МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П.О. СУХОГО»

Факультет автоматизированных и информационных систем

Кафедра «Информатика»

Специальность 1-40 04 01 «Информатика и технологии программирования»

Отчет по преддипломной практике

на тему: «Программный комплекс для автоматизации   
менеджмента фитнес-центра»

Подготовил: студент гр. ИП-41

Земчёнок В. Г.

Руководитель от предприятия:

руководитель подразделения

Гуд С. Н.

Руководитель:

старший преподаватель

Романькова Т. Л.

Дата проверки:

Дата допуска к защите:

Дата защиты:

Оценка работы:

Подписи членов комиссии: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Гомель 2021

СОДЕРЖАНИЕ

[Введение 3](#_Toc69773318)

[1 Анализ предметной области 4](#_Toc69773319)

[1.1 Анализ предметной области 4](#_Toc69773320)

[1.2 Аналитический обзор существующих аналогов 4](#_Toc69773321)

[1.3 Обзор технологий для реализации программного обеспечения 8](#_Toc69773322)

[1.4 Постановка задачи на дипломное проектирование 14](#_Toc69773323)

[2 Архитектура программного обеспечения 16](#_Toc69773324)

[2.1 Функциональная модель 16](#_Toc69773325)

[2.2 Сущности и связи между ними в проектируемом программном комплексе 18](#_Toc69773326)

[2.3 Требования к архитектуре приложения 25](#_Toc69773327)

[Заключение 27](#_Toc69773329)

[Список использованных источников 28](#_Toc69773330)

[Приложение А. Схема базы данных 16](#_Toc69773324)29

# 

# ВВЕДЕНИЕ

На сегодняшний день применение современных информационных технологий для автоматизации различных процессов находит всё больший и больший спрос. Приложения на разного рода устройствах, портативных или стационарных, значительно ускоряют и упрощают решение повседневных задач. Не обошёл процесс автоматизации и сферу спортивных и оздоровительных услуг, ведь спрос на них продолжает активно расти ­­­­­­– формирование здорового образа жизни и личностной физической культуры помогает улучшить здоровье и повысить продуктивность в повседневных активностях и на работе.

На первый взгляд, фитнес-центр – это пара тренажёров в некотором помещении, где также есть высококвалифицированный тренерский персонал, но на самом деле это большой механизм, внутренние составляющие которого работают, постоянно находясь в зависимости друг от друга. Среди них, помимо непосредственно отдела фитнеса, можно также выделить отделы дополнительных услуг, рецепции, продаж, финансовый отдел, подразделение по управлению хозяйственно-технической базой организации.

Рост спроса на услуги фитнес-центра означает увеличение клиентской базы, и, как следствие, объёма информации, обрабатываемого одним из сотрудников любого из вышеупомянутых отделов. Вместе с тем, растёт и сложность работы, что сказывается на качестве обслуживания и управления центром.

Разрабатываемый программный продукт предназначен для автоматизации основных бизнес-процессов, оптимизации процессов обработки информации, чтобы повысить качество обслуживания и обеспечить предоставление информации для принятия управленческих решений.

# 1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## 1.1 Анализ предметной области

Объектами автоматизации являются процессы, происходящие внутри фитнес-центра как организации. При исследовании предметной области, а также согласно [1][2], можно выделить следующие группы бизнес-процессов:

* учёт клиентов;
* управление расписанием и посещениями;
* учёт финансов;
* отчётность;
* маркетинг.

В учёт клиентов входит хранение и обработка информации об активных услугах клиентов, управление их картами, пакетами услуг.

При управлении расписанием и посещением занятий в первую очередь необходимо обрабатывать информацию о посещаемости занятий, иметь возможность записи клиента на занятие, обрабатывать информацию о занятости спортивных залов и площадок для планирования занятий по времени, а также подводить статистику посещений.

В финансовый учёт входят управление лицевыми счетами клиентов, расчёт заработной платы тренеров и прочее.

Отчёты, в свою очередь, тоже могут быть как по клиентам, так и по тренерам, анализ аналитики занятий и т.п.

Также важно отметить, что маркетинг играет не менее важную составляющую, например, помогает обеспечить сегментирование клиентов по категориям и прочее.

Основные роли пользователей: клиент, тренер, бухгалтер, менеджер отдела продаж, руководитель фитнес-центра.

## 1.2 Аналитический обзор существующих аналогов

На данный момент уже существуют некоторые организации, чьи программные продукты удовлетворяют большинство требований многих фитнес-центров. Например, система автоматизации *Mobifitness*.

Согласно [3], благодаря Mobifitness, спортивная организация вроде фитнес-центра может позволить себе собственную учетную систему, мобильное приложение, качественное расписание занятий и другие продукты, которые оптимизируют рабочий процесс сотрудников, повысят лояльность клиентов и продажи.

Данный программный продукт является коммерческим решением с несколькими видами тарифов, открывающих различный функционал программного комплекса. Данный программный продукт полностью соответствует требованиям, устанавливаемым разработчику программного продукта в данной предметной области. Пользователям доступен широкий выбор инструментов, среди которых стоит отметить удобный поиск занятий и их сортировку по различным критериям, например, по ФИО инструктора, по направлению занятия или конкретному его виду.

Внешний вид страницы с расписанием занятий в веб-приложении приведен на рисунке 1.1.

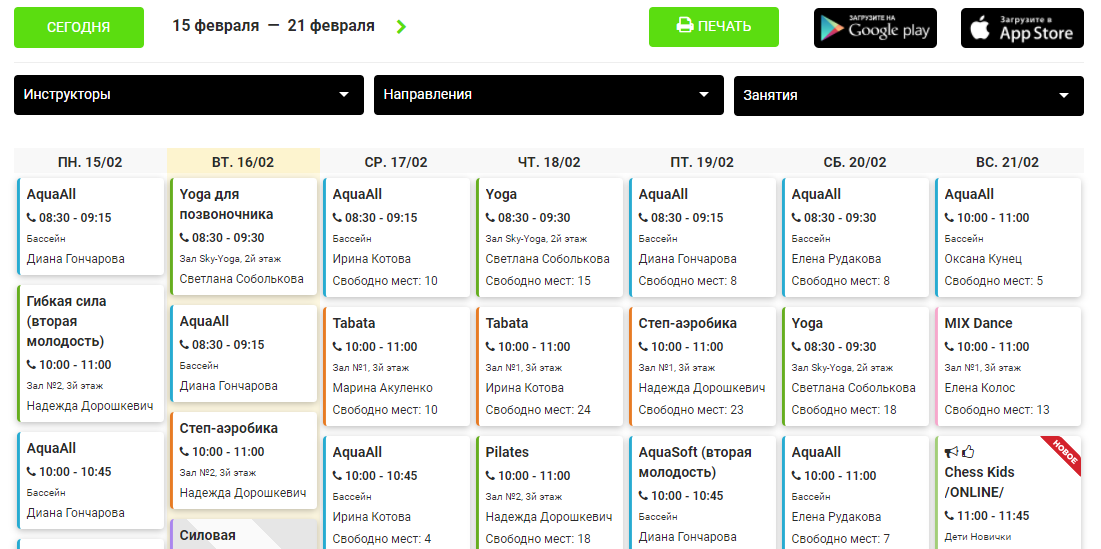


Рисунок 1.1 ­– Вид расписания занятий в веб-приложении *Mobifit*

Основной функционал – сама система автоматизации и учёта. Среди функционала «за дополнительную плату» можно отметить наличие личного кабинет тренера, расчёт их заработной платы, отсутствие отчёности или даже пользование фирменным приложением для мобильных устройств. Внешний вид такого приложения приведён на рисунке 1.2

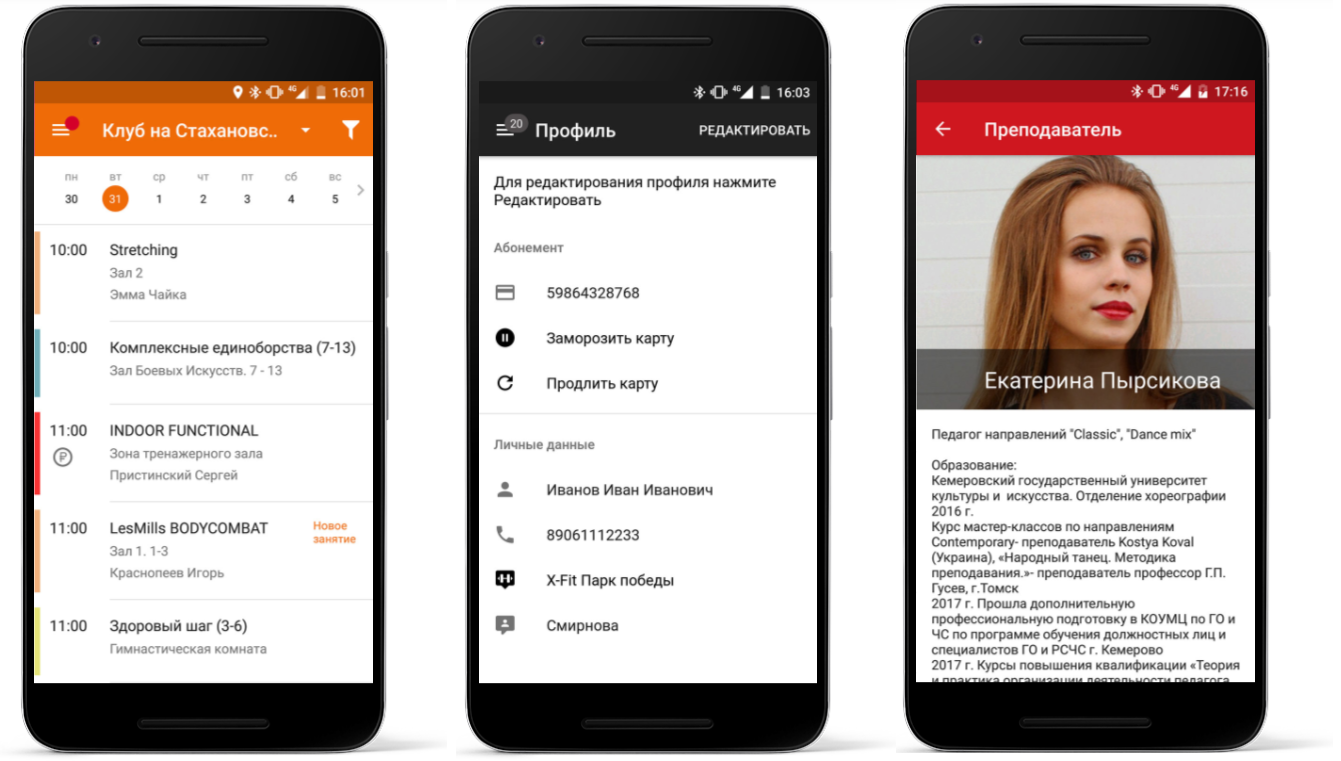
****

Рисунок 1.2 ­– Внешний вид приложения *Mobifit* на мобильном устройстве

Пользовательский интерфейс интуитивно понятен, клиенту не составит труда найти подробную информацию о месте проведения, тренере или самом занятии (вид этого фрагмента страницы приведён на рисунке 1.3.

