**3 РАЗРАБОТКА ПРИЛОЖЕНИЯ**

Итоговое приложение состоит из 3 основных компонентов:

* серверная часть;
* панель администрирования;
* клиентское приложение для настольного ПК.

**3.1 Разработка серверной части**

Разработка серверной части предусматривает несколько этапов:

1. Разработка базы данных;
2. Разработка архитектуры серверного приложения;
3. Разработка внешнего программного интерфейса приложения.

Серверная часть должна предоставлять следующие возможности:

* внешний пользовательский интерфейс на основе RESTful архитектуры;
* интерфейсы для работы с базой данных;
* системы аутентификации и авторизации;
* CRUD для сущностей модели.

**3.1.1 Разработка базы данных**

В качестве базы данных была выбрана SQL база данных от компании Microsoft – MSSQL. Диаграмма базы данных представлена в Приложении Б.

Таблица 1 – Сводка о таблицах в базе данных

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Схема | Описание | К-во столбцов |
| \_\_MigrationHistory | dbo |  | 4 |
| Criteria | dbo | Таблица, в которой хранятся критерии | 2 |
| ExerciseCriterias | dbo | Таблица, которая определяет связь упражнения с критерием | 4 |
| Exercises | dbo | Таблица, в которой хранятся упражнения | 5 |
| ProfessionCriterias | dbo |  | 4 |
| Professions | dbo | Таблица, в которой хранятся профессии | 3 |
| Roles | dbo | Таблица, в которой определены существующие роли | 3 |
| Settings | dbo | Таблица с персональными настройками пользователя для приложения | 2 |
| TrainingTimes | dbo | Таблица, в которой определено предпочтительное время перерыва для пользователя | 3 |
| UserClaims | dbo | Таблица с характеристиками пользователя | 4 |
| UserExercises | dbo | Упражнения, которые сгенерированы для определенного пользователя и определенной его тренировки | 4 |
| UserLogins | dbo | Таблица, которая определяет связи пользователя с определенным внешним провайдером аутентификации | 3 |
| UserRoles | dbo | Роли пользователя | 2 |
| Users | dbo | Таблица пользователей | 13 |
| UserTrainings | dbo | Таблица тренировок пользователя | 4 |

Таблица 2 – Описание таблицы \_\_MigrationHistory

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Описание | Тип данных | Максимальная длина | Может быть null |
| MigrationId | Идентификатор миграции | nvarchar | 150 |  |
| ContextKey | Ключ контекста Entity Framework | nvarchar | 300 |  |
| Model | Hash-код модели | varbinary | 2147483647 |  |
| ProductVersion | Версия продукта | nvarchar | 32 |  |

Таблица 3 – Описание таблицы Criteria

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Описание | Тип данных | Максимальная длина | Может быть null |
| Id | Идентификатор критерия | int | 4 |  |
| Name | Название критерия | nvarchar | 1073741823 | Да |

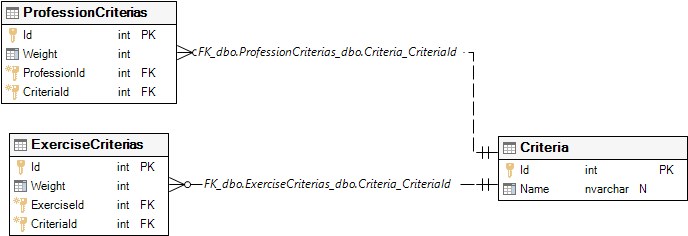


Рисунок 1 – Модель данных в отношении таблицы Criteria

Таблица – Описание таблицы ExerciseCriterias

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Описание | Тип данных | Максимальная длина | Может быть null |
| Id | Идентификатор | int | 4 |  |
| Weight | Вес отношения | int | 4 |  |
| ExerciseId | Идентификатор упражнения | int | 4 |  |
| CriteriaId | Идентификатор критерия | int | 4 |  |

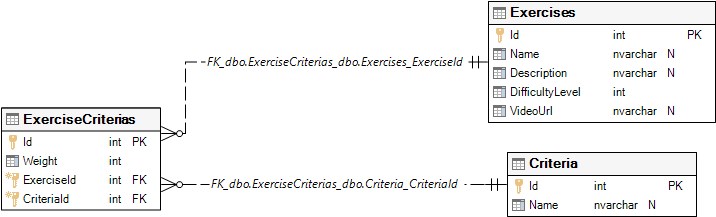


Рисунок 2 – Модель данных в отношении таблицы ExerciseCriterias

Таблица 4 – Описание таблицы Exercises

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Описание | Тип данных | Максимальная длина | Может быть null |
| Id | Идентификатор | int | 4 |  |
| Name | Название упражнения | nvarchar | 1073741823 | Да |
| Description | Описание упражнения | nvarchar | 1073741823 | Да |
| DifficultyLevel | Сложность упражнения. Значения: 0 - Легкое. 1 - Средней сложности. 2 - Тяжелое | int | 4 |  |
| VideoUrl | Ссылка на видео | nvarchar | 1073741823 | Да |

