**МАТЕМАТИЧЕСКА ГИМНАЗИЯ „АКАДЕМИК КИРИЛ ПОПОВ”**

ДИПЛОМНА РАБОТА

**ЗА ДЪРЖАВЕН ИЗПИТ ЗА ПРИДОБИВАНЕ НА ТРЕТА СТЕПЕН НА ПРОФЕСИОНАЛНА КВАЛИФИКАЦИЯ**

НА ТЕМА

**МУЗИКАЛЕН КОНКУРС „ЗВЕЗДИТЕ НА БЪЛГАРИЯ”**

Дипломант Научен ръководител

Валентин Христов Танев Росен Димитров Вълчев

Ученик от 12ж клас Учител по информатика и ит

гр. Пловдив 2022

# СЪДЪРЖАНИЕ

1. [Увод 1](#_bookmark0)
   1. [Цел и задача 2](#_bookmark1)
2. [Използвани технологии 3](#_bookmark2)
   1. [C#....... 3](#_bookmark3)
   2. [JavaScript 5](#_bookmark6)
   3. [Visual Studio 2022 5](#_bookmark7)
      1. [CLR 6](#_bookmark8)
      2. [.NET Garbage Collection 6](#_bookmark9)
   4. [ASP.NET Core 7](#_bookmark10)
      1. [Заверка 7](#_bookmark11)
      2. [Razor engine 7](#_bookmark12)
      3. [Документация 7](#_bookmark13)
      4. [Сигурност 7](#_bookmark14)
      5. [ASP.NET Code Generators 8](#_bookmark15)
      6. [Мигриращи база данни 8](#_bookmark16)
   5. [HTML5 & Css3 8](#_bookmark17)
   6. [Bootstrap 9](#_bookmark18)
   7. [Entity Framework Core 9](#_bookmark19)0
   8. [Microsoft SQL Server 10](#_bookmark20)
   9. [Какво е REST 10](#_bookmark21)
      1. [Клиент Сървър 11](#_bookmark22)
      2. [Уеб браузъра 11](#_bookmark23)
      3. [DNS 11](#_bookmark24)
      4. [JSON 12](#_bookmark26)
   10. [Какво е MVC 12](#_bookmark27)
3. [Архитектура 14](#_bookmark29)
   1. [Изглед за ползване 14](#_bookmark30)
   2. [Базата данни 15](#_bookmark32)
      1. [Таблица Asp.NetUsers 16](#_bookmark34)
      2. [Таблица Judges 17](#_bookmark35)
      3. [Таблица Asp.NetRoles 18](#_bookmark36)
      4. [Таблица Asp.NetUserRoles 18](#_bookmark37)
4. [Разработка 20](#_bookmark38)
   1. [Важни помощни класовe 20](#_bookmark39)
      1. [HomeController 20](#_bookmark40)
      2. [DbContext 20](#_bookmark42)
      3. [ServiceCollectionExtentions 21](#_bookmark44)
   2. [Изгледи 22](#_bookmark47)
      1. [Index 22](#_bookmark48)
      2. [Profile 24](#_bookmark52)
      3. [Admin 24](#_bookmark53)
      4. [Contact Us 25](#_bookmark54)
      5. [Dashboard 25](#_bookmark55)
      6. [Judges 26](#_bookmark57)
      7. [Login 26](#_bookmark58)
      8. [Register 26](#_bookmark58)
   3. [Папката Data 30](#_bookmark68)
      1. [Проектът MusicX.Data 31](my.docx)
      2. [Проектът MusicX.Data.Common 31](#_bookmark72)
      3. [Проектът MusicX.Data.Models 32](#_bookmark74)
   4. [Папката Services 30](#_bookmark68)
      1. [Проектът MusicX.Services 31](#_bookmark70)
      2. [Проектът MusicX.Services.Data 31](#_bookmark72)
      3. Проектът [MusicX.Services.Mapping 32](#_bookmark74)
      4. Проектът [MusicX.Services.Models 33](#_bookmark76)
   5. [Папката Web 30](#_bookmark68)
      1. [Проектът MusicX.Web 31](#_bookmark70)
      2. [Проектът MusicX.Web.Infrastructure 31](#_bookmark72)
      3. [Проектът MusicX.Web.BindingModels 32](#_bookmark74)
      4. Проектът [MusicX.Web.ViewModels 33](#_bookmark76)
   6. [Папката Common 30](#_bookmark68)
5. [Заключение 51](#_bookmark134)
6. [Използвана литература 53](#_bookmark135)

[Фигура 1 Графика на най-използваните програмни езици 4](#_bookmark4)

[Фигура 3 MVC 13](#_bookmark28)

[Фигура 5 Диаграма на базата данни 16](#_bookmark33)

# Увод

В съвременният свят новите технологии се развиват много бързо. Повече и повече системи се автоматизират и все повече и повече мироприятия се състоят онлайн благодарение на интернет и на съвременните приложения.

Преди…..

В днешно време обаче ….

За разработка на моята платформа за провеждане на музикален конкурс, която съм нарекъл MusicX съм използвал MVC теплейта на технологията ASP.NET Core.

## Цел и задача

Приложението има за цел да направи провеждането на музикално състезание напълно онлайн като всеки има правото да участва и едновременно публиката участва в оценяването на кандидатите едновременно и журито. MusicX улеснява значително организаторите и спестява значително време на участниците и на журито, тъй като самата концепция и идея на сайта е, че журито ще оценява само десетте участника в конкурса със събрани най-много харесвания, което прави съдийския избор много по лесен. Решението да се направи и уеб сайт е за да може и хора, които не ползват телефона си да правят снимки, да могат да качват снимки и за да може да се разглеждат и от хора, които не снимат. Трябва бавно да се премине от дигиталните камери към smart телефон-ските камери.

MusicX позволява

* + - Лесно и бързо присъединяване в музикалния конкурс
    - Създаване на профил чрез който да могат да подкрепят фаворита си
    - Разглеждат представянията на всички участници
    - Задават въпроси по всяко време на страницата Contact us
    - Да се превърне от обикновен потребител в състезател

Така хората ще могат да изразяват мнението си чрез харесване, което ще доведе до участието им под някаква форма в самия конкурс и ще увеличи способността им да влиаят на резултатите от него, а не както при типичните състезания да има монопол и журито своеволно да казва кой е най-добър без да се допитва до масите хора. Всичко това би довело до повече интерес от публиката към мироприятието, защото всеки ще бъде по ангажиран и ще следи какво се случва с фаворита му дори би могъл да агетира свои приятели също да го подкрепят, което е директен маркетинг за проекта.

Въпреки свободата която позволява концепцията на сайта, той си има администраторски акаунти, които имат възможността да променят създават и изтриват данните в него, и да го цензурира при нужда.

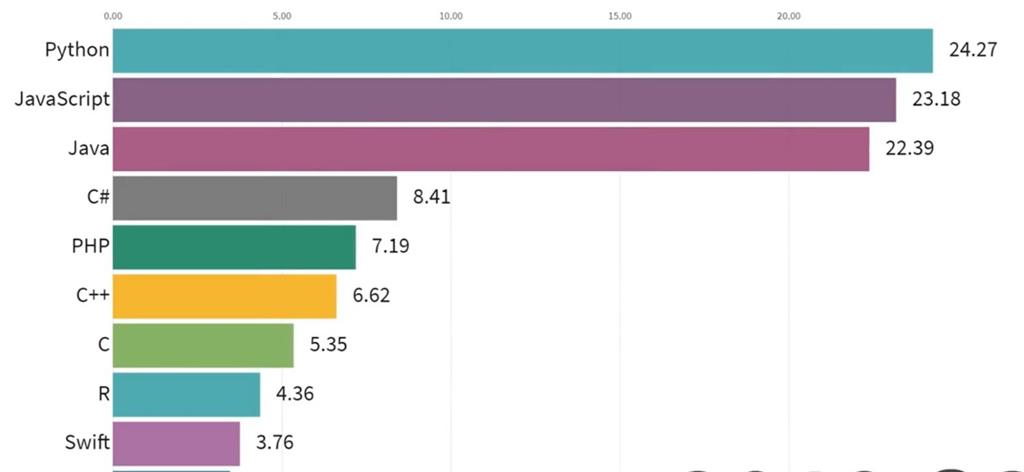
# Използвани технологии

## C#

**C#** (***C Sharp***, произнася се *Си Шарп*) е обектно ориентиран език за програмиране, разработен от Microsoft като част от софтуерната платформа .NET. Стремежът още при създаването на C# езика е бил да се създаде прост, модерен, обектно ориентиран език с общо предназначение. Основа за C# са C++, Java и донякъде езици като Delphi, VB.NET и C. Той е проектиран да балансира мощност (C++) с възможност за бързо разработване (Visual Basic и Java). Те представляват съвкупност от дефиниции на класове, които съдържат в себе си методи, а в методите е разположена програмната логика – инструкциите, които компютърът изпълнява. Програмите на C# представляват един или няколко файла с разширение .cs, в които се съдържат дефиниции на класове и други типове. Тези файлове се компилират от компилатора на C# до изпълним код и в резултат се получават асемблита – файлове със същото име, но с различно разширение (.exe или .dll).

Интересно нещо за C# и .NET е, че след като се компилира кода винаги се произвежда и dll. Файл, който съдържа код на така наречения IL език и целта е да няма значение дали даден dll. Файл да бъде използван освен всички езици на Mincrosoft като се включи в някой проект. Или иначе казано за да има language interoperability на английски.

Intermediate Language е език от **ниско** ниво със **стекова** архитектура, който отговаря за:  
  
  - Инструкции за заделяне на памет;  
  - Инструкции за предизвикване и обработка на изключения;  
  - Инструкция за извикване на виртуален метод;  
  - Позволява ефективно компилиране до машинен код за различни платформи;  
  - Винаги се компилира преди изпълнение.



*Фигура 1 Графика на най-използваните програмни езици*

През недалечната 2019 година е направено проучване на тема най-използвани и разпространени програмни езици, на което ясно можем да видим че двата главни езика, използвани в проекта MusicX са именно едни от най-модерните и съвременни езици.

Версията на C# използвана в проекта е C# 10 и съответно версията на рамката e .NET CORE 6, като и двете версии са най-актуалните, съответно с най-много добри практики имплементирани в тях, най-бързи и най-ефективни.

