**Пособие для студентов факультета ПММ.   
Разработка приложений баз данных.**

*Автор: Ростовцев Александр Андреевич 4 курс 8 группа МОЭВМ*  
*2020 год*

**Предисловие**

Данное пособие предназначено для студентов 4 курса факультета ПММ. В нём будет описываться создание с нуля web-приложения баз данных на платформе .Net с использованием HTML и JavaScript. Для хранения данных будет использоваться SQL Server, а в качестве подхода к проектированию базы данных будет выбран Code First.  
В рамках данного пособия при создании приложения не будут использоваться какие-либо специфические фреймворки платформы .Net, технологии языков C# и JavaScript а также паттерны программирования, предполагается лишь наличие начальных навыков написания кода и знание ООП, но тем не менее перед началом работы рекомендуется ознакомиться со следующими базовыми вещами, если не встречались с ними ранее:

MVC (архитектура приложений);

Razor (.Net);

Лямбда-выражения, LINQ (C#);

jQuery (JavaScript);

К данному пособию будет прикреплён архив с демонстрационным приложением, на примере которого оно создавалось. Рекомендуется обращаться к этому приложению по ходу изучения пособия.

**Введение**

Приложение баз данных - это набор форм и отчетов, с которыми работают пользователи, а также программной логики, которая реализует необходимую функциональность. Большая часть приложений при работе так или иначе обращается к базам данных, которые в реляционном виде хранятся отдельно от приложения.   
Каждое такое приложение можно разделить на три основных части:

Пользовательская часть. К ней относится всё, что видит пользователь, работая с приложением;

Серверная часть. К ней относится всё, что связано с обработкой данных, а также с управлением событий, вызванных действиями клиента на пользовательской части;

База данных. К данной части относится долгосрочное хранение какой-либо информации, связанной с приложением;

**Постановка и анализ задачи**

Разработать полноценное приложение баз данных, хранящее информацию в реляционной базе данных, обрабатывающее её на стороне сервера и имеющее интерфейс для взаимодействия с пользователем.

Для реализации данной задачи необходимо определиться с выбором следующих аспектов:

Вид приложения. Это может быть консольное, оконное или web-приложение;  
Для разработки приложения баз данных в рамках данного курса был выбран способ реализации через web-приложения.   
К плюсам данного подхода в первую очередь можно отнести:  
 I - Кроссплатформенность. Каждое современное устройство имеет браузер. Благодаря этому любой пользователь сможет комфортно использовать разработанное приложение.   
 II - Гибкость интерфейса и пользовательской части. Web-технологии позволяют разработать достаточно "дешёвую" пользовательскую оболочку клиента - она будет обрабатываться браузером и поэтому не придётся тратить много усилий для реализации её логики.  
В то же время, явным минусом данного подхода является необходимость знания языков HTML и JavaScript.

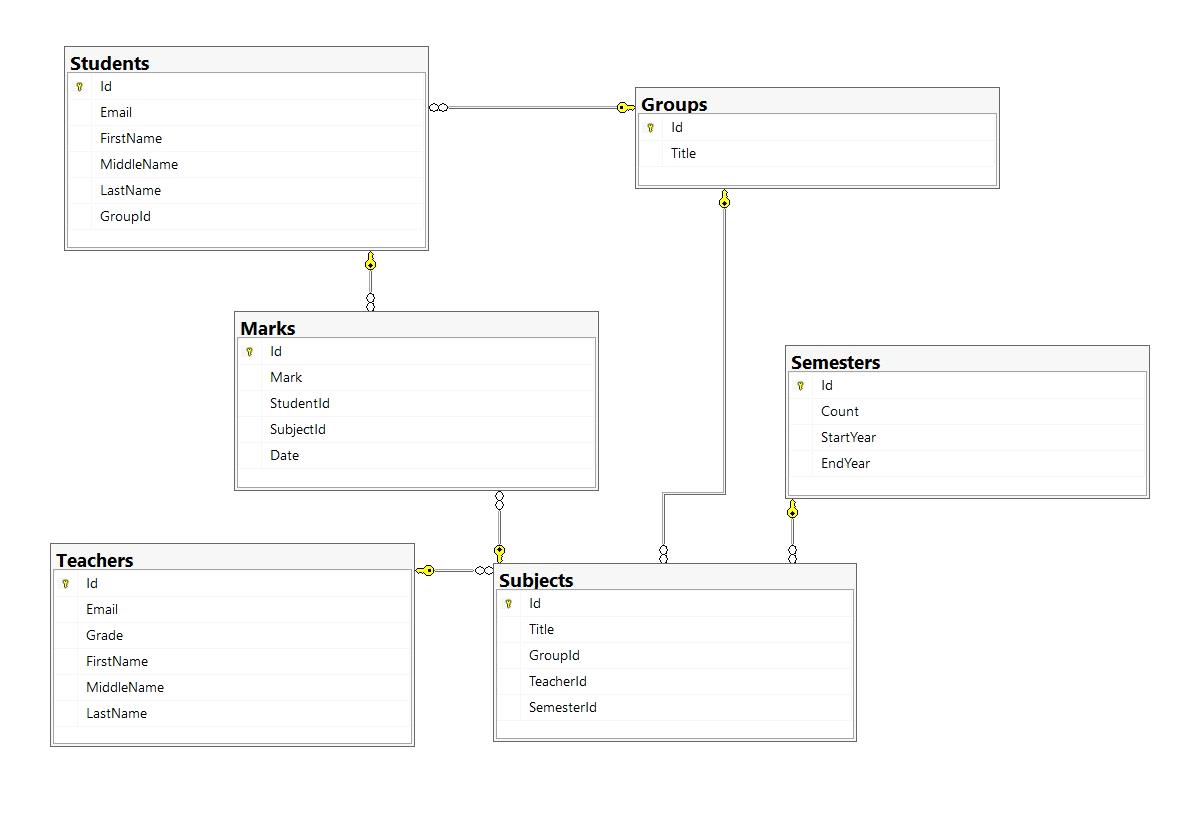
Выбор стека технологий, который даёт возможность реализовать единое цельное приложение, содержащее в себе все три основные части приложения баз данных;  
Для реализации части базы данных выбор пал на SQL Server 2019 Express. SQL Server 2019 Express является бесплатным выпуском SQL Server, который идеально подходит для разработки приложений для использования на настольных компьютерах, веб-серверах и других небольших серверах. Для реализации сервеной и пользовательской части приложения была выбрана платформа .Net; В частности - фреймворк ASP.Net Core. ASP.Net Core - свободно-распространяемый кросс-платформенный фреймворк для создания веб-приложений с открытым исходным кодом. Данная платформа разрабатывается компанией Майкрософт совместно с сообществом.

**Разработка  
Часть 1. Тема приложения**

Сперва необходимо определиться с предметной областью и бизнес-логикой приложения. В рамках данного пособия будет разработано приложение для студентов и преподавателей:   
 В базе данных будет храниться информация о преподавателях, студентах и их группах, предметах и оценках студентов по ним; Преподаватели будут иметь возможность ставить оценки студентам по своим предметам; Студенты будут иметь возможность смотреть оценки по своим предметам.  
 Наличие ролей и разграничения возможностей подразумевает необходимость авторизации.

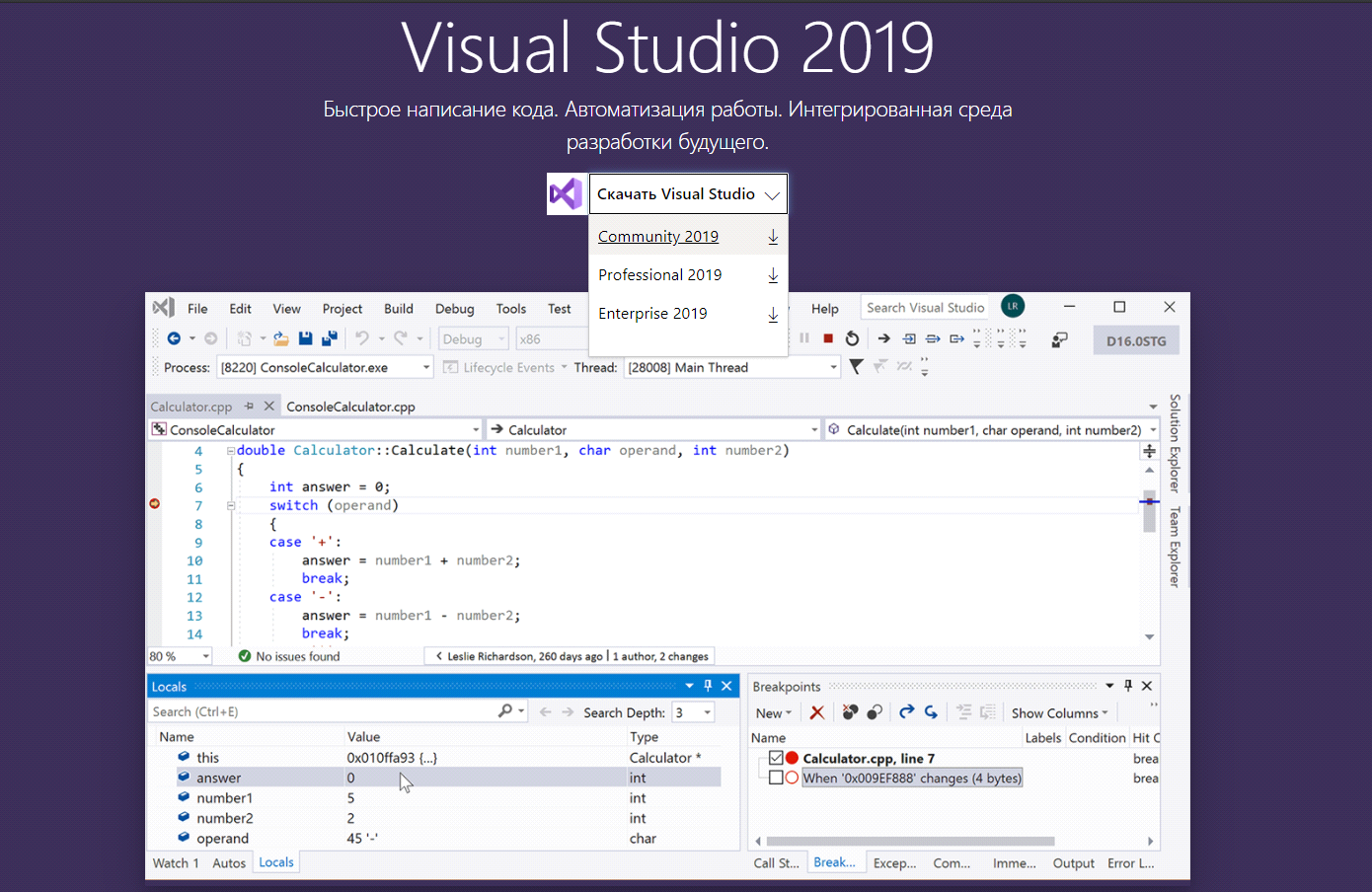
**Часть 2. Построение схемы базы данных**

