

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) Uvod u procesno-orientirani razvoj poslovnih aplikacija

#6

UPP

Procesno orientirani razvoj aplikacija (*eng. process-oriented development*) predstavlja pristup razvoju softverskih rješenja koji u središte stavlja poslovne procese – njihovo modeliranje, implementaciju i upravljanje unutar organizacije. Umjesto tradicionalnog razvoja usmjerenog primarno na podatke ili funkcionalnosti, ovaj pristup koristi standardizirane notacije poput BPMN-a te oblikuje aplikaciju oko poslovnih aktivnosti, ciljeva i tijeka rada. Takav način razvoja donosi niz prednosti: bolju usklađenost softvera s poslovnim potrebama, veću fleksibilnost te mogućnost brze prilagodbe promjenama u poslovnom okruženju. Osim toga, procesno orientirani pristup pridonosi povećanju učinkovitosti i transparentnosti poslovnih procesa te poboljšava komunikaciju između poslovnih stručnjaka, menadžera i developera. U ovoj skripti dat ćemo uvod u procesno orientirani razvoj aplikacija koristeći Camunda 8 platformu i njezin skup alata za modeliranje, implementaciju i nadzor procesno-orientiranih aplikacija.

Posljednje ažurirano: 19.1.2026.

Sadržaj

- [Upravljanje poslovnim procesima \(UPP\)](#)
- [\(6\) Uvod u procesno-orientirani razvoj poslovnih aplikacija](#)
 - [Sadržaj](#)
- [1. Uvod u procesne aplikacije](#)
 - [1.1 Camunda 8 platforma](#)
- [2. Priprema radnog okruženja](#)
 - [2.1 Pokretanje Camunda 8 Self-Managed paketa](#)
- [3. Razvoj jednostavne procesne aplikacije u Camunda 8](#)
 - [3.1 Osnovne komponente Camunda 8 platforme](#)

- [3.2 Camunda Operate aplikacija](#)
- [3.3 Camunda Tasklist aplikacija](#)
- [3.4 XOR granaanje procesa na temelju procesnih varijabli](#)
- [3.5 Kako još pokrenuti procesne instance?](#)
- [3.6 Izmjena vrijednosti procesnih varijabli tijekom izvođenja procesa](#)
- [Zadaci za vježbu 6](#)

1. Uvod u procesne aplikacije

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

1. **osigurati** da se BPMN normom služe poslovni stručnjaci koji ne razvijaju aplikacije (ili nemaju tehničko/informatičko obrazovanje) 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 predočavanje procesa.
- Detaljno razrađen **grafički model** poslovnog procesa može se pretvoriti u izvršiv oblik i na temelju tога razviti potrebna softverska rješenja.
- Pogodan za **zajednički jezik za sporazumijevanje** između poslovnih stručnjaka, **analitičara procesa i softverskih inženjera**.

[Procesno-orientirani razvoj aplikacija](#) (*eng. process-oriented application development*) nastoji približiti dva opisana "svijeta": poslovni svijet, koji definira procese i aktivnosti koje treba obaviti da bi se isporučio proizvod ili usluga, te informatički svijet, koji razvija softverska rješenja za podršku tim procesima.

Procesna aplikacija (PA) (*eng. process application*) se temelji na tijeku rada, odnosno može se reći da je svaka PA procesno usmjerena (*eng. process-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 (podatkovno-usmjerene) aplikacije (non-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 (tijek rada) implicitno je sadržan u aplikaciji**, obično određen programiranim redoslijedom prikaza korisničkih poziva, odnosno poziva programskih sučelja.
- **Aktivnosti i procesi ne postoje kao aplikacijski entiteti** - njihova se stanja ne prate izravno niti su definirani unutar aplikacije
- **Arhitektura klasične aplikacije prilagođena je funkcionalnim CRUD potrebama**; odnosno stvaranju, čitanju, ažuriranju i brisanju podatkovnih zapisa.
- Pri izmjeni poslovnih procesa (npr. zbog promjene zakonske regulative), klasične aplikacije često treba **temeljito reprogramirati**, posebice komponente njihove [poslovne logike](#) (*eng. business logic layer*)

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

- **Funkcionalnosti proizlaze iz tijeka rada koji aplikacija treba podržavati.** Polazište razvoja procesne aplikacije čini **procesni model**, dodatno razrađen i prilagođen tako da bude izravno **izvršiv**.
- Tijekovi (sljedovi) rada u aplikaciji jasno su i eksplicitno definirani te su **neovisni o korisničkim i programskim sučeljima**.
- **Aktivnosti i procesi** modelirani su kao aplikacijski entiteti nad kojima se izravno upravlja, uključujući njihova stanja i ponašanje.
- Arhitektura procesne aplikacije oslanja se na događajno (*eng. event-driven*) i porukama vođeno (*eng. message-driven*) djelovanje te na upravljanje tijekom rada (*eng. workflow management*).
- Procesne aplikacije mogu obuhvaćati i korisničke aktivnosti (*eng. user tasks*) i automatizirane aktivnosti (*eng. service tasks*) unutar istog procesa.
- Omogućuju suradnju više organizacijskih jedinica unutar organizacije te ih povezuju u jedinstven, krajnjem korisniku ili kupcu **vrijednosno orijentiran proces**.
- Visoko su prilagodljive promjenama poslovnih procesa: izmjene se provode ažuriranjem definicija aktivnosti i/ili procesa, bez potrebe za ponovnim programiranjem (ili značajno manjim opsegom programiranja) aplikacijskih komponenti.

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

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

Napomena: U praksi, granica između funkcionalnosti klasične i procesne aplikacije nije uvijek crno-bijela. Postoji veliki broj aplikacija koje kombiniraju oba pristupa (*best of both worlds*)

Sada kada smo objasnili ove osnovne razlike između klasičnih i procesnih aplikacija, možemo pokušati definirati procesnu aplikaciju na sljedeći način:

➡ **Procesna aplikacija** je softverska aplikacija čije se ponašanje i funkcionalnosti primarno određuju izvršivim modelom poslovnog procesa, a ne isključivo strukturu podataka ili skupom izoliranih operacija nad tim podacima.

Spomenuta diferencijacija na primjeru klasične webshop aplikacije i procesne webshop aplikacije može se opisati na sljedeći način:

Klasična aplikacija

- zamišljamo u kontekstu gradiva kolegija [Programsko Inženjerstvo](#) ili [Web aplikacije](#).
- Funkcionalnosti implementiramo *low-level* programiranjem gdje razmišljamo o **CRUD operacijama nad bazom podataka** (CRUD-orientirano programiranje).

Primjer 1: "Korisnik se registrira i pregledava proizvode" → CRUD operacije nad tablicama `users` i `products`, razvoj korisničkog sučelja za prikaz podataka i poziv operacija → izmjena stanja u bazi podataka. CRUD Operacije: `CREATE user`, `READ products`.

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 → izmjena stanja u bazi podataka. CRUD Operacije: `CREATE cart item`, `READ cart items`, `CREATE order`, `UPDATE order status`.

Zapamtite: Kod klasičnih aplikacija operacije su neovisne o tijeku rada/izvođenja procesa.

Backend je [stateless](#) – ne prati se cijelokupan tijek rada korisnika na poslužiteljskoj strani, već se svaka operacija (čitaj: mrežni zahtjev) izvršava izolirano, neovisno o prethodnim ili budućim operacijama.

Ovo nije slučaj kod procesnih aplikacija, gdje je tijek rada (definirani *workflow*) ključan koncept.

Ključni takeaway: Aplikacija se organizira oko podataka i CRUD operacijama nad njima.

Procesna aplikacija

- zamišljamo u kontekstu gradiva kolegija [Upravljanje poslovnim procesima](#) ili [Upravljanje projektima](#).
- Funkcionalnosti implementiramo *high-level* programiranjem gdje razmišljamo o **tijeku rada i aktivnostima koje korisnik treba obaviti** (procesno-orientirano programiranje i [process thinking](#)).

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). Svaki korak može biti različitog tipa (korisnička aktivnost, automatizirana aktivnost, ručna aktivnost itd.) i može imati različite ishode (npr. "Registracija uspješna", "Proizvodi dohvaćeni").

Primjer 2: "Korisnik dodaje proizvode u košaricu i obavlja kupnju" → Procesna aplikacija definira radne korake koje korisnik treba obaviti, npr. "Dodavanje proizvoda u košaricu", "Pregled košarice", "Unos podataka za plaćanje", "Potvrda narudžbe". Svaki radni korak može biti različitog tipa i može imati različite ishode (npr. "Plaćanje uspješno", "Plaćanje neuspješno", "Kupnja izvršena", "Proizvodi predani dostavnoj službi", itd.).

Zapamtite: Procesno-orientirane aplikacije prate tijek rada (*workflow*) na poslužiteljskoj strani kroz jasno definirane radne korake (aktivnosti) i **prate stanje procesa**.

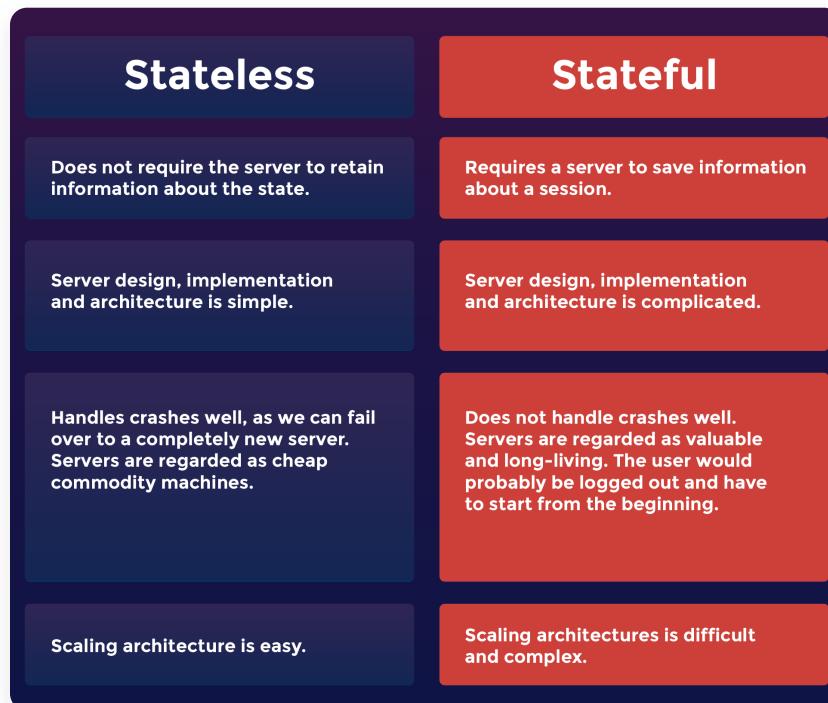
Backend je [stateful](#) – prati se cijelokupan tijek rada korisnika

Ovo nije slučaj kod klasičnih aplikacija, gdje se operacije izvršavaju izolirano, neovisno o prethodnim ili budućim operacijama - a korisnik se na neki način "snalazi sam" koristeći aplikaciju bez jasno definiranog tijeka rada.

Ključni takeaway: Aplikacija se organizira oko procesa i aktivnosti koje korisnik treba obaviti unutar definiranog tijeka rada.

Razvoj *stateful* poslovne aplikacije tehnički je zahtjevnije od razvoja *stateless* aplikacije. Glavni izazov leži u upravljanju stanjem aplikacije tijekom vremena, što uključuje praćenje napretka korisnika kroz različite faze poslovnog procesa, rukovanje prekidima i nastavcima rada te osiguravanje konzistentnosti podataka unutar dinamičnog okruženja. To zahtijeva sofisticiranje arhitekture, često uključujući korištenje mehanizme za upravljanje transakcijama i složene implementacije za obradu događaja/okidača. Osim toga, razvoj *stateful* aplikacija može povećati složenost testiranja i održavanja, jer je potrebno osigurati da se stanjem pravilno upravlja u svim mogućim scenarijima korištenja, kako ne bi došlo do gubitka podataka ili nekonzistentnosti u aplikaciji.

Sljedeća ilustracija navodi osnovne razlike između ove dvije paradigme, i naglašava zašto je *stateful* procesno-orientirani razvoj aplikacija tehnički zahtjevniji:



Slika 1. Razlike između *stateless* i *stateful* arhitekture poslužiteljske strane aplikacije.

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

Do sada smo na vježbama naučili kako ispravno modelirati poslovne procese pomoću BPMN notacije te kako definirati donošenje odluka korištenjem DMN notacije. Pritom smo se bavili i **poslovnim modeliranjem** (kako procese sagledati i oblikovati iz poslovne perspektive), ali i **tehničkom odnosno semantičkom stranom BPMN-a**, odnosno kako odabrati odgovarajuće elemente za modeliranje poslovnog procesa u kontekstu procesno orientiranog razvoja aplikacija (korisnički zadaci, servisni zadaci, ispravni događaji, ispravne skretnice, što su značke/*tokeni*, sekvenčalni i informacijski sljedovi rada i sl.).

Na kolegijima Programsko inženjerstvo i Web aplikacije, naučili ste razvijati web aplikacije koristeći prvenstveno klasični *stateless* pristup razvoju poslužiteljske strane (to ne znači da niste razmišljali u terminima procesa, već da ste to radili na način svojstven klasičnim aplikacijama).

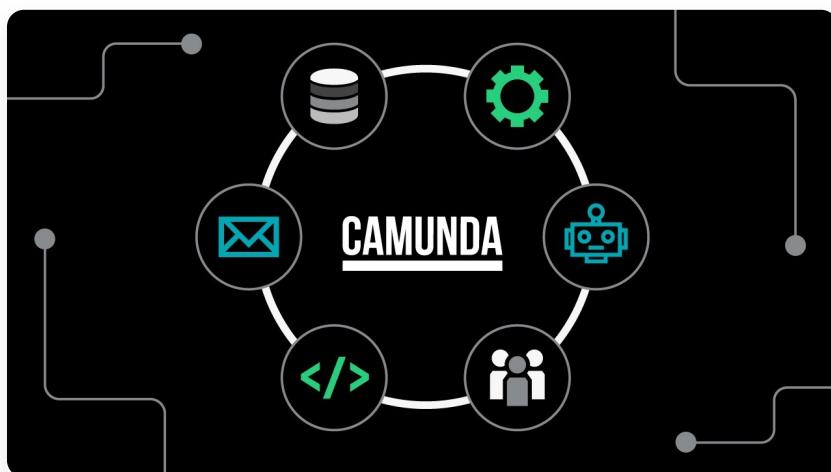
U ovoj i sljedećoj skripti, nastojat ćemo zagrebat površinu procesno-orientiranog *stateful* razvoja poslovnih aplikacija.

Kako je sam razvoj složeniji, nećemo iz nule razvijati procesne aplikacije, već ćemo koristiti **Camunda 8 platformu** koja nam omogućuje **razvoj** (eng. *development*), **izvršavanje** (eng. *execution*) i **upravljanje** (eng. *management*) procesno-orientiranim aplikacijama.

1.1 Camunda 8 platforma

Camunda 8 je moderna platforma koja služi orkestraciji, upravljanju i automatizaciji poslovnih procesa i odluka. Temelji se na otvorenim standardima poput BPMN-a i DMN-a, a omogućuje organizacijama da **modeliraju** (eng. *process modelling*), **izvršavaju** (eng. *process execution*) i **nadziru** (eng. *process monitoring*) svoje poslovne procese na učinkovit način.

Mi smo do sada koristili Camunda Modeler alat za modeliranje poslovnih procesa (BPMN) i odluka (DMN). Camunda 8 platforma, međutim, nudi mnogo više od samog modeliranja.



Slika 2. Camunda 8 platforma sastoji se od više komponenti koje omogućuju razvoj, izvršavanje i nadzor procesno-orientiranih aplikacija.

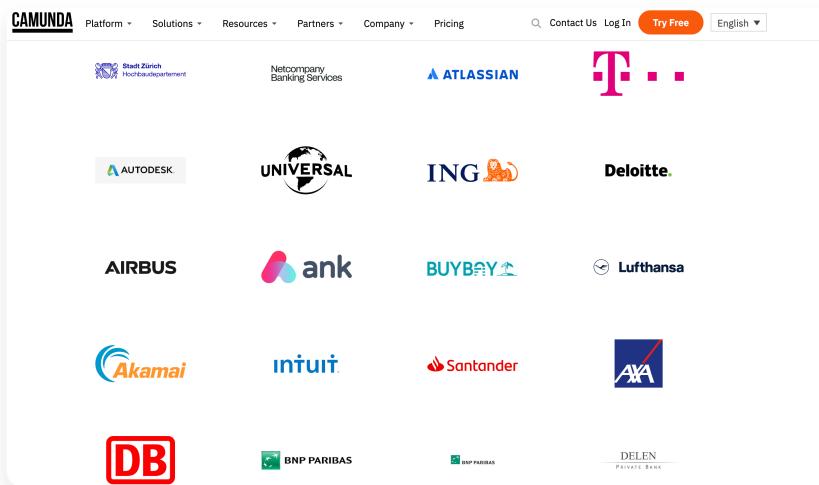
Camunda (Camunda Services GmbH) je njemačka softverska tvrtka osnovana 2008. godine, koja je započela svoje poslovanje kao BPM konzultant, a od 2013. razvija vlastitu *open-source* platformu za upravljanje poslovnim procesima - Camunda BPM. Platforma se temelji na otvorenim standardima (BPMN, DMN) i oblikovana je iskustvima iz ranijih projekata poput [Activiti](#) i [jBPM](#). Godine 2022.

Camunda je predstavila Camunda 8, *cloud-native* platformu s naglaskom na skalabilnost, performanse i orkestraciju procesa. Danas Camundu koriste brojne organizacije diljem svijeta, a zbog otvorenih standarda, dostupnih edukacijskih materijala i aktivne zajednice značajna je i u akademskom kontekstu.

Za motivaciju, u nastavku su navedene neke od većih svjetskih tvrtki koje prepoznaju vrijednost procesno-orientiranog pristupa:

- [Goldman Sachs](#) - vodeća američka investicijska banka i globalni financijski konglomerat, poznat po snažnom oslanjanju na standardizirane, automatizirane i strogo kontrolirane poslovne procese. 15 njihovih odjela koristi Camundu i njihov procesni *engine* koji obrađuje gotovo 6 milijuna aktivnosti tjedno.
 - Pročitajte više ovdje: <https://camunda.com/about/customers/goldman-sachs/>
- [T-Mobile](#) - jedan od najvećih telekomunikacijskih operatera u Europi, koristi Camundu za upravljanje složenim poslovnim procesima unutar svoje organizacije, uključujući korisničku podršku, upravljanje mrežom i interne operacije.