## JavaScript

JavaScript е един от най-използваните скриптови езици за програмиране в Интернет. Разработен е през 1995 година от Брендан Айк и осигурява моментното и динамично взаимодействие между потребител и браузър. Вярно е, че извървява дълъг път, но днес смело може да  заявим, че почти няма нещо, което JavaScript не може да направи. Благодарение на него уеб страниците, които използваме в момента, изпълняват много повече функции от това просто да зареждат данни.

Последните проучвания за 2017 показват, че 94,9% от всички уебсайтове в мрежата използват този език за програмиране. Най-популярните сред които са Google, Yahoo, Amazon, Facebook, Wikipedia и т.н..  Защо? Защото JavaScript позволява създаването на интерактивни и адаптивни интерфейси с много добър UI дизайн и динамични функционалности, изцяло с насоченост към крайния потребител.

JavaScript и Java нямат почти нищо общо по между си, освен че се използват за Обектно-ориентирано програмиране (ООП), имат близък синтаксис и сходни имена. Не мислим да задълбаваме повече по темата. Просто ще подчертаем  накратко, че:

Работят с различни плъгини

JavaScript е скриптов език, а Java не е

Функциите на JavaScript са се разширили значително през годините. Първоначално езикът просто е регулирал начинa, по който потребителят си взаимодейства с уебсайта: онлайн въпросници и форми за контакт, подаване на обратна връзка под формата на известия и важни съобщения и др. Днес обаче езикът е еволюирал до такава степен, че с него може дори да управлявате дрон. За какво още се използва JavaScript:

Създаване на мобилни приложения

Разработване на десктоп и уеб апликации

Зареждане на информация чрез AJAX

Изработване на операционни системи за смарт часовници

Разпознаване на използвания браузър и променяне на уеб страниците спрямо това

Анимиране на елементи от уеб страниците

Създаване на поп-ъп прозорци

Изготвяне на презентации под формата на уебсайт

Периодично обновяване на променливи данни без намесата на потребителя

Функционира дори и офлайн

и още, и още

## В заключение…

Предимствата на JavaScript са много. Именно те го превръщат в един от любимите езици за програмиране и на нашите девелъпъри.. Освен че разширява функционалността на уеб страниците, той е изключително лесен език за научаване и практикуване. Насочен е изцяло към осигуряване на добро потребителско преживяване, при това бързо и качествено.

Вижте и другите програмни езици, които използваме ежедневно в работата си и ако имате нужда от помощ, ние сме насреща!

## Visual Studio 2022

**Microsoft Visual Studio** е мощна интегрирана среда за разработка (на английски: *integrated development environment*, *IDE*) на софтуерни приложения за Windows и за платформата .NET Framework. Използва се за разработка на конзолни и графични потребителски интерфейс приложения, както и Windows Forms или WPF приложения, уеб сайтове, уеб приложения и уеб услуги на всички поддържани платформи от Microsoft Windows. Visual Studio предоставя мощна интегрирана среда за писане на код, компилиране, изпълнение, дебъгване (както за високо така и за машинно ниво), тестване на приложения, дизайн на потребителски интерфейс (форми, диалози, уеб страници, визуални контроли и други), моделиране на данни, моделиране на класове, изпълнение на тестове, пакетиране на приложения и стотици други функции.

* + 1. CLR

Tова е **програмната** **среда**, в която се **изпълнява** **кодът** **на** **.NET** **приложенията** (C#, F# и VB). Представлява **виртуална** **машина**, която компилира междинния код CIL (Common Intermediate Language) за конкретната хардуерна платформа и операционна система, с която работи потребителя. Използва се компилация по време на изпълнение или така нареченият Just-In-Time compiler.

**Задълженията** на CLR са няколко:  
  - Изпълнението на IL кода и JIT компилацията;  
  - Паметта и ресурсите на приложението;  
  - Безопасността на типовете;  
  - Сигурността;  
  - Code access security;  
  - Role-based security;  
  - Изключенията;  
  - Конкурентността;  
  - Връзката с неуправляван код;  
  - Процесите на debug и оптимизиране (profiling) при разработка на приложения.

* + 1. .NET Garbage Collection

Този събирател на боклуци едно от най-фините неща в .NET света, който го кара да бъде безспорен фаворит сред много програмисти особено новобранците пред технологии като C и C++. Той позволява на разработчиците да създават нови обекти без да се притесняват за заемане и освобождаване на памет. Това позволява да се пишат програми по-бързо и с по-малко грешки, като се елиминира изпускането на паметта или други проблеми с паметта. На теория това звучи добре, но на практика събирателя на боклуци си е вършил работата прекалено добре, което е довело до загуба на производителност.

“Събирането на боклуци” работи като операционната система, която дава на програмата памет, наричана heap и позволява по-бързото създаване на обекти. Всеки обект се следи дали е жив и дали той може да бъде достигнат. Ако не той се задрасква като боклук и се освобождава памет за нови обекти.

Следенето на това кои обекти все още се използват и изчисляването на това кога са се натрупали достатъчно много обекти за изчистване, че да си струва да пуснем този garbage collector да ни почисти боклука, така че да не забавим излишно програмата, са все неща които са програмирани изключително добре от разработчиците на технологичния гигант Microsoft и същевременно са ни предоставени да използваме напълно безплатно, което е огромно предимство на цялата рамка .NET

## ASP.NET Core

ASP.NET Core е безплатна софтуерна рамка за уеб разработка, с отворен код. Също така тя се явява и следващата стъпка в еволюцията ASP.NET. Тя е разработена съвместно от Microsoft и общността, която е събрала през годините на своето развитие. ASP.NET Core е модуларна софтуерна рамка, която може да върви както на пълната .NET рамка, така и на крос-платформената .NET Core. Въпреки, че е нова софтуерна рамка, изградена върху нов web stack, тя има висока степен на съвместимост с ASP.NET MVC.  
  
Спрямо своите предшественици ASP.NET Core поддържа нова функция – т.нар. „side by side versioning”. При нея различни приложения, които използват една и съща машина, могат да таргетират различни версии на ASP.NET Core, в зависимост от версиите (и нуждите) си. Това не е възможно с по-стари издания на ASP.NET.

* + 1. Заверка

Най-важното нещо, което всеки уеб сайт трябва да има, е заверка на потребителите. Писането на заверка отнема много време и трябват знания като криптиране на данни и управление на ресурси. ASP.NET Core има готов дизайн, който е имплементиран в рамката в лицето на комплекта от пакети и библиотеки наричан ASP.NET CORE Identity. Така всеки лесно с извикването на няколко реда код има една от най-добрите заверки и така се намалява времето за писане на нова заверка, която вече се използва в много уеб сайтове.

* + 1. Razor templating engine

Razor двигателят за темплейти предоставя лесен синтаксис, който ни помага да пишем HTML и код от страна на сървъра в уеб страници, използвайки C#. Важно е да се упомене, че това е език за маркиране от страна на сървъра, а не е език за програмиране.

Razor двигателят е разработен от Microsoft и е автоматично интегриран във ASP.NET Core, но това не го лимитира откъм това да бъде използван и от други платформи и езици за програмиране. Той е изключително гъвкав и удобен за ползване, като лично според мен е най-добрият сред конкурентите си.

Razor синтаксиса е компактен, което минимизира символите, които трябва да се използват, но също така е лесен за научаване.

.

* + 1. Документация

Ако имаш най-добрата рамка създадена някога, но няма добра документация и уроци за нея, никой няма да може да я ползва, затова от Microsoft са се погрижили да ни снабдят със страхотна документация за всички техни софтуерни продукти. Също таке е пълна с безплатни учебни видеа. Уроците са кратки, лесни за разбиране и точни, което прави ученето как се използва платформата още по лесно и забавно. Тук искам да вметна, че невинаги е било така и до преди няколко години документацията на Microsoft не се славеше с качеството си, за това всички програмисти използващи техните технологии страшно много оценяваме техните усилия и сме горди от състоянието на документацията в момента, защото знаем колко труд са хвърлили нашите колеги да я създадат но също много добре знаем колко много ни помага всеки ден и сме безкрайно благодарни.

* + 1. Сигурност

Сигурността е едно от най-важните, което всяко приложение трябва да има, независимо дали то е уеб страница, Android приложение или приложение за настолен компютър. ASP.NET CORE Identity използва криптирани пароли и никога не ги запаметява като обикновен текст. Използва сложни алгоритми като PBKDF2 в комбинация със HMAC-SHA256 алгоритъм за криптиране както и SQL изявления, за защита от инжектиращи атаки. Също така проверява дали потребителите не въвеждат тагове, които могат да счупят приложенията.

* + 1. ASP.NET Core Code Generation

ASP.NET Core предоставя ASP.NET Core Identity като Razor Class Library. Което значи, че когато я ползваш ти автоматично имаш достъп до всички публични неща в нея и ги използваш на готово. Но има случаи, в които искаш да помениш някои от тези неща. В такъв случай използваме code generation и си генерираме този код който бихме желали да подменим по някакъв начин при нас, за да имаме пълен достъп до него и да го подменяме по наша угода. Приложенията, които включват Identity, могат да прилагат генериране на код или на английски техниката scaffolding, за да добавят избирателно изходния код, съдържащ се в библиотеката на Identity.. Например, можете да инструктирате инструмента да генерира кода, използван при регистрацията, което е доста често срещано в практиката.

* + 1. Мигриращи база данни

Миграциите помагат за лесно създаване и модифициране на структурата на базата от данни без да се налага тя да бъде създадена наново. По този начин, системата проверява дали кода няма грешки преди да бъде променена базата данни. С миграциите може да се проследи лесно къде и кога са правени промени. Entity Framework има възможност да работи с широк набор от релационни бази данни и е доста гъвкав в това отношение.

HTML5 & CSS3

**HTML (HyperText Markup Language)** е най-популярният език за изработка и описание на уеб страници.

**HTML** е стандарт в Интернет, а правилата се определят от World Wild Web Consortium (Международната организация по стандартизация).

Началото на **HTML** датира от навлизането на Световната мрежа през 1991 година.

Тогава Тим Бърнърс-Лий от CERN (Европейската организация за ядрени изследвания) въвежда първите уеб протоколи и инструменти в Интернет.

Изработването на **HTML** документа става с помощта на така наречените тагове.

Чрез тях се създават и се определя разположението на отделните елементи в уеб страницата като заглавия, таблици, форми, бутони, текстови полета, текст, изображения, видеа и др.

