Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

Рязанский государственный радиотехнический университет

имени В.Ф. Уткина

Кафедра Автоматизированных Систем Управления

К защите

Руководитель работы:

_____

дата, подпись

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОЙ РАБОТЕ**

по дисциплине

**«Технологии программирования»**

Тема:

«Разработка информационно-поисковой системы для предметной области:

«Онлайн фитнес»

Выполнил студент группы 135

Бардин М.С.

_____

дата сдачи на проверку, подпись

Руководитель работы

Преподаватель каф. АСУ

Аникеев Д.В.

_____ _____

оценка дата защиты, подпись

Федеральное государственное бюджетное образовательное учреждение высшего образования
Рязанский государственный радиотехнический университет им.В.Ф.Уткина

Кафедра Автоматизированных Систем Управления

# З А Д А Н И Е

на курсовую работу по дисциплине

"Технологии программирования"

Студент — Бардин Максим Сергеевич группа – 135

1. Тема: Разработка информационно-поисковой системы «Онлайн фитнес»

2. Срок представления работы к защите - «_» _____ 2023г.

3. Исходные данные для проектирования:

3.1. Язык программирования – C#.
3.2. СУБД — MS SQL Compact Edition, MS SQL Express Edition
3.3. Технология объектного связывания данных – Entity Framework.
3.4. Общее количество таблиц - не менее 5.
3.5. Обязательные операции с данными — просмотр записей таблиц в виде списка, сортировка, поиск, добавление, изменение, удаление записей.
3.6. Все страницы сайт должны быть наполнены осмысленной информацией.
4. Содержание курсовой работы
4.1. Задание.
4.2. Пояснительная записка, оформленная по ГОСТ 19.404-79, содержащая:
4.2.1. Описание предметной области.
4.2.2. Структурную схему модели данных.
4.2.3. Структурную схему программы.

Руководитель работы_____ / Аникеев Д.В. /

(подпись)

Задание принял к исполнению_____ « » _____ 2023г.

(подпись)

Содержание

**Введение**

В современном мире забота о физическом здоровье и активном образе жизни стала одной из приоритетных задач для многих людей [1]. Фитнес и тренировки стали неотъемлемой частью повседневной жизни, помогая поддерживать физическую форму, повышать энергию и улучшать общее самочувствие [2]. Однако, со временем мир меняется, и появляются новые вызовы, связанные с доступностью фитнес-услуг и организацией тренировок.

Именно здесь информационно-поисковая система "Онлайн фитнес" становится незаменимым инструментом, позволяющим упростить и облегчить процесс занятий фитнесом [3]. Эта система предоставляет пользователю доступ к широкому выбору онлайн-фитнес-платформ и ресурсов, объединяет информацию о тренировках, тренерах, дает возможность выбрать оптимальный вариант в соответствии с индивидуальными предпочтениями и целями.

Важность фитнеса в современном мире трудно переоценить. Сидячий образ жизни, высокая нагрузка на работе, стресс и неправильное питание - все это приводит к ухудшению общего здоровья и возникновению различных заболеваний [4]. Регулярные тренировки и физическая активность помогают укрепить сердечно-сосудистую систему, улучшить обмен веществ, снизить риск ожирения и развития хронических заболеваний [5].

Однако, многие люди сталкиваются с преградами, мешающими им заниматься фитнесом: ограниченный доступ к спортивным клубам, занятость, отсутствие подходящих тренеров рядом с местом жительства или финансовые ограничения. В таких ситуациях онлайн-фитнес становится наиболее доступным и удобным решением, позволяющим людям заниматься спортом в любое удобное для них время и место.

Информационно-поисковая система "Онлайн фитнес" призвана упростить выбор и использование онлайн-фитнес-услуг [6]. Путем сбора и структурирования информации о различных онлайн-ресурсах, система предоставляет пользователям удобный и надежный инструмент для поиска и

выбора оптимальных вариантов фитнес-платформ, программ тренировок и тренеров. Таким образом, система значительно экономит время и средства пользователей, а также помогает им достигнуть своих фитнес-целей.

Разработка информационно-поисковой системы "Онлайн фитнес" представляет собой актуальную и важную задачу, отвечающую современным потребностям людей в здоровье и активном образе жизни.

# 1 Анализ предметной области

## 1.1 Описание предметной области

Фитнес — это активная деятельность, направленная на поддержание и улучшение физической формы, развитие мышц и укрепление здоровья [7]. Он включает в себя широкий спектр физических упражнений, тренировок и программ, направленных на достижение конкретных целей, таких как улучшение выносливости, силы, гибкости или потеря веса [8].

Фитнес стал неотъемлемой частью современной жизни, и важность его понимается все больше и больше людьми [9]. Он помогает не только поддерживать физическую форму, но и улучшать психологическое благополучие. Регулярные занятия фитнесом способствуют снижению стресса, улучшению настроения, повышению самооценки и общей жизненной энергии [10].

Существует множество различных видов фитнеса, чтобы удовлетворить разнообразные предпочтения и потребности [11]. Кардиотренировки, такие как бег, езда на велосипеде или аэробика, направлены на улучшение сердечно-сосудистой системы и потерю лишнего веса. Силовые тренировки, включая подъемы весов, отжимания и подтягивания, направлены на развитие мышц и силы. Гибкость и растяжка, включающие йогу и пилатес, помогают улучшить гибкость и координацию [12].

Однако, доступ к фитнес-услугам может быть ограничен различными факторами, такими как расстояние до спортивных клубов, занятость, финансовые возможности или отсутствие квалифицированных тренеров рядом с местом жительства. Именно здесь возникает потребность в онлайн-фитнесе.

Онлайн-фитнес предоставляет удобную альтернативу традиционным спортивным занятиям. С его помощью люди могут заниматься фитнесом в любое удобное для них время и место, используя различные онлайн-платформы, видеоуроки, тренировочные программы и тренеров. Онлайн-

фитнес становится все более популярным, и исследования показывают его эффективность и преимущества для здоровья и благополучия [13].

Однако, выбор подходящих онлайн-фитнес-ресурсов и программ может быть сложным и запутанным процессом. Пользователям может потребоваться время и усилия, чтобы найти надежные и качественные ресурсы, соответствующие их требованиям и целям.

Для облегчения этого процесса и поиска оптимальных решений создается информационно-поисковая система "Онлайн фитнес". Она предоставляет удобный и надежный инструмент, позволяющий пользователям искать, выбирать и использовать онлайн-фитнес-платформы, программы и тренеров. Система собирает и структурирует информацию о различных онлайн-ресурсах, предоставляет пользователю возможность ознакомиться с отзывами, оценками и рекомендациями, чтобы принять осознанное решение.

Таким образом, информационно-поисковая система "Онлайн фитнес" становится незаменимым инструментом для людей, стремящихся поддерживать свое физическое и психологическое благополучие через удобный доступ к качественным онлайн-фитнес-ресурсам и программам.

## 1.2 Выявление бизнес-процессов

Первом этапом в разработки информационной системы является обозначение бизнес-процессов. BPMN-модель для отображения основных процессов системы изображена на рисунке 1.1-1.2.

Описание бизнес-процессов информационной системы:

Пользователь вводит логин или пароль, с помощью базы данных мы проверяем существует ли такой логин и пароль. Если существует, то пользователь входит в аккаунт, иначе идет переадресация клиента к регистрации. Пользователь регистрируется, следом данные заносятся в базу данных.

Далее проверяется: есть ли у пользователя подписка на сервис или если она имеется, то не истек ли срок. Если подписка имеется то, спрашиваем у

пользователя хочет он внести новые продукты в аллергический список. Если подписки нет, переадресуем на страницу выбора подписки. Пользователь выбирает желаемый тариф и получает доступ к сервису.

Указав аллергические продукты информационная система составляет запрос на добавление в БД новых продуктов. Запрос выполняется внутри БД, далее уведомляем пользователя об успешном добавлении продукта в карту аллергика.

Следующем шагом является добавление новой цели – того результата, который хочет клиент. Информационная система также создает запрос на добавление новой цели и привязывает ее к клиенту.

После того, как клиент внес все добавления (изменения), ментор, прикрепленный к клиенту, указывает рекомендованные продукты для клиента и его диеты. Информационная система создает запрос на добавление новых продуктов в рекомендованный список, привязанный к клиенту.