- Pročitajte više ovdje: <https://camunda.com/case-study/deutsche-telekom/>
- Panasonic - globalni tehnološki div koji koristi Camundu za optimizaciju i automatizaciju svojih poslovnih procesa, uključujući proizvodnju, logistiku i upravljanje opskrbnim lancem.
 - Pročitajte više ovdje: <https://camunda.com/case-study/panasonic-europe/>
- Zalando - jedna od najvećih online modnih platformi u Europi, koristi Camundu za upravljanje svojim internim procesima, uključujući obradu narudžbi, logistiku i korisničku podršku.
 - Pročitajte više ovdje: <https://camunda.com/case-study/zalando/>
- Atlassian - poznata tehnološka tvrtka koja razvija alate za suradnju i upravljanje projektima, koristi Camundu za optimizaciju svojih poslovnih procesa, uključujući razvoj softvera, korisničku podršku i interne operacije.
 - Pročitajte više ovdje: <https://camunda.com/case-study/atlassian/>
- Airbus - jedan od najvećih svjetskih proizvođača zrakoplova, koristi Camundu za upravljanje složenim poslovnim procesima unutar svoje organizacije, uključujući proizvodnju, logistiku i upravljanje opskrbnim lancem.
 - Pročitajte više ovdje: <https://camunda.com/case-study/airbus/>
- Lufthansa - vodeća njemačka zrakoplovna kompanija, koristi Camundu za optimizaciju i automatizaciju svojih poslovnih procesa, uključujući upravljanje letovima, korisničku podršku i interne operacije.
 - Pročitajte više ovdje: <https://camunda.com/case-study/lufthansa/>
- A1 - jedan od jačih telekomunikacijskih operatera u regiji - koristi Camundu za upravljanje složenim poslovnim procesima unutar svoje organizacije, uključujući korisničku podršku, upravljanje mrežom i interne operacije.
 - Pročitajte više ovdje: <https://camunda.com/case-study/a1-telekom-austria-group/>



Slika 3. Neke od većih svjetskih tvrtki koje koriste Camundu za upravljanje svojim poslovnim procesima. Više informacija dostupno je na: <https://camunda.com/about/customers/>

Ovo nije plaćena reklama za Camundu! Cilj je istaknuti značaj i sve veću zastupljenost procesno-orientiranog pristupa razvoju softvera u industrijskoj praksi. Camunda se u tom kontekstu navodi isključivo kao jedan od alata koji omogućuje implementaciju takvog pristupa - a s kojim mi već imamo iskustva kroz rad na vježbama, ali i broje [besplatne edukacijske materijale](#) koje Camunda nudi, kao i besplatnu verziju ove platforme za učenje, eksperimentiranje i razvoj manjih procesnih aplikacija.

Za vas, kao buduće informatičke stručnjake, **ključno je razumjeti temeljne koncepte procesno-orientiranog razvoja**, neovisno o konkretnom alatu ili platformi. Važno je kritički sagledati zašto velike organizacije primjenjuju ovaj pristup, na koji način on doprinosi poboljšanju poslovnih procesa i učinkovitosti, te koje su njegove prednosti i ograničenja u odnosu na tradicionalne, podatkovno ili funkcionalno orijentirane pristupe razvoju aplikacija.

Koga zanima više, možete pročitati izvrsne studije slučaja na Camundinoj web stranici: <https://camunda.com/case-studies>

Tema je složena i zahtjeva puno samostalnog istraživanja i učenja. Ipak, cilj ove skripte nije iscrpno obraditi cijelokupno područje, već zagrebati površinu te pružiti temeljne koncepte i smjernice koje mogu poslužiti kao polazište za daljnje produbljivanje znanja onima koje zanima ovaj pristup razvoju softvera. Već ste naučili kako modelirati poslovne procese i to je odličan (i **nužan**) prvi korak u razumijevanju procesno-orientiranog pristupa.

Također, sva stečena znanja iz modeliranja mogu se učinkovito primjenjivati i izvan konteksta izvršavanja aplikacija ("offline"), primjerice u svrhe poslovnog modeliranja, analize i optimizacije procesa, njihova sustavnog dokumentiranja, kao i jasne prezentacije procesa različitim dionicima unutar organizacije.

U sljedećim poglavljima, upoznat ćemo se s osnovnim komponentama Camunda 8 platforme, njenim razvojnim okruženjem te kako koristiti te njezine alate za razvoj, upravljanje i nadzor poslovnih procesa.

2. Priprema radnog okruženja

Camunda 8 platformu možemo koristiti na dva načina:

1. Korištenjem [Camunda Cloud usluge](#) (SaaS rješenje u oblaku)

Ovaj pristup podrazumijeva izradu korisničkog računa i rad putem web-sučelja u oblaku. Riječ je o *fully managed* rješenju, pri čemu se Camunda brine o cijelokupnoj infrastrukturi, skalabilnosti, sigurnosti i održavanju platforme. Iako je dostupno 30-dnevno besplatno probno razdoblje za orkestraciju procesa, nakon njegova isteka potrebno je unijeti podatke o plaćanju. **Zbog toga se ovaj pristup neće koristiti u sklopu kolegija.**

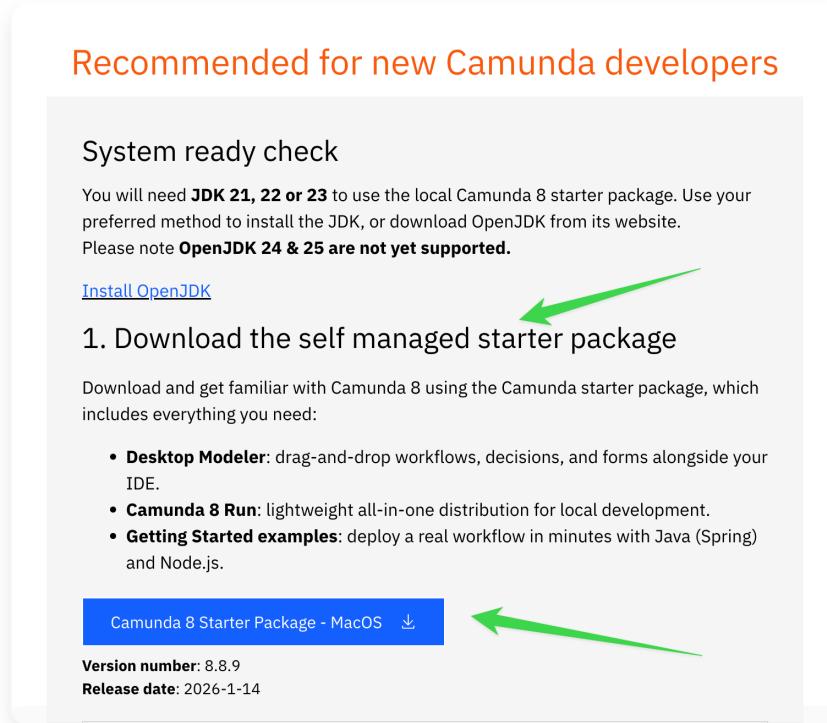
2. Korištenjem [Camunda Self-Managed rješenja](#) (on-premises / self-hosted)

Ovaj pristup zahtjeva preuzimanje i pokretanje Camunda platforme na vlastitom računalu ili poslužitelju, uz samostalno upravljanje i održavanje sustava. Iako je znatno zahtjevniji u kontekstu postavljanja i održavanja proizvodnjskog okruženja, za lokalni razvoj i testiranje procesnih aplikacija izuzetno je praktičan i jednostavan za korištenje. Dodatna prednost ovog pristupa jest nepostojanje vremenskog ograničenja korištenja platforme. **Mi ćemo koristiti ovaj pristup.**

Napomena: Termine [on-premises](#), [self-managed](#) i [fully managed](#) ćete često susresti u kontekstu korištenja gotovih softverskih rješenja. Ukratko, *on-premises* i *self-managed* označavaju rješenja koja korisnik samostalno instalira, konfigurira i održava na vlastitoj infrastrukturi (osobno računalo ili udaljeni poslužitelj). S druge strane, *fully managed* rješenja su ona koja održava i upravlja pružatelj usluge (npr. Camunda Cloud), čime se korisniku olakšava korištenje softvera bez brige o tehničkim detaljima infrastrukture i održavanja. [SaaS](#) i [PaaS](#) rješenja su najčešći oblik *fully managed* usluga.

Prvi korak je preuzeti Camunda 8 Self-Managed starter paket sa službene stranice: <https://developers.camunda.com/install-camunda-8/>

Pustite da se paket preuzima (veličina oko 1.2 GB), a za to vrijeme možete provjeriti imate li ispunjene preduvjete (JDK 21, 22 ili 23).



Slika 4. Preuzimanje Camunda 8 Self-Managed starter paketa.

Potrebno je imati instaliran Java Development Kit (JDK) verzije **21, 22 ili 23**. Ovo je vrlo važan preduvjet, novijom JDK verzijom Self-hosted paket neće raditi ispravno.

Možete preuzeti jednu od verzija JDK-a sa sljedeće poveznice: <https://jdk.java.net/archive/>

- potrebno je odabrati JDK 21, 22 ili 23 te ispravan operacijski sustav (Windows, Linux, MacOS)

Preporučena opcija je koristiti *package manager* za instalaciju JDK-a (ali i drugih potrebnih alata) na vašem OS-u:

- **Windows:** [Chocolatey](#), [Scoop](#) ili [Winget](#)
- **Linux:** [apt](#), [dnf](#), [yum](#), [pacman](#)
- **macOS:** [Homebrew](#), [MacPorts](#)

Windows OS

Ako ne koristite package manager na Windowsu, možete preuzeti Microsoft Build of OpenJDK sa sljedeće poveznice: <https://learn.microsoft.com/en-gb/java/openjdk/older-releases#openjdk-25>

- Preuzmite installer (`msi` ili `exe` datoteku) za JDK 21: `microsoft-jdk-21.0.8-windows-x64.msi`
- Pokrenite instalacijski program i slijedite upute za instalaciju JDK-a na vaše računalo.
- **Obavezno odaberite opcije:** `Modify PATH variable` i `Set or override JAVA_HOME variable`.

Nakon uspješne instalacije, možete provjeriti je li JDK ispravno instaliran otvaranjem naredbenog retka i upisivanjem sljedeće naredbe:

```
→ java --version
```

Očekivani ispis:

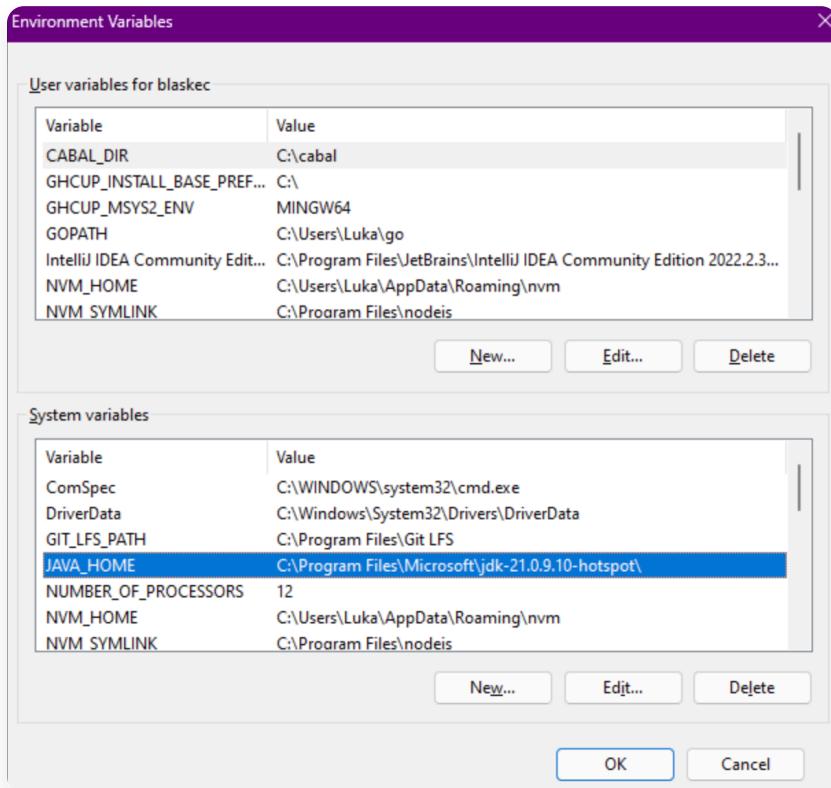
```
openjdk 21.0.9 2025-10-21
```

Napomena: Ako već imate instaliranu noviju verziju JDK-a, ovisno o načinu njegove instalacije, proces deinstalacije može varirati. Ako se koristili *installer*, na Windowsu pokušajte otvoriti Upravljačku ploču i pronaći odgovarajuću JDK verziju - obrišite ju od тамо. Ako ste ručno ekstraktirali JDK iz arhive, jednostavno izbrišite mapu u koju ste ga ekstraktirali. Nakon toga, ponovite instalacijski postupak za JDK 21. Ako koristite macOS ili Linux te neki od *package managera*, koristite isti alat za deinstalaciju postojeće verzije JDK-a prije instalacije JDK 21.

Ako ne vidite informacije o verziji Java, vjerojatno se varijabla okruženja `JAVA_HOME` nije ispravno postavila tijekom instalacije. Pokušajte restartati terminal, pokrenuti ponovo *installer* ili ručno postaviti `JAVA_HOME` varijablu okruženja - upute u nastavku.

Na Windows OS-u, **ako preuzimate JDK ručno**, potrebno je ekstraktirati preuzetu arhivu (npr. `jdk-21_windows-x64_bin.zip`) u željenu mapu, npr. `C:\Program Files\Java\jdk-21`.

- Nakon toga, potrebno je postaviti `JAVA_HOME` varijablu okruženja koja pokazuje na instaliranu JDK mapu.
- Otvorite Start i upišite "Environment Variables" ili "Advanced System Settings"
- Pod "System Variables", **trebali biste vidjeti varijablu `JAVA_HOME`** i postavljenu apsolutnu putanju do JDK mape, npr. `C:\Program Files\Java\jdk-21`
- Ako ne vidite, odaberite "New..." i unesite:
 - Variable name: `JAVA_HOME`
 - Variable value: apsolutna putanja do JDK mape, npr. `C:\Program Files\Java\jdk-21`
- **Također, otvorite varijablu `Path`** pod "System Variables", odaberite "Edit..." i provjerite postoji li unos `%JAVA_HOME%\bin`. Ako ne postoji, dodajte novi unos s tom vrijednošću.



Slika 5. Postavljanje `JAVA_HOME` varijable okruženja na Windows OS-u.

Otvorite novu instancu terminala i ponovno pokrenite naredbu:

```
→ java --version
```

2.1 Pokretanje Camunda 8 Self-Managed paketa

Jednom kad ste preuzeli Camunda 8 Self-Managed paket i instalirali ispravni JDK, spremni ste za pokretanje Camunda platforme na vašem računalu.

Otvorite novu instancu terminala i navigirajte do direktorija u koji ste ekstraktirali Camunda 8 Self-Managed paket, npr.

```
→ bash  
→ cd <putanja_do_ekstraktiranog_paketa>  
# Primjer:  
→ cd C:\Users\Korisnik\Downloads\camunda8-getting-started-bundle-8.8.9-darwin-aarch64
```

Napomena: Naziv ekstrahiranog direktorija može varirati ovisno o verziji Camunda paketa i vašem OS-u.

Provjerite sadržaj direktorija:

```
→ ls
```

Očekujete sljedeću strukturu direktorija:

```
.  
├── ai-agent-example  
├── c8run-8.8.9  
├── Camunda Modeler.app  
├── camunda-8-get-started  
├── camunda-start.sh  
└── camunda-stop.sh
```

Datoteke `camunda-start.sh` i `camunda-stop.sh` su bash skripte (`.sh`) koje služe za pokretanje i zaustavljanje Camunda 8 platforme.

Bash skriptu možete pokrenuti na način da u terminalu upišete relativnu ili absolutnu putanju do nje, npr.

```
→ ./camunda-start.sh
```

Ili naredbom `bash`:

```
→ bash camunda-start.sh
```

Nakon pokretanja skripte, trebali biste vidjeti ispis u terminalu koji pokazuje napredak pokretanja različitih komponenti Camunda 8 platforme. Na početku ispise će vam se prikazati detalji o verziji JDK-a koju koristite - provjerite da je to JDK 21, 22 ili 23.

Kod uspješnog pokretanja, vidjet ćete sljedeće *logove* u terminalu:

```
8:59PM DBG Failed to read PID from file. This is expected for the first run.  
8:59PM INF No pid for Elasticsearch  
8:59PM INF Started process: /Users/lukablaskovic/Desktop/camunda8-getting-started-bundle-  
8.8.9-darwin-aarch64/c8run-8.8.9/elasticsearch.process pid=98822  
8:59PM INF Waiting for service to start name=Elasticsearch retries_left=12  
8:59PM INF Waiting for service to start name=Elasticsearch retries_left=11  
8:59PM INF Started successfully name=Elasticsearch  
8:59PM DBG Failed to read PID from file. This is expected for the first run.  
8:59PM INF No pid for Connectors  
8:59PM INF Started process: /Users/lukablaskovic/Desktop/camunda8-getting-started-bundle-  
8.8.9-darwin-aarch64/c8run-8.8.9/connectors.process pid=98912  
8:59PM DBG Failed to read PID from file. This is expected for the first run.  
8:59PM INF No pid for Camunda  
8:59PM INF Started process: /Users/lukablaskovic/Desktop/camunda8-getting-started-bundle-  
8.8.9-darwin-aarch64/c8run-8.8.9/camunda.process pid=98913  
8:59PM INF Waiting for service to start name=Camunda retries_left=24  
8:59PM INF Waiting for service to start name=Camunda retries_left=23  
8:59PM INF has successfully been started. name=Camunda  
-----  
Access each component at the following urls with these default credentials:  
- username: demo  
- password: demo
```

```
Operate:          http://localhost:8080/operate
Tasklist:         http://localhost:8080/tasklist
Identity:         http://localhost:8080/identity

Orchestration Cluster API: http://localhost:8080/v2/
Inbound Connectors API:   http://localhost:8086/
Zeebe API (gRPC):        http://localhost:26500/

Camunda metrics endpoint: http://localhost:9600/actuator/prometheus

When using the Desktop Modeler, Authentication may be set to None.

Refer to https://docs.camunda.io/docs/guides/getting-started-java-spring/ for help getting
started with Camunda

-----
8:59PM INF All processes are running and healthy, exiting script...
```

Ovo znači da je Camunda 8 platforma uspješno pokrenuta i spremna za korištenje.