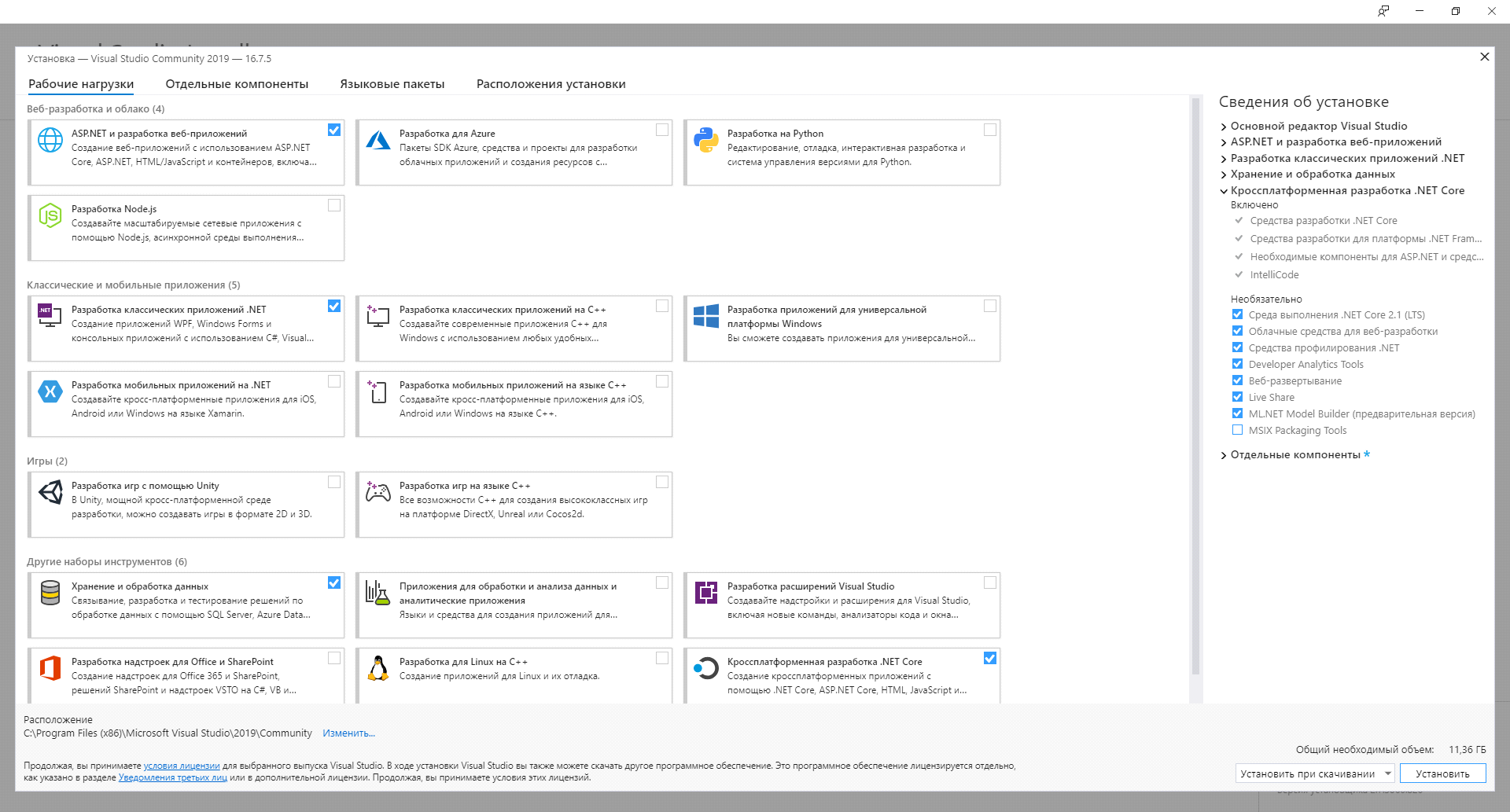
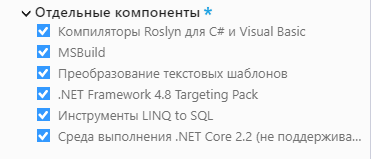
После того как тема приложения и логика его работы продумана необходимо тщательно продумать схему хранения данных. Для реализации демонстрационного приложения понадобится 6 таблиц со следующими внешними связями:

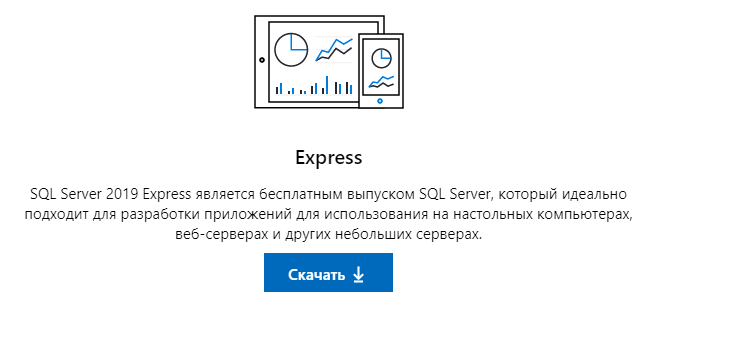
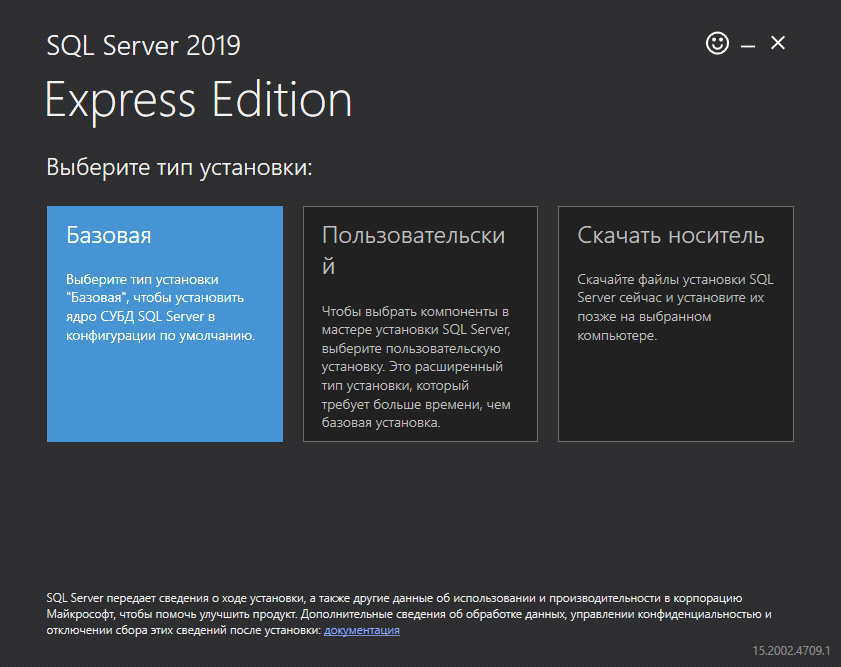


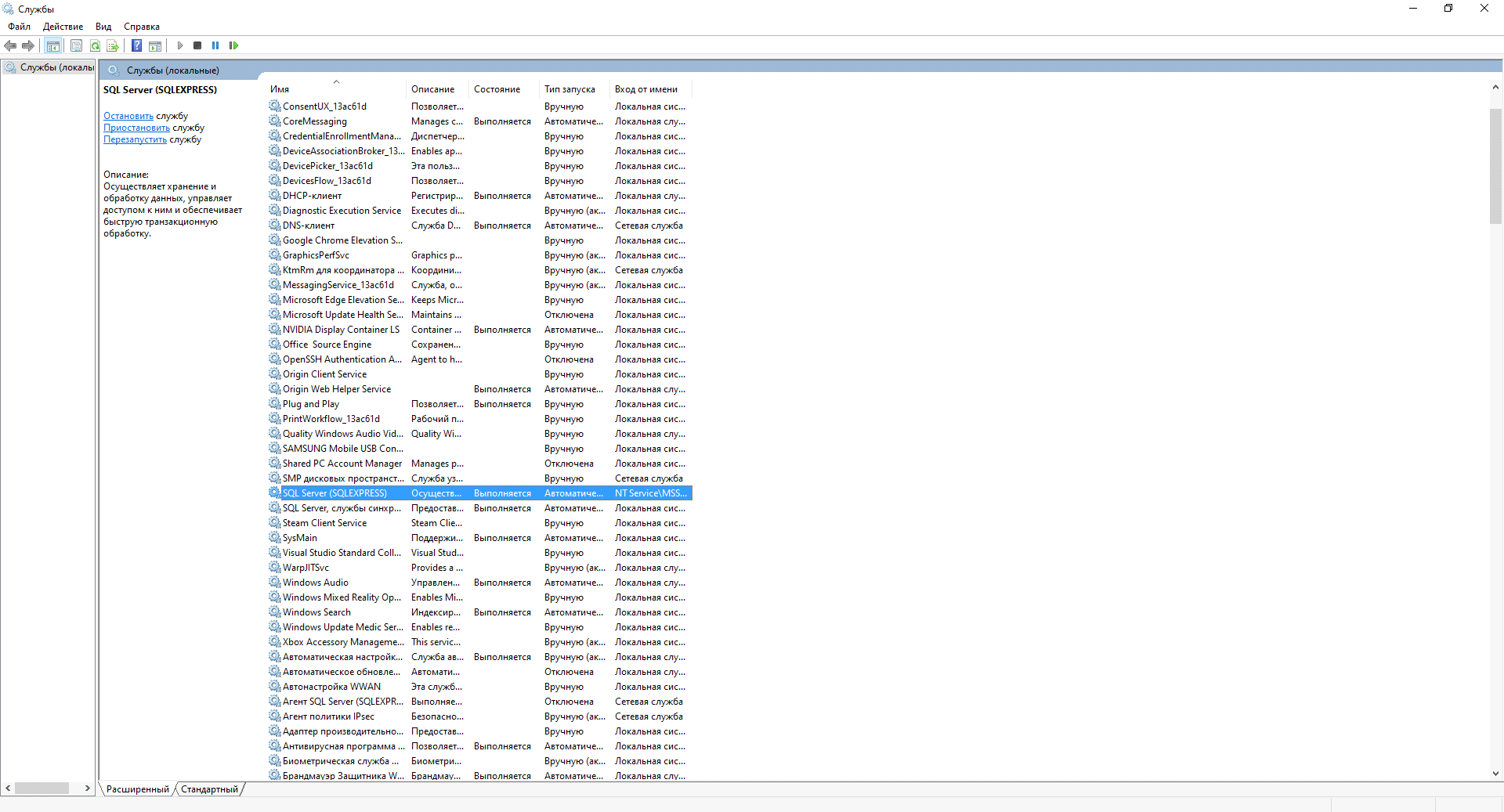
**Часть 3. Настройка окружения**

Теперь, когда часть планирования завершена, перед началом реализации необходимо установить и настроить среду разработки. Для выполнения поставленной задачи понадобятся:

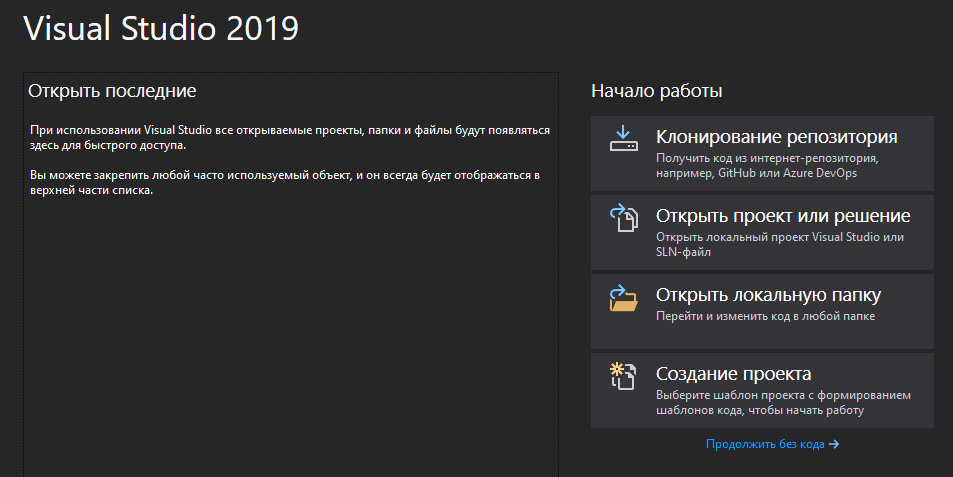
Visual Studio 2019 Community;  
Является свободно распространяемым продуктом. Скачать можно по ссылке: <https://visualstudio.microsoft.com/ru/vs/>

После скачивания в установщике необходимо выбрать следующие рабочие нагрузки:  
  
  
А также следующие отдельные компоненты:  


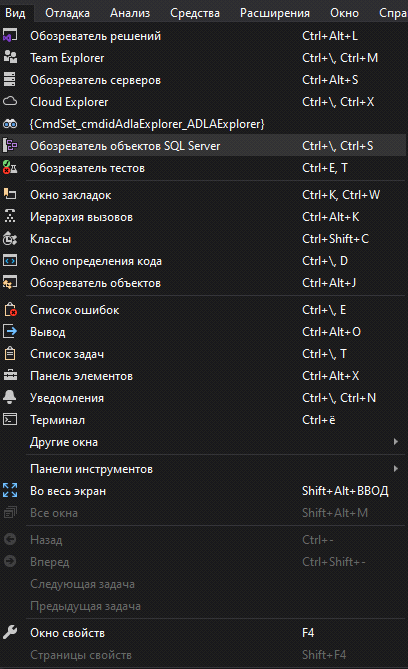
SQL Server 2019 Express;  
Является свободно распространяемым продуктом. Скачать можно по ссылке: <https://www.microsoft.com/ru-ru/sql-server/sql-server-downloads>  
После скачивания в установщике необходимо выбрать базовую установку:  


