Naučni članci / Scientific articles

Vrsta rada: Originalni naučni rad
Primljen: 23. 12. 2021.
Prihvaćen: 21. 01. 2022.
UDC:

Projektovanje i implementacija veb-aplikacije „Podrška u obrazovanju" korišćenjem Spring i Hibernate frameworka

Vladimir Bošković [1] , Svetlana Jevremović[1]

[1] ITS – Information Technology School, Zemun, Beograd, Srbija
* vladimir2417@its.edu.rs, svetlana.jevremovic@its.edu.rs

Sažetak: U ovom radu prikazani su projektovanje i implementacija veb-aplikacije „Podrška u obrazovanju" korišćenjem Spring i Hibernate frameworka. Cilj rada bio je da istraži mogućnosti projektovanja i implementacije ove veb-aplikacije, koja bi korisnicima omogućila laku pretragu i odabir obrazovnih ustanova, a ustanovama bi omogućila da lako komuniciraju sa korisnicima koji su zainteresovani za njih. Ovakva aplikacija bi omogućila korisnicima da se informišu, isplaniraju i na vreme obezbede svoje mesto u obrazovnoj ustanovi uz mogućnost određenih beneficija. Obrazovnim ustanovama bi se olakšalo poslovanje kroz efikasnu i jednostavnu evidenciju pretplaćenih korisnika, pregled zahteva za obilazak obrazovnih ustanova i slanje promotivnih poruka korisnicima. Tema je izabrana zbog nedostatka sličnih veb-aplikacija na našem tržištu. Kao metoda razvoja ove veb-aplikacije korišćena je Larmanova metoda kroz sve faze razvoja.

Ključne reči: veb-aplikacija; Spring; Hibernate; Larmanova metoda

1. Uvod

Veb-aplikacija „Podrška u obrazovanju" bazirana je na aktuelnim Java veb-tehnologijama, kao što su Spring i Hibernate. Dakle, za izradu ove aplikacije korišćen je Spring framework, uključujući Servlete, Maven, Hibernate i Thymeleaf sa serverske strane. Sa klijentske strane korišćeni su HTML, CSS, JS, i biblioteke JQuery i Bootstrap. Kao razvojno okruženje korišćen je IntelliJ IDEA Ultimate, dok su za bazu podataka korišćeni Servleti.
U prvom delu rada prikazan je Spring framework, sa posebnim naglaskom na koncepte Spring MVC modula, koji je korišćen za razvoj aplikacije „Podrška u obrazovanju". Drugi deo rada posvećen je problematici objektno-relacionog preslikavanja i perzistenciji podataka korišćenjem Hibernate ORM (object–relational mapping) alata. Centralni deo rada posvećen je izradi aplikacije „Podrška u obrazovanju".

U razvoju aplikacije korišćena je Larmanova metoda razvoja softvera kroz faze: specifikacije, analize, projektovanja, implementacije i testiranja [10]. Larmanova metoda je bazirana na iterativnom i inkrementalnom modelu životnog ciklusa softvera, predstavljena je slučajevima korišćenja i koristi objektno orijentisanu metodu projektovanja gde se koriste UML (Unified Modeling Language) dijagrami. Okrenuta je programerima, jer je prednost data fazama analize i projektovanja, a jedinstvena je po pravljenju ugovora za sistemske operacije [8].

2. Primenjene tehnologije

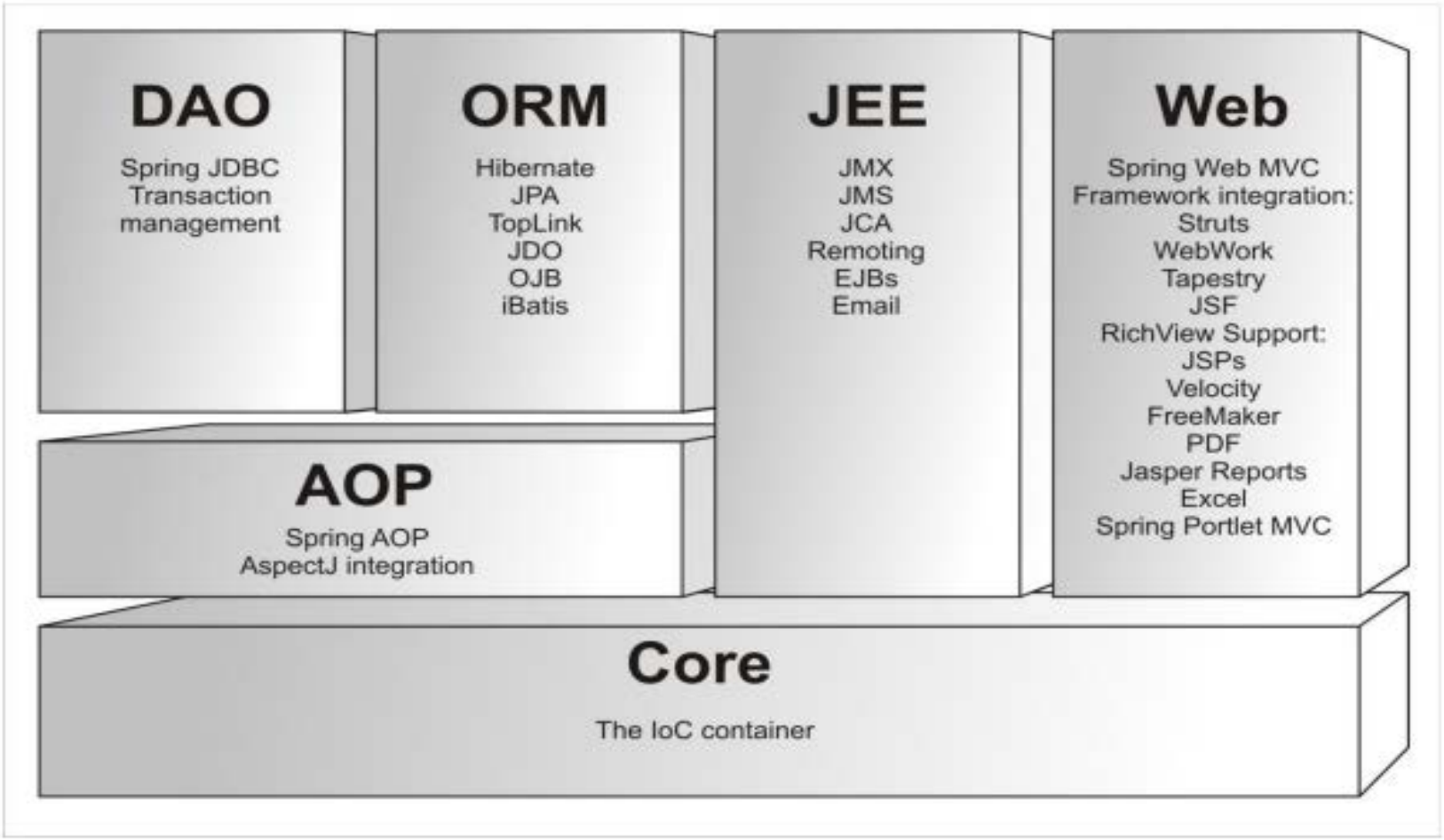
Pre analize dokumentacije o projektovanju i implementaciji veb-aplikacije „Podrška u obrazovanju" analizirane su tehnologije korišćene u toku faza razvoja aplikacije. Najviše pažnje posvećeno je osobinama i rešenjima koja Spring framework nudi u kontekstu razvoja veb-aplikacija.

2.1. Spring framework

Jedna od najbitnijih karakteristika Spring frameworka jeste njegova modularnost, koja omogućava da se prilikom projektovanja aplikacije ne koristi celokupan framework, već samo oni moduli koji su potrebni u specifičnoj situaciji. Modularan pristup omogućava da se pomoću Spring frameworka razviju Java aplikacije različitog nivoa složenosti. U zavisnosi od zahteva aplikacije, moguće je koristiti bilo koji Spring modul nezavisno od drugih modula. Na primer, može da se koristi samo Springov osnovni (core) kontejner za upravljanje slojem poslovne logike aplikacije, dok se drugi delovi aplikacije mogu realizovati korišćenjem neke druge tehnologije. Pored modularnosti, Spring omogućava i integraciju sa velikim brojem tehnologija.

