Software specification requirements

Students service CRUD Spring MVC web application

Nenad Paunov

1. Introduction

This document gives an overview of everything for this application. There will be present product requirements and functionality. Also, there will be present technologies which was used and there will be basic explanations of them.

1.1 Purpose of the documentation

This document helps our clients and our team to use this product. It will help any developer to aid in the software delivery lifecycle of the processes. This document supply detail the overview of our software.

It describes project goals and targets.

1.2 Overview of the documentation

There are several sections of this document. Section 2 give general description of the project. Section 3 gives viewpoint of the user interface Section 4 is about database and describe data requirements. Section 5 describes technologies and functions.

1.3 Setup application

For using this application, user need installed Eclipse, SQL Developer and Git. This

application can be found here: https://bitbucket.org/PaunovNenad/nenad-paunov-singidunum/src/master/

Also, user must have configured Apache Tomcat 9.0 with Eclipse. Eclipse must support Maven.

User must create new schema in database and link it in server context file. When first starting application user needs to uncomment <!-- <prop key="hibernate.hbm2ddl.auto">drop-and-create</prop> --> on line 49 in applicationContext in package nenad.paunov.singidunum.config and to put comment on line <prop key="hibernate.hbm2ddl.auto">update</prop> . Application will then drop and create tables with relations. After that undo those comments.

Credentials for user can be found in this document.

2. General Description

2.1 Product functions

Administrator have full access to project. He can view all students, professors, cities, titles,

exams, subject and countries with custom amount per page. Administrator can add, edit or remove everything. Administrator can set subjects to students and professors, set exam to students, set titles to professors, country to cities, city to professors and students and registrate exam for student.

2.2 User characteristics

There is only one type of user and that is administrator. His parameters for login are: username-test, password: test. All his functions are named in the previous section.

2.3 Use cases

- View list of professors / students / subjects / exams/ titles/ cities/ countries with pagination
- View details for choosen professor / student / subject / exam/ title/ city/ country
- Add / Edit professor / student / subject / exam/ title/ city/ country
- Delete professor / student / subject / exam/ title/ city/ country with a warning
- Assign subjects to student
- Assign professors to subject
- Register students for exam

2.4 Rules

Exam can be registered only one week before date. Student can only register exam that are for subjects in his year of study or

years before.

When creating student or professor email address must be unique in data base. Student

index number must be also unique and it need to have 10 characters.

