Technical University of Cluj-Napoca

Faculty of Automation and Computer Science

1st Semester 2019-2020

School Catalog
Web Application

Student: Elekes Lukacs-Roland

Group: 30434

Supervising teacher: Todoran Eneia

Contents

1. Abstract

The main objective of this group project is to get familiarized with the Spring Framework and to understand the steps for developing a Web Application.

Our web application is a system that represents an online school catalog. There are different types of users (admin, teacher and student) and every user gets his corresponding content after login, the admin can maintain the database system, he can add, edit, delete other accounts; the teacher can enter and edit the marks for the students that are participating at his course; and the students can log in and see their marks for every subject they have.

For developing, we used testing following the Test Driven Development and User Stories Principles, for database we used the MongoDB and for accessing it the Spring Data Framework.

For creating the web interface the MVC Web Spring framework was used.

2. Main references

- Engineering Software as a Service: An Agile Approach Using Cloud Computing by Fox, Armando, Patterson, David
- https://spring.io/docs
- https://spring.io/projects/spring-data-mongodb
- https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html

3. Project Description

3.1. Initial Mini Project

The main objective of the Mini Project was to get familiarized working in Java using the Spring Framework.

Firstly, we were using the Web MVC framework of spring to learn how to develop a small web application, how to handle URLs, how to create connection between html and Java.

Secondly, we were focusing on getting familiarized with the Spring Data part of the Framework for creating connections with databases, having access to data, learning to be able to perform CRUD operations on data.

Our application represented the model of a Web Shop, thus the we had as a model the Product and we were able to perform CRUD operations on the products. We used MongoDB to store the data. The views consisted of web pages, CSS and HTML and there was a controller that mapped the requests to the corresponding actions and methods.

Source code:

Product class:

```
@Document(collection = "products")
public class Product {
    @Id
    private String id;
    private String prodName;
    private String prodDesc;
    private Double prodPrice;

    public Product() {
    }

    public Product(String prodName, String prodDesc, Double prodPrice) {
        this.prodName = prodName;
        this.prodDesc = prodDesc;
        this.prodPrice = prodPrice;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getProdName() {
        return prodName;
    }

    public void setProdName(String prodName) {
        this.prodName = prodName;
    }

    public String getProdDesc() {
        return prodDesc;
    }

    public void setProdDesc(String prodDesc) {
        this.prodDesc = prodDesc;
    }

    public Double getProdPrice() {
        return prodPrice;
    }

    public void setProdPrice(Double prodPrice) {
        this.prodPrice = prodPrice;
    }
}
```

ProductController Class:

```
@RestController
public class ProductController {

    @Autowired
    ProductRepository productRepository;

    @RequestMapping(value = "/products", method = RequestMethod.GET)
    public ModelAndView viewProducts() {
        ModelAndView modelAndView = new ModelAndView();
        modelAndView.addObject("products", productRepository.findAll());
        modelAndView.setViewName("products");
        return modelAndView;
    }

    @RequestMapping(value = "/products/{id}", method = RequestMethod.GET)
    public ModelAndView viewProduct(@PathVariable("id") String prodId) {
        ModelAndView modelAndView = new ModelAndView();
        modelAndView.addObject("product", productRepository.findById(prodId).get());
        modelAndView.setViewName("product");
        return modelAndView;
    }

    @RequestMapping(value = "/create", method = RequestMethod.GET)
    public ModelAndView create() {
        ModelAndView modelAndView = new ModelAndView();
        modelAndView.setViewName("redirect:/products/create");
        return modelAndView;
    }

    @RequestMapping(value = "/products/create", method = RequestMethod.GET)
    public ModelAndView createProduct() {
        ModelAndView modelAndView = new ModelAndView();
        modelAndView.setViewName("create");
        return modelAndView;
    }

    @RequestMapping(value = "/save", method = RequestMethod.POST)
    public ModelAndView saveProduct(@RequestParam String prodName, @RequestParam
String prodDesc, @RequestParam Double prodPrice) {
        Product p = new Product();
        p.setProdName(prodName);
        p.setProdDesc(prodDesc);
        p.setProdPrice(prodPrice);
        productRepository.save(p);
        return new ModelAndView("redirect:/products");
    }

    @RequestMapping("/edit")
    public ModelAndView edit(@RequestParam String prodId) {
        return new ModelAndView("redirect:/products/edit/" + prodId);
    }

    @RequestMapping(value = "/products/edit/{id}")
    public ModelAndView editProduct(@PathVariable("id") String prodId) {
        ModelAndView modelAndView = new ModelAndView();
        modelAndView.addObject("product", productRepository.findById(prodId).get());
        modelAndView.setViewName("edit");
        return modelAndView;
    }

    @RequestMapping(value = "/update", method = RequestMethod.POST)
    public ModelAndView update(@RequestParam String prodId, @RequestParam String
prodName, @RequestParam String prodDesc, @RequestParam Double prodPrice) {

        Product p = productRepository.findById(prodId).get();
        p.setProdName(prodName);
        p.setProdDesc(prodDesc);
        p.setProdPrice(prodPrice);
        productRepository.save(p);
        return new ModelAndView("redirect:/products");
    }

    @RequestMapping("/delete")
    public ModelAndView delete(@RequestParam String prodId) {
        return new ModelAndView("redirect:/products/delete/" + prodId);
    }

    @RequestMapping(value = "products/delete/{id}", method = RequestMethod.GET)
    public ModelAndView deleteProduct(@PathVariable("id") String prodId) {
        productRepository.deleteById(prodId);
        return new ModelAndView("redirect:/products");
    }
}
```

ProductRepository Interface:

```
@Repository
public interface ProductRepository extends MongoRepository<Product, String> {
}
```

Some snippets from the view templates:

```
<body>
  <div id="form1-a"><section class="mbr-section mbr-section-small" data-rv-view="26"
style="background-color: rgb(255, 255, 255); padding-top: 4.5rem; padding-bottom:
4.5rem;">

    <div class="container">
        <div class="row">
            <div class="col-sm-8 col-sm-offset-2" data-form-type="formoid">
                <h2 class="mbr-section-title display-3 text-xs-center mbr-editable-
content">Edit Product</h2>
                <form action="/update" method="POST" data-form-title="Create
Product">
                    <div class="form-group" style="display: none;">
                        <input type="text" class="form-control" name="prodId"
required="" th:value="${product.id}">
                    </div>

                    <div class="form-group">
                        <input type="text" class="form-control" name="prodName"
required="" placeholder="Name*" data-form-field="Name"
th:value="${product.prodName}">
                    </div>

                    <div class="form-group">
                        <input type="text" class="form-control" name="prodDesc"
required="" placeholder="Description" data-form-field="Description"
th:value="${product.prodDesc}">
                    </div>

                    <div class="form-group">
                        <input type="number" class="form-control" name="prodPrice"
required="" placeholder="Price*" data-form-field="Price" min="0" step="0.01"
```

```
th:value="${product.prodPrice}">
                    </div>

                    <div class="text-xs-right"><button type="submit" class="btn btn-
secondary-outline mbr-editable-button">Save</button></div>
                </form>
            </div>
        </div>
    </div>
</section></div>
```

3.2. Final Project

3.2.1. Design Documents

Description:

The School Catalog Web Application is an application that different users can use. Every user is directed after login to his corresponding web page.

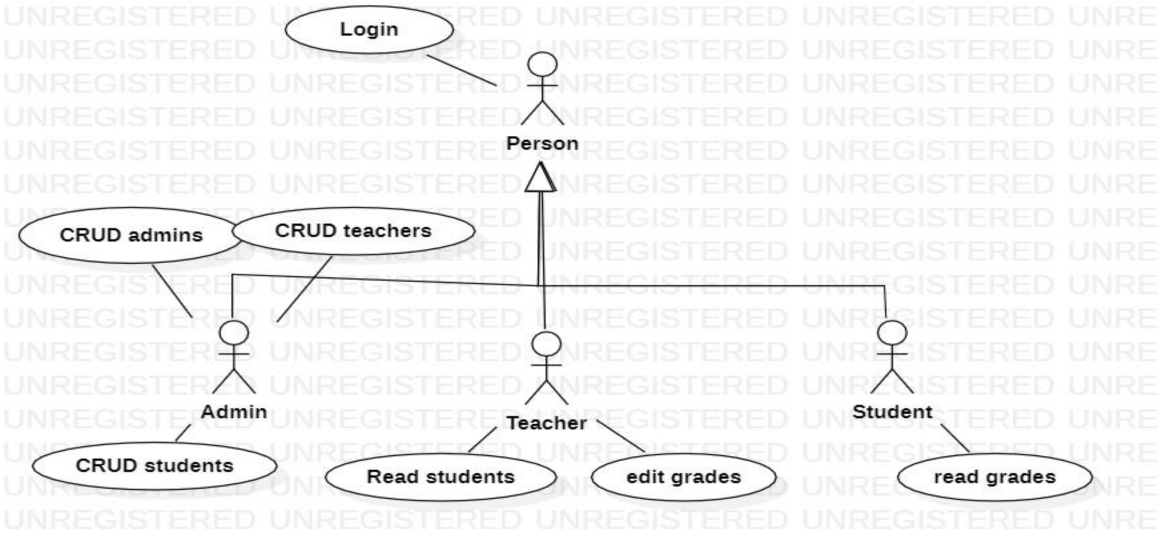
An Admin can create, edit, delete other admins, teachers or students.

A Teacher can see the students in his class and their grades, he can edit the grades.

A Student can see the grades for all subjects.

The definition of requirements and the design was done together.

The Use Case Diagram:



After defining and having a complete Use Case Diagram, we have worked on user stories with my colleague. They are based on the general form described by Mike Cohn:

As a (role) I want (something) so that (benefit).

As an *admin* I want to *create new persons*[1] so that *I will have new person*.