Arhitektura Spring frameworka obuhvata sledećih šest modula i prikazana je na slici 1 [5]:

- Core (osnovni, jezgro);
- AOP (aspect-oriented programming);
- DAO (Data Access Objects);
- ORM (object–relational mapping);
- JEE (Java Enterprise Edition);
- Web.



Slika 1. Moduli Spring frameworka [5]

Core modul je centralni modul Spring frameworka. U okviru Core modula realizovan je Inversion of Control (IoC), odnosno dependency injection (DI) mehanizam. Realizacija DI mehanizma u Core modulu ostvaruje se preko interfejsa BeanFactory, tj. preko implementacija ovog interfejsa. AOP modul obezbeđuje podršku za aspektno orijentisano programiranje, koje omogućava da se na jednostavan način razdvoje komponente sistema, implementirajući funkcionalnosti koje su logički odvojene. DAO modul obezbeđuje JDBC (Java Database Connectivity) sloj koji uklanja potrebu za klasičnim pristupanjem bazama podataka korišćenjem JDBC upravljačkih programa. Naime, korišćenje JDBC-a podrazumeva pisanje koda za pribavljanje konekcije, formiranje naredbi (statement), obradu rezultata i zatvaranje konekcije, dok se korišćenjem Spring JDBC frameworka celokupan posao apstrahuje i pojednostavljuje. ORM modul obezbeđuje sloj za integraciju Spring frameworka sa popularnim ORM alatima.

Pored Hibernate frameworka, čija je integracija sa Spring frameworkom i korišćena u ovom radu, Spring pruža mogućnost integracije sa sledećim ORM alatima: iBatis SQL Maps, JDO, Apache OJB, Oracle TopLink. ORM modul, pored integracije sa navedenim ORM alatima, pruža i mogućnost korišćenja deklarativnog upravljanja transakcijama. JEE modul omogućava integraciju Spring frameworka i EJB (Enterprise Java Bean) komponenti, kao i integraciju sa JMS (Java Message Service) komponentama, što omogućava asinhrono kreiranje, slanje i primanje poruka. Web modul sadrži kompletnu implementaciju MVC4 okvira koji se koristi za razvoj veb-aplikacija. Spring Web MVC implementacija obezbeđuje potpuno razdvajanje poslovne logike aplikacije od njenog prezentacionog sloja, omogućavajući uz to i korišćenje svih IoC osobina okvira prilikom razvoja veb-aplikacije.

2.1.1. Dependency injection

Spring IoC kontejner zasniva se na dependency injection (DI) mehanizmu. Sam naziv (u slobodnom prevodu: ubrizgavanje zavisnosti) dosta govori o odnosu između aplikacije i kontejnera, kao i o načinu na koji se razrešavaju zavisnosti između komponenti aplikacije. Aplikacija može da se posmatra kroz skup komponenti koje na određeni način sarađuju, zavise jedna od druge u toku izvršenja aplikacije. Kod Java aplikacija, ove komponente su pojavljivanja Java klasa, odnosno objekti. Poslovni objekti aplikacije kojom upravlja Spring IoC kontejner nisu zaduženi za pribavljanje resursa i kreiranje drugih objekata sa kojima sarađuju i od kojih zavise. Umesto njih, kreiranje i konfigurisanje njihovih zavisnosti vrši kontejner, što omogućava da se prilikom projektovanja poslovnih klasa pažnja fokusira na poslovnu logiku koju metode klase treba da izvrše.

Postoje tri osnovne varijante DI mehanizma koje podržava Spring kontejner:

» setter injection;
» constructor injection;
» method injection.

2.1.2. Spring kontejner

Dependency injection mehanizam predstavlja suštinu Spring frameworka, a sama implementacija ovog mehanizma realizovana je kroz dva interfejsa koji su srž Spring IoC kontejnera:

» org.springframework.beans.factory.BeanFactory
» org.springframework.context.ApplicationContext

BeanFactory je osnovni interfejs Spring kontejnera koji omogućava konfigurisanje i povezivanje beanova koristeći DI mehanizme. Kada se interfejs BeanFactory kreira, Spring framework vrši validaciju konfiguracije beanova. Svaki singleton bean se kreira prilikom pokretanja frameworka, dok se ostali beanovi kreiraju na zahtev korisnika. Pored toga, interfejs BeanFactory obezbeđuje i neke mehanizme za upravljanje životnim ciklusom beanova, pri čemu se to odnosi samo na singleton beanove, jer kod prototype beanova kontejner nakon kreiranja beana gubi kontrolu nad njim.

ApplicationContext interfejs nasleđuje BeanFactory interfejs. Njegove implementacije takođe omogućavaju kreiranje i upravljanje životnim ciklusom beanova, ali pored toga nude podršku za integraciju sa Springovim AOP modulom, upravljanje resursima za poruke (message resource support), kao i propagaciju događaja (application events propagation). Pored toga, obezbeđuje i posebne kontekste aplikacionog sloja kao što je interfejs WebApplicationContext, koji se koristi u veb-aplikacijama.

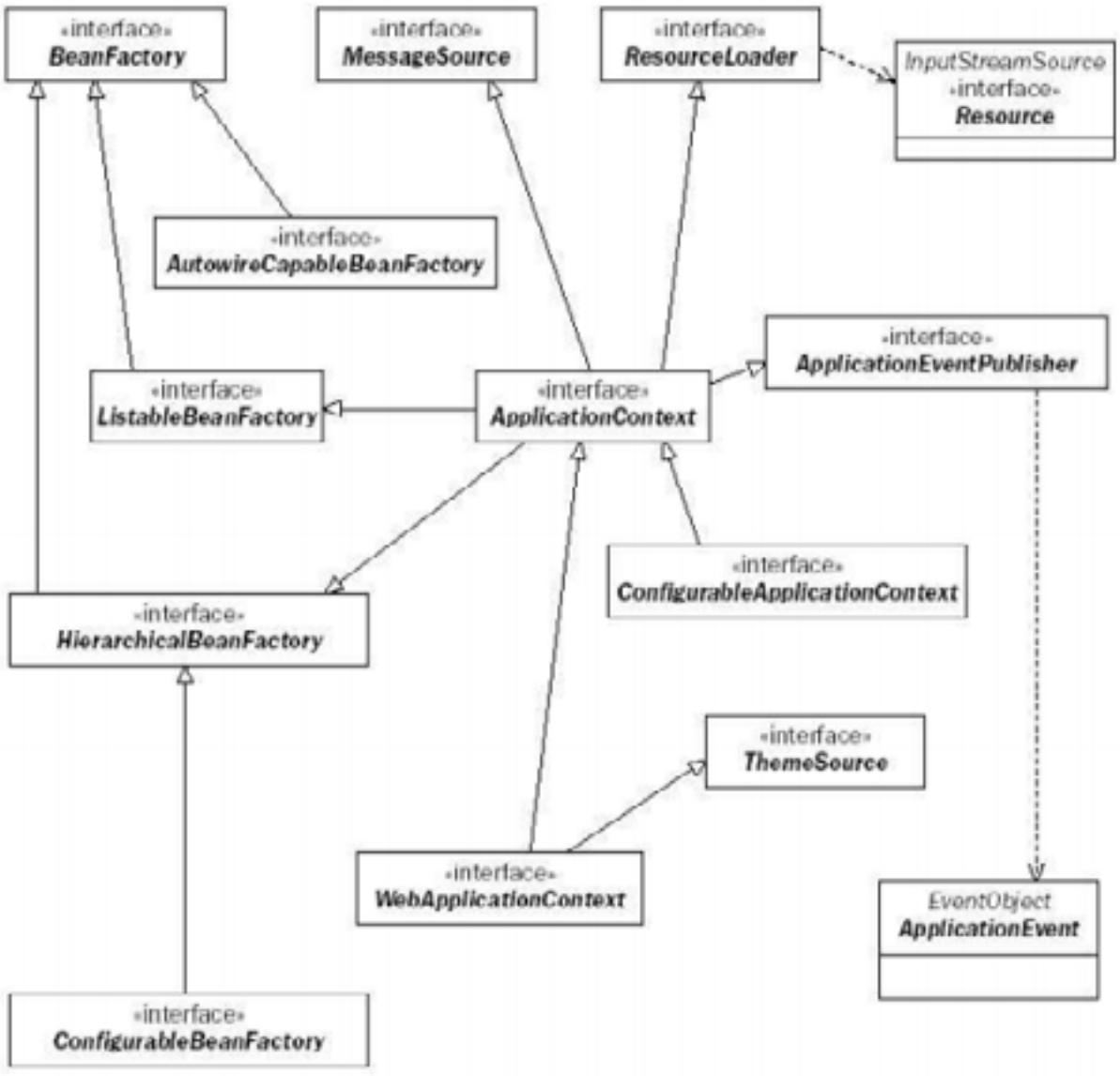
Dakle, interfejs BeanFactory obezbeđuje konfigurisanje frameworka, kao i njegove osnovne funkcionalnosti, dok interfejs ApplicationContext dodaje frameworku nove, složenije funkcionalnosti, pri čemu interfejs ApplicationContext predstavlja kompletan nadskup interfejsa BeanFactory, odnosno omogućava sve što i BeanFactory interfejs. Odnos ova dva interfejsa prikazan je na slici 2.

