršavaju "istovremeno" (paralelno). Paralelna skretnica omogućuje izvođenje aktivnosti na svim ulaznim i izlaznim putovima.

**Pri grananju** aktivira sve izlazne puteve budući da je provedena aktivnost prije skretnice.

**Pri spajanju** pokreće izlazni put tek kada su svi ulazni putevi završili.

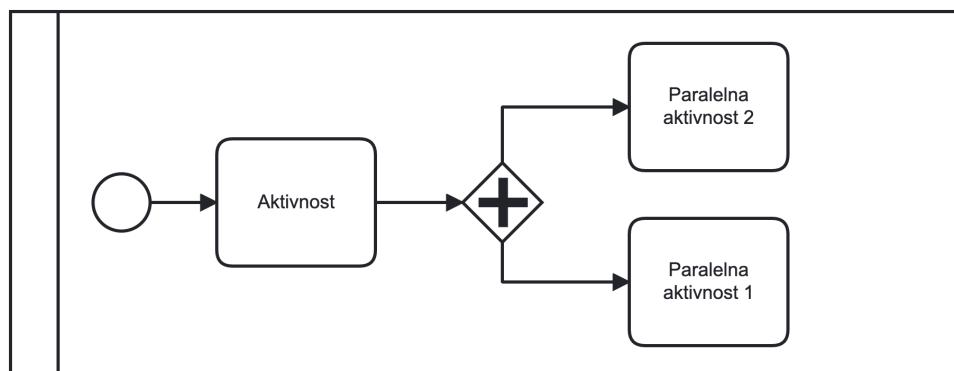


Paralelna skretnica se definira rombom, ali s oznakom +

Međutim, za razliku od XOR skretnice, kod paralelne skretnice **sve aktivnosti koje slijede nakon skretnice se izvršavaju**. Dakle, ova skretnica ekvivalentna je operatoru logičkog AND, odnosno **logičkoj konjukciji**.

Međutim, u stvarnim scenarijima, aktivnosti koje slijede nakon paralelne skretnice gotovo nikad nisu istog trajanja. Neke mogu trajati nekoliko minuta, više sati pa i nekoliko dana.

Ono što je ključno, jest da se sve aktivnosti **započinju izvršavati istovremeno**, ali sekvencijalni tok koji nastavlja izvršavanje procesa nastavlja **tek onda kada su sve aktivnosti završene**. Drugim riječima, moramo prikazati **čekanje na izvršavanje svih aktivnosti** koje slijede nakon paralelne (AND) skretnice.



Slika 5: Paralelna (AND) skretnica za definiranje paralelnog grananja

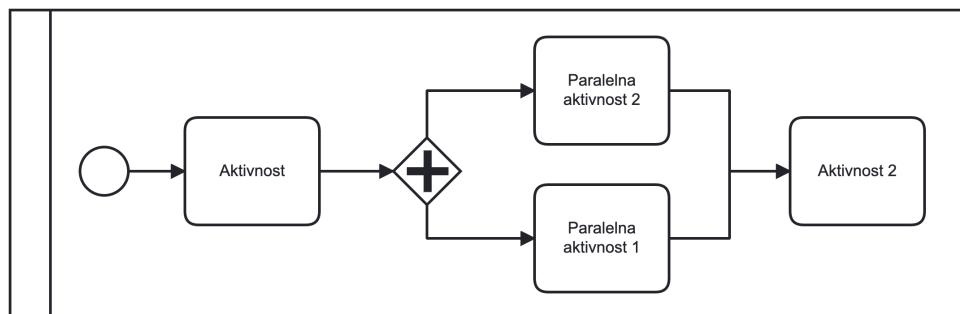
AND skretnice u pravilu **ne želimo imenovati** budući da se sve aktivnosti koje slijede izvršavaju paralelno, odnosno nema predikata kojim se uvjetuje izbor između aktivnosti.

U definiciji AND skretnice navedeno je da se aktivnosti izvršavaju paralelno, odnosno "istovremeno". Namjerno je napisano pod navodnim znakovima budući da se aktivnosti u stvarnosti ne izvršavaju sinkronizirano istovremeno, već pseudoparalelno, odnosno **konkurentno**. Što to znači? Više zadataka je pokrenuto istovremeno i u tijeku su njihova izvršavanja, ali se izvršavanje svakog zadatka odvija u vlastitom vremenskom okviru i **ne ovisi o izvršavanju drugih zadataka** (nije istovremeno s ostalima).

## 2.1 AND skretnica spajanja (eng. AND merge/join)

Kako ćemo definirati čekanje na izvršavanje svih aktivnosti?

**Česta pogreška** bila bi jednostavno povezivanje sekvenčjalnim tokom sve aktivnosti koje slijede nakon paralelne skretnice. To nije ispravno jer bi na taj način definirali da se aktivnosti izvršavaju sekvenčijalno, bez čekanja na izvršavanje svih.

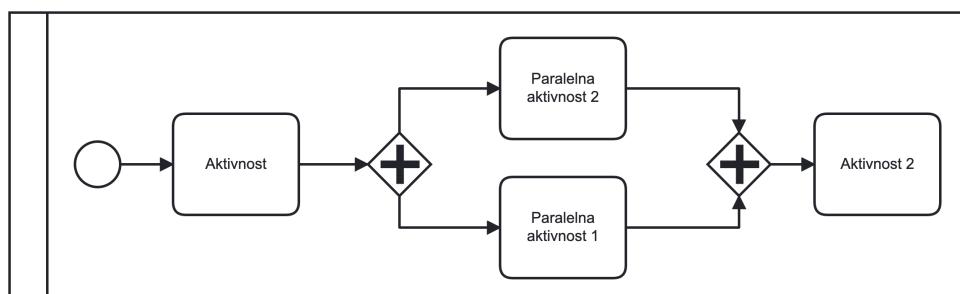


Slika 6: **Pogrešan način** povezivanja toka nakon paralelne skretnice grananja

Ono što ustvari moramo je definirati spajanje svih tokova kroz paralelnu skretnicu spajanja (eng. *parallel merge gateway*). Preciznije, **želimo prikazati čekanje na izvršavanje svih aktivnosti** kroz ekvivalentnu skretnicu spajanja.

Samim time, kod korištenja paralelnih skretnica za grupiranje, prema BPMN standardu, **obavezno je definirati ekvivalentnu paralelnu skretnicu spajanja**.

Ispravno je sljedeće:

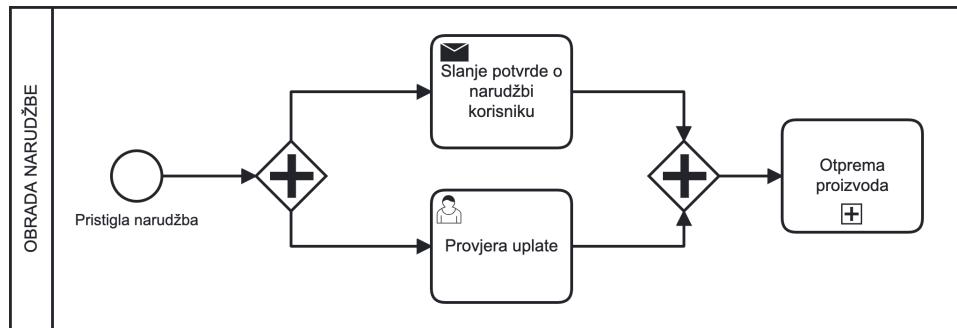


Slika 7: **Ispravan način** povezivanja toka nakon paralelne skretnice koristeći **AND merge skretnicu**

## Primjer korištenja paralelne skretnice:

Imamo web shop i želimo definirati slijed aktivnosti nakon što korisnik napravi narudžbu. Primjerice, jednom kad zaprimimo email s novom narudžbom, želimo poslati korisniku automatsku obavijest o zaprimljenoj narudžbi i paralelno provjeriti uplatu. Odnosno, potvrda se šalje automatski kroz neki servis koji smo integrirali, dok se provjera uplate obavlja kroz IT sustav.

Iskoristit ćemo AND skretnicu kako bi prikazali paralelno slanje obavijesti (potvrde) i provjeru uplate u IT sustavu.



Slika 8: Primjer korištenja AND skretnice za paralelno slanje obavijesti i provjeru uplate

Ovdje paralelno izvršavamo 2 zadatka (premda ih može biti proizvoljan broj, **ne želimo prelaziti brojku od 4-5 radi čitljivosti modela**):

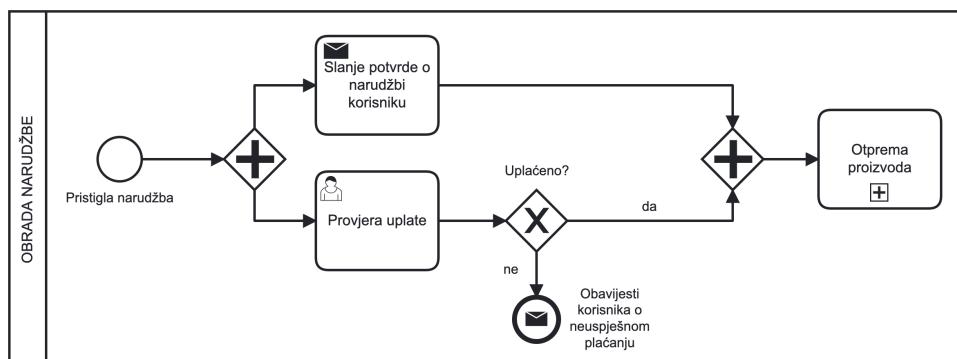
1. **Slanje potvrde o narudžbi korisniku** (*Send Task*)
2. **Provjera uplate** (*User Task*)

Koliko će se izvršavati svaka aktivnost?

- **Slanje potvrde o narudžbi korisniku** - automatski, bez čekanja ako je sustav ispravan, prosječno 4-5 sekundi
- **Provjera uplate** - ovisno o načinu plaćanja, može trajati odmah (ako je plaćanje karticom) do nekoliko dana (npr. ako je plaćanje uplatom na račun)

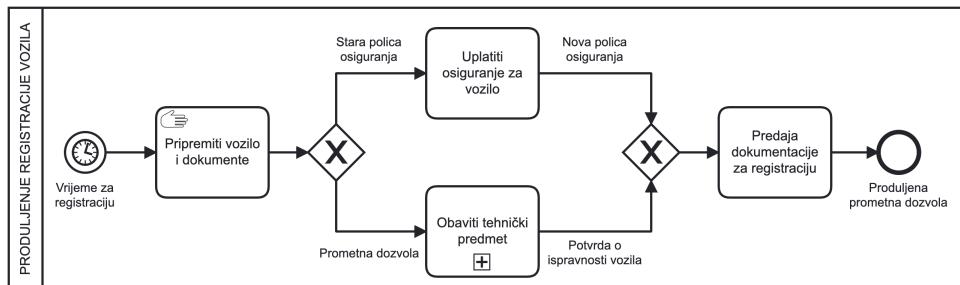
Potrebno je pričekati na izvršavanje svih aktivnosti prije nastavka procesa (otpreme proizvoda) - upravo to prikazujemo AND merge skretnicom.

Međutim, što ako uplata nije uspješna? Nema problema, **možemo kombinirati AND skretnicu s XOR skretnicom** kako bismo definirali alternativni tok.



Slika 9: Primjer kombiniranja AND i XOR skretnice za definiranje alternativnog toka

Primjer korištenja paralelne skretnice u procesu **produljenja registracije motornog vozila**. Kako je proces dovoljno poznat i ne treba ga posebno objašnjavati, prikazat ćemo samo dijagram gdje je istaknuto da se **dokumenti za registraciju mogu predati ako je prije toga (1) uspješno obavljen tehnički pregled** (koji je prikazan kao potproces jer u njemu treba riješiti postupak kako će izgledati ako prvi pregled nije bio uspješan) i **(2) uplaćeno osiguranje**.



Slika 10: Proces produljenja registracije motornog vozila

Iako je moguće prikazati aktivnosti **uplatiti osiguranje za vozilo** i potproces **obaviti tehnički pregled** kao sekvencijalne, **želimo naglasiti da se izvode paralelno i da je potrebno izvršiti oba zadatka** prije nastavka procesa - predaje dokumenata za registraciju.

### 3. Inkluzivna (eng. Inclusive) skretnica

Inkluzivna (OR) skretnica (eng. *Inclusive gateway*) koristi se za modeliranje situacija **baziranih isključivo na podacima** (vrijednostima u procesnoj instanci) gdje se **odabire jedan ili više izlaznih tokova**, odnosno provode se aktivnosti **na svim putevima za koji su ispunjeni uvjeti**.

Kao i kod XOR i AND skretnica, i kod inkluzivnih skretnica postoji skretnice **grananja i spajanja**.

- Ako inkluzivna skretnica ima više izlaznih tokova, potrebno je definirati **uvjete za svaki izlazni tok**.
- Ako inkluzivna skretnica ima samo jedan tok, onda ne mora imati definiran uvjet.



Inkluzivna skretnica se definira rombom, s oznakom kruga: o

Inkluzivnu skretnicu možemo zamisliti kao **logičku disjunkciju** (operator `OR`), odnosno **odabir jednog ili više uvjeta**. Ukoliko je zadovoljen uvjet, izvršava se odgovarajući tok. Na neki način radi se **mixu** između XOR i AND skretnice.

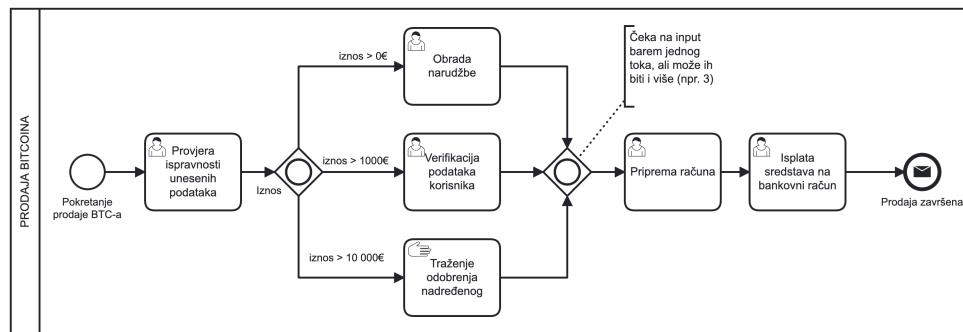
Primjer, imamo **Bitcoin mjenjačnicu** te želimo definirati slijed aktivnosti nakon što korisnik zatraži prodaju određene količine Bitcoin-a. Primitkom ponude, djelatnik mora poduzeti različite aktivnosti ovisno o cijeni transakcije (količini Bitcoin-a koja se prodaje):

- svakako moramo obraditi narudžbu za svaki iznos koji je veći od 0 eura
- za iznose veće od 1000 eura, moramo zatražiti verifikaciju podataka korisnika
- za iznose veće od 10 000 eura, moramo zatražiti odobrenje nadređenog

Ishod je uvijek isti, **isplata na bankovni račun korisnika i priprema računa**.

U opisanom procesu barem jedan uvjet će uvijek biti zadovoljen (`iznos > 0 eura`).

- mogu biti zadovoljena 2 uvjeta (`iznos > 0 eura` i `iznos > 1000 eura`)
- mogu biti zadovoljena 3 uvjeta (`iznos > 0 eura`, `iznos > 1000 eura` i `iznos > 10 000 eura`)

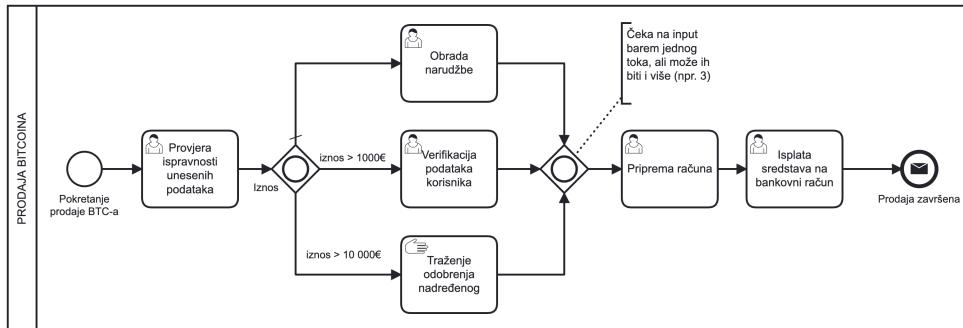


Slika 11: Proces prodaje Bitcoin-a u mjenjačnici

Kod **inkluzivnog grananja**, moramo uzeti nekoliko stvari u obzir:

- ako je jedan uvjet zadovoljen, **ne smijemo ignorirati ostale** (budući da i oni mogu biti zadovoljeni). Pratimo sve zadovoljene uvjete
- iako je moguće da je zadovoljeno više uvjeta, moguće je da nije ni jedan. Međutim, **dobro je definirati barem jedan uvjet koji će uvijek biti zadovoljen.**

**Česta greška**, iako na prvu nije očita, jest **ne definiranje uvjeta za svaki izlazni tok i definiranje defaultnog uvjeta**. Defaultni tok je tok koji smo rekli da označavamo oznakom i koji će uvijek biti zadovoljen.



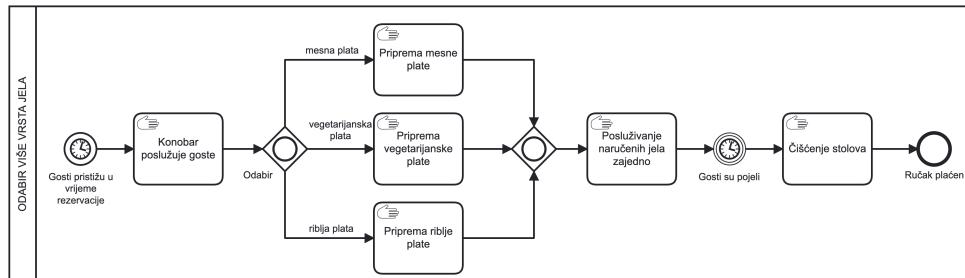
Slika 12: Proces prodaje Bitcoina u mjenjačnici s defaultnim tokom (**neispravno**)

Problem u ovom primjeru je sljedeći:

- ako je zadovoljen uvjet `iznos > 1000 eura`, neće se izvršiti *defaultni* tok i neće se izvršiti aktivnost **Obrada narudžbe**
- ako je zadovoljen uvjet `iznos > 10 000 eura`, neće se izvršiti *defaultni* tok i neće se izvršiti aktivnost **Obrada narudžbe**
- ako je zadovoljen uvjet `iznos > 1000 eura` i `iznos > 10 000 eura`, neće se izvršiti *defaultni* tok i neće se izvršiti aktivnost **Obrada narudžbe**.
- u svim drugim slučajevima (npr. iznos od 500 eura), izvršit će se samo aktivnost **Obrada narudžbe**.

**VAŽNO:** Poželjno je eksplicitno navesti uvjete za svaki izlazni tok, a *defaultni* tok koristiti samo ako je to nužno!

*Primjer posluživanja više vrsta jela u restoranu:* Skupina gostiju dolazi u restoran gdje se poslužuje hrana po prethodnoj rezervaciji za veće skupine gostiju. Nakon što se gosti smjeste, konobar donosi jelovnik i gosti biraju jelo. Radi jednostavnosti, recimo da gosti biraju između mesne, vegetarijanske i ribljе plate. Međutim, kako ima puno gostiju za stolom, vjerojatno je da će odabrati više različitih vrsta jela. Proces možemo modelirati kroz inkluzivnu skretnicu.



Slika : Proces posluživanja više vrsta jela u restoranu kroz inkluzivnu skretnicu

## 3.1 OR skretnica spajanja (eng. OR merge/join)

---

Kao i kod XOR i AND skretnica, i kod inkluzivnih skretnica postoji skretnica **spajanja** (eng. *merge/join*).

Skretnica spajanja koristi se za **spajanje više tokova u jedan tok**. U ovom slučaju, **spajamo sve tokove koji su zadovoljili uvjet**.

**Glavno pitanje** koje si možemo postaviti, što ako se zadovolji više od jednog uvjeta, mora li **OR** skretnica spajanja pričekati na sve zadovoljene uvjete prije nastavka ili ne?

Recimo da je korisnik prodao Bitcoin u iznosu od 12 000 eura. U tom slučaju, bit će zadovoljena sva 3 uvjeta. Ako su zadovoljena sva 3 uvjeta, svakako je **potrebno odraditi sve tri aktivnosti** prije nego možemo nastaviti s pripremom računa i isplatom.

- **Obrada narudžbe**
- **Verifikacija podataka**
- **Odobrenje nadređenog**

Dakle odgovor je **DA**, inkluzivna skretnica spajanja mora pričekati na sve **zadovoljene uvjete** prije nastavka. Naglasak je na pridjevu **zadovoljeni**.

Ako uvjet nije zadovoljen, inkluzivna skretnica spajanja neće čekati na izvršavanje te aktivnosti (ili tog niza aktivnosti) i nastaviti će dalje. Primjerice, ako je korisnik napravio narudžbu od 1500 eura, bit će zadovoljena 2 uvjeta, ali neće se čekati na odobrenje nadređenog (3. uvjet).

- **Obrada narudžbe**
- **Verifikacija podataka**

Ista situacija je i kod skretnice spajanja kod primjera s restoranom. Ako gosti odaberu više vrsta jela, konobar će pričekati na pripremu svih jela prije nego ih posluži. Međutim, ako su aktivne samo 2 aktivnosti (npr. mesna i vegetarijanska plata), konobar neće čekati na riblju platu.

# 4. Ukratko, kada koristiti koju skretnicu?

---

Do sad smo prošli kroz tri vrste skretnica u BPMN-u (premda ih ima još):

1. **Ekskluzivna ( `XOR` ) skretnica**
2. **Paralelna ( `AND` ) skretnica**
3. **Inkluzivna ( `OR` ) skretnica**

Kada koristiti koju skretnicu (u jednoj rečenici):

1. **`XOR` skretnica** koristi se za odabir jedne opcije između više opcija, gdje se uvijek **odabire samo jedna opcija** za koju je **predikat zadovoljen**.
2. **`AND` skretnica** koristi se za modeliranje situacija u kojima se više aktivnosti izvršava **paralelno**, a zatim se nastavlja dalje tek kada su **sve aktivnosti završene**.
3. **`OR` skretnica** koristi se za odabir jedne ili više opcija između više opcija, gdje se **odabiru sve opcije** za koje je **definirani logički izraz istinit**.

Također, vidjeli smo da za svaku skretnicu možemo definirati skretnicu grananja (*eng. split*) i skretnicu spajanja (*eng. join/merge*) kako bismo preciznije definirali tokove u procesu.

1. **`XOR` skretnica spajanja**: ako imamo dvije opcije nema ju previše smisla definirati (ali možemo), ali ako imamo više od dvije opcije, koristimo je za **čekanje na zadovoljenje jednog od uvjeta za nastavak**.
2. **`AND` skretnica spajanja**: koristimo je za **čekanje na završetak svih aktivnosti** koje su na izlaznom toku **paralelne skretnice grananja**.
3. **`OR` skretnica spajanja**: koristimo je za **čekanje na završetak svih aktivnosti** koje su na izlaznom toku **inkluzivne skretnice grananja**. Ova skretnica čeka na **završetak onih aktivnosti koje su zadovoljile** uvjet (ne moraju biti sve)

Važno je naglasiti da kod skretnica spajanja, nije nužno da su svi ulazni tokovi iz iste skretnice grananja. Možemo imati više skretnica grananja koje se spajaju u jednu skretnicu spajanja i sl.

## Zadaci za Vježbu 3

---

### 1. Wolt - dostava hrane

Modelirajte proces naručivanja hrane preko Wolt aplikacije. Proces započinje jednom kad u restoran pristigne narudžba s Wolt aplikacije. Nakon što djelatnik obradi narudžbu, paralelno se kreće u pripremu hrane i obavještavanje dostavljača. Dostavljač, kao vanjski sudionik, sudjeluje samo u procesu dostave hrane. Taj proces započinje jednom kad dostavljaču pristigne obavijest o traženoj dostavi. Dostavljač pregledava obavijest i odlučuje hoće li prihvati dostavu. Ako odbije, njegov proces tu završava i o tome obavještava restoran. Ako prihvati, obavještava restoran da će preuzeti dostavu.

Međutim, dostavu je moguće preuzeti tek kad je hrana gotova, što traje određeno vrijeme te nakon što se spakira. U međuvremenu, restoran čeka na potvrđnu informaciju od dostavljača. Ako je potvrda informacija pozitivna (dostavljač prihvata dostavu unutar 30 minuta) i hrana spakirana, tada se tok može nastaviti. Ako je potvrđna informacija negativna, tada se u sustavu zatraži novi dostavljač. Jednom kad su svi uvjeti zadovoljeni, obavještava se korisnika da je hrana na putu te se potom paralelno izrađuje račun i obavještava

dostavljača da je hrana gotova. Dostavljač čeka na tu informaciju, dostavlja robu i tu njegov proces završava, dok proces u restoranu završava izdavanjem računa i obavještavanjem dostavljača da je hrana gotova.

---

# Upravljanje poslovnim procesima (UPP)

**Nositelj:** izv. prof. dr. sc. Darko Etinger

**Asistent:** Luka Blašković, mag. inf.

**Ustanova:** Sveučilište Jurja Dobrile u Puli, Fakultet informatike u Puli



Fakultet informatike u Puli

## (4) Smjernice u modeliranju i predlošci tokova rada

#4

UPP

Cilj ove skripte pružiti je sveobuhvatan pregled principa i smjernica u modeliranju poslovnih procesa kroz BPMN 2.0 notaciju. Do sad ste naučili osnovnu BPMN notaciju, uključujući osnovne elemente, tokove i događaje. Međutim, osim poznavanja elemenata (sintakse) važno je poslovni proces modelirati na način koji je razumljiv, konzistentan svim dionicima, ali i treba biti precizan s reprezentacijom stvarnog poslovnog procesa. U ovoj skripti ćemo se fokusirati na izvršivost procesa, pravilno imenovanje i upotrebu međudogađaja, modeliranje komunikacije te predloške tokova rada te smjernice za što bolje modeliranje i teorijsko razumijevanje poslovnih procesa u kontekstu BPMN notacije.

Posljednje ažurirano: 2.1.2025.

## Sadržaj

- [Upravljanje poslovnim procesima \(UPP\)](#)
- [\(4\) Smjernice u modeliranju i predlošci tokova rada](#)
  - [Sadržaj](#)
- [1. Smjernice za modeliranje procesa](#)
  - [1.1 Aktivnosti vs Događaji](#)
  - [1.2 Koji međudogađaj odabrati?](#)
  - [1.3 Česte greške u modeliranju komunikacije između procesa](#)
  - [1.4 Nekoliko korisnih smjernica](#)
  - [1.5 Entiteti na Message flow](#)
- [2. Predlošci tokova rada](#)
  - [Sličnost na razini poslovne domene \(makrorazina\)](#)

- [Sličnost na razini aktivnosti koje čine proces \(mikrorazina\)](#)
- [2.1 Osnovni predlošci za upravljanje slijedom](#)
  - [WCP-1 Slijed \(eng. Sequence\)](#)
  - [WCP-2 Paralelno dijeljenje \(eng. Parallel Split\)](#)
  - [WCP-3 Sinkronizacija \(eng. Synchronization\)](#)
  - [WCP-4 Ekskluzivni izbor \(eng. Exclusive Choice\)](#)
  - [WCP-5 Jednostavno spajanje \(eng. Simple Merge\)](#)
- [2.2 Predlošci za grananje, sinkronizaciju i iteraciju](#)
  - [WCP-6 Višestruki izbor \(eng. Multiple Choice\)](#)
  - [WCP-7 Strukturno sinkronizirano spajanje \(eng. Structured Synchronizing Merge\)](#)
  - [WCP-8 Nesimetrično sinkronizirano spajanje \(eng. Acyclic Synchronizing Merge\)](#)
  - [WCP-9 Proizvoljno ponavljanje \(eng. Arbitrary Cycles\)](#)
- [2.3 Predlošci za okidače](#)
  - [WCP-10 Prolazni okidač \(eng. Transient Trigger\)](#)
  - [WCP-11 Stalni okidač \(eng. Persistent Trigger\)](#)

# 1. Smjernice za modeliranje procesa

---

Do sad ste naučili da postoje 3 glavna objekata toka u BPMN notaciji, to su:

1. **Događaji** (eng. *Events*)
2. **Aktivnosti** (eng. *Activities*)
3. **Skretnice** (eng. *Gateways*)

Premda su razlike između ovih objekata jasne, ponekad je teško odrediti koji objekt koristiti u određenom trenutku.

Rekli smo da:

- **Događaji** označavaju određene trenutke u procesu koji označavaju promjenu stanja, poput početka (*start event*), završetka (*end event*) ili ključnih točaka između (*intermediate event*). Oni su **pasivni elementi** i ne **podrazumijevaju akciju**, već **signaliziraju** da se određeni uvjet ispunio ili stanje promjenilo
- **Aktivnosti** predstavljaju zadatke ili skup radnji koje se trebaju izvršiti kako bi proces "napredovao". Rekli smo da su za izvođenje aktivnosti potrebni neki **resursi i vrijeme**. Radi se o **operativnim elementima procesa**.
- **Skretnice** omogućuju donošenje odluka unutar procesa, usmjeravajući tijek rada prema različitim pravcima na temelju definiranih uvjeta. Smjernice su ključne za **razgranavanje i kontrolu toka procesa**.

## 1.1 Aktivnosti vs Događaji

---

Iako su definicije aktivnosti i događaja poprilično jasne, ponekad možemo biti u nedoumici koji što odabrat. Najbolje je razmišljati o tome kako se događaji i aktivnosti razlikuju u kontekstu procesa.

Primjerice, ako se pitate "*Što se događa u procesu?*", tada je vjerojatno da je riječ o aktivnosti. S druge strane, ako se pitate "*Kada se nešto događa ili se dogodilo u procesu?*", tada je vjerojatno da je riječ o događaju.

Za primjer uzimimo proces NARUČIVANJE PROIZVODA, tada bi se **aktivnosti** mogle odnositi na:

- "Unos podataka o kupcu",
- "Unos podataka o proizvodu",
- "Plaćanje",
- "Pakiranje",
- "Dostava"

Dok bi se **događaji** mogli odnositi na:

- "Primljen zahtjev za narudžbu",
- "Proizvod poslan kupcu",
- "Plaćanje primljeno"

**Kako ispravno imenovati Aktivnost:**

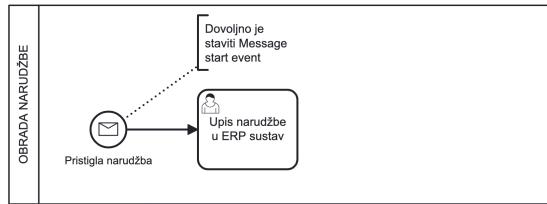
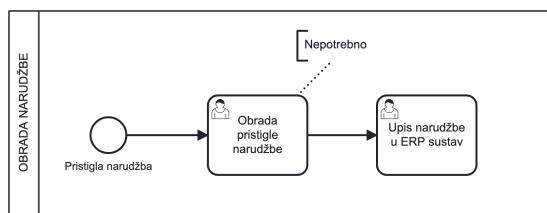
- **Glagolska imenica** koja opisuje radnju, npr. "Unos", "Plaćanje", "Pakiranje", "Dostava"
- Naglašen **objekt** na kojeg se aktivnost odnosi, npr. "Unos podataka", "Pakiranje proizvoda", "Pakiranje robe", "Dostava paketa"
- Može biti i **glagol u infinitivu**, npr. "Obavijestiti kupca", "Poslati proizvod", "Pripremiti račun"
- **Nije uobičajeno navoditi subjekt**, budući da je subjekt implicitno jasan iz konteksta modela (polje/staze), npr. "Kupac unosi podatke", "Dostavljač dostavlja paket"
- Poželjno je koristiti infinitiv, glagolsku imenicu ili iznimno glagol u 3. licu (nikako u 1. i 2. licu)
- Uobičajeno je koristiti **aktivni glagolski oblik**

**Kako ispravno imenovati Događaj :**

- **Glagolska imenica** koja opisuje **prošlo** (završeno) stanje, npr. "Primljena narudžba", "Plaćanje primljeno", "Proizvod poslan", "Pristigla narudžba"
- **Poželjno je** da sadržava informacije o subjektu i objektu, npr. "Kupac poslao narudžbu", "Dostavljač preuzeo paket", "Proizvod poslan kupcu", "Pristigao email od klijenta"
- Ako se radi o *timer eventu*, tada se koristi **vremenska oznaka**, npr. "Nakon 3 dana", "Svaki ponedjeljak", "Svaki mjesec", "Prošlo 5 minuta" itd.
- Radnja ne smije biti **u infinitivu** (kao kod aktivnosti), već se radi o **završenoj radnji** koja se dogodila u prošlosti, npr. "Proizvod poslan", "Plaćanje primljeno", "Narudžba zaprimljena"
- **Međudogađaje** je moguće imenovati i u **futuru** (budućnosti), npr. "Po primitku zahtjeva", "Kada stigne odgovor", "Jednom kad se dogodi..."
- **Nije uobičajeno koristiti infinitiv**
- **Uobičajeno je koristiti pasivan glagolski oblik**

Važno je dobro naučiti i držati se ovih smjernica imenovanja budući da će vam to olakšati odabir ispravnog elementa prilikom modeliranja procesa.