По своята същност **HTML** не е програмен език, а описание на това как искаме да изглежда съдържанието, на структурата на самото съдържание.

**HTML** е текст, които ние подаваме на браузъра, а той от своя страна го интерпретира като последователност от команди/инструкциите, които изпълнява и по този начин подрежда обектите в страницата.

CSS (Cascading Style Sheets) е език за описване на презентацията и стиловете на елементите в един HTML/XML документ. CSS е една от основните технологии, използвани в уеб, редом с HTML и JavaScript.

При създаване на HTML страница, съдържанието й се описва (маркира) с HTML код, а презентацията на това съдържание, тоест как ще изглежда то в браузъра, се описва с CSS код (стил).

CSS кодът се поставя във файл с разширение .css. След това този файл може да се използва във всяка една страница от уеб сайта. Така че стилът ще се намира на едно място и при промяна ще се отразява на всички уеб страници.

Уеб страница може да създадем като:

Нужни са ви само текстов редактор, с който да създадете **HTML** документа и браузър, с който да визуализирате съдържанието на този файл на екрана. Нямате нужда от наемане на сървър или запазване на домейн.

Създаването на **HTML** страница може да се извърши с помощта на текстов редактор (Notepad++) или инструменти (WYSIWYG), които не изискват да притежавате познания по **HTML**. Някои от тези програми са Word Microsoft, FrontPage, Macromedia Dreamweaver и други.

Основната функция на **HTML** е да определи структурата на документа от нейното съдържание като текст и мултимедиа. Със CSS и JavaScript вече може да създадем стилове на елементите от документа като: цвят, размер, яркост и много други.

## Volley

Bootstrap е безплатна, с отворен код и невероятна рамка за уеб разработка, съдържаща перфектната комбинация от шаблони за дизайн, базирани на CSS и HTML, които да се използват за форми, типография, навигации, бутони и други компоненти на интерфейса, както и разширения javaScript, които не са задължителни, Рамка за начално зареждане е разработена от хората в Twitter като част от хакатон, в който са участвали и се разглежда главно като проект на Туитър план. По-късно това бе преименувано на bootstrap и пуснато като проект с отворен код през август 2011 г. В началото на 2012 г. беше пусната нова версия на тази рамка, Bootstrap 2, която дойде заедно с много промени в по-ранната версия, само за да направи животът на уеб разработчиците много по-лесен. Тази версия поддържа отзивчив уеб дизайн, което означава, че статичните страници ще бъдат заменени с динамичното оформление, което ще се коригира при настройването на уеб страницата. След това дойде новата версия на Bootstrap 3, която преработи компонентите, за да използва мобилния първи подход и плосък дизайн. След това дойде Bootstrap 4 и след това Bootstrap 5 е най-новият.

Bootstrap е уеб рамка, която се фокусира върху опростяването на информативно разработване на уеб страници, като тези на уеб приложенията. Основната цел защо беше добавена към Bootstrap беше заради избора на размер, цвят, шрифт и оформление на проекта, който предоставя. Основният фактор е дали разработчиците ще харесат този избор и ще бъде ли лесно и достатъчно интересно да ги включат в своите проекти. След като те се добавят, те предоставят основна дефиниция на стила за HTML елементите. Крайният резултат е появата на таблици, проза и елементи от формите в уеб браузърите. Разработчиците имат и допълнителното предимство на CSS класовете, които са предоставени и дефинирани в Bootstrap, така че външният вид на съдържанието да бъде допълнително персонализиран. Има някои други функции, като цвят на таблиците на светли и тъмни нюанси, издърпване на цитати, заглавия на страници и текстове с акценти, което прави работата с обувка за начало толкова лесна. Bootstrap се предлага и с някои JavaScript функции, които работят под формата на плъгини Jquery. Предлагат се допълнителните елементи на потребителския интерфейс като съвет за инструменти, диалогови прозорци и въртележки. Всеки компонент се състои от HTML структура, някаква форма на JavaScript код и CSS декларации. Те разширяват функционалността на съществуващи елементи, свързани с интерфейса, като например функция за автоматично завършване, която да се използва за полета за въвеждане.

## Entity Framework Core

Oбектно-релационно картографиране или ORM на английски език, е техника за програмиране, която служи за трансформиране на данни, използвани в обектно-ориентирания програмен език или програма и релационната база данни (тип SQL) като персистенция. Това ще накара програмните стойности да създадат виртуална обектно-ориентирана база данни, която да съдържа данните, от които се нуждаете. Алтернативата пред използване на ORM технология са чистите SQL заявки, с които работата е много по-тромава, бавна, и уязвима за възникване на грешки или проблеми откъм сигурност

Entity Framework Core, която е стандарт за ORM в C# и .NET Core приложения. EF Core позволява mapping между релационна база и обектно-ориентиран модел чрез подходите "database first" и "code first" и предоставя мощно обектно-ориентирано API за заявки към базата данни и извършване на CRUD операции. EF core предоставя както допълнително ниво на абстракция, така и лесен начин за обработка на данните от базата.

Предшественикът на Entity Framework Core е ADO.NET, който вече е остаряла технология и повечето програмисти не я използват, ако имат възможност за избор.

Основният конкурент на Entity Framework Core е Dapper, който също е доста добра технология за работа с бази данни, предпочитана от много колеги, но аз лично никога не съм работил с нея, за това реших да работя с това, което разбирам и познавам.

## Microsoft SQL Server

**Microsoft SQL Server** е сървърна система за управление на бази от данни (и по-точно на релационни бази от данни) на компанията Microsoft. Microsoft SQL Server е предназначена за управление на големи сървърно базирани БД, за разлика от MS Access, която е desktop базирана и не е предназначена за управление на големи корпоративни бази данни.

Северноамериканската компания Microsoft представя първата версия на софтуерния продукт през 1988 г. От което можем да си направим извода, че е установен и сигурен софтуер, с минимално или никакво количество бъгове, добре поддържан от една от най-сериозните компании в света днес.

## Какво е REST

REST e архитектурен стил, който е стандарт между комуникирането на различни компютърни системи в мрежата, което позволява лесен начин на тези системи да комуникират една с друга. Тя цели да раздели директната комуникация между клиента и сървъра. Това разделяне означава, че кода на клиента може да се промени без това да засегне операцията на сървъра и обратно. Стига и двете страни да ползват един и същи формат на съобщенията, които си разменят.

Този стандарт позволява на различни клиенти да използват едни и същи действия и отговори. Тъй като сървъра не трябва да знае състоянието на клиента и обратно, те лесно могат да разбират съобщенията, които си разменят без да се знаят предишните направени действия. Това позволява на тези системи да работят по-бързо, по-надеждно и са високо мащабируеми.

Основните глаголи, които се използват са:

* Вземи – взима даден ресурс, който се търси по идентификатор или друга колекция от данни.
* Създай – създава нов ресурс
* Обнови – обновява даден ресурс по даден идентификатор
* Изтрий – изтрива даден ресурс по даден идентификатор
  + 1. Клиент Сървър

Това е много популярен модел в компютърните мрежи, който използва клиентските устройства и сървърите, всеки за специфични функции. Този модел се използва както в локалната така и в глобалната мрежа. Такива система включват уеб браузъра, уеб услугите, FTP клиенти и сървъри и DNS.

* + 1. Уеб браузъра

Уеб браузъра е компютърна програма, която се използва за отваряне на уеб сайтове или информация в една мрежа. Той пита сървъра за дадена страница и ако сървъра притежава тази страница я изпраща на уеб браузъра. Това е най-често HTML код, но може да има CSS, JavaScript, PHP, и др. езици. Следователно след като уеб браузъра получи информацията, той я визуализира в подходящ формат за четене от потребителя. За да работи всичко това, потребителя трябва да напише името на дадена страница или уеб сайт. Това се нарича URL, който съдържа в себе си HTTP протокола, който е стандартния протокол използван за обменяне на файлове през Интернет, в повечето случаи това са уеб страниците.

Най-популярните уеб браузъри са:

* Google Chrome
* Mozilla Firefox
* Microsoft Edge
* Opera

Благодарение на факта, че почти всеки телефон или компютър има така наречения софтуер браузър на себе си това означава, че може да визуализира и да отваря и работи успешно с уеб сайтове и уеб приложения, което пък от своя страна значи, че приложението MusicX може лесно да се използва независимо от конкретното устройство, което притежава потребителя.

* + 1. DNS

DNS е като телефонна книга на Интернета. Потребителите въвеждат името на сайта вместо IP адреса му. Това име се праща на локалният DNS сървър, който търси IP- то на въведеният уеб сайт и праща обратно IP-то на сървъра за да може уеб браузъра да комуникира с него. Всеки домейн в Интернет има уникален IP адрес. Много е по лесно за един човек да запомни “Youtube.com” вместо комбинацията 208.65.153.238.

* + 1. JSON

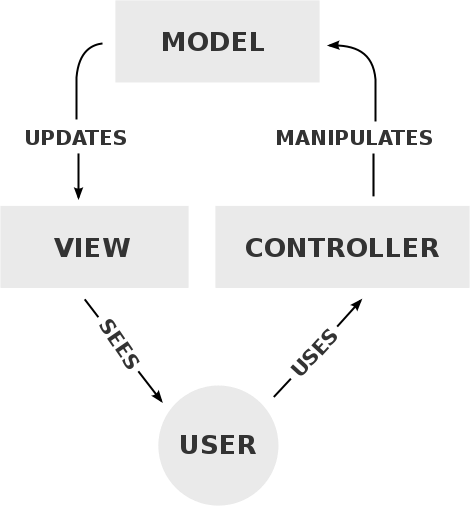
JSON, JavaScript ориентирана нотация, е формат за данни, които може да се четат от повечето езици, правейки го по известен от XML. Той се използва за транспорт и съхранение на данни, и е най-често срещан за комуникация между сървъра и браузъра. Макар че е базиран на JavaScript той е самостоятелен. Той стана много бързо известен, защото направи обмяната на данни лесно, лек е от към хардуерни изисквания и е лесно четим. JSON няма създател.

Много хора видяха потенциала за обмен на данни чрез използването на JavaScript литералите за обекти през 1996. Доугас Крокфорд, вижда потенциала на JSON и през 2002 регистрира домейна json.org. Чак през 2005, когато софтуерните и хардуерните разработчици са се нуждаели от начин за споделяне на данни между мобилни и уеб приложения, JSON става много популярен.