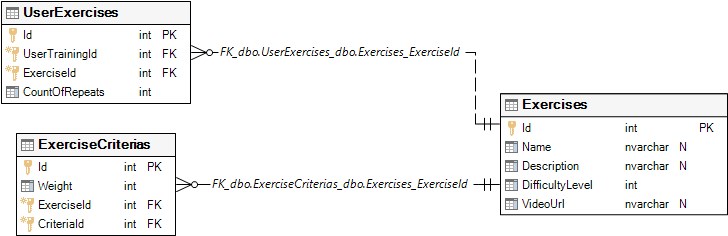


Рисунок 3 – Модель данных таблицы Exercises

Таблица 5 – Описание таблицы ProfessionCriterias

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Описание | Тип данных | Максимальная длина | Может быть null |
| Id | Идентификатор отношения | int | 4 |  |
| Weight | Вес отношения | int | 4 |  |
| ProfessionId | Идентификатор профессии | int | 4 |  |
| CriteriaId | Идентификатор критерия | int | 4 |  |

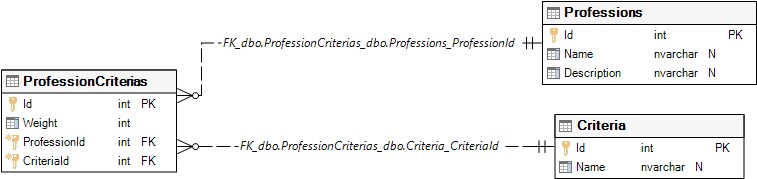


Рисунок 4 – Модель данных таблицы ProfessionCriterias

Талица 6 – Описание таблицы Professions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Описание | Тип данных | Максимальная длина | Может быть null |
| Id |  | int | 4 |  |
| Name | Название профессии | nvarchar | 1073741823 | Да |
| Description | Описание профессии | nvarchar | 1073741823 | Да |

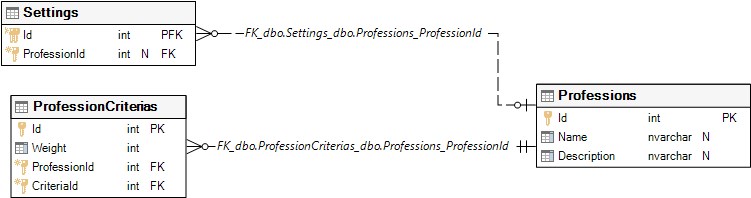
 Рисунок 6 – Модель данных таблицы Professions

Таблица 7 – Описание таблицы Roles

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Описание | Тип данных | Максимальная длина | Может быть null |
| Id | Идентификатор роли | int | 4 |  |
| CreatedUtc | Дата и время создания роли | datetime | 8 |  |
| Name | Название роли | nvarchar | 256 |  |

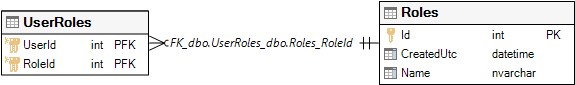


Рисунок 5 – Модель данных таблицы Roles

Таблица 8 – Описание таблицы Settings

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Описание | Тип данных | Максимальная длина | Может быть null |
| Id | Идентификатор настройки. Совпадает с идентификатором пользователя | int | 4 |  |
| ProfessionId | Идентификатор профессии. Настройка может существовать без профессии. Предполагается программное управление наличием професии | int | 4 | Да |

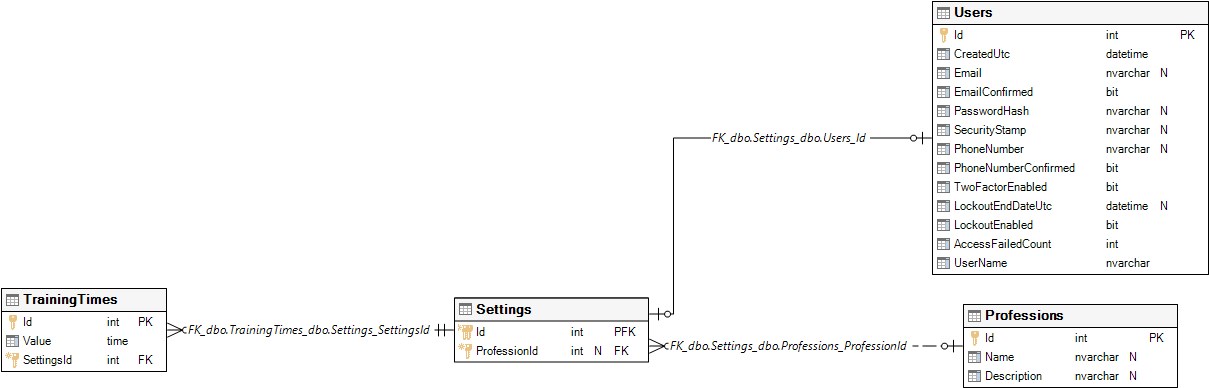


Рисунок 7 – Модель данных таблицы Settings

Таблица 9 – описание таблицы TrainingTimes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Описание | Тип данных | Максимальная длина | Может быть null |
| Id | Идентификатор предустановленного времени | int | 4 |  |
| Value | Значение времени | time | 5 |  |
| SettingsId | Идентификатор настройки, к которой относится это время | int | 4 |  |