*Primjer: Obrada pristigle narudžbe i upis narudžbe u ERP sustav*

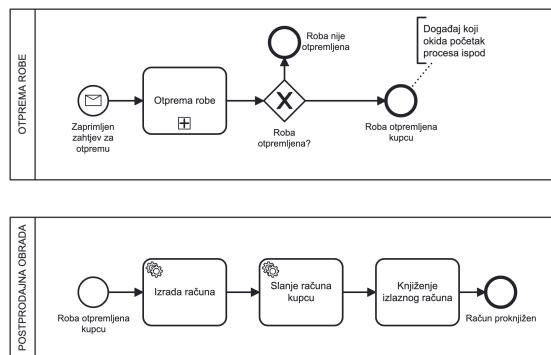


Primjer neispravno modeliranog (iznad) i ispravno modeliranog (ispod) procesa

"Pristigla narudžba" prikazujemo kao `start Event` jer se radi o završenoj radnji koja se dogodila u prošlosti. S druge strane, "Upis narudžbe u ERP sustav" prikazujemo kao `User Task` jer se radi o radnji koju je potrebno poduzeti kako bi proces "napredovao".

"Obrada pristigle narudžbe" u ovom kontekstu korisničkog zadatka nema puno smisla, jer iz opisa aktivnosti nije jasno što se ovom radnjom postiže. Prisjetite se da korisničke aktivnosti (zadaci) trebaju biti jasno i precizno definirani, a u kontekstu procesne aplikacije oni će biti **operativne radnje** gdje korisnik unosi podatke, odabire opcije i sl. kroz sučelje **web forme**.

*Primjer: Proces postprodajne usluge koji započinje jednom kad je roba otpremljena kupcu*



Primjer dvaju modeliranih procesa u zasebnim poljima

U ovom procesu "Roba otpremljena kupcu" prikazujemo kao `End Event` jer se radi o završenom događaju, koji se dogodio kao rezultat procesa otpreme. Proses POSTPRODAJNA OBRADA započinje jednom kad je roba otpremljena kupcu, a završava kad je postprodajna usluga pružena. Ovakvim imenovanjem procesa nema smisla proces POSTPRODAJNE USLUGE ugnijezditi kao potproces unutar procesa OTPREMA ROBE jer se radi o dvjema različitim procesima, odnosno proces postprodaje započinje tek kad je roba otpremljena kupcu.

Isto bi bilo moguće kada bi modelirali proces OBRADA NARUDŽBE koji bi "razbili" na potprocese OTPREMA ROBE i POSTPRODAJNA OBRADA.

## 1.2 Koji međudogađaj odabrat?

**Međudogađaji** (eng. *Intermediate Events*) koriste se za označavanje **ključnih točaka (događaja) između početka i kraja procesa**. Preciznije, koriste se za modeliranje ključnih trenutaka u procesu koji **ne predstavljaju početak ili kraj procesa**, ali svakako mogu promijeniti tijek izvođenja.

**Međudogađaji** se koriste za:



Označavanje **ključnih točaka** u procesu, tzv. **Milestone** (`Intermediate throw event`)

- Primjeri **ispravnog** imenovanja: "Tijesto se dignulo", "Vrijeme isteklo", "Paket spreman za slanje", "Hrana spremna", "Vremenska prognoza prikladna", "Proizvod na zalihi"...
- Primjeri **neispravnog** imenovanja: "Spremanje tijesta", "Priprema paketa", "Pakiranje robe", "Pohrana bilješki"...



Definiranje **nespecificiranog čekanja** u procesu, odnosno **čekanje na primitak vanjskog signala** (`Message Intermediate Catch event`)

- Primjeri **ispravnog** imenovanja: "Primljen signal", "Primljena odbijenica", "Jednom kad pristigne odgovor", "Jednom kad je gotov", "Pristigao email potvrde", "Kada pristigne poruka klijenta", "Čekanje na primitak obavijesti o...", "Po primitku zahtjeva"...
- Primjeri **neispravnog** imenovanja: "Slanje odgovora korisniku", "Klijent provjerava email...", "Djelatnik obavještava..."



Označavanje kraja **specificiranog čekanja** u procesu, ili **početak određenog vremenskog razdoblja** (`Timer Intermediate Catch event`)

- Primjeri **ispravnog** imenovanja: "Nakon 3 dana", "Svaki ponедјелjak", "Svaki мјесец", "Prošlo 5 minuta", "Stigao ponедјелjak", "Prošlo je 2 tjedna", "90 minuta", "4 sata", "Pristizanje na red u koloni", "Narudžba došla na red za obradu nakon X vremena"
- Primjeri **neispravnog** imenovanja: "Čekaj timer", "Čekanje na odgovor", "Čekanje na primitak poruke", "Čekaj", "Timer"

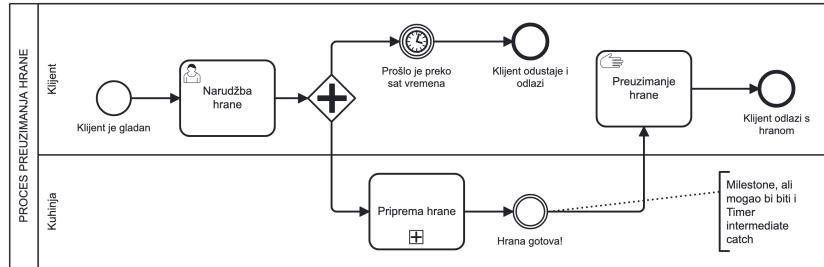


Za **signalizaciju drugih aktora** u procesu, ili **okidanje drugih procesa** (`Message Intermediate Throw event`)

- Primjeri **ispravnog** imenovanja: "Proces X signaliziran", "Klijent obaviješten", "SMS poslan", "Poslan email kupcu", "Inicijaliziran proces narudžbe", "Račun dostavljen klijentu" itd.
- Primjeri **neispravnog** imenovanje: "Pošalji email", "Slanje računa kupcu", "Obavijesti klijenta", "Pošalji SMS djelatniku"

**Zaključno:** Međudogađaje nastojite imenovati na način da jasno i precizno opisuju **trenutak** ili **stanje** u procesu koji se dogodio ili koji će se dogoditi (koji se očekuje). Koristite pasivan ton, izbjegavajte infinitiv te koristite pasivan glagolski oblik. **Izbjegavajte nazivati ove događaje kao radnje (aktivnosti)**

Primjer: *Klijent naručuje hrani iz restorana putem nekog medija, a potom čeka na dovršetak. Međutim, ako prođe preko sat vremena, klijent odustaje od narudžbe.*



Primjer modeliranog procesa u jednom polju s dvije staze

U ovom primjeru, koristi se **paralelni skretnici (AND)** te se proces dalje razlaže na način: "ono što se prije dogodi".

- ili će se narudžba dovršiti u roku od 1 sata i klijent će ju preuzeti
- ili će proći sat vremena i klijent odustaje

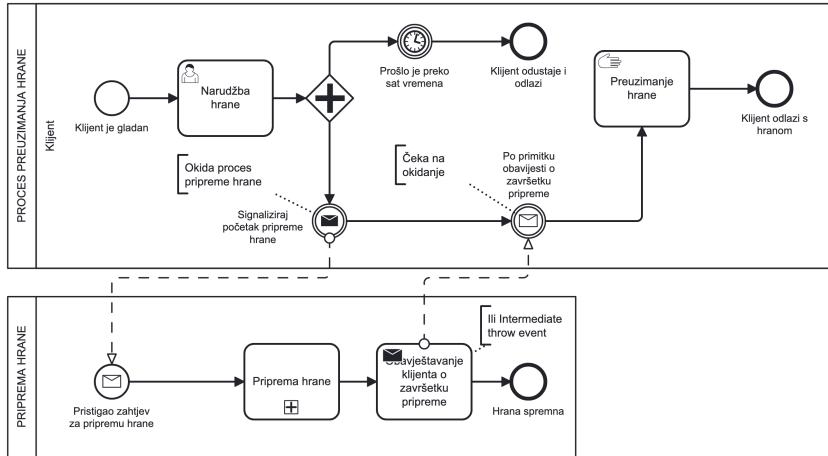
**Kako znamo unaprijed vremensko razdoblje**, možemo iskoristiti **Timer Intermediate Catch Event** za čekanje tih sat vremena ako se narudžba ne dovrši.

"Priprema hrane" je prikazana sklopljenim potprocesom (eng. collapsed subprocess) koji traje neko vrijeme, a jednom kad je hrana spremna, ovisno o kontekstu, možemo definirati *milestone* (ključnu točku) "Hrana spremna" ili "Hrana gotova!" kao **Intermediate Throw Event**. Međutim, u ovom slučaju je moguće istu stvar prikazati i **Timer Intermediate Catch event** budući da se radi o vremenskom razdoblju potrebnom za pripremu hrane gdje nam je procijenjeno vrijeme pripreme hrane unaprijed poznato.

Međutim, *milestone* i ne mora biti nužno vezan uz vremensko razdoblje, već može označavati **ključnu točku u procesu**, npr. "Vremenska prognoza prikladna" → Okini neki drugi proces ili nastavi slijed aktivnosti.

Vanjske procese moguće je okinuti pomoću **Send Task** aktivnosti, međutim ono se preciznije koristi za slanje poruka vanjskim dionicima (npr. email, SMS i sl.). Ukoliko želimo okinuti drugi proces, prikladnije je koristiti **Message Intermediate Throw Event**.

**Komunikaciju između događaja** prikazujemo kroz **Message Flow**, koji predstavlja samo **vezu informativnog karaktera** (ne utječe na sekvenčalni tijek procesa već samo pruža informaciju o komunikaciji između dva objekta).



Primjer modeliranog procesa u dva polja

U primjeru iznad, prikladno je koristiti `Send Task` aktivnost za signalizaciju međudogađaja "Primanje obavijesti o završetku pripreme" budući da se radi o slanju nekog oblika poruke. Da to nije slučaj, prikladnije bi bilo koristiti `Intermediate Throw Event` te definirati *milestone*, npr. "Hrana spremna".

## 1.3 Česte greške u modeliranju komunikacije između procesa

U BPMN modelima često moramo modelirati **komunikaciju između različitih procesa** (npr. jedan proces pokreće/okida drugi) ili inter-procesnu komunikaciju gdje se informacije razmjenjuju **između različitih staza**.

Jedna od najčešćih grešaka u modeliranju procesa odnosi se upravo na neispravno modeliranje komunikacije i sekvencialnog tijeka.

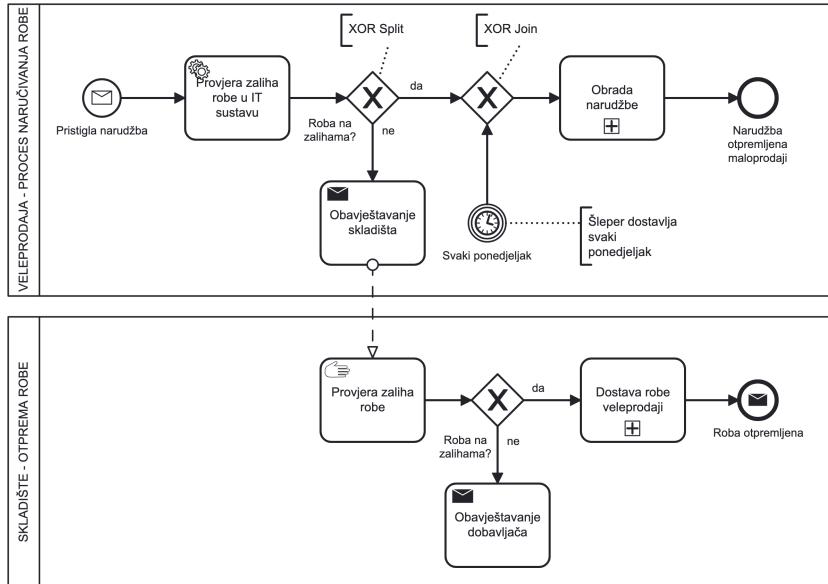
Prisjetimo se elemenata koji se koriste za komunikaciju:

- `Message Flow` - **veza** između dva objekta koja označava komunikaciju između njih (informativnog karaktera)
- `Sequence Flow` - **veza** između dva objekta koja označava sekvencialni tijek procesa (utječe na egzekuciju procesa)
- `Message Intermediate Throw Event` - **međudogađaj** koji signalizira drugom procesu da nešto učini
- `Message Intermediate Catch Event` - **međudogađaj** koji čeka na signalizaciju od drugog procesa
- `Send Task` - **aktivnost** koja šalje poruku (e-mail, SMS, itd.) vanjskom dioniku (može pokretati druge procese)
- `Receive Task` - **aktivnost** koja čeka na primitak poruke (e-mail, SMS, itd.) od vanjskog dionika kako bi sekvencialni tijek nastavio

Komunikaciju ćemo nastojati objasniti na primjeru **Veleprodaje i Skladišta** na procesima naručivanja i otpreme robe:

**Veleprodaju** predstavljamo kao zasebni proces u zasebnom polju (VELEPRODAJA - PROCES NARUČIVANJA ROBE). Proces započinje kad veleprodaja zaprimi narudžbu. Evidencijom zaliha zaključuju da nedostaje robe pa moraju kontaktirati skladište kako bi provjerili dostupnosti i naručili što nedostaje.

**Skladište** u ovom kontekstu je *outsourcani* partner čiji ćemo proces nazvati OTPREMA ROBE.

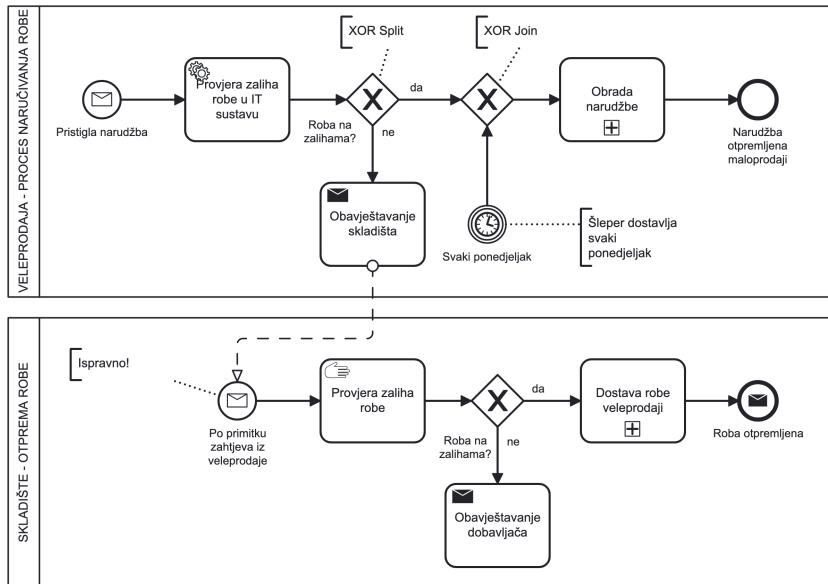


### Primjer modeliranih procesa s pogrešnom komunikacijom između 2 polja/procesa

Idemo identificirati što je dobro, a što pogrešno u ovom modelu.

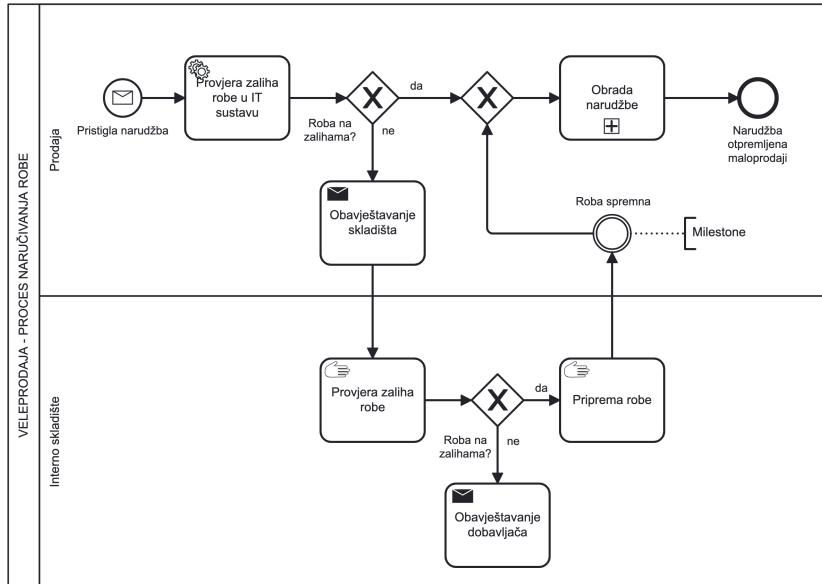
- **Dobar dio:** Budući da se radi o slanju poruke skladištu koristimo **Send Task** aktivnost za slanje poruke o nedostatku robe na zalihamu u veleprodajnom skladištu
- **Pogreška:** Međutim, SKLADIŠTE - OTPREMA ROBE je proces za sebe, definiran u vlastitom polju, a nema početni događaj. Svaki proces (definiran u vlastitom polju) ili potproces mora imati **start Event**.

Kako ispraviti ovu pogrešku? Jednostavno ćemo dodati početni događaj u polje **Skadište - Otprema robe**.



### Primjer modeliranih procesa s ispravljenom komunikacijom između 2 polja/procesa

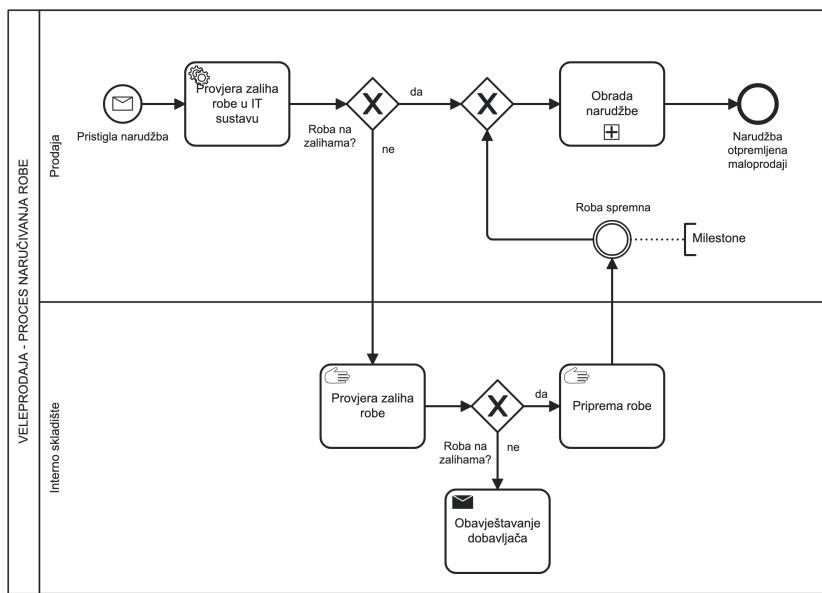
Međutim, što ako se radi o internom skladištu koje je dio iste veleprodaje? U tom slučaju komunikaciju **ne želimo modelirati kao slanje poruke** (prisjetimo se emaila, SMS-a i sl.) Dakle, možemo maknuti **Send Task** aktivnost i samo nastaviti **Sequence Flow**.



Primjer modeliranog procesa VELEPRODAJA - PROCES NARUČIVANJA ROBE **pogrešnom komunikacijom unutar istog polja**

- **✓ Dobar dio:** Budući da se radi o internom skladištu, uklanjamo `Timer Intermediate Catch event` "Svaki ponedjeljak", već na sekvenčijalni slijed između pripreme robe i XOR Mergea možemo ubaciti `milestone` "Roba spremna", iako je to više optionalno, bolje pojašnjava tijek procesa i **naglašava da je postojalo neko vremensko razdoblje** za pripremu robe.
- **✗ Pogreška:** Radi se o internom skladištu, ne modeliramo "slanje maila, SMS-a ili sl. poruke" već samo "provjeravamo" dostupnost robe u tom skladištu putem istog IT sustava, ERP-a ili sl.

Možemo jednostavno ukloniti `Send Task` aktivnost i samo nastaviti `Sequence Flow`



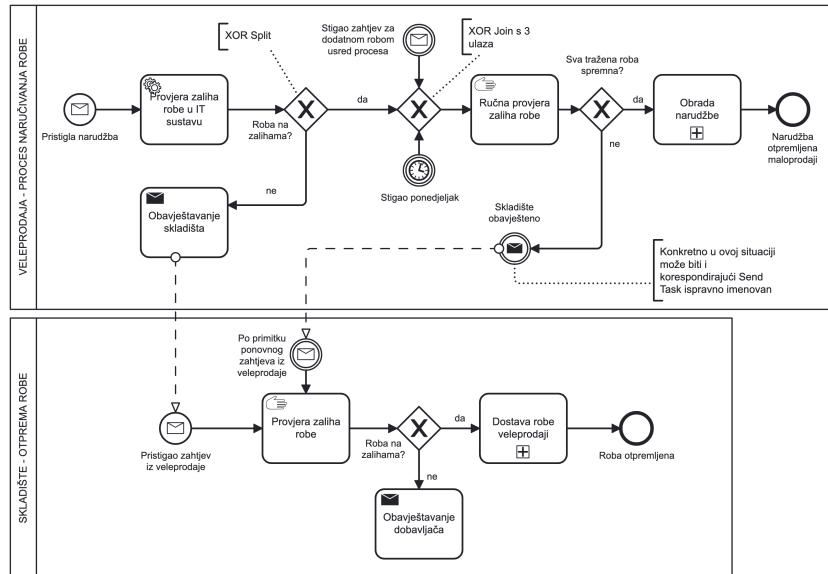
Primjer modeliranog procesa VELEPRODAJA - PROCES NARUČIVANJA ROBE s **ispravnom komunikacijom unutar istog polja**

Razmotrimo ponovno scenarij s eksternim skladištem. Što se događa ako se, nakon dolaska šlepera s robom, pojavi potreba za dodatnom narudžbom robe?

- Primjer situacije: maloprodaja je zatražila dodatne artikle koji nisu bili uključeni u prvobitnu narudžbu međutim instance procesa SKLADIŠTE - OTPREMA ROBE je već završena dok proces VELEPRODAJA - PROCES NARUČIVANJA ROBE čeka na međudogađaju dok ne stigne ponedjeljak, samim tim i šleper.

Kako bismo cijeli proces učinili jasnijim i preglednijim, možemo ga modelirati **korištenjem međudogađaja**

Message Intermediate Throw Event i Message Intermediate Catch Event:



Primjer modeliranog procesa s ponovnim pokretanjem procesa 'Otprema robe' u skladištu

- ✓ Ispravno:** U ovom slučaju, koristimo Message Intermediate Throw Event za signalizaciju potrebe za dodatnom narudžbom robe. Kada se dogodi taj međudogađaj, ponovno se stvara instance procesa VELEPRODAJA - PROCES NARUČIVANJA ROBE. Koristimo korespondirajući Message Intermediate Catch Event za hvatanje tog signala i pokretanje procesa SKLADIŠTE - OTPREMA ROBE.
- ✓ Ispravno:** Definirali smo i Message Intermediate Catch Event "Stigao zahtjev za dodatnom robom usred procesa" kako bi jasno definirali trenutak kad se za vrijeme trajanja procesa VELEPRODAJA - PROCES NARUČIVANJA ROBE pojavila potreba za dodatnom narudžbom robe. Potreba se pojavila usred procesa, dok se čeka na dolazak šlepera s prvotnom narudžbom.

Ono što vas može buniti je razlika između Message Intermediate Throw Event i Send Task aktivnosti. Radi se o vrlo sličnim objektima (bez obzira što je jedan događaj, drugi aktivnost). U primjeru iznad, oba se mogu koristiti za slanje poruke vanjskom skladištu s narudžbom. **Razlike su sljedeće:**

- Send Task - koristi se za **slanje poruke dioniku procesa** (npr. email, SMS, itd.) te **potencijalno okidanje procesa na temelju te poruke** (ipak, češće je to samo slanje poruke)
- Message Intermediate Throw Event - koristi se za **signaliziranje drugog procesa** da nešto "učini" (npr. signalizacija u procesu OTPREMA ROBE) ne nužno slanjem poruke, već **okidanje procesa na temelju događaja (event-based)**

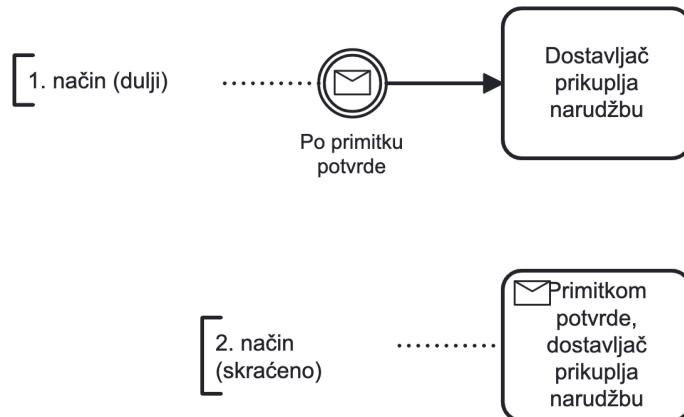
Ista distinkcija vrijedi i za Message Intermediate Catch Event i Receive Task.

U oba slučaja potrebno je držati se pravila imenovanja događaja (međudogađaja) i aktivnosti kako bi model ostao konzistentan prema BPMN sintaksi.

## 1.4 Nekoliko korisnih smjernica

- Koristite `Send Task` za eksplisitno slanje poruka dionicima procesa (npr. email, SMS, itd.)
- Koristite `Receive Task` za obradu poruka koje dolaze od dionika procesa (`Receive Task` je ustvari ekvivalentna kratica za `Message Intermediate Catch event + Task`)
- Koristite `Message Intermediate Throw Event` za signalizaciju drugim procesima da nešto učine (okidanje drugih procesa)
- Koristite `Message Intermediate Catch Event` za hvatanje signala ili poruka od drugih procesa (okidanje procesa na temelju događaja)
- Koristite `Intermediate Throw Event` za signalizaciju ključnih točaka u procesu (*milestone*)
- Koristite `Timer Intermediate Catch Event` kada proces stagnira na način da čeka na specificirano vremensko razdoblje

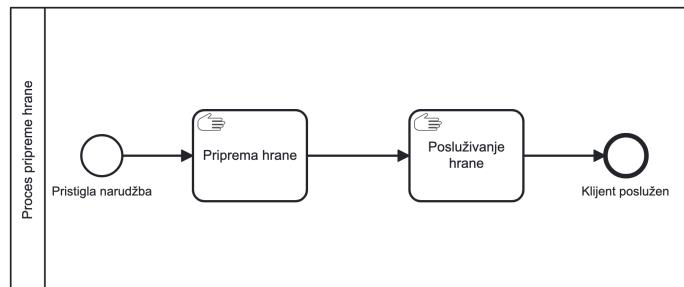
`Receive Task` nismo puno dosad koristili, međutim on je ustvari kratica za `Message Intermediate Catch event` i `Task` koji se koristi za hvatanje poruka od vanjskih dionika. Ukoliko želimo modelirati aktivnost koji čeka na primitak poruke te ju nakon tog izvršavamo, koristimo `Receive Task`.



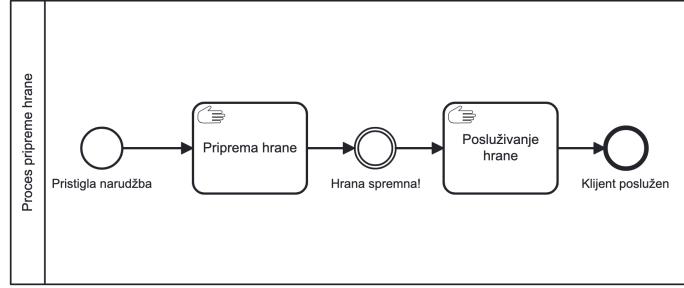
Sljedeći primjeri su ekvivalentni (Iznad je `Message Intermediate Catch event + Task`, ispod je `Receive Task`)

`Intermediate Throw Event` je korisno koristiti kada želimo naglasiti ključne točke u procesu, tzv. *milestone* (npr. "Roba spremna", "Vrijeme isteklo", "Proizvod na zalihi", "Hrana spremna"). Bez obzira, procesi tijek je moguće definirati i bez njih, ali na ovaj način možemo značajno **poboljšati čitljivost i razumljivost procesa**.

Uzmimo za primjer sljedeći super jednostavan proces pripreme i posluživanja hrane:

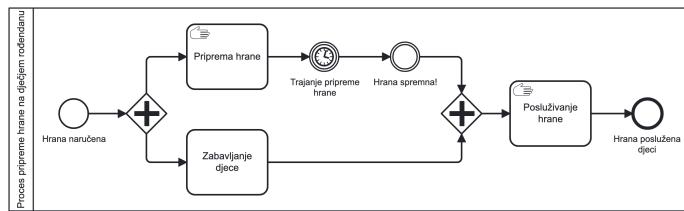


Možemo dodati `Intermediate Throw Event` "Hrana spremna" kako bismo naglasili ključnu točku u procesu. Ovaj događaj ne utječe na sekvencijalni tijek procesa, već samo signalizira da je hrana spremna za posluživanje te na taj način postižemo bolju čitljivost procesa.

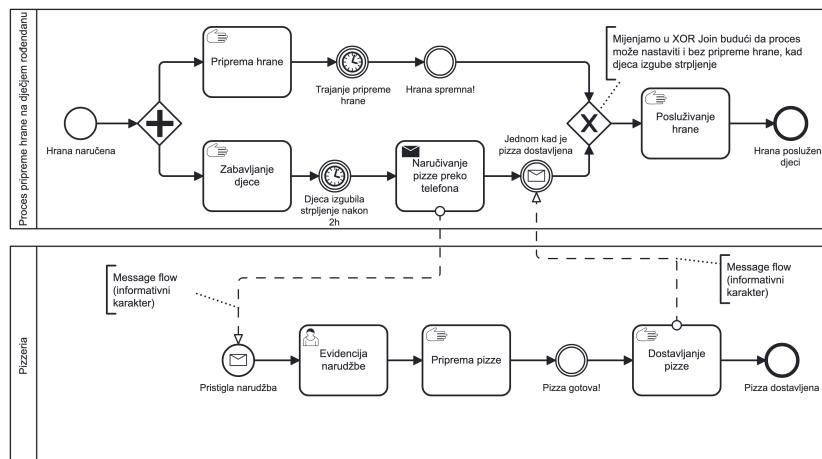


*Primjer: Što ako su naši klijenti djeca na rođendanu? Želimo zabavljati djecu dok čekaju hrana.*

Možemo dodati paralelnu aktivnost gdje zabavljamo djecu dok hrana nije gotova, a samo čekanje na spremanje hrane naglasiti kroz **Timer Intermediate Catch Event** "Trajanje pripreme hrane". Po završetku vremenskog razdoblja, proces se nastavlja.



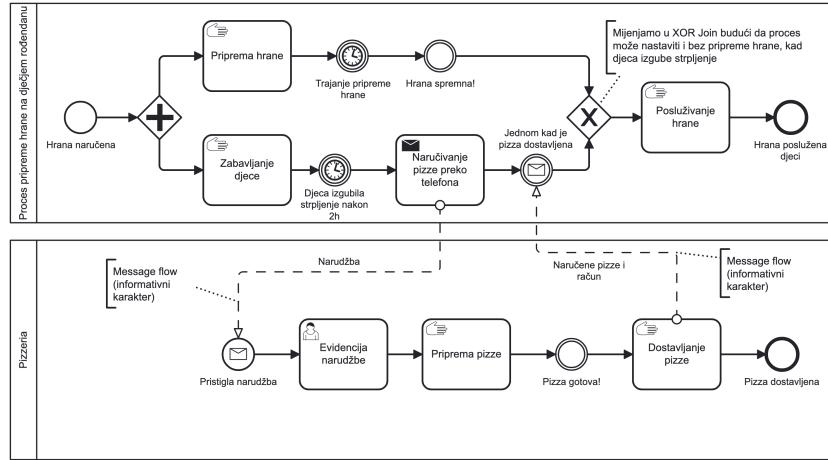
*Primjer: Zakomplificirat ćemo još malo stvari. Što ako nam hrana izgori i nemamo više ideja kako zabavljati djecu (primjerice prođe preko 2 sata)? U tom slučaju, ćemo naručiti pizzu iz obližnje pizzerije. Komunikaciju prema pizzeriji možemo prikazati kroz **Send Task** aktivnost koja se izvršava jednom kad se okine **Timer Intermediate Catch Event** - "Djeca izgubila strpljenje nakon 2 sata".*



## 1.5 Entiteti na Message flow

Uobičajeno je dodati entitete na **Message Flow** kako bi se dodatno pojasnila komunikacija između objekata. Primjerice, možemo dodati entitet "Narudžba" na **Message Flow** između **Send Task** "Naručivanje pizze preko telefona" i **Message Start Event** "Pristigla narudžba" kako bi naglasili da je poruka koja se šalje upravo **narudžba s detaljima o pizzi**.

