UNIVERZITET U BEOGRADU
FAKULTET ORGANIZACIONIH NAUKA

Završni rad

Tema: Web aplikacija za Biblioteku u Java okruženju

Mentor:
dr Xxxxxx Xxxxx, doc

Student:
Xxxx Xxxxxx xxxx/0x

BEOGRAD, 20xx

1. UVOD

U ovom radu prikazana je uprošćena Larmanova metoda razvoja softvera na studijskom primeru informacionog sistema biblioteke.

Sistem biblioteke razvijen je kao web aplikacija. Za razvoj ove aplikacije korišćene su Java okviri (frameworks) i to Tapestry, Spring i Hibernate okvir.

U prvom delu rada predstavljene su ovi okviri. Oni nude rešenja za mnoge od problema sa kojima se svakodnevno susreću programeri koji se bave razvojem *web* softverskih sistema. Neki od problema koji se javljaju tokom razvoja web softverskih sistema su čuvanje, izmena i prikazivanje velike količine podataka, zatim svakodnevno održavanje tih sistema, i drugi. Softverski sistem biblioteke koristi tronivojsku softversku arhitekturu. Sastoji se iz tri nivoa i to: nivoa korisničkog interfejsa, nivoa aplikacione logike i nivoa podataka.

Na prezentacionom nivou korišćen je *Tapestry Okvir*, na nivou aplikacione logike korišćen je *Spring Okvir*, a na nivou za pristup podacima korišćen je *Hibernate.* Tapestry okvir je izabran zbog svoje jednostavnosti prilikom izrade korisničkog interfejsa i zato što se pomoću ovog okvira jasno odvaja nivo korisničkog interfejsa od nivoa aplikacione logike. Na nivou aplikacione logike izabran je Spring okvir pomoću koga jednostavno manipulišemo domenskim Java klasama. Hibernate okvir je izabran kao jedan od najpopularnijih okvira za pristup podacima. Hibernate okvir sadrži klase za upravljanje skladištenjem i prikazivanjem podataka kojima se veoma jednostavno manipuliše.

U drugom delu rada na primeru sistema biblioteke prikazana je uprošćena Larmanova metoda razvoja softvera. Larmanova metoda se sastoji iz pet faza razvoja softvera: Prikupljanja korisničkih zahteva, Analize, Projektovanja, Implementacije i Testiranja.

2. PROGRAMSKI JEZIK JAVA

Programski jezik *Java* je u potpunosti nezavistan od operativnog sistema na kojem se izvršava, time je obezbeđena prenosivost (portabilnost) aplikacija razvijenih u *Javi*. Napisana aplikacija se može izvršiti na bilo kom operativnom sistemu, pod pretpostavkom da za taj operativni sistem postoji *JRE* (*Java Runtime Environment*), odnosno *JVM* (*Java Virtual Machine*) koja interpretira *Java* naredbe, preslikavajuci ih u naredbe konkretnog operativnog sistema. Takođe, programski jezik je u potpunosti nezavistan od konkretnih sistema za upravljanje bazama podataka (*DBMS - Database Management System*), čime se, uz obezbeđivanje odgovarajućih upravljačkih programa (drajvera) omogućava povezivanje aplikacije sa konkretnim *DBMS* sistemima, i preslikavanje opštih *Java* naredbi u naredbe konkretnog *DBMS*. Ovo je moguće zahvaljujući bibliotekama klasa i interfejsima koji su deo JDBC API-ja (Java Datbase Connectivity Application Programming Interface)

Razlikujemo tri izdanja *Java* platforme. Svako izdanje namenjeno je razvoju specifičnih aplikacija i obezbeđuje izvršno (*runtime*) okruženje za te aplikacije:

1. **Java 2 Standard Edition, J2SE** – obezbeđuje izvršno okruženje za razvoj *desktop* aplikacija i predstavlja jezgro funkcionalnosti za ostala izdanja *Java* platforme.

2. **Java Enterprise Edition, JEE** – predstavlja proširenje *J2SE* izdanja Jave koje podržava razvoj složenih (*enterprise)* aplikacija.

3. **Java 2 Micro Edition, J2ME** – definiše skup izvršnih okruženja za razvoj aplikacija za uređaje sa ograničenom memorijom, kao što su mobilni telefoni, *PDA* uređaji, *TV* uređaji.

2.1. Osnovne karakteristike programskog jezika Java

Niže su navedene osnovne karakteristike programskog jezika Java koje je čine interesantnom i privlačnom za programere:

1) Java je *jednostavnija* za upotrebu u poređenju sa drugim objektno-orijentisanim programskim jezicima. Kao razlog za tu tvrdnju može se uzeti to što Java koristi automatsku memorijsku alokaciju i upotreba smeća (*garbage collector*) za brisanje objekata iz memorije.
2) Java je distribuirana. Java je projektovana da olakša rad u mreži i razvoj distribuiranih aplikacija.
3) Java je prenosiva. Java je platformski nezavisna. Jednom napisan programski kôd može se izvršiti na bilo kom računaru za koji je napisan Java interpreter.
4) Java je interpretirana. Program je potrebno kompajlirati samo jednom. Nakon kompajliranja Java izvršnog koda dobija se Java bytecode. Ovaj bytecode je nezavisan od platforme i interpretira se od strane Java interpretera.
5) Java je pouzdana. Java poseduje više nivoa mera pouzdanosti. Java kompajler pruža nekoliko nivoa dodatnih provera kako bi identifikovao tipske greške i druge

nepravilnosti. Java runtime sistem ponavlja provere koje je izvršio java kompajler i izvršava dodatne provere kako bi proverio izvršni bajtkod (bytecode) java programa.
6) Java ima podršku za rad sa multimedijalnim sadržajem.
7) Java je višenitna. Java podržava izvršavanje više procesa istovremeno korišćenjem niti.[10]

Neke od negativnih karakteristika programskog jezika Java su brzina izvršavanja Java programa i količina memorije koja se alocira za Java program.

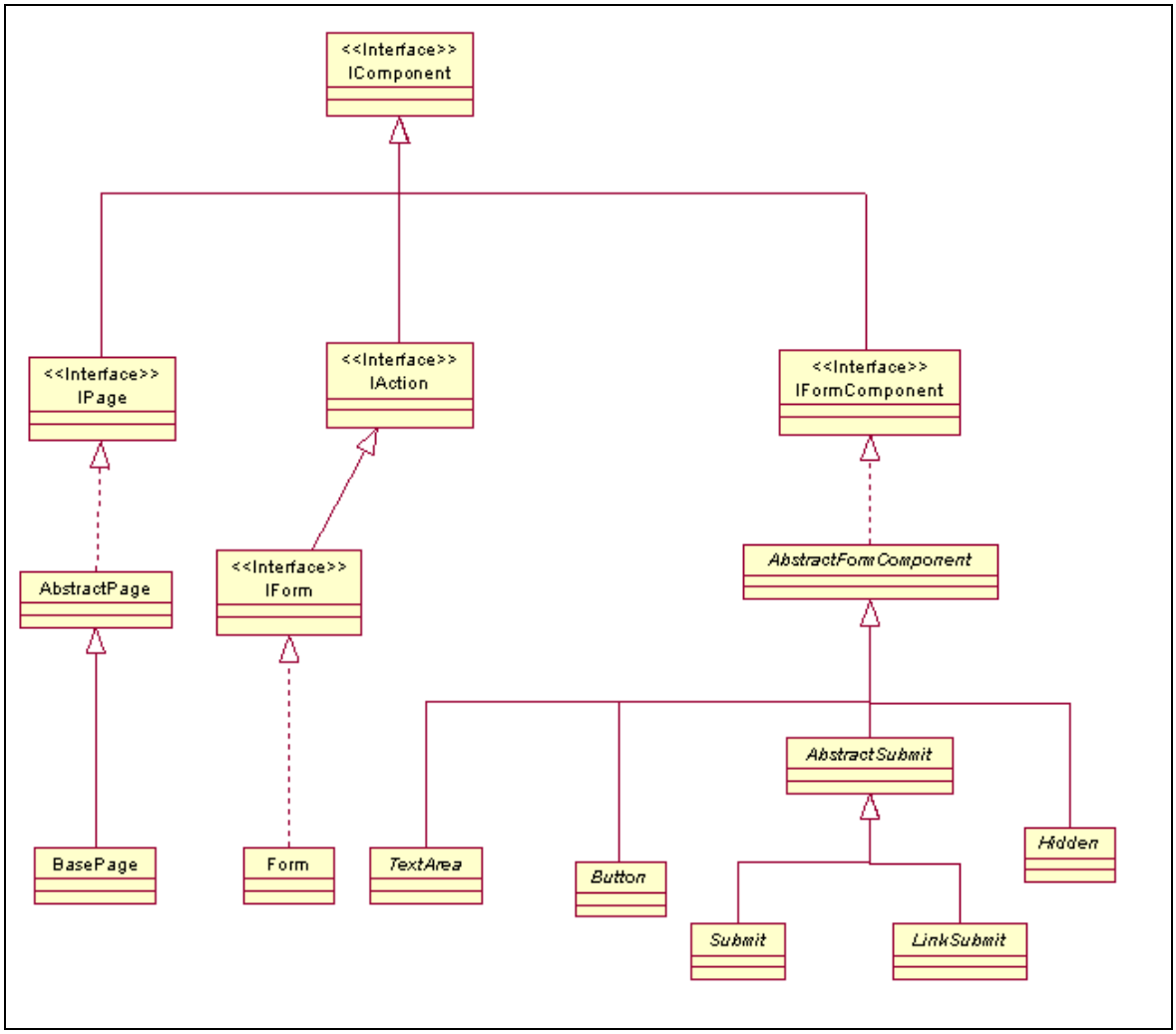
Glavni nedostatak Java programskog jezika je brzina. Ovo je posledica neophodne konverzije Java *bytecode*-a u konkretan mašinski kod za konkretan računar na kome se Java Virtuelna mašina izvršava. Smatra se da Java zauzima više memorije od drugih objektno orijentisanih programskih jezika.[10]

2.2. Tapestry okvir

Tapestry okvir (framework) je okvir za kreiranje dinamičkih i robusnih web aplikacija u Javi. Tapestri je napravljen na konceptu standardnog Java Servlet API –a. Tapestry može da radi u bilo kom servlet kontejneru ili na bilo kom aplikacionom serveru. Konkurentan je drugim okvirima kao što su Struts, Web Work i Turbine. Tapestry je razvio Howard Lewis Ship. Tapestry je deo Jakarta Projekta i Apache softver fondacije.
Zvanična verzija Tapestry okvira je 5.0. Glavna uloga ovog okvira je da olakša kreiranje dinamičkih HTML stranica. [9]

2.2.2. Osnovne komponente Tapestry okvira

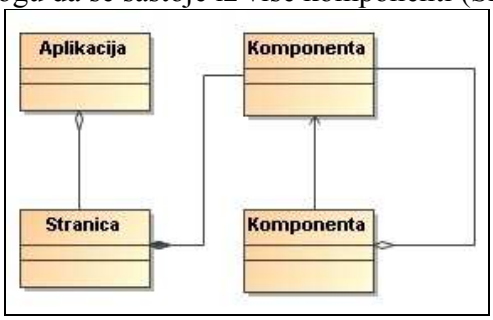
U Tapestry aplikacijama, svi elementi na web stranici (forme, linkovi, polja za unos, itd.) su predstavljeni pomoću komponenti. Sve komponente nasleđuju interfejs IComponent (org.apache.tapestry.IComponent). Dinamički sadržaji se kreiraju korišćenjem ovih komponenti (Slika 1). [9]



Slika 1. Dijagram klasa koji prikazuje relacije između Tapestry komponenti

Tapestry okvir je baziran na komponentama, a ne na operacijama. Većina web tehnologija (struts, servlets itd.) je bazirano na operacijama.

Tapestry aplikacije su sastavljene iz stranica, koje mogu da se sastoje iz manjih komponenti. Komponente mogu da se sastoje iz više komponenti (Slika 2).



Slika 2. Dijagram klasa koji prikazuje relacije između aplikacije, stranice i komponenti

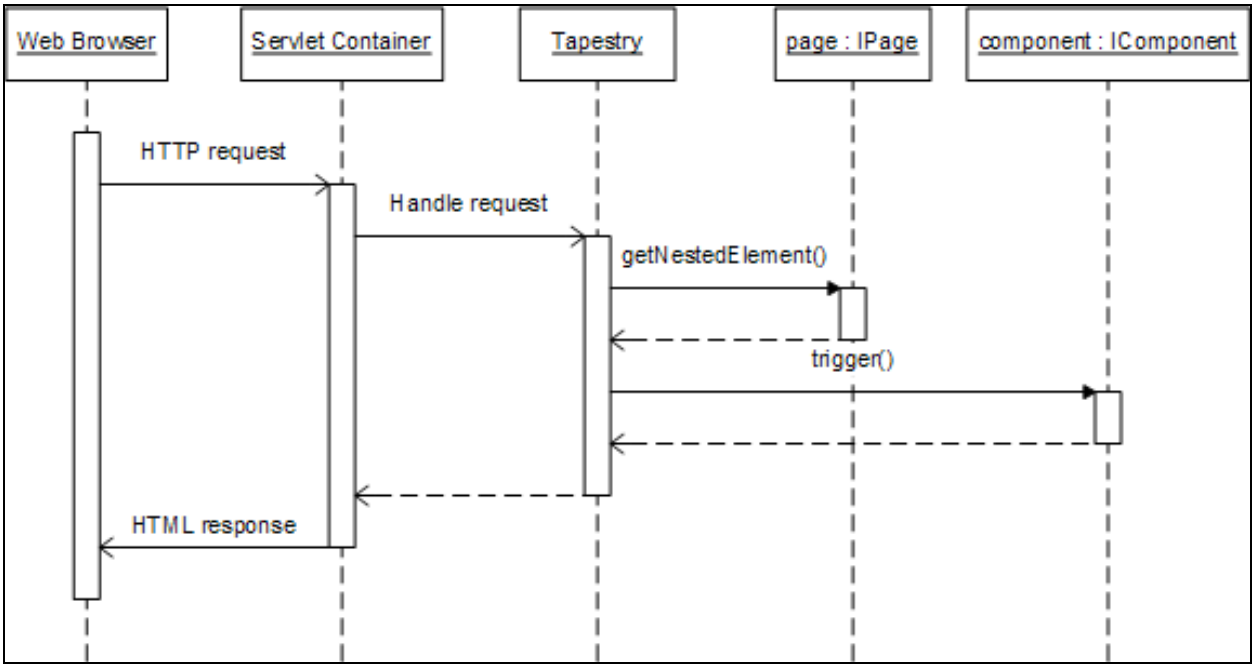
Svaka stranica poseduje jedinstveno ime, i svaka komponenta na stranici ima svoj jedinstveni identifikator. [9]

Neke Tapestry komponente mogu biti kontejneri drugih komponenata, što znači da mogu da sadrže i druge komponente. Tapestry stranice, i većina korisnički definisanih komponenti, imaju osnovni kostur (templejt), posebni HTML dokument koji opisuje strukturu i ponašanje dela komponente, sa indikatorima koji definišu da li su komponente aktivne.

Komponente mogu imati jedan ili više imenovanih parametara koji služe da detaljnije opišu komponentu. Kao i parametri Java metoda, parametri Tapestry komponenti mogu biti ulazno izlazni. Komponenta može da čita parametar i da prikaže vrednost, ili da upiše parametar da postavi vrednost.

Komponente, kao što su forma, i komponente forme (TextField, PropertySelection, Checkbox, itd.), lako se transformišu u HTML forme.[9]

Opšta dinamika poziva komponenti Tapestry okvira prikazana je dijagramom sekvenci koji je dat na Slici 3.



Slika 3: Sekvencni dijagram Tapestry okvira

Na slici su prikazane komponente Tapestry okvira koje učestvuju u životnom ciklusu koji počinje HTTP zahtevom iz web čitača i završava HTML odgovorom aplikacije.

Najpre se Servlet Container-u prosleđuje HTTP zahtev za određenom HTML stranicom. Potom Servlet Container prosleđuje zahtev Tapestry okviru. Tapestry okvir kreira stranicu, čita atribute dokumenata i prikazuje HTML pogled iz templejta koji se prosleđuje web čitaču kao odgovor na HTTP zahtev. [9]

Na slikama 4 i 5, prikazani su redom, izgled HTML Tapestry stranice za logovanje korisnika iz studijskog primera aplikacije za biblioteku, i Tapestry template stranica (Login.tml) koja služi da definiše komponente koje se nalaze na konkretnoj stranici. Kao što se na slici 3 vidi stranica za logovanje se sastoji od nekoliko komponenti tipa Form(loginForm),TextField (email i password) i Button (submit). Ove komponente su definisane html kodom, koji se nalazi na slici 4.

Kao što je ranije navedeno, jedna komponenta može da se sastoji iz više komponenti. U konkretnom primeru komponenta loginForm sadrži komponente email, password i submit.

