ВИСОКА ШКОЛА ЕЛЕКТРОТЕХНИКЕ И РАЧУНАРСТВА СТРУКОВНИХ СТУДИЈА

**Живојиновић Марко РИН-26/19**

**Размена података у оквиру Oracle BPM архитектуре**

**- примењени истраживачки рад -**



Београд, јун 2021.

САДРЖАЈ:

[1. Увод 2](#_Toc74175741)

[2. BPM системи 3](#_Toc74175742)

[2.1. Пословни процеси 3](#_Toc74175743)

[2.2. Компоненте пословног процеса 4](#_Toc74175744)

[3. Софтверске платформе за BPM 6](#_Toc74175745)

[4. Oracle BPM 7](#_Toc74175746)

[4.1. Java Servlet 7](#_Toc74175747)

[4.2. *JSF* странице и *Bean* објектИ 9](#_Toc74175748)

[4.2.1. JSF Managed Bean 9](#_Toc74175749)

[5. Закључак 11](#_Toc74175750)

[6. Литература 12](#_Toc74175751)

# Увод

Када се говори о имплементацији сложених решења намењеним за функционисање великих пословних система, постоје разне стратегије којима се може доћи до жељеног решења оваквих система. Пре свега, питање је за коју платформу је намењено софтверско решење за рад оваквих система. Данас се грубо према платформи за коју су намњене, апликације могу поделити на стандардне десктоп апликације, које се затим морају инсталирати на сваки кориснички уређај да би се омогућио рад крајњим корисницима, андроид апликације које су намењене првенствено мобилним уређајима и веб апликације. У данашње време све чешће се прибегава изради веб апликација које имају задатак да омогуће рад у датом пословном систему. Један од главних разлога зашто су веб апликације све популарније за коришћење јесте независност оперативног система корисничког уређаја од самих апликација. Све што је неопходно да би се приступило апликацији и користиле њене функционалности јесте веб претраживач који поседује готово сваки кориснички уређај данас.

На почетку развоја веб апликација, потребно је размислити о избору програмског језика, односно технологија (*framework*) које ће бити коришћене за израду апликације. Поред овога, битно је размислити о начину организације и чувања података, најчешће у оквиру одређене релационе базе података. Уколико се одабере овакав правац развоја, програмери имају потпуну слободу у коришћењу одабраних технологија и њиховог комбиновања у циљу израде веб апликација различитих намена. Међутим, уочено је да се често многи пословни системи могу свести на опис активности које корисници изводе у оквиру одређених процеса. Због ове чињенице, многе софтверске компаније су развиле своје BPM(*Business Process Management*) софтверске имплементације које имају намену да олакшају развој решења за разне пословне системе. У оквиру овог рада ће бити описан начин рада у оквиру OracleBPM платформе и поређење са другим BPMсистемима.

# BPM системи

Типичан начин рада у оквиру готово било које фирме, без обзира на делатност којом се бави се може описати кроз пословне процесе који се одвијају у оквиру те фирме. BPM служи како би се ови процеси што прецизније описали, и уочила њихова евентуална повезаност, као и комплетан опис свих активности које се могу изводити у оквиру њих. Дакле, BPM као појам није конкретно везан за развој софтверских решења која служе за израду пословних процеса који се одвијају у оквиру неке фирме. BPM представља технику која има своја правила и принципе помоћу којих се дефинишу пословни процеси. Када се говори о побољшању пословања одређене фирме, кроз BPM се не може утицати на побољшање конкретних активности који запослени изводе у оквиру неког процеса. Заправо, кроз BPM се дефинише целокупни пословни систем и његови међусобно повезани елементи. Елементи у оквиру BPM су дакле одређени догађаји који се могу уочити, активности и одлуке. Управо ови елементи су кључни фактори у сваком пословном процесу који одређују његов ток.