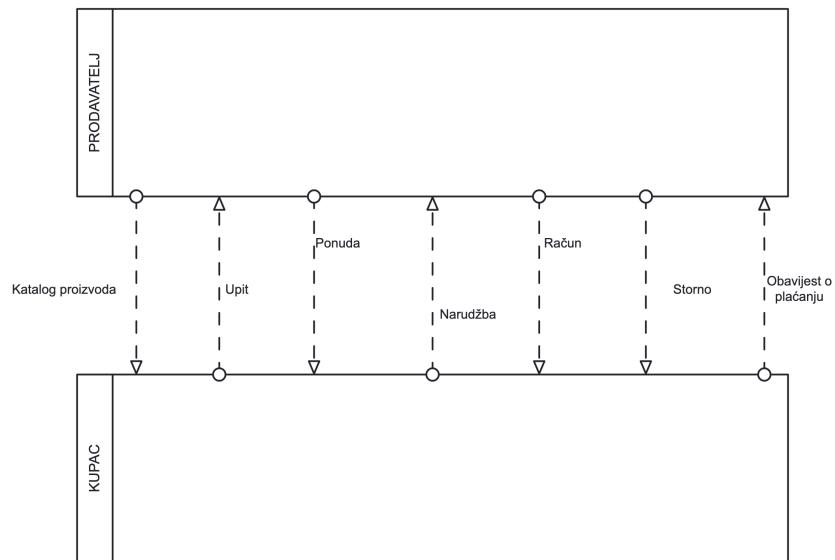
Jednako tako možemo na **Message Flow** između **Manual task** "Dostavljanje pizze" i **Message Intermediate Catch Event**: "Jednom kad je pizza dostavljena", dodati entitet "Naručene pizze i račun" kako bi naglasili da se poruka odnosi na **dostavu pizze i račun**.



**Entiteti** na **Message Flow** su korisni jer:

- **pojašnjavaju** što se šalje između objekata
- **jasno definiraju** što se očekuje od poruke
- **poboljšavaju čitljivost i razumljivost** modela

Ilustracija ispod prikazuje komunikaciju između PRODAVATELJA i KUPCA te različite entitete koji se šalju između njih, a koje definiramo na **Message Flow**.



## 2. Predlošci tokova rada

---

Poslovni procesi s kojima se susrećemo izgledaju nam međusobno vrlo različiti: čini se da svaki od njih ima svoje specifične ciljeve, da se provodi u drugom okruženju i da raspolaže drugim resursima. Premda je to točno, dublja studija ipak otkriva da u logičkoj strukturi modela procesa ima mnogo više sličnosti nego što se to čini u prvom trenutku. Ta se sličnost može utvrditi na dvjema (možemo reći **makro** i **mikro**) razinama.

### Sličnost na razini poslovne domene (makrorazina)

U dosadašnjim primjerima razmatrali smo modele koji bi se mogli primijeniti u više različitih organizacija. Tako se npr. roba široke potrošnje sa svakog veleprodajnog skladišta distribuira prema modelu koji je sličan onom koji smo spomenuli na početku vježbi (narudžba, otprema, dostava). Iako se detalji mogu razlikovati, osnovni tok poslovnog procesa je isti, odnosno aktivnosti se provode prema općoj shemi: PRIHVATITI NARUDŽBU → PROVJERITI MOGUĆNOST ISPORUKE → IZUZETI ROBU SA SKLADIŠTA → OTPREMITI ROBU KUPCU → IZRADITI RAČUN.

Ako prepoznamo tipske procese u više uspješnih organizacija u određenom poslovnom području, moći ćemo izabrati one koji najbolje odgovaraju našem poslovanju (*eng. best practice*) ili ih optimizirati i prilagoditi svojim specifičnim potrebama. Takva tipizacija procesa vodi nas do tzv. **referentnih poslovnih procesa** (obično ih nude proizvođači sustava ERP).

### Sličnost na razini aktivnosti koje čine proces (mikrorazina)

U dosadašnjim smo primjerima vidjeli da se svaki poslovni proces sastoji od niza objekata toka koji su međusobno povezani slijednim (*eng. sequential flow*) ili informacijskim vezama (*eng. message flow*). Već letimična analiza pokazuje da se u različitim procesima često ponavljaju odnosi između objekata toka, kao na primjer:

- **slijed** (AKTIVNOST A → slijedna veza → AKTIVNOST B → slijedna veza → AKTIVNOST C...)
- **izbor** (AKTIVNOST A nakon čega slijedi AKTIVNOST B ili AKTIVNOST C ili AKTIVNOST D...)
- **paralelno izvođenje** dvaju ili više aktivnosti itd.

Za navedene tipične oblike odnosa između objekata toka uobičajen je naziv **predlošci tokova rada** (*eng. workflow patterns*).

Predložaka za upravljanje tokom rada ima jako puno, a moguće ih je podijeliti u nekoliko kategorija. U nastavku će, kroz potpoglavlja, biti prikazani neki od najčešće korištenih predložaka tokova rada.

### 2.1 Osnovni predlošci za upravljanje slijedom

---

U ovoj grupi je ukupno **pet predložaka o upravljanju slijedom izvođenja aktivnosti**. Gotovo sve ste ih već nesvesno koristili u dosadašnjim primjerima modeliranja procesa. Ovdje ćemo ih još jednom navesti teorijski i ukratko objasniti.

Koristit ćemo sljedeće oznake za predloške:

- **WCP** (*Workflow Control Pattern*) - kratica za definiranje predloška
- **A** - aktivnost

- - polje
- - entitet na informacijskom vezi
- - skretnica

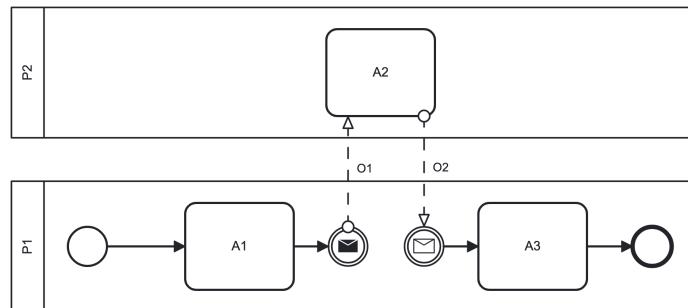
## WCP-1 Slijed (eng. Sequence)

Neka aktivnost (npr. ) može započeti ako je završena aktivnost koja joj prethodi (npr. ).



Primjer predloška WCP-1: Slijed između dvije aktivnosti

Ipak, treba podsjetiti na to kako aktivnosti modelirati kada ih izvode različiti sudionici, u različitim poljima. Koristimo za komunikaciju između polja te odgovarajuće **međudogađaje**:



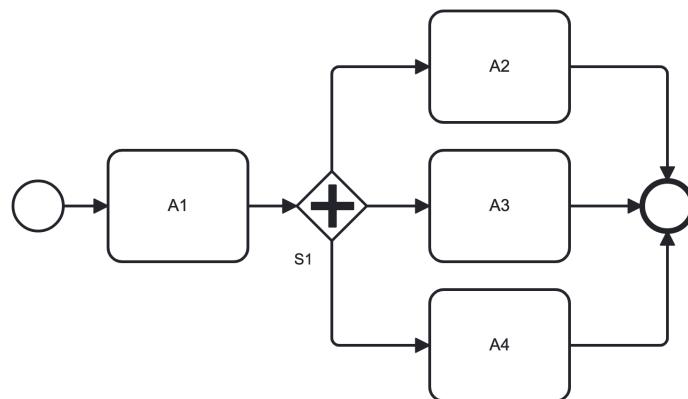
Primjer predloška WCP-1: Slijed između dvije aktivnosti u različitim poljima

## WCP-2 Paralelno dijeljenje (eng. Parallel Split)

Nakon neke aktivnosti, proces se dijeli u više paralelnih grana. To znači da nakon završetka mogu započeti aktivnosti i i te se obavljati istodobno, a iza svake od njih može slijediti neka druga aktivnost.

**Mogući početak istovremene aktivnosti ne implicira njihov istovremeneni završetak!**

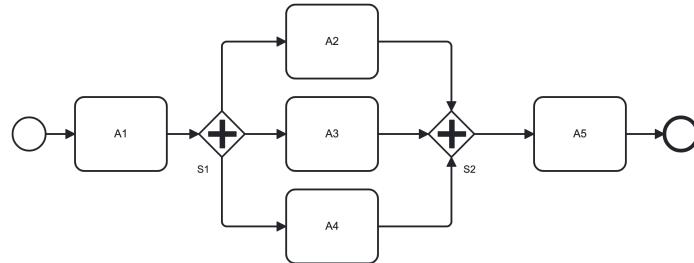
Od jedne značke (eng. token) koja ulazi u paralelnu skretnicu , uvijek se stvara (bez provjere uvjeta) onoliko kopija koliko ima izlaznih grana i svaka od tih kopija značke dalje se kreće po jednoj od paralelnih grana.



Primjer predloška WCP-2: Paralelno dijeljenje

## WCP-3 Sinkronizacija (eng. Synchronization)

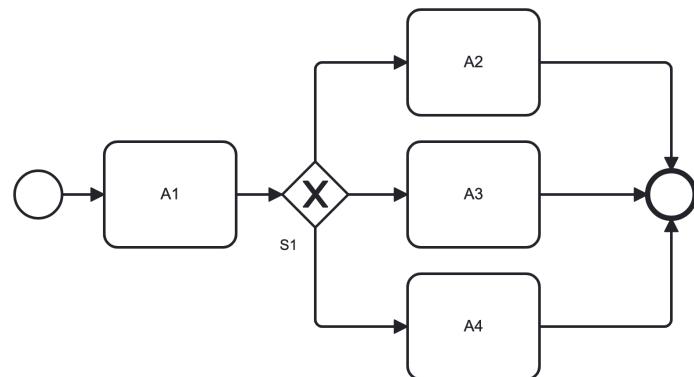
Neka aktivnost može početi ako su prije nje završene aktivnosti na svim paralelnim granama (mogu biti dvije ili više). To znači da aktivnost  $A_5$  može započeti tek nakon što su završene aktivnosti  $A_2$ ,  $A_3$  i  $A_4$ . U **paralelnoj skretnici spajanja S2** (eng. AND Merge) sve se ulazne značke uvijek spajaju, bez provjere uvjeta, u jednu izlaznu.



Primjer predloška WCP-3: Sinkronizacija

## WCP-4 Ekskluzivni izbor (eng. Exclusive Choice)

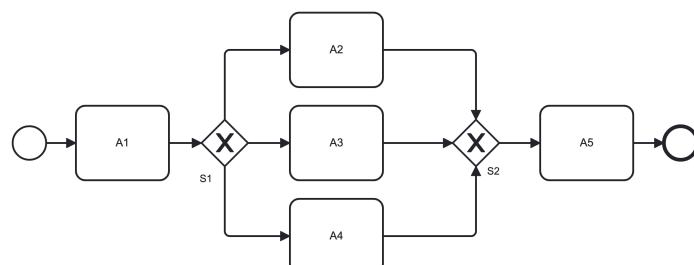
Nakon neke aktivnosti proces će se nastaviti **samo u jednoj** od više mogućih grana. To znači da će nakon  $A_1$  biti izvedena aktivnost  $A_2$  ili  $A_3$  ili  $A_4$  (odnosno slijed kojem su one na početku). Značka koja ulazi u ekskluzivnu XOR skretnicu  $S_1$  ne dijeli se, već nastavlja jednim od putova koji udovoljava uvjetu što se ispituje prije te skretnice.



Primjer predloška WCP-4: Ekskluzivni izbor

## WCP-5 Jednostavno spajanje (eng. Simple Merge)

Neka aktivnost može početi čim je izvedena neka od aktivnosti koje su se izvodile u dva ili više paralelnih sljedova. To znači da aktivnost  $A_5$  može započeti kad završe ili  $A_2$  ili  $A_3$  ili  $A_4$  (odnosno slijed kojem su one bile na kraju).. Aktivnost  $A_5$  će pokrenuti ona značka koju je ekskluzivna skretnica  $S_1$  uputila na neki od sljedova, a koje je prošla kroz ekskluzivnu skretnicu spajanja  $S_2$ .



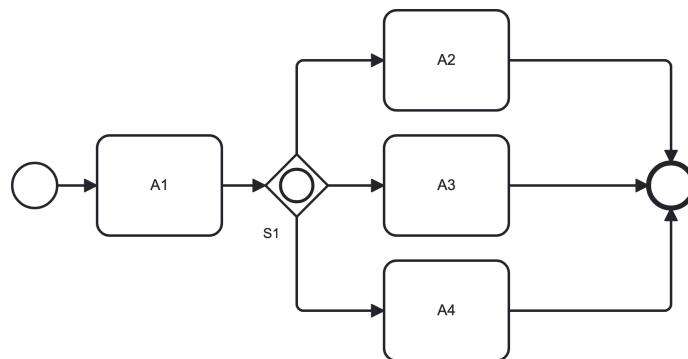
Primjer predloška WCP-5: Jednostavno spajanje

## 2.2 Predlošci za grananje, sinkronizaciju i iteraciju

U ovoj grupi su predlošci koji se koriste za grananje i sinkronizaciju toka izvođenja aktivnosti. Uobičajeno se koriste u situacijama kada je potrebno izvršiti nekoliko aktivnosti istovremeno ili kada se proces nastavlja samo ako su završene sve aktivnosti koje su se izvodile u paralelnim granama.

### WCP-6 Višestruki izbor (eng. Multiple Choice)

Nakon neke aktivnosti proces se može nastaviti u jednoj, dvjema ili u više mogućih grana, **ali najmanje u jednoj**. To znači da poslije **A1** može biti izvedena bilo koja aktivnost, ili bilo koje dvije aktivnosti ili sve tri aktivnosti od mogućih **A2**, **A3** i **A4**.

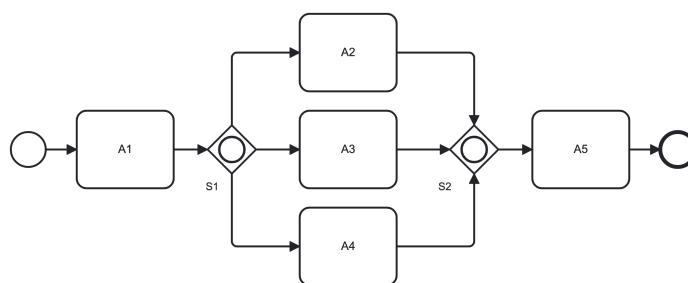


Primjer predloška WCP-6: Višestruki izbor

### WCP-7 Struktorno sinkronizirano spajanje (eng. Structured Synchronizing Merge)

Neka aktivnost može početi ako su izvedene sve aktivnosti koje su se izvodile u dvama ili u više paralelnih sljedova stvorenih ranije u procesu. To znači da **A5** može započeti kad je završila jedna ili više aktivnosti od mogućih **A2**, **A3** i **A4** koje su pokrenule kopije značaka stvorene u inkluzivnoj skretnici grananja (**S1**). Drugim riječima, u **S2** se sinkroniziraju (ili spajaju) kopije onih značaka koje su prije toga stvorene u **S1**. Bez obzira na to koliko je kopija značaka ušlo u izlaznu skretnicu **S2**, izaći će samo jedna.

U poslovnom smislu to znači da će se procesna instanca, koja je obrađena u **A1**, moći obraditi u **A5** nakon što je provedena barem jedna ili više aktivnosti iz skupa **A2**, **A3** i **A4**.

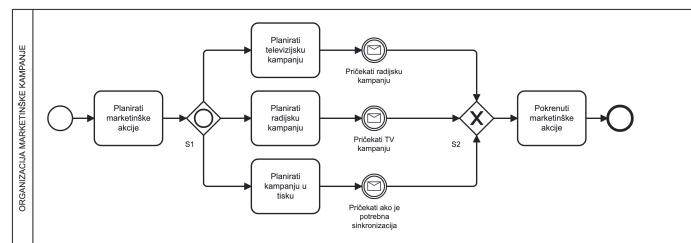


Primjer predloška WCP-7: Struktorno sinkronizirano spajanje

### WCP-8 Nesimetrično sinkronizirano spajanje (eng. Acyclic Synchronizing Merge)

Neka aktivnost može početi ako su izvedene sve aktivnosti na dva ili više paralelnih sljedova, stvorenih ranije u procesu na inkluzivnoj skretnici **S1** ali se odluka o tome što treba spajati odnosi na temelju **međudogađaja** koji prethode ekskluzivnoj skretnici spajanja **S2**.

**Nesimetrično spajanje** riješeno je kombinacijom inkluzivne skretnice **S1** (koja će stvoriti jednu, dvije ili tri značke na bilo kojem od tri slijeda) te uvjetovanih događaja na sva tri slijeda ispred konvergentne ekskluzivne skretnice **S2**. Ti će uvjetovani događaji dopustiti izvođenje aktivnosti POKRENUTI MARKETINŠKE AKCIJE kad završe one od prethodnih aktivnosti koje se moraju uskladiti.



Primjer predloška WCP-8 na primjeru procesa organizacije marketinške kampanje: Nesimetrično sinkronizirano spajanje

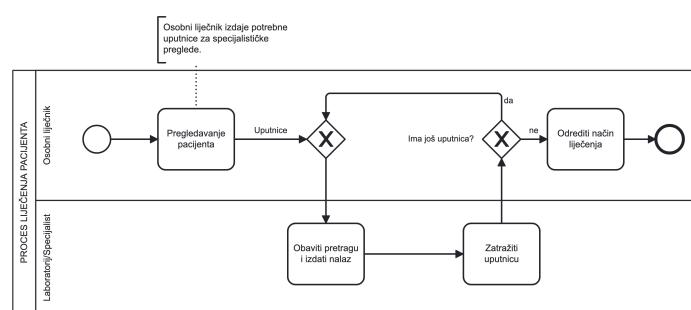
Izvođenje aktivnosti u složenom poslovnom procesu slično je izvođenju procedura u složenom programu. Simboli BPMN-a omogućuju prikaz takvih struktura kao što su **GOTO**, **WHILE...DO**, **REPEAT...UNTIL** u programiranju.

Međutim, u poslovnoj praksi česte su i druge strukture, nepoznate u strukturnom programiranju, koje opisuju ponavljanje odnosno **iteracije** pojedinačne aktivnosti ili grupe aktivnosti.

## WCP-9 Proizvoljno ponavljanje (eng. Arbitrary Cycles)

Ovaj predložak opisuje točku u procesu nakon koje se može ponoviti jedna ili više aktivnosti. Općenito, unaprijed se ne zna treba li uopće nešto ponavljati i ako treba - koliko puta, već je to specifično za svaku instancu procesa pa se stoga to naziva još i **nestrukturiranom petljom**.

Tipičan primjer za ovaj predložak može se pronaći u zdravstvu, a prikazan je na sljedećem primjeru:



Primjer predloška WCP-9: Proizvoljno ponavljanje

Primarna zdravstvena zaštita kod nas funkcioniра tako da pacijent najprije odlazi na pregled svom osobnom liječniku. Osobni liječnik nakon pregleda odlučuje koje su dodatne pretrage potrebne te za njih izdaje uputnice. Laboratorij ili specijalist će "Obaviti pretragu i izdati nalaz" te zadržati uputnicu (radi obračuna usluge), a pacijent (ako ima još uputnica) će otići na sljedeću pretragu. Osobni će liječnik "Odrediti način liječenja" na temelju nalaza u provedenim pretragama. Općenito se ne zna koliko laboratorijskih pretraga treba napraviti, već će se napraviti onoliko pretraga koliko je potrebno točno određenom pacijentu i primjerenoj njegovoj bolesti.

Ovo je zanimljiv primjer proizvoljnog ponavljanja gdje XOR skretnica spajanja **prethodi** XOR skretnici grananja.

## 2.3 Predlošci za okidače

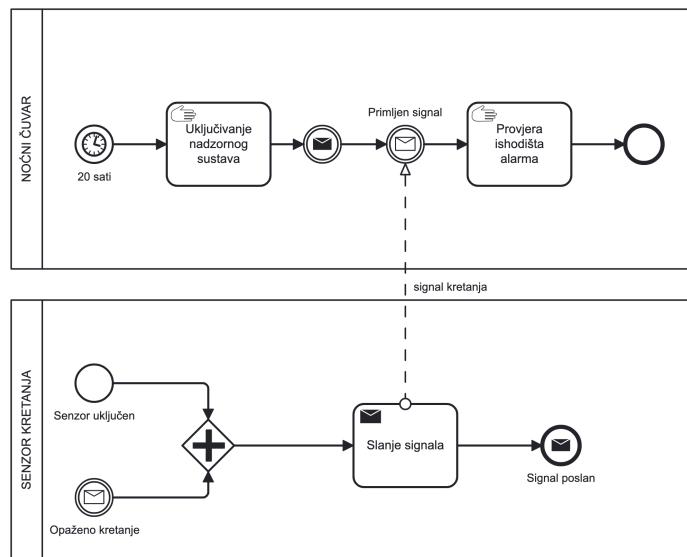
U ovom potpoglavlju prikazat ćemo nekoliko predložaka koji se koriste za modeliranje okidača u poslovnim procesima. **Okidači** su događaji koji pokreću izvođenje procesa, a mogu biti izazvani **vremenski, porukom ili signalom**.

### WCP-10 Prolazni okidač (eng. Transient Trigger)

Predložak opisuje proces u kojem izvođenje jedne aktivnosti ovisi o nekom vanjskom poticaju ili drugom procesu (odnosno, vanjski poticaj "okida" aktivnost).

Okidač koji to omogućuje zovemo prolaznim jer nestaje ako u osnovnom procesu već ne čeka instanca koja bi se mogla pokrenuti. Prolazni okidač zapravo je običan prijamni međudogađaj (npr. [Message Intermediate Catch Event](#)) koji se koristi za hvatanje signala ili poruka od drugih procesa.

Primjer opisuje rad noćnog čuvara u nadziranom objektu. Čuvar će se poslije dolaska (u 20 sati) smjestiti u kontrolnu sobu i "Uključiti nadzorni sustav" koji se sastoji od kamere i senzora kretanja. Ako senzor registrira pokret u objektu, on će "Poslati signal" u kontrolnu sobu. Ako je čuvar u sobi, on će "Provjeriti ishodište alarma". Ako pak čuvara nema, poslani signal neće biti iskorišten i propast će (zato ga zovemo prolaznim).



Primjer predloška WCP-10: Prolazni okidač

### WCP-11 Stalni okidač (eng. Persistent Trigger)

Izvođenje aktivnosti u ovom predlošku također ovisi o nekom vanjskom poticaju ili drugom poslovnom procesu (vanjski poticaj okida aktivnost). Okidač djeluje stalno i aktivan je sve dok na njega dolaze instance procesa, a modelira se također kao prijamni događaj (npr. [Message Intermediate Throw Event](#)) u osnovnom procesu.

Ishodište iz kojeg dolazi poticaj i ovdje se modelira kao predajni međudogađaj (npr. [Message Intermediate Throw Event](#)) koji šalje ciljanu obavijest određenom okidaču.

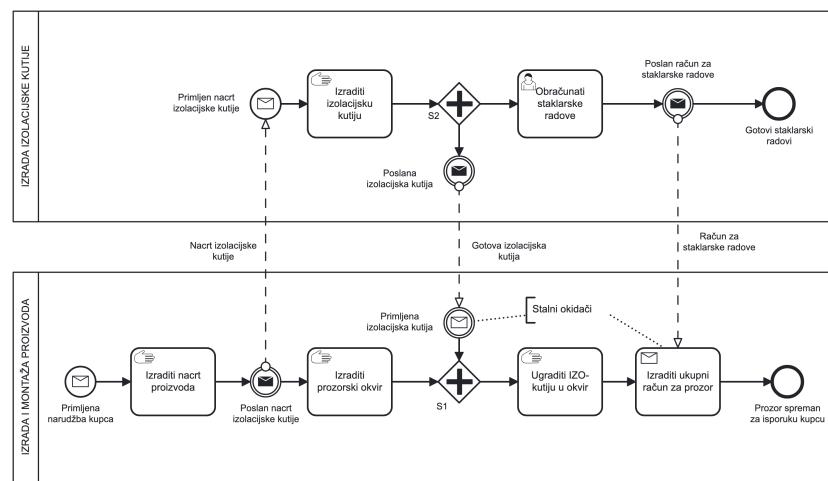
Stalni okidač djeluje slično kao prolazni, a **razlika je u tome što se vanjski poticaj ne gubi ako u osnovnom procesu trenutno nema instance koja bi na njega čekala**. Sljedeći primjer pokazuje izradu prozora. Prozor se izrađuje u tri faze: a) prozorski okvir od drva ili profilirane plastike, b) izolacijska kutija od dvije ili tri staklene ploče između kojih je inertni plin, a razmak održavaju letvice s brtvom i c) ugradnja izolacijske kutije u pripremljeni prozorski okvir.

Zbog različitih tehnologija u fazama a) i b) ti se poslovi organiziraju u dvije radionice. Prvi ćemo odjel nazvati IZRADA I MONTAŽA PROIZVODA, a drugi je staklarska radionica IZRADA IZOLACIJSKE KUTIJE. Ovdje smatramo da prvi odjel vodi posao (među kojima su kontakti s kupcima), a drugi da je kooperant (*outsourced*).

Budući da prvi odjel primi narudžbu, on će "Izraditi nacrt proizvoda" i kopiju poslati staklarskoj radionici te nastaviti s aktivnošću "Izraditi prozorski okvir". Staklarska će radionica prema nacrtu "Izraditi izolacijsku kutiju" i poslati je vodećem odjelu koji, nakon primitka gotove izolacijske kutije, može "Ugraditi izo-kutiju u okvir". Dakle, prijamni događaj "Primljena izolacijska kutija" **okidač** je za ovu aktivnosti. Upravo se u ovom detalju vidi bitna razlika između prolaznog i stalnog okidača: vanjski poticaj (ovdje je to tok "Gotova izolacija kutija") neće se izgubiti ako u okidaču "Primljena izolacijska kutija" još nema odgovarajuće instance procesa (odnosno gotovoga prozorskog okvira) već će se iskoristiti (ovdje to znači ugraditi) kad najde ta instanca (odnosno kad prozorski okvir bude gotov).

Analizom modela može se utvrditi da su u procesu zapravo **dva stalna okidača**.

Prvi smo već naveli i on je modeliran eksplisitno. Međutim, drugi okidač modeliran je implicitno i određen svojstvom prijamne aktivnosti "Izraditi ukupni račun za prozor". U tu aktivnost ulazi poruka (entitet) - "Račun za staklarske radove" iz emitirajućeg međudogađaja "Poslan račun za staklarske radove" i pokreće se ("okida") njezino izvođenje.



Primjer predloška WCP-11: Stalni okidač

# Upravljanje poslovnim procesima (UPP)

**Nositelj:** izv. prof. dr. sc. Darko Etinger

**Asistent:** Luka Blašković, mag. inf.

**Ustanova:** Sveučilište Jurja Dobrile u Puli, Fakultet informatike u Puli



Fakultet informatike u Puli

## (5) Uvod u procesne aplikacije

#5

UPP

Procesne aplikacije omogućuju automatizaciju poslovnih procesa korištenjem definiranih modela i pravila. Na ovom kolegiju naučili ste kako definirati poslovne procese korištenjem BPMN notacije, a sada ćete se upoznati s alatom Camunda 7 koji omogućuje izvođenje (egzekuciju) tih procesa. Camunda 7 je open-source platforma koja koristi BPMN za vizualno modeliranje procesa te pruža mehanizme za njihovo izvršavanje, nadzor i upravljanje. Primjenom Camunda, organizacije mogu optimizirati svoje poslovne procese i povećati učinkovitost poslovanja kroz automatizaciju zadataka i transparentno praćenje tijeka procesa.

Posljednje ažurirano: 13.12.2024.

### Sadržaj

- [Upravljanje poslovnim procesima \(UPP\)](#)
- [\(5\) Uvod u procesne aplikacije](#)
  - [Sadržaj](#)
- [1. Uvod u procesne aplikacije](#)
- [2. Camunda 7](#)
  - [2.1 Pokretanje preko Dockera](#)
- [3. Osnovne komponente Camunda 7 platforme](#)
  - [3.1 Camunda Cockpit](#)
    - [3.1.1 Egzekucija vlastitog procesa](#)
    - [3.1.2 User Task i forme](#)
  - [3.2 Camunda Tasklist](#)
    - [3.2.1 Procesne varijable i dodavanje XOR skretnice](#)
    - [3.2.2 Kako još možemo izraditi instance?](#)

- [4. Obrada vrijednosti procesnih varijabli u procesu](#)
- [Samostalni zadatak za Vježbu 5](#)

# 1. Uvod u procesne aplikacije

Od početka razvoja BPMN-a isticalo se ostvarenje dvaju (prividno) međusobno teško uskladivih ciljeva:

1. **osigurati** da se normom služe poslovni stručnjaci koji ne razvijaju aplikacije i
2. **omogućiti** softverskim inženjerima da procesni model, izведен po toj normi, preslikaju u izvršnu aplikaciju primjerenu potrebama stvarnoga poslovnog procesa.

Drugim riječima, važna namjena BPMN-a jest premošćivanje jaza u sporazumijevanju između poslovnih i informatičkih stručnjaka 😊

Too often tension exists between the developer and analyst perspectives, resulting from the lack of a common semantics and heuristics set capable of depicting process activities in a way relevant to both parties.

Promatramo li BPMN 2.0 normu općenito, s odmakom od formalno izrečenih logičkih i tehničkih pojedinosti, možemo zaključiti da ona ima sljedeća svojstva:

- sadržava skup **pravila i simbola** za modeliranje poslovnih procesa i omogućuje različite oblike za grafičko predstavljanje procesa, primjereno namjeni
- detaljno razrađen **grafički model** poslovnog procesa može se pretvoriti u izvršiv oblik i na temelju toga razviti potrebna softverska rješenja.
- pogodan za **zajednički jezik za sporazumijevanje** između poslovnih stručnjaka, **analitičara procesa i softverskih inženjera**.

**Procesna aplikacija (PA)** (*eng. process application*) se temelji na tijeku rada, odnosno može se reći da je svaka PA procesno usmjerena (*eng. workflow-oriented*). To je najšira definicija procesne aplikacije. Za preciziranje te definicije prikladno je reći što PA nije, odnosno po čemu se razlikuje od ostalih, podatkovno usmjerениh aplikacija:

**Klasične aplikacije (ne PA)** imaju sljedeća tipična svojstva:

- Funkcionalnosti, koje se ukratko mogu opisati izrekom "**upiši u bazu, pročitaj iz baze**", definirane su stanjem podataka nakon što su izvedene određene aktivnosti ili proveden cijeli proces
- **Redoslijed izvođenja aktivnosti (ili tok rada) implicitno je sadržan u aplikaciji**, obično određen programiranim redoslijedom prikaza korisničkih poziva, odnosno poziva programske sučelja
- **Aktivnosti i procesi ne postoje kao aplikacijski entiteti**
- **Arhitektura klasične aplikacije prilagođena je funkcionalnim CRUD** (*eng. Create, Read, Update, Delete*) potrebama odnosno stvaranju, čitanju, ažuriranju i brisanju podatkovnih zapisa
- Pri izmjeni poslovnih procesa (npr. zbog promjene zakonske regulative), klasične aplikacije treba **temeljito reprogramirati**, napose komponente njihove poslovne logike (*eng. business logic layer*)

S druge strane, **procesne aplikacije (PA)** imaju sljedeća tipična svojstva:

- **Funkcionalnosti su definirane tijekom rada koji aplikacija mora podržavati. Ishodište je za razvoj procesne aplikacije model procesa, dopunjen tako da bude u grupi izvršivih modela**
- Tijekovi rada u aplikaciji eksplicitni su i **neovisni o korisničkim i programskim sučeljima**

- **Aktivnosti i procesi određeni su kao aplikacijski entiteti** čijim se stanjima i definicijom izravno upravlja
- Arhitektura procesne aplikacije prilagođena je reagiranjem na **događaje** (eng. event-driven) i poruke (eng. message-driven) te **upravljanjem tijekom rada** (eng. workflow management)
- Procesne aplikacije mogu sadržavati jednako **korisničke aktivnosti** (eng. user tasks) i **automatizirane aktivnosti** (eng. service tasks)
- Procesne aplikacije **podržavaju više organizacijskih jedinica u organizaciji** i povezuju ih u cjelovit proces koji kupcu ili korisniku daje traženi proizvod ili uslugu
- **Procesne su aplikacije prilagodljive promjenama poslovnih procesa** jer nakon takvih izmjena ne treba reprogramirati aplikacijske komponente, nego procesnu aplikaciju samo opskrbiti ažuriranom definicijom aktivnosti i/ili procesa.