Primjetit ćete da Camunda **ne zauzima terminalski prozor nakon uspješnog pokretanja** - to je zato što se svi procesi pokreću u pozadini. Terminalski prozor možete slobodno zatvoriti ili koristiti za druge naredbe.

Kako biste zaustavili Camunda 8 platformu, otvorite novu instancu terminala (ili koristite postojeću), navigirajte do direktorija i pokrenite skriptu za zaustavljanje:

```
→ bash camunda-stop.sh

# ili

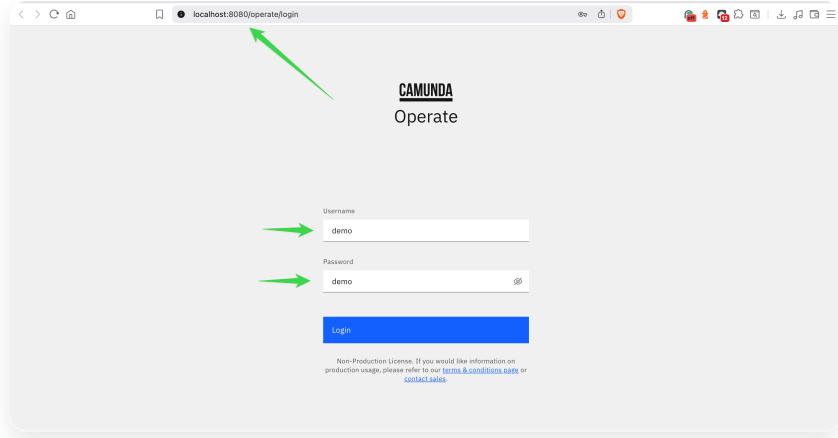
→ ./camunda-stop.sh
```

Jednom kada ste uspješno pokrenuli Camunda 8 platformu, možete pristupiti svim web aplikacijama koje su sastavni dio platforme putem web preglednika.

Primjerice, aplikaciji **Operate** možete pristupiti na sljedećoj adresi: <http://localhost:8080/operate>

Navigirat će vas na stranicu za prijavu. Prisjetite se, kako koristimo *Self-managed* varijantu, naša aplikacija se ne spaja na udaljeni Camunda Cloud, već na lokalno pokrenutu instancu Camunda platforme. Iz tog razloga, koristimo **defaultne** vjerodajnice za prijavu:

- **Korisničko ime:** `demo`
- **Lozinka:** `demo`



Slika 6. Prijava u Camunda 8 Operate aplikaciju koristeći zadane vjerodajnice. Aplikacija je dostupna na adresi <http://localhost:8080/operate>

3. Razvoj jednostavne procesne aplikacije u Camunda 8

Sada kada smo uspješno pokrenuli Camunda 8 platformu, spremni smo za razvoj naše prve procesne aplikacije koristeći Camunda alate. Prije svega, ukratko ćemo se upoznati s osnovnim komponentama Camunda 8 platforme koje ćemo koristiti za vrijeme razvoja.



Slika 7. Zeebe je *cloud-native workflow engine* koji pokreće Camunda 8 platformu i predstavlja srce Camunda 8 platforme.

3.1 Osnovne komponente Camunda 8 platforme

1. **Camunda Modeler:** Alat za modeliranje poslovnih procesa kroz BPMN i DMN notacije. Ne treba puno objašnjavati jer smo ga već koristili na svim vježbama do sada. Camunda Modeler dolazi kao samostalna aplikacija unutar Self-Managed paketa koji smo preuzeli - ali ju naravno možemo preuzeti i zasebno.
 - Modeler ćemo i dalje koristiti kao glavni alat za modeliranje poslovnih procesa i odluka. Zamislite ga kao naš VSCode za poslovne aplikacije!
 - Ako slučajno još nemate Camunda Modeler instaliran, preuzmите ga sa sljedeće poveznice: <https://camunda.com/download/modeler/>
2. **Zeebe Workflow Engine:** Temeljna komponenta Camunda 8 platforme koja omogućuje **izvršavanje BPMN procesa** te evaluaciju DMN odluka (*eng. BPMN and DMN execution*). Zeebe je *cloud-native workflow engine* namijenjen **orkestraciji i egzekuciji poslovnih procesa** modeliranih u BPMN notaciji. Temelji se na *event-driven* arhitekturi i distribuiranom *append-only* logu za **pohranu procesnog stanja**, čime se omogućuje horizontalno skaliranje, visoka dostupnost i otpornost sustava. Zeebe se integrira s vanjskim sustavima putem *workers* i *connectors* mehanizama, dok se operativni nadzor i analitika

procesa ostvaruju kroz zasebne komponente (aplikacije) Camunda 8 platforme.

- Zeebe je srce Camunda 8 platforme i omogućuje nam da naše BPMN modele "oživimo" kroz izvršavanje definiranih procesa.
- Zeebe je već pokrenut kao dio Self-Managed paketa koji smo instalirali, tako da ga ne moramo dodatno konfigurirati.
- Dokumentacija: <https://docs.camunda.io/docs/components/zeebe/zeebe-overview/>

3. **Camunda Operate:** Web aplikacija za **nadzor i upravljanje izvršenjem poslovnih procesa**.

Omogućuje pregled aktivnih i završenih procesa, praćenje njihovog stanja, analizu performansi te dijagnostiku problema tijekom izvršavanja.

- Operate ćemo koristiti za praćenje i upravljanje našim procesima nakon što ih implementiramo i pokrenemo.
- Dokumentacija: <https://camunda.com/platform/operate/>

4. **Camunda Tasklist:** Web aplikacija koja **omogućuje korisnicima upravljanje i izvršavanje korisničkih zadataka** unutar poslovnih procesa.

Korisnici mogu pregledavati svoje zadatke, dodjeljivati ih sebi ili drugima, te ih označavati kao dovršene.

- Tasklist ćemo koristiti za interakciju s korisničkim zadacima unutar naših procesa (npr. ispunjavanje obrazaca, učitavanje podataka, donošenje odluka itd.)
- Dokumentacija: <https://camunda.com/platform/tasklist/>

5. **Camunda Identity:** Komponenta za **upravljanje korisnicima, grupama i njihovim ulogama** unutar

Camunda platforme. Omogućuje definiranje pristupnih prava i autorizaciju korisnika za različite aplikacije i funkcionalnosti.

- Identity ćemo koristiti za upravljanje korisnicima koji će sudjelovati u našim procesima (npr. administratori, krajnji korisnici, menadžeri itd.)
- Dokumentacija: <https://docs.camunda.io/docs/self-managed/components/management-identity/overview>

6. **Camunda Optimize:** Alat za analitiku i izvještavanje o poslovnim procesima.

Omogućuje dubinsku analizu podataka o izvršenju procesa, identifikaciju uskih grla, praćenje ključnih pokazatelja uspješnosti (KPI) te generiranje prilagođenih izvještaja. Optimize nam pomaže u kontinuiranom poboljšanju poslovnih procesa kroz uvid u njihovu učinkovitost i performanse.

- Optimize nije uključen u Self-Managed paket, ali ga možete naknadno instalirati ako želite dodatno analizirati svoje procese.
- Dokumentacija: <https://camunda.com/platform/optimize/>

7. **Camunda Connectors:** Komponenta koja omogućuje **integraciju Camunda platforme s vanjskim sustavima i uslugama** putem unaprijed definiranih *connectora*. *Connectori* olakšavaju komunikaciju između poslovnih procesa i različitih API-ja, baza podataka, servisa u oblaku i drugih sustava.
- Connectors ćemo koristiti za povezivanje naših procesa s vanjskim sustavima (npr. slanje e-pošte, dohvaćanje podataka iz baze, slanje zahtjeva na HTTP poslužitelje itd.)
 - Dokumentacija: https://docs.camunda.io/docs/components/connectors/introduction-to-connector_s/

Pregled osnovnih komponenti Camunda 8 platforme i njihovih funkcija.

Komponenta	Funkcija	Primjer korištenja
Camunda Modeler	Alat za modeliranje poslovnih procesa (BPMN) i odluka (DMN)	<i>Modeliranje procesa narudžbe proizvoda</i>
Zeebe Workflow Engine	Izvršavanje BPMN procesa i evaluacija DMN odluka	<i>Pokretanje procesa narudžbe i praćenje njegovog stanja</i>
Camunda Operate	Nadzor i upravljanje izvršavanjem poslovnih procesa	<i>Pregled aktivnih instanci narudžbi i rješavanje incidenata - npr. deadlock</i>
Camunda Tasklist	Upravljanje i izvršavanje korisničkih zadataka unutar procesa	<i>Dodjela i dovršavanje zadataka korisničke podrške</i>
Camunda Identity	Upravljanje korisnicima, grupama i njihovim ulogama	<i>Definiranje pristupnih prava za administratore i korisnike</i>
Camunda Optimize	Analitika i izvještavanje o poslovnim procesima	<i>Analiza performansi procesa narudžbi i generiranje izvještaja</i>
Camunda Connectors	Integracija s vanjskim sustavima putem unaprijed definiranih konektora	<i>Slanje e-pošte nakon dovršetka narudžbe ili dohvati podataka iz vanjskih sustava</i>

Ukratko, za samu egzekuciju procesa koristi se **Zeebe Workflow Engine**, dok su ostale komponente (Operate, Tasklist, Identity, Optimize, Connectors) tu da nam pomognu u razvoju, upravljanju i nadzoru naših procesnih aplikacija. Samim time, Camunda 8 platforma pruža sveobuhvatno rješenje za razvoj i implementaciju procesno-orientiranih aplikacija.

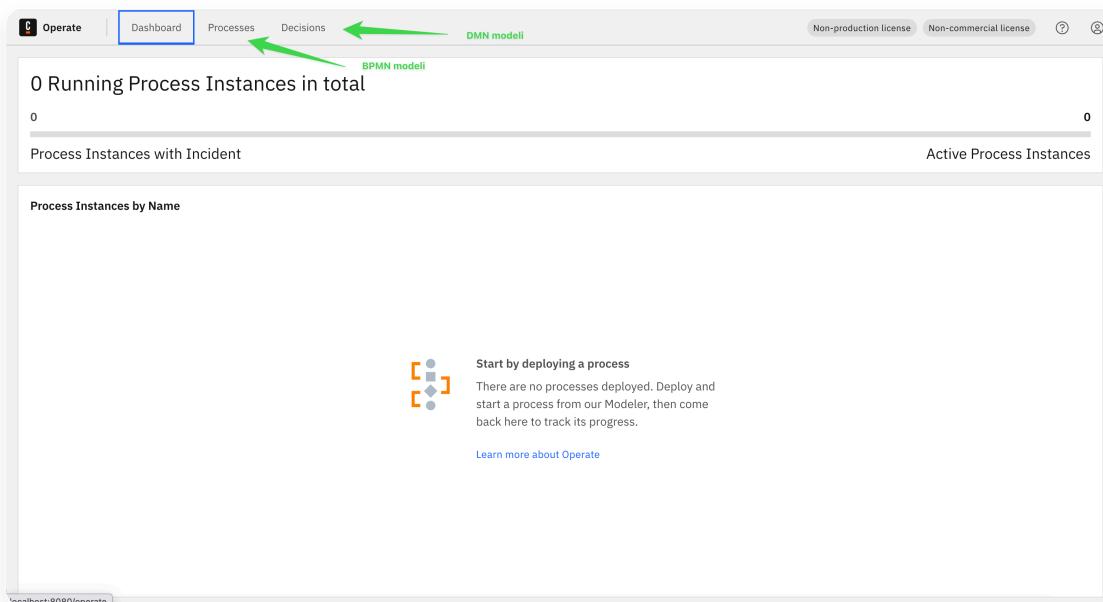
Ipak, **u praksi se često koriste custom-made rješenja aplikacija koje se integriraju s Camunda platformom putem njenih API-ja** - navedeno nećemo raditi u sklopu ovog kolegija budući da se radi o velikom poslu.

Primjerice, umjesto korištenja Camunda Tasklist aplikacije za unos korisničkih zadataka, vrlo je praktično razviti vlastitu web aplikaciju (npr. koristeći React, Angular, Vue.js ili neki drugi frontend framework) koja će se povezivati s Camunda platformom putem njenih REST API-ja. Na taj način možemo prilagoditi korisničko sučelje specifičnim potrebama naše organizacije ili projekta i opet iskoristiti sve prednosti koje Camunda platforma nudi u smislu upravljanja procesima i njihovog izvršavanja.

3.2 Camunda Operate aplikacija

Camunda Operate je web aplikacija koja nam omogućuje **nadzor i upravljanje poslovnim procesima** koji se izvršavaju unutar Camunda 8 platforme. Kroz Operate možemo pratiti stanje naših procesa, analizirati njihove performanse te dijagnosticirati eventualne probleme tijekom izvršavanja.

Početni zaslon aplikacije, nakon prijave, prikazuje pregled svih dostupnih **definicija procesa i njihovih pokrenutih instanci**.



Slika 8. Početni zaslon Camunda Operate aplikacije prikazuje pregled definicija procesa i njihovih instanci. Pod "Processes" vidimo sve definirane procese i procesne instance, a pod "Decisions" sve definirane odluke i njihove instance.

Definicija procesa (*eng. Process Definition*): predstavlja model poslovног procesa izrađen u BPMN notaciji koji je spremан за egzekuciju unutar Camunda platforme. Definicija procesa sadrži sve elemente procesa, uključujući zadatke, događaje, skretnice i tokove, te definira kako će se proces izvoditi.

Instanca procesa (*eng. Process Instance*): predstavlja pojedinačno pokretanje definiranog poslovног procesa. Svaka instance procesa ima svoje jedinstveno stanje, varijable i povijest izvršavanja, što omogućuje praćenje napretka i upravljanje tijekom njenog trajanja.

Ako otvorite karticu "Processes", vidjet ćete da **trenutno nema niti jedne definirane procesne instance** jer još nismo implementirali niti pokrenuli nijedan proces.

Kako bismo dodali novu definiciju procesa, potrebno je prvo modelirati proces u Camunda Modeler alatu, a zatim ga **implementirati** (*deploy*) na Camunda platformu.

Idemo prvo ovo pokazati na gotovim primjerima koje imamo u Self-Managed paketu.

1. Otvorite Camunda Modeler
2. U Camunda Modeleru, odaberite `File > Open File...` i navigirajte do direktorija `camunda-8-get-started/bpmn/` unutar ekstrahiranog Self-Managed paketa.
3. Odaberite datoteku `diagram_1.bpmn` i otvorite ju.

Vidjet ćete jednostavan BPMN model s 5 elemenata:

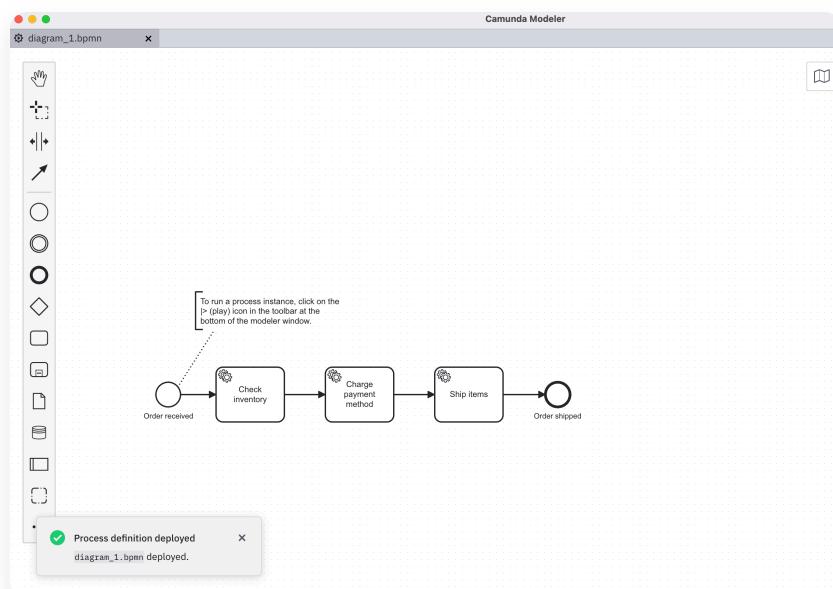
- Početni događaj - "Order received"
- Servisni zadatak - "Check inventory"
- Servisni zadatak - "Charge payment method"
- Servisni zadatak - "Ship items"
- Završni događaj - "Order shipped"

Kako bismo *deployali* procesnu aplikaciju, jednostavno ćemo kliknuti na **ikonu rakete** o donjem lijevom kutu Camunda Modelera, odabratи `Camunda 8 Self-Managed`, a kao cluster endpoint upisati Zeebe API Cluster endpoint URL (gRPC protokol) (pogledati *logove* u terminalu - **sekcija 2.1**). Autentifikaciju ostavite na `None`.

- Zadana adresa Zeebe Clustera je: `http://localhost:26500`

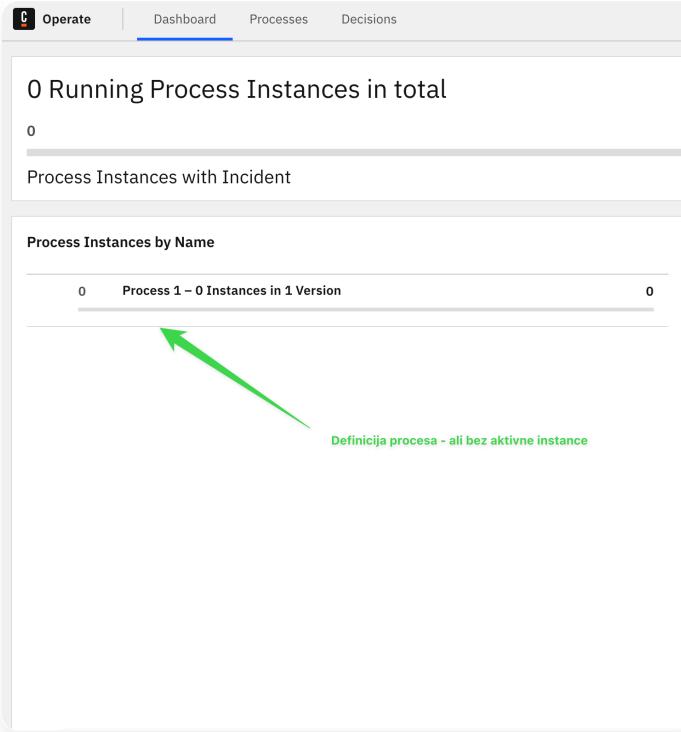
Odaberite `Deploy BPMN`.