As an *admin* I want to *edit persons* so that *the person will be modified.*

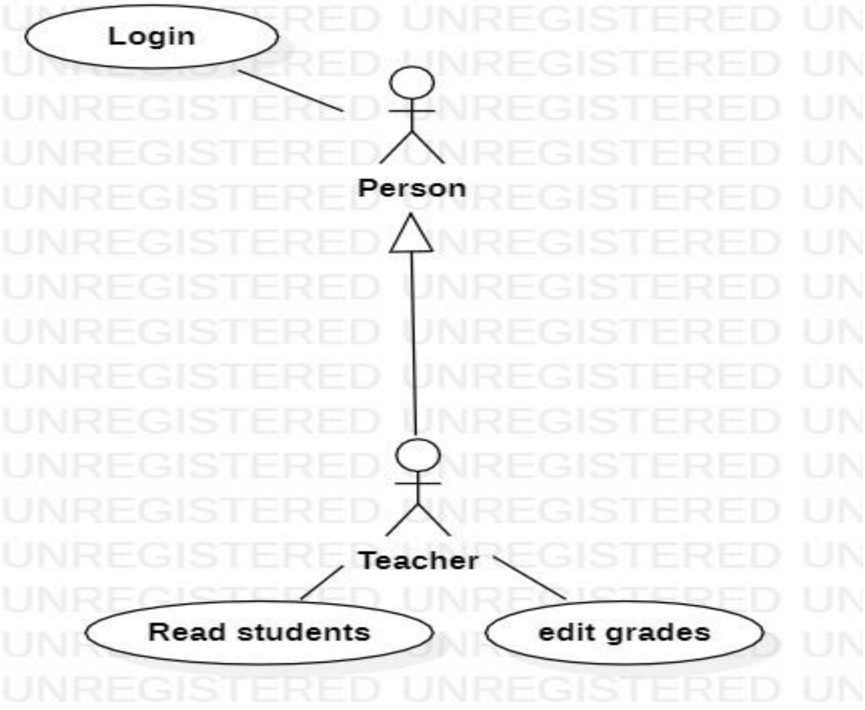
As an *admin* I want to *delete persons* so that *they will not appear in database.*

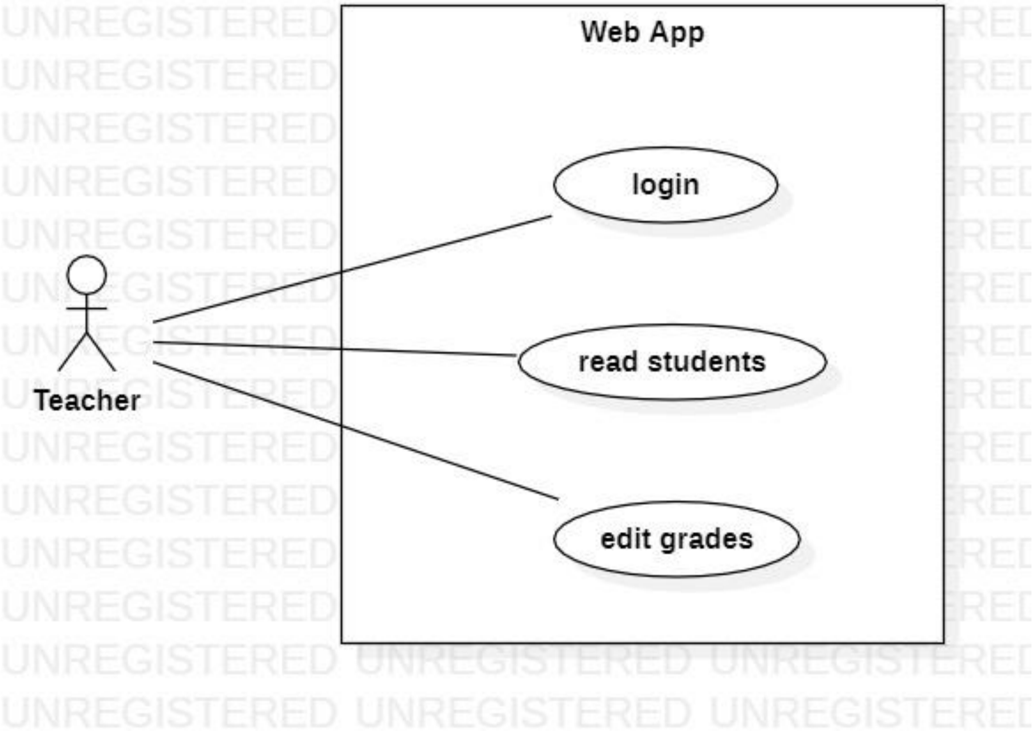
As a *teacher* I want to *see all my students* so that *I can see their grades and have an overview.*

As a *teacher* I want to *edit grades* so that *students can have their grades*.

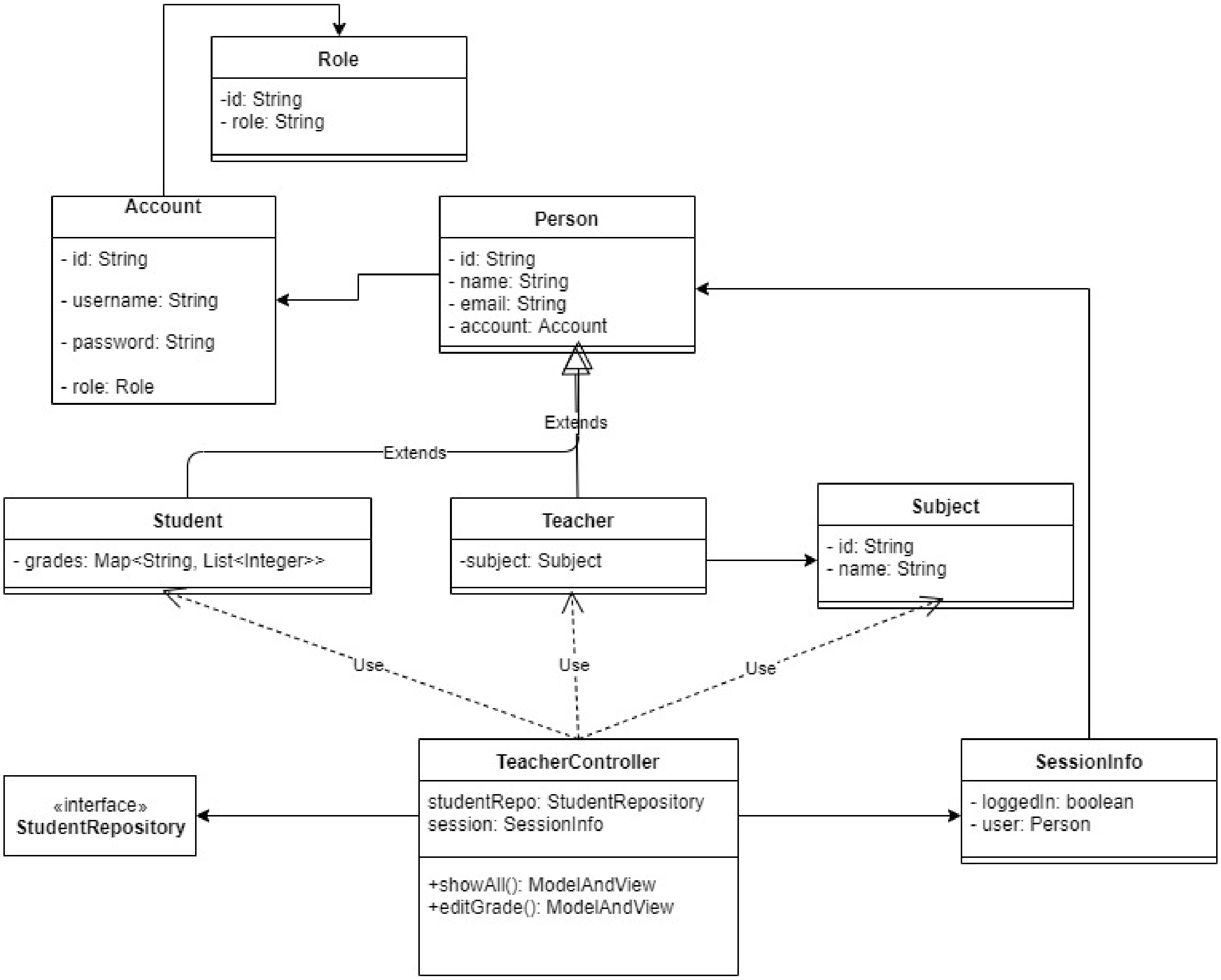
As a *student* I want to *see my grades* so that *I will know my school situation.*

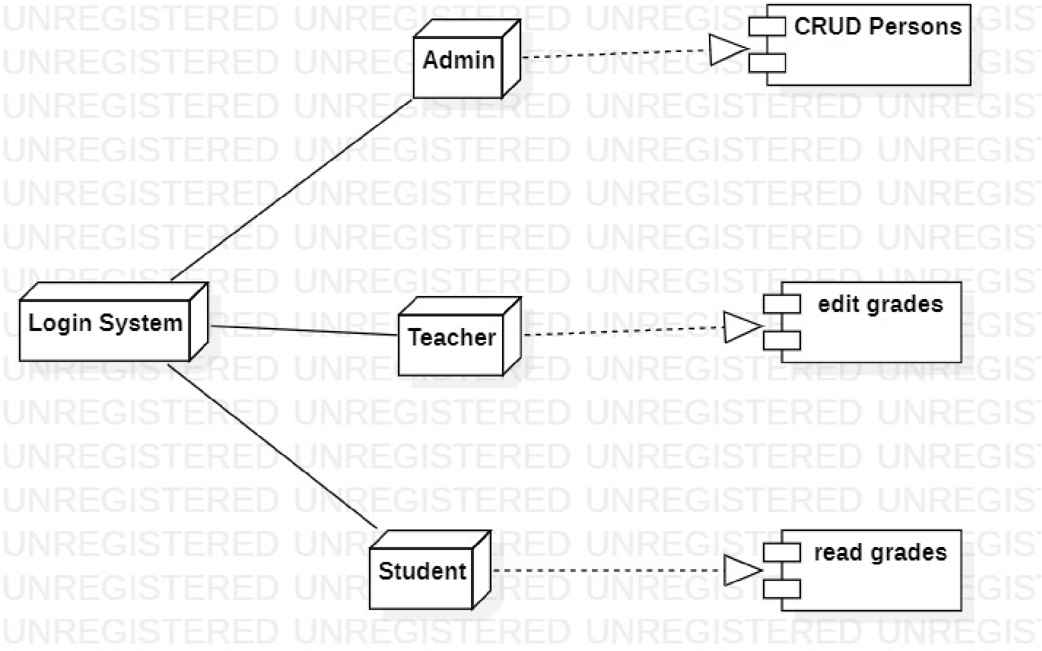
Then I have extracted from both use case diagram and user stories that I have to design and implement.


Login
Person
Teacher
Read students
edit grades


Web App
login
read students
edit grades
Teacher

To design the functionalities mentioned above, I have designed the following UML class diagram that is the abstraction of my task in the project: implementing the teacher functionalities.





The application is based on the Model View Controller design pattern, so we can see a classification of the classes.

Models:

Person is the generalization of Teacher and Student, they have everything a person has and they can do what a person can do (login). Account is an important model for the login part, every person has an account to be able to log in to the system. These classes are part of the model package.

Controller:

The TeacherController class maps the request from the view to the models and corresponding methods. It contains methods that selects all students and lists them on the web page that requests this functionality and has a method that solves the situation when the teacher wants to add or edit a grade. When this happens, the student is notified and his dates are updated (similar to the Observer design pattern).