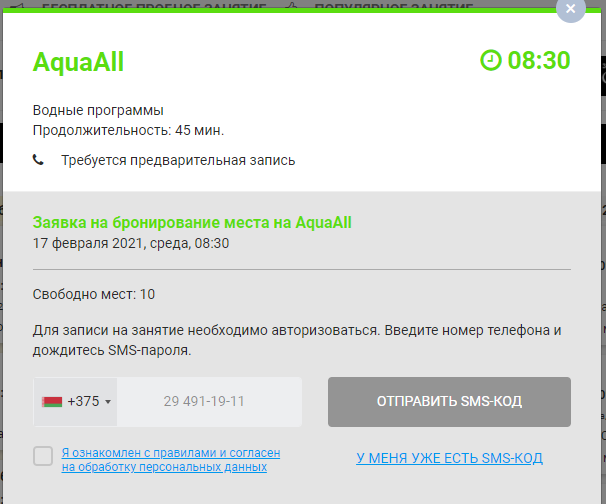


Рисунок 1.3 – Подробная информация о занятии

Помимо вышеупомянутого, программный комплекс не имеет форму обратной связи с возможностью просмотреть ответ администрации прямо в приложении.

Ещё один аналог системы менеджмента финтес-центра – программный комплекс проекта *Sport Priority.* Для фитнес-клуба данная система, согласно [4], представляет собой полнофункциональную коммуникационную площадку и систему управления клубом одновременно, для клиентов спортивного учреждения – мобильное приложение для покупки и планирования своих тренировок. Это полноценный инструмент организации своего фитнес-досуга, когда человек видит в одном месте картину своего спортивного развития целиком, при этом приложение выступает своеобразным помощником, агрегирующим информацию по услугам различных фитнес-центров и других спортивных учреждений.

В основной функционал входит и сама автоматизированная система, и мобильное приложение. Помимо того, сервис создаёт сайт со всей агрегированной информацией о спортивном учреждении и его услугах.

Пользовательский интерфейс приятный и простой для восприятия клиентом. Пример внешнего вида страницы с расписанием занятий в веб-приложении приведён на рисунке 1.4.

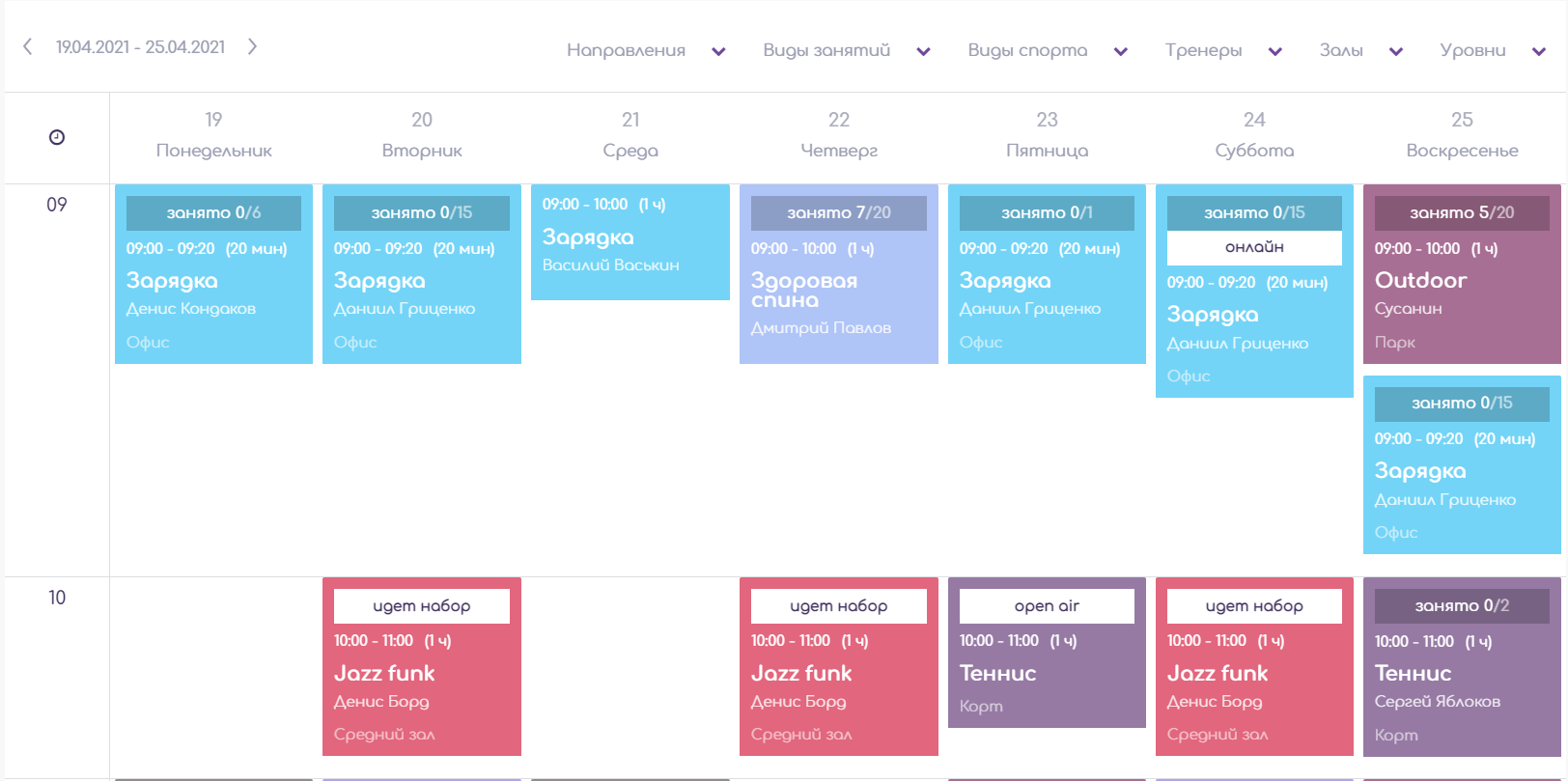
****

Рисунок 1.4 ­– Внешний вид веб-приложения

Стоит также отметить, что оно же является *посредником* между центром и его клиентом, т.е. отсутствует прямое взаимодействие, например, абонементы приобретаются в этой системе, но при этом сама система не является частью фитнес-центра, а является обособленным проектом, который собирает информацию с нескольких фитнес-центров и взаимодействует с их административно-управленческими персоналами. Это же, согласно словам разработчиков, уменьшает вероятность удаления приложения, если клиент покинул один из ранее посещаемых центров – отдельное приложение, скорее всего, будет удалено, а подобный «агрегатор» информации, где есть все спортивные клубы, будет оставлено.

В качестве третьего существующего аналога можно привести облачный сервис для управления фитнес-клубом FitBase. Согласно [5], данная система поддерживает широкий спектр полезных функций. Среди них стоит отметить наличие базы клиентов со всей историей коммуникаций и покупок, понятными статусами и возможностью учета посещений по абонементам и услугам; выполнение действий над группами (рассылки, продления и т.п.) с применением множества фильтров. Сервис также помогает менеджерам настраивать некоторый список автоматических действий, который позволяет упростить работу менеджера до пары кликов, а то и вовсе без его участия: поздравления с днём рождения, или отсылка напоминания клиенту, который давно не был в клубе. Одной из сильных сторон сервиса является реализация финансового учета: доходы фиксируются автоматически при продаже абонементов, услуг и товаров, а расходы можно внести вручную, при этом система сама высчитает необходимые показатели. Система позволяет создавать свои финансовые планы и следить за их выполнением. Пример страницы её веб-приложения приведён на рисунке 1.5.

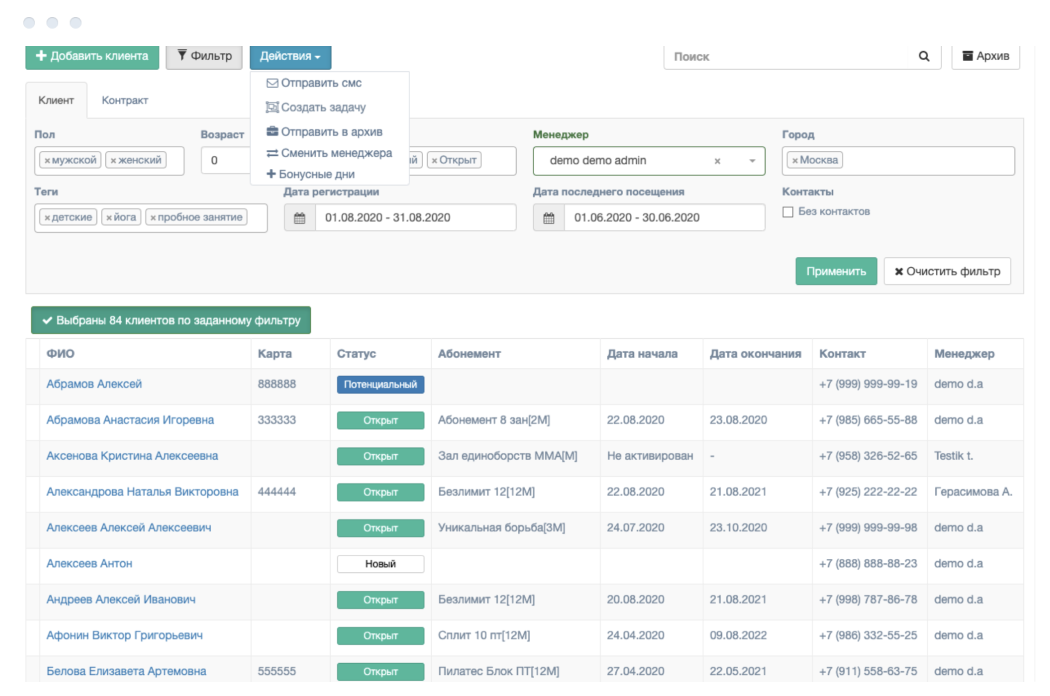


Рисунок 1.5 ­– Внешний вид окна администратора FitBase

Вместе с тем, у данной системы личный кабинет клиента отсутствует, находясь в списке функционала, который появится в будущем, т.е. является одной из перспектив развития системы, что является существенным недостатком.