Ako ste sve ispravno napravili, trebali biste vidjeti poruku o uspješnom *deploymentu* procesa.



Slika 9. Uspješan *deployment* BPMN procesa iz Camunda Modelera na Camunda 8 Self-Managed platformu.

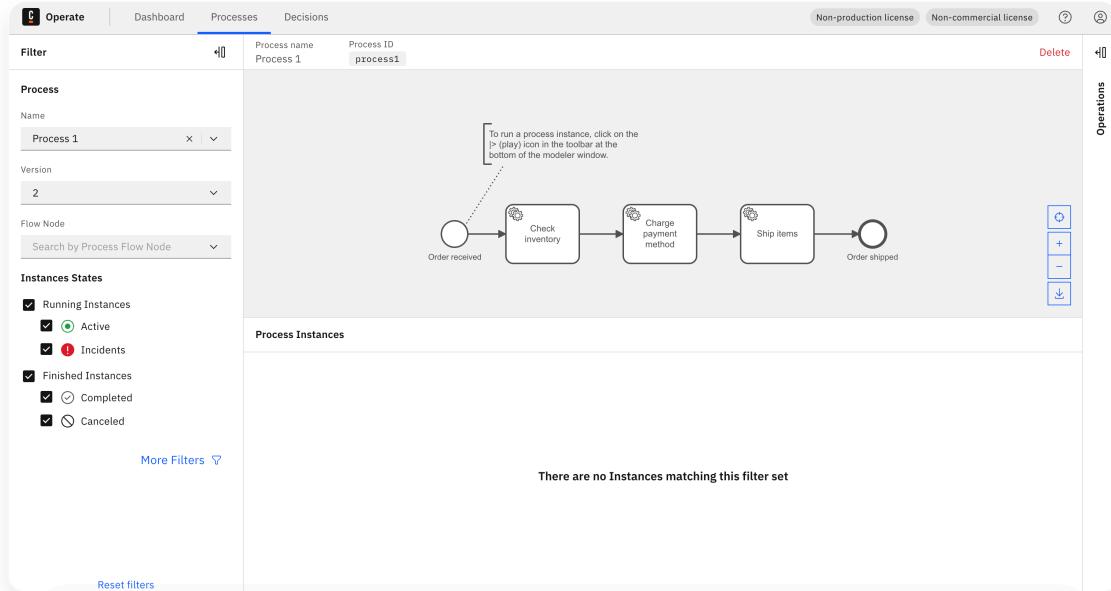
Vratite se u Operate aplikaciju i osvježite stranicu. **Sada biste trebali vidjeti novu definiciju procesa.**



Slika 10. Nova definicija procesa "Order Process" vidljiva u Camunda Operate aplikaciji nakon uspješnog *deploymenta* iz Camunda Modelera.

Ako odaberete definiciju procesa, otvorit će se "Processes" kartica koja prikazuje više detalja. Ovdje možete vidjeti sve pokrenute instance procesa, njihovo stanje, varijable i povijest izvršavanja.

Također, možete pretraživati i filtrirati instance procesa prema različitim kriterijima, kao što su status (*Completed/Canceled/Active/Incident*), naziv, verzija definicije i sl.



Slika 11. Detalji definicije procesa "Order Process" u Camunda Operate aplikaciji, uključujući pregled pokrenutih instanci procesa (trenutno ih nema).

Pokretanje procesne instance

Kako bismo pokrenuli procesnu instancu, vratit ćemo se u Camunda Modeler i otvoriti BPMN model `diagram_1.bpmn` ako već nije otvoren.

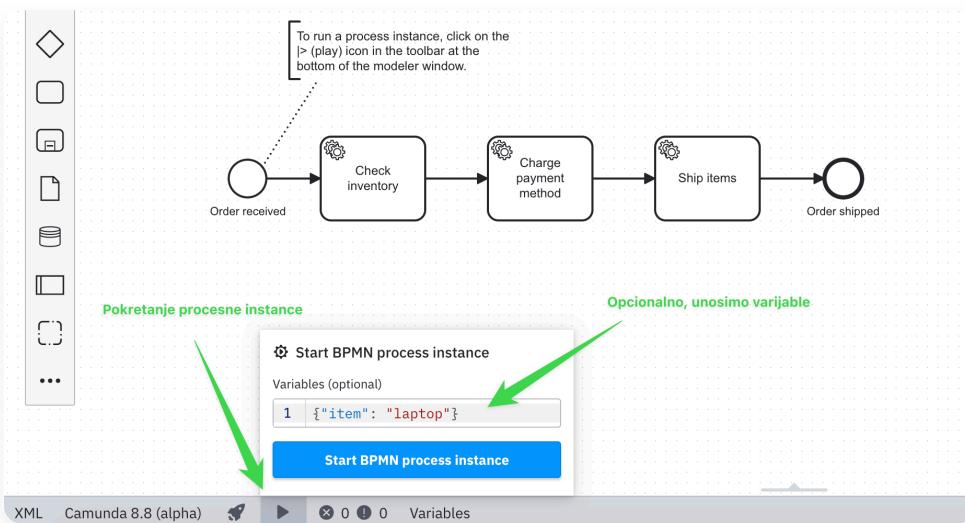
Odaberite ikonu **Run** u donjem lijevom kutu Camunda Modelera (pored ikone rakete) kako biste pokrenuli novu instancu procesa.

Opcionalno, možemo unijeti početne varijable procesa u JSON formatu, npr.

```
{  
    "item": "laptop"  
}
```

Odaberite **Start BPMN process instance**.

Ako je sve prošlo u redu, trebali biste vidjeti poruku o uspješnom pokretanju nove procesne instance za odabranu procesnu definiciju.



Slika 12. Pokretanje nove procesne instance iz Camunda Modelera s početnom varijablom.

Vratite se u Operate aplikaciju i osvježite karticu "Processes". Sada biste trebali vidjeti novu procesnu instancu u popisu.

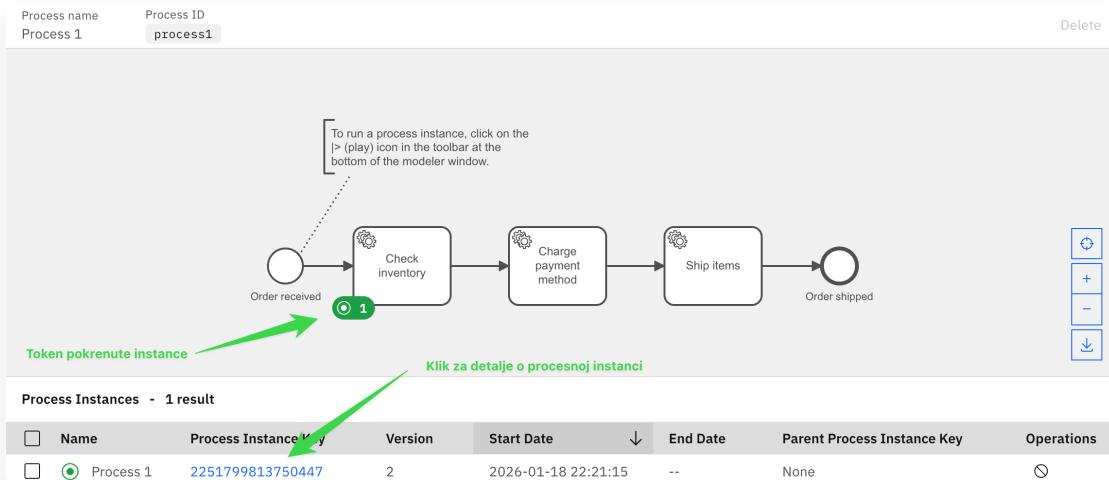
Svaka procesna instanca ima sljedeće atributе:

- **Name**: naziv procesne definicije iz koje je instanca pokrenuta (npr. "Order Process", "Process 1", itd.)
- **Process Instance Key**: jedinstveni identifikator instance procesa unutar Camunda platforme.
- **Version**: verzija procesne definicije iz koje je instanca pokrenuta.
- **Start Date**: datum i vrijeme kada je instanca procesa pokrenuta.
- **End Date**: datum i vrijeme kada je instanca procesa završena (ako je dovršena).
- **Parent Process Instance Key**: jedinstveni identifikator nadređene instance procesa ako je trenutna instanca pokrenuta kao potproces.
- **Operations**: dostupne operacije za upravljanje instancom procesa (npr. *Cancel*, *Retry Incident*, *View Details* itd.)

U prikazu ćete vidjeti da **trenutno postoji 1 aktivna procesna instanca koja se nalazi na aktivnosti "Check inventory"**.

Zelena oznaka s kružićem predstavlja **značku** tj. **token** ili **trenutnu poziciju unutar procesa gdje se instanca nalazi**.

Pored kružića se nalazi brojka **1**, što označava da postoji jedna procesna instanca čiji je token trenutno na toj aktivnosti. Kada bi pokrenuli još jedan proces iste definicije, vidjeli bismo brojku **2** na istoj aktivnosti, što bi značilo da su dvije procesne instance trenutno na toj točki u procesu.

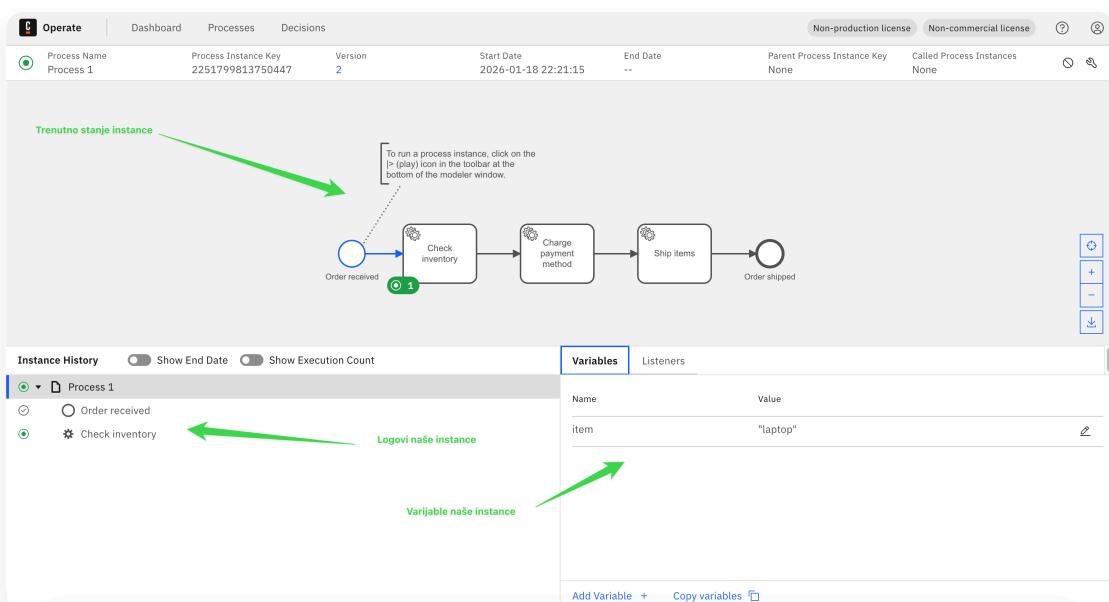


Slika 13. Pregled procesnih instanci u Camunda Operate aplikaciji s prikazom tokena na aktivnosti "Check inventory".

Kako bismo pogledali detalje o procesnoj instanci, kliknite na **Process Instance Key** vrijednost.

Uočite nekoliko kartica s detaljima:

- **Instance History:** prikazuje povijest izvršavanja instance procesa, uključujući sve aktivnosti, događaje i zadatke koji su se dogodili tijekom njenog trajanja. Ovo su logovi koji bilježe svaki korak (napredak) unutar procesa.
- **Variables:** prikazuje procesne varijable povezane s instancom procesa, uključujući njihove nazive i vrijednosti. Ovdje možemo direktno i uređivati varijable (brisati, dodavati ili mijenjati vrijednosti) kako bismo utjecali na daljnje izvršavanje procesa.
- **Diagram View:** vizualni prikaz instance našeg procesa - gdje se trenutno nalazi token, koje su aktivnosti dovršene, a koje nisu ni započele. Odabirom na pojedini BPMN element, možemo vidjeti dodatne informacije o tom elementu (npr. vrijeme početka i završetka, varijable povezane s njim, eventualne incidente itd.)



Slika 14. Detalji procesne instance u Camunda Operate aplikaciji, uključujući pregled varijabli i povijesti izvršavanja.

Procesna varijabla predstavlja podatak koji je povezan s određenom instancom procesa i koristi se za pohranu informacija tijekom njenog trajanja. Procesnim varijablama se prema zadanim postavkama može pristupati svugdje unutar procesa, što omogućuje dinamično upravljanje tokom procesa na temelju njihovih vrijednosti.

Ako se prisjetite, na prošlim vježbama (skripta 5), smo modelirali proces odobravanja kredita koristeći BPMN i DMN notacije. Podaci s kojima korisnik započinje proces (npr. željeni iznos kredita, željeni rok otplate, osobni podaci, itd.) su podaci koje bismo pohranili kao **početne vrijednosti procesnih varijabli** (npr. pokretanjem procesa iz Modelera). Tijekom izvršavanja procesa, različiti zadaci i odluke bi koristili te varijable za donošenje odluka (npr. izračun kreditnog rizika), ali također bi mogli ažurirati ili dodavati nove varijable (npr. rezultat provjere kreditnog rizika, odobreni iznos kredita, itd.) koje bi se koristile u dalnjem toku procesa.

Zašto su procesne varijable važne?

- One omogućuju dinamičnost i prilagodljivost procesa, jer se tok procesa može mijenjati na temelju njihovih vrijednosti.
- Omogućuju pohranu i prijenos podataka između različitih aktivnosti unutar procesa (prema zadanim postavkama, procesne varijablu su **globalnog dosega**)

Pitanje: Zašto se ovaj proces zaustavio na aktivnosti "Check inventory" i nije nastavio dalje prema "Charge payment method"?

► Spoiler alert! Odgovor na pitanje

Job workeri već dolaze implementirani unutar Self-Managed paketa koji smo preuzeли. Možemo odabrati `Java + Spring` implementaciju ili Node.js implementaciju. Naravno, odabrat ćemo Node.js implementaciju jer nam je to bliže.

Unutar terminala, navigirajte do direktorija `camunda-8-get-started/nodejs`.

Možete otvoriti implementaciju u VSCode-u kako biste bolje vidjeli o čemu se radi:

```
→ code .
# ili
→ code ./camunda-8-get-started/nodejs
```

Struktura ovog Node.js *worker-a* izgleda ovako:

```
.  
├── camunda-test-config.json  
├── jest.config.js  
├── node_modules  
├── package-lock.json  
├── package.json  
├── source  
└── test  
    └── tsconfig.json
```

Osim test datoteka i konfiguracijskih datoteka za [TypeScript](#) i [Jest](#), najvažniji direktorij je `source` koji sadrži implementaciju našeg *job workera*.

Unutar `source/index.ts` datoteke, uočite da se koristi [@camunda8/sdk](#) biblioteka za povezivanje s Camunda platformom i obradu zadataka, dok su pojedinačne implementacije zadataka implementirane unutar `source/workers.ts` datoteke.

```
import { Camunda8 } from "@camunda8/sdk";  
import { startWorkers } from "./workers";  
  
const client = new Camunda8({  
  CAMUNDA_AUTH_STRATEGY: "NONE",  
  ZEEBE_REST_ADDRESS: "http://localhost:8080",  
}).getCamundaRestClient();  
  
/* We inject the client to allow the workers to be tested independently using  
@camunda8/process-test. */  
startWorkers(client);  
  
console.log("Job workers started. Waiting for jobs...\n");
```

Implementacija sadrži 3 *job workera* koji obrađuju zadatke:

- `inventoryWorker` - obrađuje "Check inventory" servisni zadatak
- `paymentChargeWorker` - obrađuje "Charge payment method" servisni zadatak
- `shippingWorker` - obrađuje "Ship items" servisni zadatak

Primjer implementacije jednog od *worker-a*:

```
const inventoryWorker = client.createJobWorker<Variables, Variables>({  
  // The first type parameter is the input variables, the second is the output variables  
  type: "check-inventory",  
  timeout: 10000, // Timeout for the job worker to complete the job before it is available  
  for another worker poll  
  maxJobsToActivate: 5, // Maximum number of jobs to process concurrently  
  worker: "check-inventory-worker",  
  jobHandler: async (job, log) => {  
    log.info("Processing check-inventory job:", job.jobKey);  
    const item = job.variables.item ?? "default-item";  
    log.info(`Checking inventory for item: ${item}`);
```

```

    // Simulate checking inventory
    await new Promise((resolve) => setTimeout(resolve, 2000));
    log.info(`check-inventory job completed: ${job.jobKey}`);
    return job.complete({ item: `${item} allocated` });
},
});

```

Ove implementaciju su vrlo jednostavne i predstavljaju samo simulaciju stvarne poslovne aktivnosti (npr. provjera zaliha, naplata, otprema). U stvarnom svijetu, ove aktivnosti bi uključivale pozive na vanjske sustave, baze podataka, API-je i sl.

Ako otvorite `package.json` datoteku, vidjet ćete da su definirane potrebne ovisnosti i skripte za pokretanje *worker-a*.