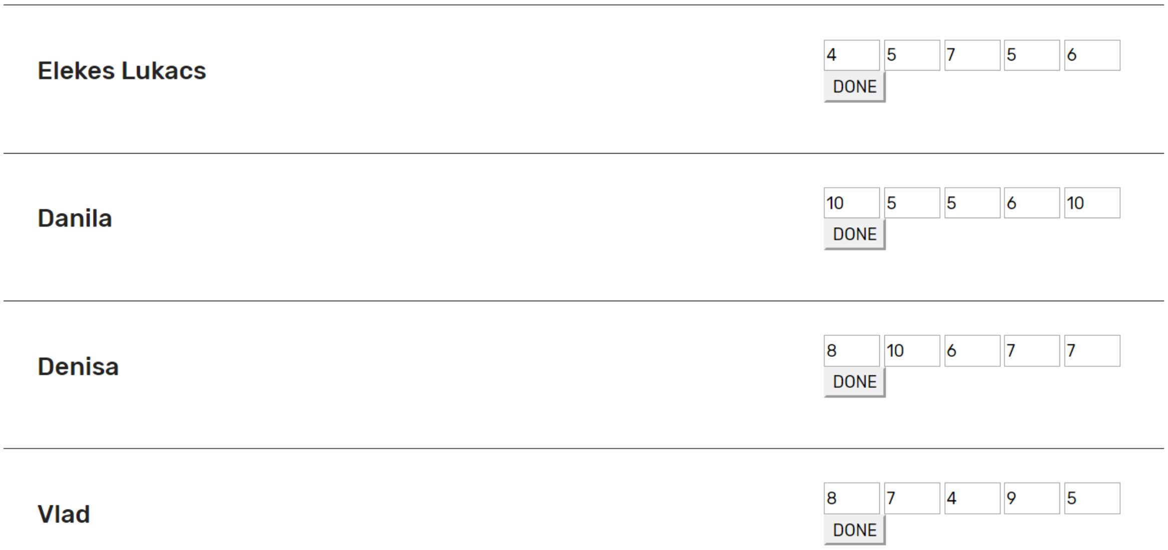
StudentRepository is an interface which makes possible to access data from the database and query some data.

SessionInfo is a class that gives information which person is associated with the account that is logged in.

Subject and Role classes are utility classes, that are entities that can be attributes of some persons.

View:

The view package consists of CSS and HTML templates. Thymeleaf was used to dynamically write HTML code, all the necessary data for parameters in the view templates are passed from the controller.



Elekes Lukacs	4	5	7	5	6	DONE
Danila	10	5	5	6	10	DONE
Denisa	8	10	6	7	7	DONE
Vlad	8	7	4	9	5	DONE

(Teacher can see the students and the grades at his subject.)

User Stories and BDD:

Behavior Driven Development was used to get familiarized with the testing and it's iterations. Some of the user stories scenarios were described by my colleague and me and were implemented. An example is presented:

Feature file:

```
Feature: teacher operations

  Scenario: teacher wants to go to teacher page
    When the teacher click the login button
    Then the teacher is at the "http://localhost:8080/teacher"
```

And the test case that implements this scenario described in natural language:

```
public class TeacherStepDefinitions extends SpringIntegrationTest{
    String response;
    String url = DEFAULT_URL;

    @When("^the teacher click the login button")
    public void the_teacher_click_the_login_button() throws Throwable{

        response = url + "teacher";

    }

//    @When("^the teacher click the done button")
//    public void the_teacher_click_the_done_button() throws Throwable{
//        response = url + "showStudents";
//    }

    @Then("^the theacher is at the \"([^\"]*)\"$")
    public void the_teacher_is_at_the_teacher_page(String s) throws Throwable{
        assertEquals(s, response);
    }
}
```

The Cucumber tool was used for this purpose.

3.2.2 Implementation, Source Code:

Teacher class:

```
@RestController
public class TeacherController {

  @Autowired
  AccountRepository accountRepo;

  @Autowired
  RoleRepository roleRepo;

  @Autowired
  SubjectRepository subjectRepo;

  @Autowired
  StudentRepository studentRepo;

  @Autowired
  TeacherRepository teacherRepo;

  @Autowired
  AdminRepository adminRepo;

  @Autowired
  SessionInfo session;

  @RequestMapping(value = "/teacher")
  public ModelAndView showAll(){
    Teacher connected = (Teacher)session.getUser();
    ModelAndView modelAndView = new ModelAndView();
    List<Student> students = studentRepo.findAll();
    for(Student s : students){
      s.getGradesForSubject(connected.getSubject());
      System.out.println(s.getArrayOfGrades().toString());
      System.out.println(s.getName());
    }

    ArrayList<Integer> note = new ArrayList<>();
    note.add(5);
    note.add(9);
    modelAndView.setViewName("teacher");
    modelAndView.addObject("students", students);
    modelAndView.addObject("teacher", connected);
    //modelAndView.addObject("groups", connected.getGroups());
    //modelAndView.addObject("subject", connected.getSubject().getName());
    modelAndView.addObject("subject", connected.getSubject());
    modelAndView.addObject("note",  note);
    return modelAndView;
  }

  @RequestMapping(value = "/showStudents")
  public ModelAndView selectGroup(@RequestParam String group){
    return new ModelAndView("redirect:/teacher/" + group);
  }

  @RequestMapping(value = "/teacher/{group}")
  public ModelAndView showGroup(@PathVariable("group") String group){
    Teacher connected = (Teacher)session.getUser();
    ModelAndView modelAndView = new ModelAndView();
    List<Student> students = studentRepo.findAll();
    //trebuie find by group

    modelAndView.setViewName("teacher");
    modelAndView.addObject("students", students);
    // modelAndView.addObject("groups", connected.getGroups());
    return modelAndView;
  }

  @RequestMapping(value = "/done/{id}")
  public ModelAndView editGrade(@PathVariable("id") String student,
@PathParam("nota1") int nota1, @PathParam("nota2") int nota2,
                                @PathParam("nota3") int nota3, @PathParam("nota4")
int nota4, @PathParam("nota5") int nota5){
    System.out.println(student);
    Student stud = studentRepo.findById(student).get();
    Teacher current = (Teacher)session.getUser();
    Subject subj = current.getSubject();
    //  ArrayList<Integer> note = new ArrayList<>();
    if(stud.getGrades().containsKey(subj.getName())){
      stud.getGrades().get(subj.getName()).clear();
      stud.getGrades().get(subj.getName()).add(nota1);
      stud.getGrades().get(subj.getName()).add(nota2);
      stud.getGrades().get(subj.getName()).add(nota3);
      stud.getGrades().get(subj.getName()).add(nota4);
      stud.getGrades().get(subj.getName()).add(nota5);
    }
    else{
      ArrayList<Integer> note = new ArrayList<>();
      note.add(nota1);
      note.add(nota2);
      note.add(nota3);
      note.add(nota4);
      note.add(nota5);

      stud.getGrades().put(subj.getName(), note);
    }
    studentRepo.save(stud);
    return new ModelAndView("redirect:/teacher");
  }
}
```

Person class:

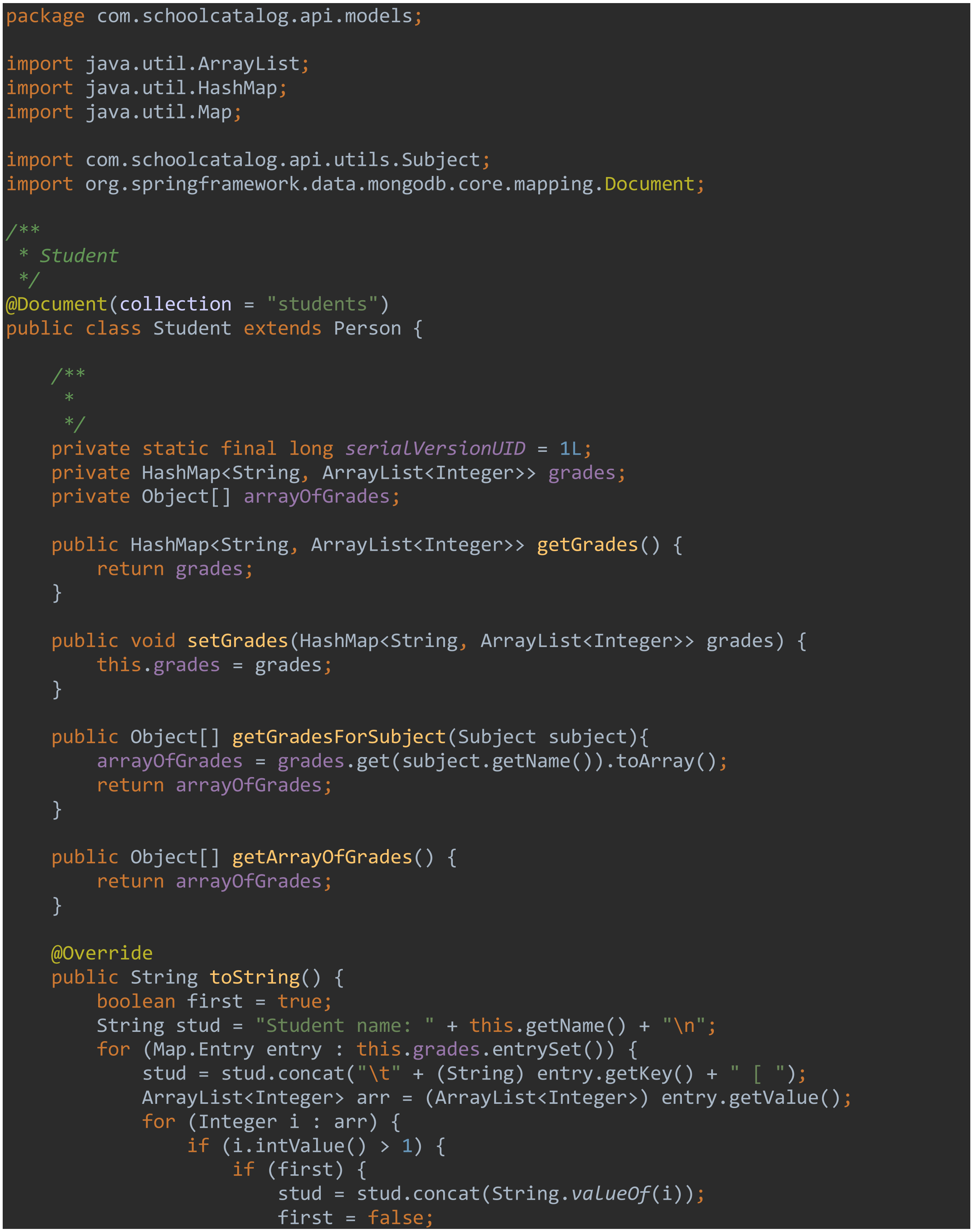
```
package com.schoolcatalog.api.models;

import java.io.Serializable;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.index.Indexed;
import org.springframework.data.mongodb.core.mapping.DBRef;

/**
 * Person
 */
public class Person implements Serializable {

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    @Id
    private String id;

    private String name;
    private String email;

    @DBRef
    @Indexed(unique = true)
    private Account account;

    public Person() {
    }

    public String getId() {
        return this.id;
    }

    public void setId(String id) {
        this.id = id;
```

```java
    }

    public String getName() {
        return this.name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public Account getAccount() {
        return this.account;
    }

    public void setAccount(Account account) {
        this.account = account;
    }
}
```

Teacher class:

```java
package com.schoolcatalog.api.models;

import com.schoolcatalog.api.utils.Subject;
import org.springframework.data.mongodb.core.mapping.Document;

/**
 * Teacher
 */
@Document(collection = "teachers")
public class Teacher extends Person {
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    private Subject subject;

    public Subject getSubject() {
        return subject;
    }

    public void setSubject(Subject subject) {
        this.subject = subject;
    }

}
```