По окончанию установки необходимо будет убедиться, что SQL Server установлен и запущен. Для этого требуется пройти открыть мониторинг служб Windows ("Пуск - Службы") и найти там   
SQLSERVER (SQLEXPRESS) и запустить его:  


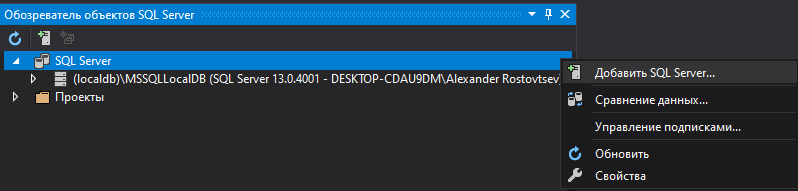
Теперь, когда установка необходимых инструментов завершена необходимо подключить Visual Studio к SQL Server. Для этого запустите Visual Studio без кода:



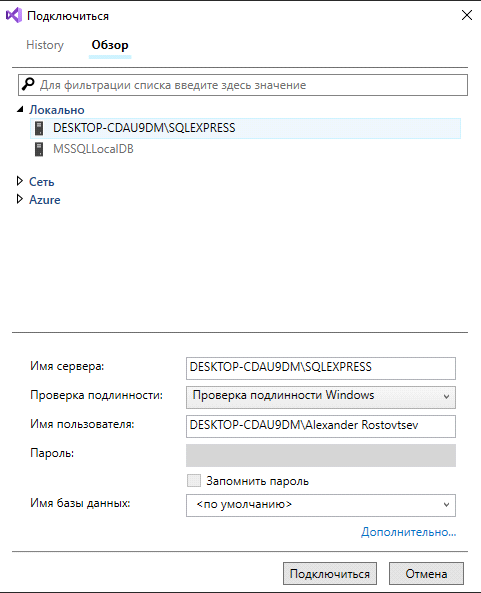
Далее нажмите "Вид" и выберите "Обозреватель объектов SQL Server":



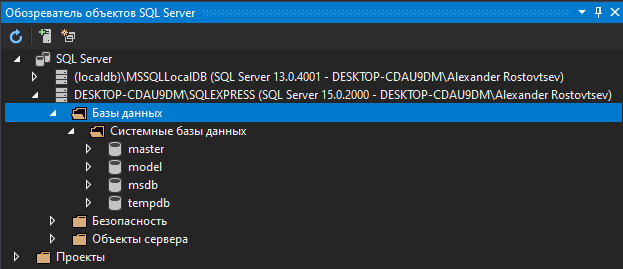
В открывшемся обозревателе необходимо добавить новый SQL Server:



В окне подключения во вкладке "Локально" нужно выбрать SQLEXPRESS. Если по каким-либо причинам данный сервер отсутствует в этой вкладке - перезапустите SQL Server в службах. Если SQLEXPRESS всё равное не появляется - введи в имя сервера "(local)\SQLEXPRESS" без кавычек, оставив остальные параметры поумолчанию:



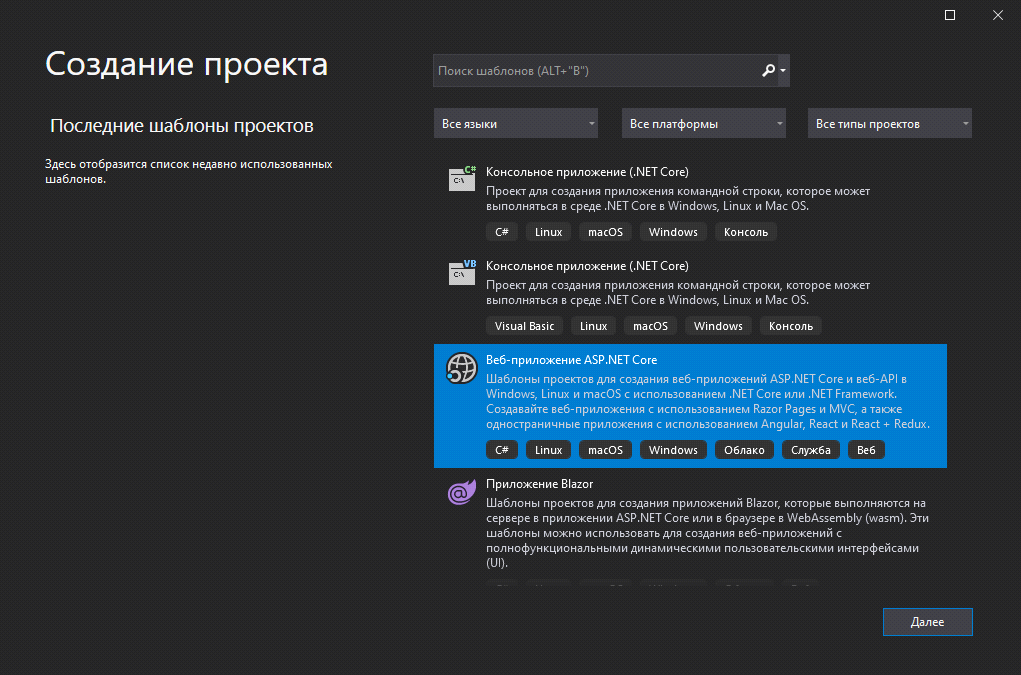
После этого добавленный сервер можно будет увидеть в обозревателе:



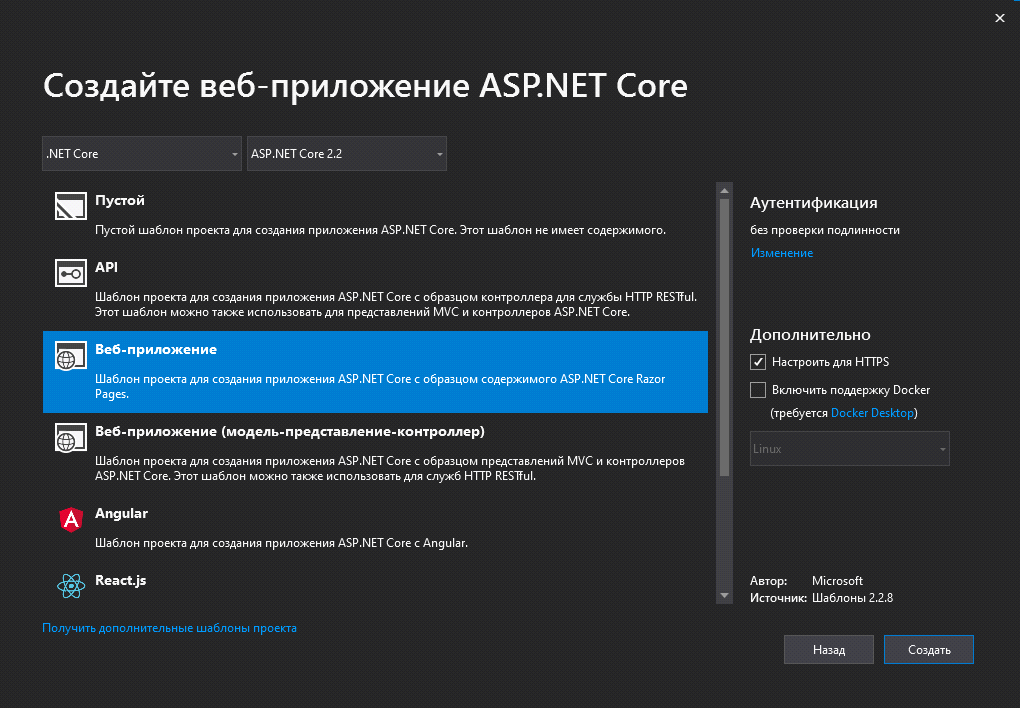
**Часть 4. Реализация**

**Пункт 1. Создание нового проекта**

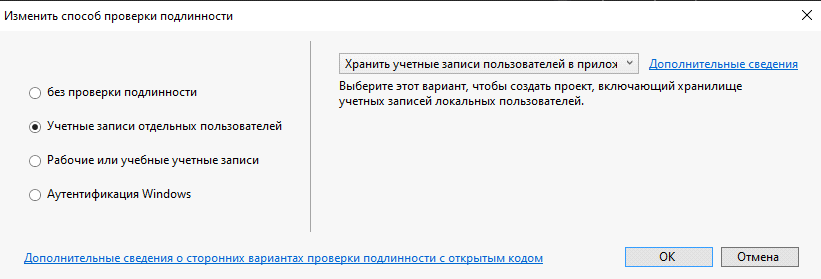
Как было упомянуто ранее в 1-ой части разработки, наличие ролей и разграничения возможностей подразумевает необходимость авторизации. К счастью, для таких случаев существует готовое решение с открытым исходным кодом. Для реализации авторизации в демонстрационном приложении будет использоваться ASP.NET Core Identity, из которого потребуется удалить весь "ненужный" функционал.  
Для начала в Visual Studio необходимо создать новый проект, выбрав "Веб-приложение ASP.NET Core:



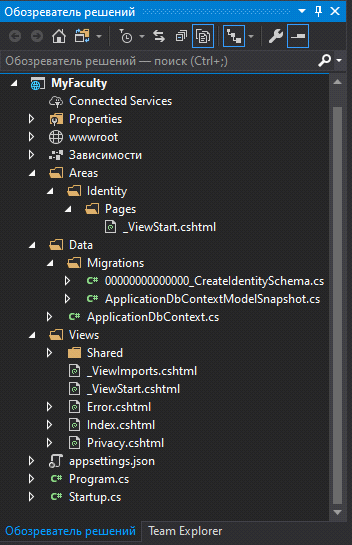
Далее нужно выбрать .NET Core и ASP.NET Core 2.2, и в списке шаблонов выбрать "Веб-приложение", после чего в правой части окна в разделе "Аутентификация", где по умолчанию выставлено "без проверки подлинности", нажать на теперь активную кнопку "Изменение":



В открывшемся окне выбрать способ проверки подлинности   
"Учётные записи отдельных пользователей":



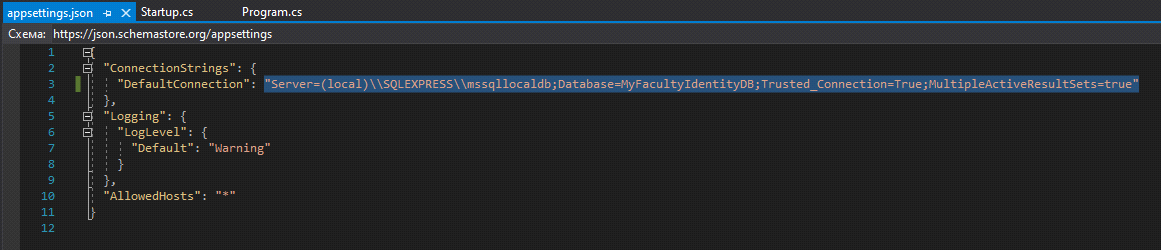
После этого нажмите "ОК" и "Создать". В обозревателе проектов станет видна архитектура созданного приложения (Вместо папки "Views" может быть папка "Pages"):