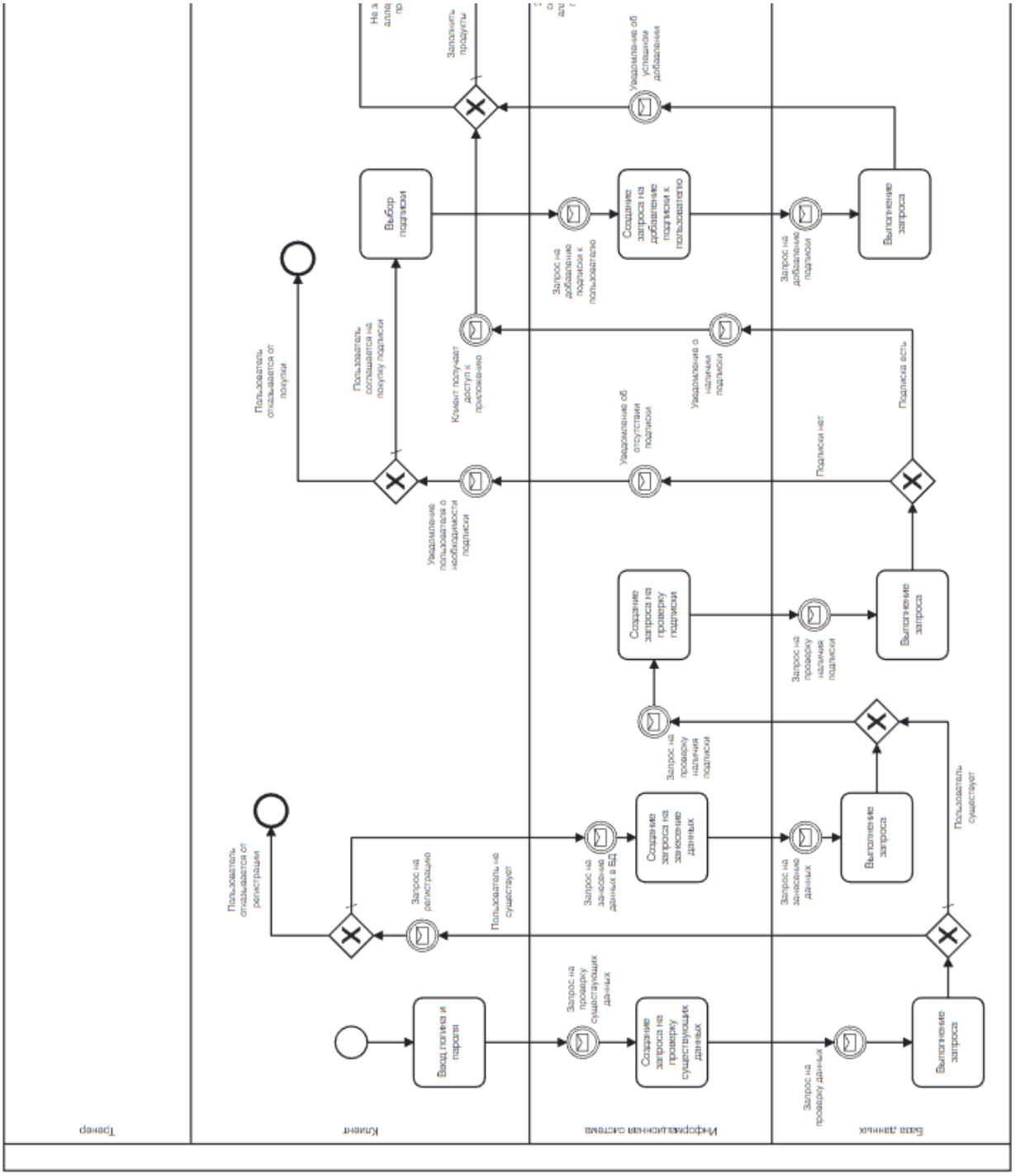


Рисунок 1.1 – Первая часть BPMN модели

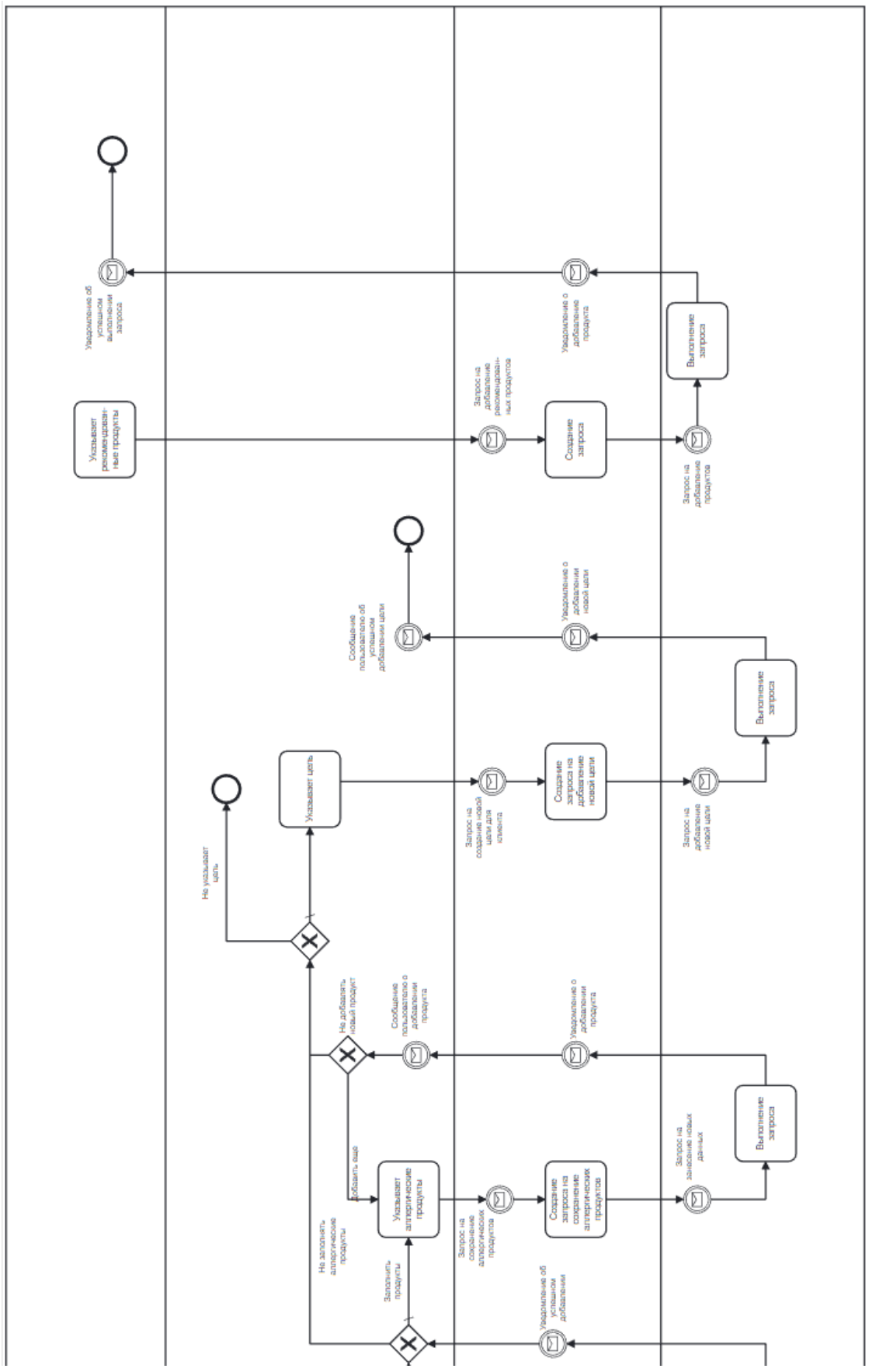


Рисунок 1.2 – Вторая часть BPMN модели

В дальнейшем следует описать работу сервиса. В самом начале клиент указывает свои начальные данные (массу тела, продукты, на которые у пользователя аллергия) и желаемые результаты (массу тела).

Ментор (тренер) – специально обученный человек, следящий за определенной группой клиентов и направляющий их на достижение указанной цели.

Ниже приведен перечень основных задач тренера.

1. Составить рацион питания.

2. Определить предполагаемую дату достижения цели.

3. Составить план тренировок.

4. Поддерживать связь с пользователем.

5. Отвечать на задаваемые клиентом вопросы.

Вышеописанные действия изображены на рисунке 1.3.

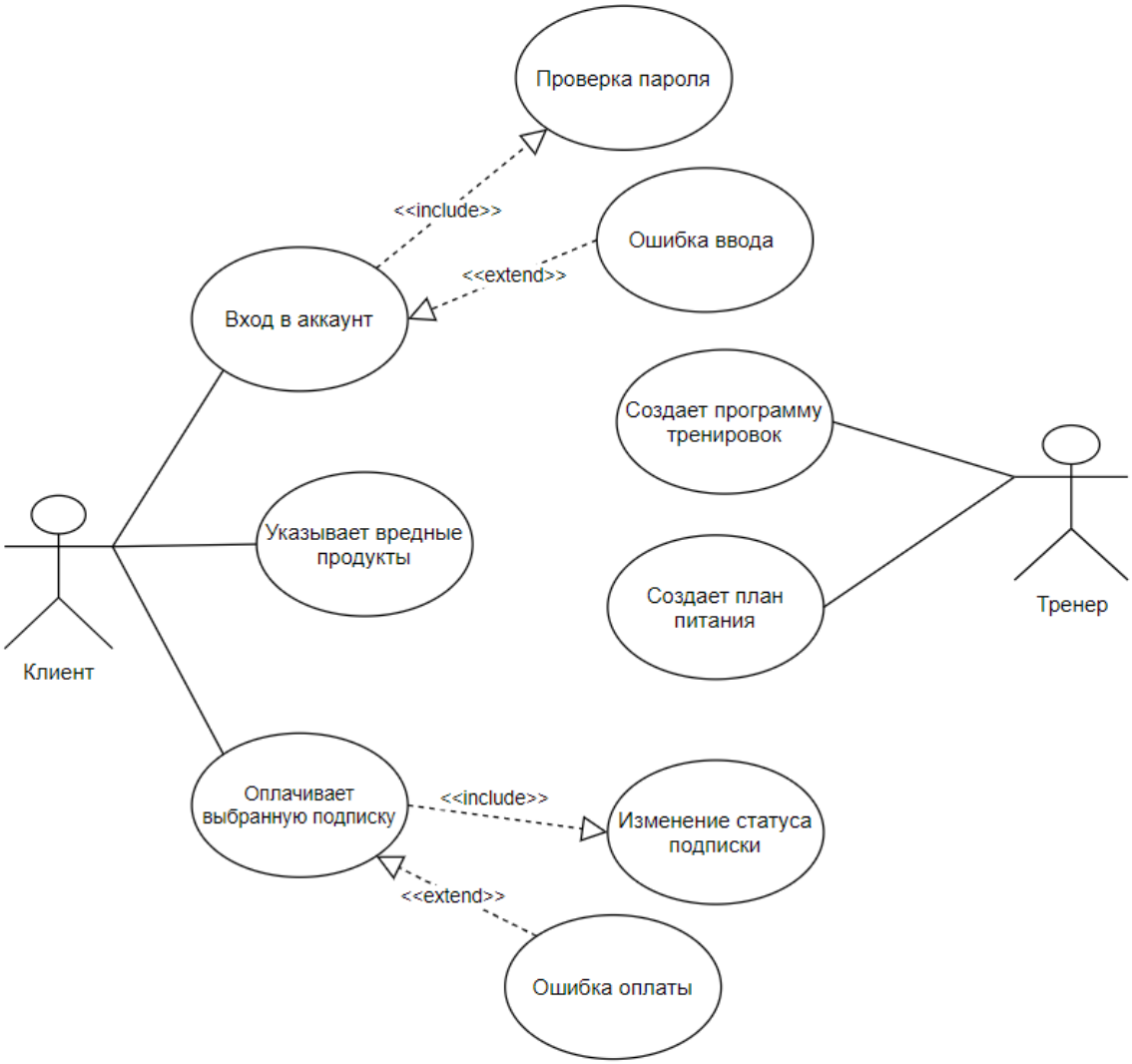


Рисунок 1.3 – Диаграмма прецедентов работы сервиса

## 2 Сравнительный анализ

На сегодняшний день Российская экономика в области фитнес-услуг во многом уступает Западной [1]. Уровень предложения в разы ниже, но хочется отметить, что за последние 5 лет Российский рынок взял темпы стремительного роста. Только в Санкт-Петербурге за последние 2 года уровень предложения увеличился почти в 1,5 раза. Емкость рынка фитнес-услуг на всероссийском уровне составляет 125 млрд. рублей. Большую часть рынка берет на себя, разумеется, Москва, но и Санкт-Петербург ничем не уступает в процентах.

Сегодня в России фитнессом занимаются около 3% населения России — это около 4,5 млн человек. Тем не менее прогресс на лицо. В 2010 году процент занимающихся составлял около 0,2% в количестве около 300 тыс. человек. И даже на сегодняшний день этого все равно недостаточно, чтобы удовлетворить желания всех клиентов, ведь больший спрос на услуги фитнесс помещений приходится только на жилые районы, где люди непосредственно проживают, так как никто не поедет на другой конец города, чтобы позаниматься спортом. По рисунку 2.1 можно отметить около половины емкости рынка фитнес услуг расположено в Москве, в Санкт-Петербурге около четверти всего рынка, а оставшуюся четверть делят между собой города миллионники и прочие города России в соотношении 2/1.

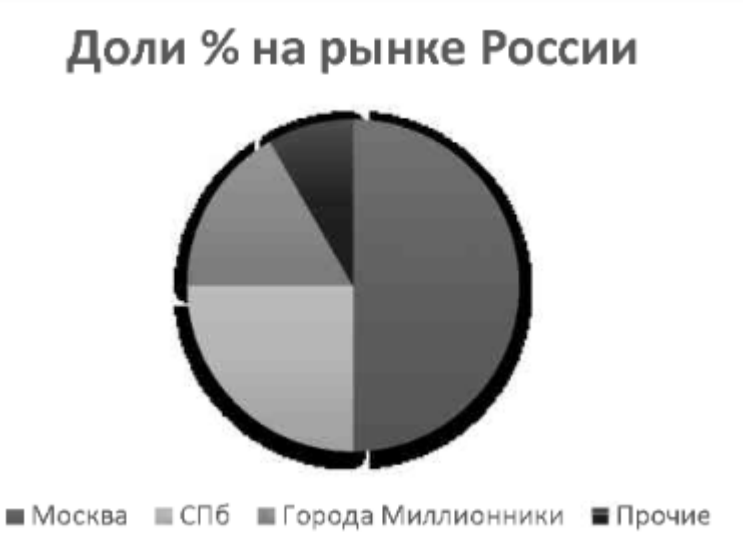


Рисунок 2.1 – Емкость рынка фитнес услуг РФ

Оценивая эффективность предложенной стратегии развития рынка фитнес- услуг следует отметить, незавершенность направленности рынка фитнес-услуг, поскольку на рынке не представлены фитнес клубы, которые распространяли бы в своей сети такую направленность. Да и введения ограничения на определенную группу общества накладывает специфические трудности при реализации стратегии, заставляя убирать многие факторы, которые могли бы наоборот помочь привлечь новых клиентов из общества в целом.

# 3 Разработка системы

## 3.1Архитектура информационной системы

Архитектура информационной системы "Онлайн фитнес" включает несколько ключевых компонентов, таких как база данных, слой доступа к данным (DAL), слой бизнес-логики (DOMAIN) и веб-часть системы [14].

База данных представляет собой хранилище, где сохраняются все данные, необходимые для функционирования системы. В данном случае, база данных содержит информацию о пользователях, тренировках, тренерах, отзывах и других сущностях, связанных с онлайн фитнесом. Она обеспечивает постоянное хранение и доступ к данным.

Слой доступа к данным (DAL) отвечает за взаимодействие с базой данных. Он предоставляет набор методов и функций для выполнения операций чтения, записи, обновления и удаления данных в базе. DAL принимает запросы от других компонентов системы, таких как слой бизнес-логики, и осуществляет соответствующие операции в базе данных.

Слой бизнес-логики (DOMAIN) содержит основную логику системы и обрабатывает бизнес-операции. Он включает различные компоненты и сервисы, связанные с функциональностью системы "Онлайн фитнес". Например, в этом слое могут быть реализованы сервисы для управления пользователями, тренировками, отзывами и другими бизнес-сущностями. DOMAIN служит промежуточным звеном между веб-частью системы и слоем доступа к данным, обеспечивая обработку запросов и предоставляя необходимые данные.

Сайт (веб-часть) является пользовательским интерфейсом системы "Онлайн фитнес" и позволяет пользователям взаимодействовать с функциональностью системы. Он предоставляет пользовательский интерфейс для регистрации, аутентификации, просмотра тренировок, выбора программы, оставления отзывов и других операций. Взаимодействие с базой данных происходит через сайт. При выполнении определенных операций, сайт

отправляет fetch-запросы на слой бизнес-логики (DOMAIN), который, в свою очередь, может отправить запросы на слой доступа к данным (DAL) для получения или обновления данных в базе данных.

Такая архитектура позволяет разделить различные компоненты системы на логические слои, что обеспечивает легкость поддержки, расширения и изменения системы в будущем. Она также обеспечивает отделение бизнес-логики от инфраструктурных компонентов, что способствует повышению гибкости и модульности системы. На рисунке 3.2 представлена UML диаграмма компонентов.

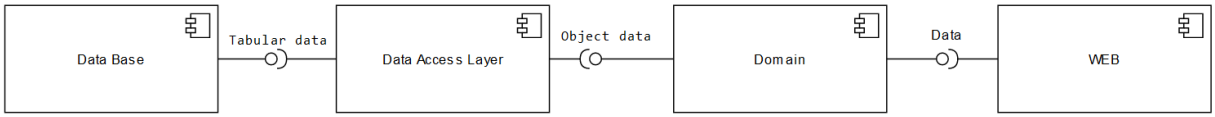


Рисунок 3.2 – Диаграмма компонентов

В архитектуре MVC (рисунок 3.3), модель представляет собой слой данных и бизнес-логики. Она отвечает за обработку данных, выполнение бизнес-логики и взаимодействие с базой данных [14]. В случае "Онлайн фитнес", модель может содержать классы и сервисы для работы с пользователями, тренировками, отзывами и другими бизнес-сущностями.

Представление отвечает за отображение данных пользователю. Это компонент, который отображает пользовательский интерфейс и предоставляет пользователю возможность взаимодействовать с системой. В случае "Онлайн фитнес", представление может быть реализовано в виде веб-страниц, форм, элементов управления и других пользовательских интерфейсных элементов.

Контроллер является посредником между моделью и представлением. Он обрабатывает пользовательские запросы, координирует взаимодействие между моделью и представлением, и обеспечивает передачу данных между ними. Контроллер может принимать входные данные от пользователя, вызывать соответствующие методы модели и обновлять представление с новыми данными [15].

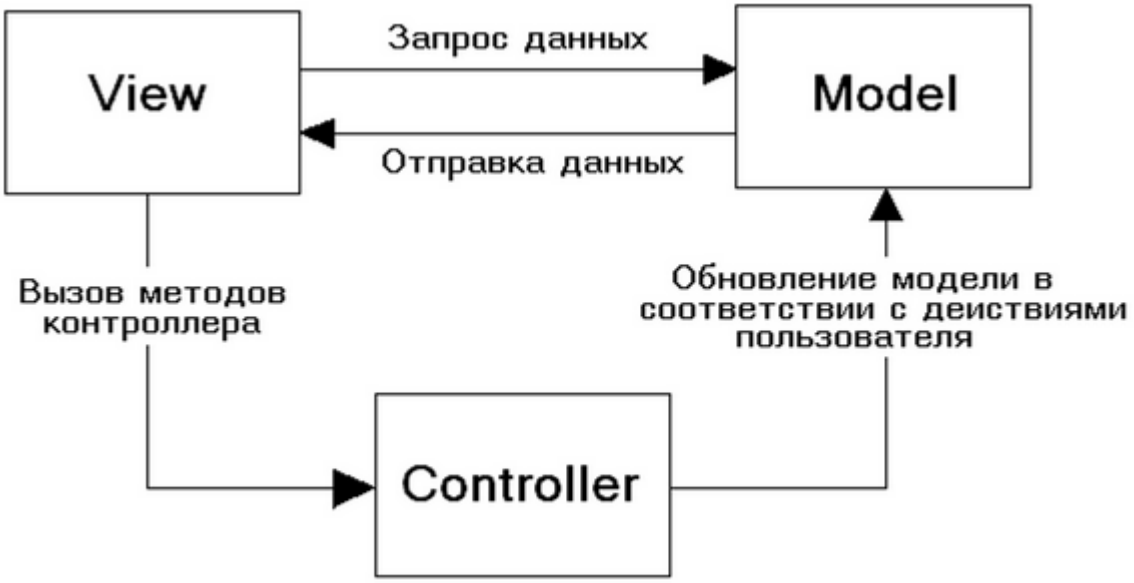


Рисунок 3.3 – Архитектура MVC

Преимущества архитектуры MVC включают разделение обязанностей и повторное использование кода [16]. Разделение компонентов позволяет улучшить читаемость, поддерживаемость и масштабируемость кода. Он также обеспечивает гибкость в изменении и расширении функциональности системы.

### 3.2 ER - моделирование

После создания use-case диаграммы, следующим шагом в разработке информационно-поисковой системы "Онлайн фитнес" является переход к ER-моделированию. ER-модель (Entity-Relationship model) позволяет описать основные сущности (entity) и их взаимосвязи в системе.

На основе анализа требований и use case диаграммы, можно определить основные сущности и связи между ними, которые будут представлены в ER-модели (рисунок 3.4).

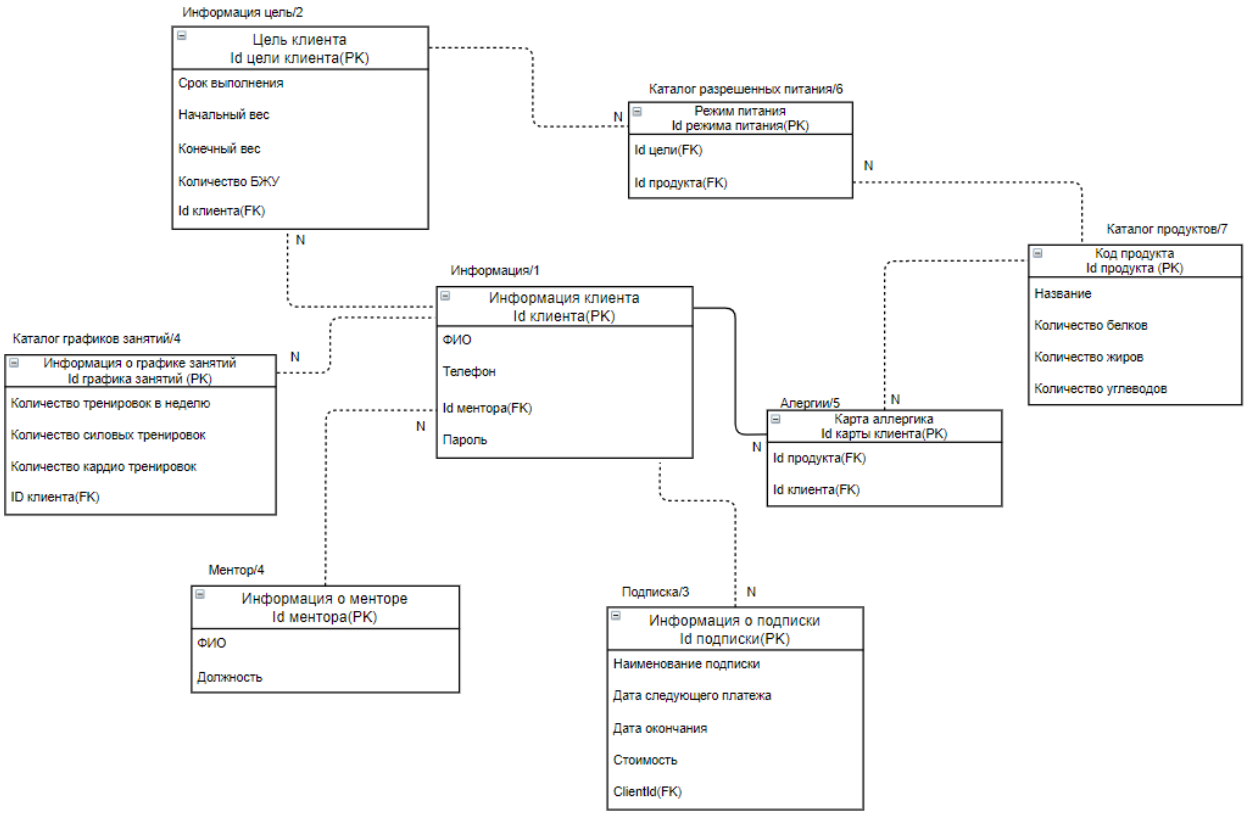


Рисунок 3.4 – Логическая ER – модель

Таблица №1. Информация о клиенте

Данная таблица предполагает хранение ФИО клиента, его номер телефона, которые предоставляется его личному ментору, ID ментора для ссылки на ментора и пароль для входа в приложение.

Таблица №2. Информация о цели клиента.

Данная таблица предполагает хранения информации о поставленной клиентом целью, то есть его начальный и конечный вес. Остальные поля таблицы заполняет ментор, то есть предполагаемый срок выполнения, количество разрешенных БЖУ (Белки Жиры Углеводы). Также в таблице содержится ссылка на Id Клиента.

Таблица №3. Информация о подписке.

Данная таблица содержит информацию о приобретенной подписки клиентом, то есть ее наименование, дата начала, дата окончания, стоимость. Также в таблице содержится ссылка на клиента.

Таблица №4. Информация о менторе.

Данная таблица содержит основную информацию о менторе, то есть его ФИО и должность.

Таблица №5. Карта аллергика.

Данная таблица является связывающим звеном, чтобы избежать связи N:N. В таблице содержится Id клиента, Id продукта.

Таблица №6. Каталог разрешенных продуктов.

Данная таблица является связывающим звеном, чтобы избежать связи N:N. В таблице содержится Id цели, Id продукта. Таблица является списком тех продуктов, которые можно употреблять одному клиенту

Таблица №7. Каталог продуктов.

Данная таблица является описанием продукта, который в дальнейшем будет распределен на разрешенный или запрещенный. Таблица содержит такие поля как: название продукта, количество белков, количество жиров, количество углеводов в продукте.

Таблица №8. Каталог графика занятий.

Данная таблица содержит в себе информацию об общем количестве тренировок, количестве силовых и кардио тренировок. Также в таблице имеется ссылка на клиента.

### 3.3 Проектирование базы данных

Проектирование базы данных является критическим этапом, определяющим эффективность и надежность хранения данных системы.

Одним из ключевых аспектов проектирования базы данных является соблюдение нормальных форм. Нормализация — это процесс организации данных в базе данных, чтобы минимизировать избыточность и обеспечить целостность данных. В базе данных могут использоваться различные нормальные формы, такие как первая нормальная форма (1NF), вторая нормальная форма (2NF), третья нормальная форма (3NF) и т. д. Каждая нормальная форма имеет свои требования к структуре и связям между таблицами в базе данных.

Проектирование базы данных играет важную роль в обеспечении эффективности системы [17]. Хорошо спроектированная база данных позволяет оптимизировать производительность операций чтения и записи данных, улучшить масштабируемость и обеспечить целостность данных. Она также облегчает сопровождение и расширение системы, поскольку изменения в структуре базы данных могут быть более простыми и безопасными.

Эффективная проектировка базы данных требует анализа и понимания требований системы, а также определения соответствующей структуры таблиц, связей и индексов. Важно учитывать различные факторы, такие как типы данных, объемы данных, ограничения целостности и требования к производительности системы.

В итоге, проектирование базы данных имеет важное значение для создания стабильной и эффективной информационной системы. Это

обеспечивает надежное хранение данных, упрощает обработку данных и способствует эффективной работе системы в целом.

После проведения анализа и обсуждения важности проектирования базы данных, перейдем к конкретной реализации базы данных для информационной системы "Онлайн фитнес" (рисунок 3.5).

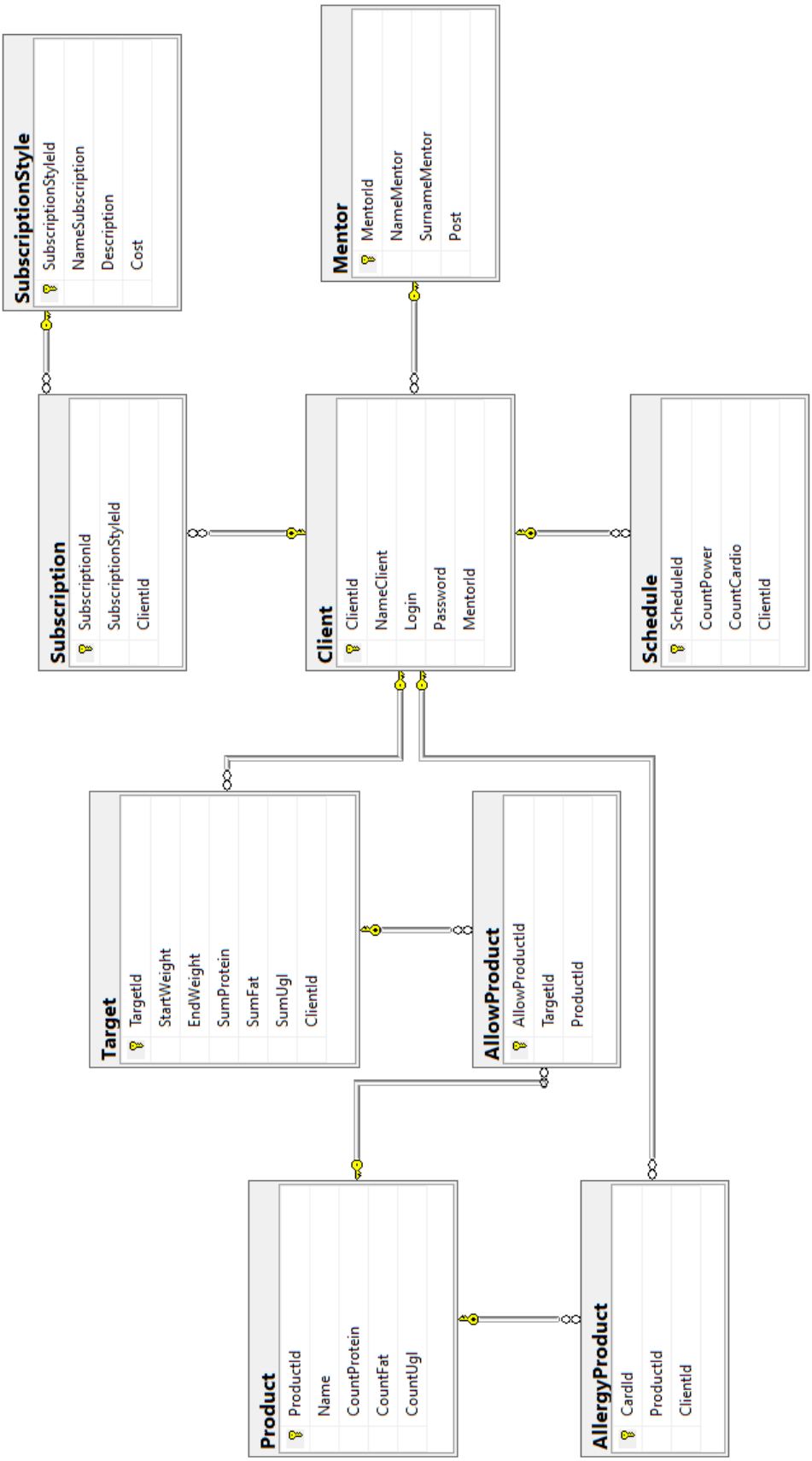


Рисунок 3.5 – База данных

## 3.4 DAL

Слой доступа к данным (DAL) является важной частью архитектуры информационной системы "Онлайн фитнес". Он отвечает за взаимодействие с базой данных и обеспечивает доступ к данным для других слоев системы, таких как слой бизнес-логики и пользовательский интерфейс [18].

Главная цель слоя DAL - абстрагировать остальные компоненты системы от специфических деталей работы с базой данных [19]. Он предоставляет единый интерфейс для выполнения операций чтения, записи, обновления и удаления данных. Это позволяет сократить дублирование кода, упростить поддержку системы и обеспечить единообразный подход к работе с данными в проекте.

Важность слоя DAL заключается в нижеописанных пунктах.

1. Разделение ответственности: Слой DAL отделяет логику доступа к данным от остальных компонентов системы. Это позволяет лучше организовать код, улучшить его читаемость и поддерживаемость. Изменения в базе данных или способе хранения данных могут быть легко внесены в слой DAL без влияния на другие компоненты системы.
2. Улучшение безопасности: Слой DAL может обеспечить контроль доступа к данным и предотвратить несанкционированный доступ или модификацию данных. Он может включать механизмы аутентификации и авторизации, защиту от инъекций и другие меры безопасности.
3. Улучшение производительности: Слой DAL может оптимизировать выполнение запросов к базе данных, включая кэширование данных, пакетную обработку и оптимизированные запросы. Это может улучшить производительность системы и снизить нагрузку на базу данных.

Таким образом выглядит дерево папок в DAL проекте (рисунок 3.6).

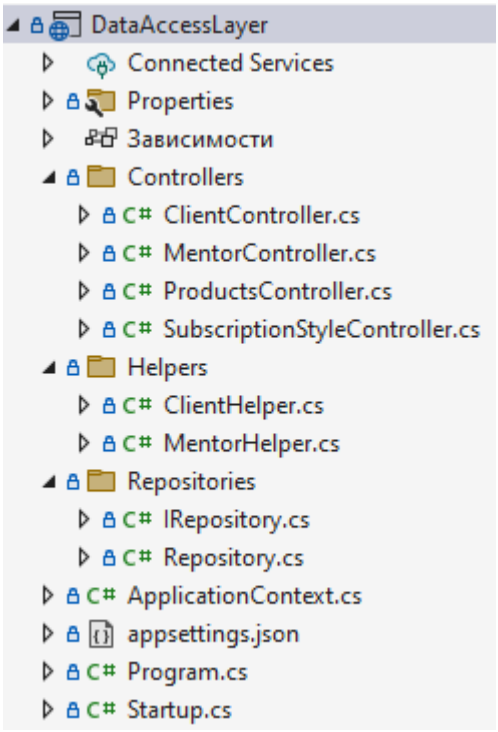


Рисунок 3.6 – Дерево папок в проекте DAL

В проекте "Онлайн фитнес" использование репозиториев с интерфейсом для DAL контроллера имеет несколько преимуществ и играет важную роль в обеспечении гибкости и удобства работы с данными.

Рассмотрим несколько причин, почему репозитории с интерфейсом нужен в контексте DAL контроллера.

1. Абстрагирование слоя доступа к данным: Репозитории предоставляют абстракцию над слоем DAL, скрывая детали конкретной реализации доступа к данным. Использование интерфейсов позволяет определить общие методы и операции для доступа к данным, независимо от конкретной базы данных или технологии. Это позволяет легко заменять или добавлять новые реализации репозиториев без необходимости изменения кода контроллеров;

2. Упрощение тестирования: Использование интерфейсов репозиториев позволяет создавать мок-объекты или поддельные реализации при написании модульных тестов. Это помогает в проведении тестирования контроллеров без реального взаимодействия с базой данных, что делает тестирование более простым, быстрым и независимым от внешних факторов;

3. Расширяемость и поддержка: Использование интерфейсов репозиториев упрощает добавление новых функций и возможностей в систему. Новые методы могут быть добавлены в интерфейс репозитория без изменения кода контроллера, что облегчает сопровождение проекта и его дальнейшее развитие;

В проекте "Онлайн фитнес" папка "models" с сущностями играет важную роль в организации и представлении данных в системе. В этой папке содержатся классы или структуры, которые представляют основные сущности и объекты, с которыми работает приложение.

Причин почему папка "models" с сущностями важна в проекте описанные в нижеприведенном списке.

1. Представление данных: Классы в папке "models" служат для представления данных и хранения информации о конкретных сущностях в системе. Каждый класс модели содержит свойства и методы, отражающие характеристики и поведение соответствующей сущности;

2. Упрощение обработки данных: Использование моделей помогает в организации и структурировании данных. Классы моделей могут содержать методы для валидации данных, обработки и преобразования информации. Это упрощает манипуляцию данными и повторное использование логики в различных компонентах системы;

3. Взаимодействие с другими слоями: Классы моделей могут использоваться как для передачи данных между различными компонентами системы, так и для сохранения данных в базе данных через слой DAL. Они служат своеобразным мостом между различными слоями приложения, обеспечивая структурированное представление данных.

В проекте "Онлайн фитнес" папка "controllers" играет важную роль в реализации бизнес-логики и обработке запросов от пользовательского интерфейса. В этой папке содержатся классы контроллеров, которые отвечают за прием и обработку запросов от клиента, взаимодействие с моделями и выполнение соответствующих действий.

Основные функции и преимущества папки "controllers" в проекте.

1. Управление потоком выполнения: Классы контроллеров являются своеобразной точкой входа для запросов от клиента. Они обрабатывают входящие запросы, определяют необходимые действия и передают управление соответствующим слоям или компонентам системы. Контроллеры координируют поток выполнения и обеспечивают связь между пользовательским интерфейсом и бизнес-логикой.

2. Взаимодействие с моделями: Классы контроллеров взаимодействуют с моделями (например, через репозитории DAL) для получения необходимых данных или изменения состояния системы. Они могут вызывать методы моделей, передавать параметры и получать результаты операций. Контроллеры обеспечивают связь между представлением данных и их обработкой.

3. Обработка и валидация запросов: Контроллеры выполняют проверку и валидацию входящих запросов от клиента. Они проверяют корректность данных, обрабатывают ошибки, управляют переходами между страницами или возвращают соответствующие ответы клиенту. Контроллеры обеспечивают безопасность и надежность обработки запросов.

4. Реализация бизнес-логики: Классы контроллеров содержат логику, связанную с бизнес-процессами и операциями системы. Они определяют, какие действия должны быть выполнены в ответ на запросы, и обеспечивают правильный порядок выполнения операций. Контроллеры реализуют бизнес-правила и обеспечивают логическую связь между различными компонентами системы.

Контроллеры в информационной системе, особенно в контексте веб-приложений, играют важную роль в обработке запросов и управлении потоком выполнения [20]. Они являются промежуточным звеном между пользовательским интерфейсом и бизнес-логикой приложения.

Важные аспекты контроллеров.

1. Обработка входящих запросов: Контроллеры получают входящие запросы от клиента, которые могут быть отправлены через различные протоколы, такие как HTTP. Запросы содержат информацию о действии, которое требуется выполнить (например, создание, чтение, обновление или удаление данных);

2. Маршрутизация запросов: Контроллеры определяют, какой метод или действие должно быть выполнено в ответ на конкретный запрос. Это осуществляется с помощью маршрутизации, которая связывает конкретный URL или путь запроса с соответствующим методом контроллера;

3. API и CRUD операции: В контексте веб-разработки, особенно с использованием архитектуры RESTful, контроллеры могут предоставлять API (Application Programming Interface) для взаимодействия с клиентскими приложениями. API позволяет выполнение различных операций над данными с использованием стандартных HTTP методов;

CRUD операции: CRUD - это акроним, который означает Create (Создание), Read (Чтение), Update (Обновление) и Delete (Удаление). Контроллеры могут предоставлять методы для выполнения этих операций над данными. Например, метод create() будет создавать новую запись, метод read() будет читать данные, метод update() будет обновлять данные, а метод delete() будет удалять данные;

4. Взаимодействие с моделями и слоем доступа к данным: Контроллеры взаимодействуют с моделями (сущностями) и слоем доступа к данным (DAL) для получения и обработки данных [21]. Они могут вызывать методы моделей для выполнения операций над данными или использовать репозитории для доступа к базе данных. Контроллеры служат связующим звеном между пользовательским интерфейсом, бизнес-логикой и хранилищем данных.

Преимущества контроллеров.

1. Организация логики приложения в структурированный способ;

2. Разделение ответственности и повышение модульности кода;

3. Удобное управление потоком выполнения и обработкой запросов;

4. Возможность использования API для взаимодействия с клиентскими приложениями.

Плавно перейдя от предыдущей темы о контроллерах, давайте рассмотрим API (Application Programming Interface) и его роль в информационных системах. API представляет собой интерфейс, который позволяет различным приложениям и сервисам взаимодействовать между собой. Он определяет набор правил и протоколов, по которым можно обмениваться данными и выполнить определенные операции.

API является важной составляющей современных информационных систем по нескольким причинам:

1. Интеграция и взаимодействие: API обеспечивает возможность интеграции различных компонентов системы или даже разных систем в целом. Он позволяет обмениваться данными и выполнять операции между различными приложениями, сервисами и платформами. Например, в нашем проекте "Онлайн фитнес" API может использоваться для интеграции с платежными системами, авторизацией пользователя или взаимодействием с мобильными приложениями;

2. Расширяемость и гибкость: API предоставляет гибкую архитектуру, которая позволяет разработчикам создавать новые функциональности или расширять существующие. Он позволяет создавать пользовательские приложения или сторонние сервисы, которые могут использовать возможности информационной системы. API может быть предоставлен как внутренним разработчикам, так и сторонним партнерам для расширения функциональности системы;

3. Стандартизация и унификация: Использование API позволяет унифицировать и стандартизировать способ взаимодействия и обмена данными. Разработчики могут использовать общие протоколы и стандарты для коммуникации с системой, что упрощает интеграцию и снижает

сложность разработки. API также может предоставлять документацию и описания интерфейсов, что упрощает взаимодействие для разработчиков;

4. Распределенная архитектура: API поддерживает распределенную архитектуру, где различные компоненты системы могут быть развернуты на разных серверах или даже в облаке. Это позволяет масштабировать систему и обеспечивает гибкость в развертывании и управлении компонентами.

В итоге, использование API в информационных системах обеспечивает гибкое взаимодействие, расширяемость и унификацию, что способствует развитию и интеграции различных компонентов и сервисов.

В нашем проекте "Онлайн фитнес" мы реализовали методы HTTP, такие как POST, GET, PUT и DELETE, для обеспечения функциональности создания, чтения, обновления и удаления данных. Эти методы являются стандартными для взаимодействия с ресурсами посредством HTTP-протокола.

Использование каждого из этих методов для каждой сущности в нашей информационной системе.

1. Метод POST (Create): Мы использовали метод POST для создания новых записей или сущностей в системе. Например, при регистрации нового пользователя, клиентское приложение отправляет POST-запрос с данными пользователя на соответствующий эндпоинт API. Затем контроллер обрабатывает этот запрос, создает новую запись пользователя и сохраняет ее в базе данных;

2. Метод GET (Read): Метод GET используется для получения данных из системы. Например, для получения списка всех доступных тренировок, клиентское приложение отправляет GET-запрос на эндпоинт API, который соответствует соответствующему контроллеру. Контроллер извлекает требуемые данные из базы данных и возвращает их в виде ответа на запрос;

3. Метод PUT (Update): Метод PUT используется для обновления существующих данных. Например, если пользователь хочет обновить

информацию о своем профиле, клиентское приложение отправляет PUT-запрос с обновленными данными на эндпоинт API. Контроллер обрабатывает запрос, обновляет соответствующую запись в базе данных с новыми данными пользователя;

4. Метод DELETE (Delete): Метод DELETE используется для удаления данных из системы.

Плавно перейдя от предыдущей темы о методах HTTP, давайте рассмотрим модели в нашем проекте. Модели представляют собой структуры данных, которые определяют атрибуты и свойства объектов в системе [22]. Они служат для описания сущностей, с которыми мы работаем, и их взаимосвязей.

Пример модели "Client" из проекта приведен в листинге 3.1.

Листинг 3.1. Модель “Client”

```
using System.ComponentModel.DataAnnotations;

namespace Domain.Models;

public class Client
{
    [Key]
    public int ClientId { get; set; }
    public string? NameClient { get; set; }
    public string? Login { get; set; }
    public string? Password { get; set; }
    public int MentorId { get; set; }

    public Mentor? Mentor { get; set; }

    public List<Subscription>? Subscriptions { get; set;}
}
```

В этой модели у нас есть следующие свойства:

ClientId: уникальный идентификатор клиента, который является первичным ключом.

NameClient: имя клиента.

Login: логин клиента.

Password: пароль клиента.

MentorId: идентификатор наставника, к которому привязан клиент.

Mentor: объект наставника, связанный с клиентом.

Subscriptions: список подписок клиента.

Эта модель представляет сущность "Клиент" в нашей информационной системе. Она определяет атрибуты, которые необходимы для представления клиента в базе данных и взаимодействия с ним в системе.

Модели позволяют нам описывать структуру данных и их связи в информационной системе. Они играют важную роль в проектировании базы данных и взаимодействии с ней.

Далее в нашем проекте мы реализовали контроллер ProductsController в слое доступа к данным (DAL). Контроллер отвечает за обработку HTTP-запросов, связанных с сущностью "Продукт".

Класс “ProductController” из проекта приведен в листинге 3.2.

Листинг 3.2. Контроллер “ClientController”

```
using Domain.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using DataAccessLayer.Repositories;

namespace DataAccessLayer.Controllers;

[ApiController]
[Route("api/DAL/Product")]
public class ProductsController : ControllerBase
{
    private readonly ILogger<ProductsController> _logger;
    private readonly IRepository _servicesProduct;

    public ProductsController(ILogger<ProductsController> logger, IRepository
servicesProduct)
    {
        _logger = logger;
        _servicesProduct = servicesProduct;
    }

    [HttpGet("ProductList")]
    public async Task<IActionResult> ProductList()
    {
        var product = await _servicesProduct.GetProduct();
        if (product == null) return NotFound();
        return Ok(product);
    }
```

```
    [HttpGet("{id}")]
    public async Task<IActionResult> ProductGet(int id)
    {
        var product = await _servicesProduct.GetProductById(id);
        return Ok(product);
    }

    [HttpGet("GetProductById/name={name}")]
    public async Task<IActionResult> ProductGet(string name)
    {
        var product = await _servicesProduct.GetProductByName(name);
        return Ok(product);
    }

    [HttpPost("AddProduct")]
    public async Task<IActionResult> AddProduct(Product product)
    {
        var result = await _servicesProduct.AddProduct(product);
        return Ok(result);
    }

    [HttpPut("UpdateProduct")]
    public async Task<IActionResult> UpdateProduct(Product product)
    {
        var result = await _servicesProduct.UpdateProduct(product);
        return Ok(result);
    }

    [HttpDelete("DeleteProduct/{Id}")]
    public async Task<IActionResult> DeleteProduct(int id)
    {
        var result = await _servicesProduct.DeleteProduct(id);
        return Ok(result);
    }
}
```

В контроллере мы определили несколько методов, каждый из которых отвечает за обработку определенного HTTP-запроса.

### 3.5 Domain

Далее в нашем проекте мы переходим к слою "Domain". Слой "Domain" представляет отдельный проект, который работает независимо от слоя доступа к данным (DAL). Он содержит бизнес-логику и модели, связанные с предметной областью нашей системы.

Слой "Domain" обеспечивает высокоуровневую обработку данных и взаимодействие с различными компонентами системы [23]. Он содержит модели данных, которые представляют сущности и их взаимосвязи в предметной области, а также логику, которая определяет, как эти данные обрабатываются и взаимодействуют между собой.

Один из ключевых аспектов слоя "Domain" — это его независимость от слоя доступа к данным (DAL). Это означает, что он не зависит от конкретной реализации хранилища данных или способа доступа к данным. Вместо этого, слой "Domain" определяет абстрактные интерфейсы и контракты, которые слой доступа к данным (DAL) должен реализовать.

Такая архитектура позволяет нам достичь разделения ответственности и улучшить модульность нашей системы [24]. Кроме того, это облегчает тестирование и поддержку кода, так как бизнес-логика и модели находятся в отдельном проекте и могут быть протестированы независимо от слоя доступа к данным.

Далее в листинге 3.3 рассмотрим конкретную реализацию контроллера в слое "Domain".

Листинг 3.3. Контроллер “ProductController”

```
using Domain.Models;
using Microsoft.AspNetCore.Mvc;
using System.Text.Json;

namespace Domain.Controllers;

[ApiController]
[Microsoft.AspNetCore.Mvc.Route("api/Domain/Product")]
public class ProductsController : ControllerBase
{
    private readonly string? _dalUrl;
    private readonly HttpClient _client;

    public ProductsController(IConfiguration conf)
    {
        _dalUrl = conf.GetValue<string>("DalUrl");
        _client = new HttpClient();
    }

    [HttpGet("ProductList")]
    public async Task<ActionResult<Product[]>> GetProductsByname(string? name)
    {
        var response = await
_client.GetAsync($"{_dalUrl}/api/DAL/Product/ProductList?name={name}");
        response.EnsureSuccessStatusCode();
        var result = await response.Content.ReadFromJsonAsync<Product[]>() ??
Array.Empty<Product>();
        if (string.IsNullOrWhiteSpace(name)) return result;
        return Array.FindAll(result, p => !string.IsNullOrWhiteSpace(p.Name) &&
p.Name.Contains(name));
    }

    [HttpPost]
    public async Task<ActionResult<Product>> PostProduct(Product product)
    {
```

```
        JsonContent content = JsonContent.Create(product);
        using var result = await
_client.PostAsync($"{_dalUrl}/api/DAL/Product/AddProduct",content);
        var dalProduct = await result.Content.ReadFromJsonAsync<Product>();
        Console.WriteLine($"{dalProduct?.Name}");

        if (dalProduct == null)
            return BadRequest();
        else
            return dalProduct;
    }

    [HttpGet("ProductId={id}")]
    public async Task<ActionResult<Product>> GetProduct(int id)
    {
        var response = await
_client.GetAsync($"{_dalUrl}/api/DAL/Product/GetProduct/id={id}");
        response.EnsureSuccessStatusCode();
        var content = await response.Content.ReadAsStringAsync();
        if (content == null) return NotFound();

        return JsonSerializer.Deserialize<Product>(content, new
JsonSerializerOptions
        {
            PropertyNameCaseInsensitive = true
        });
    }
}
```

В данном коде представлен контроллер ProductsController, который отвечает за обработку HTTP-запросов, связанных с сущностью "Product".

Контроллер наследуется от базового класса ControllerBase и имеет атрибуты [ApiController] и [Route("api/Domain/Product")], которые определяют базовый путь для всех его методов.

В конструкторе контроллера мы получаем URL слоя доступа к данным (DAL) из конфигурации приложения и создаем экземпляр класса HttpClient для взаимодействия с DAL.

Ниже приведены методы контроллера и их функциональность:

GetProductsByname: Этот метод отвечает на HTTP GET-запросы к маршруту "ProductList". Он отправляет GET-запрос к слою DAL, получает список продуктов и фильтрует его по имени (если указано). Результат возвращается в формате ActionResult<Product[]>.

PostProduct: Этот метод отвечает на HTTP POST-запросы. Он отправляет POST-запрос к слою DAL для добавления нового продукта. Данные продукта передаются в теле запроса. Результат возвращается в формате ActionResult<Product>.

GetProduct: Этот метод отвечает на HTTP GET-запросы к маршруту "ProductId={id}". Он отправляет GET-запрос к слою DAL, чтобы получить информацию о продукте с заданным идентификатором. Результат возвращается в формате ActionResult<Product>.

В методах контроллера мы используем экземпляр HttpClient для отправки HTTP-запросов к слою DAL и получения ответов. Мы также используем различные методы расширения, такие как ReadFromJsonAsync и ReadAsStringAsync, для обработки содержимого ответов в нужный формат.

### 3.6 Web

Далее рассмотрим реализацию веб-части (View) проекта, которая основана на использовании библиотеки React. В данном контексте мы будем обсуждать две популярные библиотеки для выполнения HTTP-запросов: Fetch и Axios.

Fetch является встроенной функцией в современных браузерах и предоставляет простой и гибкий способ выполнения HTTP-запросов [25]. Он основан на промисах и обладает минимальным объемом кода. Fetch предоставляет широкие возможности для настройки запросов и обработки ответов.

Axios, с другой стороны, является популярной сторонней библиотекой для выполнения HTTP-запросов как в браузере, так и в Node.js [26]. Он также основан на промисах, но предоставляет дополнительные функции, такие как автоматическая сериализация и десериализация данных, интерцепторы запросов и ответов, а также поддержку отмены запросов.

Разница между Fetch и Axios заключается в уровне абстракции и функциональности. Fetch предоставляет более низкоуровневый интерфейс, позволяя разработчику более точно настраивать запросы и обрабатывать ответы. С другой стороны, Axios предоставляет более высокоуровневый и

удобный интерфейс, который упрощает выполнение запросов и обработку данных.

Перейдем к обсуждению преимуществ одностраничных приложений (SPA) по сравнению с многостраничными приложениями (MPA).

Одностраничные приложения (SPA) представляют собой приложения, в которых весь контент загружается один раз при загрузке страницы, а затем динамически обновляется без перезагрузки страницы при взаимодействии пользователя [27]. Это достигается с помощью технологий, таких как AJAX и JavaScript-фреймворки, включая React.

Преимущества SPA.

1. Более плавный пользовательский опыт: SPA обновляет только необходимые части страницы без полной перезагрузки, что создает более быстрый и плавный пользовательский интерфейс;

2. Улучшенная отзывчивость: поскольку вся логика приложения выполняется на клиентской стороне, SPA может обрабатывать пользовательские действия мгновенно без необходимости отправки запросов на сервер и ожидания ответа;

3. Уменьшенная нагрузка на сервер: SPA загружает только необходимые данные и ресурсы, уменьшая нагрузку на сервер и снижая время отклика;

4. Улучшенная переносимость: SPA может работать на различных устройствах и платформах, включая десктопы, мобильные устройства и планшеты, благодаря использованию веб-технологий;

5. Легкая поддержка API: SPA хорошо сочетается с API-интерфейсами, такими как RESTful API, поскольку они позволяют передавать и обрабатывать данные в формате JSON.

Продолжим рассмотрение разработки веб-части проекта, сфокусируясь на использовании React Hooks. React Hooks — механизм в React, который позволяет работать полностью без классов. Он облегчает повторное

использование кода для решения общих задач. Сейчас это основной способ написания React-приложений [28]В проекте были использованы два основных: useEffect и useState.

UseEffect позволяет выполнять побочные эффекты в функциональных компонентах React [29]. Он выполняется после каждого рендера компонента и позволяет выполнять определенные действия, такие как загрузка данных, подписка на события или изменение состояния компонента. UseEffect принимает два аргумента: функцию-эффект и массив зависимостей. Функция-эффект будет вызываться при каждом рендере компонента, если указанные зависимости изменились. Это позволяет контролировать, когда и как часто эффект будет выполняться.

UseState позволяет добавлять состояние в функциональные компоненты React[30]. Он возвращает пару значений: текущее состояние и функцию для его обновления. При вызове функции обновления состояния React перерисует компонент с новым состоянием. UseState можно использовать для управления данными, формами, состоянием компонента и другими аспектами, которые требуют хранения и обновления состояния.

Преимущества использования useEffect и useState в React включают.

1. Упрощенная работа с побочными эффектами и состоянием в функциональных компонентах;
2. Более чистый и понятный код благодаря использованию декларативного стиля программирования;
3. Улучшенная производительность и оптимизация благодаря правильной настройке зависимостей useEffect;
4. Упрощенная работа с асинхронными операциями, такими как загрузка данных из API.

Приведенный в листинге 3.4 пример кода демонстрирует вывод информации о подписках на сайт с использованием React компонента Subscription.

```
import React, { useEffect, useState } from "react";
import { Button } from "react-bootstrap";
import axios from "axios";
import "../styles/main.css";

export default function Subscription() {
  const [data, setData] = useState([]);

  const fetchData = async () => {
    try {
      const res = await axios.get('/api/Domain/SubscriptionStyle');
      setData(res.data);
    } catch (error) {
      console.error(error);
    }
  };

  useEffect(() => {
    fetchData();
  }, []);

  return (
    <div>
      <h1>Subscription:</h1>
      {data.map((post) => {
        return (
          <div key={post.subscriptionStyleId}>
            <strong>
              {post.subscriptionStyleId}. {post.nameSubscription} - {post.cost}
            </strong>
            <div>
              {post.description}
              <Button style={{ marginLeft: 40 }}>Buy</Button>
            </div>
          </div>
        );
      })}
    </div>
  );
}
```

В данном примере используется useEffect для выполнения асинхронного запроса данных о подписках с помощью функции fetchData. При первом рендере компонента useEffect будет вызван и выполнит запрос к API. Полученные данные сохраняются в состоянии компонента через useState. Затем данные выводятся на сайт с помощью метода map, который создает отдельный элемент для каждой подписки. Каждая подписка отображается с указанием ее идентификатора, названия, стоимости и описания. Также для каждой подписки предоставляется кнопка "Buy".

Этот пример демонстрирует, как можно использовать React и библиотеку axios для получения данных с сервера и динамического вывода их на веб-страницу.

# 4 Тестирование

В разделе тестирования программы, одним из ключевых инструментов, который будет использован, является Swagger. Swagger представляет собой мощный инструмент для документирования и тестирования API. Он позволяет разработчикам легко и удобно взаимодействовать с API и выполнять различные CRUD операции над сущностями.

Используя Swagger, будет продемонстрирован процесс выполнения CRUD операций для сущности "Product". Через удобный пользовательский интерфейс Swagger, можно будет создавать новые продукты, обновлять существующие, удалять и получать информацию о продуктах. Этот метод тестирования позволяет разработчикам проверить функциональность и корректность работы API, а также убедиться в правильной работе с данными.

Однако, помимо тестирования через Swagger, также будет представлен вариант тестирования системы через веб-интерфейс сайта. Этот метод тестирования предоставляет более реалистическую среду, позволяющую эмулировать действия пользователей и проверять работу системы в реальных условиях.

Тестирование через веб-интерфейс сайта обеспечит проверку удобства использования, надежности и соответствия функциональных требований системы. Такой подход позволяет убедиться в корректной работе не только API, но и взаимодействия различных компонентов системы в целом.

Таким образом, тестирование программы будет осуществляться как через Swagger, так и через веб-интерфейс сайта, чтобы обеспечить полную проверку функциональности, надежности и удобства использования разрабатываемой информационной системы.

## 4.1 Тестирование с помощью swagger

Тестирование операции Get (рисунок 4.1).

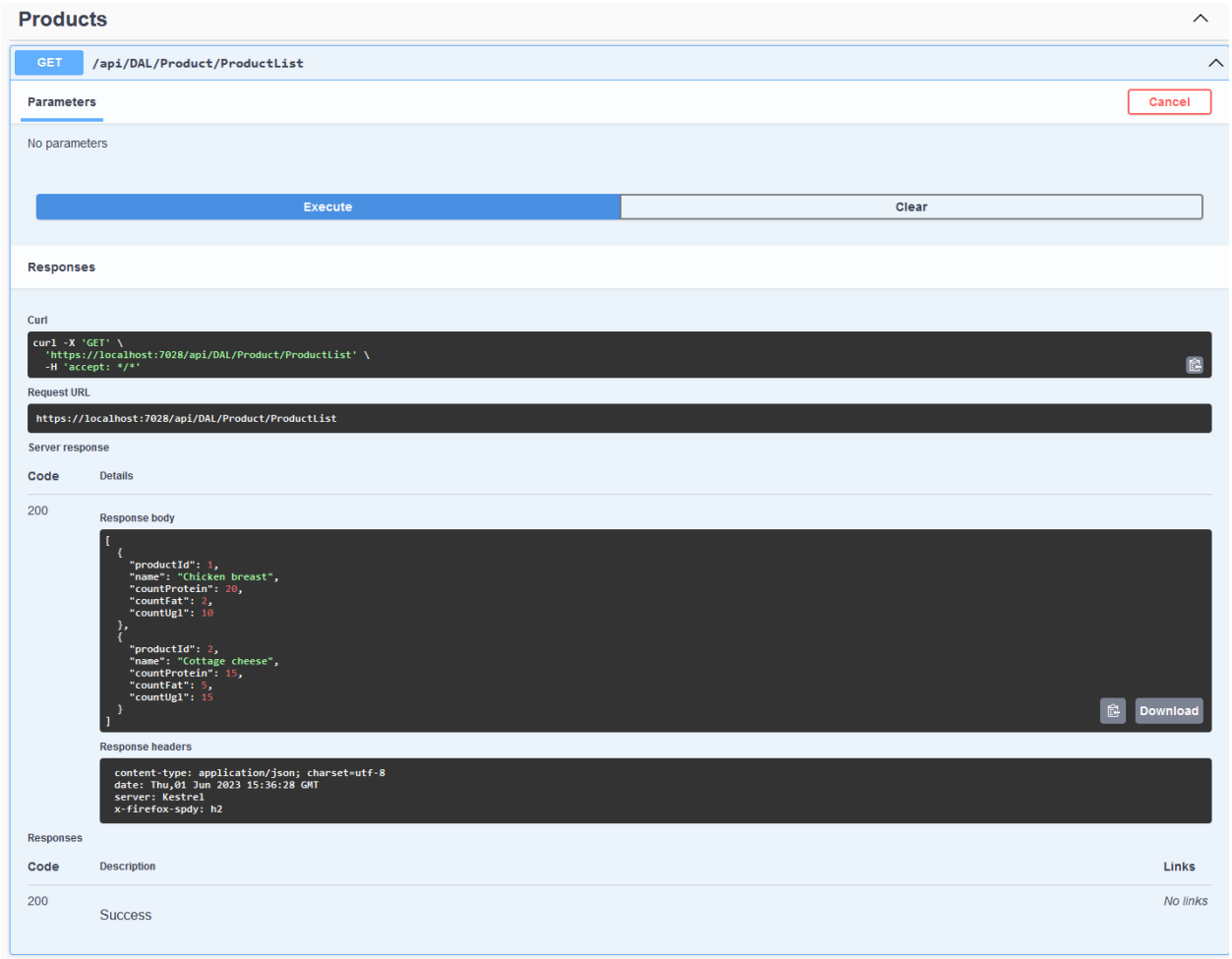


Рисунок 4.1 – Операция Get над сущностью «Product»

Тестирование операции Post (рисунок 4.2).

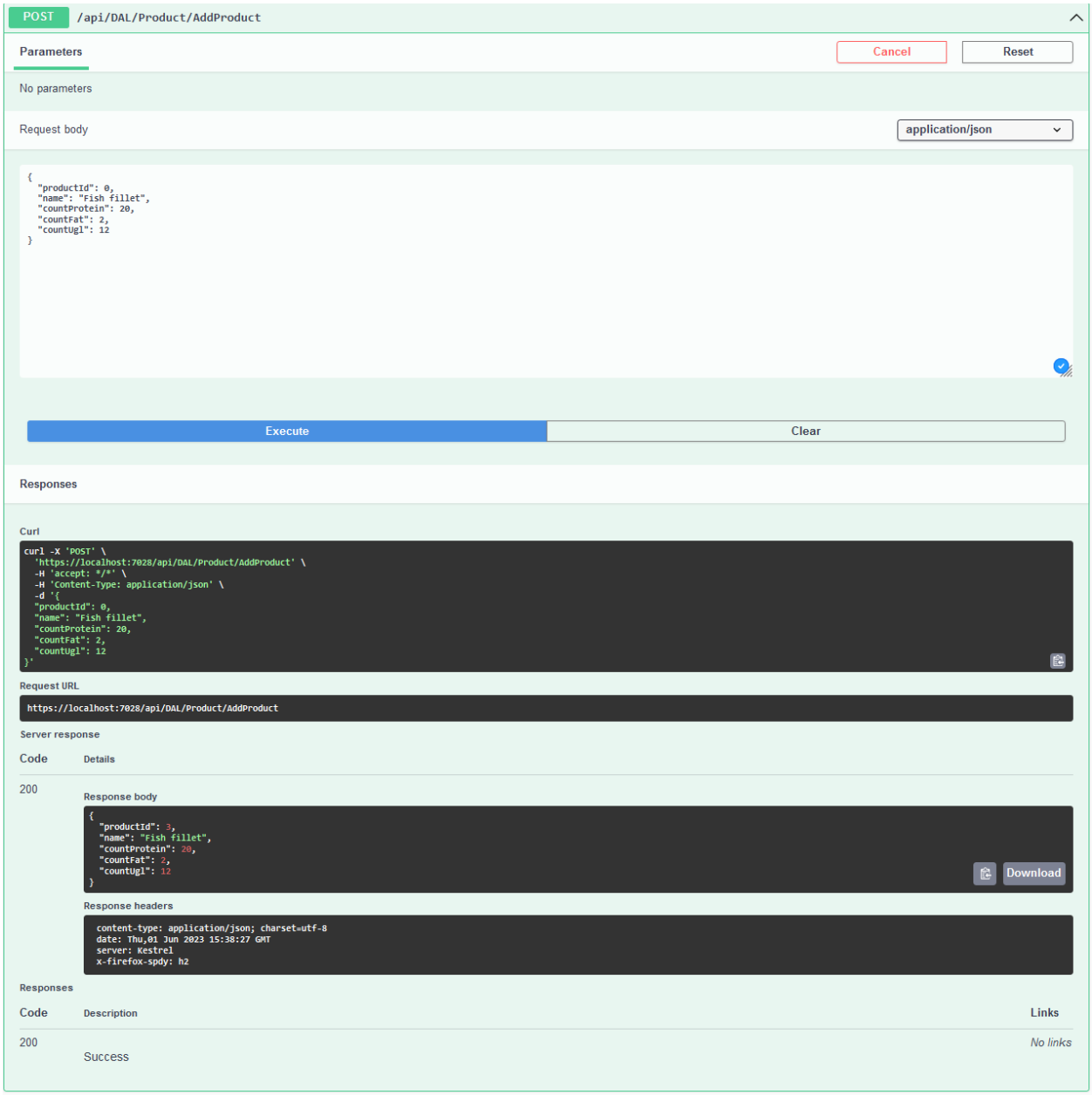


Рисунок 4.2 - Операция Post над сущностью «Product»

Тестирование операции Update (рисунок 4.3).

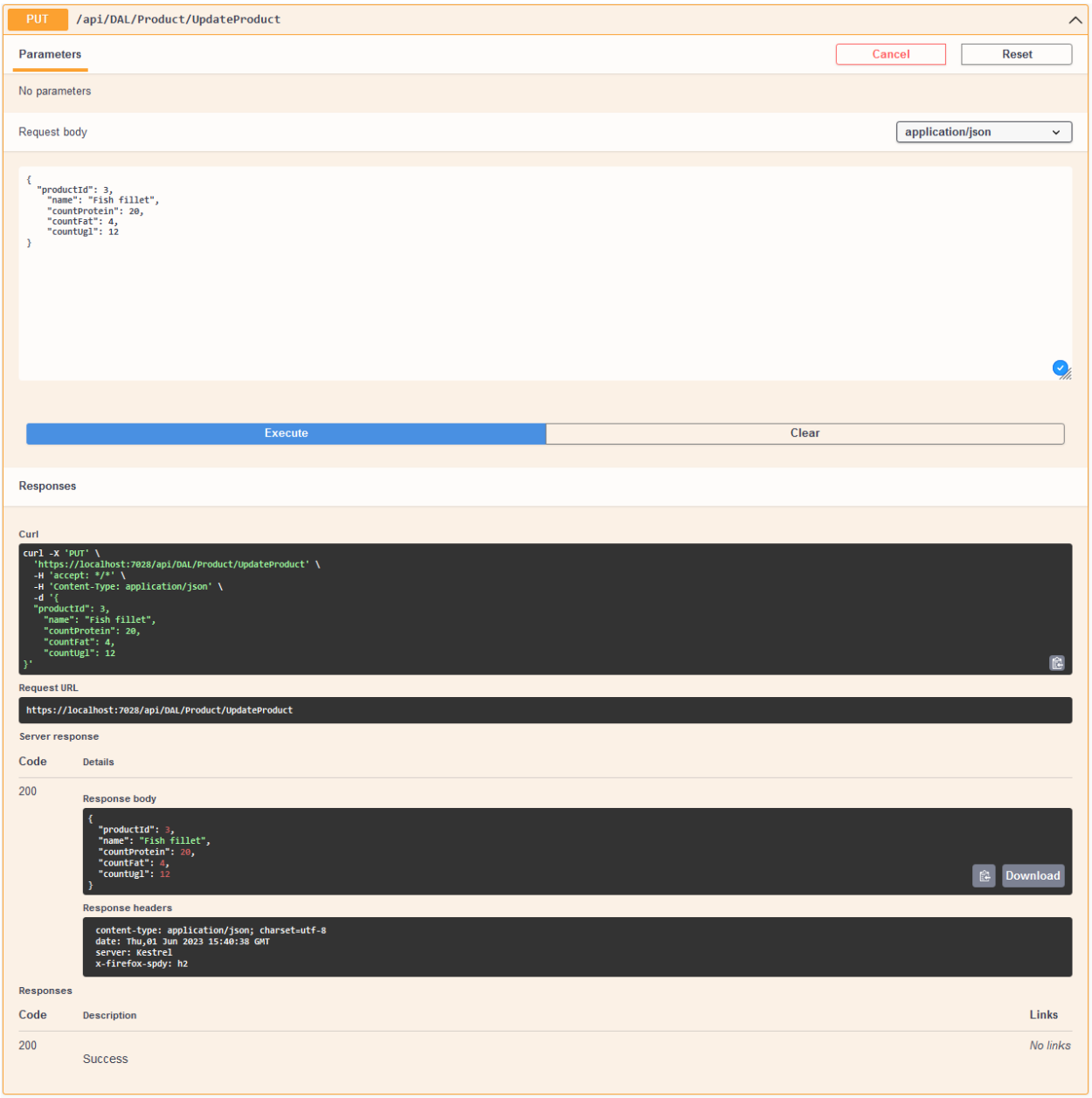


Рисунок 4.2 - Операция Update над сущностью «Product»

Тестирование слоя domain.

Для тестирования операции GET по имени продукта в swagger, будет введена букву "C" в поле "Имя продукта" (рисунок 4.3).

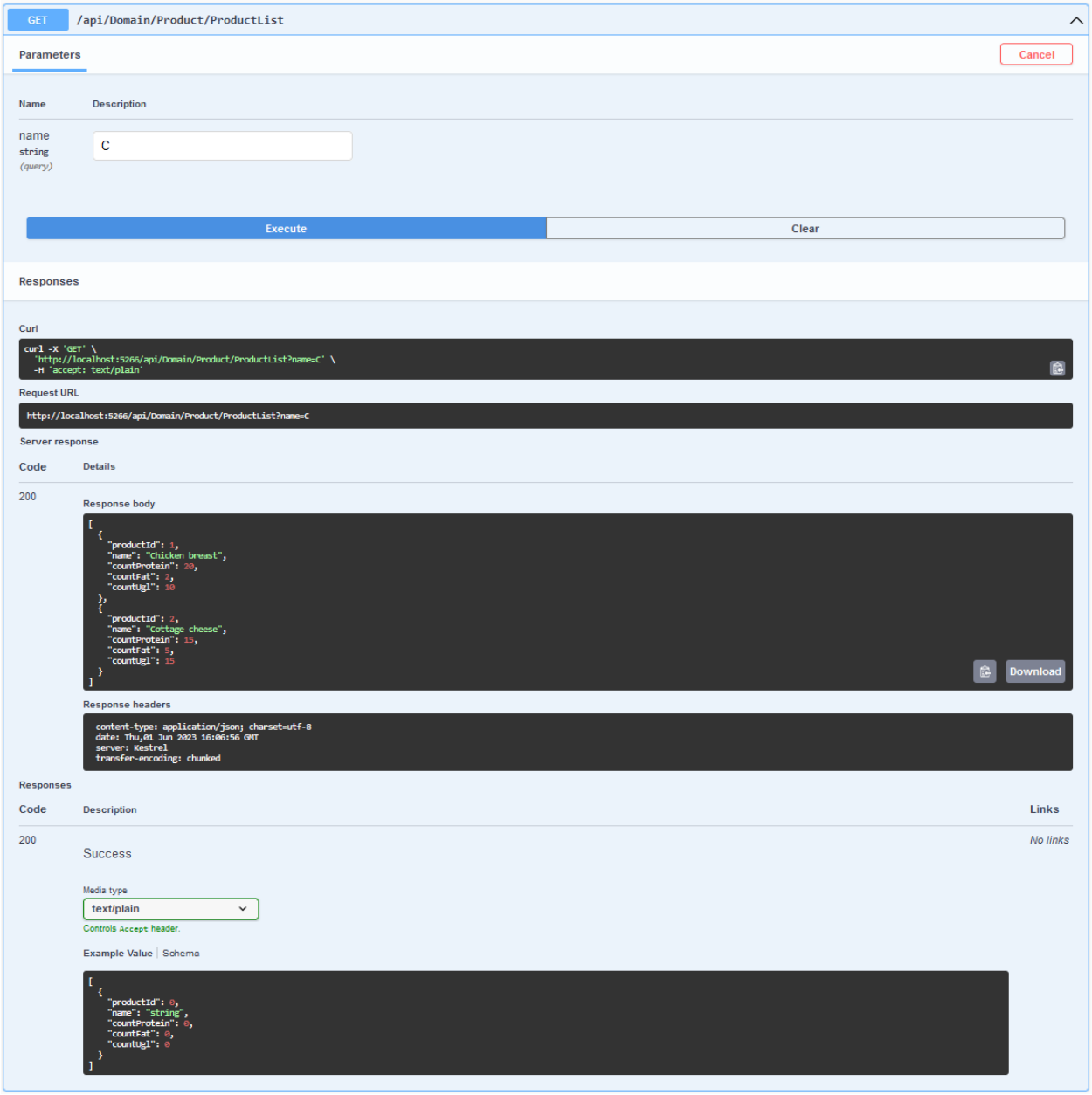


Рисунок 4.3 – Get запрос по имени «C»

### 4.2 Тестирование web слоя

На веб-интерфейсе реализованы два метода GET, которые позволяют получить таблицу с продуктами и выполнить поиск по имени продукта. При загрузке страницы пользователю отображается таблица с информацией о всех доступных продуктах, включая их название и описание (рисунок 4.4).

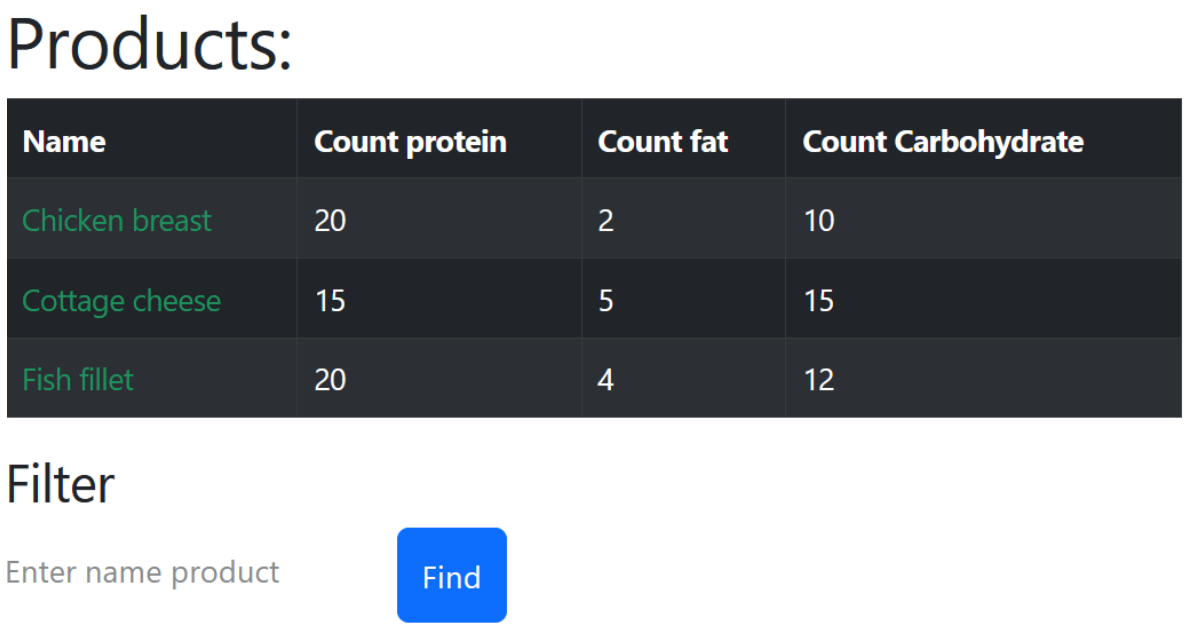


Рисунок 4.4 – Вывод всех продуктов

Также на веб-интерфейсе присутствует возможность выполнить поиск продукта по его имени. Пользователь может ввести имя продукта в соответствующее поле и нажать кнопку find (рисунок 4.5).

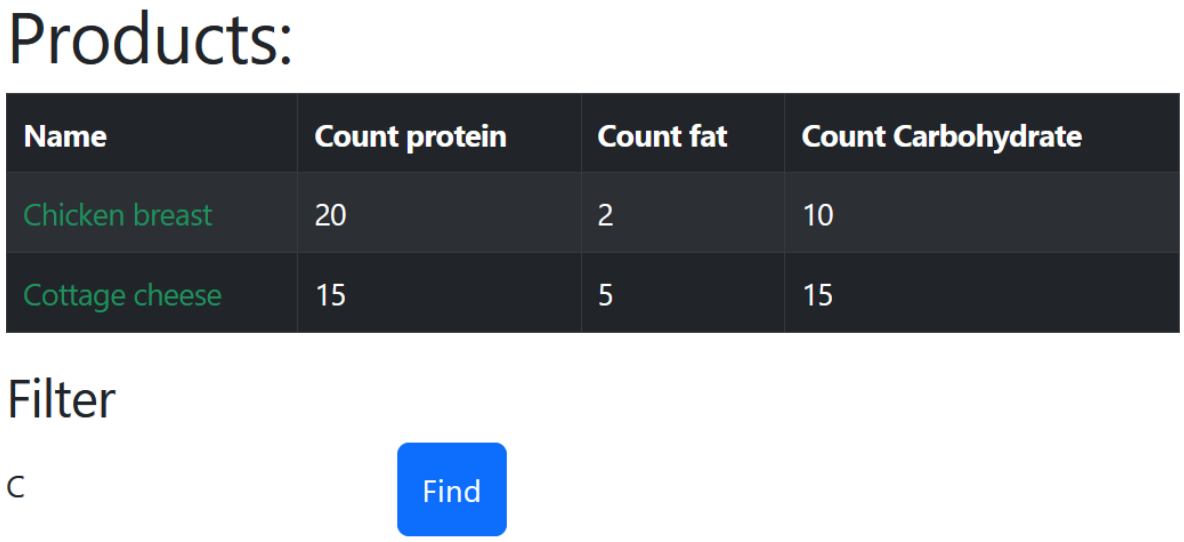


Рисунок 4.5 – Вывод продуктов, имя которых начинается на букву «С»

Далее на специальной форме пользователь может добавить продукт (рисунок 4.6).

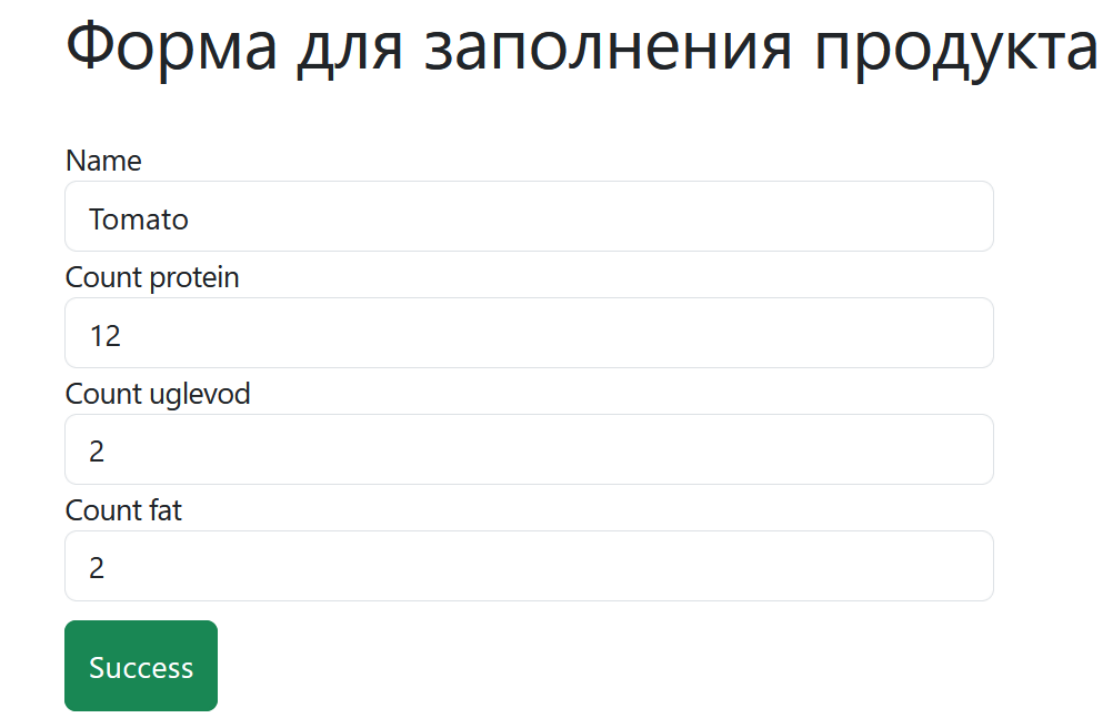


Рисунок 4.6 – Заполнение продукта

Результат выполнения (рисунок 4.7).

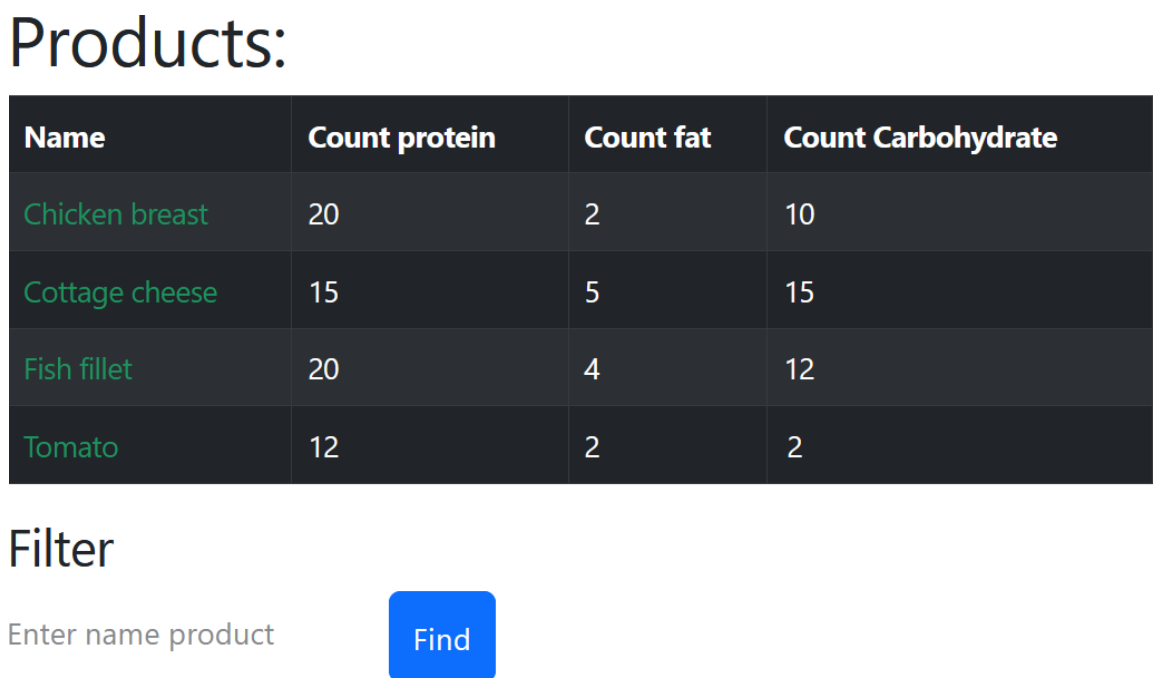


Рисунок 4.7 – Результат ввода нового продукта

## Заключение

В заключение данной курсовой работы были рассмотрены основные аспекты разработки информационно-поисковой системы "Онлайн фитнес". В процессе работы были использованы технологии, приведенные ниже.

1. ASP.NET Core: использовался для создания серверной части системы, включая API-контроллеры и управление данными;
2. Entity Framework Core: использовался для работы с базой данных, обеспечивая доступ к данным и выполнение запросов;
3. React: использовался для разработки клиентской части системы, обеспечивая динамический вывод данных на веб-страницы;
4. Axios: использовался для выполнения HTTP-запросов с клиента к серверу, обмена данными между клиентом и сервером;
5. SQL Server: использовался в качестве базы данных для хранения информации о клиентах, тренировках, подписках и других сущностях системы.

Использование этих технологий позволило создать информационно-поисковую систему "Онлайн фитнес", которая предоставляет возможность пользователям получать доступ к тренировкам, подписываться на различные программы, а также управлять своими данными в удобном и эффективном формате.

Основываясь на проведенном исследовании и разработке системы, можно сделать вывод о том, что информационно-поисковая система "Онлайн фитнес" является эффективным инструментом для организации тренировок и предоставления пользователю удобного доступа к фитнес-ресурсам и информации. Она упрощает и улучшает взаимодействие пользователей с фитнес-платформой, предоставляя широкие возможности по выбору тренировок, мониторингу прогресса и управлению подписками.

Использование современных технологий, таких как ASP.NET Core и React, позволяет создать гибкую и масштабируемую систему, способную

адаптироваться к изменяющимся потребностям пользователей и предоставлять им полноценный и удобный опыт использования.

## Список используемых источников

1) World Health Organization. Physical Activity. // who URL: https://www.who.int/news-room/fact-sheets/detail/physical-activity (дата обращения: 01.06.2023);

2) American Heart Association. Physical Activity Improves Quality of Life. // heart URL: https://www.heart.org/en/healthy-living/fitness/fitness-basics/why-is-physical-activity-so-important-for-health-and-wellbeing (дата обращения: 01.06.2023);

3) Chen, Y., Li, T., Zhou, G., Zhang, Z., & Liu, D. The development of online fitness platforms: From web2.0 to web3.0. Future Generation Computer Systems. - 2021. - 141 с;

4) Centers for Disease Control and Prevention. Physical Activity and Health. // cdc URL: https://www.cdc.gov/physicalactivity/basics/pa-health/index.htm (дата обращения: 01.06.2023);

5) Lee, I. M., Shiroma, E. J., Lobelo, F., Puska, P., Blair, S. N., & Katzmarzyk, P. T. Effect of physical inactivity on major non-communicable diseases worldwide: an analysis of burden of disease and life expectancy. - 2012. - 380 с;

6) Sillence, E., Briggs, P., Harris, P. R., & Fishwick, L. Going online for health advice: changes in usage and trust practices over the last five years. . - 2007. - 380 с;

7) American Council on Exercise. Fitness Basics // acefitness URL: https://www.acefitness.org/fitness-basics/ (дата обращения: 01.06.2023);

8) Mayo Clinic. Fitness training: Elements of a well-rounded routine. // mayoclinic URL: https://www.mayoclinic.org/healthy-lifestyle/fitness/in-depth/fitness-training/art-20044792 (дата обращения: 01.06.2023);

9) World Health Organization. Physical Activity. // who URL: https://www.who.int/news-room/fact-sheets/detail/physical-activity (дата обращения: 01.06.2023);

10) Centers for Disease Control and Prevention. Physical Activity and Mental Health. // cdc URL: https://www.cdc.gov/physicalactivity/basics/pa-health/index.htm (дата обращения: 01.06.2023);

11) Thompson, W. R. (Ed.). Worldwide survey of fitness trends for 2011. - 2010. - 310 с;

12) American Heart Association. Types of Physical Activity // heart URL: https://www.heart.org/en/healthy-living/fitness/fitness-basics/types-of-physical-activity (дата обращения: 01.06.2023);

13) Sillence, E., Briggs, P., Harris, P. R., & Fishwick, L. Going online for health advice: changes in usage and trust practices over the last five years.. - 2007. - 560 с;

14) Ding, D., Lawson, K. D., Kolbe-Alexander, T. L., Finkelstein, E. A., Katzmarzyk, P. T., van Mechelen, W., & Pratt, M. The economic burden of physical inactivity: a global analysis of major non-communicable diseases. - 2016. - 311 с;

15) Martin, R. C. The clean architecture. - 2003. - 746 с;

16) Reenskaug, T. Thing-Model-View-Editor: An Example from a Minsky Frame Editor. - 1979. - 450 с;

17) Connolly, T., & Begg, C. Database Systems: A Practical Approach to Design, Implementation, and Management. - 2014. - 543 с;

18) Ambler, S. W. Database Access Layer. - 2006. - 349 с;

19) Understanding the Data Access Layer (DAL) Design Pattern // codeproject URL: https://www.codeproject.com/Articles/10072/Understanding-the-Data-Access-Layer-Design-Patter (дата обращения: 01.06.2023);

20) Understanding ASP.NET MVC // codeproject URL: https://www.codeproject.com/Articles/25057/Understanding-ASP-NET-MVC-Part-1 (дата обращения: 01.06.2023);

21) ASP.NET Web API // dotnet.microsoft URL: https://dotnet.microsoft.com/apps/aspnet/apis (дата обращения: 01.06.2023);

22) Data Annotations in Entity Framework Core // docs.microsoft URL: https://docs.microsoft.com/en-us/ef/core/modeling/index#data-annotations (дата обращения: 01.06.2023);

23) Freeman, E., & Freeman, E. Head First Design Patterns. O'Reilly Media. - 2004. - 452 с;

24) Fowler, M. Patterns of Enterprise Application Architecture. Addison-Wesley Professional.. - 2003. - 743 с;

25) MDN Web Docs - Fetch API // developer.mozilla URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (дата обращения: 01.06.2023);

26) Axios - Promise based HTTP client // axios-http URL: https://axios-http.com/ (дата обращения: 01.06.2023);

27) Single-page application (SPA) // wikipedia URL: https://en.wikipedia.org/wiki/Single-page_application (дата обращения: 01.06.2023);

28) Введение в react hooks // hexlet.io URL: https://ru.hexlet.io/courses/js-react-hooks/lessons/intro/theory_unit (дата обращения: 01.06.2023).

29) React useEffect Hook // legacy.reactjs URL:https://legacy.reactjs.org/docs/hooks-effect.html (дата обращения: 01.06.2023);

30) React useState Hook // legacy.reactjs URL: https://legacy.reactjs.org/docs/hooks-state.html (дата обращения: 01.06.2023).