2.1.3. Rad sa Java Bean komponentama

Bean je softverska komponenta sa mogućnošću ponovnog korišćenja, kojom se može vizuelno manipulisati u odgovarajućem alatu [4]. U kontekstu aplikacija koje se baziraju na Spring frameworku, termin bean se koristi za sve objekte koje kreira Spring kontejner i kojima kontejner upravlja [5]. Prilikom definisanja beanova, konfiguraciona datoteka se sastoji od jednog (korenog) beans elementa i jednog ili više bean elemenata. Validacija ove XML datoteke vrši se u odnosu na XML DTD datoteku spring-beans.dtd, koja u potpunosti opisuje sve validne atribute i elemente koje konfiguraciona datoteka može da sadrži.

Ono što se u najvećem broju slučajeva prvo definiše jeste identifikator beana. Bitno je naglasiti da nije neophodno definisati identifikator, jer se u tom slučaju definisani bean tretira kao anoniman bean. Ime beana može da se definiše korišćenjem name ili id atributa bean elementa.

Preporuka je da se prilikom definisanja imena beana koristi id atribut, jer je on XML IDREF tipa, tako da u slučaju da ga drugi beanovi referenciraju, XML parser može odrediti da li je referenca validna. IDREF tip poseduje ograničenja zbog kojih nije pogodan za korišćenje u određenim situacijama.



Slika 2. Odnos BeanFactory i ApplicationContext interfejsa [5]

Najčešće primenjivan mehanizam kreiranja beanova jeste pomoću konstruktora beana. Prilikom definisanja beana, u okviru class atributa navodi se ime klase beana. U trenutku kada kontejneru zatreba novo pojavljivanje ovog beana, kontejner će interno izvršiti operaciju koja je ekvivalent pozivu new operatora u Java kodu.

Kreiranje beana moguće je izvršiti i pomoću statičkog factory metoda. U tom slučaju potrebno je definisati klasu čija je uloga da učauri proces kreiranja beana unutar statičke metode. Zatim se, koristeći class atribut, definiše ovaj bean, a factory-method atributom se specificira koja metoda beana je zadužena za kreiranje beana.

Sledeći pristup u kreiranju beanova je korišćenje nestatičke (instance factory) metode. U tom slučaju za kreiranje beana se koristi metoda nekog drugog beana koji je kontejner već kreirao.

2.1.4. Koncepti Spring Web MVC frameworka

MVC uzor predstavlja uzor arhitekture i sastoji se od tri ključne komponenete: Model (Model), Pogled (View) i Kontroler (Controller) [6]. Model je komponenta koja sadrži strukturu poslovnog sistema i njene operacije, odnosno, sadrži podatke i operacije za obradu podataka. View komponenta obezbeđuje korisnički interfejs preko koga korisnik komunicira sa sistemom. Takođe, on šalje korisniku izveštaje koji se dobijaju iz modela. Controller je komponenta koja je zadužena da upravlja izvršavanjem sistemskih operacija. Ona prihvata zahtev od klijenta i nakon toga poziva operaciju koja je definisana u modelu i kontroliše njeno izvršavanje.

Arhitektura Spring Web MVC frameworka podrazumeva postojanje kontroler servleta, kao centralne ulazne tačke za sve dolazeće zahteve. Ova komponenta u Spring MVC frameworku realizovana je preko klase DispatcherServlet. Sloj poslovne logike aplikacije realizovan je preko kontroler komponente. U prezentacionom delu, Spring MVC podržava različite tehnologije prikaza. Kao najznačajnija karakteristika prezentacionog sloja ističe se mogućnost ostvarenja potpune nezavisnosti između poslovne logike i konkretne prezentacione tehnologije.

Centralna komponenta Spring Web MVC frameworka jeste klasa DispatcherServlet, koja predstavlja glavnu ulaznu tačku za svaki dolazeći zahtev koji je adresiran na Spring Web MVC aplikaciju. Ova klasa je u potpunosti integrisana u Spring IoC kontejner, što omogućava korišćenje svih osobina koje poseduje Spring framework.

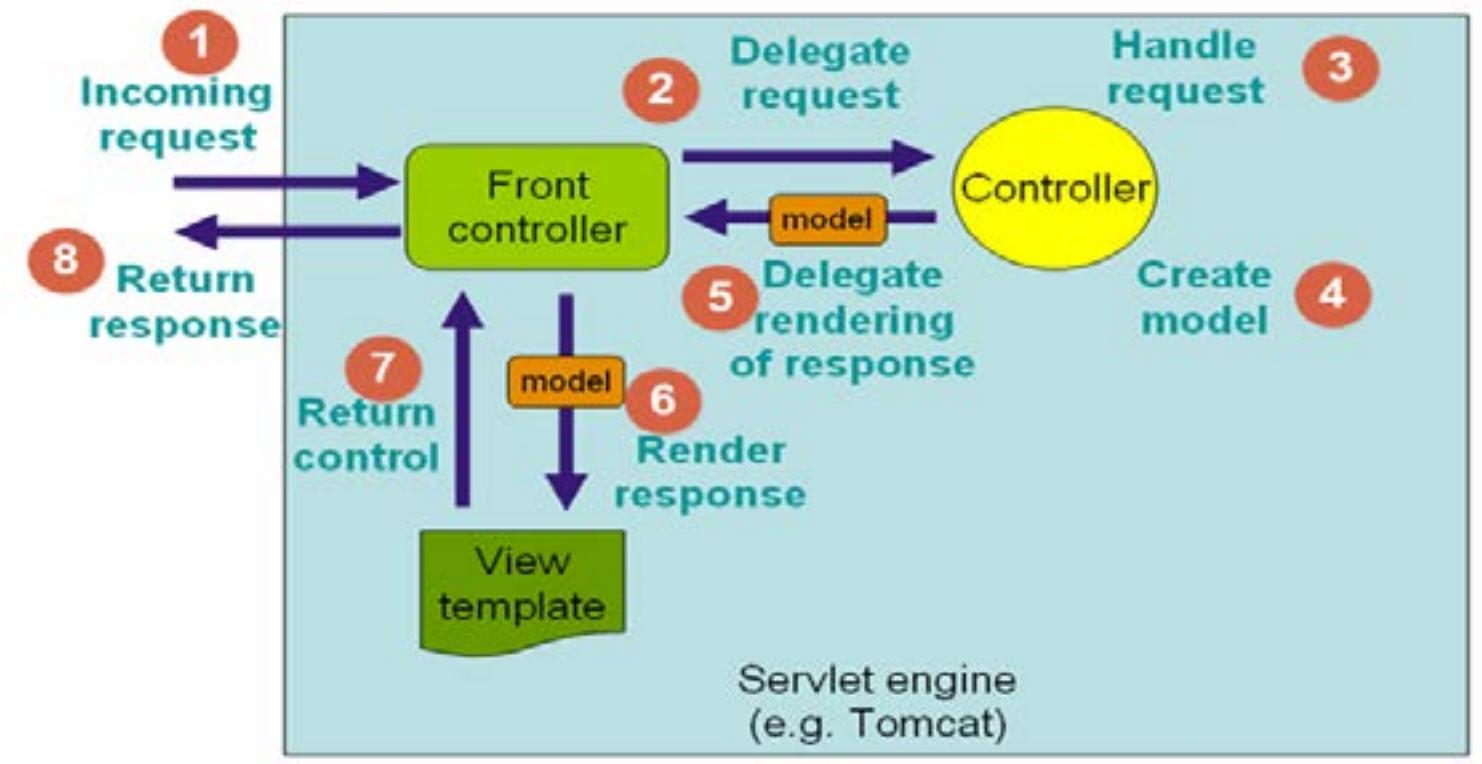
Slika 3 prikazuje konceptualni model obrade zahteva korišćenjem DispatcherServlet klase kao centralnog kontrolera. Kao što se sa slike vidi, centralni kontroler je ulazna tačka za svaki dolazeći zahtev (request). Centralni kontroler po prijemu zahteva delegira zahtev odgovarajućem kontroleru koji je zadužen za obradu zahteva. Kontroler kreira model i obrađuje zahtev, zatim predaje centralnom kontoleru model i logičko ime pogleda. Model sadrži atribute koje pogled treba da prikaže klijentu. Na osnovu logičkog imena pogleda vrši se preslikavanje ka konkretnoj realizaciji pogleda kako bi se izvršila priprema (render) prikaza koji će biti vraćen klijentu kao odgovor (response).