Slika 4. Primer Tapestry stranice

Sve tri komponente se nalaze u okviru taga forme čiji je identifikator loginForm (<t:form t:id="loginForm">). Kao što je ranije navedeno svaka komponenta Tapestry okvira poseduje određene parametre. LoginForm komponenta poseduje parametar "*id*", njen identifikator koji ima vrednost "*loginForm*". Pomoću ovog parametra komponenti se može pristupati unutar programskog koda i njome manipulisati. Komponentama tipa TextField, TextArea može se veoma jednostavno dodati validacija postavljanjem vrednosti parametra komponente *validate* na *required.* U konkretnom primeru to su text polja *email i password. (*<input type="text" t:type="passwordfield" t:value="password" t:validate="required"/>*)*

```
<t:border xmlns:t="http://tapestry.apache.org/schema/tapestry_5_0_0.xsd">
<head>
	<title>XLIBRARY</title>
</head>
<body>
	<h2>Dobro dosli u online biblioteku!</h2>
	<p><h2>${naslov}</h2></p>
	<p>${message}</p>
	<p>Ukoliko ne posedujete nalog sledite link registruj me, ukoliko
posedujete ulogujte se da imate pristup svim opcijama</p>
	<p>Unesite vas email i password</p>
	<t:form t:id="loginForm">
	<table width="100%">
```

```
      <tr>
            <td width ="15%">Email</td>
            <td width ="85%"><input type="text" t:type="textfield"
                        t:value="email" t:validate="required"/></td>
      </tr>
      <tr>
            <td>Password</td>
            <td><input type="text" t:type="passwordfield" t:value="password"
                  t:validate="required"/></td>
      </tr>
      <tr>
            <td></td>
            <td><input type="submit" value="Uloguj me!"/></td>
      </tr>
      <tr>
            <td></td>
            <td><a href="#" t:type="PageLink"
      t:page="RegisterStudent">Registruj Me!</a></td>
      </tr>
      <tr>
            <td></td>
            <td><a href="#" t:type="PageLink" t:page="Start">Refresh</a></td>
      </tr>
      </table>
    </t:form>

</body>
</t:border>
```

Slika 5. Html kod Login stranice

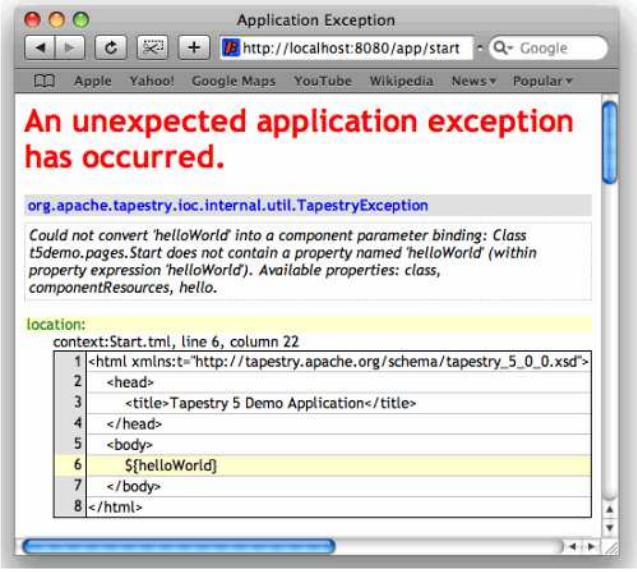
Tapestry prati komponentni pristup web razvoju, omogućavajući da se veliki deo programskog koda izdvoji izvan aplikacije, i stavi unutar okvira.

Tapestry okvir se izdvaja sledećim odlikama [9]:

- **Obrada grešaka i njihova klasifikacija**

Tapestry ima veoma precizno izveštavanje o greškama i mehanizam klasifikacije (Slika 6) Izveštavanje o greškama koristi refleksiju da funkcioniše unazad, kroz lanac grešaka, prikazujući JavaBeans atribute (property) svake greške.

Osim opisa same greške, prikazuju se podaci o servletu, aplikaciji, zahtevu, kontekstu servleta i JVM (Java Virtuela Mašina). Cilj je da se generiše izveštaj kada se pojavi greška, takav da sačuva programerima vreme koje bi proveli tražeći grešku.



Slika 6. Prikaz greške na Tapestry stranici

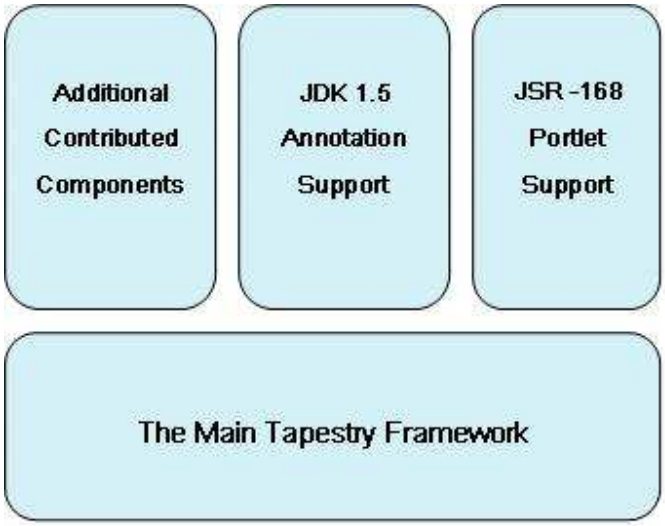
- **Tapestry strane i njihovo skladištenje**

Tapestry stranice se skladište. Da bi se kreirala stranica, obično se koristi Tapestry osnovna stranica, koja poseduje određenu funkcionalnost za ponovno korišćenje sa određenim brojem različitih atributa i metoda i dolazi uz Tapestry distribuciju u obliku kostura (template). Kada se u web čitač unese url stranice, kreira se jedno pojavljivanje (instanca) stranice što zahteva određene resurse računara. Da bismo izbegli kreiranje objekta svaki put kada novi korisnik pristupi stranici, Tapestry poseduje skladište (pool) za svaku vrstu stranice i u njemu uvek čuva po nekoliko pojavaljivanja tražene stranice.

2.2.3. Tapestry Distribucija

Tapestry distribucija je dostupna za preuzimanje na Tapestry web stranici na Apache web sajtu. Tapestry distribucija sadrži sledeće delove, prikazane na Slici 7. [9]

1. **Glavni Tapestry okvir (The main Tapestry okvir)**: Ova arhiva sadrži osnovne komponente Tapestry okvira.
2. **Priložene komponente** (**Contributed components)**: Ovo je kolekcija dodatnih komponenti za Tapestry. Sadrži komponente koje se koriste da kreiraju HTML odgovore za osvežene stranice.
3. **Podrška za anotacije (Annotation support classes):** Ova biblioteka sadrži anotacije. Ove anotacije zahtevaju JDK 1.5. Anotacije predstavljaju dodatke kodu koji naznačavaju Java kompajleru da taj deo koda tretira na drugačiji način. Mogu da obezbede neke operacije unutar Java koda koje bi inače bile specificirane u stranici ili komponenti.
4. **Portlet podrška (Portlet support):** Modularni dodatak Tapestry okviru koji omogućava Tapestry-u da kreira JSR-168 portlete.



Slika 7. Sadržaj Tapestry distribucije

Tapestry ima podršku za dodatne zavisnosti koje se mogu preuzeti u JAR arhivi. Ovo može biti urađeno raznim alatima ili ručno, dok je u mom studijskom primeru korišćen alat Maven, koji automatski preuzima sve potrebne biblioteke.

2.2.4. Prednosti i nedostaci Tapestry okvira

Tapestry je dobra zamena za JavaServer Pages. Tapestry predstavlja kompletan okvir za pravljenje kvalitetnih dinamičnih aplikacija. [9]
Navedene su neke od osnovnih prednosti Tapestry okvira:

- Poseduje mogućnost ponovnog korišćenja jer se komponente u Tapestry okviru mogu ponovo koristiti.
- U Tapestry okviru kod se sastoji iz objekata, metoda i atributa (properties), nema URL ova i upitnih parametara.
- Internacionalizacija/Lokalozacija je veoma jednostavna.
- Veoma jasno je razdvojena poslovna logika od korisničkog interfejsa i kao rezultat toga može se postići dobra saradnja izmeću programera i web projektanta. [9]

Jedan od glavnih nedostataka Tapestry okvir je nedostatak podrške alata za razvoj aplikacija. Iako je istraženo da Tapestry aplikacija može biti izrađena koristeći jednostavan alat za obradu teksta i Ant ili Maven, ne može se sa zadovoljstvom prihvatiti od strane većine Java web programera.

Drugi, ne manje bitan nedostatak je da pruža komplikovana rešenja za jednostavne probleme. Razlog je taj što su programeri koji su razvijali okvir utrošili dosta vremena da naprave rešenje kojim će pokriti složenije probleme, pa su tako jednostavniji problemi ostali zapostavljeni. Ovo može da se kompenzuje kreiranjem većeg broja komponenti za okvir, različitog nivoa i snage, što još uvek nije kompletno urađeno. Zbog ovoga mnogi misle da Tapestry još uvek nije kompletan okvir. [9]

2.3. Spring Okvir

U decembru 1996 objavljena je *JavaBeans 1.00-A* specifikacija, JavaBeans model softverske komponente za Javu. Ova specifikacija definiše skup programerskih tehnika koje omogućavaju jednostavnim Java objektima da budu ponovo korišćeni i jednostavno komponovani u složenije aplikacije.

Složenije aplikacije često zahtevaju servise kao što su podrška za transakcije, bezbednost, distribuirane kompjuterske servise koji nisu direktno obuhvaćeni *JavaBeans* specifikacijom. U martu 1998., Sun objavljuje 1.0 verziju *Enterprise JavaBeans* specifikacije (EJB). Svrha EJB specifikacije bila je jednostavniji razvoj složenih (enterprise) aplikacija.

Bez obzira na činjenicu da su mnoge uspešne aplikacije su izrađene korišćenjem EJB tehnologije, EJB nikad nije dostigao svoju svrhu: da pojednostavi razvoj složenih aplikacija.

Danas, razvoj Java komponenti se vraća svojim korenima, jednostavnom razvoju komponenti. Na taj način omogućeno je ponovno korišćenje klasa i njihovo jednostavno ugrađivanje u složenije aplikacije. Nove programerske tehnike, uključujući i *Aspektno Orjentisano Programiranje (AOP) i Dependency Injection (DI),* daju JavaBean-ovima svojstva koja su bila rezervisana za EJB. Kao rezultat ovoga nastaju "*Plain-Old Java Objects(POJOs)*". Termin POJO predstavlja Java objekat kod koga su svi atributi privatni (private) i za svaki od ovih atiributa postoje metode za pristup i izmenu njhovih vrednosti (get i set metode).

Spring Okvir je open source okvir, koji je kreirao Rod Johnson i opisao u svojoj knjizi "*Expert One-on-one: J2EE Design and Development*". Kreiran je da ublaži razvoj složenih aplikacija. [4]

Ključni koncepti koji karakterišu Spring okvir su:

Jednostavnost (Lightweight) – *Spring Okvir* može biti distribuiran kao jedinstveni JAR. Spring okvir ne zahteva da korisnički definisani objekti u aplikacijama ne zavise od specifičnih Spring klasa.

Dependency injection – Spring promoviše uparivanje pomoću tehnike zvane Dependency Injection (DI). Kada je DI uključen, objektima su pasivno date njihove zavisnosti umesto kreiranja i traženja njihove međuzavisnosti. U zasebnoj XML datoteci se definišu veze između klasa unutar aplikacije. Prilikom kreiranja novog objekta u aplikaciji koja koristi Spring okvir, automatski se i kreiraju veze tog objekta sa drugim objektima aplikacije, koje Spring okvir čita iz XML dokumenata. Na ovaj način programeri definišu veze između objekata pa u toku programiranja mogu više da se posvete usavršavanju ostalih delova aplikacije.

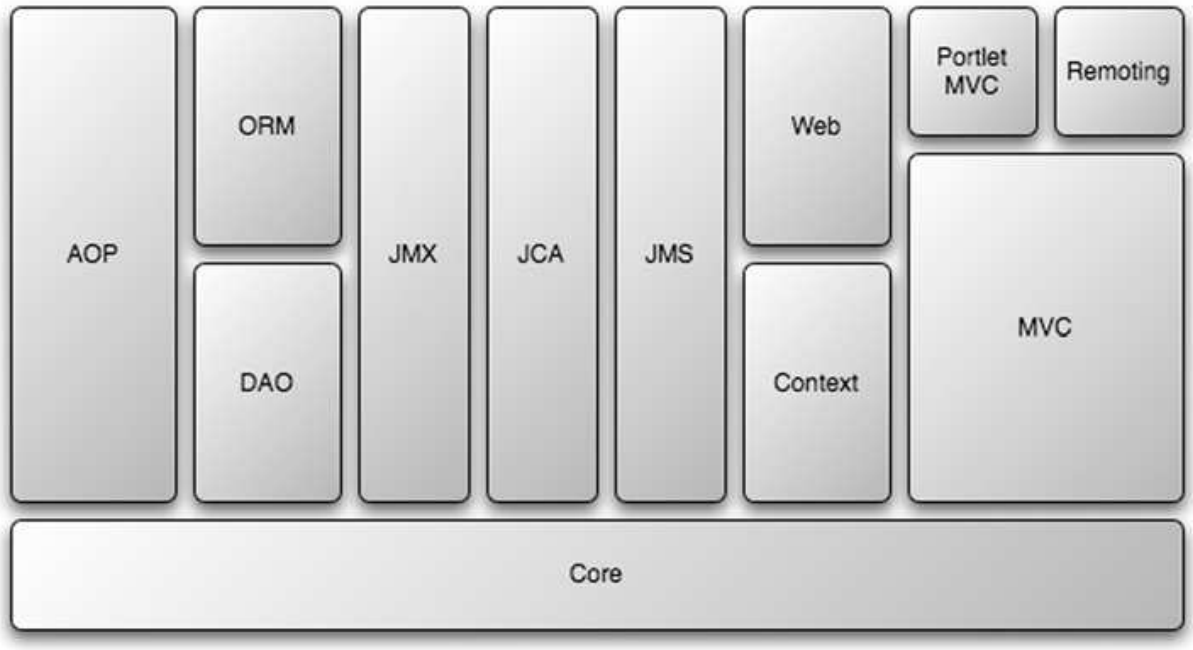
Aspektno orjentisan (Aspect-oriented) – Spring podržava aspektno orjentisano programiranje (AOP), koje omogućava razvoj poslovne logike koja je nezavisna od sistemskih servisa.

Kontejner (Container) – Spring je kontejner u smislu da sadrži i upravlja životnim ciklusom i konfiguracijom aplikacionih objekata. U Springu se može odrediti kako će svaki od aplikacionih objekata biti kreiran, kako će biti konfigurisani, i kako će biti povezani sa ostalim objektima u sistemu.

Okvir (Framework) – Spring omogućava da se konfiguriše i planira složena aplikacija koja je sastavljena od jednostavnih komponenti. U Spring okviru, aplikacioni objekti se deklarišu unutar XML datoteke.

Spring okvir je sastavljen od nekoliko modula (Slika 8) koji su izrađeni iznad osnovnog (Core) kontejnera koji čini osnovu Spring okvira. Ova modularnost omogućava korišćenje pojedinih modula u aplikaciji.

Kada se koriste kao celina, ovi moduli omogućavaju razvoj složenih aplikacija. Aplikacija može da se razvije i korišćenjem pojedinih modula, koji se mogu integrisati sa drugim okvirima i bibliotekama Svi Spring moduli su izgrađeni na osnovnom Core modulu. Ovaj modul definiše način na koji se bean-ovi kreiraju, konfigurišu i upravljaju od strane samog okvira. Klase koje se nalaze unutar ovog modula se implicitno koriste kada se konfiguriše aplikacija.[4]



Slika 8- Moduli koji čine Spring okvir

2.3.1. Dependency injection

Tradicionalno svaki objekat je odgovoran za upotrebu sopstvenih referenci na objekte sa kojima komunicira (sa svojim zavisnostima). Ovo može da vodi do veoma uparenog i koda teškog za testiranje. Kada se koristi *Dependency Injection (DI)*, objektima su date njihove zavisnosti u trenutku njihovog kreiranja. Kreiranje tih zavisnosti kao i koordinacija svakim objektom u sistemu poverena je nekom spoljnom entitetu. Drugim rečima, zavisnosti su definisane unapred. Pre kreiranja svakog od objekata definisana je njihova veza sa ostalim objektima sistema. Zavisnosti su uključene (injected) u objekte. Najčešće se definisanje zavisnosti između objekata vrši u okviru nekog XML dokumenta.[4]

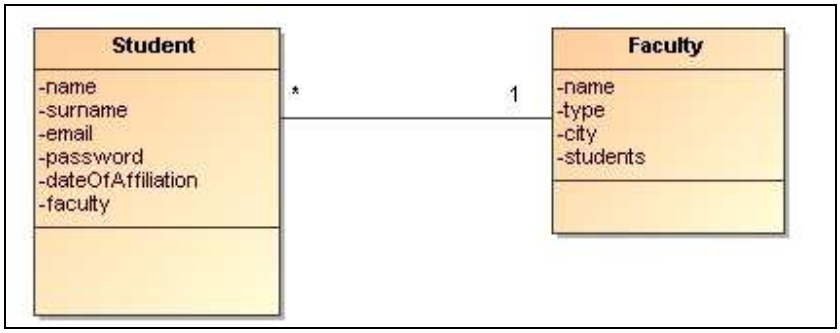
Dependency Injection (DI) predstavlja povezivanje POJO-a pomoću XML dokumenta. U tom dokumentu se definišu veze između klasa koje su deo aplikacione logike softverskog sistema.

```
<bean id="net.sf.brightside.xlibrary.metamodel.Faculty"
      class="net.sf.brightside.xlibrary.metamodel.beans.FacultyBean"
      scope="prototype">
            <property name="students">
                  <ref bean="java.util.List" />
      </property>
 </bean>
```

Slika 7: Primer XML konfiguracionog dokumenta

U navedenom primeru dat je deo XML dokumenta (Slika 7) koji se odnosi na povezivanje dva entiteta, `Student i Faculty` i njihove međusobne zavisnosti. Prema prikazanom modelu (Slika 8) entitet `Faculty` kao atribut ima jedno polje students, koje predstavlja kolekciju studenata koji pripadaju konkretnom fakultetu. Za dati atribut klase `Faculty`, može se definisati tip podatka koji će predstavljati prilikom instanciranja klase `Faculty`. Tip podatka se definiše u okviru taga *<ref bean>* kao što je prikazano na slici 7. Na konkretnom primeru, atribut `students` će prilikom svakog instanciranja klase `Faculty` biti kolekcija tipa java.util.List.

Nakon što se sve međuzavisnosti opišu XML dokumentom, Spring okvir koordinira zavisnostima. U trenutku kreiranja svakog od objekata Spring okvir kreira reference na ostale objekte sa kojima komunicira konkretan objekat.



Slika 8: Međuzavisnosti entiteta Faculty i Student

Poziv za kreiranje nove instance klase Fakultet u java kodu izgleda ovako:

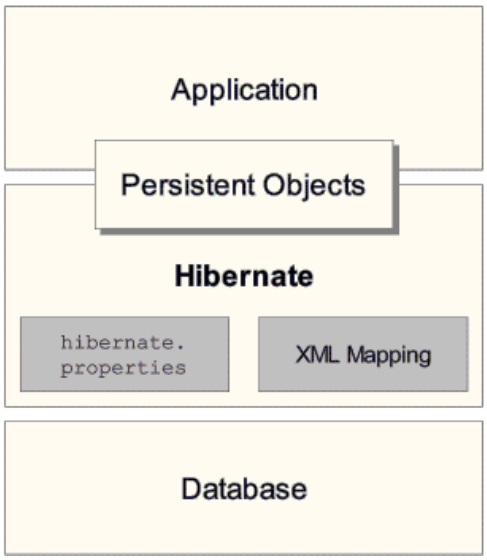
```
(Faculty) applicationContext().getBean(Faculty.class.getName())
```

Poziva se metoda `applicationContext`(), koja kreira konkretni kontekst aplikacije, čita XML document, u kome pišu zavisnosti među Java beanovima i poziva Spring da napravi novo pojavljivanje klase Faculty.