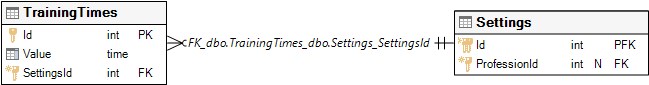


Рисунок 8 – Модель данных таблицы TrainingTimes

Таблица 10 – Описание таблицы UserClaims

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Описание | Тип данных | Максимальная длина | Может быть null |
| Id | Идентификатор характеристики | int | 4 |  |
| UserId | Идентификатор пользователя | int | 4 |  |
| ClaimType | Тип характеристики. Например, роль, возраст и тд. | nvarchar | 1073741823 | Да |
| ClaimValue | Значение характеристики | nvarchar | 1073741823 | Да |

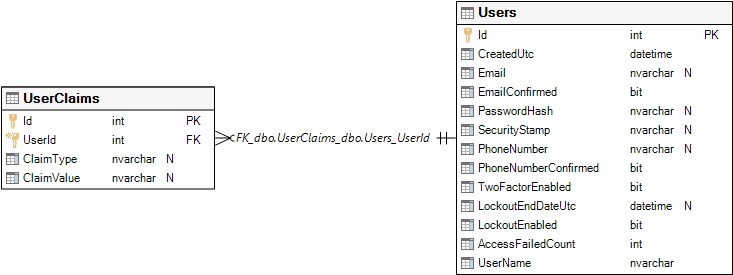


Рисунок 9 – Модель данных таблицы UserClaims

Таблица 11 – Описание таблицы UserExercises

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Описание | Тип данных | Максимальная длина | Может быть null |
| Id | Идентификатор пользовательского упражнения | int | 4 |  |
| UserTrainingId | Идентификатор тренировки | int | 4 |  |
| ExerciseId | Идентификатор упражнения | int | 4 |  |
| CountOfRepeats | Число повторений упражнения | int | 4 |  |

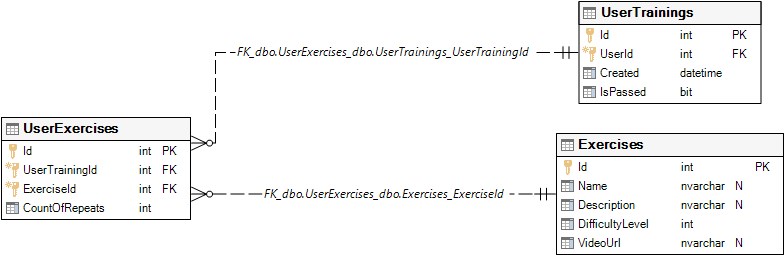


Рисунок 10 – Модель данных таблицы UserExercises

Таблица 12 – Описание таблицы UserLogins

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Описание | Тип данных | Максимальная длина | Может быть null |
| LoginProvider | Определяет логин во вшеншем сервисе | nvarchar | 128 |  |
| ProviderKey | Определяет ключ внешнего сервиса | nvarchar | 128 |  |
| UserId | Идентификатор пользователя | int | 4 |  |

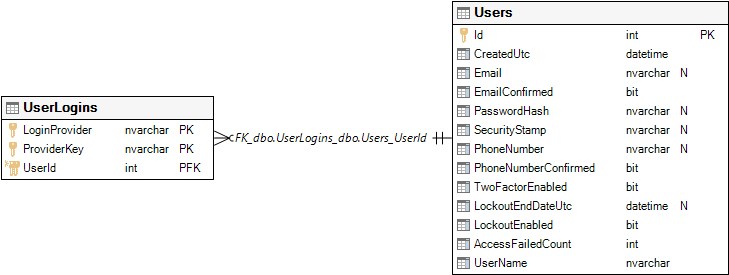


Рисунок 11 – Модель данных таблицы UserLogins

Таблица 13 – Описание таблицы UserRoles

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Описание | Тип данных | Максимальная длина | Может быть null |
| UserId | Идентификатор пользователя | int | 4 |  |
| RoleId | Идентификатор роли | int | 4 |  |

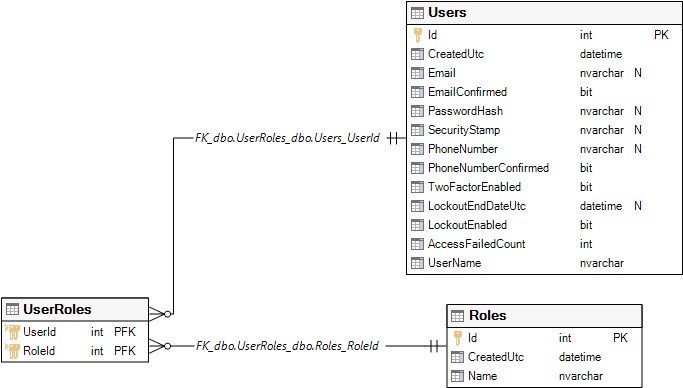


Рисунок 12 – Модель данных таблицы UserRoles

Таблица 14 – Описание таблицы Users

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Описание | Тип данных | Максимальная длина | Может быть null |
| Id | Идентификатор пользователя | int | 4 |  |
| CreatedUtc |  | datetime | 8 |  |
| Email |  | nvarchar | 256 | Да |
| EmailConfirmed |  | bit | 1 |  |
| PasswordHash | Хеш пороля пользователя | nvarchar | 1073741823 | Да |
| SecurityStamp |  | nvarchar | 1073741823 | Да |
| PhoneNumber |  | nvarchar | 1073741823 | Да |
| PhoneNumberConfirmed |  | bit | 1 |  |
| TwoFactorEnabled |  | bit | 1 |  |
| LockoutEndDateUtc |  | datetime | 8 | Да |
| LockoutEnabled |  | bit | 1 |  |
| AccessFailedCount |  | int | 4 |  |
| UserName | Имя пользователя | nvarchar | 256 |  |