## 1.3 Обзор технологий для реализации программного обеспечения

***1.3.1*** Программный комплекс будет состоять из нескольких клиентских компонентов (приложений), серверного компонента и базы данных, данными которой он манипулирует.

В данном разделе будет рассмотрен стек технологий, который представляет собой набор платформ и инструментов, используемых для разработки программного продукта. Выбор конкретного набора платформ с определёнными преимуществами при разработке способствует созданию хорошо функционирующего программного обеспечения.

Подходов к реализации клиент-серверного комплекса приложений существует достаточно много. Это и приложения, реализованные с помощью языка программирования Java, и комплексные решения, комбинирующие в себе и разработку клиентского (фронтенд) компонента, и разработку серверного (бэкенд).

Совместное использование данных между двумя или более приложениями всегда было фундаментальным требованием при разработке программного обеспечения. Одной из важных задач при проектировании архитектуры программного комплекса является корректное взаимодействие программ, в том числе разработанных на разных языках программирования и работающих на разных операционных системах.

Между двумя функциональными системами есть некоторая граница, через которую и происходит их взаимодействие – интерфейс, который обеспечивает выполнение одной системой набора операций, часто для получения некоторых данных в качестве результата, при этом нет необходимости в предоставлении информации о внутренней структуре системы и описании способа выполнения ею поставленной задачи. Такой интерфейс называется программным интерфейсом приложения (англ. *Application Programming Interface*), или *API*, он определяет правила взаимодействия с системой извне.

Для организации обмена информацией между компонентами комплекса, которые могут работать на разных устройствах, необходимо, чтобы их программное обеспечение могло запрашивать функции друг друга через сеть.

При автоматизации процессов сферы спортивных и оздоровительных услуг удобно пользоваться как мобильным приложением, так и веб-приложением для компьютера. В данном случае, эти приложения являются клиентами, которые взаимодействуют с *API* некоторого сервера, который и занимается обработкой информации. При этом отсутствует жёсткая привязка к тому, какого рода клиент или сервер может быть, главное – это соответствие клиента правилам, которые определяет сервер для взаимодействия.

*API* для веб-сервера, веб-браузера или любой другой веб-ориентированной технологии называется *Web API*. Согласно [6], данная технология позволяет посылать данные в качестве ответа на запросы различного рода клиентов, будь то поступаемые из сети, мобильных приложений, веб-приложений или других *Web API*.

Пусть одним из компонентов комплекса является клиентское *ASP.NET* веб-приложение, обеспечивающие необходимым функционалом административно-управленческий персонал и клиентов организации. Стоит учесть, что клиентам также важно, чтобы у них была возможность взаимодействовать с тренером в любой момент, поэтому актуально, чтобы в программном комплексе присутствовало приложение для мобильного устройства, например, Android приложение. При автоматизации работы любой организации важно определить, с каким источником данных будет лучше всего работать в рамках рассматриваемой предметной области. Для хранения данных существует некоторая база данных и *Web* *API,* который управляет ею, а клиентские приложения взаимодействуют с ним посредством механизма запросов и ответов. Применение подхода увеличит производительность за счёт того, что в качестве ответа будут возвращться исключительно данные, указанные в запросе. Таким образом быстрее обрабатываются запросы и выдаются ответы. Помимо того, мы можем разделить компонент на подмножество компонентов, что также может положительно сказаться на производительности, так как это может помочь распределить нагрузку внутри конкретного компонента. Исходя из вышеперечисленного, схема комплекса с серверным *Web API* приложением, приведена на рисунке 1.6.



Рисунок 1.6 – Архитектура приложения с *Web API*

*Web API* предоставляет несколько вариантов формата данных, отправляемых в ответ на запрос, например, *JSON* или *XML*, для работы с которыми существуют специальные библиотеки в языках программирования *C#* и *Java* – следовательно, разрабатываемые клиентские приложения (*ASP.NET* веб-приложение на языке программирования *C*# и *Android* приложение на *Java*) смогут корректно взаимодействовать с *API*. При необходимости расширения комплекса, например, создания другого *Web API* иликлиентских приложений для других платформ и операционных систем (например *iOS* на *IPhone*), работа над форматами выходных данных не вызовет проблем у разработчика. Пример страницы тестирования *Web API* с помощью *Swagger* приведен на рисунке 1.7.

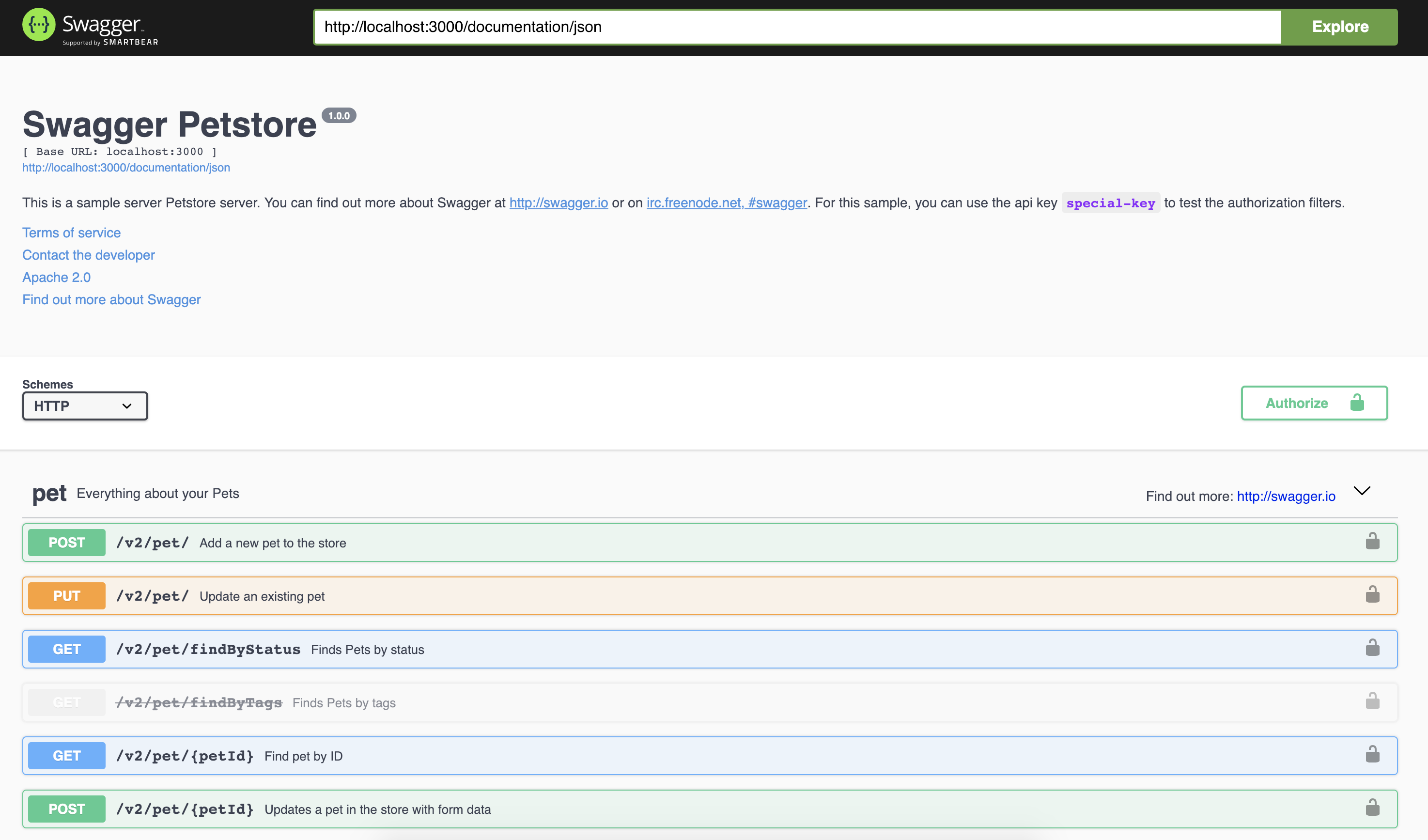


Рисунок 1.7 – Пример интерфейса некоторого *Web API*

Технология *Microsoft Azure App Service* представляет собой виртуальный сервер, облачную вычислительную платформу, преимуществами которой являтся простой процесс администрирования и, если есть необходимость, масштабирования вычислительных мощностей. В разрабатываемом комплексе на ней будет развернут *API* компонент. Это позволит свободно получить к нему доступ через сеть Интернет с любого клиентского приложения, будь то веб-приложение или приложение для мобильного устройства (*Android*, *iOS*).

***1.3.2*** Необходимо помнить, что клиентам фитнес-центра также важно получать доступ к информации по своим спортивным услугам, не имея под рукой компьютер, а также иметь возможность взаимодействовать с тренером в любой момент, поэтому актуально, чтобы в программном комплексе присутствовало приложение для мобильного устройства, например, *Android* приложение.