Klasa DispatcherServlet je, zapravo, servlet (izvedena iz klase javax.servlet.http.HttpServlet). Jedini je servlet koji je potrebno deklarisati i konfigurisati u opisivaču rasporeda (deployment descriptor) veb-aplikacije. Pored deklaracije servleta, u opisivaču rasporeda neophodno je definisaiti preslikavanja zahteva ka centralnom kontroleru (klasi DispatcherServlet).

2.2. Hibernate framework

Perzistencija podataka predstavlja jedan od fundamentalnih koncepata u razvoju softverskih sistema. Uopšteno, podaci su perzistentni ukoliko nadžive program koji ih je kreirao. Postoji nekoliko definicija koje se odnose na perzistenciju podataka u kontekstu objektno orijentisanog razvoja softvera [6].

» Objekat je perzistentan ukoliko se može materijalizovati i dematerijalizovati.
» Objekat je perzistentan ukoliko nastavi da postoji i nakon prestanka rada programa koji ga je stvorio (G. Booch).
» Materijalizacija predstavlja proces transformacije slogova baze podataka u objekte programa.
» Dematerijalizacija predstavlja proces transformacije objekta iz programa u slogove baze podataka.
» Perzistentni okvir je skup interfejsa i klasa koji omogućava perzistentnost objektima različitih klasa.



Slika 3. Konceptualni model toka obrade podataka [9]

Najrasprostranjeniji oblik čuvanja podataka u današnjim aplikacijama jeste korišćenje relacionih baza podataka, tako da se kod perzistencije podataka u Java aplikacijama najčešće podrazumeva čuvanje Java objekata u relacionoj bazi podataka. Relacione baze podataka postale su svojevrsni standard u domenu perzistencije podataka.

U objektno orijentisanim aplikacijama perzistencija treba da omogući čuvanje objekata u relacionoj bazi podataka. Pri tome se ne misli samo na čuvanje pojedinačnih objekata, već cele mreže uzajamno povezanih objekata koja reprezentuje određeni objektni model. Pored objekata koji se trajno čuvaju, u objektno orijentisanim aplikacijama postoji i veliki broj tzv. transijentnih objekata. Transijentni objekti su objekti čije je trajanje ograničeno trajanjem aplikacije koja ih je kreirala. Najčešće u ovakvim aplikacijama postoji podsistem koji treba da obezbedi materijalizaciju i dematerijalizaciju perzistentnih objekata, tj. njihovu transformaciju u oblik pogodan za čuvanje u relacionim bazama podataka – relacioni model.

Dakle, perzistentnost u objektno orijentisanim aplikacijama koje koriste relacionu bazu podataka možemo posmatrati kao proces transformacije objektnog modela u relacioni model i obratno.

2.2.1. Asocijacije

Problem asocijacija odnosi se na transformacije veza koje se uspostavljaju između objekata u objektnom modelu i veza koje se ostvaruju između relacija u relacionom modelu. U relacionom modelu veze se ostvaruju korišćenjem spoljnih ključeva, tj. spoljni ključ u referentnoj tabeli predstavlja primarni ključ u referenciranoj tabeli. U objektnom modelu postoji nekoliko vrsta asocijacija: 1-1 (one-to-one), * - * (many-to-many), 1 - * (one-to-many). Ove asocijacije je u relacionom modelu potrebno obezbediti korišćenjem spoljnih ključeva [7].

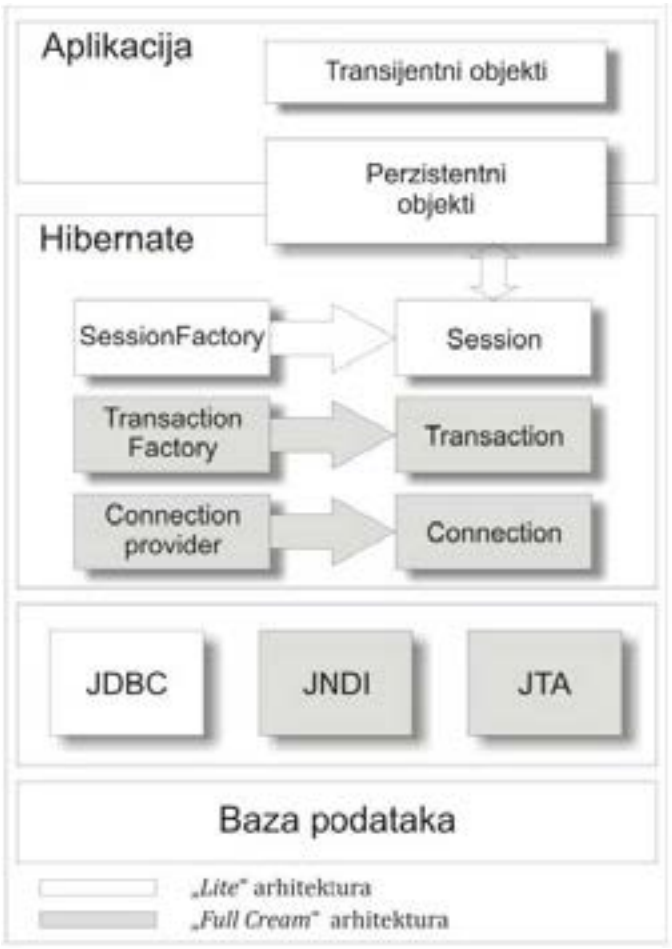
Najčešće se transformacija vrši na taj način što se oba objekta predstavljaju jednom relacijom. Na taj način podaci jednog objekta proširuju se podacima drugog objekta i postaju jedinstvena relacija. Atributi relacije postaju atributi i jednog i drugog objekta.

Many-to-many transformacija se vrši na taj način što se pored relacija koje se prave za svaki od tipova objekata pravi i dodatna agregirajuća relacija koju čine primarni ključevi relacija koje stupaju u vezu. Kada jedan objekat pravi vezu sa više objekata nekog tipa, tada se transformacija vrši tako što se naprave posebne relacije za svaki od tipova objekata, a zatim se primarni ključ objekta koji pravi vezu pamti kod svih objekata sa kojima je on u vezi, tj. primarni ključ relacije na strani one predstavlja se kao spoljni ključ relacije na strani many.