## Пословни процеси

У сваком привредном субјекту, фирми, организацији се одвијају процеси који дефинишу пословање. Типични примери процеса који се могу видети у оквиру фирми могу бити:

* Наруџбина – плаћање: Процес започиње трговац, када клијент поручи одређени производ, или закаже услугу, а завршава се када се дати производ испоручи и обави плаћање. Овај тип процеса садржи активности које су везане за верификацију наруџбине, слање нарученог производа, испоруку и уручење рачуна и потврду од стране клијента.
* Жалба клијента – решење проблема: Овај процес започиње у моменту када клијент покрене жалбу због незадовољства пруженом услугом, односно купљеним производом. Процес се не завршава док се клијент, фирма, а у најбољем случају обоје не сложе да је проблем решен.
* Подношење захтева – одобрење: Још једна доста честа врста процеса, који почиње када особа подноси пријаву везану за одређену потребу, који се може завршити одбијањем или одобрењем. Ова врста процеса се може срести у оквиру разних државних служби где грађани подносе захтеве, као што је издавање грађевинске дозволе, на пример.

Како видимо у горе наведеним примерима, пословни процеси су управо оно што се одвија у оквиру фирми и организација када је потребно извршити интеракцију са крајњим корисницима. Начин на који су ови процеси организовани и изведени одређује квалитет услуге који се пружа клијентима, као и ефикасност целокупног пословања једне фирме.

## Компоненте пословног процеса

Како се види у претходном поглављу, пословни процеси се најчешће састоје из више догађаја (*event*) и активности (*activity*). Догађаји се односе на ствари које се у неком моменту догоде, и немају предвиђено временско трајање. Догађаји могу покренути серију активности које следе након њиховог дешавања. На пример, када се деси догађај да клијент поднесе захтев за повраћај новца, запослени има задатак да спроведе активност прегледа описа захтева који је клијент упутио.

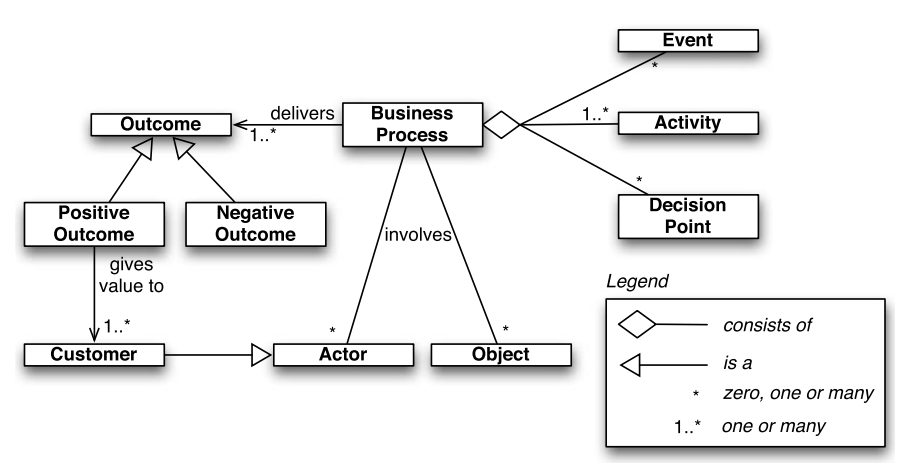
Када је потребно да човек мануелно изврши одређени задатак у оквиру активности, најчешће у смислу одређивања садржаја података који му долазе у датој активности, онда се та активност назива задатак (*task*). На нивоу задатака, потребно је одредити који запослени ће извршити конкретан задатак. Одређени задатак се не мора увек доделити конкретном запосленом, већ се може доделити одређеној корисничкој групи која постоји у оквиру фирме.

Поред догађаја и активности, процеси у главном садрже и тачке одлучивања – *decision points*. Тачке одлучивања представљају „раскрсницу“ у оквиру процеса, односно којим путем ће се наставити извршење датог процеса, најчешће у зависности од вредности неког податка. Често вредност податка који одређује даљи ток процеса бива одређена од стране запосленог који извршава одређену активност.

Процес садржи и одређене чиниоце поред претходно наведених фактора. Ови чиниоци могу бити људи, организације, или софтвер који аутоматизовано извршава задатке постављене пред њега. Процеси дакле садрже најчешће предмете пословања којима се одређена фирма бави, односно њихове описе у виду података, а такође се кроз њих могу размењивати документи који су претходно направљени.