С JSON формата е много лесно да заредиш данните в дървовидна структура и да работиш с нея. Също данните използват и по-малко място в сравнение с XML, което увеличава скоростта на пращане и намалява времето, за което да се пращат съобщения напред и назад от сървърите.

* 1. Какво е MVC

MVC се превежда като “Модел, изглед и контролер” Това е архитектурен дизайн, който се използва за интерактивни и динамични уеб сайтове. Почти всеки сайт ползва този шаблон и много приложения също. Създаден през 1978 от Тръгви Ренскайг, целта му е да помогне на потребителя да манипулира и контролира компютърните система в по визуален начин. Представете си, че искате да си направите сандвич.



*Фигура 3 MVC*

Като поръчвате този сандвич на “Изгледа”, вие имате представа как ще изглежда той. Следователно “Изгледа” казва на “Контролера” да направи този сандвич. “Модела” взима всички необходими съставки и ги праща на “Изгледа”, който сглобява сандвича.

“Контролера” знае всички комбинации за различни сандвичи, които потребителя може да иска. Той праща поръчката на “Модела”, който знае всички продукти за създаването на сандвича, и най-накрая “Изгледа” го дава на потребителя.

При използване на системата, един потребител може да създава, приема, обновява и изтрива обекти от базата данни. Както винаги той не комуникира директно с базата данни, а изпраща искане, което се обработва.

# Архитектура

В тази част от документа ще се разгледа как системата работи. Ще задълбаем повече в кода, какви подходи са използвани, защо, какви са били алтернативите и ще ви запозная със по техническата част от проекта.

## Базата данни

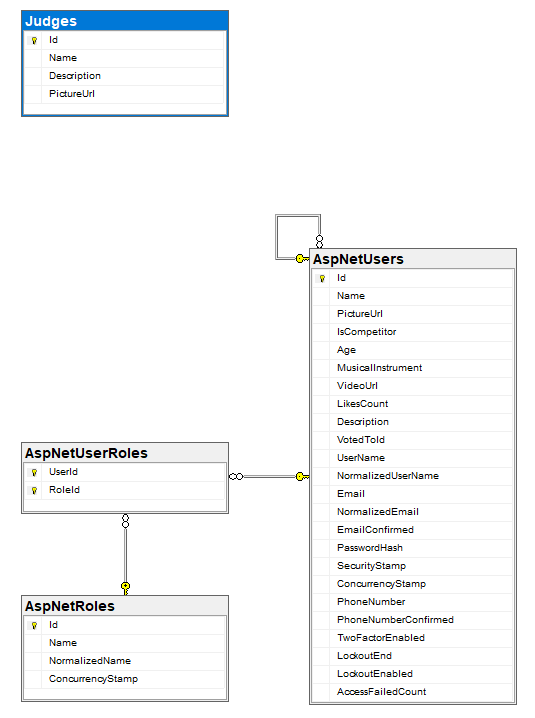
Базата данни която ще се използва ще е релационна. Създадена през 1970 от E.Ф. Кодд, той предлага да се промени метода за съхранение на данни от йерархичен модел към структура от таблици, всяка съдържаща редове и колони. Всяка таблица има уникален ключ, който идентифицира информацията в таблицата. Релацията между две таблици става чрез използването на вторичен ключ. Този ключ сочи към първичният ключ в другата таблица. Този вид база данни е доказан, че работи точно, без повторения на данни, гъвкав и сигурен е и е подходяща за нашият вид приложение.

Подходът, който е използван при изграждането на базата данни е Code First, което означава,

Базата данни ще има 9 таблици, от които повечето са служебни за ASP.NET Core Identity тоест то има нужда от тях за да работи правилно, то си ги създава, то си ги менажира.

По важните от тях са таблиците Judges, AspNetUsers, AspNetRoles, AspNetUserRoles, като другите 5 реално не се използват директно в проекта.

На следващата страница или иначе казано на фигура номер 5 можете да видите диаграмата на базата данни, данните, които се пазят във всички таблици и връзките между тях



*Фигура 5 диаграма на базата данни*

* + 1. Таблицата AspNetUsers

Таблицата AspNetUsers съдържа данните за потребителите.

Специфичното за проекта MusicX е, че концептуално той има два типа потребители, които са нормални потребители и администратори, но пък идейно в програмата нормалните потребители се делят на състезатели или иначе казано да бъде участник в музикалното състезание, а другия тип нормални потребители са такива, които имат регистрация и могат да гласуват и да подкрепят състезателите, но не са в листата със състезатели.

Това е направено като се пази полето isCompetitor, което пази информация дали дадения потребител участва в състезанието. И същевременно данните, които са специфични за състезателите също се пазят във същата таблица, но не са задължителни тоест както се казва на английски са nullable.

Тази таблица има външен ключ към себе си или референция към себе си защото всеки потребител има правото да гласува за някого, съответно пазим всеки за кого е гласувал като тази функционалност е имплементирана именно по този начин.

Също така има връзка с таблицата за роли, която е много към много и е осъществена с помощта на помощна таблица, за която ще бъде обяснено по-долу.

* + 1. Таблицата Judges

Таблица Judges ще съдържа данните за съдиите на състезанието. Тази информация е изнесена и съхранявана в базата данни за да може администраторите да подменят динамично журито и да има цялата гъвкавост и удобство, което би желал от един софтуер.

* + 1. Таблицата AspNetRoles

Тази таблица е помощна и съдържа различните роли, които потребителя може да ползва. По подразбиране всеки нов създаден потребител е нормален потребител. Но може сайта да има и администратори, които да го следят и да се грижат за него.

* + 1. Таблицата AspUserRoles

Тази таблица е ключовата таблица, в която се пази информация за това кой потребител в каква роля е. Това е помощната таблица за връзката много към много на таблиците AspNetUsers и AspNetRoles, за да може един потребител да е в много роли и едновременно с това една роля да бъде при няколко потребители.

# Разработка

## Важни помощни класова

* + 1. HomeController

Този клас се използва за да свързва сервизния или бизнес слоя на приложението с уеб приложението, като контролера е част от уеб приложението и реално обработва HTTP заявките, които биват изпращани до сървъра, като обикновено връща резултат под формата на изглед който се визуализира на потребителя. А Ако са нужни някакви данни за да се създаде изгледа то тази работа се пренасочва към сервизния слой и той набавя нужните данни. Добрите практики при контролерите са именно те да вършат само тези две неща и да са максимално изчистени откъм код, като приложението MusicX е пример за подражание в това отношение.

* + 1. DbContext

Това е клас, който служи за моделиране на базата данни в кода, като този клас се използва за генериране на миграциите, от които пък се генерира самата база данни благодарение на Entity Framework Core, като самия този клас е част от библиотеката и ние използваме техниката наследяване за да добавим неща в него.

Този клас също се използва при работа с базата данни в приложението, като абсолютно всичко свързано с нея минава първо през него. Той е страхотен посредник и следи много неща, които ние нормалните програмисти с лека ръка бихме пропуснали или пренебрегнали, а той ги прави на готово, като това да предпазва приложението ни от sql инжектиращи атаки например.

Този клас описва базата данни като декларира публични пропъртита от тип DbSet, които играят ролята на таблица и в последствие реално се превръщат в такава.

В този клас също описваме connection string-а към базата, тоест тя може лесно да бъде подменена.

* + 1. ServiceCollectionExtensions

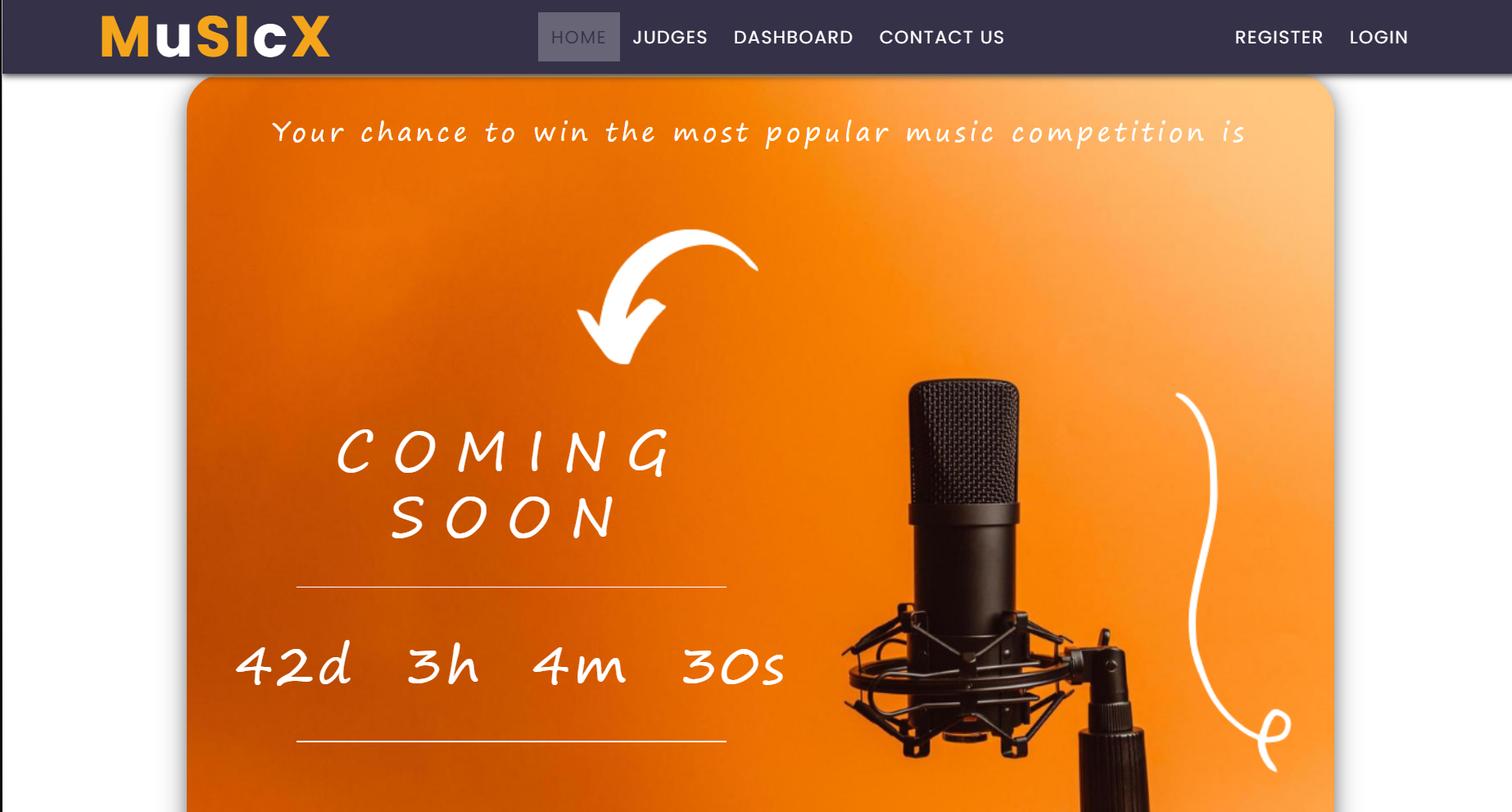
Този клас има за задача да съхранява така наречените екстеншън методи отнасящи се за интерфейса IServiceCollection, като единствения такъв метод е метода AddConventionalServices, който намира всички публични класове обозначени със интерфейсите IScopedService, ITransientService или ISingletonService и съответно ги регистрира в Dependency Injection Container – а на платформата ASP.NET Core, който е вграден и елементарен за ползване, но същевременно изключително важен, улесняващ работата на програмиста и дори повишаващ индиректно качеството на кода, защото предпоставя за по – високо ниво на абстракция.