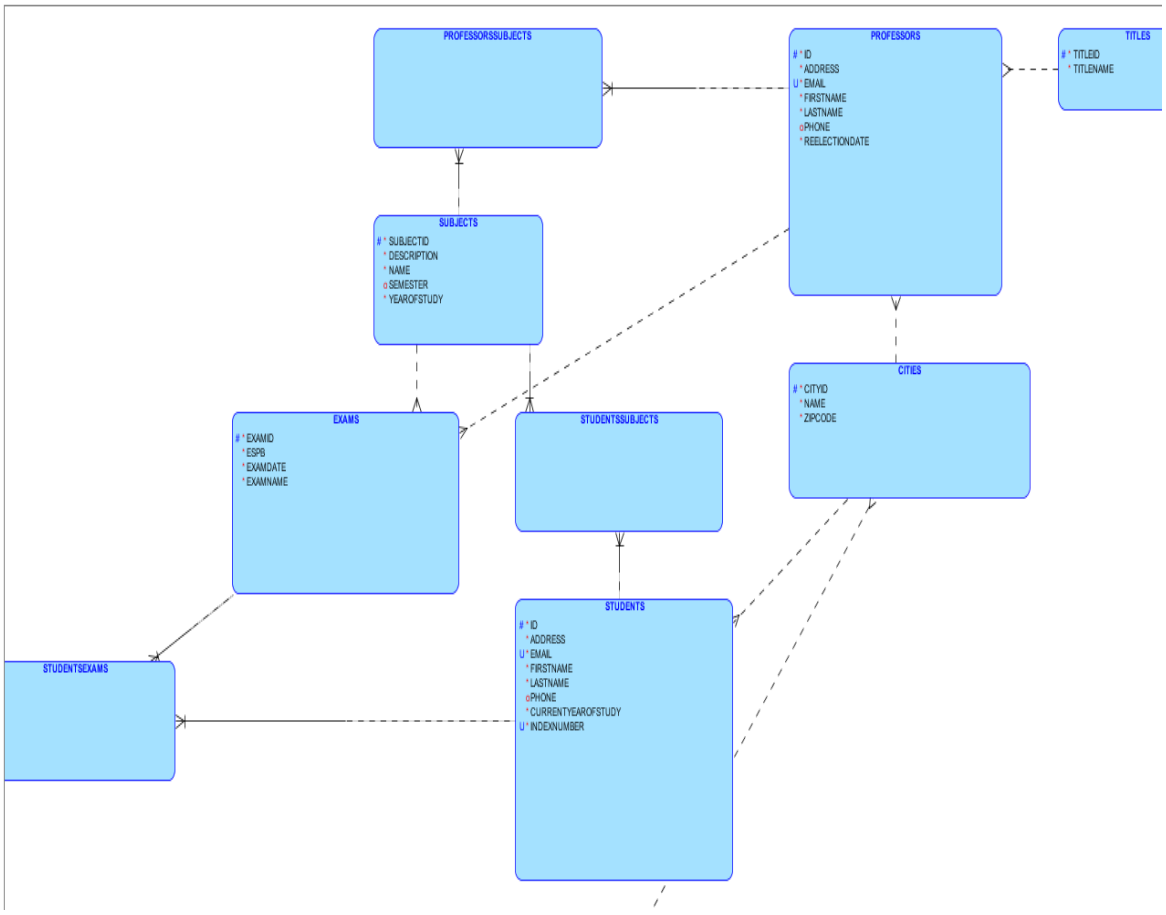
3. Used technologies

3.1 Servlet technology

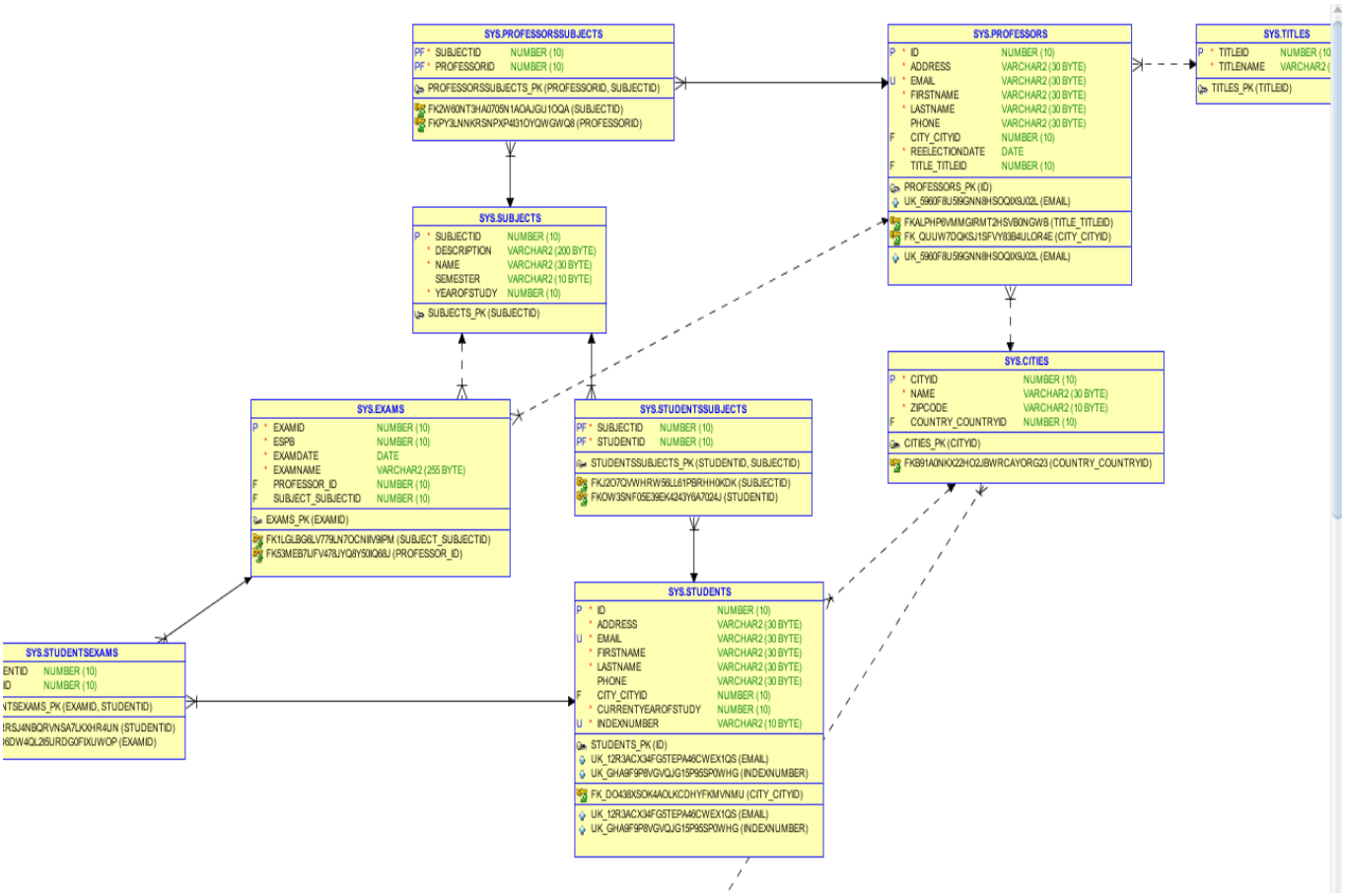
The application should be usable with any Java EE web application container that is compatible with the Servlet 4.0 and JSP 2.3 specifications. During the development of this application is used Apache Tomcat 9.0. The view technologies that are used for rendering the application are Java Server Pages (JSP) along with the Java Standard Tag Library (JSTL).