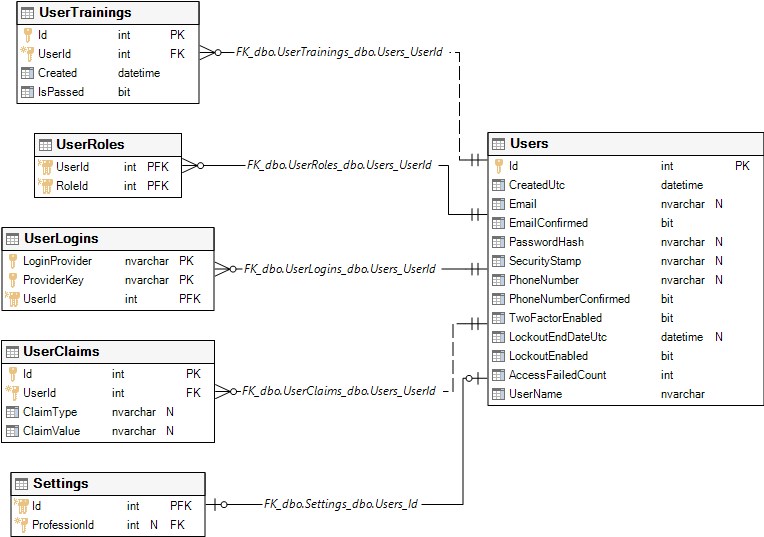


Рисунок 13 – Модель данных таблицы Users

Таблица 15 – Описание таблицы UserTrainings

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Описание | Тип данных | Максимальная длина | Может быть null |
| Id | Идентификатор тренировки | int | 4 |  |
| UserId | Идентификатор пользователя | int | 4 |  |
| Created | Время создания тренировки | datetime | 8 |  |
| IsPassed | Определяет, пройдена тренировка или нет | bit | 1 |  |

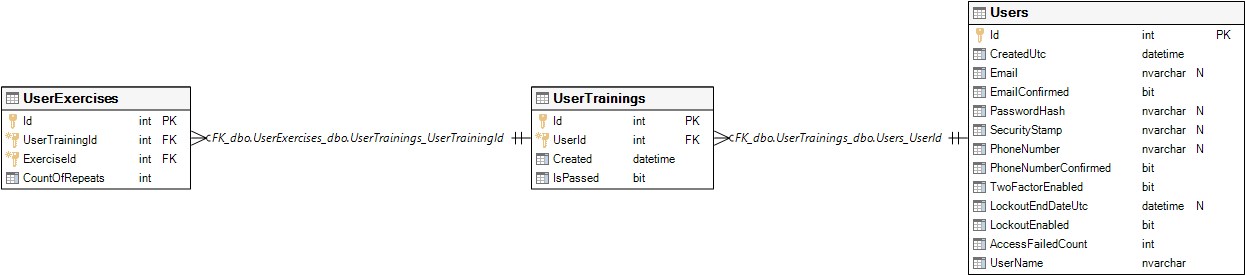


Рисунок 14 – Модель данных таблицы UserTrainings

**3.1.2 Разработка архитектуры серверного приложения**

Языком разработки дипломного проекта является C#, а основной технологией является ASP.NET MVC. Одним из наиболее подходящих шаблонов архитектурного проектирования для данного набора технологий и типа приложений является «Многоуровневая архитектура». Каждый уровень обеспечивает необходимый уровень абстракции, обычно, таких уровня 3.

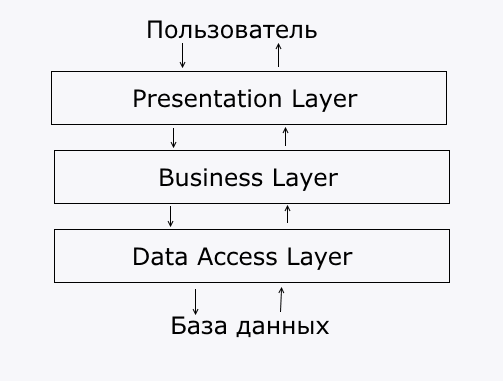


Рисунок 15 – Пример трехуровневой архитектуры

Presentation layer  ( или уровень представления) – это тот уровень, с которым непосредственно взаимодействует пользователь. Этот уровень включает компоненты пользовательского интерфейса, механизм получения ввода от пользователя. Применительно к asp.net mvc на данном уровне расположены представления и все те компоненты, который составляют пользовательский интерфейс (стили, статичные страницы html, javascript), а также модели представлений, контроллеры, объекты контекста запроса. Так же на этом слое располагается внешний программный интерфейс приложения.

