Aplikacija za praćenje studentskih praksi u tehnologiji web usluga

Milić, Matej

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Stručni studij

APLIKACIJA ZA PRAĆENJE STUDENTSKIH PRAKSI U TEHNOLOGIJI WEB USLUGA

Završni rad

Matej Milić

Osijek, 2017.

SADRŽAJ

1. UVOD

Dolaskom studenata na praksu u određenu tvrtku stavlja se veliki pritisak na mentora koji mora odvojiti vrijeme da bi pomogao studentu pri izradi projekta. U svrhu olakšavanja posla mentoru osmišljena je aplikacija te je izrađena web usluga koja omogućuje praćenje napretka rada studenata . Studenti i mentori unutar tvrtke imati će svoje korisniče račune koji će služiti za međusobnu interakciju i tako poboljšati učinkovitost studenta, odnosno olakšati mentorov posao.

Aplikacija će služiti za lakšu, bolju i bržu komunikaciju između studenta i mentora. Korisnik aplikacije ovisno o tome je li student ili mentor moći će:

- Dodavati, uređivati i predavati dnevnike rada
- Pregledavati dnevnike rada
- Dodavati komentare, uređivati i brisati ih
- Imati pregled nad osobnim podacima za dodijeljenog mentora ili studenta.

Cilj završnog rada je prikazati stečena znanja iz područja objektno orijentiranog programiranja i baza podataka. Jezici korišteni za izradu aplikacije su Java i SQL (*engl. Structured Query Language*). Pri izradi web servisa potrebno je poznavanje XML (*engl. Extensible Markup Language*) jezika za označavanje podataka i SOAP-a koji služe za komunikaciju klijenta sa web uslugom. Poglavlje 2 opisuje tehnologije korištene za izradu rada. Poglavlje 3 opisuje definiranje korisničkih zahtjeva i specifikacija aplikacije. Poglavlje 4 opisuje razvoj aplikacije. Poglavlje 5 zaključuje rad.

2. PREGLED KORIŠTENIH TEHNOLOGIJA

2.1. Java programski jezik

Java je objektno orijentirani programski jezik koji je zasnovan na sintaksi programskog jezika C++, ali za razliku od njega cijeli kod se nalazi unutar klasa, a svaki tip podatka je objekt osim primitivnih tipova (*integer, float, boolean i char*). Najveća prednost Jave je što je neovisna o platformi na kojoj se koristi, odnosno može se koristiti na bilo kojem operacijskom sustavu koji podržava Javu. Zbog toga je jedan od najkorištenijih programskih jezika na svijetu.

Osnovna svojstva Jave su [1]:

- Objektno orijentiran programski jezik
- Jednostavnost
- Neovisnost o arhitekturi računala
- Dinamičnost
- Sigurnost.

Ključ koji omogućuje Javi da riješi probleme prenosivosti i sigurnosti je to što izlaz Java kompajlera nije izvršni kod, već *bytcode*. *Bytcode* je visoko optimizirani set instrukcija kojega izvršava Java *run-time* sustav, koji se zove *Java Virtual Machine* (JVM) [2]. JVM je dizajniran kao prevoditelj za *bytcode*. Činjenica da Java program izvršava JVM omogućava da se Java program može izvršiti na bilo kojoj platformi na kojoj je JVM instaliran.

Java dolazi sa već unaprijed napisanim programima odnosno API (*engl. Application programming interface*). API je kolekcija već napisanih paketa, klasa i sučelja koji se sastoje od svojih konstruktora i ostalih metoda [3]. Postoje službeni API-i koji se nalaze u *Java Development Kit-u* i neslužbeni koji se mogu naknadno preuzeti po potrebi.

2.2. XML opisni jezik

XML je opisni jezik koji služi za razmjenu podataka između programa, ljudi i računala lokalno i preko mreže [4]. Jedan je od najraširenijih opisnih jezika u upotrebi, a sam po sebi je vrlo sličan HTML-u (*engl. HyperText Markup Language*). Najveća razlika između HTML-a i XML-a je ta što XML dolazi bez unaprijed definiranih oznaka. Također HTML služi za prikaz podataka i njihov izgled dok XML služi za prijenos i definiranje informacija. XML sam po sebi ne radi ništa,

nego prenosi informacije koje se nalaze unutar oznaka koje su definirane tako da opisuju informaciju. Primjer kôda vidljiv u programskom kôdu 2.1. prikazuje izgled XML poruke koju klijent šalje web usluzi. Imena oznaka predstavljaju parametre koji se šalju, a uz njih je još definiran i tip podatka koji je u ovom slučaju *string*.

```
<soapenv:Body>
<stud:editInfosoapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding
/">
     <userID xsi:type="xsd:string">?</userID>
     <businessMobile xsi:type="xsd:string">?</businessMobile>
     <privateMobile xsi:type="xsd:string">?</privateMobile>
     <businessMail xsi:type="xsd:string">?</businessMail>
     <privateMail xsi:type="xsd:string">?</privateMail>
</stud:editInfo>
</soapenv:Body>
```