3.2 Database technology

The application uses a relational database for data storage. Database used for developing is Oracle, but can be easily switched to MySQL or any other relational db. The following picture shows ER diagram for this project Database.


PROFESSORSSUBJECTS
PROFESSORS
* ID
* ADDRESS
U * EMAIL
* FIRSTNAME
* LASTNAME
o PHONE
* REELECTIONDATE
TITLES
* TITLEID
* TITLENAME
SUBJECTS
* SUBJECTID
* DESCRIPTION
* NAME
o SEMESTER
* YEAROFSTUDY
CITIES
* CITYID
* NAME
* ZIPCODE
EXAMS
* EXAMID
* ESPB
* EXAMDATE
* EXAMNAME
STUDENTSSUBJECTS
STUDENTS
* ID
* ADDRESS
U * EMAIL
* FIRSTNAME
* LASTNAME
o PHONE
* CURRENTYEAROFSTUDY
U * INDEXNUMBER
STUDENTSEXAMS

And Relational model:



4. Functional requirements

4.1 Application configuration

Build tool used in development of the application is Maven, so all dependencies are contained inside pom.xml file. In that way its insured that a migration of application Hosting requires minimal modifications.

All classes based on concerns of its content are organized in next packages:

4.2 Entities

For every table in DB there is a corresponding POJO class or better said Entity. In this project we are using Hibernate as Object/Relational Mapping (ORM) framework. In addition to its own “native” API, Hibernate is also an implementation of the Java Persistence API (JPA) specification.

Each entity is validated with the help of javax.validation.constraints . Entity classes are listed below:

- nenad.paunov.singidunum.entities.City
- nenad.paunov.singidunum.entities.Exam
- nenad.paunov.singidunum.entities.Country
- nenad.paunov.singidunum.entities.Person
- nenad.paunov.singidunum.entities.Title
- nenad.paunov.singidunum.entities.Student extends Person class
- nenad.paunov.singidunum.entities.Professor extends Person class
- nenad.paunov.singidunum.entities.Subject

Code snippet for one Java class:

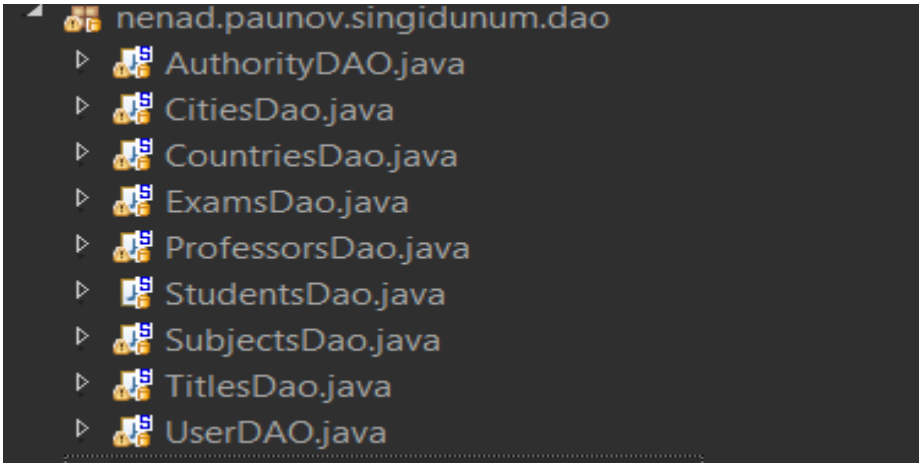
```java
package nenad.paunov.singidunum.entities;

import java.sql.Date;
@Entity
@Table(name="Exams")
public class Exam {
    @Id
    @GeneratedValue
    @Column(name = "ExamId")
    private int examId;
    @NotNull
    @Column(name="ExamName")
    private String examName;
    @NotNull
    @Range(min = 1, max = 12, message = "Please select only numbers from 1 to 12")
    @Column(name="ESPB")
    private int espb;
    @NotNull
    @Column(name = "examDate")
    private Date examDate;
    @ManyToOne
    private Subject subject;
    @ManyToOne
    private Professor professor;
    @ManyToMany(fetch = FetchType.LAZY,
            cascade =
            {
```

4.3 DAO and Services

Every entity has corresponding DAO and Service classes. DAO have methods for all operations (CRUD and others) which we need for a specific entity.

DAO classes are annotated with @Component and @Transactional so it means that every method is transaction, Service classes are annotated with @Service. In DAO classes Hibernate SessionFactory instance is injected and we use sessions for each query and operation there.

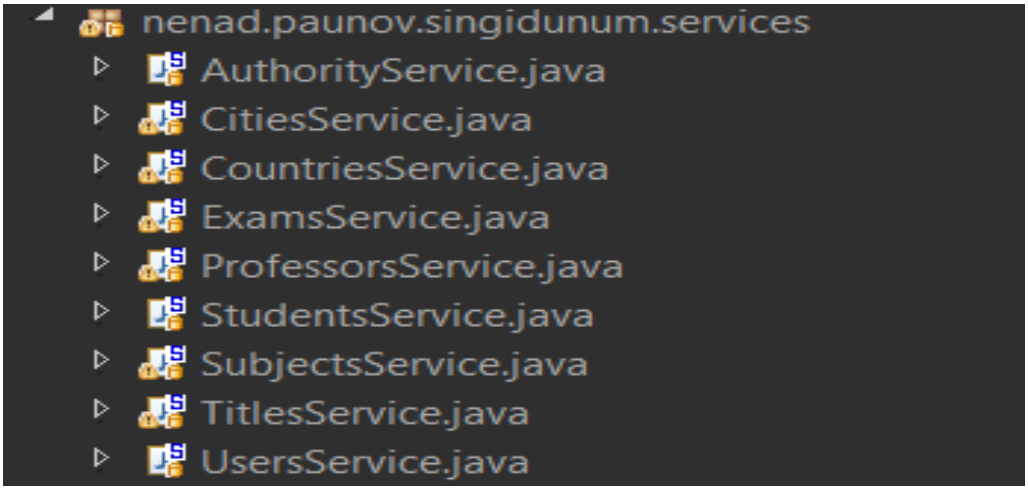
Sessions are automatically closed after the query is executed. We are using services to forward logic from DAO to Controllers.



