**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 3](#_Toc460869143)

[Глава 1. Анализ на подобни системи. Цели и задачи. 4](#_Toc460869144)

[1.1 Какво представлява процесът по избор на дипломно задание за момента. 4](#_Toc460869145)

[1.2 Цел и задачи 4](#_Toc460869146)

[Глава 2. Обзор на използваните технологии. 7](#_Toc460869147)

[2.1 ASP.NET MVC 7](#_Toc460869148)

[2.1.1 ASP.NET MVC 8](#_Toc460869149)

[2.1.2 Изглед (View) 8](#_Toc460869150)

[2.1.3 Модел (Model) 9](#_Toc460869151)

[2.1.4 Контролер (Controller) 9](#_Toc460869152)

[2.2 C# 10](#_Toc460869153)

[2.3 MS SQL - база данни 11](#_Toc460869154)

[2.3.1 Релационна база данни 11](#_Toc460869155)

[2.3.2 Отношение 11](#_Toc460869156)

[2.3.3 Нормализация на базата данни 12](#_Toc460869157)

[2.4 jQuery 14](#_Toc460869158)

[2.5 CSS 15](#_Toc460869159)

[Глава 3. Проектиране на системата. 16](#_Toc460869160)

[3.1 Съставяне на моделите за базата данни 16](#_Toc460869161)

[3.2 Съставяне на хранилище за елементите на базата (repository) 17](#_Toc460869162)

[3.3 Създаване на слой с услуги (Service layer pattern) 18](#_Toc460869163)

[3.4 Създаване на подходящи контролери и изгледи 19](#_Toc460869164)

[3.4.1 Регистрация на потребители 20](#_Toc460869165)

[3.4.2 Избор на роли в приложението 20](#_Toc460869166)

[3.4.3 Управление на дипломни задания от преподавателя 23](#_Toc460869167)

[Глава 4. Програмна реализация. 24](#_Toc460869168)

[4.1 Създаване на базата данни 25](#_Toc460869169)

[4.2 Създаване на хранилище 28](#_Toc460869170)

[4.3 Създаване на услуги (services) 29](#_Toc460869171)

[4.4 Autofac – предназначение и настройки 29](#_Toc460869172)

[4.5 Създаване на контролери 30](#_Toc460869173)

[4.6 Създаване на изгледи-модел 31](#_Toc460869174)

[4.7 Създаване на изгледи 31](#_Toc460869175)

[4.8 Automapper – настройки и приложение 35](#_Toc460869176)

[4.9 Генериране на Docx файлове 37](#_Toc460869177)

[Глава 5. Ръководство на потребителя 38](#_Toc460869178)

[5.1 Ръководство за потребител тип студент. 39](#_Toc460869179)

[5.2 Ръководство за потребител тип ръководител 40](#_Toc460869180)

[5.3 Ръководство за потребител тип администратор 41](#_Toc460869181)

[Заключение 43](#_Toc460869182)

[Списък на използвани литературни източници 44](#_Toc460869183)

[Приложения 45](#_Toc460869184)

# Увод

Намирането на подходяща диплома е важна стъпка от етапа на обучение. Когато студентът намери дипломното задание, което търси, 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. Приложението има възможност за търсене Ускорява процеса по създаване, управление на дипломни задания. Позволява намирането на дубликати.

Уводът съдържа резюме на дейностите извършени от дипломанта в процеса на дипломното проектиране.

Управление на дипломните задания (менажиране)

Лесен избор на дипломна работа

Намиране на подходящ ръководител или различни преподаватели за консултации

Повечето избори стават от сайт със статични теми като този

https://www.fmi.uni-sofia.bg/lecturers/softeng/milenp/42243543c438-437430-43443843f43b43e43c43d438-44043043143e442438

# 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.

## 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 не взети 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.

## 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 приемните им времена.

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;
* лесно управление на потребителите;
* възможност за намиране на дубликати или вече използвани теми през други години.

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.

# Обзор на използваните технологии.

## ASP.NET MVC



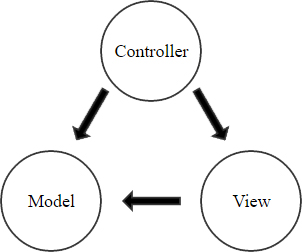
**ASP.NET** е новото поколение уеб рамка (framework), разработена от Microsoft. За първи път е публикуван през януари 2002 година с версия 1.0 на .NET Framework, и е наследник на Microsoft Active Server Pages (ASP) технология, но да не се бърка, че не е подобрена версия на ASP. ASP.NET е изградена въз основа на Common Language Runtime (CLR), което позволява на програмистите да пишат ASP.NET код като използват .NET език по избор. 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.

ASP.NET цели производителност спрямо останалите скрипт-базирани технологии (включително класическия ASP) като компилира сървърно кода в един или повече DLL файлове на Уеб сървъра. 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 сървъра, но това забавяне не би се появило отново преди следваща промяна.

ASP.NET предлага три рамки за създаване на уеб приложения: Web Forms, 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.

### 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 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

### 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.

### 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.

### 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.
* Онстраняване 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 изглет от тип ASP.NET.

Контролера е клас който се наследява от базовия клас System.Web.Mvc.Controller. Всеки публичен метод в контролера е показан като "controller action". 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.

## 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 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 Майкрософт като част от тяхната съвременна платформа за разработка и изпълнение на приложения .NET Framework, езикът е силно разпространен сред Microsoft-ориентираните 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 останали 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 Common Language Runtime (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 промени върху различни хардуерни платформи и операционни системи. Най-често C# програмите се изпълняват върху MS Windows, но .NET Framework и CLR се поддържа 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.



## MS SQL - база данни

### 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 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).

### 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 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 копие 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 другата страна, всеки от първичните ключове на изходните таблици играе ролята на външен ключ за свързващата таблица.

### 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 структурата на базата от данни, основаваща се на взаимозависимостта между данните. Структурата на таблиците се трансформират, 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 пурвичния 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

Третата нормална форма (3НФ) проверява за транзитивна зависимост. Транзитивната зависимост е подобна на частичната зависимост по това, че и двете се отнасят до атрибути, които не са напълно зависими от първичният. Зависимостта се зачита като транзитивна когато атрибут1 е зависим от атрибут2, който е зависим от първичният 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 преразгледа внимателно и задълбочено за всяка нова таблица, която се разработва. Решенията, които се взимат по време на фазатаба на изработване на дизайна играят съществена роля при цялосната цена (време), използваемост и ефективност на крайното приложение. Също така е добре да се преразгледа денормализацията на базата, за да се направи финна настройка на приложението, когато то се разрастне и се увеличи потреблението.

## jQuery



**JQuery** е 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. JQuery се използва 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, drag and drop приложения, събития, управление на бисквитки, модални прозорци и други.

## CSS

**CSS** е съкращение от **Cascading Style Sheets** 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.

# Проектиране на системата.

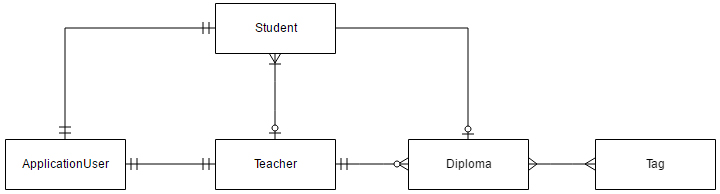
В настоящата дипломна работа ще предложа решение за изграждане на софтуерна система за разпределение и съставяне на дипломни задания. Целта на тази система е да спомогне абсолвентите и ръководителите на дипломни задания, като предложи едно лесно за употреба web базирано приложение.

## 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 начина, по който отделните елементи са свързани помежду си.



Фигура 3‑2 Схема на обектите за базата данни

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

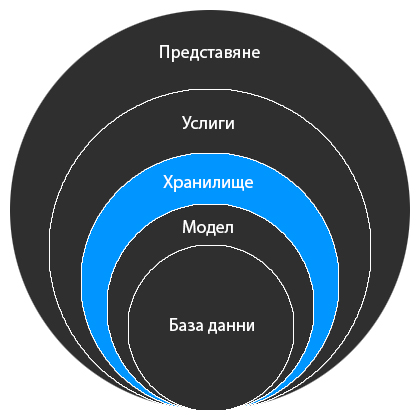
Класовете Student и Teacher са класове, в които се запазва допълнителна информация за потребителя. Изнасяйки ги в отделни класове, ще се намали отговорността на ApplicationUser класа и в същото време един потребител може да се възползва от архитектурата и на двата класа (един потребител ще може да е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

* В класа Student се запазва всякаква информация касаеща отделния студент (факултетен номер, коя диплома е избрал).
* Класът Teacher се съдържат полета с информация за конкретен учител. В този клас се записват дипломите, които преподавателя 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, която да определя дипломната работа и да съдържа проста информация като име, дата на създаване, както и дипломи, които използват този таг. Тези модели могат да се използват и за определяне на сферите на интерес (технологиите), с които даден преподавател е обвързан/запознат.

## 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. Тестване на бизнес логиката става по-лесно, като по този начин не сме зависими от базата данни и можем да работи отделно от вида и типа й. Улеснява се достъпът до ресурсите, когато имаме достъп до тях 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)
* Взимане 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

## Създаване на слой с услуги (Service layer pattern)



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

**Service layer** е 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).

Услугите, които са необходими са директно свързани с типовете модели, които има приложението.

Услуга 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

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

Услуга 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

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

* Избор на таг по ID
* Избор на таг по име
* Добавяне на таг

## Създаване на подходящи контролери и изгледи



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

За да могат да се избера 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 свой задачи (действия), които може да изпълнява

### Регистрация на потребители

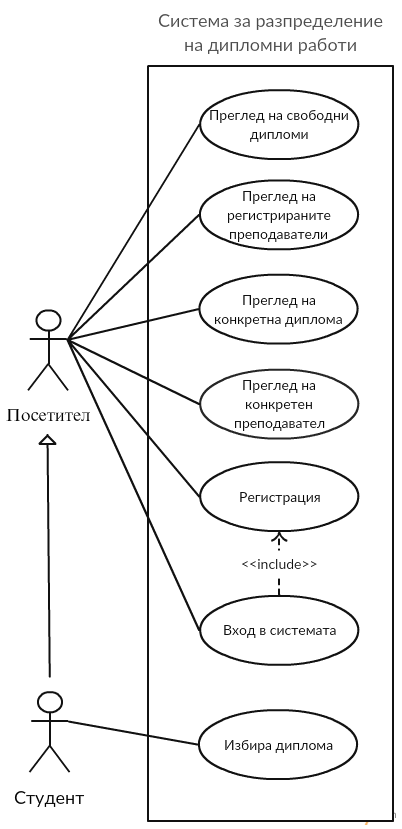
За да се регистрира нов потребител, web приложението може да използва предоставената от Microsoft архитектура. Класовете, които се използват на готово, включват основна информация за потребителите. В това число и контролер за управление на потребителя. Процесите по регистрация и вход са основни за всяко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

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, които да е обвързан с новия потребител.

### Избор на роли в приложението

Приложението трябва да има основни роли. Потребители, които да управлява самото приложение (**администратори**). Тези потребители са няколко за целия проект. За създаване на тази функционалност трябва да има контролер, които да бъде достъпен само за тях (администраторите на системата). Функционалностите, които са необходими 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).

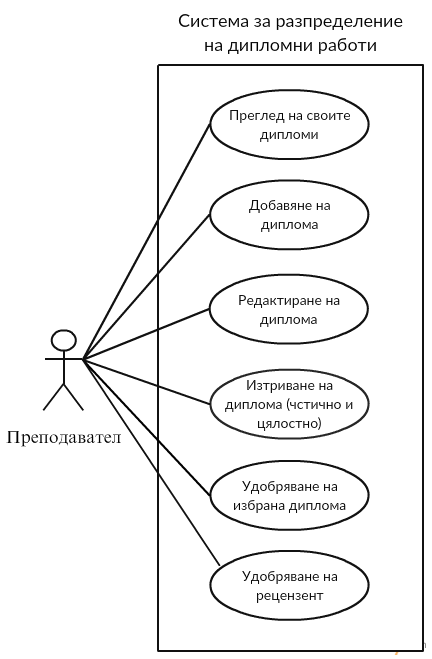
Приложените трябва да има потребители от тип студенти. Студентите са уникални потребители със способността си да избират вече създадени дипломи. Потребителите от този тип ще могат да разглеждат списъци с дипломни работи, сортирани по различни показатели. Студентите ще могат да избират тема по части от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



Фигура 3‑6Use case диаграма на посетител и студент

На фиг. 3-1 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 не регистрираните потребители. На фигурата се вижда, че системата трябва да може да се използва и като источник на информация. Не регистрираните 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.

Приложените трябва да има потребители от тип преподаватели, които да създават дипломни теми. Тези потребители трябва да могат да променият 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 одобрение, а не одобрените теми трябва да се преразгледат.



Фигура 3‑7 Use case диаграма за преподавател

На фиг. 3-2 е показана use case диаграма за потребител от тип преподавател. Показани са основните функции, които може да извърши преподавателя.

### Управление на дипломни задания от преподавателя

Процесът на управление на дипломни задания трябва да е сравнително лесен. Всеки преподавател трябва да може да създава, редактира и одобрява своите задания с няколко натискания на бутони. За целта цялата логика по тези процеси трябва да се разположи на едно място. Одобрените дипломи трябва да са лесно достъпни и лесно забелязвани. След като дадено задание бива одобрено, то не би трябвало да може да бъде редактирано.

# Програмна реализация.

В тази глава ще разледаме конкретна имплементация на приложени 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:

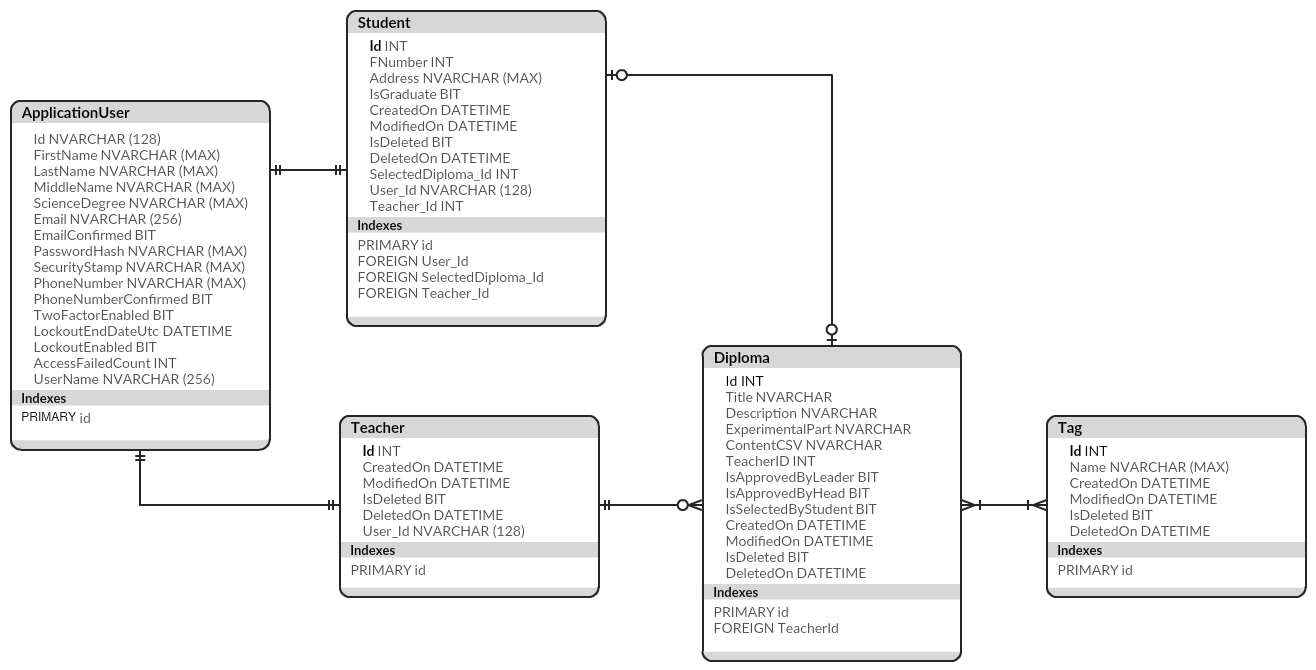
* Папка Data:
  + DDS.Data – проект, в който се намира логиката за създаване на базата данни
  + DDS.Common – проект, който съдържа обща логика за класовете (моделите)
  + DDS.Models – проект, в който са описани самите модели
* Папка Services:
  + DDS.Services.Data – тук са разположени всички услуги, както и техните интерфейси
  + DDS.Services.Web – проект, в който са разположени услуги касаещи приложението като кеширане (chache)
* 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.

## Създаване на базата данни

За да може едно приложение да е одобно 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 одобен 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 следната команда **Enable-Migrations** върху проекта DDS.Data. Тази команда създава папка с име Migrations в проекта DDS.Data, в която се генерират **Configuration.cs** файл, които 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 създадът моделите, който да бъдат използвани.



Фигура 4‑1 Схема на базата данни (ERD)

На схемата са изобразени обектите (entities), които ще бъдат използвани в настоящата система.

ApplicationUser – Този модел е наследник от генерирания за нас код. В него са добавени полета за име, през име, фамиля, 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 който 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 записа).

Student – е моделът, койти съдържащ информацията за студента необходима за заданиято, както и информация за избраната диплома и служебно булево поле за това дали студента е завършил успешно. Наследява BaseModel<int>.

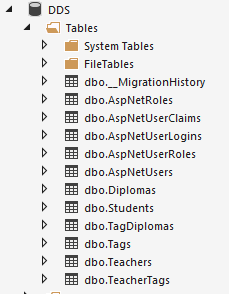
Teacher – модел, в който са поместени колекции с абсулвенти, дипломи и категории касаещи конкретния ръководител. Класът има конструктор, чиято цел е да създаде в паметта гореспоментатите колекции.

Diploma – модел, който има полета за работа с дипломни задания. Съдържа полета за заглавие, описание, описание на експерименталната част, глави, категории, ръководител и служебни полета за избор от студент, одобрение от ръководител, канцелария и технически сътридник.

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 промените, които са извършени 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 взима 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 таблицата, 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 таблиза 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>

## 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 усигурим 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 Т представлява елемент от типа BaseModel<int>.

## 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)

Основно предимство, което ни предоставя това разделение, е голямото разделение на кода. Тази архитектура е подходяща, както за малъки, 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.

## 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 архитектрура с този принцип се обръща контрола в сравнение с традиционното процедрурно програмиране: в традиционното програмиране кодът, които изразява целта на програмата извиква преизползваеми библиотеки за да извърши някаква основна операция, но чрез 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 разразне 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 се намира във вайл 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 съотвената 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();

## 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, който помещава 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.

## Създаване на изгледи

Изгледът представлява скрип, чрез който приложението разбира какво да се визуализира на екрана. Намират се в папка Views на основния проек 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 именован 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 се повтарят на всяка страница като навигация и footer. 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. Състой 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 не правилно въведена информация на екрана се изписват съответните съобщения за грешка.
* **Home/Index** – Стартов изглед за приложението. Представлява статична страница с информация за процеса по изработка на дипломи 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 дипломно 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** – Представлява 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 неоцветени. 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 ги в кошчето. Когато се натисне бутонът „Кошче“, намиращ се над списъка с дипломни работи, ръководителят е препратен към списък с всички изтрити 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** – Представлява страница с маркираните за изтрити 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 нова 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 смисъка 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.

## Automapper – настройки и приложение

В проекта DDS.Web.Infrastructure е разположена библиотеката Automapper

Конфигурация

В папка Mapping се намира интерфейс IMapFrom<T> с помоща, на който се определя кои класове могат да бъдат мапнати? 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 Т 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 мапване на полетата с еднакви имена на желаните класове. Прилагайки IHaveCustomMapping към целевия клас (например изглед-модел) и имплементирайки функцията CreateMappings, се позволява мапване 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 рефлекция, извича тези, които наследяват IMapFrom<T> и IHaveCustomMapping и им закача мапинг?. Асемблито, което се подава е на текущото приложение. Подавайки го, automapper разбира от къде да вземе всичките публични типове (класове), които наследяват кореспоменатите интерфейси. Този процес се намира в Global.asax:

var autoMapperConfig = new AutoMapperConfig();

autoMapperConfig.Execute(Assembly.GetExecutingAssembly());

Поможния клас IQueryableExtensions е разширение на възможностите на типа IQueryable, като единствения метод, които 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 пък, когато използваме Automapper и инсталирането му в Web проекта.

## Генериране на Docx файлове

Генерерирането на \*.docx файлове се усъществява с помощта на библиотеката DocX. Процесът изисква шаблонен файл, в който се намират текстови константи. Тези контейнери биват заместени с извлечените от базата информация.

# 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.

(снимка как изглежда страницата)

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. 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, която съдържа линк за смяна на парола, линк за детайли за избраната от него диплома (ако има такава), полета за въвеждане на факултетен номер и адрес (ако е студент) и полета за телефон и научна степен за всички регистрирани потребители.

(снимка на Manage)

## 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 представлява таблица, която има колоните: Заглавие, Описание, Ръководител, Категории и дата на добавяне. При натискане на ред от таблицата, студентът бива препратен към детайли за избраната диплома. Студентът 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, които използват 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.

(Снимка на избора със съобщение за данни)

Когато диплома от таблицата е избрана и потребителят е препратен към по-подробните детайли за дипломата, 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 ще бъде препратен обратно към списъка с дипломни задания и на страницата ще бъде изведено съобщение за успех. 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.

(снимка на избраната диплома – детайли)

(Снимка на списък ръководители)

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

## Ръководство за потребител тип ръководител

Когато потребител от тип ръководител се впише 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, който маркира дипломата като не избрана от студента.

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 тя бива автоматично създадена. За да създаде дипломното задание, ръководителят трябва да натисне бутона „Запази“.

(снимка с изглед на създаване на дипломно задание)

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.

Изтритите 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 изтрити дипломни задания, която 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.

## 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. 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. 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. При невалидно въведени полета, излиза съответното съобщение за грешка.

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

# Заключение

В настоящата дипломна работа беше разгледана система за съставяне и одобрение на дипломни задания, която има за цел автоматизация на процеса. Решението е изградено, използвайки всички съображения заложени 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 модалността тази функционалност може да бъде лесно добавена в случай, че приложението трябва да бъде разширено.

Заключението резюмира основните характеристики и особености на представеното инженерно решение, неговите предимства и недостатъци, резултатите от експериментите, тяхната инженерна трактовка и предложения за по-нататъшна работа върху решаването на зададения проблем.

# Списък на използвани литературни източници

# Приложения

Приложенията съдържа изходен текст на програми, формати на структури от данни, екранни форми, получени резултати, електрически схеми, графични оригинали на печатни пратки, справочни данни и др.