Завршетак процеса у може имати један или више исхода. Уколико се ради о процесу као што је захтев за повраћајем новца од стране клијента, може се десити да се процес успешно заврши (гледано са стране клијента) тако што ће му бити повраћен новац. Могућ исход је и да се деси одбијање од стране фирме, тако да новац неће бити враћен клијенту. Међутим, може се десити и да завршетак једног процеса представља услов за започињање неког новог који је условљен претходним. Ово значи, на датом примеру да се клијент може у неком моменту поново пожалити, а компанија ће имати забележен претходни случај жалбе клијента, па на основу њега у неком моменту одредити ток процеса.

На слици 1. је приказана типична структура једног пословног процеса. Као што је већ речено, један процес се може састојати од неколико догађаја, тачака одлучивања, а макар једне активности. Процес унутар себе укључује чиниоце (људи, групе људи, софтвер...), објекте (документа, подаци...), а на излазу даје неки од могућих исхода који даје одговор крајњем клијенту. Овај скуп компонената представља основну структуру која се може применити на многе видове пословања, односно делатности и помоћу које се могу описати сви различити пословни процеси који се одвијају у оквиру ових делатности.



Слика 1. Компоненте пословног процеса [1]

# Софтверске платформе за BPM

Како се види у претходном поглављу, само моделовање пословних процеса је заправо идејно решење како треба пословање једне компаније да се одвија у циљу успешног пословања. Данас постоје бројна софтверска решења за моделовање пословних процеса. У основи, сва ова решења садрже неки алат за дефинисање процеса који ће се одвијати. Њихове генералне карактеристике су на први поглед сличне, али је једно од кључних питања да ли ће одређен систем бити компатибилан са постојећим окружењем. Опште је познато да различити софтверски производи и решења захтевају различиту софтверску и хардверску подршку. Под софтвером се најчешће подразумева оперативни систем, или сервер уколико се ради о веб апликацијама. Свако софтверско решење се базира на одговарајућој технологији односно програмском језику. Фирмама у којима се постојећа решења базирају на *PHP* технологији и *Apache* серверу погодује имплементација BPMрешења које подржава ову врсту технологија. Због овога је битно препознати који тип технологија се тренутно имплементира у оквиру компаније и која BPM платформа је најпогоднија за примену у постојећу инфраструктуру. Такође, BPM платформе се ослањају на одређену релациону базу података у којој се чувају сви подаци који се тичу процеса и других активности. Нека BPM решења подржавају MySQL, нека Oracle, а нека SQL Server базе података. Најчешће се унутар фирми примењује BPM решење које подржава постојећи тип базе података.

Модерна решења BPM платформи се најчешће базирају на сервисно оријентисаној архиви. SOA (*Service Oriented Architecture*) подразумевају концепт развоја сервиса који се могу користити од стране различитих апликација. Ово значи да се пословни процес у оквиру BPM платформе може иницирати, односно контролисати чак из апликација које су независно инсталиране у постојећем окружењу.

Такође, поред алата за моделовање пословног процеса који је потребно одабрати у складу са жељеном BPM платформом, потребно је одабрати језик за моделовање процеса (*modeling language*). Овај језик, слично било ком програмском језику садржи сопствену семантику и синтаксу. Једна од кључних карактеристика BPM алата за моделовање јесте да подржи неки од популарних језика за моделовање процеса, као што су BPMN 2.0, BPQL, BPEL, EPC, UML, XML… Неке BPM платформе подржавају више језика за моделовања, док неке подржавају само један.

Сва модерна BPM решења најчешће садрже и командну таблу – *dashboard*. Овде администратори система могу да виде како се пословни процеси одвијају, као и да извуку одогварајуће анализе. Командне табле имају задатак да омогуће надгледање процеса у реалном времену, као и генерисање извештаја и графикона како би се приказао ток текућих пословних процеса. Командна табла је једна од кључних ствари како би се видела ефикасност обављања пословних процеса, као и како би се размотриле могућности даљег унапређења њихове имплементације и организације.