Services allow that communication between user (controller) is not

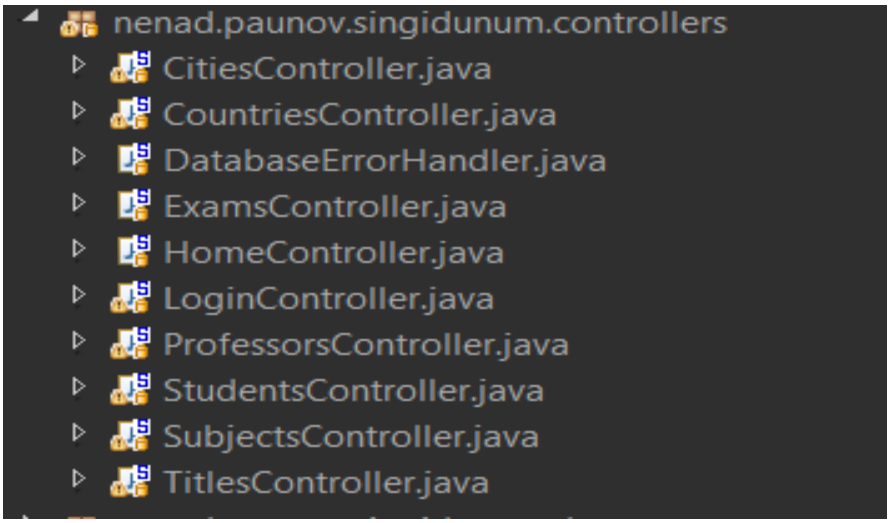
being executed directly and that every query has been isolated in a single transaction.

List of service classes can be found bellow:



4.4 Controllers

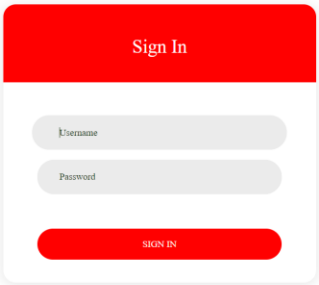
Controller package contains all controller classes. Each controller represents a collection of API endpoints for data manipulation or collecting data. Each entity has its own controller that contains several endpoints. Based on the URI, HTTP method or both different methods in the controller is being triggered. Each of them has different tasks and returns different information to the client. Every controller class is annotated with @Controller. Every method is annotated with @RequestMapping containing the path to this specific method. Service methods reference of each entity is injected in corresponding controller and it is used to get a DAO layer and to access DB.




```java
package nenad.paunov.singidunum.controllers;

import java.util.ArrayList;

@Controller
public class StudentsController {
    @Autowired
    private StudentsService studentsServices;
    @Autowired
    private CitiesService citiesService;
    @Autowired
    private SubjectsService subjectsServices;

    @RequestMapping("/students")
    public String showStudents(Model model) {
        List<Student> students = studentsServices.getAllStudents();
        model.addAttribute("students", students);
        return "students";
    }

    @RequestMapping("/student_details/{id}")
    public String showStudentDetails(@PathVariable int id, Model model) {
        Student student = studentsServices.getStudent(id);
        Set<Subject> subjects = student.getSubjects();
        model.addAttribute("subjects", subjects);
        Set<Exam> exams = student.getExams();
        model.addAttribute("exams", exams);
```

4.5 Spring Security

This project implements Spring Security to prevent unauthorized access. Only logged in users can use this application. Default user name is test and password test



All adjustments for Spring Security are in security-config.xml file which can be found in:

nenad.paunov.singidunum.config

security-context.xml

```
                    password="{noop}test" />
            </security:user-service>
        </security:authentication-provider>
        <security:authentication-provider>
            <security:jdbc-user-service
                data-source-ref="dataSource" />
        </security:authentication-provider>
    </security:authentication-manager>

    <security:http use-expressions="true">
        <security:intercept-url pattern="/static/**"
            access="permitAll" />
        <security:intercept-url pattern="/login"
            access="permitAll" />
        <security:intercept-url pattern="/error"
            access="permitAll" />
        <security:intercept-url pattern="/loggedout"
            access="permitAll" />
        <security:intercept-url pattern="/*"
            access="isAuthenticated()" />
        <security:form-login login-page="/login" />
        <security:logout logout-success-url="/loggedout" />
        <security:csrf disabled="true" />
    </security:http>
</beans>
```