Ove *job workere* možemo pokrenuti kao i svaki drugi Node.js projekt:

1. Otvorite terminal i navigirajte do direktorija `camunda-8-get-started/nodejs` i instalirajte ovisnosti sa `npm install`
2. Otvorite `package.json` i provjerite kako se pokreće aplikacija - skripta za pokretanje je definirana kao `start`, dakle: `npm start`

```

→ npm install # instalacija ovisnosti
→ npm start # pokretanje job workera (typescript aplikacije)

```

Nakon pokretanja *worker-a*, trebali biste vidjeti ispis u terminalu koji pokazuje da su* *job workeri** pokrenuti i čekaju na zadatke.

Međutim, zadaci su već definirani (naša procesna instanca je već na "Check inventory" aktivnosti), stoga će se odmah početi obrađivati.

Trebali biste vidjeti ispis u terminalu koji pokazuje napredak obrade zadataka:

```

> nodejs@1.0.0 start
> npx ts-node ./source/index.ts

Job workers started. Waiting for jobs...

info: Processing check-inventory job: {"timestamp": "2026-01-18T21:56:46.721Z"}
info: Checking inventory for item: laptop {"timestamp": "2026-01-18T21:56:46.721Z"}
info: check-inventory job completed: 2251799813750453 {"timestamp": "2026-01-18T21:56:48.723Z"}
info: Processing charge-payment job: {"timestamp": "2026-01-18T21:56:48.772Z"}
info: charge-payment job completed: 2251799813755650 {"timestamp": "2026-01-18T21:56:50.774Z"}
info: Processing ship-items job: {"timestamp": "2026-01-18T21:56:50.805Z"}
info: ship-items job completed: 2251799813755656 {"timestamp": "2026-01-18T21:56:52.807Z"}

```

Uočite da su se svi zadaci obradili jedan za drugim (Processing "naziv workera" → "job completed").

Također, u logovima možemo uočiti vrijednost procesne varijable koju smo proslijedili - u našem slučaju `item: laptop`.

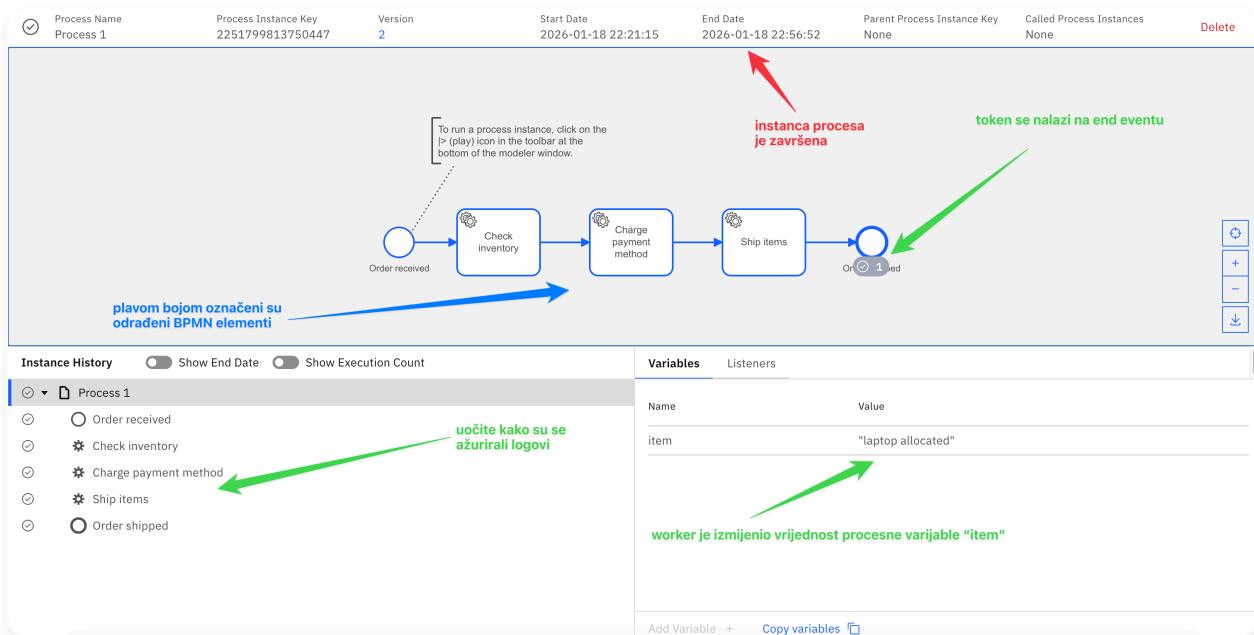
Nakon što su svi zadaci obrađeni, vratite se u Operate aplikaciju i osvježite karticu "Processes". Sada biste trebali vidjeti da je procesna instanca završena (status `Completed`), a token se nalazi na završnom događaju "Order shipped".

Osim toga, vidjet ćete **završene BPMN elemente označene plavom bojom**.

Procesni token će biti prikazan na završnom događaju, što označava da je instanca procesa uspješno dovršena.

Uočite i kako su se ažurirali logovi u kartici "Instance History" - sada možemo vidjeti sve aktivnosti koje su se dogodile tijekom izvršavanja instance, uključujući vrijeme početka i završetka svake aktivnosti ako kliknemo na pojedini BPMN element.

Za kraj, uočite i kako se promjenila vrijednost procesne varijable `item` - sada ima vrijednost `laptop allocated`, što je rezultat obrade prve zadatke `inventoryWorker`.



Slika 15. Završena procesna instanca u Camunda Operate aplikaciji s tokenom na završnom događaju "Order shipped".

To je to! Uspješno smo pokrenuli i završili našu prvu instancu poslovnog procesa koristeći Camunda 8 platformu i Node.js job *workere*.

Za vježbu možete *deployati* procesnu definiciju i pokrenuti više instanci procesa pod nazivom `diagram_2.bpmn` u direktoriju `camunda-8-get-started/bpmn/`. Ovaj proces je sličan prethodnom, ali uključuje *interrupting boundary error* evenete koji omogućuju rukovanje greškama tijekom izvršavanja procesa.

3.3 Camunda Tasklist aplikacija

Camunda Tasklist je web aplikacija koja omogućuje korisnicima upravljanje i izvršavanje korisničkih zadataka (*eng. user task*) unutar poslovnih procesa. Kroz Tasklist, korisnici mogu pregledavati svoje zadatke, dodjeljivati ih sebi ili drugima, te ih označavati kao dovršene.

Kako naši primjeri procesa ne sadrže korisničke zadatke, pravo je vrijeme za pokazati kako izraditi vlastitu procesnu aplikaciju.

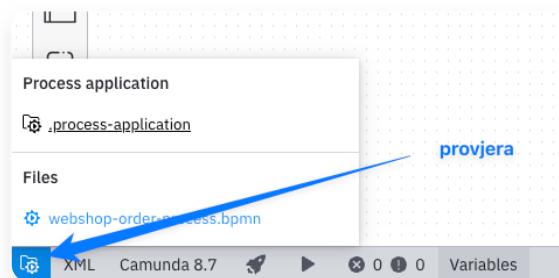
Izrada nove procesne aplikacije

Otvorite Camunda Modeler i odaberite `File > New Process Application`. Stvorite novi direktorij naziva: `webshop-order-process`. Nakon toga, izradite novi Camunda 8 BPMN dijagram unutar tog direktorija i spremite ga kao `webshop-order-process.bpmn`.

Dobit ćete sljedeću strukturu direktorija:

```
webshop-order-process
└── .process-application # skrivena datoteka koja označava da je ovo Camunda procesna
  aplikacija
    └── webshop-order-process.bpmn
```

Otvorite `webshop-order-process.bpmn` datoteku, u donjem lijevom kutu odaberite ikonu plave konfiguracijske datoteke i provjerite nalazi li se vaša BPMN datoteka unutar procesne aplikacije.



Slika 16. Provjera da je BPMN datoteka unutar Camunda procesne aplikacije.

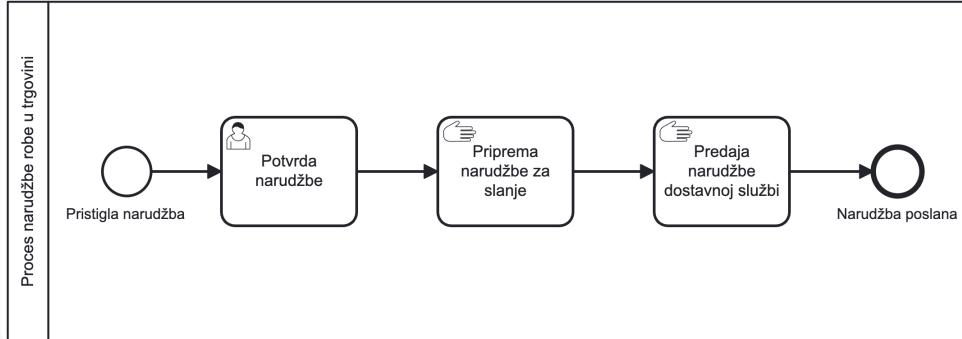
Složene procesne aplikacije vrlo često se sastoje od više BPMN i DMN datoteka koje su međusobno povezane. Stoga je važno da sve datoteke budu unutar istog direktorija procesne aplikacije.

Primjerice: možete imati 2 BPMN datoteke (glavni proces i ugniježđeni potproces) te 1 DMN datoteku (odluka) unutar iste procesne aplikacije.

Krenimo jednostavno - izradit ćemo jednostavni BPMN model "Proces narudžbe robe u trgovini" koji modelira obradu pristigle narudžbe u nekoj trgovini.

Dodajte sljedeće BPMN elemente u model:

- **Početni događaj:** "Pristigla narudžba"
- **Korisnički zadatak:** "Potvrda narudžbe"
- **Ručni zadatak:** "Priprema narudžbe za slanje"
- **Ručni zadatak:** "Predaja narudžbe dostavnoj službi"
- **Završni događaj:** "Narudžba poslana"

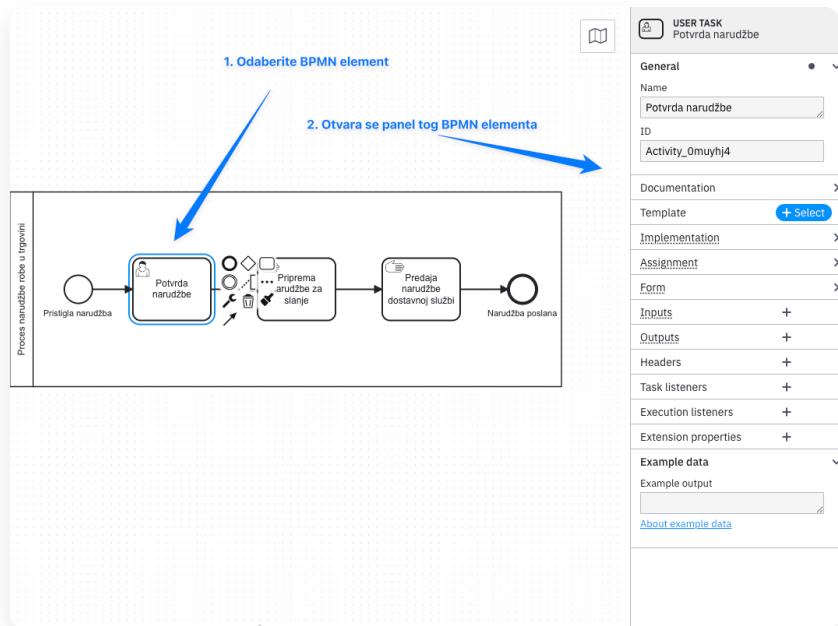


Slika 17. Početni jednostavni BPMN model "Proces narudžbe u trgovini" izrađen u Camunda Modeleru (webshop-order-process.bpmn).

Properties Panel

Sada ćemo implementirati korisnički zadatak "Potvrda narudžbe" kako bismo omogućili interakciju s korisnikom putem Camunda Tasklist aplikacije.

Odaberite element korisničkog zadatka i otvorite *Properties Panel*. Ako se panel ne prikazuje, odaberite `Window > Toggle Properties Panel` ili ga otvorite kraticom `Ctrl + P`.



Slika 18. Otvaranje "Properties Panel" u Camunda Modeleru za konfiguraciju pojedinog BPMN elemenata.

Properties Panel vam omogućuje konfiguraciju različitih svojstava odabranog BPMN elementa, uključujući njegov naziv, tip, ponašanje i druge attribute.

Svaki BPMN element ima **generalna svojstva**:

- `Name`: naziv elementa (labela) koju smo već unijeli na platnu modela.
- `ID`: jedinstveni identifikator elementa unutar BPMN modela (automatski generiran, ali se može mijenjati).
- `Documentation`: opis elementa koji može sadržavati dodatne informacije o njegovoj svrsi ili ponašanju. Ovo polje je poželjno ispuniti samo kod razvoja procesnih aplikacija.

Pod **Templates** možemo odabrat predložak za gotovi *connector* element. Ovdje možete pronaći razne cloud servise, uključujući AWS, Azure, GitHub Connector, Google Cloud Platform, OpenAI connector, RabbitMQ, Slack, Twilio, generalni RPA connector i sl. Ove predloške možete samostalno proučiti - **nećemo ih koristiti u sklopu ovog kolegija.**

Mi ćemo odabrat karticu **Implementation** koja je dostupna za **korisničke zadatke** i **business rule** zadatke.

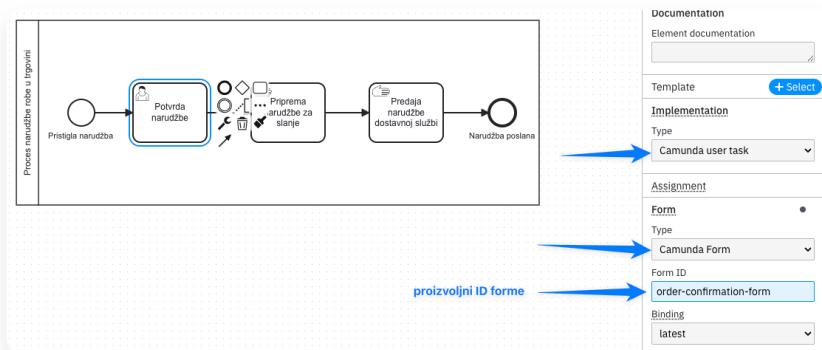
- **Camunda user task:** omogućuje konfiguraciju korisničkog zadatka unutar Camunda platforme.
- **Job worker:** omogućuje konfiguraciju servisnog zadatka koji će obrađivati vanjski job worker (npr. neki Node.js ili Java program).

Odabrat ćemo **Camunda user task** kako bismo konfigurirali naš korisnički zadatak "Potvrda narudžbe". Dalje, možemo odabrat vrstu forme koja će se koristiti za unos podataka od strane korisnika.

Možemo odabrat između:

- **Camunda Forms:** ugrađeni oblik koji dolazi s Camunda platformom (koristeći Tasklist aplikaciju za prikaz forme i Camunda Forms za izradu forme)
- **External Form Reference:** omogućuje da se forma ispunji izvan Camunda okruženja (npr. naša vlastita frontend aplikacija koja se povezuje s Camunda platformom putem API-ja)

Odabrat ćemo **Camunda Forms** kako bismo iskoristili ugrađeni oblik. Definirajte **Form ID:** `order-confirmation-form`.

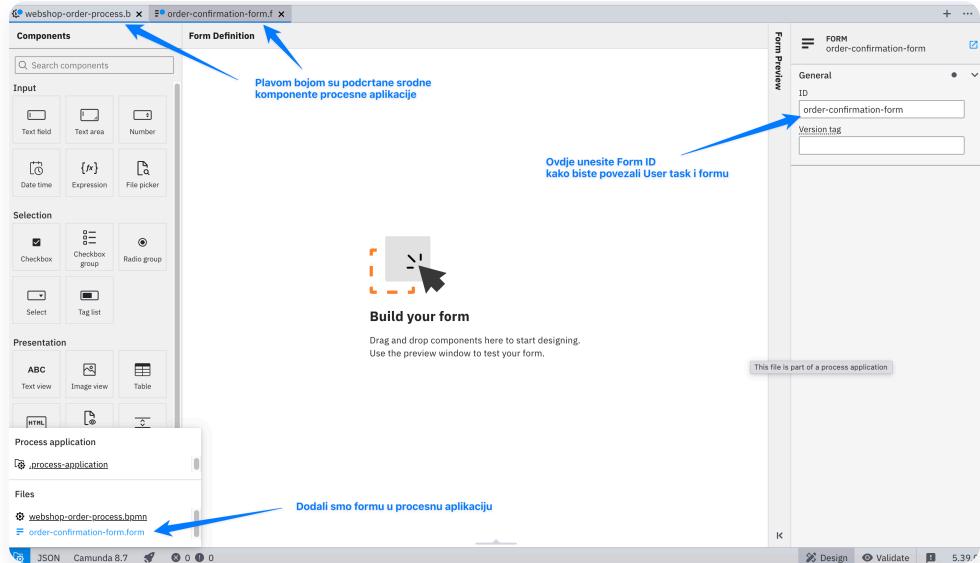


Slika 19. Konfiguracija korisničkog zadatka "Potvrda narudžbe" kao Camunda user task s ugrađenim Camunda Forms oblikom.

Izrada Camunda Forme

Formu možemo također dizajnirati unutar Camunda Modelera. Odaberite `File > New File > Form` (Camunda 8) i spremite je unutar procesne aplikacije kao `order-confirmation-form.form`. Forme u Camundi imaju nastavak `.form`.

Otvorit će vam se Camundin uređivač formi gdje možete dodavati različite elemente forme (npr. tekstualna polja, padajuće liste, radio gumbe, checkboxove, datume, brojeve itd.) kako biste prikupili potrebne informacije od korisnika.