Business layer ( или уровень бизнес-логики) содержит набор компонентов, которые отвечают за обработку полученных от уровня представлений данных, реализует всю необходимую логику приложения, все вычисления, взаимодействует с базой данных и передает уровню представления результат обработки.

Data Access layer ( или уровень доступа к данным) хранит модели, описывающие используемые сущности, также здесь размещаются специфичные классы для работы с разными технологиями доступа к данным, например, класс контекста данных Entity Framework. Здесь также хранятся репозитории, через которые уровень бизнес-логики взаимодействует с базой данных.

Для обеспечения максимально высокого уровня расширяемости и поддержки хорошим выбором будут паттерны проектирования «Репозиторий» и «Единица работы».

Паттерн «Репозиторий» позволяет абстрагироваться от конкретных подключений к источникам данных, с которыми работает программа, и является промежуточным звеном между классами, непосредственно взаимодействующими с данными, и остальной программой.

Паттерн «Единица работы» позволяет упростить работу с различными репозиториями и дает уверенность, что все репозитории будут использовать один и тот же контекст данных.

Для предоставление внешнего программного интерфейса приложения в данном дипломном проекте используется технология ASP.NET Web API 2.

Web API представляет собой веб-службу, которая может взаимодействовать с различными приложениями. При этом приложение может быть веб-приложением ASP.NET, либо может быть мобильным или обычным десктопным приложением.

Также надо отметить, что платформа Web API 2 не является частью фреймворка ASP.NET MVC и может быть задействована как в связке с MVC, так и в соединении с Web Forms. Поэтому в Web API имеется своя система версий. Так, первая версия появилась с .net 4.5. А вместе с .NET 4.5.1 и MVC 5 вышла Web API 2.0.

Для связи с базой данных используется ORM (объектно-реляционное отображение) EntityFramework 6.

Entity Framework представляет специальную объектно-ориентированную технологию на базе фреймворка .NET для работы с данными. Если традиционные средства ADO.NET позволяют создавать подключения, команды и прочие объекты для взаимодействия с базами данных, то Entity Framework представляет собой более высокий уровень абстракции, который позволяет абстрагироваться от самой базы данных и работать с данными независимо от типа хранилища. Если на физическом уровне мы оперируем таблицами, индексами, первичными и внешними ключами, но на концептуальном уровне, который нам предлагает Entity Framework, мы уже работает с объектами.

Центральной концепцией Entity Framework является понятие сущности или entity. Сущность представляет набор данных, ассоциированных с определенным объектом. Поэтому данная технология предполагает работу не с таблицами, а с объектами и их наборами.

Любая сущность, как и любой объект из реального мира, обладает рядом свойств. Например, если сущность описывает человека, то мы можем выделить такие свойства, как имя, фамилия, рост, возраст, вес. Свойства необязательно представляют простые данные типа int, но и могут представлять более комплексные структуры данных. И у каждой сущности может быть одно или несколько свойств, которые будут отличать эту сущность от других и будут уникально определять эту сущность. Подобные свойства называют ключами.

При этом сущности могут быть связаны ассоциативной связью один-ко-многим, один-ко-одному и многие-ко-многим, подобно тому, как в реальной базе данных происходит связь через внешние ключи.

Отличительной чертой Entity Framework является использование запросов LINQ для выборки данных из БД. С помощью LINQ мы можем не только извлекать определенные строки, хранящие объекты, из бд, но и получать объекты, связанные различными ассоциативными связями.

Другим ключевым понятием является Entity Data Model. Эта модель сопоставляет классы сущностей с реальными таблицами в БД.

Entity Data Model состоит из трех уровней: концептуального, уровень хранилища и уровень сопоставления (маппинга).

На концептуальном уровне происходит определение классов сущностей, используемых в приложении.

Уровень хранилища определяет таблицы, столбцы, отношения между таблицами и типы данных, с которыми сопоставляется используемая база данных.

Уровень сопоставления (маппинга) служит посредником между предыдущими двумя, определяя сопоставление между свойствами класса сущности и столбцами таблиц.

Таким образом, мы можем через классы, определенные в приложении, взаимодействовать с таблицами из базы данных.

Entity Framework предполагает три возможных способа взаимодействия с базой данных:

* Database First. Entity Framework создает набор классов, которые отражают модель конкретной базы данных;
* Model First. Сначала разработчик создает модель базы данных, по которой затем Entity Framework создает реальную базу данных на сервере;
* Code First. Разработчик создает класс модели данных, которые будут храниться в БД, а затем Entity Framework по этой модели генерирует базу данных и ее таблицы.

В данной дипломной работе использовался подход Code First. Тем не менее, имея схему базы данных, описанную выше, имеется возможность восстановить модель, используя подход Database First.

В результате серверная часть приложения состоит из следующих наборов библиотек:

* Services.DTO;
* Services.Implementations;
* Services.Interfaces;
* Data.Extensions;
* Data.Implementations;
* Data.Interfaces;
* Contracts;
* Entity;
* WebUI.

Рассмотрим каждую из библиотек в отдельности.