Razlike između klasičnih i procesnih aplikacija mogu se sažeti u sljedećoj tablici:

Svojstva aplikacije	Klasična aplikacija (OLTP ili ERP)	Procesna aplikacija
Funkcionalnosti	Definirane stanjem podataka na kraju posla	Definirane stanjem eksplicitno navedenih radnih aktivnosti. Ishodište za razvoj PA je <b>izvršivi model procesa</b>
Funkcionalna sintagma	"Upiši u bazu, pročitaj iz baze"	"Slijedni najbolji radni tok."
Aktivnosti i procesi	Ne postoji kao programske entitete	<b>Postoje kao programski entiteti</b> kojima se izravno upravlja
Arhitektura	Prilagođena CRUD operacijama	Prilagođena <b>reagiranjima na događaje i poruke</b> (event-driven & message driven)

**Napomena:** U praksi, granica između funkcionalnosti klasične i procesne aplikacije nije uvijek crno-bijela.

#### Primjer ove diferencijacije na webshop aplikaciji:

- **Klasična aplikacija** (zamišljamo u kontekstu kolegija *Programsko Inženjerstvo* ili *Web aplikacije*):
  - Funkcionalnosti implementiramo *low-level* programiranjem gdje razmišljamo o CRUD operacijama nad bazom podataka
  - *Primjer 1:* "Korisnik se registrira i pregledava proizvode" → CRUD operacije nad tablicama `users` i `products`, razvoj korisničkog sučelja, razvoj korespondirajućeg backenda za validaciju podataka i sl.
  - *Primjer 2:* "Korisnik dodaje proizvode u košaricu i obavlja kupnju" → CRUD operacije nad tablicama `cart` i `orders`, razvoj korisničkog sučelja, razvoj odgovarajućih backend komponenti, spajanje na vanjske servise za plaćanje i sl.
  - **Ključno: Aplikacija se organizira oko podataka i operacijama nad njima**

- **Procesna aplikacija** (zamišljamo u kontekstu ovog kolegija):
  - Funkcionalnosti implementiramo *high-level* programiranjem gdje razmišljamo o **tijeku rada i aktivnostima koje korisnik treba obaviti**
  - *Primjer 1:* "Korisnik se registrira i pregledava proizvode" → Procesna aplikacija definira radne korake koje korisnik treba obaviti, npr. "Registracija korisnika", "Pregled proizvoda", "Dodavanje proizvoda u košaricu". Korake definiramo kroz neki **procesni model** (u našem slučaju **BPMN**, ali može biti i drugi)
  - **Aplikacija reagira na vanjske događaje i poruke** (npr. "Korisnik je pokrenuo proces narudžbe", "Plaćanje uspješno", "Proizvod je otpremljen") → *event-driven* i *message-driven* programiranje
  - Proces narudžbe zamišljamo kao **jedan entitet** koji se sastoji od više koraka (aktivnosti) koje korisnik treba obaviti, gdje određeni koraci mogu biti različitih tipova (korisničke aktivnosti, automatizirane aktivnosti, itd.).
  - **Ključno: Jedno započinjanje aktivnosti nazivamo pokretanje procesne instance.** Npr. "korisnik Pero Perić započinje proces narudžbe u webshopu". Samim time, svaki korisnik procesne aplikacije koji započne proces ima svoju **instancu procesa**. Stanje instance procesa može se ponovno izgraditi u svakom trenutku koristeći tzv. *event logove*.

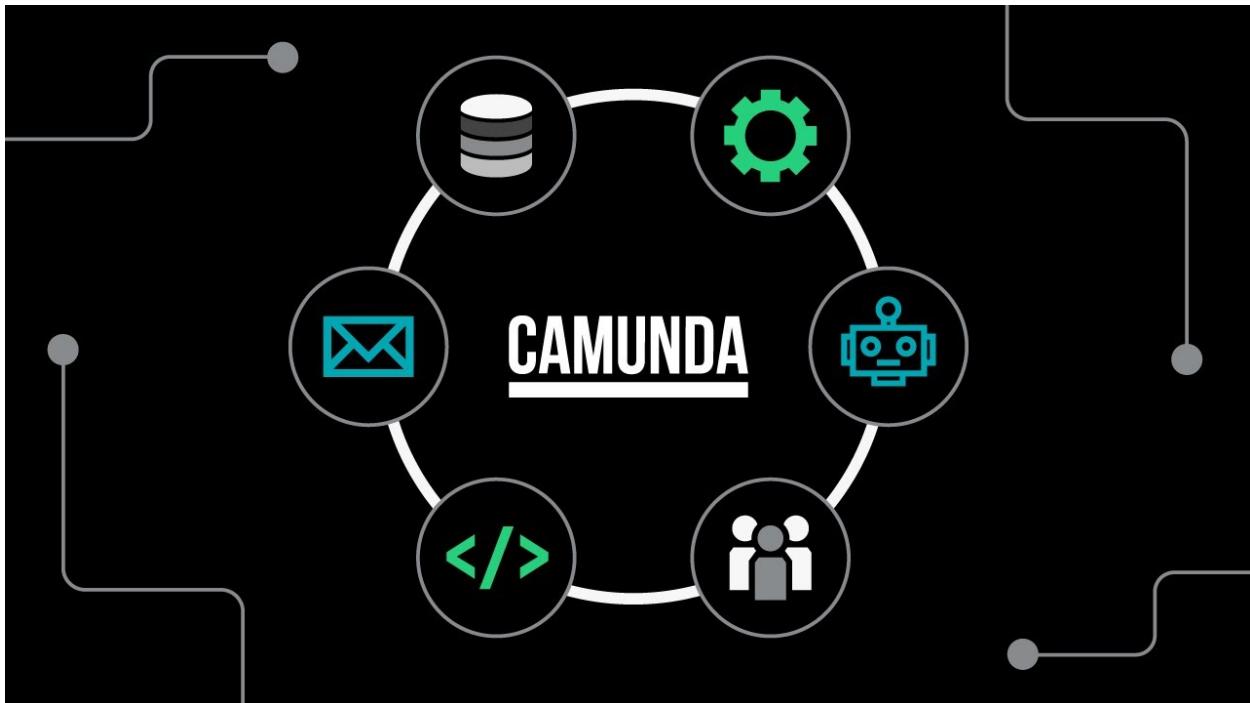
Ipak, svim komercijalnim alatima za razvoj procesnih aplikacija, kao što je Camunda, zajedničko je sljedeće svojstvo: **procesne aplikacije izvode se kao web aplikacije**.

U kontekstu ovog kolegija, procesne aplikacije ćemo interpretirati kao web aplikacije koje koriste Camunda 7 platformu za izvođenje poslovnih procesa definiranih BPMN 2.0 normom.

Do sad smo naučili kako ispravno modelirati procese u BPMN notaciji, a sada ćemo se upoznati s alatom Camunda 7 koji omogućuje izvođenje (egzekuciju) tih procesa.

## 2. Camunda 7

Camunda 7 je *open-source* platforma koja koristi BPMN za vizualno modeliranje procesa te pruža mehanizme za njihovo **izvršavanje, nadzor i upravljanje**. Primjenom Camunde, organizacije mogu optimizirati svoje poslovne procese i povećati učinkovitost poslovanja kroz automatizaciju zadataka i transparentno praćenje tijeka procesa.



Do sad ste koristili [Open Source Desktop Modeler](#) za modeliranje poslovnih procesa u BPMN notaciji, sada ćemo se upoznati s dodatnim komponentama Camunda platforme:

- **BPMN Workflow Engine**
- **DMN Decision Engine**
- **Tasklist**
- **Cockpit i Admin**

Camunda 7 nudi besplatni Community Edition paket koji uključuje sve potrebne komponente za upoznavanje procesnih aplikacija i njihovu egzekuciju lokalno na računalu, dok Enterprise Edition paket nudi dodatne mogućnosti za upravljanje i nadzor poslovnih procesa u velikim organizacijama, optimizacijske tehnike i naprednije sigurnosne mehanizme.

Također treba naglasiti da je Camunda 7 platforma EOL (End of Life) te je zadnje ažuriranje dobila u 10. mjesecu 2024. godine. Međutim, radi se o dobro razrađenom softveru koji se i dalje može koristiti za učenje i razvoj procesnih aplikacija. Camunda aktivno radi na razvoju nove verzije Camunda 8 koja donosi brojne nove mogućnosti.

Radi jednostavnije instalacije, ali i **dostupnosti materijala za učenje**, preporučuje se korištenje **Camunda 7** platforme do daljnega.

### 2.1 Pokretanje preko Dockera

Camunda 7 platformu možete vrlo jednostavno pokrenuti lokalno preko [gotovog Docker kontejnera](#).

**Docker** je besplatna platforma koja omogućuje razvoj, postavljanje i pokretanje aplikacija u kontejnerima (*eng. [Containerization](<https://en.wikipedia.org/wiki/Containerization(computing)>)*). Kontejneri omogućuju pakiranje i izvršavanje aplikacija u izoliranom okruženju, što znači da se aplikacija može pokrenuti na bilo kojem računalu koje ima Docker instaliran, bez obzira na okruženje.

Prvi korak je preuzimanje **Docker Desktop** aplikacije sa [službene stranice](#).

Ako ste na Windows OS-u, Docker Desktop zahtjeva instalaciju WSL-2 (Windows Subsystem for Linux) koji se može instalirati preko PowerShell naredbe:

```
wsl --install
```

Dodatno, moguće je omogućiti **Hyper-V i podršku za virtualizaciju** u BIOS-u računala. Ovisno o proizvođaču matične ploče, postupak se razlikuje, ali BIOS-u se obično pristupa pritiskom tipke **F2**, **F10**, **F12** ili **DEL** prilikom pokretanja računala.

Najbolji način je pretražiti na internetu kako pristupiti BIOS-u na vašem računalu te kako omogućiti Hyper-V i virtualizaciju.

Dakle, na Windowsu, Docker Desktop zahtjeva instalaciju WSL-2 **ili** Hyper-V.

## System requirements



### Should I use Hyper-V or WSL?

Docker Desktop's functionality remains consistent on both WSL and Hyper-V, without a preference for either architecture. Hyper-V and WSL have their own advantages and disadvantages, depending on your specific set up and your planned use case.

[WSL 2 backend, x86\\_64](#)   [Hyper-V backend, x86\\_64](#)   [WSL 2 backend, Arm \(Beta\)](#)

- WSL version 1.1.3.0 or later.
- Windows 11 64-bit: Home or Pro version 22H2 or higher, or Enterprise or Education version 22H2 or higher.
- Windows 10 64-bit: Minimum required is Home or Pro 22H2 (build 19045) or higher, or Enterprise or Education 22H2 (build 19045) or higher.
- Turn on the WSL 2 feature on Windows. For detailed instructions, refer to the [Microsoft documentation](#).
- The following hardware prerequisites are required to successfully run WSL 2 on Windows 10 or Windows 11:
  - 64-bit processor with [Second Level Address Translation \(SLAT\)](#)
  - 4GB system RAM
  - Enable hardware virtualization in BIOS. For more information, see [Virtualization](#).

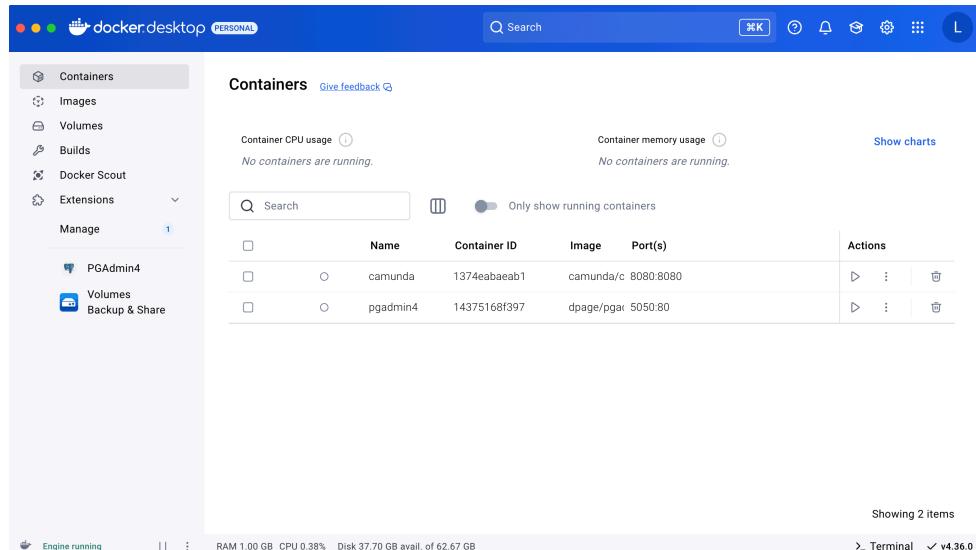
Upute za instalaciju Docker Desktop na Windows OS-u, dostupno na: <https://docs.docker.com/desktop/setup/install/windows-install/>

Docker je moguće koristiti i na **Linux** (dostupno za: Ubuntu, Debian, RHEL, Fedora) i **macOS** (dostupno za: Apple silicon, Intel chip) operacijskim sustavima bez dodatnih postavki. [Na Linuxu možete instalirati Docker i bez grafičkog sučelja preko terminala](#), međutim za početnike je preporuka instalirati grafičko sučelje - **Docker Desktop**.

Nakon što ste uspješno instalirati **Docker Desktop**, provjerite je li uspješno instaliran preko naredbe:

```
docker --version
```

Pokrenite Docker Desktop aplikaciju i prijavite se s vašim Docker računom. Ako nemate Docker račun, možete ga besplatno kreirati na [Docker Hub-u](#).



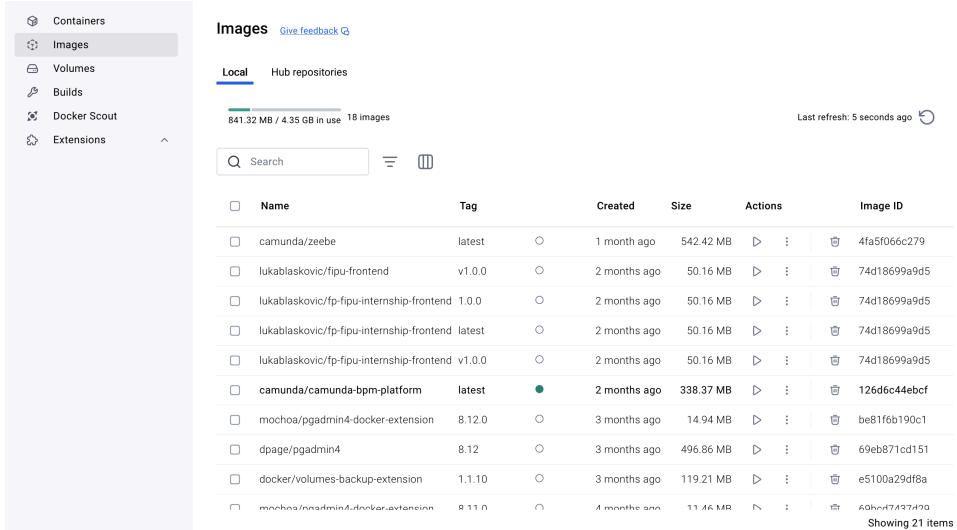
### Grafičko sučelje Docker Desktop aplikacije

Grafičko sučelje Docker Desktop aplikacije sastoji se od nekoliko osnovnih elemenata:

1. **Container** - prikaz svih pokrenutih kontejnera (dva stanja: **Running** i **Exited**). **Kontejner** smo rekli da je svaka aplikacija koja se pokreće u izoliranom okruženju. U ovom slučaju, to će biti Camunda 7 platforma.
2. **Images** - prikaz svih preuzetih Docker "slika" (eng. *Docker images*). **Docker image je predložak za pokretanje kontejnera**.
3. **Volumes** - prikaz svih Docker "volumena" (eng. *Docker volumes*). **Docker volume koristi se za trajno pohranjenje podataka, obzirom da se podaci unutar kontejnera brišu prilikom gašenja kontejnera**.
4. **Builds** - prikaz svih provedenih Docker "buildova" (eng. *Docker builds*). Ovdje su pohranjeni svi Docker buildovi, uspješni i neuspješni.
5. **Docker Scout** - napredna analiza docker images, u svrhu pronalaska ranjivosti (eng. *vulnerabilities*). Za početnike, ovo nije bitno.
6. **Extensions** - dodatne ekstenzije za Docker Desktop aplikaciju. Za početnike, ovo nije bitno.

U pravilu, za sada će nam samo biti bitni **Container** i **Images** tabovi.

**VAŽNO!** Kontejneri se uvijek pokreću preko odgovarajućeg image-a, gdje image predstavlja predložak za pokretanje kontejnera, a kontejner predstavlja realiziranu sliku (instancu) tog predloška.



Prikaz svih preuzetih Docker "slika" (eng. *Docker images*)

Docker Desktop aplikacija prikazuje sve definirane Docker images koje izradimo **(1) lokalno putem terminala ili (2) koje preuzimamo s [Docker Hub-a](#).**

Docker images lokalno gradimo koristeći **Dockerfile** datoteku, koja definira korake kontejnerizacije naše aplikacije. Ovime ćemo se detaljno baviti na kolegiju Raspodijeljeni sustavi na 1. godini diplomskog studija.

Za sada nas zanima samo dio koji se odnosi na preuzimanje gotove Docker "slike" s Docker Hub-a. **Docker Hub** je centralno mjesto za pronašljak i dijeljenje Docker images, slično kao što je GitHub centralno mjesto za pronašljak i dijeljenje izvornog koda (u obliku repozitorija).

Sliku s Docker Hub-a možemo preuzeti preko naredbe u terminalu (možete bilo gdje otvoriti terminal)

```
docker pull <image-name>:<tag>
```

Preuzet ćemo sljedeći [Camunda 7 image](#):

```
docker pull camunda/camunda-bpm-platform:latest
```

Nakon što preuzmemos sliku, možemo je pokrenuti preko naredbe:

```
docker run -d --name camunda -p 8080:8080 camunda/camunda-bpm-platform:latest
```

- `-d` - pokreće kontejner u pozadini (eng. *detached mode*)
- `--name camunda` - dodjeljuje naziv kontejneru
- `-p 8080:8080` - mapira port `8080` na lokalnom računalu (vašem) na port `8080` unutar kontejnera
- `camunda/camunda-bpm-platform:latest` - image koji pokrećemo

Nakon što pokrenemo kontejner, možemo provjeriti je li kontejner uspješno pokrenut preko Docker Desktop aplikacije.

The screenshot shows the Docker interface. On the left, a sidebar lists 'Containers', 'Images', 'Volumes', 'Builds', 'Docker Scout', and 'Extensions'. The main area is titled 'Containers' with a 'Give feedback' link. It displays 'Container CPU usage' at 163.39% / 800% (8 CPUs available) and 'Container memory usage' at 486MB / 7.57GB. A search bar contains 'camunda'. Below is a table with columns: Name, Container ID, Image, Port(s), and Actions. One row is shown for 'camunda' with Container ID 2e6e12144778, Image camunda/camunda-bpm-platform:latest, and Port(s) 8080:8080. The 'Actions' column includes a stop button, a three-dot menu, and a trash icon.

Kontejner je uspješno pokrenut i radi na portu 8080

Kontejner je, osim preko terminala, **moguće pokrenuti direktno iz grafičkog sučelja**, međutim onda je ove dodatne postavke potrebno unijeti u grafičkom sučelju.

### Praktičnije je i brže naučiti osnovne Docker naredbe u terminalu

CLI Cheat Sheet za Docker možete preuzeti na sljedećoj poveznici: [https://docs.docker.com/get-started/docker\\_cheatsheet.pdf](https://docs.docker.com/get-started/docker_cheatsheet.pdf)

Nakon što je kontejner uspješno pokrenut, možemo pristupiti Camunda 7 platformi preko web preglednika na adresi: <http://localhost:8080/camunda>

Otvorite u web pregledniku adresu: <http://localhost:8080/camunda-welcome/index.html>. Ako je kontejner uspješno pokrenut, trebali biste vidjeti sljedeći prikaz:

The screenshot shows the Camunda Platform welcome page. At the top, a red banner reads: 'Heads-up – it is time to try out Camunda 8. The last release of Camunda 7 CE will be in October 2025. If you have any more questions, please get in touch with us in the forum.' Below the banner, there are four main sections: 'Tasklist' (showing a task list for 'Approve Invoice'), 'Cockpit' (showing process instances, events, and metrics), 'Admin' (showing application and authorization management), and 'Welcome' (showing user profile and navigation links). The 'Enterprise Support' button is located in the top right corner.

Camunda 7 platforma pokrenuta lokalno preko Docker kontejnera: <camunda/camunda-bpm-platform:latest>

# 3. Osnovne komponente Camunda 7 platforme

Camunda 7 platforma koju ste pokrenuli sastoji se nekoliko ključnih komponenti:

- **Workflow Engine:** Pozadinski "motor" koji izvršava poslovne procese definirane BPMN 2.0 normom
- **Cockpit:** Monitoring i nadzor poslovnih procesa i aktivnih instanci
- **Tasklist:** Popis zadataka koje korisnici trebaju obaviti (User Tasks)
- **Admin:** Upravljanje korisnicima (User Management), grupama, ovlastima i sl.

Za samu egzekuciju procesa dovoljan je samo **Workflow Engine**, dok su **Cockpit**, **Tasklist** i **Admin** dodatne komponente koje olakšavaju upravljanje i nadzor poslovnih procesa kroz grafička sučelja.

Kao 5. komponentu ne smijemo izostaviti naš **Camunda Modeler** koji koristimo kao odvojeni alat za **modeliranje**. Međutim, vidjet ćete da modeliranje sad dobiva na još jednoj dimenziji - onoj **podatkovnoj**.

Za učenje, ovdje već imate uključena 2 poslovna procesa:

- *Invoice Receipt*
- *Review Invoice*

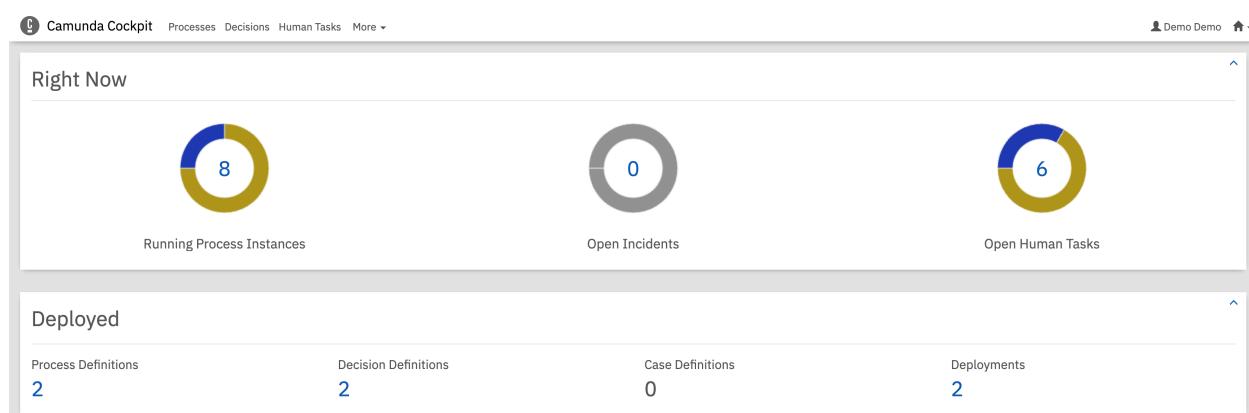
## 3.1 Camunda Cockpit

Na početnoj stranici, gdje vidite prikaz svih komponenti, odaberite **Cockpit**.

Tražit će vas da se prijavite. Korisničko ime i lozinka su `demo`.

**Camunda 7 Cockpit** predstavlja centralno mjesto za **nadzor, upravljanje i analizu poslovnih procesa** koji se izvršavaju. Kao i druge komponente (osim Workflow Enginea), Cockpit je dostupan preko web grafičkog sučelja.

Jednom kad se prijavite, vidjet ćete upravljačku ploču koja prikazuje trenutne **aktivne procese i procesne instance**.



Početna upravljačka ploča Camunda Cockpit komponente

Podsjetimo se: **Procesna instanca odnosi se na jedno pokretanje procesa**.

Svaki korisnik koji pokrene proces ima svoju instancu procesa.

Na ovom prikazu vidimo 2 aktivna procesa:

- *Invoice Receipt*
- *Review Invoice*

Od toga, postoji:

- 6 aktivnih instanci procesa *Invoice Receipt* i
- 2 aktivne instance procesa *Review Invoice*

Ako pritisnute **Processes** u gornjem izborniku, vidjet ćete popis pokrenutih **procesa**.

The screenshot shows the Camunda Cockpit interface with the 'Processes' tab selected. A header bar at the top includes the Camunda logo, 'Camunda Cockpit', and navigation links for 'Processes', 'Decisions', 'Human Tasks', and 'More'. Below the header, a message states '2 process definitions deployed'. A table lists the deployed processes: 'invoice' (Key) has 6 'Running Instances' and is named 'Invoice Receipt'; 'ReviewInvoice' (Key) has 2 'Running Instances' and is named 'Review Invoice'. The table includes columns for 'State', 'Incidents', 'Key', 'Name', and 'Tenant ID'. At the bottom right of the table, there are buttons for 'List' and 'Previews'.

### Pregled aktivnih procesa i procesnih instanci

Otvorite proces *Review Invoice* kako biste vidjeli aktivne instance tog procesa i trenutno stanje procesa kroz **dijagram toka**.

The screenshot shows the 'Review Invoice : Runtime' process details page. On the left, a sidebar displays process metadata: Definition Version (1), Version Tag (null), Definition ID (ReviewInvoice:1:37dddefa2-b6fc-11ef...), Definition Key (ReviewInvoice), Definition Name (Review Invoice), History Time To Live (45 days), Tenant ID (null), Deployment ID (37a923fc-b6fc-11ef-9b3c-0242ac11...), and Instances Running (current version: 2, all versions: 2). The main area shows the process diagram with two tasks: 'Assign Reviewer' and 'Review Invoice', connected by a flow. On the right, a table lists the 'Process Instances' with columns for 'State', 'ID', 'Start Time', and 'Business Key'. Two instances are listed: one starting at 2024-12-10T14:39:47 and another at 2024-12-10T14:39:47.

Uočite putanju u programu:

Dashboard → Processes → Review Invoice : Runtime

Sad **pregledavamo aktivne instance procesa** *Review Invoice*. Vidimo da postoje dvije.

U prvom planu vidimo dijagram ovog poslovnog procesa koji se sastoji od samo **2 aktivnosti (eng. Task)**:

- *Assign Reviewer* (User Task)
- *Review Invoice* (User Task)

Na dijagramu vidimo brojku (2) pored aktivnosti *Assign Reviewer* što znači da postoje **2 aktivne instance** ovog procesa **koje se trenutno nalazi na ovoj aktivnosti** (radnom koraku).

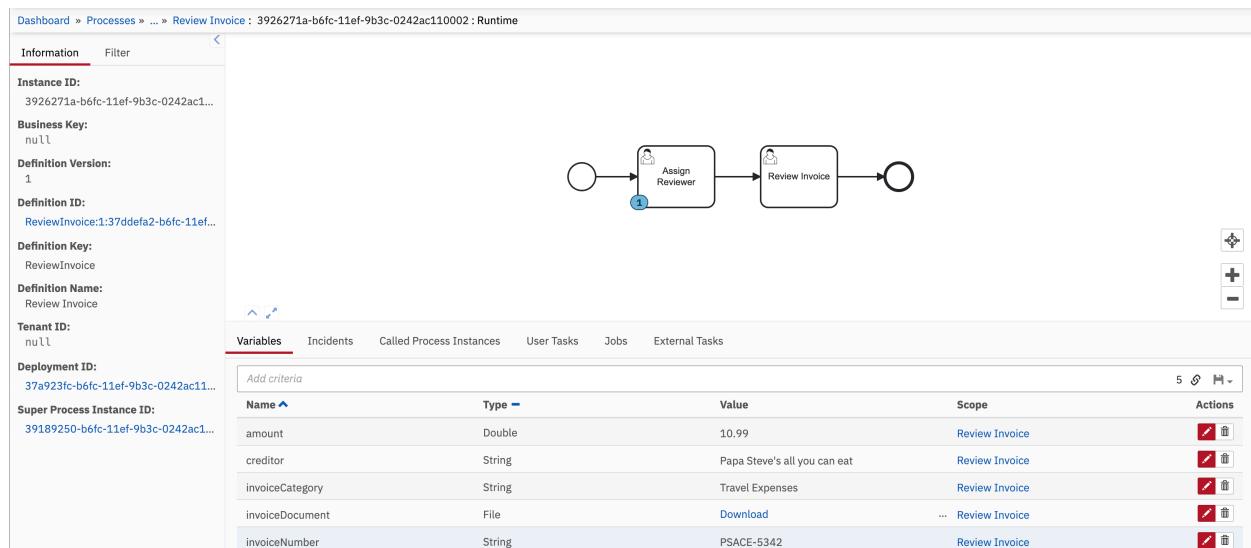
Spomenutu oznaku nazivamo **token**.

U izborniku lijevo možemo vidjeti neke općenite informacije o procesu, kao što su:

- **Definition Version** - verzija definicije procesa (u slučaju da se proces mijenja, što je čest slučaj u praksi)
- **Definition ID** - ID trenutne definicije poslovnog procesa
- **Definition Key** - ključ definicije procesa
- **History Time To Live** - koliko dugo se povijest procesa (pohranjene procesne varijable) čuva u internoj bazi podataka
- **Deployment ID** - ID trenutnog *deploymenta* poslovnog procesa
- **Instance Running** - koliko je trenutno aktivnih instanci procesa, za trenutnu verziju i sve verzije ukupno

Ako pritisnemo na jednu od dvije instance, otvorit će se još jedan prozor s detaljima o toj instanci. Ovdje su nam najzanimljivije pohranjene **procesne varijable**:

- *amount*
- *creditor*
- *invoiceCategory*
- *invoiceDocument*
- *invoiceNumber*



Detalji o jednoj instances procesa *Review Invoice* (**procesne varijable**)

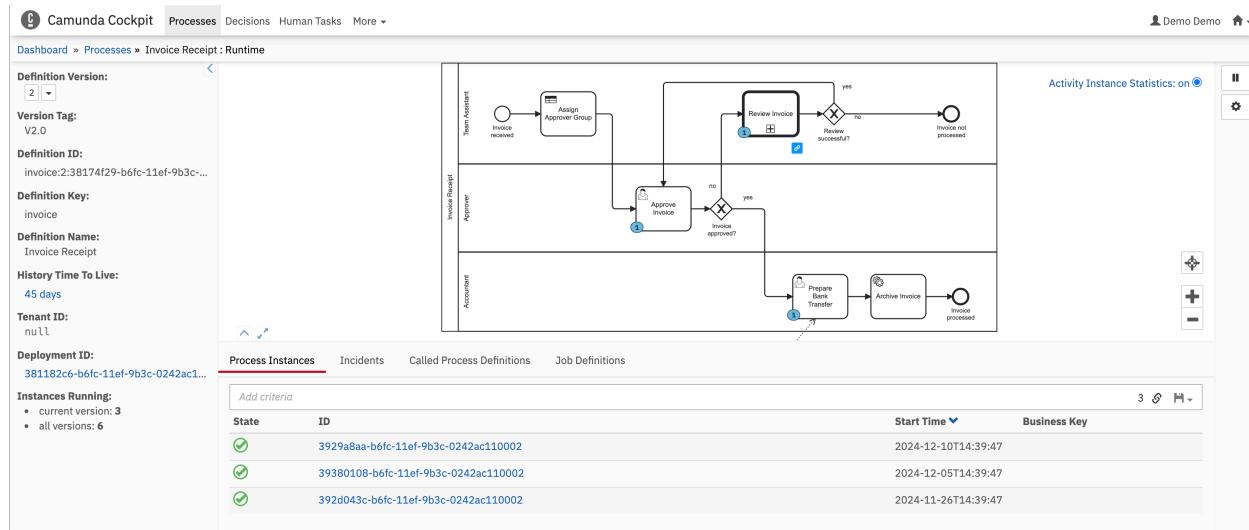
Vidimo da se svaka varijabla sastoji od:

- **naziva/ključa**
- **tipa podataka** (npr. `Integer`, `Boolean`, `Double`, `String`, `File`, itd.)
- **vrijednosti**
- **opseg-a procesa gdje je procesna varijabla vidljiva (scope)**

Vrijednosti se ovdje mogu direktno mijenjati, što je korisno **u slučaju da je potrebno ručno intervenirati u procesu**.

U prozoru **Add criteria** moguće je definirati kriterije za filtriranje podataka po **instanci, ključu i vrijednosti varijable**.

Ako se vratimo na **Dashboard → Processes** i otvorimo drugi proces *Invoice Receipt*, možemo vidjeti i ovaj proces i njegove aktivne instance.



### Pregled aktivnih instanci procesa *Invoice Receipt*

Vidimo da je proces složeniji od prethodnog, sastoji se od 3 staze (swimlanes) koje predstavljaju različite sudionike u procesu:

- **Team Assistant**
- **Approver**
- **Accountant**

Ovaj proces ima ukupno 6 aktivnih instanci, od kojih su:

- 3 u definiciji procesa V1.0 i
- 3 u definiciji procesa V2.0

Dalje, možemo uočiti nekoliko aktivnosti na svakoj stazi:

- *Assign Approver Group*
- *Approve Invoice*
- *Review Invoice*
- *Prepare Bank Transfer*
- *Archive Invoice*

Uočavate li nešto? *Review Invoice* je ustvari **potproces** koji se koristi u ovom procesu, međutim on je definiran i *deployan* kao zasebni proces koji se izvršava u Workflow engineu, a i vidjeli smo ga malo prije.

Dakle, kroz Camunda Cockpit, osim glavnog procesa koji se izvršava, **možemo na jednak način pratiti i potprocese koji se izvršavaju unutar glavnog procesa**.

Ako pogledate ovdje graf, možemo vidjeti ukupno 3 tokena: brojke `(1)` na aktivnostima:

- *Approve Invoice*
- *Prepare Bank Transfer*
- *Review Invoice*

Što to znači? 🤔

---

- **Ukupno 1 instance** procesa *Invoice Receipt* je trenutno na aktivnosti ***Approve Invoice***.
  - Npr. "Za invoice broj 12345, Approver mora odobriti račun".
- **Ukupno 1 instance** procesa *Invoice Receipt* je trenutno na aktivnosti ***Prepare Bank Transfer***.
  - Npr. "Za invoice broj 54321, Accountant mora pripremiti bankovni transfer".
- **Ukupno 1 instance** procesa *Invoice Receipt* je trenutno na aktivnosti (potprocesu) ***Review Invoice***.
  - Npr. "Za invoice broj 67890, Team Assistant mora pregledati račun".