Student class:



```java
package com.schoolcatalog.api.models;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

import com.schoolcatalog.api.utils.Subject;
import org.springframework.data.mongodb.core.mapping.Document;

/**
 * Student
 */
@Document(collection = "students")
public class Student extends Person {

    /**
     *
     */
    private static final long serialVersionUID = 1L;
    private HashMap<String, ArrayList<Integer>> grades;
    private Object[] arrayOfGrades;

    public HashMap<String, ArrayList<Integer>> getGrades() {
        return grades;
    }

    public void setGrades(HashMap<String, ArrayList<Integer>> grades) {
        this.grades = grades;
    }

    public Object[] getGradesForSubject(Subject subject){
        arrayOfGrades = grades.get(subject.getName()).toArray();
        return arrayOfGrades;
    }

    public Object[] getArrayOfGrades() {
        return arrayOfGrades;
    }

    @Override
    public String toString() {
        boolean first = true;
        String stud = "Student name: " + this.getName() + "\n";
        for (Map.Entry entry : this.grades.entrySet()) {
            stud = stud.concat("\t" + (String) entry.getKey() + " [ ");
            ArrayList<Integer> arr = (ArrayList<Integer>) entry.getValue();
            for (Integer i : arr) {
                if (i.intValue() > 1) {
                    if (first) {
                        stud = stud.concat(String.valueOf(i));
                        first = false;
                    } else {
                        stud = stud.concat(", " + i);
                    }
                }
            }
            stud = stud.concat(" ]\n");
        }
        stud = stud.concat("\n");
        return stud;
    }
}
```

The Student has an attribute arrayOfGrades, which was forced to have to be able to print the grades using thymeleaf on the views.

Account class:

```
package com.schoolcatalog.api.models;

import java.io.Serializable;

import com.schoolcatalog.api.utils.Role;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.index.Indexed;
import org.springframework.data.mongodb.core.mapping.DBRef;
import org.springframework.data.mongodb.core.mapping.Document;

/**
 * Account
 */
@Document(collection = "accounts")
public class    Account implements Serializable {

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    @Id
    private String id;

    @Indexed(unique = true)
    private String username;
    private String password;

    @DBRef
    private Role role;

    public String getId() {
        return this.id;
    }
    public void setId(String id) {
        this.id = id;
    }

    public String getUsername() {
        return this.username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return this.password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public Role getRole() {
        return this.role;
    }

    public void setRole(Role role) {
        this.role = role;
    }
}
```

Role class:

```
package com.schoolcatalog.api.utils;

import java.io.Serializable;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.index.Indexed;
import org.springframework.data.mongodb.core.mapping.Document;

/**
 * Role
 */
@Document(collection = "roles")
public class Role implements Serializable {

    @Id
    private String id;

    @Indexed(unique = true)
    private String role;

    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }

    public String getRole() {
        return role;
    }

    public void setRole(String role) {
        this.role = role;
    }
}
```

Subject class:

```
package com.schoolcatalog.api.utils;

import java.io.Serializable;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.index.IndexDirection;
import org.springframework.data.mongodb.core.index.Indexed;
import org.springframework.data.mongodb.core.mapping.Document;

/**
 * Subject
 */
@Document(collection = "subjects")
public class Subject implements Serializable {

    @Id
    private String id;

    @Indexed(unique = true, direction = IndexDirection.DESCENDING)
    private String name;


    public String getId() {
        return this.id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getName() {
        return this.name;
    }

    public void setName(String name) {
        this.name = name;
    }

}
```

StudentRepository interface:

```


/**
 * StudentRepository
 */
@Repository
public interface StudentRepository extends MongoRepository<Student, String> {
    Student findByAccount(Account account);
}
```

SessionInfo class:

```
package com.schoolcatalog.app.utils;

import com.schoolcatalog.api.models.Person;

import org.springframework.context.annotation.Scope;
import org.springframework.context.annotation.ScopedProxyMode;
import org.springframework.stereotype.Component;
import org.springframework.web.context.WebApplicationContext;

/**
 * Session
 */
@Component
@Scope(value = WebApplicationContext.SCOPE_SESSION, proxyMode =
ScopedProxyMode.TARGET_CLASS)
public class SessionInfo {

  private boolean loggedIn;
  private Person user;

  public SessionInfo() {
  }

  public boolean isLoggedIn() {
    return loggedIn;
  }

  public void setLoggedIn(boolean loggedIn) {
    this.loggedIn = loggedIn;
  }

  public Person getUser() {
    return user;
  }

  public void setUser(Person user) {
    this.user = user;
  }
}
```

teacher.html:

```
<body>
  <section class="mbr-section form3 cid-rLEKaK6Ksg" id="form3-c">

    <div class="container">
      <div class="row justify-content-center">
        <div class="title col-12 col-lg-8">
          <h2 class="align-center pb-2 mbr-fonts-style display-2" th:text =
"${teacher.name + ' - ' + subject.name}">
            Choose Group</h2>
        </div>
      </div>

      <div class="row py-2 justify-content-center">
        <div class="col-12 col-lg-6  col-md-8 " data-form-type="formoid">

          <form action="print" class="mbr-form form-with-styler">
            <div class="col-auto input-group-btn">
              <button type="submit" class="btn btn-primary  display-4">PRINT
STUDENTS</button>
            </div>
          </form>

        </div>
      </div>
    </div>
  </section>

  <section class="services5 cid-rLEKEo4ljY" id="services5-e">

    <!--Container-->
    <div class="container">
      <div class="row">
        <!--Titles-->
        <div class="title pb-5 col-12"></div>

        <!--Card-1-->
        <div class="card px-3 col-12" th:each="student : ${students}">
          <div class="card-wrapper media-container-row media-container-row">
            <div class="card-box">
              <div class="top-line pb-3">
                <h4 class="card-title mbr-fonts-style display-5"
th:text="${student.name}"> </h4>

                <form th:action="${'/done/'+student.id}">
```

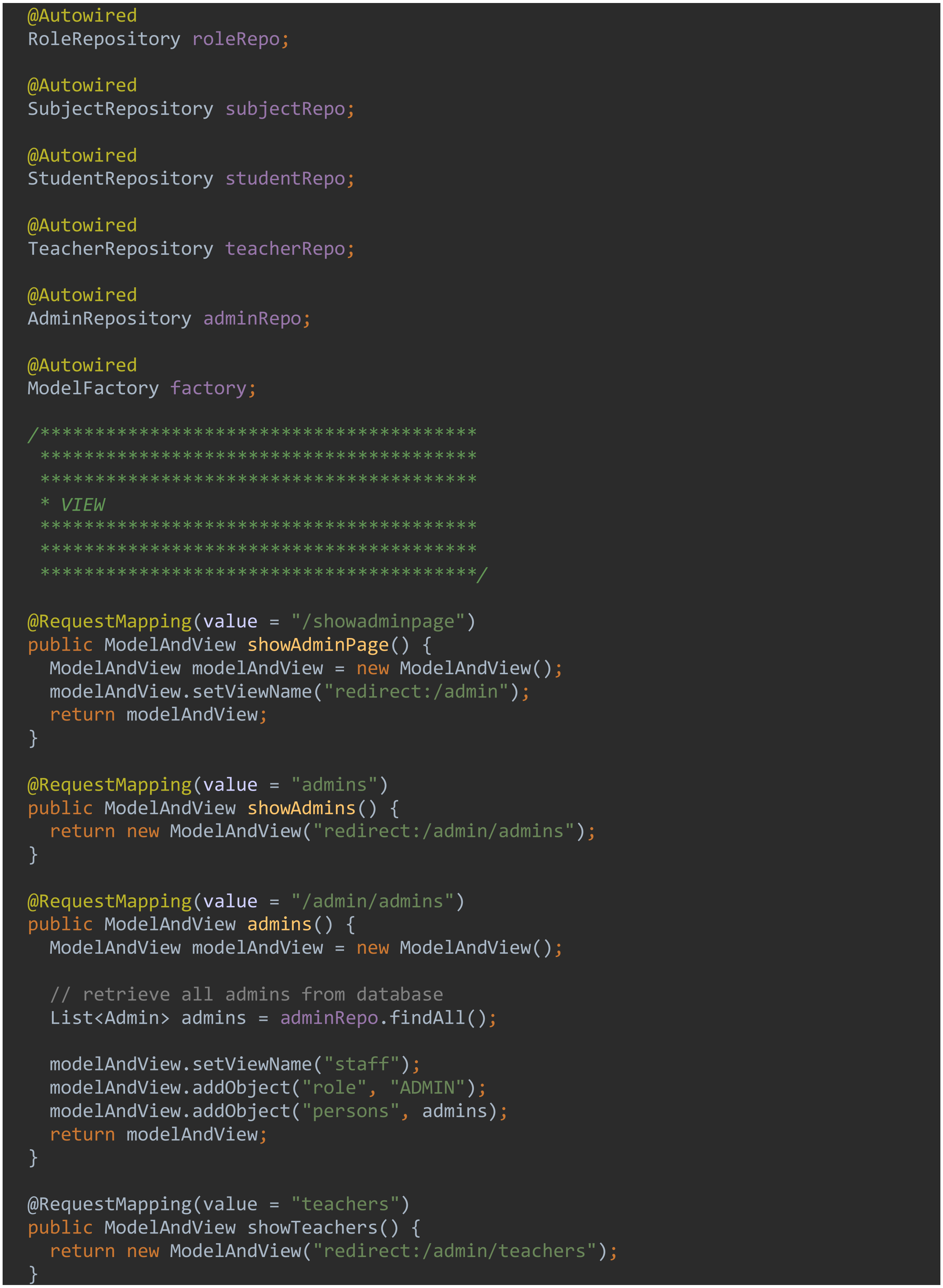
```
                  <input type="number" name="nota1" min="1" max="10"
th:value="${student.arrayOfGrades[0]}">
                  <input type="number" name="nota2" min="1" max="10"
th:value="${student.arrayOfGrades[1]}">
                  <input type="number" name="nota3" min="1" max="10"
th:value="${student.arrayOfGrades[2]}">
                  <input type="number" name="nota4" min="1" max="10"
th:value="${student.arrayOfGrades[3]}">
                  <input type="number" name="nota5" min="1" max="10"
th:value="${student.arrayOfGrades[4]}">

                  <button type="submit">DONE</button>
                </form>

              </div>
              <div class="bottom-line">

              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  </section>


  <script src="assets/web/assets/jquery/jquery.min.js"></script>
  <script src="assets/popper/popper.min.js"></script>
  <script src="assets/bootstrap/js/bootstrap.min.js"></script>
  <script src="assets/tether/tether.min.js"></script>
  <script src="assets/smoothscroll/smooth-scroll.js"></script>
  <script src="assets/theme/js/script.js"></script>
  <script src="assets/formoid/formoid.min.js"></script>

</body>
```

This is the implementation of my part in this project (and the test cases presented before).