Programski kôd 2.1. – Primjer XML kôda

2.3. SOAP komunikacijski protokol

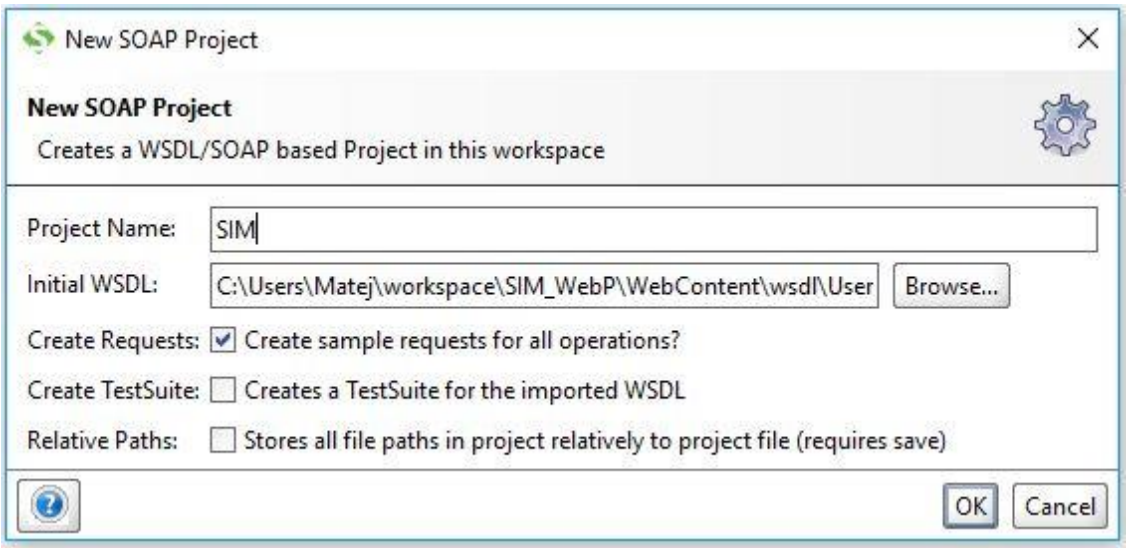
SOAP (*engl. Simple Object Access Protocol*) je komunikacijski protokol zasnovan na XML-u i služi za razmjenu poruka između računala. Razvijen je kako bi se omogućila jednostavna komunikacija tekstualnim sadržajem preko HTTP (*engl. HyperText Transfer Protocol*) komunikacijskog protokola koji je prilagođen upravo razmjeni tekstualnih sadržaja [5]. SOAP omogućuje komunikaciju između aplikacija koje se pokreću na različitim operacijskim sustavima i nije ovisan o programskom jeziku.

SOAP poruku čine slijedeći dijelovi (slika 2.2.):

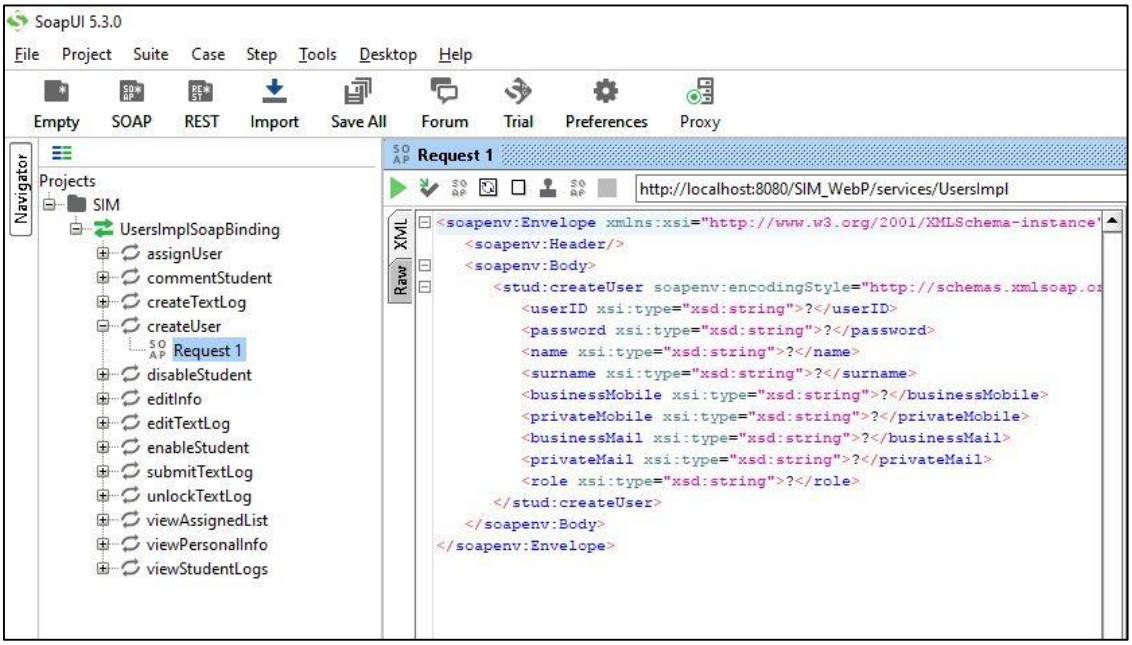
- Omotnica (*engl. Envelope*) koja prikazuje i definira da je XML dokument SOAP poruka
- Zaglavlje (*engl. Header*) je neobavezni dio, a može sadržavati opis dodatnih atributa koji su bitni pri obradi poruke
- Tijelo (*engl. Body*) sadrži podatke koji se šalju
- Greška (*engl. Fault*) je neobavezni dio koji sadrži informacije o pogrešci koja se dogodila tokom obrade poruke. Nalazi se unutar tijela poruke.

Slika 2.1. – Dijelovi SOAP poruke

Za rad sa SOAP komunikacijskim protokolom vrlo je popularan program SoapUI. To je program otvorenog koda koji služi za testiranje web usluga. Pri izradi aplikacije korišten je za testiranje i provjeru poslanih poruka i primljenih poruka od strane web usluge. Nakon pokretanja SoapUI programa potrebno je napraviti novi projekt odabrati WSDL (*engl. Web Services Description Language*) dokument (slika 2.2.) koji sadrži lokaciju web usluge i metode koje ona sadrži. WSDL je opisni jezik koji se bazira na XML-u i služi za opis funkcionalnosti koje web usluga pruža [6]. Nakon što je projekt napravljen otvara se popis svih dostupnih metoda i zahtjeva koji se šalju web usluzi (slika 2.3.).



Slika 2.2. – Izrada novog SOAP projekta



Slika 2.3. – Izgled implementiranog projekta i zahtjeva

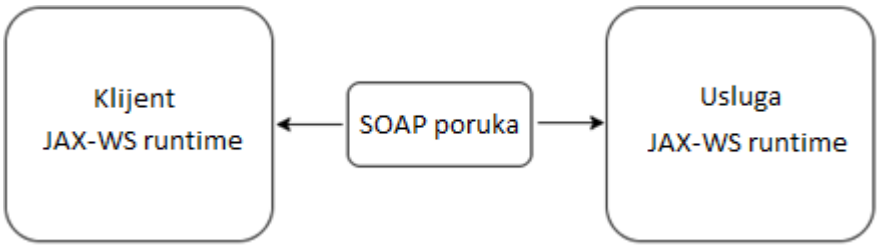
2.4. Web usluga

Web usluga (*engl. Web service*) je aplikacija koja je dostupna na internetu, a može joj se pristupiti koristeći aplikaciju koja je napisana u bilo kojem programskom jeziku i sa bilo kojeg operacijskog sustava. Web usluge koriste XML, SOAP i WSDL kako bi se omogućila komunikacija između različitih aplikacija.

