

Student web service application

Nikola Milanovic

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 software delivery lifecycle of the processes. This document supply detail 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, MySQL workbench and git. This application can be found here: <https://bitbucket.org/nimilanovi/nikola-milanovic-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. Application has SchemaGenerator file which drop and create tables with relations.

Credentials for user can be found in this document.

2. General Description

2.1 Product functions

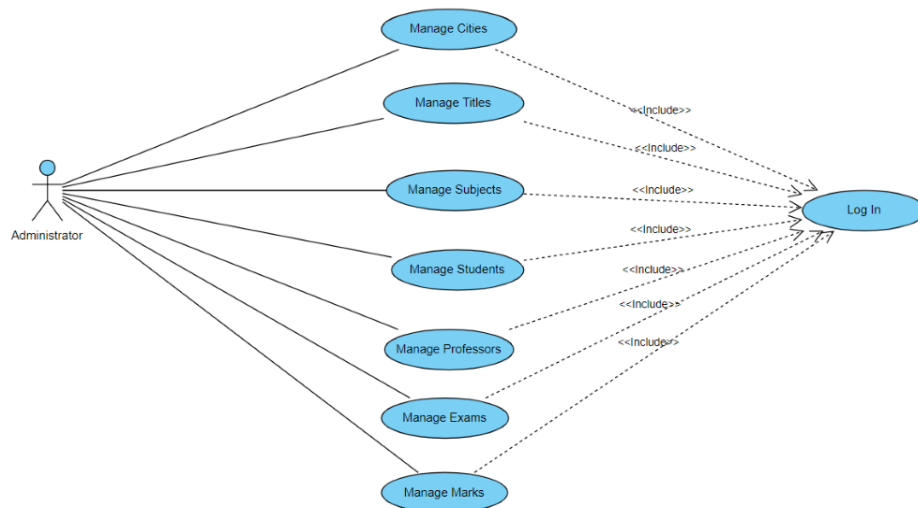
Administrator have full access to project. He can view all students, professors, cities, titles, exams, subject and marks with custom amount per page. Administrator can add, edit or remove everything. Administrator can set subjects to students and professors. Also, he can set exam to students, and add mark of exam to student.

2.2 User characteristics

There is only one type of user and that is administrator. His parameters for login are: username-admin, password-1111. All his functions are named in the previous section.

2.4 Diagram with the explanation

One the following picture we will show you use case diagram.



Use case diagram describes the privileges that an administrator has. List of manage privileges and explanation:

Cities

When administrator is logged in and he had entered cities page, he gets cities list and the following options: list all cities, create a new city, edit city, and remove city. Also, he can change his overview of the list with pagination.

Titles

When administrator is logged in and he had entered titles page, he gets titles list and the following options: list all titles, create a new title, edit title, and remove title. Also, he can change his overview of the list with pagination.

Subjects

When administrator is logged in and he had entered subjects page, he gets subjects list and the following options: list all subjects, create a new subject, edit subject, and remove subject. Also, he can change his overview of the list with pagination.

Students

When administrator is logged in and he had entered students page, he gets students list and the following options: list of all students, set exam for student, set subject for student, create a new student, edit student, and remove student. Also, he can change his overview of the list with pagination. He can also go to the individual student and get more information about that student.

Professors

When administrator is logged in and he had entered professors page, he gets professors list and the following options: list all professors, set subject for professor, create a new professor, edit professor, and remove professor. Also, he can change his overview of the list with pagination. He can also go to the individual professor and get more information about that professor.

Exams

When administrator is logged in and he had entered exams page, he gets exams list and the following options: list all exams, create a new exam, edit exam, and remove exam. Also, he can change his overview of the list with pagination.

Marks

When administrator is logged in and he had entered marks page, he gets marks list and the following options: list all marks, create a new mark, edit mark, and remove mark. Also, he can change his overview of the list with pagination.