Services.DTO предоставляет наборы DTO (data transfer object) для каждой сущности модели. Это позволяет обеспечить необходимый уровень абстракции и избежать зависимостей от сущностей DAL слоя.

Services.Implementations предоставляет реализации для интерфейсов библиотеки Services.Interfaces. Services.Interfaces в свою очередь предоставляет внешний API для уровня презентации и Web API.

Data.Extensions предоставляет набор расширений для стандартных функций EntityFramework, в частности, пагинацию и загрузку бесконечных списков.

Data.Implementations предоставляет реализации интерфейсов библиотеки Data.Interfaces. Помимо этого включает в себя всю необходимую информацию о базе данных, такую, как миграции и контекст. Здесь же реализованы паттерны «Репозиторий» и «Единица работы».

Contracts представляет из себя библиотеку контрактов. Эти контракты позволяют делать приложение более надежным в разработке и избежать большого числа проверок данных.

Entity включает в себя полную модель данных.

WebUI включает в себя интерфейсы Web API для приложения для настольного ПК, для панели администратора и для мобильного приложения. Здесь реализованы механизмы аутентификации и авторизации.

WebUI.Tests содержит в себе тесты для контроллеров WebAPI.

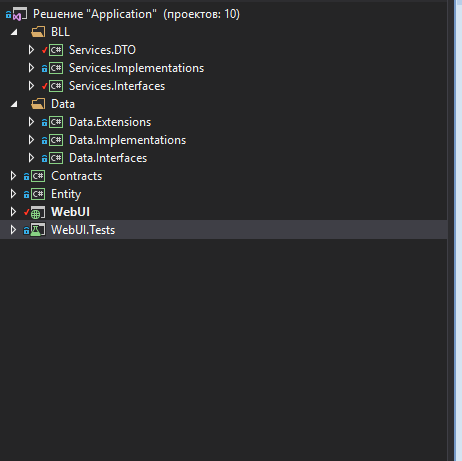


Рисунок 16 – Отображение структуры проекта в окне Обозревателя решений Visual Studio 2017

**3.2 Разработка панели администрирования**

Панель администрирования должна предоставлять интерфейс и реализацию для:

* регистрации и входу в учетную запись;
* созданию, редактированию и удалению критерий;
* созданию, редактированию и удалению профессий;
* созданию, редактированию и удалению упражнений.

В Приложении Б отображена диаграмма деятельности для основной функции, которую должна обеспечивать административная панель – создание и редактирование упражнений.

В Приложении А отображена диаграмма возможностей пользователя административной панели.

**3.2.1 Разработка программного интерфейса**

Панель администрирования разработана с помощью JavaScript библиотеки VueJs.

Веб-приложения должны представлять графический интерфейс для работы с серверной частью. Работа с серверной частью осуществляется через Web API. Для работы с серверной частью используется библиотека Axios, которая представляет собой полноценный REST клиент.

REST (или «передача состояния представления») это [архитектурный стиль](https://ru.wikipedia.org/wiki/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F) взаимодействия компонентов распределённого приложения в [сети](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C). REST представляет собой согласованный набор ограничений, учитываемых при проектировании распределённой [гипермедиа](https://ru.wikipedia.org/wiki/%D0%93%D0%B8%D0%BF%D0%B5%D1%80%D0%BC%D0%B5%D0%B4%D0%B8%D0%B0)-системы. В определённых случаях ([интернет-магазины](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82-%D0%BC%D0%B0%D0%B3%D0%B0%D0%B7%D0%B8%D0%BD" \o "Интернет-магазин), [поисковые системы](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B8%D1%81%D0%BA%D0%BE%D0%B2%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), прочие системы, основанные на данных) это приводит к повышению производительности и упрощению архитектуры. В широком смысле компоненты в REST взаимодействуют наподобие взаимодействия клиентов и серверов во [Всемирной паутине](https://ru.wikipedia.org/wiki/%D0%92%D1%81%D0%B5%D0%BC%D0%B8%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D1%83%D1%82%D0%B8%D0%BD%D0%B0). REST является альтернативой [RPC](https://ru.wikipedia.org/wiki/%D0%A3%D0%B4%D0%B0%D0%BB%D1%91%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B2%D1%8B%D0%B7%D0%BE%D0%B2_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D0%B4%D1%83%D1%80).

В сети [Интернет](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82) [вызов удалённой процедуры](https://ru.wikipedia.org/wiki/%D0%A3%D0%B4%D0%B0%D0%BB%D1%91%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B2%D1%8B%D0%B7%D0%BE%D0%B2_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D0%B4%D1%83%D1%80) может представлять собой обычный [HTTP](https://ru.wikipedia.org/wiki/HTTP)-запрос (обычно «GET» или «POST»; такой запрос называют «REST-запрос»), а необходимые данные передаются в качестве [параметров](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%BC%D0%B5%D1%82%D1%80) запроса.

Для [веб-служб](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%BB%D1%83%D0%B6%D0%B1%D0%B0), построенных с учётом REST (то есть не нарушающих накладываемых им ограничений), применяют термин «RESTful».

В RESTful-сервисах данные обычно передаются в формате JSON.