Svaka tehnologija ima svoju zadaću:

- XML služi za označavanje informacija i definiranje parametara
- SOAP prenosi poruke
- WSDL definira lokaciju i dostupnost web usluge.

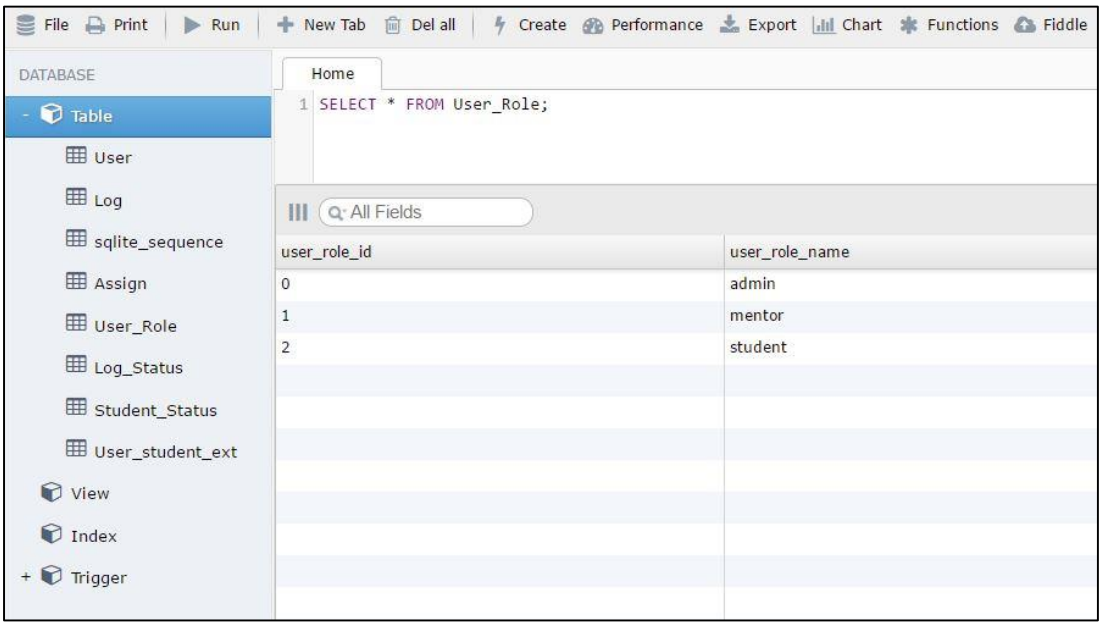
Za potrebu izrade web usluge korišten je Java API JAX-WS. To je tehnologija koja omogućuje jednostavno razvijanje web servisa koristeći Java programski jezik. JAX-WS se brine za generiranje i raščlanjivanje SOAP poruka (slika 2.4.). Klijent šalje parametre metodi web usluge i ovisno o zahtjevu, usluga vraća određeni odgovor klijentu. Ove poruke i zahtjevi preneseni su kao SOAP poruke.



Slika 2.4. – Komunikacija između JAX-WS klijenta i usluge

2.5. SQL programski jezik i baza podataka

SQL je programski jezik koji služi za izradu i upravljanje sa relacijskom bazom podataka. Omogućuje stvaranje tablica, njihovo popunjavanje i uređivanje. Za potrebu izrade aplikacije potrebna je baza podataka koja sadrži podatke o korisniku i podatke o dnevnicima rada, a korišten je popularan *engine* SQLite. Za SQLite bazu podataka nije potreban poslužitelj već se veže uz samu aplikaciju kao posebna datoteka. SQLite baza podataka zahtijeva malo ili nimalo administracije što ju čini idealnim izborom za ugradnju u sprave ili servise koji trebaju raditi automatski i bez ljudske podrške [7]. Za grafički prikaz baze podataka korištena je web aplikacija [8] koja omogućava jednostavan pregled i rad sa podacima (slika 2.5.).



Slika 2.5. – Grafički prikaz implementirane SQLite baze podataka

3. DEFINIRANJE KORISNIČKIH ZAHTJEVA I SPECIFIKACIJA ZA APLIKACIJU

3.1. Korisnički zahtjevi

Prije početka razvoja aplikacije, potrebno je utvrditi tko će je koristiti. Korisnici su podijeljeni u tri grupe. Prvu grupu korisnika čine studenti koji dolaze na praksu u tvrtku. Za te korisnike potrebno je osigurati slijedeće mogućnosti:

- Dodavanje i uređivanje dnevnika rada
- Predaja dnevnika rada
- Pregled svih izrađenih dnevnika
- Pregled i uređivanje osobnih informacija
- Pregled dodijeljenih mentora.

Drugu grupu korisnika čine mentori koji će biti dodijeljeni studentima koji dolaze na praksu. Za te korisnike potrebno je osigurati slijedeće mogućnosti:

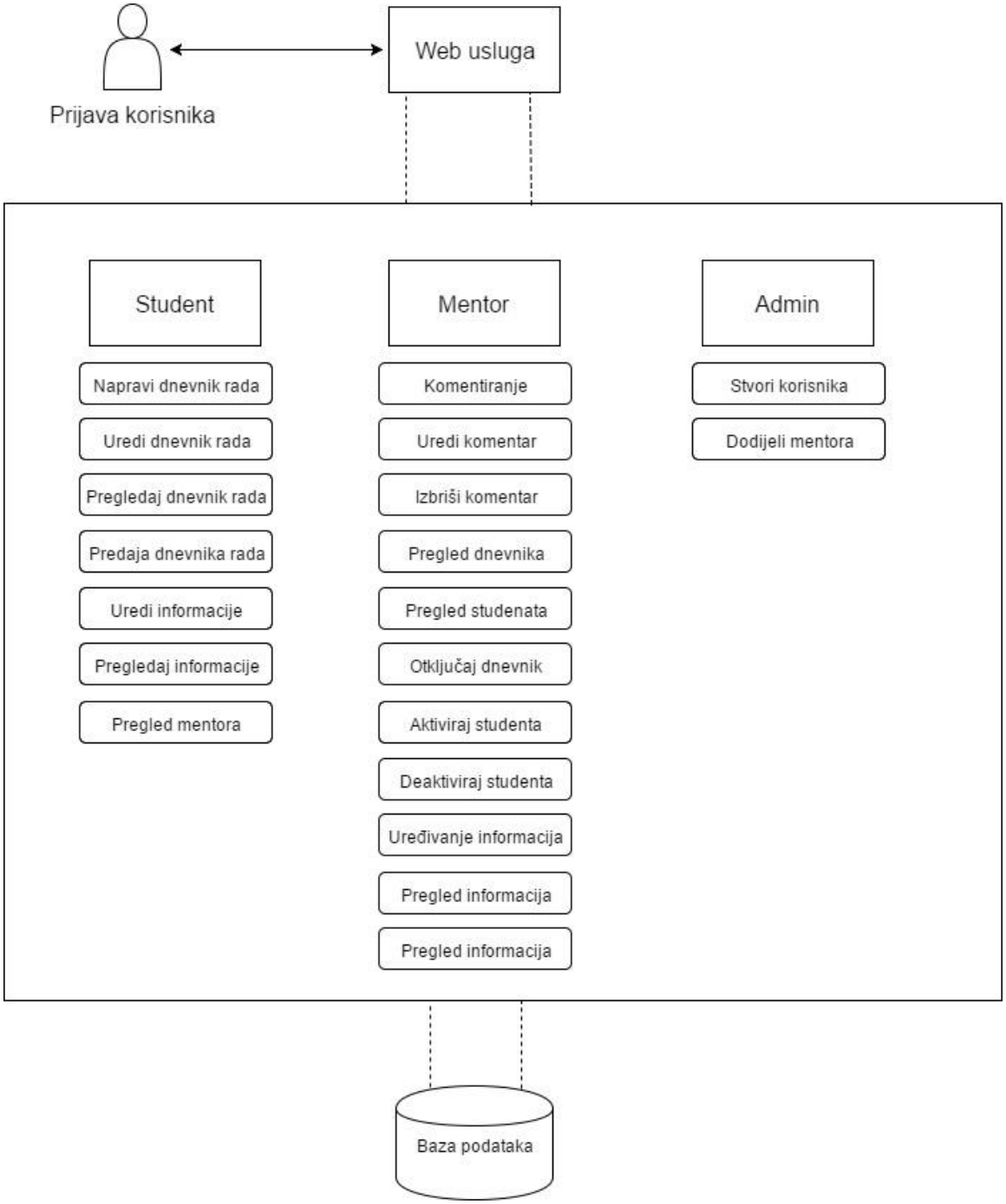
- Pregled i otključavanje dnevnika rada
- Stvaranje, uređivanje i brisanje komentara o studentu
- Aktiviranje i deaktiviranje studenata u bazi podataka
- Pregled i uređivanje osobnih informacija.

Treću grupu korisnika čine administratori. Za te korisnike potrebno je osigurati slijedeće mogućnosti:

- Stvaranje korisnika u bazi podataka
- Dodjeljivanje mentora studentu i obrnuto.

3.2. Izgled sustava

Nakon što su utvrđeni korisnički zahtjevi potrebno je utvrditi od čega će se aplikacija sastojati. Potrebno je razraditi izgled sustava, odrediti na koji način će on funkcionirati i kako će se implementirati zahtjevi korisnika. Na slici 3.1. se može vidjeti arhitektura aplikacije. Aplikacija se sastoji od dva osnovna dijela, a to su web usluga i baza podataka. Baza podataka služi za pohranjivanje podataka, a web usluga za upravljanje s tim podacima. Zahtjevi korisnika biti će dio web usluge koja služi za komunikaciju s korisnikom i s bazom podataka.

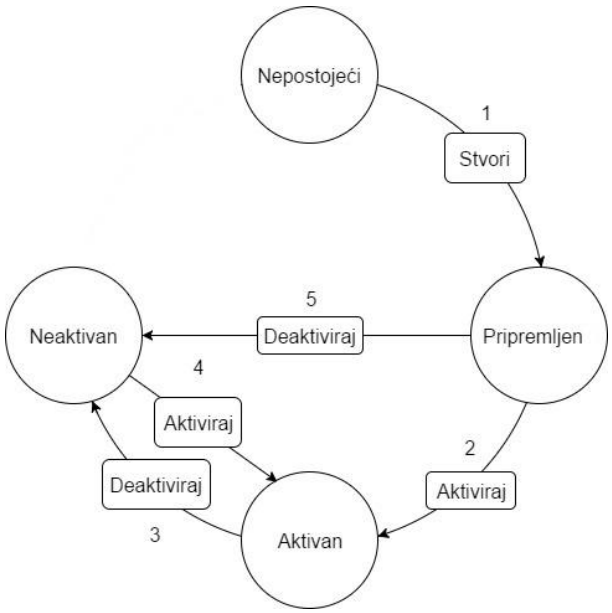


Slika 3.2. – Grafički prikaz izgleda sustava

3.3. Životni krug korisnika i dnevnika rada

Student nakon dolaska na praksu dobiva svoj korisnički račun koji je aktivan za vrijeme trajanja prakse. Nakon odlaska sa prakse korisnički račun je potrebno deaktivirati i zbog toga je oblikovan životni krug za korisničke račune sa statusom studenta i za dnevnike rada koje je taj

student napisao. Životni krug predstavlja sva moguća stanja koja će korisnik i dnevnik rada imati nakon što su dodani u bazu podataka. Korisnik sa statusom studenta imat će četiri moguća stanja prema kojima se određuju njegova prava (slika 3.2.). Između svakog stanja postoji jedna ili više mogućih operacija (tablica 3.1.).