# Java Enterprise Edition (Java EE)

Као што је већ речено, данас су софтверске компаније развиле бројна BPM решења за дизајнирање односно моделовање пословних процеса. Један од могућих избора је свакако Oracle BPM платформа. Ова платформа омогућава развој велике палете различитих апликација које заједно функционишу и размењују податке кроз њихов централни део – пословни процес. Oracle као светски позната компанија већ поседује доста признатих решења у оквиру рачунарских система и информационих технологија. Једна од најпознатијих релационих база података јесте Oracle база података, на којој се управо базира и њихова BPM архитектура пословних система. Целокупан систем се базира на *Java* програмском језику и пратећим *Java EE* (*Java Enterprise Edition*) веб технологијама.

Да би се ушло у дубља разматрања функционисања овог система неопходно је разумети основе стандардних *Java EE* веб апликација. Код стандардних *Java EE* веб апликација, могуће је одабрати неки од актуелних сервера који подржавају инсталацију оваквих апликација на њих. Као један од најпознатији свакако јесте *Apache Tomcat* сервер, који се вероватно најчешће и помиње у стручној литератури. У основи, сваки сервер (гледано са софтверске стране) поседује одређена конфигурациона подешавања, и језгро које омогућава саму инсталацију веб апликација на њега. *Java EE* технологије су главним делом сконцентрисане на рад са *Java Servlet* класама и *Java Bean* објектима.

## Java Servlet

*Servlet* представља класу која омогућава имплементацију комуникације на серверској страни веб апликација. Под овим се подразумева да се у оквиру ових класа дефинишу одговори сервера на захтев клијената. Класе које се користе су дефинисане унутар стандардних *javax.servlet* и *javax.servlet.http* пакета. Као што је већ познато, када се говори о комуникацији преко *http* протокола, најчешће се комуникација реализује преко два типа захтева који су типа *get* и *post*. Да би се ова два захтева имплементирала у оквиру веб апликација реализованих кроз *Java Servlet* технологије, најчешће се дефинишу класе које наслеђују *HttpServlet* класу.

Када желимо да дефинишемо могућност прихватања неког од захтева са клијентске стране, у оквиру класе која наслеђује *HttpServlet* класу, потребно је преклопити жељене методе (у случају *get* захтева то је *doGet* метода, а у случају *post* захтева *doPost*). Ове две методе примају као улазне аргументе два параметра, типа *HttpServletRequest* и *HttpServletResponse*. Ови параметри служе како би се у случају *HttpServletRequest* параметра извукли евентуални подаци који се шаљу, односно у случају *HttpServeltResponse* припремио одговарајући одговор сервера ка клијентској страни. У најпростијем случају, могуће је формирати одговарајући одговор кроз обичан *String* који представља *html* структуру странице која ће се генерисати на клијентској страни.



Слика 2. Основна структура типичне *Java Servlet* класе [10]

Међутим, *Java Servlet* представља једно од првобитних решења компаније *Oracle* у смислу развоја веб апликација и комуникације клијента и сервера. Такође, када се говори о *Java Servlet* технологији, незаобилазно је споменути и .*jsp* (*Java Servlet Pages*) стране. Ове стране могућност генерисања динамичке структуре странице програмски, са стране сервера. У оквиру ових страна, могуће је писати све наредбе које постоје у стандардном програмском језику *Java*. Поред овога, могуће је мешати *html* елементе који ће бити генерисани на страни и *Java* код. На овај начин се *.jsp* страна генерише на серверској страни и враћа клијенту као чист *html* код, који претраживачи једино разумеју.

*Java Servlet* технологије, као и *.jsp* стране се данас користе на веома великом броју веб апликација иако су једне од првих веб технологија које су развијене. Оне нису нестале развијањем нових веб технологија и *framework* решења других компанија односно програмских језика, а и развијањем самих *Java EE*  технологија. Модерније решење, развијено у оквиру *Java EE* платформе јесу *.jsf* стране, односно *Java Bean* објекти, о којима ће бити речи у наредном поглављу.

