

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

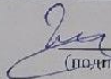
Профиль подготовки: «Вычислительные машины, комплексы, системы и сети»

БАКАЛАВРСКАЯ РАБОТА

на тему

ПРИЛОЖЕНИЕ ДЛЯ СОПРОВОЖДЕНИЯ
ТРЕНИРОВОК

Студент


(подпись, дата)

Гусев В. А.

Руководитель


(подпись, дата)

Зинкин С. А.

Нормоконтролёр


(подпись, дата)

Бычков А. С.

Работа допущена к защите (протокол заседания кафедры от 02.06.2023 г. № 14)

Заведующий кафедрой


(подпись)

М. А. Митрохин

(фамилия, инициалы)

Работа защищена с отметкой Отлично (протокол заседания ГЭК от

20.06.2023г., № 6)

Секретарь ГЭК



(подпись)

Е. И. Гурин

(фамилия, инициалы)

Пенза 2023

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«26» 04 2023 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Гусев Виталий Анатольевич гр.19BVB1 факультета вычислительной техники

2. Тема работы «Приложение для сопровождения тренировок»

Тема утверждена приказом ПГУ № 458/о от "26" апреля 2023г.

3. Руководитель бакалаврской работы д.т.н профессор кафедры ВТ ПГУ Зинкин С.А.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Необходимо спроектировать и разработать веб-приложение для сопровождения тренировок. Приложение должно предоставлять возможность для пользователя: добавлять тренировки и упражнения. Показывать историю предыдущих выполненных упражнений, а именно количество повторений и вес в подходе.

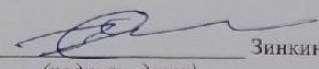
5. Перечень подлежащих разработке вопросов

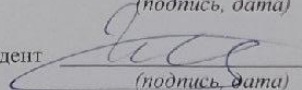
Требуется произвести анализ предметных областей: Необходимо изучить возможные альтернативы данному инструменту, другие площадки, на которых возможно базирование приложения, а также способы разработки веб-приложений как создание на базе *React + node.js*.

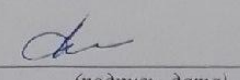
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	30	26.04.2023- 29.04.2023	
2.	Изучение конкурентов	20	30.04.2023- 01.05.2023	
3.	Анализ требований к сайту	25	02.05.2023- 04.05.2023	
4.	Выбор технологий разработки	30	05.05.2023- 08.05.2023	
5.	Проектирование и реализация базы данных	30	09.05.2023- 12.05.2023	
6.	Разработка приложения	135	13.05.2023- 27.05.2023	
7.	Тестирование приложения	30	28.05.2023- 29.05.2023	
8.	Оформление пояснительной записки	24	30.05.2023- 31.05.2023	

Дата выдачи задания "26" апреля 2023 г.

Руководитель бакалаврской работы  Зинкин С.А.
(подпись, дата)

Задание к исполнению принял студент  Гусев В.А.
(подпись, дата)

Работу к защите допустить
Декан факультета  Фионова Л.Р.
(подпись, дата)
20.06.2023

Реферат

Бакалаврская работа содержит 102 листов, 23 рисунков, 7 таблицы, 6 источников, 4 приложения

ВЕБ-ПРИЛОЖЕНИЕ, СОЗДАНИЕ ТРЕНИРОВОК, ДОБАВЛЕНИЕ ТРЕНИРОВОК, УПРАЖНЕНИЯ, БАЗА ДАННЫХ, *REACT*, *NODE.JS*, *POSTGRESQL*

Цель работы – разработать веб-приложение для сопровождения тренировок, позволяющий оптимизировать процесс сохранения результата тренировок, а также для более удобного занятия спортом.

Одним из путей достижения поставленной задачи является веб-приложение, обеспечивающий выполнение следующих функций: добавление тренировок, выполнение тренировок, создание упражнений, возможность сохранения и просмотра результатов выполнения предыдущих упражнений.

В конечном итоге разработано приложение, полностью соответствующий цели данной выпускной квалификационной работы. В ходе достижения результата, были использованы следующие технологии: *react*, *node.js*, *postgresql*. Приложение протестировано и готово к эксплуатации.

Изм.	Лист	№ докум.	Подпись	Дата
Разраб.		Гусев В.А.		
Провер.		Зинкин С.А.		
Н. Контр.		Бычков А.С.		
Утверд.		Митрохин М.А.		

ПГУ1.090301.6.001 ПЗ

Приложение для сопровождения тренировок. Пояснительная записка		Лит.	Лист	Листов
			2	105

ФВТ, каф. ВТ
гр. 19ВВВ1

Содержание

Перечень сокращений и обозначений.....	5
Введение.....	6
1 Приложение для сопровождения тренировок.	8
1.1 Разработка структуры веб-приложения.....	8
1.1.1 Анализ предметной области.....	8
1.1.2 Решение поставленной задачи.....	8
1.1.3 Анализ и структура сайта.....	9
1.1.4 Постановка задачи.....	14
1.1.5 Выбор инструментов разработки.....	14
1.1.6 Выбранные технологии.....	20
1.2 Реализация	23
1.2.1 Настройка окружения	23
1.2.2 Работа с БД	25
1.2.3 Описание доступных функций	30
1.2.4 Описание программы.....	39
2 Безопасность жизнедеятельности.....	43
2.1 Правила безопасности при использовании ПК.....	43
2.2 Правила безопасности при использовании смартфона.....	44
Заключение	49
Список использованных источников	51
Приложение А – <i>UML</i> -диаграммы	52
Приложение Б – Физическая модель базы данных.....	56

Приложение В – Листинг программы.....	58
Приложение Г - Презентация.....	94

					ПГУ1.090301.6.001 ПЗ	Лист
						4
Изм.	Лист	№ докум.	Подпись	Дата		

Перечень сокращений и обозначений

В настоящей работе применяют следующие обозначения и сокращения

БД – база данных

ПК- персональный компьютер

ПО – программное обеспечение

СУБД – система управления базами данных

ЯП – язык программирования

API (Application Programming Interface) – описание способов, которыми одна компьютерная программа может взаимодействовать с другой программой

ID (Identifier) – идентификатор объекта

IDE (Integrated development environment) – интегрированная среда разработки

SQL (Structured Query Language) – язык структурированных запросов

UML (Unified Modeling Language) – унифицированный язык моделирования

					ПГУ1.090301.6.001 ПЗ	Лист
						5
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

Использование веб-приложений в сфере тренировок становится все более популярным в настоящее время. Они предоставляют удобную платформу для тренировок и отслеживания результатов.