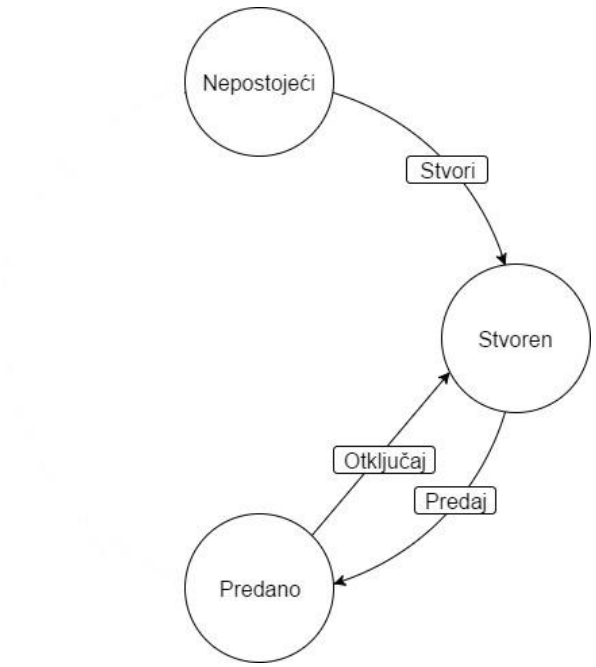


Slika 3.2. – Grafički prikaz životnog kruga korisnika sa statusom studenta

Tablica 3.1. – Opis svih stanja i operacija životnog kruga

Trenutno stanje	Operacija	Opis operacije
Nepostojeći	1 (Stvori)	Informacije o studentu su dodane u bazu podataka i status korisničkog računa je postavljen na "pripremljen"
Pripremljen	2 (Aktiviraj)	Student je aktiviran od strane mentora i njegov status se postavlja na "aktivan"
Pripremljen	5 (Deaktiviraj)	Student je deaktiviran i njegov status se postavlja na "neaktivan"
Aktivan	3 (Deaktiviraj)	Student je deaktiviran i njegov status se postavlja na "neaktivan"
Neaktivan	4 (Aktiviraj)	Student je aktiviran od strane mentora i njegov status se postavlja na "aktivan"

Dnevnik rada koji je student napisao ima tri stanja (slika 3.3.) između kojih postoji nekoliko operacija (tablica 3.2.) koje služe za upravljanje statusa dnevnika unutar same baze podataka.



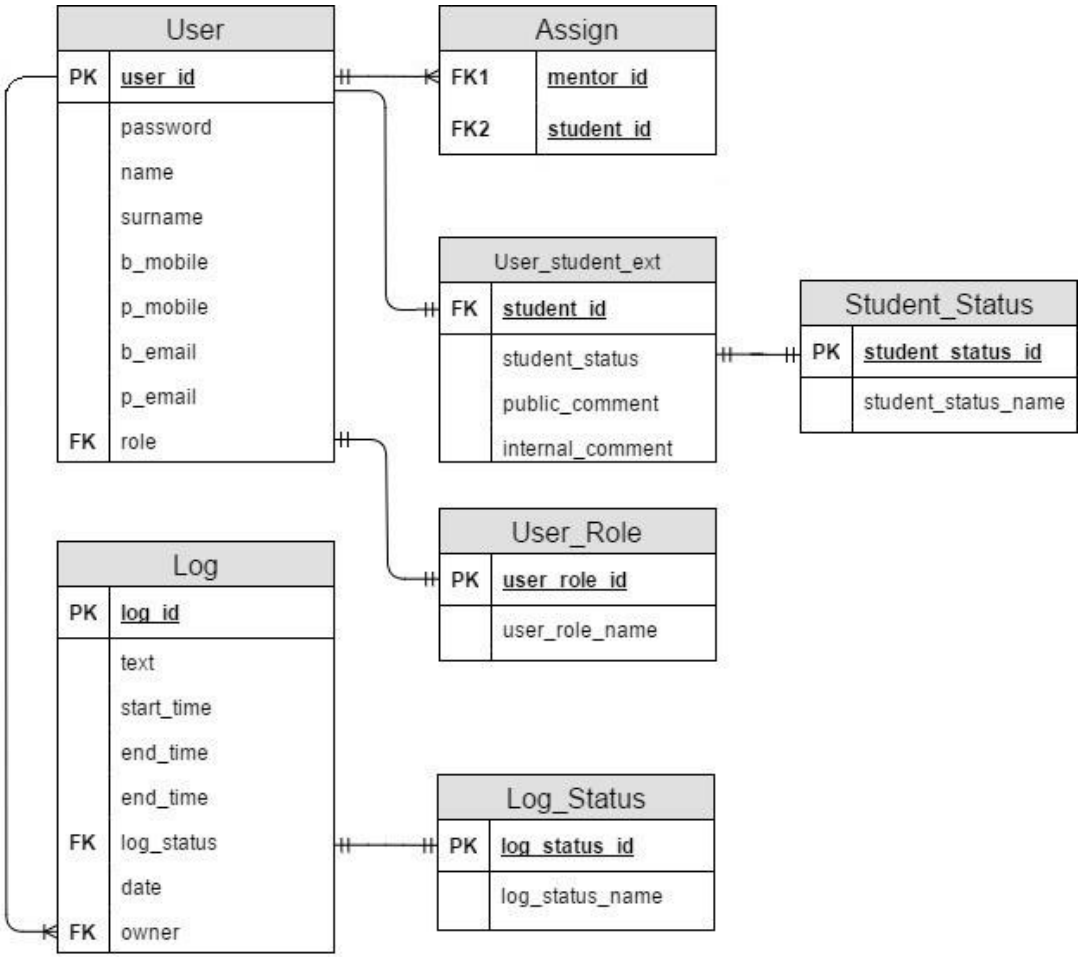
Slika 3.3. – Grafički prikaz životnog kruga dnevnika rada

Tablica 3.2. – Opis svih stanja i operacija životnog kruga

Trenutno stanje	Operacija	Opis operacije
Nepostojeći	1 (Stvori)	Student stvara novi dnevnik rada
Stvoren	2 (Predaj)	Nakon što je dnevnik predan, on se zaključava i ne može se uređivati dok ga mentor ne otključa.
Predano	3 (Otključaj)	Student je deaktiviran i njegov status se postavlja na "neaktivan"

3.4. Oblikovanje i izrada baze podataka

Nakon što su definirani svi korisnički zahtjevi potrebno je oblikovati relacijski model baze podataka. Relacijski model ja pristup izradi baze podataka kojim se unaprijed utvrđuju informacije koje baza sadrži i koje informacije će korisnik tražiti iz nje [9]. U relacijskom modelu potrebno je prikazati sve tablice koje će biti korištene i njihove veze sa ostalim tablicama unutar baze podataka. Također je potrebno definirati primarne i strane ključeve. Relacijski model je prikazan na slici 3.4.