Additional implementation and source code written by the other members is presented below:

AdminController class:

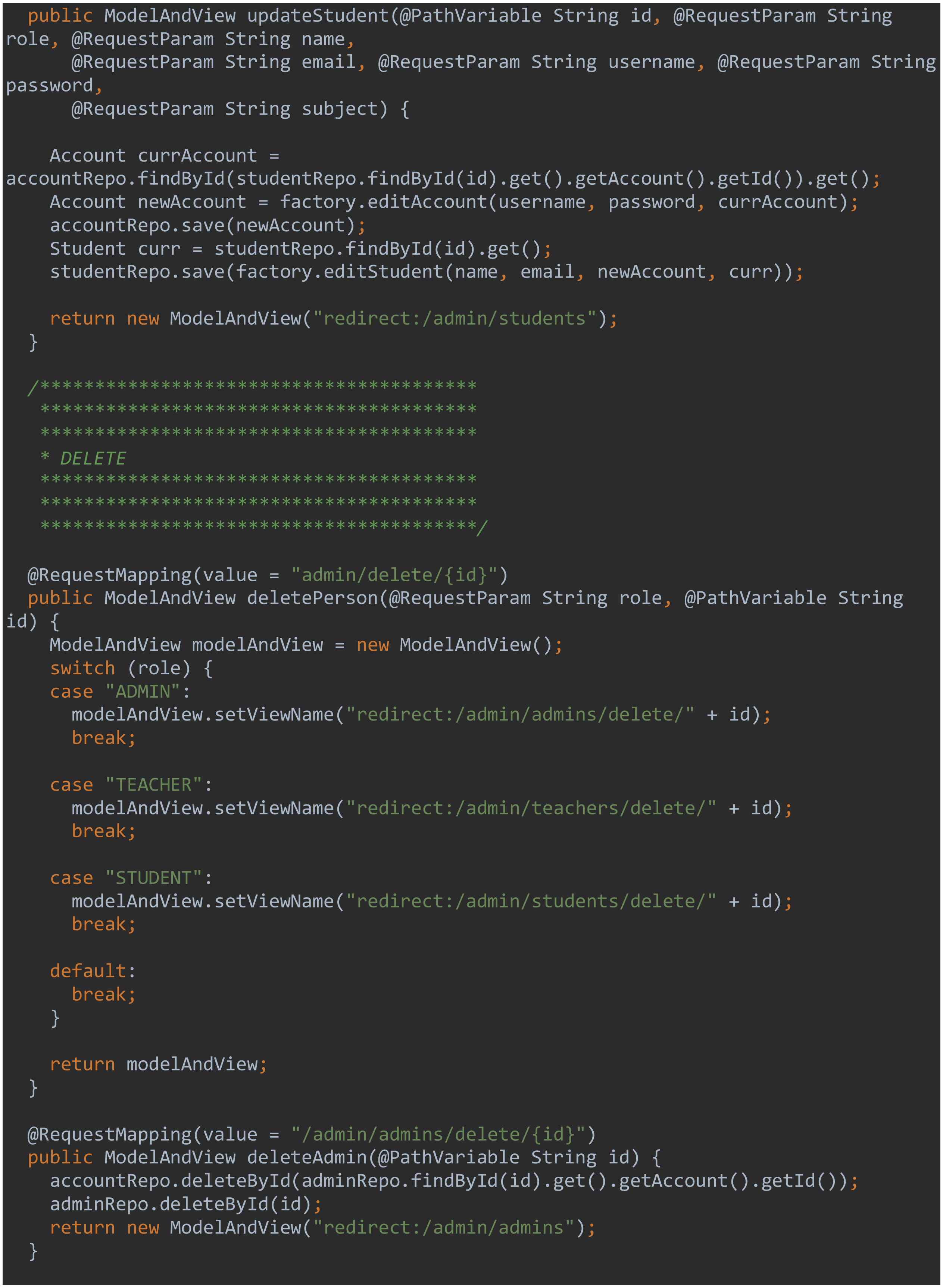
```
package com.schoolcatalog.app.controllers;


/**
 * AdminController
 */
@RestController
public class AdminController {

  @Autowired
  AccountRepository accountRepo;

```



```
@Autowired
RoleRepository roleRepo;

@Autowired
SubjectRepository subjectRepo;

@Autowired
StudentRepository studentRepo;

@Autowired
TeacherRepository teacherRepo;

@Autowired
AdminRepository adminRepo;

@Autowired
ModelFactory factory;

/*****************************************
 *****************************************
 *****************************************
 * VIEW
 *****************************************
 *****************************************
 *****************************************/

@RequestMapping(value = "/showadminpage")
public ModelAndView showAdminPage() {
  ModelAndView modelAndView = new ModelAndView();
  modelAndView.setViewName("redirect:/admin");
  return modelAndView;
}

@RequestMapping(value = "admins")
public ModelAndView showAdmins() {
  return new ModelAndView("redirect:/admin/admins");
}

@RequestMapping(value = "/admin/admins")
public ModelAndView admins() {
  ModelAndView modelAndView = new ModelAndView();

  // retrieve all admins from database
  List<Admin> admins = adminRepo.findAll();

  modelAndView.setViewName("staff");
  modelAndView.addObject("role", "ADMIN");
  modelAndView.addObject("persons", admins);
  return modelAndView;
}

@RequestMapping(value = "teachers")
public ModelAndView showTeachers() {
  return new ModelAndView("redirect:/admin/teachers");
}
```

```
@RequestMapping(value = "/admin/teachers")
public ModelAndView teachers() {
  ModelAndView modelAndView = new ModelAndView();

  // retrieve all teachers from database
  List<Teacher> teachers = teacherRepo.findAll();

  modelAndView.setViewName("staff");
  modelAndView.addObject("role", "TEACHER");
  modelAndView.addObject("persons", teachers);
  return modelAndView;
}

@RequestMapping(value = "students")
public ModelAndView showStudents() {
  return new ModelAndView("redirect:/admin/students");
}

@RequestMapping(value = "/admin/students")
public ModelAndView students() {
  ModelAndView modelAndView = new ModelAndView();

  // retrieve all students from database
  List<Student> students = studentRepo.findAll();

  modelAndView.setViewName("staff");
  modelAndView.addObject("role", "STUDENT");
  modelAndView.addObject("persons", students);
  return modelAndView;
}

/****************************************
 ****************************************
 ****************************************
 * CREATE
 ****************************************
 ****************************************
 ****************************************/

@RequestMapping(value = "/addPerson")
public ModelAndView addPerson(@RequestParam String role) {
  switch (role) {
  case "ADMIN":
    return new ModelAndView("redirect:/admin/admins/add" + role);

  case "TEACHER":
    return new ModelAndView("redirect:/admin/teachers/add" + role);

  default:
    return new ModelAndView("redirect:/admin/students/add" + role);
  }
}

@RequestMapping(value = "/admin/admins/addADMIN")
```

```
  public ModelAndView addAdmin() {
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.setViewName("addstaff");
    modelAndView.addObject("role", "ADMIN");
    return modelAndView;
  }

  @RequestMapping(value = "/admin/teachers/addTEACHER")
  public ModelAndView addTeacher() {
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.setViewName("addstaff");
    modelAndView.addObject("role", "TEACHER");
    modelAndView.addObject("subjects", subjectRepo.findAll());
    return modelAndView;
  }

  @RequestMapping(value = "/admin/students/addSTUDENT")
  public ModelAndView addStudent() {
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.setViewName("addstaff");
    modelAndView.addObject("role", "STUDENT");
    return modelAndView;
  }

  @RequestMapping(value = { "/admin/admins/save", "/admin/teachers/save",
"/admin/students/save" })
  public ModelAndView savePerson(@RequestParam String role, @RequestParam String
name, @RequestParam String email,
      @RequestParam String username, @RequestParam String password, @RequestParam
String subject) {

    String target = "";
    switch (role) {
    case "ADMIN":
      target = "/admin/admins";
      Account adminAccount = factory.createAccount(username, password,
roleRepo.findByRole(role));
      accountRepo.save(adminAccount);
      Admin admin = factory.createAdmin(name, email, adminAccount);
      adminRepo.save(admin);
      break;

    case "TEACHER":
      target = "/admin/teachers";
      Account teacherAccount = factory.createAccount(username, password,
roleRepo.findByRole(role));
      accountRepo.save(teacherAccount);
      Teacher teacher = factory.createTeacher(name, email, teacherAccount,
subjectRepo.findByName(subject));
      teacherRepo.save(teacher);
      break;

    case "STUDENT":
      target = "/admin/students";
      Account studentAccount = factory.createAccount(username, password,
roleRepo.findByRole(role));
      accountRepo.save(studentAccount);
      Student student = factory.createStudent(name, email, studentAccount,
subjectRepo.findAll());
      studentRepo.save(student);
      break;

    default:
      break;
    }

    return new ModelAndView("redirect:" + target);
  }

  /****************************************
   ****************************************
   ****************************************
   * EDIT
   ****************************************
   ****************************************
   ****************************************/

  @RequestMapping(value = "admin/edit/{id}")
  public ModelAndView editPerson(@RequestParam String role, @PathVariable String id)
{
    ModelAndView modelAndView = new ModelAndView();
    switch (role) {
    case "ADMIN":
      modelAndView.setViewName("redirect:/admin/admins/edit/" + id);
      break;

    case "TEACHER":
      modelAndView.setViewName("redirect:/admin/teachers/edit/" + id);
      break;

    case "STUDENT":
      modelAndView.setViewName("redirect:/admin/students/edit/" + id);
      break;

    default:
      break;
    }

    return modelAndView;
  }

  @RequestMapping(value = "/admin/admins/edit/{id}")
  public ModelAndView editAdmin(@PathVariable String id) {
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.addObject("person", adminRepo.findById(id).get());
    modelAndView.addObject("role", "ADMIN");
    modelAndView.setViewName("editstaff");
    return modelAndView;
  }
  @RequestMapping(value = "/admin/teachers/edit/{id}")
  public ModelAndView editTeacher(@PathVariable String id) {
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.addObject("person", teacherRepo.findById(id).get());
    modelAndView.addObject("role", "TEACHER");
    modelAndView.addObject("subjects", subjectRepo.findAll());
    modelAndView.setViewName("editstaff");
    return modelAndView;
  }

  @RequestMapping(value = "/admin/students/edit/{id}")
  public ModelAndView editStudent(@PathVariable String id) {
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.addObject("person", studentRepo.findById(id).get());
    modelAndView.addObject("role", "STUDENT");
    modelAndView.setViewName("editstaff");
    return modelAndView;
  }

  @RequestMapping(value = "/admin/admins/edit/update/{id}")
  public ModelAndView updateAdmin(@PathVariable String id, @RequestParam String role,
@RequestParam String name,
      @RequestParam String email, @RequestParam String username, @RequestParam String
password,
      @RequestParam String subject) {

    Account currAccount =
accountRepo.findById(adminRepo.findById(id).get().getAccount().getId()).get();
    Account newAccount = factory.editAccount(username, password, currAccount);
    accountRepo.save(newAccount);
    Admin curr = adminRepo.findById(id).get();
    adminRepo.save(factory.editAdmin(name, email, newAccount, curr));

    return new ModelAndView("redirect:/admin/admins");
  }

  @RequestMapping(value = "/admin/teachers/edit/update/{id}")
  public ModelAndView updateTeacher(@PathVariable String id, @RequestParam String
role, @RequestParam String name,
      @RequestParam String email, @RequestParam String username, @RequestParam String
password,
      @RequestParam String subject) {

    Account currAccount =
accountRepo.findById(teacherRepo.findById(id).get().getAccount().getId()).get();
    Account newAccount = factory.editAccount(username, password, currAccount);
    accountRepo.save(newAccount);
    Teacher curr = teacherRepo.findById(id).get();
    teacherRepo.save(factory.editTeacher(name, email, newAccount,
subjectRepo.findByName(subject), curr));

    return new ModelAndView("redirect:/admin/teachers");
  }

  @RequestMapping(value = "/admin/students/edit/update/{id}")
```