Този dependency injection container въпреки че звучи доста сложно като терминология, всъщност е доста прост откъм задачи, които трябва да свърши. Неговата работа е да знае как се създават обекти, да може да създава обекти, като за да създаде даден обект не трябва да забравяме, че трябва да може да набави подходящи стойности за параметрите в конструктора на дадения обект, което обикновено се случва като те биват регистрирани също. Регистрация означава, че задаваш на контейнера как да създаде обект. Също така той се грижи и за живота на тези обекти, като всички изброени неща до сега предпоставят за много по качествен и гъвкав код, което е ефект желан от всеки програмист и от всяка софтуерна технология.

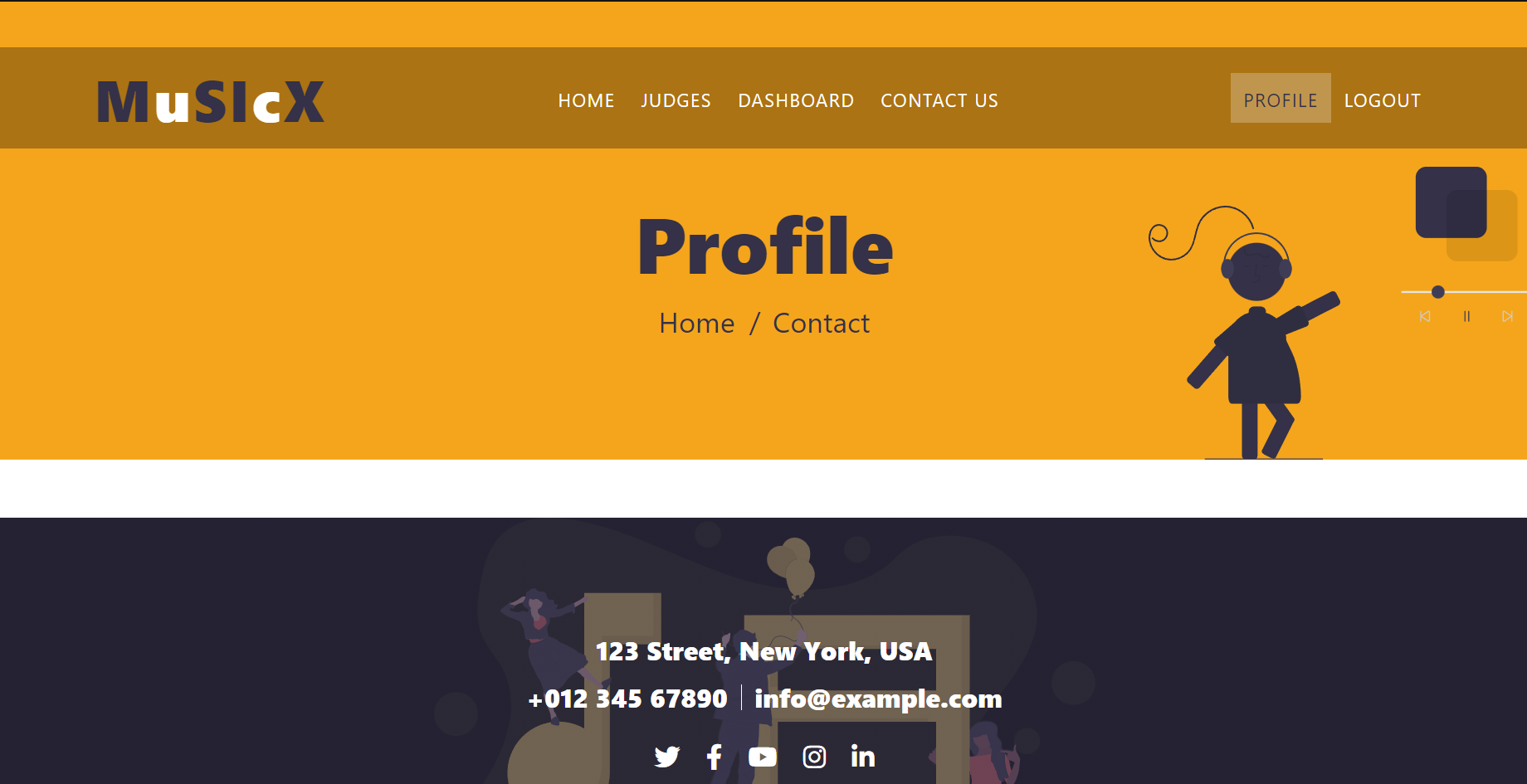
Класът ServiceCollectionExtensions е изключително важен за проекта ми защото в него е изнесена и автоматизирана всичката логика свързана с регистрация на обекти в този контейнер, който вече беше подробно обяснен. Но беше подробно обяснен именно за да стане ясно, че има доста неща за регистрация, а моя клас е направен така, че да я извършва автоматично използвайки сложни техники като рефлекшън.