2.4. Hibernate Framework

Hibernate je Java open source alat za objektno-relaciono preslikavanje. Hibernate okvir služi za preslikavanje Java klasa u relacione baze podataka i omogućava upite nad kreiranim bazama podataka. Na taj način značajno je smanjeno programersko vreme koje bi bilo utrošeno na ručno čuvanje, prikazivanje i izmenu podataka u SQL - u i JDBC - u. [13]

Više nivojska arhitektura Hibernate okvira je može se opisati ilustracijom prikazanom na slici 9.



Slika 9: Višenivnojska arhitektura Hibernate okvira

Hibernate upravlja preslikavanjem između baza podataka i Java sveta na sledeći način. Hibernate koristi POJO objekte zajedno sa XML dokumentima za preslikavanje u kojima je definisano kako se koji objekat preslikava u tabelu relacione baze podataka. Hibernate čita konfiguracione dokumente u kojima su naznačene koje klase i kako treba preslikati. Osim toga Hibernate okviru je naznačeno koji tip relacione baze podataka da koristi.

U Hibernate okviru postoje dva načina definisanja na koji način će klase postati tabele u relacionoj bazi podataka. Jedan je pomoću XML dokumenta, a drugi korišćenjem anotacija. U aplikaciji obrađenoj u ovom radu, korišćeno je anotiranje klasa. Ispred naziva klase anotacijom *@Entity* se obeleži klasa koja treba da se preslika u relacionu tabelu (Slika 10), a ispred get metode atributa koji služi kao veza među klasama (atribut *students*) dodaju se anotacije koje naznačavaju na koji će način Hibernate okvir preslikati vezu između klase FacultyBean i klase StudentBean u relacionu bazu podataka. Najčešće korišćene anotacije su @Cascade koja označava koje od operacija (insert, update, delete) će biti onemogućene, i anotacija koja označava tip veze među objektima, na primeru to je anotacija *@OneToMany*, koja označava da se na jedan objekat tipa FacultyBean referencira više objekata StudentBean. Da bi se svi podaci pravilno preslikali u bazu podataka, moraju se anotirati sve klase koje su potrebne za čuvanje podataka. U primeru Student - Faculty slično anotiranje treba uraditi i sa strane klase StudentBean.

```
@Entity
public class FacultyBean extends BaseBean implements Faculty {

	private String name;
	private String city;
	private String type;
	private List<Student> students;
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public String getCity() {
		return city;
	}
	public void setCity(String city) {
		this.city = city;
	}
	public String getType() {
		return type;
	}
	public void setType(String type) {
		this.type = type;
	}
	@OneToMany(mappedBy="faculty",
			cascade = {CascadeType.ALL},
			targetEntity=StudentBean.class)
	@Cascade({org.hibernate.annotations.CascadeType.ALL,

org.hibernate.annotations.CascadeType.DELETE_ORPHAN } )
	public List<Student> getStudents() {
		return students;
	}
	public void setStudents(List<Student> students) {
		this.students = students;
	}
	@Override
	public Serializable takeId() {
		return super.get_id();
	}
	public String toString() {
		return name;
	}
}
```

Slika 10 - Anotirana klasa FacultyBean sa definisanim Hibernate zavisnostima među objektima

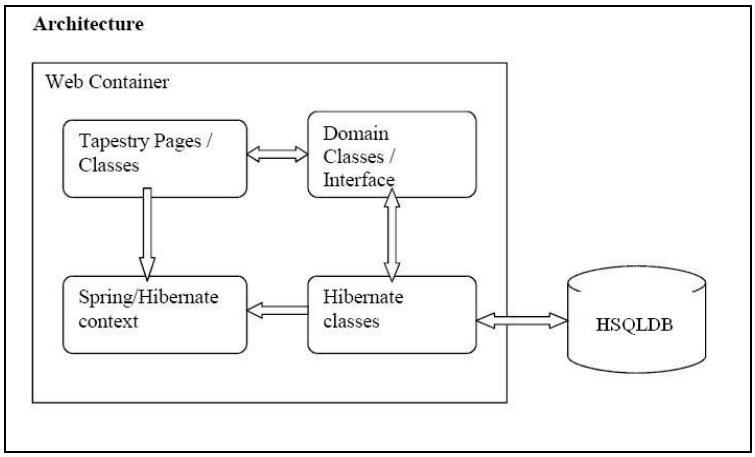
Tipičan Hibernate kod za čuvanje jednog POJO objekta u bazu izgleda ovako:

```
sessionFactory = new
Configuration().configure().buildSessionFactory();
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();
Student student = new Student();
student.setName("New Student");
student.setEmail("Email");
session.save(student);
tx.commit();
session.close();
```

Prvi korak je da se instancira HibernateSession. Sesija je glavni interfejs između Java aplikacije i Hibernate okvira. SessionFactory omogućava da se kreira Hibernate sesija čitajući konfiguracioni dokument hibernate.cfg. Posle kreiranje sesije se kreira transakcija u okviru koje će se izvršiti čuvanje jednog objekta u bazu podataka (`Transaction tx = session.beginTransaction()`). Nakon poziva komande `session.save(student)` izvrši se commit transakcije (potvrda) i sesija se zatvara i rezultat svega je sačuvan objekat klase `Student` u bazu podataka. [13]

Izvršavanje upita nad objektima koji su predstavljeni bazom podataka je veoma jednostavno. Da bi se izmenili objekti, na njih se deluje u okviru programskog koda, potom se pozove Hibernate okvir da sačuva izmene.

Kreiranje novih objekata takođe je jednostavno. Kreiraju se objekti i nakon toga se pozove *Hibernate* da ih sačuva u bazu podataka. *Hibernate* sam obezbeđuje strukturu baze podataka, tako da na programerima ostaje da se posvete poslovnoj logici svoje aplikacije i samo obeleže koje klase *Hibernate* treba da sačuva. Okvir sam generiše skript za bazu podataka, gde programer ne mora da zna kako su objekti sačuvani. [7]

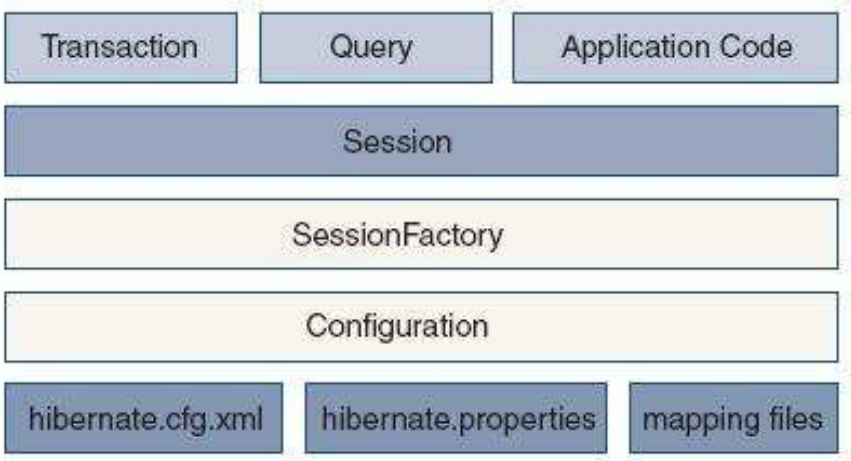


Slika 11 - Hibernate okvir unutar web aplikacije informacionog sistema biblioteke "Xlibrary"

Na slici 11 je prikazana uloga Hibernate okvira u web aplikaciji informacionog sistema biblioteke "Xlibrary". Prikazana je komunikacija između svakog od tri korišćena okvira. Kao što

je prikazano na slici, uloga Hibernate okvira je da omogući preslikavanje između Java klasa i relacione baze podataka, uloga Spring okvira je da olakša upravljanje aplikacionom logikom sistema, dok Tapestry okvir služi za izradu nivoa korisničkog interfejsa.

Hibernate okvir se sastoji iz pet osnovnih interfejsa. *Interfejsa sesije* (Session Interface), *Session Factory interfejsa*, *Configuration interfejsa*, *Transaction interfejsa* i *Query i Criteria interfejsa. (slika 12)*



Slika 12- Osnovne Hibernate Komponente

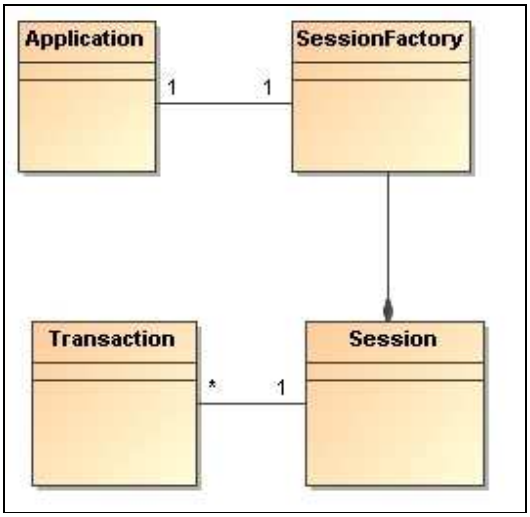
Session interfejs služi za kreiranje sesije. Objekat sesije je jednostavan i lak za kreiranje i uništavanje. Ovo je bitno jer u aplikacijama je potrebno da se kreiraju i uništavaju sesije sve vreme. Sesija predstavlja nešto između konekcije i transakcije.

Session factory interfejs. Aplikacija obuhvata sesiju instanciranu iz SessionFactory interfejsa. U poređenju sa Session interfejsom ovaj objekat je mnogo manje interesantan. On nije jednostavan, lightweight, on treba da bude deljen između mnogo aplikacionih niti. Ovo je jednostavno jedinstveni SessionFactory za celu aplikaciju.

Configuration interfejs, konfiguracioni objekat se koristi da se konfigurise *Hibernate*.

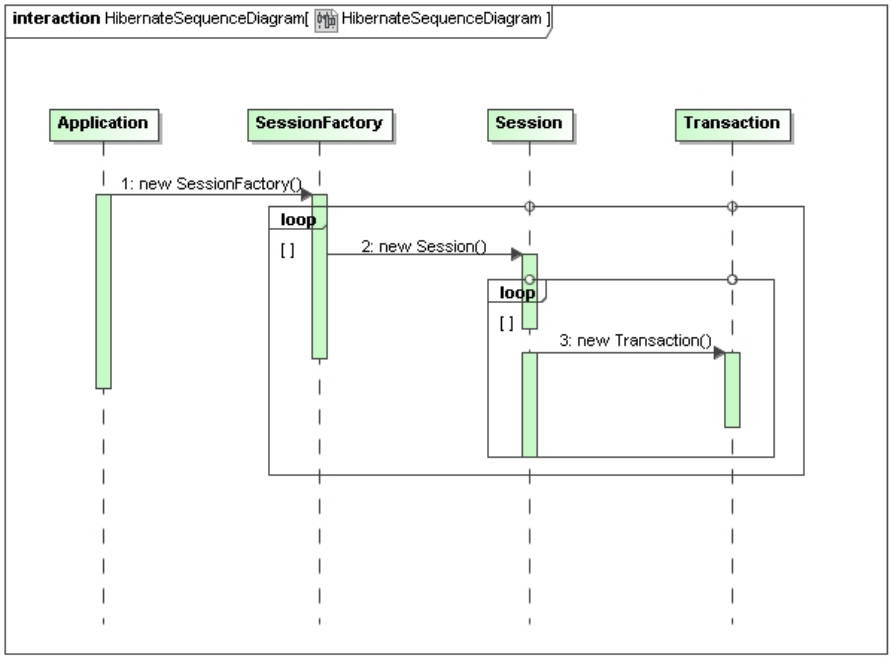
Transaction Interfejs je opcioni API. Služi za upravljanje transakcijama. *Hibernate* aplikacije mogu i ne moraju da koriste ovaj Interfejs.

Query interfejs dozvoljava da se izvode upiti nad bazom podataka i da se kontroliše kako će se oni izvršavati. **Criteria interfejs** je veoma sličan query interfejsu, dozvoljava kreiranje i izvršavanje objekto-orjentisanih kriterijuma upita.[7]



Slika 13: Hibernate Class Diagram

Na slici 13 prikazan je metamodel Hibernate okvira i uloga osnovnih interfejsa u okviru Hibernate okvira. Kao što je prikazano na slici SessionFactory je jedinstven na nivou aplikacije (Aplication), jedna aplikacija ima jedno pojavljivanje SessionFactory klase. SessionFactory može proizvesti više sesija (Session), dok u okviru jedne sesije može biti više transakcija (Transaction).



Slika 14: Hibernate sekvencni dijagram poziva komponenti

Na slici 14 prikazan je redosled pozivanja redom SessionFactory, Session i Transaction interfejsa. Najpre se u okviru aplikacije napravi jedno pojavljivanje SessionFactory klase, potom SessionFactory pravi onoliko Session pojavljivanja koliko je potrebno u aplikaciji. Dok se u okviru jednog pojavljivanja može izvršiti onoliko transakcija koliko određena programska komanda zahteva.

Ovo su ukratko opisane tri najzastupljenije tehnologije korišćene u izradi diplomskog rada *Web* aplikacije za biblioteku u Java okruženju, a njihova primena u konkretnoj aplikaciiji biće detaljnije opisana u sledećem poglavlju.

3. STUDIJSKI PRIMER

U ovom poglavlju dat je prikaz razvoja studijskog primera informacionog sistema biblioteke korišcenjem *Larmanove metode razvoja softvera*. Implementacija studijskog primera je izvršena korišćenjem *Spring, Hibernate i Tapestry* okvira.

3.1. Larmanova metoda razvoja softvera

Proces razvoja softvera prema ovoj metodi obuhvata sledeće faze:

1. Specifikacija zahteva
2. Analiza
3. Projektovanje
4. Implementacija
5. Testiranje

Specifikacija zahteva je faza intenzivne saradnje projektanta i stručnjaka koji poznaju domen problema. Zahtevi se opisuju pomocu Modela slucaja korišćenja *(Use-case Model)*, kojim se opisuje skup željenih korišćenja sistema od strane korisnika [1]. ***Faza analize*** oslanja se na dobro definisane i objašnjene slučajeve korišćenja. Ona opisuje logičku strukturu i ponašanje softverskog sistema (poslovnu logiku softverskog sistema). U fazi analize kreiraju se:

1. dijagrami sekvenci
2. ugovori
3. konceptualni model.

U fazi ***projektovanja*** se, na osnovu objektno orijentisane analize koja je prethodila, opisuje se fizička struktura i ponašanje softverskog sistema (arhitektura softverskog sistema) [1]. U fazi projektovanja se kreiraju:

1. stvarni slučajevi korišcenja
2. dijagrami saradnje
3. sekvencijalni dijagrami
4. dijagrami klasa

U ***fazi implementacije*** vrši se kodiranje softverskog sistema u nekom od objektno-orijentisanih programskih jezika. U studijskom primeru korišćen je programski jezik *Java*, pri čemu je realizacija studijskog primera data korišćenjem *Tapestry, Spring i Hibernate* okvira.

Testiranje se može podeliti u nekoliko nezavisnih jedinica. Svaka softverska komponenta se testira kao nezavisna jedinica. Nakon testiranja softverskih komponenti vrši se njihova integracija. U tom smislu se prave testovi integracije softverskih komponenti.

3.1.1. Korisnički zahtevi

Korisnički zahtevi se predstavljaju svojstva i uslove koje sistem ili šire gledajući projekat mora da zadovolji. Korisnički zahtevi mogu biti funkcionalni i nefunkcionalni. Funkcionalni zahtevi definšu zahtevane funkcije sistema, dok nefunkcionalni zahtevi definišu sve ostale zahteve. Korisnički zahtevi se kod Larmana opisuju pomoću verbalnog modela i modela slučaja korišćenja sistema. Slučaj korišćenja predstavlja način korišćenja softvera. Sastoji se iz scenarija, osnovnog i alternativnog. Verbalni model predstavlja verbalni opis onoga što se očekuje da softver radi, model u kome se opisuje svrha postojanja softvera.