## *JSF* странице и *Bean* објектИ

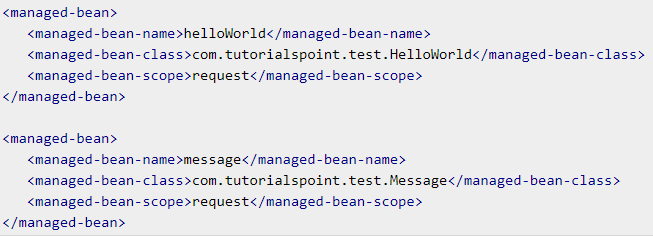
*JSF (Java Server Faces)* странице представљају модерније решење развијено такође у оквиру *Java EE* платформе компаније Oracle*.* Главна разлика у односу на претходне *.jsp* стране јесте што се сада избегава писање *Java* кода у оквиру самих страница, већ се одваја слој логичке обраде података у оквиру *Bean* објеката, а на страници је задатак да обрађене вредности укључи у оквиру своје структуре приликом приказа на клијентској страни.

Генерално, *Java EE* управо уводи појам *bean* – а (у даљем тексту **бин**) када се говори о развоју веб апликациј у оквиру ових технологија. Бин заправо представља објекат за чије инстанцирање је одговоран веб сервер на кога се инсталира веб апликација која има унутар себе декларисане класе које су конфигурисане да буду бин објекти. Бин класе по правилу има атрибуте који су *private* досега видљивости, и сваки од њих има свој пар *geter* и *seter* метода. Има више врста бин објеката који се инстанцирају на серверу:

* *JSF Managed Bean*
* *Enterprise Java Bean (EJB)*
* *CDI Bean (Context and Dependency Injection)*

### JSF Managed Bean

*Managed Bean* представља стандардну *Java* класу која је преко одговарајућих анотација, или чешће преко конфигурационе *faces-config.xml* датотеке подешена да има својства *Managed Bean* објекта. У верзији *JSF* 1.2, било је неопходно регистровање *bean* објеката искључиво у оквиру конфигурационог фајла, док су од верзије 2.0 уведене анотације над класама.



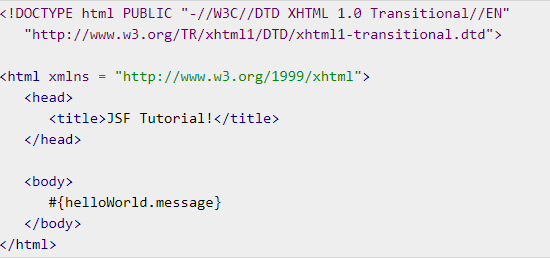
Слика 3. Подешавање *Managed Bean* у оквиру конфигурационе датотеке [11]



Слика 4. Подешавање *Managed Bean* преко анотација [11]

Што се тиче подешавања у оквиру конфигурационе датотеке, ради се о *.xml* подацима где се наводи име *Bean* објекта, класа на основу које ће се инстанцирати, односно његов тип, као и досег (*scope*) важења. Такође, подразумевано, сваки *Bean* има својство *eager* подешено на *false* вредност. Ово својство говори серверу да ли је неопходно да се инстанцира *Bean* на серверској страни пре него што се на њега позове (у оквиру неке странице).

Преко анотација су ова подешавања читљивија, и одмах се виде сва својства једне класе која ће заправо бити репрезентација *bean* објекта. Једно од важних подешавања сваког *Managed Bean* објекта јесте његов досег важења. Овај досег може бити *RequestScoped* – што значи да се *bean* рекреира на сваки *http* захтев упућен са клијентске стране ка серверу. *NoneScoped –* само при првој евалуацији стране се креира, и одмах се уништава. *ViewScoped*  - траје док траје преглед клијента у текућем прозору, при редирекцији на неку другу страну, *bean* се уништава. *SessionScoped* – траје колико год да траје сесија успостављена између клијента и сервера. *ApplicationScoped* – траје док је инсталирана апликација у оквиру које је подешен овај *bean* на сервер.