**3.2.2 Разработка структуры компонентов**

Внедрение компонентного подхода (в целом) — это то, как обычно решается такая задача. Любая компонентная система должна уменьшать общую сложность через предоставление изоляции, или естественных барьеров, скрывающих сложность одних систем от других. Хорошая изоляция также облегчает повторное использование и внедрение сервисных парадигм.

Современные веб-приложения столь же сложны, как и любые другие программные приложения, и зачастую создаются несколькими людьми, объединяющими усилия для создания финального продукта. В таких условиях, чтобы повысить эффективность, естественно искать правильные способы разделения работы на участки с минимальными пересечениями между людьми и подсистемами. Внедрение компонентного подхода (в целом) — это то, как обычно решается такая задача. Любая компонентная система должна уменьшать общую сложность через предоставление изоляции, или естественных барьеров, скрывающих сложность одних систем от других. Хорошая изоляция также облегчает повторное использование и внедрение сервисных парадигм.  
 Изначально сложность веб-приложений в основном регулировалась со стороны сервера за счет разделения приложения на отдельные страницы, что требовало от пользователя соответствующим образом переходить в браузере с одной страницы на другую. С внедрением AJAX и связанных технологий разработчики смогли отказаться от потребности делать «переходы» между разными страницами веб-приложения. Для типичных сценариев вроде чтения почты или новостей ожидания пользователей изменились. К примеру, после логина в почту, вы можете «пользоваться почтовым приложением» с одного и того же адреса (URL) и находится на этой странице целый день (т.н. [Single-Page Applications](https://en.wikipedia.org/wiki/Single-page_application), SPA). Логика клиентских веб-приложений в таких ситуациях существенно усложняется, иногда она даже становится сложнее, чем на серверной стороне. Возможным разрешением данной сложности может являться дальнейшее разделение на компоненты и изоляция логики внутри одной страницы или документа.  
 Цель веб-компонентов в уменьшении сложности за счет изоляции связанных групп кода на HTML, CSS и JavaScript для выполнения общей функциональности в пределах контекста одной страницы.

Так как веб-компоненты должны связать воедино HTML, CSS и JavaScript, необходимо учитывать модели изоляции присущие каждой из технологий:

* Изоляция стилей в CSS;
* JavaScript и области видимости (замыкания);
* Изоляция глобального объекта

Все эти механизмы покрывает система однофайловых компонентов VueJs.

[Компоненты](https://ru.vuejs.org/v2/guide/components.html) развивают идею плагинов. Каждый из них реализует какую-то свою возможность (а если нет существующих, то можно написать и свой). Если понадобится реализовать подобное в другом месте — легко переиспользовать плагин снова. Взаимодействие можно описать простым интерфейсом: отправляем в плагин входные параметры, а для обратной связи можем отслеживать события.

Всё это справедливо и для компонентов. С тем лишь отличием, что компонент может представлять собой не только одну вещь (например, красивый и функциональный select), но и какую-то часть приложения, которая должна работать и выглядеть везде единообразно (например, форма комментирования, с аватаркой, редактором и красивым select’ом).

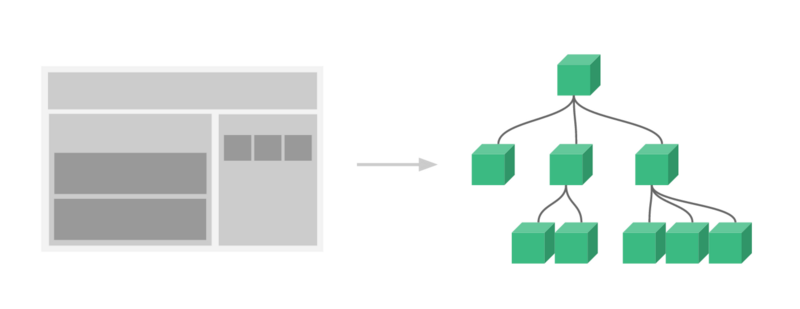


Рисунок 17 – Иллюстрация разделения страницы на компоненты

Компонентный подход позволяет избежать мешанины кода и чётко выстраивать архитектуру приложения. Любую сложную страницу всегда можно разбить на меньшие составляющие. Каждую из таких частей при выделении в компонент проще поддерживать, а при необходимости повторять разбиение внутри компонента на ещё меньшие части.

Первым заметным плюсом подобного разбиения на компоненты будет удобство в поддержке — больше не нужно держать в голове логику всего приложения, можно сосредоточиться на конкретной его части. Вносить изменения или доработки нужно будет только в одном месте. Изолированность же компонентов избавит от появления конфликтов с другими частями приложения.

Поэтому применение компонентного подхода теперь широко используется во многих фреймворках. Vue не остался в стороне и предоставляет прекрасные возможности по работе с компонентами.

Компонент и его структура

В разметке компоненты выглядят как нестандартные теги (например, <comment-form></comment-form>). Но компонент, в отличии от обычного тега, может “скрывать” под собой не только любую разметку, но и какую-то логику своей работы и использование других компонентов.