Разрабатывать приложения для ОС *Android,* согласно [7], можно в специально предназначенной для этого интерактивной среде разработки *Android Studio* от разработчиков этой операционной системы – компании *Google.* Языки программирования, поддерживаемые при этом – *Java* и *Kotlin.*

Помимо этого, в рамках платформы .*NET* существует технология разработки универсальных приложений *Xamarin* – инструмент для создания приложений на языках семейства .NET (*C#*, *F#*, *Visual Basic*), который позволяет создавать единый код, работающий на *Android*, *iOS* и *Windows* (*UWP*-приложения).

Для обеих технологий используется *xaml*-ориентированный подход к конструированию интерфейса, то есть он описывается декларативно в формате xml, вы сразу видите, как элементы расположены на форме и какие свойства имеют (рисунок 1.8)

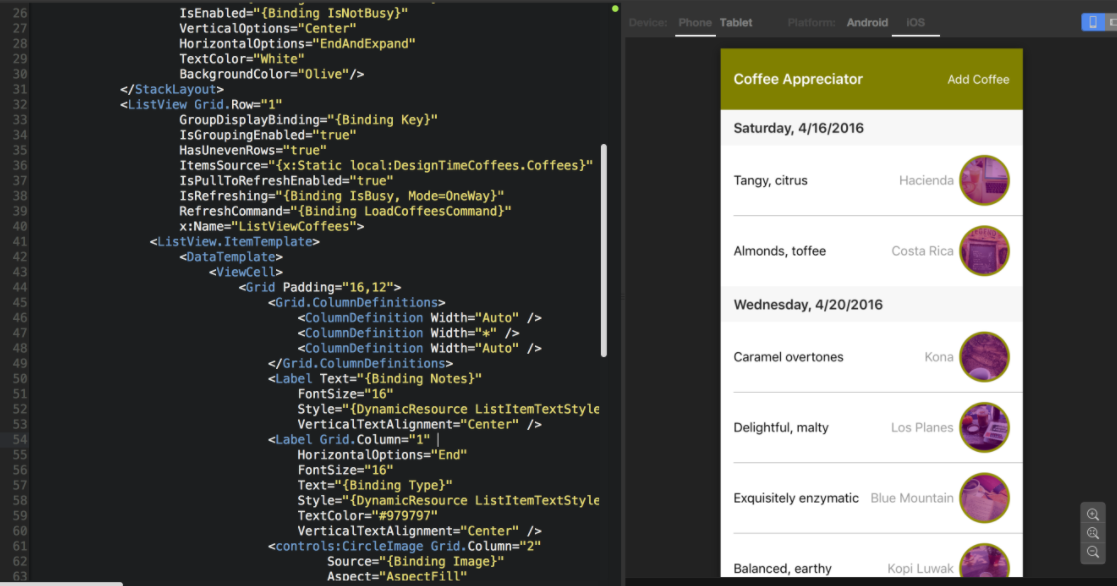


Рисунок 1.8 – Xaml-разметка и соответствующий ей вид окна приложения

***1.3.3\_***Стоит предусмотреть факт того, что административно-управленческий персонал может работать на дому в экстренных случаях, например, при ухудшении эпидемиологической обстановки. В таком случае, закрепление приложения на конкретной рабочей машине может привести к тому, что у сотрудников фитнес-центра не будет доступа к управленческому функционалу комплекса, например, со своего персонального компьютера. Здесь возникает необходимость создать клиентское веб-приложение, например используя технологии *ASP.NET Core*, которое будет доступно через сеть Интернет. Это же позволит использовать приложение на любой операционной системе.

При разработке веб-приложения с использованием технологии *ASP.NET* зачастую используют паттерн *MVC*. Использование данного архитектурного подхода, а именно разбиение приложения на модели (*model*), представления (*view*) и контроллеры (*controller*) позволяет чётко разделить ответственность между компонентами и сделать их менее зависимыми друг от друга, чтобы процесс замены компонента на другую его реализацию стал менее трудоёмким и не было необходимости вносить существенные изменения в уже рабочее приложение. Стоит также отметить, что *Web API* приложение тоже частично следует паттерну *MVC,* только результатом выполнения *action*-методов контроллера является некоторый ответ, посылаемый клиенту, который может иметь формат как xml-схемы (или веб-страница), так и другие форматы, например JSON. Схема работы приложения с моделью *MVC,* а такжеего взаимодействия с *Web API* приведена на рисунке 1.9.

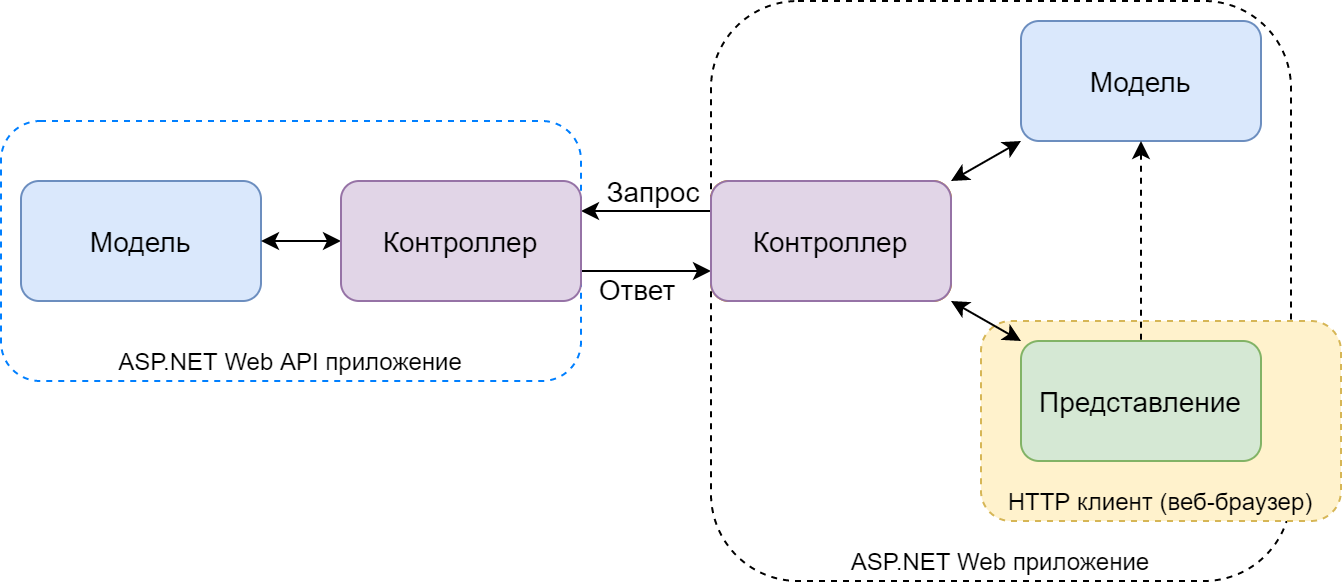


Рисунок 1.9 – Схема работы приложения с моделью MVC

При необходимости можно использовать контейнерные программные решения, например, Docker, для полной или частичной автоматизации процесса развертывания как *Web API,* так и *Web* приложений на удалённых машинах.

***1.3.4*** Для хранения и обработки данных наибольшей популярностью пользуются реляционная и нереляционная модели данных. База данных реляционного типа (*SQL*) поддерживает хранение данных в формате строго структурированных таблиц, связанных друг с другом. В таблице есть строки и столбцы, каждая строка представляет отдельную запись, а столбец — поле определённого типа данных. В каждой ячейке информация записана «по шаблону», т.е. в таблицу можно записать только те данные, структура полей которых является подмножеством описанных полей таблицы. Основное достоинство такой модели ­– надежность и неизменяемость данных, низкий риск потери информации. Конечный результат выполнения запроса здесь предсказуем. При обновлении данных их целостность гарантирована, они заменяются в одной таблице.

Согласно [8], нереляционная база данных (*NoSQL*) обеспечивает хранение данных без четко заданных связей друг с другом и четкой структуры. Хранимые данные здесь являются не структурированными связанными таблицами, а зачастую могут представлять собой некоторый набор разнородных документов, в том числе мультимедийные данные ­– изображения, видео и т.д. В отличие от реляционных баз данных, *NoSQL* базы данных не поддерживают запросы *SQL*. Примеры запросов к базам данных с этими моделями данных, приведены на рисунке 1.10