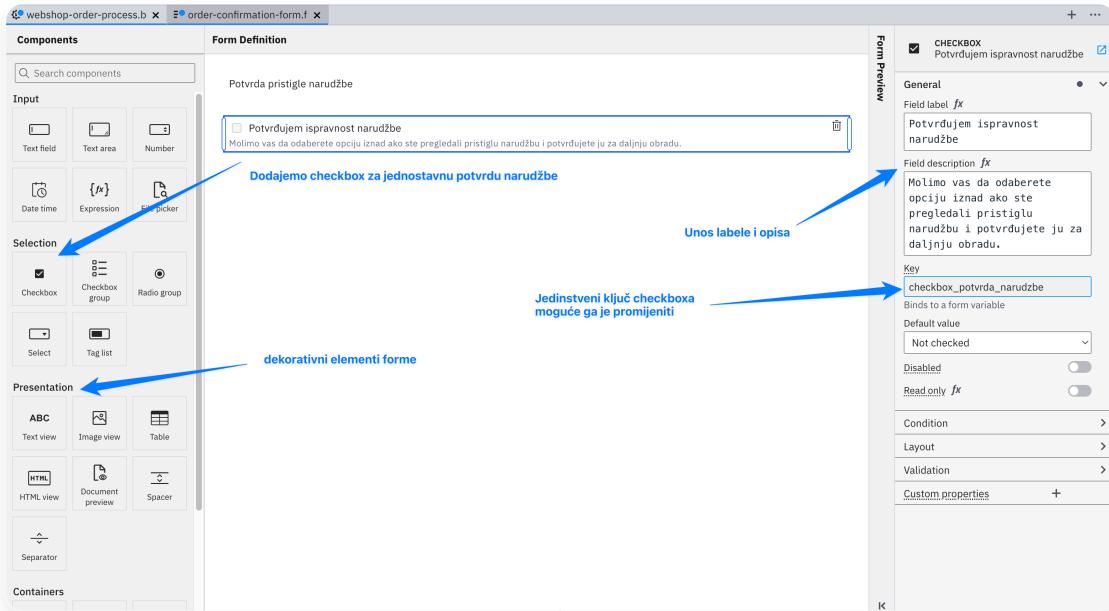
Slika 20. Izrada Camunda Forms forme "order-confirmation-form.form" unutar Camunda Modelera.

U editoru postoje razni elementi koje možemo koristiti za izradu forme, uključujući **Input elemente** gdje se očekuje slobodan unos teksta, **Selection** elemente gdje korisnik bira između unaprijed definiranih opcija, **Presentation** elemente za prikaz informacija bez unosa (uređivanje forme) te **Containers** za složene strukture i grupiranje elemenata, unos web elemenata (`iFrame`) i sl.

Odabrat ćemo jednostavni **Text View** za dodavanje naslova i dodati **Checkbox** element za potvrdu narudžbe.

Elementi forme također imaju postavke koje uređujemo preko panela s desne strane.

Uredite labelu, opis i **ključ** elementa kako biste ga lakše referencirali unutar procesa. Primjer na slici ispod:



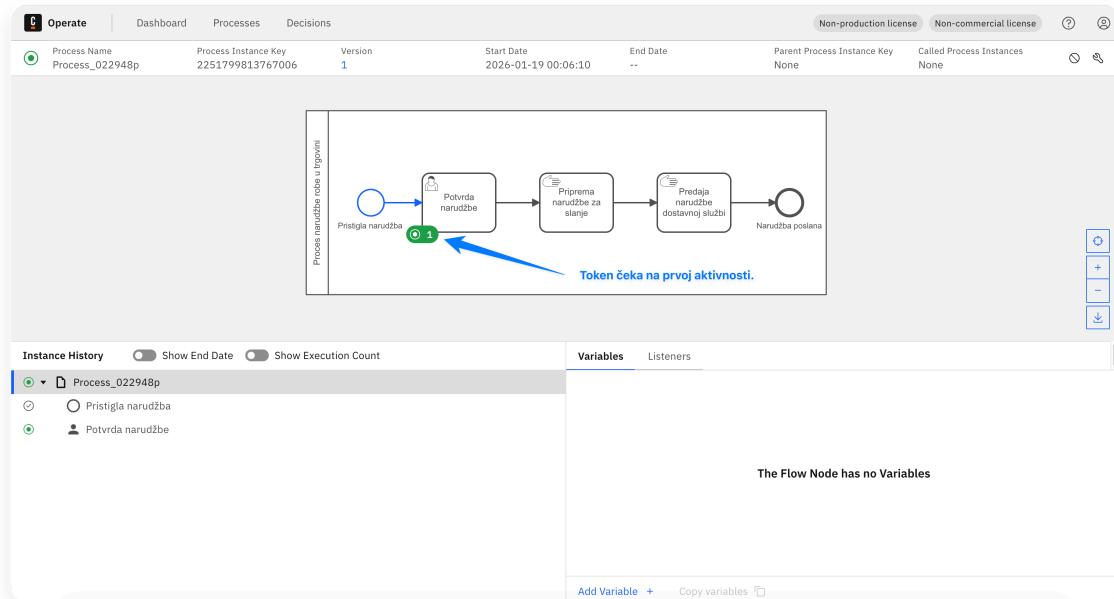
Slika 21. Konfiguracija elemenata forme "order-confirmation-form.form" unutar Camunda Modelera.

To je to za sad, vraćamo se na BPMN model i spremamo sve promjene.

Možemo pokušati *deployati* prvu verziju naše procesne aplikacije `webshop-order-process` na Camunda platformu koristeći ikonu rakete u donjem lijevom kutu Camunda Modelera.

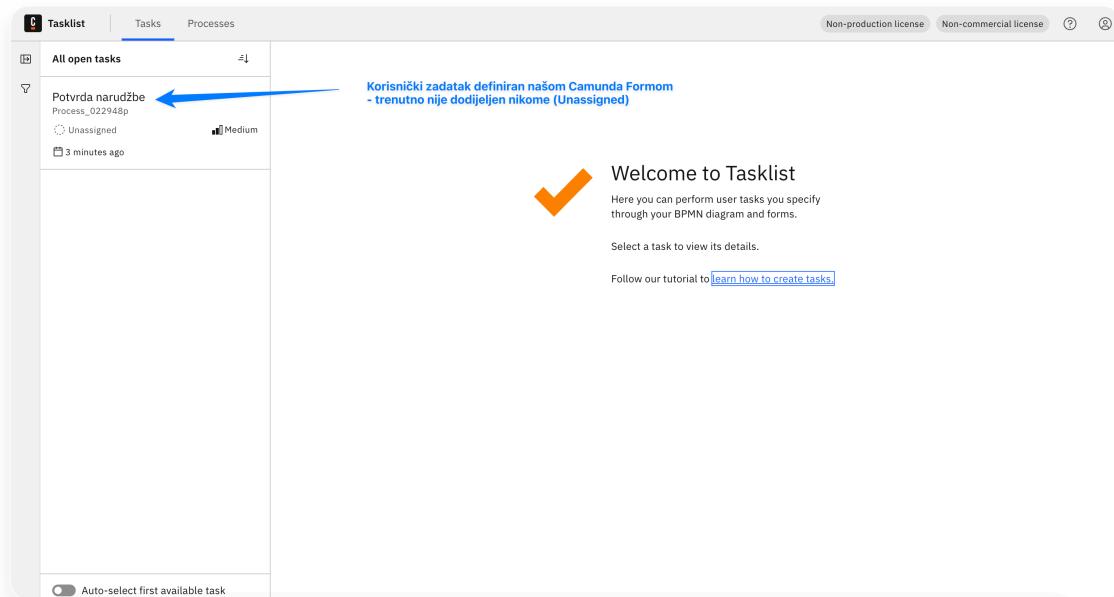
Nakon toga, pokrenite procesnu instancu iz Modelera koristeći ikonu **Run** . Za sada nećemo slati početne procesne varijable.

Otvorite Operate i provjerite stanje poslovnog procesa. Ako ste sve ispravno napravili, token procesne instance će "čekati" na korisničkom zadatku "Potvrda narudžbe".



Slika 22. Procesna instanca "webshop-order-process" u Camunda Operate aplikaciji s tokenom na korisničkom zadatku "Potvrda narudžbe".

Sada otvorite Camunda Tasklist aplikaciju kako biste vidjeli dodijeljene zadatke. Otvorite resurs `/tasklist` - trebali biste vidjeti novi zadatak pod nazivom "Potvrda narudžbe".

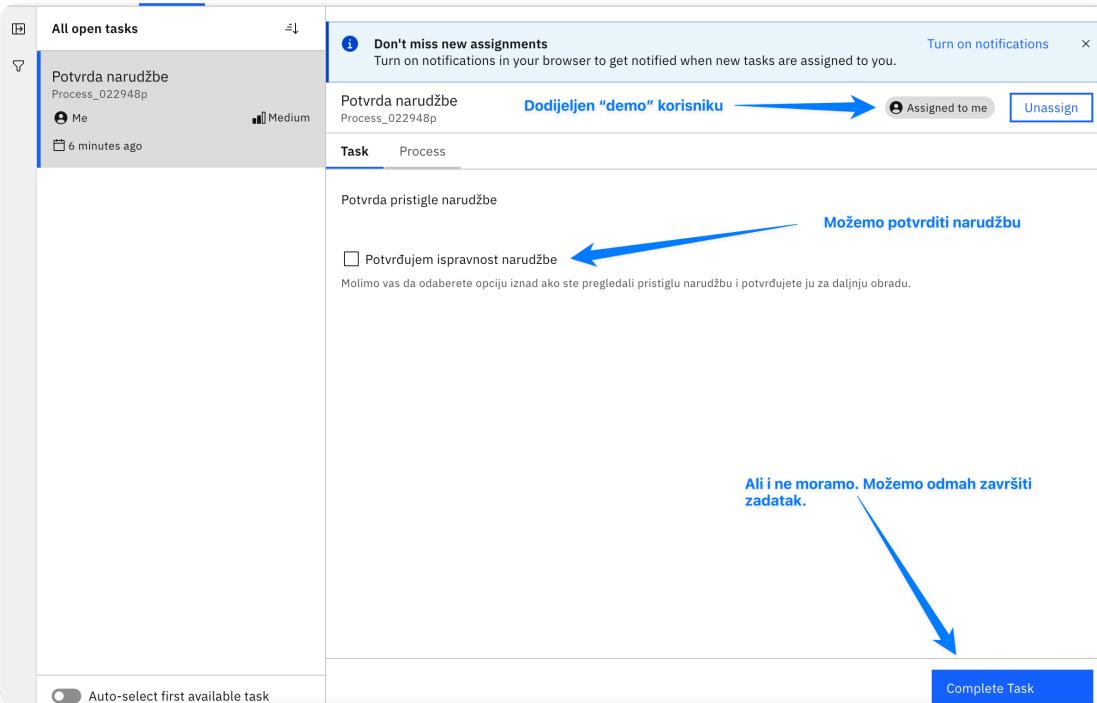


Slika 23. Pregled dodijeljenih zadataka u Camunda Tasklist aplikaciji.

Odaberite zadatak "Potvrda narudžbe" kako biste otvorili detalje zadatka i ispunili formu. Vidjet ćete da zadatak nije **dodijeljen nijednom korisniku** (Assignee: *unassigned*).

Rekli smo da u našim procesima postoje različiti dionici. Iz tog razloga, u Camundi je moguće dodijeliti zadatke određenim korisnicima ili grupama korisnika.

Za sada možemo dodijeliti zadatok sami sebi klikom na gumb `Assign to me` budući da smo prijavljeni kao `demo` korisnik koji ima administratorska prava.

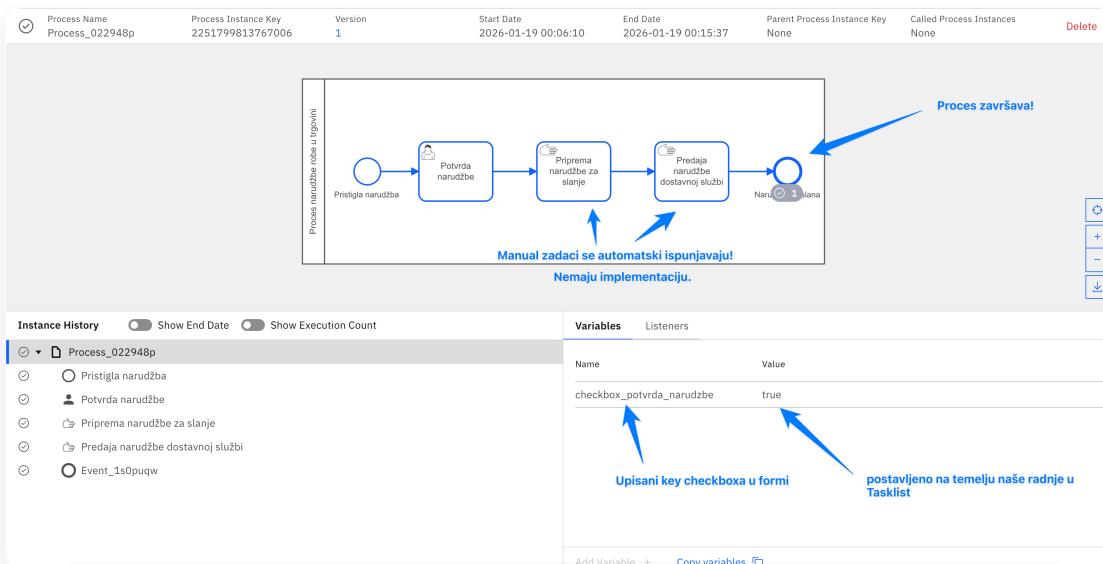


Slika 24. Dodjela zadatka "Potvrda narudžbe" korisniku `demo` u Camunda Tasklist aplikaciji.

Nakon dodjeljivanja, zadatak možemo kompletirati i bez odabira *checkboxa* za potvrdu narudžbe (npr. ako narudžba nije ispravna). Jednostavno stisnite gumb `Complete Task` u donjem desnom kutu. Možemo i potvrditi narudžbu - **trenutno nema razlike u ponašanju procesa**.

To je to! Vratimo se u Operate aplikaciju i osvježite karticu "Processes". Sada biste trebali vidjeti da je token procesne instance pomaknut na završni događaj "Narudžba poslana", što označava da je proces uspješno dovršen.

Zašto je to tako? Trenutno nemamo implementaciju ručnih (*eng. manual task*) zadataka pa stoga procesni *engine* automatski prolazi kroz te aktivnosti **bez čekanja na vanjsku interakciju**. Ovo nije greška - jednostavno nismo implementirali logiku za te zadatke.



Slika 25. Završena procesna instanca "webshop-order-process" u Camunda Operate aplikaciji s tokenom na završnom događaju "Narudžba poslana". Uočite kako se izmjenila procesna varijabla `checkbox_potvrda_narudzbe` na odabranu vrijednost u Camunda Tasklist aplikaciji.

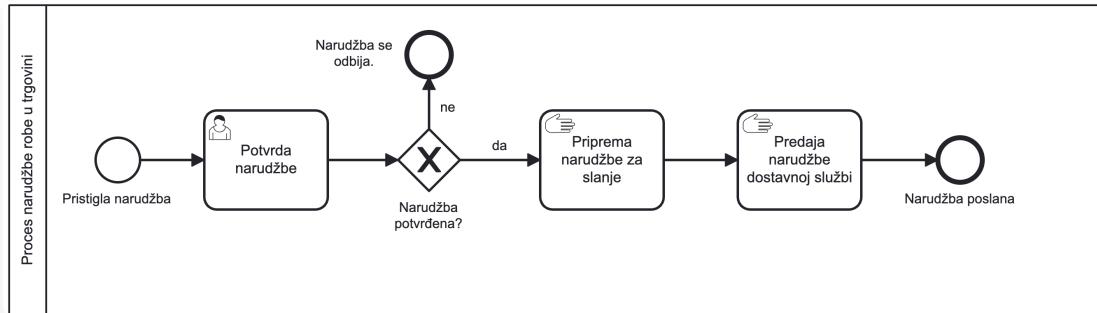
3.4 XOR grananje procesa na temelju procesnih varijabli

Sada ćemo nadograditi naš proces "webshop-order-process" kako bismo implementirali **XOR grananje toka procesa** na temelju vrijednosti procesne varijable `checkbox_potvrda_narudzbe`.

U slučaju da korisnik potvrdi narudžbu, proces će nastaviti dalje regularnim tokom, dok će u slučaju da korisnik ne potvrdi narudžbu, proces završiti.

Idemo direktno nadograditi BPMN model dodavanjem jednostavne XOR skretnice grananja nakon korisničkog zadatka "Potvrda narudžbe".

- Zašto onda? **Zato što ćemo imati vrijednost procesne varijable** `checkbox_potvrda_narudzbe` koja nam govori je li korisnik potvrdio narudžbu ili ne.



Slika 26. Nadograđeni BPMN model "webshop-order-process" s XOR skretnicom grananja toka procesa na temelju vrijednosti procesne varijable `checkbox_potvrda_narudzbe`.

Dakle, vrijednost procesne varijable `checkbox_potvrda_narudzbe` može biti `true` ili `false` ovisno o tome je li korisnik označio checkbox u formi. **To je ujedno i predikat kojim želimo modelirati labelu na XOR skretnici grananja.**

Moramo odabrati izlazne sljedove iz skretnice, te postaviti uvjete za svaki izlazni slijed - jednako kao što smo naučili modelirati.

U *Properties Panelu* za XOR skretnicu, vidjet ćete **Condition sekciju**. Ovdje pišemo uvjete sada već poznatim **FEEL izrazima**.