3. User viewpoint

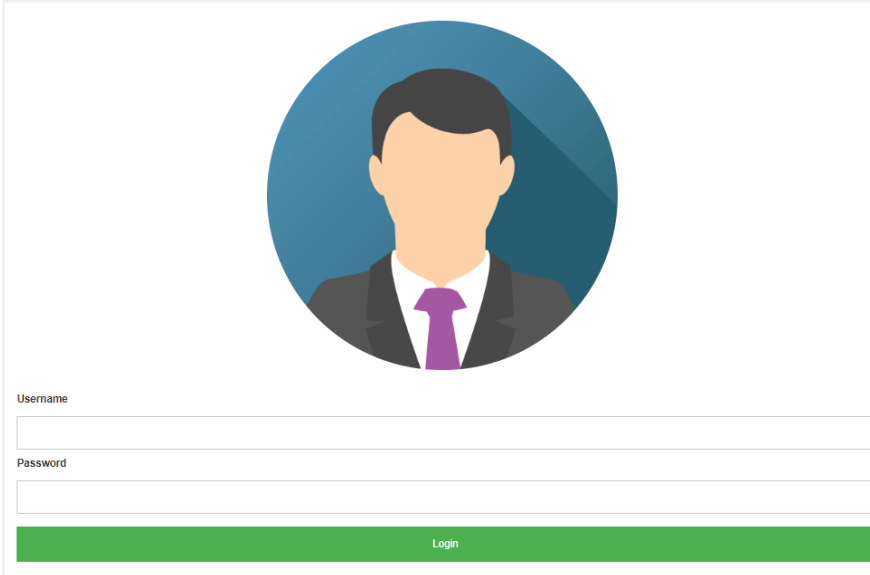
3.1 Interface

When administrator visits web site, he/she must log in with his credentials to access the site.

Credentials:

Username-admin

Password-1111

A user login interface. At the top center is a circular profile picture of a man with dark hair, wearing a dark suit, white shirt, and a purple tie. Below the profile picture are two input fields. The first is labeled 'Username' and the second is labeled 'Password'. Below these fields is a green button with the text 'Login' in white.

Username

Password

Login

When administrator is logged he has following choices:

If the administrator selects exams, list of exams will be displayed.

2

submit

ID	Date	Professor	Subject	Edit	Delete
1	2020-04-07	Sone Sone	Java	Edit exam	Delete exam
2	2020-04-05	Pera Markovic	Laravel	Edit exam	Delete exam

1 2
[Create exam](#)
[Back to index](#)

Administrator can then create new exam, remove exam or edit exam.

If administrator select choice to create new exam, he/she is introduced with following interface:

Date:

04/15/2020

Subject and Professor:

SQL - Profa Profic

SQL - Profa Profic

Laravel - Pera Markovic

Insert exam

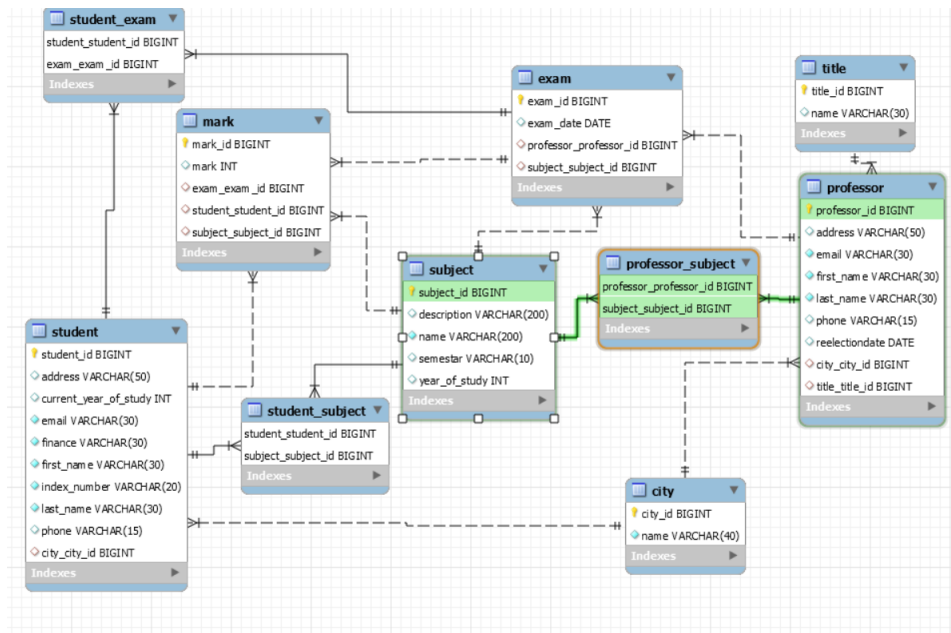
There he can choose subject and professor and apply date for the exam. After click insert, he will be redirect to page with all exams.

