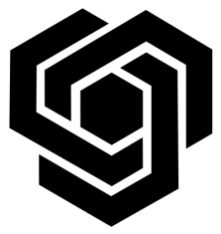
**ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ**



**Курсов проект**

Дисциплина*: Съвременни Java технологии*

Тема: *Изготвяне на електронна система от тип ученически дневник.*

Изготвил:

Христо Ботушанов, 222гр., факултетен №: 121319046

Contents

[Използвани технологии 3](#_Toc55482803)

[Spring Framework 3](#_Toc55482804)

[Spring Security 3](#_Toc55482805)

[Hibernate 4](#_Toc55482806)

[Thymeleaf 5](#_Toc55482807)

[HTML 6](#_Toc55482808)

[Цел на проекта 7](#_Toc55482809)

[Entity Relation Model 7](#_Toc55482810)

[Описание на използван код от системата 8](#_Toc55482811)

[Създаване на нов потребител 8](#_Toc55482812)

[Търсене на ученик в базата 9](#_Toc55482813)

[Валидиране на ученици 9](#_Toc55482814)

[Източници 11](#_Toc55482815)

# Използвани технологии

## Spring Framework



Spring Framework е фреймъурк за приложения и контейнер за инверсия на контрол за платформата Java. Основните функции на рамката могат да се използват от всяко приложение на Java, но има разширения за изграждане на уеб приложения върху платформата Java EE (Enterprise Edition). Въпреки че рамката не налага някакъв специфичен модел на програмиране, тя стана популярна в Java общността като допълнение към модела Enterprise JavaBeans (EJB). Spring Framework е с отворен код.

## Spring Security



Spring Security е фреймуърк, който позволява на програмиста да наложи ограничения на сигурността на базирани на Spring-framework уеб приложения чрез JEE компоненти. Накратко, това е библиотека, която може да се използва, разширена за персонализиране според нуждите на програмиста. Тъй като е член на едно и също семейство Spring, той върви гладко ръка за ръка с Spring Web MVC. Неговата основна област на действие е да обработва удостоверяването и оторизацията на ниво уеб заявка, както и нивото на извикване на метода. Може би. Най-голямото предимство на тази рамка е, че тя е мощна, но много адаптивна при изпълнението си. Въпреки че следва конвенцията на Spring за конфигурацията, програмистите могат да избират между разпоредби по подразбиране или да я персонализират според своите нужди.

## Hibernate



Hibernate ORM (или просто Hibernate) е обектно-релационен инструмент за картографиране за програмния език Java. Той осигурява рамка за картографиране на обектно-ориентиран модел на домейн към релационна база данни. Hibernate се справя с обективно-релационните проблеми с несъответствието на импеданса, като замества директните, постоянни достъпи до база данни с функции за обработка на обекти на високо ниво.

Hibernate е безплатен софтуер, който се разпространява под GNU Lesser General Public License 2.1.

Основната характеристика на Hibernate е картографиране от класове Java в таблици на бази данни и картографиране от типове данни Java на типове данни SQL. Hibernate също така предоставя възможности за заявки и извличане на данни. Той генерира SQL извиквания и освобождава разработчика от ръчното боравене и преобразуването на обекти на набора от резултати.

## Thymeleaf



Thymeleaf е Java XML / XHTML / HTML5 механизъм за шаблони, който може да работи както в уеб (базиран на сървлети), така и в не-уеб среди. Той е по-подходящ за обслужване на XHTML / HTML5 на изгледния слой на базирани на MVC уеб приложения, но може да обработва всеки XML файл, дори в офлайн среда. Той осигурява пълна интеграция на Spring Framework.В уеб приложенията Thymeleaf се стреми да бъде пълен заместител на JavaServer Pages (JSP) и реализира концепцията на Natural Templates: шаблони, които могат да се отварят директно в браузърите и които все още се показват правилно като уеб страници.

Thymeleaf е софтуер с отворен код, лицензиран под лиценза Apache 2.0.

## HTML



HTML (HTML) е стандартният език за маркиране на документи, предназначени за показване в уеб браузър. Той може да бъде подпомогна от технологии като Cascading Style Sheets (CSS) и скриптови езици като JavaScript.

Уеб браузърите получават HTML документи от уеб сървър или от локално хранилище и визуализират документите в мултимедийни уеб страници. HTML описва семантично структурата на уеб страница и първоначално включва реплики за външния вид на документа.