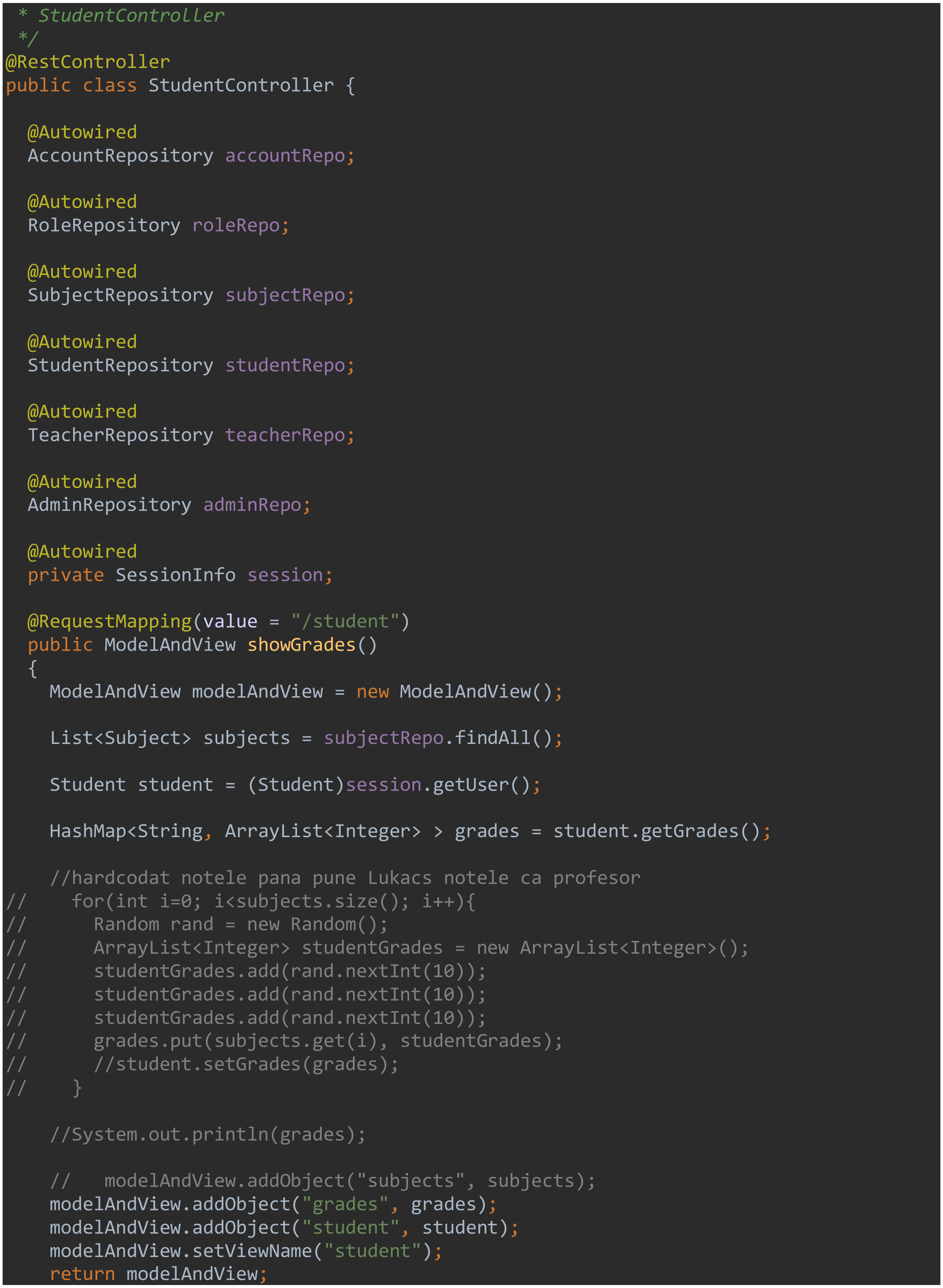
```
  public ModelAndView updateStudent(@PathVariable String id, @RequestParam String
role, @RequestParam String name,
      @RequestParam String email, @RequestParam String username, @RequestParam String
password,
      @RequestParam String subject) {

    Account currAccount =
accountRepo.findById(studentRepo.findById(id).get().getAccount().getId()).get();
    Account newAccount = factory.editAccount(username, password, currAccount);
    accountRepo.save(newAccount);
    Student curr = studentRepo.findById(id).get();
    studentRepo.save(factory.editStudent(name, email, newAccount, curr));

    return new ModelAndView("redirect:/admin/students");
  }

  /****************************************
   ****************************************
   ****************************************
   * DELETE
   ****************************************
   ****************************************
   ****************************************/

  @RequestMapping(value = "admin/delete/{id}")
  public ModelAndView deletePerson(@RequestParam String role, @PathVariable String
id) {
    ModelAndView modelAndView = new ModelAndView();
    switch (role) {
    case "ADMIN":
      modelAndView.setViewName("redirect:/admin/admins/delete/" + id);
      break;

    case "TEACHER":
      modelAndView.setViewName("redirect:/admin/teachers/delete/" + id);
      break;

    case "STUDENT":
      modelAndView.setViewName("redirect:/admin/students/delete/" + id);
      break;

    default:
      break;
    }

    return modelAndView;
  }

  @RequestMapping(value = "/admin/admins/delete/{id}")
  public ModelAndView deleteAdmin(@PathVariable String id) {
    accountRepo.deleteById(adminRepo.findById(id).get().getAccount().getId());
    adminRepo.deleteById(id);
    return new ModelAndView("redirect:/admin/admins");
  }
  @RequestMapping(value = "/admin/teachers/delete/{id}")
  public ModelAndView deleteTeacher(@PathVariable String id) {
    accountRepo.deleteById(teacherRepo.findById(id).get().getAccount().getId());
    teacherRepo.deleteById(id);
    return new ModelAndView("redirect:/admin/teachers");
  }

  @RequestMapping(value = "/admin/students/delete/{id}")
  public ModelAndView deleteStudent(@PathVariable String id) {
    accountRepo.deleteById(studentRepo.findById(id).get().getAccount().getId());
    studentRepo.deleteById(id);
    return new ModelAndView("redirect:/admin/students");
  }
}
```

LoginController class:

```
package com.schoolcatalog.app.controllers;

/**
 * LoginController
 */
@Controller
public class LoginController {

  @Autowired
  AdminRepository adminRepo;

  @Autowired
  TeacherRepository teacherRepo;

  @Autowired
  StudentRepository studentRepo;

  @Autowired
  AccountRepository accountRepo;

  @Autowired
  RoleRepository roleRepo;

  @Autowired
  SessionInfo session;

  @RequestMapping(value = "/login", method = RequestMethod.POST)
  public ModelAndView login(@RequestParam("username") String username,
@RequestParam("password") String password) {

    // check if account exists and if exists what type it is
    Account account = accountRepo.findByUsername(username);
    if (account == null) {
      System.err.println("This account does not exist");
      return null;
    }
    if (!account.getPassword().equals(password)) {
      System.err.println("Incorrect password");
      return null;
    }

    ModelAndView modelAndView = new ModelAndView();
    switch (account.getRole().getRole()) {
    case "ADMIN":
      Admin connectedAdmin = adminRepo.findByAccount(account);
      modelAndView.setViewName("redirect:/admin");
      session.setUser(connectedAdmin);
      break;

    case "TEACHER":
      Teacher connectedTeacher = teacherRepo.findByAccount(account);
      modelAndView.setViewName("redirect:/teacher");
      session.setUser(connectedTeacher);
      break;

    case "STUDENT":
      Student connectedStudent = studentRepo.findByAccount(account);
      modelAndView.setViewName("redirect:/student");
      session.setUser(connectedStudent);
      break;

    default:
      break;
    }

    return modelAndView;
  }

  @RequestMapping(value = "/login", method = RequestMethod.GET)
  public ModelAndView login() {
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.setViewName("login");
    return modelAndView;
  }

  @RequestMapping(value = "/admin", method = RequestMethod.GET)
  public ModelAndView showAdminPage() {
    ModelAndView modelAndView = new ModelAndView();
    modelAndView.setViewName("admin");
    return modelAndView;
  }
}
```

StudentController class:

```
package com.schoolcatalog.app.controllers;



/**
```



```
 * StudentController
 */
@RestController
public class StudentController {

  @Autowired
  AccountRepository accountRepo;

  @Autowired
  RoleRepository roleRepo;

  @Autowired
  SubjectRepository subjectRepo;

  @Autowired
  StudentRepository studentRepo;

  @Autowired
  TeacherRepository teacherRepo;

  @Autowired
  AdminRepository adminRepo;

  @Autowired
  private SessionInfo session;

  @RequestMapping(value = "/student")
  public ModelAndView showGrades()
  {
    ModelAndView modelAndView = new ModelAndView();

    List<Subject> subjects = subjectRepo.findAll();

    Student student = (Student)session.getUser();

    HashMap<String, ArrayList<Integer> > grades = student.getGrades();

    //hardcodat notele pana pune Lukacs notele ca profesor
//    for(int i=0; i<subjects.size(); i++){
//      Random rand = new Random();
//      ArrayList<Integer> studentGrades = new ArrayList<Integer>();
//      studentGrades.add(rand.nextInt(10));
//      studentGrades.add(rand.nextInt(10));
//      studentGrades.add(rand.nextInt(10));
//      grades.put(subjects.get(i), studentGrades);
//      //student.setGrades(grades);
//    }

    //System.out.println(grades);

    //   modelAndView.addObject("subjects", subjects);
    modelAndView.addObject("grades", grades);
    modelAndView.addObject("student", student);
    modelAndView.setViewName("student");
    return modelAndView;
  }

}
```

ConcurencyController class:

```
package com.schoolcatalog.app.concurrency;

@Controller
public class ConcurrencyController {

    @Autowired
    StudentRepository studentRepo;

    @RequestMapping(value = "print")
    public ModelAndView print() throws IOException {
        ArrayList<Student> students = (ArrayList<Student>) studentRepo.findAll();

        ParallelMergeSort pm = new ParallelMergeSort(students);
        Producer producer = new Producer(students);
        Consumer consumer = new Consumer();

        Thread thread = new Thread(pm);
        thread.start();

        Thread t1 = new Thread(producer);
        Thread t2 = new Thread(consumer);
        t1.start();
        t2.start();

        return new ModelAndView("redirect:/teacher");
    }
}
```

