**Русенски университет “Ангел Кънчев”**

Факултет: Природни науки и образование

Катедра: Информатика и информационни технологии

**ДОПУСКАМ ДО ЗАЩИТА**

Ръководител на специали-

зиращата катедра:……..………

(доц.д-р Румен Русев)

**ОБЯСНИТЕЛНА ЗАПИСКА**

**НА ДИПЛОМЕН ПРОЕКТ**

**Тема**: **Инструмент за прогнозиране на крайна оценка по дисциплина.**

Дипломант: ………………… **ОДОБРЯВАМ**

(Божидар Атанасов) Ръководител:…………………… (Камелия Шойлекова)

Русе, 2025 г.

**Съдържание**

[**ДОПУСКАМ ДО ЗАЩИТА** 1](#_Toc193136289)

[Дипломант: ………………… **ОДОБРЯВАМ** 1](#_Toc193136290)

[1. Увод 5](#_Toc193136291)

[2. Обзор на съществуващи решения. Изводи. Цели и задачи 6](#_Toc193136292)

[**2.1. Разглеждане на съществуващи решения** 6](#_Toc193136293)

[**2.1.1. Cope** 6](#_Toc193136294)

[**2.1.2 Админ Плюс** 7](#_Toc193136295)

[**2.1.3 Administrator’s Plus** 7](#_Toc193136296)

[*Фигура 2.3 Administrator’s Plus* 8](#_Toc193136297)

[**2.2 Анализ на съществуващите системи** 8](#_Toc193136298)

[**2.3 Изводи. Цели и задачи** 8](#_Toc193136299)

[**2.3.1 Изводи** 8](#_Toc193136300)

[**2.3.2 Цели и задачи** 9](#_Toc193136301)

[3. Архитектурен и логически модел на системата 10](#_Toc193136302)

[**3.1 Архитектурен модел на системата** 10](#_Toc193136303)

[**3.1.1 Използвани технологии** 10](#_Toc193136304)

[**3.1.1.3 Visual Studio** 10](#_Toc193136305)

[**3.1.1.5 Entity Framework Core** 11](#_Toc193136306)

[**3.2 Архитектура на системата** 12](#_Toc193136307)

[**3.3 Логически модел на системата** 13](#_Toc193136308)

[**3.3.1 Диаграми на случаите на употреба** 13](#_Toc193136309)

[**3.3.2 Диаграми на дейностите – Activity Diagrams** 14](#_Toc193136310)

[4. Описание на базата данни 17](#_Toc193136311)

[**4.1 Релационен модел на базата данни** 17](#_Toc193136312)

[**4.2. Описание на таблиците и връзките между тях** 19](#_Toc193136313)

[**4.2.1 Таблица Roles** 19](#_Toc193136314)

[**4.2.2 Таблица UserRoles** 19](#_Toc193136315)

[**4.2.3 Таблица Users** 19](#_Toc193136316)

[**4.2.4 Таблица Grades** 19](#_Toc193136317)

[**4.2.5 Таблица SemesterSubject** 20](#_Toc193136318)

[**4.2.6 Таблица SemesterUsers** 20](#_Toc193136319)

[**4.2.7 Таблица Semesters** 20](#_Toc193136320)

[**4.2.8 Таблица Subjects** 21](#_Toc193136321)

[**4.2.9 Таблица Formulas** 21](#_Toc193136322)

[**4.2.10 Таблица EFMigrationsHistory** 21](#_Toc193136323)

[5. Програмна реализация 21](#_Toc193136324)

[**5.1 Инжектиране на зависимости в Razor Pages (Dependency Injection)** 24](#_Toc193136325)

[**5.2 Основни страници** 24](#_Toc193136326)

[**5.2.1 Начална страница (Index)** 24](#_Toc193136327)

[**5.2.2 Страница Register** 26](#_Toc193136328)

[**5.2.3 Страница Logout** 27](#_Toc193136329)

[**5.2.4 Страница Admin – BrowseGrades** 28](#_Toc193136330)

[**5.2.5 Страница Admin - BrowseSemesters** 30](#_Toc193136331)

[**5.2.6 Страница Admin – BrowseStudents** 31](#_Toc193136332)

[**5.2.7 Страница Admin - BrowseSubjects** 31](#_Toc193136333)

[**5.7.8 Страница Admin - BrowseTeachers** 32](#_Toc193136334)

[**5.2.9 Страница Admin - Index** 32](#_Toc193136335)

[**5.2.10 Страница Student - SemesterOverview** 33](#_Toc193136336)

[**5.2.11 Страница Student - Index** 35](#_Toc193136337)

[**5.2.12 Страница Teacher – BrowseSubjects** 36](#_Toc193136338)

[**5.2.13 Страница Teacher – CourseDetails** 37](#_Toc193136339)

[**5.2.14 Страница Teacher – Index** 37](#_Toc193136340)

[**5.2.15 Страница Shared** 38](#_Toc193136341)

[6. Ръководство за работа със системата 38](#_Toc193136342)

[**6.1 Инсталиране на системата.** 38](#_Toc193136343)

[**6.1.1 Нужни ресурси за инсталирането на системата** 38](#_Toc193136344)

[**6.1.2 Стартиране на проекта** 39](#_Toc193136345)

[**6.2 Регистрация на нови потребители** 40](#_Toc193136346)

[**6.3 Употреба на приложението като студент** 41](#_Toc193136347)

[**6.4 Употреба на приложениетж като преподавател** 42](#_Toc193136348)

[**6.5 Употреба на приложението като администратор** 46](#_Toc193136349)

[7. Основни резултати, изводи и препоръки 51](#_Toc193136350)

[8. Използвана литература 51](#_Toc193136351)

[Cope, „tgw-calculator,“ 17 3 2025. [Онлайн]. Available: https://copeseeds.co.uk/services/tgw-calculator/. 51](#_Toc193136352)

[9. Таблица с фигури 51](#_Toc193136353)

[Фигура 1 Copeseeds 6 51](#_Toc193136354)

[[2] AdminPlus, URL: https://adminplus.bg/, посетен на 07.02.2024г. 51](#_Toc193136355)

[[3] Administrator's Plus, URL: https://www.rediker.com/, посетен на 07.02.2024г. 51](#_Toc193136356)

[[4] ASP.NET Core, URL https://dotnet.microsoft.com/en-us/apps/aspnet, посетен на 10.02.2024г. 51](#_Toc193136357)

[[5] Microsoft SQL Server, URL: https://www.microsoft.com/en-us/sql-server/sql-server-downloads, посетен на 10.06.2024г. 51](#_Toc193136358)

[[6] Entity Framework Documentation Hub, URL: https://learn.microsoft.com/en-us/ef, посетен на 10.06.2024г. 51](#_Toc193136359)

[[7] Какво представлява унифицирания език за моделиране (UML), URL: https://www.microsoft.com/bg-bg/microsoft-365/visio/uml, посетен на 20.07.2024г. 51](#_Toc193136360)

# **1. Увод**

Фермерство, дори в тази модерна ера на квантови компютри, изкуствен интелект и телефони по-мощни от настолни компютри един човек не може да живее без храна. В миналото всеки е разполагал с площ за фермерство или е плащал на друг за да може. С това е изхранвал както себе си така и семейството си. В днешно време това също е възможно, но повечето хора извършват тази дейност не за изхрани, а като хоби през уикенда.

В днешно време фермери с хиляди декари площ, притежавана от тях или плащайки рента на нея, извършват тази дейност. От това зависи тяхното преживяване, инвестиции и дори в някой случай преживяването на други хора. Те не могат да правят грешки и трябват да намалят случайните фактори до миниум. За това повечето се консултират с агрономи и използват калкулатори.

Този проект не за използване на изкуствен интелект да замени агрономи. Идеята и темата за калкулиране на параметри предварително е много стара. Едни основни неща който един фермер трябва лесно да може да калкулира са: сеитбената норма (колко растения трябва да посади от семки за да поникна толкова колкото очаква), сеитбена норма за смески, с какви препарати, с какви дози и как да пръска при борба с неприятели. Също да е запознат с одобрени за използване на Българския пазар препрати от ББХ (Начало - Българска агенция по безопасност на храните) (<https://bfsa.egov.bg/wps/portal/bfsa-web/home>).

Системата ще подържа 3 вида калкулатора:

* Калкулатор за сеитбената норма
* Калкулатор за сеитбената норма за смески
* Калкулатор за препарати с борба с неприятели

Също ще подържа:

* Уикипедия с информация за предоставените препрати в системата
* Система за профили за запазване на резултати

# **2. Обзор на съществуващи решения. Изводи. Цели и задачи**

## **2.1. Разглеждане на съществуващи решения**

<https://www.agrocalculator.bg/>

<https://www.agrii.co.uk/calculator-tools/seed-rate-calculator/>

### **2.1.1. Cope**

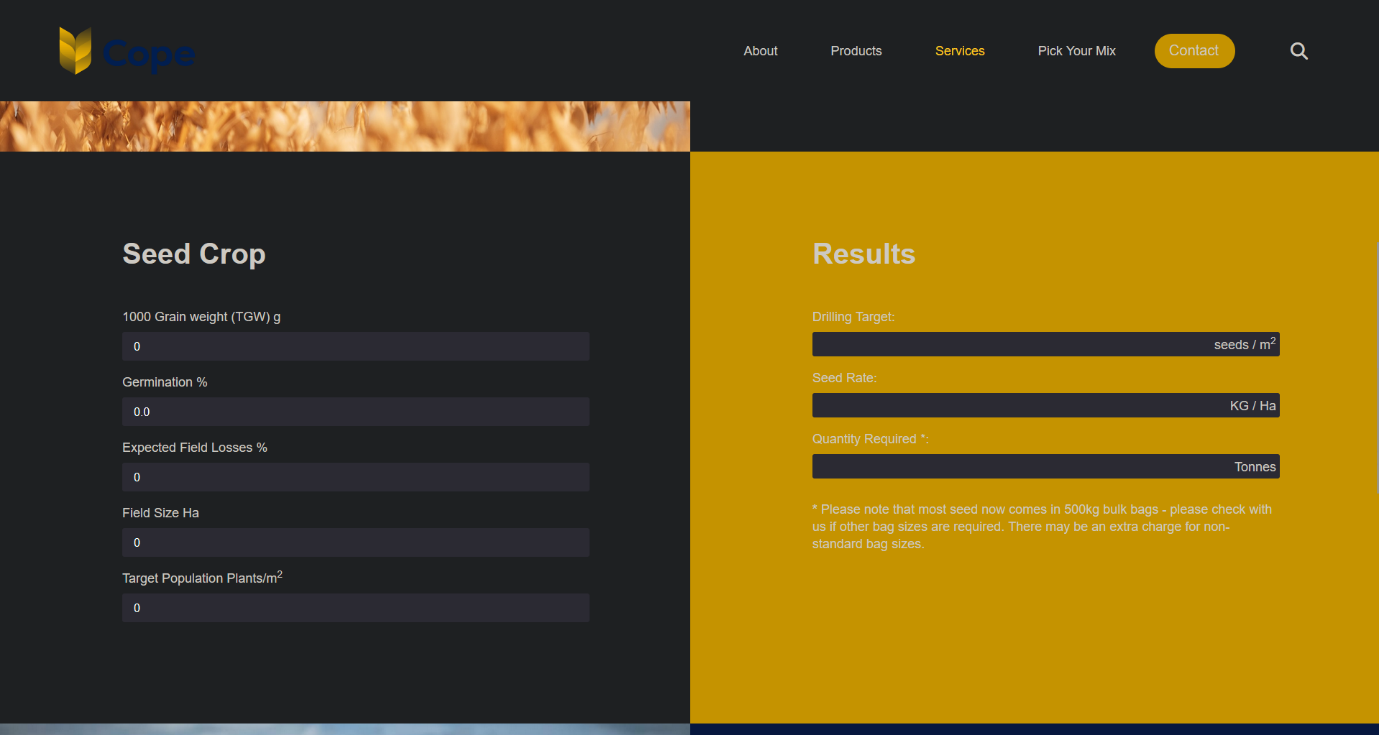
Страница направена от английска фирма занимаваща се с продажбата на семена и торова. Не е главния продукт. Продукта може да се види на Фигура 1 [1].

**Предимства:**

* Малко възможности води до лесна употреба.
* На английски и е достъпен до друг пазар извън България.

**Недостатъци**:

* Липсват данни от резултата.
* Използва хектари, единица главно използвана в Англия.



Фигура 1 Copeseeds

### **2.1.2 Реколта digital**

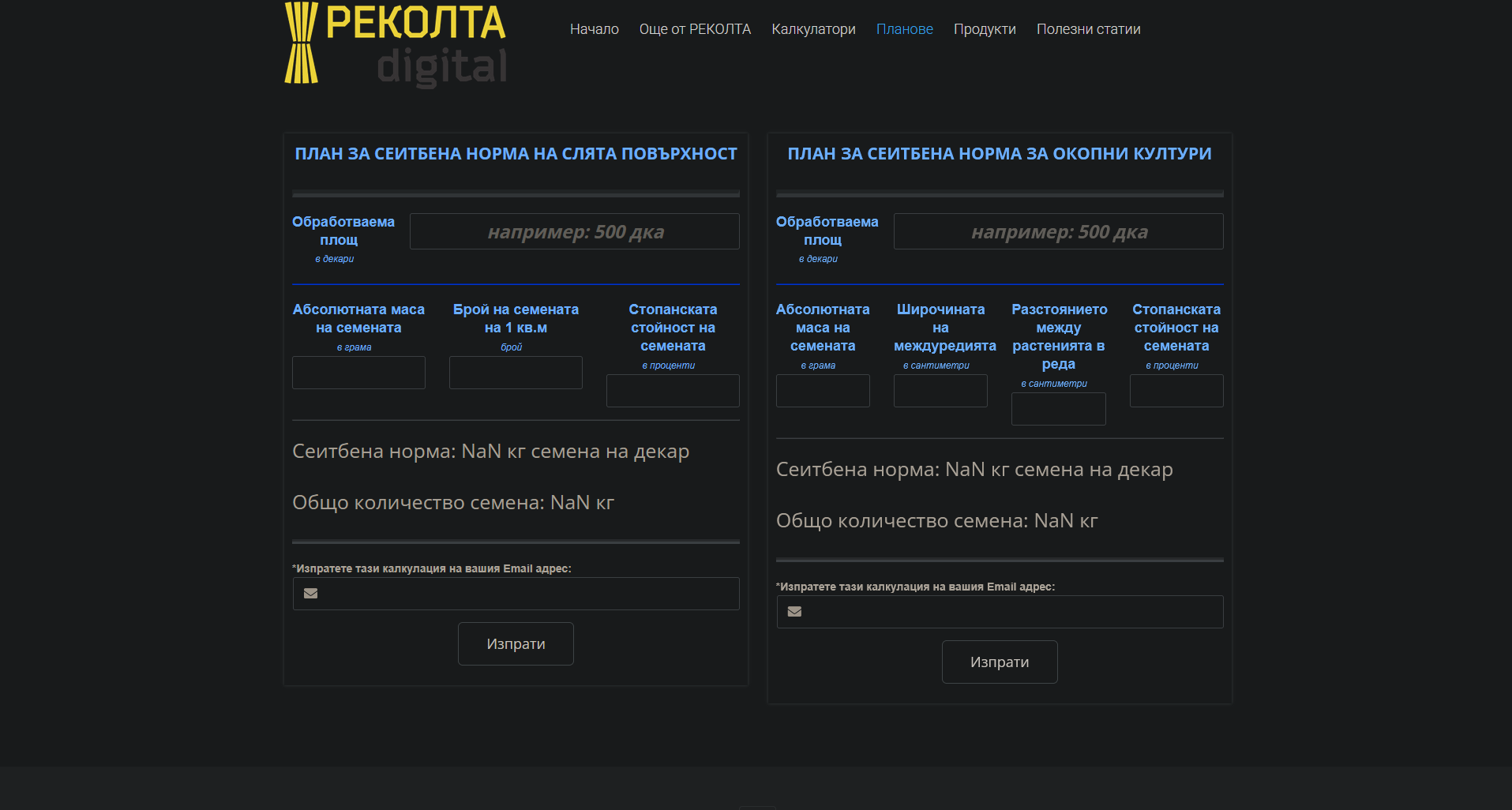
Подобно на Cope те същото предлагат система за смятане на сеитбена норма. Те също предлагат подробни калкулатор за основни култури като пшеница и рапица засягащи всякакви черти от отглеждането на тези култури. Продукта може да се види на Фигура 2 [2].

Предимства:

* На български е като предлага сметки с български единици.
* Има още информация и други калкулатори.
* Може да бъде запазен резултата чрез пращане по емайл.

Недостатъци:

* Калкулатора калкулира малко информация за сеитбени норми.
* Няма информация за смески или за пръскане



Фигура 2 Реколта digital

### **2.1.3 Agrii**

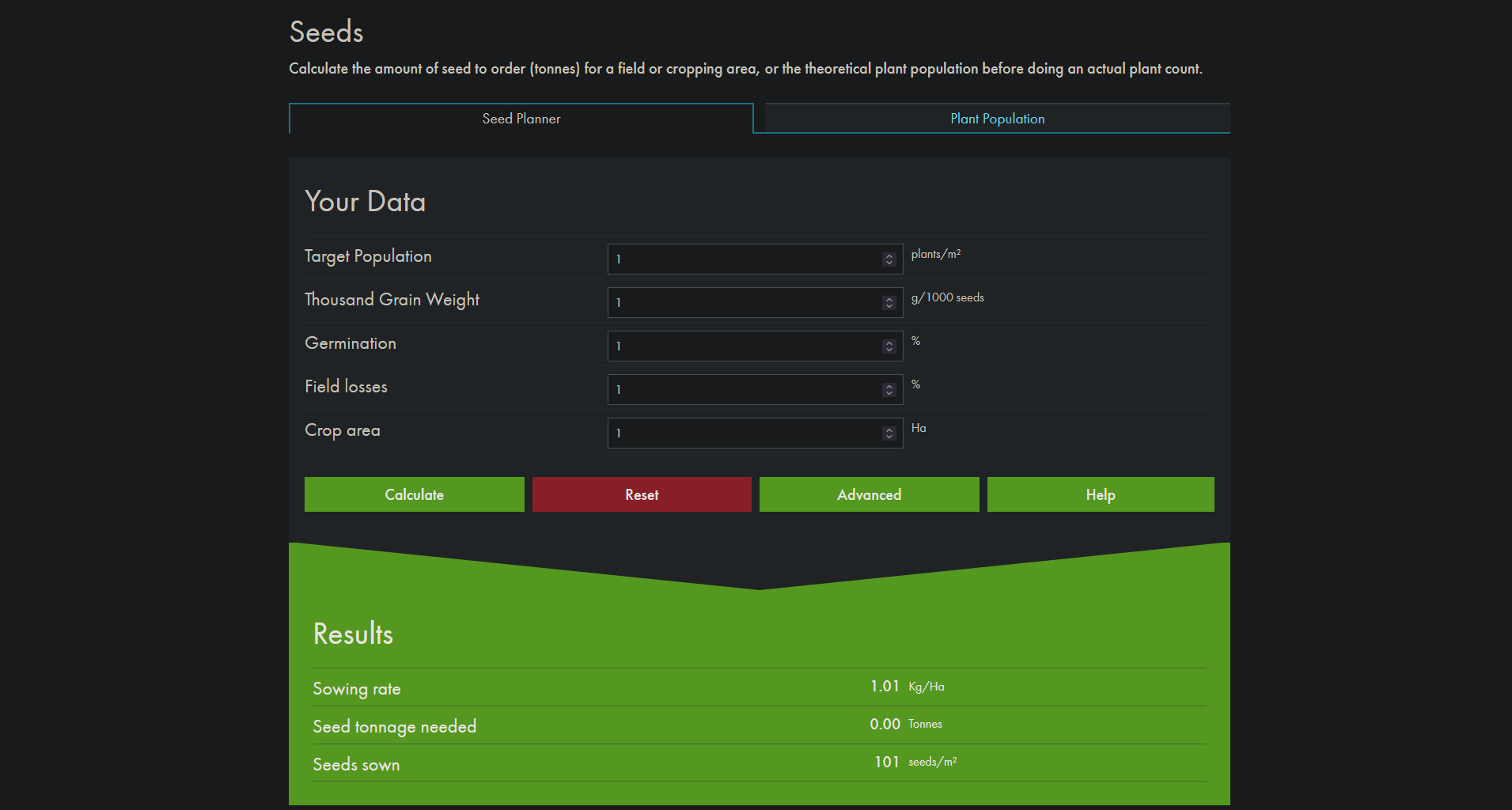
Agrii, подобни на Cope, е англиска фирма занимаваща се с консултация и пордажаба на семена и торове. Не е главният им продукт [3].

**Предимства:**

* Добре направен с дизайн и функционалност
* Има помощно меню за обясняване на всяко поле какво прави
* Има възможност за запазване на сметка в емайл или да бъде изтеглена като PDF

**Недостатъци:**

* На английски е
* Работи в хектари
* Има малки досадни бъгове в UI/UX



Фигура 3 Agrii

### **2.1.4 agrocalculator.bg**

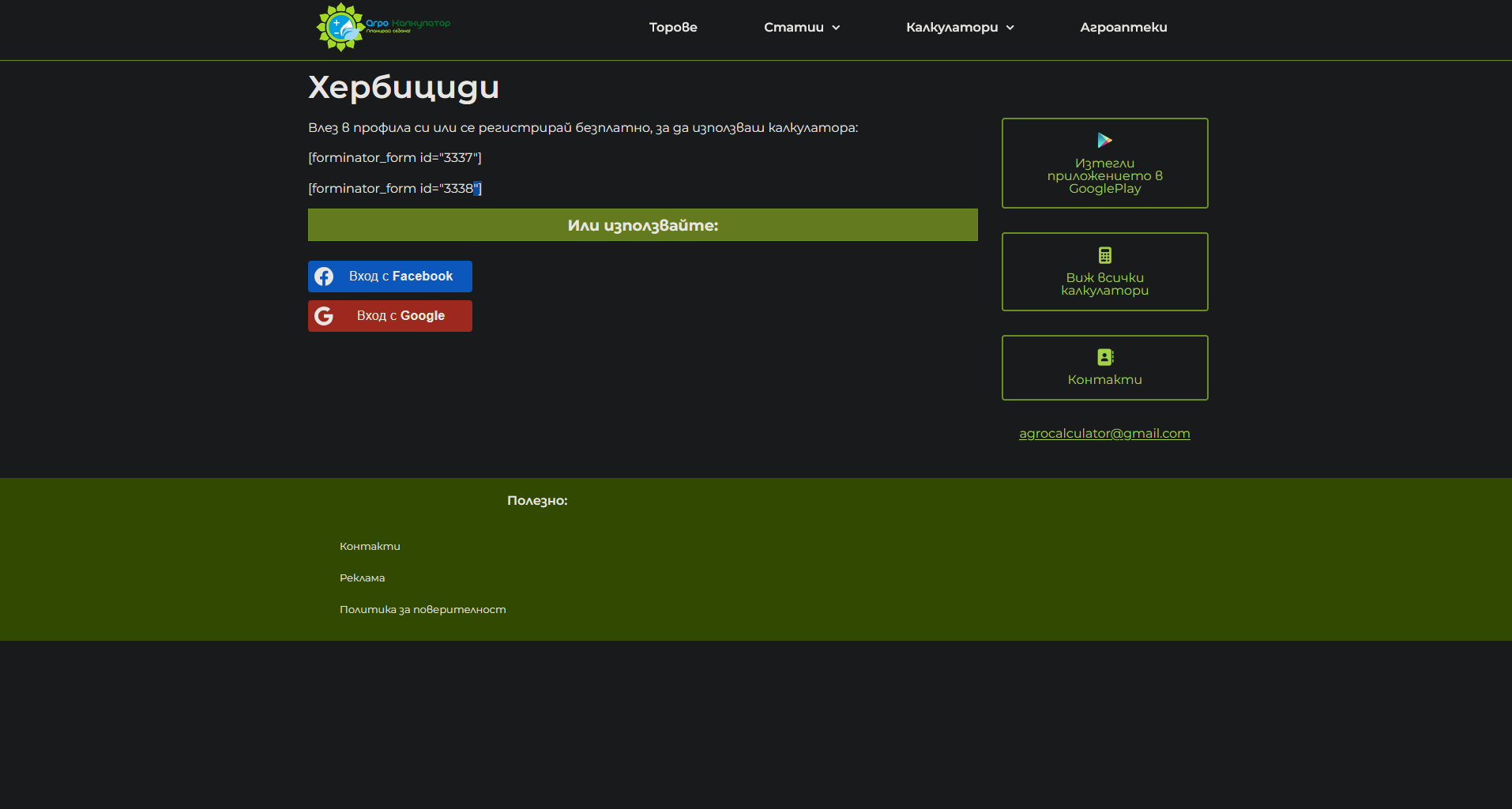
Agrocalculator.bg е български сайт, съвкупност от няколко калкулатора със земеделски цели. За съжаление не работи [4].

**Предимства:**

* Имат мобилно приложение

**Недостатъци:**

* Сайта и калкулаторите не работят



Фигура 4 agrocalculator.bg

### **2.1.5 preparati.info**

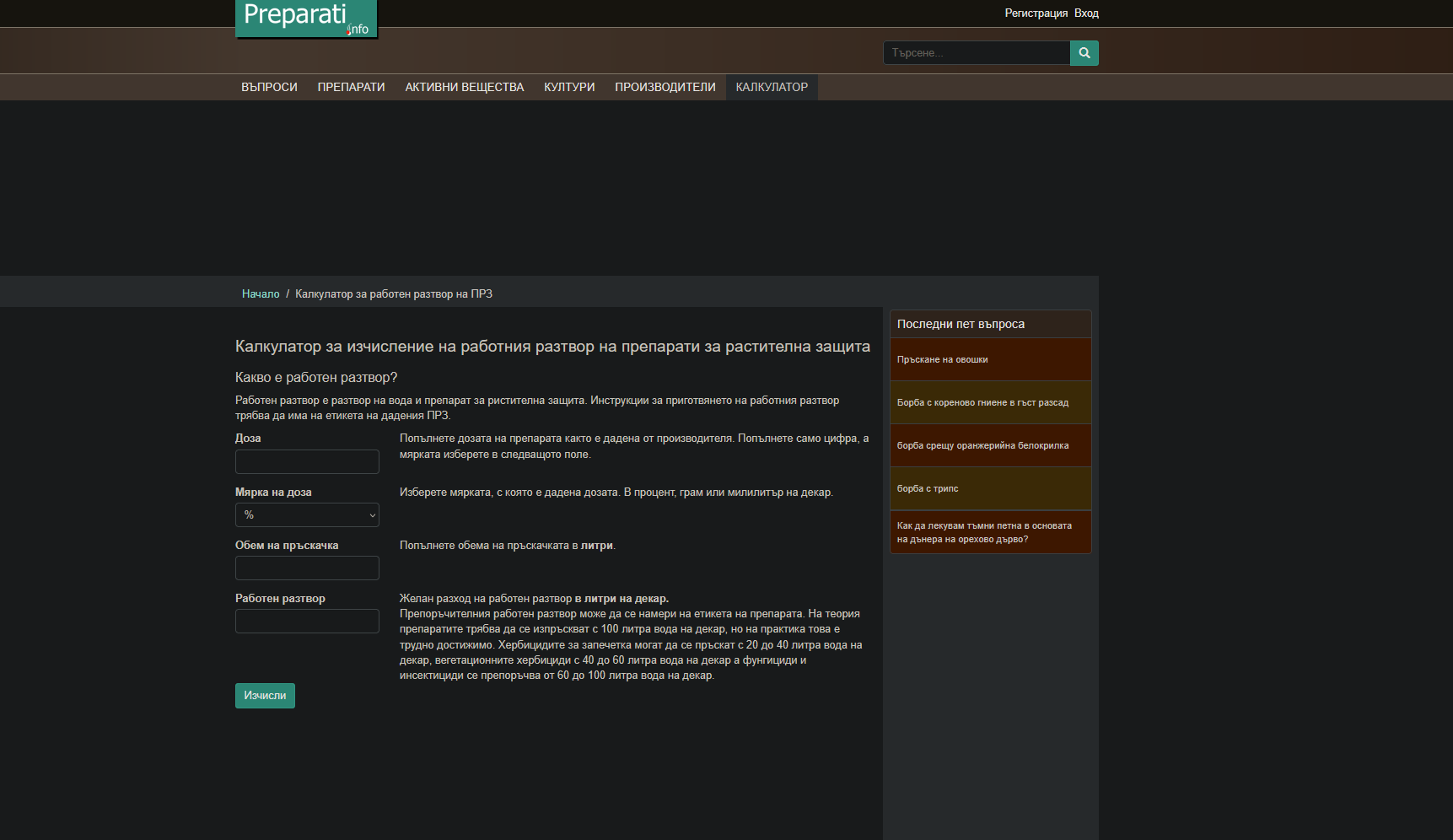
preparati.info български сайт, съдържащ информация за препарати за борба с неприятели в България одобрени от ББХ [5].

**Предимства:**

* Има много богата уикипедия с текущо използвани и забранени препарати
* Има калкулатор за работен разтвор, една лесна но всеки му нужна сметка
* В калкулатора обяснява всяко поле за какво е
* Добър UI/UX

**Недостатъци:**

* Няма начин за запазване на сметката



Фигура 5 preparati.info

## **2.2 Анализ на съществуващите системи**

Сравнение на функционалните възможности на различните продукти в таблица 1.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Характеристики | Cope | Реколта digital | Agrii | agrocalculator.bg | preparati.info |
| Уеб Базирана система | ДА | ДА | ДА | ДА | ДА |
| Предлагат повече от един калкулатор | НЕ | ДА | НЕ | ДА | НЕ |
| Предлага пресмятане на работен разтвор | НЕ | НЕ | НЕ | НЕ | ДА |
| Има достъп до потвърдени научно данни | НЕ | НЕ | НЕ | НЕ | ДА |

Таблица 1 Сравнение на функционалности

## **2.3 Изводи. Цели и задачи**

### **2.3.1 Изводи**

От направяният преглед на съществуващите решения се виждат някои недостатъци:

* Не всички продукти имат повече от един калкулатор
* Не всички продукти предлагат смятане на работен разтвор
* Не всички продукти имат достъп до научно доказани данни

### **2.3.2 Цели и задачи**

Целта на дипломният проект е да се разработи уеб базирана система за пресмятане на най-важните неща за фермер и любител фермер с достъп до научно доказани данни

За постигането на тази цел са формулирани следните задачи:

* Създаване на уеб базирано приложение, което да притежава лесен и достъпен дизайн за употреба;
* Регистрация на потребители, управление на правата им на достъп;
* Запазване на направени вече калкулации за по-късен анализ и употреба
* Показване на статистика за текущо направени калкулации и поредица от тях

# **3. Архитектурен и логически модел на системата**

## **3.1 Архитектурен модел на системата**

### **3.1.1 Използвани технологии**

За създаването на дипломният проект са използвани следните технологии: Visual Studio Code, git, Docker, yarn, React, NextJS, Prisma, PostgreSQL, Typescript, react-redux, zod, ShadCN, Tailwind CSS, Postcss, Lucide icons, Github, Github Actions, Netcup VPS, Jest.

#### **3.1.1.1 Visual Studio Code**

Когато става въпрос за разработка, едно от най-важните неща са инструментите, които се използват. Visual Studio Code е гъвкава среда за разработка. То е текстов редактор с вградени инструменти и клавишни комбинации за лесна употреба от един програмист. В сравнение с пакети като Visual Studio и Eclipse даже има малко функционалност, но разполага с бурен и жив пазар за разширения (extensions) който го правят изключителна мощна среда позволяваща на един програмист да следи промените от написването им до пускането им в производство (production) [6].

#### **3.1.1.2 git**

Проблем който всеки програмист среща е как да пази версии на кода. В стари дни (и днес) файловете са на един споделен диск в някакво облачно пространство или се пренасят от компютър на компютър с още едно \_final добавено отзад. Linus Torvalds предоставя решение на този проблем през 2005 с FOSS (free and open source – безплатен с отворен код) инструмент наречен git. Този инструмент позволява създава на лесно разбираема нишка от промени чрез който програмист може да работи спокоен че винаги има старата версия на една команда разстояние [7].

#### **3.1.1.3 Docker**

Docker е инструмент и платформа за контейнеризация, която позволява разработчиците да създават, разпространяват и стартират приложения в изолирани контейнери. Контейнерите осигуряват лека, преносима и последователна среда, която включва всички зависимости на приложението, което улеснява разработката, тестовете и внедряването в различни среди. Docker използва образи (images) за дефиниране на приложения и техните зависимости, а контейнерите се изпълняват от тези образи. Това елиминира проблеми, свързани с несъвместимости между операционни системи и библиотеки, подобрявайки мащабируемостта и ефективността на софтуерните решения [8].

## **3.1.1.4 React**

React е популярна JavaScript библиотека, разработена от Facebook през 2013 г., която се използва за изграждане на динамични и интерактивни потребителски интерфейси. Тя следва компонентно-базиран подход, което улеснява повторното използване на код и поддръжката на големи приложения. Основно предимство на React е използването на виртуален DOM, който оптимизира обновяването на потребителския интерфейс, подобрявайки производителността. Освен това React поддържа еднопосочен поток на данни, което прави управлението на състоянието по-предвидимо. Благодарение на своята гъвкавост и мащабируемост, React е предпочитан избор за разработка на уеб и мобилни приложения.

## **3.1.1.5 NextJS**

Next.js е мощен JavaScript фреймуърк, базиран на React, разработен от Vercel, който улеснява създаването на бързи и оптимизирани уеб приложения. Той поддържа сървърно рендериране (SSR) и статична генерация (SSG), което подобрява производителността и SEO оптимизацията на сайтовете. Освен това, Next.js предлага автоматична маршрутизация, вградена поддръжка за API маршрути, лесна интеграция с CSS, Sass и Tailwind CSS, както и функции като динамични маршрути и инкрементално статично обновяване (ISR). Тези характеристики го правят отличен избор за изграждане на мащабируеми, интерактивни и кросплатформени уеб приложения [6].

## **3.1.1.6 Prisma**

Prisma е модерен ORM (Object-Relational Mapping) инструмент за бази данни, който улеснява работата с SQL бази като PostgreSQL, MySQL, SQLite, SQL Server и CockroachDB. Той предоставя удобен типизиран API за взаимодействие с базата данни, като автоматично генерира TypeScript или JavaScript код на базата на дефинирана схема. Prisma включва мощни инструменти като миграции на базата (Prisma Migrate), визуален интерфейс за заявки (Prisma Studio) и поддръжка за GraphQL и REST API. Благодарение на строгата типизация и интеграцията с TypeScript, Prisma подобрява сигурността и предвидимостта при работа с бази данни [9].

## **3.1.1.7 PostgreSQL**

PostgreSQL е гъвкава и мощна релационна база данни с отворен код, която работи на множество платформи, включително Linux, Windows и macOS. Тя е изключително стабилна, поддържа ACID транзакции, мащабиране и разширяемост чрез разширения. PostgreSQL е предпочитан избор за уеб, корпоративни и облачни приложения поради своята надеждност и висока производителност [10].

#### **3.1.1.8 Typescript**

TypeScript е обектно-ориентиран език за програмиране, разработен от Microsoft и пуснат през 2012 година като надграждане на JavaScript. Основната му цел е да осигури по-добра структура и мащабируемост при разработката на уеб приложения, като добавя статична типизация и модерни функционалности. TypeScript е строго типизиран, което означава, че всяка променлива има конкретен тип, известен по време на компилацията. Това позволява на програмистите да откриват грешки още преди изпълнението (run time) на кода и улеснява поддръжката на големи проекти. За разлика от него, JavaScript е динамично типизиран език, което позволява по-гъвкаво, но и по-рисковано управление на променливите. Например, една променлива в JavaScript може да съдържа число (int), а по-късно да бъде заменена със низ (string), което може да доведе до грешки по време на изпълнение. TypeScript елиминира този проблем, като осигурява по-голяма предвидимост и стабилност при разработката на сложни приложения [7].

#### **3.1.1.4 Microsoft SQL Server**

Microsoft SQL Server служи за управление на релационни бази данни и е разработен от Microsoft през 1989 година. През годините преминава през множество различни подобрения. Основната му функция е съхранението и извличането на данни, както се изисква от други софтуерни приложения. Продуктът се предлага в различни версии, разпределени според типа и нуждите на потребителя. В случая за изготвянето на тази дипломна работа ще се използва Express edition. Той е предназначен за тестови проекти (тип хоби) и предлага всички необходими инструменти [5].

## **3.1.1.5 Entity Framework Core**

Entity Framework (EF) Core е обектно-релационен модел (Object-Relational Model, накратко ORM) за данни. Позволява на .NET разработчиците да работят с база данни, използвайки .NET обекти. Това означава, че разработчикът не е длъжен да пише SQL заявки, тъй като те се генерират чрез C# обекти. EF Core използва класове в C# за представяне на таблици в базата данни, а връзките между таблиците се моделират чрез релации между обектите. Базата данни се актуализира чрез миграции, които проследяват промените в структурата на обектите [6].

Друго предимство на EF Core е поддръжката на Eager loading. Това е метод за оптимизация на заявките, при който информацията, която може да е необходима в бъдеще, се зарежда предварително чрез връзките между обектите. Например, с една заявка могат да бъдат заредени 4 различни обекта, свързани помежду си. При lazy loading, когато трябва да се използва свързан обект, се изпраща отделна заявка за неговото зареждане. Това води до множество заявки към базата, което може да забави изпълнението и да доведе до потенциално кеширане на неактуална информация.

#### **3.1.1.6 Bootstrap**

Bootstrap е популярен фронт-енд фреймуърк (frontend framework), който подпомага изобразяването на информацията по достъпен начин. Една от най-полезните му функционалности е автоматичното преоразмеряване към различните размери на екрана. Това предоставя възможността за единно изживяване при употребата на компютър и на мобилно устройство. Освен това, Bootstrap позволява лесно модифициране на елементите, за да се приспособят към специфичните нужди на проекта.

#### **3.1.1.7 CSS**

CSS (Cascading Style Sheets) се използва за контролиране на външния вид и форматирането на уеб страниците. Позволява лесното разделение на уеб структурата от облика на елементите. Употребява се заедно с HTML.

#### **3.1.1.8 CSHTML**

CSHTML е файлов формат, разработен от Microsoft. Той позволява на ASP.NET приложенията да вмъкват C# код в HTML на уеб страницата. Това позволява динамичното изменение на страниците, преди да бъдат заредени от крайния потребител. Пример за това е употребата на HTML таг <h1>, който обозначава заглавен текст, като в него се вмъкне символът @. Това позволява на компилатора да обработи следващото логическо съчетание: <h1>Добре дошли, @Model.UserName!</h1>. На крайния потребител ще се покаже „Добре дошли, Иван!“, ако, разбира се, свойството UserName е публично достъпно и инициализирано.

#### **3.1.1.9 JavaScript**

JavaScript е лесен за научаване и използване език в програмирането, използван е от уеб браузърите. Той е обектно-ориентиран и поддържа функционален стил на програмиране. Често се използва за улесняване на визуални промени на страницата и за извършване на клиентска валидация на данните, преди те да бъдат изпратени към сървърната логика на приложението.

#### **3.1.1.10 AJAX**

Ajax (Asynchronous JavaScript and XML) е технология, която позволява на уеб страниците да комуникират с уеб сървъра асинхронно. След първоначалното зареждане на страницата е възможно, при нужда, да се зареждат допълнителни данни. Това намалява времето за зареждане и позволява повече функционалност да бъде включена в една страница.

#### **3.1.1.11 GIT**

Тип разпределителна система на версиите на едно приложение. Спомага при проследяването на развитието на приложение. Решава множество проблеми свързани с конфликтите при подобрения на код, в среда където множества хора променят сходни файлове. Копието на един проект се нарича repository. То е напълно самостоятелно и позволява самостоятелност при работата над подобрението на проекта. Позволява се подразделение на repository, чрез branches (клони).

## **3.2 Архитектура на системата**

Първата стъпка при asp.net е обработката на HTTP Request на изпратената заявка. Тоест ако получим заявка към сървъра <https://localhost/Admin> тя ще започне да се обработва чрез middleware pipeline (това се намира в Program.cs файла):

1. Първото действие на middleware pipeline e да подготви обекта, който ще стартира приложението. Към него се прилагат множество services (услуги):

* Add Razor Pages – тук директно се споменава, че се използва този тип структура от asp.net.
* Add Session – Добавя се поддържането на сесия от към страна на сървъра. Това позволява динамично да се запазва всякаква допълнителна информация. И да може да се търси по-лесен начин от различните елементи. Пример за това е запазването на потребителя, който се е логнал.
* Add Db Context – Чрез този елемент се добавя връзката на приложението с базата данни. Може да се достъпи от всеки един контролер след като се инжектира в конструкторът му. Допълнително в случаят се добавя и Use Sql Server което обработва свързването на проекта с базата данни на Microsoft Sql Server.
* Накрая се добавя Ajax поддръжката към Razor Pages структурата. Това става чрез Add Newtonsoft Json.

1. След това се добавя пренасочване на Http заявки да станат от тип Https. Чрез Use Https Redirection.
2. Добавена е статичната обработка на търсене на файлове Use Static Files.
3. Добавя се автоматичната обработка на пътя към контролерите Use Routing. Това позволява крайният адрес на заявката да бъде обработен и след това да се насочи вътрешно в приложението към правилният контролер. EndPoint от HttpContext.
4. Вече се активира употребата на сесията чрез Use Session.
5. От тук нататък може да се добавят различни Middlewares. В случая е добавен мой собствен Login Middleware. Като той определя дали дадена заявка да бъде обработена или не и също обработва вписването на потребителите в системата.
6. След това се извиква Map Razor Pages, което превръща страниците според това как са добавени в архитектурата на файловете към крайни точки.
7. Накрая се извиква Run и приложението се стартира.

Пояснение на LoginMiddleware: При стартирането му първо се запазва оставащата опашка от делегатите (методите, които предстои да се изпълнят) в middleware pipeline. След това се анализира крайната точка и се проверява дали има вписан потребител (тъй като сесията на сървъра се анулира след 30 минути неактивност).

Накрая се открива търсената крайна точка от потребителя и се проверява дали поисканият контролер е достъпен за него. В случай че не е, потребителят се пренасочва обратно към главна страница в приложението отговаряща на неговата роля.

## **3.3 Логически модел на системата**

Представяне на системата чрез Unified Modelling Language (UML) модели. UML е стандартен език за моделиране на софтуер. Използват се различни диаграми и текст, за да се опише развитието на действието на софтуерната система [7].

### **3.3.1 Диаграми на случаите на употреба**

Това е една от най-използваните диаграми, тъй като визуално представя функционалността на системата от гледната точка на потребителя. Главната идея е да се демонстрира взаимодействието на потребителите със системата. Действията се наричат случаи на употреба, а хората или другите взаимодействащи системи се наричат актьори.

Диаграмата на случаите на употреба на разработваното приложение е показана на Фигура 3.1.

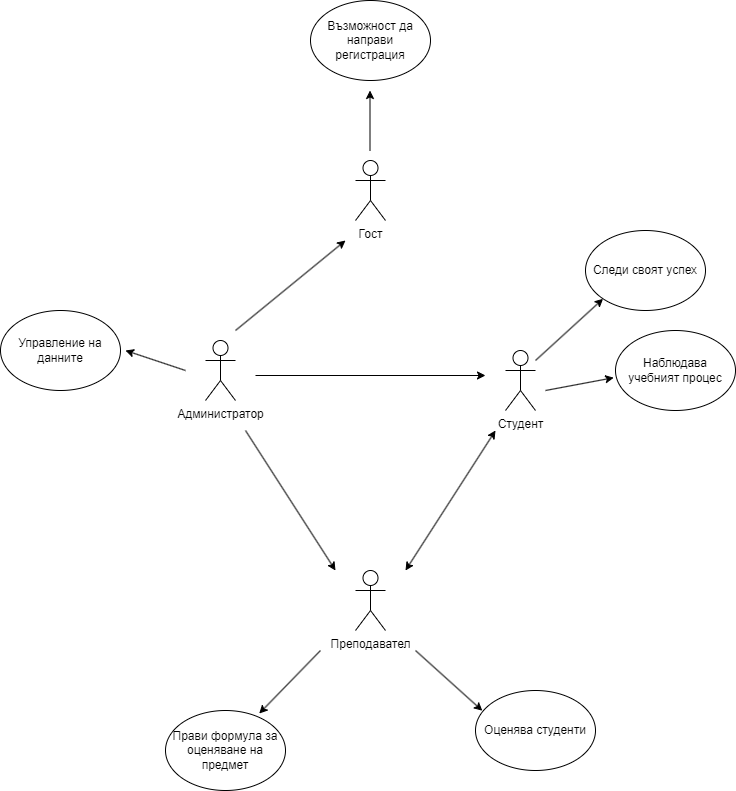
Основните потребители на системата са четири на брой.

**Гост** – потребител който не е регистриран в приложението. Има достъп да разгледа информация свързана с приложението. Също така може свободно да започне регистрация на Студентски акаунт.

**Студент** – вече регистриран потребител. Има достъп до това да разглежда различните специалности на които е записан. След избор на специалност може да види текущите изучавани предмети. И също така текущият си успех, предвиденият успех и крайната оценка ако има такава.

**Преподавател** – има достъп до предметите по които преподава, може да сменя формулата за крайната оценка. Може да оценява студентите и да слага крайна оценка. Планирано е да може да има седмичен график с предметите по които преподава.

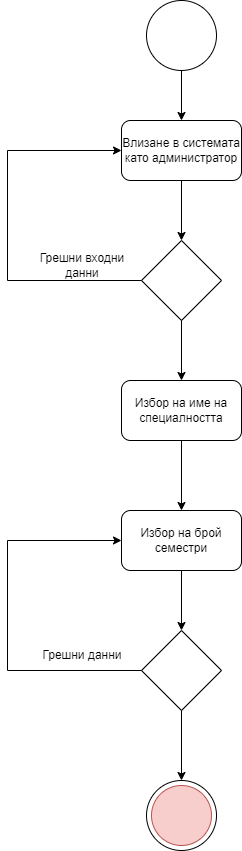
**Администратор** – това е тип потребител, който има достъп до всички ресурси на приложетнието. Може да създава учебни процеси, да добавя предмети за изучаване. Да създава нови предмети и да променя преподавателите им. Може да регистрира нови потребители и да завършва тяхната регистрация.



*Фигура 3.1 – Диаграма на случаите на употреба*

### **3.3.2 Диаграми на дейностите – Activity Diagrams**

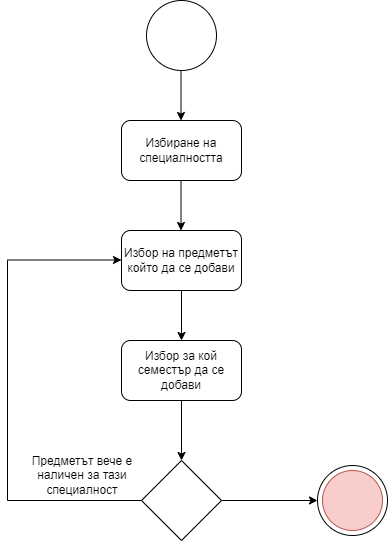
Това е друг тип UML диаграма, която се използва, за да се опишат работните процеси или дейности използвани в системата. Тя представя визуално последователността на действия и решения взети в даден процес



*Фигура 3.2 Диаграма на дейностите – създаване на нова специалност*

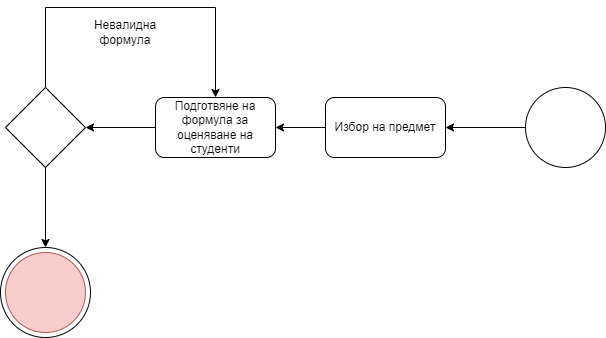
Процесът е представен на Фигура 3.2. се отнася за създаването на нова специалност.

Единствено администраторите могат да добавят нови специалности. За да се случи това трябва да се избере подходящо наименование на специалността и валиден брой на семестрите. Винаги четен брой. След това успешно се създава нова специалност.



*Фигура 3.3 Диаграма на дейностите – Добавяне на към специалност*

На Фигура 3.3 е показан процеса по добавяне на нов предмет към специалност. Тази дейност отново може да се извърши единствено от администратор. При случай, че предмет вече съществува за дадена специалност, той не се запазва.



*Фигура 3.4 Диаграма на дейностите – Добавяне на нова формула за пресмятане на крайната оценка на предмет.*

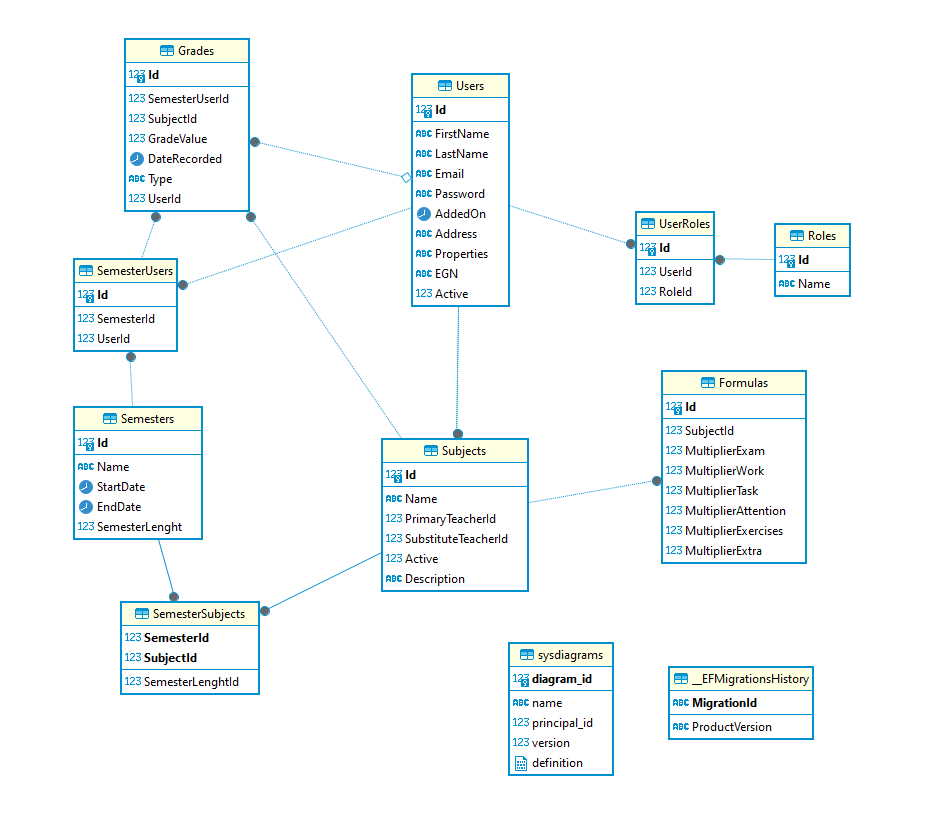
Чрез Фигура 3.4 се показва процеса по добавянето на нова формула, това действие може да се извършва от всички преподаватели. Всяка една дисциплина може да има по една формула.

# **4. Описание на базата данни**

За реализацията на системата е използвана релационна база от данни. Използвана е СУБД (RDBMS) SQL Server. DBeaver е използван като среда за визуализиране на таблиците.

## **4.1 Релационен модел на базата данни**

На Фигура 4.1 е показан релационния модел на проектираната база данни, с таблиците и техните връзки.



*Фигура 4.1 Релационен модел на базата данни*

**4.1.1. Основни таблици**

* EFMigrationsHistory – Създадена от Entity Framework – Тази таблица проверява версиите на изпълнените миграции върху базата.
* Roles – Съдържа всички валидни роли които могат да се използва от един потребител
* UserRoles – Съдържа информация за всички потребители и тяхната роля
* Users – Съдържа информация за всеки потребител
* Grades – Съдържа информация за всички оценки които се въвеждат в системата
* SemesterUsers – Съдържа информация за всеки потребител и специалността която изучава, позволя един потребител да изучава множество специалности
* Semesters – Съдържа информация за всички специалности
* SemesterSubjects – Съдържа информация за всеки изучаван предмет в дадена специалност и през кой семестър се изучава
* Subjects – Съдържа информация за всеки един учебен предмет
* Formulas – Съдържа информация за специалните формули приложени към оценяването на даден предмет

## **4.2. Описание на таблиците и връзките между тях**

### **4.2.1 Таблица Roles**

Таблицата Roles съхранява всички достъпни за употреба роли.

* Id – Уникален идентификатор (тип integer)
* Name – Наименованието на типа роля (тип string)
* UserRoles – Връзка с таблица UserRoles (тип ICollection<UserRole>)

### **4.2.2 Таблица UserRoles**

Съдържа информацията за всеки потребител и типа роля, която му е присвоена.

* Id – Уникален идентификатор (тип integer)
* UserId – Идентификатор на потребителя (тип integer)
* RoleId – Идентификатор на ролята (тип integer)
* User – Връзка с таблица Users (тип User)
* Role – Връзка с таблица Roles (тип Role)

### **4.2.3 Таблица Users**

Съдържа информацията за всеки регистриран потребител.

* Id – Уникален идентификатор (тип integer)
* FirstName – Първото име на потребителя (String)
* LastName – Второто име на потребителя (String)
* Email – Имейлът на човека (String)
* Password – Паролата, кеширана (String)
* Address – Адресът на потребителя (може да е празен) (String)
* Properties – Специални акаунтови настройки (може да са празни) (String)
* EGN – Единен граждански номер (може да е празен) (String)
* Active – Проверка дали акаунта е активен (Boolean)
* AddedOn – Дата на вписване на записа в базата (DateTime)
* UserRoles – Връзка с таблица UserRoles (тип ICollection<UserRole>)
* Grades – Връзка с таблица Grades (тип ICollection<Grade>)
* PrimarySubjects – Връзка с таблица Subjects (тип ICollection<Subject>) \*по грешка се казва PrimarySubjects – реално представлява връзката за главен учител на предмет
* SubstituteSubjects – Връзка с таблица Subjects (тип ICollection<Subject>) \*по грешка се казва SubstituteSubjects – реално е връзката за допълнителния преподавател към даден предмет

### **4.2.4 Таблица Grades**

Съдържа информацията, свързана с оценките по различните предмети.

* Id – Уникален идентификатор (Integer)
* SemesterUserId – Идентификатор към таблица Семестри (Integer)
* SubjectId – Идентификатор към таблица Предмети (Integer)
* GradeValue – Оценката попълнена от преподавателя (Decimal)
* Type – Типът на оценката (String)
* DateRecorded – Ден на вписване на оценката (DateTime)
* SemesterUser – Връзка с таблица SemesterUsers
* Subject – Връзка с таблица Subjects

### **4.2.5 Таблица SemesterSubject**

Съдържа информацията за това кой предмет в коя специалност се изучава.

* SemesterId – Уникален идентификатор по Семестъра (Integer)
* SubjectId – Уникален идентификатор по Предметът (Integer)
* SemesterLenghtId – Обозначава предметът през кой семестър се изучава (Integer)
* Semester – Връзка с таблица Semesters
* Subject – Връзка с таблица Subjects

### **4.2.6 Таблица SemesterUsers**

Таблицата съдържа информация за всяка дисциплина и за специалността, в която се изучава. Всяка дисциплина може да бъде добавена към дадена специалност само веднъж.

* Id – Уникален идентификатор (Integer)
* SemesterId – Идентификатор към таблица Семестри (Integer)
* UserId – Идентификатор към таблица Users (Integer)
* Semester – Връзка с таблица Semesters
* User – Връзка с таблица Users

### **4.2.7 Таблица Semesters**

Съдържа информацията за всяка една специалност

* Id – Уникален идентификатор (Integer)
* Name – Наименованието на специалността (String)
* SemesterLenght – Броят на семестрите за специалността
* StartDate – Времето когато започва обучението, по предназначение се запазва днешната дата на вписване на записа (DateTime)
* EndDate – Времето когато ще приключи обучението, по предназначение се пресмята използвайки Стартовата дата и се прибавя броя на семестърите. (DateTime)
* SemesterSubject – Връзка с таблица SemesterSubjects (ICollection<SemesterSubject>)
* SemesterUser – Връзка с таблица SemesterUser (ICollection<SemesterUser>)

### **4.2.8 Таблица Subjects**

Таблицата съдържа информация за всички дисциплини.

* Id – Уникален идентификатор (Integer)
* Name – Наименование на предмета (String)
* Description – Кратко описание на предмета (Може да е празно) (String)
* PrimaryTeacherId – Главният преподавател (Integer)
* SubstituteTeacherId – Допълнителният преподавател (Integer)
* Active – Логичен флаг, служи да покаже дали предметът може да се добавя за изучаване в нови специалности (Boolean)
* PrimaryTeacher – Връзка с таблица Users
* SubstituteTeacher – Връзка с таблица Users

### **4.2.9 Таблица Formulas**

Таблицата съдържа информация за всички формули, по които се изчислява финалната оценка по даден предмет. Всеки предмет може да има само една формула. Настоящата версия на проекта поддържа 6 вида формули, всяка от които може да има обща тежест от 100. Всяка формула може да приема тежести в диапазона от 0 до 100.

* SubjectId – Предметът на който принадлежи формулата (Integer)
* MultiplierExam – Формула за изпитни оценки
* MultiplierWork – Формула за оценки от тип работа (Специални семестриални курсови задачи)
* MultiplierTask – Формула за оценки от тип задачи в лекционни занятия
* MultiplierAttention – Формула за оценяване при участие в учебния процес
* MultiplierExercises – Формула за оценки от тип задачи в упражнителни занятия
* MultiplierExtra – Формула за високи успехи от Олимпиади или други извънредни участия
* Subject – Връзка с таблица Subjects

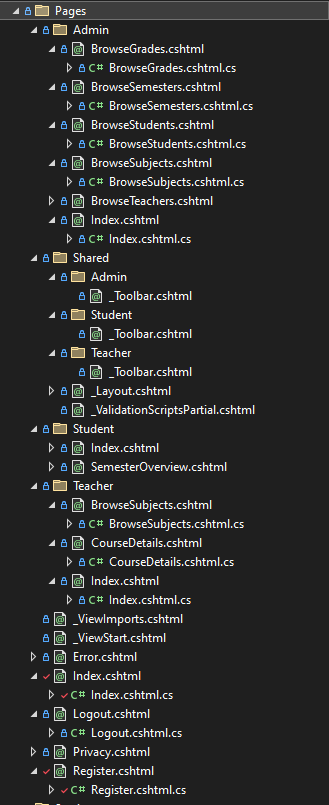
### **4.2.10 Таблица EFMigrationsHistory**

Таблица, в която се пази историята за всички направени миграции към базата. Използва се ако има нужда да се смъкне една версия назад или напред.

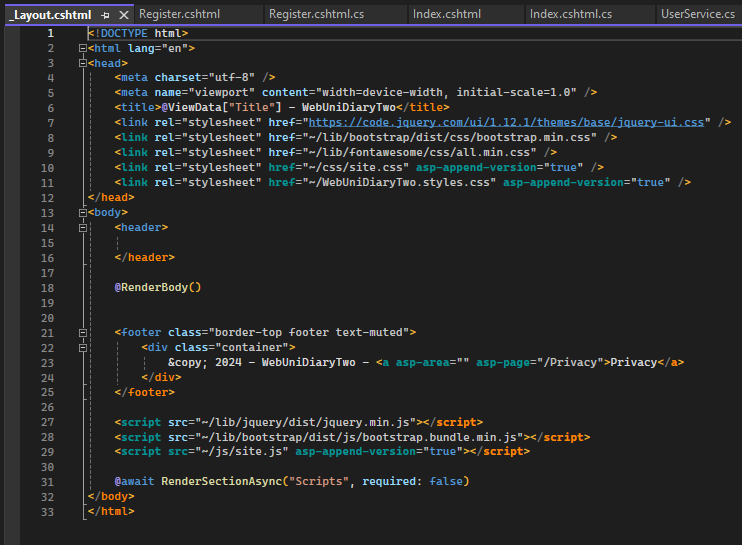
# **5. Програмна реализация**

Проектът се състои от три основни компонента: Admin, Teacher и Student, както и три помощни страници: Index, Register и Logout. По-рано беше споменато, че всеки HttpRequest се обработва от LoginMiddleware, като също така се използва UserService, който отговаря за хеширането на паролите.

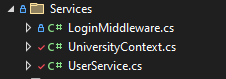
RazorPages поддържа автоматично маршрутизиране към страниците, когато се търси даден адрес. Това улеснява организацията на архитектурата на приложението. Ако възникне нужда от ново действие, свързано с администратора, е достатъчно просто да се създаде нова страница в Pages/Admin.

Също така там се обработва визията на страницата добавянето на Header + Body + Footer и също така зависимостите на Bootstrap, Ajax, Fontawesome и други. Може да се види на Фигура 5.2.

*Фигура 5.1 – Всички Страници и тяхната йерархия*



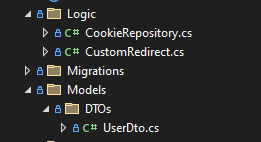
*Фигура 5.2 – Template \_Layout.cshthml*

Освен страниците, приложението притежава и различни услуги. Те се намират в папка Services това са LoginMiddleware, UniversityContext (Db relation) и UserService. Приложени са на Фигура 5.3.

*Фигура 5.3 – Раздел Services*

Препоръчително е всеки вид обработка на информация да бъде изнесен в отделен Service, за да се предотвратят проблеми в приложението и при нужда от бъдещи промени на дадена логика, тя да функционира по един и същ начин навсякъде. Засега само най-базовата логика е изнесена.

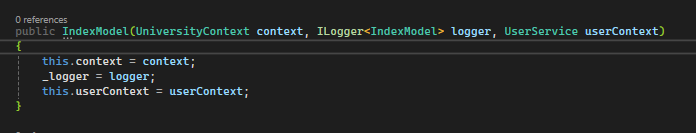
Допълнително, разработката разполага само с един модел DTO (Data Transfer Object), и това е UserDto.cs. Планира се всеки обект да има свой собствен модел за пренос на данни, чрез който да се прилагат проверки и ограничения за това какво може да се извършва с него.

Част от помощната логика на приложението е изнесена в директорията Logic/, където се намират CookieRepository.cs и CustomRedirect.cs. Тези два обекта обработват бисквитките, запазвани в браузъра, и пренасочването на регистрираните потребители. Например, ако вече сме логнати, няма нужда localhost/Register да ни насочва към формата за регистрация.

*Фигура 5.4 – Допълнителната логика и Моделите за DTOs*

## **5.1 Инжектиране на зависимости в Razor Pages (Dependency Injection)**

Инжектирането на зависимости (Dependency Injection, DI) е важен модел за управление на софтуерните зависимости, който улеснява изграждането на гъвкав и проверим код. В контекста на Razor Pages в ASP.NET Core, DI се използва широко за предоставяне на услуги и компоненти на страниците и техните модели (Page Models), без да е необходимо те сами да създават тези зависимости.



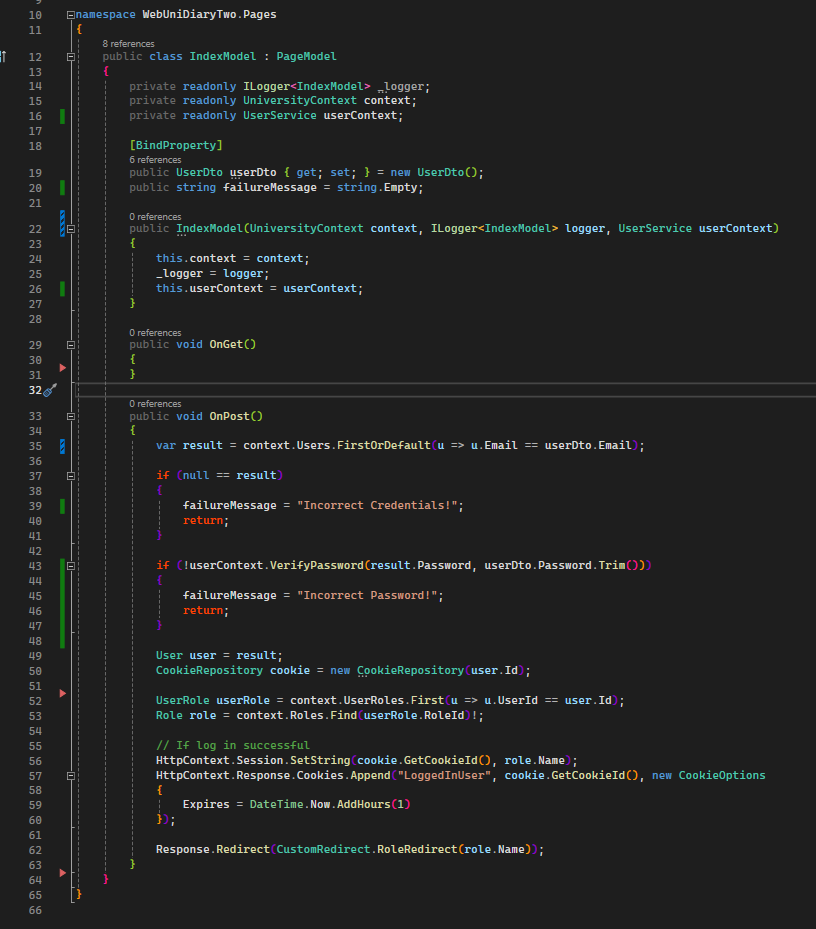
*Фигура 5.5 – Начална на приложението и нейният конструктор.*

На горната фигура е демонстрирано как чрез конструктора се добавя връзката с базата данни UniversityContext, обработката на данни, свързани с потребителя, чрез UserService и интерфейсът ILogger, използван за извеждане на логове в конзолата на сървъра.

## **5.2 Основни страници**

### **5.2.1 Начална страница (Index)**

Използва се за показване на началната страница, където потребителите имат опцията да се впишат в приложението или да се регистрират.



*Фигура 5.6 – Как точно изглежда една .cshtml.cs страница*

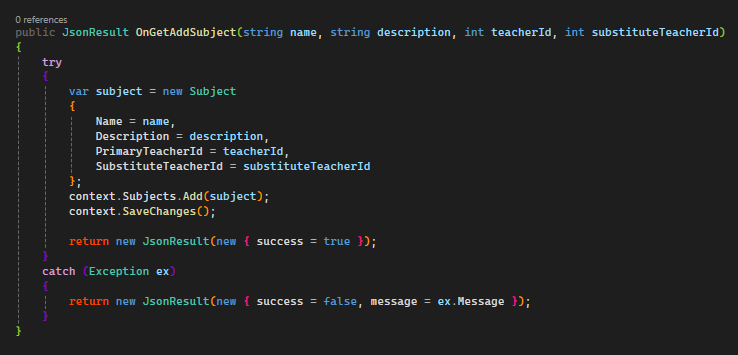
Всяка Razor страница съдържа методи като OnGet() и OnPost(), които са известни като Handler Methods. Те отговарят за това какво се случва, когато страницата се заявява (GET) или когато изпращаме данни към контролера на страницата (POST).

OnGet(): Изпълнява се, когато страницата се зарежда чрез GET заявка. Обикновено тук се инициализират данни, които трябва да бъдат показани на страницата.

OnPost(): Изпълнява се, когато потребителят изпрати данни чрез формуляр на страницата (POST заявка). Обикновено тук се обработват и валидират изпратените данни.

Освен стандартните OnGet() и OnPost() методи, всяка Razor Page може да поддържа и именовани Named Handlers, като например OnGetAddSubject(). Това са разширени методи, които се използват за обработка на специфични действия или събития, често активирани чрез AJAX заявки или допълнителни действия на потребителя.

Пример за разширен Handler е показан във Фигура 5.7. Той е OnGet() хандлър със специфично име AddSubject.



*Фигура 5.7 – Разширен Handler – за допълнителна логика.*

Как се вика този хандлър -> url: '@Url.Page("/Admin/BrowseSubjects", new { handler = "AddSubject" })'

Както се забелязва, използван е OnGet(), за да се изпратят данни, но реално трябва да е OnPost(). Поради неясна грешка, Ajax заявката не се обработваше от приложението, което наложи употребата на OnGet() handler. В бъдещи версии това ще бъде коригирано.

Относно Index() страницата: Тя служи като главна страница и помага на потребителите да се впишат в системата. Използва DTO (data transfer object) - userDto, за да валидира данните, и след успешна валидация потребителят се запазва в сесията и му се създава бисквитка, за да остане вписан. Още една област за подобрение е, че CookieRepository няма Factory обект, който да генерира бисквитката. В момента това действие е директно записано в самото репозитори.

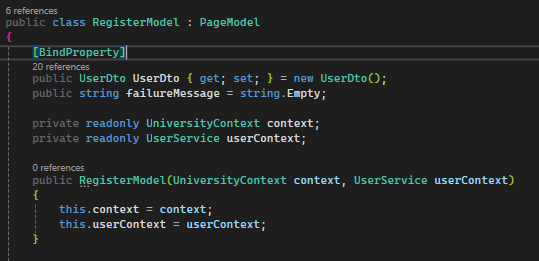
### **5.2.2 Страница Register**

На тази страница се обработват данни, свързани с регистрацията на нов потребител, и се проверява дали потребителската роля съществува в базата данни. Ако не съществува, тя се добавя. Хеширането на пароли се извършва чрез PasswordHasher, наличен в библиотеката Microsoft.AspNetCore.Identity. Накратко, използва се хеширащият алгоритъм PBKDF2, който прави разбиването на хеша много трудно, като хеширането е 256-битово.

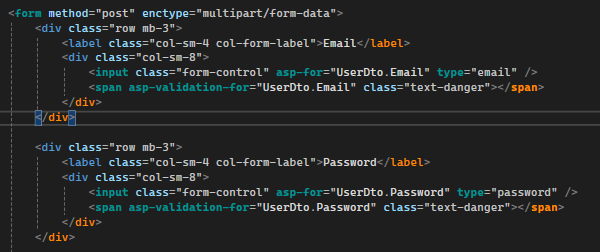
След успешната регистрация на нов потребител, контролерът генерира нова бисквитка за този потребител и я запазва в сесията на сървъра. По този начин потребителят автоматично се вписва в системата и се пренасочва към началната страница според своята роля.

Използва се само методът public void OnPost().

По-рано в началната страница беше споменато, че приложението използва UserDto. Тук ще разгледаме как чрез [BindProperty] регистрационната страница валидира потребителската заявка.



*Фигура 5.8 – Страница Register контролер*

**

*Фигура 5.9 – Страница Register html*

Фигури 5.8 и 5.9 демонстрират как в C# се обявява, че даден обект ще валидира своите променливи и при проблем <span asp-validation-for> ще съдържа съответната стойност. Това се случва след изпращането на заявката, и ако възникне проблем, потребителят ще бъде пренасочен обратно към същата страница.

### **5.2.3 Страница Logout**

Тази страница се използва единствено за отписването на вписан потребител от сесията му. Бисквитката с неговия уникален ключ за достъп се изтрива от браузъра, след което той се премахва от сесията на сървъра.

Страницата няма визуална част, а потребителят винаги се пренасочва към началната страница чрез Response.Redirect("/"). В бъдеще е планирано да се запазва входният достъп на потребителя в базата данни, така че дори при рестартиране на сървъра, ако браузърът съдържа бисквитка, потребителят да бъде вписан автоматично.

### **5.2.4 Страница Admin – BrowseGrades**

Това е страницата, на която се показват всички оценки на всички студенти за определен предмет и специалност. За да се реализира това, както е показано на Фигура 5.10, трябва да се извлекат данните за конкретната дисциплина на дадена специалност, както и да се намерят оценките за тази дисциплина. Използва се eager loading, за да може с една заявка да се съберат всички нужни данни:

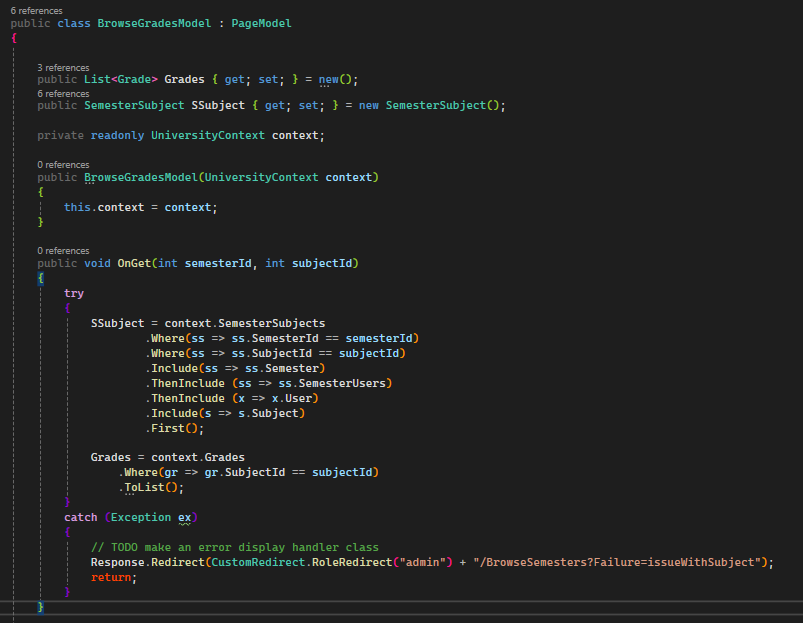
* Данни за специалността (Semester)
* Данни за дисциплината (Subject)
* Данните за студентите (User)

По този начин, чрез вече дефинираните връзки в ERM, C# обектите могат логически да се свържат и да се заредят.

При проблем или липса на данни, администраторът ще бъде върнат на предишната страница.

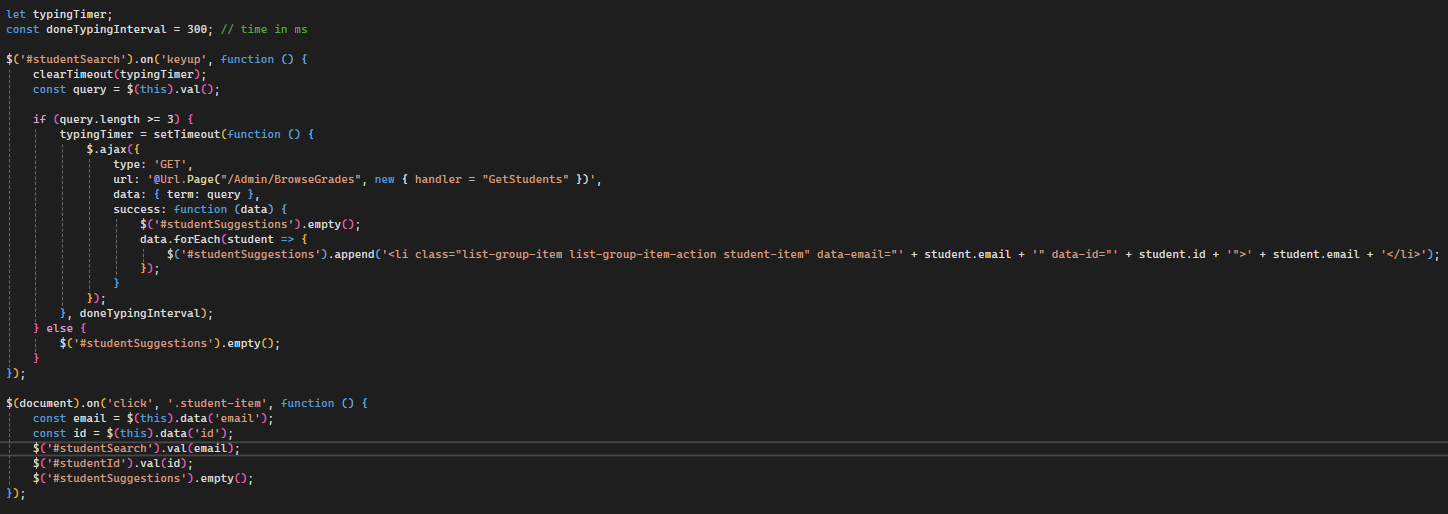
На тази страница се поддържа серия от допълнителни AJAX handlers.

1. AddGrades – IActionResult метод, който при подаване на семестъра, дисциплина, оценката и типа на оценката, ще направи опит да я запази в базата данни. При успех ще презаредим страницата за да се визуализира новата оценка.

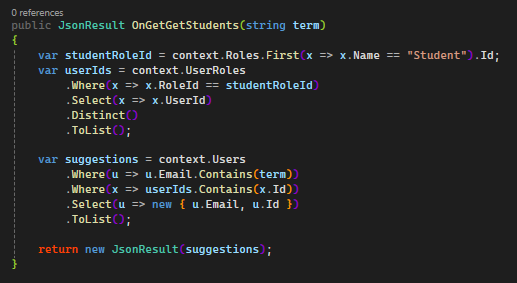


*Фигура 5.10 – Страница BrowseGrades използваща eager loading*

1. AddStudent – Метод, с който лесно можем да добавим нов студент към специалността. Отново е реализиран чрез IActionResult и изисква подаването на студентския номер и семестъра на студента. Извършват се серии от проверки, преди студентът да бъде добавен към специалността. Първо, потребителят трябва да е с роля "студент", както и студентът да не присъства вече в тази специалност.
2. GetStudent – Този метод предоставя списък с имейл адреси на студенти в реално време, докато се попълва формата за добавяне на нов студент, което улеснява администраторът при избор на правилния потребител. Методът връща JsonResult. На Фигури 5.11.1 и 5.11.2 е показано как работи тази функционалност. Чрез HTML append старите данни се изтриват и се заменят с нови имейл адреси. В момента заявки се изпращат само ако потребителят е написал повече от 3 букви, а интервалът между заявките е над 1/3 от секундата. След това потребителят може да избере правилния студент чрез кликване с мишката.



*Фигура 5.11.1 – JS ajax метод за попълване на потребители, когато се търси чрез имейл адрес*

**

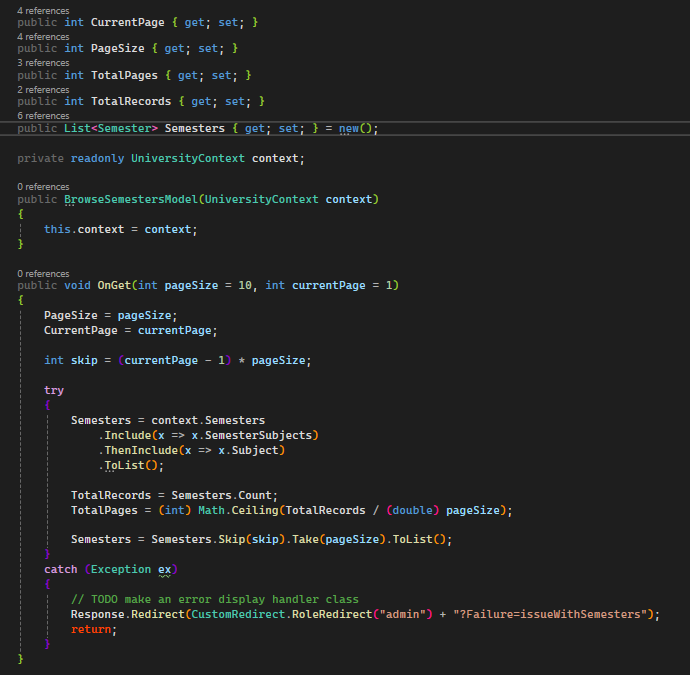
*Фигура 5.11.2 – C# частта която отговаря за обработването на ajax request*

### **5.2.5 Страница Admin - BrowseSemesters**

На тази страница администраторите на приложението могат да видят всяка добавена специалност. Тук те могат да добавят нови специалности, да добавят предмети към тях или да променят данни по вече съществуващи специалности.

Контролерът на страницата притежава един OnGet() метод, който филтрира и организира страниците на списъка със специалности. Използването на тази техника позволява разделянето на голям обем данни на по-малки части, разпределени на множество страници. Потребителят има възможността да определи броя на показваните данни на екрана. Това се реализира чрез следене на текущата страница, общия брой данни, възможни за показване на екрана, броя на необходимите страници за показване на всички данни, както и общия брой записи.

След като разполагаме с тази информация, можем да филтрираме заявката към базата данни, за да получим само нужните данни. На Фигура 5.12 е демонстриран OnGet() методът на този контролер. Следващите страници, които използват странициране, следват същата логика.



*Фигура 5.12 – Странициране на BrowseSemesters*

Освен това, чрез AJAX заявки е имплементирана възможността за добавяне на нова специалност. Методът AddSemester връща IActionResult и очаква име на специалността и продължителност, подадена като брой на семестрите. Не е възможно добавянето на нечетен брой семестри. При успех страницата се презарежда.

Други два AJAX метода са тези за редактиране на специалност и вземане на данни за специалност, които работят заедно. GetSemester и EditSemester връщат IActionResult. При промяна на броя на семестрите се обновява и крайната дата на специалността.

Последните два AJAX метода на тази страница са свързани с добавянето на нова дисциплина към даден семестър на специалност. AddSubject служи за добавянето на новата дисциплина и връща IActionResult, докато методът GetSubject подпомага autocomplete функцията на формата. Когато потребителят въведе 3 букви от името на дисциплината, се прави заявка към базата данни и се връщат всички дисциплини, съдържащи тази комбинация.

### **5.2.6 Страница Admin – BrowseStudents**

Използва се, за да се покажат всички студенти, регистрирани в приложението. Контролерът има OnGet() метод и няколко AJAX метода.

Както при предишните страници, в OnGet() се реализира логиката, свързана със страницирането на страниците. Първо се събират всички потребители със студентска роля, след което те се разделят според лимита на записи, които могат да се видят на една страница.

От тази страница, освен разглеждането на всички студенти, може да се добавя нов студент чрез AJAX метода AddStudent, който е от тип IActionResult. Важно е, че имейлът винаги трябва да бъде уникален, за да се добави нов запис.

Друга функционалност е промяната на статуса на студент – дали е активен или не. Това се извършва чрез AJAX метода ToggleStatus, като се изпращат потребителският номер и флагът дали новият статус е true или false. Методът връща IActionResult.

Последната основна функционалност е промяната на данни на студент. Първо, данните на даден потребител се попълват във формата чрез AJAX заявка GetStudent, а след това, при изпращане на промените, те се обработват от EditStudent. И двата метода са от тип IActionResult.

### **5.2.7 Страница Admin - BrowseSubjects**

Тази страница се използва, за да се покажат всички налични предмети, добавени в приложението. Контролерът на страницата предлага OnGet() метод, който подготвя данните за страницираенто, както при предишните страници. Освен заявката за предметите, тук се прави и такава за подготовка на данните за преподавателите, свързани с тези предмети.

Контролерът също така предоставя възможност чрез AJAX за добавяне на нови предмети, редактиране на съществуващи предмети, автоматично предсказване на имейл адресите на преподавателите при добавяне на нов предмет или редактиране на съществуващ, както и AJAX заявка за логическа деактивация на дисциплините.

* AddSubject – връща JsonResult – обработва добавянето на нова дисциплина.
* EditSubject – връща IActionResult – отговаря за редактирането на вече съществуваща дисциплина.
* GetSubject – връща IActionResult – връща търсена дисциплина, като се подаде Id.
* ToggleStatus – връща IActionResult – обновява статуса на дисциплината чрез Boolean флаг.
* GetEmail – връща JsonResult – връща потребители отговарящи на даден имейл адрес. Като тези потребители могат да се единсвено преподаватели.

### **5.7.8 Страница Admin - BrowseTeachers**

Страницата се използва, за да се покажат всички преподаватели в приложението. Контролерът отново предлага OnGet() метод, който отговаря за подготвянето на данните на страницата. Също така, чрез AJAX заявки се обработва логиката по добавянето на нов преподавател, редактирането на данни на вече съществуващи преподаватели, както и логическото деактивиране и активиране на акаунтите.

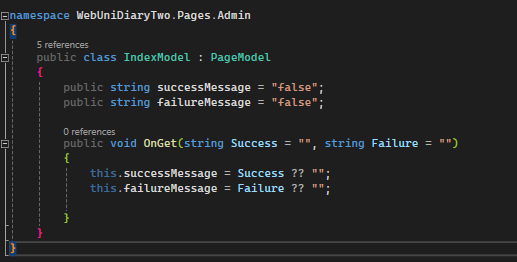
Всички ajax заявки:

* AddTeacher – връща IActionResult - Служи за добавяне на нов преподавател. Имейл адресът трябва да е уникален. При успешно добавяне страницата се презарежда.
* EditTeacher – връща IActionResult – Изпълнява промятната на данни свързани с вече съществуващ преподавател.
* GetTeacher – връща IActionResult – Спомага попълването на данните на формата за редактиране на преподавател
* ToggleStatus – връща IActionResult – обновява логическия статус на преподавателя

### **5.2.9 Страница Admin - Index**

Това е главната страница на админският панел. В нея има линкове по подразбиране за бърз достъп към дадена страница. Тези линкове могат да се конфигурират.

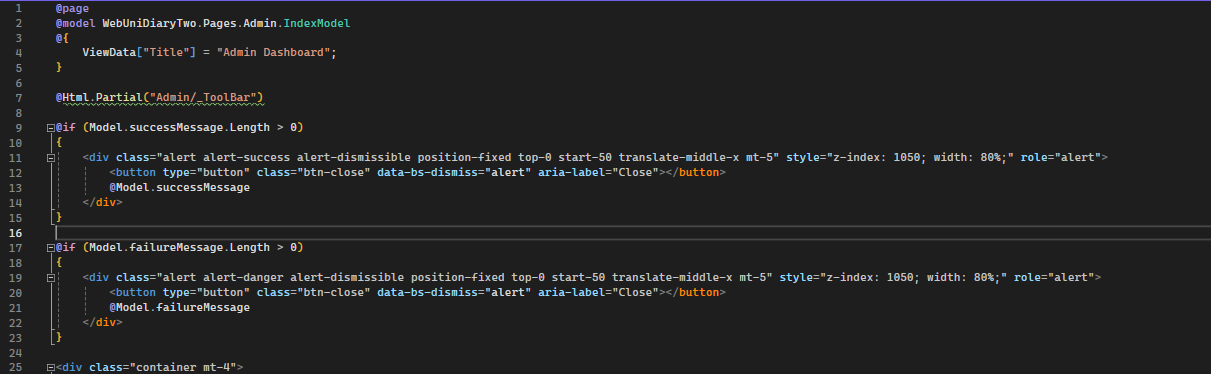
Допълнително главната страница се използва за редирект, при случай че има проблем с приложението. Или е извършено някакво действие успешно, което няма страница за пренасочване.



*Фигура 5.13 – Как изглежда контролера на Admin/Index страницата*

На Фигура 5.13 и 5.14 може да се види как изглежда index страницата. По този начин се извеждат всякакви информационни съобщения в приложението.

Допълнително чрез @Html.Partial() се вмъкват помощните изгледи. В случая на Фигура 5.14 се вижда как се добавя администраторския панел с линковете, в горната част на страницата.



*Фигура 5.14 – Част от Admin/Index страницата*

### **5.2.10 Страница Student - SemesterOverview**

Тази страница се използва от студентите, като тук те могат да видят своите оценки по предметите, които изучават, в зависимост от избраната специалност.

Контролерът има само един OnGet() метод, който приема номера на специалността. След това се извличат данни единствено за съответния потребител и тази специалност. Ако студентът не изучава тази специалност, той ще бъде пренасочен към главната страница.

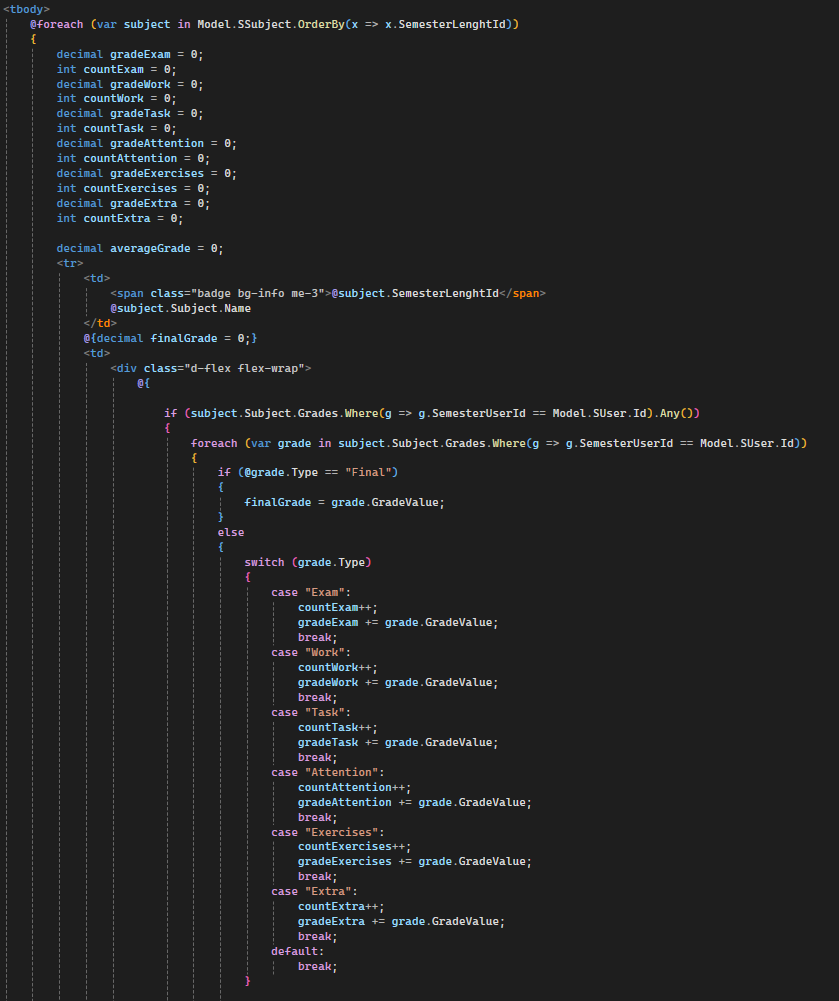
Първо се извличат всички оценки и се инициализират променливите gradeExam, gradeWork, finalGrade и други. При всяко добавяне се брои броят на въведените оценки с помощта на други променливи като countExam, countWork и т.н. След това за всеки тип оценка резултатът се дели на броя оценки, умножава се по тежестта на формулата и се дели на 100.

Как точно работи логиката по изчисляването на препоръчителната крайна оценка по дадена дисциплина. Първо се намират всички оценки за дадена дисциплина за даден студент. След това се филтрират по категории (Примерно оценки от изпит, оценки по задачи и други). Пресмята се индивидуално броят на оценките по групи, после на всяка група се сумира средната оценка, тя се умножава по тежеста на формулата свързана с тази група и накрая се дели на 100. По този начин получаваме успехът за дадена група, който използваме за да получим крайната оценка.

Пример: Има 3 оценки от тип Exam (5, 6, 6). Събираме ги и получаваме 17, след което делим 17 на 3 и получаваме 5,666. Накрая умножаваме по тежестта (примерно 80% от оценката) – 5,666 \* 80 = 453,333, и разделяме на 100. Крайният резултат е 4,53 (след закръгляне до втория знак). По този начин, след като студентът получи и другия тип оценка, оставащите 20% също могат да се добавят и да формират крайния успех по предмета.

На Фигура 5.15 и Фигура 5.16, са показани съответно страниците, на които е приложена тази логика.

В бъдеще е препоръчително логиката за изчисляване да бъде изнесена в Repository, което значително ще улесни повторното използване на тази логика на различни страници.



*Фигура 5.15 – Инициализирането на променливите свързани с успеха*

**

*Фигура 5.16 – Пресмятането на различните оценки и средният успех*

### **5.2.11 Страница Student - Index**

Страницата Student/Index е началната страница за всеки вписан студент. Тя предоставя достъп до преглед на всички специалности, които студентът изучава, както и информация за продължителността на всяка специалност.

Контролерът на страницата съдържа OnGet() метод, който проверява дали логнатият студент има записани специалности, и ако има, показва какво изучава.

### **5.2.12 Страница Teacher – BrowseSubjects**

На тази страница преподавателите могат да видят всички предмети, на които преподават. Контролерът отговаря за OnGet() метод и няколко AJAX заявки.

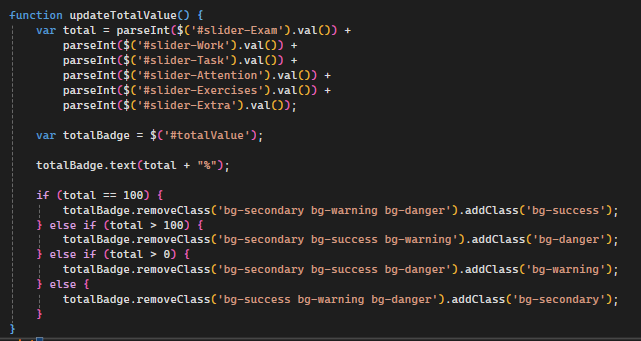
Първо, предметите се сортират по това дали преподавателят е главен или помощен преподавател.

Първата AJAX функция, GetFormula, отговаря за връщането на текущата формула за дадена дисциплина, ако изобщо има такава. Резултатът е IActionResult.

Другият AJAX метод, SaveFormula, служи за запазване на правилно попълнена формула за дадена дисциплина. Важно е, че общият процент на всички полета във формулата трябва да бъде точно 100%. Само тогава резултатът се запазва.

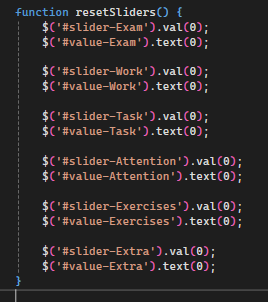
Освен AJAX заявки, на страницата има и няколко помощни JS метода.

Първият метод е за визуално известяване при промяна на стойностите във формата, показващ дали текущата настройка е валидна. Това е илюстрирано на Фигура 5.17.



*Фигура 5.17 – Js метод за визуално известяване на потребителя дали текущата настрайка на формулата е валидна*

Друг помощен метод е resetSliders(), който служи за връщане на стойностите обратно на 0%. Това е необходимо в случаите, когато избраният предмет няма предварително зададени стойности, и формата трябва да се върне към настройките по подразбиране. Методът също позволява на потребителя да нулира данните, ако не е доволен от крайния резултат. Действието на този метод е показано на Фигура 5.18.

*Фигура 5.18 – Js метод, служещ за възстановяване на стойностите на формата*

### **5.2.13 Страница Teacher – CourseDetails**

Това е страницата, където преподавателите могат да нанасят оценки, да проверяват текущия успех на студентите и да въвеждат крайна оценка. След като крайната оценка вече е въведена, не могат да се нанасят нови оценки по тази дисциплина за съответния студент.

Контролерът на страницата предоставя OnGet() метод и два AJAX метода.

За да се визуализира правилно страницата, в OnGet() метода трябва да се подадат правилният номер на семестъра и номерът на дисциплината. След това от базата данни се зареждат информацията за дисциплината, оценките и наличната формула за изчисляване на средния успех по дисциплината. Допълнително, оценките на тази страница се групират по тип и по успех, за улеснение на преподавателя.

Контролерът също така отговаря и за 2 ajax метода:

* SubmitGrade – Тук се обработва запазването на нова оценка, важно е тя да е реално положително число. По-голямо или равно на 2 и по-малко или равно на 6. При всеки друг случай оценката е невалидва.
* SubmitFinalGrade – Тук се запазва финалната оценка по предмета. Отново трябва да отговаря на условията за валидна оценка и чак тогава се запазва в базата данни. Важен момент е, че крайната оценка се въвежда по преценка на преподавателя. Дори средният успех да е 4,5 преподавателя има свободата да нанесе друга оценка различна от 5.

Страницата предлага валидация дали има подготвена формула за дисциплината. И показва визуално ако липсва такава, за да може преподавателят да въведе нова при този случай.

### **5.2.14 Страница Teacher – Index**

Това е началната страница за всички преподаватели. На нея могат да се видят линкове за бърз достъп до други страници, които могат да бъдат конфигурирани от настройките на акаунта.

Контролерът на страницата съдържа OnGet() метод, който подготвя съобщенията по страницата. Страницата е подобна на тази на администраторите.

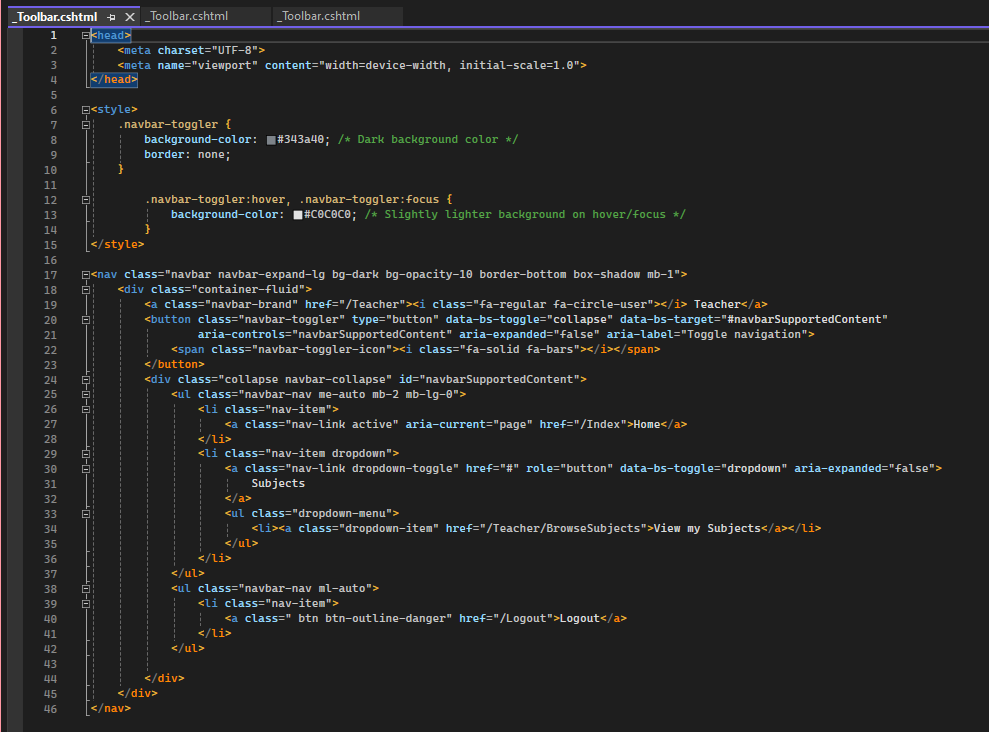
### **5.2.15 Страница Shared**

Тук се съдържат всички споделени HTML темплейти, използвани в другите страници.

На Фигура 5.19 е даден пример за това как изглежда една споделена страница. Вграденият CSS се използва за падащите менюта при избора на линк. Останалата част от HTML е свързана с навигационния панел. По този начин всеки от контролерите получава линкове, които са свързани с неговата функционалност.

Пример за това как се извиква shared view -> @Html.Partial("Teacher/\_ToolBar")

Общо споделените изгледи са 4: Admin/\_Toolbar.cshtml, Teacher/\_Toolbar.cshtml, Student/\_Toolbar.cshtml, \_Layout.cshtml – отговарящ за изгледа на всяка една страница, както бе споменато по-рано.



*Фигура 5.19 – Shared View / Teacher / \_Toolbar.cshtml*

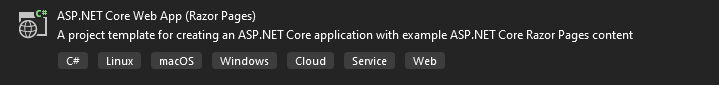
# **6. Ръководство за работа със системата**

## **6.1 Инсталиране на системата.**

### **6.1.1 Нужни ресурси за инсталирането на системата**

За успешна инсталация са нужни следните софтуерни изисквания:

* Visual Studio 2022.
* Добавка ASP.NET Core Web App (Razor Pages) (Фигура 6.1).
* .NET 8.0 – Configured for HTTPS, Do not use top-level statements.
* Microsoft SQL Server.



*Фигура 6.1 – ASP.NET*

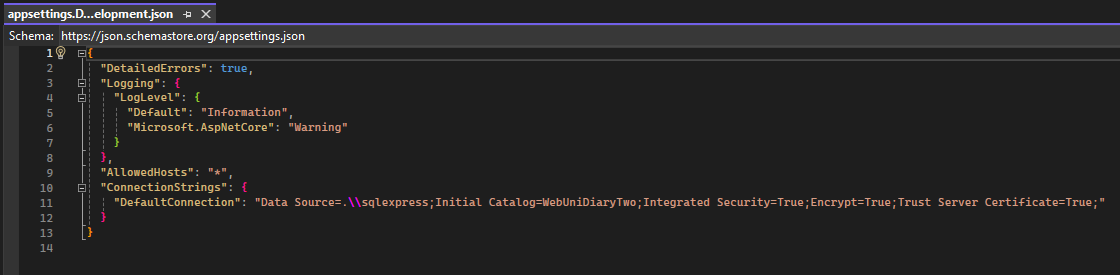
### **6.1.2 Стартиране на проекта**

Първо трябва да се потвърди, че connection стрингът е настроен правилно. Фигура 6.2 показва следния пример:

"DefaultConnection":"Data Source=.\\sqlexpress;Initial Catalog=WebUniDiaryTwo; Integrated Security=True;Encrypt=True;Trust Server Certificate=True;"

След това, при нужда, трябва да се конфигурира SQL Server, като задължително версията му трябва да бъде над 2014, тъй като eager loading е достъпен от тази версия нагоре.

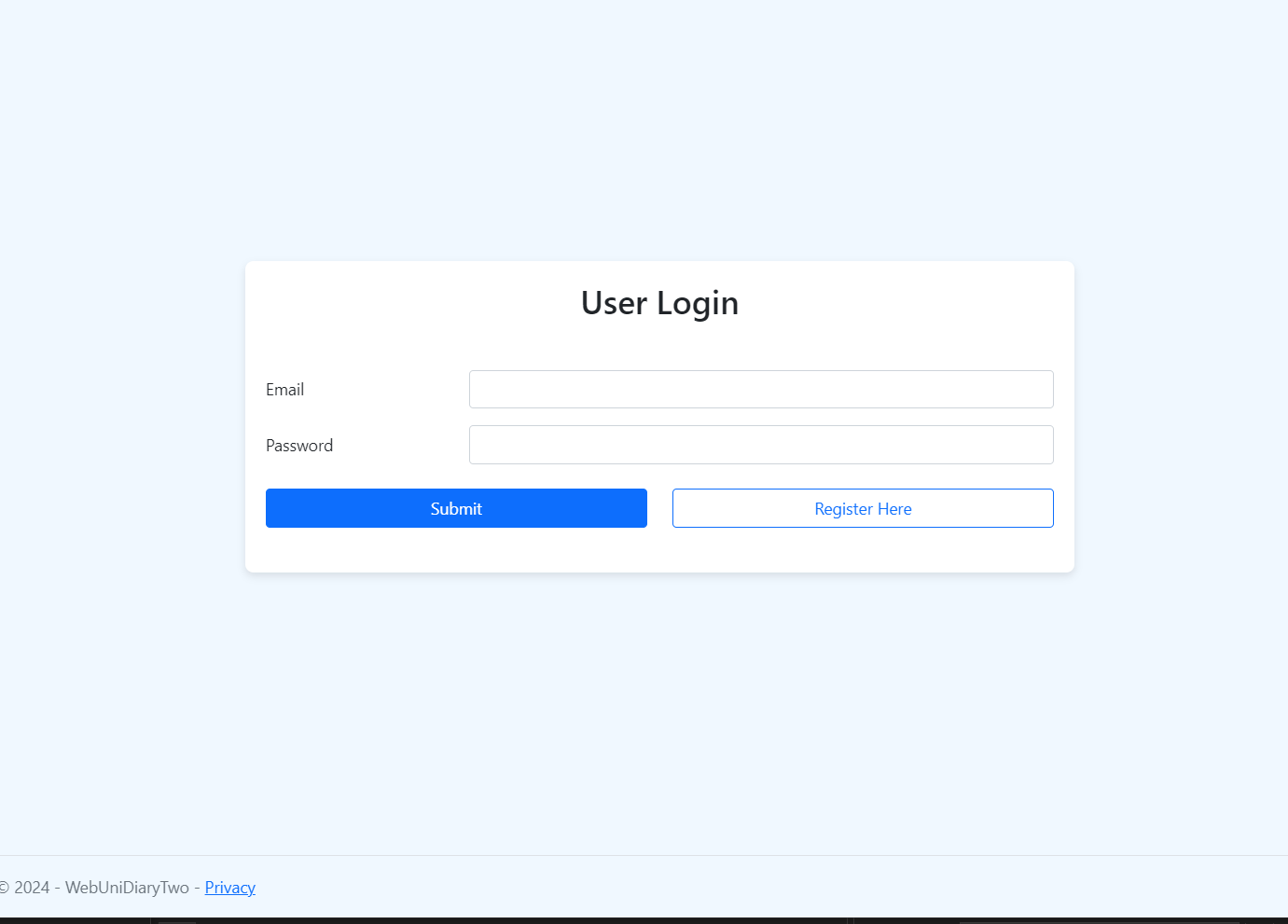
При първото стартиране на SQL Server, трябва да се добави потребител от всеки тип (администратор, преподавател и студент), за да могат потребителските групи да бъдат създадени.



*Фигура 6.2 – Конфигурационни данни*

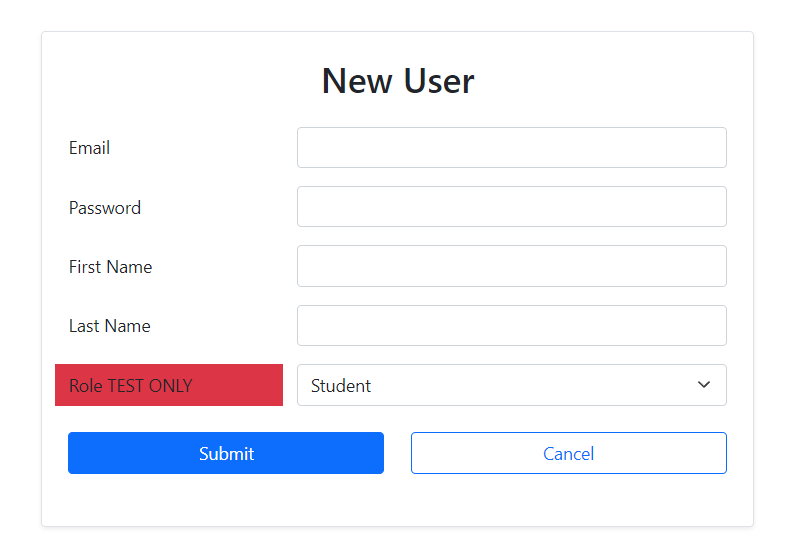
След това може да се стартира поректа. При успех ще се покаже Log In страницата след като се стартира преложението. Фигура 6.3 показва точно това.

Тук потребителите имат опцията да влязат в собственият си акаунт или да се регистрират.



*Фигура 6.3 – Начална страница на системата*

## **6.2 Регистрация на нови потребители**

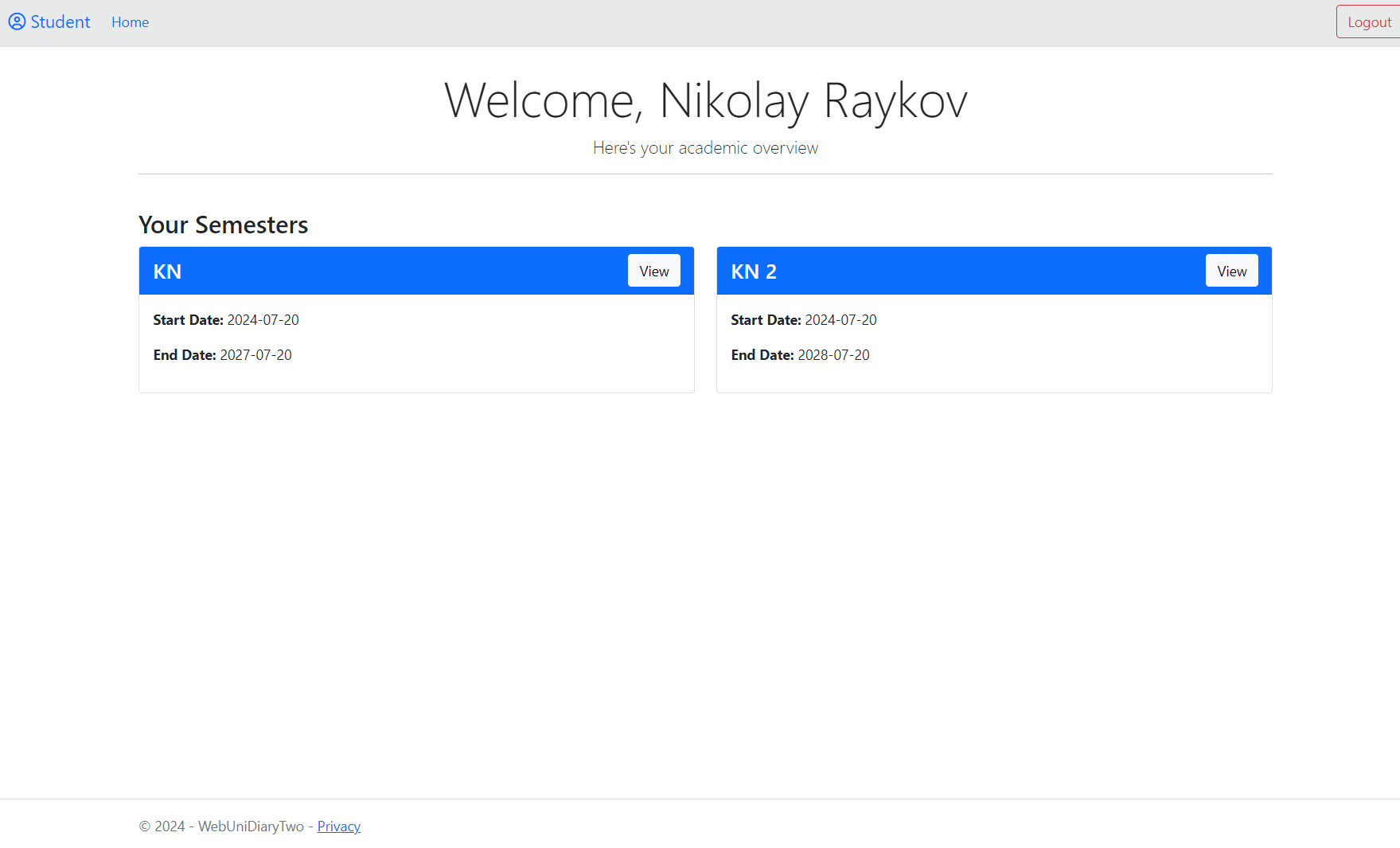


*Фигура 6.4 – Регистрационната форма*

Регистрирането на нови потребители се извършва след избиране на опцията Register Here, както е показано на Фигура 6.3. След това ще се зареди страницата, показана на Фигура 6.4, където могат да се попълнят данните и да се създаде нов студент. Допълнително, за целите на демонстрацията, от тук могат да се добавят също преподаватели и администратори.

### **6.3 Употреба на приложението като студент**

След успешно вписване в платофомата (Фигура 6.3) всеки студент ще бъде приветсван от главната страница на контролера.

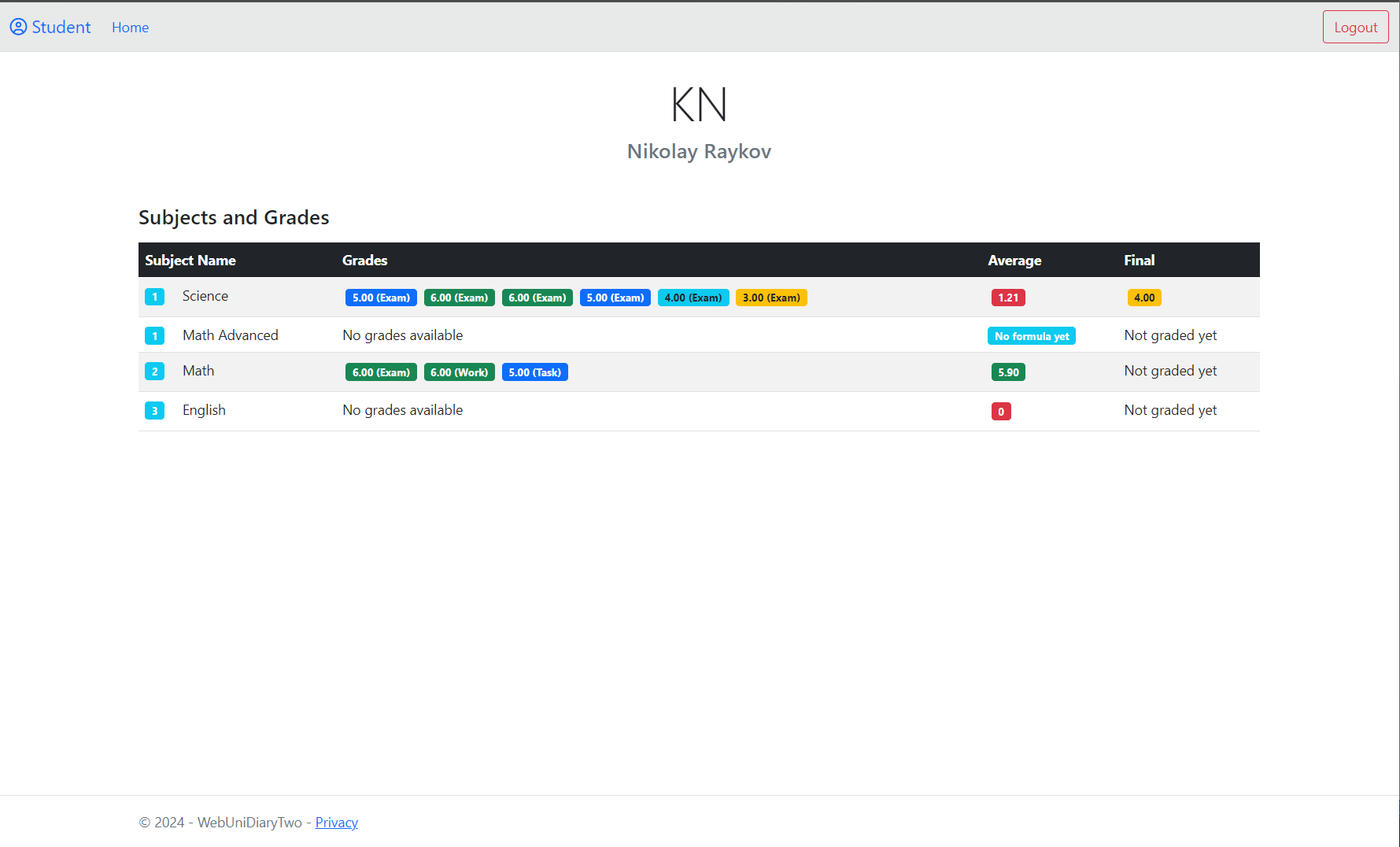


*Фигура 6.5 – Успешно вписан студент, страница Student/Index*

Тук всеки студент може да избере и разгледа своя учебен план или да прегледа профила си в приложението.

Също така приеложението поддържа опцията за изучаване на множество специалности от един студент. При избор на специалност се показват всички дисциплини, текущият успех, средната пресметната оценка по дисциплината и, ако преподавателят е оценил студента, крайната оценка.

Както е показано на Фигура 6.6, може да се види страницата и различни примерни ситуации: как изглежда дисциплина с крайна оценка, пресмятането на средната оценка по формула, както и случаите, в които студентът няма нанесени оценки или дисциплината няма зададена формула за пресмятане на крайната оценка.

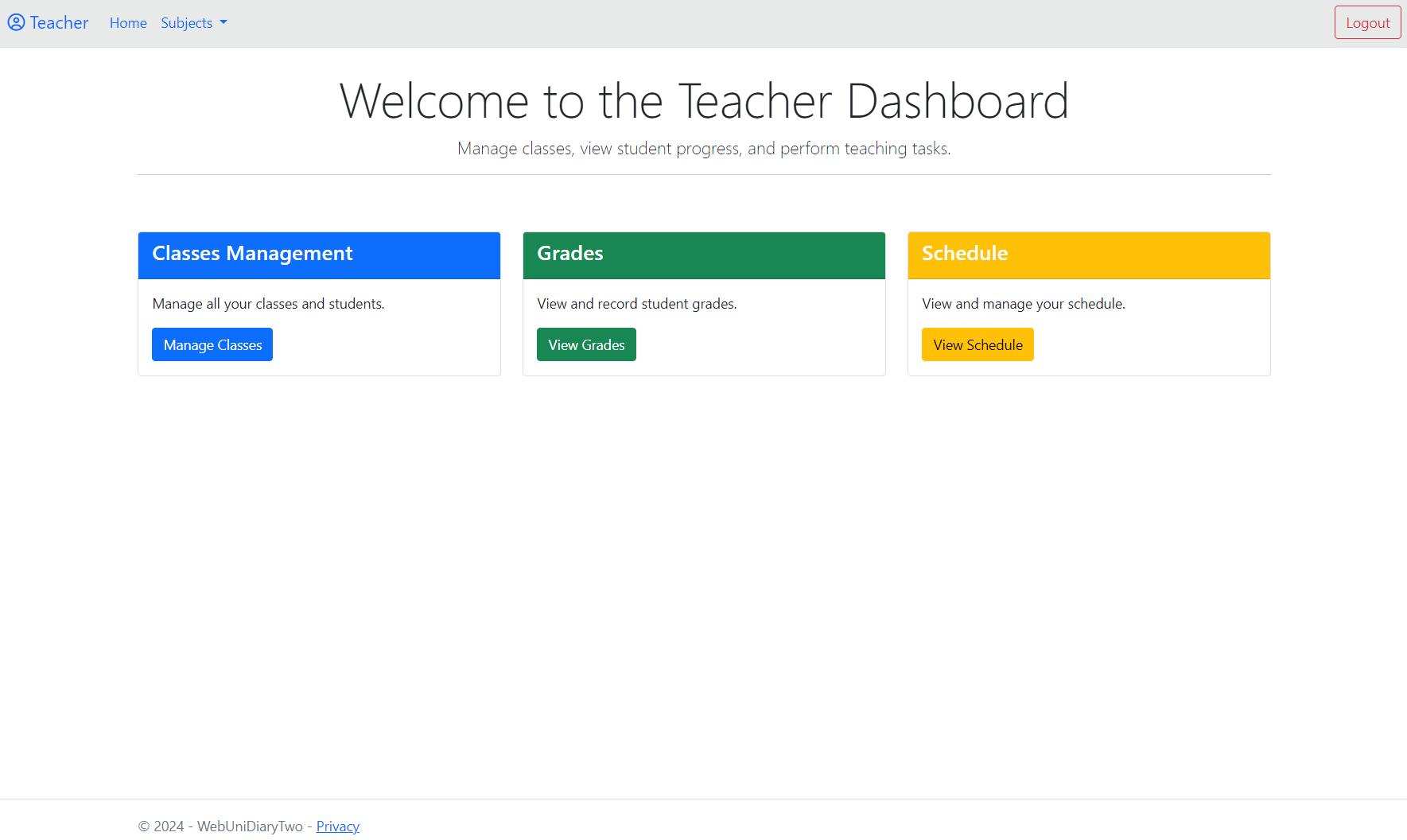


*Фигура 6.6 – SemesterOverview може да се види академичното състояние на студентът*

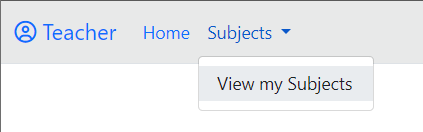
### **6.4 Употреба на приложениетж като преподавател**

След вписване в системата (Фигура 6.3) преподавателите ще бъдат пренасочени към главната страница контролера.

От тук на татък те имат достъп до всички инструменти предназначени за преподавателите.



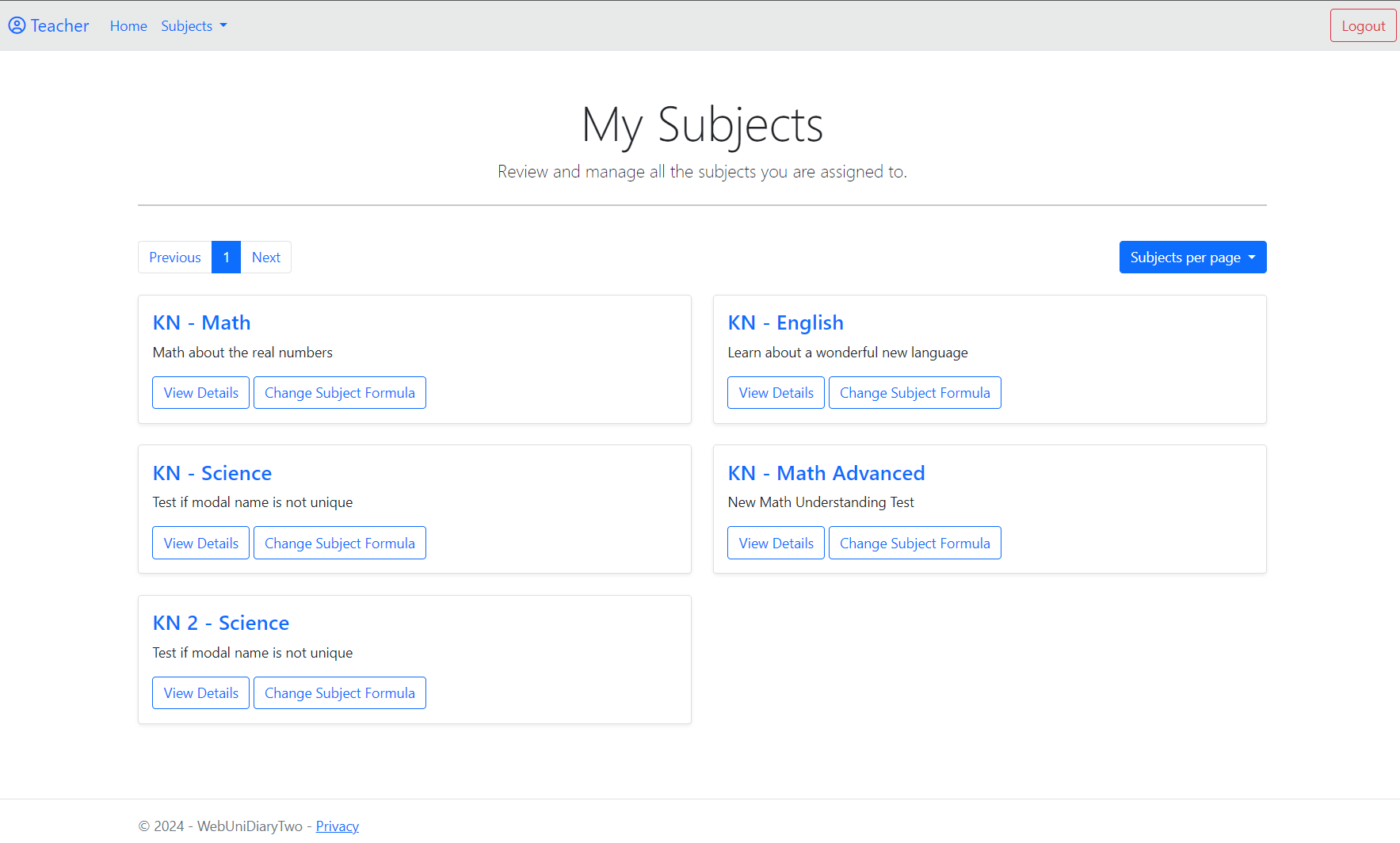
*Фигура 6.7 – Главна страница на Преподавател – Teacher/Index*



*Фигура 6.8 – Разглеждане на предметите на даден преподавател*

При избор на опцията за разглеждане на предметите (Фигура 6.8), преподавателят се пренасочва към Teacher/BrowseSubjects, откъдето има достъп до всеки предмет, по който преподава. Предметите са подредени според специалностите, които ги изучават. На Фигура 6.9 може да се види как изглежда примерна страница.

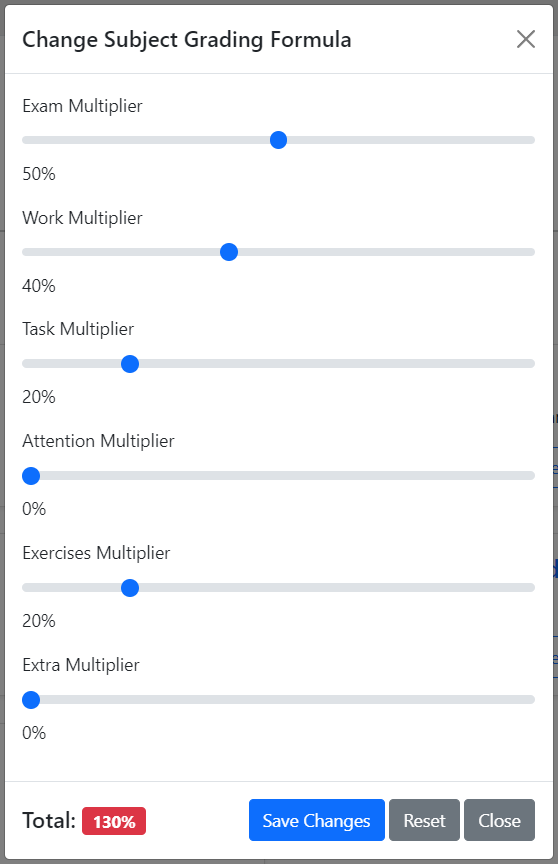
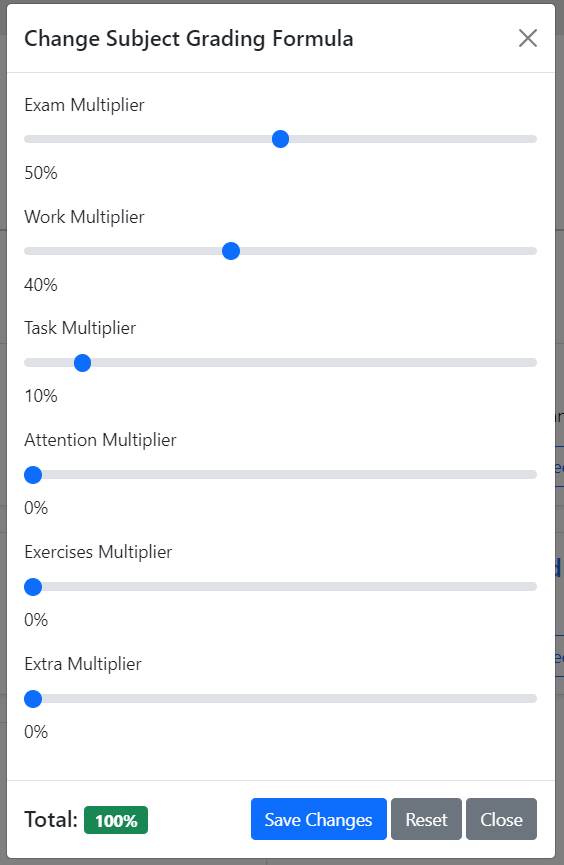
Освен възможността да прегледа всеки предмет, преподавателят може също да променя формулата, с която се изчислява успехът по дисциплината.



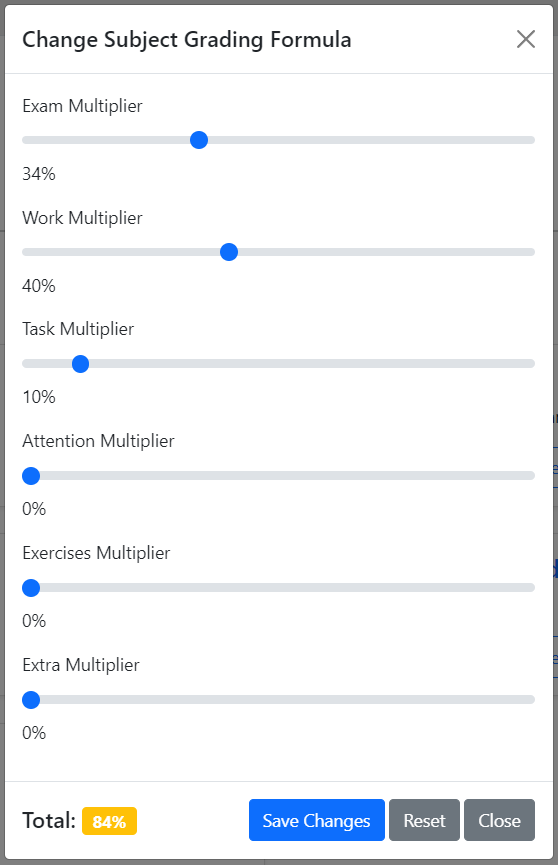
Фигура 6.9 – Страница Teacher/BrowseSubjects

При избор на Change Subject Formula се зарежда модално меню, в което се показва текущата формула, ако има такава. И потребителят има пълен достъп да променя процентите по пресмятането на крайната оценка.

В процесът по промяната на процентите има няколко възможни ситуации:

* Когато общият резултат е 100%, формулата може да се запази успешно в базата данни – пример за това е Фигура 6.10.
* Ако формулата надхвърля 100%, то тя се води за неуспешна и не може да бъде запазена. Пример за това е Фигура 6.11.
* Последен вариант, ако формулата е под 100%, то тя се води за незавършена и не може да се запази в приложението. Пример за това е Фигура 6.12

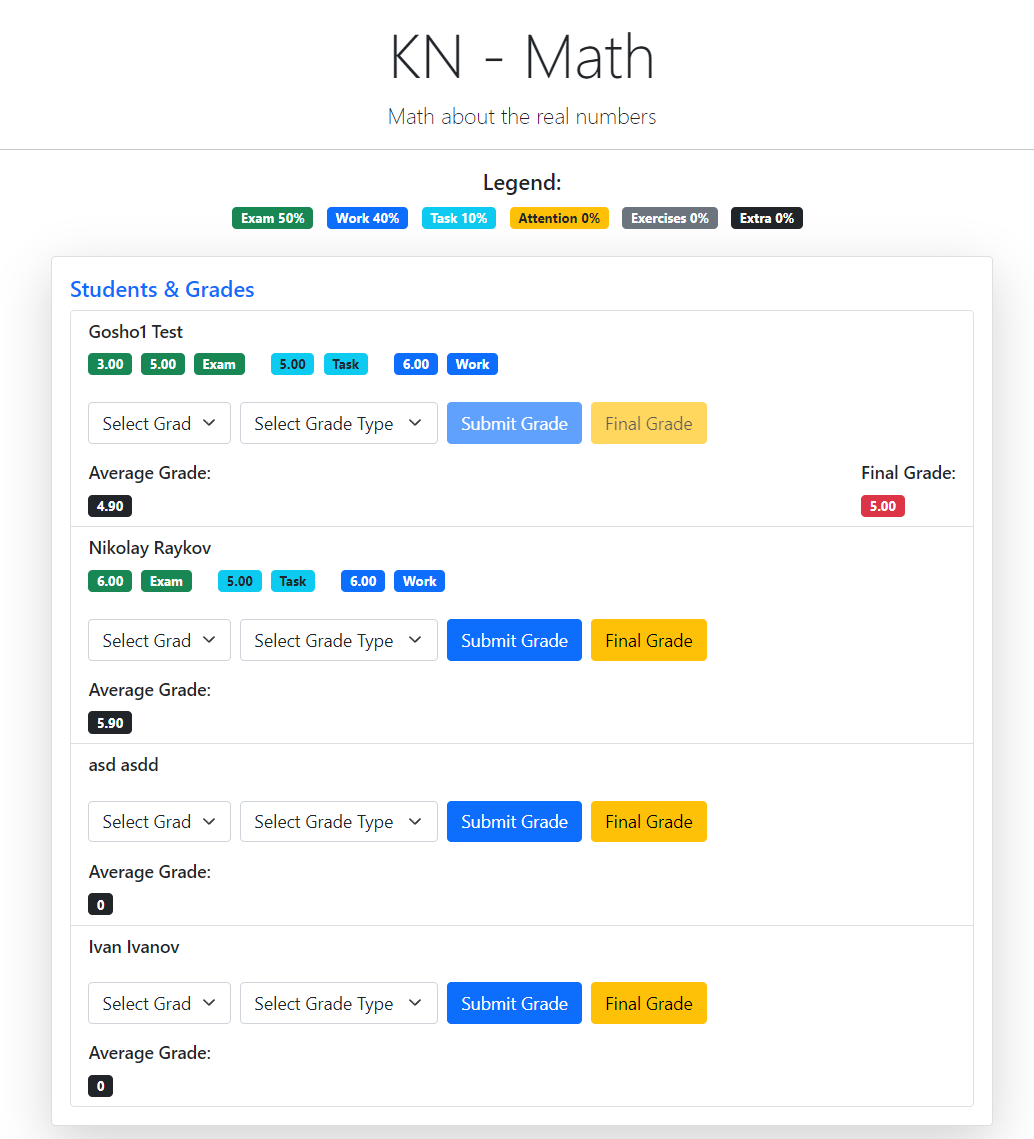
*Фигура 6.10 – Успешна формула Фигура 6.11 – Не успешна формула*

**

*Фигура 6.12 – Недовършена формула*

При избор на View Details от Фигура 6.9 за някой от предметите, преподавателят има възможността да разгледа подробните данни за този предмет и студентите от специалността. В новия изглед се предлага възможност за нанасяне на оценки и оформянето на крайната оценка.

Пример за това е Фигура 6.13, където е показан примерен дневник на студентите и техните оценки.



*Фигура 6.13 – CourseDetails – Студенти изучаващи предмет от дадена специалност*

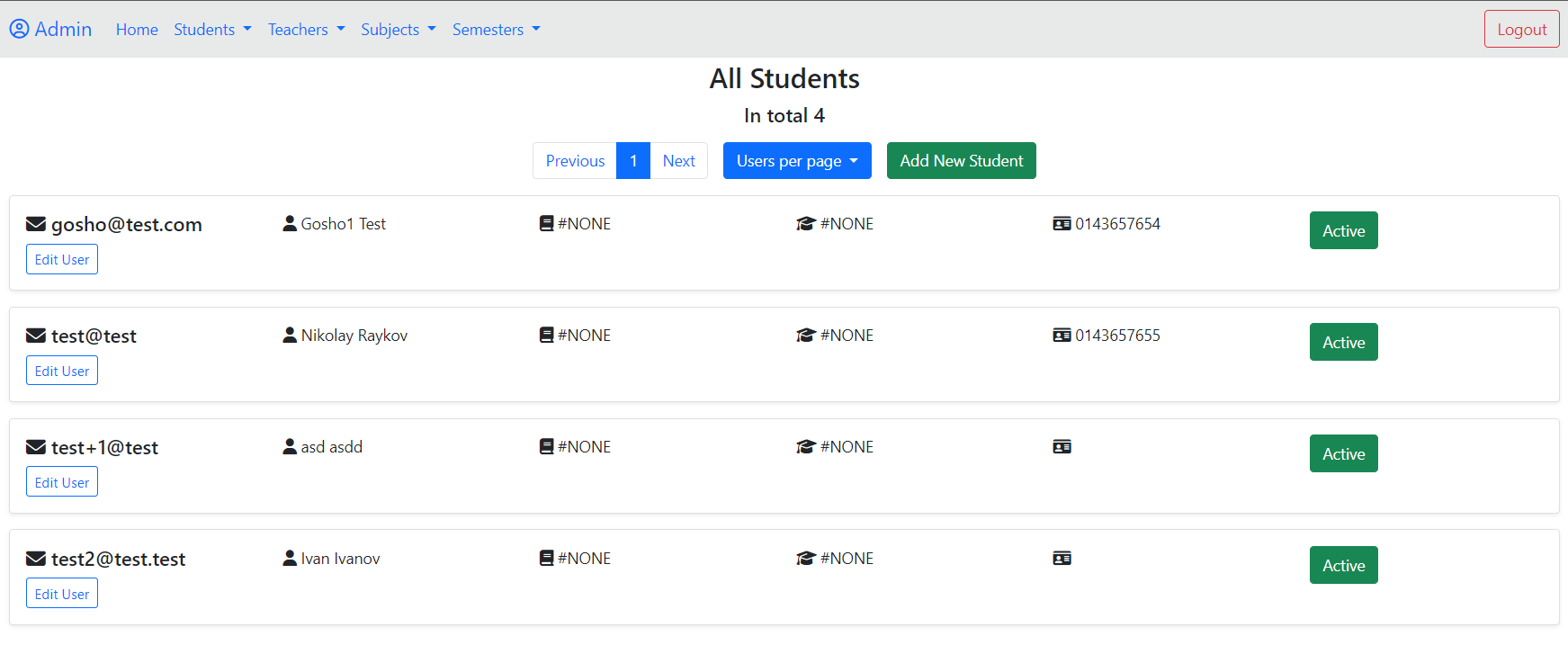
Преподавателят лесно може да избере дадана оценка и да я нанесе стига студентът да няма нанесена крайна оценка.

### **6.5 Употреба на приложението като администратор**

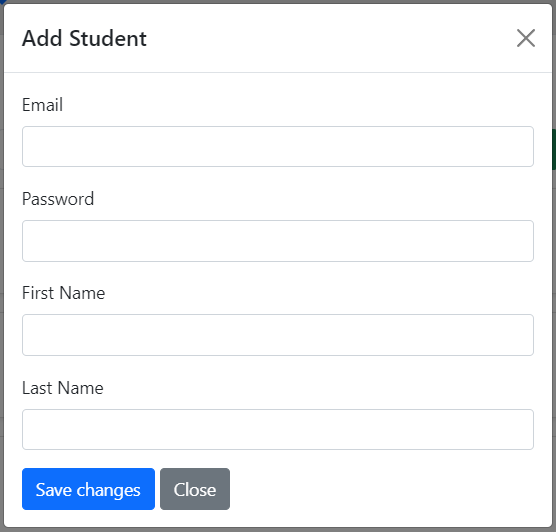
При успешно вписване в системата, администраторът ще бъде приветстван от началната страница Admin/Index. Тук са представени различни линкове за бърз достъп до различни страници.

Първо ще разгледаме страницата с всички студенти на приложението. Фигура 6.14 показва тази страница, където лесно може да се прегледа всеки студент, както и да се добави нов при нужда. Страницата също така поддържа редактиране на вече съществуващи студенти.

Добавянето и редактирането се извършват чрез модална форма на същата страница. Пример за това е показан на Фигура 6.15.

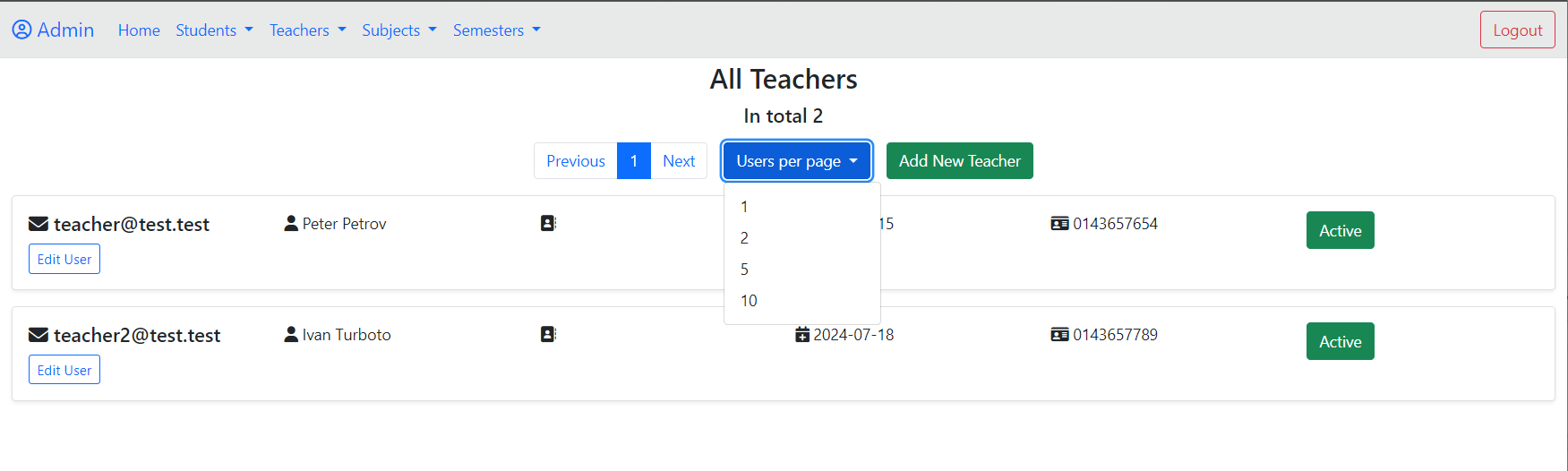


*Фигура 6.14 – Страница Admin/BrowseStudents*



*Фигура 6.15 – Модал форма за добавяне на нов студент*

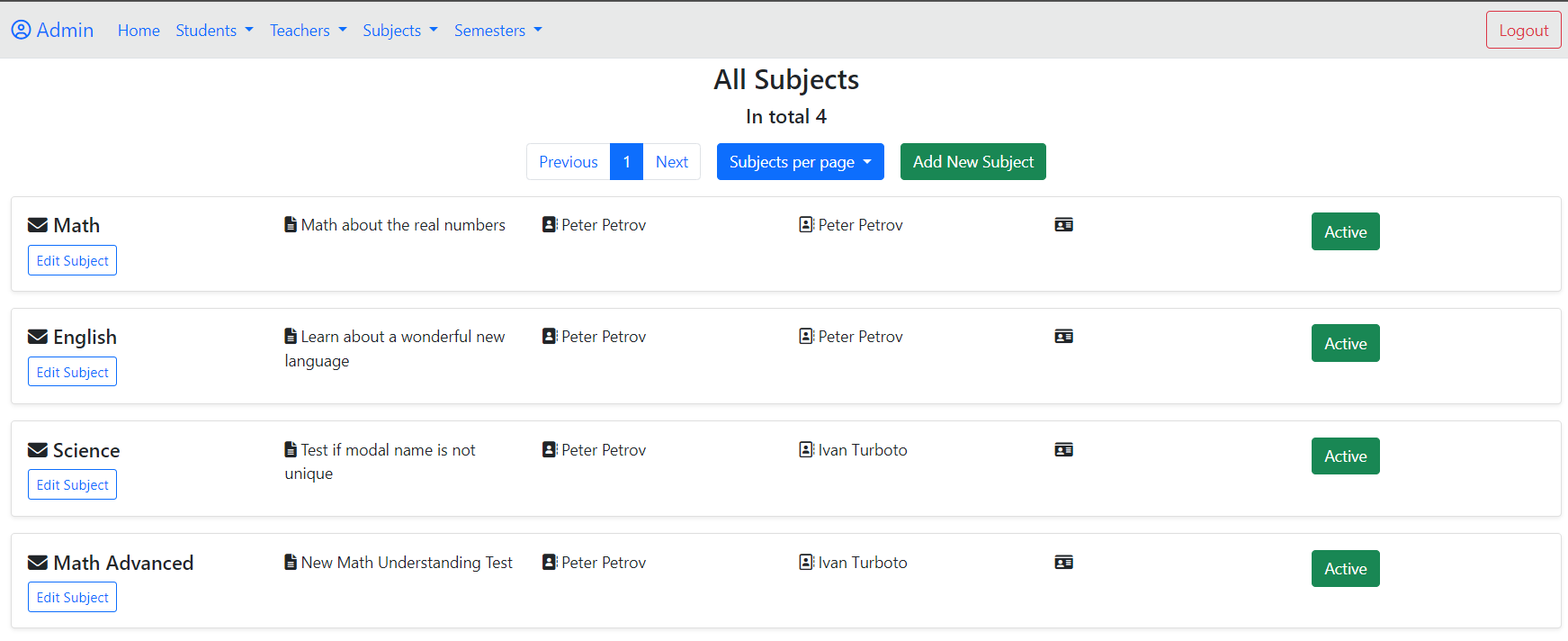
Друга страница, достъпна за администраторите, е тази с всички преподаватели. Тя се намира на **Admin/BrowseTeachers**. Тук се предлага идентична функционалност като при страницата за студентите – добавяне, редактиране и преглеждане на преподаватели.



*Фигура 6.16 – Страница Admin/BrowseTeachers*

Следващата страница е предназначена за показване на учебните предмети. Тук могат да се добавят, обновяват и преглеждат предмети. Страницата е подобна на предишните две.

По-специфичното е, че всеки предмет трябва да има преподавател, а в случай че е само един, той се добавя и като заместник. В бъдеще ще бъде добавена поддръжка за уникална икона за всеки предмет, която преподавателите ще могат да променят. В момента текущата икона е просто запълнено писмо.

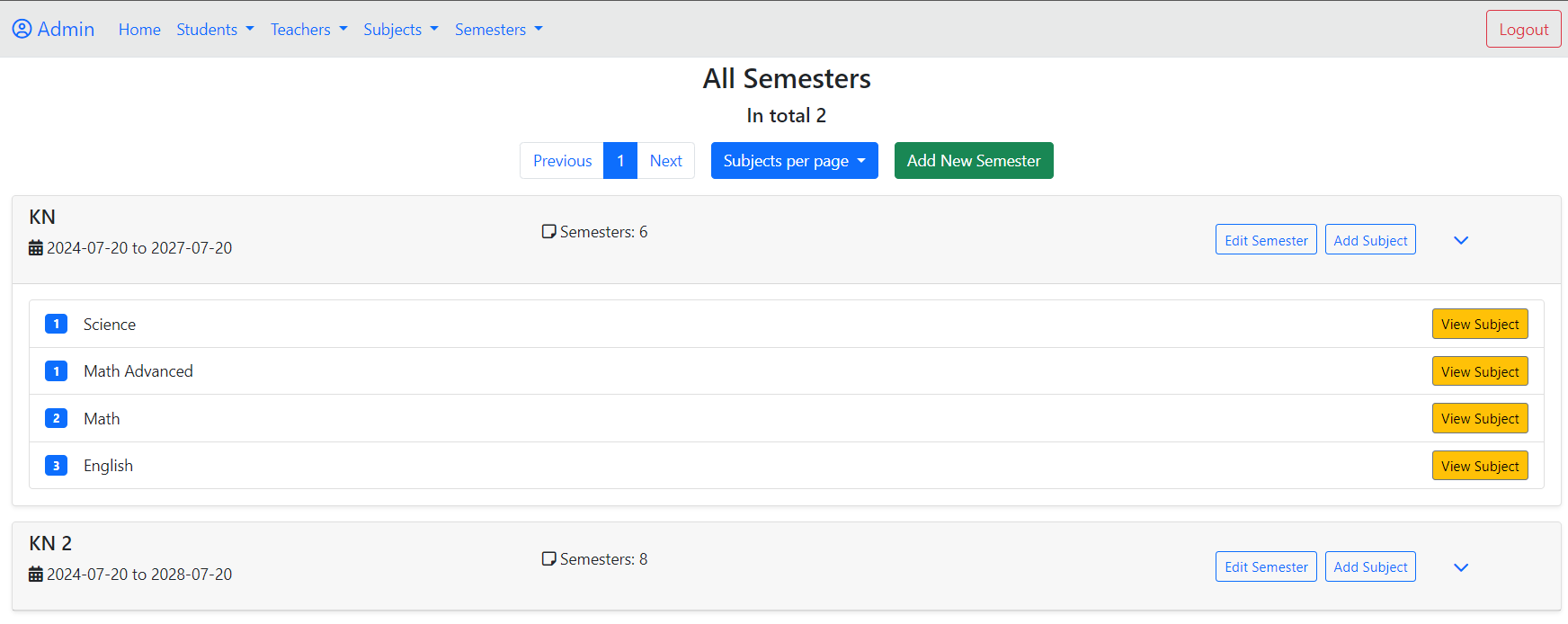


*Фигура 6.17 – Странциа Admin/BrowseSubjects*

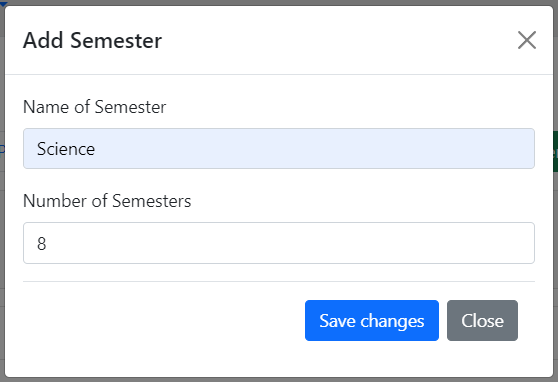
Последната страница, достъпна за администраторите на този етап, е страницата за всички специалности. Тук може да се види всяка специалност и нейният учебен план. Фигура 6.18 показва тази страница.

Дисциплините са подредени според семестъра, в който се изучват. Поддържа се промяна на данните за специалността. Добавянето на нова специалност е показано на Фигура 6.19, както и добавянето на нов предмет към дадена специалност чрез бутона Add Subject. Пример за тази функционалност е показан на Фигура 6.20.

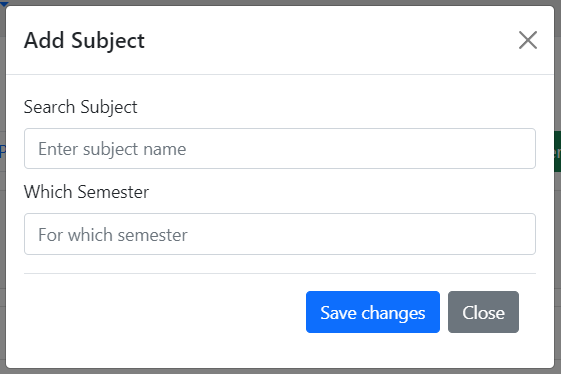
Няколко важни стъпки: специалностите трябва да имат четен брой семестри, в противен случай не могат да бъдат запазени в системата. Действително, не е възможно да има учебна година, състояща се от само един семестър.



*Фигура 6.18 – Страница Admin/BrowseSemesters*

**

*Фигура 6.19 – Модална форма за добавяне на нова Специалност*

**

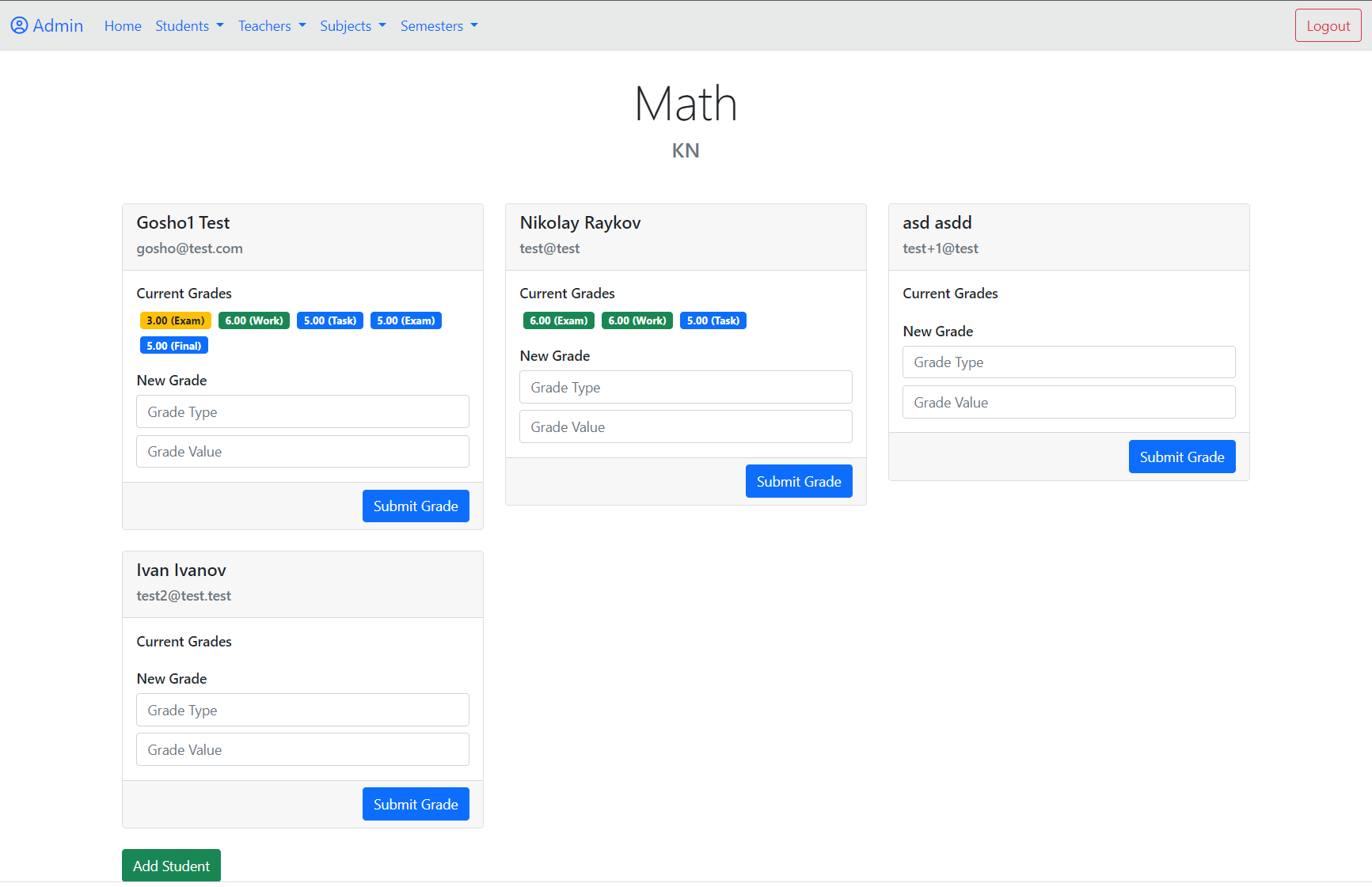
*Фигура 6.20 - Модално меню за добавяне на нов предмет*

При избор на бутон View Subject от Фигура 6.18 се отваря отделна страница, къде се показват всички студенти и техният текущ успех. В момента не се поддържа изтриване на оценките, но вбъдеще ще бъде достъпно чрез натискане на желаната оценка.

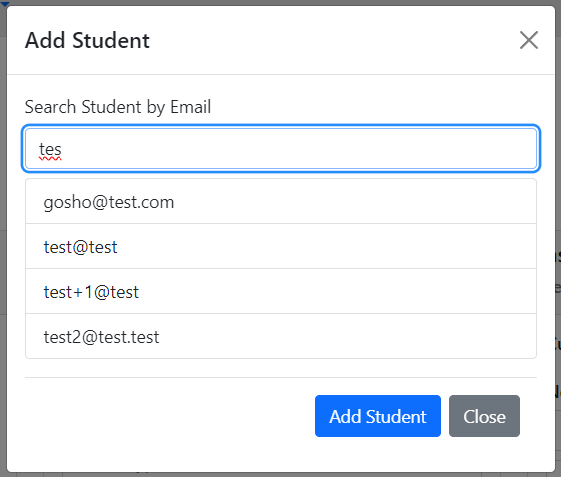
Тук освен да се разглеждат оценките на различните студенти, могат и да се добавят нови, при случай че липсват или е настъпила грешка при преподавателя.

Тази страница Admin/BrowseGrades може да се види на Фигура 6.21, друга важна функционалност е добавянето на нов студент да изучава дадена специалност. Това се извършва от бутон Add Student. И може да се прави по всяко време.

Пример за това може да се разгледа на Фигура 6.22. Тук чрез ajax заявка се предлагат студенти отговарящи на текстовият критерии. При намиране на правлният нов студент, администратора трябва единствено да натисне потребителя, който иска да добави.



*Фигура 6.21 – Всички студенти от дадена специалност и техният успех по даден предмет*



Фигура 6.22 – Добавяне на нов студент, чрез търсене по имейл

# **7. Основни резултати, изводи и препоръки**

Основният резултат е създаването на приложение, пригодено както за настолни компютри, така и за мобилни устройства, което позволява съхранението на учебни данни – било то за потребители или учебни предмети. Това е постигнато чрез използването на модерни технологии и утвърдени принципи в разработката.

Проектът улеснява модификацията на данните и ефективното разрешаване на проблемите, свързани с оценяването на студентите, като предлага ясен и лесен модел за изчисляване на примерни крайни оценки.

Системата позволява редактирането на множество данни, без риск от загуба на предишна информация, като същевременно гарантира гъвкавост и устойчивост при бъдещи актуализации. Приложението е проектирано така, че да бъде лесно за поддръжка и надграждане, което го прави подходящо за продължително използване и разширяване в образователната сфера.

# **8. Използвана литература**

|  |  |
| --- | --- |
| [1] | Cope, „tgw-calculator,“ [Онлайн]. Available: https://copeseeds.co.uk/services/tgw-calculator/. [Отваряно на 17 3 2025]. |
| [2] | Р. digital, „planove,“ Реколта digital, [Онлайн]. Available: https://digital.rekolta.bg/planove/. [Отваряно на 17 3 2025]. |
| [3] | Agrii, „seed-rate-calculator,“ Agrii, [Онлайн]. Available: https://www.agrii.co.uk/calculators/seed-rate-calculator/. [Отваряно на 17 3 2025]. |

# **9. Таблица с фигури**

[Фигура 1 Copeseeds 6](#_Toc193137641)

[Фигура 2 Реколта digital 7](#_Toc193137642)

[Фигура 3 Agrii 8](#_Toc193137643)

[1] Какво представлява Школо, URL: <https://bg.wikipedia.org/wiki/%D0%A8%D0%BA%D0%BE%D0%BB%D0%BE>, посетен на 02.02.2024г.

### [2] AdminPlus, URL: https://adminplus.bg/, посетен на 07.02.2024г.

### [3] Administrator's Plus, URL: <https://www.rediker.com/>, посетен на 07.02.2024г.

### [4] ASP.NET Core, URL https://dotnet.microsoft.com/en-us/apps/aspnet, посетен на 10.02.2024г.

### [5] Microsoft SQL Server, URL: https://www.microsoft.com/en-us/sql-server/sql-server-downloads, посетен на 10.06.2024г.

### [6] Entity Framework Documentation Hub, URL: https://learn.microsoft.com/en-us/ef, посетен на 10.06.2024г.

### [7] Какво представлява унифицирания език за моделиране (UML), URL: <https://www.microsoft.com/bg-bg/microsoft-365/visio/uml>, посетен на 20.07.2024г.