Слика 5. Пример *.jsf* странице [11]

Што се тиче структуре .*jsf* страница, оне могу садржати стандардане *html* елементе, као и *xml* елементе. Ове технологије се убрајају под *.xhtml* странице, па се могу срести и под овом екстензијом. Најчешће се срећу под екстензијом *.jsf* и представљају структуру *.xml* елемената, који приликом поступа који се назива рендеровање (*render*) генеришу *html* садржај који ће у одговору бити приказан на клијентској страни. У оквиру самог *Java EE* пакета су дефинисани *xml* елементи који ће имати одговарајућу репрезентацију у виду *html* садржаја на клијентској страници.

Неки од типичних елеманта јесу елементи који се тичу комуникације са крајњим корисником, као што су дугмићи – *commandButton*, који морају бити угњеждени у оквиру *form* елемента како би испунили своју функционалност. Дефинисање одговарајућих акција ових дугмића се спроводи кроз атрибуте над њима, који директно указују на одговарајуће методе у оквиру *Managed Bean* објеката на које су повезани. Приликом позива одговарајуће методе, почиње *request* фаза, а *response*, односно одговор сервера се догађа на крају последње линије дате методе. Уколико се дефинише *action* атрибут над *commandButton* компонентом, онда је неопходно да метода врати *String* вредност, која може представљати назив неке друге *.jsf* странице. На овај начин се врши редирекција између страна помоћу ових технологија.

### Enterprise Java Bean (EJB)

У основи, *EJB* бинови се могу поделити на сесијске (*Session beans*) и оне које контролишу извесни догађаји и поруке (*Message – driven beans*). Оба типа имају своје специфичне намене у оквиру *Java EE* архитектуре. Коришћење *EJB* у оквиру апликација се првенствено своди на слој који се бави радом са подацима. Под овим се подразумева најчешће рад са базом података, који се тиче дохватања, брисања и ажурирања података из одговарајућих табела. Поред ових функционалности, могуће је увек писати делове кода који ће имати задатак да обраде неки специфичан захтев.

*EJB* сесијског типа се користи како би се обавила одређена операција, или дохватили одређени подаци. Захтев за обављањем ових задатака се увек упућује са клијентске стране. У литератури која се бави *Java EE* архитектуром се често среће назив пословна логика (*business logic)*, када се говори о томе шта је задатак једног *EJB* објекта. Управо се под овим подразумевају неке од честих операција које један задатак треба да изврши.

Сами сесијски бинови се могу поделити према дужини трајања на: *stateful, stateless* и *singleton* бинове. *Stateful* сесијски бинови аутоматски чувају тренутно стање података у виду *Java* објеката на страни сервера непосредно након инстанцирања бина. У случају да апликација која позива *stateful* бин нема више потребе за његовим коришћењем, он се уклања. Због овога се овај тип бинова и назива *stateful*, јер има могућност чувања свог тренутног стања, привремено. За разлику од њих, *stateless* бинови немају могућност привременог чувања свог стања, односно статуса података на страни сервера. У моменту позива одређене методе из *stateful* бина, он се креира, извршава се дата пословна логика која је дефинисана у оквиру дате методе, и цео бин се уклања, укључујући све референце ка подацима које је садржао. Уопштено говорећи, када се у литератури објектно оријентисаних програмских језика каже да је одређена класа *singleton* типа, ово подразумева да се увек користи само једна инстанца те класе за остварење потребних функционалности. Због овога су и *singleton* бинови добили управо тај назив. То значи да ће *EЈB* који је проглашен за *singleton* увек имати једну инстанцу, која је задужена за опслуживање свих захтева клијената. Животни век ових бинова је практично док год је постављена апликација са датим бином на серверу.

На слици 6. је приказан основни начин декларисања сесијског *EJB* *stateful* типа. Када се генерално говори о сесијским *EJB* биновима, по правилу је неопходно да сваки бин имплементира два интерфејса, један који ће бити реализован као локални, и он се означава са *@Local* анотацијом. Други интерфејс треба да буде анотиран са *@Remote* анотацијом. У оквиру сваког од њих је потребно дефинисати методе које желимо да позивамо кроз њих. Управо се најчешће сами интерфејси користе у оквиру апликација које желе да користе одређене функционалности једног *EJB* бина. Интерфејс који је означен *@Local* анотацијом, користи се у оквиру исте апликације у оквиру које је сам бин декларисан, док се *@Remote* интерфејс користи у свим другим апликацијама. Један од разлога овог концепта јесте зато што је управо могуће ограничити приступ одређеним методама самог *EJB* бина у зависности да ли се приступа из исте апликације, или друге.