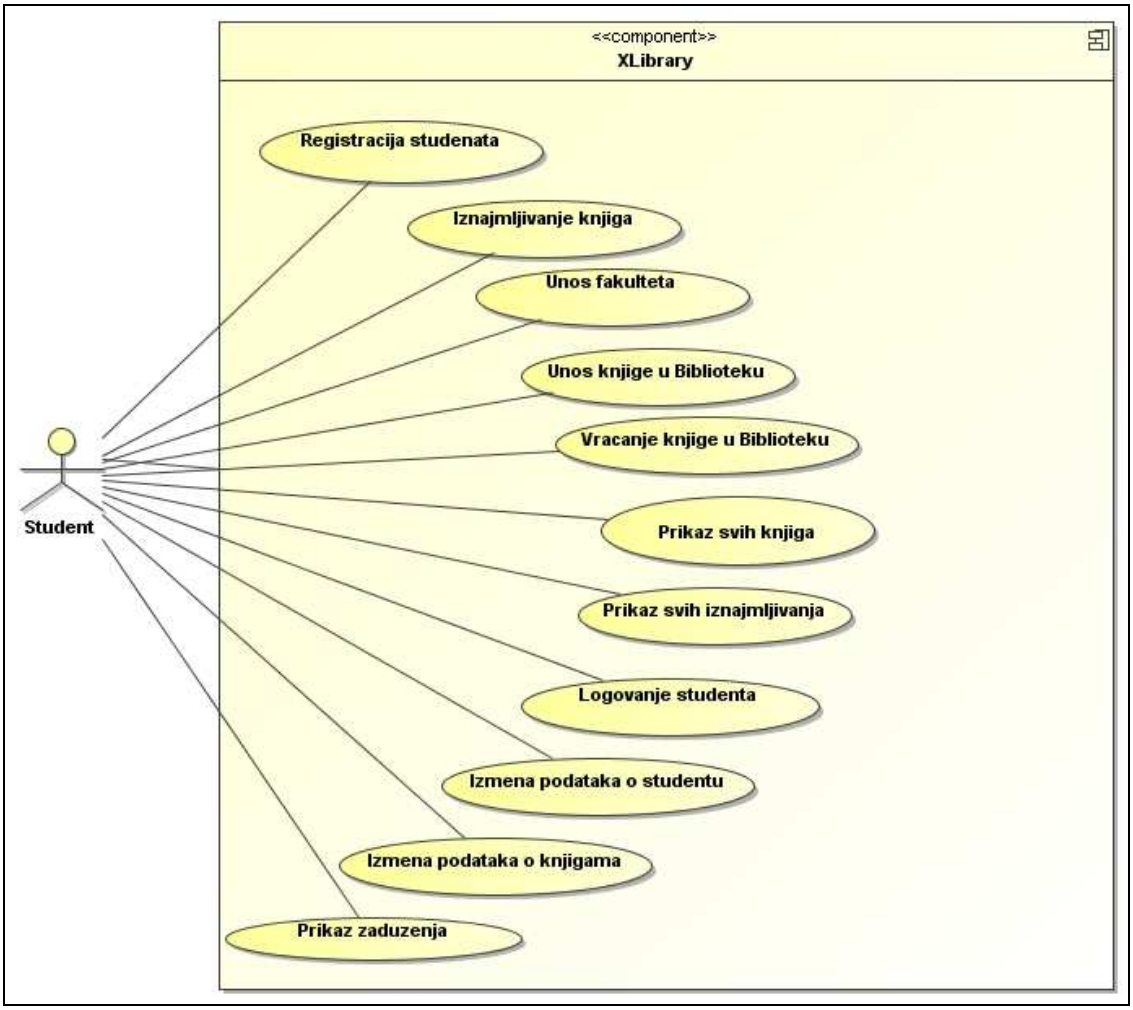
3.1.1.1. Verbalni model

Potrebno je projektovati i implementirati Informacioni sistem biblioteke. Postoje dve vrste korisnika sistema, administrator i student. Administratoru treba omogućiti unos, i izmenu knjiga u biblioteci, unos novog studenta, i izmenu podataka o studentima, i dodeljivanje administratorskih prava novim korisnicima. Student ima mogućnost da se registruje, iznajmi, vrati knjigu, pregleda sve slobodne knjige koje se trenutno nalaze u biblioteci, izmeni svoje podatke i pregleda svoja zaduženja i prethodna iznajmljivanja.

3.1.1.2. Slučajevi Korišćenja

Na osnovu verbalnog modela uočeni su sledeći slučajevi korišćenja:

1. Logovanje studenata
2. Registracija studenata
3. Izmena podataka o studentu
4. Unos knjige u biblioteku
5. Izmena podataka o knjigama
6. Pregled svih knjiga
7. Iznajmljivanje knjige
8. Vraćanje knjige
9. Pregled svih zaduženja
10. Unos fakulteta



Slika 15: Pregled slučajeva korišćenja

3.1.1.3. Opis slučaja korišćenja

U ovoj fazi dajemo tekstualni opis slučaja korišćenja (*SK*). Tekstualni SK ima sledeću strukturu:

1. Naziv SK
2. Aktore SK
3. Učesnike SK
4. Preduslovi koji moraju biti zadovoljeni da bi SK poceo da se izvršava
5. Osnovni scenario izvršenja SK
6. Alternativna scenarija izvršenja SK

SK1: Logovanje korisnika

Naziv: Logovanje korisnika
Aktori: Korisnik
Ucesnici: Korisnik i sistem
Preduslov: Sistem je uključen i prikazuje formu za logovanje korisnika.

Osnovni scenario:
1. Korisnik unosi svoju email adresu i password (APUSO)
2. Korisnik poziva sistem da uloguje studenta (APSO)
3. Sistem proverava unete podatke (SO)
4. Sistem obaveštava korisnika o uspešnom logovanju (IA)

Alternativna scenarija:
4.1. Student nije ulogovan, sistem javlja poruku: ”Neuspešno logovanje”.

SK2: Registracija studenata

Naziv: Registracija studenata
Aktori : Korisnik (student)
Učesnici: Korisnik i sistem
Preduslovi: Sistem je uključen, prikazuje formu za registraciju studenata.

Osnovni scenario:
1. Korisnik unosi ime, prezime, email i password (APUSO)
2. Korisnik poziva sistem da ga registruje (APSO)
3. Sistem registruje studenta (SO)
4. Sistem prikazuje poruku “Uspešna registracija”(IA)

Alternativna scenarija:
4.1. Sistem ne može da sačuva studenta i ispisuje poruku “Student nije sačuvan”.

SK 3: Izmena podataka o studentu

Naziv: Izmena podataka o studentu
Aktori: Korisnik (student)
Učesnici: Korisnik i sistem
Preduslovi: Student je ulogovan, sistem je uključen i prikazuje formu za izmenu podataka o studentu.

Osnovni scenario:
1. Korisnik unosi nove podatke (email, password, ime, prezime) (APUSO)
2. Korisnik poziva sistem da sačuva promene (APSO)
3. Sistem čuva promene (SO)

4. Sistem prikazuje poruku o uspešnom pamćenju promena (IA)

Alternativna scenarija:
4.1. Sistem ne može da sačuva studenta, prikazuje poruku “Student nije sačuvan”.

SK4: Unos knjige u biblioteku

Naziv: Unos knjige u biblioteku
Aktori: Korisnik
Učesnici: Korisnik i sistem
Preduslovi: Student je ulogovan, sistem uključen, prikazuje formu za unos nove knjige.

Osnovni scenario:
1. Korisnik unosi naslov, autora, datum izdavanja i broj primeraka knjige (APUSO)
2. Korisnik poziva sistem da sačuva knjigu (APSO)
3. Sistem čuva knjigu (SO)
4. Sistem prikazuje poruku o uspešnom čuvanju (IA)

Alternativna scenarija:
4.1. Sistem ne može da sačuva novu knjigu i ispisuje poruku “Knjiga nije sačuvana”

SK 5: Izmena podataka o knjigama

Naziv: Izmena podataka o knjigama
Aktori: Korisnik (student)
Učesnici: Korisnik i sistem
Preduslovi: Student je ulogovan, sistem je uključen i prikazuje formu za izmenu knjige.

Osnovni scenario:
1. Korisnik unosi naziv knjige (APUSO)
2. Korisnik poziva sistem da pronađe knjigu (APSO)
3. Sistem pronalazi knjigu (SO)
4. Sistem prikazuje traženu knjigu (IA)
5. Korisnik menja podatke (naslov, autora, datum izdavanja, broj primeraka) (APUSO)
6. Korisnik poziva sistem da sačuva promene (APSO)
7. Sistem čuva promene (SO)
8. Sistem prikazuje poruku o uspešnom pamćenju promena (IA)

Alternativna scenarija:
4.1. Sistem ne može da pronađe knjigu i prikazuje se poruka “Ne postoji tražena knjiga”.
8.2. Sistem ne može da sačuva knjigu i prikazuje se poruka “Knjiga nije sačuvana”.

SK 6: Pregled svih knjiga

Naziv: Pregled svih knjiga
Aktori: Korisnik
Učesnici: Korisnik i sistem
Preduslovi: Student je ulogovan, sistem je uključen i prikazuje početnu formu.

Osnovni scenario:
1. Korisnik poziva sistem prikaže knjige (APSO)
2. Sistem pronalazi sve knjige (SO)
3. Sistem prikazuje sve knjige (IA)

Alternativna scenarija:
3.1. Sistem ne može da prikaže knjige i prikazuje poruku “Neuspešan prikaz”.

SK 7: Iznajmljivanje knjige

Naziv: Iznajmljivanje knjige
Aktori: Korisnik (student)
Učesnici: Korisnik i sistem
Preduslovi: Student je ulogovan, sistem je uključen i prikazuje formu za iznajmljivanje knjiga .

Osnovni scenario:
1. Korisnik bira knjigu iz liste koju želi da iznajmi (APUSO)
2. Korisnik poziva sistem da sačuva iznajmljivanje (APSO)
3. Sistem čuva iznajmljivanje (SO)
4. Sistem prikazuje poruku o uspešnom čuvanju (IA)

Alternativna scenarija:
4.1. Sistem ne može da sačuva iznajmljivanje i prikazuje poruku “Neuspešno iznajmljivanje”.

SK 8: Vraćanje knjige

Naziv: Vraćanje knjige
Aktori: Korisnik
Učesnici: Korisnik i sistem
Preduslovi: Student je ulogovan, sistem je uključen i prikazuje formu sa pregledom svih zaduženja.

Osnovni scenario:
1. Korisnik poziva sistem da mu prikaže knjigu za dato zaduženje (APSO)

2. Sistem pronalazi knjigu (SO)
3. Sistem prikazuje knjigu (IA)
4. Korisnik poziva sistem da razduži zaduženje (APSO)
5. Sistem razdužuje knjigu (SO)
6. Sistem prikazuje poruku o uspešnom razduživanju (IA)

Alternativna scenarija:
2.1. Sistem ne može da prikaže knjigu i prikazuje poruku "Neuspežan prikaz"
4.1. Sistem ne može da razduži knjigu i prikazuje poruku "Došlo je do greške prilikom razduživanja knjige"

SK 9: Pregled svih zaduženja

Naziv: Pregled svih zaduženja
Aktori: Korisnik
Učesnici: Korisnik i sistem
Preduslovi: Student je ulogovan, sistem je uključen i prikazuje početnu formu.

Osnovni scenario:
1. Korisnik poziva sistem prikaže prethodna zaduženja za trenutno ulogovanog studenta (APSO)
2. Sistem pronalazi sva prethodna zaduženja (SO)
3. Sistem prikazuje sva zaduženja (IA)

Alternativna scenarija:
3.1. Sistem ne može da prikaže sva zaduženja i prikazuje poruku "Neuspešan prikaz".

SK10: Unos fakulteta

Naziv: Unos fakulteta
Aktori: Korisnik
Učesnici: Korisnik i Sistem
Preduslovi: Student je ulogovan, sistem je uključen, prikazuje formu za unos fakulteta.

Osnovni scenario:
1. Korisnik unosi podatke o fakultetu (naziv, tip i grad) (APUSO)
2. Korisnik poziva sistem da sačuva podatke (APSO)
3. Sistem čuva fakultet (SO)
4. Sistem prikazuje poruku o uspešnom čuvanju (IA)

Alternativna scenarija:
4.1. Sistem može da sačuva fakultet i ispisuje poruku "Greška prilikom čuvanja".

3.1.2. Analiza

U fazi analize opisuje se logička struktura i ponašanje softverskog sistema (poslovna logika softverskog sistema). Ponašanje softverskog sistema opisujemo pomoću sistemskih dijagrama sekvenci, koji se prave za svaki SK, i pomoću ugovora o sistemskim operacijama, koje se dobijaju na osnovu sistemskih dijagrama sekvenci. Struktura softverskog sistema je opisana pomoću konceptualnog modela.

3.1.2.1. Ponašanje sistema

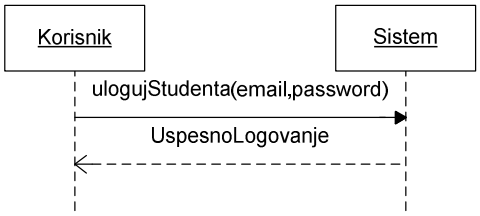
Ponašanje sistema opisujemo preko sistemskih dijagrama sekvenci.

3.1.2.1.1. Sistemski dijagram sekvenci

Sistemski dijagram sekvenci prikazuje, za izdvojeni scenario SK, akcije u određenom redosledu koji uspostavljaju interakciju između aktora i softverskog sistema.

SK1: Logovanje korisnika

1. Korisnik poziva sistem da uloguje studenta (APSO)
2. Sistem obaveštava korisnika o uspešnom logovanju (IA)

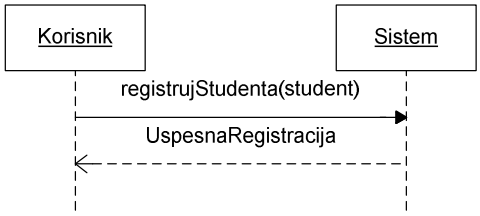


Uvedene sistemske operacije:

1. ulogujStudenta(email,password)

SK2: Registracija studenata

1. Korisnik poziva sistem da ga registruje (APSO)
2. Sistem prikazuje poruku o uspešnoj registraciji (IA)

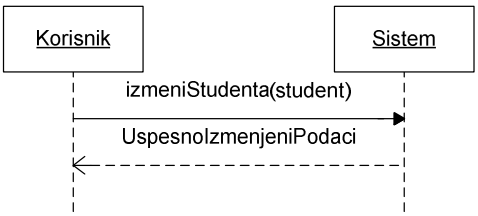


Uvedene sistemske operacije:

2. registrujStudenta(student)

SK 3: Izmena podataka o studentu

1. Korisnik poziva sistem da sačuva promene o studentu (APSO)
2. Sistem prikazuje poruku o uspešnom pamćenju promena (IA)

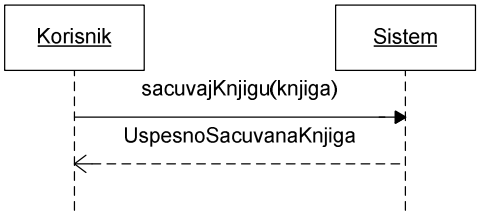


Uvedene sistemske operacije:

3. izmeniStudenta(student)

SK4: Unos knjige u biblioteku

1. Korisnik poziva sistem da sačuva knjigu (APSO)
2. Sistem prikazuje poruku o uspešnom čuvanju (IA)

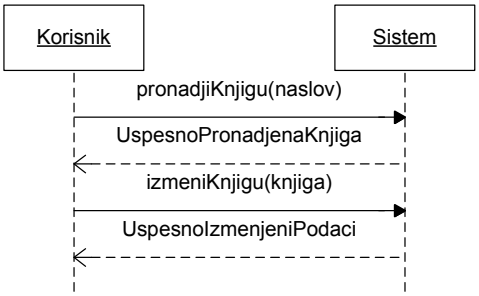


Uvedene sistemske operacije:

4. sacuvajKnjigu(Knjiga)

SK 5: Izmena podataka o knjigama

1. Korisnik poziva sistem da pronađe knjigu (APSO)
2. Sistem prikazuje traženu knjigu (IA)
3. Korisnik poziva sistem da sačuva promene (APSO)
4. Sistem prikazuje poruku o uspesnom pamćenju promena (IA)

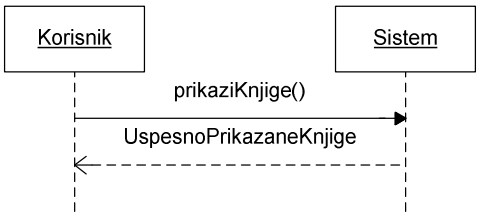


Uvedene sistemske operacije:

5. prikaziKnjigu(naslov)
6. izmeniKnjigu(knjiga)

SK 6: Pregled svih knjiga

1. Korisnik poziva sistem prikaže knjige (APSO)
2. Sistem prikazuje sve knjige (IA)

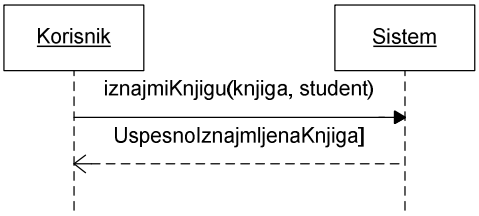


Uvedene sistemske operacije:

7. prikaziKnjige()

SK 7: Iznajmljivanje knjige

1. Korisnik poziva sistem da sačuva iznajmljivanje (APSO)
2. Sistem prikazuje poruku o uspešnom čuvanju (IA)

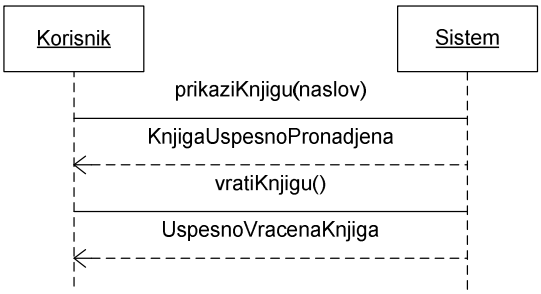


Uvedene sistemske operacije:

8. iznajmiKnjigu(knjiga, student)

SK8: Vraćanje knjige

1. Korisnik poziva sistem da prikaže zaduženu knjigu
2. Sistem prikazuje knjigu
3. Korisnik poziva sistem da razduži knjigu
4. Sistem razdužuje knjigu

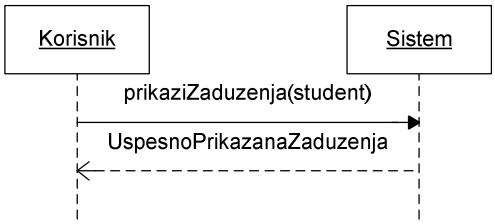


Uvedene sistemske operacije:

9. vratiKnjigu()

SK9: Pregled svih zaduženja

1. Korisnik poziva sistem prikaže prošla zaduženja (APSO)
2. Sistem prikazuje sva zaduženja (IA)

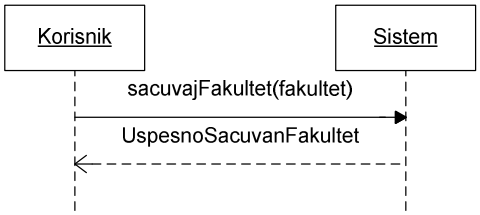


Uvedene sistemske operacije:

10. prikaziZaduzenja()

SK10: Unos fakulteta

1. Korisnik poziva sistem da sačuva podatke o fakultetu (APSO)
2. Sistem prikazuje poruku o uspešnom čuvanju(IA)



Uvedene sistemske operacije:

11. sacuvajFakultet()

Uočene su sledeće sistemske operacije koje treba projektovati:

1. ***sacuvajKnjigu(****Knjiga****) --> saveBook****(Book)***
2. ***prikaziKnjigu(****naslov****) --> getBook****(title)***
3. ***vratiKnjigu(****Iznajmljivanje****) --> returnBook****(BorrowBook)***
4. ***izmeniKnjigu(****Knjige****) --> updateBook(****Book****)***
5. *List<Knjiga>* ***prikaziSveKnjige()*** *--> List<Book>* ***getAllBooks****()*
6. ***iznajmiKnjigu(****Knjiga, Student****) --> borrowBook****(Book, Student)***
7. *List<Iznajmljivanja>* ***prikaziSvaIznajmljivanja****()--> List<BorrowBook>* ***getAllBorrows****()*
8. ***ulogujStudenta(****email, password****) --> loginStudent****(email, password)***
9. ***registrujStudenta(****Student****) --> registerStudent****(Student)***
10. ***izmeniStudenta(****Student****) --> updateStudent(****Student****)***
11. ***sacuvajFakultet(****Fakultet****) --> saveFaculty****(Faculty)***

3.1.2.1.2. Definisanje ugovora o sistemskim operacijama

Za svaku od uočenih sistemskih operacija prave se ugovori (*contracts*). Ugovori opisuju ponašanje sistemske operacije. Jedan ugovor vezan je za jednu sistemsku operaciju.