**Пункт 2. Подключение приложения к базе данных**

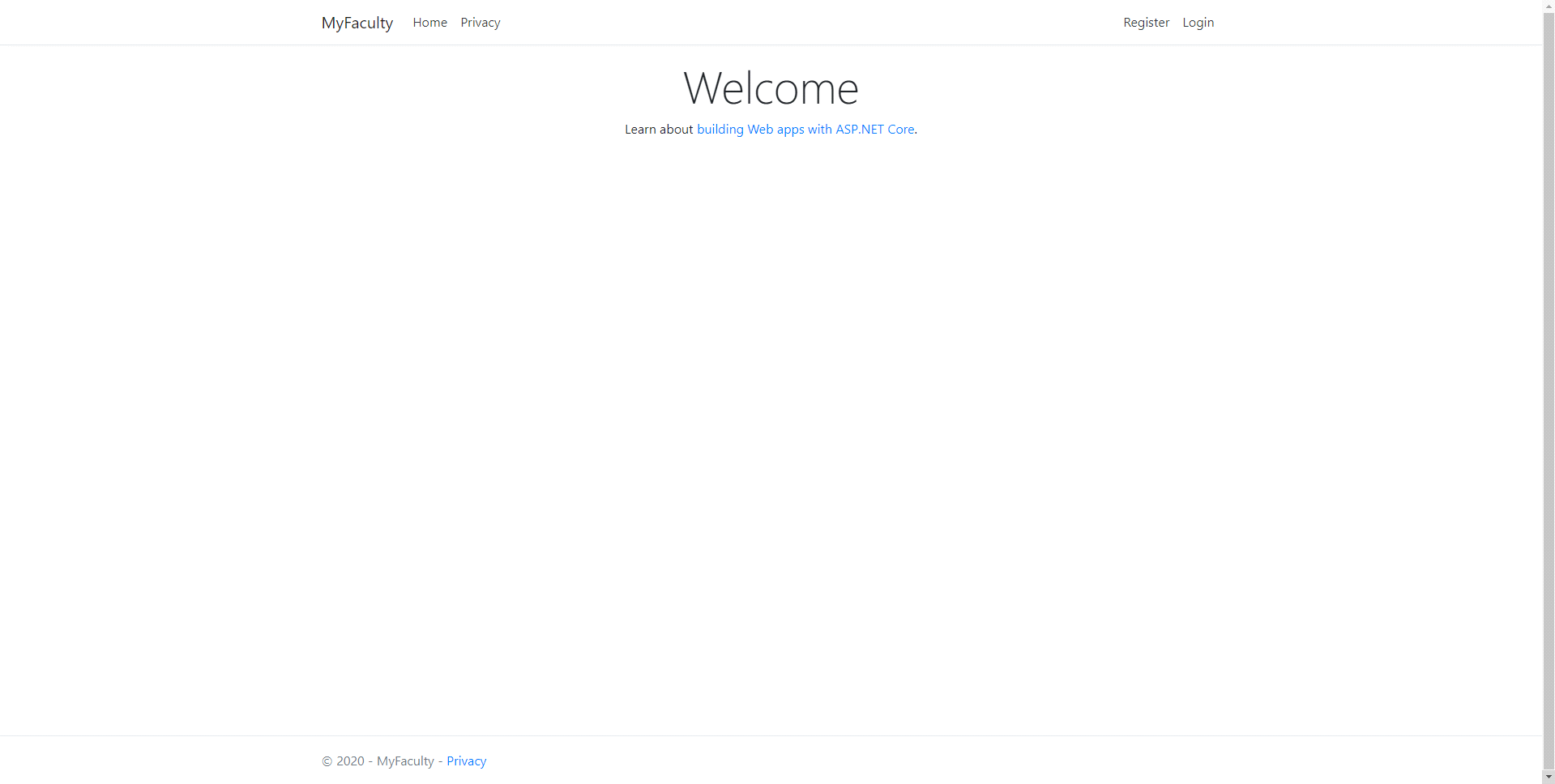
Сразу же откройте appsettings.json и в блоке ConnectionStrings в DefaultConnection вставьте следующую строку:  
**"Server=(local)\\SQLEXPRESS;Database=MyFacultyIdentityDB;Trusted\_Connection=True;MultipleActiveResultSets=true"**

Данная строка называется строкой подключения. Указав строку подключения вы обеспечиваете своему приложению подключение к SQL Server:



**Пункт 3. Первый запуск приложения. Знакомство с миграциями**

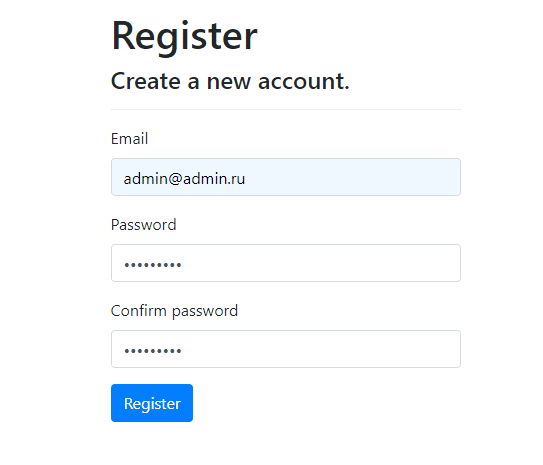
Сгенерированное шаблонное приложение можно запустить сразу после его подключения к базе данных. Более того, сделать это рекомендуется. Для того чтобы запустить Веб-приложение необходимо на верхней панели Visual Studio выбрать для запуска IIS Express и запустить проект. При первом запуске может потребоваться сделать сертификат IIS Express доверенным. После этого приложение запустится и вы увидите пустую стартовую страницу:



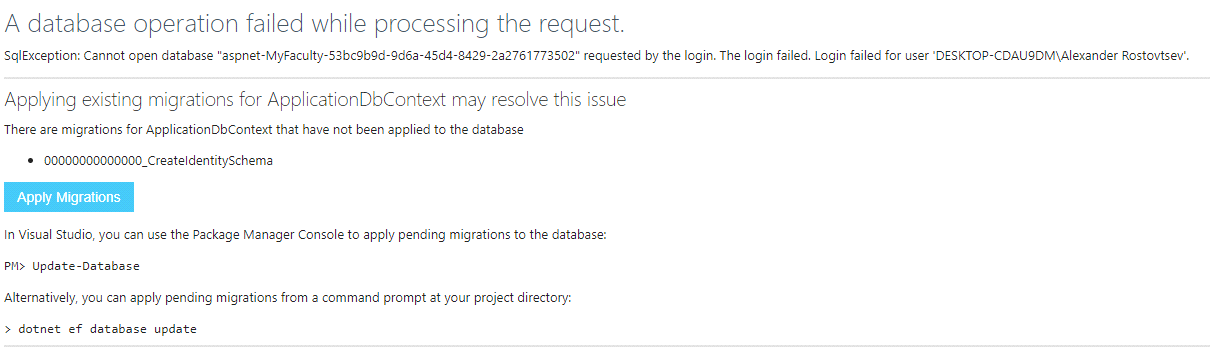
Работа с приложением начнётся с попытки регистрации первого пользователя;  
Нажмите на кнопку "Register" и попытайтесь зарегистрироваться; В дальнейшем в данном пособии будет использоваться пользователь со следующими данными:

**Login: admin@admin.ru**

**Password: 123456Az!**

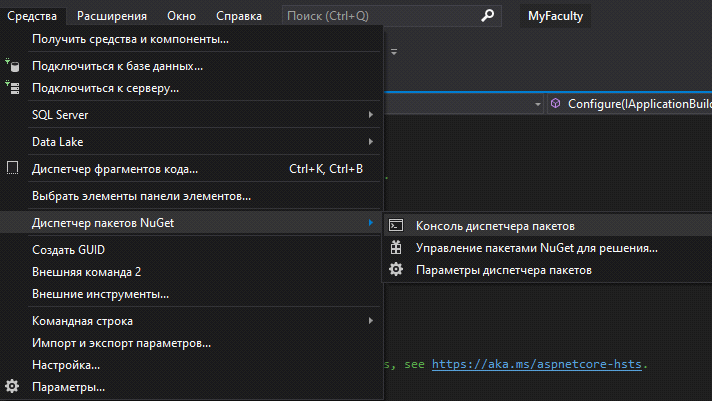


После первой попытки зарегистрировать нового пользователя вы гарантированно получите ошибку следующего вида:

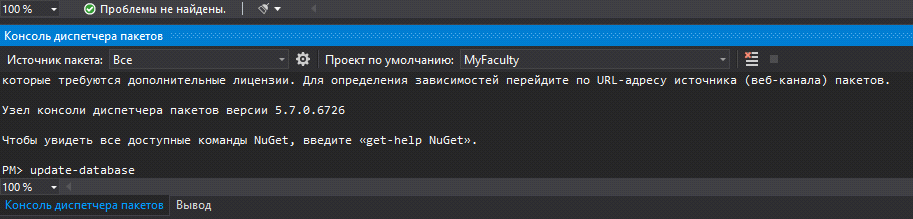


Это значит, что SQL Server сейчас находится в состоянии, не готовом для корректной работы приложения, в то время как приложение содержит в себе инструкции по приведению его в нужное состояние. Такие инструкции называются **миграциями**.   
На данном этапе полезно будет изучить содержимое двух файлов в папке *Migrations* вашего проекта.  
Чтобы применить уже существующую миграцию можно нажать на кнопку "Apply Migrations" в браузере, но этот способ сработает только в первый раз. Т.к. скорее всего к этой процедуре придётся прибегать ещё не единожды и чтобы поскорее познакомиться с процессом применения миграций мы сделаем это иначе.

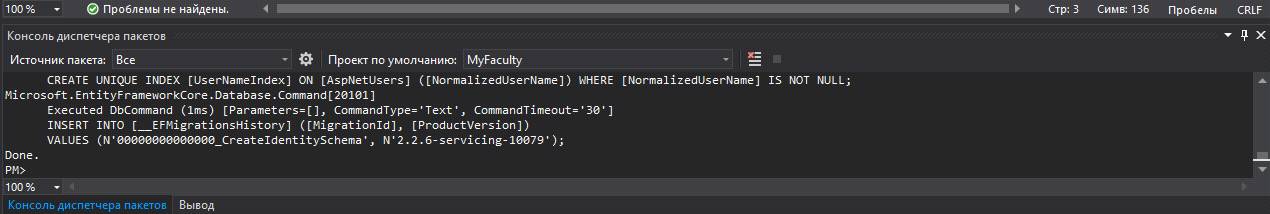
Закройте окно браузера с веб-приложением и в Visual Studio выберите "Средства - Диспетчер пакетов NuGet - Консоль диспетчера пакетов":



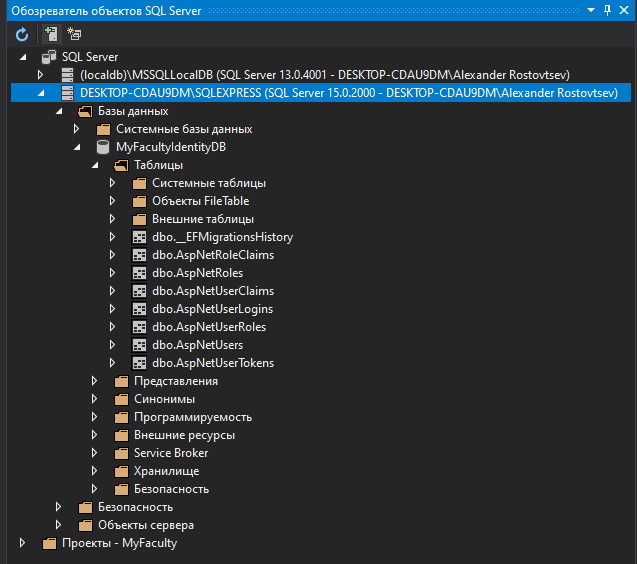
В открывшемся окне необходимо ввести команду "update-database" и нажать Enter:



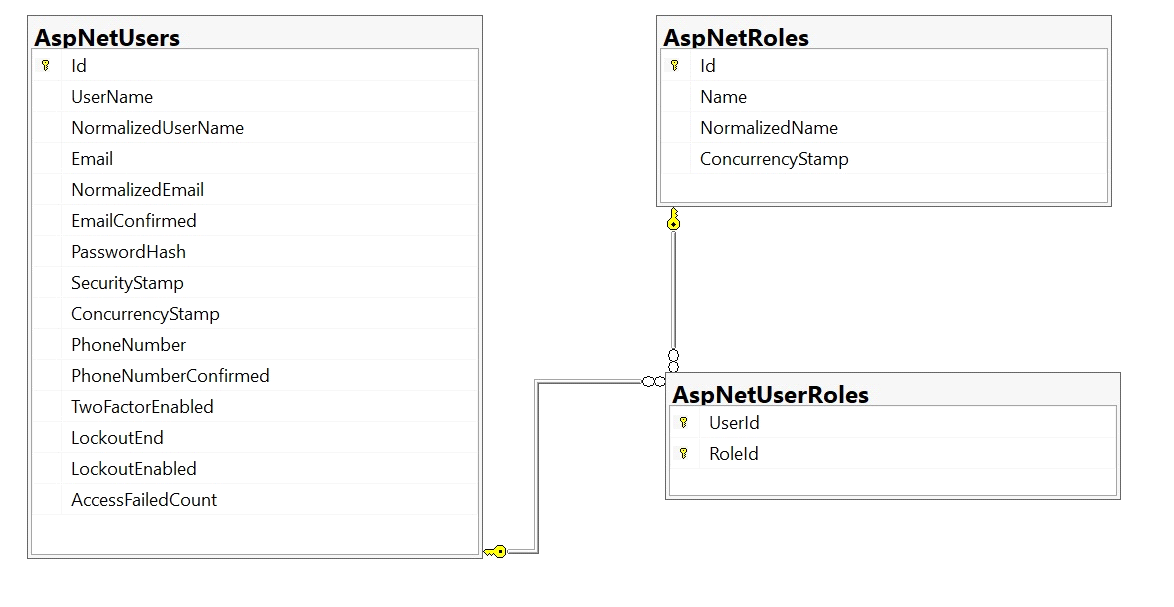
Если строка подключения указана верно то вы получите сообщение об успешном завершении:



А в обозревателе объектов SQL Server на вашем сервере появится новая база данных с именем, которое вы указали в строке подключения:



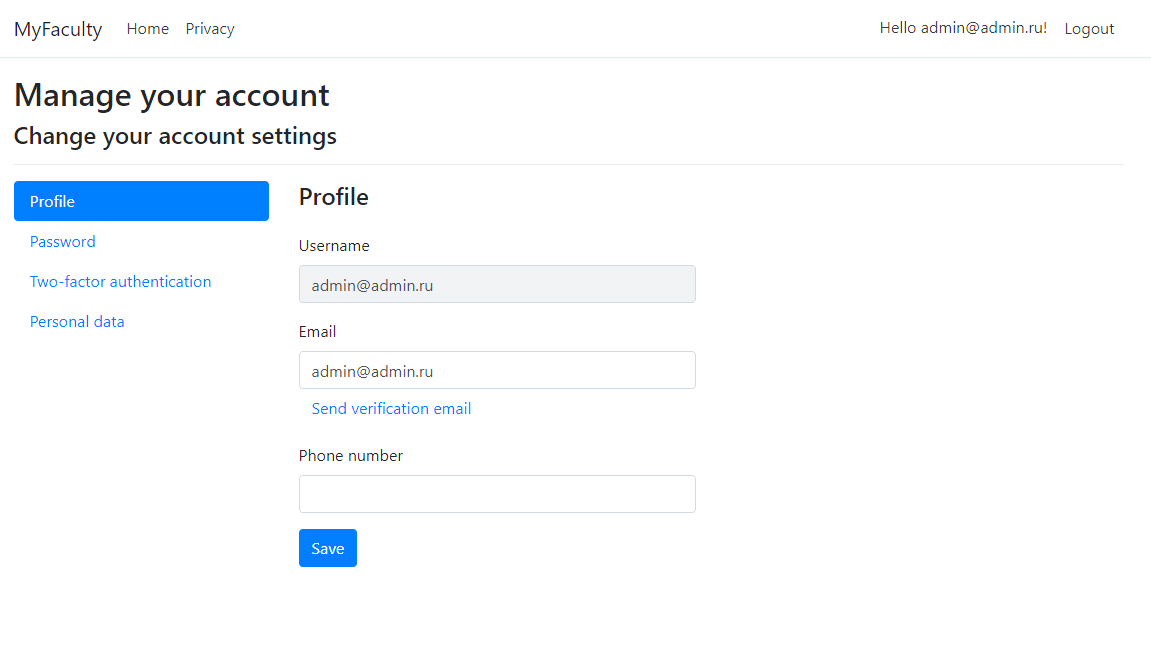
Таблицы AspNetRoles, AspNetUserRoles и AspNetUsers являются основными в этой базе данных, остальные являются вспомогательными и в дальнейшем нас интересовать не будут. Поэтому на данном этапе вы можете добавить в свою схему ещё три таблицы:



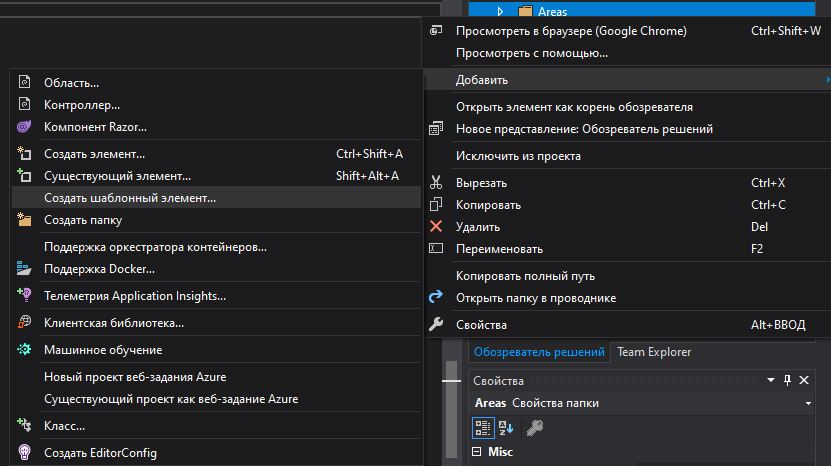
Миграции являются достаточно удобным средством по управлению базой данных без применения таких реляционных сред разработки как   
SQL Server Managment Studio. С помощью подобных инструкций можно удалять, создавать и вносить изменения в существующие базы данных и в их таблицы.   
После применения актуальных миграций SQL Server перейдёт в состояние готовности поддержания корректной работы вашего приложения. Запустите проект ещё раз и зарегистрируйте нового пользователя.

**Пункт 4. ASP.NET Core Identity**

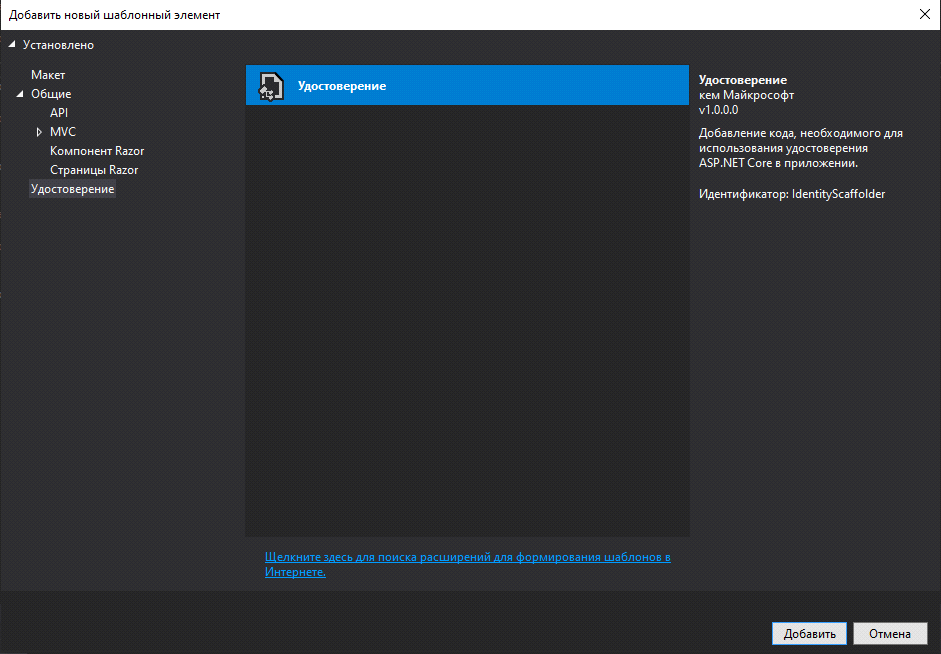
После успешной регистрации вы сможете изучить интерфейс, предоставляемый ASP.NET Core Identity:



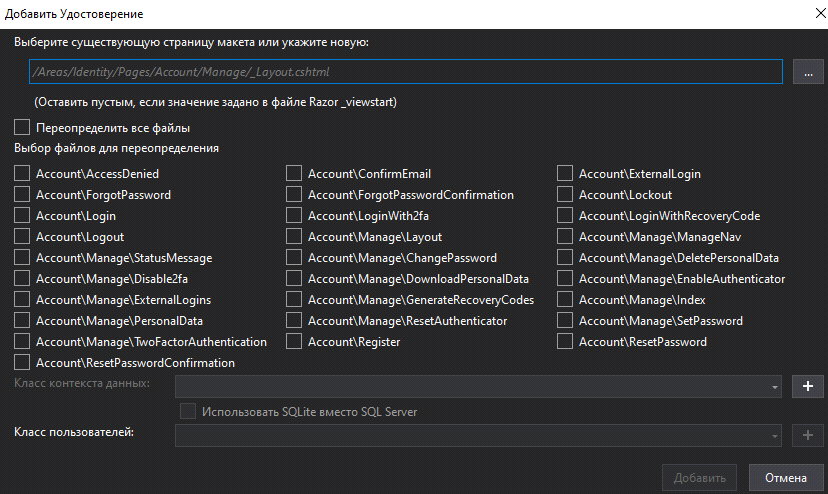
Здесь можно увидеть и возможность верификации вашего почтового адреса, и включение двухфакторной аутентификации, и даже возможно загрузки каких-либо персональных данных. На своё усмотрение вы можете оставить, удалить или внести изменения в предоставленный функционал. Для этого щёлкните правой кнопкой мыши по папке *Areas* в вашем проекте и выберите "Добавить" а затем   
"Создать шаблонный элемент...":



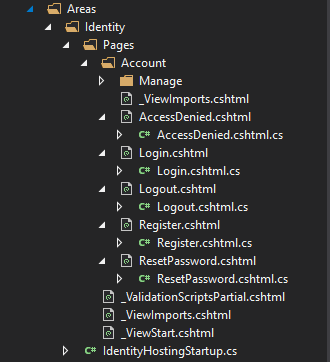
В открывшемся окне нужно будет выбрать "Удостоверение" и нажать "Добавить":



Затем потребуется выбрать пункты, в которые вы захотите внести изменения:



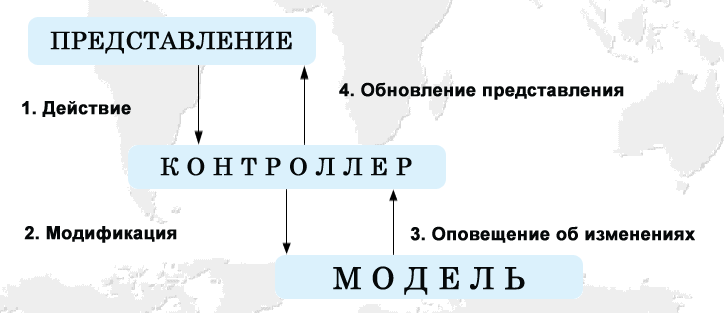
После выбора необходимых вам пунктов нажмите "Добавить". Сразу после этого в дереве проекта в папке *Areas* появятся файлы, отвечающие за выбранные функции решения. Файлы .cshtml содержат в себе внешний вид указанных страниц, а их подфайлы .cs отвечают за функционал страницы:



Так как этот пункт не является обязательным для выполнения работы, и к тому же никак не повлияет на функционал остальной части приложения, ему не будет уделяться пристальное внимание. Студенту предлагается самостоятельно, ориентируясь на прикреплённый архив, по желанию вносить или не вносить изменения в шаблонный функционал, исходя из собственного видения реализации своего приложения.

**Пункт 5. Дополнение архитектуры приложения**

Идеи веб-приложения баз данных с разделением на клиентскую, серверную части и часть базы данных отвечает архитектура MVC   
(Model - View - Controller):



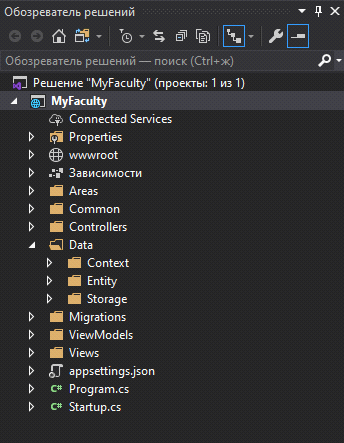
Представление - это страница, написанная на языке HTML, которая отрисовывается в браузере и с которой взаимодействует пользователь. С представления действия пользователя отправляются в контроллер. В дальнейшем эти действия будут называться запросами.

Контроллер - это обработчик какого либо представления или групп представлений. В зависимости от запросов пользователя контроллер совершает манипуляции с данными и модифицирует модель.

Модель - это объект, который формируется в соответствии с сущностью из базы данных, которую необходимо отобразить на представлении путём преобразований. Вместе с изменениями в модели должны происходить и изменения в сущности, путём обратных преобразований.

После изменения модели внешний вид отображаемых данных должен быть обновлён до актуального состояния. Сделать это можно несколькими способами. Проще всего обновить представление. Тогда при загрузки страницы данные заново пройдут путь от сущности в базе данных через модель и контроллер и попадут в представление и гарантированно будут актуальными. В качестве альтернативы можно прибегнуть к JavaScript и менять представление сразу, в зависимости от действий контроллера.