4. Database

4.1 Database connection and environment

For this application we used MySQL workbench. This application also creates database tables and relations between them, you just need to setup it. Instructions to setup database: First thing we need to do is open the MySQL workbench and create new schema. We will name it student_project. User that we will use has user name root, and password root. We are connected via the local host on port 3306 by default. We are generating tables from java classes using hibernate.

4.2 Database logical model



4.3 Java class

Example of java class that is used as entity for the database:

```
@Entity
@Table(name="student")
public class Student implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="student_id", nullable = false)
    private long studentId;

    @Column(name="index_number", unique=true, nullable = false, length=20)
    @Size(min = 10, message= "Index must contain 10 chars.")
    private String indexNumber;

    @Column(length=50)
    @Size(min = 3, max = 40, message = "Address must be between 3 and 50 characters")
    private String address;

    @Column(name="current_year_of_study", precision=7)
    @NotNull
    private int currentYearOfStudy;

    @Column(length=30, nullable=false, unique = true)
    @Email(message= "Not email format")
    private String email;

    @Column(nullable=false, length=30)
    private String finance;

    @Column(name="first_name", nullable=false, length=30)
    @Size(min=3, max=30, message="Name must be between 3 and 30 characters")
    private String firstName;
```

@Entity annotation means this class is an entity in our database.

@Table annotation describes the name of this entity in our database.

@Id annotation describes that property is primary key of the table.

@GeneratedValue annotation means that primary key will be automatically assigned.

@Column annotation describes attributes of columns in database.

@Size annotation describes validation of that property.


```

private String lastName;

@Column(length=15)
@Size(min=6, max=15, message="Phone must be between 6 and 15 characters")
private String phone;

@OneToMany(mappedBy="student", cascade= CascadeType.ALL)
private Set<Mark> marks;

@ManyToOne(fetch=FetchType.LAZY)
private City city;

@ManyToMany
@JoinTable(
    name="student_exam"
    , joinColumns={
        @JoinColumn(name="student_student_id")
    }
    , inverseJoinColumns={
        @JoinColumn(name="exam_exam_id")
    }
)
private Set<Exam> exams;

@ManyToMany
@JoinTable(
    name="student_subject"
    , joinColumns={
        @JoinColumn(name="student_student_id")
    }
    , inverseJoinColumns={
        @JoinColumn(name="subject_subject_id")
    }
)
private Set<Subject> subjects;

```

First relation is one-to-many between the student and marks. It is one-to-many which means that many marks belong to one student.

Second relation is many-to-one between the student and city. It is many-to-one which means that many students are stored in one city (Student have one city).

Third and other relations are many-to-many. It is many-to-many which means that students can have many exams and many exams can have many students. It is same for subjects.

5. Technologies and functions

Back-end of this application is based on Java. Spring framework is used for application back-end. For front-end we used JSP. Apache Tomcat application server is used for running application. Hibernate is used for map Java classes to database tables. For tracking changes in source code during software development we used GIT.

5.1 Application configuration

Migration of application hosting requires minimal modifications. Application use Maven build tool, so all dependencies can be found in the pom.xml. Application configuration is based on XML files. Classes are organized in packages. Database connection is located in a server context.