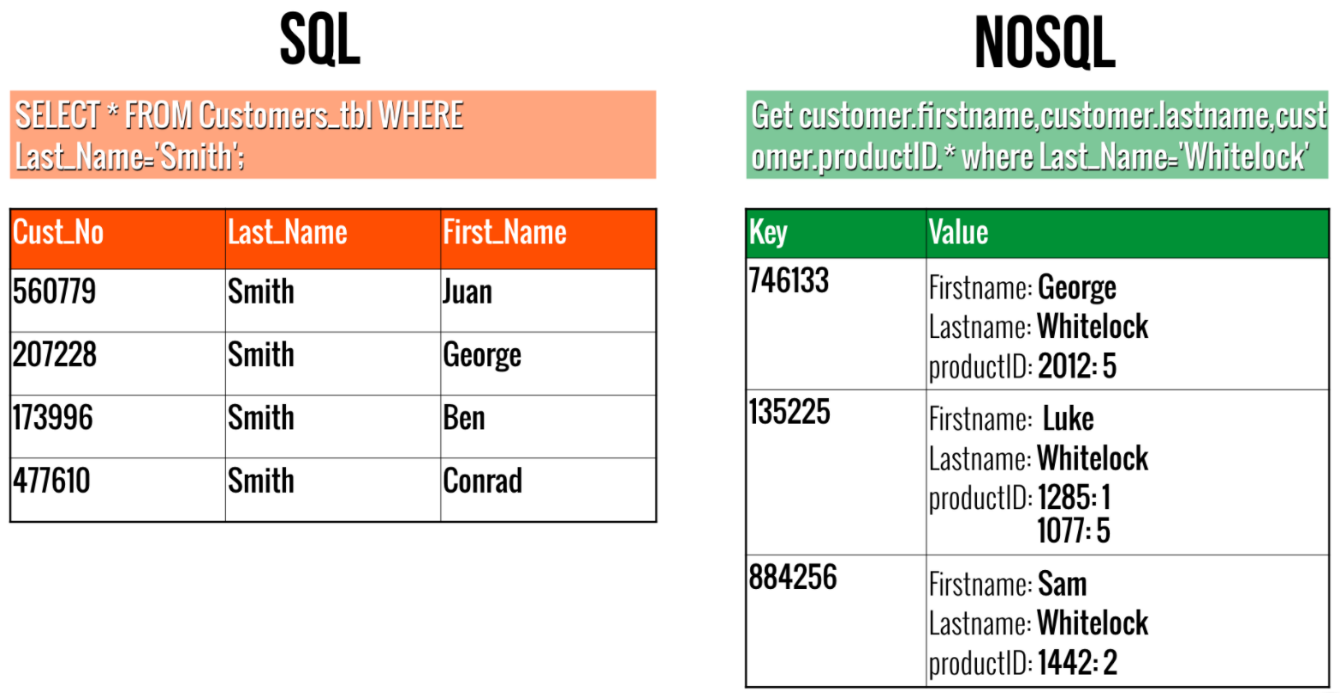


Рисунок 1.10 – Запросы к SQL и NoSQL базам данных

Для развертывания реляционной базы данных и управления ею можно использовать сервисы *Microsoft Azure SQL Database* или *Amazon Relational   
Database Service.* Оба сервиса являются облачными базами данных, запускаемых на некоторой облачной вычислительной платформе. Они обечпечивают обеспечивает экономичное и масштабируемое использование ресурсов при одновременной автоматизации трудоемких задач администрирования, таких как выделение аппаратного обеспечения, настройка базы данных, установка исправлений и резервное копирование. Единственное различие в том, что *Amazon RDS –* это контейнерный сервис, соответственно, программный компонент в данном случае будет помещён в виртуальный контейнер. *Azure* *SQL Database* ­являетсясервисом управляемой базы данных, т.е. здесь отсутствует дополнительная контейнерная оболочка.

## 1.4 Постановка задачи на дипломное проектирование

В проектируемом программном обеспечении необходимо реализовать комплекс приложений для автоматизации менеджмента фитнес-центром с различных сторон данного учреждения как организации. Это означает, что программное обеспечение должно упростить работу как административно-управленческого персонала, так и персонала, непосредственно оказывающего спортивные услуги, а также обеспечить клиентов удобным инструментом для взаимодействия с фитнес-центром и услугами, которые он предоставляет. Данный комплекс должен упростить процессы обработки информации, чтобы повысить качество обслуживания и обеспечить предоставление информации для принятия управленческих решений.

В процессе разработки провести построение функциональной модели комплекса, а также логическое и физическое проектирование базы данных.

Так как программное обеспечение представляет комплекс клиентских и серверных приложений, следует распределить логику программного обеспечения между этими компонентами, причём клиентские приложения должны полностью обеспечивать функционал клиента вне зависимости от платформы, на которой они реализованы.

Клиентская часть управляется непосредственно клиентами, поэтому важно, чтобы она имела пользовательский интерфейс, который будет одновременно и многофункциональный, и интуитивно понятный, простой. Необходимо реализовать корректное взаимодействие между клиентами и серверным компонентом комплекса, представляющее собой формирование запросов и получение ответов от сервера.

Серверный компонент должен представлять собой некоторый *API*, расположенный и развёрнутый на отдельной независимой машине со своими вычислительными ресурсами.

Функционал программного комплекса должен обеспечивать:

* возможность ведения справочной информации, а именно добавления, изменения или удаления справочников и управление их содержанием, с возможностью сохранения в некоторую базу данных;
* возможность отправки фотографий, например, для установки в качестве фотографии учётной записи пользователя или фотографии спортивного комплекса, где расположен фитнес-центр;
* возможность сортировки и фильтрации данных, например, занятий по тренеру или виду занятий в расписании центра;
* поиск данных;
* формирование отчетов в виде таблицы прямо на веб-странице приложения с возможностью экспорта в файл формата *MS Excel*;
* обеспечить управление статусом занятий, в том числе их отмену и перенос, своевременное информирование клиентов о вносимых изменениях.

Система должна предоставлять возможность локализации своих компонентов.

Компоненты программного комплекса должны быть сопровождены модульными (юнит) тестами.

Необходимо также провести тестирование программного комплекса как бизнес-приложения, соответствующее реальным условиям эксплуатации на предприятии, оно же тестирование *AQA* (*Automated Quality Assurance*).

# 2 АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

## 2.1 Функциональное моделирование

Под функциональным моделированием понимается процесс построения функциональных моделей объекта автоматизации, либо отдельных его процессов. Функциональная модель представляет собой ориентированный граф, вершинами которого являются выполняемые функции, а дугами либо элементы подсистемы, либо потоки информации, вещества или энергии.

Согласно [9], IDEF0 – Function Modeling – методология функционального моделирования, предназначенная для формализации и описания бизнес-процессов. Отличительной особенностью IDEF0 является её акцент на соподчинённость объектов. В IDEF0 рассматривается логические отношения между работами, а не их временная последовательность

* функция на диаграмме представлена блоком, имеющим 3 входа (снизу, слева, сверху) и один выход (справа);
* потоки информации об управлении или ограничениях входят в блок сверху;
* информация, которая подвергается обработке, показана с левой стороны блока;
* результаты выхода показаны с правой стороны;
* механизм (человек или автоматизированная система), который осуществляет операцию, представляется дугой, входящей в блок снизу [9].

На рисунке 2.1 изображена диаграмма IDEF0 декомпозиции первого уровня функциональной модели программного комплекса для автоматизации менеджмента фитнес-центра.

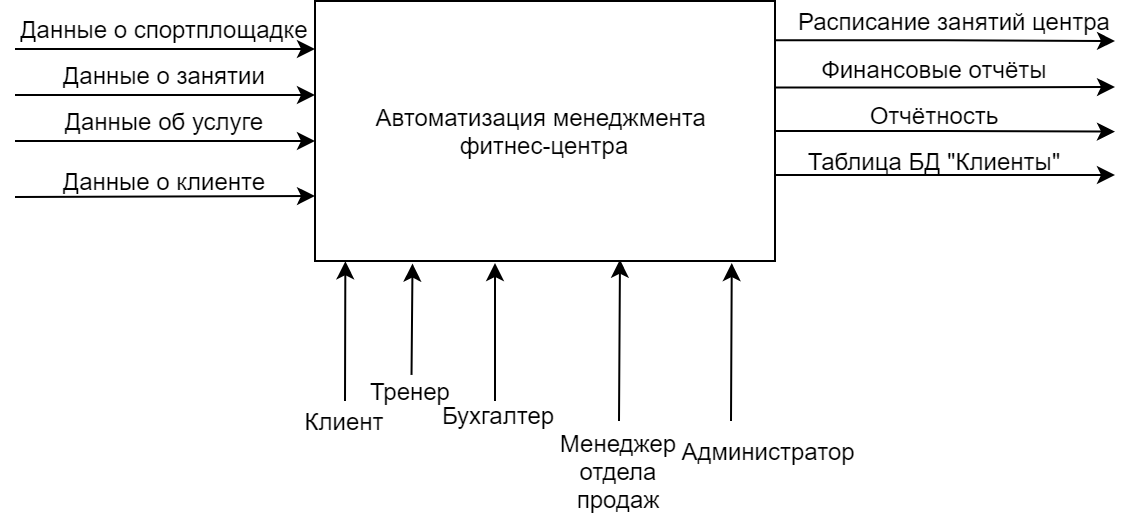


Рисунок 2.1 ­– Диаграмма IDEF0 декомпозиции первого уровня

Диаграммы декомпозиции предназначены для детализации функций и получаются при разбиении контекстной диаграммы на крупные подсистемы (функциональная декомпозиция) и описывают каждую подсистему и их взаимодействие.

Диаграмма IDEF0 декомпозиции второго уровня приведена на рисунке 2.2.