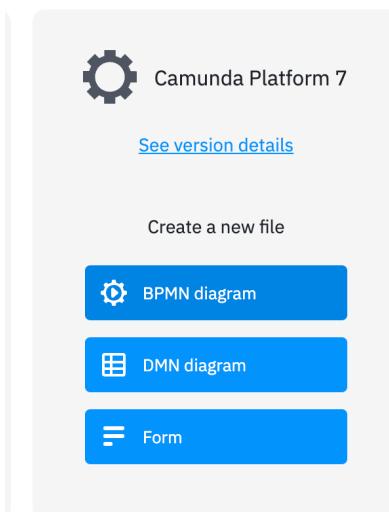
Ako stisnemo na neku od ovih aktivnosti s tokenom, **filtrirat će nam se one instance koje se trenutno nalaze na toj aktivnosti.**

Dodatno, pored potprocesa možemo odabrati opciju `"Show Called Process Definition"` koja će nam otvoriti novi prozor **s detaljima o tom potprocesu i njegovim aktivniminstancama**, dakle klikom na "Review Invoice", otvorit će se `Dashboard -> Processes -> Invoice Receipt -> Review Invoice: Runtime`.

### 3.1.1 Egzekucija vlastitog procesa

Prije nego što krenemo pregledavati druge komponente (`Tasklist`, `Admin`), idemo pokušati definirati vlastiti poslovni proces, pokrenuti ga te pratiti njegovo izvođenje unutar Camunda Cockpita.

Otvorite Camunda Modeler i odaberite novi BPMN dijagram za Camunda 7 platformu.



| Odaberite Camunda Platform 7 -> BPMN diagram

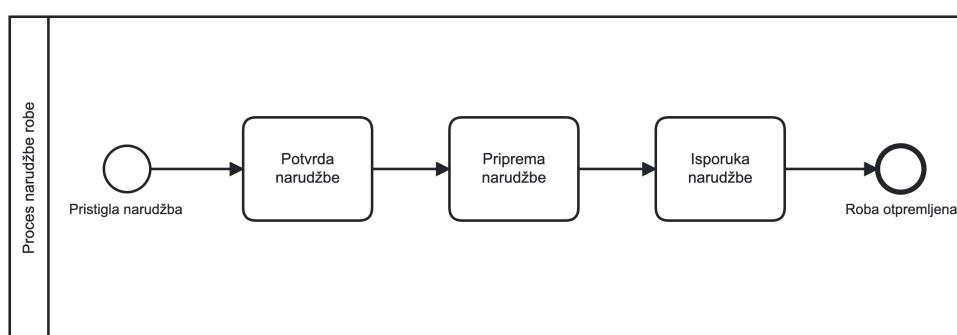
Dijagram pohranite lokalno, na proizvoljnu lokaciju.

Definirat ćemo jednostavan proces koji definira **obradu narudžbe proizvoda u webshopu**. Nazovite ga: `webshop-order.bpmn`

**Krenimo jednostavno**, definirat ćemo proces koji se sastoji od 3 aktivnosti u jednoj stazi:

- *Potvrda narudžbe*
- *Priprema narudžbe*
- *Isporuka narudžbe*

Za sada nećemo definirati dopunske atribute aktivnosti, niti skretnice. Napravite jednostavan linearni proces s 3 aktivnosti i početnim i završnim događajem.



| Jednostavan proces narudžbe u webshopu

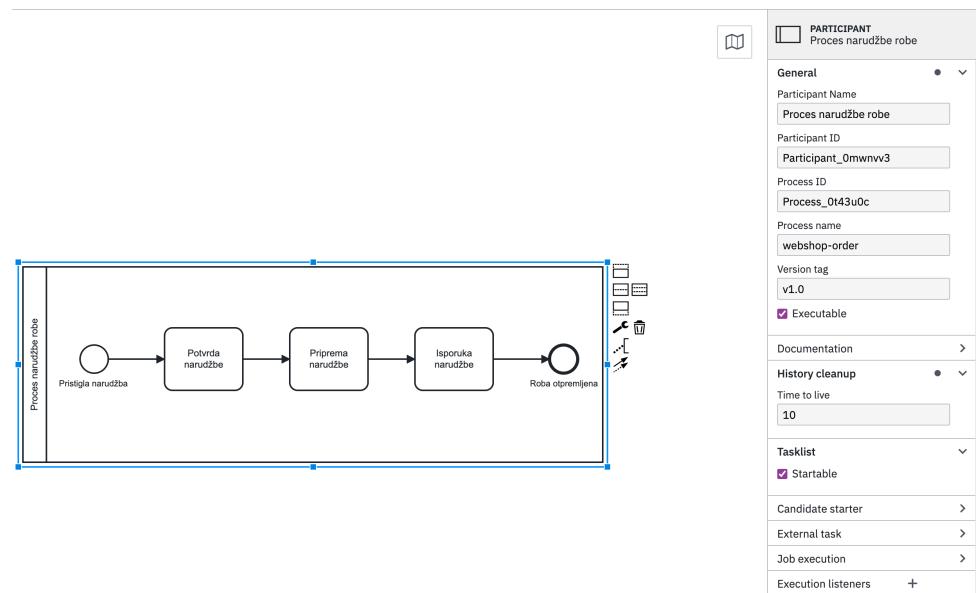
Prije nego možemo *deployati* ovaj proces, moramo definirati nekoliko stvari:

- **Process name** - ime procesa (npr. *Webshop Order*)

- **Version tag** - verzija procesa (npr. v1.0)
- **Time to live** - koliko dugo se povijest procesa čuva u bazi podataka (unesite proizvoljnu vrijednost)
- **Process ID** - jedinstveni identifikator procesa (npr. *narudzba\_robe*) (čisto da vidite da ne mora biti isto kao ime procesa, ovaj podatak ćemo kasnije najviše koristiti)

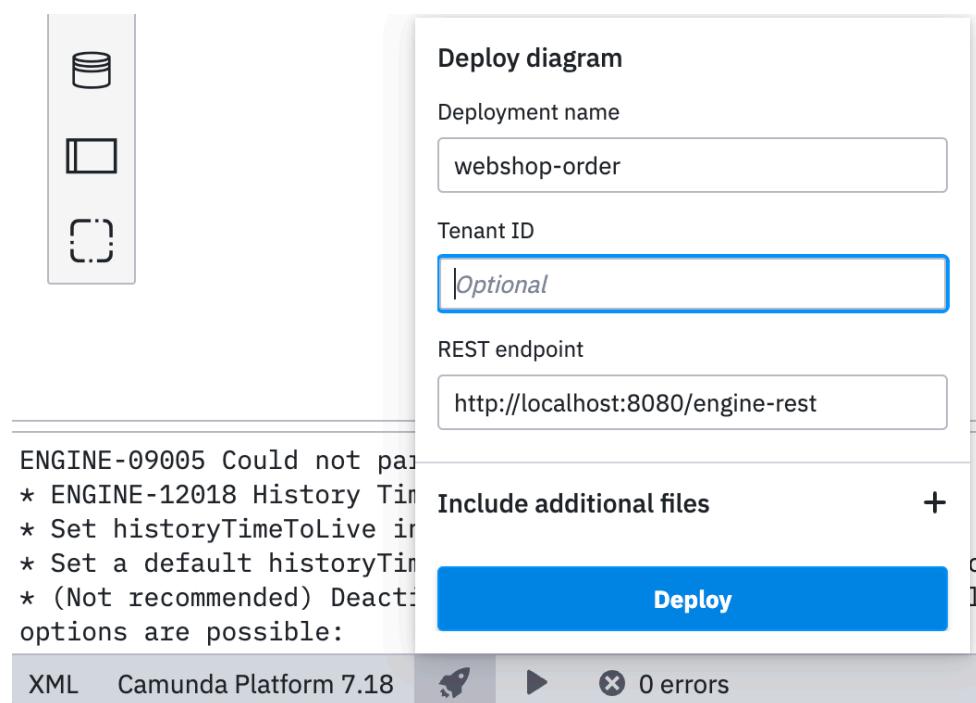
Stisnite na *pool* gdje je sadržan proces, trebao bi vam se otvoriti s desne strane prozor s postavkama procesa (**Properties panel**).

Ako vam se ovaj prozor ne prikazuje, odaberite `Window -> Toggle Properties Panel`. Jednom kad se otvori, **unesite tražene vrijednosti**:



#### Postavke procesa u **Properties panelu**

Sada možete deployati diagram pritiskom na ikonu rakete ( u donjem lijevom kutu



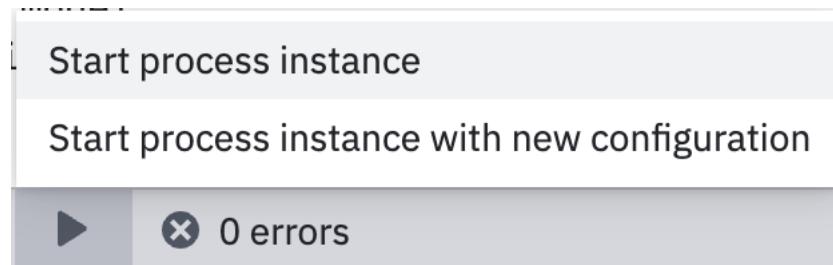
Provjerite da se PORT REST endpointa poklapa s portom na kojem je pokrenuta Camunda platforma, odnosno PORT na koji je mapiran Docker kontejner.

Trebali biste dobiti poruku o uspješnom deploymentu definicije procesa. Sada otvorite Camunda Cockpit i pregledajte procese:

Ako otvorite proces, vidjet ćete da **nema aktivnih instanci**. To je zato što nismo pokrenuli niti jednu.

U realnom okruženju, proces će se pokrenuti nekim događajem ili korisničkom interakcijom. Međutim, tijekom razvoja procesne aplikacije, možemo ručno pokrenuti proces:

**Vratite se u Modeler**, na dnu odaberite strelicu pored ikone rakete ( ) i odaberite `Start process instance`.



Pokretanje nove instance procesa direktno unutar **Camunda Modelera**

Trebali biste dobiti obavijest o uspješnom pokretanju instance procesa.

Ako se vratite u Camunda Cockpit i osvježite stranicu očekivali biste novu instancu procesa koja čeka na zadatku "**Potvrda narudžbe**", međutim to nije slučaj. Zašto?

### 3.1.2 User Task i forme

Nećete vidjeti novu instancu procesa, niti će se ona pojaviti u Cockpitu jer je instanca već završila (odnosno započela, odradila zadatke i završila). Naime, proces koji smo definirali je jednostavan linearni proces koji se sastoji od 3 aktivnosti i završnog događaja. Ove 3 aktivnosti koje smo definirali nemaju nikakve dodatne postavke/atribute, odnosno **nismo definirali način na koji korisnik može dati "input" u proces**, npr. **obraditi narudžbu, pripremiti narudžbu** i sl.

Ako se prisjetite, rekli smo da za radnje koje obavlja korisnik preko informacijskog sustava (u našem slučaju je to Camunda Cockpit) koristimo **korisnički radni korak/zadatak** (`User task`) kao **opisni atribut aktivnosti**.

Izmijenit ćemo proces tako da aktivnost "**Potvrda narudžbe**" bude `User task`.

Međutim to nije sve, moramo definirati i neki način kako će korisnik potvrditi tu narudžbu, kojom interakcijom? To ćemo definirati preko **formi**. Camunda 7 dozvoljava da definirate 3 vrste formi:

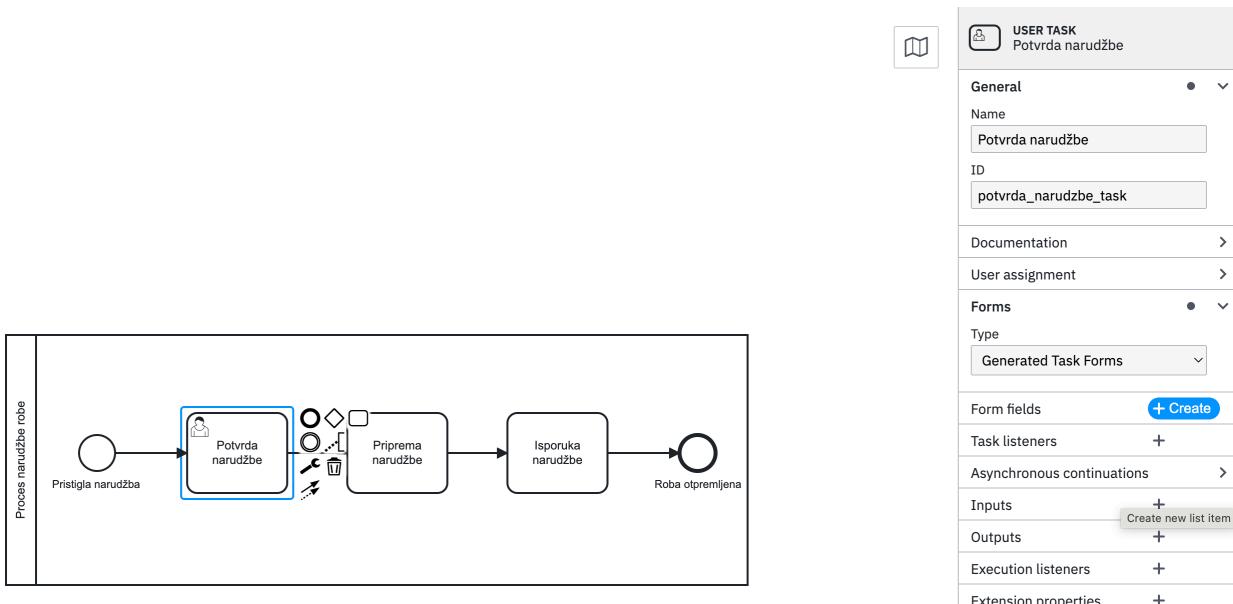
1. `Camunda Forms`
2. `Embedded or External Task Forms`

### 3. Generated Task Forms

Općenito, forme definiraju **način na koji korisnik može unijeti podatke u proces**.

1. Camunda Forms su bazirane na JSON zapisu i direktno su integrirane u Camunda Modeler kroz **Form Editor** (File -> New file -> Form (Camunda platform 7)). Potrebno je dodati referencu na formu, a one će se automatski prikazati u Cockpitu.
2. Embedded or External Task Forms su forme koje se mogu definirati izvan Camunda Modelera, npr. preko HTML/CSS/JS. S njima je moguće komunicirati preko REST API poziva.
3. Generated Task Forms su forme koje se generiraju automatski na temelju varijabli koje su definirane u opcijama **Form fields** u Properties panelu za **User Task**.

Mi ćemo za sad odabrati **treću opciju** obzirom da je ujedno i najjednostavnija.



Unos ID-a i odabir tipa forme: "**Generated Task Forms**"

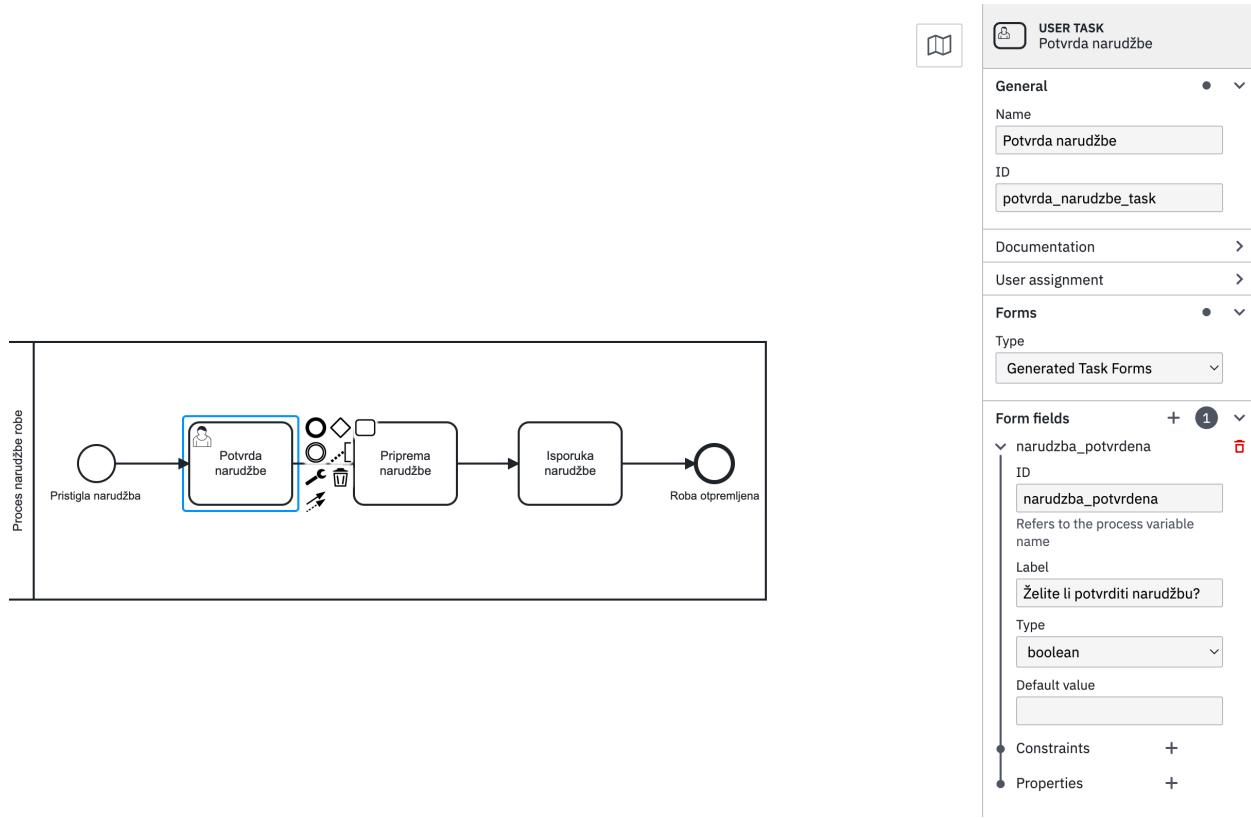
Svakoj aktivnosti, pa i ovoj, možete dodati ID, npr. `potvrda_narudzbe_task`.

1. Pod **Forms** odaberite **Generated Task Forms**
2. Odaberite **Form Fields** i dodajte jedno polje. Nazovite ga `narudzba_potvrdena` i postavite tip podatka na `Boolean`. Pod **Label** možete unijeti labelu koju će korisnik vidjeti za navedeno polje, npr. "Želite li potvrditi narudžbu?".

Uočite dodatne opcije (`Constraints`, `Properties`), za sada ćemo ih ignorirati.

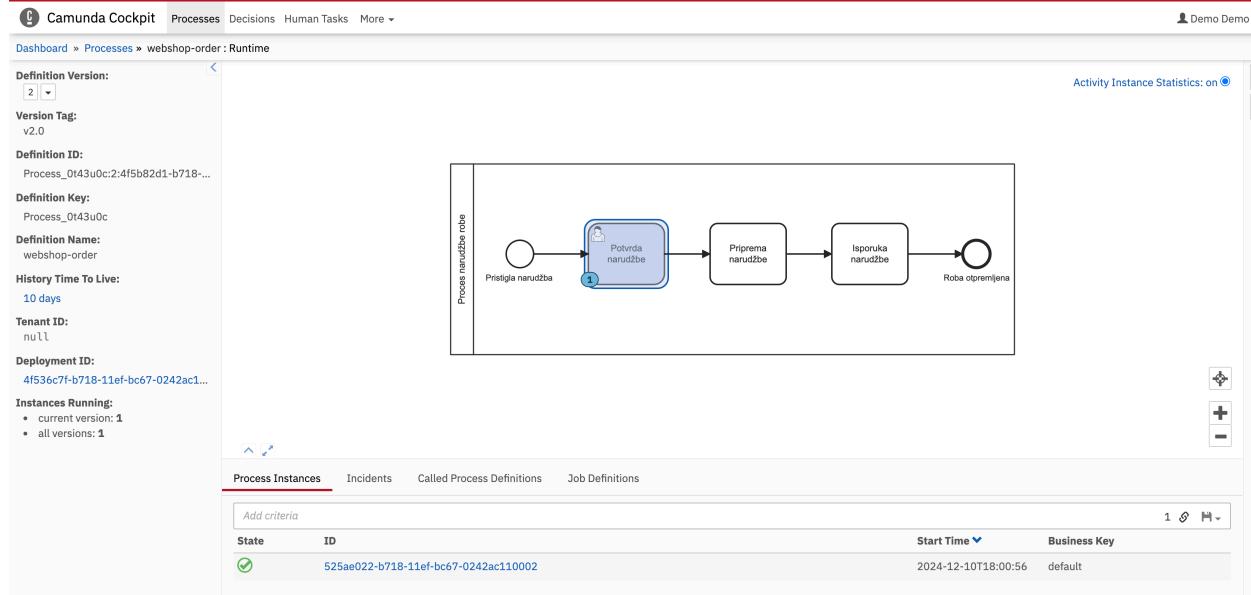
Dignite verziju procesa na `v2.0` i *deployajte* ga.

Nakon toga, startajte novu instancu procesa kroz Modeler.



### Dodavanje novog polja u formu

Otvorite instance ovog procesa, vidjet ćete **1 aktivnu instancu** koja ima token na aktivnosti "Potvrda narudžbe".



Pregled aktivne instance procesa s tokenom na aktivnosti "Potvrda narudžbe"

Kako bi završili ovu aktivnost, moramo potvrditi narudžbu putem nove forme koju smo definirali 😎

## 3.2 Camunda Tasklist

**Camunda Tasklist** je web aplikacija unutar Camunde 7 koja omogućuje korisnicima da pregledavaju i obavljaju korisničke zadatke (**User Task**) koji su im dodijeljeni u poslovnim procesima.

Otvorite **Tasklist** preko početne stranice Camunda platforme.

Ako vas traži korisničke podatke, možete ponovo unijeti **demo** kao korisničko ime i lozinku.

Lijevo ćete vidjeti grupirane zadatke po kategorijama. Zadatke je moguće grupirati po korisnicima, po procesima, po prioritetu, po datumu, itd. Možete vidjeti nekoliko predefiniranih grupa, ali i zadatke različitih korisnika koji su uneseni po defaultu u Camunda platformu, radi lakšeg učenja (John, Marry, Peter).

Odaberite **All Tasks** kako biste vidjeli sve zadatke.

Obzirom da ste prijavljeni kao **demo**, koji ima **administratorske ovlasti**, vidjet ćete sve zadatke koji su trenutno aktivni u Camunda platformi.

Dakle, možete se samostalno dodijeliti na zadatak **Potvrda narudžbe**; pritisnite na **claim** skroz desno.

The screenshot shows the Camunda Tasklist interface. On the left, there's a sidebar with categories like 'My Tasks', 'My Group Tasks', 'Accounting', 'John's Tasks', 'Marry's Tasks', 'Peter's Tasks', and 'All Tasks (7)'. The main area displays a list of tasks. One task, 'Potvrda narudžbe', is highlighted and shown in more detail. This task is associated with a process 'webshop-order (v. v2.0)' and has a due date set for 'Set follow-up date'. The task details include 'Invoice Amount: 10.99' and 'Invoice Number: PSACE-5342'. Below this, there are other tasks: 'Prepare Bank Transfer', 'Approve Invoice', and another 'Assign Reviewer' task. At the bottom right of the main view, there are buttons for 'Save' and 'Complete'.

Pregled svih zadataka (All Tasks) u **Tasklist** aplikaciji

U sadržaju forme možete vidjeti polje **narudzba\_potvrdena**, kojem ste dodijelili labelu "želite li potvrditi narudžbu?" i tip podatka **Boolean**.

Možete odabrati **checkbox** i pritisnuti na **complete** kako biste završili ovaj korisnički zadatak, ali ga i ne morate odabrati jer je polje **narudzba\_potvrdena** definirano kao **Boolean** tip podatka, što znači da je moguće unijeti i **true** i **false** vrijednosti.

Što će se dogoditi nakon što završite ovaj zadatak? 😊

Što ako ga završimo s **true** vrijednošću? A što ako ga završimo s **false** vrijednošću?

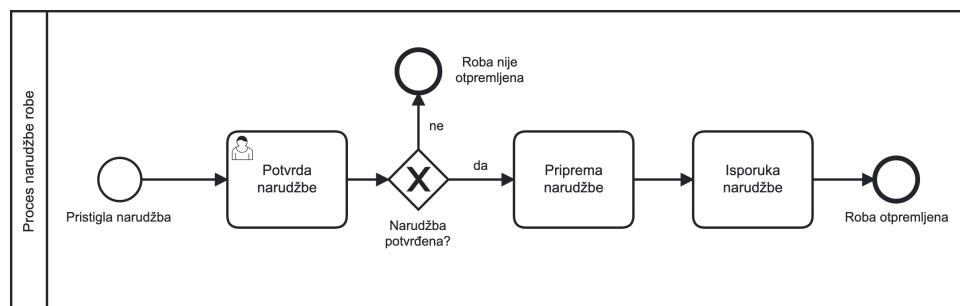
Instanca procesa završava u oba slučaja.

### 3.2.1 Procesne varijable i dodavanje **xor** skretnice

Vidjeli smo u modeliranju, da je uobičajeno da se nakon `User Taska` dodaje `xor` skretnica koja **određuje sljedeći korak** u procesu ovisno o rezultatu korisničkog zadatka, iako to ne mora uvijek biti slučaj.

Nadogradit ćemo proces dodavanjem `xor` skretnice koja će ovisno o rezultatu zadatka `"Potvrda narudžbe"` podijeliti proces na dva toka:

- ako je narudžba potvrđena, proces ide dalje na korak `"Priprema narudžbe"`
- ako narudžba nije potvrđena, proces završava



Dodavanje `xor` skretnice u definiciju procesa s alternativnim slijedom

Što je ovdje podatak/varijabla? 🤔

U ovom slučaju, podatak je `narudzba_potvrdena` koji je definiran u formi korisničkog zadatka `"Potvrda narudžbe"`. Ovaj podatak je **procesna varijabla** koja se pohranjuje u **procesnu instancu** i može se kasnije koristiti bilo gdje u procesu. Naziv procesne varijable smo definirali prilikom definiranja forme (`ID - Refers to the process variable name`).

Drugim riječima, procesna varijabla je **automatski pohranjena u procesnoj instanci** jednom kad se aktivnost `"Potvrda narudžbe"` završi.

Ono što moramo napraviti je definirati koristeći [Expression Language \(EL\)](#) izraze na izlaznim tokovima **XOR skretnice**.

Na izlaznim tokovima smo već napisali labele:

- da
- ne

**Međutim to nije dovoljno kod razvijamo procesne aplikacije!**

Osnovna sintaksa za **provjeru vrijednosti procesne varijable** je:

```
 ${varijabla == "vrijednost"}
```

U našem slučaju `"vrijednost"` je `true` ili `false`.

Dakle, na izlaznom toku koji ide prema `"Priprema narudžbe"` napišite izraz:

```
 ${narudzba_potvrdena == true}
```

Na izlaznom toku koji ide prema završnom događaju napišite izraz:

```
 ${narudzba_potvrdena == false}
```

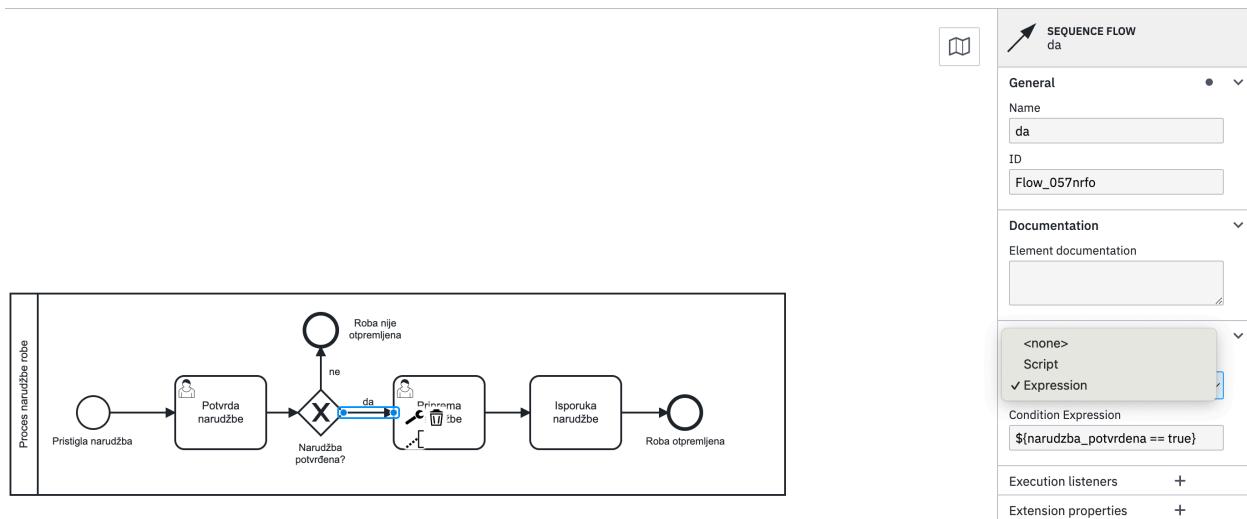
Alternativno, moguće je `Boolean` izraze napisati i skraćeno:

```
 ${narudzba_potvrdena}
```

odnosno ako nije potvrđena:

```
 ${!narudzba_potvrdena}
```

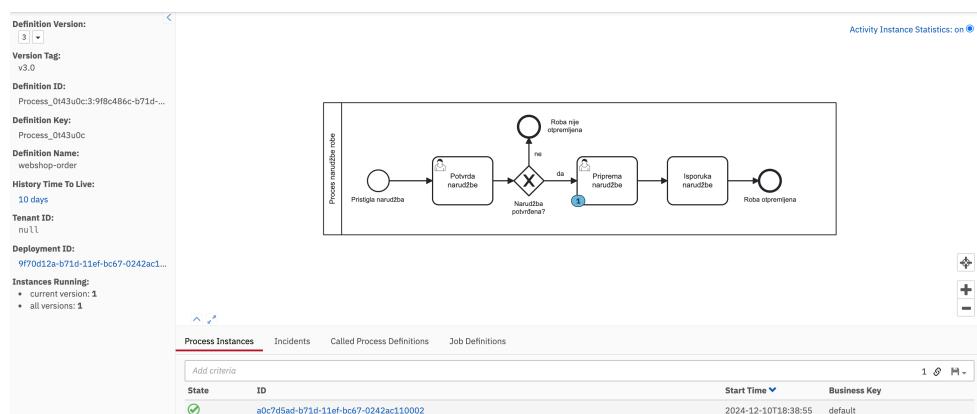
Odaberite strelice i definirajte `Condition Expression` za svaki izlazni tok (2):



Dodavanje izraza na izlazne tokove XOR skretnice (na izlazni tok "ne" dodan je izraz  
 `${!narudzba_potvrdena}` )

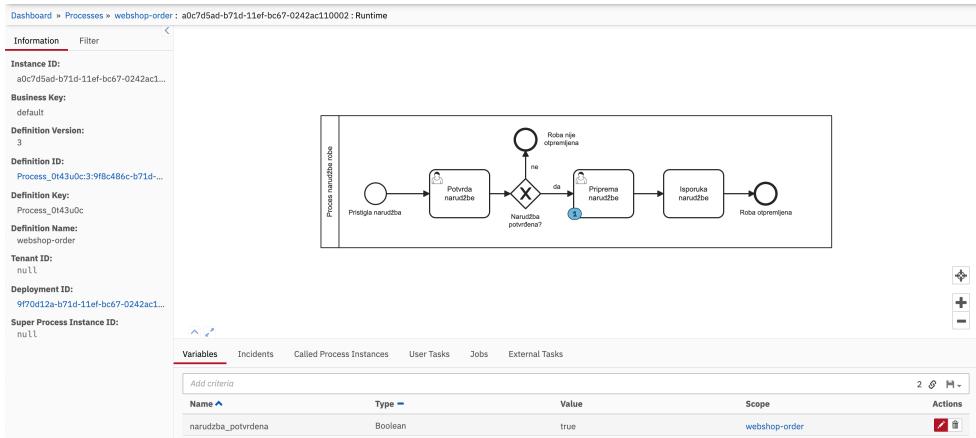
Ako sad ispunite formu i odaberete `true`, proces će ići prema aktivnosti `"Priprema narudžbe"`. Ako odaberete `false`, proces će završiti.