Слика 6. Пример декларисања *stateful EJB* бина [12]

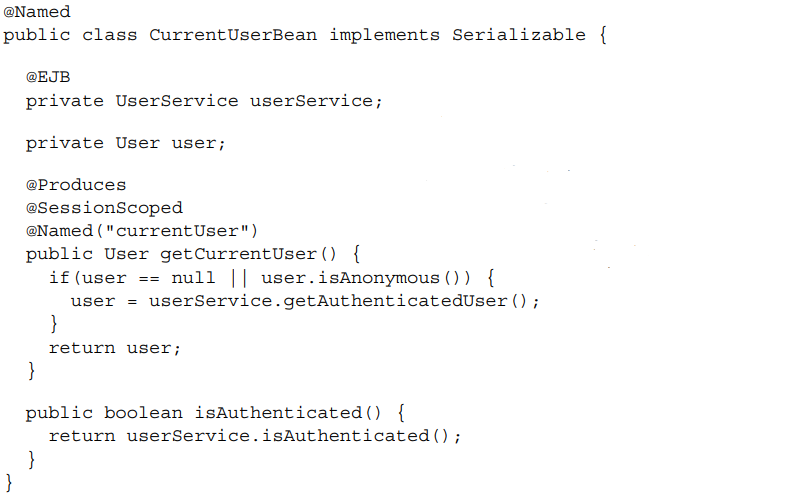
Попут сесијских бинова, тако и *message – driven* бинови имају задатак да изврше одређену пословну логику. Главна разлика у односу на сесијске бинове јесте што се позив метода *message – driven* бинова никада не дешава директно са клијентске стране. Уместо тога, позиви метода ових бинова се догађају када се дешавају одређени догађаји, који онда шаљу поруку преко сервера.

*EJB* бинови представљају једну од кључних компоненти *Java EE* архитектуре управо због тога што имају задатак да се баве самом пословном логиком која заправо диктира сам смисао одређене веб апликације*. EJB* обухвата доста концепата који ће бити обрађени детаљније у следећим деловима рада, на конкретним примерима.

### *CDI* бинови (*Context and Dependency Injection*)

*CDI* бинови практично представљају проширење концепата *EJB* бинова и стандардних *JSF* *Managed* бинова. У оквиру ових бинова, могуће је извршити и одговарајућу пословну логику, али их је такође могуће користити на исти начин као и стандардне *JSF Managed* бинове. *CDI* у погледу рада са подацима не замењује *EJB*, већ управо користи одређени *EJB* који је задужен за дохватање одређених података из базе података и њихово ажурирање. *CDI* бинови такође имају своје векове трајања. За њих, као и за друге претходно наведене бинове, важе анотације *@ApplicationScoped*, *@RequestScoped, @SessionScoped*, међутим поред њих садрже још две могуће опције - *@Dependent* и *@ConversationScoped*.

*ConversationScoped* могућност је уведена када је неопходно да животни век бина траје док има више узастопних захтева са клијентске стране, како би се очували подаци који се тренутно чувају у оквиру њега. *Dependent* опција је подразумевана, и најчешће се и користи, када се врши *Inject* операција овог бина. Да би се постигло његово аутоматско креирање, најчешће се у оквиру неког другог бина наведе анотација *@Inject* над атрибутом типа класе које је *CDI* бин који желимо да користимо на том месту, а сервер изврши инстанцирање објекта ове класе који ће нам онда бити на располагању.