2.2.2. Arhitektura Hibernate frameworka

Hibernate framework je veoma fleksibilan alat koji omogućava nekoliko različitih pristupa prilikom odabira servisa koje okvir pruža, a koji će se koristiti u razvoju aplikacije. Tri ključna servisa (komponente) Hibernate okvira su: upravljanje konekcijama (connection management), upravljanje transakcijama (transaction management) i objektno–relaciono preslikavanje (object–relational mapping).

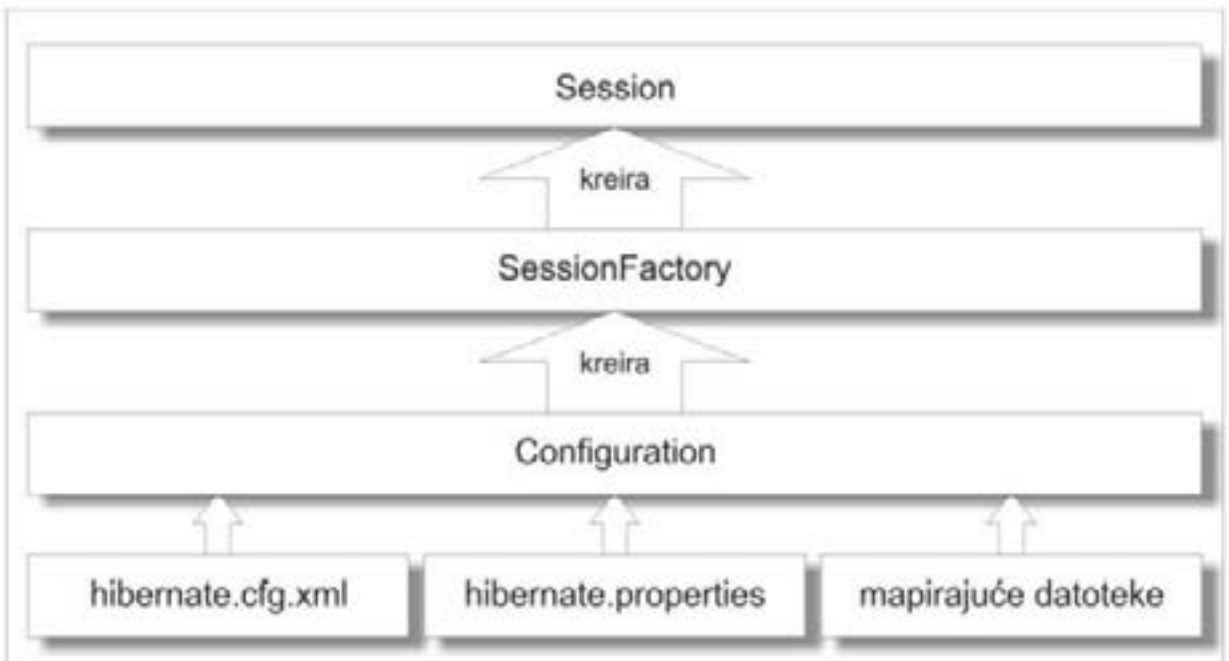
Slika 4 prikazuje dve osnovne arhitekture: „Lite" i „Full cream" arhitekturu. Ovo je klasifikacija u zavisnosti od toga koje komponenete okvira se koriste u razvoju aplikacije. „Lite" arhitektura podrazumeva korišćenje samo komponente za objektno–relaciono preslikavanje, a upravljanje transakcijama i obezbeđivanje JDBC konekcija je prepušteno aplikaciji. „Full Cream" arhitektura podrazumeva korišćenje sve tri komponente.



Slika 4. Arhitektura Hibernate frameworka

2.2.3. Konfigurisanje Hibernate frameworka

Hibernate, kao perzistentni framework, može da komunicira sa velikim brojem različitih SUBP-a i može da se izvršava u različitim okruženjima. Ovakva prilagodljivost Hibernatea različitim SUBP-ovima i okruženjima u kojima se izvršava zahteva različite načine konfigurisanja frameworka. Bez obzira na okruženje i bazu podataka sa kojom aplikacija komunicira, konfigurisanje frameworka može se logički podeliti u dve celine. U prvom delu obezbeđuju se konfiguracioni podaci koji su neophodni frameworku da bi pristupio bazi podataka, a drugi deo čine konfiguracioni podaci kojima se obezbeđuje preslikavanje između perzistentnih klasa aplikacije i odgovarajućih tabela u relacionoj bazi podataka. Centralna klasa, pomoću koje se vrši konfigurisanje i pokretanje Hibernate frameworka, jeste klasa org.hibernate.cfg.Configuration. Prilikom kreiranja, ovoj klasi je potrebno obezbediti konfiguracione podatke, na osnovu kojih Configuration objekat kreira jedno pojavljivanje (singleton) klase



Slika 5. Konfigurisanje Hibernate frameworka

Jedno pojavljivanje SessionFactory klase u suštini predstavlja potpuno konfigurisan Hibernate framework, koji omogućava komunikaciju sa jednom bazom podataka. Pored toga što Configuration objekat kreira jedinstveno (singleton) pojavljivanje SessionFactory klase, stanje ovog objekta nije moguće menjati nakon kreiranja. Klasa SessionFactory je zadužena za kreiranje objekata klase Session prilikom svake interakcije sa bazom podataka.

3. Projektovanje i implementacija veb-aplikacije „Podrška u obrazovanju“

Detaljna priprema dokumentacije usledila je nakon istraživanja adekvatnih tehnologija. Prvo će biti prikazan verbalni opis modela na osnovu kojeg će biti utvrđeni slučajevi korišćenja aktora sistema.

3.1. Specifikacija zahteva

Potrebno je projektovati i implementirati veb-aplikaciju koja bi krajnjim korisnicima pružila pomoć pri odabiru obrazovnih ustanova. Slika 6 prikazuje da postoje tri vrste učesnika: korisnik, administrator i ustanova.