Provjerite rezultat u Camunda Cockpitu:



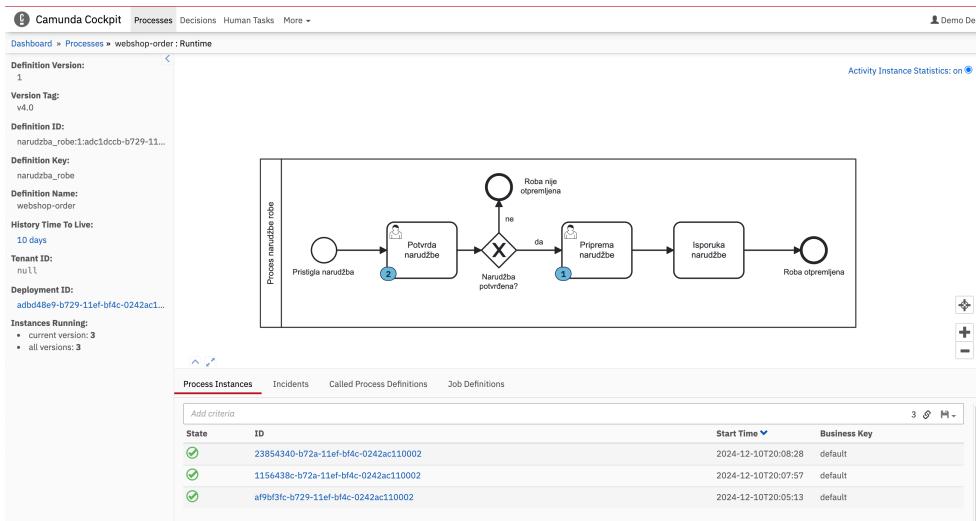
Token (1) se nalazi na aktivnosti `"Priprema narudžbe"`, što znači da **jedna aktivna instanca procesa čeka na ovom koraku**

Ako otvorite pregled procesne instance u Cockpitu, vidjet ćete da je procesna varijabla `narudzba_potvrdena` pohranjena u procesnu instancu i ima vrijednost `true`.



Pregled procesne instance s pohranjenom procesnom varijablu `narudzba_potvrdena`

Pokrenite još dvije instance ovog procesa kroz Modeler:



Prikaz 3 aktivne instance procesa, od kojih 2 čekaju na aktivnosti "Potvrda narudžbe", a jedna na "Priprema narudžbe"

### 3.2.2 Kako još možemo izraditi instance?

Osim ručnog pokretanja procesa preko Modelera, moguće je pokrenuti proces preko **REST API-ja**, ali i direktno iz **Camunda Tasklista**.

Do sad smo direktno izradivali novu instancu procesa preko Modelera, iako je ovo praktično za testiranje, u stvarnom okruženju korisnici naravno neće imati pristup modeleru.

Problemi su sljedeći:

- Pristigla je narudžba - gdje su podaci o narudžbi? Kako ih unijeti?
- Kako krajnji korisnik može pokrenuti instancu procesa?

Procesne varijable možemo, osim kroz različite aktivnosti, **definirati i na početku**, kod započinjanja procesa. Proces koji modeliramo započinje primitkom narudžbe, logično je da onda i podaci o narudžbi budu procesne varijable.

Konkretno, podaci o narudžbi razlikovat će se u svakoj procesnoj instanci, samim tim je logično da ih ne definiramo unutar procesa (npr. u start eventu), već se unose **prilikom pokretanja procesa**.

## 1. Način: Izrada procesne instance s varijablama preko Camunda Tasklista

- otvorite **Tasklist sučelje** i odaberite `start process` u gornjem desnom kutu

Camunda Tasklist      Keyboard Shortcuts      Create task      Start process      Demo      Demo      ▾

Odaberite `webshop-order` proces:

- unesite `Business Key` (proizvoljno): predstavlja jedinstveni ključ procesne instance (npr. u stvarnosti može biti ID narudžbe)
- dodajte varijable pritiskom na "Add a variable" i unesite neke podatke o narudžbi u obliku ključ:vrijednost parova

Start process: webshop-order

You can set variables, using a generic form, by clicking the "Add a variable" link below.

Add a varia... +	Name	Type	Value
Remo... x	proizvod	String	Crna jakna
Remo... x	cijena	Double	150.25
Remo... x	kolicina	Integer	2

Back      Close      Start

### Pokretanje procesa s varijablama preko Tasklista

Dobit ćete poruku `"Process has been started."`

Provjerite procesne varijable pohranjene u procesnu instancu u **Cockpitu**.

## 2. Način: Izrada procesne instance s varijablama preko REST API-ja

Jednom kad se Camunda platforma pokrene, automatski se pokreće i REST API koji omogućuje komunikaciju s platformom preko HTTP protokola. REST API je dokumentiran i možete pronaći sve informacije na [sljedećoj poveznici](#).

Ako otvorite Modeler, vidjet ćete da je REST endpoint sljedeći:

`http://localhost:8080/engine-rest`

Otvorite **Postman** ili **Thunder Client**, možete poslati GET zahtjev na `http://localhost:8080/engine-rest/process-definition` kako biste dobili sve definicije procesa:

```

39      "name": "Invoice Receipt",
40      "version": 2,
41      "resource": "invoice.v2.bpmn",
42      "deploymentId": "9da07cc9-b728-11ef-bf4c-0242ac110002",
43      "diagram": null,
44      "suspended": false,
45      "tenantId": null,
46      "versionTag": "V2.0",
47      "historyTimeToLive": 45,
48      "startableInTasklist": true
49    },
50    {
51      "id": "narudzba_robe:1:adc1dccb-b729-11ef-bf4c-0242ac110002",
52      "key": "narudzba_robe",
53      "category": "http://bpmn.io/schema/bpmn",
54      "description": null,
55      "name": "webshop-order",
56      "version": 1,
57      "resource": "webshop-order.bpmn",
58      "deploymentId": "adbd48e9-b729-11ef-bf4c-0242ac110002",
59      "diagram": null,
60      "suspended": false,
61      "tenantId": null,
62      "versionTag": "v4.0",
63      "historyTimeToLive": 10,
64      "startableInTasklist": true
65    }
66  ]

```

Uočite proces `webshop-order` kao drugu vrijednost u JSON listi

Za pokretanje procesa, koristimo **POST metodu i endpoint** `http://localhost:8080/engine-rest/process-definition/key/<ProcessID>/start`, gdje je `<ProcessID>` ključ procesa, npr. `webshop-order` ili `narudzba_robe` - ovisno kako ste ga definirali u Modeleru (pogledati poglavlje 3.1.1).

**PARTICIPANT**  
Proces narudžbe robe

**General**

Participant Name: Proces narudžbe robe

Participant ID: Participant\_Omwnvv3

Process ID: narudzba\_robe

Process name: webshop-order

Version tag: v4.0

Executable

Ključ procesa je `narudzba_robe`

**Endpoint za pokretanje procesa je:**

`http://localhost:8080/engine-rest/process-definition/key/narudzba_robe/start`

Međutim, kako bi pokrenuli proces s varijablama, moramo dodati i varijable u **tijelu HTTP zahtjeva**.

Varijable je potrebno omotati u JSON objekt, unutar ključa `variables`.

Npr. sljedeći JSON objekt započinje instancu procesa `narudzba_robe` s varijablama `proizvod`, `cijena` i `kolicina`:

```
{  
    "variables": {  
        "proizvod": {  
            // ključ varijable  
            "value": "Majica", // vrijednost varijable  
            "type": "String" // tip varijable  
        },  
        "cijena": {  
            "value": 70,  
            "type": "Double"  
        },  
        "kolicina": {  
            "value": 2,  
            "type": "Integer"  
        }  
    }  
}
```

The screenshot shows a REST client interface with the following details:

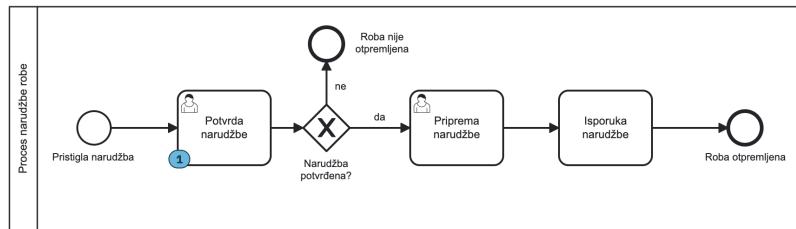
- URL:** http://localhost:8080/engine-rest/process-definition/key/narudzba\_robe/start
- Method:** POST
- Body:** JSON (selected)
- JSON Content:**

```
1 {  
2     "variables": {  
3         "proizvod": {  
4             // ključ varijable  
5             "value": "Majica", // vrijednost varijable  
6             "type": "String" // tip varijable  
7         },  
8         "cijena": {  
9             "value": 70,  
10            "type": "Double"  
11        },  
12         "kolicina": {  
13             "value": 2,  
14             "type": "Integer"  
15         }  
16     }  
17 }
```
- Response Headers:** 200 OK, 382 ms, 505 B
- Response Body:**

```
1 {  
2     "links": [  
3         {  
4             "method": "GET",  
5             "href": "http://localhost:8080/engine-rest/process-instance/7b0bd8d1-b72d-11ef-bf4c-0242ac110002",  
6             "rel": "self"  
7         },  
8     ],  
9     "id": "7b0bd8d1-b72d-11ef-bf4c-0242ac110002",  
10    "definitionId": "narudzba_robe:1:adc1dccb-b729-11ef-bf4c-0242ac110002",  
11    "businessKey": null,  
12    "caseInstanceId": null,  
13    "ended": false,  
14    "suspended": false,  
15    "tenantId": null  
16 }
```

Izrada procesne instance s varijablama preko REST API-ja

Provjerite procesnu instancu i pohranjene varijable u **Cockpitu**.



Variables	Incidents	Called Process Instances	User Tasks	Jobs	External Tasks
Add criteria					
Name	Type	Value	Scope	Actions	
cijena	Double	70	webshop-order		
kolicina	Integer	2	webshop-order		
proizvod	String	Majica	webshop-order		

Pregled procesne instance definirane preko REST API-ja i njezinih pohranjenih varijabli

# 4. Obrada vrijednosti procesnih varijabli u procesu

Sad kad smo dodali mogućnost unosa podataka u proces preko **Tasklista i REST API-ja**, možemo **iskoristiti te podatke u procesu**.

Procesne varijable možemo dohvaćati na jednak način, bez obzira na način unosa, a to je sintaksom:

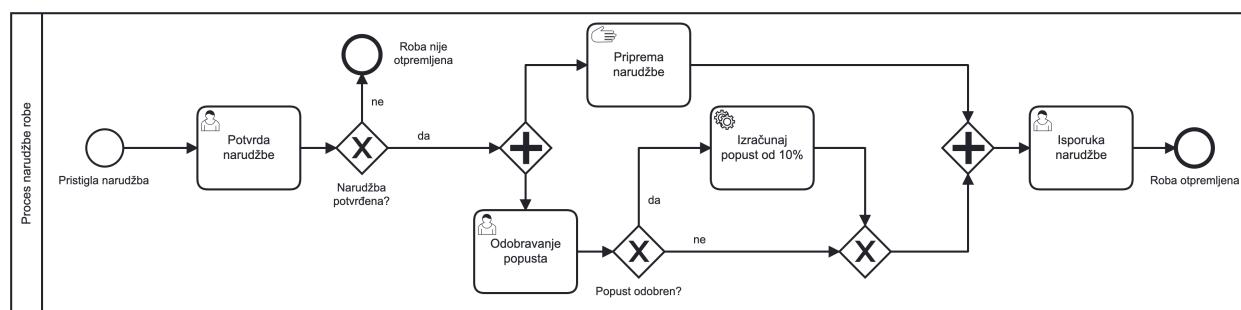
```
 ${naziv_procesne_varijable} .
```

Nadogradit ćemo proces `narudzba_robe` tako da se na temelju unesenih podataka o narudžbi, izračuna ukupna cijena narudžbe. Slijed procesa se sad paralelno dijeli na `"Priprema narudžbe"` i novi User Task - `"Odobravanje popusta od 10%"`. Korisnik može odobriti ili odbiti popust na narudžbu, a prilikom odobravanja/odbijanja mora unijeti i svoje ime i prezime. Ako korisnik odobri popust, isti se mora izračunati i pohraniti u procesnu varijablu. Ako korisnik odbije popust, proces nastavlja na jednak način ali bez izračuna popusta.

Koje atribute ćemo koristiti za ovaj nadograđeni proces?

- `"Potvrda narudžbe"` - **User Task**
- `"Odobravanje popusta"` - **User Task**
- `"Izračunaj popust od 10%"` - **Service Task**
- `"Priprema narudžbe"` - **Manual Task**
- `"Isporuka narudžbe"` - **User Task** - čisto da nam instance ne završi odmah, inače bi bio *manual task* ili *potproces*

Nakon `"Odobravanje popusta"` želimo izračunati ukupnu cijenu narudžbe i pohraniti ju u novu procesnu varijablu `ukupna_cijena`. Ova aktivnost ide u `XOR split` skretnicu `"Popust odobren?"` koja ovisno o rezultatu ide na `"Izračunaj popust od 10%"` ili direktno u `AND merge` skretnicu.



Prvo ćemo definirati formu za `Odobravanje popusta`. Odaberite `Generated Task Forms` i dodajte polja:

- `popust_odobren` - tip podatka `Boolean`, labela: `"Želite li odobriti popust od 10%?"`
- `djelatnik_ime` - tip podatka `String`, labela: `"Molimo unesite vaše ime"`
- `djelatnik_prezime` - tip podatka `String`, labela: `"Molimo unesite vaše prezime"`

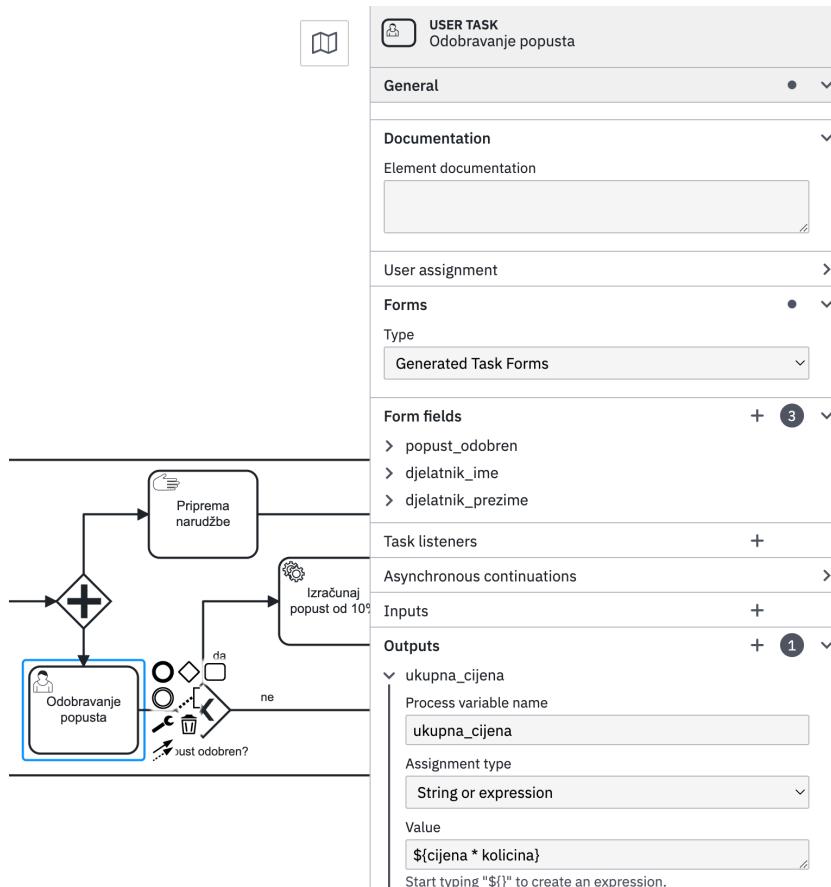
**Gotovo svaka aktivnost može kao rezultat svog izvršavanja pohraniti neku procesnu varijablu.**

Moguće je iskoristiti vrijednosti postojećih varijabli te unutar `Expression` izraza izračunati nove vrijednosti.

**Mi želimo izračunati ukupnu cijenu narudžbe** koristeći procesne varijable `cijena` i `kolicina` te pohraniti rezultat u novu varijablu `ukupna_cijena`.

Odaberite `Outputs` i dodajte novu varijablu `ukupna_cijena`. Odaberite `String or expression` te kao vrijednost pohranite:

```
 ${cijena * kolicina}
```



Dodavanje 3 polja u Form fields korisničkog zadatka "Odobravanje popusta" te dodavanje izračunate procesne varijable `ukupna_cijena` kao izlaznu vrijednost (**Outputs**) ovog zadatka

Na izlaznom toku `Da` dodajte izraz:

```
 ${popust_odobren}
```

Na izlaznom toku `Ne` dodajte izraz:

```
 ${!popust_odobren}
```

Izlazni tok `Da` ide na aktivnost "Izračunaj popust od 10%", a izlazni tok `Ne` ide direktno u AND merge skretnicu.

Na "Izračunaj popust od 10%" aktivnosti ćemo odabrati **servisni zadatak**. Servisni zadatak je aktivnost koja se izvršava automatski, bez korisničke interakcije, i može izvršiti neki posao, npr. izračunati popust. Odaberite jednostavni izraz:

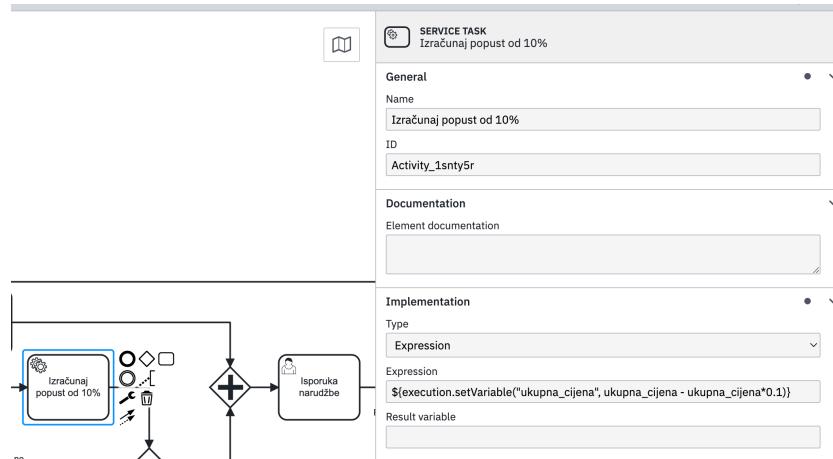
Implementation` -> `Type` -> `Expression

Sad možemo definirati izraz koji će promijeniti vrijednost procesne varijable `ukupna_cijena`:

```
 ${execution.setVariable("varijabla", vrijednost)}
```

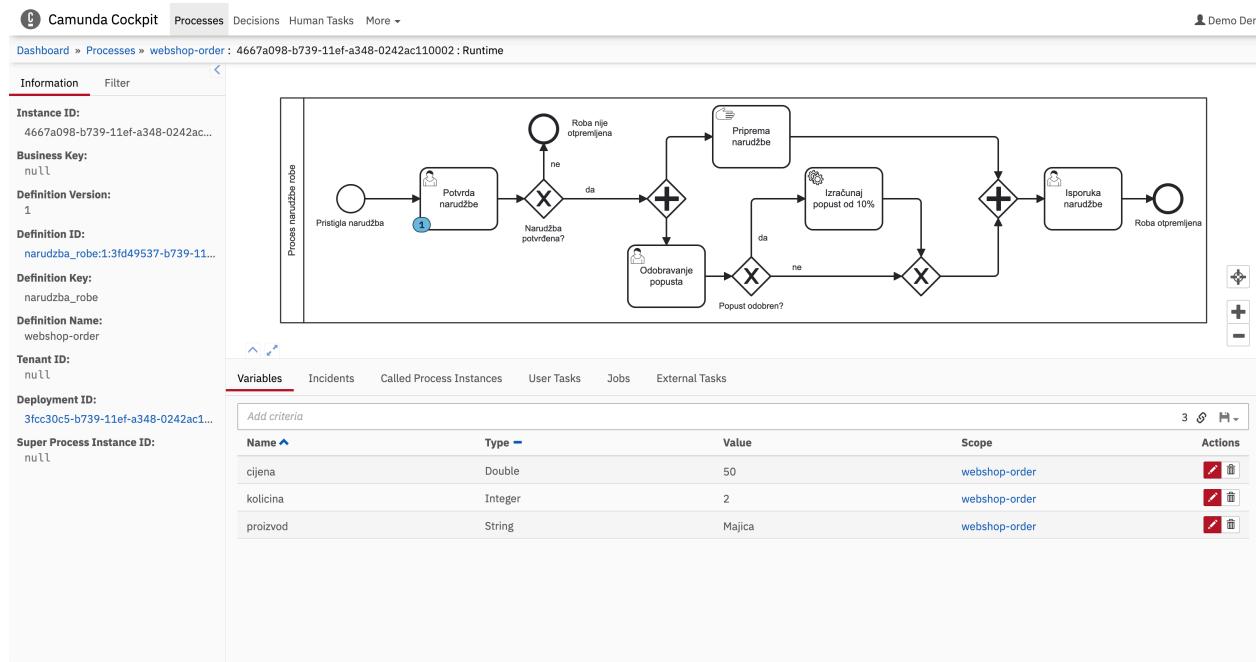
U našem slučaju:

```
 ${execution.setVariable("ukupna_cijena", ukupna_cijena - ukupna_cijena*0.1)}
```



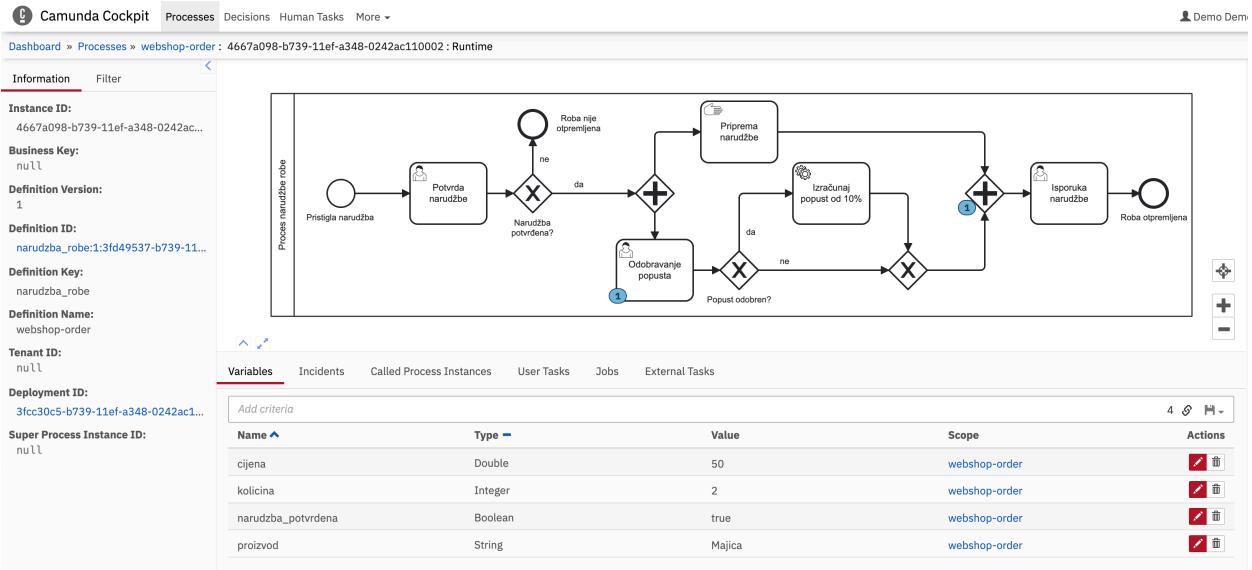
Definiranje izraza za izračun popusta od 10% na servisnom zadatku "Izračunaj popust od 10%"

To je to! **Redployajte novu verziju procesa i pokrenite novu instancu procesa kroz REST API ili Tasklist.** Dodajte početne procesne varijable: `proizvod`, `cijena`, `kolicina` i pratite tijek procesa kroz **Cockpit**.



Početno stanje instance procesa s varijablama `proizvod`, `cijena` i `kolicina`, čekanje na aktivnost "Potvrda narudžbe"

Nakon potvrde narudžbe, paralelno se izvršavaju aktivnosti "Priprema narudžbe" i "Odobravanje popusta". Međutim, proces čeka na egzekuciju "Odobravanje popusta" budući da je to User Task.



Prikaz aktivne instance procesa s tokenom na aktivnosti "Odobravanje popusta" i AND merge skretnici budući da se manualni taskovi preskaču

Otvaramo **Tasklist** i odabiremo zadatak "odobravanje popusta". Unosimo podatke i odobravamo popust.

Tasklist interface:

- My Tasks (3)
- Odobravanje popusta (webshop-order) - Demo Demo
- Assign Reviewer (Review Invoice) - Demo Demo
- Assign Reviewer (Review Invoice) - Demo Demo

Task details for Odobravanje popusta:

**Odobravanje popusta**  
webshop-order (v. v5.0)

**Form**

**Odobravate li popust od 10%?**

**Molimo unesite vaše ime**  
Marko

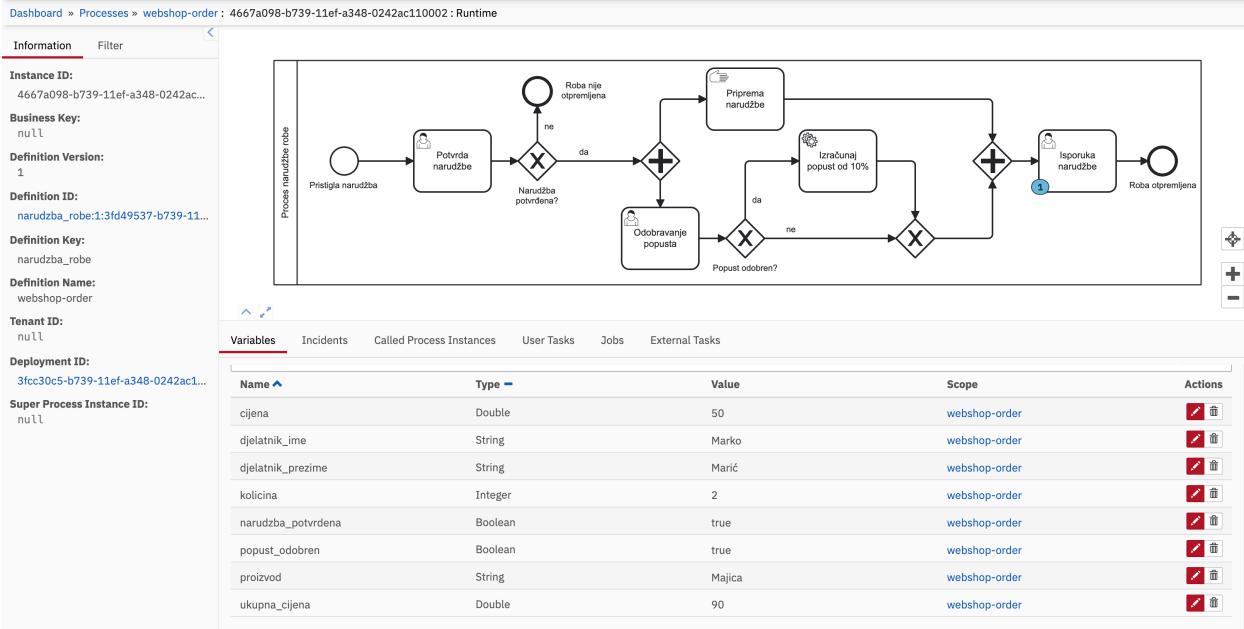
**Molimo unesite vaše prezime**  
Marić

**Buttons:** Save, Complete

Generirana forma za "Odobravanje popusta" s unesenim podacima za procesne varijable:

popust\_odobren, djetatnik\_ime i djetatnik\_prezime

Otvorite **Cockpit** i pogledajte stanje procesne instance i unesenih varijabli. Vidjet ćete da se izračunao popust od 10% i pregazio vrijednost procesne varijable ukupna\_cijena, koja je bila 100. Token se sada nalazi na aktivnosti "Isporuka narudžbe", kako se instanca ne bi završila odmah (premda nismo definirali kako dalje).



Prikaz aktivne instance procesa s tokenom na aktivnosti "Isporuka narudžbe" nakon izračuna popusta od 10% i pohranjenih varijabli.

# Samostalni zadatak za Vježbu 5

---

Modelirajte jednostavni proces prijave studentske prakse na Fakultetu informatike u Puli. Postoje 3 sudionika u procesu prakse:

1. **Student**
2. **Poslodavac**
3. **Profesor**

Proces započinje kod studenta odabirom zadatka za praksu. Student ispunjava web formu gdje unosi svoje ime, prezime, JMBAG i šifru zadatka (izmislite podatke).

Sljedeći korak je odobravanje prakse od strane profesora. Profesor pregledava podatke studenta i šifru zadatka u web sučelju, a nakon toga odobrava ili odbija prijavu. Ako prijava nije prihvaćena, proces se vraća na studenta i njegovu aktivnost ispunjavanja web forme. Ako profesor prihvati prijavu, proces se nastavlja kod poslodavca. Poslodavac provodi intervju sa studentom, a nakon toga odlučuje hoće li ga prihvati ili odbiti. Ako ga odbije, proces se ponovno vraća na studenta i njegov unos podataka. Ako ga prihvati, proces ide prema studentu koji sad mora unijeti kratak opis zadatka, datum izvođenja prakse te ime i prezime mentora koji mu je dodijeljen te istovremeno prema profesoru kojeg se samo obavještava. Nakon izvršavanja tih paralelnih aktivnosti, proces se završava.

Nakon što ste modelirali proces, implementirajte procesnu aplikaciju u **Camundi 7**:

- Dodajte definirane korisničke aktivnosti i korespondirajuće forme
- Definirajte procesne varijable i njihove vrijednosti
- Definirajte skretnice i uvjete na izlaznim tokovima
- Obavještavanje sudionika procesa ne implementirate

Predajete isključivo `.bpnn` datoteku procesa i aplikacije definirane u Camunda Modeleru.

---

## Upravljanje poslovnim procesima (UPP)

---

**Nositelj:** izv. prof. dr. sc. Darko Etinger

**Asistent:** Luka Blašković, mag. inf.

**Ustanova:** Sveučilište Jurja Dobrile u Puli, Fakultet informatike u Puli



Fakultet informatike u Puli

## (6) Servisna arhitektura procesne aplikacije

---

#6

UPP

Servisna arhitektura Camunda aplikacije obuhvaća dizajn i implementaciju raspodijeljenog sustava temeljenog na malim servisima (mikroservisi) koji komuniciraju preko REST API-ja. Camunda 7, kao jezgra procesne aplikacije pruža mogućnost izvođenja poslovnih procesa, njihovo upravljanje i praćenje, ali i integraciju s mikroservisima i vanjskim sustavima. Mikroservisi su ništa drugo nego neovisne aplikacije koje obavljaju specifične zadatke, u ovom kontekstu možemo ih zamisliti kao jednostavne REST API poslužitelje koji obavljaju određene radnje koje sami definiramo. U ovoj skripti naučit ćete integrirati jednostavne mikroservise s Camunda procesnim engineom kroz servisne zadatke i Express.js poslužitelje.

 Posljednje ažurirano: 5.1.2025.

## Sadržaj

---

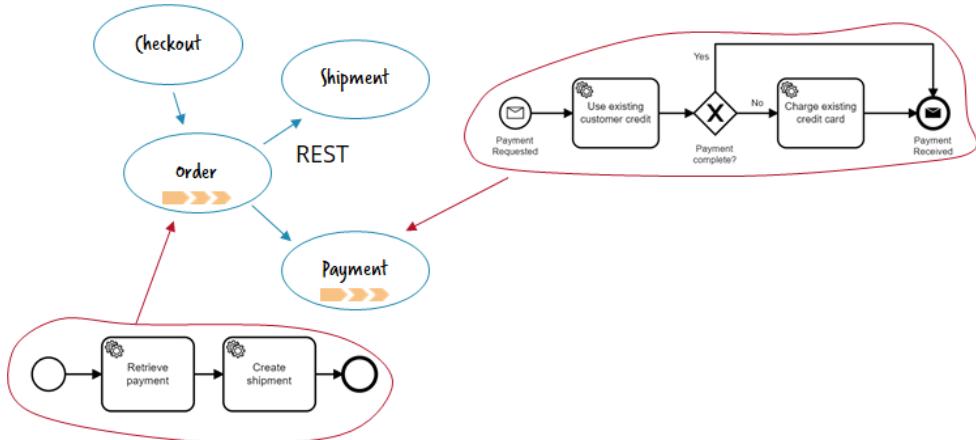
- [Upravljanje poslovnim procesima \(UPP\)](#)
- [\(6\) Servisna arhitektura procesne aplikacije](#)
  - [Sadržaj](#)
- [1. Servisni zadaci \(eng. Service Task\)](#)
  - [1.1 Priprema poslužitelja](#)
  - [1.2 Slanje HTTP GET zahtjeva](#)
  - [1.3 Dohvaćanje statusnog koda \(`statusCode`\)](#)
  - [1.4 Dohvaćanje tijela odgovora \(`response`\)](#)
  - [1.5 Slanje HTTP POST zahtjeva](#)
- [2. Otpremni zadaci \(eng. Send Task\)](#)
  - [2.1 Priprema poslužitelja za automatsko slanje e-maila](#)
  - [2.2 Email.js - priprema predloška](#)
  - [2.3 Implementacija slanja e-mail poruke](#)
  - [2.4 Definiranje `Send Task` aktivnosti](#)

# 1. Servisni zadaci (eng. Service Task)

Do sam smo iz vježbi najmanje govorili o servisnim zadacima (*eng. Service task*), a oni su zapravo jedan od najvažnijih elemenata procesnih aplikacija. Naučili ste da servisne zadatke koristimo za izvođenje vanjskih, automatiziranih operacija gdje imamo ulazne podatke, želimo napraviti nekakvu transformaciju ili akciju, te dobiti izlazne podatke.