Слика 7. Имплементација *CDI* бина [12]

На слици 7 је приказана једна типична имплементација *CDI* бина који се може инстанцирати преко *@Inject* анотације у оквиру неког другог бина, а такође захваљујући *@Named* анотацији над самом класом, могуће га је користи на начин на који се користе и стандардни *JSF* бинови, директно се реферишући на њега у оквиру неке стране. Овде је приказан такође начин позива методе *getAuthenticatedUser* из *UserService EJB* објекта. Као што се инстанцирају *CDI* бинови преко *@Inject* анотације, на исти начин је могуће постичи инстанцирање *EJB* винова помоћу анотације *@EJB*. На овај начин се *CDI* бинови могу користити као индиректни посредници до података преко *EJB* бинова који се инстанцирају у оквиру њих, а њих саме је могуће користити у другим биновима, као и позивати њихове функционалности директно у оквиру *JSF* страница.

# Закључак

*Hadoop* и његови алати захваљујући својим особинама омогућавају одличан начин за складиштење и обраду великих количина података. Већ су развијена бројна решења на основу *Hadoop* система, а у овом раду је приказана једна могућа варијанта за употребу овог система. Сама анализа сентимената представља веома широку грану за истраживање. Битно је имати довољно добре библиотеке и правилно их употребити у оквиру *MapReduce* функционалности која је обезбеђена у *Hadoop* систему. Могућа надоградња решења приказаног у овом раду би се могла огледати у додавању *HBase* базе података за складиштење резултата анализе, као и у проширивању броја чворова, односно другачијег конфигурисања самог *HDFS* система. Програмски језик *Java* има огромну заједницу програмера који свакодневно развијају нове библиотеке које пружају начине за интеграцију са разним технологијама у области информационих технологија, па тако и *Hadoop* алата. Због тога што је *Hadoop* отвореног кода, постоји читава заједница његових програмера који стално раде на његовом усавршавању и надоградњи постојећих верзија његовог екосистема. Како се количина података свакодневно увећава, потреба за *Hadoop* системом и другим алатима за брзо и ефикасно обрађивање података ће сигурно све више расти у будућности.

# Литература

[1] M. Dumas, M. La Rosa, J. Mendling, H. A. Reijers „Fundamentals of Business Process Management“, 2018.

[2] <https://www.scirp.org/journal/paperinformation.aspx?paperid=88720>, приступано 20.05.2021

[3] D. Panda, R. Rahman, R. Cuprak, M. Remijan „EJB 3 in action“, 2014.

[4] <https://hadoop.apache.org/docs>, приступано 20.05.2020.

[5] <https://www.dezyre.com/hadoop-tutorial/hadoop-mapreduce-tutorial->, приступано 20.05.2020.

[6] V.K. Vavilapalli, A. C. Murthy... „Apache Hadoop YARN: yer another resource negotiator“, 2013.

[7] D. Šušak, Z. Đurić, „Analiza Apache Hadoop – a i mogućnosti upotrebe“, 2019.

[8] <https://www.edureka.co/blog/hadoop-ecosystem>, приступано 25.05.2020.

[9] <https://beyondcorner.com/learn-apache-hbase/facebook-messenger-case-study-with-apache-hbase/>, приступано 25.05.2020.

[10] <https://www.tutorialspoint.com/servlets/index.htm>, приступано 20.05.2021.

[11] <https://www.tutorialspoint.com/jsf>, приступано 25.05.2021.

[12] <https://www.tutorialspoint.com/ejb/> , приступано 25.05.2021.

[13] <https://docs.microsoft.com/bs-latn-ba/azure/hdinsight/hdinsight-hadoop-manage-ambari>, приступано 25.05.2020.

[14] S. Alenezi, S. Mesbah, „Big Data Spatial Analytics in Social Networks using Hadoop“, 2015

[15] J. Lin, D. Ryaboy, „Scaling Big Data Mining Infrastructure: The Twitter Experience“, 2013

[16] A. Jain, V. Bhatnagar, „Crime Data Analysis Using Pig with Hadoop“, 2015

[17] L. J. Sheela, „A Review of Sentiment Analysis in Twitter Data Using Hadoop“, 2016

[18] <https://developer.twitter.com/en/docs/tweets/search/api-reference/get-search-tweets>, приступано 25.05.2020.