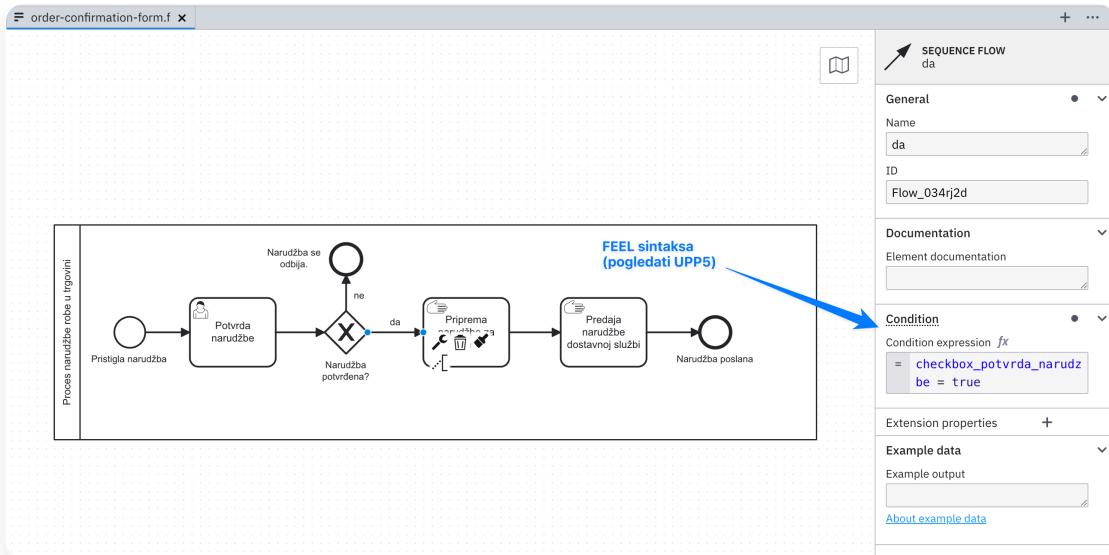
Za prvi izlazni slijed koji vodi prema aktivnosti "Priprema narudžbe za slanje", definirat ćemo FEEL uvjet:

```
checkbox_potvrda_narudzbe = true
```

Za drugi izlazni slijed koji vodi prema završnom događaju "Narudžba se odbija", definirat ćemo FEEL uvjet:

```
checkbox_potvrda_narudzbe = false
```

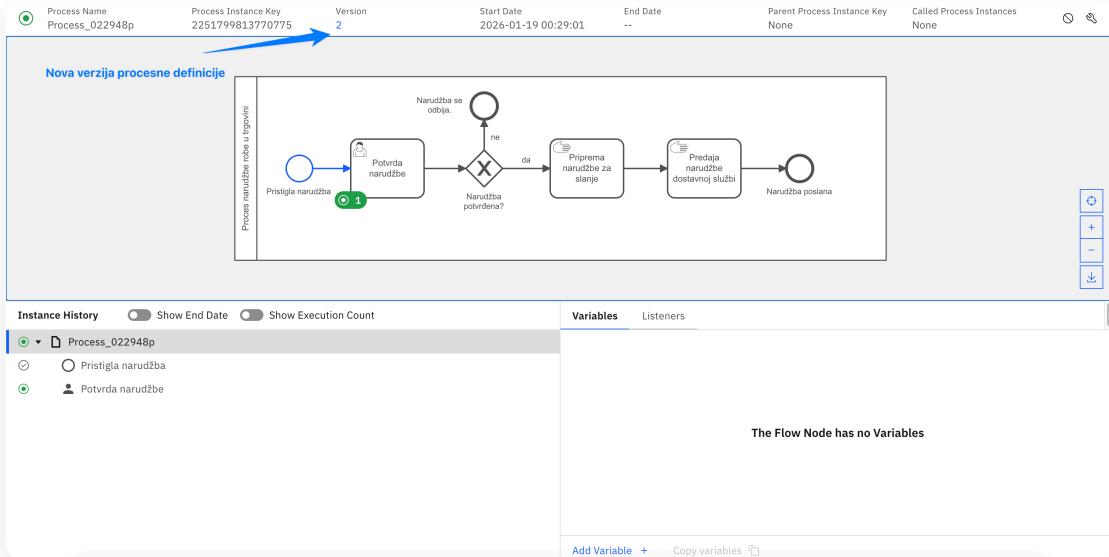
Simple as that!



Slika 27. Definiranje FEEL uvjeta za izlazne slijedove iz XOR skretnice grananja toka procesa u Camunda Modeleru.

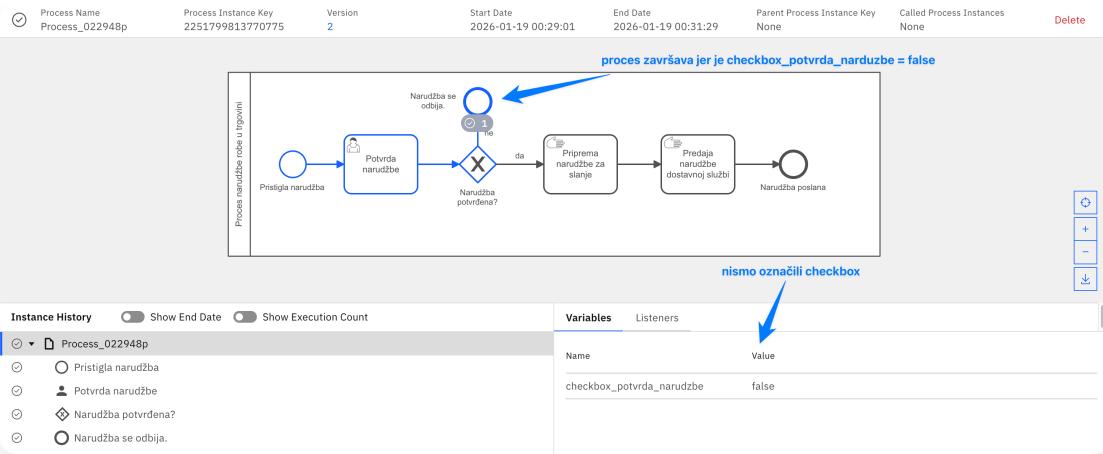
Spremite sve promjene i *deployajte* nadograđeni BPMN model na Camunda platformu. Nakon toga, pokrenite novu procesnu instancu.

Otvorite Operate aplikaciju i provjerite stanje procesne instance. Token bi trebao biti na korisničkom zadatku "Potvrda narudžbe". Također, primjetite da je sada definicija procesa nadograđena na verziju 2.



Slika 28. Procesna instanca "webshop-order-process" u Camunda Operate aplikaciji s tokenom na korisničkom zadatku "Potvrda narudžbe" nakon nadogradnje BPMN modela na verziju 2.

Pokrenite Camunda Tasklist aplikaciju i otvorite zadatak "Potvrda narudžbe". Ovaj put, **nemojte označiti checkbox za potvrdu narudžbe i kompletirajte zadatak.**



Slika 29. Završena procesna instanca "webshop-order-process" u Camunda Operate aplikaciji s tokenom na završnom događaju "Narudžba se odbija" nakon što korisnik nije potvrdio narudžbu.

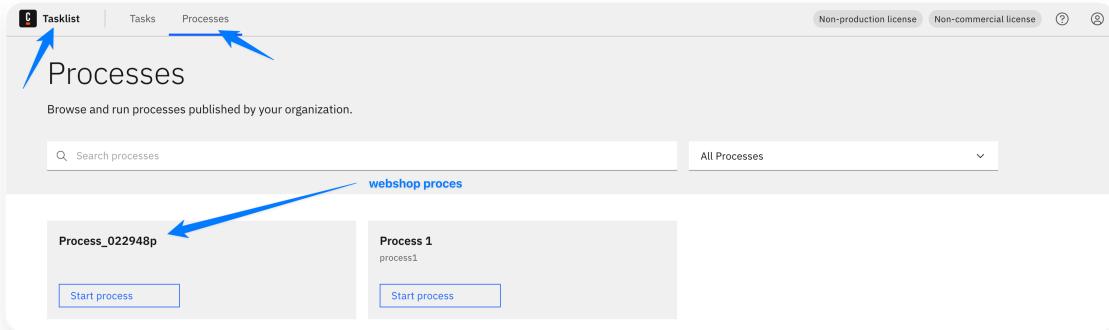
3.5 Kako još pokrenuti procesne instance?

Sigurno se pitate - zašto pokrećemo procesne instance iz Camunda Modelera? To nam nema smisla ako se aplikacija nalazi u proizvodnjkom okruženju. Tada želimo da joj korisnici pristupaju putem lijepih web i mobilnih sučelja koje smo izradili.

To je točno. Pokretanje procesnih instanci iz Camunda Modelera je zgodno za razvoj i testiranje, ali u stvarnom svijetu, procesne instance se obično pokreću na druge načine - **najčešće putem REST API poziva iz naših vlastitih aplikacija**.

Osim toga, želimo proslijediti naše podatke o narudžbi kao početne procesne varijable prilikom pokretanja procesa (možemo i to kroz Modeler, ali nije skroz praktično).

Osim preko REST API-ja, procesne instance možemo pokretati i preko Tasklist aplikacije (ručno pokretanje procesa). Jednostavno otvorite aplikaciju, odaberite karticu **Processes** i odaberite procesnu definiciju za koju želite pokrenuti novu instancu.



Slika 30. Pokretanje nove procesne instance iz Camunda Tasklist aplikacije.

Naš Zeebe engine radi na [gRPC](#) protokolu. Radi se o komunikacijskom protokolu koji omogućuje visoku efikasnu komunikaciju između klijenata i servera koristeći HTTP/2 kao transportni sloj - često se koristi u mikroservisima i raspodijeljenim sustavima zbog svoje brzine i niske latencije (priprema za kolegij - [Raspodijeljeni sustavi](#) za one koji nastavljaju diplomski studij).

Međutim, Zeebe također nudi REST API sloj putem HTTP protokola preko kojeg možemo slati HTTP zahtjeve i obavljati različite operacije nad procesniminstancama, procesnim definicijama, varijablama i sl., uključujući:

- *Deployanje procesne definicije ([POST /deployments](#))*
- *Pokretanje procesne instance ([POST /process-instances](#))*
- *Izvršavanje korisničkog zadatka ([POST /user-tasks/{userTaskKey}/completion](#))*

Dokumentacija za Camunda Zeebe REST API dostupna je na: [Camunda Zeebe REST API Reference](#).

Otvorite Postman ili curl, i testirajte topologiju vaše Camunda platforme slanjem HTTP zahtjeva na endpoint `/topology`:

```
→ curl http://localhost:8080/v2/topology
```

Ispis:

```
{"brokers": [{"nodeId": 0, "host": "localhost", "port": 26501, "partitions": [{"partitionId": 1, "role": "leader", "health": "healthy"}]}, {"version": "8.8.9"}], "clusterSize": 1, "partitionsCount": 1, "replicationFactor": 1, "gatewayVersion": "8.8.9", "lastCompletedChangeId": "-1"}
```

Ovo potvrđuje da je naš Zeebe engine dostupan putem REST API-ja na <http://localhost:8080>.

Ipak, kako koristimo Self-Managed paket, moramo dodati router prefix `/v2` u URL jer je Camunda platforma konfiguirirana da koristi ovaj prefix za REST API pozive. Ovo nam se ispisuje prilikom pokretanja Camunde pod **Orchestration Cluster API**.

Ako otvorite Camunda dokumentaciju REST API-ja za `POST /process-instances`, vidjet ćete upute kako koristiti ovaj endpoint, odnosno kako strukturirati HTTP zahtjev za pokretanje nove procesne instance.

The screenshot shows the Camunda REST API documentation for the `Create process instance` endpoint. The URL is `http://localhost:8080/v2/process-instances`. The REQUEST section includes a CURL example and a detailed description of the Body required fields. The Body is defined as `curl -L 'http://localhost:8080/v2/process-instances' -H 'Content-Type: application/json' -H 'Accept: application/json' -d '{ "processDefinitionId": "new-account-onboarding-wc", "processDefinitionVersion": -1, "variables": {}, "tenantId": "customer-service", "operationReference": 0, "startInstructions": [] }'`. The REQUEST section also includes fields for Base URL (`http://localhost:8080/v2`), Body (REQUIRED), and examples like `Example (from schema)`, `By process definition key`, and `By pr`.

Slika 31. Dokumentacija za Camunda Zeebe REST API endpoint `POST /process-instances` za pokretanje nove procesne instance. Uočite različite primjere HTTP klijenata (curl, JavaScript, Python, Java, Go, C#).

Poslat ćemo sljedeći curl zahtjev za pokretanje nove procesne instance bez početnih procesnih varijabli:

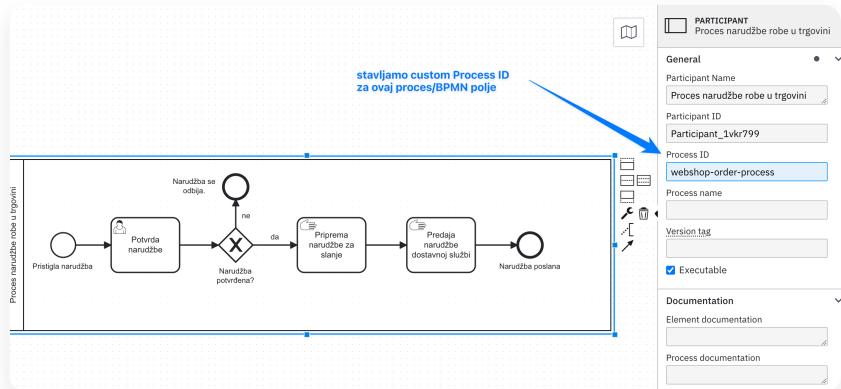
- zastavica `-L` omogućuje praćenje eventualnih preusmjerenja ([HTTP redirects](#))
- šaljemo dva HTTP zaglavla: `Content-Type: application/json` i `Accept: application/json`
- šaljemo JSON tijelo zahtjeva s podacima o procesnoj definiciji koju želimo pokrenuti (`processDefinitionId` i `processDefinitionVersion`) te praznim skupom početnih varijabli (`variables: {}`)

OPREZ! Potrebno je zamijeniti vrijednost `processDefinitionId` s ID-om vaše **procesne definicije** koju želite pokrenuti.

Primjer slanja HTTP zahtjeva za pokretanje instance bez početnih varijabli:

```
→ curl -L 'http://localhost:8080/v2/process-instances' \
-H 'Content-Type: application/json' \
-H 'Accept: application/json' \
-d '{
  "processDefinitionId": "Process_022948p",
  "processDefinitionVersion": 2,
  "variables": {}
}'
```

Kako nam Camunda ne bi sama dodjeljivala ID procesne definicije, možemo sami to napraviti u Camunda Modeleru. Odaberite **polje** i u postavkama mu izmijenite ID na npr. `webshop-order-process`.



Slika 32. Postavljanje prilagođenog ID-a procesne definicije "webshop-order-process" unutar Camunda Modelera.

Prije sljedećeg slanja, obavezno *deployajte* procesnu definiciju na Camunda platformu kako biste osigurali da je najnovija verzija dostupna. Provjerite u Operate aplikaciji jesu li izmjene vidljive.

Nakon slanja zahtjeva, trebali biste dobiti HTTP odgovor s podacima o pokrenutoj procesnoj instanci:

```
{
  "processDefinitionId": "webshop-order-process",
  "processDefinitionVersion": 1,
  "tenantId": "<default>",
  "variables": {},
  "processDefinitionKey": "2251799813770739",
  "processInstanceKey": "2251799813775821",
  "tags": []
}
```

Idemo ovo preslikati u Postman kako bismo lakše započinjali nove procesne instance ubuduće. Zaglavlja ne moramo ručno dodavati u ovom slučaju budući da ih Postman automatski dodaje prilikom slanja JSON tijela zahtjeva.

The screenshot shows the Postman application interface. At the top, it says "camunda / započni instancu procesa webshop-order". Below that, a "POST" button is highlighted, and the URL "http://localhost:8080/v2/process-instances" is entered. The "Body" tab is selected, showing a raw JSON payload:

```

1 {
2   "processDefinitionId": "webshop-order-process",
3   "processDefinitionVersion": 1,
4   "variables": {}
5 }
```

Below the body, the response status is "200 OK" with a timestamp of "18 ms" and a size of "2.12 KB". The response body is identical to the one sent:
1 {
2 "processDefinitionId": "webshop-order-process",
3 "processDefinitionVersion": 1,
4 "tenantId": "<default>",
5 "variables": {},
6 "processDefinitionKey": "2251799813770739",
7 "processInstanceKey": "2251799813775821",
8 "tags": []
9 }

Slika 33. Uspješno instanciranje procesne instance "webshop-order-process" putem Postman HTTP klijenta.

Kako bismo definirali početne vrijednosti procesnih varijabli, možemo ih jednostavno dodati unutar JSON tijela zahtjeva. Dodat ćemo sljedeće početne procesne varijable:

- `customerName`: ime i prezime kupca (npr. "Ivan Horvat")
- `orderId`: jedinstveni identifikator narudžbe (npr. "ORD-1001")
- `orderTotal`: ukupni iznos narudžbe (npr. 299.99)
- `items`: niz artikala u narudžbi (npr. ["laptop", "mouse", "keyboard"])

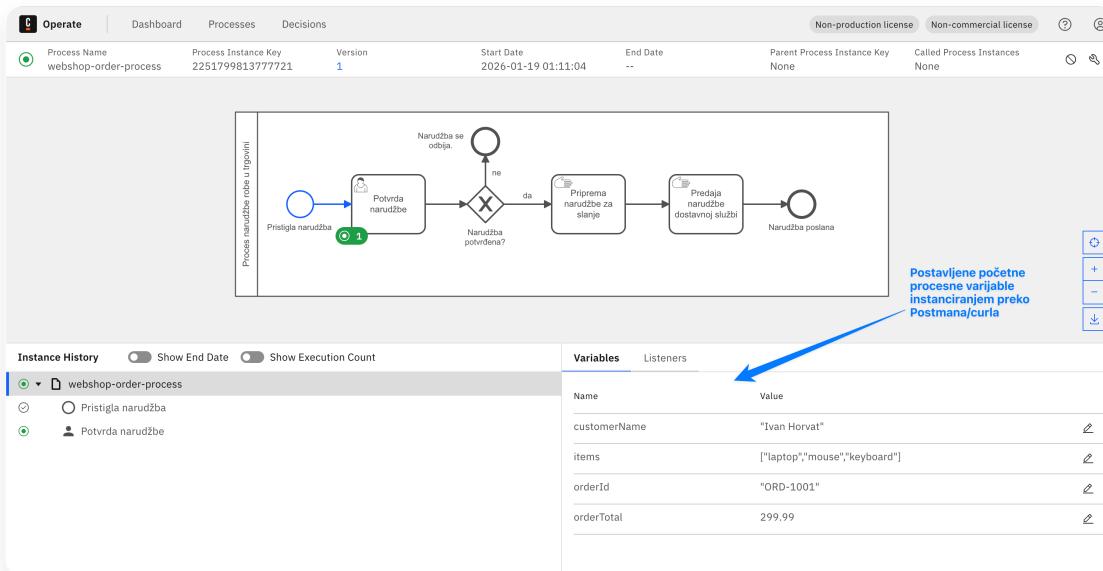
Primjer slanja HTTP zahtjeva za pokretanje instance s početnim procesnim varijablama:

```

→ curl -L 'http://localhost:8080/v2/process-instances' \
-H 'Content-Type: application/json' \
-H 'Accept: application/json' \
-d '{
  "processDefinitionId": "webshop-order-process",
  "processDefinitionVersion": 1,
  "variables": {
    "customerName": "Ivan Horvat",
    "orderId": "ORD-1001",
    "orderTotal": 299.99,
    "items": ["laptop", "mouse", "keyboard"]
  }
}'

```

Otvorite Operate aplikaciju i provjerite novu procesnu instancu. Uočite kako su početne procesne varijable dostupne unutar instance.



