МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Информационных систем и технологий

Специальность 1-40 01 01 «Программное обеспечение информационных технологий»

Специализация 1-40 01 01 10 «Программное обеспечение информационных технологий (программирование интернет-приложений)»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту на тему:**

Web-приложение « Отслеживание прогресса в тренажерном зале »

Выполнил студент Валдайцев Александр Денисович

(Ф.И.О.)

Руководитель проекта ст. препод. Дубовик М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доц. Смелов В.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты ст. препод. Дубовик М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер ст. препод. Дубовик М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2023

**Содержание**

[**ВВЕДЕНИЕ** 3](#_Toc135368411)

[**1 Постановка задачи** 4](#_Toc135368412)

[**1.1. Анализ прототипов** 4](#_Toc135368413)

[**1.1.1. Fitness Online** 4](#_Toc135368414)

[**1.1.2. Daily Strength** 5](#_Toc135368415)

[**1.1.3. Pocket Trainer** 6](#_Toc135368416)

[**1.2. Требования к проекту** 7](#_Toc135368417)

[**2 Проектирование web-приложения** 8](#_Toc135368418)

[**2.1. Архитектура приложения** 8](#_Toc135368419)

[**2.2. UML-Диаграммы** 9](#_Toc135368420)

[**2.2.1. Диаграмма вариантов использования** 9](#_Toc135368421)

[**2.2.2. Диаграмма развертывания** 11](#_Toc135368422)

[**2.3. Проектирование базы данных** 12](#_Toc135368423)

[**2.4. Хранимые процедуры** 16](#_Toc135368424)

[**2.5. Функции** 16](#_Toc135368425)

[**2.6. Представления** 17](#_Toc135368426)

[**2.7. Технологии базы данных** 17](#_Toc135368427)

[**2.8. Спецификация функциональных требований** 18](#_Toc135368428)

[**3 Обоснование технических приемов программирования** 19](#_Toc135368429)

[**4 Тестирование, экспериментальные исследования и анализ полученных результатов** 20](#_Toc135368430)

[**4.1. Ошибка при регистрации существующего логина** 20](#_Toc135368431)

[**4.2. Ошибка при некорректных данных для входа** 20](#_Toc135368432)

[**4.3. Ошибка при поиске несуществующей услуги** 21](#_Toc135368433)

[**Заключение** 22](#_Toc135368434)

[**Список литературы** 23](#_Toc135368435)

[**ПРИЛОЖЕНИЕ А** 24](#_Toc135368436)

## **ВВЕДЕНИЕ**

В современном мире здоровый образ жизни и физическая активность играют все более важную роль, и тренажерные залы становятся популярным местом для достижения фитнес-целей и поддержания хорошей физической формы. Однако, для многих людей сложно отслеживать свой прогресс и оценить достижения, особенно когда речь идет о долгосрочных тренировках и множестве разнообразных упражнений. В рамках данного курсового проекта было принято решение разработать приложение, предназначенное для удобного отслеживания и контроля прогресса в тренажерном зале.

Целью создания данного проекта является облегчение процесса отслеживания и оценки физической активности пользователей в тренажерном зале. Приложение будет предоставлять возможность составлять собственные программы тренировок, запускать тренировку в режиме реального времени и отслеживать прогресс выполнения упражнений.

Новизна данного проекта заключается в комбинации удобства использования и точности отслеживания прогресса в тренажерном зале. В отличие от традиционных методов, таких как использование записных книжек или таблиц, данное приложение предоставляет пользователю удобный и интуитивно понятный интерфейс для создания тренировок и отслеживания их выполнения.

Актуальность и необходимость создания данного проекта обусловлены несколькими факторами. Во-первых, современные люди все больше осознают важность здоровья и физической активности, и тренажерные залы становятся для них естественным местом для занятий. Однако многие из них испытывают сложности в отслеживании своего прогресса и мотивации к достижению фитнес-целей. Во-вторых, с развитием технологий web-приложения становятся незаменимым инструментом для повседневной жизни. Поэтому разработка приложения для отслеживания прогресса в тренажерном зале будет актуальным и полезным решением для широкой аудитории.

В основной части будут затронуты все аспекты разработки проекта и обоснованы некоторые технические приёмы, к которым приходилось прибегнуть, с целью реализации задачи.

Основные цели и задачи приложения:

* поддержка ролей администратора и пользователя;
* составление программы тренировок;
* отслеживание прогресса в упражнениях и тренировках;
* измерение времени выполнения тренировки;
* обновление данных аккаунта.

В пояснительной записке представлена краткая информацию о похожих продуктах, архитектуре, реализации проекта, руководстве пользователя.

## **Постановка задачи**

Основной задачей курсового проекта является создание Web-приложения для отслеживания прогресса в тренажерном зале. Для этого необходимо выполнить следующие задачи:

* возможность аутентификации и авторизации пользователей;
* разделение ролей администратора и пользователя;
* организация безопасного соединения посредством SSL;
* хранение данных в базе данных;
* добавление, удаление и редактирование администратором упражнений;
* возможность просмотра и обновления персональных данных.

## **Анализ прототипов**

Были проанализированы цели и задачи, поставленные в данном курсовом проекте, а также рассмотрены аналогичные примеры их решений. На основании анализа всех достоинств и недостатков данных альтернативных решений были сформулированы требования к данному программному средству.

# **Fitness Online**

Одно из самых популярных альтернативных решений – программное средство «Fitness Online – Жизнь в новом теле». Fitness Online – виртуальная энциклопедия с интерактивной статистикой, тренировочными программами и подробными инструкциями для новичков, рискнувших заглянуть в тренажерный зал. Рекомендации разработчики выдают последовательно: сначала погружают спортсменов в основы биомеханики и физиологии, а после рассказывают о профилактике травм и работе различных мышечных групп.

Инструкции представлены и в текстовом виде, и в чуть более наглядном – в формате анимированных заставок и видеороликов. Дополнительно предусмотрен разбор заданий на местном форуме, а заодно и в чатах с реальными фитнес-инструкторами.

Преимуществами являются встроенный дневник для записи результатов тренировок и фиксации рекордов, а также специальная энциклопедия, включающая 550+ упражнений с инструкциями, наряду с возможностью связи с тренерами в чате. Также немаловажными преимуществами являются наличие раздела с меню и диетами для начинающих и выступающих спортсменов и подробная статистика с анализом физической активности и рекомендациями.

Из недостатков программного средства можно выделить то, что изолирующие упражнения находятся в приоритете перед базовыми, а тренировочные программы для новичков слишком простые. Также редкие технические неполадки способны прервать тренировочные сессии.

Интерфейс интернет-ресурса представлен на рисунке 1.1.

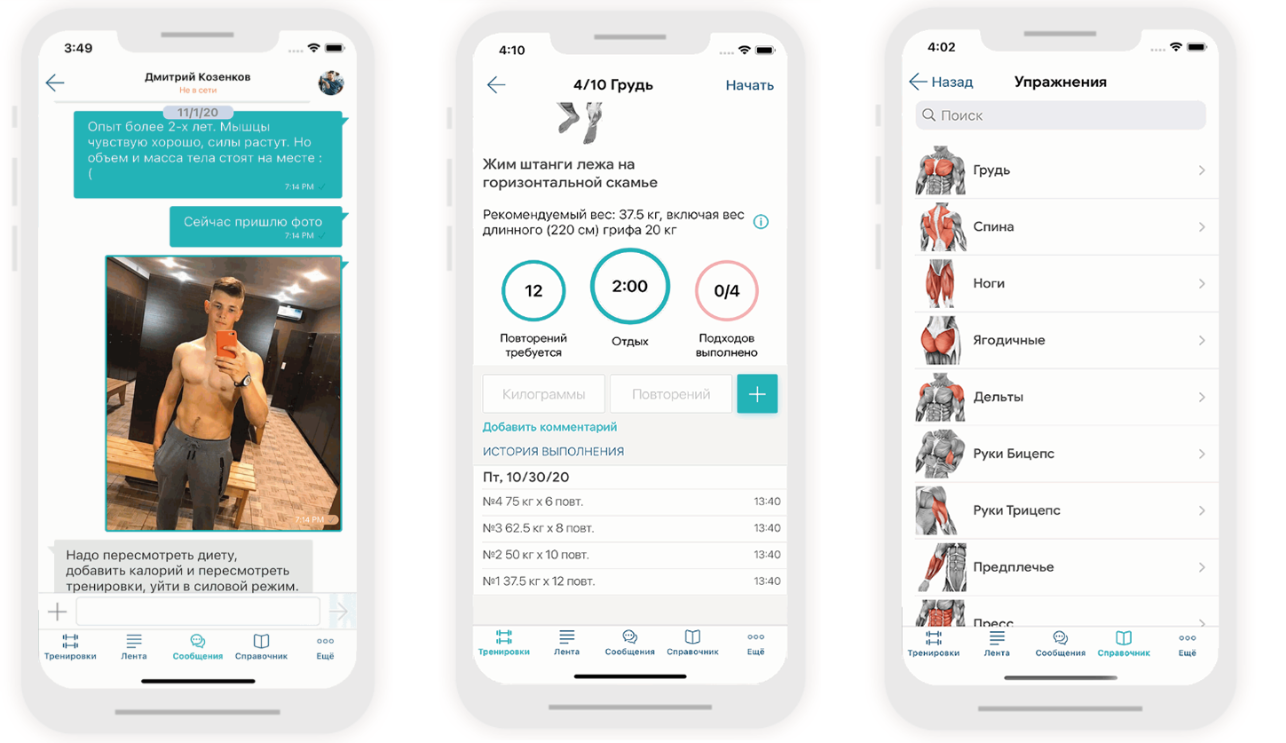


Рисунок 1.1 – Программное средство «Fitness Online»

# **Daily Strength**

Еще одним альтернативным решением задач, поставленных в данном курсовом проекте, является программное средство «Daily Strength» – электронный дневник для бодибилдеров и пауэрлифтеров с развернутой и интерактивной статистикой, эффективными тренировочными программами и подробными инструкциями для новичков и профессионалов.