Описывать составляющие компонентов можно просто в коде страниц. В этом случае есть [некоторые ограничения](https://ru.vuejs.org/v2/guide/components.html#%D0%9E%D1%81%D0%BE%D0%B1%D0%B5%D0%BD%D0%BD%D0%BE%D1%81%D1%82%D0%B8-%D0%BF%D0%B0%D1%80%D1%81%D0%B8%D0%BD%D0%B3%D0%B0-DOM-%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D0%B0) по использованию и именованию (для браузера всё должно быть валидно). Vue также [имеет поддержку JSX](https://ru.vuejs.org/v2/guide/render-function.html#JSX).

Для удобной организации компонентов и их составляющих предлагается использовать [vue-файлы](https://ru.vuejs.org/v2/guide/single-file-components.html). Такой файл состоит из трёх частей:

* <template> для шаблона с разметкой компонента;
* <script> с логикой компонента;
* <style> со стилями компонента.

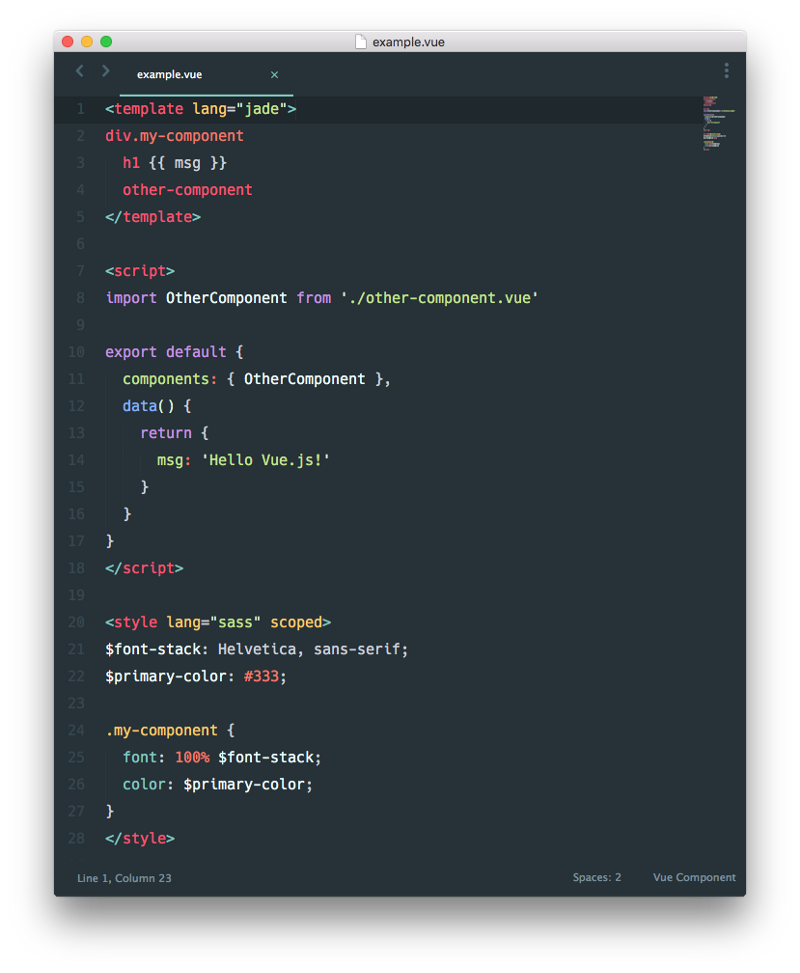


Рисунок 18 – Пример файла компонента.

**3.3 Разработка приложения для настольных ПК**

Приложение для настольных ПК должно предоставлять интерфейс и реализацию для:

* регистрации и входу в учетную запись;
* просмотру всех упражнений;
* настройки профессии;
* настройки предпочтительного времени;
* оповещения о необходимости провести профилактическую тренировку;
* просмотра и отслеживания результатов выполнения тренировки.

В Приложении Б отображена диаграмма деятельности для основной функции, которую должно обеспечить приложение для настольного ПК – выполнение профилактической разминки.

В Приложении А отображена диаграмма возможностей пользователя приложения для настольного ПК.

**3.3.1 Разработка программного интерфейса**

Работа программного интерфейса очень схожа с работой программного интерфейса административной части. Т.к. для разработки приложения для настольных ПК используется Electron, то есть возможность применить библиотеку Axios, которая реализует RESTful клиент.

Основной особенностью разработки программного интерфейса является способ взаимодействия приложений, написанных с помощью Electron, с возможностями платформы, на которой запускается такое приложение.

В Electron доступен ряд API-интерфейсов, поддерживающих разработку настольных приложений в обоих процессах - main process и render process. Для доступа к этим процессам API Electron вам необходимо включить в проект модуль «electron».

Все электронные API-интерфейсы назначаются типам процессов. Многие из них могут использоваться только из основного процесса, некоторые из них только из процесса рендеринга. В документации для каждого отдельного API указывается, из какого процесса он может быть использован.

Поскольку возможна связь между процессами, процесс визуализации может вызвать основной процесс для выполнения задач. Electron поставляется с модулем, называемым remote, который предоставляет API, обычно доступные только в основном процессе. Чтобы создать BrowserWindow из процесса рендеринга, необходимо использовать remote модуль как промежуточный модуль.

Все API, доступные в Node.js, доступны и в Electron.