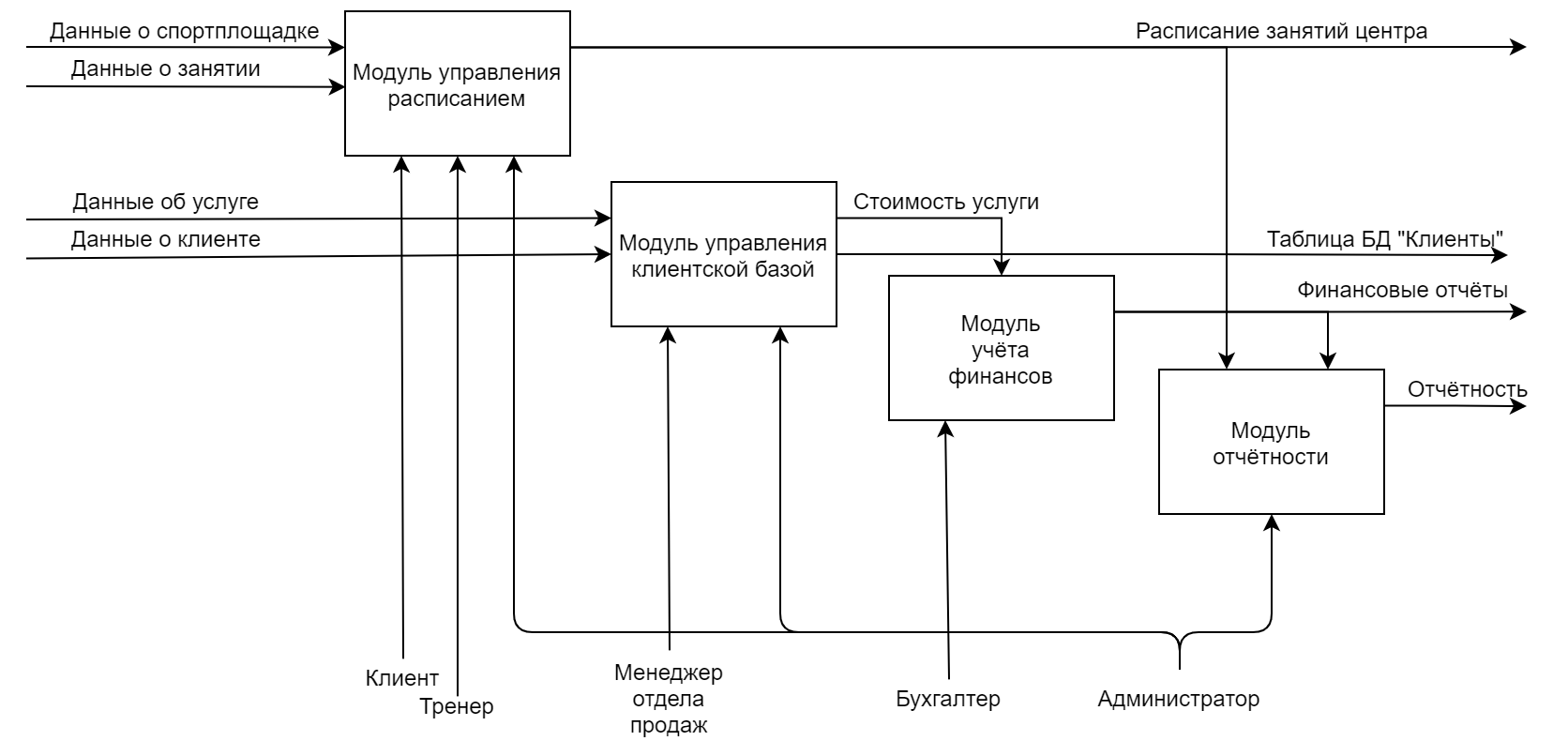


Рисунок 2.2 – Диаграмма IDEF0 второго уровня декомпозиции

При функциональном моделировании необходимо также разработать диаграмму прецедентов на основе результатов исследования предметной области. Диаграмма прецедентов приведена на рисунке 2.3

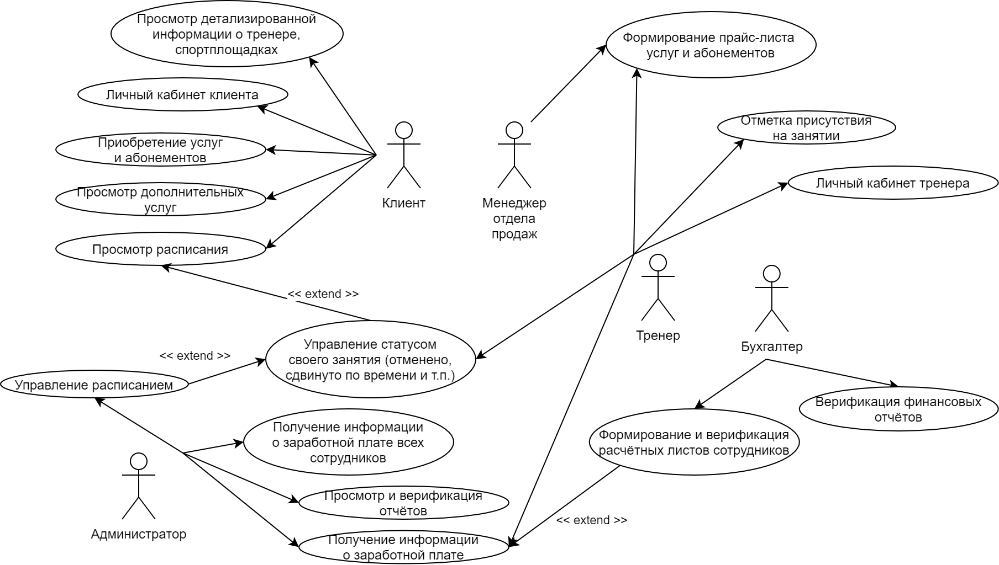


Рисунок 2.3 – Диаграмма прецедентов

## 2.2 Разработка информационной модели

***2.2.1*** На основе анализа предметной области можно выделить следующие сущности:

* пользователь/клиент (*User*);
* тренер (*Trainer*);
* специализация (*Specialization*);
* группа клиентов (*Group*);
* услуги, оказываемые специалистом в рамках программы абонемента (*Service*);
* карта абонемента (*AbonementCard*);
* абонемент, закреплённый за группой клиентов (*Abonement*);
* место оказания спортивных услуг (*Venue*);
* занятие как услуга, оказываемая тренером, закреплённая за местом и ограниченная по времени (*Event*);
* занятие как встреча, фактически ­– мероприятие, закреплённое за занятием-услугой, проводимое для группы клиентов в назначенную   
  дату (*Meeting*).

Сущность «Пользователь» должна хранить информацию о пользователе приложения (клиенте фитнес-центра). В ней должны присутствовать следующие характеристики:

* фамилия;
* имя;
* отчество;
* пол;
* дата рождения;
* адрес проживания;
* телефон;
* баланс.

Сущность «Тренер» должна хранить информацию о тренере фитнес-центра. В ней должны в качестве характеристик должны присутствовать идентификатор пользователя и специализация.

Сущность «Специализация» представляет собой описание сферы деятельности тренера.

Сущность «Специализация» представляет собой описание сферы деятельности тренера.

Сущность «Группа клиентов» представляет собой объединение клиентов в рамках одного занятия.

Сущность «Услуга» представляет собой некоторую услугу, предоставляемую фитнес-центром клиенту в рамках программы абонемента. В ней должны присутствовать следующие характеристики:

* наименование услуги;
* цена;
* идентификатор специализации.

Сущность «Карта абонемента» представляет собой свидетельство о том, что клиент может воспользоваться услугами абонемента определённое число раз в течение некоторого времени. В ней должны присутствовать следующие характеристики:

* идентификатор пользователя;
* идентификатор абонемента;
* дата начала действия карты;
* число посещённых занятий;
* статус карты.

Сущность «Абонемент» представляет собой непосредственно абонемент. Среди его характеристик стоит упомянуть:

* наименование;
* срок действия (в днях);
* число посещений;
* цена;
* число дней приостановки;
* идентификатор группы.

Сущность «Место оказания спортивных услуг» полностью соответствует своему названию. В качестве характеристик имеет наименование и адрес.

Сущность «Занятие как услуга» имеет следующие характеристики:

* длительность занятия в минутах;
* идентификатор услуги;
* идентификатор тренера;
* идентификатор места оказания услуги.

Сущность «Занятие как мероприятие» характеризуется наличием:

* даты проведения занятия;
* идентификатора занятия как услуги;
* идентификатора группы.

***2.2.2*** Связь между сущностями «Пользователь» и «Тренер» – один к одному. Это означает, что тренер может иметь только один идентификатор пользователя, а пользователь, если является тренером, то только одним.

Связь между сущностями «Пользователь» и «Карта абонемента» – один ко многим. Это означает, что карта может принадлежать только одному пользователю, при этом пользователь может иметь несколько карт абонементов.

Связь между сущностями «Абонемент» и «Карта абонемента» – один ко многим. Это означает, что карта может относиться только к одному абонементу, при этом абонемент доступен всем клиентам и может иметь несколько карт у разных клиентов.

Связь между сущностями «Специализация» и «Тренер» – один ко многим. Это означает, что тренеру ставится в соответствие только одна специализация, при этом могут существовать несколько тренеров одной специализации.

Связь между сущностями «Специализация» и «Тренер» – один ко многим. Это означает, что тренеру ставится в соответствие только одна специализация, при этом могут существовать несколько тренеров одной специализации.

Связь между сущностями «Специализация» и «Услуга» – один ко многим. Это означает, что у одной специализации имеется множество услуг, каждая из которых может быть оказана тренерами только одной специализации.

Связь между сущностями «Услуга» и «Занятие как услуга» – один к одному. Это означает, что между ними ставится прямое соответствие.

Связь между сущностями «Место оказания услуги» и «Занятие как услуга» – один к одному, что устанавливает прямое соответствие между сущностями.

Связь между сущностями «Тренер» и «Занятие как услуга» – один ко многим, что означает наличие у тренера множества услуг, но конкретное занятие проводится только одним тренером.

Связь между сущностями «Занятие как услуга» и «Занятие как мероприятие» – один ко многим, что означает, что занятие-услугу можно проводить несколько раз в разные дни, но каждому мероприятию соответствует только одно занятие-услуга.

Связь между сущностями «Группа» и «Занятие как мероприятие» – один ко многим, что означает, что группа может посетить несколько мероприятий, при этом за каждым мероприятием закреплена только одна определённая группа.

Связь между сущностями «Абонемент» и «Группа» – один ко многим, что означает, что сразу несколько групп может быть иметь один и тот же абонемент, но группа имеет не более одного абонемента.

Связь между сущностями «Абонемент» и «Услуга» – много ко многим, что означает, что услуга может находиться в списке сразу у несколих абонементов, и у каждого абонемента есть несколько услуг. Данную связь можно заменить, введя новую промежуточную таблицу «Услуги абонемента». Данная сущность будет иметь связь много к одному и с сущностью «Услуга», и с сущностью «Абонемент».

Логическая модель данных приведена на рисунке 2.4.