Slika 3.4. – Relacijski model baze podataka

Na temelju relacijskog modela izrađuje se baza podataka. Prva tablica koju je potrebno napraviti je tablica *User* koja će sadržavati osobne podatke o korisnicima, njihovo jedinstveno korisničko ime *user_id* i njihovu ulogu *role*. Potrebno je postaviti postavit ograničenje na tablicu (*engl. Constraint*) koje se koristi kako bi se ograničilo dodavanje određenih podataka u tablicu. To nam osigurava točnost i pouzdanost podataka u tablici i ukoliko dođe do kršenja ograničenja, akcija je prekinuta [10]. Ograničenje će biti postavljeno na atribut *role* koji može biti samo mentor, student ili administrator (programski kôd 3.1.).

```
CREATE TABLE User (
user_id CHAR(5) PRIMARY KEY NOT NULL,
password VARCHAR(20) NOT NULL,
name VARCHAR(20) NOT NULL,
surname VARCHAR(20) NOT NULL,
b_mobile VARCHAR(20) NOT NULL UNIQUE,
p_mobile VARCHAR(20) NOT NULL UNIQUE,
b_email VARCHAR(20) NOT NULL UNIQUE,
p_email VARCHAR(20) NOT NULL UNIQUE,
role VARCHAR(10) NOT NULL,
CONSTRAINT fk_role FOREIGN KEY (role) REFERENCES User_Role(user_role_id) ON
DELETE CASCADE ON UPDATE NO ACTION)
```

Programski kôd 3.1. – SQL kôd za tablicu *User*

Tablica *User_Role* na koju se poziva tablica *User* sadrži imena i identifikacijske oznake korisničkih uloga (programski kôd 3.2.). Identifikacijske oznake biti će brojevi 0, 1 i 2 (slika 3.4.) i ukoliko se u tablicu *User* doda neka druga vrijednost akcija će biti prekinuta.

```
CREATE TABLE User_Role (
user_role_id INTEGER PRIMARY KEY NOT NULL,
user_role_name VARCHAR(10) NOT NULL)
```

Programski kôd 3.2. – SQL kôd za tablicu *User_Role*

user_role_id	user_role_name
0	admin
1	mentor
2	student

Slika 3.4. – Grafički prikaz tablice *User_Role*

Tablica *User_student_ext* će služiti za pohranu komentara mentora i sadržavati će trenutni status studenta (programski kôd 3.3.). Potrebno je postaviti ograničenje da se atribut *student_id* poziva na atribut *user_id* iz tablice *User*. Drugo ograničenje će biti postavljeno na atribut *student-_status* koji može biti aktivan, neaktivan i pripremljen.

```
CREATE TABLE User_student_ext (
student_id CHAR(5) NOT NULL,
student_status VARCHAR(15) NOT NULL,
public_comment VARCHAR(500),
internal_comment VARCHAR(500),
CONSTRAINT fk_student FOREIGN KEY (student_id) REFERENCES User(user_id) ON
DELETE CASCADE ON UPDATE NO ACTION
CONSTRAINT fk_status FOREIGN KEY (student_status) REFERENCES
Student_Status(student_status_id) ON DELETE CASCADE ON UPDATE NO ACTION)
```

Programski kôd 3.3. – SQL kôd za tablicu *User_Role*

Tablica *Student_Status* na koju se poziva tablica *User_student_ext* sadrži imena i identifikacijske oznake korisničkih uloga (programski kôd 3.4.). Identifikacijske oznake biti će brojevi 0, 1 i 2 (slika 3.5.) i ukoliko se u tablicu *User_student_ext* doda neka druga vrijednost akcija će biti prekinuta.

```
CREATE TABLE Student_Status (
student_status_id INTEGER PRIMARY KEY NOT NULL,
student_status_name VARCHAR(15) NOT NULL)
```

Programski kôd 3.4. – SQL kôd za tablicu *Log_Status*

student_status_id	student_status_name
0	Preparation
1	Enabled
2	Disabled

Slika 3.5. – Grafički prikaz tablice *Student_Status*

Za tablicu *User_student_ext* potrebno je napraviti okidač (*engl. Trigger*) koji će se aktivirati kada se u tablicu *User* doda novi korisnik koji ima ulogu studenta. Okidač je skup akcija koje se automatski pokrenu nakon što se pozove određena operacija promjene (*insert, update, delete*) nad zadanom tablicom [11]. Okidač se može vidjeti u programskom kôdu 3.5.

```
CREATE TRIGGER student_status
AFTER INSERT
ON User
WHEN (new.role = '2')
BEGIN
INSERT INTO User_student_ext (student_id, student_status) VALUES
(new.user_id, '0');
END
```

Programski kôd 3.5. – SQL kôd za implementirani okidač

4. RAZVOJ APLIKACIJE ZA PRAĆENJE STUDENTSKIH PRAKSI

4.1. Izrada sučelja

Na početku je potrebno izraditi sučelje (*engl. Interface*) u kojem će biti definirane sve metode koje će web usluga sadržavati. Sučelje je nacrt za klase, nije nužno potrebno i služi za organizaciju strukture objektno-orijentiranog programiranja. Sučelje definira metode za klase tako što specificira njihova imena, povratni tip i argumenti pojedine metode [12]. Programski kôd 4.1. prikazuje metodu *createTextLog* iz sučelja *SIMInterface* koja će služiti za dodavanje novih korisnika u bazu podataka. Definirani su svi argumenti koje će metoda primati i tip podatka za pojedini argument. Oznaka *@WebMethod* je anotacija koja označava da je ova metoda dio web usluge. Anotacije pružaju dodatne informacije o programu koji nije dio samog programa i nemaju direktan efekt na operaciju koju anotiraju [13].

```
@WebMethod
public Response createTextLog(
            @WebParam (name="userID") String userID,
            @WebParam (name="text") String text,
            @WebParam (name="startTime") Date startTime,
            @WebParam (name="endTime") Date endTime);
```