Тренироваться разработчики предлагают и в функциональном стиле (кроссфит), и по классической силовой методике, последовательно нагружая руки, ноги, плечи, трицепс и бицепс, грудь и спину. Сразу выбирать конкретный подход необязательно – никто не мешает экспериментировать и комбинировать идеи.

Преимуществами данного аналога являются подробные программы для новичков и профессионалов с плавающим графиком занятий, встроенная библиотека из более чем трёхсот упражнений с видео-инструкциями, а также Подробная статистика с анализом результатов и трекером рекордов.

Главным недостатком является техническое несовершенство программного средства: присутствуют периодические проблемы с переводом интерфейса и инструкций на русский язык, некорректно работает экспорт и импорт настроек и статистики. Также таймер работает с ограничениями – нельзя добавить отдых или время разминки.

Интерфейс данного программного средства представлен на рисунке 1.2.

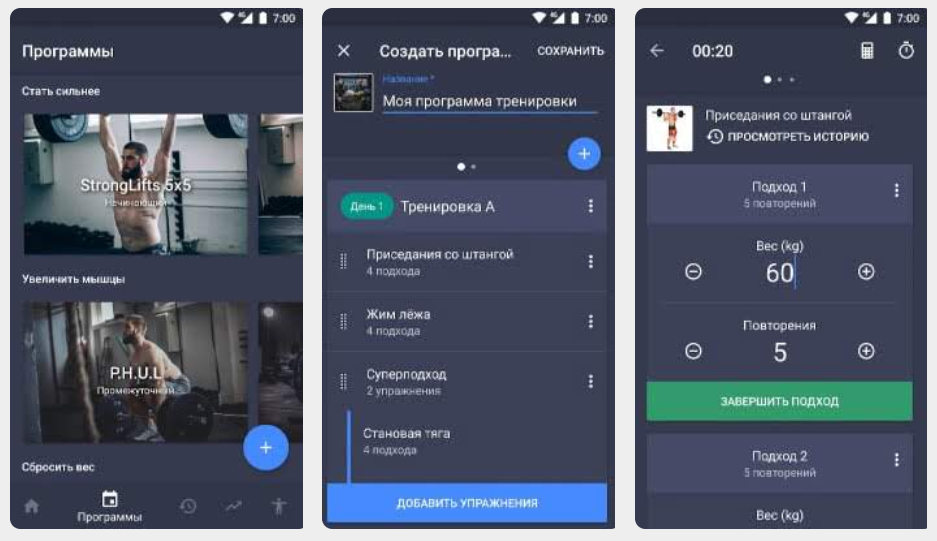


Рисунок 1.2 – Программное средство «Daily Strength»

# **Pocket Trainer**

Ещё одной альтернативой решения поставленных задач является программное средство «Pocket Trainer». У данного прототипа присутствует подробный анализ тренировочных сессий (за неделю, месяц или полгода) и специальные подсказки для новичков, такие как визуальные инструкции и текстовые рекомендации. Главной особенность данного аналога является наличие виртуального тренера. Специальный инструктор мотивирует экспериментировать, защищает от спортивной стагнации и травм, а заодно подсказывает, как разнообразить тренировочный процесс. Также виртуальный тренер не засчитывает пропущенные подходы и не дает поблажек по выходным.

Основными преимуществами данного программного средства являются: наличие сотен испытаний настраиваемой сложности для новичков и профессионалов, встроенный дневник со статистикой, недавними занятиями и анализом результатов, конфигуратор упражнений для экспериментов, таймер и секундомер для проведения занятий на выносливость, а также синхронизация настроек и статистики со сторонними платформами и сервисами.

Из недостатков можно выделить тот факт, что приложение на данный момент прекратило поддержку, также недоступны некоторые тренировочные форматы, такие как суперсеты или круговые сессии. Также немаловажным недостатком является то, что встроенное радио автоматически прерывает воспроизведение фоновой музыки.

Интерфейс интернет-ресурса представлен на рисунке 1.3.

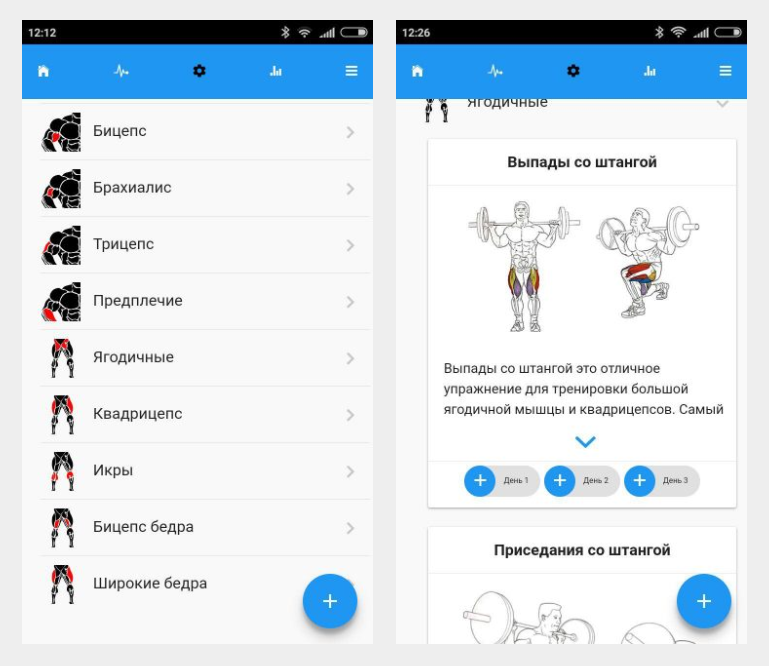


Рисунок 1.3 – Интернет-ресурс «BeatBeat.ru»

## **Требования к проекту**

Обзор вышеперечисленных известных аналогов позволяет проанализировать все преимущества и недостатки альтернативных возможностей и позволяет сформулировать список требований, предъявляемых к программному средству, разрабатываемому в данном курсовом проекте. Программное средство должно обеспечивать возможность выполнения перечисленных ниже функций:

* регистрация и авторизация пользователя;
* обеспечение безопасного соединения посредством SSL;
* наличие секундомера и подсчета общего времени тренировки;
* возможность просмотра и редактирования профиля;
* создание собственных программ тренировок;
* возможность просмотра завершённых тренировок.

## **Проектирование web-приложения**

# **Архитектура приложения**

Архитектура программного средства определяет структуру и организацию компонентов системы, их взаимодействие и распределение функций. Серверная часть проекта, реализованная с использованием фреймворка Nest.js и применяет трехуровневую архитектуру для организации компонентов. Эта архитектура разделяет приложение на три основных уровня: контроллеры, службы и репозитории.

* контроллеры (controllers) отвечают за обработку входящих запросов и отправку ответов клиентам. Они определяют конечные точки (endpoints) API и определяют логику обработки запросов. Контроллеры служат связующим звеном между запросами и сервисами приложения, они обрабатывают запросы и передают их соответствующим службам для выполнения бизнес-логики;
* поставщиками (providers) можно назвать большую часть стандартных классов – сервисы, репозитории, фабрики, хелперы и так далее. Главная идея поставщиков в том, что они могут быть внедрены с помощью механизма Dependency Injection. Основные поставщики – это службы (services). Службы содержат бизнес-логику приложения, обрабатывают полученные от контроллеров данные, выполняют операции и манипуляции с ними, могут взаимодействовать с репозиториями для доступа к данным или с другими службами для выполнения определенных задач, отвечают за выполнение основной функциональности приложения;
* модули (modules) служат для организации структуры и содержат в себе импорты и экспорты компонентов и поставщиков, которые должны находиться в области видимости данного модуля, либо экспортироваться из этого модуля и внедряться в другие модули приложения.

Трёхуровневая архитектура Nest.js помогает разделить логику приложения на отдельные компоненты, улучшает читаемость и поддерживаемость кода. Контроллеры обрабатывают внешние запросы, службы выполняют бизнес-логику, а модули образуют общую структуру приложения.