UG1: saveBook
Operacija: saveBook(b:Book)
Veza sa SK: SK5
Preduslovi:
Postuslovi: Kreirana je nova knjiga.

UG2: updateBook
Operacija: updateBook(b:Book)
Veza sa SK: SK6
Preduslovi: Knjiga postoji.
Postuslovi: Izmenjena je postojeća knjiga.

UG3: borrowBook
Operacija: borrowBook(b:Book,s:Student)
Veza sa SK: SK8
Preduslovi: Knjiga postoji, student je ulogovan.
Postuslovi: Iznajmljena je knjiga

UG4: getBook
Operacija: getBook(title:String)
Veza sa SK: SK6
Preduslovi: Knjiga postoji.
Postuslovi: Pronađena je knjiga.

UG5: getAllBooks
Operacija: List<Book> getAllBooks()
Veza sa SK: SK6
Preduslovi: Postoji bar jedna knjiga.
Postuslovi: Prikazane su sve knjige.

UG6: getAllBorrows
Operacija: List<BorrowBook> getAllBorrows(s:Student)
Veza sa SK: SK9
Preduslovi: Postoji iznajmljivanje za studenta.
Postuslovi: Prikazana sva iznajmljivanja.

UG7: loginStudent
Operacija: loginStudent(email:String, password:String)
Veza sa SK: SK1

Preduslovi: Postoji student sa traženim podacima.
Postuslovi: Student ulogovan.

UG8: registerStudent
Operacija: registerStudent(s:Student)
Veza sa SK: SK2
Preduslovi: Postoji bar jedan registrovan fakultet.
Postuslovi: Registrovan student.

UG9: getStudent
Operacija: getStudent(email:String)
Veza sa SK: SK7
Preduslovi: Postoji bar jedan student.
Postuslovi: Pronađen student.

UG10: returnBook
Operacija: returnBook(b: BorrowBook)
Veza sa SK: SK8
Preduslovi: Postoji zadužena knjiga.
Postuslovi: Izmenjeni podaci o zaduženju i izmenjeni podaci o broju knjiga u biblioteci.

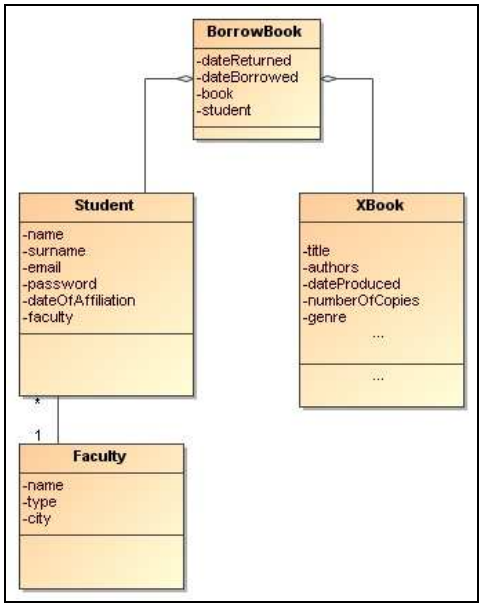
UG11: updateStudent
Operacija: updateStudent(s:Student)
Veza sa SK: SK3
Preduslovi: Postoji bar jedan student.
Postuslovi: Izmenjeni podaci o studentu.

UG12: saveFaculty
Operacija: saveFaculty(f:Faculty)
Veza sa SK: SK10
Preduslovi:
Postuslovi: Sačuvan fakultet.

3.1.2.2. Struktura sistema

Struktura softverskog sistema se opisuje pomoću konceptualnog (domenskog) modela i relacionog modela.

3.1.2.2.1. Konceptualni (Domenski) model



Slika 16: Konceptualni model

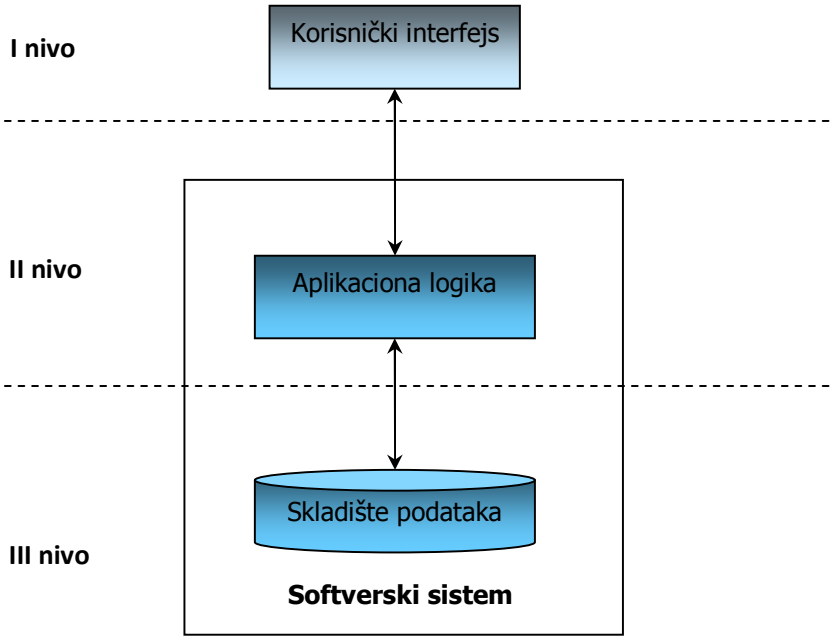
Konceptualni model opisuje konceptualne klase domena problema. Konceptualni model sadrži konceptualne klase (domenske objekte) i veze između konceptualnih klasa.

3.1.3. Projektovanje

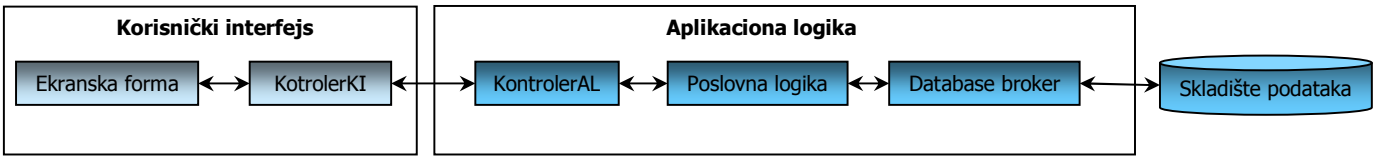
Faza projektovanja opisuje fizicku strukturu i ponašanje softverskog sistema (arhitekturu softverskog sistema). Projektovanje arhitekture softverskog sistema obuhvata projektovanje aplikacione logike, skladišta podataka i korisnickog interfejsa. U okviru aplikacione logike se projektuju kontroler, poslovna logika i database broker. Projektovanje poslovne logike obuhvata projektovanje logicke strukture i ponašanja softverskog sistema.

3.1.3.1. Arhitektura softverskog sistema

Projektovanju strukture i ponašanja softverskog sistema predhodi definisanje arhitektura softverskog sistema. U radu je korišćena klasicna tro-nivojsku arhitekturu softverskog sistema prikazana na Slici 17:

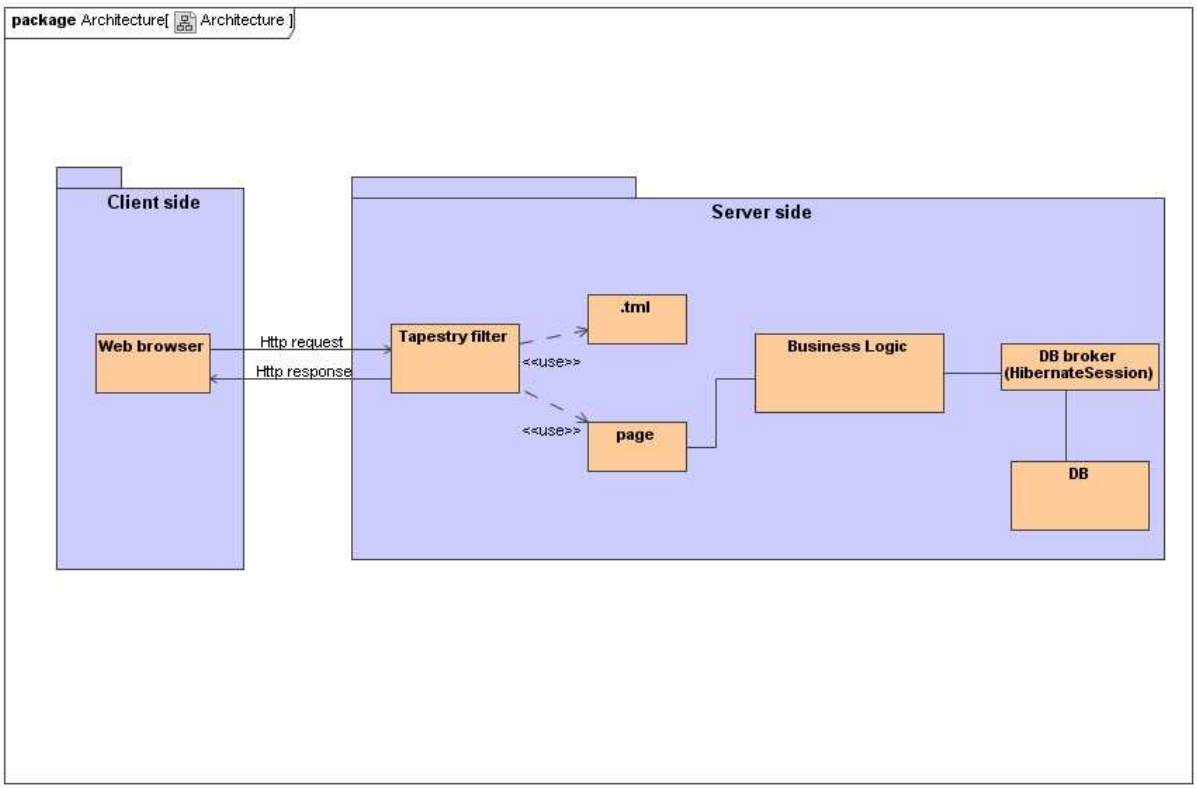


Slika 17: Tronivojska arhitektura



Slika 18: Tronivojska arhitektura – detaljniji prikaz

Na slici 18 dat je prikaz tronivojske arhitekture. Na slici se jasno razlikuju tri nivoa i to, nivo korisničkog interfejsa, nivo aplikacione logike i nivo skladišta podataka. Novo korisničkog interfejsa čine Ekranska forma i KontrolerKI, na srednjem nivou (nivou aplikacione logike) nalaze se KontrolerAL, Poslovna logika sistema i Database broker, i na najnižem nivou nalazi se skladište podataka.



Slika 19: Tronivojska arhitektura – softverski sistem biblioteke Xlibrary

Na slikama 17 i 18 dat je opšti prikaz tronivojske softverske arhitekture. Na slici 19 prikazana je arhitektura korišćena na studijskom primeru sistema biblioteke. Kao što je ranije navedeno, nivo korisničkog interfejsa implementiran je pomoću Tapestry Okvira, aplikacione logike sistema pomoću Spring Okvira, a preslikavanje iz objektnog u relacioni model izvršeno je pomoću Hibernate okvira, klase HibernateSession koja u konkretnom slučaju igra ulogu Database brokera. (Slika 19). HibernateSession je Java klasa koja je zadužena za otvaranje i zatvaranje konekcije prema bazi i za čuvanje, izmenu i brisanje slogova u datoj bazi podataka. HibernateSession brine o svemu: otvaranju, zatvaranju sesije, izmeni promena koje smo napravili pozivom konkretne naredbe. Na programerima ostaje samo da pozovu naredbu za čuvanje, izmenu ili brisanje objekata.

3.1.3.2. Aplikaciona logika

U okviru aplikacione logike projektuju se kontroler aplikacione logike, poslovna logika i database broker.

3.1.3.2.1. Kontroler aplikacione logike

Kontroler aplikacione logike je odgovoran da prihvati zahtev za izvršenje sistemske operacije od klijenta i da ga prosledi do poslovne logike koja je odgovorna za izvršenje sistemske operacije.

3.1.3.2.2. Poslovna logika

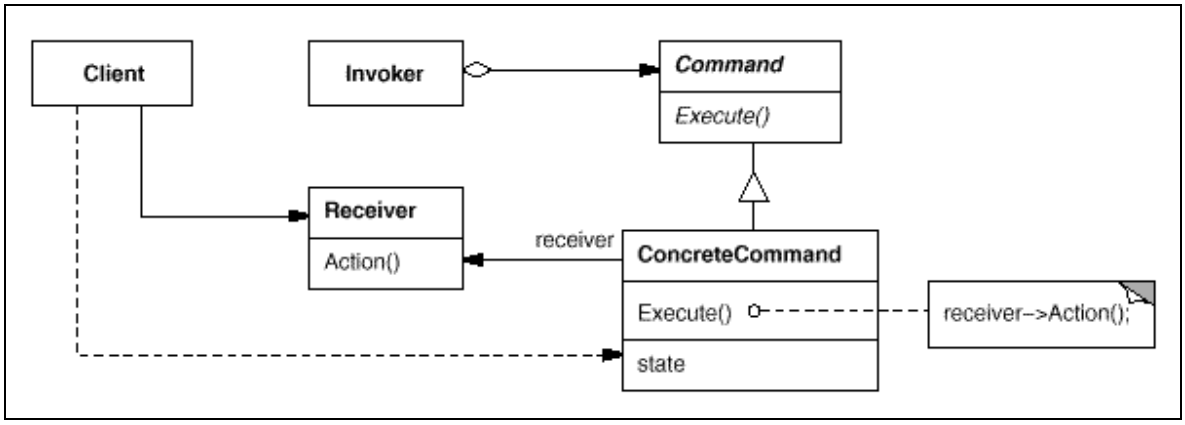
Prilikom projektovanja sistemskih operacija treba treba se držati sledećih preporuka:

U početku projektovanja SO treba napraviti konceptualne realizacije za svaku SO. Konceptualne realizacije treba da budu direktno povezane sa logikom problema.

Može se pretpostaviti da se podaci čuvaju u bazi. u tom smislu mogu se pozvati neke od osnovnih operacija baze bez ulaženja u način njihove realizacije. Aspekte realizacije koji se odnose na konekciju sa bazom, perzistentnost i transakcije treba izbeći u početku projektovanja SO. U kasnijim fazama navedeni aspekti treba da se povežu sa projektovanim rešenjima SO.

Konceptualna realizacija se može opisati preko objektnog pseudokoda, dijagrama saradnje, sekvencnih dijagrama, dijagrama aktivnosti, dijagrama prelaza stanja. U nastavku za svaki ugovor dajemo prikaz konceptualnog rešenja SO korišćenjem dijagrama sekvenci.

Sve sistemske operacije su implementirane korišćenjem Command uzora. Command uzora se implementira tako što definišemo interfejs. Command, koji sadrži metodu execute(). Svaka komanda, koja predstavlja jednu sistemsku operaciju, implementira ovaj interfejs i redefiniše metodu execute.



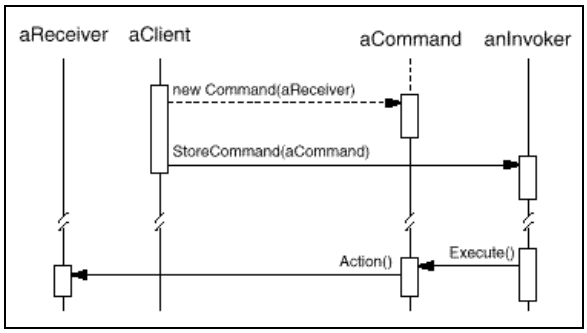
Slika 20: Struktura Command uzora [11]

Na slici 20 prikazana je struktura Command uzora. Niže su prikazani sledeći učesnici:

- *Komanda (Command)*
  - definiše interfejs za izvršavanje operacije
- *Konkretna komanda (ConcreteCommand)*
  - predstavlja vezu između objekta primaoca (Receiver) i akcije.
  - implementira Execute metodu tako što poziva koknretnu operaciju na objektu primaocu (Receiver).
- *Klijent (Client-Application)*
  - kreira objekat konkretne komande i setuje njegov objekat primalac (receiver).
- *Inicijator (Invoker)*
  - šalje zahtev za komandom.
- *Objekat primalac (Receiver)*
  - prepoznaje operaciju koja ide uz zahtev.

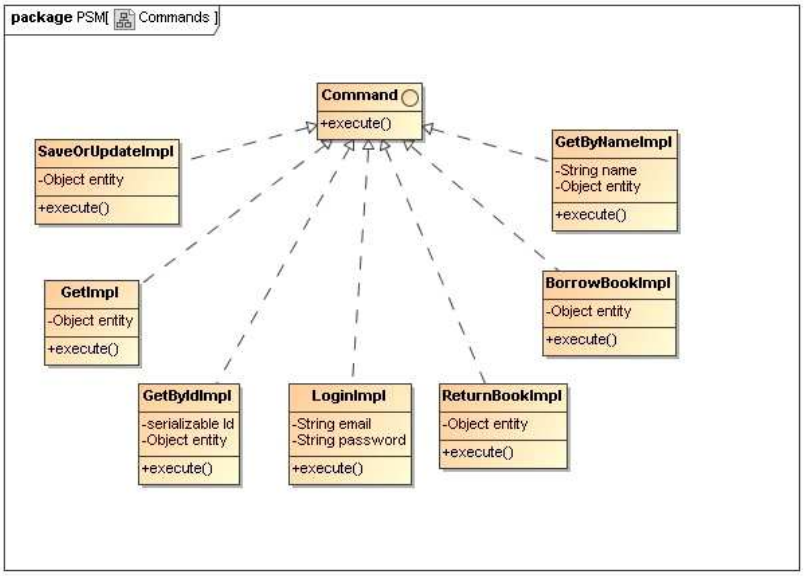
Najpre Inicijator kreira objekat komande i podešava primaoca. Objekat primalac skladišti objekat komande. Zatim inicijator zadaje zahtev pozivanjem Execute metode. Objekat konkretne komande izvršava operaciju na objektu primaocu da bi izvršio zahtev inicijatora.