Programski kôd 4.1. – Java kôd za metodu *createUser*

4.2. Definiranje konstanti

Prije početka implementacije sučelja potrebno je definirati konstante koje će se koristiti u kodu. Konstante su vrijednosti koje će se često koristiti u pisanju koda i potrebno je napraviti klasu koja će ih sadržavati. Potrebno ih je definirati radi lakšeg pisanja samoga koda i njegove preglednosti. Nakon što ih definiramo njihove vrijednosti možemo mijenjati unutar klase i automatski će se promijeniti u cijelom kodu što olakšava ispravljanje eventualnih grešaka ili ažuriranje koda. Konstante će biti definirane u klasi *SIMConstants* i imati će atribut *static final* što znači da je njihova vrijednost konačna i da se ne može mijenjati. Ključna riječ *static* najčešće se upotrebljava radi upravljanja memorijom i koristi se za referenciranje zajedničkih obilježja više različitih objekata [14]. Programski kôd 4.2. prikazuje klasu *SIMConstants*, konstante koje će biti

korištene pri implementacije (*LOG_STATUS, USER_ROLE, STUDENT_STATUS*) i njihove vrijednosti.

```
public class SIMConstants {

	public static final int LOG_STATUS_LOCKED = 0;
	public static final int LOG_STATUS_UNLOCKED = 1;
	public static final int USER_ROLE_ADMIN = 0;
	public static final int USER_ROLE_MENTOR = 1;
	public static final int USER_ROLE_STUDENT = 2;
	public static final int STUDENT_STATUS_DISABLED = 2;
	public static final int STUDENT_STATUS_ACTIVE = 1;
	public static final int STUDENT_STATUS_PREPEARATION = 0;
}
```

Programski kôd 4.2. – Java kôd za klasu *SIMConstants*

4.3. Definiranje poruka za iznimke

Ukoliko se prilikom unosa podataka upiše pogrešna vrijednost program će baciti iznimku za tu vrijednost i odgovarajuću poruku. Klasa *Response* definira poruke koje će biti poslane korisniku ukoliko dođe do pogreške pri upisu podatka. Programski kôd 4.3. prikazuje definirane poruke.

```
public static final Response OK = new Response(1, "SUCCESS");
public static final Response ERROR_INVALID_NAME =
new Response(2, "''{0}'' is not a valid name. Only letters and '-' sign are
allowed with length of 1-20 characters");
public static final Response ERROR_INVALID_EMAIL =
new Response(3, "''{0}'' is not a vaild email address!");
public static final Response ERROR_INVALID_PHONE_NUMBER =
new Response(4, "''{0}''  is not a valid phone number! Only numbers are
allowed with length of 5-25 digits");
public static final Response ERROR_INVALID_ID =
new Response(5, "Syntax error: ''{0}'' is invalid ID");
public static final Response ERROR_EMPTY_PARAMETERS =
new Response(6, "Please specify all parameters!");
public static final Response ERROR_INACTIVE_STUDENT_STATUS =
new Response(7, "Student is not active");
```

Programski kôd 4.3. – Java kôd za poruke iz klase *Response*

Greške će biti poslane korisniku u obliku XML poruke, a svaka poruka se sastoji od dva dijela. Prvi dio je *integer* koji će označavati broj poruke odnosno *error code* i svaka poruka ima različitu oznaku. Drugi dio poruke jest sama poruka, odnosno tekst koji će biti poslan korisniku. Oznaka

"{0}" je *placeholder* i na njegovo mjesto dolazi argument zbog kojega je došlo do pogreške. Programski kôd 4.4. prikazuje XML poruku koja će biti poslana korisniku ukoliko je upisan pogrešan korisnički *userID*.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Body>
      <ns1:disableStudentResponse
      soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:ns1="http://studentInternshipManager">
      <disableStudentReturn href="#id0"/>
      </ns1:disableStudentResponse>
         <multiRef id="id0" soapenc:root="0"
      soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      xsi:type="ns2:Response"
      xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:ns2="http://studentInternshipManagerExceptions">
      <args soapenc:arrayType="xsd:anyType[1]" xsi:type="soapenc:Array">
        <args xsi:type="soapenc:string">
            NepostojeciKorisnik
        </args>
      </args>
      <code xsi:type="xsd:int">
            5
      </code>
      <message xsi:type="xsd:string">
            Syntax error: 'NepostojeciKorisnik' is invalid ID
      </message>
      </multiRef>
   </soapenv:Body>
</soapenv:Envelope>
```

Programski kôd 4.4. – XML poruka za pogrešan *userID*

4.4. Implementacija sučelja

Nakon što su sve konstante i poruke definirane potrebno je implementirati osmišljeno sučelje. Implementacija sučelja biti će podijeljena na dva dijela odnosno dvije klase. Prva klasa *SIMImpl* će služiti za provjeru primljenih parametara i da li oni zadovoljavaju sve zadane uvijete. Druga klasa *DatabaseCommunication* će služiti za komunikacijom sa bazom podataka odnosno zapisivanje i uzimanje podataka iz nje. Na ovaj način se odvaja logika programa od rada sa bazom podataka što povećava preglednost koda i omogućava se lagano ažuriranje koda ukoliko to bude potrebno u budućnosti. Programski kôd 4.5. prikazuje metodu *createTextLog* iz klase *SIMImpl*. Metoda će prvo provjeriti da li poslani *userID* ima status studenta, ukoliko je uvjet zadovoljen provjerava se da li je

student aktivan i zatim se poziva metoda *createTextLog* iz klase *DatabaseCommuncation*. U slučaju da jedan od uvjeta nije zadovoljen korisniku se šalje poruka da je *userID* pogrešan, ili da student nije aktivan.

```
@Override
    public Response createTextLog(
            @WebParam (name="userID") String userID,
            @WebParam (name="text") String text,
            @WebParam (name="startTime") Date startTime,
            @WebParam (name="endTime") Date endTime)
      {
      try
          {
            if(dbcom.getStudentStatus(userID) == USER_ROLE_STUDENT)
                  {

                  if(dbcom.getUserRole(userID) == STUDENT_STATUS_ACTIVE)
                  {
                        dbcom.createTextLog(userID, text, startTime,
                        endTime);
                        return Response.OK;
                  }

                  else
                  {
                        return
                        Response.ERROR_INACTIVE_STUDENT_STATUS.fill(dbcom.g
                        etUserRole(userID));
                  }
            }

            else
                  {
                        throw new InvalidUserIDException(userID);
                  }
            }

      catch(InvalidUserIDException e)
            {
                  return Response.ERROR_INVALID_ID.fill(e.getMessage());
            }
      }
```

Programski kôd 4.5. – Metoda *createTextLog* iz klase *SIMImpl*

Programski kôd 4.6. prikazuje metodu *createTextLog* iz klase *DatabaseCommunication*. Metoda služi za zapisivanje primljenih parametara u bazu podataka. Prvo je potrebno napraviti povezivanje (*engl. Connection*) sa bazom podataka kao što je prikazanu u programskom kôdu 4.7. Nakon što je uspostavljena veza sa bazom podataka potrebno dodati parametre u bazu podataka *SQL* naredbom *INSERT INTO*. U svrhu skraćivanja koda napravljena je *for* petlja koja će upisati sve parametre na njihova zadana mjesta.