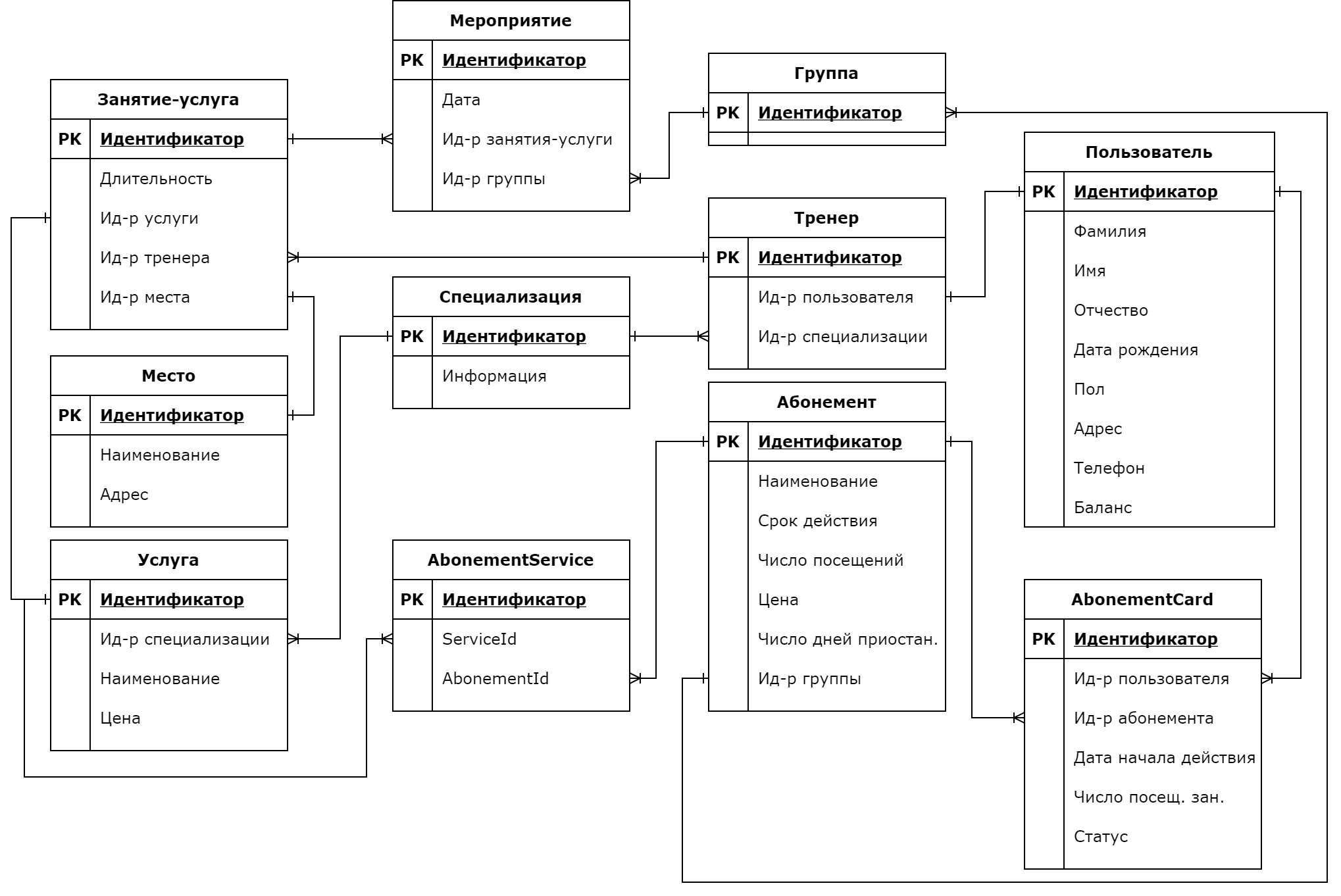


Рисунок 2.4 – Логическая модель данных

***2.2.3*** На основе результатов анализа предметной области, результатов концептуального и логического проектирований, была создана физическая модель базы данных со следующими таблицами:

* user;
* abonementCard;
* abonement;
* trainer;
* group;
* meeting;
* specialization;
* abonementService;
* venue;
* service;
* event.

Для реализации локализации необходимо иметь дополнительную сущность «Язык» с характеристиками «наименивание» и «код», которой соответствует таблица Language. Каждый пользователь имеет дополнительное поле-идентификатор языка.

Описание полей таблиц приведено в таблицах 2.1 – 2.12.

Таблица 2.1 – Поля таблицы User

|  |  |  |  |
| --- | --- | --- | --- |
| Ключевое  поле, тип ключа | Наименование поля | Тип хранимых данных | Может принимать значение NULL |
| Первичный | UserId | nvarchar(128) | нет |
| – | Surname | nvarchar(50) |
| Name | nvarchar(50) |
| LastName | nvarchar(50) | да |
| Telephone | nvarchar(20) |
| BirthDate | datetimeoffset | нет |
| Sex | int |
| Address | nvarchar(200) |
| Balance | decimal |
| Внешний | LanguageId | int |

Таблица 2.2 – Поля таблицы AbonementCard

|  |  |  |  |
| --- | --- | --- | --- |
| Ключевое  поле, тип ключа | Наименование поля | Тип хранимых данных | Может принимать значение NULL |
| Первичный | Id | int | нет |
| Внешний | UserId | nvarchar(128) |
| AbonementId | int |
| – | StartDate | datetimeoffset |
| VisitCount | int |
| Status | int |

Таблица 2.3 – Поля таблицы Abonement

|  |  |  |  |
| --- | --- | --- | --- |
| Ключевое  поле, тип ключа | Наименование поля | Тип хранимых данных | Может принимать значение NULL |
| Первичный | Id | string | нет |
| – | Name | nvarchar(50) |
| Days | int |
| Attendances | intInt |
| Cost | decimal |
| StopDays | int |
| Внешний | GroupId | int |

Таблица 2.4 – Поля таблицы Trainer

|  |  |  |  |
| --- | --- | --- | --- |
| Ключевое  поле, тип ключа | Наименование поля | Тип хранимых данных | Может принимать значение NULL |
| Первичный | Id | int | нет |
| Внешний | UserId | nvarchar(128) |
| SpecializationId | int |

Таблица 2.5 – Поля таблицы Group

|  |  |  |  |
| --- | --- | --- | --- |
| Ключевое  поле, тип ключа | Наименование поля | Тип хранимых данных | Может принимать значение NULL |
| Первичный | Id | int | нет |

Таблица 2.6 – Поля таблицы Meeting

|  |  |  |  |
| --- | --- | --- | --- |
| Ключевое  поле, тип ключа | Наименование поля | Тип хранимых данных | Может принимать значение NULL |
| Первичный | Id | int | нет |
| – | Date | datetimeoffset |
| Внешний | EventId | int |
| GroupId | int |

Таблица 2.7 – Поля таблицы Specialization

|  |  |  |  |
| --- | --- | --- | --- |
| Ключевое  поле, тип ключа | Наименование поля | Тип хранимых данных | Может принимать значение NULL |
| Первичный | Id | int | нет |
| – | Info | nvarchar(200) | да |

Таблица 2.8 – Поля таблицы AbonementService

|  |  |  |  |
| --- | --- | --- | --- |
| Ключевое  поле, тип ключа | Наименование поля | Тип хранимых данных | Может принимать значение NULL |
| Первичный | Id | int | нет |
| Внешний | ServiceId |
| AbonementId |

Таблица 2.9 – Поля таблицы Venue

|  |  |  |  |
| --- | --- | --- | --- |
| Ключевое  поле, тип ключа | Наименование поля | Тип хранимых данных | Может принимать значение NULL |
| Первичный | Id | int | нет |
| – | Name | nvarchar(200) |
| Address | nvarchar(500) |

Таблица 2.10 – Поля таблицы Event

|  |  |  |  |
| --- | --- | --- | --- |
| Ключевое  поле, тип ключа | Наименование поля | Тип хранимых данных | Может принимать значение NULL |
| Первичный | Id | int | нет |
| – | Minutes |
| Внешний | ServiceId |
| TrainerId |
| VenueId |

Таблица 2.11 – Поля таблицы Language

|  |  |  |  |
| --- | --- | --- | --- |
| Ключевое  поле, тип ключа | Наименование поля | Тип хранимых данных | Может принимать значение NULL |
| Первичный | Id | int | нет |
| – | Name | nvarchar(15) |
| Code | nvarchar(5) |

Таблица 2.12 – Поля таблицы Service

|  |  |  |  |
| --- | --- | --- | --- |
| Ключевое  поле, тип ключа | Наименование поля | Тип хранимых данных | Может принимать значение NULL |
| Первичный | Id | int | нет |
| – | Name | nvarchar(20) |
| Price | decimal |
| Внешний | SpecializationId | int |

Описание соответствий сущностей логической модели и таблиц физической модели приведено в таблице 2.12.

Таблица 2.12 – Соответствие сущностей и таблиц

|  |  |
| --- | --- |
| Сущность логической модели | Таблица физической модели |
| Пользователь | User |
| Карта абонемента | AbonementCard |
| Абонемент | Abonement |
| Тренер | Trainer |
| Группа клиентов | Group |
| Занятие как мероприятие | Meeting |
| Специализация | Specialization |
| Место оказания спорт. услуг | Venue |
| Занятие как услуга | Event |
| Услуга | Service |

Согласно связям сущностей логической модели, были построены связи между таблицами физической модели. Схема созданной базы данных приведена в приложении А.

**2.3 Требования к архитектуре приложения**

При организации программного комплекса, компоненты которого разнородны и используют для манипулирования базой данных один *Web API,* его архитектуру можно организовать так, чтобы он был *RESTful.*

В 2000 году был описан архитектурный подход *REST* (англ. *Representational State Transfer* или«передача состояния представления») – архитектурный подход к описанию веб-сервисов, основанных на гипермедиа-данных (информация различного рода, такая как аудио, видео, изображения, простой текст, гиперссылки и прочее), набор ограничений, правил и требований, которым должен следовать некоторый сервис для того, чтобы быть *RESTful*, т.е. сервисом, выполняющим все эти требования.

*REST* определяет следующий набор архитектурных ограничений:

* клиент-серверная архитектура;
* отсутствие состояния у сервера;
* кэшируемость;
* многослойность компонентов;
* единство интерфейса.

*RESTful* сервис должен соблюдать разделение обязанностей между клиентом и сервером, где клиент представляет собой некоторый функционал по отображению, управлению и фильтрации данных. Каждый запрос от клиента к серверу должен быть самодостаточным и не зависеть от других запросов, при этом нет необходимости в поддержании постоянного соединения между клиентом и сервером, в запрос должна быть встроена информация о клиенте, которую должен знать сервер – этим обеспечивается отсутствие состояния у сервера. Также должна быть возможность кэширования (сохранения данных) при выборке данных для последующего повторого использования без необходимости заново получать эти данные. Такой процесс должен быть доступен как на стороне сервера, так и на стороне клиента, и контролироваться одной из сторон. Приветствуется наличие посредников между клиентом и сервером – например, сервер журналирования, кэширования или др.