## Изгледи

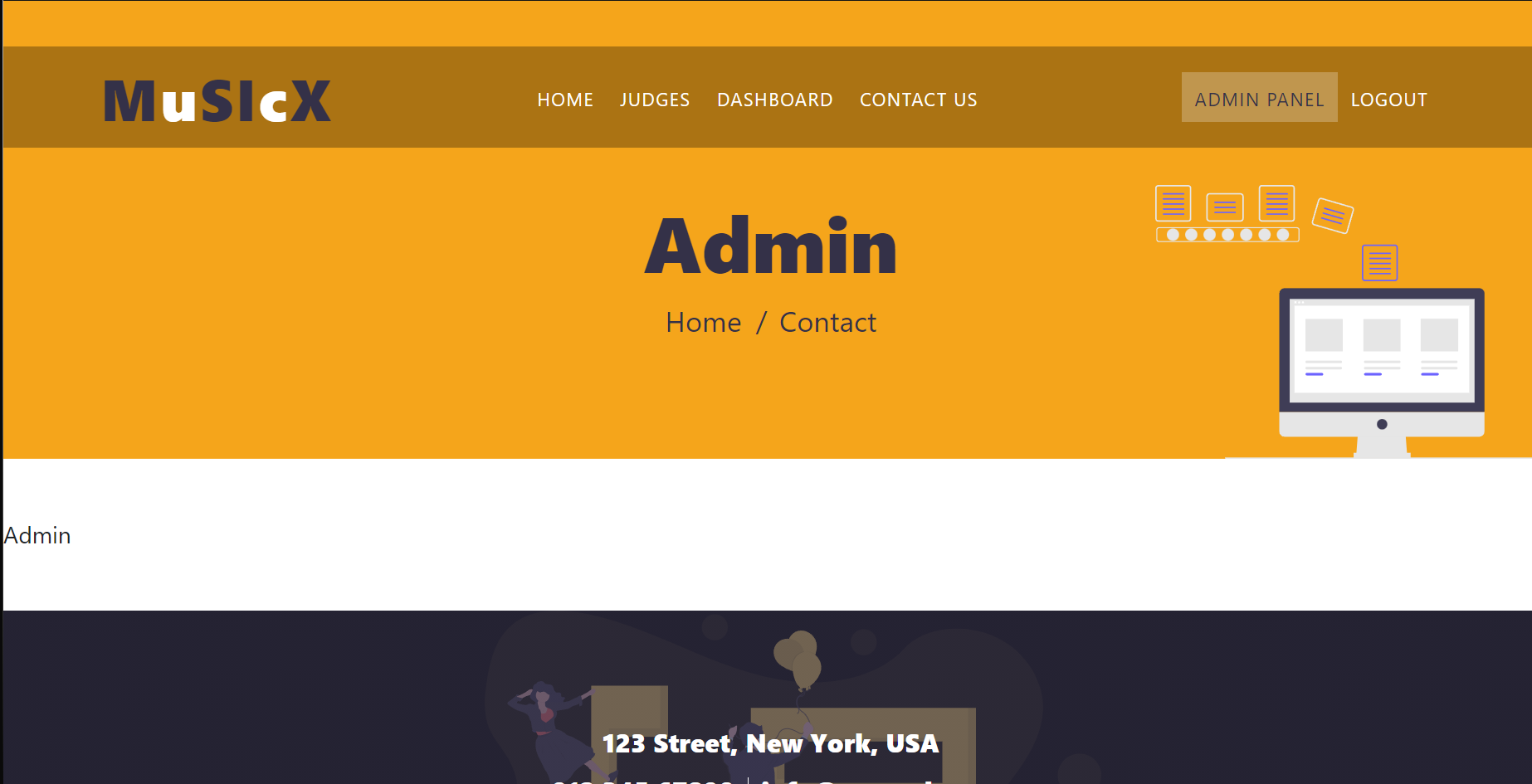
* + 1. Index



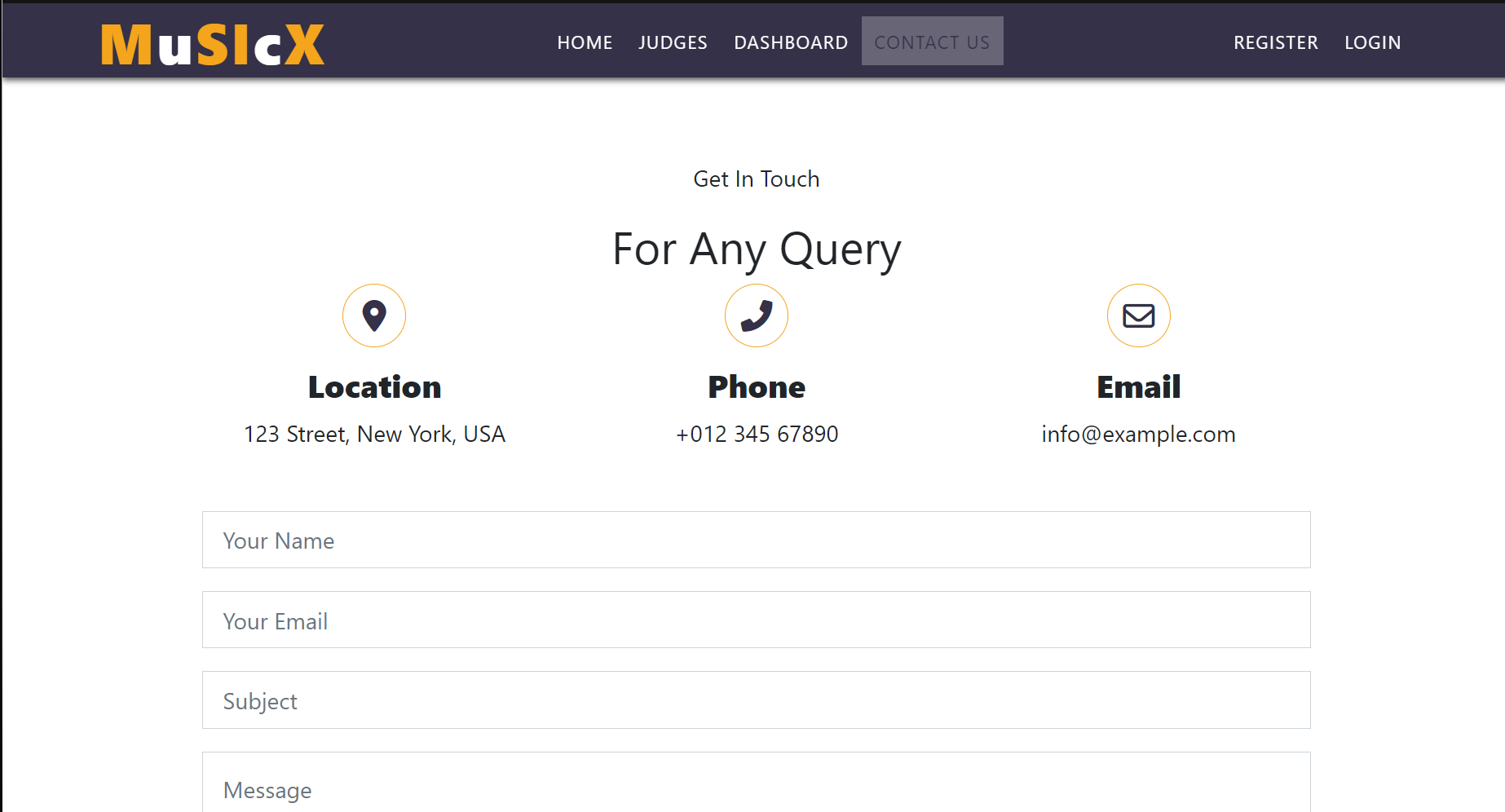
* + 1. Profile



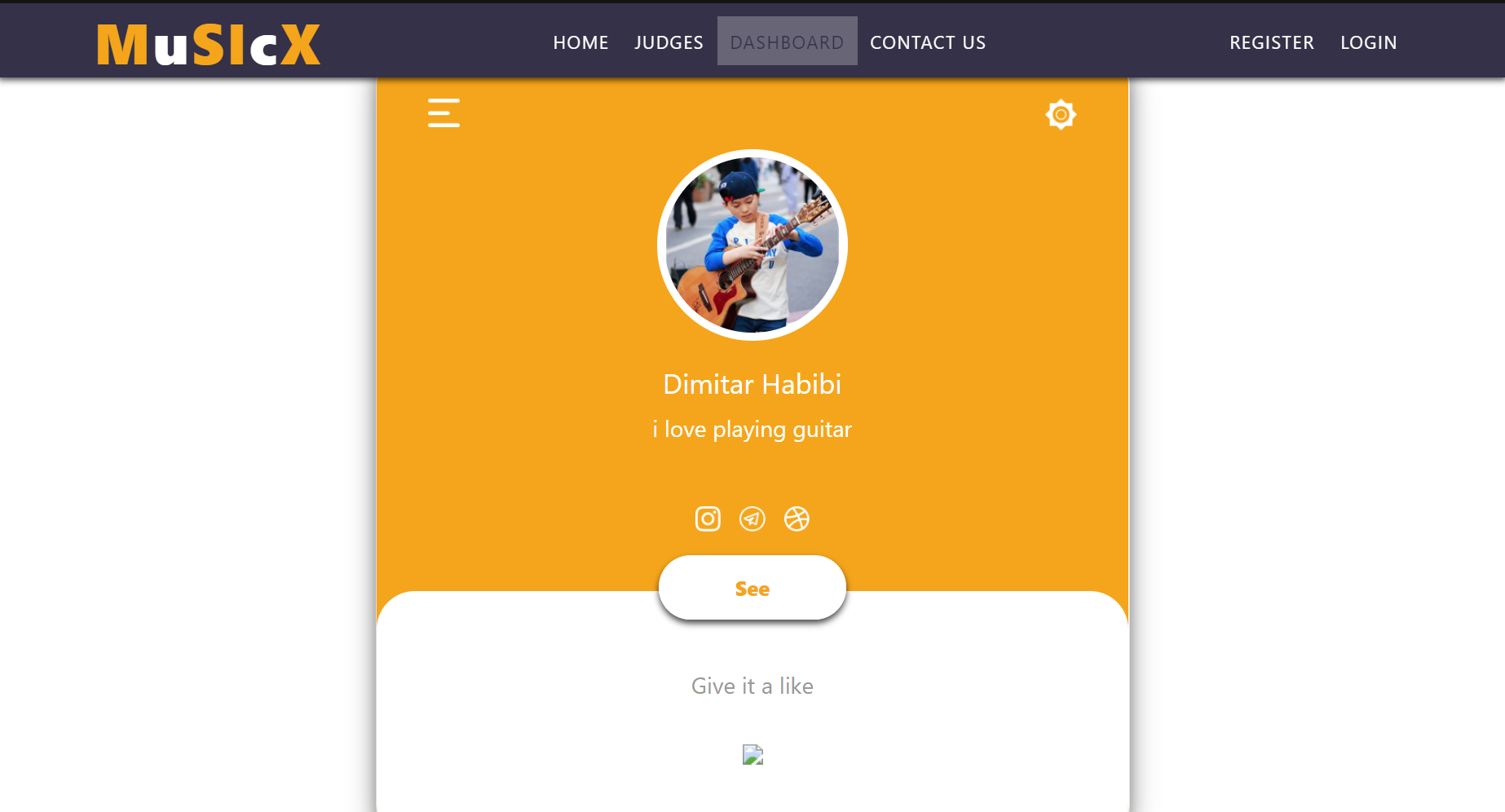
* + 1. Admin



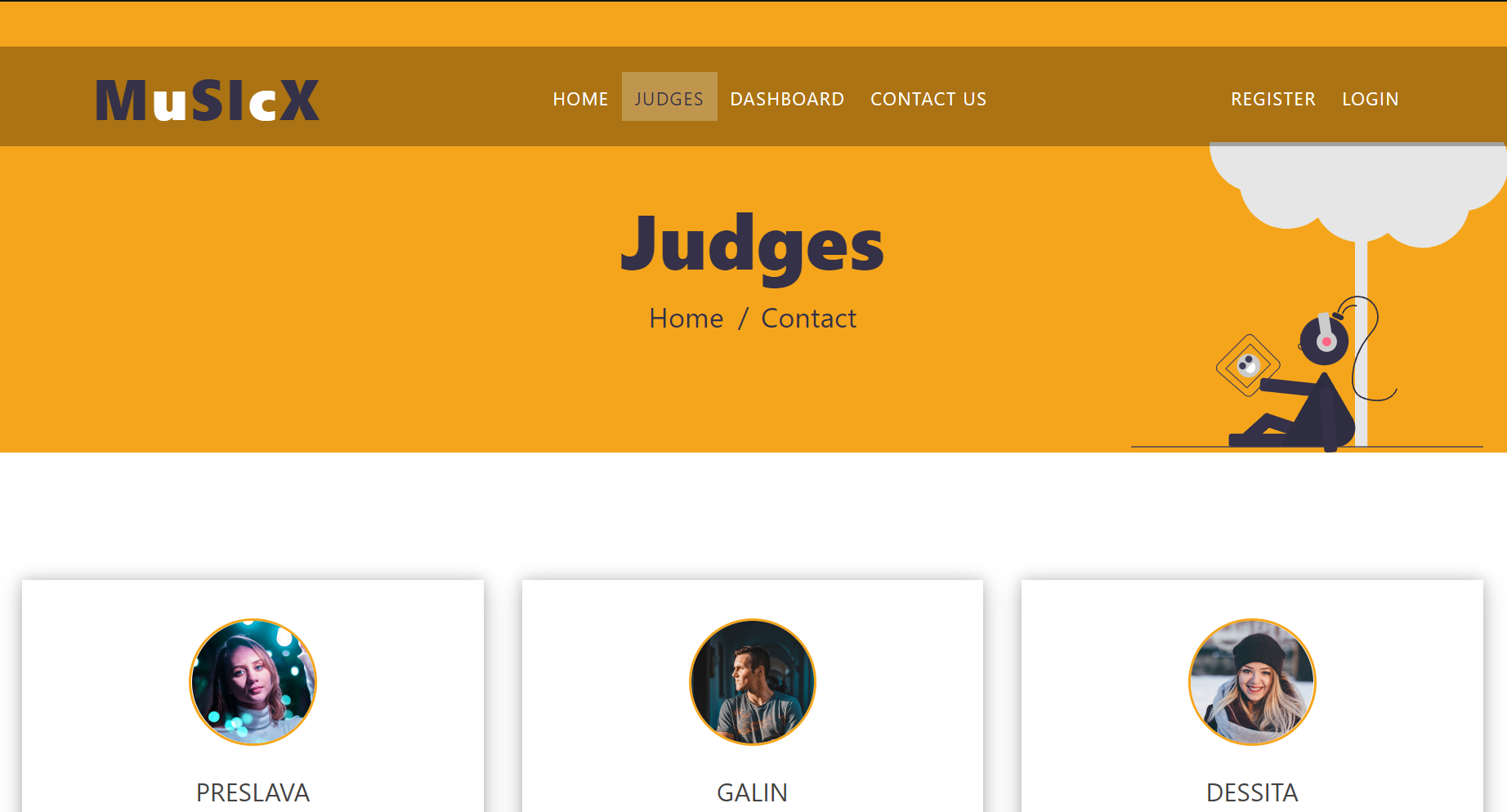
* + 1. ContactUs



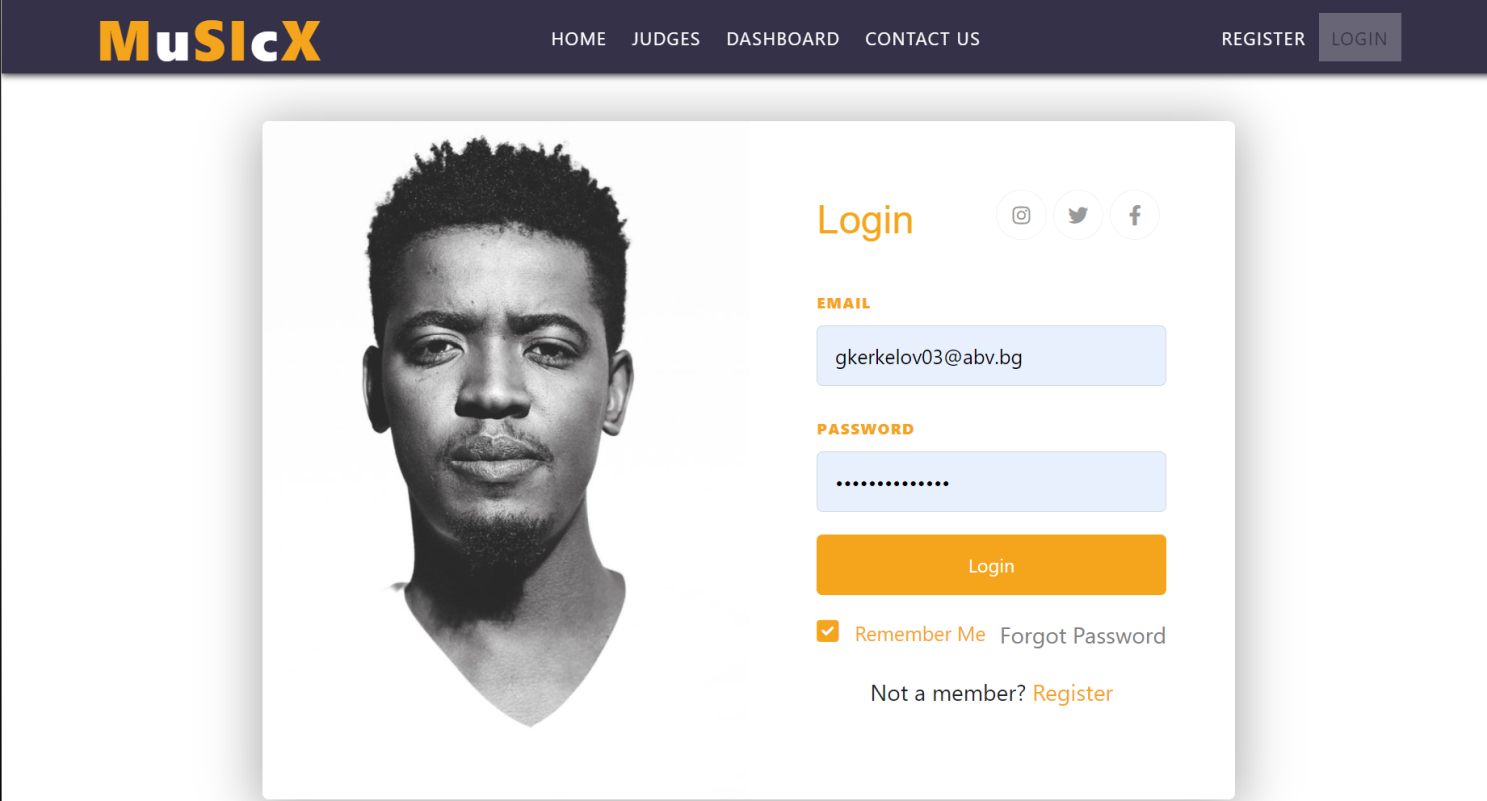
* + 1. Dashboard



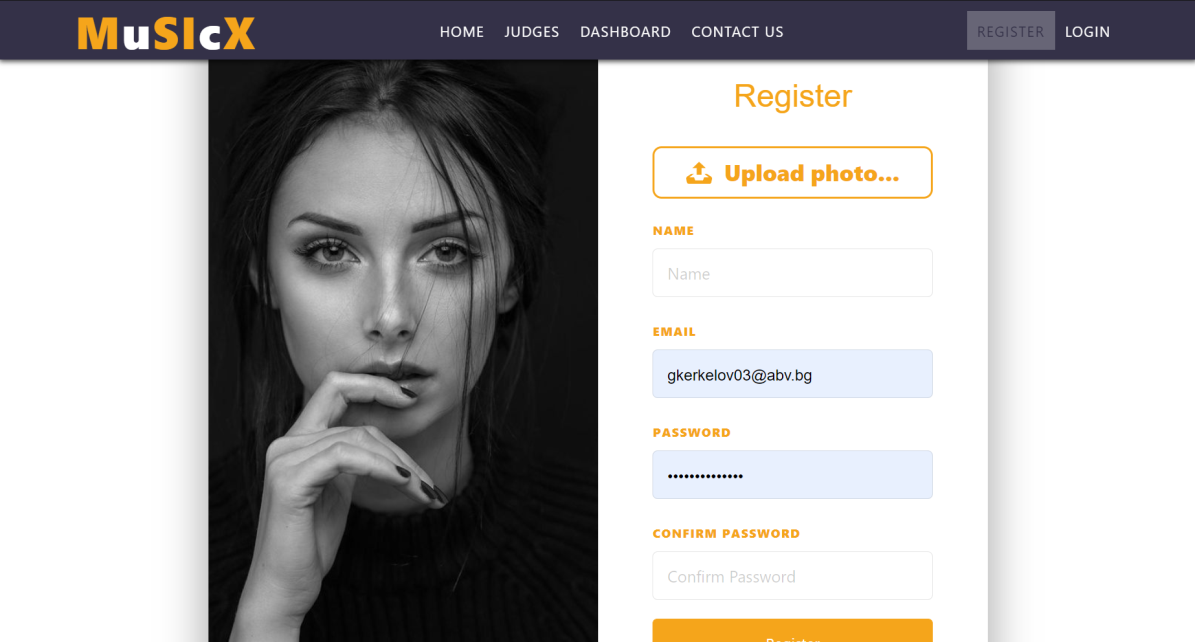
* + 1. Judges



* + 1. Login



* + 1. Register



* 1. Папката Data
     1. Проектът MusicX.Data

В този проект са имплементирани неща като сиидване на данните, което означава, че проекта сам си добавя примерни данни за демо цели или за представяне пред публика или чисто и просто лесно тестване при разработка.

DbContext класа също е описан в този проект, като за него вече обяснихме колко е ключов.

Миграциите също се намират в този проект, както и репозиторитата, чрез които е използвам така наречения Repository шаблон за дизайн.

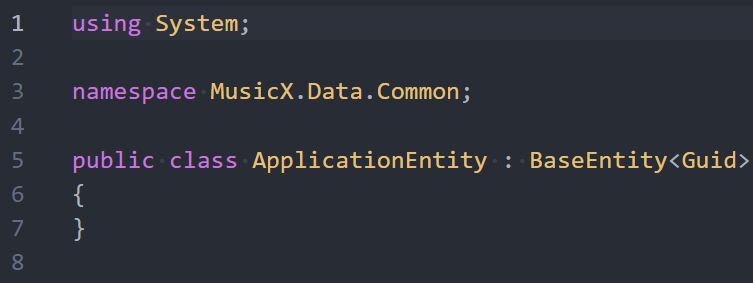
Последното нещо което съдържа този проект са различни видове конфигурации, като ApplicationUserConfiguration и IdentityOptionsProvider

* + 1. Проектът MusicX.Data.Common

В този проект са изнесени нещата, които се повтарят в проектите MusicX.Data и MusicX.Data.Моdels

Той е дом за класовете Application Entity и BaseEntity, като те представляват базови класове за всички модели на базата данни, които също така се наричат ентитита или на английски entities, за това съм кръстил тези класове по този начин.

Ако разгледаме по – отблизо класа ApplicationEntity може да ни се стори излишен и изобщо да не успеем да проумеем каква работа върши изобщо той и за какво е нужен. Ето го и него:



Това е буквално празен клас, но всъщност придава голяма гъвкавост на проекта, защото ако сменим параметъра на базовото ентити което наследява, то ще се промени в целия проект, а иначе навсякъде в проекта щеше да бъде използвано базовото ентити и щеше да се наложи навсякъде да се сменя ръчно, ако решим, че искаме вече primary ключовете на таблиците от базата да се сменят от Guid на int примерно.

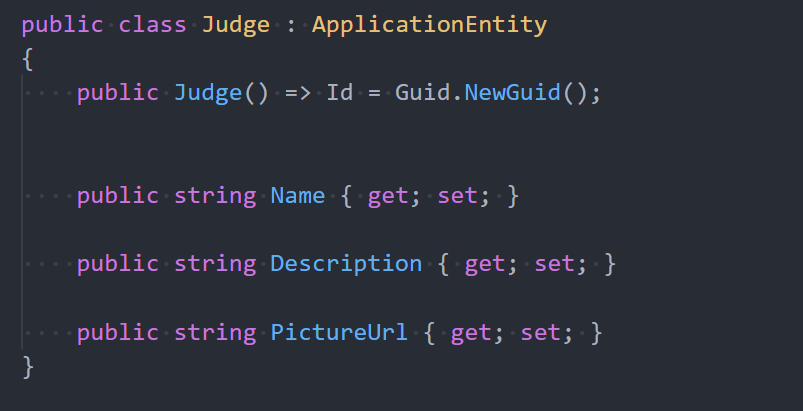
* + 1. Проектът MusicX.Data.Models

В този проект са изнесени всички модели на базата данни, защото те имат потенциала да станат огромно количество и за по добра четимост, по добро разпределение на задачите и по подреден проект като цяло съм взел решението да си имат собствен проект, като това е именно той.

Моделът на база данни показва логическата структура на базата данни, включително връзките и ограниченията, които определят как данните могат да бъдат съхранявани и достъпвани.

Тези модели се използват когато пишем създаваме и използваме класа DbContext и всички данни от базата се map-ват към конкретният за тях тип модел.

Ето и как изглежда един модел:



Той е просто клас с публични пропъртита и без функционалност използван за пренос на данни и за шаблон.

# Папката Services

* + 1. Проектът MusicX.Services

В този проект стоят всички сървиси, които не са свързани с работата на базата данни и техните абстракции, като конкретно те са CloudinaryService и RandomService. Първият от изброените е по-интересният, който ще разгледаме по-подробно, а втория просто има за задача да генерира случайни числа.

Cloudinary е доставчик на Cloud услуги, което означава, че предлага услугата да пази ваши данни. Това което потребителите печелят е че може да използват данните от всякъде не е нужно те да бъдат свалени на текущото устройство. В моя проект използвам техните услуги за да съхранявам всички картинки, защото е много по-удобно и ефективно в сравнение с това да ги съхраняваш на сървъра или в базата данни.

CloudinaryService класа е клас, който се грижи за това да си комуникира с уеб API-то на платформата Cloudinary, което ми беше доста трудно за имплементация, но мисля, че абсолютно си заслужаваше, защото веднъж, когато го setup-неш няма никакви проблеми, то си работи безотказно, лесно и бързо.

* + 1. Проектът MusicX.Services.Data

В този проект са отделени всички сървиси, които работят с базата от данни, като конкретно за моя проект те са CompetitorsService и JudgesService, които държат в себе си функционалността да връщат всички състезатели или всички участници в журито. Като използват вътрешно репозиторитата от папката Data.

Ето и как изглежда единият от тях:



* + 1. Проектът MusicX.Services.Mapping

В този проект стои кода свързан със setup-ването на AutoMapper, като по-точно регистрирането на мапингите чрез помощта на помощните интерфейси IMapFrom, IMapTo и IHaveCustomMappings, като отново идеята тук е те да се регистрират автоматично чрез техниката рефлекшън, за да направи работата на разработчика максимално лека.

Какво е AutoMapper. Това е библиотека грижеща се за това да свързва пропъртитата на един обект към пропъртитата на друг за да спести това на нас хората, което е изключително досадна задача, която за жалост се налага ние програмистите да извършваме множество пъти, ако искаме кода ни да бъде качествен. Всички големи езици и платформи за програмиране си имат еквивалент на този софтуер като просто конкретно този си подхожда най-добре с технологиите, които аз съм използвам в проекта ми MusicX.

* + 1. Проектът MusicX.Services.Models

В този проект са всички модели на сървисите, като не е задължително те да да си имат свои, биха могли и да работят с тези на базата данни, но е добра практика да си имат свои поради факта, че по този начин всяка от отделните части на проекта е самостоятелна и зависи възможно най-малко от другите. Това е ключово в разработката на софтуер, защото бихме могли да преизползваме цели слоеве от приложението в друго приложение.

Точно както моделите на базата данни моделите на сървисите са прости класове единствено съдържащи пропъртита, на които им липсва функционалност.

Минуса, който имаме, когато изберем да усложним проекта си по този начин (с добавяне на такъв тип модели заради предимствата изброени по – горе) е, че предразполага за писането на така наречения boilerplate code или това е код, който горе долу нищо толкова смислено не прави и обикновено може да се автоматизира или избегне писането му, но го пишем, защото нещата няма да работят без него и се нуждаят от него. В случая на сървиз моделите този boilerplate code е мапването от единия тип модели в другия тип модели, но заради AutoMapper това става изключително лесно и удобно.

* 1. Папката Уеб
     1. Проектът MusicX.Web

Този проект е най-важния за работата на приложението. Реално той е този който бива стартиран и от който използва в себе си абсолютно всички други проекти, като по принцип не е задължително да бъдат разделени изобщо, тоест всичко би могло да бъде в този проект, но такъв тип архитектура при по-голям проект като MusicX, би била пагобна от гледна точка на така наречения developer experience, тоест би било в пъти по – трудно на програмистите да се ориентират в такъв проект, да работят, да го следят като прогрес и би ги забавило изключително много. Също така ще върже ръцете им откъм гъвкавост, защото те не биха могли да преизползват отделните части на проекта в друго приложение, което е един от основните принципи и цели в програмирането.

В този проект се съхраняват контролерите, изгледите и статичните файлове.

* + 1. Проектът MusicX.Web.Infrastructure

В този проект е изнесено всичко от проекта MusicX.Web, което не е жизнено важно или което има под някаква форма повече код или специфични изисквания за подредба на някои файлове, като е изнесено именно за да не замърсява главния проект, за да може в него директно да се използват тези неща и да се подобри четимостта му.

В този проект стоят неща като View components, tag helpers, partial views, и също така в конкретно моя проект съм изнесъл и страниците за регистрация и влизане, защото ги считам за второстепенни един вид и съм оставил само най-важните неща в главния проект.

* + 1. Проектът MusicX.Web.BindingModels

В този проект са изнесени всички класове, които служат за приемане на данни в контролерите или иначе казано това за така наречените обекти за трансфер на данни, които биват използвани като модели за контролерите, тоест те нямат функционалност единствено служат за това имената на пропъртитата им да бъдат сравнени с данни пратени от потребителя в HTTP заявката, за да може тази информация да бъде смляна до обект от тип Binding модела, а не да е в чист вид, защото е много по удобно за програмиста и позволява допълнителни функционалности като валидация, което е изключително важно и широко използвано, също така е направено изключително лесно за ползване в платформата ASP.NET Core

* + 1. Проектът MusicX.Web.ViewModels

Моделът на изглед представлява данните, които искаме да покажем в нашия изглед/страница, независимо дали се използват за статичен текст или за входни стойности (като текстови полета и падащи списъци), които могат да бъдат добавени към базата данни (или редактирани). Те са различни като типове от моделите на базата данни и сервизните модели, като те отново нямат функционалност и чрез пропъртита описват структурата на данните, с които се налага да работим във изгледите. Имаме нужда от тях защото бихме могли да нямаме нужда от всички данни които модела на базата данни пази за конкретните обекти, а само част от тях, съответно не трябва да даваме достъп до тези данни. Това също е един от основните принципи, които програмистите се стремят да спазват, а именно ако даден код да има възможността да използва само код, от който има нужда, а всеки друг да бъде недостъпен за него, за да не се дава излишна власт в ръцете на този, който го консумира.

* 1. Папката Common

В тази папка са поставени класове, които биват достъпвани от цялото приложение или неща, общи за цялото приложение. Това са глобални константи, конфигурации, конвенционални интерфейси и др.

В моя проект това са интерфейси като ITransientService и IScopedService. Наричам ги конвенционални, защото единствената им функция е някой да ги имплементира, като те не реално не изискват никакви методи или пропъртита, тоест са празни, но пък когато бъдат имплементирани имам възможността да ги изследвам с рефлекшън и да ги достъпя програмно, което ми помага да автоматизирам различни важни процеси в приложението, като регистрация в контейнера за зависимости.

# Заключение

# Използвана литература

<https://bg.wikipedia.org/wiki/Microsoft_Visual_Studio15/06/2018> Visual Studio 2022

<https://softuni.bg/blog/common-language-runtime> CLR

<https://docs.microsoft.com/bg-bg/dotnet/standard/garbage-collection/> Garbage Collection <https://speedflow.bg/blog/what-is-javascript-what-it-is-used-for/> JavaScript

https://softuni.bg/blog/asp-dot-net-core-and-mvc/ ASP.NET Core

https://www.c-sharpcorner.com/article/learn-about-razor-view-engine/ Razor View Engine

[https://docs.microsoft.com/en-us/aspnet/core/security/authentication/scaffold- identity?view=aspnetcore-6.0&tabs=visual-studio](https://docs.microsoft.com/en-us/aspnet/core/security/authentication/scaffold-%20%20%20%20%20%20%20%20%20%20identity?view=aspnetcore-6.0&tabs=visual-studio) Asp.Net Core Code Generation

<https://www.hwlibre.com/bg/orm-object-relational-mapping/> ORM технология

<https://softuni.bg/courses/entity-framework-core> Entity Framework Core

<https://searchdatamanagement.techtarget.com/definition/relational-database> Релационна база данни

<https://bg.wikipedia.org/wiki/Microsoft_SQL_Server> Microsoft SQL Server

<https://www.lucidchart.com/pages/database-diagram/database-models> Entity

<https://blog.sqlizer.io/posts/json-history/>JSON

<https://www.lifewire.com/introduction-to-client-server-networks-817420>клиент-сървър

<https://www.lifewire.com/what-is-a-browser-446234>уеб браузър

<https://www.cloudflare.com/learning/dns/what-is-dns/>DNS

[https://www.ibm.com/support/knowledgecenter/en/SSGMCP\_5.3.0/com.ibm.cics.ts.webservi](https://www.ibm.com/support/knowledgecenter/en/SSGMCP_5.3.0/com.ibm.cics.ts.webservices.doc/concepts/dfhws_definition.html) [ces.doc/concepts/dfhws\_definition.html](https://www.ibm.com/support/knowledgecenter/en/SSGMCP_5.3.0/com.ibm.cics.ts.webservices.doc/concepts/dfhws_definition.html) уеб услуги

<https://www.codecademy.com/articles/what-is-rest>REST

<https://stackoverflow.com/questions/11064316/what-is-viewmodel-in-mvc> View Model