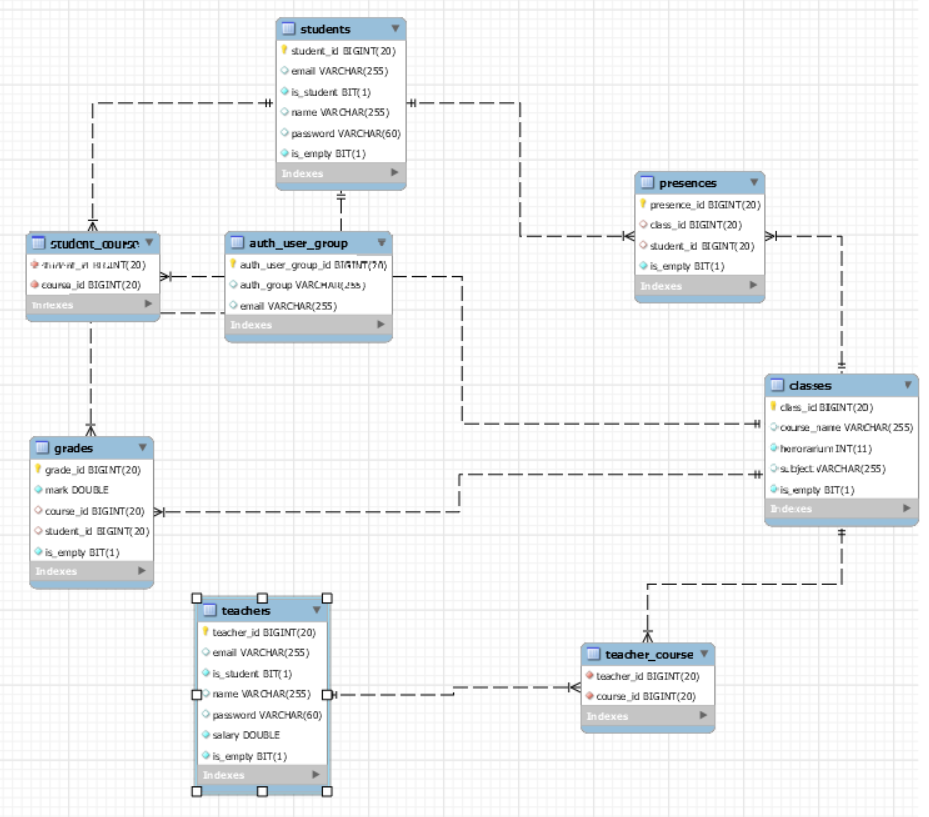
HTML елементите са градивните елементи на HTML страниците. С HTML конструкции изображения и други обекти като интерактивни форми могат да бъдат вградени в изобразената страница. HTML предоставя средство за създаване на структурирани документи, като обозначава структурна семантика за текст като заглавия, абзаци, списъци, връзки, цитати и други елементи. HTML елементите са очертани с тагове, написани с помощта на ъглови скоби. Етикети като <img /> и <input /> директно въвеждат съдържание в страницата. Други маркери като <p> обграждат и предоставят информация за текста на документа и могат да включват други тагове като поделементи. Браузърите не показват HTML маркерите, но ги използват, за да интерпретират съдържанието на страницата.HTML може да вгражда програми, написани на скриптов език като JavaScript, което влияе върху поведението и съдържанието на уеб страниците. Включването на CSS определя външния вид и оформлението на съдържанието. Консорциумът на World Wide Web (W3C), бивш поддържащ HTML и настоящ поддържащ CSS стандартите, насърчава използването на CSS върху изричен презентационен HTML от 1997 г.

# Цел на проекта

Целта на този проект е изграждането на електронна система, която да замени хартиения училищен дневник с уеб сайт. В тази система ученици ще могат да се записват за предмети, които изучават. Могат да видят кои учители преподават по тези предмети. Могат да видят оценките си по предметите.

Учителите от своя страна могат да нанасят оценки на учениците и да следят техните присъвствия.

# Entity Relation Model



# Описание на използван код от системата

## Създаване на нов потребител

public ServiceReturnResult createUser(ViewModel viewModel) {

Създаваме обекта който ще държи Entity-то, по което ще създадем новия потребител

        User resultUser = new User();

В този обект ще вкючим резултата от създаването на новия потребител

        ServiceReturnResult userCreateResult = new ServiceReturnResult();

Тази променлива ще съдържа в себе си дали новия потребител трябва да е ученик или учител

        boolean isStudent;

Проверяваме какъв тип нов потребител трябва да направим

        isStudent = UserEntityHelper.isUserStudent(viewModel);

