

ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ
ФАКУЛТЕТ ПО КОМПЮТЪРНИ СИСТЕМИ И ТЕХНОЛОГИИ

Катедра “Компютърни системи”

Дипломна работа

Тема: Софтуерна система за разпределяне и съставяне на дипломни задания

Изготвил: Стефан Николов Петров
Факултет: КСТ
Фак. №: 121314032
Проверил: гл. ас. д-р Антония Ташева

Съдържание

Увод	4
Глава 1. Анализ на подобни системи. Цели и задачи.....	5
1.1 Какво представлява процесът по избор на дипломно задание за момента	5
1.2 Цел и задачи	5
Глава 2. Обзор на използваните технологии.....	8
2.1 ASP.NET MVC	8
2.1.1 ASP.NET MVC.....	9
2.1.2 Изглед(View).....	9
2.1.3 Модел (Model)	10
2.1.4 Контролер (Controller)	10
2.2 C#.....	11
2.3 SQL - база данни.....	12
2.3.1 Релационна база данни	12
2.3.2 Отношение	12
2.3.3 Нормализация на базата данни	13
2.4 jQuery	15
2.5 CSS	16
Глава 3. Проектиране на системата.	17
3.1 Съставяне на моделите за базата данни	17
3.2 Съставяне на хранилище за елементите на базата (repository)	19
3.3 Създаване на слой с услуги (Service layer pattern).....	20
3.4 Създаване на подходящи контролери и изгледи	21
3.4.1 Регистрация на потребители	22
3.4.2 Избор на роли в приложението	22

Глава 4. Програмна реализация.....	26
4.1 Създаване на базата данни.....	27
4.2 Създаване на хранилище.....	31
4.3 Създаване на услуги (services).....	32
4.4 Autofac – предназначение и настройки	33
4.5 Създаване на контролери.....	34
4.6 Създаване на изгледи	34
4.7 Automapper – настройки и приложение.....	39
4.8 Генериране на Docx файлове.....	40
Глава 5. Ръководство на потребителя.....	43
5.1 Ръководство за потребител тип студент.....	45
5.2 Ръководство за потребител тип ръководител	47
5.3 Ръководство за потребител тип администратор	51
Заклучение	54
Списък на използвани литературни източници	55
Приложения	56
1. Модели	56
1.1 DDS.Data.Common.....	56
1.2 DDS.Data.Models.....	59
1.3 DDS.Data.....	62
Глава 2. Услуги	64
2.1 DDS.Services.Data.Interfaces	64
2.2 DDS.Services.Data	66
2.3 DDS.Services.Web	73
Глава 3. Основна част на приложението	74

3.1	DDS.Common	74
3.2	DDS.Web.....	75
3.2.1	App_Start	76
3.2.2	Area – Administration	82
3.2.3	ViewModels	90
3.2.4	Views.....	91
3.2.5	Controllers	101

Увод

Намирането на подходяща диплома е важна стъпка от етапа на обучение. Когато студентът намери желаното от него дипломно задание, изработването на дипломната работа става много по-лесно и приятно.

В настоящата дипломна работа ще се разгледа система за създаване и разпределение на дипломни задания. Системата е предназначена както за студенти, така и за ръководители. Позволява лесен избор на дипломни работи измежду задания предложени от всички преподаватели, както и следене на процеса на утвърждение на темата. Улеснява процеса по създаване и редактиране на дипломни задания. Има изграден интерфейс за преглед на дипломни задания и тяхното одобрение от ръководител. Позволява намирането на дубликати чрез използването на алгоритъм за сравнение на заглавията.

В системата има изградено администриране на потребители. Позволява промяна на ролите на даден потребител.

Текущата дипломна работа е изградена следвайки начина и методите на избор и одобрение на заданията в Технически университет – София, катедра „Компютърни системи“. За момента системата поддържа само една катедра, но инфраструктурата позволява лесна интеграция на логика за всички катедри, в случай че приложението трябва да се разшири.

Глава 1. Анализ на подобни системи. Цели и задачи.

Изборът на дипломно задание е важна и отговорна задача. От този избор до голяма степен зависи и крайният резултат на дипломанта. В този момент, процесът на избиране на дипломно задание в Технически университет – София, не е много сложен, но е сравнително „муден“. За да може един студент да намери подходяща тема, той трябва да търси преподаватели лично, което е обвързано понякога с пътуване до сградите на университета и търсене на кабинети. Като се има в предвид, че една голяма част от студентите трябва и да работят, то това спестено време би било полезно в техния професионален живот.

1.1 Какво представлява процесът по избор на дипломно задание за момента.

За избор на дипломно задание, студентът първо трябва да намери преподавател в сферата, в която е избраната тема. Повечето ръководители имат списъци с дипломни задания на хартиен носител, от които студентът да избира. В случай, че преподавателят няма търсеното задание налично, студентът трябва да намери друго подходящо задание или друг преподавател. Този кръговрат на търсене води до загуба на време не само на студента, но и на преподавателите, предразполага за чакане пред кабинета на ръководителя и запълване на приемното му време.

След като студентът намери желана от него тема (или подобна), тя трябва да бъде доставена в канцеларията на факултета за проверка на студента, дали има невзети изпити и след това дипломата бива изпратена на технически ръководител за подпис от страна на декана.

Този процес на проверки и одобрения значително забавя избора на задание. За да се ускорят нещата е необходима автоматизация.

1.2 Цел и задачи

Света, в който живеем непрекъснато се стреми към подобряване на условията за труд. Използването на нови технологии позволява автоматизация на процесите и увеличава, количеството и качеството на извършената работа. Една от основните цели на настоящата дипломна работа е да автоматизира процеса по избиране и одобряване на дипломни задания.

Използването на web приложение събира всички дипломи на едно място. Създаването на общо сборище за дипломни задания позволява лесно търсене и дава възможност за избор на тема от богат списък. Web приложението позволява лесен достъп до всички теми от разстояние, което значително намалява стоенето пред кабинетите и търсенето на преподаватели в приемното им време.

Задачите, поставени пред разработката на такова приложение, са:

- всички дипломни задания да са на едно място;
- да се даде възможност за търсене по теми, категории или преподавател;
- да се улесни съставянето на задания;
- да се ускори модифицирането на задания;
- възможност за намиране на дубликати или вече използвани теми през други години;
- лесно управление на потребителите.

Събирането на всички дипломни задания на едно място позволява цялостно преглеждане на списъка от студента. Това увеличава възможността за намиране на подходяща тема на дипломно задание в случай, че студентът не се е спрял на определена тема. От друга страна, когато студентът знае каква тема търси, той може да намери подходящо дипломно задание чрез използване на търсачка, която да търси из цялата база. Търсенето по категории спомага за отсяването само на задания с желана технология. По този начин студентът може да намери дипломно задание, което е в неговата сфера на интерес. От друга страна, ако студентът предпочита дипломно задание при някой ръководител, той ще може да избере своето задание от негов списък.

Когато съставените заданията са дигитални, те могат да бъдат бързо променяни. Тъй като повечето ръководители в университетите в София разполагат само със списъци от дипломни задания на хартиен носител, то тази система би позволила да променят детайли по някое задание по всяко време и без много сили. За разлика от повечето списъци от задания, в такава автоматизирана система може да се намери по-цялостна информация за заданието. В системата трябва да има категории, които да представляват сферата на дипломата. Придружено от категориите може да има и кратко описание, което да даде по-ясен поглед върху самата тема.

Една такава система за съставяне и управление на дипломни задания трябва да има модули за администриране и управление на заданията. Управлението на толкова дипломни

задания е трудоемка задача сама по себе си. Това е един от най-големите плюсове на една такава автоматизирана система, която ще може да сравнява заданията с останалите вече избрани такива. При намиране на дубликати трябва да показва на администратора съобщение, по което той да определи дали дадена дипломна работа трябва да бъде одобрена или трябва да се върне за преразглеждане.

Глава 2. Обзор на използваните технологии.

2.1 ASP.NET MVC



ASP.NET

ASP.NET е новото поколение уеб рамка (framework), разработена от Microsoft. За първи път е публикуван през януари 2002 година с версия 1.0 на .NET Framework, и е най-успешен наследник на Microsoft Active Server Pages (ASP) технология, но да не се бърка, че не е подобрена версия на ASP. ASP.NET е изградена въз основа на CommonLanguage Runtime (CLR), което позволява на

програмистите да пишат ASP.NET код като използват .NET език по избор.

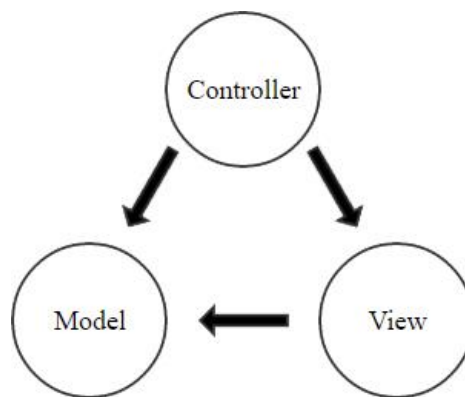
ASP.NET цели производителност спрямо останалите скрипт-базирани технологии (включително класическия ASP) като компилира сървърно кода в един или повече DLL файлове на Уеб сървър. Тази компилация става автоматично, когато страницата бива заредена за пръв път (което от своя страна означава, че програмистът не трябва да изпълнява отделни компилации за страниците). Тази характеристика осигурява лекота на разработване, предлагана от скриптовите езици с производителността на бинарните операции. Трябва да се има предвид обаче, че самата компилация може да причини забележимо забавяне при потребителя когато редактираната страница бива изискана за пръв път от Уеб сървър, но това забавяне не би се появило отново преди следваща промяна.

ASP.NET предлага три рамки за създаване на уеб приложения: WebForms, ASP.NET MVC, и ASP.NET Web Pages. И трите рамки са стабилни и добре развити, и с всяка от тях могат да се създадат големи уеб приложения. Всички рамки предоставят навсякъде предимствата и характеристиките на ASP.NET.

2.1.1 ASP.NET MVC



ASP.NET MVC е платформа, създадена от Microsoft, която служи за изработване на уеб приложения, използвайки модела **Model-View-Controller** (MVC). Платформата използва C#, HTML, CSS, JavaScript, XML и бази данни. ASP.NET MVC е съвременно средство за изграждане на уеб приложения, което не замества изцяло уеб формите. Платформата включва нови тенденции в разработката на уеб приложения, притежава много добър контрол върху HTML и дава възможност за създаване на всякакви приложения. ASP.NET MVC може да бъде много лесно тествана и допълвана, защото е изградена от отделни модули, които са изцяло независими едни от други. Чрез платформата се създават цялостни приложения, които се стартират, а не единични скриптове (като при PHP например).



Фигура 2-1 Схема на Модел-Изглед-Контролер

2.1.2 Изглед(View)

Изгледите са тези, които определят как ще бъде визуализиран потребителският интерфейс (UI) на приложението. В ASP.NET MVC се поддържат средства (engines) за генериране на изгледи.

Когато потребител взаимодейства с изглед, данните се предават от изгледа до метод за действие, който от своя страна може да създаде друг изглед. Едно MVC приложение може да има няколко контролери, всеки от които може да съдържа множество методи за действие, а всяко действие може да създаде различен изглед. Изгледите са организирани в папки, като името им се определя от това на свързания контролер.

2.1.3 Модел (Model)

Моделът представлява част от приложението, което реализира домейн логиката, също известна като бизнес логика. Домейн логиката обработва данните, които се предават между базата данни и потребителския интерфейс. Например, в една система за инвентаризация, моделът отговаря за това дали елемент от склада е наличен. Моделът може да бъде част от заявлението, което актуализира базата данни, когато даден елемент е продаден или доставен в склада. Често моделът съхранява и извлича официална информация в базата данни.

Изглед модел (ViewModel)

Изглед модела позволява да се оформят няколко изгледа от един или повече модела от данни или източници в един обект. Този модел е оптимизиран за потребление и изпълнение.

2.1.4 Контролер (Controller)

Контролери са класове, които се създават в MVC приложението. Намират се в папка Controllers. Всеки един клас, който е от този тип, трябва да има име, завършващо с наставка "Controller". Контролерите обработват постъпващите заявки, въведени от потребителя и изпълняват подходящата логика за изпълнение на приложението. Класът контролер е отговорен за следните етапи на обработка:

- Намиране и извикване на най-подходящия метод за действие (action method) и валидиране, че може да бъде извикан;
- Взимане на стойности, които да се използват като аргументи в метода за действие;
- Отстраняване на всички грешки, които могат да възникнат по време на изпълнението метода за действие;
- Осигуряване на клас WebFormViewEngine по подразбиране за отваряне на страници с изглед от тип ASP.NET.

Контролерът е клас, който се наследява от базовия клас System.Web.Mvc.Controller. Всеки публичен метод в контролера е показан като "controlleraction". Ако искаме даден метод да не бъде извикан, трябва да сложим "NonAction" атрибут върху неговото име. По подразбиране "Index()" действието е извикано за контролера, когато друго такова не е изрично упоменато.

2.2 C#



C# е език от високо ниво, който прилича на Java и C++ и донякъде на езици като Delphi, VB.NET и C. Всички C# програми са обектно-ориентирани. Те представляват съвкупност от дефиниции на класове, които съдържат в себе си методи, а в методите е разположена програмната логика и инструкциите, които компютърът изпълнява. В днешно време C# е един от най-популярните езици за програмиране. На него пишат милиони разработчици по цял свят. Тъй като C# е разработен от Microsoft като част от тяхната съвременна платформа за разработка и изпълнение на приложения .NET Framework, езикът е силно разпространен сред Microsoft-ориентирани фирми, организации и индивидуални разработчици. Езикът C# и платформата .NET Framework се поддържат и контролират от Microsoft, но постепенно се отварят и за външния свят чрез миграция към отворен код с усилията на .NET фондацията (www.dotnetfoundation.org) стартирала през 2014 г. Поради години наред затворено развитие на езика C# и .NET платформата, останалите големи световни софтуерни корпорации като Google, Apple, IBM, Oracle и SAP базират своите решения на Java или други платформи и не използват C# като основен език за разработка на своите продукти. Езикът C# се разпространява заедно със специална среда, върху която се изпълнява, наречена CommonLanguageRuntime (CLR). Тази среда е част от платформата .NET Framework, която включва CLR, пакет от стандартни библиотеки, предоставящи базова функционалност, компилатори, дебъгери и други средства за разработка. Благодарение на нея CLR програмите са преносими и след като веднъж бъдат написани, могат да работят почти без промяна върху различни хардуерни платформи и операционни системи. Най-често C# програмите се изпълняват върху MS Windows, но .NET Framework и CLR се поддържат и за мобилни телефони и други преносими устройства, базирани на Windows Mobile. Под Linux, FreeBSD, MacOS X и други операционни системи, C# програмите могат да се изпълняват върху свободната .NET Framework имплементация Mono, която обаче не се поддържа официално от Microsoft. Тенденциите са Microsoft постепенно да започне да поддържа C# и .NET под Linux.

2.3 SQL - база данни

2.3.1 Релационна база данни

Релационна база данни е тип база данни, която съхранява множество данни във вид на релации, съставени от записи и атрибути (полета) и са възприемани от потребителите като таблици. Релационните бази данни понастоящем преобладават при избора на модел за съхранение на финансови, производствени, лични и други видове данни.

Софтуерът, който се използва за организиране и управление на този вид бази данни се нарича най-общо система за управление на релационни бази данни (СУРБД).

2.3.2 Отношение

Отношение (relationship - в някои източници с това значение е натоварен терминът релация) се нарича зависимост, съществуваща между две таблици, когато записи от първата таблица могат да се свържат по някакъв начин със записи от втората таблица. Три са възможните видове отношения, още известни като кардиналности или кардинални числа (cardinality):

- „едно към едно“ (1:1);
- „едно към много“ (1:N);
- „много към много“ (M:N).

Отношение „едно към много“

Отношението от вид „едно към едно“ е налице, когато всеки запис от една таблица е свързан с най-много един запис от втора таблица и всеки запис от втората таблица е свързан най-много с един запис от първата таблица. Този вид отношение е специално, защото е единственото, при което двете таблици могат да споделят един общ първичен ключ. Възможно е обаче и първичните им ключове да са различни и отношението да се създава с използване на външен ключ, като първичният ключ на едната таблица, без значение коя, се включи в структурата на другата.

Отношение „едно към много“

Отношение „едно към много“ между две таблици съществува тогава, когато един запис от първата таблица, наречена родителска, може да бъде свързан с много записи от втората таблица, наречена дъщерна, но запис от дъщерната таблица може да бъде свързан само с един запис от родителската таблица. Отношението между двете таблици се създава като копието на първичния ключ на родителската таблица се включи в структурата на дъщерната таблица, за която той представлява външен ключ. В литературата се среща и кардиналността „много към едно“ (N:1), която е вариант на „едно към много“. Това е най-често срещаният вид отношение между таблици.

Отношение „много към много“

Отношението „много към много“ съществува, когато един запис от едната таблица може да се свърже с много на брой записи от втората таблица и един запис от втората може да се свърже с много на брой записи от първата таблица. За да се създаде на практика това отношение, се използва нова, свързваща или асоциираща таблица, която съдържа копия на първичните ключове на двете таблици. От една страна свързващата таблица представлява сложен първичен ключ на отношението, а от друга страна, всеки от първичните ключове на изходните таблици играе ролята на външен ключ за свързващата таблица.

2.3.3 Нормализация на базата данни

Нормализацията изпълнява следните задачи:

- Изключване на повтаряща се информация в таблиците;
- Декомпозиция на един типов обект на няколко;
- Минимизиране на аномалиите при съхраняване, изтриване и промяна на данните;
- Създаване на отворена към бъдещи промени структура;
- Създаване на структура, свеждаща до минимум влиянието на структурни изменения върху вече създадени приложения.

Нормализацията, т.е. привеждането в нормална форма, включва набор от практики по отстраняването на повторения сред данните, което от една страна води до икономия на памет и повишено бързодействие, а от друга страна предпазва от аномалии при

манипулирането на данните (вмъкване, актуализиране и изтриване) и от загуба на тяхната цялост. В процеса на нормализация се осигурява оптимална структура на базата от данни, основаваща се на взаимозависимостта между данните. Структурата на таблиците се трансформира, с цел да се оптимизират функционалните зависимости на съставните им атрибути.

Има пет нормални форми (НФ):

- Първа НФ: Премахване на повтарящите се атрибути

За да бъде таблицата в първа нормална форма, трябва да са изпълнени следните три условия:

- В таблицата да няма дублиращи се записи;
- Таблицата да няма повтарящи се атрибути;
- Записите в колона или атрибут да са от един и същи тип данни.

- Втора НФ: Отстраняване на многозначните данни

Втора нормална форма (2НФ) гарантира, че всеки атрибут е в пълна функционална зависимост с ключ. Тя се основава на зависимостите в смисъла, че атрибут, който не е част от първичния ключ трябва да бъде зависим от всички негови атрибути. Ако атрибут зависи от един от атрибутите на първичния ключ, но не зависи от останалите, той става частично зависим, което нарушава втора нормална форма.

- Трета НФ : Премахване на атрибутите, които не са зависими от ключа

Трета нормална форма (3НФ) проверява транзитивна зависимост. Транзитивната зависимост е подобна на частичната зависимост по това, че и двете се отнасят до атрибути, които не са напълно зависими от първичния. Зависимостта се зачита като транзитивна когато атрибут₁ е зависим от атрибут₂, който е зависим от първичния ключ.

- Четвърта НФ : Отделяне на многозначните зависимости

Чрез четвъртата нормална форма (4НФ), от два независими атрибута се създава първичен ключ чрез трети атрибут. Но, ако двата атрибута не могат еднозначно да определят информацията без трети атрибут, тогава дизайнът нарушава четвъртата нормална форма.

- Пета НФ : Отделяне на семантично свързаните многозначни зависимости

Петата нормална форма (5НФ) осигурява метод за проектиране на сложни отношения свързани с многобройни (обикновено три или повече)обекти.Удовлетворяването на изискванията на третата нормална форма обикновено е достатъчно.

Нормализацията на базата данни трябва да се преразглежда внимателно и задълбочено за всяка нова таблица, която се разработва. Решенията, които се взимат по време на фазатана изработване на дизайна, играят съществена роля при цялостната цена (време), използваемост и ефективност на крайното приложение. Също така е добре да се преразглежда денормализацията на базата, за да се направи фина настройка на приложението, когато то се разрастне и се увеличи потреблението.



2.4 jQuery

jQueryе една от най-известните и най-използваните библиотеки, алтернатива на **JavaScript**, публикувана в началото на 2006 от Джон Резиг. В основата си jQuery опростява достъпа до всеки елемент на дадена уеб-страница, като по този начин позволява лесно изграждане на динамична функционалност в страниците. JQuery е създаден с цел да промени начина, по който се пише JavaScript. Синтаксисът му е сравнително лесен за научаване и употреба, а възможностите и гъвкавостта на jQuery подпомагат за изключителни резултати. Поддържа се от всички съвременни браузъри, което е още една от многото причини за голямата му популярност.

jQuery е безплатна библиотека с отворен код, лицензиран под MIT лиценз. Използва се в 55% от 10000-те най-посещавани сайтове, което я прави най-популярната JavaScript библиотека днес.

jQuery архитектурата позволява на разработчиците да създават приставки (plug-in), като по този начин разширяват нейната функционалност. В момента има на разположение в интернет над 16 хиляди jQuery приставки, които обхващат широк спектър от функционалности, като помощни приложения тип Ajax, уеб услуги, мрежови масиви от данни, динамични списъци, XML и XSLT инструменти, draganddrop приложения, събития, управление на бисквитки, модални прозорци и други.

2.5 CSS



CSS е съкращение от **CascadingStyleSheets** и представлява отделен език, съдържащ множество "инструменти", с които може да се промени външния вид на HTML страниците. От определена гледна точка CSS е нещо като "надстройка" на HTML. Официално спецификацията на CSS се поддържа от W3C.

CSS е създаден с цел да бъдат разделени съдържанието и структурата на уеб страниците отделно от тяхното визуално представяне. Преди стандартите за CSS, установени от W3C през 1995 г., съдържанието на сайтовете и стила на техния дизайн са писани в една и съща HTML страницата. В резултат на това HTML кода се превръща в сложен и нечетлив, а всяка промяна в проекта на даден сайт изисквала корекцията да бъде нанасяна в целия сайт страница по страница. Използвайки CSS, настройките за форматиране могат да бъдат поставени в един единствен файл, и тогава промяната ще бъде отразена едновременно на всички страници, които използват този CSS файл.

CSS позволява да се определя как да изглеждат елементите на една HTML страница - шрифтове, размери, цветове, фонове и др. CSS кодът се състои от последователност от стилови правила, всяко от които представлява селектор, последван от свойства и стойности.

Селекторите се използват, за да покажат към кои елементи на HTML документа трябва да бъде прилаган съответният стил. Съществуват много видове селектори. Някои селектори позволяват постигане на динамичност на страницата до определена степен. Например, само с помощта на CSS могат да бъдат направени изскачащи менюта, хипервръзки, които при посочване променят цвета си и др.

Глава 3. Проектиране на системата.

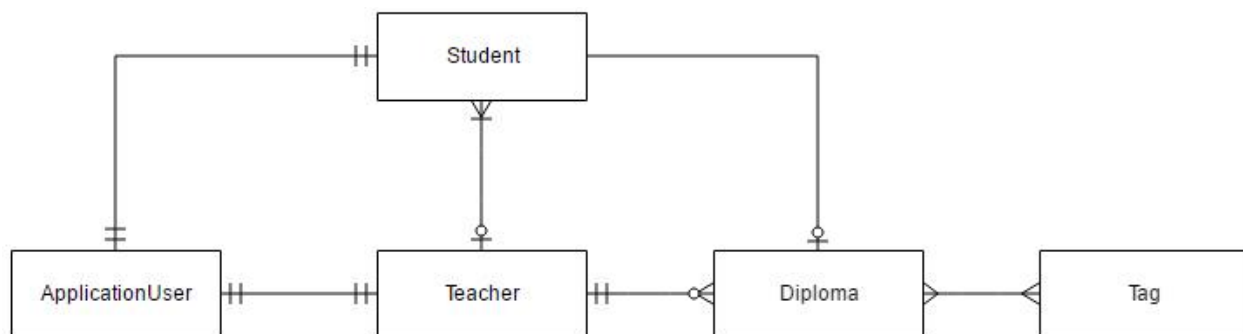
В настоящата дипломна работа ще предложи решение за изграждане на софтуерна система за разпределение и съставяне на дипломни задания. Целта на тази система е да автоматизира процеса на създаване и избиране на теми и така да спомогне абсолвентите и ръководителите на дипломни задания, като предложи едно лесно за употреба web базирано приложение.

3.1 Съставяне на моделите за базата данни



Фигура 3-1 Мястото на моделите в общата архитектура

При съставяне на план за приложение е необходимо да се поставят солидни основи. В този проект основата е правилно съставената база данни и начинът, по който отделните елементи са свързани помежду си.



Фигура 3-2 Схема на обектите за базата данни

Приложението за съставяне на дипломни работи се нуждае от потребител, с който да се извършват операции за идентификация в системата. ApplicationUser е такъв модел. То пази в себе си информацията за отделния потребител като имена, телефон, email и т.н.

Student и Teacher са класове, в които се запазва допълнителна информация за потребителя. Изнасяйки ги в отделни класове, ще се намали отговорността на ApplicationUser и в същото време един потребител може да се възползва от архитектурата и на двата класа.

- В класа Student се запазва всякаква информация, касаеща отделния студент (факултетен номер, адрес, коя диплома е избрал);
- В класа Teacher се съдържат полета с информация за конкретен учител. В този клас се записват дипломите, които преподавателят е добавил, както и студентите, които е поел като дипломанти.

Класът Diploma има за цел да пази информация за индивидуална диплома като полета за заглавие, съдържание, дата на задаване, както и служебна информация за дипломата, която да бъде използвана в логическата част на приложението.

Класът Tag представлява ключова дума, определяща дипломната работа и да съдържаща информация като име, дата на създаване, както и дипломи, които използват тази категория. Тези модели могат да се използват и за определяне на сферите на интерес (технологиите), с които даден преподавател е обвързан/запознат.

3.2 Съставяне на хранилище за елементите на базата (repository)



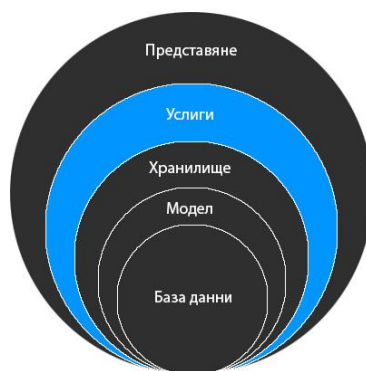
Фигура 3-3 Хранилището (repository) като част от системата

Използвайки repository pattern, отделяме основната част на приложението (web частта) от базата данни. Тестването на бизнес логиката става по-лесно, като по този начин не е зависима от базата данни и може да работи отделно от вида и типа ѝ. Улеснява се достъпът до ресурсите, когато се достъпват от няколко места, като прилага консистентни правила и логика за работа. Този шаблон улеснява поддръжката и разчитането на кода като разделя бизнес логиката от данните и услугите.

В този слой се дефинират някои основни действия свързани с моделите като:

- Връщане на всички елементи от даден тип (модел или таблица от базата данни);
- Вземане на запис по ключ от базата данни;
- Добавяне на елемент от даден тип към базата данни;
- Изтриване на елемент от базата данни (маркиране на елемента като изтрит и пълно изтриване);
- Записване на промените по базата данни.

3.3 Създаване на слой с услуги (Service layer pattern)



Фигура 3-4 Мястото на слой услуги в архитектурата на приложението

Service layer модел, прилаган при дизайн, ориентиран към услугите, и има за цел да ги организира в рамките на едно общо място като логически слоеве. Услуги, които се категоризират с определена функционалност обуславят един слой. Това помага за намаляване на сложността на управлението на слоевете на услуги, тъй като услугите, които принадлежат към един и същи слой, адресират по-малък набор от дейности.

Употребата на услуги (services) улеснява допълнително работата с данните, както и компонентното тестване (unit testing) на контролерите на web приложението. Достъпът през service дава възможност за настройка на върнатата информация според нуждите на приложението (например сортиране по име на студентите).

Необходимите услуги са директно свързани с типовете модели, които има приложението.

Услугата за операции с дипломи, трябва да има някои фундаментални операции като:

- Избиране на диплома по ID;
- Избиране на всички дипломи;
- Избиране на дипломи, принадлежащи на даден преподавател.
- Добавяне на дипломи.

Услугата за работа с преподаватели, трябва да има следните операции:

- Избор на преподавател по ID;
- Избор на преподавател по ID на потребителския модел (ApplicationUser) ;
- Добавяне на диплома към колекцията от дипломи на преподавателя;
- Добавяне на студент към колекцията от дипломанти на преподавателя.

- Добавяне на преподаватели.

Услугата за работа със студенти, трябва да има следните операции:

- Избор на студент по ID;
- Избор на студент по ID на потребителския модел (ApplicationUser) ;
- Избор на студент по факултетен номер.
- Добавяне на студенти.

Услугата за работа с таг, трябва да има следните операции:

- Избор на таг по ID;
- Избор на таг по име;
- Добавяне на таг.

3.4 Създаване на подходящи контролери и изгледи



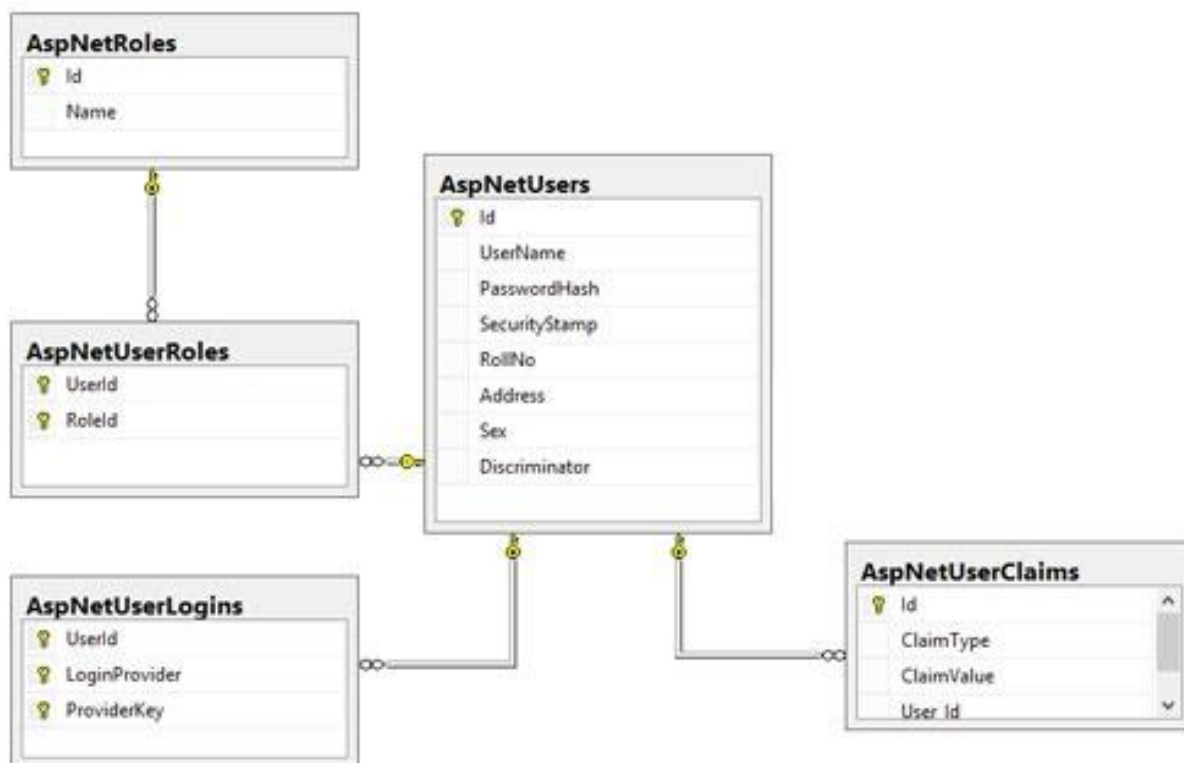
Фигура 3-5 Мястото на контролерите и изгледите като представителен слой на приложението

За да могат да се изберат подходящи контролери за приложението, е необходимо да се изследват целите на самото приложение. То има няколко основни изисквания като:

- Приложението трябва да може да поддържа регистрации;
- Трябва да има потребители от различни роли (администратори, студенти, преподаватели);
- Всеки от типовете потребители има свои задачи (действия), които може да изпълнява.

3.4.1 Регистрация на потребители

За да се регистрира нов потребител, web приложението може да използва предоставената от Microsoft архитектура. Класовете, които се използват на готово, включват основна информация за потребителите, в това число и контролер за управление на потребителя. Процесите по регистрация и вход са основни за всяко едно приложение, което желае да има разделение и различни функционалности за отделни посетители.



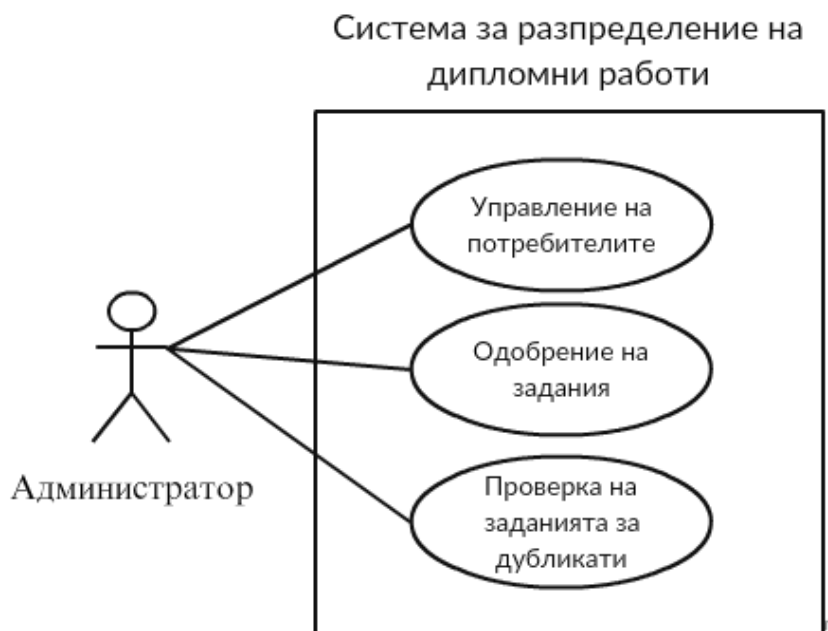
Фигура 3-6 Схема на готовите класове, които да бъдат разширени

За да могат да се използват, готовите класове трябва да бъдат разширени за нуждите на приложението. Тъй като основната задача на проекта е да се улесни процеса на избор на диплома от абсолвентите, при регистрирането на потребител е подходящ момент за създаването на обект от тип студент, който да е обвързан с новия потребител и да му се зададе роля от тип студент.

3.4.2 Избор на роли в приложението

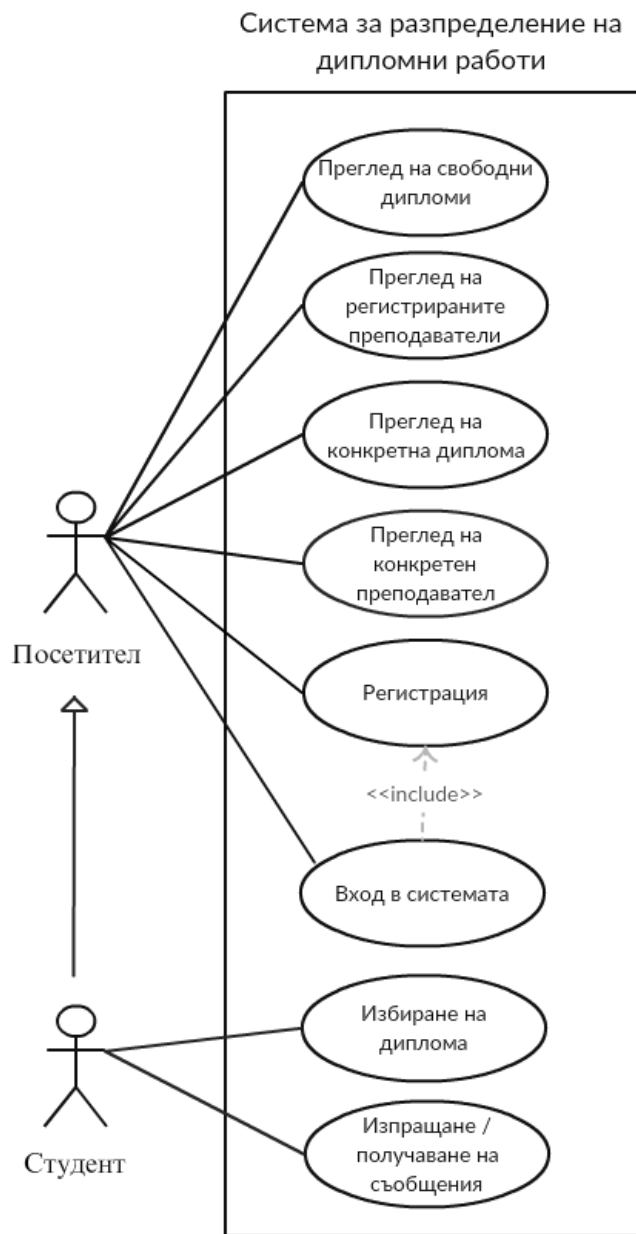
Приложението трябва да има основни роли - потребители, които да управляват самото приложение (**администратори**). Тези потребители са няколко за целия проект. За създаване на тази функционалност трябва да има контролер, който да бъде достъпен само

за тях (администраторите на системата). Функционалностите, които е необходимо да има, са промяна на част от потребителската информация и промяна на правата на останалите потребители (студенти, преподаватели). Тези потребители са подходящи и да играят роля на най-високо ниво в управлението на цялостния процес по избиране и одобрение на дипломна работа (одобрение на дипломата от декана на факултета).



Фигура 3-7 Use case диаграма на основните действия на администратор

Приложението трябва да има потребители от тип студенти. Студентите са уникални потребители със способността си да избират вече създадени дипломи. Потребителите от този тип ще могат да разглеждат списъци с дипломни работи, сортирани по различни показатели. Студентите трябва да могат да избират тема по части от име, по различни технологии или по даден преподавател. По този начин самият процес по намиране на подходяща диплома се улеснява и автоматизира.

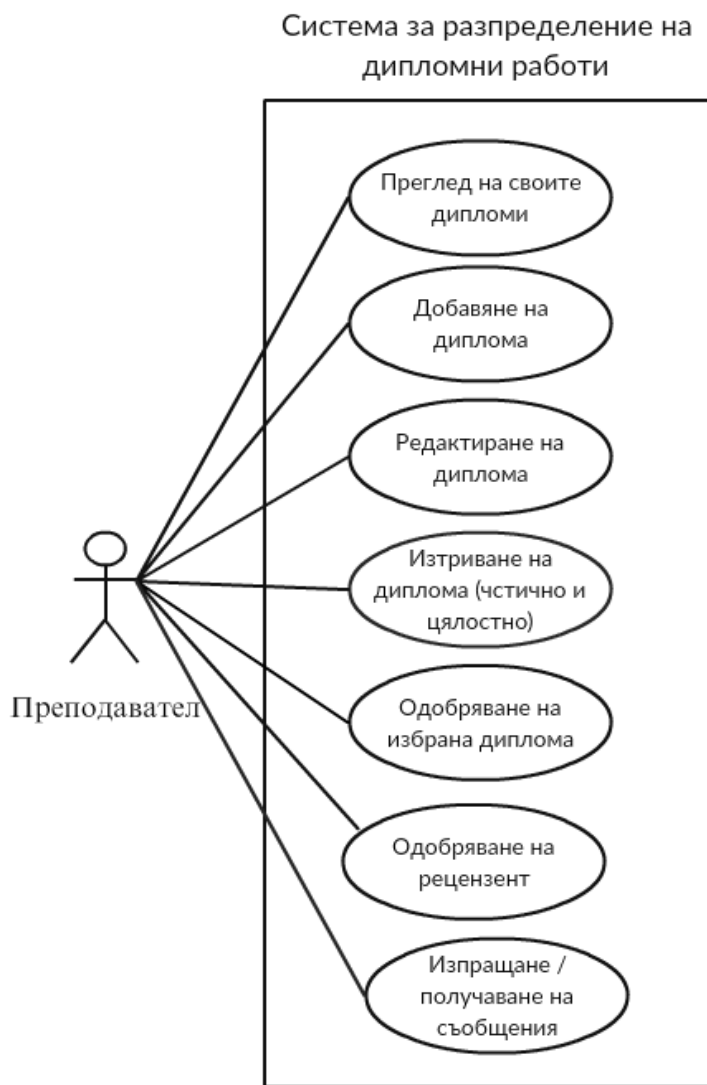


Фигура 3-8 Use case диаграма на посетител и студент

На фиг. 3-8 е представена use case диаграма на система за разпределение и съставяне на дипломни работи за посетител и студент. Представени са основните действия, които могат да се извършат от регистрираните и нерегистрираните потребители. На фигурата се вижда, че системата трябва да може да се използва и като източник на информация. Нерегистрираните потребители могат да разглеждат дипломи и теми, предоставени от

конкретен преподавател, но само след регистрация потребителя (в случая студента) може да избере конкретна тема като желана от него/нея за изработване.

Приложението трябва да има потребители от тип преподаватели, които да създават дипломни теми. Тези потребители трябва да могат да променят своите задания, както и да ги изтриват. Също така преподавателите трябва да могат да одобряват или не дипломни теми, предложени от даден студент, като одобрените теми се дават на следващото ниво за утвърждаване, а неодобрените теми трябва да се преразгледат.



Фигура 3-9 Use case диаграма за преподавател

На фиг. 3-9 е показана use case диаграма за потребител от тип преподавател. Показани са основните функции, които може да извърши преподавателя.

Глава 4. Програмна реализация.

В тази глава ще разгледаме конкретна имплементация на приложения за съставяне и разпределение на дипломни задания. Приложението е направено следвайки съображенията разгледани в предходната глава.

Web приложението е разделено на няколко под проекта всеки, от които е натоварен с определена цел и задача. Проектите са разделени в папки както следва:

- Папка Data:
 - DDS.Data – проект, в който се намира логиката за създаване на базата данни
 - DDS.Common – проект, който съдържа обща логика за класовете (моделите)
 - DDS.Models – проект, в който са описани самите модели
- Папка Services:
 - DDS.Services.Data – тук са разположени всички услуги, както и техните интерфейси
 - DDS.Services.Web – проект, в който са разположени услуги, касаещи приложението като кеширане (cache)
- Папка Web:
 - DDS.Web – стартов проект за приложението. Тук се намира цялата логика за обработка на извлечената от базата информация, настройки на приложението (routing), контролерите, изгледите и т.н.
 - DDS.Web.Infrastructure – проект, в който се намира логиката за външни операции, използвани в проекта (automapper, docx генератор)
- DDS.Common – е проект, отделен за обща логика за цялото приложение. Там се намират константите, използвани в кода.

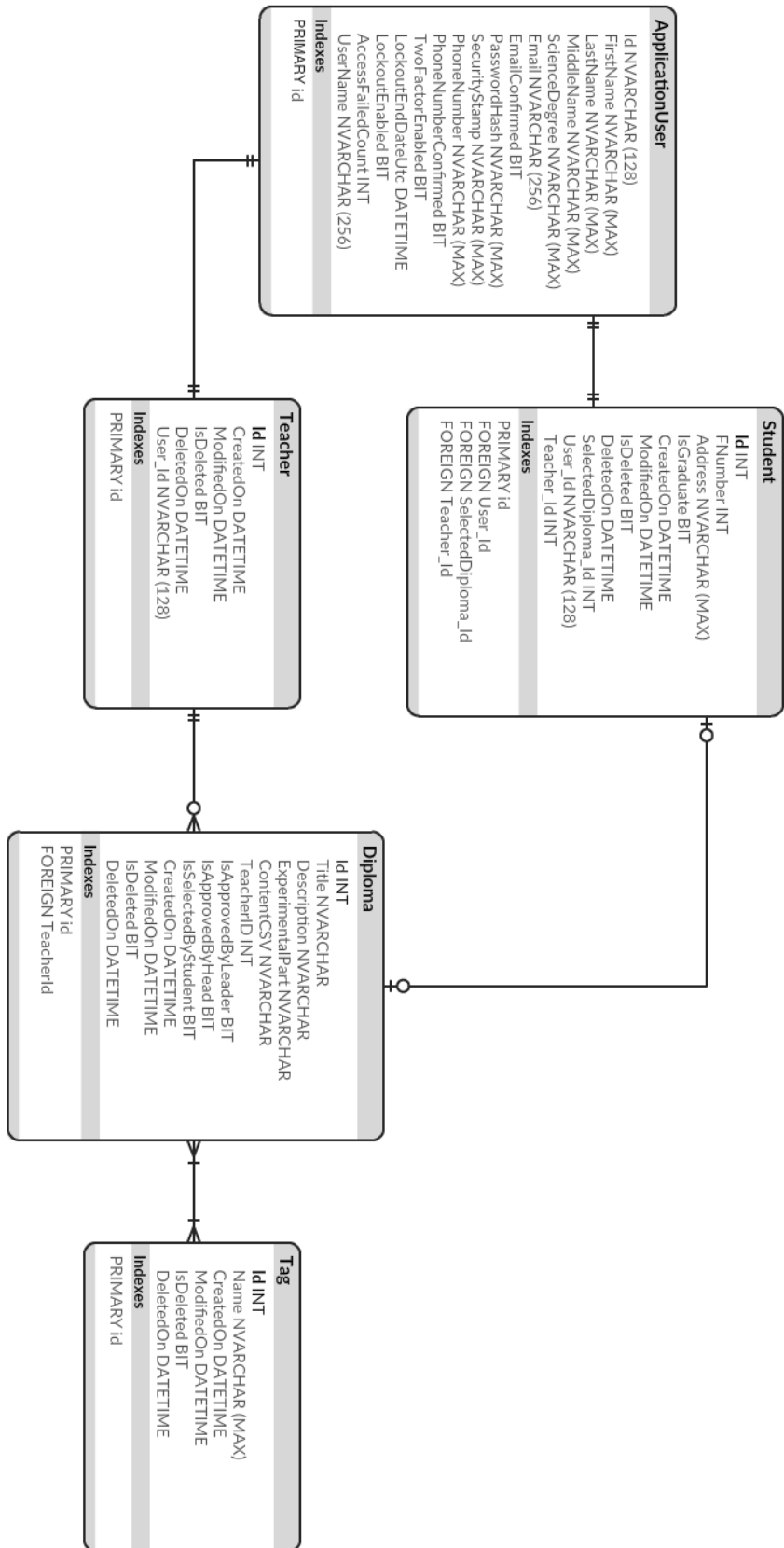
4.1 Създаване на базата данни

За да може едно приложение да е удобно за разширение и поддръжка, е важно да се използват технологии и методи, които са доказано ефективни. За целите на настоящото приложение се използва среда за разработка Visual Studio 2015 Community. Софтуерът е много удобен и позволява използването на най-новото поколение рамки (frameworks).

За създаването на базата данни се използва модела *code first*, т.е. създаване на моделите преди създаването на самата схема на базата. Visual Studio позволява лесен достъп до настройките на миграциите на проекта, чрез вградената конзола за инсталиране на пакети (**Package Manager Console**). За стартиране на миграциите за приложението се изпълнява командата **Enable-Migrations** върху проекта DDS.Data. Тази команда създава папка с име Migrations в проекта DDS.Data, в която се генерират **Configuration.cs** файл, който съдържа логиката за настройка на миграциите, както и информация, с която да се запълни базата при първоначално пускане.

В папката Migrations се помещава и файловете с логиката по създаване на базата. Тези файлове се създават с командата **Add-Migration** в конзолата за инсталиране на пакети, но преди това трябва да се създадат моделите, които да бъдат използвани.

На схема 4-1 са изобразени обектите (entities), които ще бъдат използвани в настоящата система.



Фигура 4-1Схема на базата данни (ERD)

ApplicationUser – Този модел е наследник от генерирания за нас код. В него са добавени полета за име, презиме, фамилия, научни степени, поле за ID на информацията за студент и поле за ID на информация за учител. Този модел се използва от всеки един регистриран потребител. Наследявайки **IdentityUser**, в този клас се намира и допълнителна информация като email, телефон, информация за вписване, потребителско име.

BaseModel<T> представлява базов модел, в който се намира поле ID за отделните наследници (модели). Имплементира интерфейсите **IAuditInfo** и **IdeletableEntity**, чрез които съдържа, дата на създаване, дата на последно модифициране, булево поле за това дали конкретен запис е изтрит, както и дата на изтриване (ако бъде изтрит запис).

Student – е моделът, който съдържа информацията за студента, необходима за заданието, както и информация за избраната диплома и служебно булево поле за това дали студента е завършил успешно. Наследява **BaseModel<int>**.

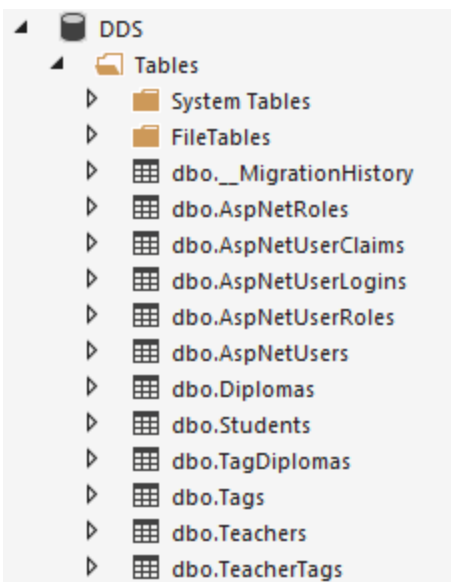
Teacher – модел, в който са поместени колекции с абсолвенти, дипломи и категории, касаещи конкретния ръководител. Класът има конструктор, чиято цел е да създаде в паметта гореспоменатите колекции.

Diploma – модел, който има полета за работа с дипломни задания. Съдържа полета за заглавие, кратко описание на темата, описание на експерименталната част, глави, категории, ръководител и служебни полета за избор от студент, одобрение от ръководител, канцелария и технически сътрудник (подпис от декан).

Tag – модел, който има поле за име на категорията, както и колекция от дипломи и ръководители обвързани с конкретната категория.

След като тези класове са създадени и се изпълни команда “Add-Migrations Initial”(в **Packet Manager Console**), се генерира файл с информация за схемата на базата данни. Този файл съдържа извършените промените и стъпките, които трябва да се изпълнят, за да се премине в новото състояние на базата, както и връщането в предишно такова.

Генерираните таблици изглеждат така:



Фигура 4-2 Таблиците генерирани от така създадените обекти

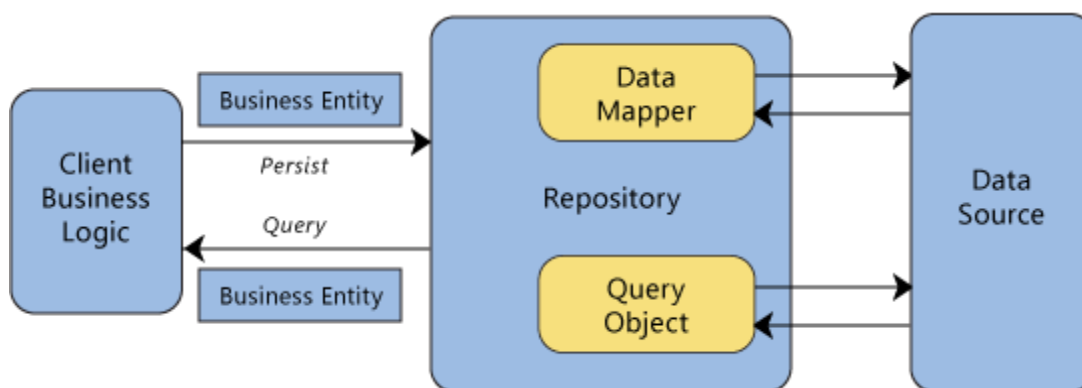
Таблица MigrationsHistory съдържа информация за състоянието, в което се намира базата, като в нея се запазват последователно извършените миграции. Когато се правят промени по приложението (добавяне на обекти към база), се създава нова миграция с помощта на командата Add-Migration. Данните от обекта се записват във файл с името на миграцията и при обновяване на базата с команда update-database, в тази таблица се записва името на извършената миграция. В случай, че се налага връщане на промените, от тази таблица се взема името на файла с миграцията, която трябва да се изпълни.

Таблиците AspNetRoles, AspNetUserClaims, AspUserLogins, AspUserRoles, AspNetUsers са служебни таблици, генерирани от Entity Framework Migrations.

- AspNetUsers представлява таблица, която съдържа информацията за потребителя. Генерира се от класа ApplicationUser, който е наследник и разширява IdentityUser.
- AspNetRoles съдържа информация за ролите, които един потребител може да получи. Съдържа основен ключ на ролята и име. За целите на тази дипломна работа, са създадени ролите Administrator, Student, Teacher.
- AspUserRoles е междинна таблица за потребителите и ролите им. Съдържа колони за основен ключ на потребител и основен ключ на роля.
- AspNetUserClaims се генерира от IdentityUserClaim<TKey>
- AspUserLogins се генерира от IdentityUserLogin<TKey>

4.2 Създаване на хранилище

Използването на репоситори позволява разделянето на логиката по извличане на данните и попълването им в моделите от бизнес логиката, която се извършва върху моделите. Бизнес логиката трябва да бъде агностична към данните, идващи от слоя за данни. Това позволява смяната на източника, без да се отрази на работата на приложението.



Фигура 4-3 Репоситори шаблон

В текущата дипломна работа е използван този шаблон. Интерфейсът за хранилището е **IDbRepository{T}.cs** и съдържа следните функции:

- **IQueryable<T> All()** – използва се за извличане на query на всички данни от даден тип без вече изтритите (например студенти).
- **IQueryable<T> AllWithDeleted()** – използва се за извличане на query на всички данни от даден тип, включително изтритите.
- **T GetById(TKey id)** – връща обект от даден тип, който бива намерен по ID
- **void Add(T entity)** – функция за добавяне на елемент към базата данни
- **void Delete(T entity)** – функция за изтриване на елемент към базата данни, като маркира елемента за изтрит
- **void UnDelete(T entity)** – функция за премахване на маркера за изтриване на елемент от базата данни
- **void HardDelete(T entity)** – функция за изтриване на елемент към базата данни
- **void Save()** – функция за запазване на промените

Наследявайки и работейки само с `IDbRepository{T}` можем да осигурим функционалността, която ни дава като интерфейс и по този начин може да се напише конкретна имплементация за различните видове източници на информация. В случая конкретната имплементация е `DbRepository<T>`, където `T` представлява елемент от типа `BaseModel<int>`.

4.3 Създаване на услуги (services)

Слоят с услуги определя набор от налични операции, които скриват бизнес логиката на приложението. Предназначението на слоя е да намали обвързването и подобри поддръжката на кода. Услугите са разпределени в отделен проект, независим от MVC инфраструктурата. В него се намират услуги за всеки отделен модел.

- **BaseService<T>** - базова услуга, която има за цел да дефинира основни операции върху модели, наследяващи `BaseModel<int>`. Такива модели са `Student`, `Teacher`, `Diploma`, `Tag`. В тази услуга се намират основните операции върху хранилището като четене, запис и изтриване на модели от базата.
- **DiplomasService** – услуга, предназначена за операции върху дипломните задания.
- **StudentsService** – услуга, предназначена за операции върху студентите.
- **TeachersService** – услуга, предназначена за операции върху преподавателите например добавяне на студент или диплома към колекцията.
- **TagsService** – услуга, предназначена за специфични операции върху категориите (проверка за съществуваща такава)
- **MessagesService** – услуга, която наследява `BaseService` и няма имплементирани методи, тъй като изпращането на съобщения е опростено.

Основно предимство, което ни предоставя това разделение, е голямото разделение на кода. Тази архитектура е подходяща както за малки, така и за значително по-сложни проекти. В случай на нужда от разширение, много лесно може да се добави необходимата логика в определеното за нея място. Добавянето на нови модели от типа `BaseModel<int>` наследява автоматично операциите за четене и запис.

4.4 Autofac – предназначение и настройки

В софтуерното инженерство **Inversion of control (IoC)** - инверсия на контрол) е принцип, при който част от кода на приложението получава редът на изпълнение на операциите от обща рамка. В софтуерната архитектура с този принцип се обръща контрола в сравнение с традиционното процедурно програмиране: в традиционното програмиране кодът, който изразява целта на програмата, извиква преизползваеми библиотеки за да извърши някаква основна операция, но inversion of control (IoC) рамката (framework) е тази, която извиква необходимия код за изпълнение. Използва се за повишаване на модулността на приложението и улеснява бъдещи разширения.

В дипломната разработка за извършване на IoC се грижи Autofac. Библиотеката управлява зависимостите между класовете така, че приложението да остане лесно за промяна, когато се разрасне по размер и сложност. Постига се чрез третирането на класовете от .Net като компоненти. Autofac има предимство пред други подобни библиотеки със сравнително лесното си прилагане.

Библиотеката, която е инсталирана в дипломната разработка се казва Autofac.MVC5, която зависи и инсталира Autofac библиотеката.

Конфигурацията на Autofac е във файл AutofacConfig, намиращ се в папка App_Start на проекта Web. Класът AutofacConfig изпълнява неща строго специфични за Autofac и неговата интеграция в MVC. В него се намира метод за регистрация на услуги. Целта е инстанцирането на контролерите без празни конструктори, използвайки интерфейси. Това, което ще направи Autofac е да обвърже интерфейси с конкретни имплементации. Когато бива извикан обект от инстанциран интерфейс, библиотеката ще върне обект от съответната имплементация. Всички услуги на приложението са регистрирани в Autofac чрез подаването на цялото асембли, използвайки един от интерфейсите (в случая IDiplomasService).

```
var servicesAssembly = Assembly.GetAssembly(typeof(IDiplomasService));  
builder.RegisterAssemblyTypes(servicesAssembly).AsImplementedInterfaces();
```

За да се стартира Autofac е необходимо да се извика методът RegisterAutofac() в Global.asax на приложението по следния начин:

```
AutofacConfig.RegisterAutofac();
```

4.5 Създаване на контролери

Системата за създаване и избор на дипломни задания се нуждае от няколко сравнително малки контролера. Контролерите за всеки един от типовете потребители, повишава сигурността и улеснява навигирането из тях. Следвайки предоставената архитектура на MVC, контролерите се намират в проект DDS.Web в папка Controllers.

- **BaseController** – базов контролер, който наследява Controller и добавя обща логика за всички контролери.
- **ManageController** – контролер, който има грижата за промяна на настройките на потребителя (смяна на парола, добавяне на телефон и т.н.)
- **AccountController** – контролер, който се занимава с вписването на потребителите, както и регистрацията на нови такива.
- **HomeController** – контролер, съдържащ основните действия (actions), които може да извърши един посетител (посещаване на началната страница, показване на списък с дипломи, детайли за определена диплома, избиране на задание от студент, показване на списък с ръководители и детайли за определен такъв)
- **ManageDiplomasController** – контролер, който се занимава с логика по операциите необходими на ръководител за създаване и редактиране на дипломи, изтриване (маркиране за изтрити и пълно изтриване) на задания, както и одобряване на задания, избрани от студенти.

4.6 Създаване на изгледи

Изгледът представлява скрипт, чрез който приложението разбира какво да се визуализира на екрана. Намират се в папка Views на основния проект DDS.Web.MVC. Конвенцията казва, че имената на изгледите трябва да отговарят на имената на методите в контролерите, които ги извикват, т.е. когато се извика метод от даден контролер от тип ActionResult, изгледа, който трябва да се визуализира, се намира в папка с името на контролера, именуван по същия начин, както извикания метод. Приложението разполага с един основен изглед _Layout, намиращ се в папка Shared. Целта на изгледа е да покаже основните части на Web приложението, които са част от всяка страница като навигация и футър. В изгледа се извиква метод @RenderBody(), който има за цел да визуализира конкретен изглед в себе си.

- **Account/Register** - Изглед на формата за регистрация на нов потребител. Състои се от полета за въвеждане на информация за потребителя като имена, email, потребителско име и парола. Използва изглед-модел RegisterViewModel. При подаване на попълнената информация се извиква метод Register от контролер Account, който проверява въведената информация и при правилно попълнени полета създава нов потребител от тип ApplicationUser. При въвеждане на невалидна информация, на екрана се показват съответните насочващи потребителя съобщения.
- **Account/Login** – Изглед, който позволява на потребители да влезнат в своя профил чрез подаване на потребител и парола. При подаване на информацията се извиква метод Login от контролер Account. При неправилно въведена информация на екрана се изписват съответните съобщения за грешка.
- **Home/Index** – Стартов изглед за приложението. Представява статична страница с информация за процеса по изработка на дипломни работи.
- **Home/Diplomas** – Изглед, който визуализира дипломите, подадени му от контролера. Използва CommonDiplomaViewModel. От този списък с дипломи студентите могат да изберат конкретна диплома, което води до изглед за детайли на дипломни задания.
- **Home/Details** – Изглед, показващ детайлите за дипломна работа. Използва изглед-модела DisplayDiplomaViewModel. Студентът има възможност да избере дипломното задание, ако вече не е избрал такова, чрез бутона „Избери“. При избора на дипломна работа се извиква методът Select в контролер Home, който присвоява дипломното задание към вписания студент и маркира заданието като избрано.
- **Home/Tag** – Изглед, който визуализира свободните дипломи по избрана категория. Използва CommonDiplomaViewModel. Има възможност за търсене на дипломно задание по част от заглавие или описание.
- **Home/TeacherDetails** – Този изглед показва информация за ръководител като телефон и e-mail и неговите свободни задания. Използва изглед-модела

TeacherDiplomasViewModel. При избор на дипломна работа от списъка, потребителят е препратен към страницата за детайли към това задание.

- **Home/Teachers** – Представява списък с ръководители. Използва TeacherViewModel. Към информацията за преподавателя са изброени и всички категории на неговите задания. При избор на ръководител, потребителят е препратен към страницата с детайлите на преподавателя.
- **Manage/ChangePassword** – Изглед, който позволява на потребителя да смени паролата си. Използва изглед-модела ChangePasswordViewModel. При подаване на информацията се извиква функцията ChangePassword от контролера Manage. При неправилно въведена информация на екрана се изписва подходящо съобщение.
- **Manage/Index** – Изглед, показващ информация за потребителя. Съдържа форма за попълване или промяна на тази информация като извиква функцията SaveInfo от контролера Manage. Има препратка към страница за смяна на паролата.
- **ManageDiplomas/Index** – Изглед, който визуализира списък с дипломни задания на конкретен ръководител. В зависимост от статуса на заданието, то се оцветява в определен цвят с цел по-добра ориентация. Задания, които са избрани от студенти биват оцветени в жълт цвят. Тези, които са одобрени от ръководителя – в синьо, а напълно одобрените задания – в зелено. Свободните задания остават неочветени. Под списъкът със задания е разположена легенда за цветовете и тяхното значение. Когато едно задание е свободно, то може да бъде редактирано или изтривано от преподавателя, премествайки го в кошчето. Когато се натисне бутонът „Кошче“, намиращ се над списъка с дипломни работи, ръководителят е препратен към списък с всички изтрети от него задания. На този изглед, отляво на кошчето е разположен и бутон за създаване на нова дипломна работа, препращащ към съответната страница. Ръководителят има възможност да търси дипломни задания по част от заглавие и описание. Допълнителна опция е търсене по константни ключови думи:

- „избрани“ – в списъка ще се визуализират само дипломни задания, избрани от студент.
 - „одобрени“ – в списъка ще се визуализират само одобрените дипломни задания от ръководителя.
 - „свободни“ – в списъка ще се визуализират всички свободни дипломни задания.
- **ManageDiplomas/Create** – Изглед за създаване на нова дипломна работа. Визуализира форма за попълване на данните на дипломното задание, състояща се от полета за:
 - Тема на дипломното задание
 - Кратко описание
 - Въвеждане на експериментална част
 - Съставяне на съдържанието на дипломното задание, което има бутон позволяващ добавянето на допълнително поле за нова точка
 - Динамично падащо меню за избор на технологии/категории на дипломната работа

При натискане на бутона „Запази“ се извиква функцията Create в контролер ManageDiplomas. Възможна е отмяна на операцията, което ще върне преподавателя на списъка с дипломни задания.
 - **ManageDiplomas/Deleted** – Представява страница с маркираните за изтрети от ръководителя задания. Когато списъкът е празен, над него се показва подходящо съобщение. На тази страница преподавателят има възможност да върне заданието чрез бутона „Върни“ или напълно да изтрие дипломната работа чрез бутона „Изтрий?“. Над списъка има бутон за създаване на ново дипломно задание, който препраща към съответния изглед, както и бутон за пълното изтриване на всички дипломни задания от списъка. Страницата разполага и с търсачка по част от заглавие или описание.
 - **ManageDiplomas/Details** – Страница, която показва детайли за определена дипломна работа като служебната информация за дипломното задание (тема, описание, съдържание, експериментална част, линкове към категориите, кога е създадено и последно променяно, дали е избрано от студент, дали вече е

одобрено от ръководител и дали е одобрено от декан), информация за студента, избрал конкретното задание (ако има такъв). Ако дипломната работа не е одобрена от ръководителя, се визуализира бутон за одобрение, който извиква функцията Approve в контролер ManageDiplomas.

- **ManageDiplomas/Edit** – Това е изглед, който позволява редактиране на дипломно задание. Използва общия изглед-модел DisplayDiplomaViewModel. Дава възможност за промяна на всяко поле от информацията за заданието, включително и добавяне на нови точки към съдържанието. При желание за запазване на промените се натиска бутонът „Запази“. Ръководителят може да се върне към списъка с дипломни задания чрез бутона „Отмени“.

Създаване на изгледи: Area Administration

- **Administration/Index** – Изглед, който показва списък с всички потребители. Показва техните лични данни, потребителско име и роля. От тази страница може да се избере потребител с цел промяна на някое от полетата.
- **Administration/Edit** – Изглед на форма за промяна на личните данни на избрания потребител. Състои се от полета за въвеждане на информация за потребителско име, фамилия, e-mail и падащо меню за потребителски права. Чрез този изглед администраторите могат да променят правата на даден потребител като могат да присвояват на потребител само по една роля. Предишната роля бива изтрита.
- **ManageDiplomas/Index** – Показва изглед, който визуализира списък с дипломни задания вече одобрени от техните ръководители. В зависимост от статуса на заданието, то се оцветява в определен цвят. Заданията, които са одобрени от ръководителя, са оцветени в синьо, а тези, които са одобрени от страна на декана се оцветяват в зелено. Под списъка със задания е разположена легенда за цветовете и тяхното значение. Тук потребителят има възможност да търси дипломни задания по част от заглавие или описание. При избиране на дипломна работа от списъка, се пренасочва към страница с детайли на дипломната работа и опции за нейното одобрение.
- **ManageDiplomas/Details** - Страница, която показва детайли за определена дипломна работа като служебната информация за дипломното задание (тема,

описание, съдържание, експериментална част, линкове към категориите, кога е създадено и последно променяно, дали е избрано от студент, дали вече е одобрено от ръководител и дали е одобрено от декан), информация за студента избрал конкретното задание. Ако дипломната работа не е одобрена от канцелария и декан, се визуализира бутон за одобрение, който извиква функцията `Approve` в контролер `ManageDiplomas`.

4.7 AutoMapper – настройки и приложение

В проекта `DDS.Web.Infrastructure` е разположена библиотеката `AutoMapper`

Конфигурация

В папка `Mapping` се намира интерфейс `IMapFrom<T>` с помощта, на който се определя кои класове могат да бъдат конфигурирани (`mapped`). Когато е необходимо един изглед-модел да бъде „преведен“ от даден обект, било то от източника на данни (базата данни) или от друг изглед-модел, той трябва да наследи интерфейса `IMapFrom<T>`, където `T` е типът на източника на информация. За да се извърши обратна връзка е създаден интерфейс `IMapTo<T>`.

Помощният интерфейс `IHaveCustomMapping` дава възможност за изпълняване на специфичен мапинг. `AutoMapper` позволява автоматично конфигуриране на полетата с еднакви имена на желаните класове. Прилагайки `IHaveCustomMapping` към целевия клас (например изглед-модел) и имплементирайки функцията `CreateMappings`, се позволява конфигуриране на полета с различни имена и прилагане на прости операции върху данните преди записване.

Класът `AutoMapperConfig` събира всички класове, които има в подаденото му асембли чрез рефлексия, извлича тези, които наследяват `IMapFrom<T>` и `IHaveCustomMapping` и извършва конфигурирането им. Асемблито, което се подава, е на текущото приложение. Подавайки го, `automapper` разбира от къде да вземе всичките публични типове (класове), които наследяват гореспоменатите интерфейси. Този процес се намира в `Global.asax`:

```
var autoMapperConfig = new AutoMapperConfig();  
autoMapperConfig.Execute(Assembly.GetExecutingAssembly());
```

Помощния клас `IQueryableExtensions` е разширение на възможностите на типа `IQueryable`, като единствения метод, който е необходим е `IQueryable<TDestination>`

To<TDestination>. Целта на този метод е да премахне ограничението, което AutoMapper налага в инсталираната версия.

```
public static IQueryable<TDestination> To<TDestination>(this IQueryable source,
    params Expression<Func<TDestination, object>>[] membersToExpand)
{
    return source.ProjectTo(AutoMapperConfig.Configuration, membersToExpand);
}
```

С така дефинирания клас, не се налага подаване на конфигурацията на библиотеката всеки път, когато използваме AutoMapper и инсталирането му в Web проекта.

4.8 Генериране на Docx файлове

След като дипломата бъде одобрена и подписана от декан, тя може да бъде изтеглена като текстов файл. Генерирането на *.docx файловете се осъществява с помощта на библиотеката DocX. Процесът изисква шаблонен файл, в който се намират текстови константи. Тези контейнери биват заместени с извлечената от базата информация.

Примерен код на процеса по генериране на *.docx файл:

```
Dictionary<string, string> properties = new Dictionary<string, string>();
properties.Add("[USER_NAME]", "Стефан Николов Петров");
var doc = DocX.Create(newFilePath);
doc.ApplyTemplate(templateFilePath);
foreach (var prop in properties)
{
    doc.ReplaceText(prop.Key, prop.Value);
}
doc.Save();
```

Създава се променлива от тип Dictionary<string,string>, която да запази контейнерите (като текстов ключ) и съответните им стойности (като текст). След това се създава обект в паметта от тип DocX. Обекта първоначално се попълва с шаблон като се подава физическият път до него (функция ApplyTemplate(string filePath)). След като вече шаблона съществува, се прилага цикъл върху речника и за всяка стойност се прилага функцията ReplaceText(string key, string value), която заменя контейнерите с техните стойности. Накрая се записва физически файл на подадената директория с Save()

Шаблонът, който се използва се намира в Content\DocXFiles\Templates и се казва **DiplomaTemplate.docx**. Когато дипломата избрана от студента бъде одобрена, на неговата страница с детайли за дипломно задание се появява бутон за изтегляне на попълненото задание. Генерираното задание се запазва на сървъра на директория Content\DocXFiles и се именуват с факултетния номер на студента. Шаблонът може да се види на следващата страница.

Шаблонът е приложен отделно от дипломната работа поради специфичния си формат (DiplomaTemplate.pdf). Тази страница не се включва във физическата част на дипломната изработка.

Глава 5. Ръководство на потребителя

В тази глава се разглежда кратко ръководство на потребителя за системата за съставяне и разпределение на дипломни задания. Приложението е предназначено за няколко типа потребители.

При отваряне на сайта се зарежда началната страница. Най-отгоре на страницата има навигация, която е различна за всеки тип потребител. Ако все още не си регистриран потребител на сайта, в навигационната лента има опции за достъп до начална страница, списък с дипломни задания, списък с ръководители, както и регистрация и вход.



Фигура 5-1 Част от началната страница

Нерегистрираният потребител може да се регистрира чрез опцията „Регистрация“, която препраща към страница за регистрация, състояща се от полета за име, презиме, фамилия, e-mail, потребителско име, парола, потвърждение на парола. Под полетата за попълване на данни има бутони за регистрация, чрез който се създава потребител и се вписва автоматично. При неправилно въведена информация се появяват подходящи съобщения за допуснатата грешка.

Регистрация.

Регистрация на нов потребител

- Полето за потребителско име е задължително!
- Не е въведено потвърждение на паролата!

Име	<input type="text" value="Стефан"/>
Презиме	<input type="text" value="Николов"/>
Фамилия	<input type="text" value="Петров"/>
Е-майл	<input type="text" value="steff.kn@abv.bg"/>
Потребителско име	<input type="text"/>
Парола	<input type="password" value="....."/>
Потвърди паролата	<input type="password"/>

Фигура 5-2 Форма за регистрация с съобщение за грешка

Ако потребителят вече има регистрация, може да се впише чрез натискане на бутона „Вход“ от навигацията, който отвежда потребителя на форма за вход. Тя се състои от полета за потребителско име и парола. При попълнени данни без грешка, препраща към началната страница и в зависимост от типа потребител навигацията получава допълнителни бутони.

За да попълни информация за себе си, потребителят може да натисне бутона „Здравей ...“ от навигацията, който ще го препрати към страница с настройки на лична информация (фигура 5-3). Страницата е различна в зависимост дали потребителят е студент. Всички потребители разполагат с линк за смяна на паролата и полета за въвеждане на телефонен номер и научни степени (ако има такива). Когато студент посети страницата за настройки, тя съдържа допълнително линк за детайли за избраната от него диплома (ако има такава), полета за въвеждане на факултетен номер и адрес. При въвеждане на грешна информация на потребителя се показват съобщения за съответните пропуски.

Промяна на данните на потребителя.

Парола: [[Смени паролата си](#)]

- Полето за телефон е задължително!
- Полето за адрес е задължително!

Фак. Номер

Адрес

Телефон

Научни степени

Фигура 5-3 Форма за промяна на лична информация и съобщение за грешка

5.1 Ръководство за потребител тип студент.

Когато студентът се впише в системата, бива препратен към началната страница. В навигационната лента на този потребител се намира бутон за списък с дипломни задания и списък с ръководители.

Списъкът с дипломни задания (фигура 5-4) представлява таблица със следните колони: заглавие, описание, ръководител, категории и дата на добавяне. При натискане на ред от таблицата, студентът бива препратен към детайли за избраната диплома. Той може да избере име от колонката с ръководители, което го препраща към детайлите за него и списък с дипломни задания, които той приема. От колонката с категории студентът може да избере технология, което ще го препрати към списък с дипломи, използващи тази категория. Студентът има възможност да сортира дипломните задания по заглавие или дата на добавяне. Над списъка с дипломни задания се намира търсачка, чрез която студентът може да търси дипломни задания по част от заглавие и описание.

Дипломи

					сисТЕМА	Търси
#	Заглавие	Описание	Ръководител	Категории	Дата на добавяне	
1	Софтуерна система за разпределяне и съставяне на дипломни задания	Софтуерна система за автоматизация на процесите по разпределяне и съставяне на дипломни задания	ас. Георги Стоянов	HTML CSS C# ASP.NET MVC jQuery MYSQL	Sep 9 2016 2:53PM	

Фигура 5-4 Изглед на списъка с дипломни задания, от които да избира студентът

Когато диплома от таблицата е избрана и потребителят е препратен към подробните детайли за дипломата (фигура 5-5), той има опцията да избере дипломно задание чрез бутона „Избери“ или да се върне обратно. При избор на дипломно задание бива проверено дали потребителят е въвел факултетен номер, адрес, телефон. В случай, когато студентът не е попълнил информацията той бива помолен, чрез съобщение на същата страница, да го стори, за да може да избере дипломно задание. При успешно избиране на дипломна работа студентът бива препратен обратно към списъка с дипломни задания и на страницата се извежда съобщение за успех.

Детайли на дипломната работа

Ръководител	ас. Георги Стоянов
Тема	Софтуерна система за разпределяне и съставяне на дипломни задания
Описание	Софтуерна система за автоматизация на процесите по разпределяне и съставяне на дипломни задания
Съдържание	<ol style="list-style-type: none">1. Анализ на подобни системи. Цели и задачи.2. Обзор на използваните технологии.3. Проектиране на системата.4. Програмна реализация.5. Ръководство на потребителя.
Експериментална част	Програмен код, CD
Създадена	9.9.2016 г. 14:53:37
Последно променена	9.9.2016 г. 15:15:08
<div><button>Избери</button><button>Обратно</button></div>	

Фигура 5-5 Изглед на детайлите на желана дипломна работа и опция за избор

Ако студентът иска да види детайлите на избраната си диплома, той може да отиде на настройките на потребителя и да избере линка за детайли на дипломното си задание. На тази страница той може да изпраща съобщения към ръководителя си.

Когато студентът желае да избере диплома на конкретен ръководител или да прегледа дипломните задания, с които той/тя разполага, може да посети страницата с ръководители (фигура 5-6). Тя се състои от списък с ръководители, за всеки от които е изведена информация за име, e-mail и дисциплини (категориите на дипломните задания, с които разполага). Когато преподавател бъде избран от този списък, потребителят бива препратен към страница с детайлите на ръководителя (фигура 5-7). Тя включва телефонен номер и списък със свободни дипломни задания. При избор на дипломно задание, студентът бива препратен към детайлите на дипломата.

Преподаватели

Ръководител	Емайл	Дисциплини									
ас. Георги Стоянов	georgi@abv.bg	HTML	CSS	Java	JavaScript	CoffeeScript	SASS	C++	C	C#	
Константина Соколова	kosi@abv.bg	Java	C#								

Фигура 5-6 Част от изгледа със списък от преподаватели

Детайли за преподавателя

Ръководител ас. Георги Стоянов
Емайл georgi@abv.bg
Телефон 0888223344

#	Заглавие	Описание	Категории	Дата на добавяне
1	Разработка на софтуерни инструменти за електронно обучение чрез използването на портални приложения или отделни модули (темата е добавена 2009-10-02)	Разработка на софтуерни инструменти за електронно обучение	Java CoffeeScript SASS	9.9.2016 г. 14:34:28
2	Разработка на портални приложения или отделни модули	Разработка на портални приложения или отделни модули Разработка на портални приложения или отделни модули	CSS JavaScript C++	9.9.2016 г. 14:34:28
3	Разработка на софтуерни инструменти (системи, услуги) в електронното обучение	Разработка на софтуерни инструменти (системи, услуги) в електронното обучение (като цяло и най-вече в областта на тестването)	HTML Java CoffeeScript C	9.9.2016 г. 14:34:28

Фигура 5-7 Детайли на преподавател и дипломите, с които разполага

5.2 Ръководство за потребител тип ръководител

Когато потребител от тип ръководител се впише, той бива препратен към началната страница. В навигационната лента на потребителя освен списък с дипломни задания и списък с ръководители се намира и бутон за препратка към дипломните му задания. На тази страница ръководителят може да управлява всичките си дипломни задания. Тук потребителят може да създава, променя или изтрива дипломни задания. Визуализира таблица с вече създадени теми, включваща заглавие, описание, дата на добавяне и опции за промяна или изтриване, ако заданието е свободно или краен срок на заданието, ако е избрано от студент. В случай, че крайният срок е минал, се появява и допълнителен бутон, чрез който може да се освободи темата и тя да бъде маркирана като неизбрана от студента.

Нова

Кошче

Търся...

Търси

#	Заглавие	Описание	Дата на добавяне	
1	Инструменти за работа с цифрови хранилища	Проектиране и разработка на приложения, за работа с цифрови хранилища.	Sep 9 2016 2:34PM	<div>Промени</div> <div>Изтрий</div>
2	Обмен на данни и оперативна съвместимост между системи за електронно обучение	Темата/направлението включва запознаване и реализация/адаптация на някоя от популярните спецификации и стандарти за електронно обучение (като например портфолио, пакетирание на съдържанието, учебни обекти, стандарт за оценяване), голям набор от спецификации са налични на адрес: http://www.imsglobal.org/specificationdownload.cfm	Sep 9 2016 2:34PM	<div>9.9.2017 г.</div> <div>Освободи</div>
3	Разработка на RCP клиенти (plugin-s към NetBeans или Eclipse), бази данни, услуги, java сървърни приложения	Проектиране и разработка на приложения, базирани на софтуерни приставки	Sep 9 2016 2:34PM	<div>9.9.2017 г.</div> <div>Освободи</div>
4	Разработка на портални приложения или отделни модули	Разработка на портални приложения или отделни модули	Sep 9 2016 2:34PM	<div>Промени</div> <div>Изтрий</div>
5	Разработка на софтуерни инструменти за електронно обучение чрез използването на портални приложения или отделни модули (темата е добавена 2009-10-02)	Разработка на софтуерни инструменти за електронно обучение	Sep 9 2016 2:34PM	<div>Промени</div> <div>Изтрий</div>

Страница 1 от 2

1

2

»

Избрана от студент

Одобрена от ръководител

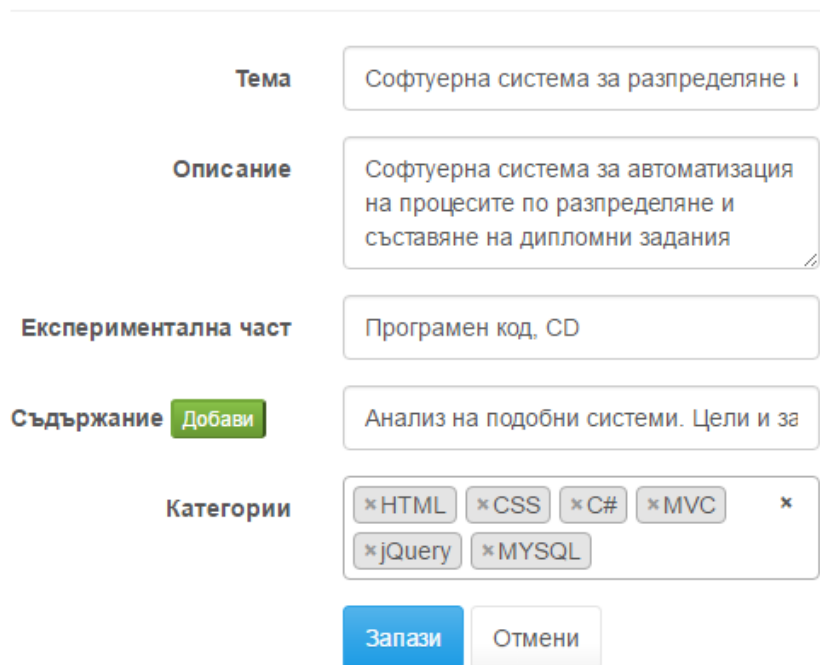
Одобрена и подписана от декан

Фигура 5-8 Изглед на списъка с дипломни работи на преподавателя

Редовете на таблицата се оцветяват в различни цветове в зависимост от статуса на дипломното задание. Ако е одобрено от ръководител, редът е син, ако е одобрено от декан – зелен. В случаите, когато дипломата е свободна, тя не се оцветява.

За да създаде ново дипломно задание, потребителят трябва да натисне бутона „Нова“, намиращ се в горния ляв ъгъл. Формата за създаване на дипломни задания се състои от полета за тема, описание, експериментална част, съдържание и технологии. До полето за съдържание се намира бутон „Добави“, позволяващ добавяне на нови редове към съдържанието. Технологиите представляват категории за дипломното задание. Те се въвеждат чрез динамично падащо меню. Ръководителят може да търси по име на категория и ако тя не съществува, то бива автоматично създадена. За да създаде дипломното задание, ръководителят трябва да натисне бутона „Запази“.

Създаване на диплома



The form is titled "Създаване на диплома" (Creating a diploma). It contains several input fields and buttons:

- Тема** (Topic): A text input field containing "Софтуерна система за разпределяне на ресурси" (Software system for resource distribution).
- Описание** (Description): A text input field containing "Софтуерна система за автоматизация на процесите по разпределяне и съставяне на дипломни задания" (Software system for automation of the processes of distribution and composition of diploma assignments).
- Експериментална част** (Experimental part): A text input field containing "Програмен код, CD" (Program code, CD).
- Съдържание** (Content): A text input field containing "Анализ на подобни системи. Цели и задачи" (Analysis of similar systems. Goals and tasks). To the left of this field is a green button labeled "Добави" (Add).
- Категории** (Categories): A list of tags including *HTML, *CSS, *C#, *MVC, *jQuery, and *MYSQL. There is also a small * icon to the right of the tags.
- At the bottom, there are two buttons: "Запази" (Save) in blue and "Отмени" (Cancel) in grey.

Фигура 5-9 Форма за създаване на диплома работа

Ако потребителят иска да изтрие задание, той може да го направи чрез бутона „Изтрий“, намиращ се в последната колонка на таблицата. При първото натискане на бутона, той се променя на „Изтрий?“ и чака потвърждение чрез второ натискане. Ако потребителят натисне някъде извън този бутон, той се връща в първоначалното си състояние.

Изтрите дипломни задания се преместват в „Кошче“. Достъпът до него се осъществява чрез натискане на бутона „Кошче“, намиращ се горе вляво до бутон „Нова“. Страницата представлява таблица с изтрети дипломни задания и съдържа информацията за дипломните задания и бутони „Върни“, който изважда дипломното задание от кошчето и „Изтрий?“, който изтрива дипломното задание от базата данни. В горната лява част на страницата се намират бутон за ново дипломно задание и бутон за изпразване на кошчето. Страницата съдържа и търсачка, търсеща по част от заглавие или описание.

За да промени свободно задание, ръководителят трябва да натисне бутона „Помени“, намиращ се в последната колона на таблицата с неговите дипломни задания. Потребителят е препратен към страница, подобна на тази за създаване на ново дипломно задание, като

полетата са попълнени с информацията за дадена диплома. За да запише промените, потребителят трябва да натисне бутонът „Запази“, а в противен случай – „Отмени“.

В горния десен ъгъл на страницата се намира търсачка, чрез която ръководителят може да търси дипломи по част от заглавие или описание. Търсачката позволява и търсене по ключови думи:

- „свободни“, в таблицата се появяват само свободните задания.
- „избрани“, на страницата се визуализират само избраните от студенти дипломни задания.
- „одобрени“ – само одобрените дипломни работи от ръководителя и декана задания.

Одобрението на дипломни задания става като преподавателят посети страницата за детайли на съответното дипломно задания. Това може да се направи като се селектира диплома от списък със задания. На страницата (фиг. 5-9) се показва информация за студента и за дипломното задание. Ако дипломата е избрана от студент и все още не е одобрена от преподавател, на страницата се изобразява бутон „Одобри“, както и поле за съобщение до студента в случай, когато трябва да се обсъди въпрос относно дипломното задание или данните на студента.

Информация за студента

Име	Стефан Николов Петров
Фак. номер	121314032
Email	steff.kn@abv.bg
Адрес	гр. Кюстендил, кв "Запад", бл. 71, ап 21
Телефон	0897018034

Информация за дипломната работа

Тема	Софтуерна система за разпределяне и съставяне на дипломни задания
Описание	Софтуерна система за автоматизация на процесите по разпределяне и съставяне на дипломни задания
Съдържание	1. Анализ на подобни системи. Цели и задачи. 2. Обзор на използваните технологии. 3. Проектиране на системата. 4. Програмна реализация. 5. Ръководство на потребителя.
Експериментална част	Програмен код, CD
Категории	HTML CSS C# ASP.NET MVC jQuery MYSQL
Създадена	9.9.2016 г. 14:53:37
Последно променена	9.9.2016 г. 15:16:12
Избрана от студент	Избрана
Одобрена от ръководител	Не одобрена
Одобрена от канцелария	Не одобрена
<div><button>Одобри</button><button>Обратно</button></div>	

Съобщение за не одобрение

Фигура 5-10 Изглед на страницата за детайли на избрано дипломно задание

След одобрение, на страница за детайли срещу поле „Одобрена от ръководител“ се изписва „Одобрена“ и бутона за одобрение бива скрит.

5.3 Ръководство за потребител тип администратор

Когато администраторът се впише, той се препраща към началната страница. В навигационната му лента се появява бутон за администрация на потребителите, който препраща към страница с таблица с всички потребители (фигура 5-10). Данните, които се визуализират за потребителите са име, фамилия, e-mail, телефон, потребителско име, права. Над списъка се намира търсачка, която позволява търсене на потребител по всички полета.

						<input type="text" value="Търся..."/> <input type="button" value="Търси"/>
Име	Фамилия	Емейл	Телефон	Псевдоним	Права	
Admin	Admin	admin@admin.com		admin@admin.com	Administrator	<input type="button" value="Промени"/>
Александър	Симеонов	a.simeonov@abv.bg	0800123456	sasho	Student	<input type="button" value="Промени"/>
Божидаря	Таскова	boji@abv.bg		boji	Student	<input type="button" value="Промени"/>
Георги	Стойанов	georgi@abv.bg		georgi	Teacher	<input type="button" value="Промени"/>
Драгомир	Тодоров	drago@abv.bg		drago	Student	<input type="button" value="Промени"/>
Константина	Соколова	kosi@abv.bg		kosi	Teacher	<input type="button" value="Промени"/>

Фигура 5-11 Част от списък с потребители, които администратора може да променя

До всеки запис се намира бутон „Промени“, който препраща към страница за промяна на избрания потребител. Страницата за промяна включва полета за име, фамилия, e-mail и права, които се попълват автоматично с текущите данни на избрания потребител.

Промяна на правата на потребителя.

Име
Александър

Фамилия
Симеонов

Емейл
a.simeonov@abv.bg

Права

Student ▼

Фигура 5-12 Формна за промяна на правата на потребител

Потребителите от тази роля разполагат с опция за крайно утвърждение на дипломни задания. Одобрението на заданието става аналогично на процеса при ръководителите. Отивайки на страницата на детайлите на дипломното задание, администратора може да види информация за студента и дипломното задание. Там потребителят може да избере да одобри темата чрез натискане на бутон „Одобри“.

Избрана от студент	Избрана
Одобрена от ръководител	Одобрена
Одобрена от канцелария	Не одобрена
<input type="button" value="Одобри"/> <input type="button" value="Обратно"/>	

Фигура 5-13 Бутон за одобрение на диплома от администратор

Ако съществуват заглавия близки до избраната тема, те ще бъдат изредени под информацията за дипломата, като списък от линкове, препращащи към детайлите за съответната диплома. В този списък фигурират само одобрени от ръководителите теми.

Информация за дипломната работа

Ръководител	Константина Соколова
Тема	Софтуерна система за социални Web 2.0 инструменти
Описание	Темата/направлението включва разработването на софтуерни инструменти с използване на java сървърни технологии.
Съдържание	1. Проектиране 2. Web 2.0 3. Приложения 4. Обучение
Експериментална част	Проектиране
Категории	CSS ASP.NET MVC jQuery
Създадена	11.9.2016 г. 16:34:28
Последно променена	11.9.2016 г. 16:38:51
Избрана от студент	Избрана
Одобрена от ръководител	Одобрена
Одобрена от канцелария	Не одобрена
<input type="button" value="Одобри"/> <input type="button" value="Обратно"/>	

[Разработване на социални Web 2.0 инструменти](#)

[Разработване на социални Web 1.0 инструменти](#)

Activate V
Get PDF

Фигура 5-14 Част от страницата за одобрение от администратор, показваща дипломи със сходни имена

Когато потребителят е приключил и желае да напусне приложението, той може да натисне бутона „Изход“, който ще го отпише.

Заклучение

В настоящата дипломна работа беше разгледана система за съставяне и одобрение на дипломни задания, която има за цел автоматизация на процеса. Решението е изградено, като са използвани всички съображения заложи в глава 1 и глава 3. Изграденото web приложение е разделено на логически слоеве, които позволяват разширение и лесна заменяемост. Също така приложението е съставено от отделни проекти. Тъй като в текущата дипломна работа е обърнато по-голямо внимание на функционалната и инженерната част, дизайнът на сайта е сравнително прост и използва Bootstrap.

Цялата логика по извличане на информация от базата данни и създаването на нови модели е изнесена в отделен проект. За достъп до тези данни се използва хранилище. Използването на шаблон repository позволява заменяемост на източника на данни, като единственото нещо, което трябва да се направи е създаването на нов клас от тип хранилище. Всички услуги са разположени в слой на услугите и са отделени в собствен проект. Добавянето на услуги изисква само добавянето на нов клас в този проект.

Основната цел на изграденото приложение е то да бъде лесно достъпно и всички операции да се извършват бързо. Функционалности като избиране на дипломно задание от студента, одобряване на дипломно задание от ръководителя и декан се извършват с натискането на един бутон.

В текущото дипломно решение не са разгледани методи за подписване на заданията. Поради модулността на приложението тази функционалност може да бъде лесно добавена в случай, че приложението трябва да бъде разширено.

Списък на използвани литературни източници

1. ASP.NET MVC - [https://msdn.microsoft.com/en-us/library/dd381412\(v=vs.108\).aspx](https://msdn.microsoft.com/en-us/library/dd381412(v=vs.108).aspx)
2. C# Programming Guide - <https://msdn.microsoft.com/en-us/library/67ef8sbd.aspx>
3. Официален сайт на Технически университет – София, ФКСУ- <http://cs.tu-sofia.bg/>
4. Официален сайт на jQuery - <https://jquery.com/>
5. CSS - Cascading Style Sheets на сайта на World Wide Web Consortium - <https://www.w3.org/Style/CSS/>
6. Статия „ASP.NET MVC Solution Architecture – Best Practices“ – <https://chsakell.com/2015/02/15/asp-net-mvc-solution-architecture-best-practices/>
7. Статия „C#: Calculating Percentage Similarity of 2 strings“ - <http://social.technet.microsoft.com/wiki/contents/articles/26805.c-calculating-percentage-similarity-of-2-strings.aspx>
8. Itzik Ben-Gan, Microsoft SQL Server 2012 T-SQL Fundamentals, Microsoft Press 2012

Приложения

1. Модели

1.1 DDS.Data.Common

BaseModel{TKey}.cs

```
namespace DDS.Data.Common.Models
{
    using System;
    using System.ComponentModel.DataAnnotations;
    using System.ComponentModel.DataAnnotations.Schema;

    public abstract class BaseModel<TKey> : IAuditInfo, IDeletableEntity
    {
        [Key]
        public TKey Id { get; set; }

        public DateTime CreatedOn { get; set; }

        public DateTime? ModifiedOn { get; set; }

        [Index]
        public bool IsDeleted { get; set; }

        public DateTime? DeletedOn { get; set; }
    }
}
```

IAuditInfo.cs

```
namespace DDS.Data.Common.Models
{
    using System;

    public interface IAuditInfo
    {
        DateTime CreatedOn { get; set; }

        DateTime? ModifiedOn { get; set; }
    }
}
```

IDeletableEntity.cs

```
namespace DDS.Data.Common.Models
{
    using System;

    public interface IDeletableEntity
    {
        bool IsDeleted { get; set; }

        DateTime? DeletedOn { get; set; }
    }
}
```

```
}
```

IDbRepository{T}.cs

```
namespace DDS.Data.Common
{
    using System.Linq;

    using DDS.Data.Common.Models;

    public interface IDbRepository<T> : IDbRepository<T, int>
        where T : BaseModel<int>
    {
    }

    public interface IDbRepository<T, in TKey>
        where T : BaseModel<TKey>
    {
        IQueryable<T> All();

        IQueryable<T> AllWithDeleted();

        T GetById(TKey id);

        void Add(T entity);

        void Delete(T entity);

        void UnDelete(T entity);

        void HardDelete(T entity);

        void Save();
    }
}
```

DbRepository{T}.cs

```
namespace DDS.Data.Common
{
    using System;
    using System.Data.Entity;
    using System.Linq;

    using DDS.Data.Common.Models;

    // TODO: Why BaseModel<int> instead BaseModel<TKey>?
    public class DbRepository<T> : IDbRepository<T>
        where T : BaseModel<int>
    {
        public DbRepository(DbContext context)
        {
            if (context == null)
            {
                throw new ArgumentException("An instance of DbContext is required to use this repository.", nameof(context));
            }
        }
    }
}
```

```

        this.Context = context;
        this.DbSet = this.Context.Set<T>();
    }

    private IDbSet<T> DbSet { get; }

    private DbContext Context { get; }

    public IQueryable<T> All()
    {
        return this.DbSet.Where(x => !x.IsDeleted);
    }

    public IQueryable<T> AllWithDeleted()
    {
        return this.DbSet;
    }

    public T GetById(int id)
    {
        return this.All().FirstOrDefault(x => x.Id == id);
    }

    public void Add(T entity)
    {
        this.DbSet.Add(entity);
    }

    public void Delete(T entity)
    {
        entity.IsDeleted = true;
        entity.DeletedOn = DateTime.Now;
    }

    public void UnDelete(T entity)
    {
        entity.IsDeleted = false;
        entity.DeletedOn = null;
    }

    public void HardDelete(T entity)
    {
        this.DbSet.Remove(entity);
    }

    public void Save()
    {
        this.Context.SaveChanges();
    }
}

```

1.2 DDS.Data.Models

ApplicationUser.cs

```
namespace DDS.Data.Models
{
    using System.ComponentModel.DataAnnotations;
    using System.Security.Claims;
    using System.Threading.Tasks;

    using Microsoft.AspNet.Identity;
    using Microsoft.AspNet.Identity.EntityFramework;

    public class ApplicationUser : IdentityUser
    {
        [Required(ErrorMessage = "Моля въведете име.")]
        public string FirstName { get; set; }

        [Required(ErrorMessage = "Моля въведете презиме.")]
        public string LastName { get; set; }

        [Required(ErrorMessage = "Моля въведете фамилия.")]
        public string MiddleName { get; set; }

        public string ScienceDegree { get; set; }

        // what will happen with Ph.D. (maybe adding in future) this will be better
    }
}

solution
    public Student Student { get; set; }

    public Teacher Teacher { get; set; }

    public async Task<ClaimsIdentity>
GenerateUserIdentityAsync(UserManager<ApplicationUser> manager)
    {
        var userIdentity = await manager.CreateIdentityAsync(this,
DefaultAuthenticationTypes.ApplicationCookie);
        return userIdentity;
    }
}
```

Student.cs

```
namespace DDS.Data.Models
{
    using DDS.Data.Common.Models;

    public class Student : BaseModel<int>
    {
        public int FNumber { get; set; }
        public string Address { get; set; }
        public ApplicationUser User { get; set; }
        public Diploma SelectedDiploma { get; set; }
        public bool IsGraduate { get; set; }
    }
}
```

Diploma.cs

```
namespace DDS.Data.Models
{
    using System.Collections.Generic;
    using System.ComponentModel.DataAnnotations;
    using System.ComponentModel.DataAnnotations.Schema;
    using DDS.Data.Common.Models;

    [Table("Diplomas")]
    public class Diploma : BaseModel<int>
    {
        [Required(ErrorMessage = "Заглавието е задължително")]
        [MinLength(40, ErrorMessage = "Заглавието трябва да бъде поне 40 символа.")]
        public string Title { get; set; }

        [Required(ErrorMessage = "Описанието е задължително")]
        public string Description { get; set; }

        [Required(ErrorMessage = "Експерименталната част е задължителна")]
        public string ExperimentalPart { get; set; }

        public string ContentCSV { get; set; }

        public virtual ICollection<Tag> Tags { get; set; }

        public int TeacherID { get; set; }

        [ForeignKey("TeacherID")]
        public virtual Teacher Teacher { get; set; }

        public bool IsApprovedByLeader { get; set; }

        public bool IsApprovedByHead { get; set; }

        public bool IsSelectedByStudent { get; set; }
    }
}
```

Tag.cs

```
namespace DDS.Data.Models
{
    using System.Collections.Generic;

    using DDS.Data.Common.Models;

    public class Tag : BaseModel<int>
    {
        public string Name { get; set; }

        public virtual ICollection<Diploma> Diplomas { get; set; }

        public virtual ICollection<Teacher> Teachers { get; set; }
    }
}
```

Teacher.cs

```
namespace DDS.Data.Models
{
    using System.Collections.Generic;

    using Common.Models;

    public class Teacher : BaseModel<int>
    {
        public Teacher()
        {
            this.Students = new HashSet<Student>();
            this.Diplomas = new HashSet<Diploma>();
            this.Tags = new HashSet<Tag>();
        }

        public ApplicationUser User { get; set; }

        public ICollection<Student> Students { get; set; }

        public ICollection<Diploma> Diplomas { get; set; }

        public virtual ICollection<Tag> Tags { get; set; }
    }
}
```

Message.cs

```
namespace DDS.Data.Models
{
    using System.ComponentModel.DataAnnotations;
    using DDS.Data.Common.Models;

    public class Message : BaseModel<int>
    {
        public string SenderUserId { get; set; }

        public string ResieverUserId { get; set; }

        public Diploma SelectedDiploma { get; set; }

        [Required]
        public string MessageSend { get; set; }

        public bool IsRead { get; set; }
    }
}
```

1.3 DDS.Data

ApplicationDbContext.cs

```
namespace DDS.Data
{
    using System;
    using System.Data.Entity;
    using System.Data.Entity.Validation;
    using System.Linq;
    using Common.Models;
    using DDS.Data.Models;
    using Microsoft.AspNet.Identity.EntityFramework;

    public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
    {
        public ApplicationDbContext()
            : base("DefaultConnection", throwIfV1Schema: false)
        {
        }

        public IDbSet<Student> Students { get; set; }

        public IDbSet<Teacher> Teachers { get; set; }

        public IDbSet<Diploma> Diplomas { get; set; }

        public IDbSet<Message> Messages { get; set; }

        public IDbSet<Tag> Tags { get; set; }

        public static ApplicationDbContext Create()
        {
            return new ApplicationDbContext();
        }

        public override int SaveChanges()
        {
            this.ApplyAuditInfoRules();
            try
            {
                return base.SaveChanges();
            }
            catch (DbEntityValidationException ex)
            {
                // Retrieve the error messages as a list of strings.
                var errorMessages = ex.EntityValidationErrors
                    .SelectMany(x => x.ValidationErrors)
                    .Select(x => x.ErrorMessage);

                // Join the list to a single string.
                var fullErrorMessage = string.Join("; ", errorMessages);

                // Combine the original exception message with the new one.
                var exceptionMessage = string.Concat(ex.Message, " The validation errors
are: ", fullErrorMessage);
            }
        }
    }
}
```

```

        // Throw a new DbEntityValidationException with the improved exception
message.
        throw new DbEntityValidationException(exceptionMessage,
ex.EntityValidationErrors);
    }
}

protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);

    // configure Student
modelBuilder.Entity<Student>()
    .HasKey(s => s.Id);

    // Map one-to-zero or one relationship
modelBuilder.Entity<Student>()
    .HasRequired(s => s.User)
    .WithOptional(u => u.Student);

    // configure Diploma
modelBuilder.Entity<Diploma>()
    .HasKey(s => s.Id);

    // Map one-to-many relationship
modelBuilder.Entity<Diploma>()
    .HasRequired<Teacher>(d => d.Teacher)
    .WithMany(t => t.Diplomas);

    // configure Teacher
modelBuilder.Entity<Teacher>()
    .HasKey(s => s.Id);

    // Map one-to-zero or one relationship
modelBuilder.Entity<Teacher>()
    .HasRequired(t => t.User)
    .WithOptional(u => u.Teacher);

modelBuilder.Entity<Teacher>()
    .HasMany<Student>(s => s.Students);

modelBuilder.Entity<Teacher>()
    .HasMany<Diploma>(s => s.Diplomas);
}

private void ApplyAuditInfoRules()
{
    // Approach via @julielerman: http://bit.ly/123661P
    foreach (var entry in
        this.ChangeTracker.Entries()
            .Where(
                e =>
                    e.Entity is IAuditInfo && ((e.State == EntityState.Added) ||
(e.State == EntityState.Modified))))
    {
        var entity = (IAuditInfo)entry.Entity;
        if (entry.State == EntityState.Added && entity.CreatedOn ==
default(DateTime))
    }
}

```



```

        {
            entity.CreatedOn = DateTime.Now;
        }
        else
        {
            entity.ModifiedOn = DateTime.Now;
        }
    }
}
}
}
}

```

Глава 2. Услуги

2.1 DDS.Services.Data.Interfaces

IadministrationService.cs

```

namespace DDS.Services.Data.Interfaces
{
    using System.Linq;
    using DDS.Data.Models;

    public interface IAdministrationService
    {
        IQueryable<ApplicationUser> GetAll();

        ApplicationUser GetById(string userId);

        IQueryable<Student> GetAllStudents();

        IQueryable<Student> GetAllStudentsByCreateDate();

        IQueryable<Teacher> GetAllTeachers();

        IQueryable<Teacher> GetAllTeachersByCreateDate();

        void DeleteTeacher(Teacher entity);

        void CreateTeacher(Teacher entity);

        void EditTeacher(Teacher entity);

        void DeleteStudent(Student entity);

        void CreateStudent(Student entity);

        void EditStudent(Student entity);
    }
}

```

IBaseServices.cs

```

namespace DDS.Services.Data.Interfaces
{
    using System.Linq;

```

```

using DDS.Data.Common.Models;

public interface IBaseServices<T>
    where T : BaseModel<int>
{
    void Create(T entity);

    void Delete(T entity);

    void UnDelete(T entity);

    void HardDelete(T entity);

    void Edit(T entity);

    void Save();

    T GetObjectById(int id);

    IQueryable<T> GetAll();

    IQueryable<T> GetDeleted();

    IQueryable<T> GetById(int id);
}
}

```

IDiplomasService.cs

```

namespace DDS.Services.Data.Interfaces
{
    using System.Linq;
    using System.Threading.Tasks;
    using DDS.Data.Models;

    public interface IDiplomasService : IBaseServices<Diploma>
    {
        IQueryable<Diploma> GetByTeacherId(int id);

        IQueryable<Diploma> GetRandomDiplomas(int count);
    }
}

```

ITagsService.cs

```

namespace DDS.Services.Data.Interfaces
{
    using DDS.Data.Models;

    public interface ITagsService : IBaseServices<Tag>
    {
        Tag EnsureCategory(string name);
    }
}

```

ITeachersService.cs

```

namespace DDS.Services.Data.Interfaces

```

```

{
    using System.Collections.Generic;
    using System.Linq;
    using DDS.Data.Models;

    public interface ITeachersService : IBaseServices<Teacher>
    {
        IQueryable<Teacher> GetByUserId(string id);

        IEnumerable<Diploma> GetAllDiplomas(int id);

        void AddDiploma(int teacherID, Diploma entity);

        void AddStudent(int teacherID, Student entity);
    }
}

```

IStudentsService.cs

```

namespace DDS.Services.Data.Interfaces
{
    using System.Linq;
    using DDS.Data.Models;

    public interface IStudentsService : IBaseServices<Student>
    {
        IQueryable<Student> GetByFNumber(int number);

        IQueryable<Student> GetByUserId(string userId);

        IQueryable<Student> GetStudentWithSelectedDiplomaByUserID(string userId);

        IQueryable<Student> GetStudentWithSelectedDiploma(int studentId);

        void AddDiploma(int studentId, Diploma entity);

        void RemoveDiploma(int studentId);
    }
}

```

IMessagesService.cs

```

namespace DDS.Services.Data.Interfaces
{
    using DDS.Data.Models;

    public interface IMessagesService : IBaseServices<Message>
    {
    }
}

```

2.2 DDS.Services.Data

BaseService.cs

```

namespace DDS.Services.Data
{
    using System;

```

```

using System.Linq;

using DDS.Data.Common;
using DDS.Data.Common.Models;
using Interfaces;

public class BaseService<T> : IBaseServices<T>
    where T : BaseModel<int>
{
    private IDbRepository<T> items;

    public BaseService(IDbRepository<T> items)
    {
        this.items = items;
    }

    internal IDbRepository<T> Items
    {
        get
        {
            return this.items;
        }
    }

    /// <summary>
    /// Marks entity of type BaseModel<int> as deleted. Calls Items.Save() method!
    /// </summary>
    /// <param name="entity">Given entity of type BaseModel</param>
    public void Delete(T entity)
    {
        this.Items.Delete(entity);
        this.Items.Save();
    }

    public void UnDelete(T entity)
    {
        this.Items.UnDelete(entity);
        this.Items.Save();
    }

    public void HardDelete(T entity)
    {
        this.Items.HardDelete(entity);
        this.Items.Save();
    }

    /// <summary>
    /// Adds entity of type BaseModel<int>. Calls Items.Save() method!
    /// </summary>
    /// <param name="entity">Given entity of type BaseModel<int></param>
    public void Create(T entity)
    {
        this.Items.Add(entity);
        this.Items.Save();
    }

    /// <summary>

```

```

    /// Edits the date and time modified and deleted as well as the IsDeleted property
for the entity. Calls Items.Save() method!
    /// </summary>
    /// <param name="entity">Given entity of type BaseModel<int></param>
    public virtual void Edit(T entity)
    {
        this.Items.GetById(entity.Id).ModifiedOn = DateTime.Now;
        this.Items.GetById(entity.Id).IsDeleted = entity.IsDeleted;
        if (entity.IsDeleted)
        {
            this.Items.GetById(entity.Id).DeletedOn = DateTime.Now;
        }
        else
        {
            this.Items.GetById(entity.Id).DeletedOn = null;
        }

        this.Items.Save();
    }

    /// <summary>
    /// Returns entity of type <paramref name="T"/>
    /// </summary>
    /// <param name="id">id of the entity to look for</param>
    /// <returns>Entity of type <paramref name="T"/></returns>
    public T GetObjectById(int id)
    {
        return this.Items.GetById(id);
    }

    public IQueryable<T> GetById(int id)
    {
        return this.Items.All().Where(d => d.Id == id);
    }

    /// <summary>
    /// Returns entities of type <paramref name="T"/>
    /// </summary>
    /// <returns>IQueryable of type <paramref name="T"/></returns>
    public IQueryable<T> GetAll()
    {
        return this.Items.All();
    }

    /// <summary>
    /// Returns entities of type <paramref name="T"/>
    /// </summary>
    /// <returns>IQueryable of type <paramref name="T"/></returns>
    public IQueryable<T> GetDeleted()
    {
        return this.Items.AllWithDeleted().Where(x => x.IsDeleted == true);
    }

    public void Save()
    {
        this.Items.Save();
    }
}

```

```
}
```

AdministrationService.cs

```
namespace DDS.Services.Data
{
    using System;
    using System.Linq;
    using DDS.Data;
    using DDS.Data.Common;
    using DDS.Data.Models;
    using Interfaces;

    public class AdministrationService : IAdministrationService
    {
        private ApplicationDbContext db = new ApplicationDbContext();
        private IDbRepository<Student> students;
        private IDbRepository<Teacher> teachers;

        public AdministrationService()
        {
            this.students = new DbRepository<Student>(this.db);
            this.teachers = new DbRepository<Teacher>(this.db);
        }

        public IQueryable<ApplicationUser> GetAll()
        {
            var allUsers = this.db.Users.OrderBy(u => u.FirstName).ThenBy(u =>
u.LastName);

            return allUsers;
        }

        public ApplicationUser GetById(string userId)
        {
            var user = this.db.Users.Find(userId);

            return user;
        }

        public IQueryable<Student> GetAllStudents()
        {
            var allStudents = this.db.Students.OrderBy(s => s.User.FirstName).ThenBy(s =>
s.User.LastName);
            return allStudents;
        }

        public IQueryable<Student> GetAllStudentsByCreateDate()
        {
            var allStudents = this.db.Students.OrderBy(s => s.CreatedOn);
            return allStudents;
        }

        public IQueryable<Teacher> GetAllTeachers()
        {
            var allTeachers = this.db.Teachers.OrderBy(t => t.User.FirstName).ThenBy(t =>
t.User.LastName);
            return allTeachers;
        }
    }
}
```

```

    }

    public IQueryable<Teacher> GetAllTeachersByCreateDate()
    {
        var allTeachers = this.db.Teachers.OrderBy(t => t.CreatedOn);
        return allTeachers;
    }

    public void DeleteTeacher(Teacher entity)
    {
        entity.IsDeleted = true;
        entity.DeletedOn = DateTime.Now;
        this.db.SaveChanges();
    }

    public void CreateTeacher(Teacher entity)
    {
        this.db.Teachers.Add(entity);
        this.db.SaveChanges();
    }

    public void EditTeacher(Teacher entity)
    {
        this.db.Teachers.FirstOrDefault(t => t.Id == entity.Id).DeletedOn =
entity.DeletedOn;
        this.db.Teachers.FirstOrDefault(t => t.Id == entity.Id).IsDeleted =
entity.IsDeleted;
        this.db.Teachers.FirstOrDefault(t => t.Id == entity.Id).ModifiedOn =
entity.ModifiedOn;
        this.db.SaveChanges();
    }

    public void DeleteStudent(Student entity)
    {
        entity.IsDeleted = true;
        entity.DeletedOn = DateTime.Now;
        this.db.SaveChanges();
    }

    public void CreateStudent(Student entity)
    {
        this.db.Students.Add(entity);
        this.db.SaveChanges();
    }

    public void EditStudent(Student entity)
    {
        this.db.Students.FirstOrDefault(t => t.Id == entity.Id).ModifiedOn =
DateTime.Now;
        this.db.SaveChanges();
    }
}
}

```

DiplomasService.cs

```

namespace DDS.Services.Data
{

```

```

using System;
using System.Data.Entity;
using System.Linq;
using DDS.Data.Common;
using DDS.Data.Models;
using Interfaces;

public class DiplomasService : BaseService<Diploma>, IDiplomasService
{
    public DiplomasService(IDbRepository<Diploma> diplomas)
        : base(diplomas)
    {}

    public IQueryable<Diploma> GetByTeacherId(int id)
    {
        return this.Items.All().Where(d => d.TeacherID == id);
    }

    public IQueryable<Diploma> GetRandomDiplomas(int count)
    {
        return this.Items.All().OrderBy(x => Guid.NewGuid()).Take(count);
    }
}

```

StudentsService.cs

```

namespace DDS.Services.Data
{
    using System.Data.Entity;
    using System.Linq;
    using Data.Interfaces;
    using DDS.Data.Common;
    using DDS.Data.Models;

    public class StudentsService : BaseService<Student>, IStudentsService
    {
        public StudentsService(IDbRepository<Student> students)
            : base(students)
        {}

        public IQueryable<Student> GetByFNumber(int number)
        {
            return this.Items.All().Where(s => s.FNumber == number);
        }

        public IQueryable<Student> GetByUserId(string userId)
        {
            return this.Items.All().Where(t => t.User.Id == userId);
        }

        public void AddDiploma(int studentId, Diploma entity)
        {
            this.Items.GetById(studentId).SelectedDiploma = entity;
            this.Items.GetById(studentId).SelectedDiploma.Id = entity.Id;
        }

        public void RemoveDiploma(int studentId)

```



```

        {
            this.Items.GetById(studentId).SelectedDiploma = null;
        }

        public IQueryable<Student> GetStudentWithSelectedDiplomaByUserID(string userId)
        {
            return this.Items.All().Include(s => s.SelectedDiploma).Where(t => t.User.Id
== userId);
        }
        public IQueryable<Student> GetStudentWithSelectedDiploma(int studentId)
        {
            return this.Items.All().Include(s => s.SelectedDiploma).Where(t => t.Id ==
studentId);
        }
    }
}

```

TagsService.cs

```

namespace DDS.Services.Data
{
    using System.Linq;
    using DDS.Data.Common;
    using DDS.Data.Models;
    using Interfaces;

    public class TagsService : BaseService<Tag>, ITagsService
    {
        public TagsService(IDbRepository<Tag> tags)
            : base(tags)
        {}

        public Tag EnsureCategory(string name)
        {
            var tag = this.Items.All().FirstOrDefault(x => x.Name == name);
            if (tag != null)
            {
                return tag;
            }

            tag = new Tag { Name = name };
            this.Items.Add(tag);
            this.Items.Save();
            return tag;
        }
    }
}

```

TeachersService.cs

```

namespace DDS.Services.Data
{
    using System.Collections.Generic;
    using System.Data.Entity;
    using System.Linq;
    using Data.Interfaces;
    using DDS.Data.Common;
    using DDS.Data.Models;

```

```

public class TeachersService : BaseService<Teacher>, ITeachersService
{
    public TeachersService(IDbRepository<Teacher> teachers)
        : base(teachers)
    {
    }

    public IQueryable<Teacher> GetByUserId(string id)
    {
        return this.Items.All().Where(t => t.User.Id == id);
    }

    public IEnumerable<Diploma> GetAllDiplomas(int id)
    {
        var teacher = this.Items.All()
            .Where(d => d.Id == id)
            .Include(e => e.Diplomas);
        return teacher.First().Diplomas;
    }

    public void AddDiploma(int teacherID, Diploma entity)
    {
        this.Items.GetById(teacherID).Diplomas.Add(entity);
        this.Items.Save();
    }

    public void AddStudent(int teacherID, Student entity)
    {
        this.Items.GetById(teacherID).Students.Add(entity);
        this.Items.Save();
    }
}

```

2.3 DDS.Services.Web

ICacheService.cs

```

namespace DDS.Services.Web
{
    using System;

    public interface ICacheService
    {
        T Get<T>(string itemName, Func<T> getDataFunc, int durationInSeconds);

        void Remove(string itemName);
    }
}

```

HttpCacheService.cs

```

namespace DDS.Services.Web
{
    using System;
    using System.Web;
}

```

```

using System.Web.Caching;

public class HttpCacheService : ICacheService
{
    private static readonly object LockObject = new object();

    public T Get<T>(string itemName, Func<T> getDataFunc, int durationInSeconds)
    {
        if (HttpRuntime.Cache[itemName] == null)
        {
            lock (LockObject)
            {
                if (HttpRuntime.Cache[itemName] == null)
                {
                    var data = getDataFunc();
                    HttpRuntime.Cache.Insert(
                        itemName,
                        data,
                        null,
                        DateTime.Now.AddSeconds(durationInSeconds),
                        Cache.NoSlidingExpiration);
                }
            }
        }

        return (T)HttpRuntime.Cache[itemName];
    }

    public void Remove(string itemName)
    {
        HttpRuntime.Cache.Remove(itemName);
    }
}

```

Глава 3. Основна част на приложението

3.1 DDS.Common

GlobalConstants.cs

```

namespace DDS.Common
{
    public class GlobalConstants
    {
        public const string AdministratorRoleName = "Administrator";
        public const string StudentRoleName = "Student";
        public const string TeacherRoleName = "Teacher";
        public const string ManagementRoleName = "Management";
        public const int PageSize = 5;
        public const char ContentSeparator = ',';
    }
}

```

3.2 DDS.Web

Startup.cs

```
using Microsoft.Owin;

using Owin;

[assembly: OwinStartupAttribute(typeof(DDS.Web.Startup))]

namespace DDS.Web
{
    public partial class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            this.ConfigureAuth(app);
        }
    }
}
```

Global.asax

```
namespace DDS.Web
{
    using System.Data.Entity;
    using System.Reflection;
    using System.Web;
    using System.Web.Mvc;
    using System.Web.Optimization;
    using System.Web.Routing;

    using Data;
    using Data.Migrations;

    using Infrastructure.Mapping;

    #pragma warning disable SA1649 // File name must match first type name
    public class MvcApplication : HttpApplication
    #pragma warning restore SA1649 // File name must match first type name
    {
        protected void Application_Start()
        {
            ViewEngines.Engines.Clear();
            ViewEngines.Engines.Add(new RazorViewEngine());

            Database.SetInitializer(new
MigrateDatabaseToLatestVersion<ApplicationDbContext, Configuration>());
            AutofacConfig.RegisterAutofac();
            AreaRegistration.RegisterAllAreas();
            FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
            RouteConfig.RegisterRoutes(RouteTable.Routes);
            BundleConfig.RegisterBundles(BundleTable.Bundles);

            var autoMapperConfig = new AutoMapperConfig();
            autoMapperConfig.Execute(Assembly.GetExecutingAssembly());
        }
    }
}
```

```
}  
}
```

3.2.1 App_Start

ApplicationRoleManager.cs

```
namespace DDS.Web.App_Start  
{  
    using Data;  
    using Microsoft.AspNet.Identity;  
    using Microsoft.AspNet.Identity.EntityFramework;  
    using Microsoft.AspNet.Identity.Owin;  
    using Microsoft.Owin;  
  
    public class ApplicationRoleManager : RoleManager<IdentityRole>  
    {  
        public ApplicationRoleManager(IRoleStore<IdentityRole, string> roleStore)  
            : base(roleStore)  
        {  
        }  
  
        public static ApplicationRoleManager Create(  
            IdentityFactoryOptions<ApplicationRoleManager> options, IOwinContext context)  
        {  
            return new ApplicationRoleManager(  
                new RoleStore<IdentityRole>(context.Get<ApplicationDbContext>()));  
        }  
    }  
}
```

ApplicationSignInManager.cs

```
namespace DDS.Web  
{  
    using System.Security.Claims;  
    using System.Threading.Tasks;  
    using DDS.Data.Models;  
    using Microsoft.AspNet.Identity.Owin;  
    using Microsoft.Owin;  
    using Microsoft.Owin.Security;  
  
    public class ApplicationSignInManager : SignInManager<ApplicationUser, string>  
    {  
        public ApplicationSignInManager(ApplicationUserManager userManager,  
            IAuthenticationManager authenticationManager)  
            : base(userManager, authenticationManager)  
        {  
        }  
  
        public static ApplicationSignInManager  
        Create(IdentityFactoryOptions<ApplicationSignInManager> options, IOwinContext context)  
        {  
            return new  
            ApplicationSignInManager(context.GetUserManager<ApplicationUserManager>(),  
            context.Authentication);  
        }  
    }  
}
```

```

    }

    public override Task<ClaimsIdentity> CreateUserIdentityAsync(ApplicationUser user)
    {
        return
user.GenerateUserIdentityAsync((ApplicationUserManager)this.UserManager);
    }
}

```

ApplicationUserManager.cs

```

namespace DDS.Web
{
    using System;
    using DDS.Data;
    using DDS.Data.Models;
    using Microsoft.AspNet.Identity;
    using Microsoft.AspNet.Identity.EntityFramework;
    using Microsoft.AspNet.Identity.Owin;
    using Microsoft.Owin;

    // Configure the application user manager used in this application. UserManager is
    // defined in ASP.NET Identity and is used by the application.
    public class ApplicationUserManager : UserManager<ApplicationUser>
    {
        public ApplicationUserManager(IUserStore<ApplicationUser> store)
            : base(store)
        {
        }

        public static ApplicationUserManager
Create(IdentityFactoryOptions<ApplicationUserManager> options, IOwinContext context)
        {
            var manager = new ApplicationUserManager(
                new UserStore<ApplicationUser>(context.Get<ApplicationDbContext>()));

            // Configure validation logic for usernames
            manager.UserValidator = new UserValidator<ApplicationUser>(manager)
            {
                AllowOnlyAlphanumericUserNames = false,
                RequireUniqueEmail = true
            };

            // Configure validation logic for passwords
            manager.PasswordValidator = new PasswordValidator
            {
                RequiredLength = 6,
                RequireNonLetterOrDigit = true,
                RequireDigit = true,
                RequireLowercase = true,
                RequireUppercase = true,
            };

            // Configure user lockout defaults
            manager.UserLockoutEnabledByDefault = true;
            manager.DefaultAccountLockoutTimeSpan = TimeSpan.FromMinutes(5);
        }
    }
}

```

```

        manager.MaxFailedAccessAttemptsBeforeLockout = 5;

        // Register two factor authentication providers. This application uses Phone
and Emails as a step of receiving a code for verifying the user
        // You can write your own provider and plug it in here.
        manager.RegisterTwoFactorProvider(
            "Phone Code",
            new PhoneNumberTokenProvider<ApplicationUser> { MessageFormat = "Your
security code is {0}" });
        manager.RegisterTwoFactorProvider(
            "Email Code",
            new EmailTokenProvider<ApplicationUser> { Subject = "Security Code",
BodyFormat = "Your security code is {0}" });
        manager.EmailService = new EmailService();
        manager.SmsService = new SmsService();
        var dataProtectionProvider = options.DataProtectionProvider;
        if (dataProtectionProvider != null)
        {
            manager.UserTokenProvider =
                new
DataProtectorTokenProvider<ApplicationUser>(dataProtectionProvider.Create("ASP.NET
Identity"));
        }

        return manager;
    }
}

```

AutofacConfig.cs

```

namespace DDS.Web
{
    using System.Data.Entity;
    using System.Reflection;
    using System.Web.Mvc;

    using Autofac;
    using Autofac.Integration.Mvc;
    using Controllers;
    using Data;
    using Data.Common;
    using Services.Data.Interfaces;
    using Services.Web;

    public static class AutofacConfig
    {
        public static void RegisterAutofac()
        {
            var builder = new ContainerBuilder();

            // Register your MVC controllers.
            builder.RegisterControllers(typeof(MvcApplication).Assembly);

            // OPTIONAL: Register model binders that require DI.
            builder.RegisterModelBinders(Assembly.GetExecutingAssembly());
            builder.RegisterModelBinderProvider();
        }
    }
}

```

```

        // OPTIONAL: Register web abstractions like HttpContextBase.
        builder.RegisterModule<AutofacWebTypesModule>();

        // OPTIONAL: Enable property injection in view pages.
        builder.RegisterSource(new ViewRegistrationSource());

        // OPTIONAL: Enable property injection into action filters.
        builder.RegisterFilterProvider();

        // Register services
        RegisterServices(builder);

        // Set the dependency resolver to be Autofac.
        var container = builder.Build();
        DependencyResolver.SetResolver(new AutofacDependencyResolver(container));
    }

    private static void RegisterServices(ContainerBuilder builder)
    {
        builder.Register(x => new ApplicationDbContext())
            .As<DbContext>()
            .InstancePerRequest();
        builder.Register(x => new HttpCacheService())
            .As<ICacheService>()
            .InstancePerRequest();

        var servicesAssembly = Assembly.GetAssembly(typeof(IDiplomasService));
        builder.RegisterAssemblyTypes(servicesAssembly).AsImplementedInterfaces();

        builder.RegisterGeneric(typeof(DbRepository<>))
            .As(typeof(IDbRepository<>))
            .InstancePerRequest();

        builder.RegisterAssemblyTypes(Assembly.GetExecutingAssembly())
            .AssignableTo<BaseController>().PropertiesAutowired();
    }
}

```

RouteConfig.cs

```

namespace DDS.Web
{
    using System.Web.Mvc;
    using System.Web.Routing;

    public static class RouteConfig
    {
        {
            public static void RegisterRoutes(RouteCollection routes)
            {
                routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

                routes.MapRoute(
                    name: "Default",
                    url: "{controller}/{action}/{id}",

```



```

                defaults: new { controller = "Home", action = "Index", id =
UrlParameter.Optional },
                namespaces: new[] { "DDS.Web.Controllers" });
        }
    }
}

```

FilterConfig.cs

```

namespace DDS.Web
{
    using System.Web.Mvc;

    public static class FilterConfig
    {
        public static void RegisterGlobalFilters(GlobalFilterCollection filters)
        {
            filters.Add(new HandleErrorAttribute());
        }
    }
}

```

BundleConfig.cs

```

namespace DDS.Web
{
    using System.Web.Optimization;

    public static class BundleConfig
    {
        // For more information on bundling, visit
http://go.microsoft.com/fwlink/?LinkId=301862
        public static void RegisterBundles(BundleCollection bundles)
        {
            RegisterScripts(bundles);
            RegisterStyles(bundles);
        }

        private static void RegisterScripts(BundleCollection bundles)
        {
            bundles.Add(new ScriptBundle("~/bundles/jquery").Include("~/Scripts/jquery-
{version}.js"));
            bundles.Add(new
ScriptBundle("~/bundles/jqueryval").Include("~/Scripts/jquery.validate*"));
            bundles.Add(new
ScriptBundle("~/bundles/bootstrap").Include("~/Scripts/bootstrap.js"));
            bundles.Add(new
ScriptBundle("~/bundles/select2").Include("~/Scripts/select2/select2.full.min.js"));
        }

        private static void RegisterStyles(BundleCollection bundles)
        {
            bundles.Add(new
StyleBundle("~/Content/css").Include("~/Content/cerulean.bootstrap.css",
"~/Content/Site.css", "~/Content/select2.min.css"));
        }
    }
}

```

```
}
```

Startup.Auth.cs

```
namespace DDS.Web
{
    using System;
    using App_Start;
    using DDS.Data;
    using DDS.Data.Models;
    using Microsoft.AspNet.Identity;
    using Microsoft.AspNet.Identity.Owin;
    using Microsoft.Owin;
    using Microsoft.Owin.Security.Cookies;
    using Owin;

    public partial class Startup
    {
        // For more information on configuring authentication, please visit
        http://go.microsoft.com/fwlink/?LinkId=301864
        public void ConfigureAuth(IAppBuilder app)
        {
            // Configure the db context, user manager and signin manager to use a single
            instance per request
            app.CreatePerOwinContext(ApplicationDbContext.Create);

            app.CreatePerOwinContext<ApplicationUserManager>(ApplicationUserManager.Create);

            app.CreatePerOwinContext<ApplicationSignInManager>(ApplicationSignInManager.Create);

            app.CreatePerOwinContext<ApplicationRoleManager>(ApplicationRoleManager.Create);

            app.UseCookieAuthentication(new CookieAuthenticationOptions
            {
                AuthenticationType = DefaultAuthenticationTypes.ApplicationCookie,
                LoginPath = new PathString("/Account/Login"),
                Provider = new CookieAuthenticationProvider
                {
                    OnValidateIdentity =
                    SecurityStampValidator.OnValidateIdentity<ApplicationUserManager, ApplicationUser>(
                        validateInterval: TimeSpan.FromMinutes(30),
                        regenerateIdentity: (manager, user) =>
                        user.GenerateUserIdentityAsync(manager))
                }
            });
            app.UseExternalSignInCookie(DefaultAuthenticationTypes.ExternalCookie);
            app.UseTwoFactorSignInCookie(DefaultAuthenticationTypes.TwoFactorCookie,
            TimeSpan.FromMinutes(5));

            app.UseTwoFactorRememberBrowserCookie(DefaultAuthenticationTypes.TwoFactorRememberBrowserC
            ookie);
        }
    }
}
```

3.2.2 Area – Administration

AdministrationAreaRegistration.cs

```
namespace DDS.Web.Areas.Administration
{
    using System.Web.Mvc;

    public class AdministrationAreaRegistration : AreaRegistration
    {
        public override string AreaName => "Administration";

        public override void RegisterArea(AreaRegistrationContext context)
        {
            context.MapRoute(
                "Administration_default",
                "Administration/{controller}/{action}/{id}",
                new { action = "Index", id = UrlParameter.Optional },
                new[] { "DDS.Web.Areas.Administration.Controllers" });
        }
    }
}
```

AdministrationController.cs

```
namespace DDS.Web.Areas.Administration.Controllers
{
    using System;
    using System.Linq;
    using System.Net;
    using System.Threading.Tasks;
    using System.Web;
    using System.Web.Mvc;
    using App_Start;
    using Data.Models;
    using DDS.Common;
    using DDS.Web.Controllers;
    using DDS.Web.Infrastructure;
    using Microsoft.AspNet.Identity;
    using Microsoft.AspNet.Identity.Owin;
    using Services.Data.Interfaces;
    using ViewModels;

    [Authorize(Roles = GlobalConstants.AdministratorRoleName)]
    public class AdministrationController : BaseController
    {
        private readonly IAdministrationService users;

        private ApplicationRoleManager roleManager;
        private ApplicationUserManager userManager;

        public AdministrationController(
            IAdministrationService users)
        {
            this.users = users;
        }
    }
}
```

```

        public ApplicationUserManager UserManager
        {
            get
            {
                return this.userManager ??
this.Request.GetOwinContext().GetUserManager<ApplicationUserManager>();
            }

            private set
            {
                this.userManager = value;
            }
        }

        public ApplicationRoleManager RoleManager
        {
            get
            {
                return this.roleManager ??
this.Request.GetOwinContext().Get<ApplicationRoleManager>();
            }

            private set
            {
                this.roleManager = value;
            }
        }

        [Authorize(Roles = GlobalConstants.AdministratorRoleName)]
        public ActionResult Index(string sortOrder, string currentFilter, string
searchString, int? page)
        {
            this.ViewBag.CurrentSort = sortOrder;
            this.ViewBag.NameSortParm = string.IsNullOrEmpty(sortOrder) ? "name_desc" :
string.Empty;
            this.ViewBag.DateSortParm = sortOrder == "Date" ? "date_desc" : "Date";

            if (searchString != null)
            {
                page = 1;
            }
            else
            {
                searchString = currentFilter;
            }

            this.ViewBag.CurrentFilter = searchString;

            var allusers = this.users.GetAll().ToList();

            var viewModel = allusers.Select(
                user => new UserViewModel
                {
                    Id = user.Id,
                    UserName = user.UserName,
                    FirstName = user.FirstName,
                    Email = user.Email,
                    LastName = user.LastName,

```

```

        PhoneNumber = user.PhoneNumber,
        SelectedRole = this.UserManager.GetRoles(user.Id).FirstOrDefault(),
        RolesList = this.UserManager.GetRoles(user.Id)
                                .Select(role => new SelectListItem { Text =
role, Value = role, Selected = true })
    });

    if (!string.IsNullOrEmpty(searchString))
    {
        searchString = searchString.ToLower();
        viewModel = viewModel.Where(s =>
s.FirstName.ToLower().Contains(searchString) ||
s.LastName.ToLower().Contains(searchString)
        || s.Email.ToLower().Contains(searchString) ||
s.UserName.ToLower().Contains(searchString) || (s.PhoneNumber != null &&
s.PhoneNumber.Contains(searchString)));
    }

    if (viewModel.LongCount() <= 0)
    {
        this.TempData["Message"] = "Не са намерени потребители!";
    }

    return this.View(viewModel);
}

// GET: Administration/Administration/Edit/5
public ActionResult Edit(string id)
{
    if (id == null)
    {
        return this.RedirectToAction("Index");
    }

    var user = this.users.GetById(id);
    if (user == null)
    {
        return this.HttpNotFound();
    }

    var allRoles = this.RoleManager.Roles.Select(
        role => new SelectListItem
        {
            Text = role.Name,
            Value = role.Name,
            Selected = false
        });

    UserViewModel userViewModel = new UserViewModel
    {
        Id = user.Id,
        FirstName = user.FirstName,
        LastName = user.LastName,
        Email = user.Email,
        RolesList = this.UserManager.GetRoles(user.Id)
                                .Select(role => new SelectListItem { Text =
role, Value = role, Selected = true })
                                .Concat(allRoles)
    }
}

```

```

        .GroupBy(r => r.Text)
        .Select(grp => grp.FirstOrDefault())
        .OrderBy(r => r.Text)
        .ToList()
    };

    return this.View(userViewModel);
}

// POST: Administration/UserViewModels/Edit/5
// To protect from overposting attacks, please enable the specific properties you
want to bind to, for
// more details see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<ActionResult> Edit(UserViewModel model)
{
    if (this.ModelState.IsValid)
    {
        var userRoles = await this.UserManager.GetRolesAsync(model.Id);

        if (!this.UserManager.IsInRole(model.Id, model.SelectedRole))
        {
            var addResult = await this.UserManager.AddToRoleAsync(model.Id,
model.SelectedRole);

            var result = await this.UserManager.RemoveFromRolesAsync(model.Id,
userRoles.ToArray());

            if (!result.Succeeded)
            {
                //this.ModelState.AddModelError("", result.Errors.First());
                return this.RedirectToAction("Index", "Home");
            }
        }

        var user = this.UserManager.FindById(model.Id);
        user.FirstName = model.FirstName;
        user.LastName = model.LastName;
        user.Email = model.Email;
        this.UserManager.Update(user);

        var teacher = this.users.GetAllTeachers().FirstOrDefault(t => t.User.Id ==
model.Id);

        if (this.UserManager.IsInRole(model.Id, GlobalConstants.TeacherRoleName))
        {
            if (teacher == null)
            {
                user.Teacher = new Teacher()
                {
                    CreatedOn = DateTime.Now,
                };

                this.UserManager.Update(user);
            }
            else
            {

```

```

        teacher.IsDeleted = false;
        teacher.ModifiedOn = DateTime.Now;
        teacher.DeletedOn = null;
        this.users.EditTeacher(teacher);
    }
}
else
{
    if (teacher != null)
    {
        teacher.IsDeleted = true;
        teacher.DeletedOn = DateTime.Now;
        this.users.EditTeacher(teacher);
    }
}

var student = this.users.GetAllStudents().FirstOrDefault(t => t.User.Id ==
model.Id);

if (this.UserManager.IsInRole(model.Id, GlobalConstants.StudentRoleName))
{
    if (student == null)
    {
        user.Student = new Student()
        {
            CreatedOn = DateTime.Now,
        };

        this.UserManager.Update(user);
    }
    else
    {
        student.IsDeleted = false;
        student.ModifiedOn = DateTime.Now;
        student.DeletedOn = null;
        this.users.EditStudent(student);
    }
}
else
{
    if (student != null)
    {
        student.IsDeleted = true;
        student.DeletedOn = DateTime.Now;
        this.users.EditStudent(student);
    }
}

return this.RedirectToAction("Index");
}

return this.View(model);
}
}
}

```

ManageDiplomasController.cs

```
namespace DDS.Web.Areas.Administration.Controllers
{
    using System.Data.Entity;
    using System.Linq;
    using System.Web.Mvc;
    using Common;
    using Infrastructure.Mapping;
    using PagedList;
    using Services.Data.Interfaces;
    using ViewModels;
    using Web.Controllers;
    using Web.ViewModels.ManageDiplomas;
    using Web.ViewModels.Shared;

    [Authorize(Roles = GlobalConstants.AdministratorRoleName)]
    public class ManageDiplomasController : BaseController
    {
        private readonly ITeachersService teachers;
        private readonly IDiplomasService diplomas;
        private readonly IStudentsService students;
        private readonly ITagsService tags;

        public ManageDiplomasController(
            ITeachersService teachers,
            IDiplomasService diplomas,
            IStudentsService students,
            ITagsService tags)
        {
            this.teachers = teachers;
            this.diplomas = diplomas;
            this.students = students;
            this.tags = tags;
        }

        // GET: ManageDiplomas
        public ActionResult Index(string sortOrder, string currentFilter, string
searchString, int? page)
        {
            this.ViewBag.CurrentSort = sortOrder;
            this.ViewBag.NameSortParm = string.IsNullOrEmpty(sortOrder) ? "name_desc" :
string.Empty;
            this.ViewBag.DateSortParm = sortOrder == "Date" ? "date_desc" : "Date";

            if (searchString != null)
            {
                page = 1;
            }
            else
            {
                searchString = currentFilter;
            }

            this.ViewBag.CurrentFilter = searchString;

            //var teacher =
this.teachers.GetByUserId(this.User.Identity.GetUserId()).FirstOrDefault();
```



```

        var diplomas = this.diplomas.GetAll().Where(d => d.IsApprovedByLeader)
            .ToCommonDiplomaViewModel();

        if (!string.IsNullOrEmpty(searchString))
        {
            if (searchString.ToLower() == "одобрени")
            {
                diplomas = diplomas.Where(d => d.IsApprovedByLeader ||
d.IsApprovedByHead);
            }
            else if (searchString.ToLower() == "избрани")
            {
                diplomas = diplomas.Where(d => d.IsSelectedByStudent);
            }
            else
            {
                diplomas = diplomas.Where(s => s.Title.Contains(searchString) ||
s.Description.Contains(searchString));
            }
        }

        if (diplomas.LongCount() <= 0)
        {
            this.TempData["Message"] = "Не са намерени дипломи!";
        }
        else
        {
            switch (sortOrder)
            {
                case "name_desc":
                    diplomas = diplomas.OrderByDescending(s => s.Title);
                    break;

                case "Date":
                    diplomas = diplomas.OrderBy(s => s.CreatedOn);
                    break;

                case "date_desc":
                    diplomas = diplomas.OrderByDescending(s => s.CreatedOn);
                    break;

                default:
                    // Title ascending
                    diplomas = diplomas.OrderBy(s => s.Title);
                    break;
            }
        }

        int pageSize = 5;
        int pageNumber = page ?? 1;

        return this.View(diplomas.ToPagedList(pageNumber, pageSize));
    }

    public ActionResult Details(int? id)
    {
        if (id == null)
        {

```

```

        return this.RedirectToAction("Index", "ManageDiplomas");
    }

    int intId = id ?? 0;
    var diploma = this.diplomas.GetObjectById(intId);
    if (diploma == null)
    {
        this.TempData["Message"] = "Дипломата не бе намерена!";
        return this.RedirectToAction("Index", "ManageDiplomas");
    }

    var result = new StudentDiplomaViewModel();

    var studentDetails = this.students.GetAll()
        .Where(s => s.SelectedDiploma.Id ==
diploma.Id)
        .Include(s => s.User)
        .To<SimpleStudentViewModel>()
        .FirstOrDefault();

    result.Student = studentDetails;

    var diplomaModel = this.diplomas.GetAll()
        .Where(d => d.Id == intId)
        .Include(d => d.Teacher)
        .Include(d => d.Tags)
        .To<DisplayDiplomaViewModel>();

    result.Diploma = diplomaModel.FirstOrDefault();
    result.Diploma.ContentCSV = diploma.ContentCSV.Split(new char[] {
GlobalConstants.ContentSeparator },
System.StringSplitOptions.RemoveEmptyEntries).ToList();
    result.Diploma.TeacherID = diploma.TeacherID;
    result.Diploma.Tags = diploma.Tags.Select(t => new SelectListItem
    {
        Text = t.Name,
        Disabled = true,
        Selected = true,
        Value = t.Id.ToString()
    });

    var teacher = this.teachers.GetById(diploma.TeacherID).Include(t =>
t.User).FirstOrDefault();
    result.Diploma.TeacherName = string.Format("{0} {1} {2}",
teacher.User.ScienceDegree, teacher.User.FirstName, teacher.User.LastName).Trim();

    var dipl = this.diplomas.GetAll().Where(d => d.IsApprovedByLeader).Select(d =>
new DiplomaTitleViewModel { Title = d.Title, Id = d.Id }).ToList();

    var duplicates = dipl.Where(d =>
Infrastructure.StringComparer.CalculateSimilarity(d.Title, result.Diploma.Title) >= 0.75
&& d.Id != result.Diploma.Id);
    result.Duplicates = duplicates;
    return this.View(result);
}

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Approve(int id)

```

```

        {
            var diploma = this.diplomas.GetObjectById(id);
            diploma.IsApprovedByHead = true;
            this.diplomas.Save();

            return this.RedirectToAction("Details", "ManageDiplomas", new { @id = id });
        }
    }
}

```

3.2.3 ViewModels

UserViewModel.cs

```

namespace DDS.Web.Areas.Administration.ViewModels
{
    using System.Collections.Generic;
    using System.ComponentModel;
    using System.Web.Mvc;
    using Data.Models;
    using Infrastructure.Mapping;

    public class UserViewModel : IMapFrom<ApplicationUser>
    {
        public string Id { get; set; }

        [DisplayName("Име")]
        public string FirstName { get; set; }

        [DisplayName("Фамилия")]
        public string LastName { get; set; }

        [DisplayName("Емейл")]
        public string Email { get; set; }

        [DisplayName("Телефон")]
        public string PhoneNumber { get; set; }

        [DisplayName("Псевдоним")]
        public string UserName { get; set; }

        [DisplayName("Права")]
        public IEnumerable<SelectListItem> RolesList { get; set; }

        public string SelectedRole { get; set; }

        public List<string> RolesValues { get; set; }
    }
}

```

DiplomaTitleViewModel.cs

```

namespace DDS.Web.Areas.Administration.ViewModels
{
    using DDS.Data.Models;
    using DDS.Web.Infrastructure.Mapping;
}

```

```

public class DiplomaTitleViewModel : IMapFrom<Diploma>
{
    public int Id { get; set; }

    public string Title { get; set; }
}

```

3.2.4 Views

Administration

Edit.cshtml

```

@model DDS.Web.Areas.Administration.ViewModels.UserViewModel

@{
    ViewBag.Title = "Редактиране на права";
}

@using (Html.BeginForm("Edit", "Administration", FormMethod.Post))
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <h4>Промяна на правата на потребителя.</h4>
        <hr />
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        @Html.HiddenFor(model => model.Id)

        <div class="form-group">
            @Html.LabelFor(model => model.FirstName, htmlAttributes: new { @class = "text-right col-md-2" })
            @Html.DisplayFor(model => model.FirstName, new { htmlAttributes = new { @class = "form-control col-lg-10" } })
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.LastName, htmlAttributes: new { @class = "text-right col-lg-2" })
            @Html.DisplayFor(model => model.LastName, new { htmlAttributes = new { @class = "form-control col-lg-10" } })
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Email, htmlAttributes: new { @class = "text-right col-lg-2" })
            @Html.DisplayFor(model => model.Email, new { htmlAttributes = new { @class = "form-control col-lg-10" } })
        </div>

        <div class="form-group">
            @Html.Label("Права", new { @class = "control-label col-lg-2" })
            @Html.DropDownListFor(model => model.SelectedRole, Model.RolesList, new { @class = "form-control col-lg-2" })
        </div>
    </div>

```

```

        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Запази" class="btn btn-success" />
                @Html.ActionLink("Обратно", "Index", null, new { @class = "btn btn-
default" })
            </div>
        </div>
    </div>
}
<div>
</div>

```

Administration

Index.cshtml

```

@model IEnumerable<DDS.Web.Areas.Administration.ViewModels.UserViewModel>

@using DDS.Web.Helpers;
@{
    ViewBag.Title = "Списък с потребители";
}

@if (TempData["Message"] != null)
{
    <div class="row">
        @Html.Alert("", this.TempData["Message"].ToString(), AlertType.Info)
    </div>
}

<div class="row">
    <div class="col-xs-6">
    </div>
    @using (Html.BeginForm("Index", "Administration", FormMethod.Get))
    {
        @Html.Partial("~/Views/Shared/_SearchBoxPartial.cshtml")
    }
</div>
<br />

<table class="table table-responsive">
    <tr>
        <th>
            @Html.DisplayNameFor(model => model.FirstName)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.LastName)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Email)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.PhoneNumber)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.UserName)
        </th>
    </tr>

```

```

        <th>
            @Html.DisplayNameFor(model => model.RolesList)
        </th>
    </tr>

    @foreach (var item in Model)
    {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.FirstName)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.LastName)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Email)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.PhoneNumber)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.UserName)
            </td>
            <td>
                <span>
                    @foreach (var role in item.RolesList)
                    {
                        @Html.Label(role.Text, new { @class = "control-label " + role.Text
    })
                    }
                </span>
            </td>
            <td>
                @Html.ActionLink("Промени", "Edit", new { id = item.Id }, new { @class =
"btn-sm btn-default" })
            </td>
        </tr>
    }
</table>

```

ManageDiplomas

Index.cshtml

```

@model PagedList.IPagedList<DDS.Web.ViewModels.Shared.CommonDiplomaViewModel>

@using DDS.Web.Helpers;
<link href="~/Content/PagedList.css" rel="stylesheet" type="text/css" />

@{
    ViewBag.Title = "Менажиране на дипломи";
}

@if (TempData["Message"] != null)
{
    <div class="row">
        @Html.Alert("", this.TempData["Message"].ToString(), AlertType.Info)
    </div>
}

```

```

    </div>
}
<br />
<br />
<br />

<div class="row">
    <div class="col-xs-6">
    </div>
    @using (Html.BeginForm("Index", "ManageDiplomas", FormMethod.Get))
    {
        @Html.Partial("~/Views/Shared/_SearchBoxPartial.cshtml")
    }
</div>

<table class="table table-hover table-responsive">
    <thead>
        <tr>
            <th class="">#</th>
            <th class="col-md-5">
                @Html.ActionLink("Заглавие", "Index", new { sortOrder =
ViewBag.NameSortParm, currentFilter = ViewBag.CurrentFilter })
            </th>
            <th class="col-md-5">
                @Html.DisplayName("Описание")
            </th>
            <th class="col-md-2">
                @Html.ActionLink("Дата на добавяне", "Index", new { sortOrder =
ViewBag.DateSortParm, currentFilter = ViewBag.CurrentFilter })
            </th>
        </tr>
    </thead>
    <tbody>
        @{ int index = (Model.PageNumber - 1) * Model.PageSize;}
        @foreach (var item in Model)
        {
            <tr class="@((item.IsApprovedByHead ? "bg-success" :
                item.IsApprovedByLeader ? "bg-info" :
                item.IsSelectedByStudent ? "bg-warning" :
                ""))">

                <td class="">
                    @{index++;}
                    @index
                    @Html.Hidden("RedirectToDetails", Url.Action("Details",
"ManageDiplomas", new { id = item.Id }))
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.Title)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.Description)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.CreatedOn)
                </td>
            </tr>
        }
    </tbody>
</table>

```

```

</table>
<br />
<div class="row">
    @Html.Partial("_Pager", Model, new ViewDataDictionary { { "ActionName", "Index" } })
</div>

<div class="row">
    <div class="col-sm-12">
        <div class="row">
            <div class="col-sm-1">
                <div class="alert-warning well-sm"></div>
            </div>
            <div class="col-sm-3">
                <p>Избрана от студент</p>
            </div>
            <div class="col-sm-1">
                <div class="alert-info well-sm"></div>
            </div>
            <div class="col-sm-3">
                <p>Одобрена от ръководител</p>
            </div>
            <div class="col-sm-1">
                <div class="alert-success well-sm"></div>
            </div>
            <div class="col-sm-3">
                <p>Одобрена и подписана от декан</p>
            </div>
        </div>
    </div>
</div>

@section scripts{
    <script type="text/javascript">
        window.onload = function () {
            $('table > tbody > tr').on('click', function () {
                var url = $(this).find("#RedirectToDetails").val();
                location.href = url;
            });

            $('.alert').fadeIn('fast').delay(3000).fadeOut('slow');
        }

        $(function () {
            $("a.delete-link").click(function () {
                var deleteLink = $(this);
                deleteLink.hide();
                var confirmButton = deleteLink.siblings(".delete-confirm");
                confirmButton.show();

                var cancelDelete = function () {
                    removeEvents();
                    showDeleteLink();
                };

                var deleteItem = function () {
                    removeEvents();
                    confirmButton.hide();
                    $.post(

```



```

        '@Url.Action("Delete")',
        AddAntiForgeryToken({ id: confirmButton.attr('data-delete-id') }))
        .done(function () {
            var parentRow = deleteLink.parents("tr:first");
            parentRow.fadeOut('fast', function () {
                parentRow.remove();
            });
        }).fail(function (data) {
            alert("error");
        });
        return false;
    };

    var removeEvents = function () {
        confirmButton.off("click", deleteItem);
        $(document).on("click", cancelDelete);
        $(document).off("keypress", onKeyPress);
    };

    var showDeleteLink = function () {
        confirmButton.hide();
        deleteLink.show();
    };

    var onKeyPress = function (e) {
        //Cancel if escape key pressed
        if (e.which == 27) {
            cancelDelete();
        }
    };

    confirmButton.on("click", deleteItem);
    $(document).on("click", cancelDelete);
    $(document).on("keypress", onKeyPress);

    return false;
});

AddAntiForgeryToken = function (data) {
    data.__RequestVerificationToken =
$('input[name=__RequestVerificationToken]').val();
    return data;
};

});
</script>
}

```

ManageDiplomas

Details.cshtml

```

@model DDS.Web.ViewModels.ManageDiplomas.StudentDiplomaViewModel
@using DDS.Web.Areas.Administration.ViewModels
@{
    ViewBag.Title = "Детайли";
}

@if (Model.Diploma.IsSelectedByStudent)

```

```

{
    <h4>Информация за студента</h4>
    <dl class="panel dl-horizontal">
        <dt class="">
            @Html.DisplayNameFor(model => model.Student.FirstName)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.Student.ScienceDegree) @Html.DisplayFor(model
=> model.Student.FirstName) @Html.DisplayFor(model => model.Student.MiddleName)
@Html.DisplayFor(model => model.Student.LastName)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.Student.FNumber)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.Student.FNumber)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.Student.Email)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.Student.Email)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.Student.Address)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.Student.Address)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.Student.PhoneNumber)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.Student.PhoneNumber)
        </dd>
    </dl>
}

<h4>Информация за дипломната работа</h4>
<dl class="dl-horizontal">
    <dt>
        @Html.DisplayNameFor(model => model.Diploma.TeacherName)
    </dt>

    <dd>
        @Html.ActionLink(Model.Diploma.TeacherName, "TeacherDetails", "Home", new { Area =
string.Empty, id = Model.Diploma.TeacherID }, new { @class = "btn-link" })
    </dd>

    <dt>
        @Html.DisplayNameFor(model => model.Diploma.Title)
    </dt>

```

```

</dt>

<dd>
    @Html.DisplayFor(model => model.Diploma.Title)
</dd>

<dt>
    @Html.DisplayNameFor(model => model.Diploma.Description)
</dt>

<dd>
    @Html.DisplayFor(model => model.Diploma.Description)
</dd>

<dt>
    @Html.DisplayNameFor(model => model.Diploma.ContentCSV)
</dt>
<dd>
    <ol>
        @foreach (var item in Model.Diploma.ContentCSV)
        {
            <li> @item </li>
        }
    </ol>
</dd>

<dt>
    @Html.DisplayNameFor(model => model.Diploma.ExperimentalPart)
</dt>

<dd>
    @Html.DisplayFor(model => model.Diploma.ExperimentalPart)
</dd>

<dt>
    @Html.DisplayNameFor(model => model.Diploma.Tags)
</dt>

<dd>
    @foreach (var item in Model.Diploma.Tags)
    {
        @Html.ActionLink(@item.Text, "Tag", "Home", new { id = item.Value }, new {
@class = "btn-link" }) <text> </text>
    }
</dd>

<dt>
    @Html.DisplayNameFor(model => model.Diploma.CreatedOn)
</dt>
<dd>
    @Html.DisplayFor(model => model.Diploma.CreatedOn)
</dd>

@if (Model.Diploma.ModifiedOn != null)
{
    <dt>

```

```

        @Html.DisplayNameFor(model => model.Diploma.ModifiedOn)
    </dt>

    <dd>
        @Html.DisplayFor(model => model.Diploma.ModifiedOn)
    </dd>
}

@if (Model.Diploma.IsDeleted)
{
    <dt>
        @Html.DisplayNameFor(model => model.Diploma.DeletedOn)
    </dt>

    <dd>
        @Html.DisplayFor(model => model.Diploma.DeletedOn)
    </dd>
}

@if (Model.Diploma.IsSelectedByStudent)
{
    <dt>
        @Html.DisplayNameFor(model => model.Diploma.IsSelectedByStudent)
    </dt>
    <dd>
        <span class="success" style="padding: 5px">Избрана</span>
    </dd>

    <dt>
        @Html.DisplayNameFor(model => model.Diploma.IsApprovedByLeader)
    </dt>
    <dd>
        @if (Model.Diploma.IsApprovedByLeader)
        {
            <span class="text-success" style="padding: 5px">Одобрена</span>
        }
        else
        {
            <span class="text-warning" style="padding: 5px">Не одобрена</span>
        }
    </dd>

    <dt>
        @Html.DisplayNameFor(model => model.Diploma.IsApprovedByHead)
    </dt>
    <dd>
        @if (Model.Diploma.IsApprovedByHead)
        {
            <span class="text-success" style="padding: 5px">Одобрена</span>
        }
        else
        {
            <span class="text-warning" style="padding: 5px">Не одобрена</span>
        }
    </dd>
}
<dt></dt>

```

```

<dd>
    <div class="row">
        @if (Model.Diploma.IsApprovedByLeader && !Model.Diploma.IsApprovedByHead)
        {
            using (Html.BeginForm("Approve", "ManageDiplomas", new { Id =
Model.Diploma.Id }, FormMethod.Post))
            {
                @Html.AntiForgeryToken()
                <input type="submit" value="Одобри" class="btn btn-primary col-sm-1"
/>
            }
        }
        @Html.ActionLink("Обратно", "Index", null, htmlAttributes: new { @class = "btn
btn-default col-sm-1" })
    </div>
</dd>
</dl>

<div class="row">
    <div class="col-md-12">
        @if (Model.Duplicates != null)
        {
            foreach (var item in Model.Duplicates)
            {
                <div class="row">
                    <div class="col-sm-12">
                        @Html.ActionLink(item.Title, "Details", new { id = item.Id }, new
{ @class = "btn btn-link" })
                    </div>
                </div>
            }
        }
    </div>
</div>

```

Area Administration - Web.config

```
<?xml version="1.0"?>

<configuration>
  <configSections>
    <sectionGroup name="system.web.webPages.razor"
type="System.Web.WebPages.Razor.Configuration.RazorWebSectionGroup,
System.Web.WebPages.Razor, Version=3.0.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35">
      <section name="host" type="System.Web.WebPages.Razor.Configuration.HostSection,
System.Web.WebPages.Razor, Version=3.0.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35" requirePermission="false" />
      <section name="pages"
type="System.Web.WebPages.Razor.Configuration.RazorPagesSection,
System.Web.WebPages.Razor, Version=3.0.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35" requirePermission="false" />
    </sectionGroup>
  </configSections>

  <system.web.webPages.razor>
    <host factoryType="System.Web.Mvc.MvcWebRazorHostFactory, System.Web.Mvc,
Version=5.2.3.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" />
    <pages pageBaseType="System.Web.Mvc.WebViewPage">
      <namespaces>
        <add namespace="System.Web.Mvc" />
        <add namespace="System.Web.Mvc.Ajax" />
        <add namespace="System.Web.Mvc.Html" />
        <add namespace="System.Web.Routing" />
        <add namespace="System.Web.Optimization" />
        <add namespace="DDS.Web" />

      </namespaces>
    </pages>
  </system.web.webPages.razor>

  <appSettings>
    <add key="webpages:Enabled" value="false" />
  </appSettings>

  <system.webServer>
    <handlers>
      <remove name="BlockViewHandler"/>
      <add name="BlockViewHandler" path="*" verb="*" preCondition="integratedMode"
type="System.Web.HttpNotFoundHandler" />
    </handlers>
  </system.webServer>
</configuration>
```

3.2.5 Controllers

AccountController.cs

```
namespace DDS.Web.Controllers
{
    using System;
    using System.Linq;
```

```

using System.Threading.Tasks;
using System.Web;
using System.Web.Mvc;
using Common;
using DDS.Data.Models;
using DDS.Web.ViewModels.Account;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.Owin;
using Microsoft.Owin.Security;
using Services.Data.Interfaces;

[Authorize]
public class AccountController : BaseController
{
    // Used for XSRF protection when adding external logins
    private const string XsrfKey = "XsrfId";

    private readonly IStudentsService students;

    private ApplicationSignInManager signInManager;

    private ApplicationUserManager userManager;

    public AccountController(
        IStudentsService students)
    {
        this.students = students;
    }

    public AccountController(ApplicationUserManager userManager,
        ApplicationSignInManager signInManager)
    {
        this.UserManager = userManager;
        this.SignInManager = signInManager;
    }

    public ApplicationSignInManager SignInManager
    {
        get
        {
            return this.signInManager ??
this.HttpContext.GetOwinContext().Get<ApplicationSignInManager>();
        }

        private set
        {
            this.signInManager = value;
        }
    }

    public ApplicationUserManager UserManager
    {
        get
        {
            return this.userManager ??
this.HttpContext.GetOwinContext().GetUserManager<ApplicationUserManager>();
        }
    }
}

```

```

        private set
        {
            this.userManager = value;
        }
    }

    private IAuthenticationManager AuthenticationManager =>
this.HttpContext.GetOwinContext().Authentication;

    // GET: /Account/Login
    [AllowAnonymous]
    public ActionResult Login(string returnUrl)
    {
        this.ViewBag.ReturnUrl = returnUrl;
        return this.View();
    }

    // POST: /Account/Login
    [HttpPost]
    [AllowAnonymous]
    [ValidateAntiForgeryToken]
    public async Task<ActionResult> Login(LoginViewModel model, string returnUrl)
    {
        if (!this.ModelState.IsValid)
        {
            return this.View(model);
        }

        // This doesn't count login failures towards account lockout
        // To enable password failures to trigger account lockout, change to
shouldLockout: true
        var result =
            await
            this.SignInManager.PasswordSignInAsync(
                model.UserName,
                model.Password,
                model.RememberMe,
                shouldLockout: false);
        switch (result)
        {
            case SignInStatus.Success:
                return this.RedirectToLocal(returnUrl);
            case SignInStatus.LockedOut:
                return this.View("Lockout");
            case SignInStatus.RequiresVerification:
                return this.RedirectToAction(
                    "SendCode",
                    new { ReturnUrl = returnUrl, model.RememberMe });
            case SignInStatus.Failure:
            default:
                this.ModelState.AddModelError(string.Empty, "Invalid login attempt.");
                return this.View(model);
        }
    }

    // GET: /Account/Register
    [AllowAnonymous]
    public ActionResult Register()

```



```

    {
        return this.View();
    }

    // POST: /Account/Register
    [HttpPost]
    [AllowAnonymous]
    [ValidateAntiForgeryToken]
    public async Task<ActionResult> Register(RegisterViewModel model)
    {
        if (this.ModelState.IsValid)
        {
            var hasher = new PasswordHasher();
            var user = new ApplicationUser
            {
                UserName = model.UserName,
                Email = model.Email,
                FirstName = model.FirstName,
                MiddleName = model.MiddleName,
                LastName = model.LastName,
                SecurityStamp = Guid.NewGuid().ToString(),
                PasswordHash = hasher.HashPassword(model.Password),
            };

            var result = await this.UserManager.UserValidator.ValidateAsync(user);
            if (result.Succeeded)
            {
                var student = new Student()
                {
                    User = user
                };
                this.students.Create(student);
                this.UserManager.AddToRole(user.Id, GlobalConstants.StudentRoleName);
                await this.SignInManager.SignInAsync(user, isPersistent: false,
rememberBrowser: false);

                // For more information on how to enable account confirmation and
password reset please visit http://go.microsoft.com/fwlink/?LinkID=320771
                // Send an email with this link
                // string code = await
UserManager.GenerateEmailConfirmationTokenAsync(user.Id);
                // var callbackUrl = Url.Action("ConfirmEmail", "Account", new {
userId = user.Id, code = code }, protocol: Request.Url.Scheme);
                // await UserManager.SendEmailAsync(user.Id, "Confirm your account",
                "Please confirm your account by clicking <a href=\"\" + callbackUrl + "\">here</a>");
                return this.RedirectToAction("Index", "Home");
            }

            this.AddErrors(result);
        }

        // If we got this far, something failed, redisplay form
        return this.View(model);
    }

    // GET: /Account/ResetPassword
    [AllowAnonymous]
    public ActionResult ResetPassword(string code)

```

```

    {
        return code == null ? this.View("Error") : this.View();
    }

    // POST: /Account/ResetPassword
    [HttpPost]
    [AllowAnonymous]
    [ValidateAntiForgeryToken]
    public async Task<ActionResult> ResetPassword(ResetPasswordViewModel model)
    {
        if (!this.ModelState.IsValid)
        {
            return this.View(model);
        }

        var user = await this.UserManager.FindByNameAsync(model.Email);
        if (user == null)
        {
            // Don't reveal that the user does not exist
            return this.RedirectToAction("ResetPasswordConfirmation", "Account");
        }

        var result = await this.UserManager.ResetPasswordAsync(user.Id, model.Code,
model.Password);
        if (result.Succeeded)
        {
            return this.RedirectToAction("ResetPasswordConfirmation", "Account");
        }

        this.AddErrors(result);
        return this.View();
    }

    // POST: /Account/LogOff
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult LogOff()
    {
        this.AuthenticationManager.SignOut(DefaultAuthenticationTypes.ApplicationCookie);
        return this.RedirectToAction("Index", "Home");
    }

    protected override void Dispose(bool disposing)
    {
        if (disposing)
        {
            if (this.userManager != null)
            {
                this.userManager.Dispose();
                this.userManager = null;
            }

            if (this.signInManager != null)
            {
                this.signInManager.Dispose();
                this.signInManager = null;
            }
        }
    }

```

```

    }

    base.Dispose(disposing);
}

private void AddErrors(IdentityResult result)
{
    foreach (var error in result.Errors)
    {
        this.ModelState.AddModelError(string.Empty, error);
    }
}

private ActionResult RedirectToLocal(string returnUrl)
{
    if (this.Url.IsLocalUrl(returnUrl))
    {
        return this.Redirect(returnUrl);
    }

    return this.RedirectToAction("Index", "Home");
}

internal class ChallengeResult : HttpUnauthorizedResult
{
    public ChallengeResult(string provider, string redirectUri, string userId =
null)
    {
        this.LoginProvider = provider;
        this.RedirectUri = redirectUri;
        this.UserId = userId;
    }

    public string LoginProvider { get; set; }

    public string RedirectUri { get; set; }

    public string UserId { get; set; }

    public override void ExecuteResult(ControllerContext context)
    {
        var properties = new AuthenticationProperties { RedirectUri =
this.RedirectUri };
        if (this.UserId != null)
        {
            properties.Dictionary[XsrfKey] = this.UserId;
        }

        context.HttpContext.GetOwinContext().Authentication.Challenge(properties,
this.LoginProvider);
    }
}
}

```

BaseController.cs

```
namespace DDS.Web.Controllers
{
    using System.Web.Mvc;
    using AutoMapper;
    using DDS.Services.Web;
    using Infrastructure.Mapping;

    public abstract class BaseController : Controller
    {
        public ICacheService Cache { get; set; }

        protected IMapper Mapper
        {
            get
            {
                return AutoMapperConfig.Configuration.CreateMapper();
            }
        }
    }
}
```

ErrorController.cs

```
namespace DDS.Web.Controllers
{
    using System.Net;
    using System.Web.Mvc;

    public class ErrorController : BaseController
    {
        public ActionResult Error()
        {
            //this.Response.TrySkipIisCustomErrors = true;
            this.Response.StatusCode = (int)HttpStatusCode.InternalServerError;
            return this.View("Error");
        }

        public ActionResult PageNotFound()
        {
            //this.Response.TrySkipIisCustomErrors = true;
            this.Response.StatusCode = (int)HttpStatusCode.NotFound;
            return this.View("PageNotFound");
        }
    }
}
```

HomeController.cs

```
namespace DDS.Web.Controllers
{
    using System.Collections.Generic;
    using System.Data.Entity;
    using System.Linq;
    using System.Text;
    using System.Web;
    using System.Web.Mvc;
    using Common;
    using Data.Models;
    using Infrastructure;
    using Infrastructure.Mapping;
    using Microsoft.AspNet.Identity;
    using Microsoft.AspNet.Identity.Owin;
    using PagedList;
    using Services.Data.Interfaces;
    using ViewModels.ManageDiplomas;
    using ViewModels.Shared;

    public class HomeController : BaseController
    {
        private readonly IDiplomasService diplomas;
        private readonly ITeachersService teachers;
        private readonly IStudentsService students;
        private readonly IMessagesService messages;
        private readonly ITagsService tags;

        private ApplicationUserManager userManager;

        public HomeController(
            IDiplomasService diplomas,
            ITeachersService teachers,
            IStudentsService students,
            IMessagesService messages,
            ITagsService tags)
        {
            this.diplomas = diplomas;
            this.teachers = teachers;
            this.students = students;
            this.messages = messages;
            this.tags = tags;
        }

        public ApplicationUserManager UserManager
        {
            get
            {
                return this.userManager ??
this.Request.GetOwinContext().GetUserManager<ApplicationUserManager>();
            }

            private set
            {
                this.userManager = value;
            }
        }
    }
}
```

```

/// <summary>
/// Returns the index page (start up page)
/// </summary>
/// <returns>Index page</returns>
public ActionResult Index()
{
    var tags =
        this.Cache.Get(
            "tags",
            () => this.tags.GetAll().To<TagViewModel>().ToList(),
            30 * 60);
    return this.View();
}

public ActionResult DownloadFile(int? id)
{
    if (id == null)
    {
        return this.RedirectToAction("Index", "Home");
    }

    int intId = id ?? 0;
    var diploma = this.diplomas.GetObjectById(intId);
    if (diploma == null)
    {
        this.TempData["Message"] = "Дипломата не бе намерена!";
        return this.RedirectToAction("Index", "Home");
    }

    var result = new StudentDiplomaViewModel();

    var studentDetails = this.students.GetAll()
        .Where(s => s.SelectedDiploma.Id ==
diploma.Id)
        .Include(s => s.User)
        .To<SimpleStudentViewModel>()
        .FirstOrDefault();

    result.Student = studentDetails;

    if (result.Student != null)
    {
        if (string.IsNullOrEmpty(result.Student.Address) ||
string.IsNullOrEmpty(result.Student.Email)
            || string.IsNullOrEmpty(result.Student.FirstName) ||
string.IsNullOrEmpty(result.Student.MiddleName)
            || string.IsNullOrEmpty(result.Student.LastName) ||
string.IsNullOrEmpty(result.Student.PhoneNumber)
            || result.Student.FNumber == 0)
        {
            return this.RedirectToAction("Index", "Home");
        }
    }
    else
    {
        return this.RedirectToAction("Index", "Home");
    }
}

```

```

        var diplomaModel = this.diplomas.GetAll()
            .Where(d => d.Id == intId)
            .Include(d => d.Teacher)
            .Include(d => d.Tags)
            .To<DisplayDiplomaViewModel>();

        result.Diploma = diplomaModel.FirstOrDefault();
        result.Diploma.ContentCSV = diploma.ContentCSV.Split(new char[] {
GlobalConstants.ContentSeparator },
System.StringSplitOptions.RemoveEmptyEntries).ToList();
        result.Diploma.TeacherID = diploma.TeacherID;
        result.Diploma.Tags = diploma.Tags.Select(t => new SelectListItem
        {
            Text = t.Name,
            Disabled = true,
            Selected = true,
            Value = t.Id.ToString()
        });

        var teacher = this.teachers.GetById(diploma.TeacherID).Include(t =>
t.User).FirstOrDefault();
        result.Diploma.TeacherName = string.Format("{0} {1}. {2}",
teacher.User.ScienceDegree, teacher.User.FirstName[0], teacher.User.LastName).Trim();

        if (result.Diploma != null)
        {
            if (result.Diploma.ContentCSV.Count == 0 ||
string.IsNullOrEmpty(result.Diploma.ExperimentalPart)
                || result.Diploma.Tags.Count() == 0 ||
string.IsNullOrEmpty(result.Diploma.TeacherName)
                || string.IsNullOrEmpty(result.Diploma.Title) ||
string.IsNullOrEmpty(result.Diploma.ModifiedOn.ToString()))
            {
                return this.RedirectToAction("Index", "Home");
            }
        }
        else
        {
            return this.RedirectToAction("Index", "Home");
        }

        Dictionary<string, string> properties = new Dictionary<string, string>();

        properties.Add("[USER_NAME]", string.Format("{0} {1} {2} {3}",
result.Student.ScienceDegree, result.Student.FirstName, result.Student.MiddleName,
result.Student.LastName));
        properties.Add("[USER_FNUMBER]", result.Student.FNumber.ToString());
        properties.Add("[USER_ADDRESS]", result.Student.Address);
        properties.Add("[USER_TEL]", result.Student.PhoneNumber);
        properties.Add("[USER_EMAIL]", result.Student.Email);
        properties.Add("[DOC_TITLE]", result.Diploma.Title.ToString());
        properties.Add("[DOC_BEGINDATE]",
result.Diploma.ModifiedOn.Value.ToString("dd/MM/yyyy"));
        properties.Add("[DOC_ENDDATA]",
result.Diploma.ModifiedOn.Value.AddYears(1).ToString("dd/MM/yyyy"));
        properties.Add("[DOC_EXP]", result.Diploma.ExperimentalPart);
        properties.Add("[TEACHER]", result.Diploma.TeacherName);
        properties.Add("[CONSULTANT]", string.Empty);

```

```

properties.Add("[DEP_HEAD]", "доц. д-р М. Лазарова");
properties.Add("[DEAN]", "проф. д-р Д. Гоцева");

int i = 1;
foreach (var item in result.Diploma.ContentCSV)
{
    string placeholder = string.Format("[DOC_{0}]", i);
    properties.Add(placeholder, string.Format("{0}. {1}", i++, item));
}

while (i <= 8)
{
    string placeholder = string.Format("[DOC_{0}]", i++);
    properties.Add(placeholder, string.Empty);
}

StringBuilder tags = new StringBuilder();

foreach (var tag in result.Diploma.Tags)
{
    tags.Append(tag.Text + ", ");
}

string tagsString = tags.Remove(tags.Length - 2, 2).ToString();

properties.Add("[DOC_TECH]", tagsString.ToString());

var filePath = DocXGenerator.Generate(properties,
this.Server.MapPath("~/Content/DocxFiles/Templates/DiplomaTemplate.docx"));
var fileName = string.Format("{0}.docx", properties["[USER_FNUMBER]"]);
return this.File(filePath, "application/vnd.openxmlformats-
officedocument.wordprocessingml.document", fileName);    }

/// <summary>
/// Returns details for a diploma by id of the diploma
/// </summary>
/// <param name="id">ID of the diploma</param>
/// <returns>Returns details for a diploma</returns>
public ActionResult Details(int? id)
{
    if (id == null)
    {
        return this.RedirectToAction("Diplomas", "Home");
    }

    var userID = this.User.Identity.GetUserId();
    var currentStudent = this.students.GetByUserId(userID).Include(s =>
s.SelectedDiploma).FirstOrDefault();

    if (currentStudent != null)
    {
        if (currentStudent.SelectedDiploma != null)
        {
            this.TempData["DiplomaIsSelected"] = "Вече сте избрали диплома";

            if (currentStudent.SelectedDiploma.Id == id)
            {
                this.TempData["IsOwnDiploma"] = true;
            }
        }
    }
}

```



```

        }
    }

    int intId = id ?? 0;
    var diploma = this.diplomas.GetObjectById(intId);

    if (diploma == null)
    {
        this.TempData["DiplomasNotFound"] = "Дипломата не бе намерена!";
        return this.RedirectToAction("Diplomas", "Home");
    }

    var result = this.diplomas.GetById(intId)
        .Include(d => d.Tags)
        .Include(d => d.Teacher)
        .To<DisplayDiplomaViewModel>().FirstOrDefault();

    result.ContentCSV = diploma.ContentCSV.Split(new char[] {
GlobalConstants.ContentSeparator },
System.StringSplitOptions.RemoveEmptyEntries).ToList();
    result.Tags = diploma.Tags.Select(t => new SelectListItem
    {
        Text = t.Name,
        Disabled = true,
        Selected = true,
        Value = t.Id.ToString()
    });

    var teacher = this.teachers.GetById(diploma.TeacherID).Include(t =>
t.User).FirstOrDefault();
    result.TeacherName = string.Format("{0} {1} {2}", teacher.User.ScienceDegree,
teacher.User.FirstName, teacher.User.LastName).Trim();

    var userId = this.User.Identity.GetUserId();
    var messages = this.messages.GetAll().Where(mes => mes.SelectedDiploma.Id ==
diploma.Id
                                                                && (mes.ResieverUserId ==
userId
                                                                || mes.SenderUserId ==
userId))

    .To<MessageViewModel>().ToList();
    foreach (var message in messages)
    {
        message.SenderUser = this.UserManager.FindById(message.SenderUserId);
        message.ResieverUser = this.UserManager.FindById(message.ResieverUserId);
    }

    this.ViewBag.Messages = messages;
    this.ViewBag.CurrentUserId = userId;

    return this.View(result);
}

[HttpPost]
[ValidateAntiForgeryToken]
[Authorize(Roles = GlobalConstants.StudentRoleName)]

```

```

        public ActionResult Select(int id)
        {
            var selectedDiploma = this.diplomas.GetById(id).Include(d =>
d.Teacher).FirstOrDefault();
            var student =
this.students.GetByUserId(this.User.Identity.GetUserId()).Include(s =>
s.User).FirstOrDefault();

            if (student != null)
            {
                if (student.Address == null || student.FNumber == 0 ||
student.User.PhoneNumber == null)
                {
                    this.TempData["UpdateProfile"] = "Моля попълнете личната си информация
преди да изберете дипломна работа!";
                    return this.RedirectToAction("Details", new { id = id });
                }
            }

            if (selectedDiploma.Teacher != null)
            {
                selectedDiploma.Teacher.Students.Count();
                selectedDiploma.Teacher.Students.Add(student);
            }

            if (selectedDiploma != null && student != null)
            {
                selectedDiploma.IsSelectedByStudent = true;
                student.SelectedDiploma = selectedDiploma;
                this.students.Save();
            }

            this.TempData["DiplomaIsSelectedSuccesfully"] = string.Format("Дипломата
\\{0}\\ е избрана успешно!", selectedDiploma.Title);

            return this.RedirectToAction("Diplomas");
        }

        /// <summary>
        /// DONE
        /// </summary>
        /// <param name="sortOrder">Order to be used for sorting</param>
        /// <param name="currentFilter">Filter</param>
        /// <param name="searchString">Searching string</param>
        /// <param name="page">Page that we are at</param>
        /// <returns>Sorted diplomas</returns>
        public ActionResult Diplomas(string sortOrder, string currentFilter, string
searchString, int? page)
        {
            this.ViewBag.CurrentSort = sortOrder;
            this.ViewBag.NameSortParm = string.IsNullOrEmpty(sortOrder) ? "name_desc" :
string.Empty;
            this.ViewBag.DateSortParm = sortOrder == "Date" ? "date_desc" : "Date";

            if (searchString != null)
            {
                page = 1;
            }
        }
    }

```

```

        else
        {
            searchString = currentFilter;
        }

        this.ViewBag.CurrentFilter = searchString;

        var diplomas = this.diplomas.GetAll().Where(d =>
!d.IsSelectedByStudent).To<CommonDiplomaViewModel>());

        if (!string.IsNullOrEmpty(searchString))
        {
            diplomas = diplomas.Where(s => s.Title.Contains(searchString)
|| s.Description.Contains(searchString));
        }

        if (diplomas.LongCount() <= 0)
        {
            this.TempData["DiplomasNotFound"] = "Не са намерени дипломи!";
        }
        else
        {
            switch (sortOrder)
            {
                case "name_desc":
                    diplomas = diplomas.OrderByDescending(s => s.Title);
                    break;
                case "Date":
                    diplomas = diplomas.OrderBy(s => s.CreatedOn);
                    break;
                case "date_desc":
                    diplomas = diplomas.OrderByDescending(s => s.CreatedOn);
                    break;
                default:
                    // Title ascending
                    diplomas = diplomas.OrderBy(s => s.Title);
                    break;
            }
        }

        int pageSize = GlobalConstants.PageSize;
        int pageNumber = page ?? 1;
        return this.View(diplomas.ToPagedList(pageNumber, pageSize));
    }

    /// <summary>
    /// Done
    /// </summary>
    /// <returns>Returns all teachers with their tags</returns>
    public ActionResult Teachers()
    {
        var teachers = this.teachers.GetAll().Include(t =>
t.Tags).To<TeacherViewModel>();

        return this.View(teachers);
    }

    /// <summary>

```

```

    /// DONE
    /// </summary>
    /// <param name="id">Id of the teacher</param>
    /// <returns>Returns a teacher from a given ID and hes/hers diplomas</returns>
    public ActionResult TeacherDetails(int? id)
    {
        if (id == null)
        {
            return this.RedirectToAction("Teachers", "Home");
        }

        int intId = id ?? 0;

        var teacher =
this.teachers.GetById(intId).To<TeacherViewModel>().FirstOrDefault();

        if (teacher == null)
        {
            this.TempData["Message"] = "Учителят не бе намерена!";
            return this.RedirectToAction("Teachers", "Home");
        }

        TeacherDiplomasViewModel teacherDiplomaVM = new TeacherDiplomasViewModel();
        teacherDiplomaVM.TeacherDetails = teacher;
        teacherDiplomaVM.Diplomas = this.diplomas.GetAll().Where(d => d.TeacherID ==
teacher.Id && !d.IsSelectedByStudent).ToList();

        return this.View(teacherDiplomaVM);
    }

    /// <summary>
    /// Done
    /// </summary>
    /// <param name="id">Id of the tag</param>
    /// <param name="teacherId">Id of the teacher</param>
    /// <param name="sortOrder">Order to be used for sorting</param>
    /// <param name="currentFilter">Filter</param>
    /// <param name="searchString">Searching string</param>
    /// <param name="page">Page that we are at</param>
    /// <returns>Returns diplomas by tag</returns>
    public ActionResult Tag(int? id, int? teacherId, string sortOrder, string
currentFilter, string searchString, int? page)
    {
        if (id == null)
        {
            return this.RedirectToAction("Diplomas");
        }

        this.ViewBag.CurrentSort = sortOrder;
        this.ViewBag.NameSortParm = string.IsNullOrEmpty(sortOrder) ? "name_desc" :
string.Empty;
        this.ViewBag.DateSortParm = sortOrder == "Date" ? "date_desc" : "Date";

        if (searchString != null)
        {
            page = 1;
        }
        else

```

```

    {
        searchString = currentFilter;
    }

    this.ViewBag.CurrentFilter = searchString;

    int intId = id ?? 0;
    var tag = this.tags.GetObjectById(intId);

    var diplomas = this.diplomas.GetAll()
        .Where(d => !d.IsSelectedByStudent && (d.TeacherID == teacherId ||
teacherId == null))
        .To<CommonDiplomaViewModel>()
        .ToList()
        .Where(d => d.Tags.Contains(tag));

    if (!string.IsNullOrEmpty(searchString))
    {
        diplomas = diplomas.Where(s => s.Title.Contains(searchString)
|| s.Description.Contains(searchString));
    }

    if (diplomas.LongCount() <= 0)
    {
        this.TempData["DiplomasNotFound"] = "Не са намерени дипломи!";
    }
    else
    {
        switch (sortOrder)
        {
            case "name_desc":
                diplomas = diplomas.OrderByDescending(s => s.Title);
                break;
            case "Date":
                diplomas = diplomas.OrderBy(s => s.CreatedOn);
                break;
            case "date_desc":
                diplomas = diplomas.OrderByDescending(s => s.CreatedOn);
                break;
            default:
                // Title ascending
                diplomas = diplomas.OrderBy(s => s.Title);
                break;
        }
    }

    int pageSize = GlobalConstants.PageSize;
    int pageNumber = page ?? 1;
    return this.View(diplomas.ToPagedList(pageNumber, pageSize));
}

public JsonResult GetAllTags(string searchTerm, int pageSize, int pageNumber)
{
    var resultList = new List<TagViewModel>();
    if (!string.IsNullOrEmpty(searchTerm))
    {
        resultList = this.tags.GetAll().To<TagViewModel>()
            .Where(n => n.text.ToLower()

```

```

        .Contains(searchTerm.ToLower()))
        .ToList();
    }
    else
    {
        resultList = this.tags.GetAll().To<TagViewModel>().ToList();
    }

    var results = resultList.OrderBy(t => t.text).Skip((pageNumber - 1) *
    pageSize)
        .Take(pageSize)
        .ToList();

    return this.Json(new { Results = results, Total = resultList.Count },
    JsonRequestBehavior.AllowGet);
}

[ValidateAntiForgeryToken]
public ActionResult SendMessage(FormCollection formCollection, int diplomaId, int
teacherId)
{
    string message = formCollection["message"];
    if (this.ModelState.IsValid)
    {
        var senderId = this.User.Identity.GetUserId();
        var resieverId = this.teachers.GetById(teacherId)
            .Include(s => s.User)
            .Select(s => s.User.Id)
            .FirstOrDefault();

        Diploma selectedDiploma =
        this.diplomas.GetById(diplomaId).FirstOrDefault();

        var messageEntity = new Message()
        {
            SenderUserId = senderId,
            ResieverUserId = resieverId,
            MessageSend = message,
            SelectedDiploma = selectedDiploma,
            IsRead = false,
        };

        this.messages.Create(messageEntity);

        return this.RedirectToAction("Details", new { Id = diplomaId });
    }
}
}

```

ManageController.cs

```

namespace DDS.Web.Controllers
{
    using System.Data.Entity;
    using System.Linq;
    using System.Threading.Tasks;

```

```

using System.Web;
using System.Web.Mvc;
using Common;
using DDS.Web.ViewModels.Manage;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.Owin;
using Microsoft.Owin.Security;
using Services.Data.Interfaces;

[Authorize]
public class ManageController : BaseController
{
    // Used for XSRF protection when adding external logins
    private const string XsrfKey = "XsrfId";

    private readonly IStudentsService students;
    private readonly ITeachersService teachers;
    private ApplicationSignInManager signInManager;
    private ApplicationUserManager userManager;

    public ManageController(IStudentsService students, ITeachersService teachers)
    {
        this.students = students;
        this.teachers = teachers;
    }

    public ManageController(ApplicationUserManager userManager,
ApplicationSignInManager signInManager)
    {
        this.UserManager = userManager;
        this.SignInManager = signInManager;
    }

    public enum ManageMessageId
    {
        AddPhoneSuccess,
        ChangePasswordSuccess,
        SetTwoFactorSuccess,
        SetPasswordSuccess,
        RemoveLoginSuccess,
        RemovePhoneSuccess,
        Error
    }

    public ApplicationSignInManager SignInManager
    {
        get
        {
            return this.signInManager ??
this.HttpContext.GetOwinContext().Get<ApplicationSignInManager>();
        }

        private set
        {
            this.signInManager = value;
        }
    }
}

```

```

public ApplicationUserManager UserManager
{
    get
    {
        return this.userManager ??
this.HttpContext.GetOwinContext().GetUserManager<ApplicationUserManager>();
    }

    private set
    {
        this.userManager = value;
    }
}

private IAuthenticationManager AuthenticationManager =>
this.HttpContext.GetOwinContext().Authentication;

// GET: /Manage/Index
public async Task<ActionResult> Index(ManageMessageId? message)
{
    this.ViewBag.StatusMessage = message == ManageMessageId.ChangePasswordSuccess
        ? "Паролата Ви е променена."
        : message ==
ManageMessageId.SetPasswordSuccess
        ? "Паролата Ви е зададена."
        : message ==
ManageMessageId.SetTwoFactorSuccess
        ? "Your two-factor authentication
provider has been set."
        : message ==
ManageMessageId.Error
        ? "Възникна грешка."
        : message ==
ManageMessageId.AddPhoneSuccess
        ? "Вашият телефон бе
добавен."
        : message ==
ManageMessageId.RemovePhoneSuccess
        ? "Вашия
телефон бе изтрит."
        : string.Empty;

    var userId = this.User.Identity.GetUserId();
    var user = this.UserManager.FindById(userId);

    var model = new IndexViewModel
    {
        UserId = userId,
        HasPassword = this.HasPassword(),
        PhoneNumber = await this.UserManager.GetPhoneNumberAsync(userId),
        TwoFactor = await this.UserManager.GetTwoFactorEnabledAsync(userId),
        Logins = await this.UserManager.GetLoginsAsync(userId),
        BrowserRemembered = await
this.AuthenticationManager.TwoFactorBrowserRememberedAsync(userId),
        ScienceDegree = user.ScienceDegree,
    };

    if (this.UserManager.IsInRole(model.UserId, GlobalConstants.StudentRoleName))

```



```

        {
            var student = this.students.GetByUserId(model.UserId).Include(s =>
s.SelectedDiploma).FirstOrDefault();

            if (student.SelectedDiploma != null)
            {
                model.DiplomaId = student.SelectedDiploma.Id;
                this.TempData["HasDiploma"] = true;
            }

            model.Address = student.Address;
            model.FNumber = student.FNumber;
            model.IsStudent = true;
            this.TempData["Student"] = true;
        }
        else
        {
            model.IsStudent = false;
        }

        return this.View(model);
    }

    // POST: /Manage/SaveInfo
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult SaveInfo(IndexViewModel model)
    {
        if (this.ModelState.IsValid)
        {
            var user = this.UserManager.FindById(model.UserId);

            user.PhoneNumber = model.PhoneNumber;
            user.ScienceDegree = model.ScienceDegree;
            this.UserManager.Update(user);

            var student = this.students.GetByUserId(model.UserId).Where(s =>
!s.IsDeleted).FirstOrDefault();
            if (student != null)
            {
                user.Student = student;
                user.Student.FNumber = model.FNumber;
                user.Student.Address = model.Address;
                this.students.Save();
            }

            return this.RedirectToAction("Index", "Home", null);
        }

        return this.View("Index", model);
    }

    // GET: /Manage/ChangePassword
    public ActionResult ChangePassword()
    {
        return this.View();
    }

```

```

// POST: /Manage/ChangePassword
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<ActionResult> ChangePassword(ChangePasswordViewModel model)
{
    if (!this.ModelState.IsValid)
    {
        return this.View(model);
    }

    var result =
        await
            this.UserManager.ChangePasswordAsync(
                this.User.Identity.GetUserId(),
                model.OldPassword,
                model.NewPassword);
    if (result.Succeeded)
    {
        var user = await
this.UserManager.FindByIdAsync(this.User.Identity.GetUserId());
        if (user != null)
        {
            await this.SignInManager.SignInAsync(user, isPersistent: false,
rememberBrowser: false);
        }

        return this.RedirectToAction("Index", new { Message =
ManageMessageId.ChangePasswordSuccess });
    }

    this.AddErrors(result);
    return this.View(model);
}

// GET: /Manage/SetPassword
public ActionResult SetPassword()
{
    return this.View();
}

// POST: /Manage/SetPassword
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<ActionResult> SetPassword(SetPasswordViewModel model)
{
    if (this.ModelState.IsValid)
    {
        var result = await
this.UserManager.AddPasswordAsync(this.User.Identity.GetUserId(), model.NewPassword);
        if (result.Succeeded)
        {
            var user = await
this.UserManager.FindByIdAsync(this.User.Identity.GetUserId());
            if (user != null)
            {
                await this.SignInManager.SignInAsync(user, isPersistent: false,
rememberBrowser: false);
            }
        }
    }
}

```

```

        return this.RedirectToAction("Index", new { Message =
ManageMessageId.SetPasswordSuccess });
    }

    this.AddErrors(result);
}

// If we got this far, something failed, redisplay form
return this.View(model);
}

protected override void Dispose(bool disposing)
{
    if (disposing && this.userManager != null)
    {
        this.userManager.Dispose();
        this.userManager = null;
    }

    base.Dispose(disposing);
}

private void AddErrors(IdentityResult result)
{
    foreach (var error in result.Errors)
    {
        this.ModelState.AddModelError(string.Empty, error);
    }
}

private bool HasPassword()
{
    var user = this.UserManager.FindById(this.User.Identity.GetUserId());
    return user?.PasswordHash != null;
}

private bool HasPhoneNumber()
{
    var user = this.UserManager.FindById(this.User.Identity.GetUserId());
    return user?.PhoneNumber != null;
}
}
}

```

ManageDiplomasController.cs

```

namespace DDS.Web.Controllers
{
    using System;
    using System.Collections.Generic;
    using System.Data.Entity;
    using System.Linq;
    using System.Web;
    using System.Web.Mvc;
    using Common;
    using Data.Models;

```

```

using Infrastructure.Mapping;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.Owin;
using PagedList;
using Services.Data.Interfaces;
using ViewModels.ManageDiplomas;
using ViewModels.Shared;

[Authorize(Roles = GlobalConstants.TeacherRoleName)]
public class ManageDiplomasController : BaseController
{
    private readonly ITeachersService teachers;
    private readonly IDiplomasService diplomas;
    private readonly IStudentsService students;
    private readonly ITagsService tags;
    private readonly IMessagesService messages;

    private ApplicationUserManager userManager;

    public ManageDiplomasController(
        ITeachersService teachers,
        IDiplomasService diplomas,
        IStudentsService students,
        ITagsService tags,
        IMessagesService messages)
    {
        this.teachers = teachers;
        this.diplomas = diplomas;
        this.students = students;
        this.tags = tags;
        this.messages = messages;
    }

    public ApplicationUserManager UserManager
    {
        get
        {
            return this.userManager ??
this.Request.GetOwinContext().GetUserManager<ApplicationUserManager>();
        }

        private set
        {
            this.userManager = value;
        }
    }

    // GET: ManageDiplomas
    public ActionResult Index(string sortOrder, string currentFilter, string
searchString, int? page)
    {
        this.ViewBag.CurrentSort = sortOrder;
        this.ViewBag.NameSortParm = string.IsNullOrEmpty(sortOrder) ? "name_desc" :
string.Empty;
        this.ViewBag.DateSortParm = sortOrder == "Date" ? "date_desc" : "Date";

        if (searchString != null)
        {

```

```

        page = 1;
    }
    else
    {
        searchString = currentFilter;
    }

    this.ViewBag.CurrentFilter = searchString;

    var teacher =
this.teachers.GetByUserId(this.User.Identity.GetUserId()).FirstOrDefault();
    var diplomas = this.diplomas.GetByTeacherId(teacher.Id)
        .To<CommonDiplomaViewModel>();

    if (!string.IsNullOrEmpty(searchString))
    {
        if (searchString.ToLower() == "одобрени")
        {
            diplomas = diplomas.Where(d => d.IsApprovedByLeader ||
d.IsApprovedByHead);
        }
        else if (searchString.ToLower() == "избрани")
        {
            diplomas = diplomas.Where(d => d.IsSelectedByStudent);
        }
        else if (searchString.ToLower() == "свободни")
        {
            diplomas = diplomas.Where(d => !d.IsSelectedByStudent);
        }
        else
        {
            diplomas = diplomas.Where(s => s.Title.Contains(searchString) ||
s.Description.Contains(searchString));
        }
    }

    if (diplomas.LongCount() <= 0)
    {
        this.TempData["Message"] = "Не са намерени дипломи!";
    }
    else
    {
        switch (sortOrder)
        {
            case "name_desc":
                diplomas = diplomas.OrderByDescending(s => s.Title);
                break;

            case "Date":
                diplomas = diplomas.OrderBy(s => s.CreatedOn);
                break;

            case "date_desc":
                diplomas = diplomas.OrderByDescending(s => s.CreatedOn);
                break;

            default:
                // Title ascending

```

```

        diplomas = diplomas.OrderBy(s => s.Title);
        break;
    }
}

int pageSize = 5;
int pageNumber = page ?? 1;
return this.View(diplomas.ToPagedList(pageNumber, pageSize));
}

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Approve(int id)
{
    var diploma = this.diplomas.GetObjectById(id);
    diploma.IsApprovedByLeader = true;

    this.diplomas.Save();

    return this.RedirectToAction("Details", "ManageDiplomas", new { @id = id });
}

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Release(int id)
{
    var diploma = this.diplomas.GetObjectById(id);
    diploma.IsSelectedByStudent = false;
    diploma.IsApprovedByLeader = false;
    diploma.IsApprovedByHead = false;
    this.diplomas.Save();

    var student = this.students.GetAll().Where(s => s.SelectedDiploma.Id ==
id).FirstOrDefault();
    student.SelectedDiploma = null;

    var teacher =
this.teachers.GetByUserId(this.User.Identity.GetUserId()).FirstOrDefault();
    teacher.Students.Remove(student);

    this.students.Save();
    this.teachers.Save();

    return this.RedirectToAction("Details", "ManageDiplomas", new { @id = id });
}

[HttpGet]
public ActionResult Create()
{
    return this.View();
}

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Create(CreateDiplomaViewModel viewModel)
{
    if (this.ModelState.IsValid)
    {

```

```

        var teacher =
this.teachers.GetByUserId(this.User.Identity.GetUserId()).FirstOrDefault();

        // create diploma
var diploma = new Diploma()
{
    Teacher = teacher,
    Title = viewModel.Title,
    Description = viewModel.Description,
    ExperimentalPart = viewModel.ExperimentalPart,
    ContentCSV = string.Join(GlobalConstants.ContentSeparator.ToString(),
viewModel.ContentCSV),
};

var listOfTags = new List<Tag>();
foreach (var viewModelTag in viewModel.TagsNames)
{
    var tagId = 0;
    Tag tag;
    if (int.TryParse(viewModelTag, out tagId))
    {
        tag = this.tags.GetObjectById(tagId);
    }
    else
    {
        tag = this.tags.EnsureCategory(viewModelTag);
    }

    listOfTags.Add(tag);
}

diploma.Tags = listOfTags;

// add diploma to teacher
this.teachers.AddDiploma(teacher.Id, diploma);
this.TempData["Message"] = "Дипломата е създадена!";

return this.RedirectToAction("Index", "ManageDiplomas");
}

return this.View(viewModel);
}

public ActionResult Edit(int? id)
{
    if (id == null)
    {
        this.TempData["Message"] = "Дипломата не е намерена!";
        return this.RedirectToAction("Index", "ManageDiplomas");
    }

    int intId = id ?? 0;

    var diploma = this.diplomas.GetObjectById(intId);
    var diplomaModel = this.diplomas.GetById(intId)
        .Include(d => d.Tags)

```

```

.To<DisplayDiplomaViewModel>().FirstOrDefault();
/*this.diplomas.GetById(intId).To<EditDiplomaViewModel>().FirstOrDefault();*/

    if (diploma == null)
    {
        this.TempData["Message"] = "Дипломата не бе намерена!";
        return this.RedirectToAction("Index", "ManageDiplomas");
    }

    diplomaModel.ContentCSV = diploma.ContentCSV.Split(new char[] {
GlobalConstants.ContentSeparator },
System.StringSplitOptions.RemoveEmptyEntries).ToList();
    diplomaModel.Tags = diploma.Tags.Select(t => new SelectListItem
    {
        Text = t.Name,
        Disabled = false,
        Selected = true,
        Value = t.Id.ToString()
    });
    return this.View(diplomaModel);
}

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Edit(DisplayDiplomaViewModel viewModel)
{
    if (this.ModelState.IsValid)
    {
        var diploma = this.diplomas.GetById(viewModel.Id).FirstOrDefault();
        diploma.Title = viewModel.Title.Trim();
        diploma.Description = viewModel.Description.Trim();
        diploma.ExperimentalPart = viewModel.ExperimentalPart.Trim();
        diploma.ContentCSV =
string.Join(GlobalConstants.ContentSeparator.ToString(), viewModel.ContentCSV);

        diploma.Tags.Clear();
        this.diplomas.Save();

        var tags = new List<Tag>();

        foreach (var tag in viewModel.TagsNames)
        {
            var tagIndex = 0;

            if (int.TryParse(tag, out tagIndex))
            {
                tags.Add(this.tags.GetById(tagIndex).FirstOrDefault());
            }
            else
            {
                tags.Add(this.tags.EnsureCategory(tag));
            }
        }

        diploma.Tags = tags;
        this.diplomas.Save();
    }
}

```



```

        this.TempData["Message"] = "Дипломата е променена!";
        return this.RedirectToAction("Index", "ManageDiplomas");
    }

    return this.View(viewModel);
}

public ActionResult Details(int? id)
{
    if (id == null)
    {
        return this.RedirectToAction("Index", "ManageDiplomas");
    }

    int intId = id ?? 0;
    var diploma = this.diplomas.GetObjectById(intId);
    if (diploma == null)
    {
        this.TempData["Message"] = "Дипломата не бе намерена!";
        return this.RedirectToAction("Index", "ManageDiplomas");
    }

    var result = new StudentDiplomaViewModel();
    var studentDetails = this.students.GetAll()
        .Where(s => s.SelectedDiploma.Id ==
diploma.Id)
        .Include(s => s.User)
        .To<SimpleStudentViewModel>()
        .FirstOrDefault();

    result.Student = studentDetails;
    var diplomaModel = this.diplomas.GetAll()
        .Where(d => d.Id == intId)
        .Include(d => d.Teacher)
        .Include(d => d.Tags)
        .To<DisplayDiplomaViewModel>();

    result.Diploma = diplomaModel.FirstOrDefault();
    result.Diploma.ContentCSV = diploma.ContentCSV.Split(new char[] {
GlobalConstants.ContentSeparator },
System.StringSplitOptions.RemoveEmptyEntries).ToList();
    result.Diploma.TeacherID = diploma.TeacherID;
    result.Diploma.Tags = diploma.Tags.Select(t => new SelectListItem
    {
        Text = t.Name,
        Disabled = true,
        Selected = true,
        Value = t.Id.ToString()
    });
    var teacher = this.teachers.GetById(diploma.TeacherID).Include(t =>
t.User).FirstOrDefault();
    result.Diploma.TeacherName = string.Format("{0} {1} {2}",
teacher.User.ScienceDegree, teacher.User.FirstName, teacher.User.LastName).Trim();
    var userId = this.User.Identity.GetUserId();
    var messages = this.messages.GetAll().Where(mes => mes.SelectedDiploma.Id ==
diploma.Id && (mes.ResieverUserId == userId || mes.SenderUserId == userId))

    .To<MessageViewModel>().ToList();

```

```

        foreach (var message in messages)
        {
            message.SenderUser = this.UserManager.FindById(message.SenderUserId);
            message.ResieverUser = this.UserManager.FindById(message.ResieverUserId);
        }
        result.MessageBox = messages;
        result.CurrentUserId = userId;
        return this.View(result);
    }

    // GET: Deleted
    public ActionResult Deleted(string sortOrder, string currentFilter, string
searchString, int? page)
    {
        this.ViewBag.CurrentSort = sortOrder;
        this.ViewBag.NameSortParm = string.IsNullOrEmpty(sortOrder) ? "name_desc" :
string.Empty;
        this.ViewBag.DateSortParm = sortOrder == "Date" ? "date_desc" : "Date";

        if (searchString != null)
        {
            page = 1;
        }
        else
        {
            searchString = currentFilter;
        }
        this.ViewBag.CurrentFilter = searchString;
        var teacher =
this.teachers.GetByUserId(this.User.Identity.GetUserId()).FirstOrDefault();
        var diplomas = this.diplomas.GetDeleted()
                                .Where(x => x.Teacher.Id == teacher.Id)
                                .To<CommonDiplomaViewModel>();
        if (!string.IsNullOrEmpty(searchString))
        {
            diplomas = diplomas.Where(s => s.Title.Contains(searchString)
|| s.Description.Contains(searchString));
        }
        if (diplomas.LongCount() <= 0)
        {
            this.TempData["Message"] = "Няма намерени дипломи!";
            this.TempData["NotFound"] = true;
        }
        else
        {
            switch (sortOrder)
            {
                case "name_desc":
                    diplomas = diplomas.OrderByDescending(s => s.Title);
                    break;

                case "Date":
                    diplomas = diplomas.OrderBy(s => s.CreatedOn);
                    break;

                case "date_desc":
                    diplomas = diplomas.OrderByDescending(s => s.CreatedOn);
                    break;
            }
        }
    }

```

```

        default:
            // Title ascending
            diplomas = diplomas.OrderBy(s => s.Title);
            break;
    }
}
int pageSize = 5;
int pageNumber = page ?? 1;
return this.View(diplomas.ToPagedList(pageNumber, pageSize));
}

[ValidateAntiForgeryToken]
public ActionResult Delete(int? id)
{
    if (id == null)
    {
        return this.RedirectToAction("Index", "ManageDiplomas");
    }
    int intId = id ?? 0;
    var diploma = this.diplomas.GetObjectById(intId);

    if (diploma == null)
    {
        this.TempData["Message"] = "Дипломата не бе намерена!";
        return this.RedirectToAction("Index", "ManageDiplomas");
    }
    this.diplomas.Delete(diploma);
    this.TempData["Message"] = "Дипломата е изтрита!";
    return this.RedirectToAction("Index", "ManageDiplomas");
}

// Get: HardDelete
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult HardDelete(int? id)
{
    if (id == null)
    {
        return this.RedirectToAction("Deleted");
    }

    int intId = id ?? 0;
    var diploma = this.diplomas.GetDeleted().FirstOrDefault(d => d.Id == intId);

    if (diploma == null)
    {
        this.TempData["NotFound"] = true;
        this.TempData["Message"] = "Дипломата не бе намерена!";
        return this.RedirectToAction("Deleted");
    }
    this.diplomas.HardDelete(diploma);
    return this.RedirectToAction("Deleted");
}

public ActionResult HardDeleteAll()
{

```

```

        var teacherID =
this.teachers.GetByUserId(this.User.Identity.GetUserId()).FirstOrDefault().Id;
        var diplomas = this.diplomas.GetDeleted().Where(d => d.TeacherID ==
teacherID).ToList();
        foreach (var diploma in diplomas)
        {
            this.diplomas.HardDelete(diploma);
        }
        return this.RedirectToAction("Index");
    }

    public ActionResult Return(int? id)
    {
        if (id == null)
        {
            return this.RedirectToAction("Index", "ManageDiplomas");
        }
        int intId = id ?? 0;
        var diploma = this.diplomas.GetDeleted().FirstOrDefault(d => d.Id == id);
        if (diploma == null)
        {
            this.TempData["Message"] = "Дипломата не бе намерена!";
            return this.RedirectToAction("Index", "ManageDiplomas");
        }
        this.diplomas.UnDelete(diploma);
        this.TempData["Message"] = "Дипломата е върната!";
        return this.RedirectToAction("Index", "ManageDiplomas");
    }
    [ValidateAntiForgeryToken]
    public ActionResult SendMessage(FormCollection formCollection, int diplomaId, int
studentId)
    {
        string message = formCollection["message"];
        if (this.ModelState.IsValid)
        {
            var senderId = this.User.Identity.GetUserId();
            var resieverId = this.students.GetById(studentId)
                .Include(s => s.User)
                .Select(s => s.User.Id)
                .FirstOrDefault();
            var senderUser = this.UserManager.FindById(senderId);
            var resieverUser = this.UserManager.FindById(resieverId);
            Diploma selectedDiploma =
this.diplomas.GetById(diplomaId).FirstOrDefault();
            var messageEntity = new Message()
            {
                SenderUserId = senderId,
                ResieverUserId = resieverId,
                MessageSend = message,
                SelectedDiploma = selectedDiploma,
                IsRead = false,
            };
            this.messages.Create(messageEntity);
        }
        return this.RedirectToAction("Details", new { Id = diplomaId });
    }
}
}
}

```