Для доступа к данным используется Docker в связке с базой данных PostgreSQL. Docker – это платформа для создания и управления контейнерами, которые содержат приложения и их зависимости. База данных PostgreSQL является реляционной системой управления базами данных.

Docker позволяет создавать изолированные контейнеры, которые могут содержать необходимые компоненты и настройки, включая базу данных. Это упрощает развертывание и масштабирование приложения, а также обеспечивает согласованность окружения при разработке и развертывании.

Клиентская часть проекта использует фреймворк React.js. React.js – это библиотека JavaScript для создания пользовательских интерфейсов. Он позволяет разработчикам создавать компоненты, которые могут обновляться эффективно и отзывчиво. React.js использует виртуальный DOM (Virtual DOM) для эффективного обновления только необходимых элементов интерфейса. Это улучшает производительность при работе с динамическими интерфейсами и упрощает разработку переиспользуемых компонентов.

Взаимодействие между сервером и клиентом осуществляется посредством Web API. Сервер предоставляет API (Application Programming Interface), который определяет доступные конечные точки и форматы данных, используемые для обмена информацией с клиентским приложением.

Клиентское приложение (React.js) отправляет HTTP-запросы на сервер, используя соответствующие конечные точки, и получает ответы, обычно в формате JSON. Это позволяет серверной и клиентской частям взаимодействовать и передавать данные между собой, обеспечивая функциональность и взаимодействие приложения.

# **UML-Диаграммы**

UML-диаграммы (Unified Modeling Language) представляют из себя графические инструменты, используемые для визуального представления и моделирования различных аспектов системы или программного средства. Они предоставляют универсальный набор символов, нотаций и правил для описания структуры, поведения и взаимодействия компонентов системы.

UML-диаграммы играют важную роль в разработке программного обеспечения, позволяя команде разработчиков и заинтересованным сторонам лучше понять и визуализировать архитектуру, функциональность, взаимодействие и динамику системы. Они помогают улучшить коммуникацию, согласование и совместное понимание в рамках проекта, а также служат важным инструментом для документирования, анализа и проектирования программных систем.

Далее будут более подробно рассмотрены диаграммы развёртывания и вариантов использования в рамках данного курсового проекта.

# **Диаграмма вариантов использования**

Диаграмма вариантов использования (Use case) – это тип UML-диаграммы, который моделирует функциональность системы с точки зрения ее пользователей (актёров). Она позволяет идентифицировать и описать различные варианты использования системы, то есть взаимодействие между актёрами и системой в рамках определенных сценариев. Диаграмма использования отображает функциональные возможности системы и позволяет лучше понять ее цели и требования.

На диаграмме вариантов использования актёры представляют роли, которые взаимодействуют с системой, а вариант использования описывает конкретное действие или сценарий, которым пользователь может воспользоваться для достижения определенной цели. Отношения между актёрами и вариантами использования показывают, какие действия могут выполнять актёры в рамках каждого варианта использования. Существует два типа отношений – «расширить» и «включить», которые обозначают соответственно необязательное и обязательное к выполнению действия.

Диаграмма вариантов использования изображена на рисунке 2.1.

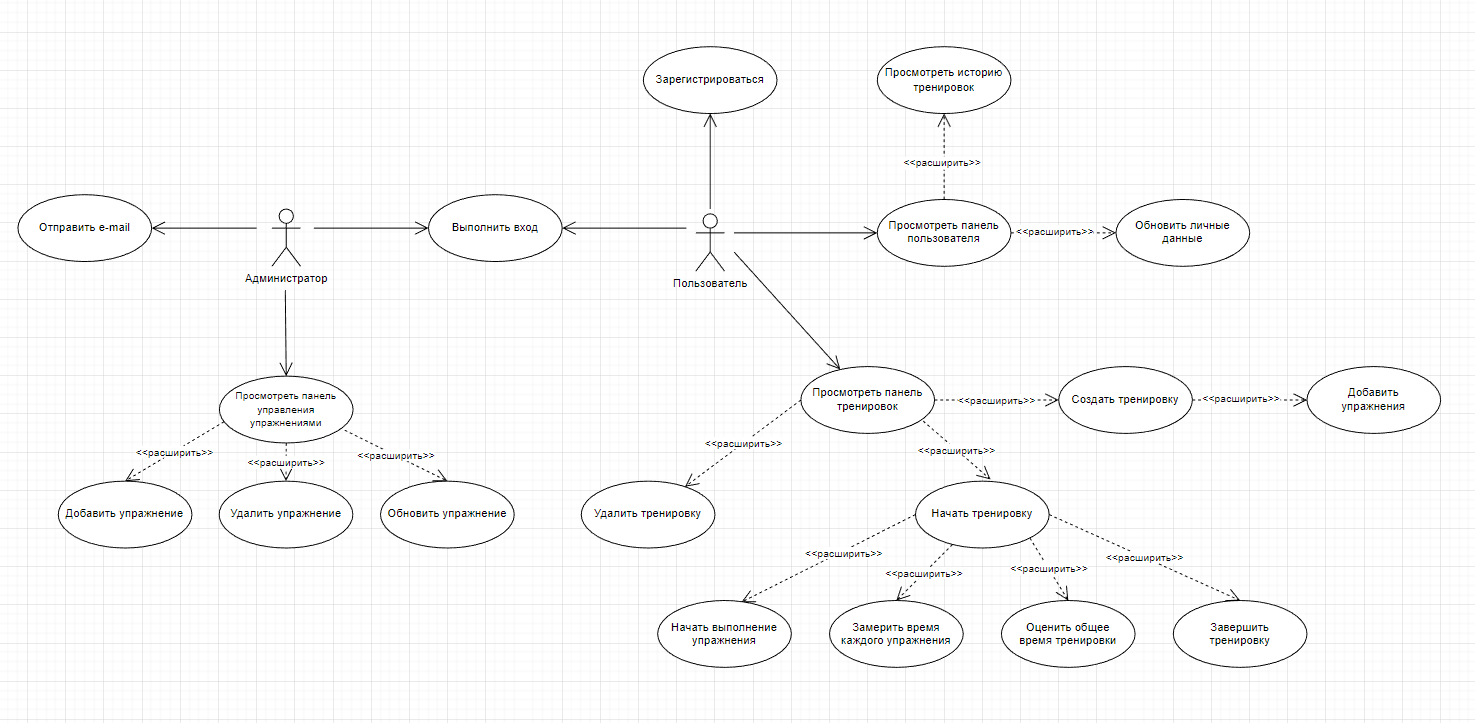


Рисунок 2.1 – Диаграмма вариантов использования

Исходя из данной диаграммы, видно, что в приложении существуют два актёра, то есть две роли: администратора и пользователя. Основными вариантами использования администратором являются отправка e-mail и просмотр панели администратора, включающая в себя панель управления упражнениями, что позволяет ему создавать, редактировать и удалять упражнения. В данном случае используются связь «расширить», так как данные действия не являются обязательными.

Актёр пользователя обладает более широким спектром вариантов использования, включающих в себя возможность просмотреть панель тренировок с дальнейшей возможность либо начать уже существующую, либо создать новую тренировку. При начале тренировки пользователь имеет возможность начать выполнение любого из упражнений в данной тренировки, и измерить время его выполнения. Также пользователь имеет вариант использования просмотра своего профиля, в котором отображается история тренировок, а также присутствует возможность обновления личных данных. Все связи вариантов использования актёра пользователя также имеют тип «расширить».

# **Диаграмма развертывания**

Диаграмма развертывания – это UML-диаграмма, которая моделирует физическое развертывание компонентов и архитектуры системы на аппаратном обеспечении. Она позволяет визуализировать, как компоненты и ресурсы системы распределены по физическим узлам (устройствам), таким как серверы, компьютеры, мобильные устройства и так далее.

На диаграмме развертывания показывается структура системы в терминах узлов и соединений между ними.

Компоненты системы представлены в виде узлов, а связи между узлами показывают, как компоненты взаимодействуют друг с другом и с внешними ресурсами (например, базами данных, сетями). Диаграмма развертывания позволяет лучше понять архитектуру системы, ее физическую конфигурацию и взаимодействие с окружающей средой.

Диаграмма развертывания web-приложения представлена на рисунке 2.2.

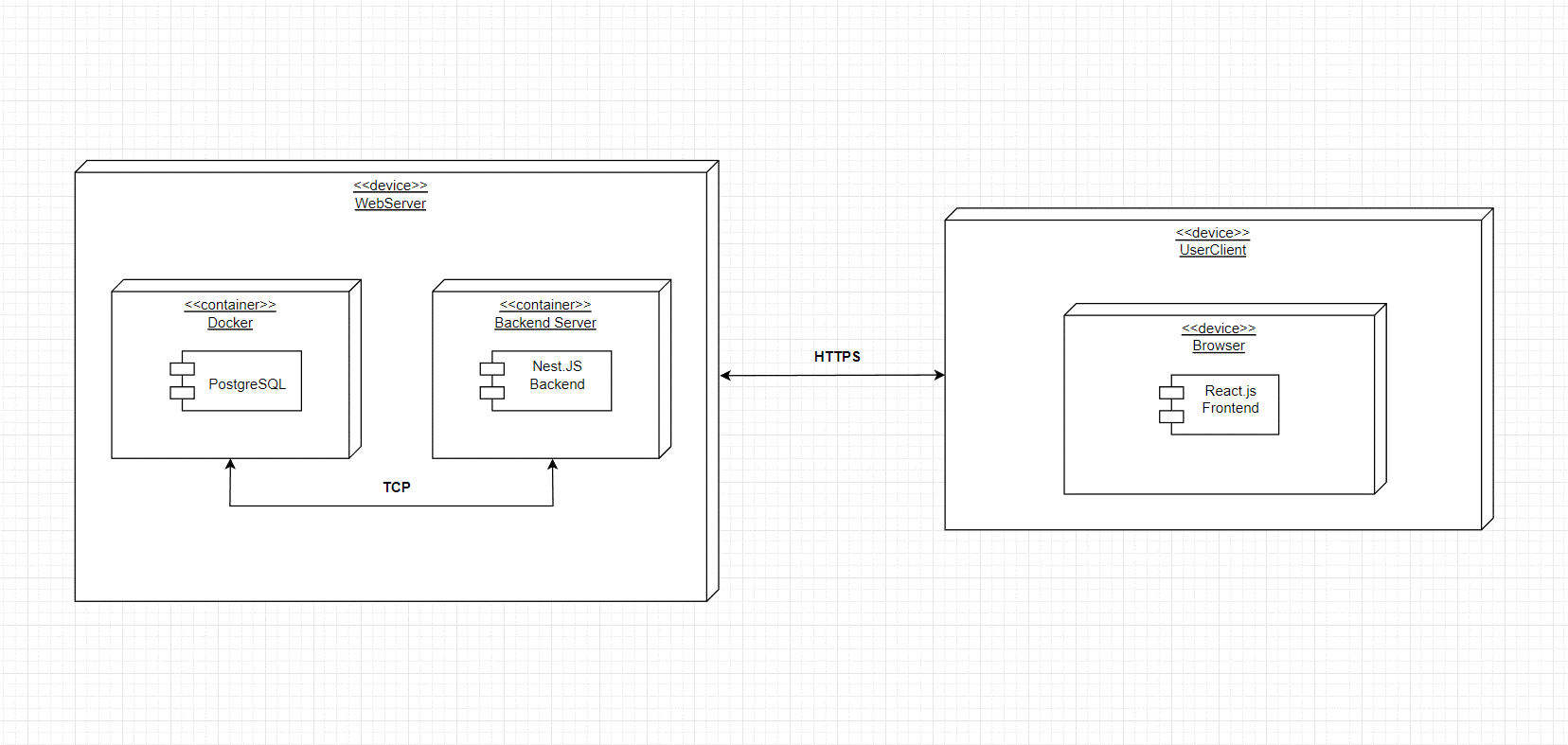


Рисунок 2.1 – Диаграмма развертывания

Как видно из данной диаграммы, сервер состоит из Backend-сервера на Nest.js, и Docker-контейнера, в котором находится реляционная база данных PostgreSQL. Клиент и сервер взаимодействуют между собой посредством HTTPS, обеспечивая безопасное соединение с помощью сертификата.

# **Описание базы данных**

Процесс проектирования базы данных состоит из следующих этапов:

* сбор информации;
* определение сущностей;
* определение атрибутов для каждой сущности;
* определение связей между сущностями;
* нормализация.

На этапе сбора информации необходимо точно определить, как будет использоваться база данных, и какая информация будет в ней храниться. Далее следует этап определения сущностей и на нем определяются сущности, из которых будет состоять база данных.

Сущность – это объект в базе данных, в котором хранятся данные. Сущность может представлять собой нечто вещественное (дом, человек, предмет, место) или абстрактное (банковская операция, отдел компании, маршрут автобуса). В физической модели сущность называется таблицей. Сущности состоят из атрибутов (столбцов таблицы) и записей (строк в таблице). Обычно базы данных состоят из нескольких основных сущностей, связанных с большим количеством подчиненных сущностей. Основные сущности называются независимыми: они не зависят ни от какой-либо другой сущности. Подчиненные сущности называются зависимыми: для того, чтобы существовала одна из них, должна существовать связанная с ней основная таблица.

Следующий этап – это определение атрибутов созданных сущностей. Атрибут представляет свойство, описывающее сущность. Атрибуты часто бывают числом, датой или текстом. Все данные, хранящиеся в атрибуте, должны иметь одинаковый тип и обладать одинаковыми свойствами.

В физической модели атрибуты называют колонками. После определения сущностей необходимо определить все атрибуты этих сущностей. На диаграммах атрибуты обычно перечисляются внутри прямоугольника сущности. На этапе определения связей работа происходит с ключами сущностей. Ключом называется набор атрибутов, однозначно определяющий запись. Ключи делятся на два класса: простые и составные.

Простой ключ состоит только из одного атрибута. Например, в базе «Паспорта граждан страны» номер паспорта будет простым ключом: ведь не бывает двух паспортов с одинаковым номером.

Составной ключ состоит из нескольких атрибутов. В той же базе «Паспорта граждан страны» может быть составной ключ со следующими атрибутами:  
фамилия, имя, отчество, дата рождения. Это – как пример, т. к. этот составной ключ, теоретически, не обеспечивает гарантированной уникальности записи.

Первичным ключом называется совокупность атрибутов, однозначно идентифицирующих запись в таблице (сущности). Один из возможных ключей становится первичным ключом. На диаграммах первичные ключи часто изображаются выше основного списка атрибутов или выделяются специальными символами.

Любой возможный ключ, не являющийся первичным, называется альтернативным ключом. Сущность может иметь несколько альтернативных ключей. Внешним ключом называется совокупность атрибутов, ссылающихся на первичный или альтернативный ключ другой сущности. Если внешний ключ не связан с первичной сущностью, то он может содержать только неопределенные значения. Если при этом ключ является составным, то все атрибуты внешнего ключа должны быть неопределенными. На диаграммах атрибуты, объединяемые во внешние ключи, обозначаются специальными символами.

Реляционные базы данных позволяют объединять информацию, принадлежащую разным сущностям. Отношение – это ситуация, при которой одна сущность ссылается на первичный ключ второй сущности. Они определяются в процессе проектирования базы. Для этого следует проанализировать сущности и выявить логические связи, существующие между ними. Тип отношения определяет количество записей сущности, связанных с записью другой сущности. Отношения делятся на три основных типа:

* один-к-одному: каждой записи первой сущности соответствует только одна запись из второй сущности. А каждой записи второй сущности соответствует только одна запись из первой сущности. Например, есть две сущности: Люди и Свидетельства о рождении. И у одного человека может быть только одно свидетельство о рождении.
* один-ко-многим: каждой записи первой сущности могут соответствовать несколько записей из второй сущности. Однако каждой записи второй сущности соответствует только одна запись из первой сущности. Например, есть две сущности: Заказ и Позиция заказа. И в одном заказе может быть много товаров.
* многие-ко-многим: каждой записи первой сущности могут соответствовать несколько записей из второй сущности. Однако и каждой записи второй сущности может соответствовать несколько записей из первой сущности. Например, есть две сущности: Автор и Книга. Один автор может написать много книг. Но у книги может быть несколько авторов.

Далее следует этап нормализации. Нормализацией называется процесс удаления избыточных данных из базы данных. Каждый элемент данных должен храниться в базе в одном и только в одном экземпляре. Существует пять распространенных форм нормализации. Как правило, база данных приводится к третьей нормальной форме.

В процессе нормализации выполняются определенные действия по удалению избыточных данных. Нормализация повышает быстродействие, ускоряет сортировку и построение индекса, уменьшает количество индексов на сущность, ускоряет операции вставки и обновления. Нормализованная база данных обычно отличается большей гибкостью. При модификации запросов или сохраняемых данных в нормализованную базу обычно приходится вносить меньше изменений, а внесение изменений имеет меньше последствий.

Чтобы преобразовать сущность в первую нормальную форму, следует исключить повторяющиеся группы значений и добиться того, чтобы каждый атрибут содержал только одно значение, списки значений не допускаются.  
Другими словами, каждый атрибут, в сущности, должен храниться только в одном экземпляре.

Для соответствия второй нормальной форме сущности должны быть в первой нормальной форме. Таблица во второй нормальной форме содержит только те данные, которые к ней относятся. Значения не ключевых атрибутов сущности зависят от первичного ключа. Если более точно, то атрибуты зависят от первичного ключа, от всего первичного ключа и только от первичного ключа.

В третьей нормальной форме исключаются атрибуты, не зависящие от всего ключа. Любая сущность, находящаяся в третьей нормальной форме, находится также и во второй. Это самая распространенная форма базы данных.  
В третьей нормальной форме каждый атрибут зависит от ключа, от всего ключа и ни от чего, кроме ключа.

Ограничения – это правила, за соблюдением которых следит система управления базы данных. Ограничения определяют множество значений, которые можно вводить в столбец или столбцы. Организовав данные в таблицы и определив связи между ними, можно считать, что была создана модель, правильным образом отражающая бизнес-среду. Теперь нужно обеспечить, чтобы данные, вводимые в базу, давали правильное представление о состоянии дела. Иными словами, нужно обеспечить выполнение деловых правил и поддержку целостности базы данных.

Например, ваша компания занимается доставкой книг. Вы вряд ли примете заказ от неизвестного клиента, ведь тогда вы даже не сможете доставить заказ. Отсюда бизнес-правило: заказы принимаются только от клиентов, информация о которых есть в базе данных. Корректность данных в реляционных базах обеспечивается набором правил. Правила целостности данных делятся на четыре категории.

Для реализации необходимого функционала была создана база данных, таблицы которой можно разделить на 3 логические группы: таблицы для хранения информации о продуктах, таблицы для хранения информации о заказах пользователей и таблицы для хранения пользовательских данных.

Рассмотрим таблицы базы данных на примере диаграммы, представленной на рисунке 2.3.

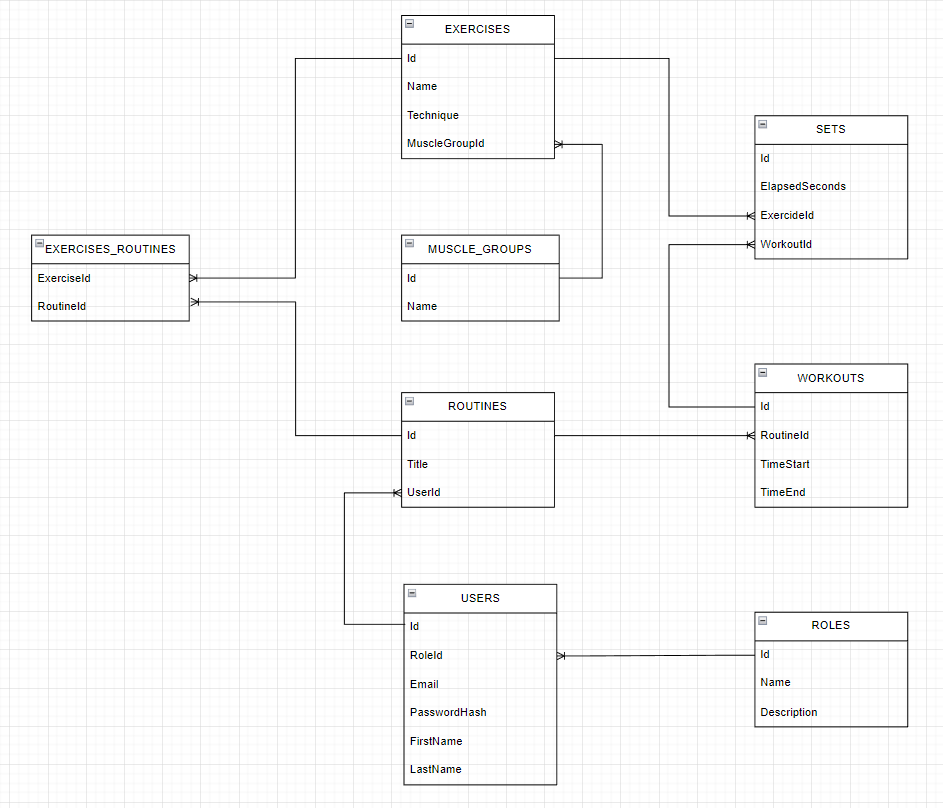


Рисунок 2.3 – Диаграмма базы данных

База данных состоит из восьми таблиц, связанных между собой внешними ключами, представленные в таблицах 2.1 – 2.7.

Таблица 2.1 – Роли

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание столбца |
| id | UUID | Уникальный идентификатор записи в таблице |
| name | VARCHAR | Название роли |
| description | TEXT | Описание роли |
| users | User[] | Связь с моделью User (множество пользователей с данной ролью) |

Таблица 2.2 – Группы мышц

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | UUID | Уникальный идентификатор записи в таблице |
| name | VARCHAR | Название группы мышц |
| exercise | Exercise[] | Связь с моделью Exercise (множество упражнений) |

Таблица 2.3 – Пользователи

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | UUID | Уникальный идентификатор записи в таблице |
| email | VARCHAR | Адрес электронной почты пользователя |
| password | VARCHAR | Хэшированный пароль пользователя |
| firstName | VARCHAR | Имя пользователя |
| lastName | VARCHAR | Фамилия пользователя |
| roleId | UUID | Идентификатор роли, связь с таблицей Role |
| routine | Routine[] | Связь с моделью Routine (множество рутин) |
| role | Role | Связь с моделью Role (роль пользователя) |

Таблица 2.4 – Упражнения

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | UUID | Уникальный идентификатор записи в таблице |
| name | VARCHAR | Название упражнения |
| technique | TEXT | Техника выполнения упражнения |
| muscleGroupId | UUID | Идентификатор группы мышц, связь с таблицей MuscleGroup |
| muscleGroup | MuscleGroup | Связь с моделью MuscleGroup (группа мышц) |
| routine | Routine[] | Связь с моделью Routine (множество рутин, в которых участвует упражнение) |
| set | Set[] | Связь с моделью Set (множество сетов упражнения) |

Таблица 2.5 – Тренировки

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | UUID | Уникальный идентификатор записи в таблице |
| title | VARCHAR | Название тренировки |
| userId | UUID | Идентификатор пользователя, связь с таблицей User |
| user | User | Связь с моделью User (пользователь тренировки) |
| workout | Workout[] | Связь с моделью Workout (множество воркаутов) |
| exercise | Exercise[] | Связь с моделью Exercise (множество упражнений) |

Таблица 2.6 – Воркауты

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | UUID | Уникальный идентификатор записи в таблице |
| routineId | UUID | Идентификатор тренировки, связь с таблицей Routine |
| timeStart | TIMESTAMPTZ | Время начала воркаута |
| timeEnd | TIMESTAMPTZ | Время окончания воркаута |
| set | Set[] | Связь с моделью Set (множество подходов воркаута) |
| routine | Routine | Связь с моделью Routine (тренировка воркаута) |

Таблица 2.7 – Подходы

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | UUID | Уникальный идентификатор записи в таблице |
| elapsedSeconds | INTEGER | Продолжительность выполнения подхода в секундах |
| exerciseId | UUID | Идентификатор упражнения, связь с таблицей Exercise |
| workoutId | UUID | Идентификатор воркаута, связь с таблицей Workout |
| exercise | Exercise | Связь с моделью Exercise (упражнение подхода) |
| workout | Workout | Связь с моделью Workout (воркаут подхода) |

## **Разработка Web-приложения**

Разработка веб-приложений обычно включает две основные части: backend (серверная часть) и frontend (клиентская часть). Backend и frontend могут разрабатываться практически независимо друг от друга, особенно при использовании подхода разработки Web API.

В идеальном случае проект веб-приложения следует разрабатывать в формате, который позволяет параллельную разработку backend и frontend. Обычно этот подход называется разделение ответственности. Согласно этому подходу, backend отвечает за обработку бизнес-логики, обработку запросов и взаимодействие с базой данных. Frontend, с другой стороны, отвечает за пользовательский интерфейс, визуализацию данных и взаимодействие с backend через Web API.

Один из распространенных подходов к разработке таких проектов – разработка backend API в первую очередь. Backend разрабатывается и тестируется независимо от клиентской части. Он предоставляет Web API с задокументированными конечными точками, форматами данных и правилами взаимодействия. Frontend-разработчики могут затем использовать эту документацию, чтобы создать клиентскую часть, взаимодействующую с API.

Frontend взаимодействует с API посредством HTTP-запросов и ответов. Фреймворки и библиотеки, такие как React.js, предоставляют инструменты для создания клиентской части приложения, которая может отправлять асинхронные запросы на сервер, обрабатывать полученные данные и обновлять пользовательский интерфейс. Взаимодействие между фронтендом и API может осуществляться с помощью Fetch, Axios или других инструментов для работы с HTTP-запросами.

# **Разработка серверной части**

При разработке серверной части Web-приложения вначале были созданы модули для каждой таблицы в базе данных. Были разработаны модели, которые определяют структуру данных и их взаимосвязи. Затем были реализованы основные CRUD-операции для работы с каждой таблицей.

Далее было принято решение разработать более специфические запросы к базе данных, которые предположительно будут использоваться в клиентской части (frontend) разработке. Учитывая потребности клиентской части, были созданы методы, обеспечивающие нужную функциональность. Это включало запросы данных с определенными фильтрами или сортировкой, а также запросы, возвращающие агрегированные или связанные данные из нескольких таблиц.

Особое внимание уделялось разработке модуля аутентификации. Был отдельно создан модуль, отвечающий за процесс аутентификации пользователей. В рамках этого проекта было решено реализовать стратегию JSON Web Token (JWT), позволяющую создавать аутентификационные токены и обеспечивать безопасную и эффективную аутентификацию и авторизацию пользователей.

Для разработки стратегии JWT был разработан класс JwtStrategy, наследуемый от класса PassportStrategy и представленный на рисунке 3.1.

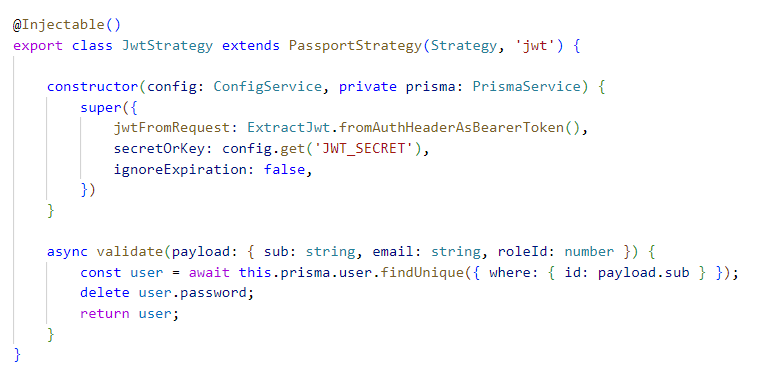


Рисунок 3.1 – Класс JwtStrategy

Также для реализации стратегии в сервисе аутентификации service.auth был создан метод signToken(), создающий JSON Web Token на основании идентификатора пользователя, идентификатора роли пользователя и электронной почты пользовтеля. Данная функция представлена на рисунке 3.2.



Рисунок 3.2 – Метод signToken()

Также для упрощения аутентификации и авторизации был создан декоратор GetUser(), который получает объект user из запроса request. Кастомный декоратор представлен на рисунке 3.3.



Рисунок 3.3 – Декоратор GetUser()

Наряду с этим, был разработан охранник для ролей RolesGuard, имплементирующий CanActivate, что позволяет использовать декоратор Roles() и передавать в него массив строковых значений ролей, тем самым охраняя определённые конечные точки от доступа пользователей, не имеющих данной роли. Данный guard представлен на рисунке 3.4.



Рисунок 3.4 – Класс RolesGuard

Также для доступа к данным используется ORM Prisma. Для получения дополнительного уровня абстракции был создан класс PrismaClient, помеченный декоратором Injectable(), который будет внедряться во все сервисы, которые нуждаются в обращении к базе данных через Prisma. Данный класс представлен на рисунке 3.5.

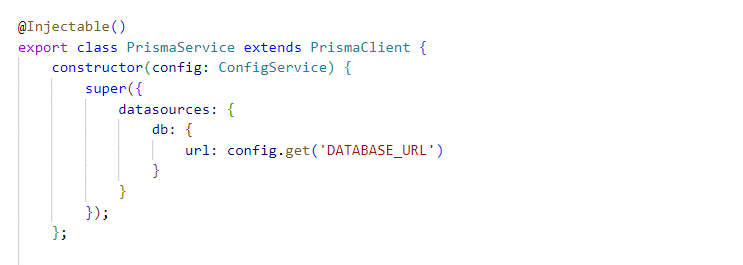


Рисунок 3.4 – Класс PrismaService

В качестве одного из примеров реализации CRUD операций для каждой таблицы рассмотрим модуль Routine, а конкретно routine.controller, в котором содержатся основные эндпоинты, соотвествующие RESTful API. Данный контроллер представлен на рисунке 3.5.

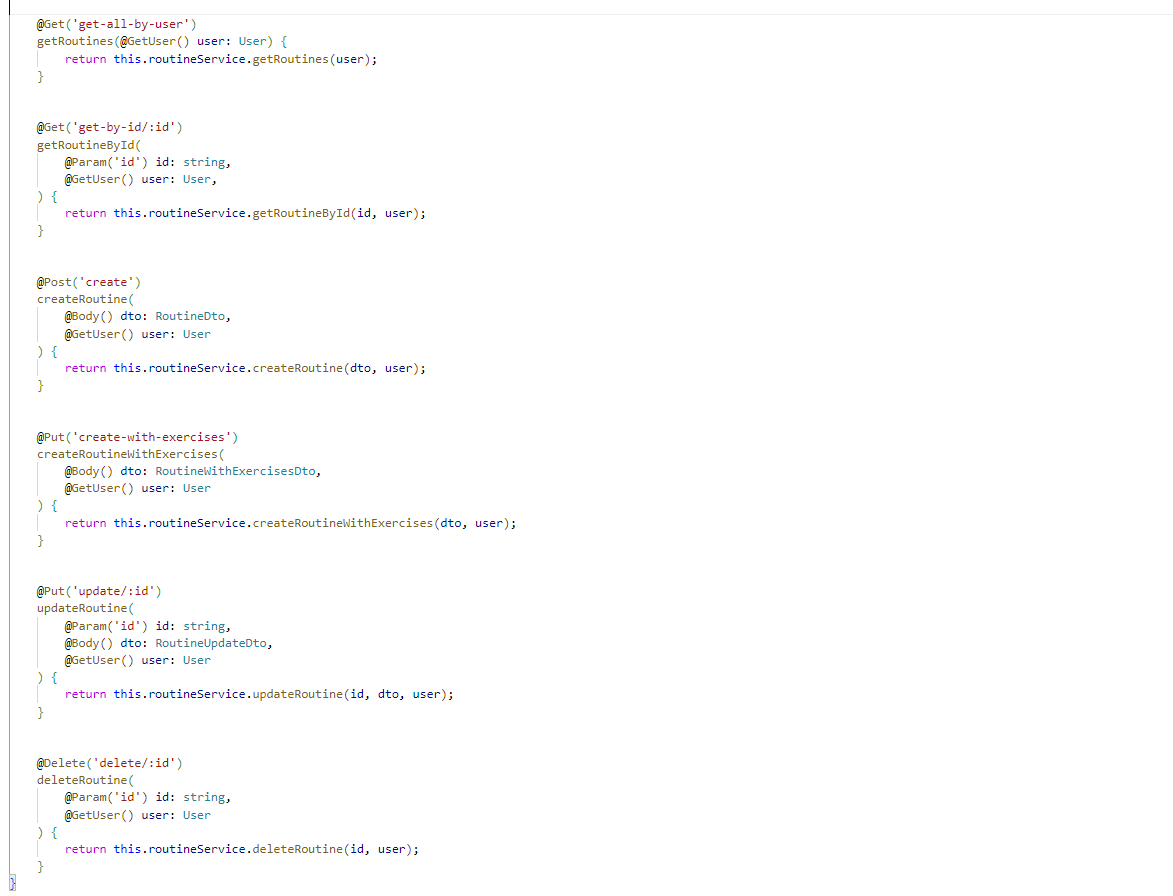


Рисунок 3.5 – Контроллер Routine

Также рассмотрим сервис, соответствующий данному контроллеру, содержащий бизнес-логику и представленный на рисунке 3.6.



Рисунок 3.6 – Сервис Routine

# **Разработка клиентской части**

При разработке клиентской части веб-приложения на React, используются ключевые концепции, такие как состояния (states), компоненты и React Router.

Состояния (states) в React представляют собой данные, которые определяют текущее состояние компонента. Компоненты React могут иметь свои собственные состояния, которые могут изменяться во время выполнения приложения. Изменение состояний обычно приводит к перерисовке компонента и обновлению пользовательского интерфейса.

Компоненты являются основными строительными блоками в React. Они представляют из себя независимые и переиспользуемые части пользовательского интерфейса. Компоненты React могут быть классовыми или функциональными. Классовые компоненты наследуются от базового класса React.Component и имеют свое состояние и методы жизненного цикла, тогда как функциональные компоненты представляют собой простые функции, которые принимают входные параметры и возвращают JSX-элементы.

Для обеспечения аутентификации и создания страницы со входом пользователя был создан компонент SignInPage, представленный на рисунке 3.7.



Рисунок 3.7 – Компонент SignInPage

В основном файле проекта App.js прописана маршрутизация приложения с помощью библиотеки react-router. Маршруты, прописанные в данном классе App, представлены на рисунке 3.8.



Рисунок 3.8 – Компонент App

Для отображения панели администратора был разработан компонент AdminDashboard, включающий в себя компонент AdminExercises. Данный компонент отображает все упражнения в виде плиток с кнопками добавления, удаления и редактирования. Данный компонент представлен на рисунке 3.9.



Рисунок 3.9 – Компонент AdminExercises

Также в качестве общих компонентов реализована навигационная панель NavBar, принимающая в props объект user и представленная на рисунке 3.10.

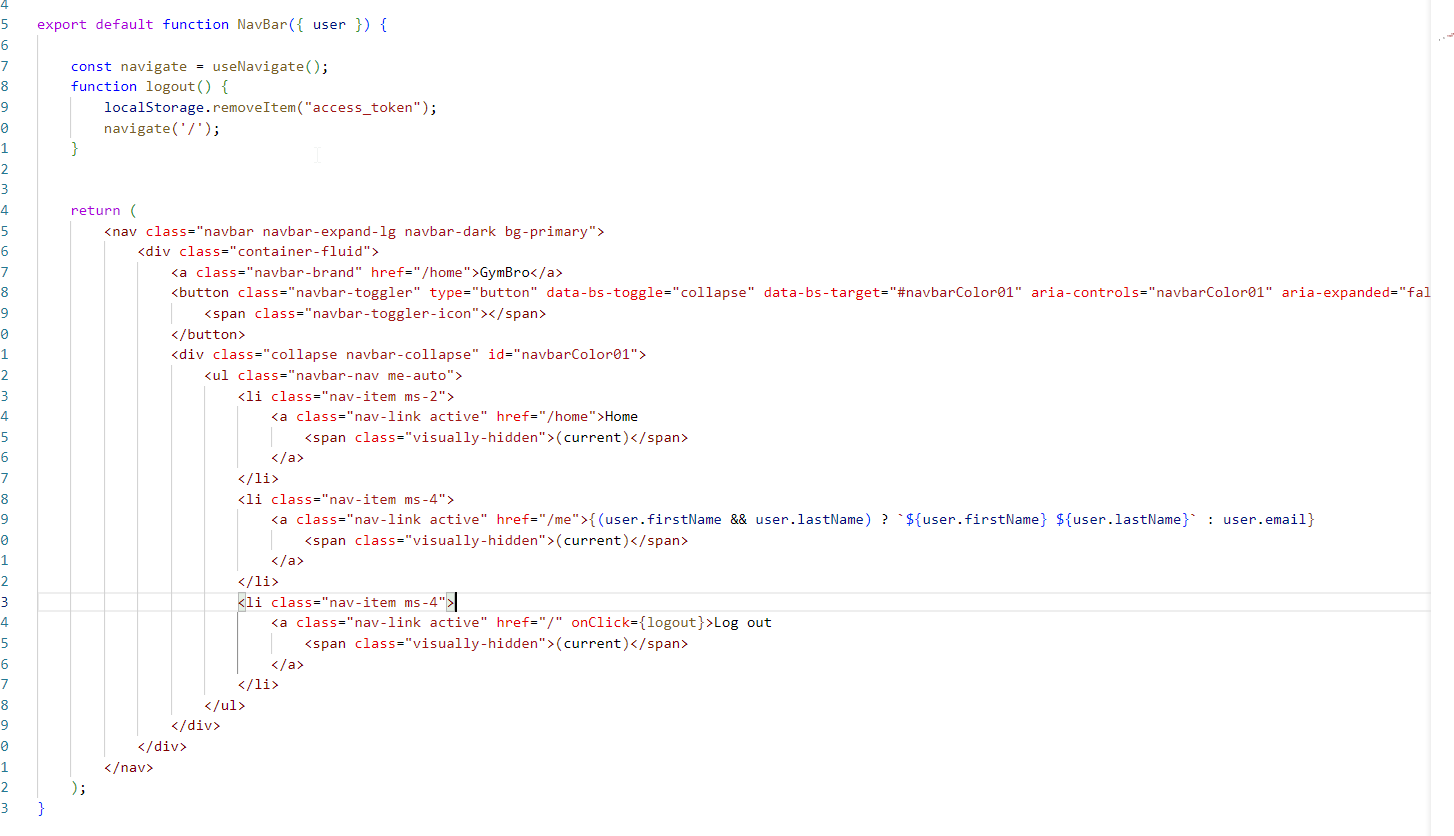


Рисунок 3.10 – Компонент NavBar

## **Тестирование Web-приложения**

Тестирование отыгрывает важную роль при разработке любого программного продукта. Чем качественнее тестирование, тем лучше в итоге должен выйти конечный продукт.

# **Ошибка при регистрации существующего логина**

При регистрации пользователя может возникнуть ситуация, при которой данная электронная почта уже существует в базе данных. Ошибка отображена на рисунке 4.1.

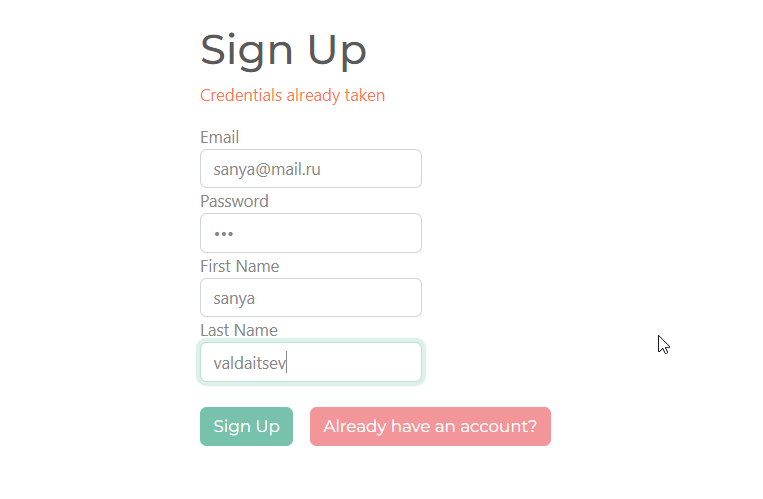


Рисунок 4.1 – Ошибка при регистрации существующего e-mail

# **Ошибка** **при некорректных данных для входа**

При предоставлении некорректных данных для входа (несуществующий логин или почта, неверный пароль) также должна возвращаться ошибка и сообщение. Ошибка отображена на рисунке 4.2.

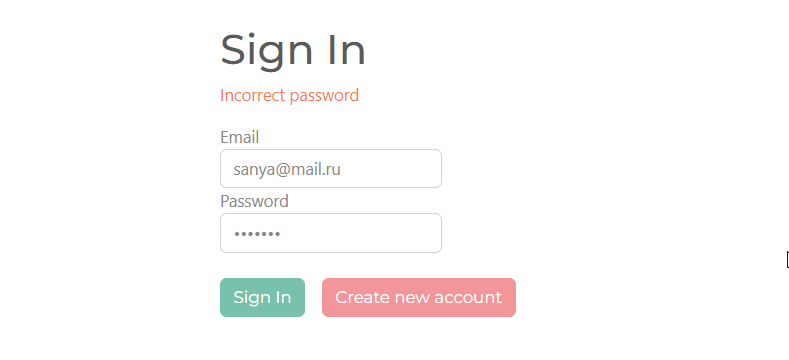


Рисунок 4.2 – Ошибка при некорректных данных для входа

# **Ошибка при пустом значении электронной почты**

При вводе пустых значений в форму входа или регистрации происходит валидация на frontend. Ошибка отображена на рисунке 4.3.

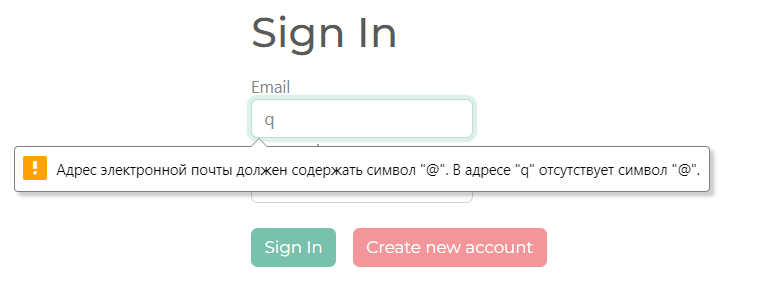


Рисунок 4.3 – Ошибка при пустых значения

# **Ошибка при создание тренировки с занятым именем**

Так как каждый пользователь может иметь некоторое количество тренировок, их названия должны быть уникальными. На рисунке 4.4 представлен текст ошибки в случае, когда было задано неуникальное название тренировки.

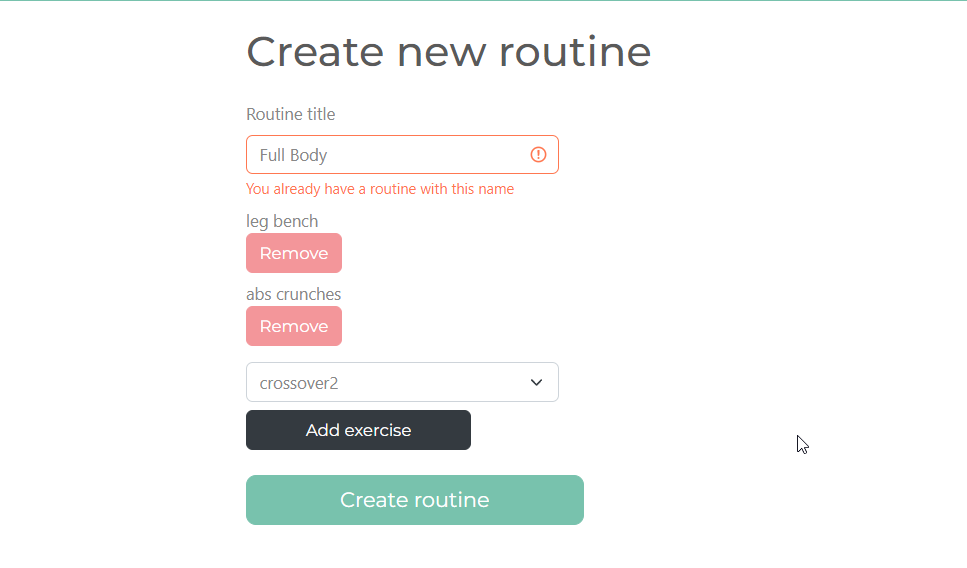


Рисунок 4.4 – Ошибка при задании занятого имени тренировки

# **Ошибка при вводе несуществующего маршрута**

Любое Web-приложение должно грамотно обрабатывать типичные ошибки пользователей, в том числе и ошибку 404 – Не найдено. Визуальное отображение в клиентском браузере ошибки при вводе несуществующего маршрута представлено на рисунке 4.5.

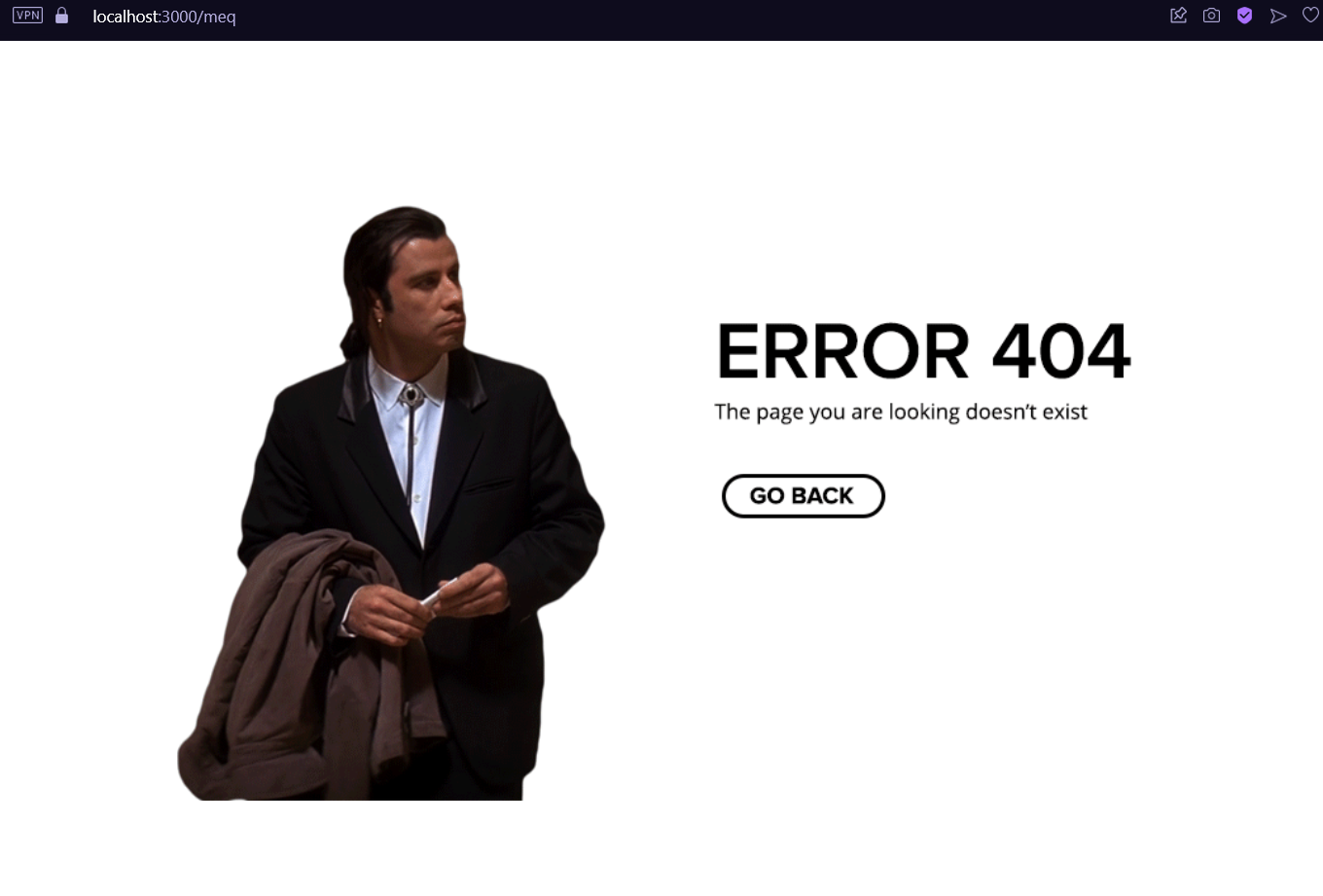


Рисунок 4.4 – Ошибка при вводе несуществующего маршрута

# **Заключение**

Результатом выполнения курсового проекта стала база данных в базе данные SQL Server в СУБД SQL Server Management Studio 18.

Были реализованы основные требования, указанные в листе задания вместе со следующими пунктами:

* Регистрация и авторизация пользователя;
* Поиск услуги;
* Оформление заказа;
* Уведомление пользователя о заказе;
* Просмотр корзины;
* Просмотр статуса заказа;
* Импорт и экспорт данных таблицы в XML формат;
* Заполнение таблицы на 100 000 записей;
* Технология мультимедийного хранения данных.

Проект не может окончательно претендовать на полноценный коммерческий продукт, поскольку для демонстрации технологии и работы с базой данных были опущены некоторые атрибуты современных площадок, реализация которых не относится к основной теме курсового проекта (например, платежные системы).

Протестировав все компоненты приложения, можно прийти к заключению, что все основные требования выполнены и приложение работает исправно.

# **Список литературы**

1. METANIT.COM Сайт о программировании [Электронный ресурс] / Режим доступа: https://metanit.com – Дата доступа: 15.10.2021.

2. DBMS\_XMLDOM [Электронный ресурс] / Режим доступа: <https://docs.oracle.com/cd/B1930601/appdev.102/b14258/dxmldom.htm#i1076719> – Дата доступа: 20.10.2022.

3. Developing and Using Stored Procedures [Электронный ресурс] / Режим доступа: <https://docs.oracle.com/cd/B2835901/appdev.111/b28843/tdddgprocedures.htm> – Дата доступа: 26.10.2022.

4. Stackoverflow.com [Электронный ресурс] / Режим доступа: https://stackoverflow.com – Дата доступа: 08.12.2022

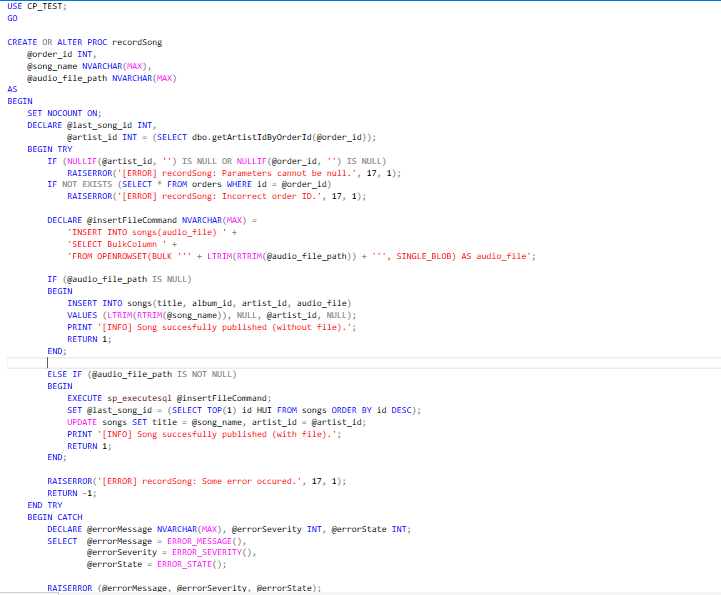
5. Блинова, Е.А. Курс лекций по Базам данным / Е.А. Блинова. – Минск: БГТУ, 2019. – 175 с.

# **ПРИЛОЖЕНИЕ А**

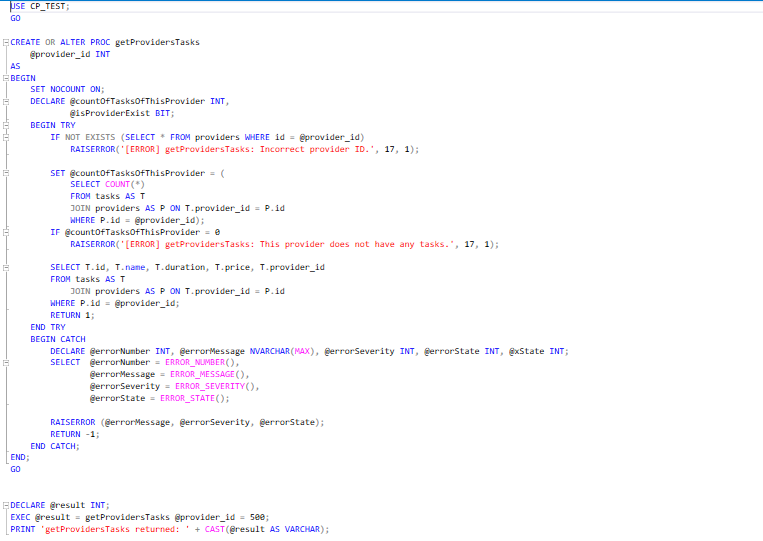
Процедура searchTask



Процедура recordSong



Процедура getProvidersTasks



Процедура payForAnorder