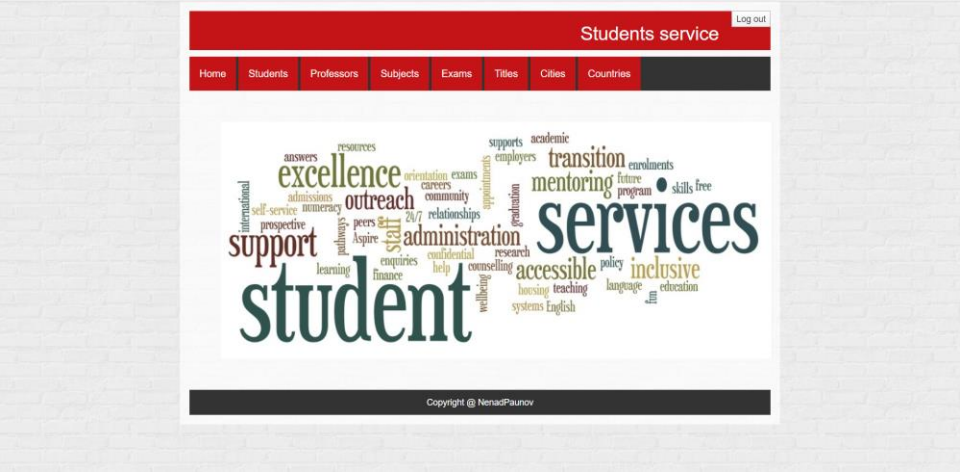
4.6 Front end technologies

In development of this applications user interface are used next technologies:

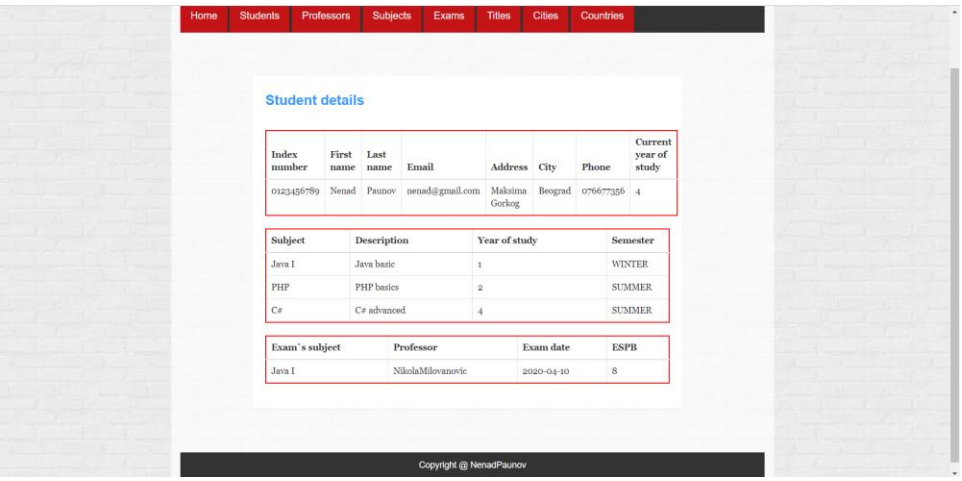
- JSP
- CSS
- HTML
- JSP - Standard Tag Library
- JavaScript
- jQuery