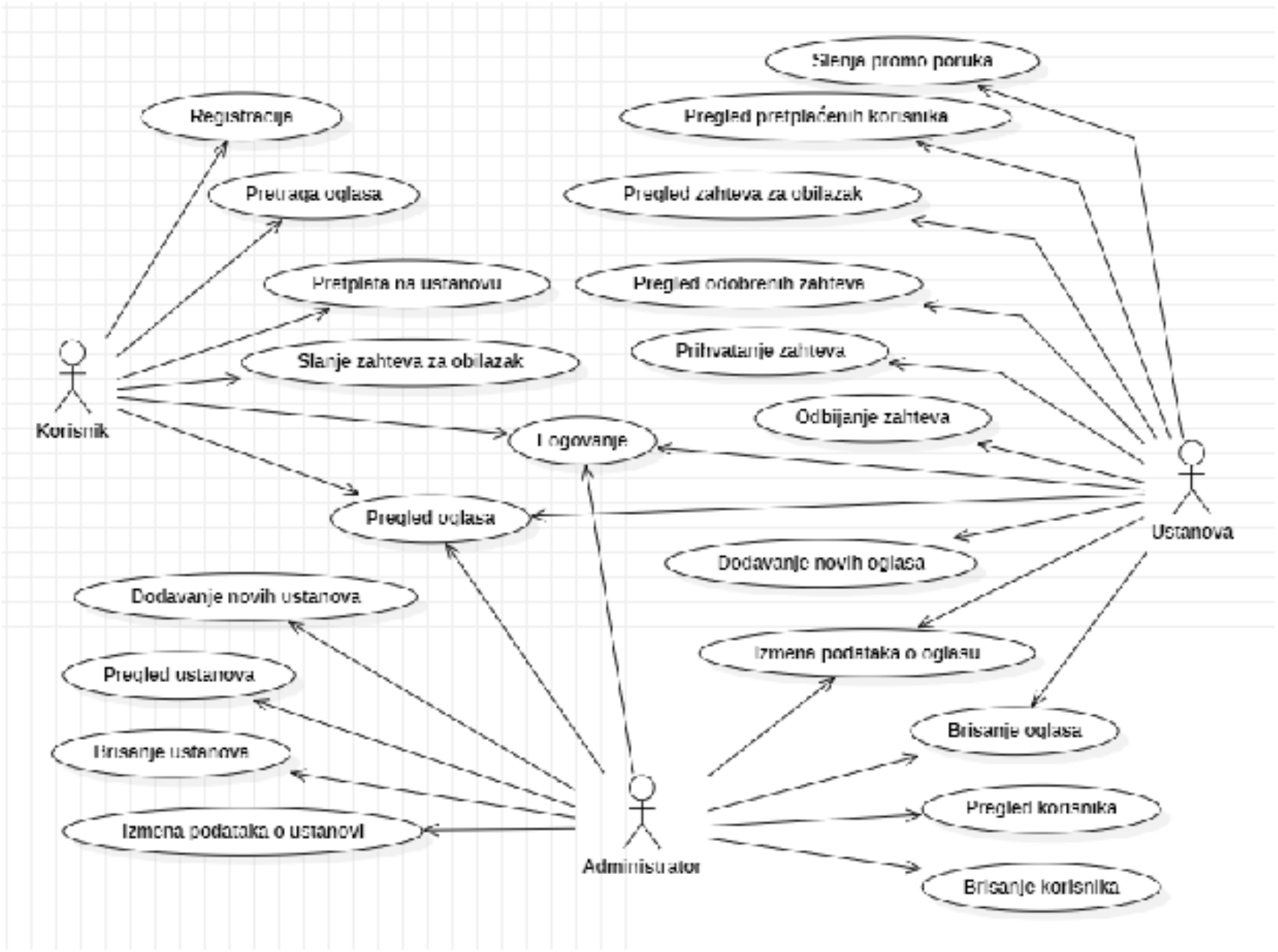
Korisniku sistem treba da omogući registrovanje na sajt, ukoliko već nije registrovan. Nakon registracije, korisniku stiže na mejl poruka dobrodošlice i sada ima mogućnost logovanja na sajt. Nakon logovanja, imao bi mogućnost da pretražuje oglase na osnovu više kriterijuma, koji se biraju na početnoj strani, a zatim se ispisuju svi oglasi za željene parametre. Ukoliko je korisnik zainteresovan za neki od oglasa, klikom na oglas, može videti detalje oglasa, kao i detalje same ustanove. Dodatna mogućnost je besplatna pretplata na određenu ustanovu gde korisnik potvrđuje da želi da prima promotivne mejlove od ustanove. Prilikom svake pretplate korisnik ostvaruje poene. Nakon što skupi određeni broj poena, korisniku se dodeljuje elektronski poklon-vaučer. Poklon-vaučer mu daje mogućnost da ostvari popust na školarinu za određene ustanove. Ako želi da sazna više o obrazovnoj ustanovi, može zakazati obilazak ustanove čija je realizacija moguća nakon potvrde zaposlenog u ustanovi. Ukoliko ustanova ne potvrdi zahtev u roku od 24 sata pred obilazak, zahtev se automatski briše.

Administratoru sistema aplikacija treba da omogući dodavanje novih ustanova, kao i izmenu podataka o ustanovi i brisanje ustanova. Prilikom dodavanja nove ustanove automatski bi se poslao mejl ustanovi sa parametrima za logovanje. Takođe, administrator bi imao mogućnost izmene i brisanja oglasa i korisnika.

Ustanova dobija parametre za logovanje od administratora aplikacije putem mejla. Nakon logovanja, ustanova može da postavlja, menja ili briše oglase. Takođe, ako se korisnik pretplati na ustanovu nakon što je ušao na njen oglas, onda ustanova, kada se uloguje, može videti koji korisnici su se pretplatili i poslati im promo-poruku. Ukoliko je neko od korisnika zahtevao obilazak ustanove, te zahteve ustanova može prihvatiti ili odbiti. Ako ustanova prihvati zahtev, korisniku će se poslati mejl sa obaveštenjem da je zahtev prihvaćen; u suprotnom, dobiće mejl da je zahtev odbijen.

3.1.1. Slučajevi korišćenja

Na osnovu verbalnog modela uočeni su sledeći slučajevi korišćenja: Logovanje, Registracija, Pretraga oglasa, Pregled oglasa, Pretplata na ustanovu, Slanje zahteva za obilazak, Dodavanje novih ustanova, Pregled ustanova, Izmena podataka o ustanovi, Brisanje ustanova, Dodavanje novih oglasa, Izmena podataka o oglasu, Brisanje oglasa, Pregled korisnika, Brisanje korisnika, Pregled pretplaćenih korisnika, Slanje promotivnih poruka, Pregled zahteva za obilazak, Pregled odobrenih zahteva za obilazak, Prihvatanje zahteva za obilazak, Odbijanje zahteva za obilazak (slika 6).



Slika 6. Use-case dijagram svih slučajeva korišćenja

Zbog ograničenog obima rada, u nastavku sledi primer jednog opisa slučaja korišćenja (SK4).

SK4: Pregled oglasa

Naziv: Pregled oglasa

Aktori: Korisnik, Administrator, Ustanova

Učesnici: Korisnik, Administrator, Ustanova i sistem

Preduslovi: Sistem je aktivan, korisnik ulogovan i prikazana je stranica sa oglasima

Osnovni scenario:

1. Korisnik poziva sistem da prikaže detalje oglasa (APSO)
2. Sistem pronalazi detalje oglasa (SO)
3. Sistem prikazuje korisniku detalje oglasa (IA)

Alternativni scenario:

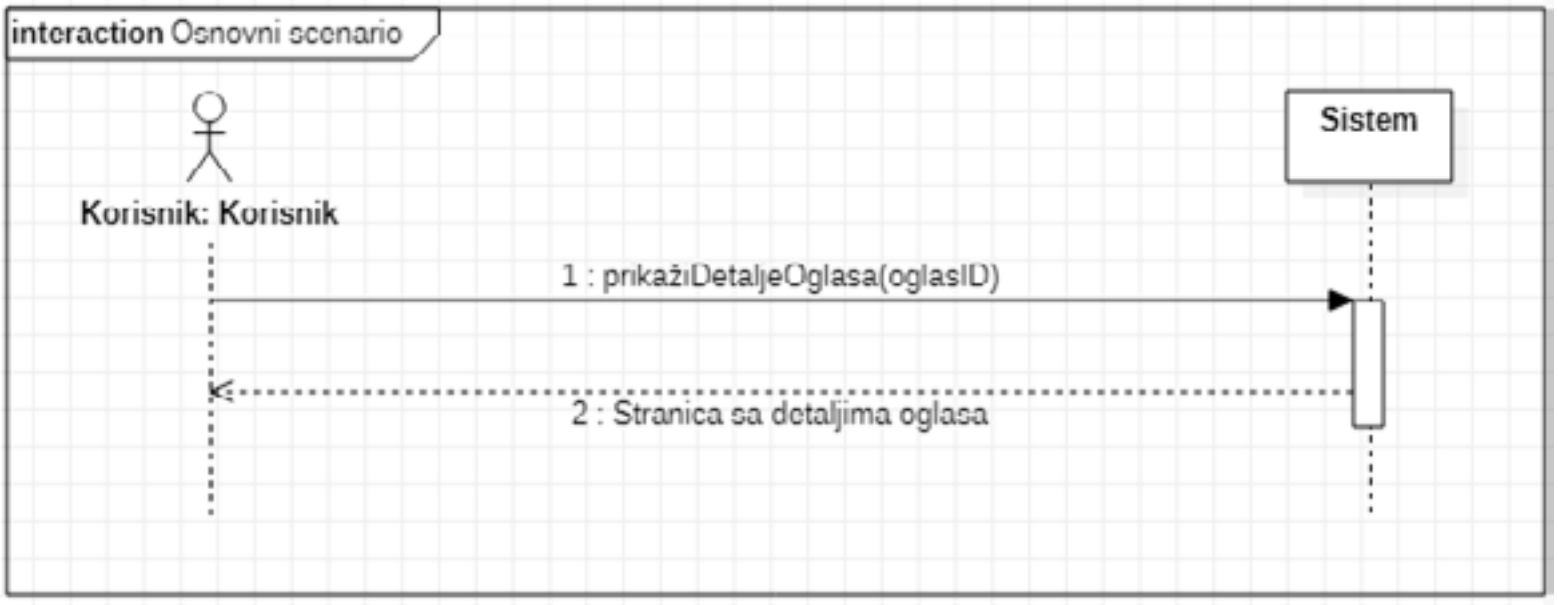
2.1. Sistem ne može da uspostavi vezu sa bazom podataka, prikazuje odgovarajuću poruku (IA)

3.2. Analiza

U fazi analize opisuje se logička struktura i ponašanje softverskog sistema. Struktura softverskog sistema se opisuje pomoću konceptualnog i relacionog modela. Ponašanje sistema se opisuje pomoću dijagrama sekvenci (DSSK), koji se prave za svaki slučaj korišćenja i pomoću ugovora o sistemskim operacijama dobijenih na osnovu prethodnih dijagrama. Sledi primer jednog sistemskog dijagrama sekvenci.

DSSK4: Pregled oglasa

Osnovni scenario:
1. Korisnik poziva sistem da prikaže detalje oglasa (APSO)
2. Sistem prikazuje korisniku detalje oglasa (IA)



Slika 7. DSSK4 – Pregled oglasa

Uvedena je sistemska operacija:
1. prikažiDetaljeOglasa(oglasID)

Za svaku od uočenih sistemskih operacija prave se ugovori, koji opisuju šta operacija radi, a ne i kako, pri čemu se jedan ugovor vezuje za jednu sistemsku operaciju [8]. Sledi primer jednog ugovora (UG18).

Ugovor UG18: prikaziZahteveZaObilazak

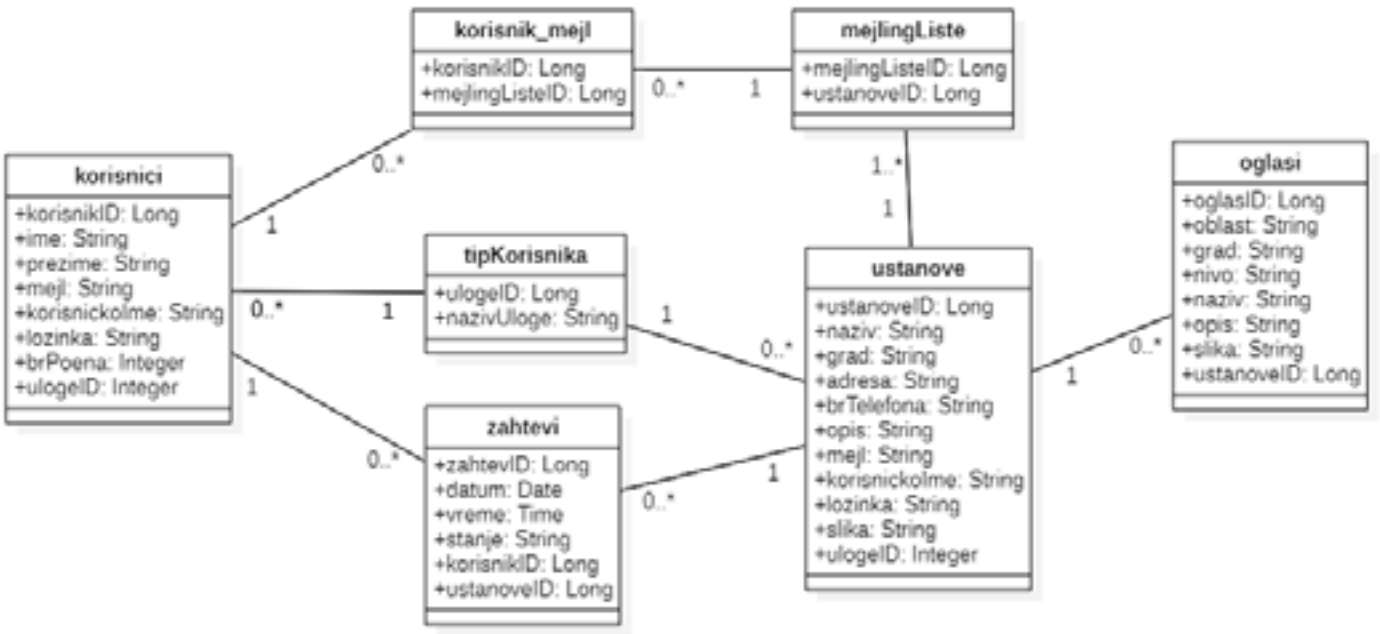
Operacija: prikaziZahteveZaObilazak(ustanoveID)

Veza sa SK: SK18

Preduslovi: Zahtevi za obilazak postoje u bazi podataka

Postuslovi: Prikazana je lista zahteva za obilazak

Nakon definisanja svih ugovora kreira se konceptualni model na osnovu podataka iz funkcionalnog zahteva i slučajeva korišćenja (slika 8).



Slika 8. Konceptualni model

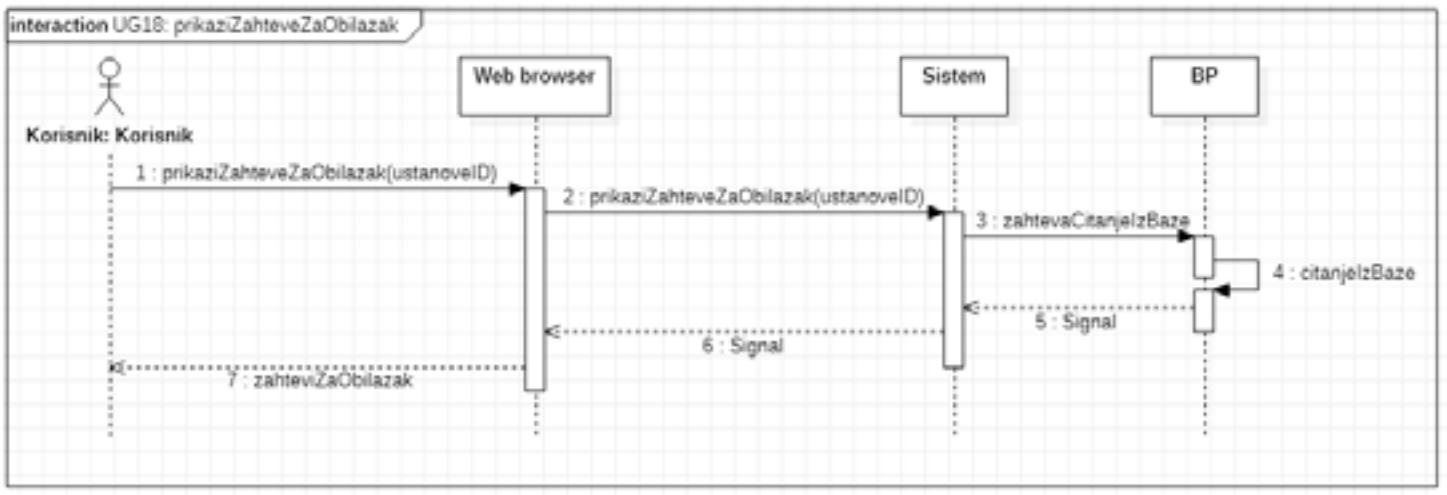
Na osnovu konceptualnog, pravi se i relacioni model, koji predstavlja dalju osnovu za projektovanje baze podataka.
tipKorisnika(ulogeID, nazivUloge)
korisnici(korisnikID, ime, prezime, mejl, korisnickoIme, lozinka, brPoena, ulogeID)
korisnici(ulogeID) references tipKorisnika(ulogeID)
ustanove(ustanoveID, naziv, grad, adresa, brTelefona, opis, mejl, korisnickoIme, lozinka, slika, ulogeID)
ustanove(ulogeID) references tipKorisnika(ulogeID)
oglasi(oglasID, oblast, grad, nivo, naziv, opis, slika, ustanoveID)
oglasi(ustanoveID) references ustanove(ustanoveID)
mejlingListe(mejlingListeID, ustanoveID)
mejlingListe(ustanoveID) references ustanove(ustanoveID)
korisnik_mejl(korisnikID, mejlingListeID)
korisnik_mejl(korisnikID) references korisnici(korisnikID)
korisnik_mejl(mejlingListeID) references mejlingListe(mejlingListeID)
zahtevi(zahtevID, datum, vreme, stanje, korisnikID, ustanoveID)
zahtevi(korisnikID) references korisnici(korisnikID)
zahtevi(ustanoveID) references ustanove(ustanoveID)