Consumer class:

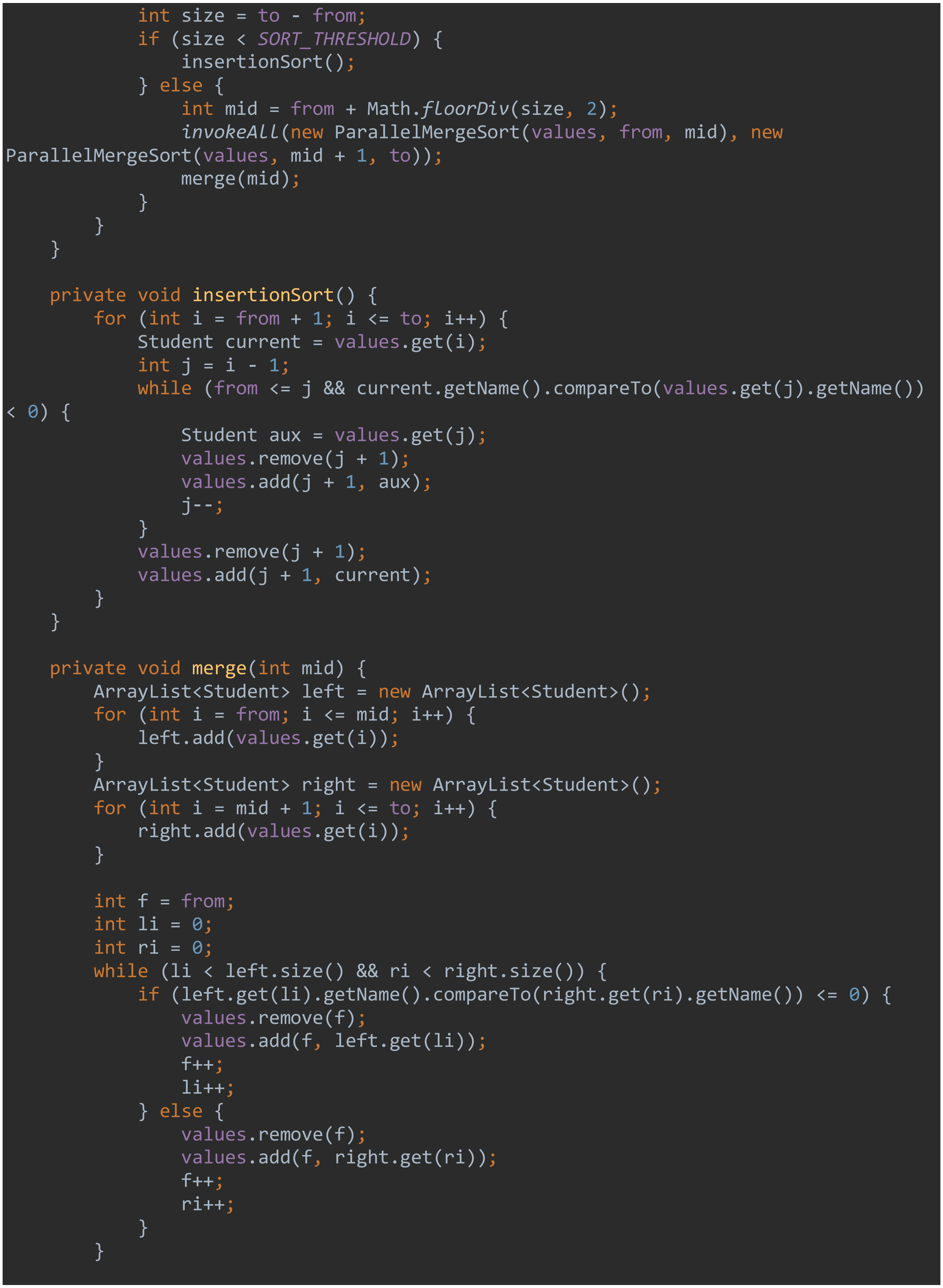
```
package com.schoolcatalog.app.concurrency;


public class Consumer  implements Runnable{
//
//    private FileWriter fileWriter;
//    private PrintWriter printWriter;

    public Consumer() throws IOException {
//        this.fileWriter = new FileWriter("students.txt", true);
//        this.printWriter = new PrintWriter(this.fileWriter);
    }

    @Override
    public void run() {
        System.out.println("Consumer thread started.");
        while (true) {
            try {
                Student student = MyUtils.queue.take();
                System.out.println("take " + student.getName() + " from queue");
                FileWriter fr =new FileWriter("outputFiles/" + student.getName() +
student.getId() + ".txt");
                PrintWriter pr = new PrintWriter(fr);
                pr.println(student.toString());
                pr.close();
                fr.close();
            } catch (InterruptedException | IOException e) {
                e.printStackTrace();
            }

        }

    }
}
```

MyUtils class:

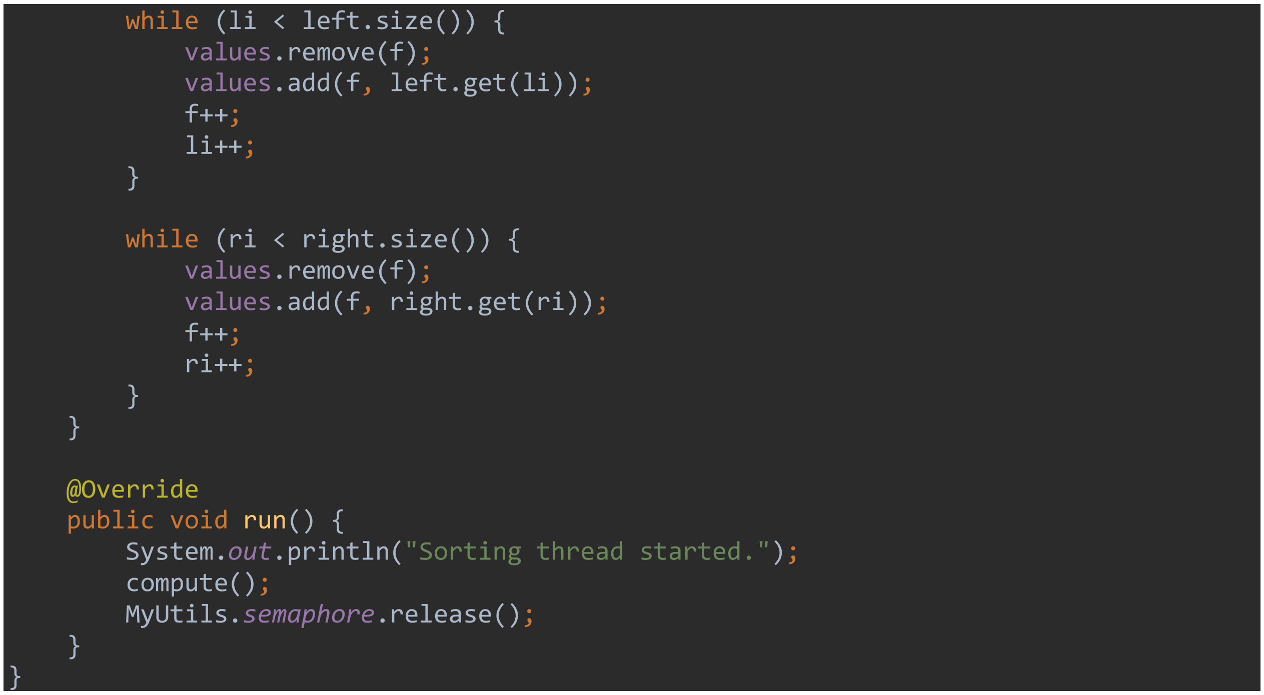
```
package com.schoolcatalog.app.concurrency;

public class MyUtils {

    public static ArrayBlockingQueue<Student> queue = new
ArrayBlockingQueue<Student>(3);
    public static Semaphore semaphore = new Semaphore(0);

}
```

ParallelMergeSort class:

```
public class ParallelMergeSort extends RecursiveAction implements Runnable {

    private static final int SORT_THRESHOLD = 3;

    private ArrayList<Student> values;
    private int from;
    private int to;

    public ParallelMergeSort(ArrayList<Student> values) {
        this(values, 0, values.size() - 1);
    }

    public ParallelMergeSort(ArrayList<Student> values, int from, int to) {
        this.values = values;
        this.from = from;
        this.to = to;
    }

    @Override
    protected void compute() {
        if (from < to) {
```