5.2 Application security

This application is secured with FilterChainProxy. Security configuration file can be found in config package. There is only one user which is created in xml file. Just authorized user can use this application. IF unauthorized user tries to reach any page of application, he will be redirected to login form.

5.3 Entities

As we said, Hibernate is used for mapping. Every table have corresponding Entity class which is located in entities package. Every class in entities package describe table in database. Every property in the class corresponding a column in the table. In addition to its own API, Hibernate also implements the Java Persistence API specification. Every class contains annotations. Description of that is in database section. In a database package can be found a SchemaGenerator.java file which create tables in the database.

5.4 Logger

This application has enabled aop(Aspect-Oriented-Programming) logger. All transactions with the database are tracked and printed in application console. Every DAO class is tracked.

5.5 DAO and Services

Every entity has corresponding DAO and Service classes. That classes implements corresponding interfaces. DAO have methods for all operations (CRUD and others) which we need for specific entity. DAO classes are annotated with `@Repository`, 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 query is executed. Every method in Service classes is annotated with `Transactional`. We are using services to forward logic from DAO to Controllers.

```
@Repository
public class StudentDAO implements CrudDAOInterface<Student>, StudentRelationsDAOInterface<Subject, Exam> {

    private SessionFactory sessionFactory;

    @Autowired
    public StudentDAO(SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;
    }

    @Override
    public List<Student> getAll() {
        Session session = sessionFactory.getCurrentSession();
        Query<Student> query = session.createQuery("from Student", Student.class);
        return query.getResultList();
    }

    @Override
    public Student getOneById(long id) {
        Session session = sessionFactory.getCurrentSession();
        Student student = session.get(Student.class, id);
        return student;
    }

    @Override
    public void add(Student s) {
        Session session = sessionFactory.getCurrentSession();
        session.saveOrUpdate(s);
    }
}
```

```
import java.util.List;

@Service("studentService")
public class StudentService implements CrudServiceInterface<Student>, StudentRelationsServiceInterface<Subject, Exam> {

    private StudentDAO studentDao;

    @Autowired
    public void setStudentDao(StudentDAO studentDao) {
        this.studentDao = studentDao;
    }

    @Transactional
    @Override
    public List<Student> getAll() {
        return studentDao.getAll();
    }

    @Transactional
    @Override
    public Student getOneById(long id) {
        return studentDao.getOneById(id);
    }

    @Transactional
    @Override
    public void create(Student s) {
        this.studentDao.add(s);
    }
}
```

5.6 Controllers

All controllers can be found in controller package. Each entity has controller. Controller is a collection of API endpoints for data manipulation or collecting data. Each controller has a several methods for CRUD and other operations for his entity. Service forwards DAO logic to Controllers.

```
@Controller
@RequestMapping("/students")
public class StudentController {

    @Autowired
    private StudentService studentService;
    @Autowired
    private CityService cityService;
    @Autowired
    private SubjectService subjectService;
    @Autowired
    private ExamService examService;

    private Set<Subject> subjectCart;
    private Set<Exam> examCart;

    @RequestMapping("")
    public String allStudents(Model model) {
        model.addAttribute("students", studentService.getAll());
        return "students";
    }

    @RequestMapping("/{page}/{num}")
    public String paginatedStudents(@PathVariable("page") int page, @PathVariable("num") int num, Model model) {
        model.addAttribute("pages", Math.ceil((double)studentService.getAll().size()/num));
        model.addAttribute("num", num);
        model.addAttribute("students", studentService.getPaginated(page, num));
        return "students";
    }

    @RequestMapping("/{id}")
    public String oneStudent(@PathVariable("id") long id, Model model) {
        model.addAttribute("student", studentService.getOneById(id));
        model.addAttribute("subjects", studentService.allSubjects(id));
        model.addAttribute("exam", studentService.getExam(id));
        return "students";
    }
}
```

Every controller is annotated as `@Controller` and have `@RequestMapping` annotation so we can know which URL we are targeting to reach controllers methods. There are several mappings: `@RequestMapping`, `@PostMapping`, `@PutMapping` etc...