Так архитектура приложения должна быть дополнена следующим образом:



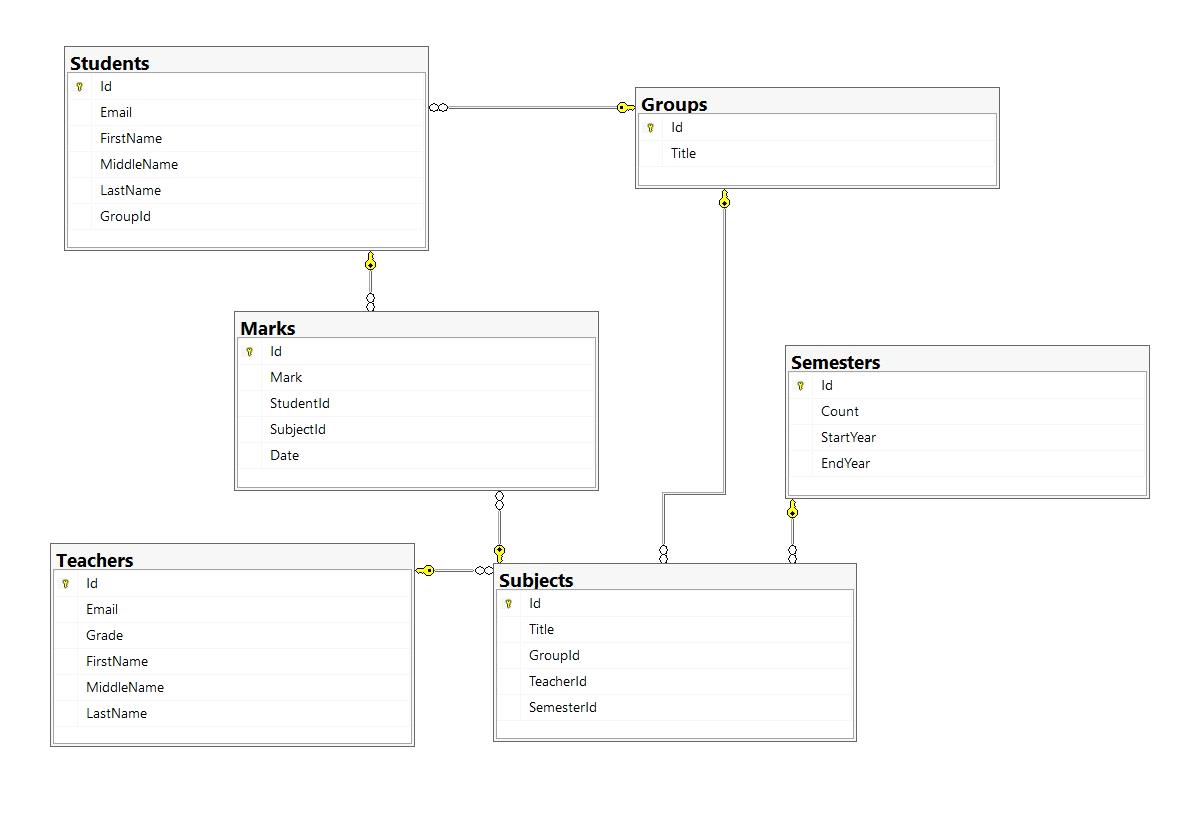
*Controllers* будет содержать в себе все обработчики.

*Data* будет содержать в себе всё, что необходимо для манипуляции с данными:   
*Context -* контексты для общения с базами данных;  
*Entity -* классы, отвечающие сущностям в базах данных;  
*Storage -* классы для манипуляции с данными;

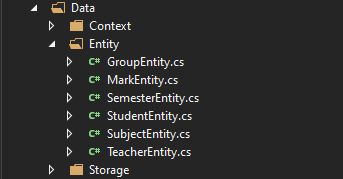
*ViewModels* будет содержать в себе классы моделей;

**Пункт 6. Добавление классов сущностей**

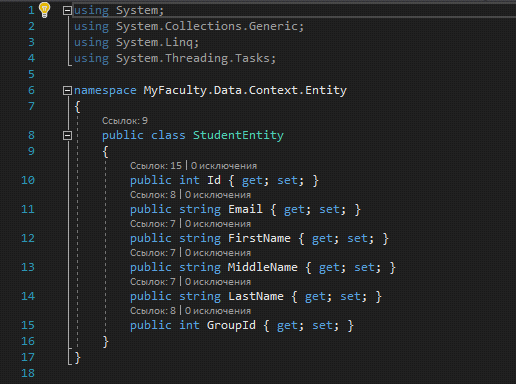
Сперва обратимся к схеме базы данных:



Спроектированная для демонстрационного приложения база данных содержит 6 таблиц. Соответственно на данном этапе в должны быть добавлены 6 классов сущностей:



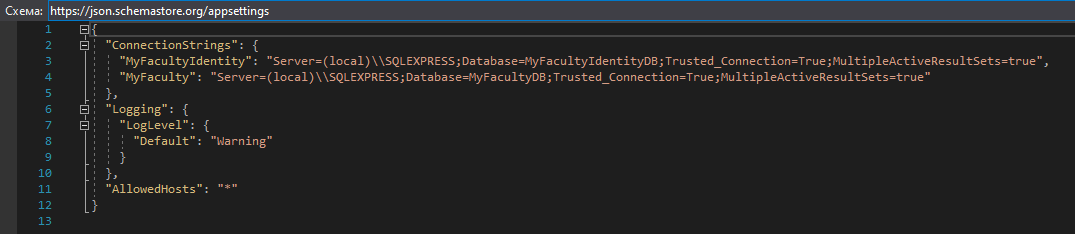
Каждый класс сущности будет соответствовать одной таблице и содержать в себе в качестве свойств только все столбцы таблицы, которой он соответствует.   
Класс сущности студента в качестве примера:



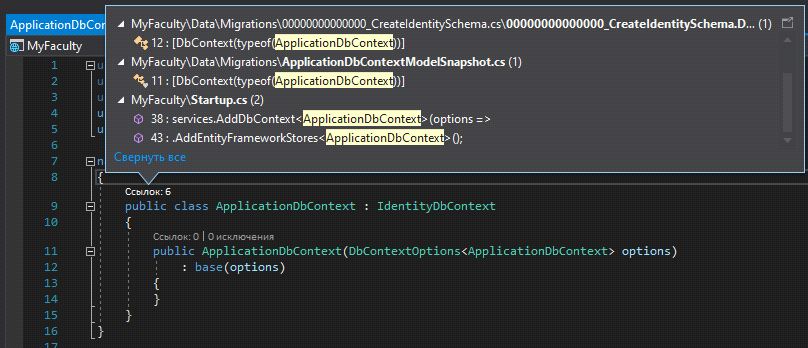
**Пункт 7. Контексты и манипуляции с данными**

Контекст нужен для общения приложения с базой данных. На одну базу данных приходится один контекст. На данном шаге потребуется выполнить сразу несколько действий, соблюдая особенную внимательность:

1. В appsettings.json добавить новую строку подключения для будущей базы данных приложения и переименовать старую, которая относится к базе данных Idenity:

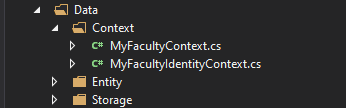


2. Переименовать ApplicationDbContext и поместить его в папку *Context.* ApplicationDbContext является именем по умолчанию для первого контекста приложения. Обратите внимание, что помимо переименования класса контекста необходимо так же переименовать его во всех местах, которые на него ссылаются. В этом может помочь Visual Studio: зайдите в класс контекста и вы увидите сколько на него есть ссылок:

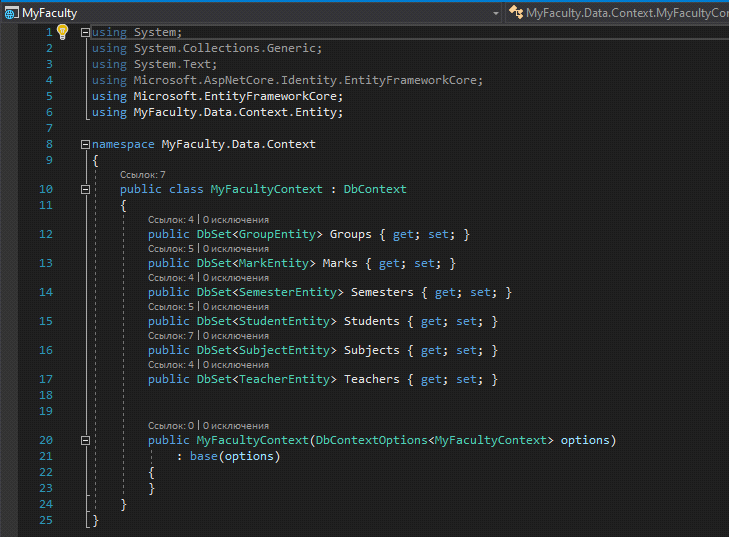


Пройдите по каждой ссылке и переименуйте её. Затем переименуйте имя класса.

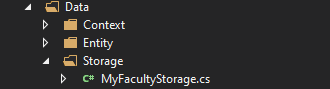
3. Добавьте новый контекст, необходимый для общения с будущей базой данных и поместите его в одну папку с уже существующим:



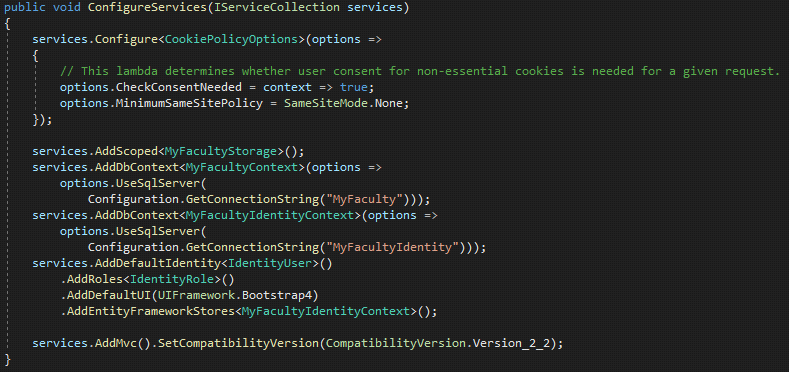
Новый класс контекста должен знать о всех таблицах в базе данных, за которую он отвечает. Для демонстрационного приложения новый контекст выглядит следующим образом:



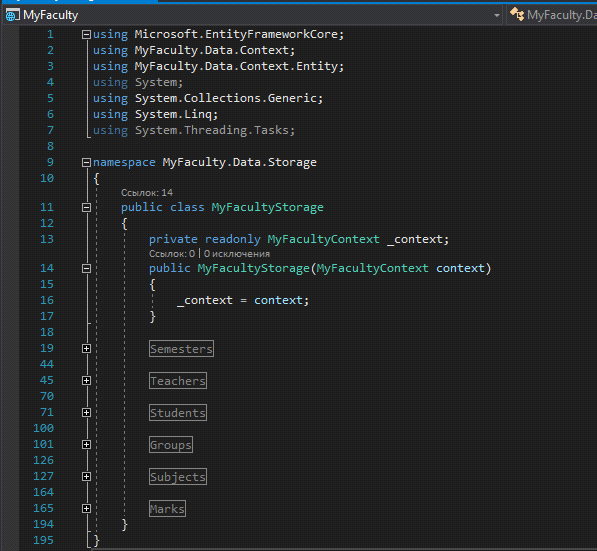
4. Добавить новый класс, отвечающий за манипуляцию с данными:



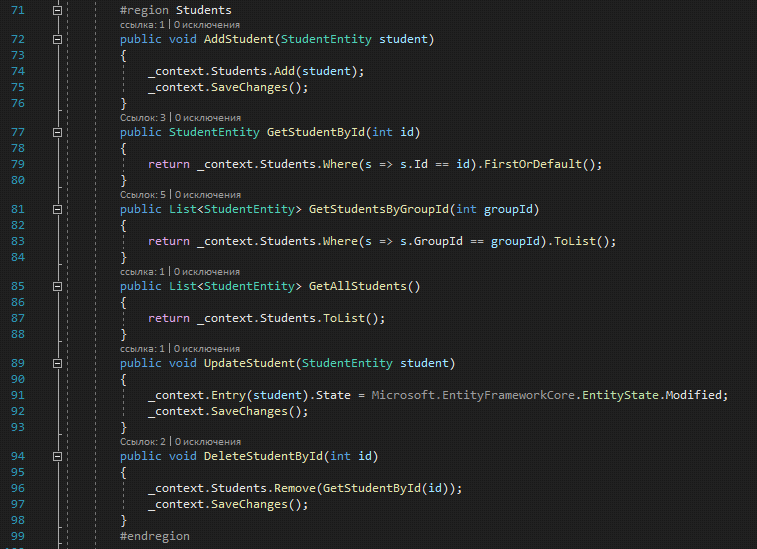
5. Внести в Startup изменения, связанные с переименованием строки подключения и добавить информацию о новом контексте и о классе для манипуляции с данными. Функция ConfigureServices должна быть изменена следующим образом:



6. Заполнить класс для манипуляции с данными необходимыми вам методами и функциями. Для реализации данного класса очень полезен язык запросов LINQ. Прежде чем приступать к данному действию ознакомьтесь с его возможностями.  
После заполнения данный класс должен выглядеть следующим образом:



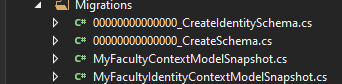
Каждый регион содержит в себе методы, отвечающие за определённую таблицу. Рассмотрим получившиеся методы для таблицы Students:



Стоит отметить, в рамках данного класса следует выполнять только различные CRUD (Create Read Update Delete) операции и не выполнять никаких проверок. Так, например, заботиться о ссылочной целостности и о каскадном удалении будет необходимо вне этого класса, при обращении к его методам и функциям.