**Camunda** podržava različite načine implementacije servisnih zadataka, npr. kroz JavaDelegate sučelje (Java), Expressione, Script Task (JavaScript, Groovy, Python, Ruby), ali i kroz REST API pozive (HTTP protokol). Mi ćemo se fokusirati na posljednji princip, odnosno na REST API pozive.

Za slanje HTTP zahtjeva koristit ćemo **Camunda7 Connectors API**, preciznije [http-connector](#) modul. Ovaj modul dolazi s Camunda platformom te ga nije potrebno naknadno instalirati.



Ilustracija servisne arhitekture procesne aplikacije

## 1.1 Priprema poslužitelja

Prije nego što krenemo s implementacijom servisnih zadataka, potrebno je pripremiti poslužitelj na koji ćemo slati HTTP zahtjeve. Poslužitelj može definirati u bilo kojem programskom jeziku/razvojnog okruženju, no mi ćemo koristiti Node.js i Express.js.

Izradite novi direktorij `express-server` i inicijalizirajte novi Node.js projekt:

```
mkdir express-server
cd express-server
npm init -y
```

Instalirajte Express.js:

```
npm install express
```

U `package.json` podesite `"type": "module"`, kako biste mogli koristiti `ES6` module te definirajte osnovni `Express.js` poslužitelj:

```
// express-server/index.js

import express from "express";

const PORT = 8000;

const app = express();
app.use(express.json());

app.get("/", (req, res) => {
  res.send("Pozdrav iz Express poslužitelja!");
});

app.listen(PORT, () => {
  console.log(`Poslužitelj sluša na adresi http://localhost:${PORT}`);
});
```

Provjerite radi li poslužitelj slanjem GET zahtjeva na `http://localhost:8000`.

Kako bismo mogli nesmetano slati zahtjeve s Camunda platforme, potrebno je omogućiti CORS (Cross-Origin Resource Sharing) politiku. Instalirajte `cors` paket:

```
npm install cors
```

Dodajemo `cors` middleware u `Express.js` poslužitelj:

```
// express-server/index.js

import express from "express";
import cors from "cors";

const PORT = 8000;

const app = express();
app.use(express.json());
app.use(cors());

app.get("/", (req, res) => {
  res.send("Pozdrav iz Express poslužitelja!");
});

app.listen(PORT, () => {
  console.log(`Poslužitelj sluša na adresi http://localhost:${PORT}`);
});
```

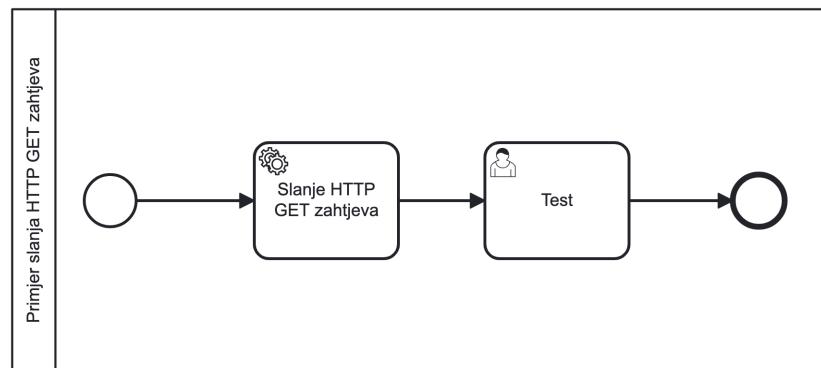
To je to! Za sada. Vraćamo se u Camunda Modeler 

## 1.2 Slanje HTTP GET zahtjeva

Definirat ćemo jednostavni proces koji se sastoji od jednog servisnog zadatka koji šalje HTTP GET zahtjev na naš Express.js poslužitelj.

Kako bismo vidjeli što se dešava, odnosno kako naša procesna instanca ne bi odmah završila, dodat ćemo i **User Task** neposredno nakon **Service Task** elementa.

Dodajte novo polje te u postavkama postavite osnovne podatke kako bi mogli izraditi procesnu instancu.



Jednostavna procesna definicija koja se sastoji od **Service Task** i **User Task** elemenata

Definirat ćemo i jednostavnu formu za **User Task** element:

USER TASK  
Test

General

Name: Test

ID: Activity\_1mzdr55

Documentation

Element documentation

User assignment

Forms

Type: Generated Task Forms

Form fields

+ 1

test

ID: test

Refers to the process variable name

Label: Želite li nastaviti?

Type: boolean

Default value

Constraints

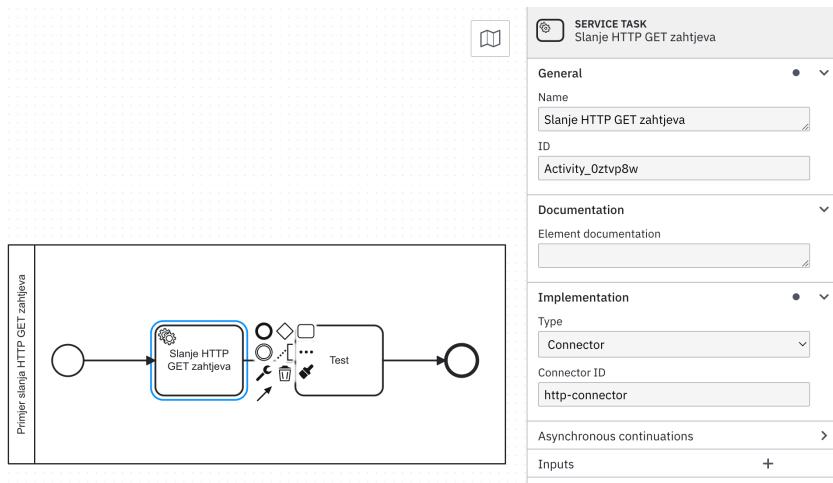
Properties

Dodavanje polja u **User Task** formu

Ako već niste, pokrenite **Camunda 7** preko Dockera:

```
docker run -d --name camunda -p 8080:8080 camunda/camunda-bpm-platform:latest
```

Za kraj, kako ne bi dobili grešku, moramo odabratи **connector** implementaciju servisnog zadatka, a za ID postaviti **http-connector**.



Deployajte procesnu definiciju na *Camunda Engine* i pokrenite novu procesnu instancu.

Vidimo da je *deployment* procesne definicije bio uspješan, ali procesnu instancu nije moguće pokrenuti. U konzoli Camunda Modelera vidjet ćete grešku:

```
HTCL-02005 Request url required. [ start-instance-error ]
```

Ova greška se javlja jer nismo definirali **URL** na koji će se poslati HTTP zahtjev.

Već iz web aplikacija znamo da su **obavezni dijelovi HTTP zahtjeva** `URL` i `method`, a **opcionarni dijelovi** su `payload` i `headers`. Isto primjenjujemo i ovdje.

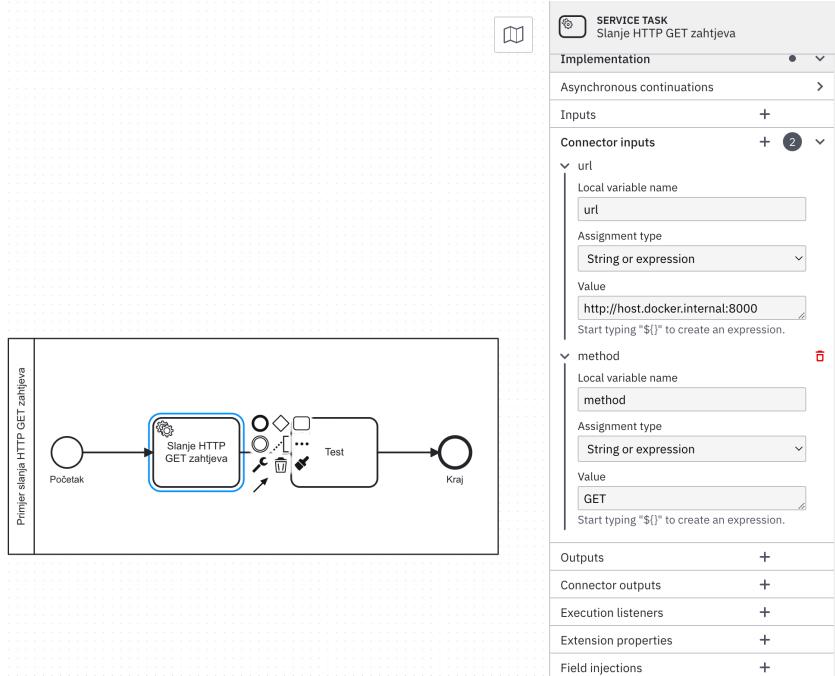
- `url`: ciljani URL gdje sluša naš poslužitelj
- `method`: HTTP metoda koju koristimo (GET, POST, PUT, DELETE, PATCH)
- `payload`: tijelo HTTP zahtjeva tj. *key-value* podaci koje šaljemo (JSON, XML)
- `headers`: dodatno zaglavlje HTTP zahtjeva (npr. Content-Type, Authorization)

Ove podatke navodit ćemo u **Connector inputs** polju servisnog zadatka.

- **Local variable name** = `url`, **Assignment type** = `String or expression`, **Value** = `http://host.docker.internal:8000`
- **Local variable name** = `method`, **Assignment type** = `String or expression`, **Value** = `GET`

Razlog zašto koristimo `http://host.docker.internal:8000` umjesto `http://localhost:8000` je taj što se `localhost` u kontekstu kontejnera ne referencira na naše računalo, već na internu adresu docker kontejnera. Više informacija o tome na: <https://docs.docker.com/engine/network/>

`http://host.docker.internal` je adresa koja **omogućava pristup resursima na domaćinu iz Docker containera**.



Dodavanje `Connector inputs` polja (`url` i `method`) u servisni zadatak "Slanje HTTP GET zahtjeva"

Za kraj, dodat ćemo i jednostavan `console.log` unutar definicije rute na Express.js poslužitelju kako bismo se uvjerili da je zahtjev uspješno poslan.

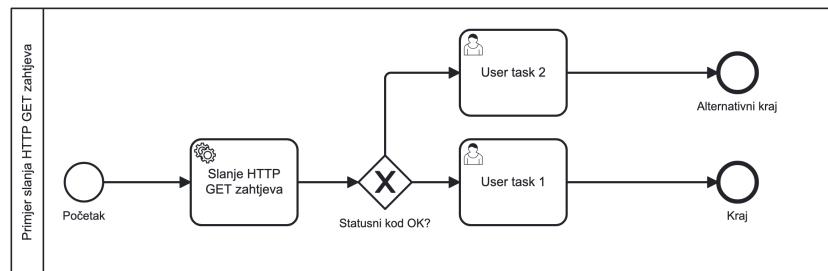
Redeployajte procesnu definiciju i pokušajte ponovno pokrenuti procesnu instancu. Pazite da je Express.js poslužitelj pokrenut.

Ako ste sve napravili ispravno, trebali biste vidjeti ispis u konzoli Express.js poslužitelja koji dokazuje da je GET zahtjev uspješno poslan 🚀

## 1.3 Dohvaćanje statusnog koda (statusCode)

Recimo da želimo preusmjeriti tok procesa koristeći XOR skretnicu temeljem statusnog koda koji dobijemo kao odgovor na HTTP zahtjev. Ukoliko je statusni kod 200, preusmjerit ćemo tok na `User Task 1`, a u suprotnom na `User Task 2`.

Dakle, želimo implementirati sljedeći sekvenčijalni tok:



Što se tiče `Connector outputs` polja, dostupne su sljedeće varijable:

- `response`: tijelo odgovora HTTP zahtjeva (`string`)
- `statusCode`: statusni kod HTTP odgovora (`Integer`)
- `responseHeaders`: zaglavlje HTTP odgovora (`Map<String, String>`)

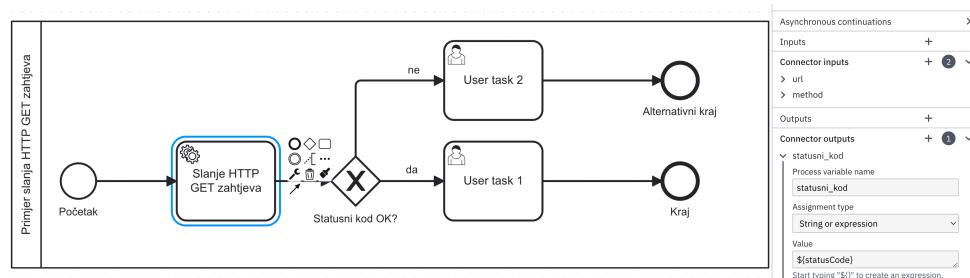
Kako bismo mogli dohvatiti statusni kod, potrebno je dodati novu varijablu u `connector outputs` polje. Međutim, uočite da je prvo polje zapisa naziva `Process variable name`, dakle varijablu možemo nazvati kako god želimo, ona će postati **procesna varijabla** dostupna u cijelom procesu (kao što smo ih definirali i u prethodnim vježbama).

Namjerno ćemo ju nazvati `statusni_kod` kako biste uočili razliku. Za dohvaćanje samog statusnog koda, moramo koristiti Camunda Expression `#{statusCode}`

- **Process variable name** = `statusni_kod`, **Type** = `String or expression`, **Value** = `#{statusCode}`

Na XOR skretnicu dodajte sljedeće uvjete:

- `#{statusni_kod == 200}` : nastavlja sekvencijalni flow prema `User Task 1`
- `#{statusni_kod != 200}` : nastavlja sekvencijalni flow prema `User Task 2`



Za kraj, unutar Express.js poslužitelja vratite statusni kod 200 kako biste preusmjerili tok na `User Task 1`.

```
// express-server/index.js

app.get("/", (req, res) => {
  console.log("Zahtjev primljen!");
  res.status(200).send("Pozdrav iz Express.js poslužitelja!");
});
```

Redeployajte procesnu definiciju i pokušajte ponovno pokrenuti procesnu instancu.

Ako otvorite pregled procesne instance u **Cockpitu**, vidjet ćete da je tok preusmjeren prema `User Task 1` jer je statusni kod 200. Pohranjenu procesnu varijablu `statusni_kod` možete vidjeti u detaljima procesne instance.

- **Name** = `statusni_kod`, **Type** = `Integer`, **Value** = `200`, **Scope** = `connector_GET`

Information

Instance ID: 6cef23a5-caa9-11ef-81aa-0242ac110002

Business Key: null

Definition Version: 5

Definition ID: connector\_GET:5:6a6c7064-caa9-11ef-81aa-0242ac110002

Definition Key: connector\_GET

Definition Name: connector\_GET

Tenant ID: null

Deployment ID: 6a54a2a2-caa9-11ef-81aa-0242ac110002

Super Process Instance ID: null

Variables

Name	Type	Value	Scope	Actions
statusni_kod	Integer	200	connector_GET	<span style="color: green;">✓</span> <span style="color: red;">✗</span>

Testirajte i drugi uvjet tako da promijenite statusni kod u Express.js poslužitelju na npr. 404.

## 1.4 Dohvaćanje tijela odgovora (response)

Ukoliko želimo dohvatiti tijelo odgovora HTTP zahtjeva, koristimo varijablu `response`. U ovom primjeru, želimo dohvatiti tijelo odgovora i pohraniti ga u procesnu varijablu kako bismo ga mogli koristiti u dalnjem toku procesa.

Možemo isto testirati odmah, budući da naš poslužitelj metodom `res.send` šalje odgovor koji je zapravo tijelo odgovora (nije JSON).

Dodajte novu procesnu varijablu `odgovor` koja će pohraniti tijelo odgovora HTTP zahtjeva. Varijablu dodajte na jednak način: kao **Connector output** varijablu s Expression izrazom  `${response}` .

Redeployajte procesnu definiciju i pokušajte ponovno pokrenuti procesnu instancu.

Možemo vidjeti procesnu varijablu `odgovor` u detaljima procesne instance u **Cockpitu**.

Information

Instance ID: 414c38a0-caaa-11ef-81aa-0242ac110002

Business Key: null

Definition Version: 6

Definition ID: connector\_GET:6:3fa2803f-caaa-11ef-81aa-0242ac110002

Definition Key: connector\_GET

Definition Name: connector\_GET

Tenant ID: null

Deployment ID: 3f9d01fd-caaa-11ef-81aa-0242ac110002

Super Process Instance ID: null

Variables

Name	Type	Value	Scope	Actions
odgovor	String	Pozdrav iz Express.js poslužitelja!	connector_GET	<span style="color: green;">✓</span> <span style="color: red;">✗</span>
statusni_kod	Integer	200	connector_GET	<span style="color: green;">✓</span> <span style="color: red;">✗</span>

Međutim, iz web aplikacija znamo da nije uobičajeno slati tekstualne odgovore na ovaj način, već koristimo JSON. Vratit ćemo jednostavan objekt `key-value` parova kao JSON odgovor te ga zatim pohraniti u procesnu varijablu.

Recimo da naš API simulira dohvaćanje podataka o korisniku. Dodat ćemo novu rutu u Express.js poslužitelj koja vraća JSON odgovor s imenom i prezimenom korisnika.

```
app.get("/user", (req, res) => {
  console.log("Zahtjev primljen na /user");
  res.status(200).json({
    ime: "Marko",
    prezime: "Marić",
  });
});
```

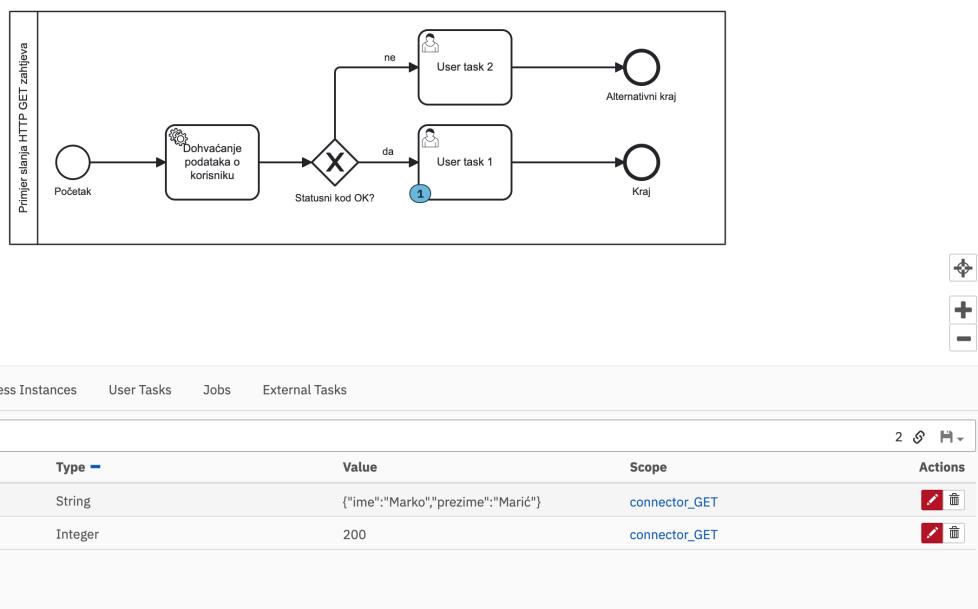
Service task ćemo sad preimenovati u "Dohvaćanje podataka o korisniku" te promijeniti URL na `http://host.docker.internal:8000/user`.

Redeployajte procesnu definiciju i pokušajte ponovno pokrenuti procesnu instancu.

Ako provjerite detalje procesne instance u **Cockpitu**, vidjet ćete da je tijelo odgovora pohranjeno u procesnu varijablu `odgovor`, međutim kao `String`.

- Name = `odgovor`, Type = `String`, Value = `{"ime": "Marko", "prezime": "Marić"}`, Scope = `connector_GET`

Ukoliko želimo koristiti taj odgovor kao objekt, moramo odraditi proces **deserializacije**.



Ideja je da pohranimo tijelo odgovora u dvije nove procesne varijable `ime` i `prezime`

U Camundi postoji mnogo načina za odraditi ovu transformaciju, primjerice isto je moguće postići koristeći **Script Task** element u kojeg ćemo direktno pisati JavaScript ili Java kod. Međutim, mi ćemo nastojati iskoristiti stvari rješavati na što jednostavniji način, koristeći **Expressione**.

[Camunda Spin](#) biblioteka nudi jednostavan API za rad s XML i JSON struktogram. Međutim, jednako kao i Connector, nije ju potrebno naknadno instalirati jer dolazi s Camunda platformom.

**Camunda Spin Expressione** počinjemo pisati sa slovom `s`. Sintaksa je sljedeća:

```
 ${S("JSON").prop("key")}
```

Budući da je naš JSON odgovor pohranjen u procesnu varijablu `odgovor`, možemo upotrijebiti procesnu varijablu direktno kao argument Spin Expressiona.

```
 ${S(odgovor).prop("ime")}
```

Za kraj, sam proces deserijalizacije radimo naredbom `.value()`:

```
 ${S(odgovor).prop("ime").value()}
```

To je to! Sad možemo odraditi proces deserijalizacije u jednom Expression izrazu, bez potrebe za pisanjem koda.

Ostalo je jedino pitanje - gdje dodajemo ove Expressione?

Svaka aktivnost (zadatak) pa tako i `Service Task` ima dostupna polja za **Input/Output Mapping**:

- `Inputs`: **podaci koji se koriste kao ulazni parametri za aktivnost**. Ovdje možemo definirati koje procesne varijable će se koristiti kao ulazni parametri za tu aktivnost.
- `Outputs`: **rezultati aktivnosti**. Ovdje možemo definirati koje procesne varijable će se koristiti kao izlazni parametri za tu aktivnost.

U skripti `UPP5` već ste vidjeli kako koristimo ova polja, no mi ćemo sada iskoristiti `Outputs` polje kako bismo deserijalizirali tijelo odgovora.

Zašto `Outputs`? Jer želimo da **rezultat** aktivnosti "Dohvaćanje podataka o korisniku" budu procesne varijable `ime` i `prezime`, a ne `odgovor`.

**Oprez:** pazite da ne pomiješate Input s Connector inputima i Output s Connector outputima.

Connector inputi i outputi su vezani uz HTTP zahtjev Connector API-ja, dok su **Input/Output Mapping vezani uz samu aktivnost**.

Napravit ćemo mali rezime prije nego nastavimo:

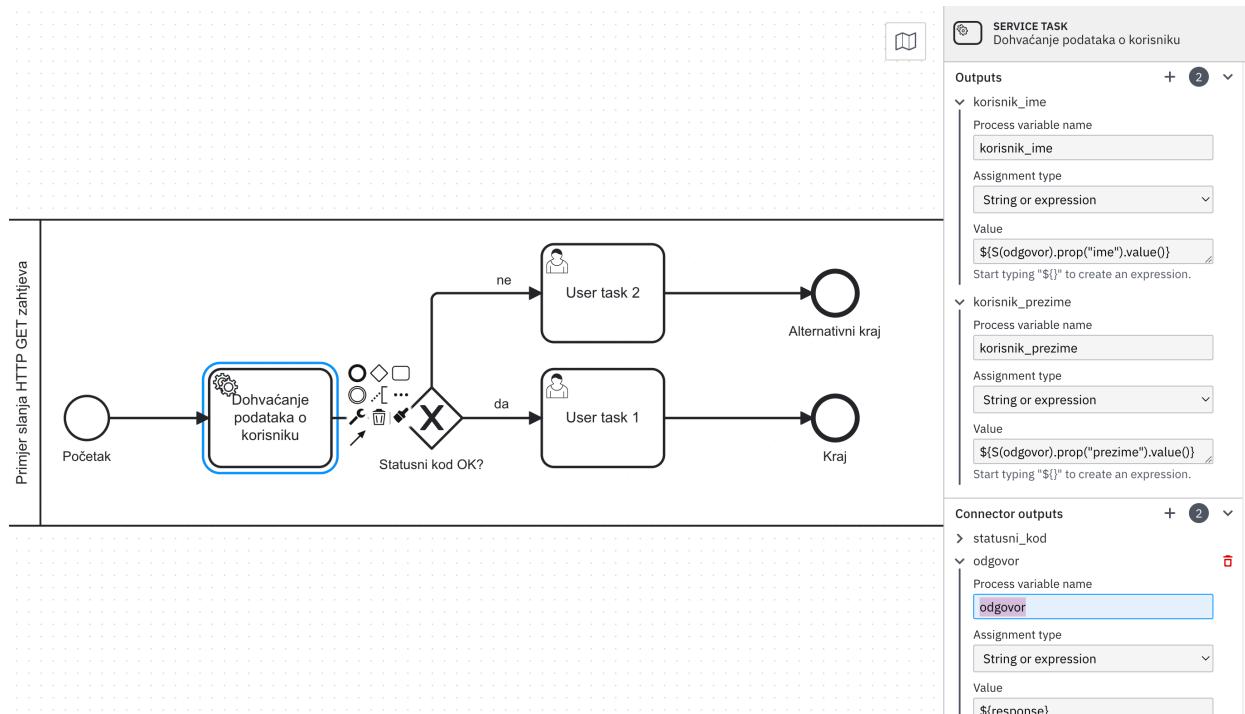
1. Procesni *engine* nailazi na `Service Task` "Dohvaćanje podataka o korisniku"
2. Navedeni `Service Task` implementiran je kao `Connector`, preciznije `http-connector` jer smo rekli da koristimo tu implementaciju za slanje HTTP zahtjeva (iako ih ima više)
3. U `Connector inputs` polju definirali smo URL i metodu HTTP zahtjeva, konkretno to je slanje GET zahtjeva koji dohvaca podatak o korisniku s našeg Express.js poslužitelja
4. U `Connector outputs` polju definirali smo da želimo pohraniti **statusni kod i tijelo odgovora** u procesne varijable
5. Kako je tijelo odgovora JSON, želimo ga deserijalizirati i pohraniti u procesne varijable `ime` i `prezime`. Sad već imamo taj JSON u procesnoj varijabli `odgovor`, pa koristimo Spin Expressione za deserijalizaciju unutar polja `Outputs` za `Service Task` element.

Dakle, **S Expressioni** koje ćemo koristiti u `Outputs` polju su sljedeći:

- `ime: ${S(odgovor).prop("ime").value()}`
- `prezime: ${S(odgovor).prop("prezime").value()}`

Na isti način možemo deserijalizirati bilo koji drugi primitivni tip podatka.

Redeployajte procesnu definiciju i pokušajte ponovno pokrenuti procesnu instancu.



**Deserijalizacija JSON podataka** iz tijela odgovora u procesne varijable `ime` i `prezime`

Provjerite vrijednosti svih procesnih varijabli u Cockpitu.

## 1.5 Slanje HTTP POST zahtjeva

Naučili smo kako poslati HTTP GET zahtjev, dohvatiti statusni kod i tijelo odgovora, te deserijalizirati JSON odgovor. Sada ćemo vidjeti kako poslati HTTP POST zahtjev, odnosno **kako poslati vrijednosti procesnih varijabli na poslužitelj**.

Idemo implementirati jednostavni proces gdje korisnik unosi svoje prezime, a servisni zadatak šalje to prezime na poslužitelj kako bi dohvatio ukupne podatke o korisniku (`ime`, `prezime`, `username`, `email`). Ako poslužitelj ne pronađe korisnika, vraća statusni kod 404. U suprotnom vraća statusni kod 200 i JSON objekt koji predstavlja korisnika.

Podatke, iako bi trebali biti pohranjeni u bazi podataka, ćemo pohraniti samo *in-memory*.

Dodat ćemo novu rutu u Express.js poslužitelj koja simulira dohvaćanje korisnika **na temelju prezimena**.

Iako prema standardu REST protokola, ovu radnju bi modelirali GET metodom i **parametrom rute** (npr. `/user/Marić`) ili **query parametrima** (npr. `/user?prezime=Marić`), mi ćemo koristiti **POST metodu i poslati prezime u tijelu zahtjeva**, budući da ova distinkcija nije bitna u kontekstu ovog kolegija:

Definirat ćemo nekoliko korisnika u memoriji:

```
// express-server/index.js
```

```
// legendarni trio
let korisnici = [
  {
    ime: "Marko",
    prezime: "Marić",
    username: "marko.maric",
    email: "mmaric@gmail.com"
  },
  {
    ime: "Pero",
    prezime: "Perić",
    username: "ppppp.pero",
    email: "pero123@gmail.com"
  },
  {
    ime: "Ana",
    prezime: "Anić",
    username: "ana.anic",
    email: "aanic@gmail.com"
  }
]
```

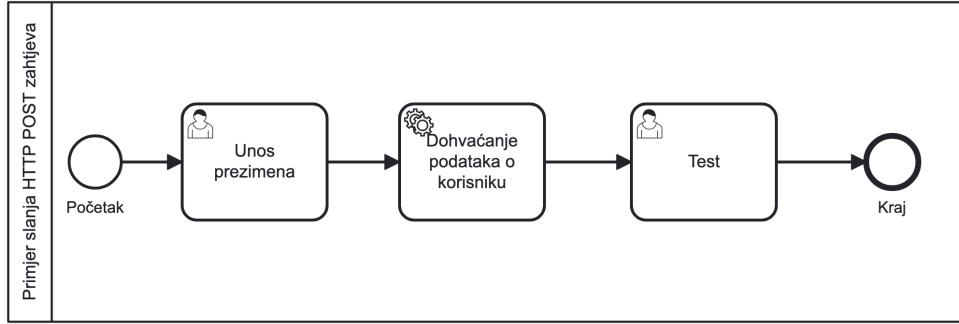
Dodajemo definiciju POST rute na poslužitelju:

```
// express-server/index.js

app.post("/user", (req, res) => {
  console.log("Zahtjev primljen na /user");
  let prezime = req.body.prezime;
  let korisnik = korisnici.find((korisnik) => korisnik.prezime === prezime); // pronalazak
  korisnika
  if (!korisnik) {
    return res.status(404).json({ message: "Korisnik nije pronađen." });
  } else {
    return res.status(200).json(korisnik);
  }
});
```

Implementirat ćemo sljedeći proces: "Primjer slanja HTTP POST zahtjeva":

1. Korisnik unosi svoje prezime preko **User Task** elementa "Unos prezimena"
2. **Service Task** "Dohvaćanje podataka o korisniku" **Šalje prezime na poslužitelj HTTP POST metodom** i vraća objekt korisnika
3. **User Task** "Test" samo stopira proces kako bi mogli vidjeti rezultate u Cockpitu



Procesna definicija sa servisnim zadatkom koji šalje HTTP POST zahtjev na poslužitelj

Dalje, definirat ćemo jednostavnu formu za unos prezimena u **User Task** elementu "Unos prezimena".

USER TASK  
Unos prezimena

General

ID  
Activity\_1tec2gw

Documentation

User assignment

Forms

Type  
Generated Task Forms

Form fields

prezime	+	1
ID	prezime	Refers to the process variable name
Label	Molimo da unesete prezime korisnika	
Type	string	
Default value		

Dodavanje polja za unos prezimena u **User Task** formu

Za **Service Task** element, postavite sljedeće **Connector inputs** vrijednosti:

- **Local variable name** = `url`, **Assignment type** = `String or expression`, **Value** = `http://host.docker.internal:8000/user`
- **Local variable name** = `method`, **Assignment type** = `String or expression`, **Value** = `POST`

Tijelo zahtjeva definiramo u varijabli `payload`. Međutim, moramo ga poslati u onom obliku koji poslužitelj očekuje, a to je JSON format s ključem `prezime` i vrijednošću koju korisnik unese (procesna varijabla `prezime`).

Prije slanja, dodat ćemo u implementaciji POST rute na poslužitelju `console.log` kako bismo ispisali tijelo zahtjeva.

```
// express-server/index.js

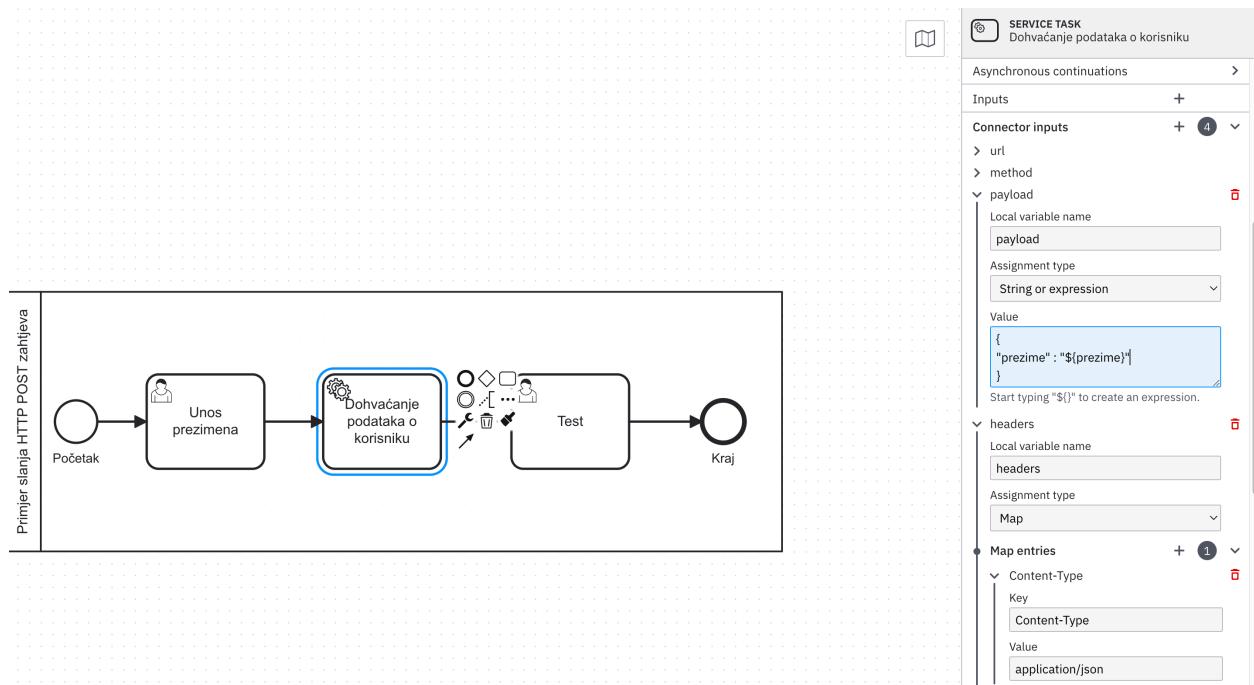
app.post("/user", (req, res) => {
  console.log("Zahtjev primljen na /user");
  console.log(req.body); // ispis tijela zahtjeva
  let prezime = req.body.prezime;
  let korisnik = korisnici.find((korisnik) => korisnik.prezime === prezime);
  if (!korisnik) {
    return res.status(404).json({ message: "Korisnik nije pronađen." });
  } else {
    return res.status(200).json(korisnik);
  }
});
```

U `Connector inputs` polju dodajemo novu varijablu `payload`.