```
public void createTextLog (String userID, String text, Date startTime, Date
endTime)
  {

    String[] args = {userID, text, startTime.toString(),
endTime.toString()};

    try
      {
        conn = openConnection();
        stmt = conn.prepareStatement("INSERT INTO Log (Owner, text,
        start_time, end_time, log_status)" + "VALUES (?, ?, ?, ?, 1)");

        for(int i = 0; i < args.length; i++)
          {
            stmt.setString((i+1), args[i]);
          }
        stmt.execute();
      }

        catch (SQLException e){
              e.printStackTrace();
        }
        finally{
              if (conn != null){
                    try{
                          conn.close();
                    }
                    catch (SQLException e){
                          e.printStackTrace();
                    }
              }
        }
  }
```

Programski kôd 4.6. – Metoda *createTextLog* iz klase *DatabaseCommunication*

```
Connection openConnection(){
        try{
              Class.forName(dbClass);
          Connection conn = DriverManager.getConnection(dbURL, username,
password);
          conn.createStatement().execute("PRAGMA foreign_keys = ON");
          return conn;
        }
        catch (ClassNotFoundException e){
              e.printStackTrace();;
        }
        catch (SQLException e){
              e.printStackTrace();;
        }
        return conn;
  }
```

Programski kôd 4.7. – Metoda *openConnection*

5. ZAKLJUČAK

Aplikacija za praćenje studentskih praksi u tehnologiji web usluga korisnicima pruža jednostavno upravljanje studentskom praksom i olakšava cijeli proces od dolaska studenta u tvrtku do njegovog odlaska. Ovom aplikacijom mentori mogu pratiti i komentirati napredak studenta što olakšava njihov posao jer se ponekad ne mogu posvetiti studentu na željeni način. Student koji je na praksi ima mogućnost dodavanja dnevnika rada, njihov pregled i uređivanje istih, dok mentori imaju mogućnost pregleda studentovih dnevnika rada i njihovo komentiranje putem web usluge.

Izrada praktičnog dijela završnog rada zahtijevala je istraživanje, proučavanje i detaljno planiranje aplikacije. U fazi planiranja utvrđeni su zahtjevi korisnika i izgled sustava koji zadovoljava iste. Osmišljeni su životni krugovi za korisnike aplikacije i njihove dnevnike rada. Osmišljena je implementirana baza podataka koja je trebala zadovoljiti sve korisničke zahtjeve. Izrađena je web usluga koja će se koristiti u aplikaciji te povezivanje usluge sa bazom podataka uz korištenje Java i SQL programskih jezika. Korištena je SQLite baza podataka koja ne zahtjeva poslužitelja nego se veže uz aplikaciju kao zasebna datoteka. Za testiranje web usluge korišten je program SoapUI koji omogućuje isprobavanje metoda dostupnih na web usluzi.

LITERATURA

[1] Java documentation, *About the Java technology*, http://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html, svibanj 2017.

[2] Java A Beginer's Guide Fifth Edition, Herbert Schildt, kolovoz 2017.

[3] Technopedia, *API*, https://www.techopedia.com/definition/25133/application-programming-interface-api-java, kolovoz 2017.

[4] W3, XML essentials, *Browser Statistics*, https://www.w3.org/standards/xml/core, svibanj 2017.

[5] Wikipedia, *SOAP*, https://hr.wikipedia.org/wiki/SOAP, svibanj 2017.

[6] Wikipedia, WSDL, https://en.wikipedia.org/wiki/Web_Services_Description_Language, rujan 2017.

[7] Wikipedia, *SQLite*, https://hr.wikipedia.org/wiki/SQLite, svibanj 2017.

[8] SQLite, *SQLite Manager* https://sqliteonline.com, svibanj 2017.

[9] Wikipedia, *Relation Model*, https://en.wikipedia.org/wiki/Relational_model#SQL_and_the_relational_model, lipanj 2017.

[10] W3 schools, *Constraints*, https://www.w3schools.com/sql/sql_constraints.asp, lipanj 2017.

[11] w3resource, *Triggers*, http://www.w3resource.com/mysql/mysql-triggers.php, lipanj 2017.

[12] LearnJava, *Interfaces*, http://www.learnjavaonline.org/en/Interfaces, lipanj 2017

[13] Java documentation, *Annotations*, https://docs.oracle.com/javase/tutorial/java/annotations/, kolovoz 2017.

[14] javaTpoint, *Static keyword* https://www.javatpoint.com/static-keyword-in-java, kolovoz 2017.

SAŽETAK

U završnom radu razvijena je aplikacija za praćenje studentskih praksi u tehnologiji web usluga, koja korisnicima omogućava upravljanje studentskim praksama i olakšava proces odrađivanja studentskih praksi. Aplikacija je namijenjena studentima koji će pomoću nje dodavati i uređivati svoje dnevnike rada, dok će mentori komentirati i pregledavati iste. U teorijskom dijelu rada opisane su korištene tehnologije, te je opisan proces razvoja web usluge i baze podataka. Praktični dio rada se sastoji od utvrđivanja korisničkih zahtjeva, a potom izrade baze podataka i web usluge koja će zadovoljavati sve zahtjeve. Za razvoj web usluge i baze podataka korišteni su programski jezici Java i SQL.

Ključne riječi: baze podataka, studentske prakse, web usluge

ABSTRACT

In this bachelor thesis application for managing student internship was developed, which allows users to manage student internships and eases the process of completing them. Application is intended for students who will use it to add and edit their daily logs, while mentors view and comment them. The theoretical part describes technologies used and the process of developing the web service and the database. Practical part of the work consists of defining user requirements and then developing the database and the web service which will satisfy all requirements. Java and SQL are programming languages used for developing the web service and the database.

Keywords: database, student internship, web service

ŽIVOTOPIS

Matej Milić rođen je 12. siječnja 1994. godine u Osijeku. Od 2000. do 2008. pohađa Osnovnu školu Ljudevita Gaja u Osijeku. Godine 2008. upisuje Elektrotehničku i prometnu školu Osijek koju završava 2012. obranom završnog rada i polaganjem ispita državne mature. Godine 2014. upisuje Elektrotehnički fakultet Osijek na Sveučilištu Josipa Juraja Strossmayera u Osijeku na stručni studij informatike.

Matej Milić