**Пункт 8. Работа с миграциями**

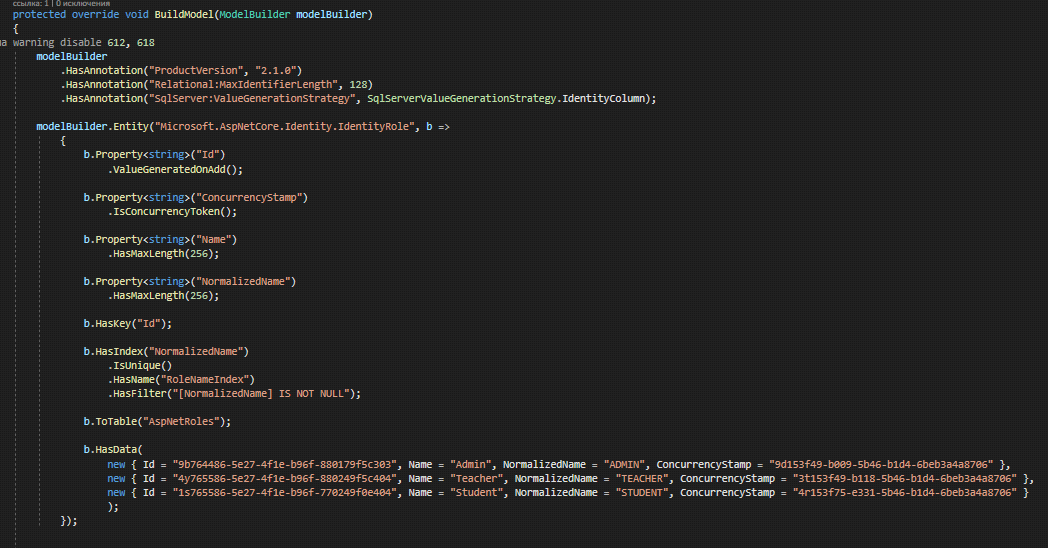
Для начала переименуем уже существующие миграции для базы Identity чтобы не путаться и добавим новые пустые для будущей базы данных приложения:



По задумке база данных приложения изначально должны быть пуста, а база данных Identity должна содержать в себе лишь три роли (учителя, студента и администратора), одну учётную запись для администратора и запись о соответствии единственной учётной записи роли администратора. Поэтому сперва внесём изменения в уже существующую инструкцию для базы данных Idenity. В файле Schema для Identity сразу после инструкций о создании таблицы с ролями пользователей добавим информацию о том, какие записи должны быть добавлены в неё:



После этого инструкции необходимо продублировать в файл ModelSnapshot:

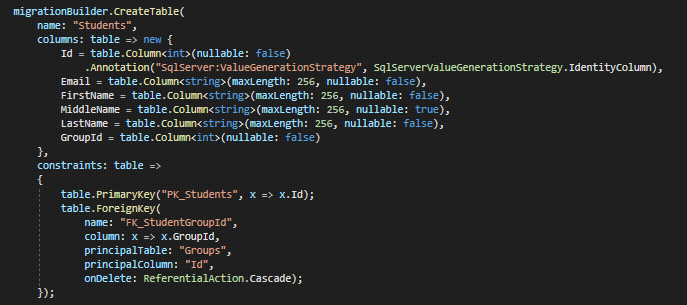


Точно так же по аналогии необходимо сделать для таблицы с пользователями и для таблицы, которая содержит связи многие ко многим между пользователями и ролями. Можно заметить, что записи в таблицах базы Identity содержат в себе GUID в том числе для ID. Если вы затрудняетесь сгенерировать GUID самостоятельно - вы можете скопировать миграции для базы Identity из прикреплённого архива. Так же важно отметить что в таблице с пользователями не хранятся пароли, а лишь их хэш-код. Взять хэш-код для предзаполнения таблицы админской учётной записью вы можете из записи с пользователем, которого мы регистрировали ранее, из поля PasswordHash. Так, например, для пароля **123456Az!** хэш-код будет следующим:

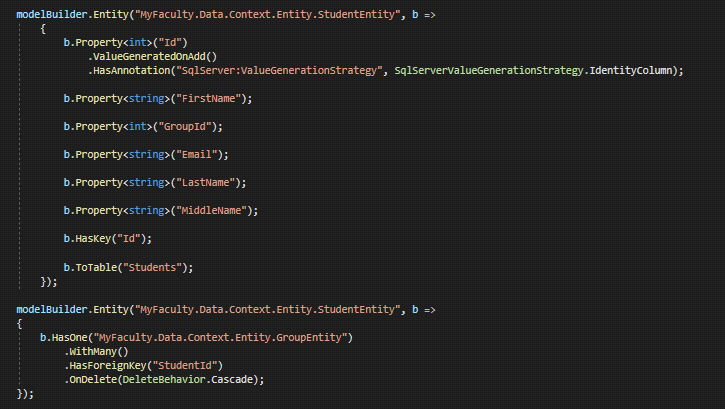
**AQAAAAEAACcQAAAAELR0zVUvO3Qkr0hXT4MoaP/jRrxmndq9p0WK2vVtO6yrsJy3YRrqfx6Qjj172DZBIA==**

Теперь необходимо написать инструкции для создания пустой базы данных для вашего приложения. Сделать это можно по аналогии с миграциями для базы данных Identity - они выйдут очень похожими. Не забывайте про первичные и внешние ключи, согласно вашей схеме базы данных.   
Рассмотрим получившиеся инструкции для создания таблицы со студентами:

Schema:



ModelSnapshot:



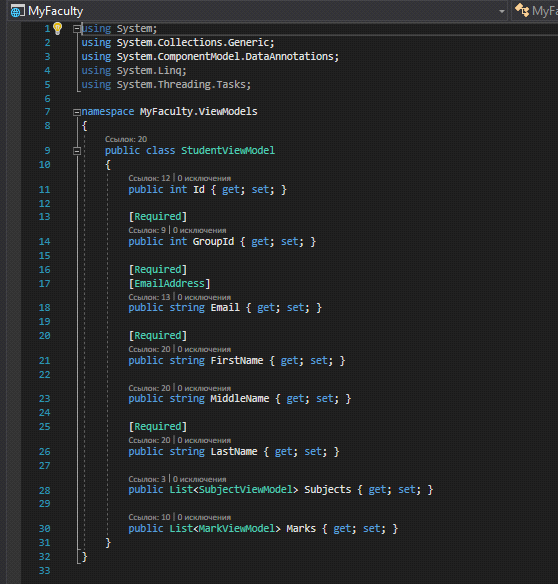
Аналогичные инструкции должны быть написаны для создания каждой таблицы в вашей базе данных. После того, как данный пункт будет завершён будет необходимо будет применить их к SQL Server. Т.к. база Identity на данный момент уже существует. На данном шаге её можно просто удалить и при применении инструкций она будет создана заново и с уже предзаполненной админской учётной записью. Для применения миграций понадобится уже известная нам команда   
update-database, но т.к. мы создали ещё один контекст, данную команду придётся с этого момента вызывать с параметром -Context; Так, чтобы поднять обе базы данных нужно будет ввести две команды:

update-database -Context MyFacultyIdentityContext  
update-database -Context MyFacultyContext

**Пункт 9. Создание моделей**

Как было сказано ранее, любая сущность из базы данных для отображения на представлении должна быть преобразована в модель. Модель может содержать в себе столько же свойств, сколько и сущность, но не меньше. Таким образом, если вам не нужно никаким образом дополнять модель, соответствующую какой-либо сущности то их классы будут иметь одно и то же число свойств, а преобразование из сущности в модель и обратно будет являться тождественным сопоставлением полей. Так или иначе зачастую модель должна быть дополнена чем-либо, поэтому в классе модели будут появляться самые разные дополнительные свойства. Стоит учитывать, что хорошим тоном является наличие только одной модели у каждой сущности. Так, например, в списке группы и в списке с оценками по какому-либо предмету должна быть использована одна и та же модель, за исключением того, что при выводе списка группы не понадобятся оценки студента по какому-либо предмету - за этим следует следить при выполнении преобразований из сущности в модель и оставлять ненужные свойства пустыми.

Рассмотрим в качестве примера получившуюся модель студента:



Помимо имеющихся у сущности свойств у модели появились два дополнительных: список всех предметов, которые относятся к этому студенту и список его оценок. Обратите внимание, что в свойствах, содержащих в себе списки предметов и оценок коллекции имеют тип модели а не сущности.  
 Кроме того в классах моделей можно указать ограничения на получаемые данные. Так, например, свойство Email имеет ограничение Requiered - это означает что оно не должно быть пусто и EmailAdress - это означает, что оно должно пройти проверку на корректность почтового адреса, а свойство MiddleName не имеет никаких ограничений и вообще не является обязательным.

**Пункт 10. Контроллеры и представления**

Для описания работы контроллеров в качестве примера будет использоваться контроллер, отвечающий за часть приложения, связанную с преподавателями - TeacherController. Сразу стоит сделать важное замечание:

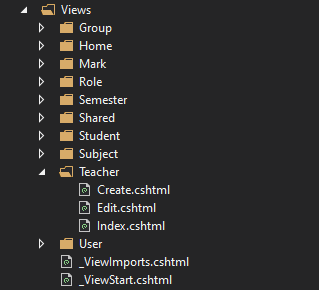
**Имя каждого контроллера должно оканчиваться на "Controller".**

Это очень важное замечание. Если не соблюдать данное условие то класс просто не будет воспринят приложением как контроллер, даже если он будет находиться в папке *Controllers.*Забегая вперёд следует сделать ещё одно не менее важное замечание, касающееся именования:

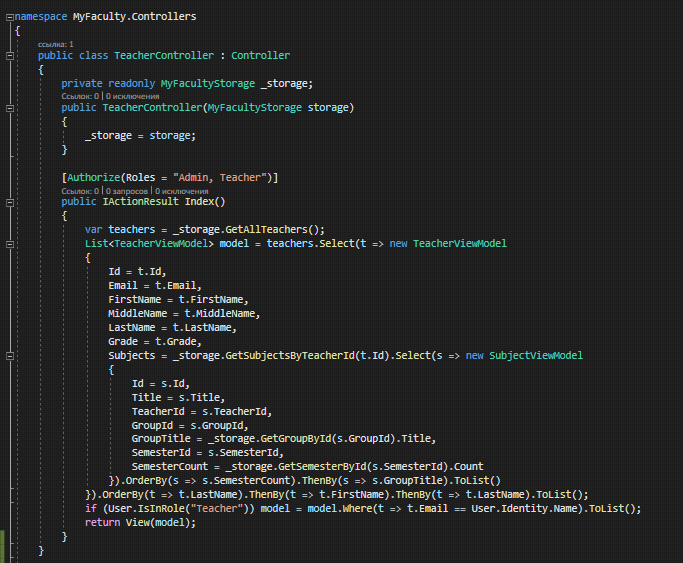
**Название функций и методов контроллера должно совпадать с именами представлений, которые обрабатываются этим контроллером. А название папки с группой представлений - с именем контроллера.**

Тут всё просто. Если у вас есть функция по добавлению нового преподавателя, называющаяся "AddTeacher", то представление, на котором будет находится форма по заполнению информации о новом преподавателе должна называться точно так же, а папка, в которой лежит эта страница - "Teachers".