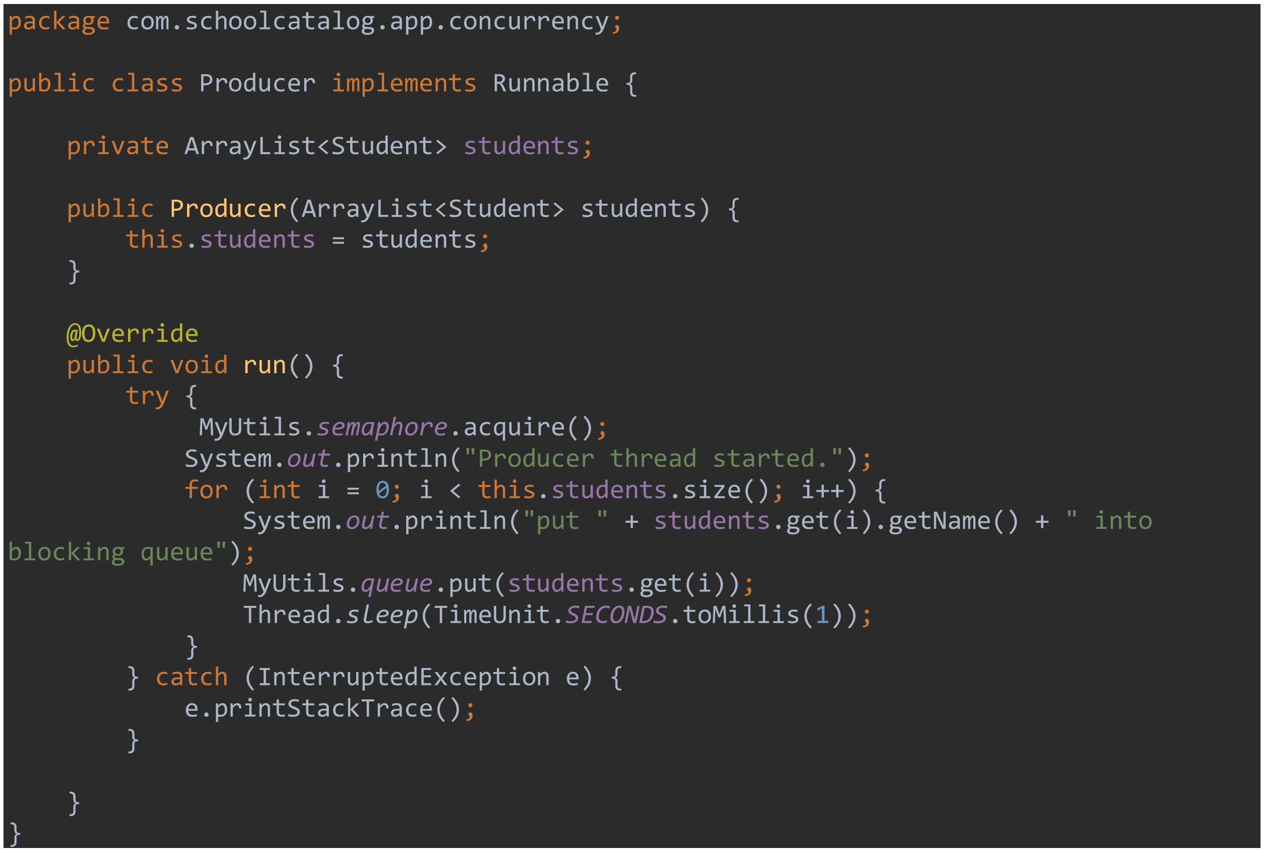


```
            int size = to - from;
            if (size < SORT_THRESHOLD) {
                insertionSort();
            } else {
                int mid = from + Math.floorDiv(size, 2);
                invokeAll(new ParallelMergeSort(values, from, mid), new
ParallelMergeSort(values, mid + 1, to));
                merge(mid);
            }
        }
    }

    private void insertionSort() {
        for (int i = from + 1; i <= to; i++) {
            Student current = values.get(i);
            int j = i - 1;
            while (from <= j && current.getName().compareTo(values.get(j).getName())
< 0) {
                Student aux = values.get(j);
                values.remove(j + 1);
                values.add(j + 1, aux);
                j--;
            }
            values.remove(j + 1);
            values.add(j + 1, current);
        }
    }

    private void merge(int mid) {
        ArrayList<Student> left = new ArrayList<Student>();
        for (int i = from; i <= mid; i++) {
            left.add(values.get(i));
        }
        ArrayList<Student> right = new ArrayList<Student>();
        for (int i = mid + 1; i <= to; i++) {
            right.add(values.get(i));
        }

        int f = from;
        int li = 0;
        int ri = 0;
        while (li < left.size() && ri < right.size()) {
            if (left.get(li).getName().compareTo(right.get(ri).getName()) <= 0) {
                values.remove(f);
                values.add(f, left.get(li));
                f++;
                li++;
            } else {
                values.remove(f);
                values.add(f, right.get(ri));
                f++;
                ri++;
            }
        }
```



```
        while (li < left.size()) {
            values.remove(f);
            values.add(f, left.get(li));
            f++;
            li++;
        }

        while (ri < right.size()) {
            values.remove(f);
            values.add(f, right.get(ri));
            f++;
            ri++;
        }
    }

    @Override
    public void run() {
        System.out.println("Sorting thread started.");
        compute();
        MyUtils.semaphore.release();
    }
}
```

Producer class:



```
package com.schoolcatalog.app.concurrency;

public class Producer implements Runnable {

    private ArrayList<Student> students;

    public Producer(ArrayList<Student> students) {
        this.students = students;
    }

    @Override
    public void run() {
        try {
             MyUtils.semaphore.acquire();
            System.out.println("Producer thread started.");
            for (int i = 0; i < this.students.size(); i++) {
                System.out.println("put " + students.get(i).getName() + " into 
blocking queue");
                MyUtils.queue.put(students.get(i));
                Thread.sleep(TimeUnit.SECONDS.toMillis(1));
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

    }
}
```

Admin Class:

```
package com.schoolcatalog.api.models;

import org.springframework.data.mongodb.core.mapping.Document;

/**
 * Admin
 */
@Document(collection = "admins")
public class Admin extends Person {
    //CRUD operations for each type of user

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    public Account createAccount() {

        return null;
    }

    public Admin createAdmin() {
        return null;
    }

    public Teacher createTeacher() {

        return null;
    }

    public Student createStudent() {

        return null;
    }
}
```

ModelFactory interface:

```
package com.schoolcatalog.api.interfaces;


/**
 * DatabaseInterface
 */
public interface ModelFactory {

    Admin createAdmin(String name, String email, Account account);

    Teacher createTeacher(String name, String email, Account account, Subject
subject);

    Student createStudent(String name, String email, Account account, List<Subject>
subjects);

    Account createAccount(String username, String password, Role role);

    Account editAccount(String username, String password, Account currAccount);

    Admin editAdmin(String name, String email, Account account, Admin currAdmin);

    Teacher editTeacher(String name, String email, Account account, Subject subject,
Teacher currTeacher);

    Student editStudent(String name, String email, Account account, Student
currStudent);
}
```

Repository interfaces:

```
package com.schoolcatalog.api.repositories;

/**
 * AccountRepository
 */
@Repository
public interface AccountRepository extends MongoRepository<Account, String> {
    Account findByUsername(String username);
}
```

```
import org.springframework.stereotype.Repository;

/**
 * AdminRepository
 */
@Repository
public interface AdminRepository extends MongoRepository<Admin, String> {
    Admin findByAccount(Account account);
}
```

```
/**
 * RoleRepository
 */
@Repository
public interface RoleRepository extends MongoRepository<Role, String> {
    Role findByRole(String role);
}
```

```

/**
 * SubjectRepository
 */
@Repository
public interface SubjectRepository extends MongoRepository<Subject, String> {
    Subject findByName(String subject);
}
```

```
/**
 * TeacherRepository
 */
@Repository
public interface TeacherRepository extends MongoRepository<Teacher, String> {
    Teacher findByAccount(Account account);
}
```

ApiApplication class:

```
@SpringBootApplication
public class ApiApplication {

    public static void main(String[] args) {
        SpringApplication.run(ApiApplication.class, args);
    }

}
```

DemoApplication class:

```
@SpringBootApplication
@ComponentScan({"com.schoolcatalog.*"})
public class DemoApplication {

  @Bean
  RmiProxyFactoryBean rmiProxy() {
    RmiProxyFactoryBean bean = new RmiProxyFactoryBean();
    bean.setServiceInterface(ModelFactory.class);
    bean.setServiceUrl("rmi://localhost:1099/database");

    return bean;
  }

  public static void main(String[] args) {
    SpringApplication.run(DemoApplication.class, args);
  }
}
```

```
@SpringBootApplication
public class DemoApplication {

  @Bean
  RemoteExporter registerRMIExporter() {
    RmiServiceExporter exporter = new RmiServiceExporter();
    exporter.setServiceName("database");
    exporter.setServiceInterface(ModelFactory.class);
    exporter.setService(new ModelFactoryImplementation());

    return exporter;
  }

  public static void main(String[] args) {
    SpringApplication.run(DemoApplication.class, args);
  }
}
```

Testing:

student.feature:

```
Feature: student operations

  Scenario: student wants to go to student page
    When the student click the login button
    Then the student is at the "http://localhost:8080/student"
```

```
package com.schoolcatalog.app;


public class StudentStepDefinitions extends SpringIntegrationTest {
    String response;
    String url = DEFAULT_URL;

    @When("^the student click the login button")
    public void the_student_click_the_login_button() throws Throwable{

        response = url + "student";

    }

    @Then("^the student is at the \"([^\"]*)\"$")
    public void the_student_is_at_the_student_page(String s) throws Throwable{
        assertEquals(s, response);
    }



}
```

admin.feature:

```
Feature: admin operations

  Scenario: admin wants to go to students page
```

```
    When the admin click the students button
    Then the admin is at the "http://localhost:8080/admin/students"

  Scenario: admin wants to go to teachers page
    When the admin click the teachers button
    Then the admin is at the "http://localhost:8080/admin/teachers"

  Scenario: admin wants to go to admins page
    When the admin click the admins button
    Then the admin is at the "http://localhost:8080/admin/admins"
```

```
package com.schoolcatalog.app;

public class AdminStepDefinitions extends SpringIntegrationTest {
    String response;
    String url = DEFAULT_URL + "admin";

    @When("^the admin click the students button")
    public void the_admin_click_the_students_button() throws Throwable {
        // Write code here that turns the phrase above into concrete actions
        response = url + "/students";
        // throw new PendingException();
    }

    @When("^the admin click the teachers button")
    public void the_admin_click_the_teachers_button() throws Throwable {
        // Write code here that turns the phrase above into concrete actions
        response = url + "/teachers";
        // throw new PendingException();
    }

    @When("^the admin click the admins button")
    public void the_admin_click_the_admins_button() throws Throwable {
        // Write code here that turns the phrase above into concrete actions
        response = url + "/admins";
        // throw new PendingException();
    }


    @Then("^the admin is at the \"([^\"]*)\"$")  //
            public void the_admin_is_at_the_student_page(String string1) throws
Throwable {
        // Write code here that turns the phrase above into concrete actions
        // System.out.println(response);
        assertEquals(string1, response);
        // throw new PendingException();
    }
}
```

```
package com.schoolcatalog.app;
```

```
@SpringBootTest
@RunWith(Cucumber.class)
@CucumberOptions(format = { "json:target/REPORT_NAME.json", "pretty",
      "html:target/HTML_REPORT_NAME" }, features = { "src/test/java/resources/",   })
//@CucumberOptions(features = "src/test/resources")
public class DemoApplicationTests {

   //@Test
   //void contextLoads() {
   //}

}
```

TDD:

```
@RunWith(SpringRunner.class)
@SpringBootTest(classes = DemoApplication.class,
        webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
public class CatalogAppTDD {
    @Autowired
    private TestRestTemplate restTemplate;

    @LocalServerPort
    private int port;

    private String getRootUrl() {
        return "http://localhost/:" + port + "/api/v1";
    }

    @Test
    public void testCreateAdmin() {
        Admin admin = new Admin();
        Role role = new Role();
        role.setRole("ADMIN");
        Account adminAccount = AdminController.createAccount("admin2", "pass", role);
        admin.setName("Paul");
        admin.setEmail("admin@yahoo.com");
        admin.setAccount(adminAccount);

        ResponseEntity<Admin> postResponse = restTemplate.postForEntity(getRootUrl()
+ "/admins", admin, Admin.class);
        Assert.assertNotNull(postResponse);
        Assert.assertNotNull(postResponse.getBody());
    }

    @Test
    public void testGetAdmin() {
        Admin admin = restTemplate.getForObject(getRootUrl() + "/admins",
Admin.class);
        System.out.println(admin.getName());
        Assert.assertNotNull(admin);
    }

}
```

4. References

- Engineering Software as a Service: An Agile Approach Using Cloud Computing by Fox, Armando, Patterson, David
- https://spring.io/docs
- https://spring.io/projects/spring-data-mongodb
- https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html
- https://www.baeldung.com/thymeleaf-in-spring-mvc
- https://www.baeldung.com/spring-thymeleaf-path-variables
- https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/bind/annotation/RequestParam.html
- https://springframework.guru/spring-requestmapping-annotation/
- Design Patterns: Elements of Reusable Object-Oriented Software by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
- https://en.wikipedia.org/wiki/Behavior-driven_development
- https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/testing.html