4.7 Some snippets of Web pages

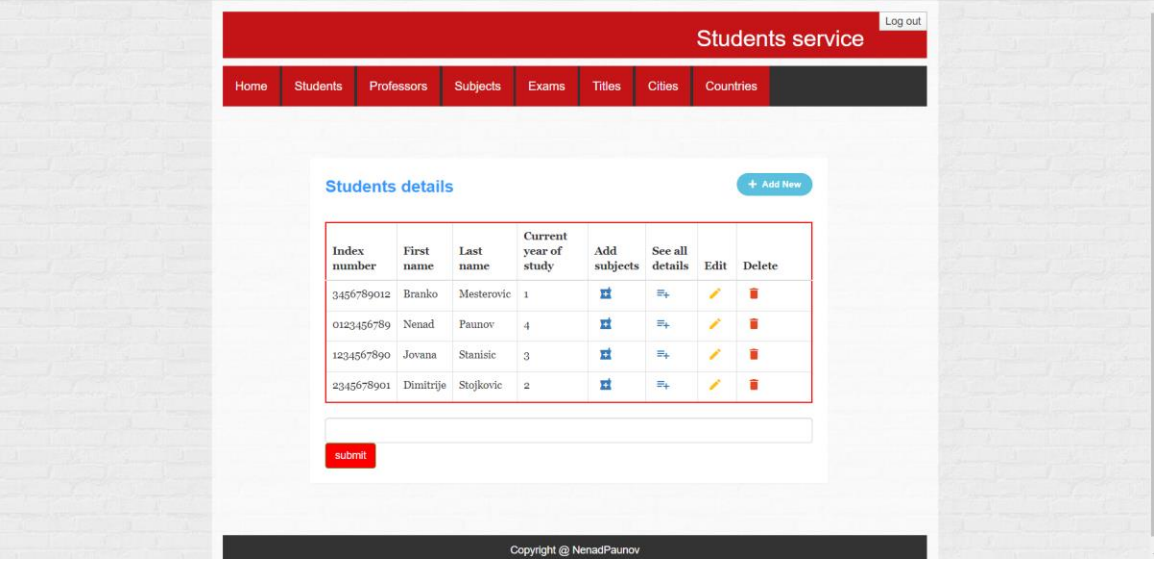
Home page:



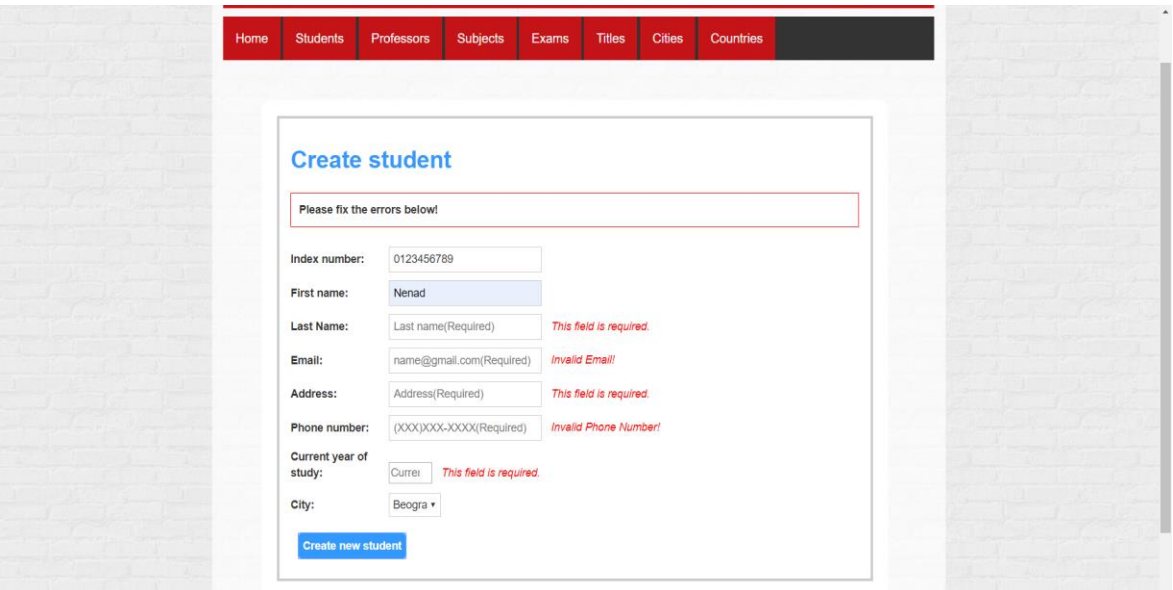
Student details:



All students:



Creating student:



Deleting student: