



Природо-математическа гимназия „Васил Друмев“

гр. Велико Търново, ул. „Вела Благоева“

ДИПЛОМЕН ПРОЕКТ

За придобиване на трета
степен на професионална
квалификация

Тема: Изграждане на комплекс от методи, средства и
подходи при разработване на приложение:

**“СИСТЕМА ЗА ОРГАНИЗИРАНЕ НА УЧЕНИЧЕСКИ
СЪСТЕЗАНИЯ”**

Ученик:

Николай Миленов Симеонов

Ръководител:

Мариела Маринова

Професия: „Приложен програмист“

Специалност: „Приложно програмиране“

Велико Търново
2024г.

Съдържание

1. Увод	2
2. Използвани технологии	3
Програмни езици	3
Framework-ове	10
Конвенции	12
Библиотеки	16
Среди за разработка и контрол на версиите	18
3. Архитектура на Базата Данни	20
Таблицы	20
Връзки	24
Диаграма на базата	28
4. Представяне и обясняване на уеб приложението	29
5. Как да използваме	42
6. Заключение	48

1. Увод

Предмет на дипломната работа е разработване на приложение, целящо да улесни работата на учителите и организаторите на учебни състезания и олимпиади. Системата се ползва от учители и ученици, като учителите създават и организират състезания, а учениците могат да ги преглеждат филтрират по предмет, дата и други фактори и да се записват за тях. Системата има администратор, който управлява потребителите. Функционалностите включват: онагледяване на информацията за отделните състезания, водене на статистика, разделение за състезанията и потребителите по определени критерии и пълен контрол над организацията на учебни състезания.

Уеб системата поддържа три потребителски роли:

- **Администратор-** Одобрява акаунтите на учители и ученици. Има възможност да преглежда, променя, филтрира и изтрива вече регистрираните потребители, като преди необратими промени бива предупреден, че действието е необратимо. Нерегистрираните потребители може да бъдат филтрирани, одобрявани и отхвърляни. Той има правото да създава, изтрива, променя състезания както и да вади статистика за учениците записани за дадено състезание.
- **Учител-** Учителят може да вижда основния прозорец на приложението, в който се визуализират всички активни състезания. Той има право да го преглежда, променя, деактивира, изтрива и да вади статистика за записаните ученици.
- **Ученик-** Ученикът може да вижда основния прозорец на приложението, в който се визуализират всички активни състезания. Той има право да се записва за тях, като при натискане на бутона записване, състезанието изчезва от основната страница и се премества в страницата „Моите състезания“.

Целта на дипломната работа е показване на знания свързани със изработването на функционално уеб приложение, което включва неговата архитектура, логика и потребителски интерфейс.

2. Използвани технологии

Програмни езици

- **C#:**

C# е обектно-ориентиран програмен език, разработен от Microsoft. Той е изключително популярен за създаване на софтуерни приложения за различни платформи като десктоп, уеб и мобилни устройства. Ето някои от предимствата и ползите на C#:

- **Обектно-ориентиран подход:** C# поддържа основните концепции на обектно-ориентираното програмиране (ООП), като инкапсулация, наследяване и полиморфизъм. Това прави кода по-структуриран, поддържаем и разширяем.
- **Мощна платформа за разработка:** C# е част от платформата .NET на Microsoft, която предлага богат набор от инструменти и библиотеки за разработка на различни видове приложения.
- **Типобезопасност:** C# е статично типизиран език, което означава, че типовете на данните се проверяват по време на компилацията. Това помага за намаляване на грешките в кода и улеснява поддръжката му.
- **Интеграция с други технологии:** C# се интегрира добре с различни технологии на Microsoft, като **SQL Server, Azure, Windows Forms, WPF, ASP.NET** и други. Това позволява лесна разработка на приложения, които се взаимодействат с различни системи и услуги.
- **Широка общност и ресурси за обучение:** Поради популярността си, C# разполага с голяма общност от разработчици и обширен набор от уроци, ръководства и ресурси за обучение. Това прави изучаването и развиването на умения в C# достъпно и удобно.
- **Крос-платформеност:** С развитието на **.NET Core** и по-късно на **.NET 5** и **.NET 6**, C# става все по-крос-платформен език, който може да се използва за разработка на приложения за различни операционни системи като **Windows, Linux** и **macOS**.
- **Висока производителност:** C# е известен с високата си производителност, която го прави подходящ за разработка на приложения с високи изисквания към бързодействие и ефективно използване на ресурсите на системата.

Всички тези фактори правят C# изключително полезен език за разработка на различни видове софтуерни приложения, което го прави предпочитан избор за много програмисти и организации.

- **HTML 5:**

HTML е съкращение от "**HyperText Markup Language**" (Хипертекстов език за означаване), който е стандартен език за описване и структуриране на уеб страници.

HTML използва различни елементи, наречени тагове, за да определи структурата и съдържанието на уеб документите.

Основната цел на **HTML** е да позволи на уеб разработчиците да създават документи, които могат да бъдат изобразени в уеб браузъри. Тези документи могат да включват текст, изображения, връзки към други уеб страници, мултимедийни елементи и форми за въвеждане на данни от потребителя. **HTML** се използва за създаване на различни видове уеб съдържание, включително:

- **Уеб страници:** **HTML** се използва за създаване на уеб страници, които могат да съдържат текст, изображения, линкове, таблици и други елементи.
- **Уеб приложения:** **HTML** се използва за създаване на интерактивни уеб приложения, които могат да предлагат различни функционалности като форми за въвеждане на данни, анимации, мултимедия и други.
- **Електронни бюлетини и имейл шаблони:** **HTML** се използва за създаване на електронни бюлетини и имейл шаблони, които могат да се изпращат на потребителите по електронна поща.
- **Мобилни приложения:** **HTML** се използва за създаване на мобилни приложения с помощта на технологии като **Apache Cordova** или **PhoneGap**.

HTML работи заедно с други технологии като **CSS (Cascading Style Sheets)** за стилизиране на уеб страници и **JavaScript** за добавяне на интерактивност и динамично съдържание.

Съвременните уеб страници и приложения обикновено използват комбинация от **HTML**, **CSS** и **JavaScript** за да предложат богато и атрактивно потребителско изживяване.

HTML 5 е петата версия на стандарта за маркиране на хипертекстови документи (**HTML**), който се използва за създаване на уеб страници.

HTML 5 включва множество нови функции и подобрения спрямо предишните версии, които го правят по-мощен и гъвкав инструмент за уеб разработка. Ето някои от основните тагове и атрибути в **HTML 5**:

- **<html>**: Определя началото на **HTML** документа и съдържа целия **HTML** код.
- **<head>**: Съдържа метаданни за документа, като заглавие, метатагове, връзки към външни файлове за стилове и скриптове и други.
- **<title>**: Задава заглавието на документа, което се показва в заглавната част на прозореца на браузъра или във вкладката на страницата.
- **<body>**: Съдържа съдържанието на уеб страницата, като текст, изображения, линкове и други елементи.
- **<div>**: Използва се за създаване на контейнери за групиране на други елементи и за прилагане на стилове чрез **CSS**.
- ****: Подобно на **<div>**, но обикновено се използва за групиране на текстови елементи.
- **<p>**: Определя параграф, който съдържа текст.

- **<a>**: Създава хипервръзка към други уеб страници, документи, изображения и други ресурси.
- ****: Вмъква изображение в документа.
- **** и ****: Създават неномериран и номериран списък, съответно. ****: Определя елемент в списък.
- **<table>**, **<tr>**, **<td>**: Създават таблица, ред и клетка в таблица.
- **<form>**: Създава формуляр за въвеждане на данни от потребителя.
- **<input>**: Използва се за създаване на поле за въвеждане на данни във формуляр.
- **<textarea>**: Създава текстово поле с множество редове във формуляр. Това са само някои от основните тагове и атрибути в **HTML 5**. В **HTML 5** има още много други тагове и възможности за създаване на богато и интерактивно съдържание за уеб.

- **CSS:**

CSS е съкращение от "**Cascading Style Sheets**", което е език за стилизиране и оформление на уеб страници. **CSS** позволява на уеб разработчиците да контролират външния вид на уеб страници, като определят цветове, шрифтове, размери, позициониране и други стилови елементи.

Основната цел на **CSS** е да раздели съдържанието от представянето му, като позволява на разработчиците да променят външния вид на множество елементи на уеб страницата едновременно, като променят само една линия код.

CSS се използва за следните цели:

- **Стилизиране на текст**: **CSS** позволява да се променят различни аспекти на текста като шрифт, размер, цвят, подчертаване и т.н.
- **Стилизиране на елементи на страницата**: **CSS** позволява да се променят визуалните атрибути на различни елементи на уеб страницата като фонове цветове, граница, размери, позиции и други.
- **Работа със специфични устройства**: **CSS** позволява да се създадат различни стилове, които да се прилагат в зависимост от устройството, на което се визуализира уеб страницата, като например мобилни устройства, таблети или десктоп компютри.
- **Анимации и преходи**: **CSS** позволява да се създават анимации и преходи, които да подобрят визуалното изживяване на потребителите, като например анимирани менюта, промени на цветове и други.
- **Поддръжка на медийни елементи**: **CSS** позволява да се стилизират медийни елементи като изображения, видео и аудио файлове, като се контролират техните размери, позиции и други аспекти.

CSS обикновено се използва заедно с **HTML** и **JavaScript**, за да се създаде уеб страница със структура, стил и интерактивност. Стандартите на **CSS** се поддържат от всички модерни уеб браузъри, което гарантира, че стиловете, които се дефинират с

помощта на **CSS**, ще бъдат правилно интерпретирани и показани на различните устройства и браузъри.

- **href**: Атрибут за `<a>`, който определя адреса на дестинацията на хипервръзката.
- **src**: Атрибут за ``, който указва пътя към изображението.
- **alt**: Алтернативен текст за ``, който се показва, ако изображението не може да бъде заредено.
- **id**: Уникален идентификатор за елемента, което е полезно за стилизиране или манипулиране с **JavaScript**.
- **class**: Клас на елемента, който се използва за групиране на елементи със сходни стилове или за прилагане на JavaScript функционалност.
- **style**: Атрибут, който позволява директно дефиниране на **CSS** стилове за даден елемент.
- **title**: Показва заглавието на елемента, което се показва при *ховър* или при фокусиране върху него.

Това са някои от основните **HTML** тагове и атрибути, които се използват за създаване на уеб страници. С тях можете да създадете разнообразни и интересни уеб преживявания за вашите потребители.

- **JavaScript:**

Предимствата и ползите на **JavaScript**:

- **Интерактивност на уеб страниците**: **JavaScript** позволява създаването на интерактивни елементи и приложения в уеб страниците, като например форми за въвеждане на данни, анимации, интерактивни менюта и други. Това прави уеб страниците по-привлекателни и полезни за потребителите.
- **Клиентска страна**: **JavaScript** се изпълнява в брауъра на потребителя, което го прави идеален за разработка на функционалности, които се изискват в реално време, без да се налага презареждане на страницата. Това включва валидация на форми, динамично обновяване на съдържанието и други.
- **Широка поддръжка**: **JavaScript** е поддържан от всички модерни уеб браузъри, което означава, че кодът, който създадете, ще работи в различни браузъри и операционни системи без проблеми.
- **Лесно изучаване и употреба**: **JavaScript** е език с прост синтаксис, който е лесен за научаване за начинаещи и за разработчици, които идват от други програмни езици.
- **Богат набор от библиотеки и фреймуърк-ове**: Съществуват множество библиотеки и фреймуърки като **jQuery**, **React.js**, **Angular.js**, **Vue.js** и други, които улесняват разработката на сложни уеб приложения и повишават производителността на разработчиците.
- **Разширяемост**: **JavaScript** позволява внедряване на допълнителна функционалност чрез използване на външни библиотеки и плъгини, които са лесни за интегриране във вашите проекти.
- **Поддръжка на мобилни устройства**: **JavaScript** се използва за разработка на мобилни приложения с помощта на фреймуърк-ове като **React Native** или

ionic, което позволява създаването на крос-платформени приложения, които работят на различни мобилни устройства.

Тези са само някои от предимствата и ползите на **JavaScript**, които го правят един от най-широко използваните езици за уеб разработка в момента.

Основните понятия, конструкции и методи в **JavaScript**:

Понятия:

- **Променливи:** Контейнери за съхранение на данни.
- **Типове данни:** Включват числа, стрингове, булеви стойности, масиви, обекти и други.
- **Функции:** Блокове код, които могат да се изпълняват при нужда, използвайки параметри и връщайки стойност.
- **Условни оператори:** Като if, else if и else, които позволяват на програмата да взема решения в зависимост от определени условия.
- **Цикли:** Като for, while и do-while, които позволяват на програмата да изпълнява определени действия повторно.
- **Масиви:** Структури от данни, които могат да съдържат набор от елементи, индексирани с числови стойности.
- **Обекти:** Съдържат ключове и стойности, които позволяват организиране на данните по различни категории.
- **Събития:** Действия, които могат да бъдат предизвикани от потребителя или от системата, като кликуване на мишката или зареждане на страницата.

Конструкции и методи:

- **console.log():** Метод за извеждане на информация в конзолата на браузъра, което е полезно за дебъгване и проверка на данни.
- **document.querySelector()** и **document.querySelectorAll():** Методи за избор на елементи от HTML документа по CSS селектори.
- **addEventListener():** Метод за добавяне на събития към HTML елементи, като кликуване, навлизане на мишката и други.
- **Array.push():** Метод за добавяне на нови елементи към края на масива.
- **Array.pop():** Метод за премахване на последния елемент от масива.
- **String.length:** Свойство, което връща дължината на низа.
- **Math.random():** Метод за генериране на случайно число между 0 и 1.
- **setTimeout()** и **setInterval():** Функции, които позволяват на програмата да изчака определено време преди да изпълни определени действия или да повтаря действията периодично.

Това са някои от основните понятия, конструкции и методи в JavaScript, които са от съществено значение за разработката на уеб приложения и интерактивни уеб страници.

- **SQL:**

Ползите и предимствата на **SQL**:

- **Лесен за използване:** **SQL** е декларативен език за заявки, който е лесен за научаване и разбиране дори за хора без техническо образование.
- **Мощен и гъвкав:** **SQL** позволява на потребителите да изпълняват сложни заявки за данни, включително извличане, филтриране, сортиране, групиране и обединяване на данни от различни източници.
- **Скорост и ефективност:** **SQL** е оптимизиран за бърза обработка на големи обеми от данни, като осигурява ефективно търсене и сортиране на информацията.
- **Скалируемост:** **SQL** базите данни могат да се мащабират вертикално и хоризонтално, което позволява увеличаване на капацитета и производителността им спрямо нуждите на приложението.
- **Сигурност:** **SQL** предлага механизми за сигурност и управление на достъпа до данните, като ролево базиран достъп и криптиране на данните, което гарантира защитата на чувствителната информация.
- **Стандартизиран:** **SQL** е стандартизиран език, който е признат и поддържан от множество вендори и организации, което гарантира съвместимост и преносимост на данните между различни системи и платформи.
- **Разнообразие от инструменти и ресурси:** Съществуват множество инструменти и ресурси за работа с **SQL**, включително мениджмънт системи за бази данни (**DBMS**) като **MySQL**, **PostgreSQL**, **Microsoft SQL Server** и **Oracle**, както и богата документация, онлайн курсове и форуми за поддръжка и обучение.
- **Поддръжка на транзакции:** **SQL** поддържа транзакции, което позволява на потребителите да извършват група от операции като една атомарна и неделима операция, гарантираща цялостността на данните.

Тези са някои от основните ползи и предимства на **SQL**, които го правят незаменим инструмент за управление и манипулиране на данни в различни бизнес среди.

Основни понятия:

- База данни - Съхранява структурирани данни в таблици.
- Таблица - Съдържа редове и колони, които организират данните по категории.
- Колона - Съдържа специфичен вид данни за всеки ред в таблицата.
- Ред - Съдържа конкретните данни за всеки запис в таблицата.
- Ключ - Уникален идентификатор за реда или комбинация от колони, който позволява бързо търсене и сортиране на данните.

Основни **SQL** команди:

- **SELECT** - Избира данни от таблица или комбинация от таблици.
- **INSERT INTO** - Добавя нов ред в таблицата със специфични стойности за всяка колона.
- **UPDATE** - Променя данни във вече съществуващи редове.
- **DELETE FROM** - Премахва редове от таблицата.

- **CREATE TABLE** - Създава нова таблица в базата данни със специфични колони и типове данни.
- **ALTER TABLE** - Променя структурата на съществуваща таблица, като добавя, променя или премахва колони.
- **DROP TABLE** - Премахва цялата таблица от базата данни.
- **JOIN** - Обединява данни от две или повече таблиците в един резултатен набор, базиран на общи стойности в определени колони.
- **WHERE** - Филтрира резултатите на **SELECT** заявката базирано на дадени критерии.
- **GROUP BY** - Групира резултатите от **SELECT** заявката базирано на стойности в определени колони.
- **ORDER BY** - Сортира резултатите от **SELECT** заявката във възходящ или низходящ ред базирано на стойности в определени колони.
- **HAVING** - Прилага условия върху резултатите от **GROUP BY** операцията.

Това са някои от основните понятия, команди и конструкции в **SQL**, които се използват за създаване и управление на релационни бази данни. **SQL** е език с широко приложение в областта на базите данни и е важен инструмент за разработчиците и администраторите на данни.

- **Razor:**

Razor е синтаксис за шаблони, който се използва в рамките на **ASP.NET** за създаване на динамични уеб страници. Ето някои от неговите ползи и същност:

- ***Прост за използване:*** **Razor** е лесен за научаване и използване, тъй като синтаксисът му е подобен на стандартния **HTML** и включва минимални промени.
- ***Интегриран в ASP.NET:*** **Razor** е интегриран в **ASP.NET**, което го прави част от екосистемата на платформата и улеснява работата с него за разработчиците, които вече използват **ASP.NET**.
- ***Динамични уеб страници:*** **Razor** позволява вписването на код на **C#** или **VB.NET** директно в **HTML** файловете, което дава възможност за генериране на динамично съдържание в уеб приложенията.
- ***Интерполация на данни:*** С помощта на **Razor** можем да вмъкваме данни от моделите или други източници директно в **HTML** шаблоните, което улеснява визуализацията на данните за потребителите.
- ***Лесна интеграция с кода на C# или VB.NET:*** **Razor** позволява използването на цялата функционалност на **C#** или **VB.NET**, включително операции, логика и извикване на методи, което прави генерирането на динамично съдържание много по-лесно и удобно.
- ***Четим и лесен за разбиране синтаксис:*** Синтаксисът на **Razor** е четим и лесен за разбиране, което улеснява сътрудничеството между разработчиците и прави кода по-прозрачен и поддържаем.

Същността на **Razor** е възможността да създаваме динамични уеб страници, които да показват персонализирано съдържание за потребителите, използвайки лесен и удобен синтаксис за вписване на код и данни в **HTML** шаблоните.

Framework-ове

- **ASP.Net:**

ASP.NET платформа, разработена от Microsoft, която позволява на програмистите да създават уеб сайтове, уеб услуги и уеб приложения. Тя предоставя мощни инструменти и функционалности за създаване на динамично генерирани уеб страници, използвайки .NET програмни езици като C# или VB.NET.

Ето някои от основните причини за използването на **ASP.NET**:

- **Бързо разработване на уеб приложения:** ASP.NET предлага множество готови компоненти, контролери и шаблони, които ускоряват процеса на разработка на уеб приложенията.
- **Мощен и сигурен:** Платформата включва вградена система за сигурност, управление на сесии, автентикация и авторизация, което прави уеб приложенията, разработени с ASP.NET, мощни и сигурни.
- **Интеграция с други технологии на Microsoft:** ASP.NET се интегрира лесно с други технологии на Microsoft, като SQL Server, Azure, Windows Server и други, което улеснява разработката и управлението на уеб приложенията.
- **Мащабируемост и производителност:** ASP.NET е мащабируема платформа, която може да се справи с големи обеми от трафик и данни. Тя предлага оптимизирани решения за увеличаване на производителността и ефективността на уеб приложенията.
- **Поддръжка и общност:** ASP.NET е широко използвана технология с голяма общност от разработчици, което означава, че има множество ресурси, форуми и обучения за поддръжка и учене.
- **Крос-платформена поддръжка:** С появата на .NET Core, ASP.NET стана крос-платформен, което означава, че уеб приложенията могат да бъдат разработвани и изпълнявани на различни операционни системи.

В общи линии, **ASP.NET** е мощна, сигурна и мащабируема платформа за създаване на уеб приложения, която предоставя инструменти и ресурси за бързо развитие и управление на уеб проекти.

- **Identity Framework:**

ASP.NET Identity е *фреймуърк* за управление на потребителска идентичност в уеб приложенията, разработена от Microsoft. Тя предоставя инструменти за регистрация, управление на потребителите, удостоверяване (автентикация) и управление на правата (авторизация).

Ето някои от основните причини за използването на **ASP.NET Identity**:

- **Гъвкавост и персонализация:** **ASP.NET Identity** позволява на разработчиците да персонализират и адаптират системата за управление на потребителската идентичност според специфичните изисквания на техните уеб приложения.
- **Безопасност:** Идентичността на потребителите се управлява по сигурен начин, като се включват механизми за *хеширане* на пароли, защита от CSRF атаки, управление на сесии и други.
- **Интеграция с външни доставчици за идентичност:** **ASP.NET Identity** поддържа интеграция с външни доставчици за идентичност като **Google**, **Facebook**, **Twitter** и други, което позволява на потребителите да се автентифицират с техните съществуващи акаунти в социалните мрежи.
- **Лесна интеграция с ASP.NET приложения:** **ASP.NET Identity** е проектирана за лесна интеграция с уеб приложения, базирани на **ASP.NET MVC**, **ASP.NET Web Forms**, **ASP.NET Core** и други.
- **Поддръжка на различни типове автентикация:** Платформата поддържа различни методи за автентикация, включително паролна, двуфакторна, посредством SMS и други.
- **Готови за употреба функционалности:** **ASP.NET Identity** предоставя готови функционалности за регистрация, вход, изход, потвърждение на имейл, възстановяване на парола и други, които улесняват разработката на уеб приложения.

В общи линии, **ASP.NET Identity** е мощна и гъвкава рамка за управление на потребителската идентичност, която предоставя инструменти и функционалности за сигурно и удобно управление на потребителите в уеб приложенията.

• Entity Framework:

Entity Framework е фреймуърк разработен от **Microsoft**, който позволява лесна връзка между базата данни и уеб приложението. С помощта на **Entity Framework**, можем да работим с база данни чрез обекти на езика **C#**, като таблиците в базата данни се представят като класове, а записите в тях като обекти.

Entity Framework ни предоставя вградена библиотека (**EntityFramework.Schema**) за работа с атрибути, като например:

- **[Key]:** Обозначава основния ключ на таблицата.
- **[InverseProperty]:** Позволява задаване на връзки между полета в различни класове, когато има връзка "един към много" или "много към много".
- **[MaxLength]:** Задава максималната дължина на стойностите в дадено поле.
- **[MinLength]:** Задава минималната дължина на стойностите в дадено поле.
- **[Pattern]:** Задава шаблон за валидация на стойностите в дадено поле.

Entity Framework също така предлага мощни разширения като **EntityFramework.Diagnostics**, **EntityFramework.Tools** и други, които улесняват работата с базата данни чрез модели и миграции. Миграциите позволяват автоматично генериране на скриптове за промени в базата данни, което улеснява контрола над версиите и автоматизира процеса на актуализация на схемата на базата данни.

Като цяло, **Entity Framework** е мощен инструмент за работа с бази данни в уеб приложенията, който предоставя лесен и удобен начин за взаимодействие с базата данни, като използва обекти и модели на програмния език C#.

- **Bootstrap:**

Bootstrap е популярен *фронт енд фреймуърк*, разработен от **Twitter**, който предоставя готови компоненти, шаблони и стилове за бързо разработване на уеб приложения и уеб сайтове. Някои от основните ползи и причини за използването на **Bootstrap** са:

- **Бързо разработване:** **Bootstrap** предоставя готови компоненти и стилове, които ускоряват процеса на разработка на уеб приложенията и уеб сайтовете.
- **Мобилна първа подход:** **Bootstrap** е изцяло отзивчив и мобилен първенец, което означава, че уеб сайтовете, изградени с него, автоматично се адаптират и оптимизират за различни устройства и екрани.
- **Консистентен дизайн:** **Bootstrap** предоставя единно оформление и дизайн за уеб приложенията, което прави интерфейса по-привлекателен и професионален.
- **Множество компоненти:** **Bootstrap** включва голямо разнообразие от компоненти като бутони, форми, навигация, карусели, модални прозорци и други, които могат да се използват за бързо добавяне на функционалности в уеб приложенията.
- **Лесно персонализиране:** **Bootstrap** позволява на разработчиците да персонализират и променят стиловете и компонентите с помощта на собствени CSS правила и теми.
- **Поддръжка на браузъри:** **Bootstrap** е съвместим с всички основни браузъри, което гарантира, че уеб приложенията ще работят правилно за всички потребители.
- **Активна общност:** **Bootstrap** има голяма и активна общност от разработчици, което означава, че има множество ресурси, документация и обучения за поддръжка и учене.

Това са някои от основните ползи и причини за използването на **Bootstrap**, което го прави предпочитан избор за много разработчици при създаването на модерни и атрактивни уеб приложения и уеб сайтове.

Конвенции

- **MVC:**

MVC (Model-View-Controller) е архитектурен шаблон за създаване на софтуерни приложения, който разделя приложението на три основни компонента: Model, View и Controller.

В контекста на ASP.NET, моделът представлява бизнес обектите и данните, които приложението използва. Те могат да бъдат представени като класове, които съдържат логика и свойства за работа с данните.

Моделът може да бъде реализиран като класове на C# или VB.NET, които съдържат свойства, методи и валидация за данните.

Ползите от модела в ASP.NET включват разделянето на бизнес логиката от визуалната част на приложението, улесняване на тестването и повторното използване на моделите в различни части на приложението.

- **View (Изглед):**

- В ASP.NET, изгледът представлява **HTML**, **CSS** и **JavaScript** код, който дефинира визуалната част на уеб приложението. Той може да бъде представен чрез **Razor** страница или **HTML** файл.
- Изгледът показва данните от модела на потребителя и приема вход от потребителя чрез форми и контроли.
- Ползите от изгледа в ASP.NET включват лесно поддържане на визуалната част на приложението, разделение на отговорностите между разработчиците и дизайнерите и възможност за многократно използване на изгледите в различни части на приложението.

- **Controller (Контролер):**

- В ASP.NET, контролерът е клас, който приема заявките от потребителите, обработва ги и връща подходящ отговор. Той координира взаимодействието между модела и изгледа.
- Контролерът съдържа методи, които обработват заявките от потребителите и извършват съответните операции върху модела. Той също така избира подходящия изглед за показване на резултатите от операциите.
- Ползите от контролера в ASP.NET включват централизиране на логиката за управление на заявките, улесняване на тестването и възможност за повторно използване на контролерите в различни части на приложението.

- **Service Architecture:**

Service Architecture в контекста на ASP.NET представлява подход за организиране на приложението чрез разделяне на функционалността му на отделни услуги (**services**), които се отговарят за конкретни задачи или функции. Това е важна практика за създаване на поддържаеми, разширяеми и лесни за тестване уеб приложения. Ето какво представлява този подход и какви са неговите ползи и същност:

- **Разделяне на отговорностите:** **Service Architecture** помага за ясно разделяне на отговорностите между различните части на приложението. Всяка услуга е отговорна за конкретна функционалност или бизнес логика, което прави кода по-четим и поддържаем.

- **Разширяемост и гъвкавост:** Чрез **Service Architecture** е лесно да добавяте нови функционалности към приложението, като просто добавяте нови услуги или разширявате вече съществуващите. Това прави приложението по-гъвкаво и подходящо за бъдещи промени и изисквания.
- **Тестваемост:** Подходът на **Service Architecture** прави кода по-лесен за тестване, тъй като всяка услуга може да бъде тествана поотделно, без да е необходимо да се тества цялото приложение. Това води до по-добра качествена анализ и по-малко грешки в приложението.
- **Лесна поддръжка:** Чрез ясно разделението на функционалността, **Service Architecture** прави кода по-лесен за поддръжка и разширение. Разработчиците могат лесно да навигират в кода и да разберат какво прави всяка част от него.
- **Рециклиране на код:** Чрез **Service Architecture** е по-лесно да се рециклира кодът, тъй като функционалността, която е изолирана в услуги, може да бъде използвана повторно в различни части на приложението. Това води до по-малко дублиране на кода, по-малко грешки и по-лесно поддръжка на приложението.
- **Самостоятелност на услугите:** В **Service Architecture** всяка услуга е самостоятелна и не зависи от други части на приложението. Това означава, че всяка услуга може да бъде разработвана, тествана и поддръжана независимо от останалите части на приложението. Това прави кода по-модулен, по-лесен за разбиране и поддръжка.

Същността на **Service Architecture** е в създаването на чист и структуриран код, който е лесен за поддръжка и разширение. Това е важен аспект на добрите практики в разработката на софтуер, който помага за създаването на висококачествени уеб приложения.

• **Application Builder Extension:**

Application Builder Extension може да се използва за автоматизиране на различни процеси, свързани с разработката, конфигурирането и управлението на уеб приложения. Ето някои от примерите как може да се използва този инструмент за автоматизация:

- **Генериране на проекти и компоненти:** **Extension**-а може да се използва за автоматично създаване на нови проекти или компоненти в рамките на съществуващи проекти. Например, може да се създаде шаблон за генериране на ново уеб приложение със стандартна конфигурация и основни компоненти като контролери, модели и изгледи.
- **Управление на зависимости:** **Extension**-а може да се използва за автоматично добавяне и конфигуриране на зависимости на проектите, като например инсталиране на **NuGet** пакети, настройка на връзки с бази данни или интеграция с външни **API**.
- **Генериране на код и файлове:** Може да се използва за автоматично генериране на код и файлове, като например контролери, изгледи, конфигурационни файлове или скриптове. Това улеснява и ускорява процеса на създаване на нови функционалности или компоненти в приложението.

- **Автоматично тестване на кода:** **Extension**-а може да се използва за автоматично изпълнение на тестове на приложението или за провеждане на преглеждане на кода, като се намират потенциални проблеми или неконсистентности в кода.
- **Деплоймънт и обновяване на приложения:** Може да се използва за автоматично генериране на скриптове за деплоймънт и обновяване на приложенията, което улеснява и оптимизира процеса на доставка и развитие на софтуера.

Чрез използването на **Application Builder Extension** за автоматизация на различни аспекти на разработката на уеб приложения в **ASP.NET**, разработчиците могат да постигнат по-голяма продуктивност, по-малко човешки грешки и по-голяма ефективност в управлението на своите проекти.

- **CamelCase и PascalCase**

CamelCase и **PascalCase** са две практики за именуване на променливи, функции, класове и други елементи в програмирането. Ето какво представляват и как се различават:

- **CamelCase:** При **CamelCase**, първата дума се пише с малка буква, а всяка следваща започва с главна буква, като всички думи се сливат без интервали. Използва се за имена на локални полета.

Пример: **myVariableName**, **calculateTotalAmount**, **getUserInfo**.

- **PascalCase:** При **PascalCase**, всяка дума започва с главна буква и всички думи се сливат без интервали. Обикновено се използва за имена на класове, интерфейси, методи и други типове елементи, които представляват типове или абстракции. Използва се за именуване на методи, интерфейси, класове, публични полета и др.

Пример: **MyClass**, **CalculateTotalAmount**, **GetUserInfo**.

- **GUID**

GUID (Globally Unique Identifier) е уникален идентификатор, който се използва за идентифициране на обекти или данни в различни системи или приложения. В основата си, **GUID** е 128-битово числово стойност, която се представя като низ от 32 шестнадесетични цифри, разделени на четири групи по осем цифри, разделени с тирета. Например: „6B29FC40-CA47-1067-B31D-00DD010662DA“.

Ето някои от същността и ползите на **GUID**:

- **Уникалност:** Всеки **GUID** е уникален в рамките на глобалното пространство. Това означава, че вероятността два различни обекта да имат еднакъв **GUID** е изключително малка, като практически е невъзможно.
- **Глобална уникалност:** **GUID**-овете се генерират така, че да бъдат уникални не само в рамките на едно приложение, но и в рамките на целия свят. Това ги прави изключително полезни при разпределени системи и приложения, където е важно да се гарантира уникалността на идентификаторите.
- **Спецификация и стандартизация:** Формата и генерацията на **GUID** са стандартизирани от **Microsoft** и други организации, като например **ITU-T (International Telecommunication Union - Telecommunication Standardization Sector)**. Това осигурява съвместимост и интероперазивност между различни системи и платформи.
- **Използване в бази данни и системи за управление на данни:** **GUID** се използва широко в бази данни и системи за управление на данни като уникални идентификатори на записи или обекти. Те са особено полезни при синхронизацията на данни между различни източници или при скалируеми системи с разпределени данни.
- **Сигурност:** **GUID**-овете могат да бъдат използвани за криптиране, като например генериране на случайни ключове за криптиране и дигитални подписи.

Общо взето, **GUID** предоставят ефективен начин за уникално идентифициране на обекти или данни в различни системи и приложения, като гарантират уникалността и глобалната идентификация на тези елементи.

Библиотеки

- **LINQ (Language Interpreted Query)**

LINQ (Language Integrated Query) е библиотека в **C#**, която позволява извършването на заявки върху данни директно в езика, без да се налага да се използват специфични езици за заявки като **SQL**. В контекста на **ASP.NET**, **LINQ** може да се използва за обработка на данни от бази данни, колекции, **XML** файлове и други източници на данни.

Ползите на **LINQ** в **ASP.NET** включват:

- **Интеграция с езика C#:** **LINQ** е интегрирана в **C#**, което прави синтаксиса на заявките много по-близък до програмния език, който използвате за разработката на уеб приложения. Това улеснява разработката и поддръжката на кода.
- **Декларативен синтаксис:** **LINQ** предоставя декларативен синтаксис за създаване на заявки, което улеснява четимостта и разбирането на кода, особено за хора, които не са запознати с **SQL** или други езици за заявки.
- **Съвместимост с различни източници на данни:** **LINQ** може да се използва за обработка на данни от различни източници, включително бази данни, **XML** файлове, колекции от обекти и други.
- **Проверка на типовете по време на компилация:** **LINQ** предлага типизирано синтаксис, който позволява проверка на типовете по време на

компиляция, което намалява риска от грешки по време на изпълнение на програмата.

Някои от основните методи, предоставени от **LINQ**, са:

- **Where:** Извлича елементите от колекцията, които отговарят на определено условие.
- **Select:** Трансформира елементите на колекцията, като извлича определени атрибути или прилага функции върху тях.
- **OrderBy/OrderByDescending:** Сортира елементите на колекцията във възходящ или низходящ ред според определен критерий.
- **First/FirstOrDefault:** Връща първия елемент от колекцията или първия елемент, който отговаря на условие.
- **GroupBy:** Групира елементите на колекцията според определен критерий.
- **Join:** Обединява две колекции въз основа на определен критерий.
- **Aggregate:** Извършва агрегатни операции като сбор, средно аритметично и други.

Тези методи се използват за създаване на заявки, които манипулират и обработват данни по удобен и ефективен начин.

• **EntityFramework.Configurations**

EntityFramework.Configurations е част от **Entity Framework**, който се използва за конфигуриране и настройване на модела на данните, който **Entity Framework** използва за взаимодействие с базата данни. Тази компонента позволява на разработчиците да контролират различни аспекти на създаването, мигрирането и управлението на базата данни чрез код.

Някои от основните ползи на **EntityFramework.Configurations** включват:

- **Конфигуриране на модела на данните:** Позволява на разработчиците да дефинират модела на данните, като определят класове, които съответстват на таблиците в базата данни, и свойства, които съответстват на колоните в тези таблици.
- **Управление на връзките между таблиците:** Предоставя възможност за дефиниране на връзките между различните таблици в базата данни, като например едно към много, много към много и други.
- **Конфигуриране на индекси и ключове:** Позволява на разработчиците да дефинират индекси и ключове върху колоните в таблиците, което ускорява търсенето и заявките към базата данни.
- **Контрол на поведението при мигриране и създаване на базата данни:** Предоставя възможност за дефиниране на правила и настройки за мигрирането на базата данни, като например автоматичното създаване на базата данни, извършване на миграции на данни и други.
- Чрез използването на **EntityFramework.Configurations**, разработчиците могат да персонализират и управляват модела на данните в своите приложения, което допринася за по-голяма гъвкавост и ефективност в работата с базата данни.

Среди за разработка и контрол на версиите

- **Visual Studio 2022 Community**

Visual Studio Community 2022 е безплатна интегрирана среда за разработка (**IDE**), предоставена от **Microsoft**, която позволява на разработчиците да създават различни видове приложения за уеб, мобилни устройства, настолни компютри и други. Ето някои от неговите основни характеристики и ползи:

- **Безплатна за използване:** **Visual Studio Community** е напълно безплатна за употреба от физически лица, учебни заведения и малки фирми с до 5 служители. Това я прави достъпна за широк кръг от разработчици.
- **Интегрирана среда за разработка:** Предоставя интегрирана среда за разработка, която обединява всички необходими инструменти за програмиране, от текстовия редактор и компилатора до дебъгера и инструментите за управление на версиите.
- **Многоплатформена поддръжка:** Поддържа разработка на приложения за различни операционни системи и платформи, включително уеб, мобилни устройства (**Android** и **iOS**), настолни компютри (**Windows, macOS, Linux**) и облачни услуги.
- **Богата екосистема:** Предлага широка гама от интегрирани инструменти и библиотеки за разработка на различни видове приложения, като например **ASP.NET, Xamarin, .NET Core, Azure** и други.
- **Лесно за употреба и учене:** Има интуитивен и лесен за употреба потребителски интерфейс, който улеснява на новите разработчици да започнат да работят и да се научат на различните функционалности.
- **Обширна документация и общност:** Предлага обширна документация, онлайн ресурси и активна общност от разработчици, които могат да помогнат на потребителите да решат проблеми и да споделят знания и опит.

Всички тези характеристики правят **Visual Studio Community 2022** мощен инструмент за разработка на софтуер, който е достъпен за различни категории на разработчици и проекти.

- **GitHub и Github Desktop**

- **GitHub** е онлайн платформа за управление на изходен код и проекти. Позволява на разработчиците да съхраняват, управляват и споделят своите проекти с други, както и да сътрудничат по тях. Ето някои от основните същности и ползи на **GitHub**:
 - **Версионирание на кода:** **GitHub** предлага система за контрол на версиите (**VCS**), която позволява на разработчиците да следят промените в своя код, да създават и сливат различни версии на проектите си и да се възстановяват при нужда от предишни версии.

- **Сътрудничество:** Разработчиците могат лесно да сътрудничат помежду си, като правят коментари към кода, отварят проблеми (**issues**), правят и преглеждат пълно функционални заявки за теглене (**pull requests**) и още.
 - **Управление на задачи и проблеми:** Разработчиците могат да създават, преглеждат и управляват задачи и проблеми (**issues**) в **GitHub**, което помага за организирането на работата и следенето на напредъка на проектите.
- **GitHub Desktop** е официален клиент на **GitHub**, който предоставя графичен потребителски интерфейс за работа с репозитории на **GitHub**. Ето някои от ползите и същността на **GitHub Desktop**:
- **Лесен за използване:** **GitHub Desktop** предлага интуитивен и лесен за използване интерфейс, който позволява на разработчиците да управляват своите репозитории на **GitHub** без да използват команден ред.
 - **Управление на репозитории:** Разработчиците могат да създават, клонират, актуализират и качват репозитории на **GitHub** чрез **GitHub Desktop**, като също така могат да преглеждат историята на промените и да правят комити.
 - **Сътрудничество и комуникация:** **GitHub Desktop** предоставя възможност за сътрудничество и комуникация между разработчиците, като позволява на потребителите да правят и преглеждат коментари към кода и да работят в екип по проекти.
 - **Интеграция с GitHub:** **GitHub Desktop** се интегрира напълно с платформата на **GitHub**, което позволява на разработчиците да синхронизират своите локални промени с уеб-базираните репозитории и да използват всички функционалности на **GitHub** пряко от клиента.

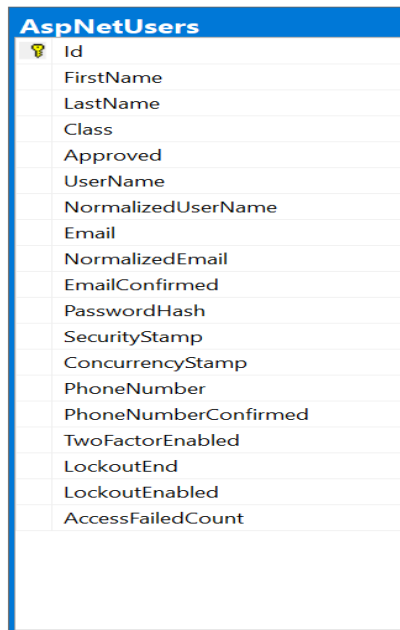
3. Архитектура на Базата Данни

Таблицы

- **AspNetUsers:**

AspNetUsers (Фигура 1) таблицата представлява мястото където се съхраняват потребителите и техните данни. В таблицата се намират полетата:

- **Id:** В това поле се съдържа идентификатора на потребителя, под формата на низ.
- **FirstName:** В това поле се съдържа Името на потребителя под формата на низ.
- **LastName:** В това поле се съдържа фамилията на потребителя под формата на низ.
- **Class:** В това поле се съдържа Класът на ученика въведен от него при регистрация под формата на низ.
- **Approved:** В това поле се съдържа булева променлива която показва дали потребителя е бил приет от администратора.
- **Username** и **NormalizedUserName:** Тези две полета съдържат потребителското име на потребителя като едното го държи с главни и малки букви, а другото само с главни, с цел при влизане в профила да няма значение дали потребителя е въвел потребителското си име със същите главни и малки букви като при регистрирането.
- **Email** и **NormalizedEmail:** Тези две полета съдържат имейла на потребителя като едното го държи с главни и малки букви, а другото само с главни, с цел при влизане в профила да няма значение дали потребителя е въвел имейла си със същите главни и малки букви като при регистрирането.
- **EmailConfirmed:** Поле, съдържащо булева променлива, която показва дали потребителя е потвърдил своя имейл.
- **PasswordHash:** Поле, което съдържа паролата на потребителя в криптиран вид.
- **SecurityStamp:** Това поле съдържа уникален идентификатор, който се използва за повишаване на сигурността на приложението. Когато потребителят промени своята парола или други лични данни, този идентификатор се обновява.
- **ConcurrencyStamp:** Поле, съдържащо маркер за конкурентност, който се използва за проверка на версията на потребителския запис при



Id	FirstName	LastName	Class	Approved	Username	NormalizedUserName	Email	NormalizedEmail	EmailConfirmed	PasswordHash	SecurityStamp	ConcurrencyStamp	PhoneNumber	PhoneNumberConfirmed	TwoFactorEnabled	LockoutEnd	LockoutEnabled	AccessFailedCount
----	-----------	----------	-------	----------	----------	--------------------	-------	-----------------	----------------	--------------	---------------	------------------	-------------	----------------------	------------------	------------	----------------	-------------------

Фигура 1


конкурентни операции. Когато се прави изменение в потребителския запис, този маркер се обновява, за да се гарантира, че няма конфликт при заявките за изменение на записа.

- **PhoneNumber**: Поле, което съдържа телефонния номер на потребителя.
- **PhoneNumberConfirmed**: Поле, съдържащо булева променлива, която обозначава дали потребителя е потвърдил телефонния си номер.
- **TwoFactorEnabled**: Поле, съдържащо булева променлива, показваща дали потребителя е избрал да включи **TwoFactor Authentication**
- **LockoutEnabled**: Поле, съдържащо булева променлива което показва дали функцията за заключване на профила е активирана за даден потребител. Когато това поле е истина, потребителят може да бъде деактивиран за определен период от време, ако има прекалено много неуспешни опити за влизане.
- **LockoutEnd**: Това е поле, което указва края на периода на деактивиране на профила, ако такъв е налице. Полето е попълнено само когато потребителят е бил деактивиран.
- **AccessFailedCount**: Променлива съдържаща броя провалени влизания в профила.

■ **AspNetRoles**

AspNetRoles (Фигура 2) е автоматично генерирана от **ASP.Net** таблица предназначена да съхранява потребителските роли, използвани в приложението.

- **Id**: Поле, съдържащо идентификатора на ролята под формата на низ.
- **Name**: Поле, съдържащо името на ролята с главни и малки букви.
- **NormalizedName**: Поле, съдържащо името на ролята в нормализиран вид (Само с главни букви).
- **ConcurrencyStamp**: Поле, съдържащо маркер за конкурентност, който се използва за проверка на версията на ролята при конкурентни операции.



AspNetRoles	
	Id
	Name
	NormalizedName
	ConcurrencyStamp

Фигура 2

■ **AspNetUserRoles**

AspNetUserRoles (Фигура 3) е таблица посредник служеща на много към много връзката между **AspNetUsers** и **AspNetRoles**, като всеки запис в таблицата представлява връзката между потребител и една от неговите роли, като в случая на наето приложение всеки има само по една.

- **UserId:** Държи идентификатора на потребител под формата на низ.
- **RoleId:** Държи идентификатора на роля под формата на низ.


AspNetUserRoles	
	UserId
	RoleId

Фигура 3

▪ Subjects

Subjects (Фигура 4) е таблицата държаща всички учебни предмети достъпни в приложението.

- **Id:** Държи идентификатора на учебния предмет под формата на низ.
- **SubjectName:** Държи името на учебния предмет.



Subjects	
	Id
	SubjectName

Фигура 4

▪ ApplicationUserSubjects

ApplicationUserSubjects (Фигура 5) таблицата представлява много към много връзката между потребителите и учебните предмети. Само учителите могат да бъдат асоциирани с учебни предмети, като един учител може да преподава по няколко предмета.

- **TeacherSubjectId:** Държи идентификатора на учебния предмет, по който преподава учителя.
- **UsersId:** Държи идентификатора на учителя.

ApplicationUserSubject	
	TeacherSubjectsId
	UsersId


Фигура 5

▪ Competitions

Competitions (Фигура 6) таблицата представлява мястото където бива записвана информацията за всички състезания налични в приложението.

- **Id:** Държи идентификатора на състезанието под формата на низ.
- **Name:** Държи името на състезанието под формата на низ.
- **Description:** Държи описанието на състезанието под формата на низ.
- **RegistrationDeadline:** Държи крайната дата за записване под формата на дата.
- **Location:** Държи мястото на провеждане на състезанието под формата на низ.
- **MaxParticipants:** Държи максималния брой ученици, които могат да се запишат за състезанието.

- **IsFull:** Държи стойност на булева променлива, която когато е вярна означава, че свободните места в състезанието са били запълнени.
- **IsActive:** Държи стойност на булева променлива, която когато е вярна означава, че състезанието все още е активно.
- **CompetitionTypeId:** Държи идентификатора на типа на състезанието, който е бил избран при създаване.
- **SubjectId:** Държи идентификатора на предмета по който е състезанието.
- **CurrentParticipants:** Държи броя на записани ученици към момента.
- **DateOfConduct:** Държи дата на провеждане под формата на дата.



Competitions	
	Id
	Name
	Description
	RegistrationDeadline
	Location
	MaxParticipants
	IsFull
	IsActive
	CompetitionTypeId
	SubjectId
	CurrentParticipants
	DateOfConduct

Фигура 6

▪ ApplicationUserCompetitions

ApplicationUserCompetitions (Фигура 7) представлява междинната таблица на **AspNetUsers** и **Competitions**. Всеки запис представлява един ученик записан за едно състезание.

- **CompetitionId:** Съдържа идентификатора на състезанието.
- **UserId:** Съдържа идентификатора на ученика.

ApplicationUserCompetition	
	CompetitionId
	UserId

Фигура 7

▪ CompetitionTypes

CompetitionTypes (Фигура 8) е таблицата в която се съдържат всички типове състезания (национални, общински, областни олимпиади и др.).

- **Id:** Съдържа идентификатора на типа състезание.
- **Type:** Съдържа името на типа състезание под формата на низ.

CompetitionTypes	
	Id
	Type

Фигура 8

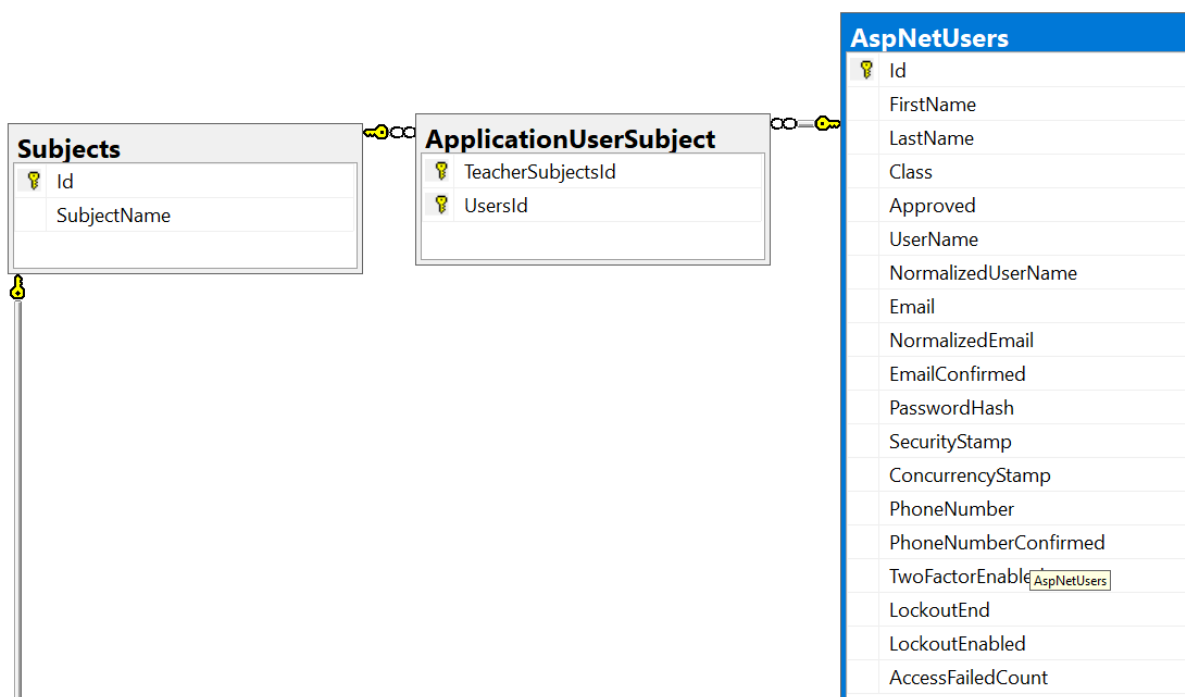
Връзки

- Потребители → Предмети

Връзката между Потребителите (таблица **AspNetUsers**) и предметите (**Subjects**) е от тип много към много, което значи че много учители могат да преподават по много предмети. Връзката се осъществява чрез междинна (mapping) таблица, която обозначава връзката между всеки един потребител и всеки един предмет свързан към него.

Пример: Учител едно преподава по: Математика, География и Физика. Учител две преподава по Математика, История и Философия. И двамата учители преподават по повече от един предмет, и двамата преподават по Математика.

Връзката е представена чрез *фигура 9*.

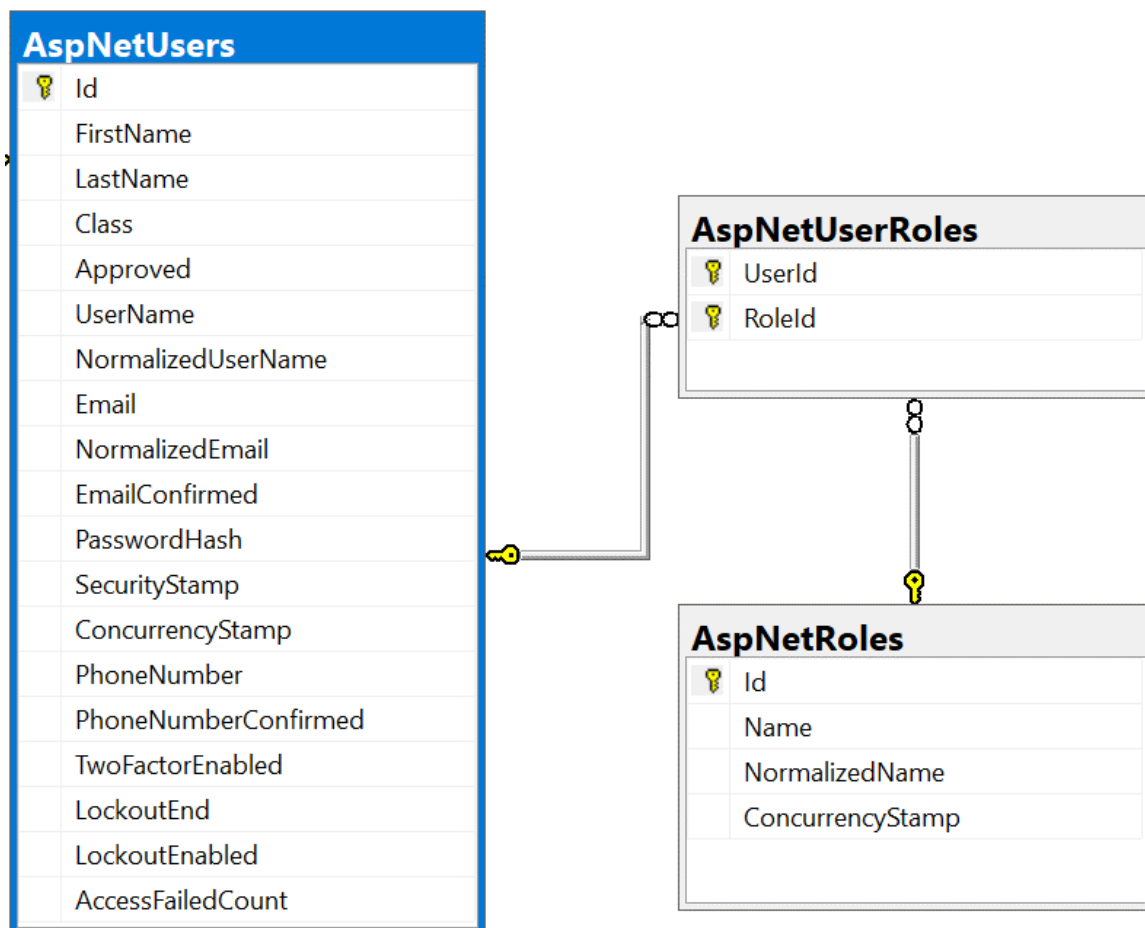


Фигура 9

- Потребители → Роли

Връзката между потребителите (таблица **AspNetUsers**) и ролите (таблица **AspNetRoles**) работи по аналогичен начин на тази между потребителите и учебните предмети. Връзката е много към много и отново ползва междинна таблица.

Връзката е представена чрез *фигура 10*.

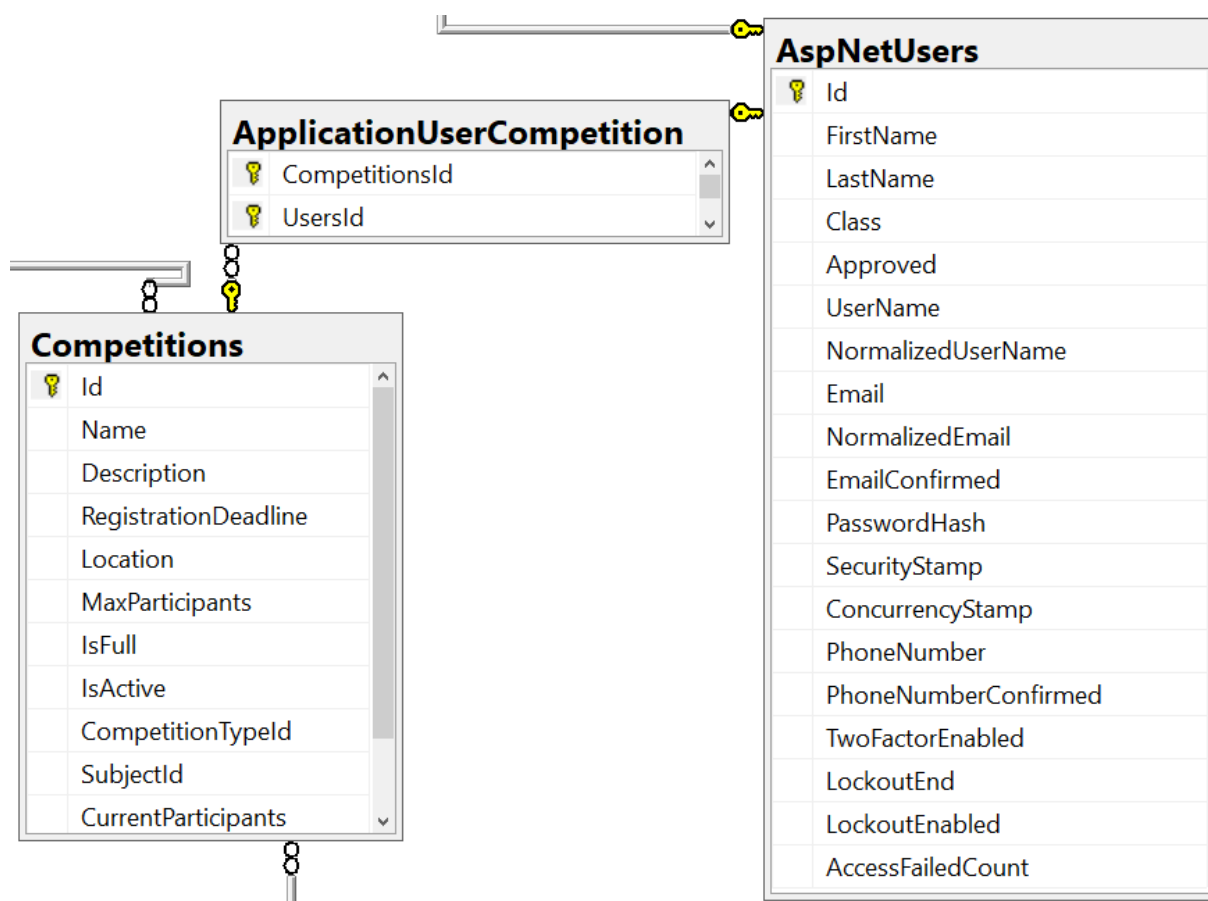


Фигура 10

- Потребители → Състезания

Връзката потребители (таблица **AspNetUsers**) и Състезания (таблица **Competitions**) е много към много изразена чрез междинна таблица като по горе показаните връзки. Същността на релацията е, че много потребители могат да бъдат записани за много състезания.

Релацията е показана на *фигура 11*.

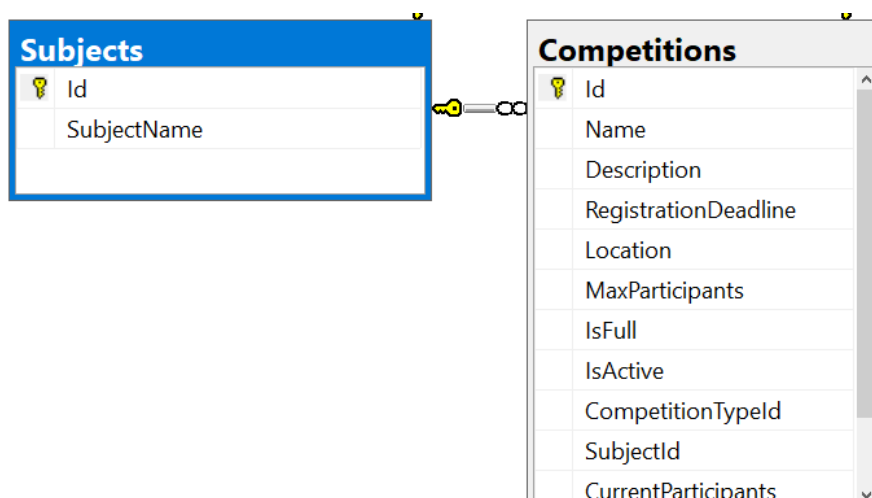


Фигура 11

- Състезания → Предмети

Връзката между Състезанията (таблица **Competitions**) и Учебните предмети (таблица **Subjects**) е едно към много. Това означава че много състезания могат да са по един и същи предмет, но едно състезание не може да е по много предмети.

Връзката е онагледена чрез **фигура 12**.

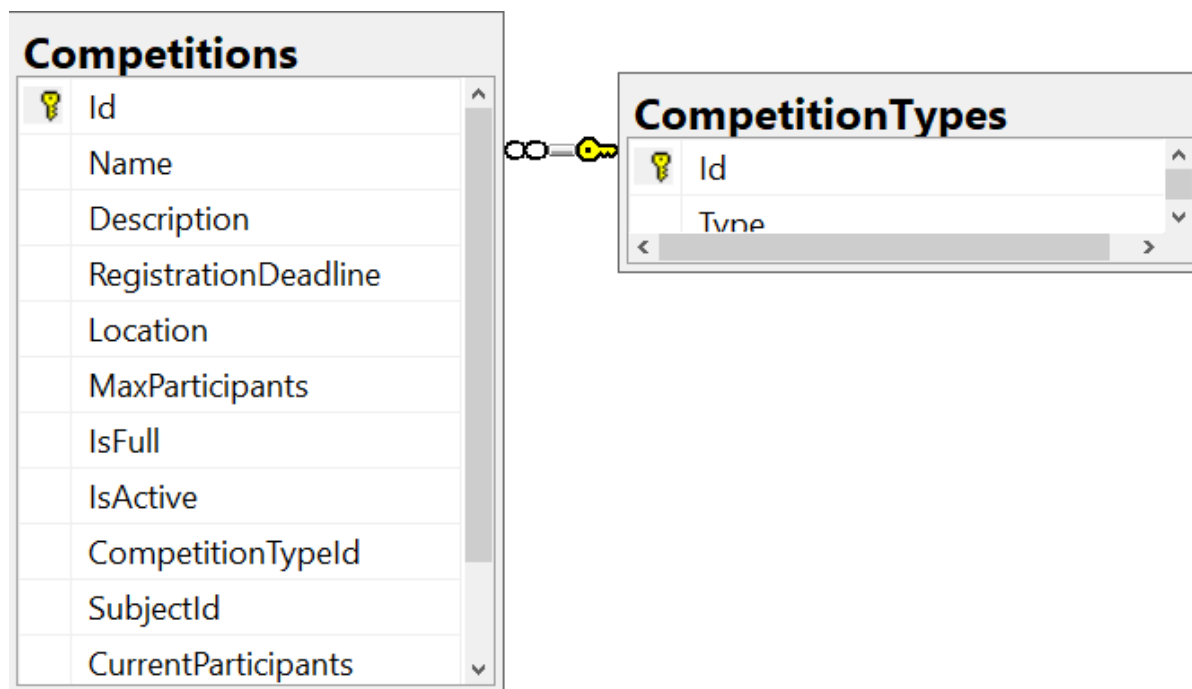


Фигура 12

- Състезания → Типове състезания

Връзката между Състезанията (таблица **Competitions**) и Типовете състезания (таблица **CompetitionTypes**) е аналогична на връзката Състезания → Учебни предмети, като едно състезание може да е само един тип, но много състезания могат да бъдат от един и същи тип.

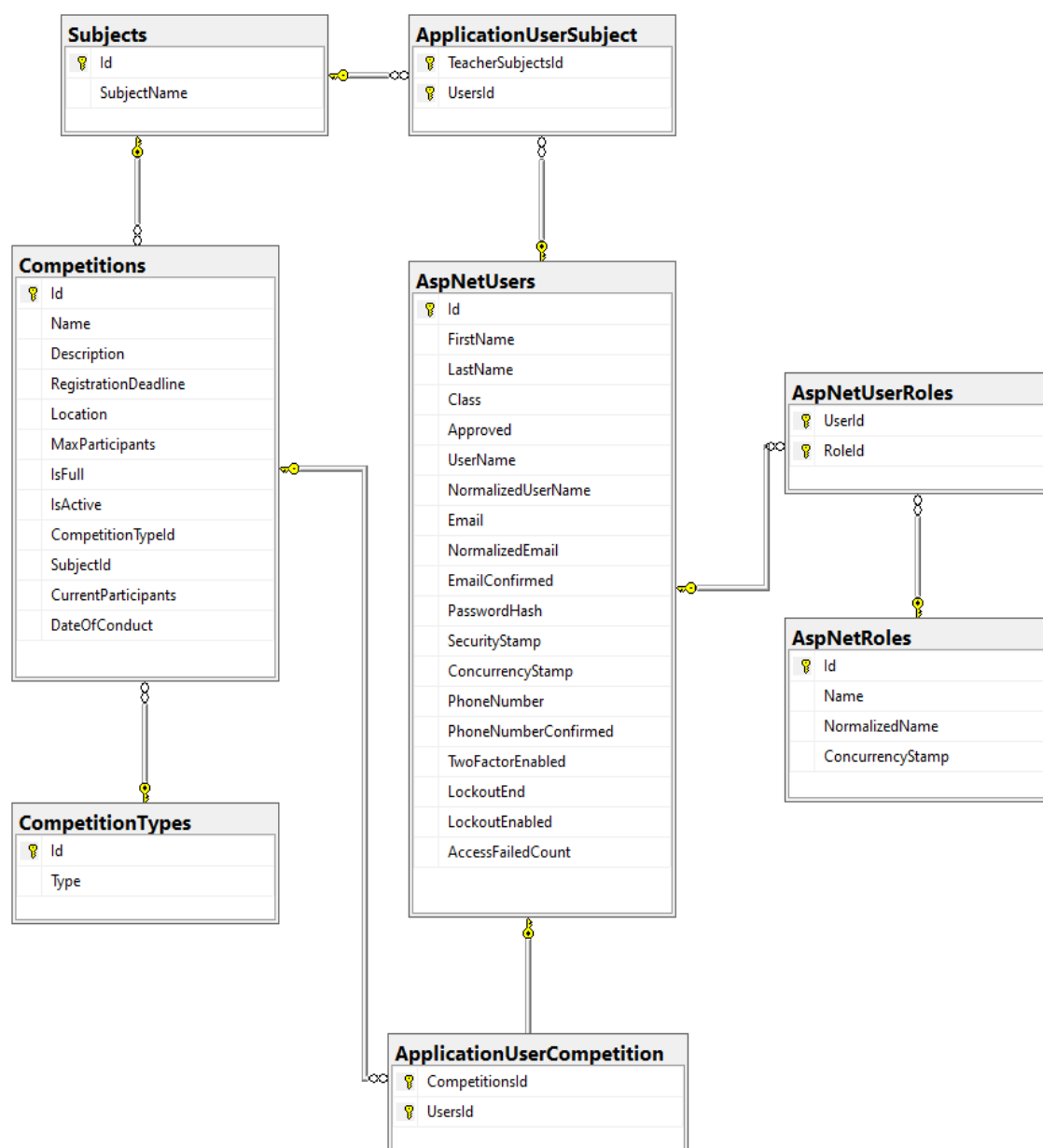
Връзката е онагледена чрез *фигура 13*.



Фигура 13

Диаграма на базата

- Цялата база и всички нейни връзки и таблици са представени във *Фигура 14*.



Фигура 14

4. Представяне и обясняване на уеб приложението

- Структура
 - Какво използваме от вече създадената файлова система:

Разглеждайки автоматично генерираната файлова система (*Фигура 23*) на ASP.Net MVC проекта намираме голям набор от папки и файлове. Единствените елементи от файловата система за които няма да говорим са: **ConnectedServices**, **Dependencies**, папката **Properties**, **wwwroot** и папката **Areas**, понеже не са използвани в проекта.

- Папка Контролери (**Controllers**)

Автоматично генерираната папка **Controllers** държи основната бизнес логика на приложението. В нея добавяме нови контролери за всеки раздел функционалности на приложението. В нашия случай контролерите са 5 (**CompetitionController**, **CompetitionTypeController**, **HomeController**, **SubjectController** и **UserController**), за които ще говорим по-долу. Имената на контролерите са много важни понеже ASP.Net използва специален начин за обръщане към контролери като кум тях се обръщаме само с основната част от името на контролера.

Пример: към **UserController** се обръщаме само с **User** и след това съответната функция. Това е нагледно и в **routing**-а (нагледно във *Фигура 15*), като този линк ще ни отведе във формата за влизане в акаунт.

<https://localhost:44304/User/Login>

Фигура 15

Пример за контролер:

```
public class CompetitionController : Controller
{
    private ICompetitionService _competitionService;
    private ISubjectService _subjectService;
    private ICompetitionTypeService _competitionTypeService;
    private protected SignInManager<ApplicationUser> _signInManager;

    public CompetitionController(ICompetitionService competitionService, ISubjectService
subjectService, ICompetitionTypeService competitionTypeService,
SignInManager<ApplicationUser> signInManager)
    {
        _competitionService = competitionService;
        _subjectService = subjectService;
        _competitionTypeService = competitionTypeService;
        _signInManager = signInManager;
    }

    [HttpGet]
    [Authorize(Roles = "Admin, Teacher, Student")]

    public async Task<IActionResult> GetAll(CompetitionGetAllViewModel? model, int?
pageSize, int? pageNumber)
```

```

{
    var list = await _competitionService.GetAll();

    model.Subjects = await _subjectService.GetAll();

    if (model.Subject != null)
    {
        model.Subject = await _subjectService.GetById(model.Subject.Id);
    }
...}}

```

На примерния код виждаме декларацията на класа, наследяващ стандартен клас **Controller**. След декларацията на класа правим обект на всички услуги, които ще ползваме, които в случая са услугите отговарящи за състезания, предмети и типове състезания, както и услугата **SignInManager**, която ни позволява да използваме вече готови функции за създаване на нови потребители и влизане в акаунт. След декларацията на услугите имаме конструктор, който позволява те да бъдат използвани при извикване на контролера.

В края на откъса код виждаме метод за вземане на всички потребители от базата, а по скоро неговата **Get** част, която предоставя всичко нужно за изобразяване в потребителската част. След обявяването на типа на метода и авторизацията му виждаме че в тялото на метода вместо да изписваме наново всеки достъп до базата за изкарване на всички предмети и типове, използваме вече създадените услуги за тази цел. Контролера има също **Post** част на всеки метод, която отговаря за действията извършени след натискане на бутон за изпращане или след каквото и да е действие на потребителя свързано с обновяване на данни.

- Папка Данни (**Data**)

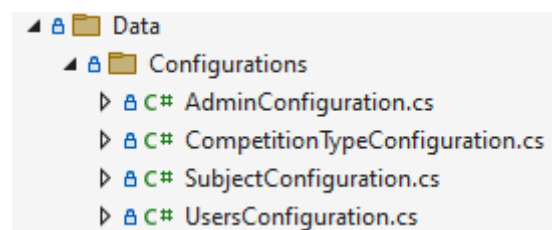
В папката *Данни* намираме основните файлове и папки отговарящи за връзката с базата.

- Папка Конфигурации (**Configurations**)

Ръчно добавената папка *Конфигурации* държи файловете, които използваме за да вкараме потребители, учебни предмети и типове състезания при стартирането на проекта.

В случая имаме 4 конфигурации (*Фигура 16*), отговарящи за създаване на администратор, потребителски и учителски акаунт, типове състезания и учебни предмети.

Като пример за конфигурация ще разгледаме тази за администраторски акаунт, понеже тя е най-сложна и се препокрива с останлите като логика.



Фигура 16

Примерен код:

```
1 public class AdminConfiguration : IEntityTypeConfiguration<ApplicationUser>
2 {
3     public void Configure(EntityTypeBuilder<ApplicationUser> builder)
4     {
5         var admin = new ApplicationUser
6         {
7             Id = "255a2f5d-901c-4966-b091-b5ed871a3c4d",
8             FirstName = "admin",
9             LastName = "admin",
10            Email = "admin@email.com",
11            UserName = "admin",
12            NormalizedEmail = "admin@email.com".Normalize(),
13            NormalizedUserName = "admin".Normalize(),
14            Approved = true
15        };
16        var passwordHasher = new PasswordHasher<ApplicationUser>();
17        admin.PasswordHash = passwordHasher.HashPassword(null, "*****");
18
19        builder.HasData(admin);
20    }
21 }
```

Разглеждайки кода ред по ред започваме от декларацията на класа н ред 1. За да използваме конфигурации разширяваме класа с наследяването на **IEntityTypeConfiguration** интерфейса, задавайки му тип **ApplicationUser**.

От интерфейса идва метода **Configure** ред 3, който понеже както ще обясним по-късно е част от **OnModelCreating** метода в контекста, трябва да получи като аргумент **EntityTypeBuilder<ApplicationUser>**.

След това на ред 5 създаваме нов обект от класа **ApplicationUser** и го попълваме с нужните данни. Използваме **Normalize** метода на редове 12 и 13 за да сме сигурни че при въвеждане на имейла при логин няма да има грешки, понеже данните които ще вкараме от тук в базата няма да бъдат конфигурирани допълнително от програмата.

На ред 16 създаваме нов обект на класа **PasswordHasher** от тип **ApplicationUser**.

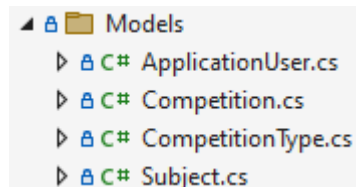
На следващия ред използваме този обект за да криптираме паролата на потребителя.

На ред 19 подаваме на **builder**-а (който е част от интерфейса който ползваме) потребителя, който създадохме за да той да се погрижи за неговото вкарване в базата.

В тези конфигурации е много важно да ползваме точния модел използване за създаване на базата за да може контекста да знае за коя таблица е конфигурацията. В конфигурациите можем също да разполагаме и с връзките между таблиците и като цяло всичко което искаме да се случва при стартиране на програмата.

- Папка Модели (**Models**)

Моделите в **ASP.Net** отговарят за създаването на таблиците в базата данни. Всеки модел представлява клас, описващ това което искаме да се появи в базата данни като таблица. В нашето приложение има 4 модела (Разширение на потребителя: **ApplicationUser**, състезание: **Competition**, учебен предмет: **Subject** и тип състезание: **CompetitionType**, показани на *Фигура 17*). Ще разгледаме като пример за модел **Competition**, понеже той съдържа всеки тип връзка и променлива, използвани в проекта.



Фигура 17

Примерен код:

```

1 public class Competition
2 {
3     [Key]
4     public Guid Id { get; set; }
5
6     [StringLength(100, MinimumLength = 3)]
7     public string Name { get; set; } = null!;
8
9     [StringLength(300, MinimumLength = 3)]
10    public string Description { get; set; } = null!;
11
12    public int CurrentParticipants { get; set; }
13
14    public DateTime RegistrationDeadline { get; set; }
15
16    public DateTime DateOfConduct { get; set; }
17
18    [StringLength(100, MinimumLength = 3)]
19    public string Location { get; set; } = null!;
20
21    public int MaxParticipants { get; set; }
22
23    public bool IsFull { get; set; }
24
25    public bool IsActive { get; set; }
26
27    public CompetitionType CompetitionType { get; set; } = null!;
28
29    public Subject Subject { get; set; } = null!;
30
31    public List<ApplicationUser> Users { get; set; } = null!;
32 }

```

Разглеждайки примерния код виждаме всички полета от таблицата в базата с няколко допълнителни.

Като за начало на ред три можем да видим обяснените в [Framework-ове EntityFramework.Schema](#) атрибути за обозначаване на полетата. В случая е поставен атрибут [**Key**] обозначаващ главен ключ на таблицата.

Полетата в таблицата са създадени по идентичен начин използвайки атрибути като максимална и минимална дължина за да заменим писането на **SQL** код. Те представляват просто публични променливи с гет и сет действия, като за да нямаме предупреждения за съществуването на колекции и компонентни обекти се налага да използваме „= null!;“, за да предупредим контекста за естеството на полето.

Последните три полета, са според мен най-интересни, понеже това са полетата, които наричаме навигационни. Навигационното поле показва, че таблицата, която гледаме в момента има връзка към таблицата, чиито модел извикваме като поле. 27 и 29 ред са примери за връзка 1 към много като те показват, че едно състезание може да има съответно само един тип и да бъде по само един предмет. Този тип връзка няма нужда от така нареченото обратно поле показващо зависимостта в другата таблица. 31 ред съответно е пример за връзка много към много. Като създадем лист или колекция от обекти по модела на таблицата, с която желаем тази връзка показваме на **EntityFramework**, че искаме едно състезание да може да има много ученици записани за него, но съответно за да може да се създаде мапинг таблицата между Състезания и потребители трябва в модела на потребителите да има лист от състезания, за да програмата да разбере, че искаме много ученици да може да са записани за много състезания

○ Контекст (**ApplicationDbContext**)

Контекстът на приложението служи, за да програмиста да може да определи кои класове да се ползват за модели, като ги направи на така наречените **DbSet**-ове. В контекста също се добавя метод, с точно определено име, който отговаря за конфигурирането на базата.

Код на контекста:

```
public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
{
    public DbSet<Subject> Subjects { get; set; }
    public DbSet<CompetitionType> CompetitionTypes { get; set; }
    public DbSet<Competition> Competitions { get; set; }

    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
    }

    protected override void OnModelCreating(ModelBuilder builder)
    {
        builder.ApplyConfiguration(new AdminConfiguration());
        builder.ApplyConfiguration(new UsersConfiguration());
        builder.ApplyConfiguration(new SubjectConfiguration());
        builder.ApplyConfiguration(new CompetitionTypeConfiguration());
    }
}
```

```
base.OnModelCreating(builder);  
}
```

Тук виждаме декларацията на класа контекст и това, че той наследява готов клас отговарящ за идентичностите (потребителите). Първите 4 реда от класа са декларациите на гореспоменатите **DbSet**-ове, показващи кои модели, да бъдат разчетени и превърнати в таблици.

След тях виждаме декларация на конструктора на контекста, който приема настройки наследени от класа обозначен при декларацията на контекста, тоест няма нужда да ги попълваме при създаване на обект на контекста.

В края на контекст класа намираме и метода, който изпълнява конфигурациите, които сме направили. Този метод също е с важно име понеже е стандартен и за да бъде разпознат и изпълнен трябва да се казва така и да приема тези атрибути.

- Папка Разширения (**Extensions**)

Ръчно добавената папка *Разширения* (Фигура 18) съдържа всички разширения добавени от потребителя.

```

0 references
public static class ApplicationBuilderExtension
{
    1 reference
    public static IApplicationBuilder SeedRoles(this IApplicationBuilder app)
    {
        using IServiceScope scopedServices = app.ApplicationServices.CreateScope();

        IServiceProvider services = scopedServices.ServiceProvider;

        UserManager<ApplicationUser> userManager = services.GetRequiredService<UserManager<ApplicationUser>>();
        RoleManager<IdentityRole> roleManager = services.GetRequiredService<RoleManager<IdentityRole>>();

        Task.Run(async () =>
        {
            if (!await roleManager.RoleExistsAsync("Admin"))
            {
                IdentityRole adminRole = new IdentityRole("Admin");
                await roleManager.CreateAsync(adminRole);
            }

            if (!await roleManager.RoleExistsAsync("Teacher"))
            {
                IdentityRole adminRole = new IdentityRole("Teacher");
                await roleManager.CreateAsync(adminRole);
            }

            if (!await roleManager.RoleExistsAsync("Student"))
            {
                IdentityRole userRole = new IdentityRole("Student");
                await roleManager.CreateAsync(userRole);
            }

            ApplicationUser admin = await userManager.FindByNameAsync("admin");
            await userManager.AddToRoleAsync(admin, "Admin");

            ApplicationUser teacher = await userManager.FindByNameAsync("teacher");
            await userManager.AddToRoleAsync(teacher, "Teacher");

            ApplicationUser student = await userManager.FindByNameAsync("student");
            await userManager.AddToRoleAsync(student, "Student");
        })
        .GetAwaiter()
        .GetResult();

        return app;
    }
}

```

Фигура 18

Това разширение се изпълнява в класа **Program.cs**, който е обяснен по-долу. Неговата функция е автоматично добавяне на основните роли и намирането на потребителите вкарани от горе обяснените конфигурации, за да им бъдат зададени роли. За тази цел се използва услугата на **ASP.Net: RoleManager**, която отговаря за работата с роли.

- Папка Миграции (**Migrations**)

В папката *Миграции* стоят всички автоматично генерирани миграции (обяснени в [Framework-ове](#)). Те служат за контрол на версиите и в примерната миграция (Фигура 19) виждаме че те всъщност представляват преведен **SQL** код (примерната снимка е само част от миграция).

```

public partial class dateOfCnduct : Migration
{
    /// <inheritdoc />
    0 references
    protected override void Up(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.AddColumn<DateTime>(<
            name: "DateOfConduct",
            table: "Competitions",
            type: "datetime2",
            nullable: false,
            defaultValue: new DateTime(1, 1, 1, 0, 0, 0, 0, DateTimeKind.Unspecified));

        migrationBuilder.UpdateData(
            table: "AspNetUsers",
            keyColumn: "Id",
            keyValue: "0778238c-1e1b-4e5d-9ea6-1a01e9264086",
            columns: new[] { "ConcurrencyStamp", "PasswordHash", "SecurityStamp" },
            values: new object[] { "c5965000-f49a-4a74-aedf-3c3d99f53f9a",
                "AQAAAAIAAYagAAAAEPGLJAE0SIJ0hNre0QAZ/6oKoq010UbK3+h9Ls/Og8mAG425k8UJqspj3gXAIxBDRg==",
                "d3291787-b404-441b-8e08-7fe13de5d19b" });

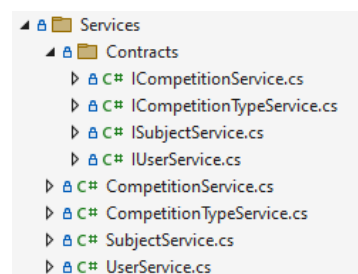
        migrationBuilder.UpdateData(
            table: "AspNetUsers",
            keyColumn: "Id",
            keyValue: "255a2f5d-901c-4966-b091-b5ed871a3c4d",

```

Фигура 19

■ Папка Услуги (Services)

Папката *Услуги* (Фигура 20) съдържа всички услуги и техните *Интерфейси* (Договори), които се намират в подпапката **Contracts** (И действат като посредник между услугата и нейния ползвател: контролера). Броя на услугите, най-често отговаря на контролерите, като в случая те са: **CompetitionService**, **CompetitionTypeService**, **SubjectService** и **UserService**.



Фигура 20

Пример за *Интерфейс*:

```

public interface IcompetitionTypeService
{
    Task Add(CompetitionTypeViewModel model);
    Task Delete(Guid id);
    Task<List<CompetitionTypeViewModel>> GetAll();
    Task Update(CompetitionTypeViewModel model);
    Task<CompetitionTypeViewModel> GetById(Guid id);
}

```

Това е интерфейса на услугата **CompetitionTypeService**. В него се декларирани всички методи, които ще са достъпни до ползвателите на услугата.

Пример за *Услуга*:

```
< references
public class CompetitionTypeService : ICompetitionTypeService
{
    private ApplicationDbContext context;

    0 references
    public CompetitionTypeService(ApplicationDbContext context)
    {
        this.context = context;
    }

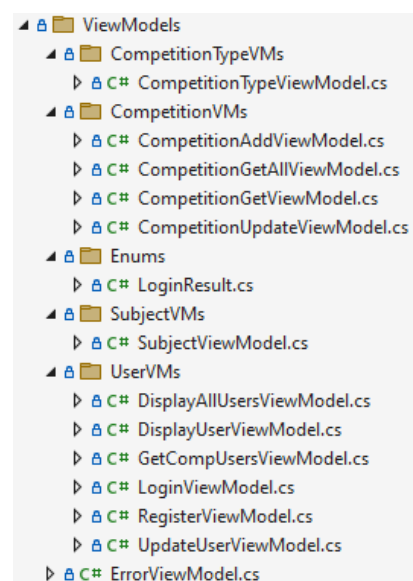
    2 references
    public async Task Add(CompetitionTypeViewModel model)
    {
        var forDb = new CompetitionType
        {
            Type = model.Type
        };
        await context.AddAsync(forDb);
        await context.SaveChangesAsync();
    }

    2 references
    public async Task Delete(Guid id)
    {
        context.Remove(await context.CompetitionTypes.FindAsync(id));
        await context.SaveChangesAsync();
    }
}
```

Това е най-простия пример за услуга, понеже показва основата на нейната функционалност. Методи декларирани в интерфейса, приемащи данни от техния ползвател и връщащи изисквания резултат.

- Папка *Модели за изгледите (ViewModels)*

Папката *Модели за изгледите* (Фигура 21) държи всички псевдомодели на приложението. Псевдомоделите представляват посредник между базата, контролерите и изгледите. Тяхната цел е запазване на сигурността, понеже позволяват само на нужната информация да присъства, и гъвкавост, понеже позволяват на допълнителни променливи да бъдат добавяни за специфични нужди. За всеки тип действие с модел се изисква *Модел за изглед* специално за това действие.



Фигура 21

Примери за *Модели за изглед*:

```
public class CompetitionAddViewModel
{
    [Required(ErrorMessage = "Полето Име е задължително.")]
    4 references
    public string Name { get; set; } = null!;

    [Required(ErrorMessage = "Полето Описание е задължително.")]
    4 references
    public string Description { get; set; } = null!;

    [Required(ErrorMessage = "Полето Срок за регистрация е задължително.")]
    [DataType(DataType.Date)]
    4 references
    public DateTime RegistrationDeadline { get; set; }

    [Required(ErrorMessage = "Полето Дата на Провеждане е задължително.")]
    [DataType(DataType.Date)]
    3 references
    public DateTime DateOfConduct { get; set; }

    [Required(ErrorMessage = "Полето Местоположение е задължително.")]
    4 references
    public string Location { get; set; } = null!;

    [Required(ErrorMessage = "Полето Максимален брой участници е задължително.")]
    [Range(1, int.MaxValue, ErrorMessage = "Максималният брой участници трябва да бъде по-голям от 0.")]
    4 references
    public int MaxParticipants { get; set; }

    [Required(ErrorMessage = "Полето Предмет е задължително.")]
    3 references
    public Guid SubjectId { get; set; }

    2 references
    public List<SubjectViewModel>? Subjects { get; set; } = null!;

    [Required(ErrorMessage = "Полето Тип състезание е задължително.")]
    3 references
    public Guid CompetitionTypeId { get; set; }
}
```

Този *Модел за изглед* представлява всички данни и техните верификации нужни на приложението да създаде ново състезание.

Пример за допълнителни полета в *Изгледен модел*:

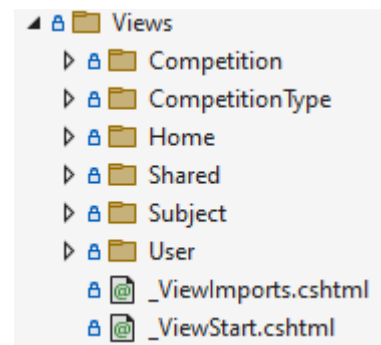
```
public class CompetitionTypeViewModel
{
    [Key]
    7 references
    public Guid Id { get; set; }
    [MinLength(1)]
    [MaxLength(50)]
    11 references
    public string Type { get; set; } = null!;

    6 references
    public List<CompetitionTypeViewModel>? AllTypes { get; set; }
}
```

Тук допълнителното поле е листа, държащ всички типове. Той е нужен за да може на една и съща страница да можем да видим всички типове състезания, както и да можем да добавяме нови, което изисква други полета. Така позволяваме на програмата в една страница да обяви няколко функционалности. Опция е също да *направим Обединителен модел* в който са вкарани като обекти два други *Модела за изглед*.

- Папка Изгледи (Views)

Папката *Изгледи* отговаря за съхраняването на изгледите аз всеки метод, който ги изисква. В случая на моето приложение изгледите са разделени на 6 под-папки (*Фигура 22*). По една за всеки *контролер* и една за споделени *изгледи*. Всеки *изглед* трябва да носи името на метода в *контролера*, който ще го вика, за да може да бъде намерен правилно. Папката съответно трябва да носи името на *контролера*.



Фигура 22

Пример за изглед:

```
@using KursovaRabota.ViewModels.CompetitionTypeVMs
@model CompetitionTypeViewModel

<form asp-controller="CompetitionType" asp-action="Add" method="post">
  <div class="form-group">
    <label asp-for="Type">Буг състезание</label>
    <input asp-for="Type" class="form-control" />
    <span asp-validation-for="Type" class="text-danger"></span>
  </div>

  <button type="submit" class="btn btn-primary">Добаи мун</button>
</form>

<form style="display:flex; flex-direction:row; justify-content: space-between; align-content: center" method="get" asp-action="Add" asp-controller="CompetitionType">
  <div style="display:flex; flex-direction: column" class="form-group">
    <label asp-for="@Model.Name">Филмпураи по име:</label>
    <input asp-for="@Model.Name" class="form-control" />
    <span asp-validation-for="@Model.Name" class="text-danger"></span>
  </div>

  <button style="max-height: 50px" class="btn btn-primary" type="submit">Филмпураи</button>
</form>

<div style="display:flex; justify-content:space-evenly; flex-wrap:wrap">
  @if (Model.AllTypes != null)
  {
    @for (int i = 1; i <= Model.AllTypes.Count; i++)
    {
      <div class="card" style="display:flex ;flex-direction:column; justify-content:space-around; width: 350px; height: 100px ">
        <label>@i: @Model.AllTypes[i-1].Type</label>
        <div style="display:flex; flex-direction:row; justify-content:space-evenly">
          <a class="btn btn-primary" asp-action="Update" asp-controller="CompetitionType" asp-route-id="@Model.AllTypes[i-1].Id">Промену</a>
          <button onclick="confirmDelete('@Model.AllTypes[i-1].Id')" class="btn btn-danger">Изтриване</button>
        </div>
      </div>
    }
  }
</div>

<script>
function confirmDelete(id) {
  var result = confirm("Сигурни ли сме, че искаме да изтриеме твоя състезание?");

  if (result) {
    window.location.href = '/CompetitionType/Delete?id=' + id;
  }
}
</script>
```

Този *изглед* отговаря за показването на всички видове състезания, тяхното добавяне и филтриране. В началото виждаме кой модел се използва. В самия код забелязваме наличието на **C#** код обяснен в [Програмни езици](#). В края на кода имаме прост **JavaScript**, отговарящ за предупреждаване преди изтриване.

- appsettings.js

Файлът **appsettings.js** отговаря за конфигурацията на връзката с базата. Никога не трябва да споделяме този файл или да го качваме някъде където е достъпен, защото в някои случаи той държи потребител, парола и **ip**-адрес на **SQL** сървър.

Пример за **appsettings.js**:

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=****; Database=****; User Id=****; Password=*****;
TrustServerCertificate=True;"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

▪ Program.cs

Файлът **Program.cs** отговаря за стартирането и конфигурирането на цялото приложение. Той държи в себе си всичко нужно за стартиране и конфигуриране на проекта при пускане

Примерна част от **Program.cs**:

```
public class Program
{
    0 references
    public static void Main(string[] args)
    {
        var builder = WebApplication.CreateBuilder(args);

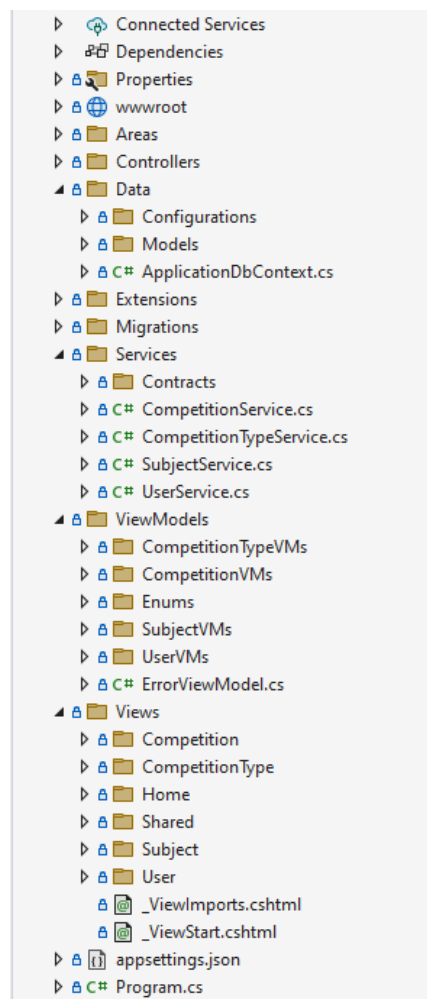
        // Add services to the container.
        var connectionString = builder.Configuration.GetConnectionString("DefaultConnection") ?? throw new InvalidOperationException("Connection string 'DefaultConnection' not found.");
        builder.Services.AddDbContext<ApplicationDbContext>(options =>
            options.UseSqlServer(connectionString));
        builder.Services.AddDatabaseDeveloperPageExceptionFilter();

        builder.Services.AddDefaultIdentity<ApplicationUser>(options => options.SignIn.RequireConfirmedAccount = false)
            .AddRoles<IdentityRole>()
            .AddEntityFrameworkStores<ApplicationDbContext>();
        builder.Services.AddControllersWithViews();

        builder.Services.AddScoped<IUserService, UserService>();
        builder.Services.AddScoped<ISubjectService, SubjectService>();
        builder.Services.AddScoped<ICompetitionTypeService, CompetitionTypeService>();
        builder.Services.AddScoped<ICompetitionService, CompetitionService>();

        var app = builder.Build();
```

Тази част от файла отговаря за инициализирането на всички услуги, които ще ползва приложението.

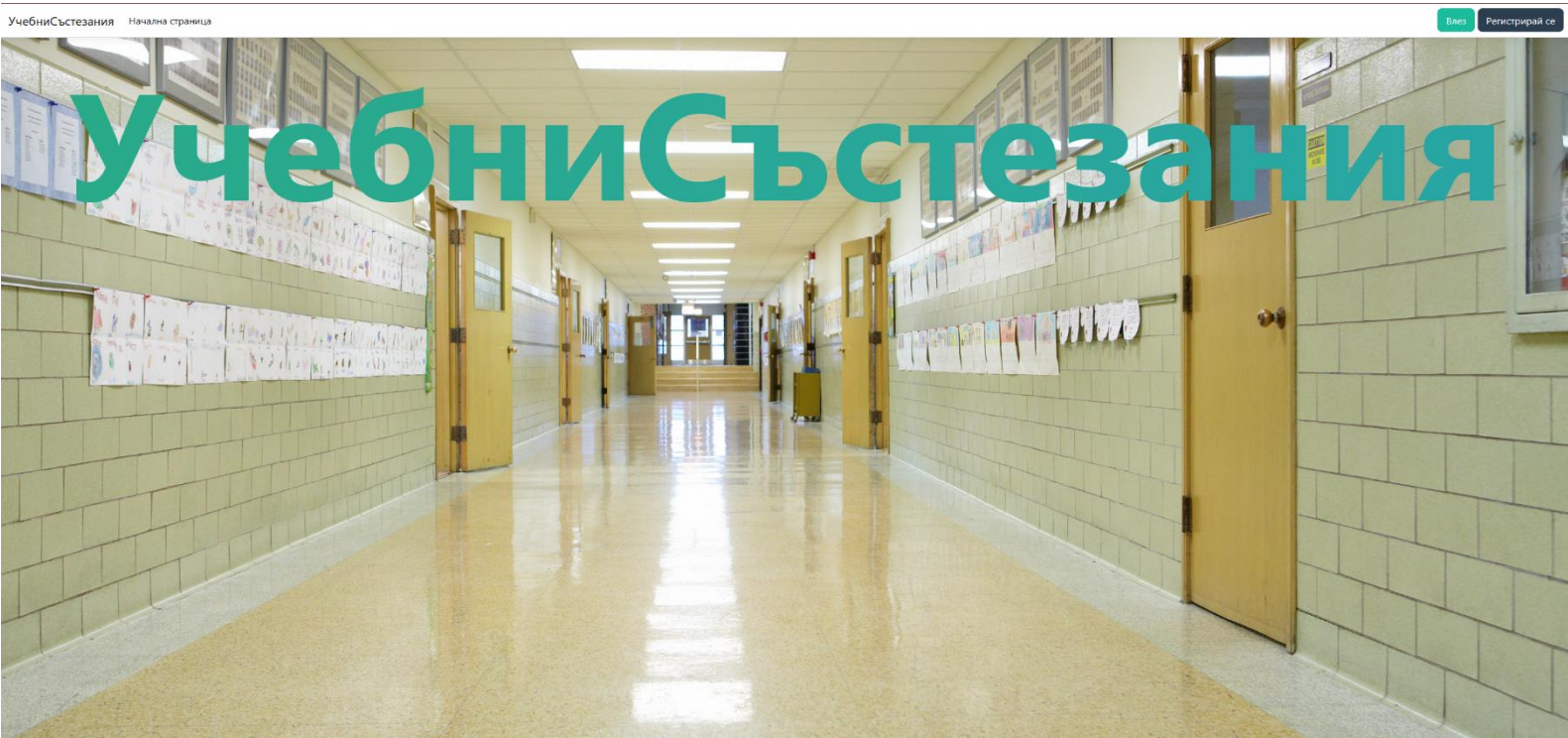


Фигура 23

5. Как да използваме

- Първи стъпки

Влизайки в сайта потребителя е посрещнат с начална страница (*Фигура 24*), на която единствените му опции са да влезе в акаунта си или да се регистрира.

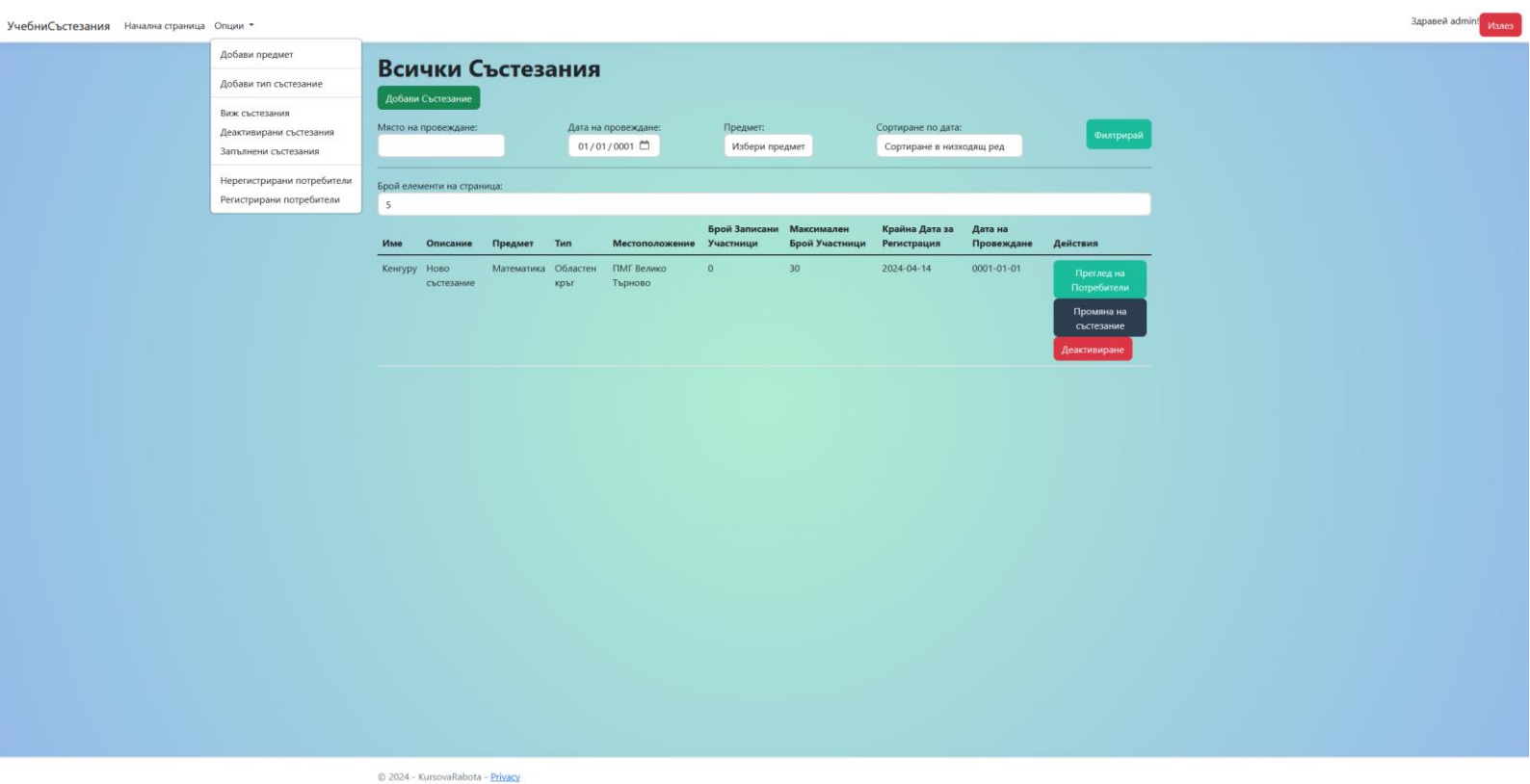


© 2024 - KursovaRabota - Privacy

Фигура 24

- Начална страница
 - Началната страница (*Фигура 25*) за всички потребители показва всички състезания налични в системата. В нейната навигационна лента опциите за всяка роля са различни.
 - Ученикът може да вижда само всички състезания в системата, които са активни и всички състезания за които се е записал. Учителя има достъп до бутона, който добавя нови състезания на началната страница, освен този бутон той има достъп до страниците: **Неактивни състезания**, **Запълнени състезания**, **Добави предмет**, **Добави тип състезания**.
 - Администраторът има достъп до всяка страница в системата, като единствените различни от тези на учителя са: **Регистрирани потребители** и **Нерегистрирани потребители**.

Началната страница за администратора:



Фигура 25

На страницата имаме опции за филтриране по място на провеждане, дата на провеждане, предмет и сортиране по дата на провеждане. Филтрацията е аналогична за всички страници свързани със състезанията. За всяко състезание администратора и учителя имат опциите да извадят статистика за учениците, да променят състезанието и да го деактивират, като тогава то ще се появи на страницата с деактивирани състезания.

Ученикът има само един бутон и той е записване за състезание. Също така той няма бутон за добавяне на състезание.

- Страница за добавяне на учебни предмети

Страницата за добавяне на учебни предмети (*Фигура 26*) е достъпна само за администратора и учителите. На нея има опция за добавяне на нов предмет, Филтриране по име на предмет, Промяна и изтриване на вече съществуващи предмети.

Учебни Състезания Начална страница Опции ▾

Здравей admin! Излез

Учебен Предмет

Добави предмет

Филтрирай по Име:

Филтрирай

1: Математика

Промени

Изтриване

2: География

Промени

Изтриване

3: История

Промени

Изтриване

4: Български език

Промени

Изтриване

- Страница за добавяне на типове състезания

Страницата за типове състезания (Фигура 27) е аналогична на тази за учебни предмети.



© 2024 - KursovaRabota - [Privacy](#)

Фигура 27

- Страници за Състезания

Всички страници за състезания са аналогични като изглед като само **GetAll** страницата има бутон за добавяне.

Страница: Всички състезания за ученика (Фигура 28)

УчебниСъстезания

Начална страница

Ученически опции

Зареей student

Излез

Всички състезания

Мои състезания

Добави Състезание

Място на провеждане:

Дата на провеждане:

01 / 01 / 0001

Предмет:

Избери предмет

Сортиране по дата:

Сортиране в низходящ ред

Филтрирай

Брой елементи на страница:

5

Име	Описание	Предмет	Тип	Местоположение	Брой Записани Участници	Максимален Брой Участници	Крайна Дата за Регистрация	Дата на Провеждане	Действия
Кенгуру	Ново състезание	Математика	Областен кръг	ПМГ Велико Търново	0	30	2024-04-14	0001-01-01	Записване

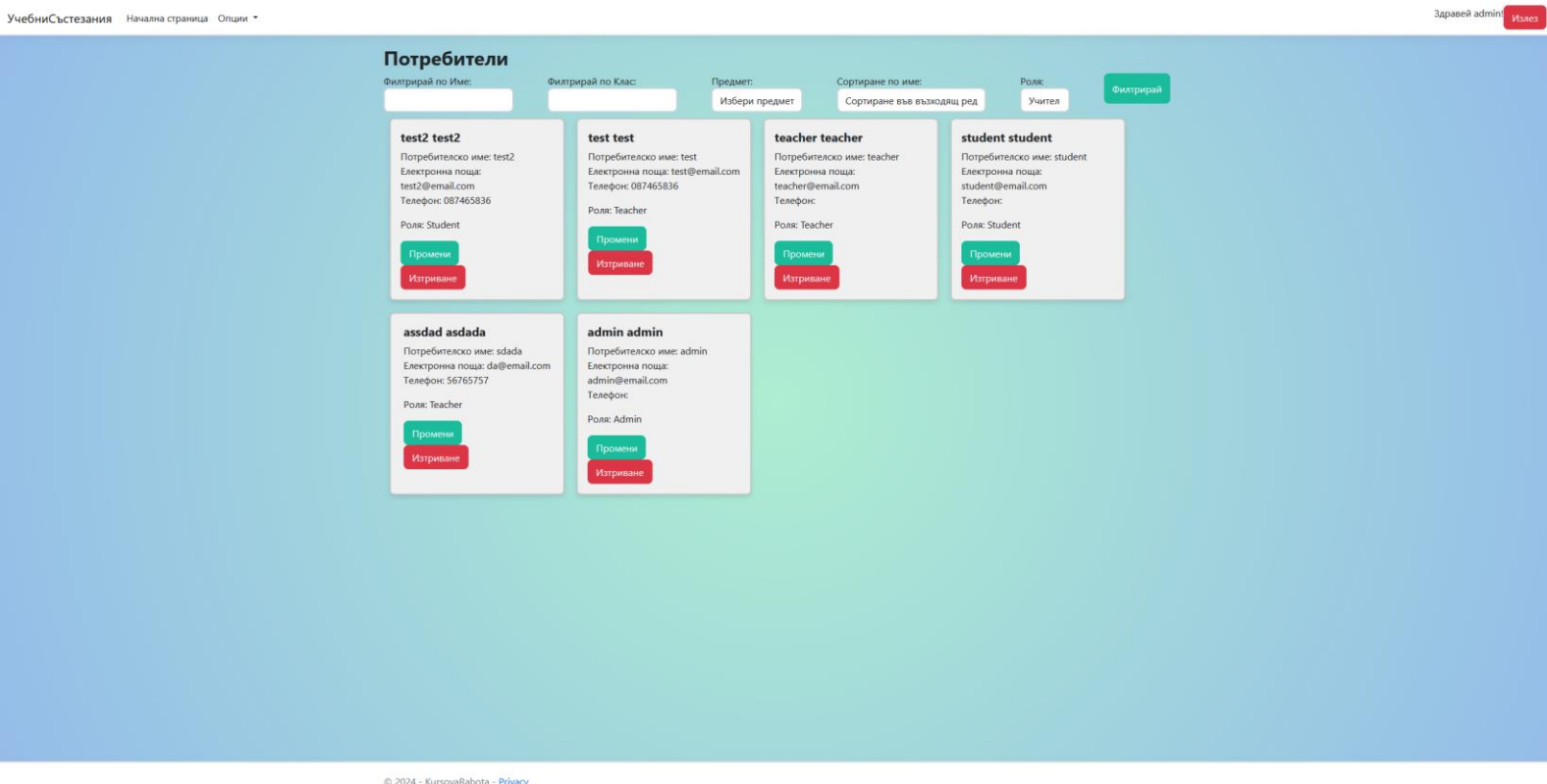
© 2024 - KursovaRabota - Privacy

Фигура 28

- Страници за Потребители

Страниците за потребители са две: **Регистрирани потребители** (Фигура 29) и **Нерегистрирани потребители** (Фигура 30). На страницата **Регистрирани потребители** администратора може да види, промени и изтрие потребителите на системата. Той може да ги сортира по име (възходящ и низходящ ред) и да ги филтрира по Име (независимо

дали цяло, първо или фамилно), клас, предмет по който преподават (ако са учители) и по роля (ученици и учители).



Фигура 29

На двете страници в картичките на потребителите е представена само най-важната информация за потребителя и неговата роля.

Страницата за **Нерегистрирани потребители** (Фигура 30) администратора има същите опции за сортиране и филтрация, като единствената разлика е, че може само да Одобри и Изтрие потребителите.

Нерегистрирани потребители

Филтрирай по Име:	Филтрирай по Клас:	Предмет:	Сортиране по име:	Роля:	Филтрирай
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	
<div>asdasd asdasdad</div> <div>Потребителско име: aDADASDA</div> <div>Електронна поща: admAsASin@email.com</div> <div>Телефон: 9789799789</div> <div>Желана роля: Teacher</div> <div>Одобри</div> <div>Изтриване</div>					

Фигура 30

6. Заключение

В заключение Дипломния проект „СИСТЕМА ЗА ОРГАНИЗИРАНЕ НА УЧЕНИЧЕСКИ СЪСТЕЗАНИЯ“ показва всички основни технологии, нужни за създаване на напълно функционален уеб-сайт следвайки всички конвенции и добри практики свързани с езиците **C#**, **HTML**, **CSS** и **SQL**.

Приложението цели да улесни работата на учители свързана с организация на учебни състезания, да им позволи да ги филтрират и сортират, също както да виждат статистика за учениците записващи се за тях. Програмата също цели да улесни учениците в намирането на подходящите за тях състезания олимпиади и т.н.

Приложението е създадено по технологиите изучавани в паралелката „Приложен Програмист“ в ПМГ Васил Друмев от 8ми до 12ти клас и е напълно функционална и готова за употреба с малки мащаби.