3.3. Projektovanje

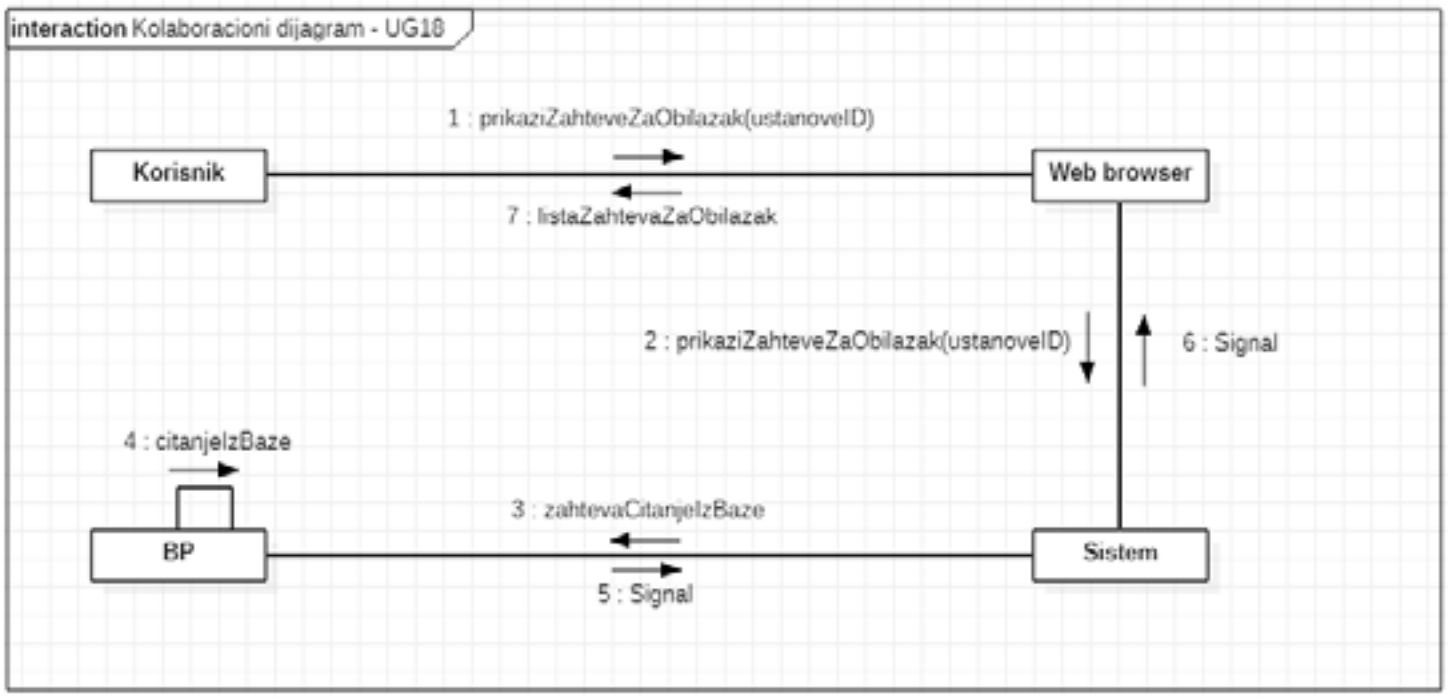
Faza projektovanja opisuje fizičku strukturu i ponašanje softverskog sistema, odnosno arhitekturu sistema. Kao takva obuhvata projektovanje aplikacione logike i projektovanje logičke strukture i ponašanja softverskog sistema.

3.3.1. Projektovanje aplikacione strukture

Ugovor UG18: prikaziZahteveZaObilazak (slika 9 i slika 10)
Operacija: prikaziZahteveZaObilazak(ustanoveID)
Preduslovi: Zahtevi za obilazak postoje u bazi podataka
Postuslovi: Prikazana je lista zahteva za obilazak



Slika 9. Dijagram sekvenci UG18 – PrikaziZahteveZaObilazak



Slika 10. Kolaboracioni dijagram UG18 – PrikaziZahteveZaObilazak

3.3.2 Projektovanje korisničkog interfejsa

Sledi primer definisanja jednog dela korisničkog interfejsa za aplikaciju „Podrška u obrazovanju".

SK4: Pregled oglasa

Preduslovi: Sistem je aktivan, korisnik je ulogovan i prikazana je stranica sa oglasima (slika 11).

Slika 11. Projektovanje korisničkog interfejsa – SK4

Osnovni scenario:

1. Korisnik poziva sistem da prikaže detalje oglasa klikom na
   Opis akcije: Korisnik klikom na oglas pozivva sistem da prikaže detalje o
2. Sistem pronalazi detalje oglasa
3. Sistem prikazuje korisniku detalje oglasa (slika 12)

Kliknite na oglas za
više detalja.

Slika 12. Projektovanje korisničkog interfejsa – SK4

Alternativni scenario:
Sistem ne može da uspostavi vezu sa bazom podataka i prikazuje odgovarajuću poruku (slika 13)

Slika 13. Projektovanje korisničkog interfejsa – SK4

3.4. Implementacija i testiranje

U fazi implementacije vrši se kodiranje sistema primenom određenih tehnologija. Za izradu ove veb-aplikacije korišćene su sledeće tehnologije: Spring framework, uključujući Servlete, Maven, Hibernate i Thymeleaf sa serverske strane. Sa klijentske strane korišćeni su HTML, CSS, JS, i biblioteke JQuery i Bootstrap. U fazi testiranja, tokom razvoja aplikacije „Podrška u obrazovanju", vršeno je testiranje funkcionalnosti same aplikacije, pri čemu su unošeni razni podaci radi utvrđivanja i otklanjanja eventualnih nedostataka.

4. Zaključak

U radu je kreirana veb-aplikacija koja olakšava pronalaženje željenih institucija radi daljeg školovanja zainteresovanih korisnika i time doprinosi samom razvoju obrazovanja u našoj zemlji. Aplikacija svakako može doprineti daljem razvoju nečije karijere. Za izradu aplikacije odabrana je Spring tehnologija, jer ona doprinosi bržem, lakšem i sigurnijem programiranju veb-aplikacija. U daljem radu ispitivale bi se mogućnosti da se veb-aplikacija „Podrška u obrazovanju" obogati novim funkcionalnostima.

Reference

1. Vlajić, S., Savić, D., Stanojević, V., Antović, I., Milić, M. Projektovanje softvera – Napredne Java tehnologije. Beograd: Zlatni Presek, 2008.
2. Jevremović, S. Java programiranje veb aplikacija. Beograd: ITS, 2016.
3. Spring Framework – Reference Documentation, Spring Framework, 2020. Available at: www.springframework.org
4. JavaBeans Documentation, Oracle, 2020. Available at: http://java.sun.com/javase/technologies/desktop/javabeans/docs/spec.html
5. Smeets, B., Ladd, S. Building Spring 2 Enterprise Applications. US: Apress, 2007.
6. Vlajić, S. Projektovanje programa. Beograd: FON, 2004.
7. Hibernate – Reference Documentation, Hibernate, 2020. Available at: http://www.hibernate.org/
8. Anđelić, S. WPF i ASP.NET Framework - projektovanje i implementacija softvera. Beograd: ITS, 2016.
9. Spring Web MVC Framework Flow, 2015. Available at: https://www.onlinetutorialspoint.com/spring/spring-web-mvc-framework.html
10. Vlajić, S. Projektovanje softvera, skripta. Beograd: FON, 2009.