Избираме типа валидация спрямо типа на потребителя

        VMValidator validator = isStudent ? new StudentVMValidator() : new TeacherVMValidator();

Създаваме обект от тип декоратор, който ще държи ViewModel-a и после ще го валидира

        ModelDecorator decorator = new ModelDecorator(viewModel);

Валидираме ViewModel-a

        userCreateResult.addErrorMsg(decorator.validateModel(validator));

Проверяваме дали информацията е валидна, ако не е връщаме грешка

        if (!userCreateResult.isSuccessful()) {

            return userCreateResult;

        }

Инициализираме нов обект от тип резултат, който ще върне информация от factory класа

        ServiceReturnResult factoReturnResult = new ServiceReturnResult();

В зависимост дали новия потребител трябва да е учител или ученик използваме правилния factory клас за създаването на ново Entity

        factoReturnResult = isStudent ? studentFactory.getEntity(viewModel) : teacherFactory.getEntity(viewModel);

Проверяваме дали са възникнали грешки по време на създаването на entity-то, ако не е било успешно, връщаме грешка

        if (!factoReturnResult.isSuccessful()) {

            return factoReturnResult;

        }

Взимаме Entity обекта от factory резултата

        resultUser = (User) factoReturnResult.getReturnResultObject();

Използване на вътрешен метод за добавяне на потребител в базата

        saveUser(resultUser);

След успешно записване на потребител, използваме сървиса authGroupservice, за да му добавим роля в базата с данни

        authGroupService.addAuth(viewModel);

        userCreateResult.setReturnResultObject(resultUser);

връщаме резултата

        return userCreateResult;

    }

## Търсене на ученик в базата

    @Override

    public ServiceReturnResult findStudentEntityById(String id) {

Декларираме променливите, които ще поемат резултатите от функциите

        Long longId;

        Optional<Student> foundStudentOptional;

        StudentViewModel studentViewModel;

        ServiceReturnResult convertLongResult;

Инициализираме резултата, който ще върнем

        this.returnResult = new ServiceReturnResult();

Парсваме индентификатора към число

        convertLongResult = NumberHandler.parseStringToLong(id);

Проверяваме дали са изникнали грешки при парсването

        if (!convertLongResult.isSuccessful()) {

            return convertLongResult;

        }

Взимаме стойността от резултата

        longId = (Long) convertLongResult.getReturnResultObject();

Извикваме findId метода от базата данни

        foundStudentOptional = repository.findById(longId);

        if (foundStudentOptional.isEmpty()) {

            this.returnResult.addErrorMsg("Student not found");

            return returnResult;

        }

        this.returnResult.setReturnResultObject(foundStudentOptional.get());

        return this.returnResult;

    }

## Валидиране на ученици

Наследява се базовия валидатор клас, на който ще използваполетата за да валидиира ViewMode-a

public class StudentVMValidator extends VMValidatorBase {

    private StudentViewModel student;

    public StudentVMValidator () {

        super();

    }

    @Override

    public List<String> validate() {

Инициализираме масив с текстови полете, в който под формата на текст ще вкарваме грешките получили се при валидацията

        List<String> result = new ArrayList<String>();

Проверяваме дали обекта е бил зареден

        if (student == null) {

            result.add("Student validator not initialized");

            return result;

        }

Валидираме полетата на ViewModel-a

        result.addAll(emailValidator.validateEmail(student.getEmail()));

        result.addAll(stringValidator.validateString(student.getName(), "The student's name", 50));

        result.addAll(stringValidator.validateString(student.getPassword(), "Student's passowrd", null));

Връщаме резултата

        return result;

    }

С този метод инициализираме обекта като му подаваме ViewModel

    @Override

    public boolean injectModel(ViewModel model) {

Проверяваме дали модела е реален

        if (model == null || model.isEmpty()) {

            return false;

        }

Прикрепяме ViewModel-a към обекта от този клас

        student = (StudentViewModel) model;

        return true;

    }

}

# Източници

1. <https://www.thymeleaf.org/>
2. <https://spring.io/projects/spring-security>
3. <https://spring.io/>
4. https://www.w3schools.com/html/
5. Mak, Gary (September 1, 2010). Spring Recipes: A Problem-Solution Approach (Second ed.). Apress. p. 1104. ISBN 1-4302-2499-1.
6. Walls, Craig (November 28, 2010). Spring in Action (Third ed.). Manning. p. 700. ISBN 1-935182-35-8.
7. Walls, Craig; Breidenbach, Ryan (August 16, 2007). Spring in Action (Second ed.). Manning. p. 650. ISBN 1-933988-13-4.