Sledeći dijagram opisuje interakciju između navedenih objekata (Slika 21). Ilustrovano je kako Command razdvaja inicijatora (Invoker) od primaoca (Receiver) i kako se izvršava zahtev.[2]



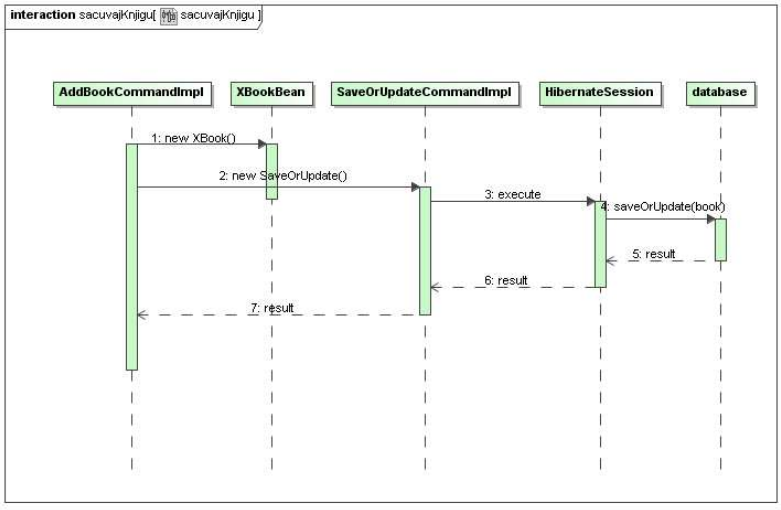
Slika 21: Sekvencni dijagram komunikacije objekata Command uzora

Command uzor učauruje koncept komande u objekat. Objekat primalac zadržava reference na objekat komande radije nego na inicijatora. Inicijator šalje komandu objektu komande izvršavajući njegovu specifičnu metodu. Objekat komande je odgovoran za prenos komande specifičnom primaocu da bi se posao završio. Inicijator ima apstraktnu komandu i zahteva njeno izvršenje tako što poziva njenu apstraktnu metodu execute(). Ova komanda je prevedena u specifičnu akciju kod primaoca pomoću raznih konkretnih objekata komandi. Takođe je moguće za komandu da se sastoji iz kolekcije komandi, koje se nazivaju "Macro command". Pozivanje execute metode makro komande izvršava se preko niza komandi. [11]
Na slici 22 prikazana je implementacija sistemskih operacija u vidu komandi korišćenjem Command uzora u konkretnom studijskom primeru.



Slika 22: Implementacija Sistemskih Operacija

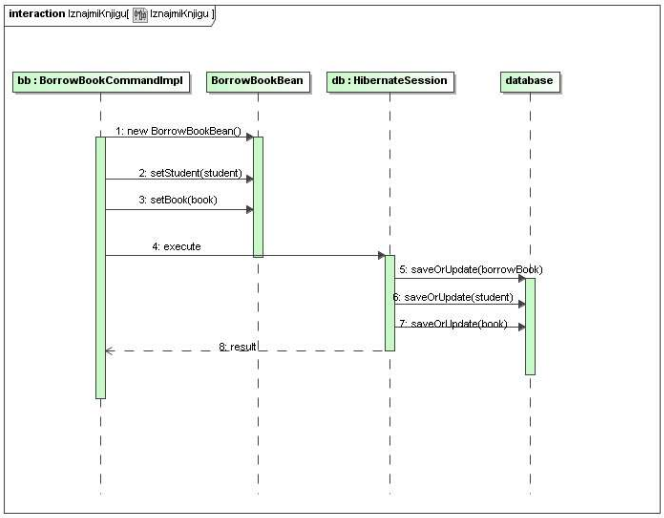
UG1: saveBook
Operacija: saveBook (k:Book)
Veza sa SK: SK4
Preduslovi:
Postuslovi: Kreirana je nova knjiga.



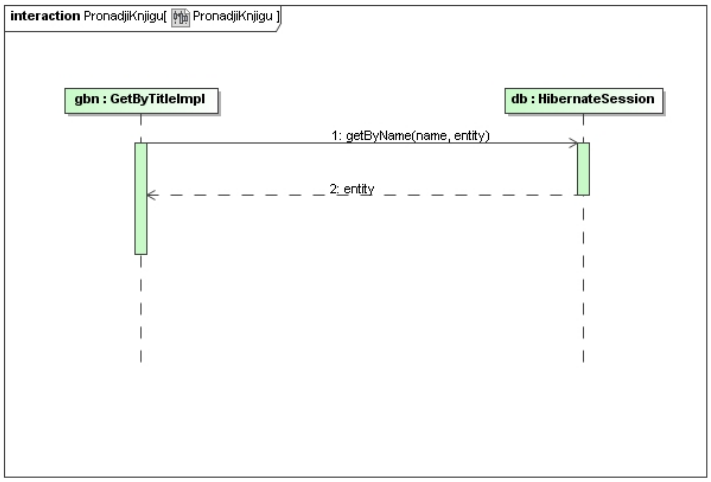
UG2: updateBook
Operacija: updateBook (k:Book)
Veza sa SK: SK5
Preduslovi: Knjiga postoji.
Postuslovi: Izmenjena je postojeća knjiga.

Ugovor1 i Ugovor2 koriste identičnu komandu koja čuva objekat bilo kog tipa (SaveOrUpdateImpl.java)

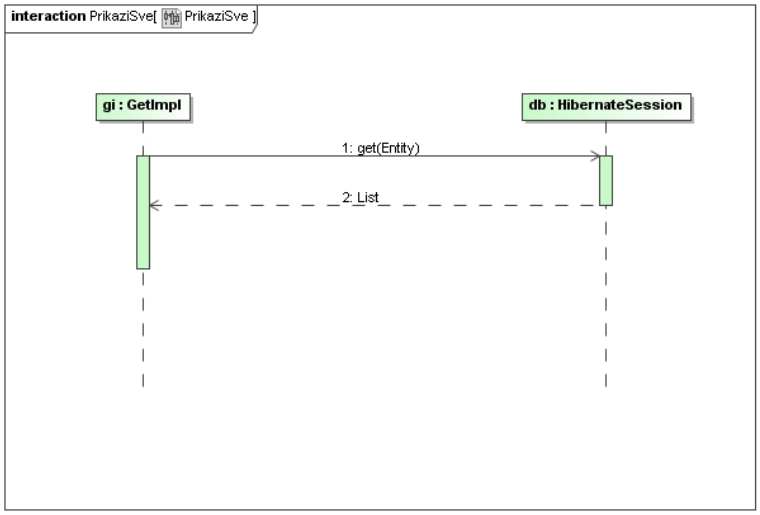
UG3: borrowBook
Operacija: borrowBook (k:Book,s:Student)
Veza sa SK: SK8
Preduslovi: Knjiga postoji, student je ulogovan.
Postuslovi: Iznajmljena je knjiga.



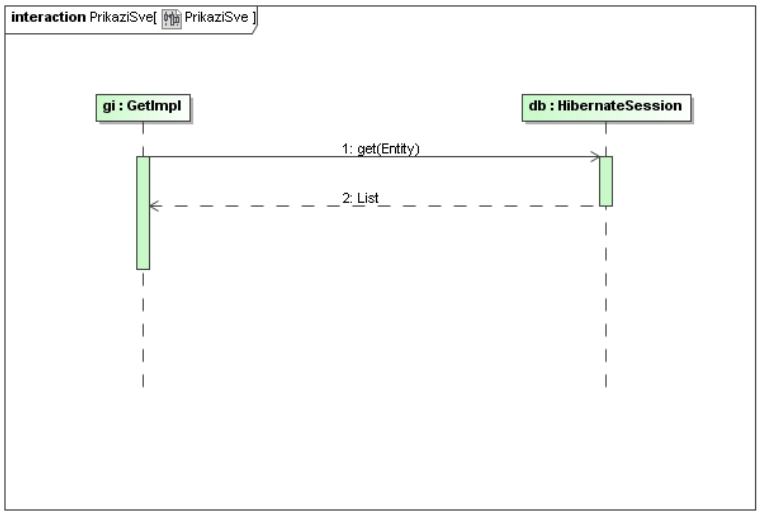
UG4: getBook
Operacija: getBook(title:String)
Veza sa SK: SK6
Preduslovi: Knjiga postoji.
Postuslovi: Pronađena je knjiga.



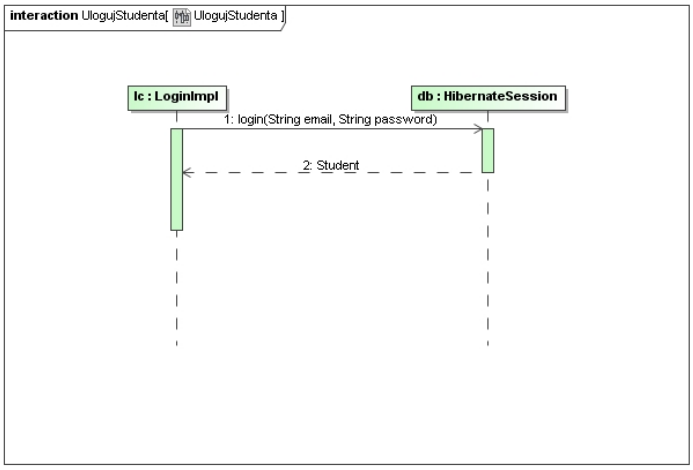
UG5: getAllBooks
Operacija: List<Book> getAllBooks()
Veza sa SK: SK6
Preduslovi: Postoji bar jedna knjiga.
Postuslovi: Prikazane su sve knjige.



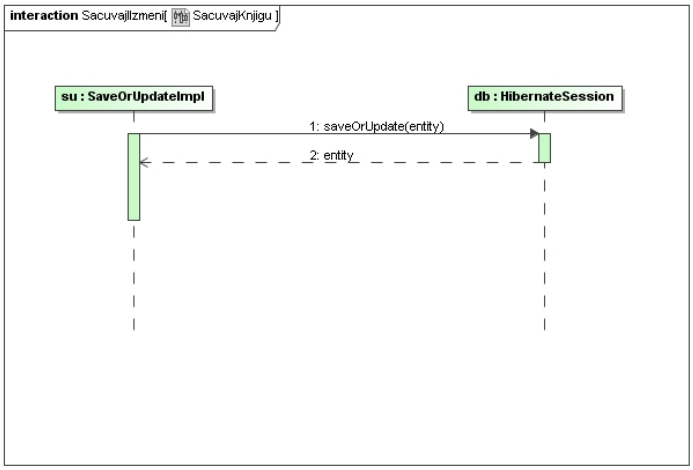
UG6: getAllBorrows
Operacija: List<BorrowBook> getAllBorrows(s:Student)
Veza sa SK: SK9
Preduslovi: Postoji iznajmljivanje za studenta.
Postuslovi: Prikazana sva iznajmljivanja.



UG7: loginStudent
Operacija: loginStudent(email:String, password:String)
Veza sa SK: SK1
Preduslovi: Postoji student sa traženim podacima.
Postuslovi: Student ulogovan.



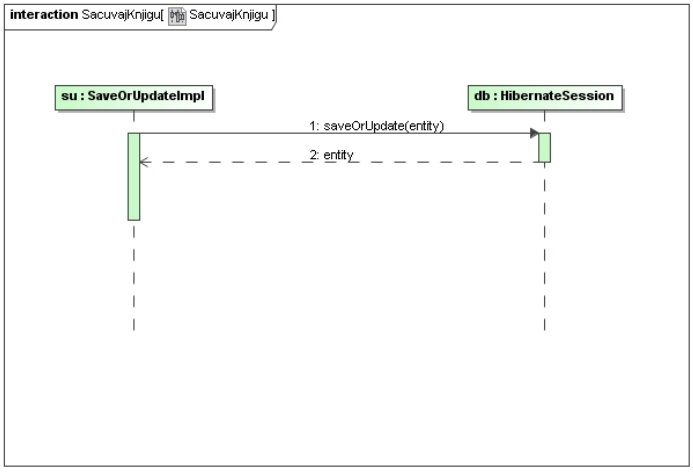
UG8: registerStudent
Operacija: registerStudent(s:Student)
Veza sa SK: SK2
Preduslovi: Postoji bar jedan registrovan fakultet.
Postuslovi: Registrovan student.



UG9: updateStudent
Operacija: updateStudent(s:Student)
Veza sa SK: SK3
Preduslovi: Postoji bar jedan student.
Postuslovi: Izmenjeni podaci o studentu.

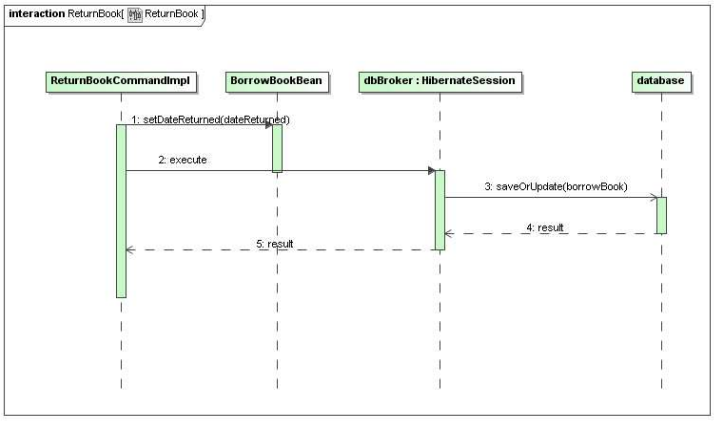
Ugovor8 i Ugovor9 koriste identičnu komandu koja čuva objekat bilo kog tipa (SaveOrUpdateImpl.java).

UG10: saveFaculty
Operacija: saveFaculty(f:Faculty)
Veza sa SK: SK10
Preduslovi:
Postuslovi: Sačuvan fakultet.



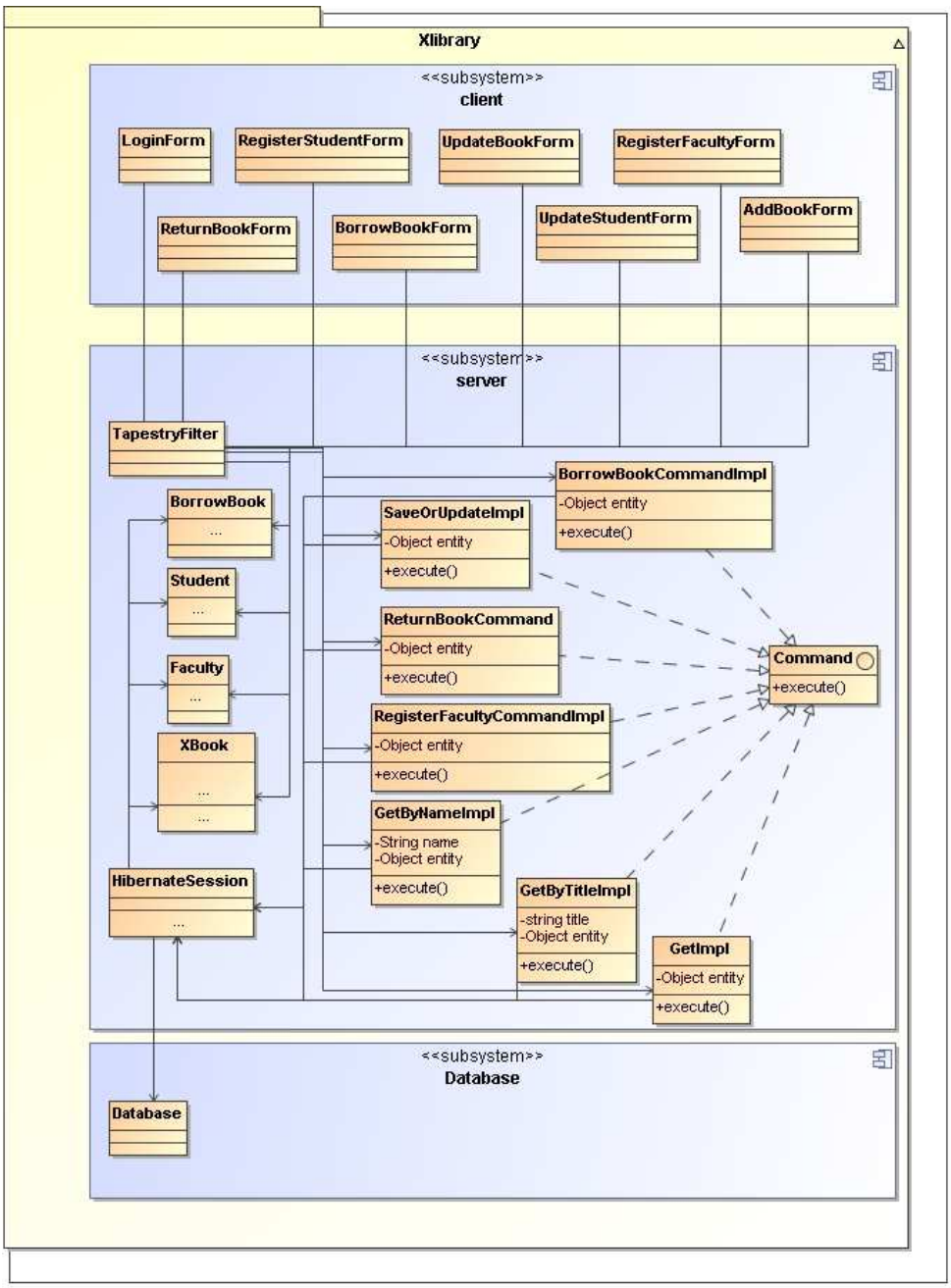
Ugovor10 koristi komandu koja čuva objekat bilo kog tipa (SaveOrUpdateImpl.java)

UG11: returnBook
Operacija: returnBook(borrow:BorrowBook)
Veza sa SK: SK10
Preduslovi: Postoji zadužena knjiga.
Postuslovi: Izmenjeni podaci o zaduženju i izmenjeni podaci o broju knjiga u biblioteci.