- **Local variable name** = `payload`, **Assignment type** = `String or expression`, **Value** = `{"prezime": "${prezime}"}`

Dodatno, moramo poslati i zaglavje (`header`) kako bi poslužitelj znao da se radi o JSON formatu.

- **Local variable name** = `headers`, **Assignment type** = `Map`, **Map entries** (Key = `Content-Type`, Value = `application/json`)



**Obavezno** je potrebno proslijediti i `Content-Type` zaglavje kako bi poslužitelj ispravno interpretirao tijelo zahtjeva. Tada u `payload` možemo jednostavno pisati JSON objekt kao string.

Budući da svaki korisnik sadrži podatke o imenu, prezimenu, korisničkom imenu i e-mailu, dodat ćemo **preostale izlazne procesne varijable** u Outputs polju `Service Task` elementa.

- `ime: ${S(odgovor).prop("ime").value()}`
- `username: ${S(odgovor).prop("username").value()}`
- `email: ${S(odgovor).prop("email").value()}`

Redeployajte procesnu definiciju i pokušajte ponovno pokrenuti procesnu instancu.

Ako ste sve napravili ispravno, nakon unosa prezimena putem Tasklist aplikacije, vidjet ćete ispis tijela zahtjeva na poslužitelju, a zatim i dohvaćene podatke o korisniku u Cockpitu.

Camunda Cockpit   Processes Decisions Human Tasks More ▾

Dashboard » Processes » connector\_POST : 3d7593c6-cabd-11ef-b5d6-0242ac10002 : Runtime

Information Filter

Instance ID: 3d7593c6-cabd-11ef-b5d6-0242ac1...

Business Key: null

Definition Version: 8

Definition ID: connector\_POST:8:3b81b305-cabd-1...

Definition Key: connector\_POST

Definition Name: connector\_POST

Tenant ID: null

Deployment ID: 3b77c7f3-cabd-11ef-b5d6-0242ac1...

Super Process Instance ID: null

Diagram:

```
graph LR; Start((Početak)) --> Unos[Unos prezimena]; Unos --> Dohv[Dohvaćanje podataka o korisniku]; Dohv --> Test[Test]; Test --> End((Kraj));
```

Primer slajanja HTTP POST zahtjeva

Variables

Name	Type	Value	Scope	Actions
email	String	pero123@gmail.com	connector_POST	[edit] [trash]
ime	String	Pero	connector_POST	[edit] [trash]
odgovor	String	{"ime":"Pero","prezime":"Perić","username": "pero123@gmail.com","statusni_kod": 200}	connector_POST	[edit] [trash]
prezime	String	Perić	connector_POST	[edit] [trash]
statusni_kod	Integer	200	connector_POST	[edit] [trash]
username	String	ppppp.pero	connector_POST	[edit] [trash]

## 2. Otpremni zadaci (eng. Send Task)

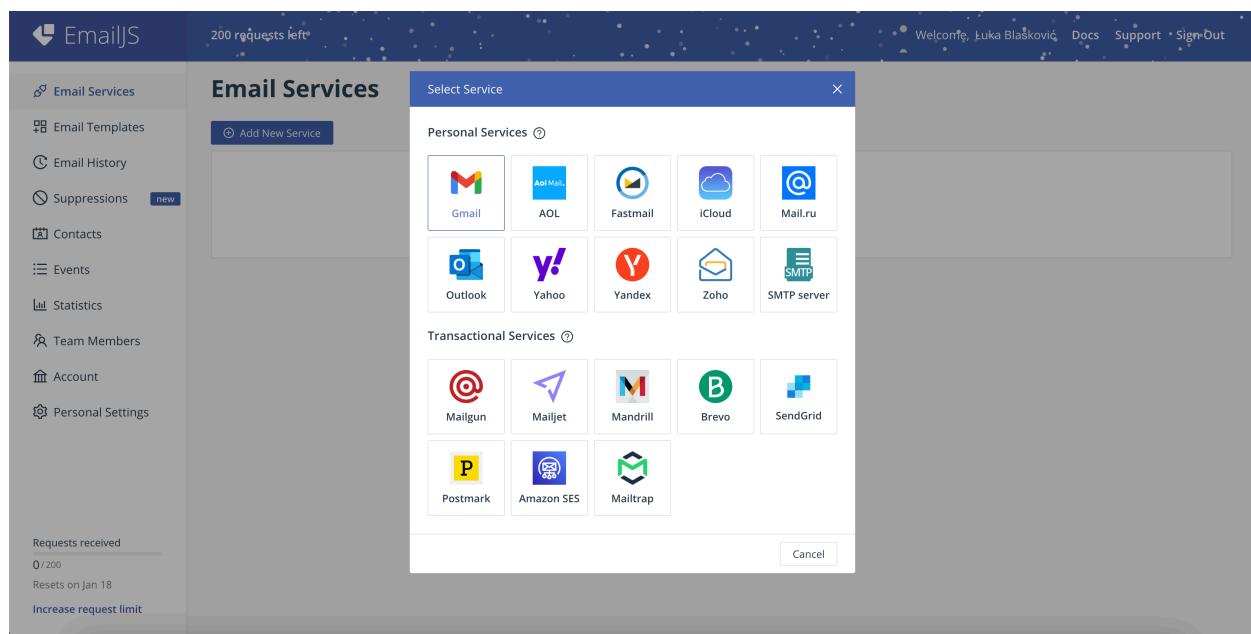
**Otpremni zadaci** (end. *Send Task*) predstavljaju aktinosti koje uključuju slanje poruka vanjskim dionicima, ali ih možemo koristiti i za pokretanje novih procesa.

Kao i kod servisnih zadataka, `Send Task` implementacija može se realizirati na različite načine, ali i korištenjem Connector API-ja za slanje HTTP zahtjeva koji smo već naučili. Prema tome, iskoristit ćemo istu implementaciju za slanje HTTP zahtjeva na poslužitelj (mikroservis) koji služi za slanje e-mail poruka.

Servis za slanje e-mail poruka može biti bilo koji, primjerice *Nodemailer*, *SendGrid*, *Mailgun*, itd. U našem slučaju, koristit ćemo [Email.js](#) servis. Bez obzira što je *Email.js* primarno namijenjen za korištenje u frontend aplikacijama, možemo ga koristiti i u backend aplikacijama preko REST API-ja.

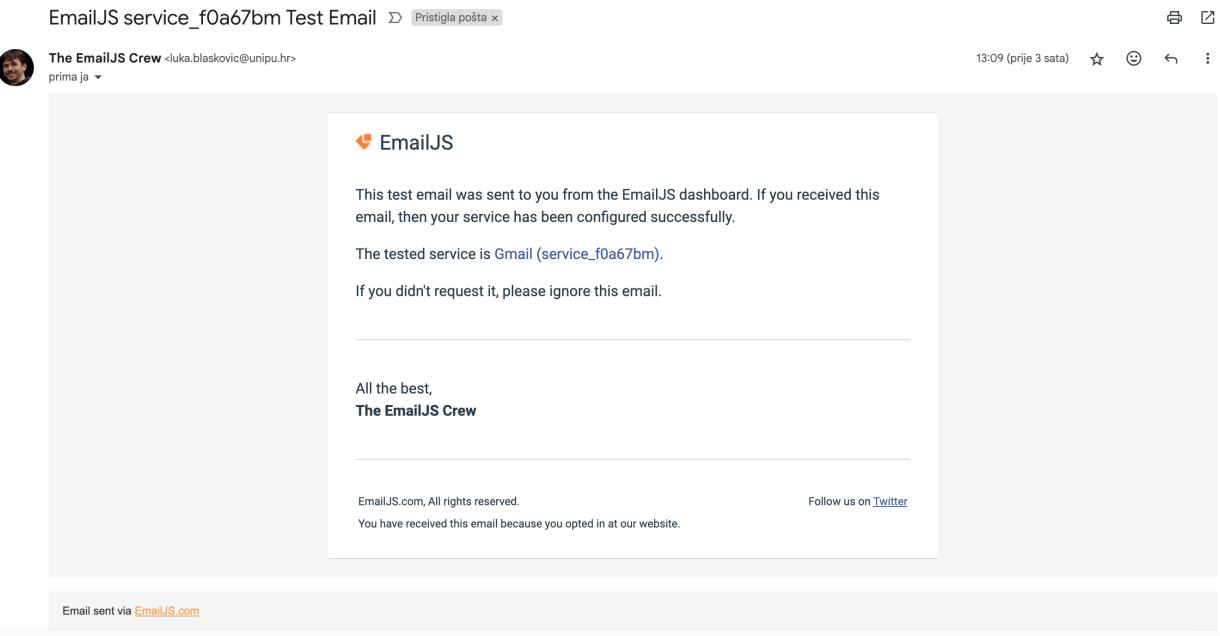
Registrirat ćemo novi račun na *Email.js* servisu, a nakon toga povezati naš Gmail račun putem kojeg ćemo slati e-mail (najjednostavniji način). U produkciji, vjerojatno nećete htjeti koristiti osobni Gmail račun za slanje e-mail poruka, već **SMTP** (eng. *Simple Mail Transfer Protocol*) poslužitelj vaše organizacije ili neki od gore navedenih servisa.

Izradite novi račun na [Email.js](#) servisu i povežite svoj **Gmail** račun.



Prilikom povezivanja morate dozvoliti pristup vašem Gmail računu i omogućiti **Slanje e-poruka u vaše ime**. Jednom kad povežete račun, možete poslati testnu poruku kako biste se uvjерili da je sve ispravno konfiguirirano.

Trebali biste dobiti sljedeći email u vašem sandučiću koji ste poslali "sami sebi":



## 2.1 Priprema poslužitelja za automatsko slanje e-maila

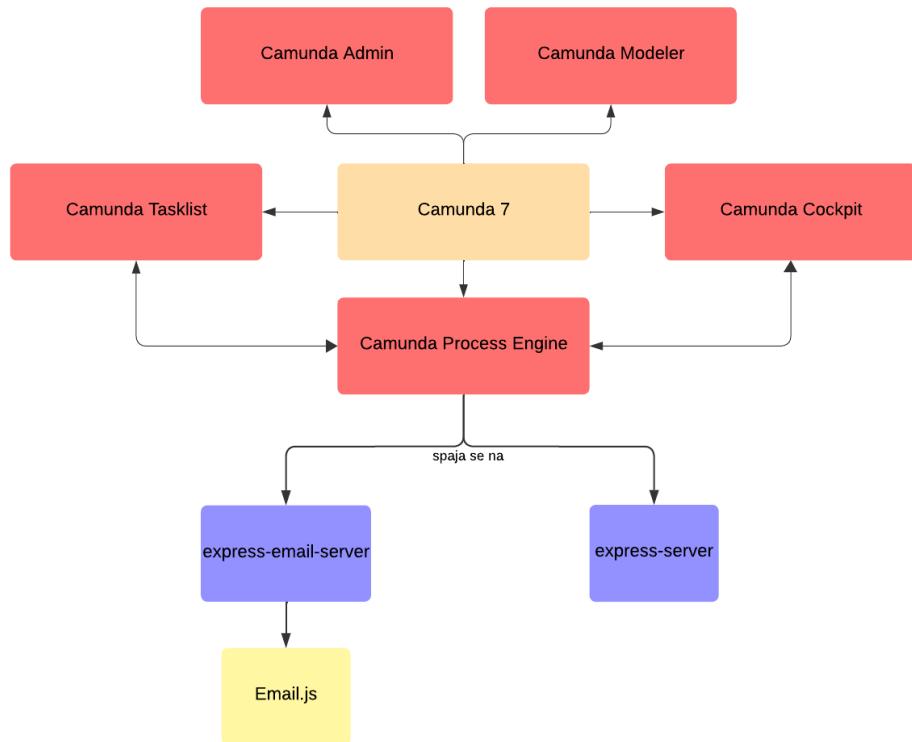
Sljedeći korak je implementacija Express.js poslužitelja koji će služiti kao **posrednik između Camunda Enginea i Email.js servisa**. Ideja je sljedeća:

1. Camunda Engine šalje HTTP POST zahtjev na naš Express.js poslužitelj s podacima o korisniku
2. Express.js poslužitelj obrađuje zahtjev i šalje e-mail poruku korisniku putem *Email.js* servisa
3. *Email.js* servis šalje e-mail poruku korisniku

Izradite novi direktorij `express-email-server`, inicijalizirajte novi Node.js projekt i instalirajte Express.js:

Naravno, moguće je iskoristiti postojeći Express.js poslužitelj koji smo definirali ranije, međutim kako su same procesne aplikacije bazirane na raspodijeljenoj arhitekturi, nije loše držati se te paradigme i odvojiti poslužitelje.

Prije nego nastavimo, nije loše pogledati kako do sada izgleda **raspodijeljena arhitektura naše procesne aplikacije**:



Ilustracija raspodijeljene arhitekture procesne aplikacije bazirane na Camundi 7

Naš `express-email-server` poslužitelj slušat će na portu `3000`:

```
// express-email-server/index.js

import express from "express";
import cors from "cors";

const PORT = 3000;

const app = express();
app.use(express.json());
app.use(cors());

app.get("/", (req, res) => {
  console.log("Zahtjev primljen!");
  res.status(200).send("Pozdrav iz express-email-server poslužitelja!");
});

app.listen(PORT, () => {
  console.log(`Poslužitelj sluša na adresi http://localhost:${PORT}`);
});
```

Kako koristimo Email.js servis na poslužiteljskoj strani, isto moramo dozvoliti u postavkama Email.js-a.

Otvorite sljedeću poveznicu: <https://dashboard.emailjs.com/admin/account/security>

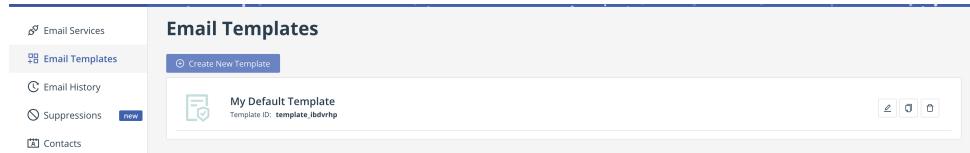
U odjeljku **API Settings** omogućite: "Allow EmailJS API for non-browser applications" i spremite promjene.

## 2.2 Email.js - priprema predloška

Prije nego krenemo s pisanjem koda, kreirat ćemo predložak e-mail poruke koji ćemo koristiti za slanje e-mail poruke korisniku.

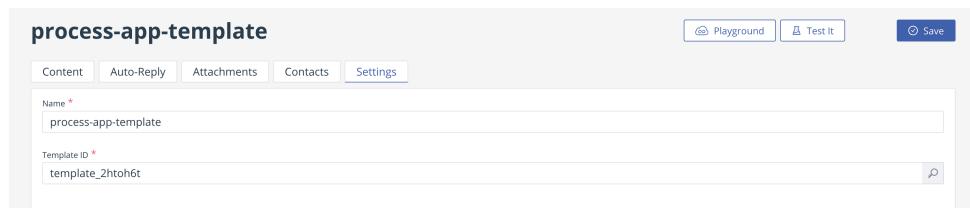
Predložak (*eng. Template*) možete kreirati na sljedećoj povezničkoj stranici: <https://dashboard.emailjs.com/admin/templates> ili odabirom **Email Templates** u glavnom izborniku.

Kliknite na **Create New Template** i izradite novi predložak. Nazovite ga `process-app-template`.



### Email.js - kreiranje novog predloška e-mail poruke (**Email Templates**)

Otvorite postavke predloška (*eng. Settings*) i promijenite njegov naziv, potom kopirajte negdje **ID predloška** (*eng. Template ID*) jer će nam trebati kasnije. Pohranite promjene.



U **Content** odjeljku možete definirati sadržaj e-mail poruke, uključujući naslov (*eng. Subject*) te sadržaj emaila (*eng. Content*).

Osim vizualnog editora, možete koristiti i HTML editor za preciznije uređivanje sadržaja e-mail poruke.

Odaberite **Edit Content** i uredite sadržaj emaila. Poslat ćemo korisniku jednostavni email s podacima o korisniku (ime, prezime, email, username) koje smo dohvatali iz procesa.

Kako su podaci o korisniku pohranjeni u procesnim varijablama, koje se proslijeđuju u tijelu HTTP POST zahtjeva na ovaj poslužitelj, moramo ih dohvatiti i koristiti u predlošku e-mail poruke. Za to koristimo tzv. **placeholders** koje pišemo duplim vitičastim zagradama `{ }`.

Dakle, naše varijable možemo koristiti na sljedeći način:

```
Ime: {{ime}}
Prezime: {{prezime}}
Korisničko ime: {{username}}
E-mail: {{email}}
```

*Primjer predloška:*

---

Pozdrav!

Šaljemo Vam podatke o korisniku:

**Ime:** {{ime}}  
**Prezime:** {{prezime}}  
**Email:** {{email}}  
**Username:** {{username}}

Lijepi pozdrav i ugodan dan,

Vaša Camunda!

---

Kao naslov emaila možemo postaviti: "Camunda: Podaci o korisniku"

Za pošiljatelja navodite vaš email koji ste koristili prilikom registracije Gmail servisa, a kao naziv pošiljatelja možete staviti "Moja Camunda" ili sl.

**Sve skupa trebalo bi izgledati ovako:**

## process-app-template

Content Auto-Reply Attachments Contacts Settings

Subject \*  
Camunda: Podaci o korisniku

Content \*  
 Desktop  Mobile

 Edit Content

Pozdrav!

Šaljemo Vam podatke o korisniku:

**Ime:** {{ime}}

**Prezime:** {{prezime}}

**Email:** {{email}}

**Username:** {{username}}

Lijepi pozdrav i ugodan dan,

Vaša Camunda!

To Email \*  
lukablasovic2000@gmail.com

From Name  
Moja Camunda

From Email \*  
 Use Default Email Address 

Reply To

Bcc

Cc

Primjer Email.js predloška e-mail poruke

Spremite promjene.

## 2.3 Implementacija slanja e-mail poruke

Sada kada smo pripremili predložak e-mail poruke, možemo implementirati Express.js poslužitelj koji će služiti kao posrednik između *Camunda Enginea* i *Email.js* servisa.

Provjerite još jednom jeste li omogućili **Allow EmailJS API for non-browser applications** u postavkama *Email.js* servisa kako bi stvari radile kako treba.

Definirajte POST rutu `/send-email` koja će obrađivati zahtjeve za slanje e-mail poruka.

```
app.post("/send-email", async (req, res) => {
  res.send("Zahtjev primljen!");
});
```

Kako šaljemo zahtjeve na REST API *Email.js* servisa, moramo koristiti neku HTTP klijentsku biblioteku. U ovom slučaju, koristit ćemo [Axios](#).

Instalirajte **Axios** biblioteku:

```
npm install axios
```

Instalirat ćemo i `dotenv` biblioteku kako bismo mogli koristiti `.env` datoteku za pohranu osjetljivih podataka kao što su **API ključevi** *Email.js* servisa i ID predloška.

```
npm install dotenv
```

Izrađujemo `.env` datoteku u korijenskom direktoriju projekta i dodajemo sljedeće 4 varijable:

```
SERVICE_ID= service_xxxxxxx (kopirati iz Email Services, odabir Gmail servisa)
TEMPLATE_ID= template_xxxxxxx (kopirati iz Email Templates/Settings, odabir predloška
process-app-template)
PUBLIC_KEY=Public Key (kopirati iz postavka Email.js servisa - Account/General)
PRIVATE_KEY=Private key (kopirati iz postavka Email.js servisa - Account/General)
```

Environment varijable učitavmao koristeći `dotenv` biblioteku:

```
// express-email-server/index.js

import dotenv from "dotenv";
dotenv.config();
const { SERVICE_ID, TEMPLATE_ID, PUBLIC_KEY, PRIVATE_KEY } = process.env;
```

Unutar `/send-email` rute definirat ćemo `try-catch` blok te unutar njega definirati Axios kod za slanje POST zahtjeva na *Email.js* servis te obradu eventualnih grešaka.

URL endpointa gdje šaljemo POST zahtjev je sljedeći: <https://api.emailjs.com/api/v1.0/email/send>

```
// express-email-server/index.js
```

```

app.post("/send-email", async (req, res) => {
  try {
    const response = await axios.post(
      "https://api.emailjs.com/api/v1.0/email/send", // Email.js REST API endpoint
      {
        service_id: SERVICE_ID,
        template_id: TEMPLATE_ID,
        user_id: PUBLIC_KEY,
        accessToken: PRIVATE_KEY,
      },
      {
        headers: {
          "Content-Type": "application/json", // uključujemo Content-Type zaglavlje
        },
      }
    );
    // obrada uspješnog odgovora
    res.status(200).json({
      message: "Email uspješno poslan!",
      data: response.data,
    });
  } catch (error) {
    // obrada greške
    console.error(
      "Greška prilikom slanja emaila: ",
      (error.response && error.response.data) || error.message
    );
    res.status(500).json({
      error: "Greška prilikom slanja emaila!",
      details: (error.response && error.response.data) || error.message,
    });
  }
});

```

Za kraj, **moramo proslijediti podatke o korisniku u tijelu POST zahtjeva**. Kako su podaci o korisniku pohranjeni u procesnim varijablama, moramo ih dohvatiti i koristiti u tijelu zahtjeva.

```

// express-email-server/index.js

app.post("/send-email", async (req, res) => {
  try {
    const { ime, prezime, email, username } = req.body; // dohvaćanje podataka o korisniku
    const response = await axios.post(
      "https://api.emailjs.com/api/v1.0/email/send",
      {
        service_id: SERVICE_ID,
        template_id: TEMPLATE_ID,
        user_id: PUBLIC_KEY,
        accessToken: PRIVATE_KEY,
        // podaci koji se koriste u predlošku e-mail poruke
      }
    );
    res.status(200).json({
      message: "Email uspješno poslan!",
      data: response.data,
    });
  } catch (error) {
    console.error("Greška prilikom slanja emaila: ", error.message);
    res.status(500).json({
      error: "Greška prilikom slanja emaila!",
      details: error.message,
    });
  }
});

```

```

    template_params: {
      ime: ime,
      prezime: prezime,
      email: email,
      username: username,
    },
  },
  {
    headers: {
      "Content-Type": "application/json",
    },
  }
);
res.status(200).json({
  message: "Email uspješno poslan!",
  data: response.data,
});
} catch (error) {
  console.error(
    "Greška prilikom slanja emaila: ",
    (error.response && error.response.data) || error.message
  );
res.status(500).json({
  error: "Greška prilikom slanja emaila!",
  details: (error.response && error.response.data) || error.message,
});
}
});

```

Testirajte ovu POST rutu korištenjem nekog HTTP klijenta (npr. [Postman](#), [Insomnia](#), [ThunderClient](#)). Primjer tijela zahtjeva:

```
{
  "ime": "Pero",
  "prezime": "Perić",
  "email": "pperic@gmail.com",
  "username": "pperic123"
}
```

Ako ste sve točno napravili, email bi se trebao poslati na vašu e-mail adresu, a kao rezultat dobiti ćete sljedeće tijelo odgovora i statusni kod 200:

```
{
  "message": "Email uspješno poslan!",
  "data": "OK"
}
```

*Primjer zaprimljenog e-maila:*

Camunda: Podaci o korisniku ✉ Pristigla pošta 🖨️ 🔗

 Moja Camunda <luka.blaskovic@unipu.hr>  
prima ja ▾

Pozdrav!

Šaljemo Vam podatke o korisniku:

**Ime:** Pero  
**Prezime:** Perić  
**Email:** pperic@gmail.com  
**Username:** pperic123

Lijepi pozdrav i ugodan dan,  
Vaša Camunda!

Email sent via [EmailJS.com](#)

✉ Odgovor ⟳ Prosljedi ✉

Primjer e-mail poruke koju smo poslali koristeći *Email.js* servis

To je to! Uspješno smo implementirali Express.js poslužitelj za slanje e-mail poruka putem *Email.js* servisa

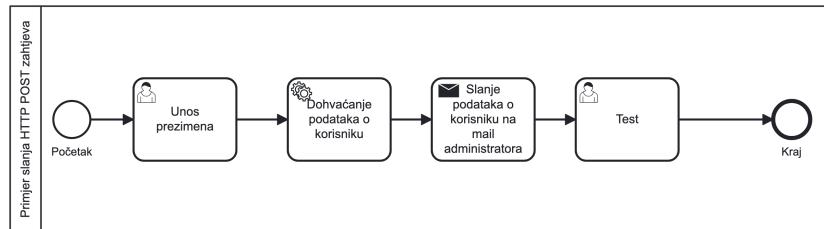


Sljedeći korak je integracija ovog poslužitelja s *Camunda Engineom*, preciznije `Connector` implementacija na `Send Task` elementu.

## 2.4 Definiranje Send Task aktivnosti

Postupak definiranja **Send Task** elementa je identičan kao i kod **Service Task** elementa. Koristeći Connector API implementirat ćemo na jednak način slanje POST zahtjeva na poslužitelj **express-email-server**.

Nadogradit ćemo proces "Primjer slanja HTTP POST zahtjeva" **Send Taskom** - "Slanje podataka o korisniku na mail administratora", gdje prepostavljamo da smo administrator mi, odnosno email koji smo definirali na Email.js servisu.



Procesna definicija s dodanim **Send Task** elementom

Odabiremo **Send Task** element te jednako kao i kod servisnog zadatka, pod **Implementation** odabiremo **Connector** te kao **Connector Id** unosimo **http-connector**.

Dalje, dodajemo sljedeće **Connector inputs**:

- **Local variable name** = `url`, **Assignment type** = `String or expression`, **Value** = `http://host.docker.internal:3000/send-email`
- **Local variable name** = `method`, **Assignment type** = `String or expression`, **Value** = `POST`
- **Local variable name** = `headers`, **Assignment type** = `Map`, **Map entries** (Key = `Content-Type`, Value = `application/json`)
- **Local variable name** = `payload`, **Assignment type** = `String or expression`, **Value** = `{"ime": "${ime}", "prezime": "${prezime}", "email": "${email}", "username": "${username}"}`

Pazite da se imena procesnih varijabli podudaraju s imenima varijabli koje referenciramo Expressionom u **payload** varijabli!

The screenshot shows the BPMN process definition and the configuration of the 'Slanje podataka o korisniku na mail administratora' task. The task is highlighted with a blue border. The configuration panel on the right shows the following settings:

- Connector inputs**:
  - Map
  - Map entries
    - Content-Type
      - Key: Content-Type
      - Value: application/json- payload**:
  - Local variable name: payload
  - Assignment type: String or expression
  - Value:

```
{  
    "ime": "${ime}",  
    "prezime": "${prezime}",  
    "email": "${email}",  
    "username": "${username}"  
}
```

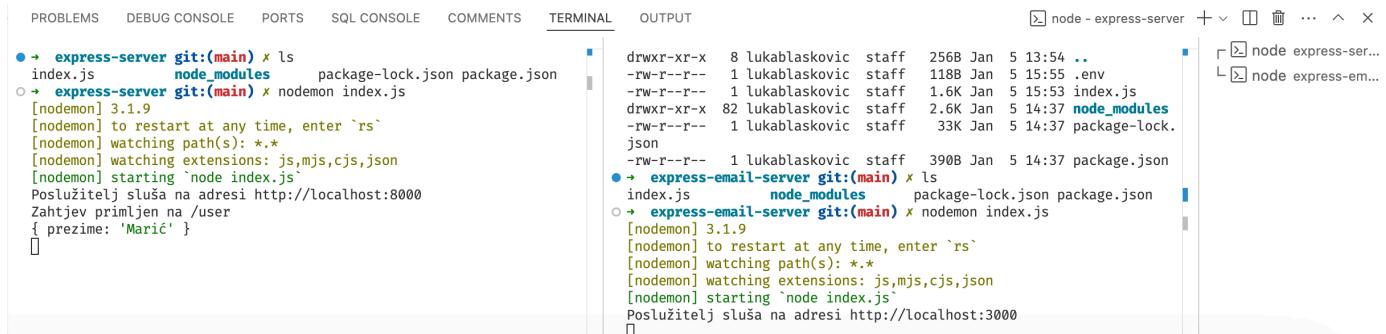
Dodavanje `Connector inputs` za `Send Task` element, primjer definiranja tijela zahtjeva i zaglavlja

Kao odgovor, dovoljno nam je samo pohraniti statusni kod odgovora u **izlaznu procesnu varijablu**

**Connectora** (`Connector outputs`):

- **Local variable name** = `emailStatus`, **Assignment type** = `String or expression`, **Value** =  
 `${statusCode}`

Prije nego testirate procesnu definiciju, provjerite da ste pokrenuli oba poslužitelja (`express-server` i `express-email-server`) te da su dostupni na odgovarajućim portovima. Ako koristite VS Code, oba poslužitelja možete pokrenuti u zasebnim terminalima, najpregleđnije je odvojiti ih u zasebne radne prozore (`Terminal -> Split Terminal`).



```
PROBLEMS DEBUG CONSOLE PORTS SQL CONSOLE COMMENTS TERMINAL OUTPUT node - express-server + v ... ^ x
● ➔ express-server git:(main) ✘ ls
index.js      node_modules      package-lock.json package.json
○ ➔ express-server git:(main) ✘ nodemon index.js
[nodemon] 3.1.9
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
Poslužitelj sluša na adresi http://localhost:8000
Zahtjev primljen na /user
{ prezime: 'Marić' }
[nodemon] 3.1.9
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
Poslužitelj sluša na adresi http://localhost:3000
node express-server... ↴
node express-em... ↴
```

Redeployajte procesnu definiciju i pokušajte ponovno pokrenuti procesnu instancu.

Ako ste sve ispravno definirali, nakon što unesete prezime u `User Task` elementu "Unos prezimena", dešava se sljedeće:

1. Poziva se servisni zadatak koji pronađe korisnika na temelju prezimena u servisu `express-server`
2. Dohvaćeni podaci se pohranjuju u odgovarajuće procesne varijable te se koristeći `Send Task` element šalju na poslužitelj `express-email-server` koji šalje e-mail poruku korisniku
3. Ako je poslužitelj ispravno implementiran, trebali biste primiti e-mail poruku na vašu e-mail adresu

Primjerice, unijeli smo prezime "Marić" i dohvatali podatke o korisniku Marku Mariću. Nakon toga, podaci o korisniku su poslati na našu e-mail adresu.

Camunda Cockpit   Processes   Decisions   Human Tasks   More ▾   Demo Der

Dashboard » Processes » connector\_POST : 58469790-cb78-11ef-945f-0242ac110002 : Runtime

**Information**   Filter

**Instance ID:** 58469790-cb78-11ef-945f-0242ac1...

**Business Key:** null

**Definition Version:** 1

**Definition ID:** connector\_POST:1:568ba11f-cb78-1...

**Definition Key:** connector\_POST

**Definition Name:** connector\_POST

**Tenant ID:** null

**Deployment ID:** 56711dfd-cb78-11ef-945f-0242ac11...

**Super Process Instance ID:** 56711dfd-cb78-11ef-945f-0242ac11...

**Primer stanja HTTP POST zadjeva**

```

graph LR
    Start((Početak)) --> Unos[Unos prezimena]
    Unos --> Dohvacanje[Dohvaćanje podataka o korisniku]
    Dohvacanje --> Slanje[Slanje podataka o korisniku na mail administratora]
    Slanje --> Test[Test]
    Test --> Kraj((Kraj))
  
```

**Variables**

Add criteria					
Name	Type	Value	Scope	Actions	
email	String	mmaric@gmail.com	connector_POST		
emailStatus	Integer	200	connector_POST		
ime	String	Marko	connector_POST		
odgovor	String	{"ime": "Marko", "prezime": "Marić", "use...}	connector_POST		
prezime	String	Marić	connector_POST		
statusni_kod	Integer	200	connector_POST		
username	String	marko.maric	connector_POST		

Primjer rezultata procesne instance u Cockpitu