Сфера спортивных и оздоровительных услуг постоянно развивается, появляются новые виды спортивных активностей, здесь и возникает необходимость добавления в приложение нового функционала. *REST* позволяет быть сервисам расширяемыми, гибкими и предсказуемыми (за счёт соблюдения этих ограничений, которые не несут вреда, а только устанавливают рамки, следуя которым упрощается разработка и поддержка продукта). Стоит также отметить, что часто ошибочно отождествляются *REST* и *HTTP*, на самом деле *RESTful* могут быть и любые другие протоколы, которые следуют вышеперечисленным ограничениям.

Основываясь вышеперечисленном и на результатах анализа технологий для реализации программного комплекса, можно прийти к выводу, что при организации программного комплекса, компоненты которого разнородны и используют для манипулирования базой данных один *Web API,* его архитектуру можно организовать так, чтобы он был *RESTful*, что позволит повысить производительность, масштабируемость, обеспечит прозрачность всей системы взаимодействия, упростит дальнейшую разработку и поддержку продукта.

При анализе архитектуры комплекса можно выделить компоненты:

* база данных *MS SQL*, доступная на сервисах *Microsoft Azure;*
* клиентское веб-приложение, разработанное с использованием паттерна *MVC* с использованием технологии *ASP.NET Core*;
* клиентское *Android*-приложение;
* *RESTful WebAPI* приложение, разработанное с помощью технологии *ASP.NET Core WebAPI* платформы .*NET*.

Из дополнительных технологий, применяемых в комплексе, стоит упомянуть *Docker* для автоматизации развёртывания и управления *API* и *Microsoft Azure App Service*.ss

Схема разрабатываемого программного комплекса приведена на рисунке 2.5.

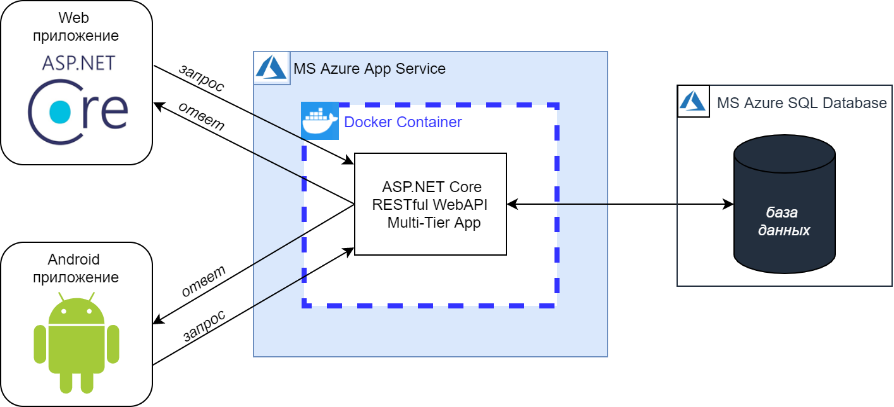


Рисунок 2.5 – Архитектура разрабатываемого программного комплекса

# ЗАКЛЮЧЕНИЕ

В ходе прохождения преддипломной практики была проанализирована предметная область разрабатываемого продукта, был произведён аналитический обзор существующих аналогов на рынке приложений. На основе результатов анализа предметной области были выделены группы основных бизнес-процессов и роли пользователей. Также был произведён обзор технологий для реализации программного обеспечения.

Было произведено функциональное моделирование, логическое и физическое проектирование базы данных, анализ информационной модели программного комплекса. Также в ходе выполнения преддипломной практики были изучены методы реализации функциональных возможностей программы.

Были подготовлены и написаны главы дипломного проекта:

* введение;
* анализ предметной области;
* архитектура программного комплекса.

В соответствии с календарно-тематическим планом были пройдены следующие этапы:

* прохождение курса по адаптации;
* обзор и анализ информации по разработке веб-приложений и приложений для мобильных устройства;
* разработка функциональной модели приложения;
* разработка структуры и состава информационного обеспечения программного комплекса;
* подготовка отчёта по преддипломной практике.

За время прохождения преддипломной практики все поставленные задачи были выполнены в полном объёме.

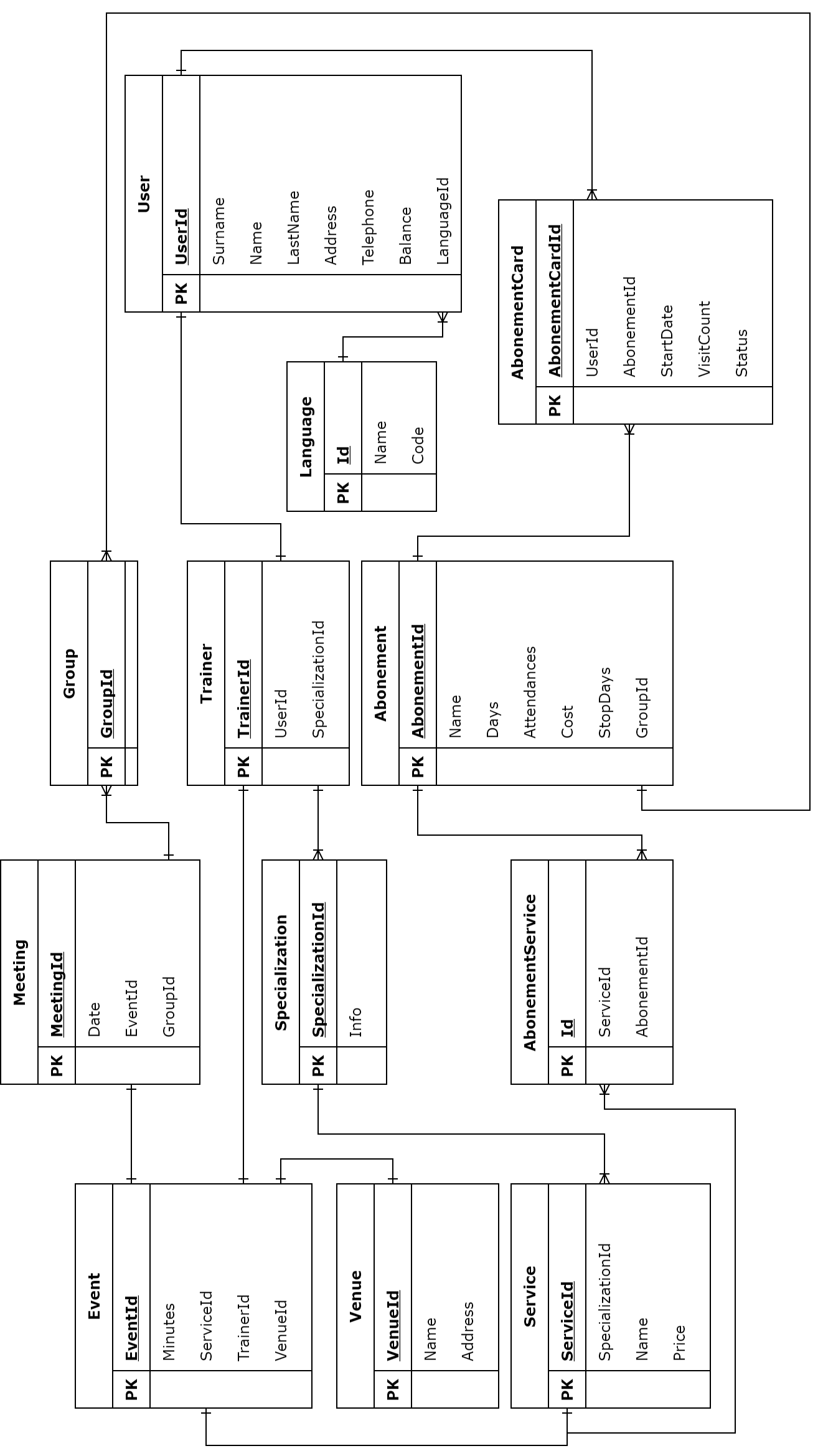
**Список использованных источников**

1. В. В. Вавилов. Основы менеджмента в фитнес-индустрии. – «Автор»,  2015. – 240 с.
2. Темякова Т. В. Особенности организации процесса оказания услуг в фитнес-клубах. – Научно-методический электронный журнал «Концепт». – 2017. – Т. 44. – С. 75–78.
3. Программное обеспечение для управления фитнес-клубом [Электронный ресурс]. – Mobifit, 2021. – Режим доступа: https://mobifitness.ru/. – Дата доступа – 21.04.2021.
4. Sport Priority – сервис планирования спорта и активного отдыха. [Электронный ресурс]. – Sport Priority, 2021. – Режим доступа: https://sportpriority.com/. – Дата доступа – 21.04.2021.
5. Облачный сервис для управления и автоматизации работы фитнес-клуба [Электронный ресурс]. – Fitbase , 2021. ­– Режим доступа: https://fitbase.io/. – Дата доступа – 21.04.2021.
6. Документация по ASP.NET [Электронный ресурс]. – Microsoft Docs, 2021. – Режим доступа: https://docs.microsoft.com/ru-ru/aspnet/core/?view=  
   aspnetcore-5.0. – Дата доступа – 21.04.2021.
7. Documentation ­[Электронный ресурс]. – Android Development, 2021. – Режим доступа: https://developer.android.com/docs. – Дата доступа – 21.04.2021.
8. Что такое NoSql? Нереляционные базы данных, модели данных с гибкой схемой [Электронный ресурс]. – AWS, 2021. – Режим доступа: https://aws.amazon.com/ru/nosql/ – Дата доступа – 21.04.2021.
9. Черемных, С.В. Структурный анализ систем: IDEF-технологии /   
   С.В. Черемных, И.О. Семенов, В.С. Ручкин. – Москва: Финансы и статистика. 2015. -208 с.

# ПРИЛОЖЕНИЕ А

(обязательное)

**Схема базы данных**

****