U studijskom primeru sistema biblioteke, 12 sistemskih operacija implementirano je pomoću samo 7 komandi, koje kao parametar primaju objekat sa kojim rade, a povratna vrednost zavisi od svrhe za koju su projektovane.

Nakon projektovanja sistemskih operacija i konceptualnog modela dobijamo izgled konačne strukture softverskog sistema (slika 23).



Slika 23: Struktura softverskog sistema Xlibrary

3.1.3.2.3. Database Broker

Database broker predstavlja jednu moguću realizaciju perzistentnog okvira. Za obezbeđivanje perzistentnosti moguće je koristiti različite implementacije. U radu je korišćen *Hibernate okvir*. U koknretnom studijskom primeru ulogu brokera uzima HibernateSession koji je zadužen za čuvanje promena u bazi podataka.

3.1.3.3. Projektovanje skladišta podataka

Na osnovu softverskih klasa strukture projektovali smo tabele (skladišta podataka) relacionog sistema za upravljanje bazom podataka. Umesto nas, Hibernate je sam napravio skript za bazu čitajući anotacije za entitete koje smo želeli da smestimo u bazu podataka.
Niže su dati skript za kreiranje baze podataka i relacioni model za informacioni sistem biblioteke.

```
create table BorrowBookBean (_id bigint not null, dateBorrowed
timestamp, dateReturned timestamp, book__id bigint, student__id
bigint, primary key (_id))

create table FacultyBean (_id bigint not null, name varchar(255), type
varchar(255), city varchar(255), primary key (_id))

create table StudentBean (_id bigint not null, name varchar(255),
email varchar(255), password varchar(255), dateOfAffiliation
timestamp, surname varchar(255), admin bit not null, faculty__id
bigint, primary key (_id))

create table XBookBean (_id bigint not null, dateProduced timestamp,
title varchar(255), authors varchar(255), ISBN varchar(255),
numberOfCopies integer not null, genre varchar(255), publisher
varchar(255), primary key (_id))
```

Skript za kreiranje HSql baze podataka

Niže je dat relacioni model:

Book (BookId, Title, Authors, Genre, Publisher, DateOfPublish)
Student: (StudentId, Name, Surname, Email, Password, FacultyId)
BorrowBook: (StudentId, BookId, DateBorrowed, DateReturned)
Faculty: (FacultyId, Name, City, Type)

Softverski sistem Xlibrary koristi HSQLDB – Hypersonic SQL Data Base za skladištenje podataka. Bazu generiše i njome upravlja Hibernate okvir, na osnovu anotacija koje se nalaze u Bean klasama i XML dokumenata. Ono što je bitno pomenuti, a vezano je za ogrničenja u bazi, odnosno nad kolonama tabele, je to da se postavljena ograničenja odnose na jedinstvenost podataka u određenim kolonama.

U klasi XBook ograničenje je na naslovu knjige – title, iz istog razloga, jer ne mogu da postoje dve knjige sa istim imenom:

```
@Table(name="XBook",
uniqueConstraints= @UniqueConstraint(columnNames={"ISBN"}))
```

U klasi Student ograničenje je na emailu studenta – email, jer je potrebno da svaki korisnik ima jedinstveni email.

```
@Table(name="Student",
uniqueConstraints= @UniqueConstraint(columnNames={"email"}))
```

3.1.3.4. Struktura korisničkog interfejsa

Korisnički interfejs pretstavlja realizaciju ulaza i/ili izlaza softverskog sistema. Korisnički interfejs se sastoji od:

1. Ekranske forme čija je odgovornost da:
   - prihvata podatke koje unosi aktor
   - prihvata događaje koje pravi aktor
   - poziva kontrolera grafičkog interfejsa, prosleđujući mu prihvaćene podatke

Svaku od navedenih odgovornosti ekranska forma obavlja pomoću grafičkih elemenata.

2. Kontroler KI koji je odgovoran da:
   - prihvati podatke od ekranske forme
   - konvertuje podatke (koji se nalaze u grafičkim elementima) u objekat koji pretstavlja ulazni argument SO koja će biti pozvana
   - šalje zahtev za izvršenje SO do aplikacionog servera (softverskog sistema).
   - prihvata objekat(izlaz) softverskog sistema koji nastaje kao rezultat izvršenja SO
   - konvertuje objekat u podatke grafičkih elemenata
   - prikaže podatke na ekranskoj formi

Kontroler korisničkog interfejsa je implementiran putem Tapestry mehanizma, koji je zadužen za sve poslove kreiranja i upravljanja događajima vizuelnih elemenata aplikacije.

3.1.3.6. Projektovanje korisničkog interfejsa

Korisnički interfejs predstavlja realizaciju ulaza i/ili izlaza softverskog sistema. Korisnički interfejs sastoji se od ekranskih formi i kontrolera korisničkog interfejsa. Ekranske forme i kontroler korisničkog interfejsa realizovani su korišćenjem *Tapestry* okvira.

Na slici 24 je prikazana početna stranica aplikacije za ulogovanog studenta. Student se može kretati kroz aplikaciju i koristiti njene funkcionalnosti praćenjem linkova koji su sa leve strane početne stranice, dok se u srednjem delu prikazuju stranice na kojima korisnik trenutno radi. Kao što postoji korisnički deo aplikacije, tako postoje i funkcionalnosti vezane za

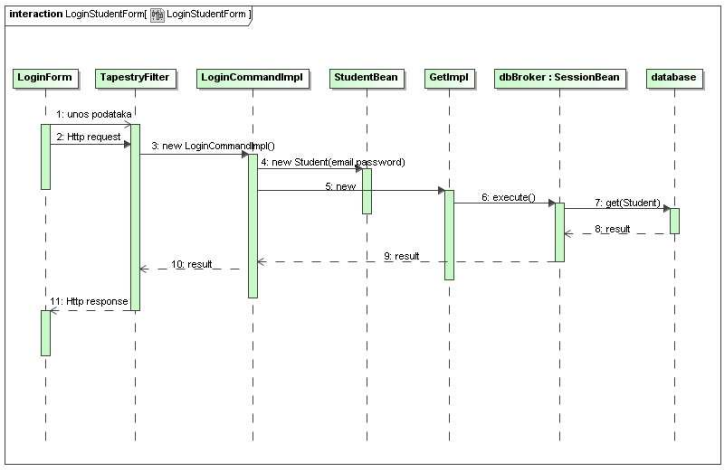
administraciju aplikacije. Na slici 25 je prikazana početna stranica administratorskog dela aplikacije.

Slika 24: Izgled glavnog menija - krajnji korisnik(student)

Slika 25: Izgled glavnog menija - krajnji korisnik(administrator)

SK1: Logovanje korisnika

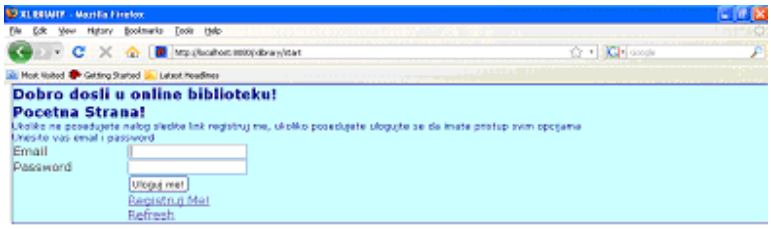
Naziv: Logovanje korisnika
Aktori: Korisnik
Ucesnici: Korisnik i sistem
Preduslov: Sistem je uključen i prikazuje formu za Logovanje korisnika.



Slika 26: Sekvencni dijagram logovanja studenta

Osnovni scenario:

1. Korisnik unosi svoj email i password (APUSO)



2. Korisnik poziva sistem da uloguje studenta (APSO)

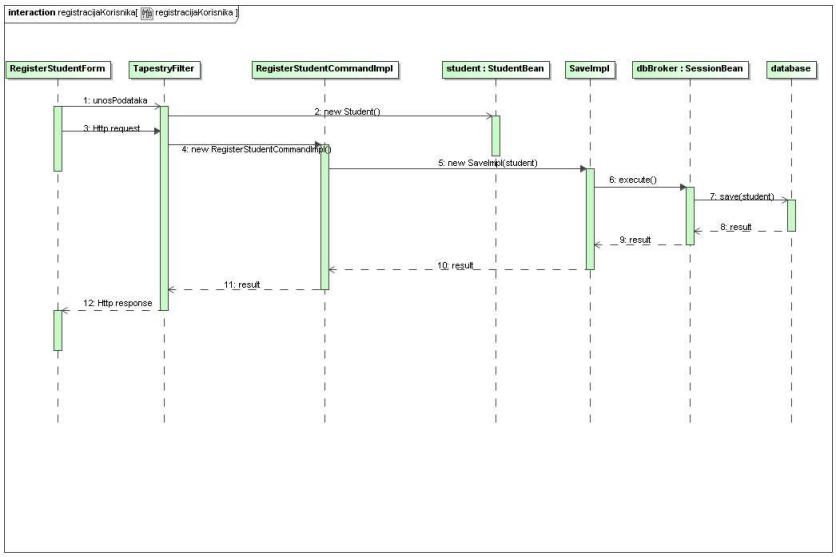
Opis akcije: Po unosu validnog emaila i passworda korisnik pritiska dugme "Uloguj me" ili pritiska tiplu "Enter" nakon čega se poziva sistemska operacija LoginStudent() i nakon uspešne provere podataka studentu se omogućava pristup glavnom meniju.

3. Sistem proverava unete podatke (SO)
4. Sistem obaveštava korisnika o uspešnom logovanju (IA)

Alternativna scenarija:

4.1. Sistem obaveštava korisnika porukom o neuspešnom logovanju i zahteva da ponovo unese podatke.

SK2: Registracija studenata

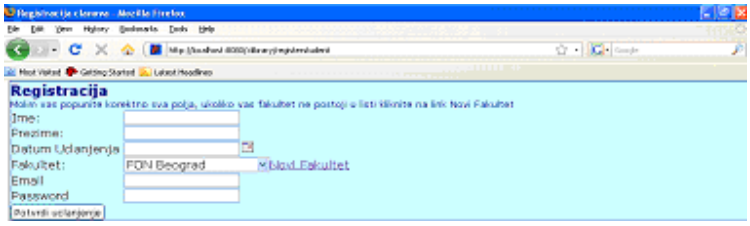


Slika 28: Sekvencni dijagram za slučaj korišćenja registracije studenta

Naziv: Registracija studenata
Aktori : Korisnik (student)
Učesnici: Korisnik i sistem
Preduslovi: Sistem je uključen, prikazuje formu za registraciju studenata.

Osnovni scenario:

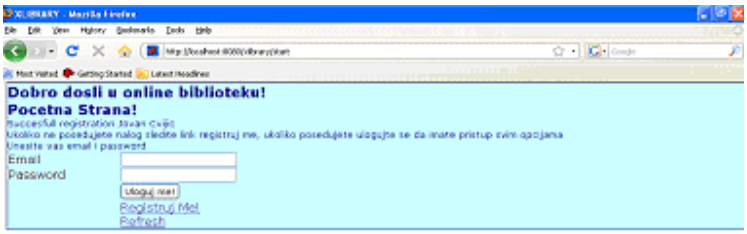
1. Korisnik popunjava potrebne podatke (APUSO)



2. Korisnik poziva sistem da ga registruje (APSO)

   Opis akcije: Korisnik nakon unosa ličnih podataka pritiska dugme "Registruj me" nakon čega se poziva sistemska operacija SaveStudent() koja upisuje podatke o studentu u bazu podataka.

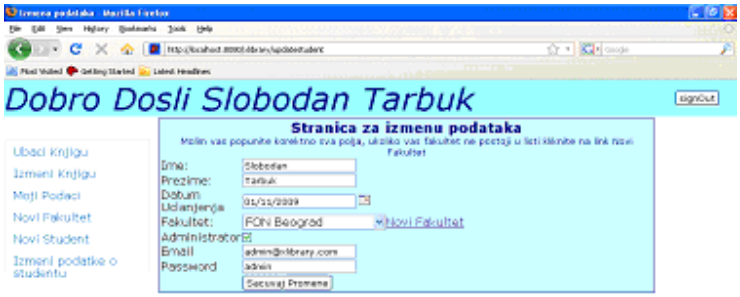
3. Sistem registruje studenta (SO)
4. Sistem prikazuje poruku o uspešnoj registraciji (IA)



Alternativna scenarija:
4.1. Sistem javlja grešku, ne može da registruje studenta i ispisuje poruku “Molim vas unesite sve podatke ispravno”.

SK 3: Izmena podataka o studentu

Naziv: Izmena podataka o studentu
Aktori: Korisnik (student)
Učesnici: Korisnik i sistem
Preduslovi: Student je ulogovan, sistem je uključen i prikazuje *formu* za izmenu podataka o studentu.

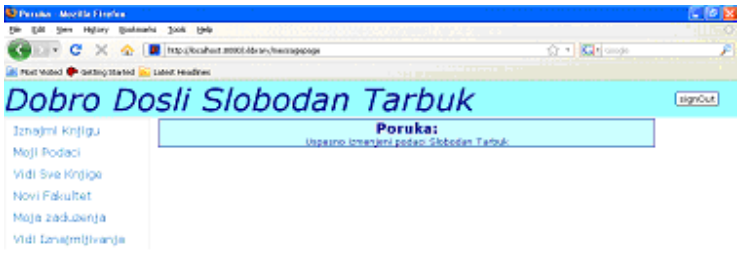


Osnovni scenario:

1. Korisnik popunjava potrebne podatke (APUSO)
2. Korisnik poziva sistem da sačuva promene (APSO)

   Opis akcije: Korisnik klikom na hyperlink "Moji podaci" poziva sistemsku operaciju SaveStudent() koja čuva izmenjene podatke o trenutno ulogovanom studentu.

3. Sistem čuva promene (SO)
4. Sistem prikazuje poruku o uspesnom pamćenju promena (IA)



Alternativna scenarija:

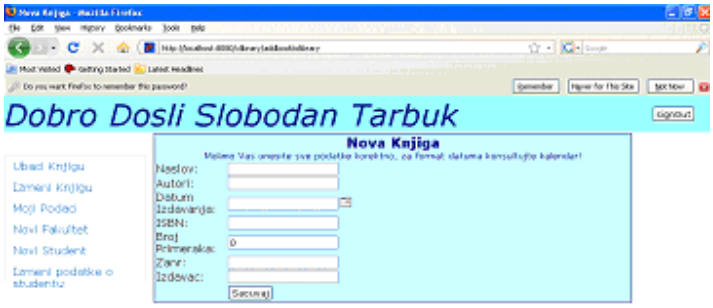
4.1. Sistem javlja grešku, prikazuje poruku “Neuspešno čuvanje”.

SK4: Unos knjige u biblioteku

Naziv: Unos knjige u biblioteku
Aktori: Korisnik
Učesnici: Korisnik i sistem
Preduslovi: Student je ulogovan, sistem uključen, prikazuje formu za unos nove knjige.

Osnovni scenario:

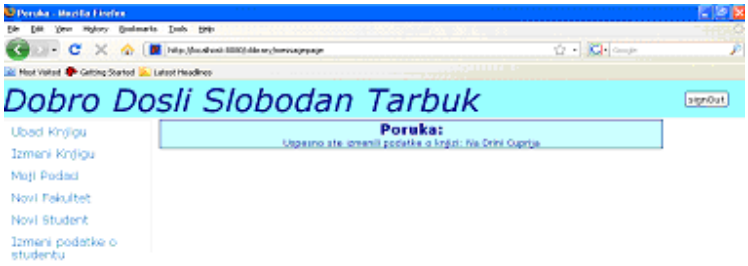
1. Korisnik unosi podatke o knjizi (APUSO)



2. Korisnik poziva sistem da sačuva knjigu (APSO)

   Opis akcije: Korisnik klikom na dugme "Sačuvaj" poziva sistemsku operaciju SaveBook() koja čuva podatke o knjizi u bazu podataka.

3. Sistem čuva knjigu (SO)
4. Sistem prikazuje poruku o uspešnom čuvanju (IA)



Alternativna scenarija:

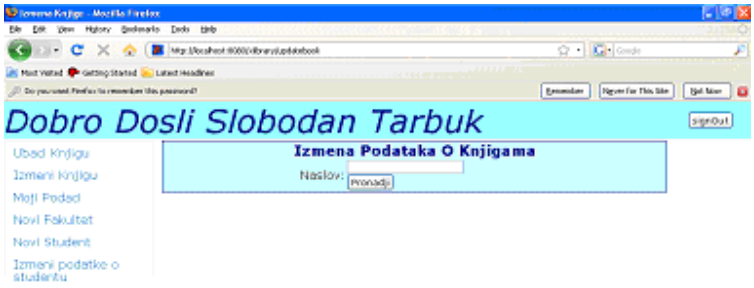
4.1. Sistem javlja grešku, ne može da sačuva novu knjigu i ispisuje poruku “Greška prilikom čuvanja”

SK 5: Izmena podataka o knjigama

Naziv: Izmena podataka o studentu
Aktori: Korisnik (student)
Učesnici: Korisnik i sistem
Preduslovi: Student je ulogovan, sistem je uključen i prikazuje formu za izmenu knjige.

Osnovni scenario

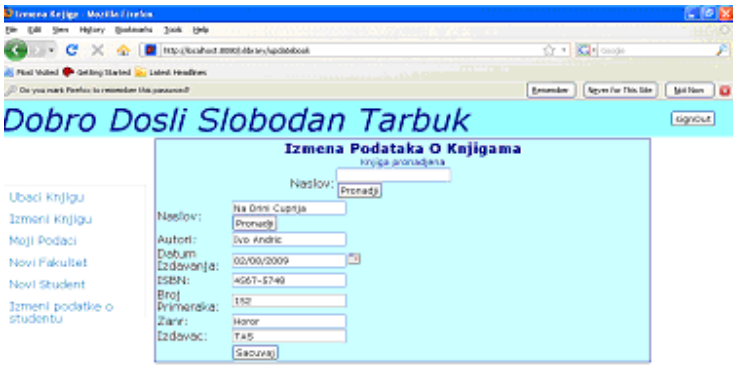
1. Korisnik unosi naziv knjige (APUSO)



2. Korisnik poziva sistem da pronađe knjigu (APSO)

   Opis akcije: Korisnik pritiskom na dugme "Pronadji" poziva sistemsku operaciju GetBook() koja pronalazi traženu knjigu u bazi podataka.