Таким образом для наглядности создадим сначала пустые представления, связанные с преподавателями:

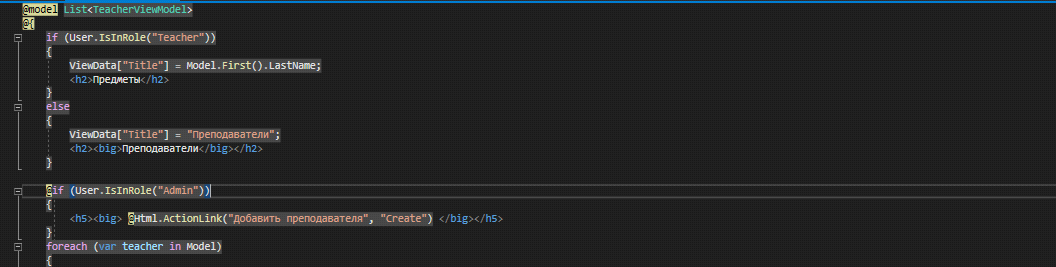


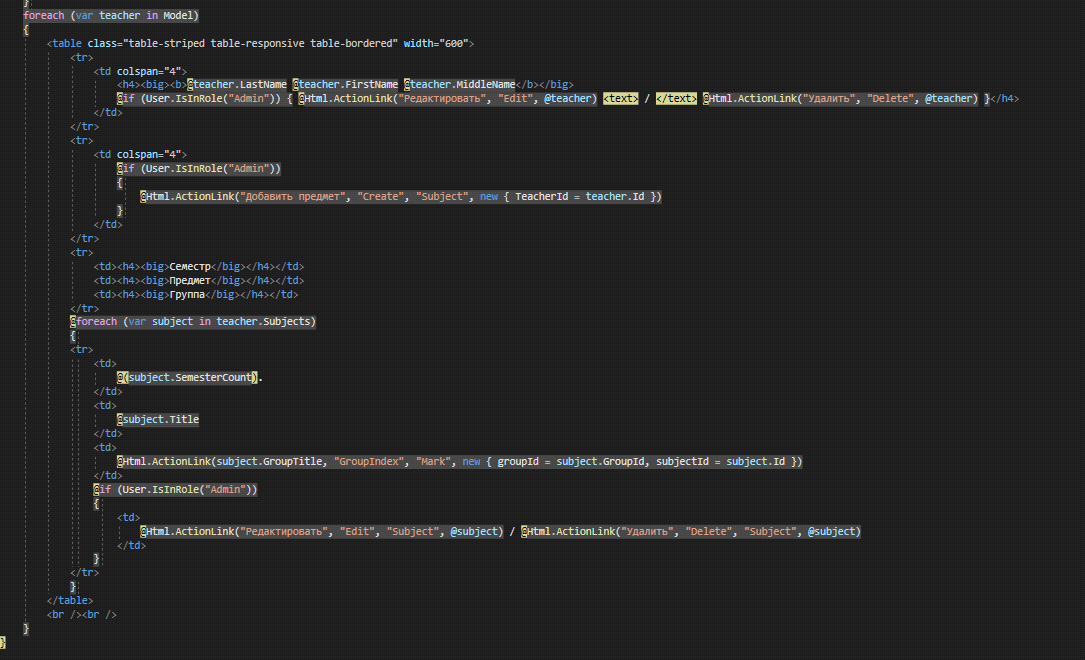
Затем вернёмся в класс TeacherController и создадим его первый метод - Index. Index традиционно является методом контроллера по умолчанию. Он отличается от остальных методов тем, что при обращении к нему адресная строка в браузере не будет дополнительно содержать в себе название метода, а только имя контроллера:



Начнём рассматривать получившийся пример по порядку с самого начала:  
Контроллер в качестве поля содержит в себе класс для манипуляции с данными и имеет публичный конструктор. Конструкторы всех контроллеров вызываются при запуске приложения автоматически, а параметры в них передаются благодаря корректно заполненному ранее файлу Startup.   
Метод контроллера Index содержит ограничение. Данное ограничение похоже на те, что мы видели ранее при создание классов моделей; Оно гласит что для использования данного метода пользователь обязательно должен быть авторизован и к тому же состоять в одной из указанных ролей. Таким образом даже если студент или неавторизованный пользователь введёт в адресную строку корректный Url он не получит никакого результата.   
Результаты действия метода различаются для разных ролей. Вы можете видеть, что условный оператор в предпоследней строчке проверят, является ли авторизованный пользователь преподавателем, и, если да - показывает ему только его данные.

После всех действий происходит возврат представления, именованного таким же образом, как и метод, содержащее в себе получившуюся модель.  
В свою же очередь представление должно ожидать какую-либо модель. Рассмотрим представление, соответствующее методу Index:





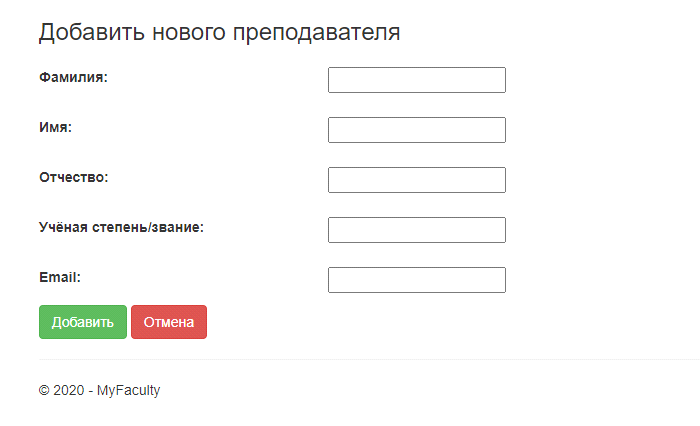
Код представления обязательно должен содержать в себе строчку с объявлением типа модели до первого обращения к ней.

Представления в веб-приложениях ASP.Net Core используют движок Razor, который позволяет использовать код С# при разработке html-страниц. Для вставки кода на языке C# необходимо пользоваться спецсимволом @.   
На данном представлении видите неоднократную проверку роли пользователя ещё раз, но только в этот раз это необходимо для динамической отрисовки представления. Преподаватель не должен видеть админский функционал.

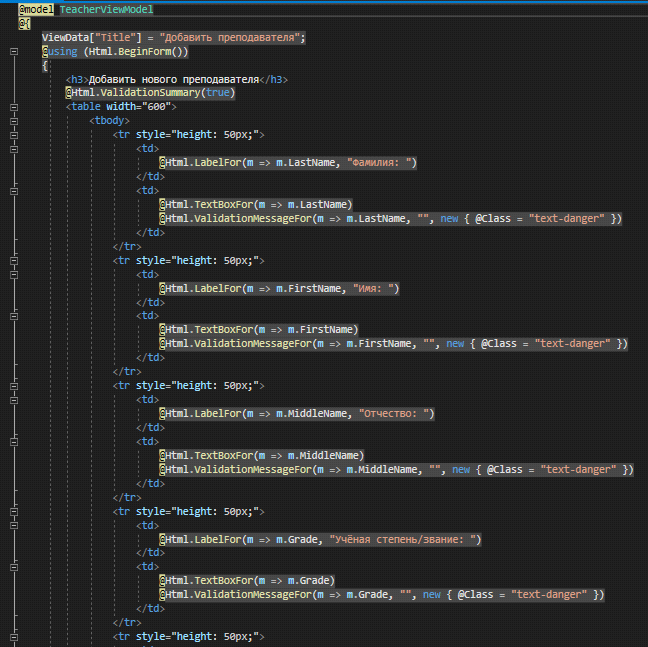
Рассмотрим далее функции и представление, необходимые для добавления нового преподавателя. Начнём с контроллера:

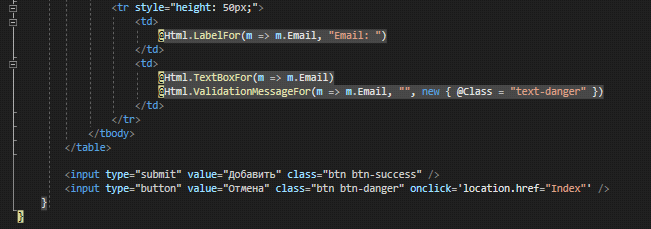


Для реализации добавления необходимо всего два действия. Первое действие является HttpGet методом и всего лишь возвращает страницу с пустой формой для заполнения модели:

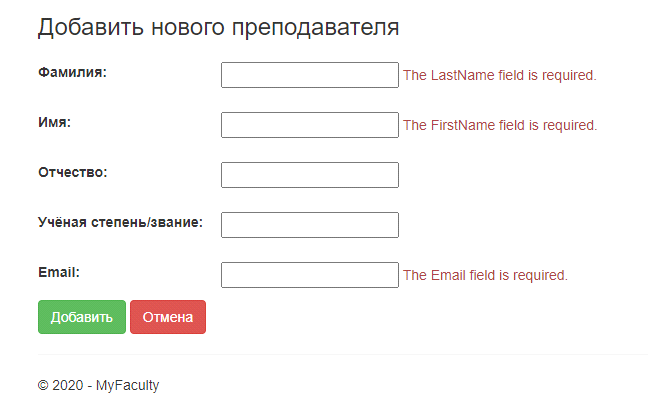


Рассмотрим код данной страницы:



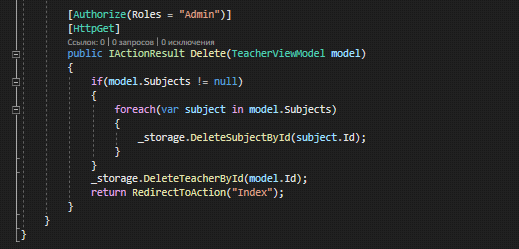


На этой странице используется самая обычная форма с несколькими полями для заполнения свойств модели. Валидация происходит на стороне клиента согласно ограничениям, указанным в классе модели. При попытке добавить невалидную модель вы получите сообщение около каждого невалидного поля:



После того как необходимые поля заполнены по нажатию кнопки с типом submit модель будет передана во в HttpPost метод контроллера, преобразована в сущность и добавлена в базу данных. Методы и представление для редактирования будут выглядеть аналогичным образом за одним исключением - HttpGet метод будет возвращать страницу не для пустой модели, а для уже заполненной.

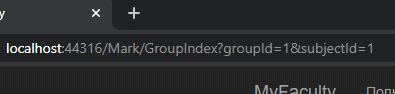
Важное внимание стоит уделить методу для удаления какой-либо записи:



Метод для удаления это всегда HttpGet метод. Как и было сказано ранее, именно при разработке контроллера необходимо следить за соблюдением ссылочной целостности и за необходимыми каскадными удалениями. Таким образом, при удалении какого-либо преподавателя должны быть удалены все предметы, которые имеют внешний ключ, соответствующий этому преподавателю.   
Т.к. удаление, в отличие от добавления и редактирования записей происходит в рамках одной страницы то её содержимое необходимо каким-то образом обновить. Самый простой способ, который и продемонстрирован в примере - обновить страницу.

**Заключение**

После последовательно и внимательного выполнения студентом всех частей и пунктов данного пособия результатом будет веб-приложение, которое отвечает всем трём необходимым частям приложения базы данных. В пособии уделялось внимание лишь самым необходимым и важным частям. Студенту предлагается самостоятельно рассмотреть следующие вопросы, которые не были затронуты ранее:

Улучшение разграничения доступа. Сейчас доступ к действиям контроллеров ограничен по ролям. Например для выставления оценок по своим предметам преподавателю необходимо просто нажать на нужную кнопку в интерфейсе приложения. В то время в адресной строке появится следующая Url-ссылка:  
Подменив параметры groupId и subjectId преподаватель возможно сможет попасть на страницы с оценками по предметам других преподавателй, ведь ограничение выставлено только на роль. Подумайте, как можно исправить данную брешь в безопасности и сделать Url-ссылки менее информативными?

Улучшение интерфейса приложения. Представления демонстрационного приложения созданы с помощью Razor движка и используют HtmlHelpers, чьи возможности почти не были раскрыты. В качестве вёрстки использована самая простая табличная вёрстка. В собственно разработанном приложении сделайте интерфейс более лаконичным и прибегните к использованию css-классов а так же картинок и, возможно, увеличьте объём JavaScript-кода в проекте для расширения возможностей интерфейса. Для размещения картинок, файлов css и JavaScript следует использовать *wwwroot* в дереве проекта:  