Slika 34. Procesna instanca "webshop-order-process" u Camunda Operate aplikaciji s početnim procesnim varijablama definiranim putem REST API-ja.

Ove procesne varijable dostupne su unutar cijele procesne instance i možemo ih koristiti za donošenje odluka, prikaz podataka u formama, te u job workerima. Možemo ih ažurirati za vrijeme izvođenja procesa, a njihove vrijednosti možemo pratiti i izmjenjivati i unutar Camunda Operate aplikacije.

3.6 Izmjena vrijednosti procesnih varijabli tijekom izvođenja procesa

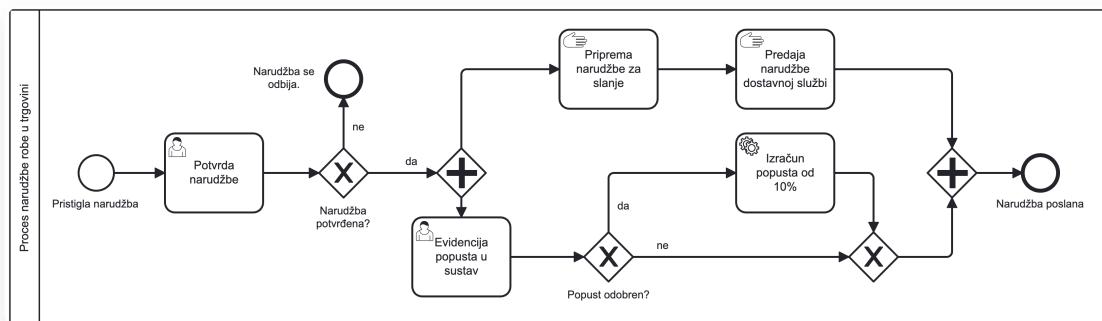
Naučili smo kako definirati jednostavne procesne aplikacije koristeći Camunda Modeler, kako pokretati procesne instance putem Modelera i REST API-ja, te kako koristiti Camundine aplikacije Operate i Tasklist za upravljanje procesnim instancama i zadacima. Sada ćemo naučiti kako možemo mijenjati vrijednosti procesnih varijabli tijekom izvođenja procesa koristeći **servisne zadatke** (*service tasks*), a na sljedećim vježbama i koristeći vanjske servise.

Nadogradit ćemo proces `webshop-order-process` tako da omogućimo djelatniku trgovine da odobri popust za narudžbu od 10%. Ako djelatnik odobri popust, vrijednost procesne varijable `orderTotal` će se smanjiti za 10%. Ako ne odobri, vrijednost ostaje nepromijenjena i proces nastavlja dalje.

Idemo modelirati ovu logiku u našem BPMN modelu.

Koje BPMN elemente dodajemo u model?

- Nakon XOR skretnice grananja, na izlazu "da", dodajemo AND skretnicu grananja kako bismo paralelno izvršili dvije aktivnosti:
 - "Priprema narudžbe" (ručni zadatak)
 - "Odobravanje popusta" (korisnički zadatak)
- Ako djelatnik odobri popust, dodajemo servisni zadatak "Izračunaj popust od 10%" koji će ažurirati vrijednost procesne varijable `orderTotal`.
- Nakon toga, dodajemo AND skretnicu spajanja i XOR skretnicu spajanja kako bismo spojili ispravno tokove procesa i nastavili prema aktivnosti "Isporuka narudžbe".



Slika 35. Nadograđeni BPMN model "webshop-order-process" s logikom za odobravanje popusta i izmjenu vrijednosti procesne varijable `orderTotal`.

Napomena: **Procesnu definiciju ne možete *deployati* dok imate grešaka** (pogledati Camunda konzolu na dnu). Nedostaju implementacije zadatka "Evidencija popusta u sustav", vrijednosti na XOR skretnicu "Popust odobren" i definicija servisnog zadatka "Izračun popusta od 10%".

Prikaz vrijednosti procesnih varijabli u Formi

Ipak, kako bismo unaprijedili korisničko iskustvo, želimo djelatniku u formi prikazati podatke koje je korisnik unio prilikom narudžbe. Nadogradit ćemo našu Camunda formu `order-confirmation-form.form` kako bismo prikazali sljedeće informacije:

- Ime i prezime kupca (`customerName`)
- ID narudžbe (`orderId`)
- Ukupni iznos narudžbe (`orderTotal`)
- Popis artikala u narudžbi (`items`)

Otvorite formu `order-confirmation-form.form` unutar Camunda Modelera i dodajte **tablicu**.

Nazvat ćemo tablicu "Pristigla narudžba" i dodati joj 4 stupca (prema ključevima u procesnim varijablama):

- `customerName`
- `orderId`

- `orderTotal`
- `items`

Za svaki stupac dodajte naslov i odgovarajući ključ (mi ćemo dodati sufiks `_form` svakom kako bismo razlikovali varijable u formi od onih u procesu).

The screenshot shows the Camunda Forms interface with the following details:

- Components** sidebar: Shows various input types like Text field, Date time, Selection (Checkbox, Radio group, Select, Tag list), and Presentation (ABC, Text view, Image view, Table, HTML view, Document preview, Spacer).
- Form Definition**: The main workspace contains a table named "Pristigla narudžba" with columns: Customer name, Order ID, Order Total, and Items. Below the table is a note: "Potvrđujem ispravnost narudžbe. Molimo vas da odaberete opciju iznad ako ste pregledali pristiglu narudžbu i potvrđujete ju za daljnju obradu."
- Form Preview**: A panel on the right side shows the data source configuration for the table:
 - Headers source**: Type is set to "List of items".
 - Header items** (under "Pristigla narudžba") are defined with keys corresponding to process variables:
 - Customer name: Key `customer_name_form`
 - Order ID: Key `order_id_form`
 - Order Total: Key `order_total_form`
 - Items: Key `items_forms`

Slika 36. Dodavanje tablice "Pristigla narudžba" u Camunda Forms formu "order-confirmation-form.form".

Podatke ćemo napuniti FEEL izrazom pod **Data Source**:

```
[  
{  
    customer_name_form: customerName,  
    order_id_form: orderId,  
    order_total_form: orderTotal,  
    items_forms: string join(items, ", ")  
}  
]
```

Ovdje koristimo FEEL funkciju `string join()` kako bismo niz artikala iz procesne varijable `items` pretvorili u jedan string razdvojen zarezima za prikaz u stupcu tablice. **Pod ključeve navodimo definirane ključeve stupaca tablice, a kao vrijednosti se referenciramo na procesne varijable.**

Zatim, izradit ćemo novu Camunda formu `popust_form.form` gdje ćemo jednostavno dodati checkbox za odobravanje popusta. Pohranite novu formu unutar procesne aplikacije.

Dodat ćemo u formu i sliku koja označava 10%, čisto da izgleda malo zanimljivije. Procesnu varijablu odobrenog popusta nazvat ćemo `popust_odobren`.

Slika 37. Camunda Forms forma "popust_form.form" za odobravanje popusta s checkbox elementom.

Obavezno moramo još dodati ispravne FEEL uvjete na izlazne slijedove iz XOR skretnice "Popust odobren?":

- Za izlazni slijed "da":

```
popust_odeobren = true
```

- Za izlazni slijed "ne":

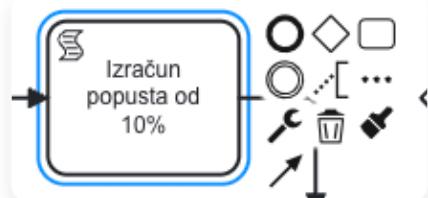
```
popust_odeobren = false
```

Script task za izračun popusta

Sada moramo implementirati servisni zadatak "Izračun popusta od 10%" koji će ažurirati vrijednost procesne varijable `orderTotal` smanjenjem za 10%. Ipak, kako nemamo vanjski job worker za ovaj zadatak, moramo koristiti jednostavni **script task** koji će izvršiti skriptu unutar samog procesnog enginea.

Ne koristi li se servisni zadatak za to? Da, ali u kontekstu procesnog aplikacija na Camunda 8 platformi, servisni zadaci se interpretiraju kao vanjski servisi koji rade određene poslove izvan procesnog enginea. Budući da nemamo vanjski servis za izračun popusta, koristit varijantu **script taska** koji omogućuje izvršavanje skripti unutar procesa.

Script task omogućuje sinkrono izvršavanje koda unutar procesnog enginea bez potrebe za vanjskim servisom. Ovo je korisno za jednostavne zadatke poput jednostavnih izračuna ili manipulacije podacima. Za sve ozbiljnije zadatke, preporučuje se korištenje servisnih zadataka s vanjskim job workerima.



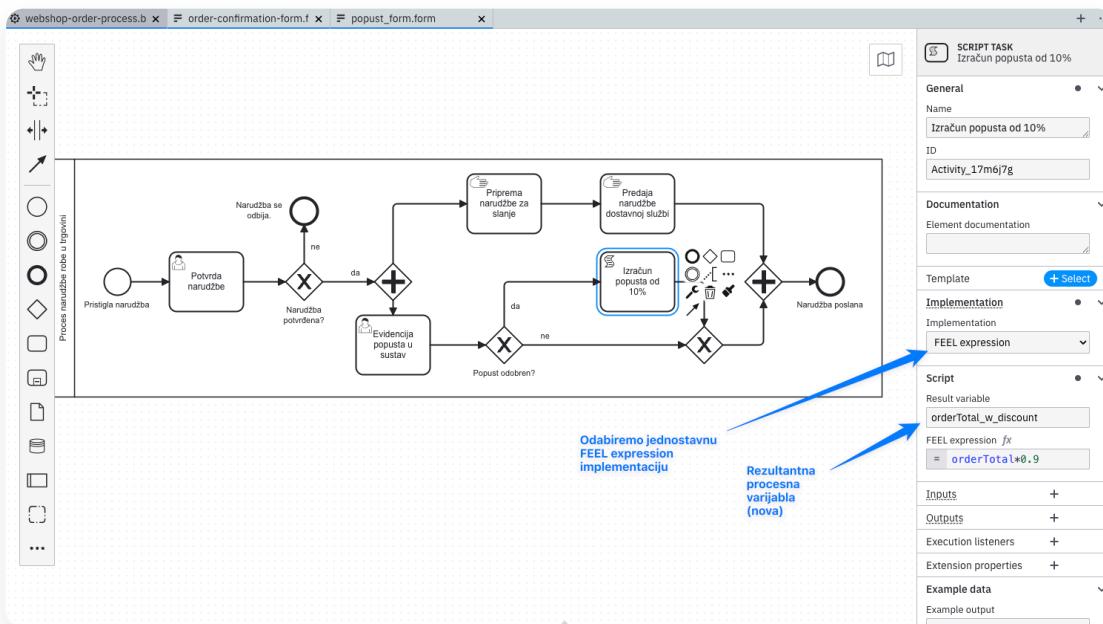
Slika 38. Zamjena servisnog zadatka "Izračun popusta od 10%" sa skriptnim taskom unutar Camunda Modelera.

Izmijenite postavke *Implementation* u *Properties Panelu* za odabrani **script task** u `FEEL Expression`.

Nakon toga, definirajte naziv **rezultantne procesne varijable**, npr. `orderTotal_w_discount` te unesite sljedeći FEEL izraz za izračun popusta:

```
orderTotal * 0.9
```

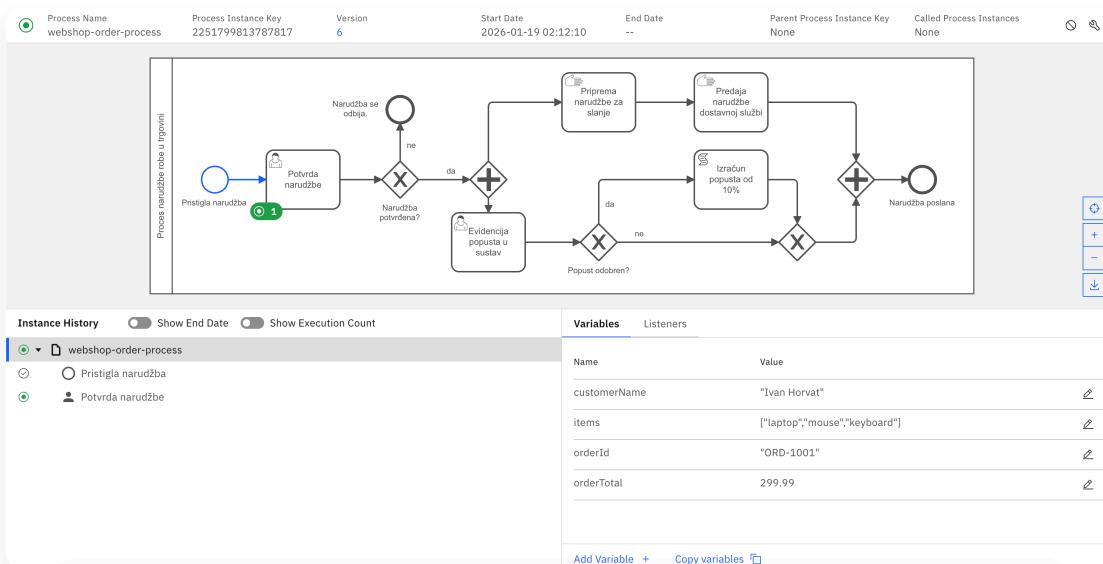
- vrijednost procesne varijable `orderTotal` se množi s 0.9 kako bi se izračunala nova vrijednost s popustom od 10%



Slika 39. Konfiguracija *script taska* s FEEL izrazom za izračun popusta unutar Camunda Modelera.

Spremite sve promjene i *deployajte* nadograđeni BPMN model na Camunda platformu. Nakon toga, pokrenite novu procesnu instancu s početnim procesnim varijablama putem Postmana. **Pazite da odaberete točnu verziju procesne definicije.**

Početno stanje u Operate aplikaciji: čekanje na potvrdu narudžbe. Dakle, prebacujemo se u Tasklist aplikaciju.



Slika 40. Procesna instanca "webshop-order-process" u Camunda Operate aplikaciji s tokenom na korisničkom zadatku "Potvrda narudžbe" nakon nadogradnje BPMN modela.

Ako otvorite zadatak "Potvrda narudžbe" u Tasklist aplikaciji, vidjet ćete da je tablica ispunjena podacima iz procesnih varijabli kao što smo definirali u formi FEEL izrazom.

Customer name	Order ID	Order Total	Items
Ivan Horvat	ORD-1001	299.99	laptop, mouse, keyboard

Slika 41. Pregled zadatka "Potvrda narudžbe" u Camunda Tasklist aplikaciji s ispunjenom tablicom podacima iz procesnih varijabli.

Potvrđujemo narudžbu i odmah prelazimo na zadatak "Odobravanje popusta". Ovdje ćemo označiti checkbox za odobravanje popusta i kompletirati zadatak.

Nakon odobravanja popusta, *script task* će automatski izračunati novu vrijednost procesne varijable `orderTotal_w_discount`, umanjujući 10% u odnosu na početnu vrijednost `orderTotal`.

Gotovi smo! Vratimo se u Operate aplikaciju i osvježimo karticu "Processes". Proces ćete pronaći pod filterom **Finished Instances**: `Completed`. Odaberite instancu i pregledajte je li se uspješno izračunala vrijednost procesne varijable `orderTotal_w_discount`.

Name	Value
<code>checkbox_potvrda_narudzbe</code>	true
<code>customerName</code>	"Ivan Horvat"
<code>items</code>	["laptop","mouse","keyboard"]
<code>orderId</code>	"ORD-1001"
<code>orderTotal</code>	299.99
<code>orderTotal_w_discount</code>	269.991
<code>popust_odobren</code>	true

Slika 42. Završena procesna instance "webshop-order-process" u Camunda Operate aplikaciji s izmijenjenom vrijednošću procesne varijable `orderTotal_w_discount` nakon odobrenja popusta.

Nekoliko savjeta za razvoj u Camundi 8

Kako se snalaziti u izraditi Camunda 8 procesnih aplikacija?

- Radite u malim koracima (iterativno) i često *deployajte* svoje promjene na Camunda platformu
- Iskoristite primjere iz ove skripte kao vodič za izradu vlastitih procesnih aplikacija
- Koristite službenu Camunda dokumentaciju za dodatne informacije o svojstvima koja možete mijenjati različitim BPMN elementima
- Testirajte svoje procese često kako biste osigurali da sve radi kako treba i obratite pažnju na greške u Camunda konzoli (na dnu Modelera)
- Pomognite si generirati složenije koristeći genAI te ih testirajte poput [FEEL Playground](#) alata.
- Definirajte si osnovno okruženje u Postmanu kako biste mogli brže testirati svoje procese putem REST API-ja

Zadaci za vježbu 6

Modelirajte jednostavni proces prijave studentske prakse na Fakultetu informatike u Puli. Postoje 3 sudionika u procesu prakse (modelirajte 1 polje i 3 staze):

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

Proces započinje kod studenta odabirom zadatka za praksu. Student ispunjava Camunda 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. (podatke pošaljite kao početne procesne varijable).

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 prihvatiti ili odbiti. Ako ga odbije, proces se ponovno vraća na studenta i njegov unos podataka.

Ako poslodavac prihvati studenta, proces se grana prema studentu koji sad mora unijeti kratak opis zadatka, datum izvođenja prakse te ime i prezime mentora koji mu je dodijeljen, te se paralelno grana prema profesoru kojeg se samo obavještava (simulacija). Nakon izvršavanja tih paralelnih aktivnosti, proces se završava.

Nakon što ste modelirali proces, implementirajte procesnu aplikaciju u **Camundi 8**:

- Dodajte definirane korisničke aktivnosti i korespondirajuće Camunda forme
- Definirajte procesne varijable i njihove vrijednosti (REST API)
- Definirajte skretnice i uvjete na izlaznim tokovima
- Obavještavanje sudionika procesa ne implementirate (možete pustiti manual task - automatski se preskače)

Predajete zip datoteku koja sadrži sve potrebne datoteke procesne aplikacije uključujući screenshotove iz Operate i Tasklist aplikacija koji dokazuju ispravno izvođenje procesa.