3. Sistem pronalazi knjigu (SA)
4. Sistem prikazuje traženu knjigu (IA)



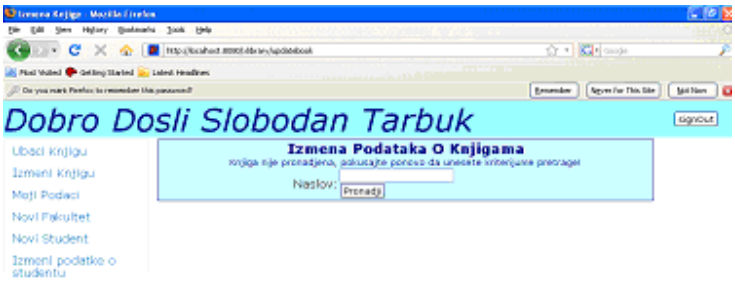
5. Korisnik popunjava potrebne podatke (APUSO)
6. Korisnik poziva sistem da sačuva promene (APSO)

   Opis akcije: Korisnik pritiskom na dugme "Sacuvaj" poziva sistemsku operaciju SaveBook() koja čuva izmenjene podatke o knjizi u bazu podataka.

7. Sistem čuva promene (SO)
8. Sistem prikazuje poruku o uspešnom pamćenju promena (IA)

Alternativna scenarija:

4.1. Sistem javlja grešku, prikazuje poruku “Ne postoji tražena knjiga”.



8.2. Sistem javlja grešku, prikazuje poruku “Neuspešno čuvanje”.

SK 6: Pregled svih knjiga

Naziv: Pregled svih knjiga
Aktori: Korisnik
Učesnici: Korisnik i sistem
Preduslovi: Student je ulogovan, sistem je uključen, student ulogovan, i prikazuje početnu formu.

Osnovni scenario:

1. Korisnik poziva sistem prikaže knjige (APSO)

   Opis akcije: Korisnik u klikom na hyperlink "Vidi Sve Knjige" u okviru glavnog menija poziva sistemsku operaciju GetAllBooks() koja kao rezultat vraća kolekciju knjiga koje se nalaze u bazi podataka.

2. Sistem pronalazi sve knjige (SO)
3. Sistem prikazuje sve knjige (IA)

Alternativna scenarija:

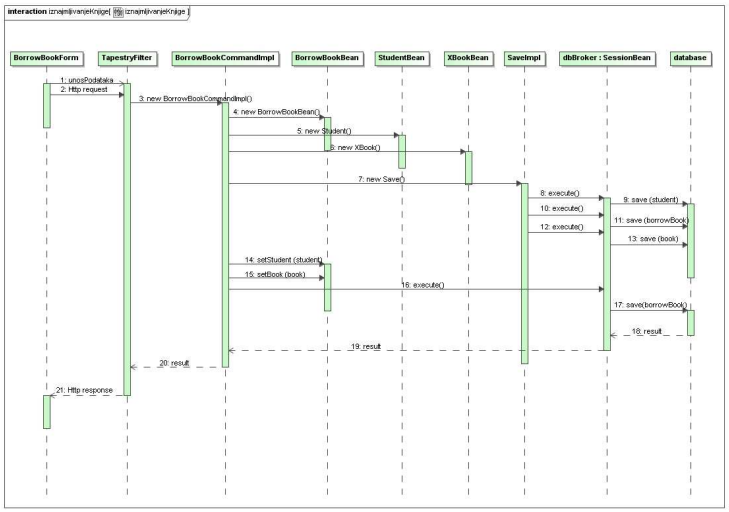
3.1. Sistem javlja grešku, ne može da prikaže knjige i prikazuje poruku “Greška prilikom prikazivanja”.

SK 7: Iznajmljivanje knjige

Naziv: Iznajmljivanje knjige
Aktori: Korisnik (student)
Učesnici: Korisnik i sistem
Preduslovi: Student je ulogovan, sistem je uključen i prikazuje *formu* za iznajmljivanje knjiga.

Slika 30: Iznajmljivanje knjige

Na slici 31 prikazan je sekvencni dijagram za slučaj korišćenja Iznajmljivanje knjige. Nakon poziva korisnika da iznajmi knjigu, sistemu se šalje Http zahtev koji TapestryFilter objekat obrađuje, i pravi novi objekat klase BorrowBookCommandImpl. Navedena klasa kreira nove objekte klase BorrowBookBean, StudentBean, XBookBean i SaveImpl. Komanda pozivima



Slika 31: Sekvencni dijagram za slučaj korišćenja iznajmljivanje knjige

svoje metode execute čuva u bazi novostvoreni objekat borrowBook, a objekte student i book modifikuje ukoliko je došlo do nekih promena. Nakon ovoga objekat klase BorrowBookCommandImpl objektu borrowBook setuje vrednosti student, book i poziva svoju metodu execute koja poziva crud(osnovnu) operaciju čuvanja objekta u bazu. Na formu se preko TapestryFilter objekta šalje rezultat o iznajmljivanju knjige u vidu Http odgovora.

Osnovni scenario:

1. Korisnik popunjava polja i bira knjigu iz liste koju želi da iznajmi (APUSO)
2. Korisnik poziva sistem da sačuva iznajmljivanje (APSO)

   Opis akcije: Korisnik klikom na dugme "Iznajmi knjigu" poziva sistemsku operaciju BorrowBook() koja pravi novo zaduženje jedne knjige za trenutno ulogovanog studenta i čuva ga u bazi podataka.

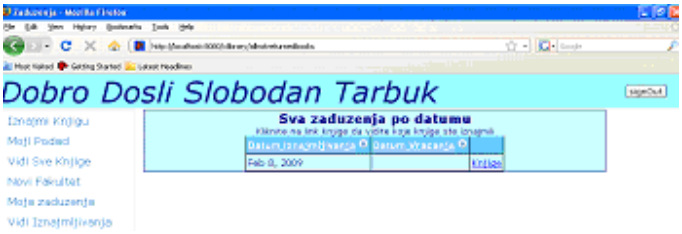
3. Sistem čuva iznajmljivanje (SO)
4. Sistem prikazuje poruku o uspešnom čuvanju (IA)

Alternativna scenarija:

4.1. Sistem javlja grešku, ne može da sačuva iznajmljivanje i prikazuje poruku “Greška prilikom prikazivanja rezultata”.

SK 8: Vraćanje knjige

Naziv: Vraćanje knjige
Aktori: Korisnik
Učesnici: Korisnik i sistem
Preduslovi: Student je ulogovan, sistem je uključen i prikazuje formu sa pregledom svih zaduženja.



Osnovni scenario:

1. Korisnik poziva sistem da mu prikaže knjigu za dato zaduženje (APSO)

   Opis akcije: Korisnik pritiskom na hyperlink "Knjiga" poziva sistemsku operaciju GetBooksForBorrow() koja pronalazi zaduženu knjigu za trenutno ulogovanog studenta.

2. Sistem pronalazi knjigu (SO)

3. Sistem prikazuje knjigu (IA)

4. Korisnik poziva sistem da razduži zaduženje (APSO)

   Opis akcije: Korisnik pritiskom na dugme "Vrati Knjigu" poziva sistemsku operaciju ReturnBook() koja menja status zaduženja knjige na "vraćena" i čuva promene u bazi podataka.

5. Sistem razdužuje knjigu(SO)
6. Sistem prikazuje poruku o uspešnom razduživanju knjige (IA)

Alternativna scenarija:

2.1. Sistem ne može da prikaže knjigu i prikazuje poruku “Neuspežan prikaz”

4.1. Sistem ne može da razduži knjigu i prikazuje poruku “Došlo je do greške prilikom razduživanja knjige”

SK 9: Pregled svih iznajmljivanja

Naziv: Pregled svih iznajmljivanja
Aktori: Korisnik
Učesnici: Korisnik i sistem
Preduslovi: Student je ulogovan, sistem je uključen i prikazuje početnu *formu* za ulogovanog studenta.

Osnovni scenario:

1. Korisnik poziva sistem prikaže prethodna iznajmljivanja (APSO)

   Opis akcije: Korisnik pritiskom na hyperlink "Sva iznajmljivanja" poziva sistem da prikaže sva prethodna iznajmljivanja za trenutno ulogovanog korisnika.

2. Sistem pronalazi sva iznajmljivanja (SO)

3. Sistem prikazuje sva iznajmljivanja (IA)

Alternativna scenarija:

3.1. Sistem javlja grešku, ne može da prikaže sva iznajmljivanja i prikazuje poruku "Greška prilikom prikazivanja".

SK10: Unos fakulteta

Naziv: Unos fakulteta
Aktori: Korisnik
Učesnici: Korisnik i sistem
Preduslovi: Student je ulogovan, sistem je uključen, prikazuje formu za unos fakulteta.

Osnovni scenario:

1. Korisnik unosi potrebne podatke (APUSO)

2. Korisnik poziva sistem da sačuva podatke (APSO)

   Opis akcije: Korisnik pritiskom na dugme "Dodaj Fakultet" poziva sistemsku operaciju SaveFaculty() koja dodaje novi fakultet u bazu podataka.

3. Sistem čuva fakultet (SO)
4. Sistem prikazuje poruku o uspešnom čuvanju (IA)

Alternativna scenarija:

4.1. Sistem javlja grešku, ne može da sačuva fakultet i ispisuje poruku "Greška prilikom čuvanja".

3.1.4. Implementacija

Implementacija studijskog primera izvršena je u programskom jeziku Java. Kao razvojno okruženje korišćen je razvojni alat Eclipse SDK 3.3.1. Sistem za upravljanje bazom podataka je HSQL. Kao alat za build aplikacije korišćen je Maven 2.0.7. Server pomoću koga se pokreće aplikacija je embeded Jetty aplikacioni server.

Projektovani software je u fazi implementacije razdvojen na *client-server* arhitekturu, tako što su sve ekranske forme i kontroler korisničkog interfejsa nalaze na strani *client*-a (Tapestry okvir), dok se na strani *server*-a nalazi aplikaciona logika i skladište podataka tj. *softverski sistem*.

3.1.5. Testiranje

Konstantno pisanje testova je jedna od glavnih odlika izrade kvalitetnog softvera. Testovi se pišu i pre nego što su implementirane metode klasa koje se testiraju, da bi se postavili uslovi koje treba da zadovolje implementirane metode.

Za testiranje softvera korišćeni su Java Unit testovi, Java okvir za pisanje i izvršavanje unit testova, TestNG. TestNG je okvir za testiranje inspirisan na JUnit i NUnit alatima, ali on uvodi neke nove funkcionalnosti koje ga čine moćnijim i lakšim za upotrebu, kao na primer:

- JDK 5 anotacije
- Fleksibilna test konfiguracija
- Podrška za data driven testove (testovi koji se izvršavaju nad tabelama, posebno za svaki red)
- Moćni model izvršavanja
- Podržan je mnogim alatima i plug-in ovima (Eclipse, IDEA, Maven...)

TestNG je dizajniran da pokrije sve kategorije testova: unit testove, testove funkcionalnosti, integracione testove. [8]

```
public class StudentBeanTest {
      private StudentBean studentBeanUnderTest;
      @BeforeMethod
      public void startUp(){
            studentBeanUnderTest = new StudentBean();
```

```
            studentBeanUnderTest.setBorrows(new
LinkedList<BorrowBook>());
            faculty = new FacultyBean();
            studentBeanUnderTest.setFaculty(faculty);
      }
      @Test
      public void testNameSetNull(){
            String name = "Jovana";
            studentBeanUnderTest.setName(name);
            assertEquals(name, studentBeanUnderTest.getName());
            studentBeanUnderTest.setName(null);
            assertNull(studentBeanUnderTest.getName());
      }
}
```

Naveden je primer jednog jednostavnog TestNG testa koji delom testira funkcionalnosti metoda klase StudentBean. Anotacijom @Test se obeležava da ta metoda predstavlja test metodu, koju će kompajler prepoznati i izvršiti. Anotacijom @BeforeMethod se obeležava metoda u kojoj inicijalizujemo početne vrednosti objekata koje testiramo, kompajler pročita anotaciju, inicijalizuje stanje i izvrši test. Ovakvim testovima testirane su sve funkcionalnosti studijskog primera i tako unapred izbegnute greške koje se mogu javiti u kasnijim fazama razvoja softvera i integracije sa drugim delovima.

4. ZAKLJUČAK

U toku izrade ovog rada upoznao sam se sa alatima i okvirima koji znatno olakšavaju razvoj softvera i omogućavaju brži razvoj aplikacija, bez preterane brige o detaljima. Spoznao sam mogućnosti Tapestry okvira koji se tiču veoma lake upotrebe i izrade komponenti, prikazom podataka, validacijom formi. Takođe smo se upoznali sa prednostima Spring okvira i DependencyInjection-a. Upoznao sam se i sa mogućnostima objektno-relacionog mapera Hibernate, pomoću koga se može izrađivati softver ne vodeći mnogo računa kako se sve to negde smešta i čuva, i na koji se to način vraća do programera, iskorišćene su sve prednosti navedenih tehnologija.

Ukratko su opisane njihove osnovne osobine i namena, prednosti i mane, a fokus je bio na njihovim osnovnim funkcionalnostima koje su iskorišcene u aplikaciji studijskog primera web aplikacije za biblioteku. Opisan je način konfigurisanja preko eksternih XML konfiguracionih dokumenata. Poslovi objektno-relacionog preslikavanja kao i poslovi komunikacije sa bazom podataka su opisani u poglavlju koji opisuje Hibernate.

Korišćenjem navedenih okvira omogućeno je brže i lakše pisanje aplikacija. Korišćenjem Spring okvira rešio sam probleme povezivanja Java objekata i definisanja njihovih zavisnosti. Hibernate okvir je omogućio bezbedno i jednostavno čuvanje i prikazivanje podataka, dok korišćenjem Tapestry okvira sam naučio da veoma jednostavno prikažem na ekranu sve ono što je definisano korisničkim zahtevima. U velikoj meri je pomoglo i konstantno pisanje testova u toku izrade aplikacije, jer je time izbegnuto otkrivanje grešaka u kasnijim fazama razvoja i njihovo teško ispravljanje.

Prilikom izrade aplikacije nailazio sam na probleme. Problemi su se uglavnom odnosili na sitne greške u kodu koje su nastajale prilikom brzog pisanja. Ovi problemi su uočeni na vreme u ranim fazama razvoja aplikacije, jer je konstantno pisanje testova u toku razvoja softvera u mnogome olakšalo njihovo pronalaženje. Rešavanjem tih problema došao sam do zaključka da dobro napisana dokumentacija i dobro napisani testovi u mnogome olakšavaju razvoj aplikacije, i da se tako mogu izbeći mnoge sitne greške koje mogu da nas koštaju sati bespotrebnog traženja u kasnijim fazama razvoja softvera.

Aplikacija je pisana u *Eclipse* razvojnom okruženju, i korišćenjem *Jetty* aplikacionog servera, koji zajedno, uz pomenute tehnologije, predstavljaju dobar osnov za razvijanje poslovnih i web aplikacija.

5. LITERATURA

[1] Dr Siniša Vlajić: *Projektovanje programa (Skripta)*, FON, Beograd 2006.
[2] Dr Siniša Vlajić, Dušan Savić, Vojislav Stanojević, Ilija Antović, Miloš Milić: *Projektovanje softvera - Napredne java tehnologije* FON, Beograd 2008.
[3] http://www.onjava.com/pub/a/onjava/2005/09/21/what-is-hibernate.html
[4] Manning – Spring in action 2nd Edition – Craig Walls with Ryan Breidenbach
[5] Tapestry 5 Building *Web* Applications - Alexander Kolesnikov
[6] Hibernate quickly - PATRICK PEAK, NICK HEUDECKER
[7] Hibernate In Action – Christian Bauer, Gavin King
[8] http://testng.org/doc/
[9] http://www.javaenum.com/tapestry/
[10] http://arizonacommunity.com/articles/java_32001.shtml
[11] Addison-Wesley - 1995 - Erich Gamma - Design Patterns – GoF
[12] http://java.sun.com/
[13] http://www.allapplabs.com/hibernate/hibernate_tutorials.htm

BESPLATNI GOTOVI SEMINARSKI, DIPLOMSKI I MATURSKI RAD.

RADOVI IZ SVIH OBLASTI, POWERPOINT PREZENTACIJE I DRUGI EDUKATIVNI MATERIJALI.

WWW.SEMINARSKIRAD.ORG

WWW.MATURSKIRADOVI.NET

WWW.MATURSKI.NET

WWW.SEMINARSKIRAD.INFO

WWW.MATURSKI.ORG

WWW.ESSAYSX.COM

WWW.FACEBOOK.COM/DIPLOMSKIRADOVI

NA NAŠIM SAJTOVIMA MOŽETE PRONAĆI SVE, BILO DA JE TO SEMINARSKI, DIPLOMSKI ILI MATURSKI RAD, POWERPOINT PREZENTACIJA I DRUGI EDUKATIVNI MATERIJAL. ZA RAZLIKU OD OSTALIH MI VAM PRUŽAMO DA POGLEDATE SVAKI RAD, NjEGOV SADRŽAJ I PRVE TRI STRANE TAKO DA MOŽETE TAČNO DA ODABERETE ONO ŠTO VAM U POTPUNOSTI ODGOVARA. U BAZI SE NALAZE GOTOVI SEMINARSKI, DIPLOMSKI I MATURSKI RADOVI KOJE MOŽETE SKINUTI I UZ NJIHOVU POMOĆ NAPRAVITI JEDINSTVEN I UNIKATAN RAD. AKO U BAZI NE NAĐETE RAD KOJI VAM JE POTREBAN, U SVAKOM MOMENTU MOŽETE NARUČITI DA VAM SE IZRADI NOVI, UNIKATAN SEMINARSKI ILI NEKI DRUGI RAD RAD NA LINKU IZRADA RADOVA. PITANjA I ODGOVORE MOŽETE DOBITI NA NAŠEM FORUMU ILI NA MATURSKIRADOVI.NET@GMAIL.COM