Множество компаний разных масштабов успешно внедряют веб-приложений в свою деятельность. От малых стартапов до крупных корпораций, все больше людей осознают преимущества использования онлайн-платформ для тренировок. Это позволяет им достичь хорошего результата и поддерживать физическую форму.

Веб-приложения для тренировок обладают несколькими преимуществами по сравнению с традиционными занятиями спортом. Во-первых, они доступны пользователям в любое время и из любой точки мира. Во-вторых, пользователи имеют возможность выбирать тренировку из списка исходя из обстоятельств. Кроме того, приложения могут предлагать персонализированные рекомендации, учитывая предпочтения и историю тренировок клиентов.

Создание и разработка веб-приложения для сопровождения тренировок требует учета различных аспектов, таких как дизайн, функциональность, безопасность и удобство использования для пользователей. Эффективное приложение должно обеспечивать простую навигацию, быструю загрузку страниц, удобную систему взаимодействия.

Таким образом, веб-приложения для тренировок становятся неотъемлемой частью современного тренинга. Они предоставляют удобную платформу для тренировок и отслеживания прогресса, привлекая все большее количество людей, желающих поддерживать свое здоровье.

Цели выпускной квалификационной работы:

– освоение актуальных и современных средств программной разработки;

					ПГУ1.090301.6.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

- изучение нюансов разработки веб-приложений;
- реализация веб-приложения для сопровождения тренировок.

Для достижения поставленных целей, необходимо обозначить следующие задачи, решаемые в ходе выполнения выпускной квалификационной работы:

- провести анализ протекания действующих процессов деятельности приложения для тренировок;
- поставить задачи;
- провести анализ существующих приложений и их квалификации;
- провести анализ структуры приложения;
- реализовать веб-приложение;
- протестировать приложение.

					ПГУ1.090301.6.001 ПЗ	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

1 Приложение для сопровождения тренировок.

1.1 Разработка структуры веб-приложения

1.1.1 Анализ предметной области

Перед началом разработки любого продукта в сфере IT, необходимо провести анализ для определения наилучшего пути достижения желаемого результата. Это включает изучение различных вариантов разработки, архитектурных решений и других факторов. В данном разделе мы рассмотрим ключевые теоретические аспекты, необходимые для создания веб-приложения.

Для начала необходимо проанализировать, какими путями в данный момент времени люди тренируются и хранят информацию о предыдущих тренировках.

Главным ресурсом служат такие вещи как: телефон с каким-либо приложением для записи (блокнот, заметки), бумажные носители (дневники тренировок). Информацию о тренировках можно найти в интернете, получить от сертифицированного тренера, найти в методических указаниях. К сожалению, бумажные носители не всегда удобно таскать с собой, они быстро приходят в негодность, информация в книгах со временем устаревает, поэтому куда удобнее использовать телефон в качестве источника для хранения и поиска информации.

1.1.2 Решение поставленной задачи

Веб-приложение – это современный вариант реализации тренировочного процесса, программный продукт, позволяющий пользователям сети интернет тренироваться и хранить результат в одном приложении, экономя время и ресурсы.

Их используют и крупные коммерческие фирмы, и на государственной службе, и на образовательном уровне. Яркими примерами можно назвать:

					ПГУ1.090301.6.001 ПЗ	Лист
						8
Изм.	Лист	№ докум.	Подпись	Дата		

- веб-приложение «GymUp». Дневник тренировок;
- веб-приложение «GymKeeper». Приложение для тренировок и хранения результата;
- веб-приложение «Тренировки в тренажерном зале». Приложение для тренировок в тренажерном зале;
- приложение «Карманный-тренер». Мобильное приложение для тренировок.

Главными плюсами веб-приложений являются:

- экономия времени. В большинстве случаев использование приложения для тренировок куда быстрее, чем занятие в тренажерном зале под руководством тренера;
- возможность анонимного использования приложения;
- приложение доступна в любой момент, круглосуточно;
- пользователь может вернуться к тренировкам в любой момент;
- максимально простой интерфейс. Пользователю не нужно рассмотреть все другие тренировки, ведь он может найти всё по удобным критериям или создать свою тренировку. Весь интерфейс — это структурированная сеть кнопок;
- экономия. Пользователи могут сэкономить на услугах тренера или занятий в тренажерном зале;
- возможность сбора данных о действиях пользователей.

1.1.3 Анализ и структура сайта

Приложение для сопровождения тренировок представляет из себя программный продукт, в котором пользователь может создавать упражнения и тренировки, отслеживать свой прогресс в них.

Архитектура приложения для сопровождения тренировок состоит из трех основных составляющих частей:

- клиентская часть;
- серверная часть;
- база данных.

Клиентская часть – графическая часть приложения, предназначенная для взаимодействия пользователя с программным продуктом. В традиционном веб-приложении клиентский код отвечает за взаимодействие с пользователем и визуализацию пользовательского интерфейса. Кроме того, клиентская часть сайта отвечает за проверку данных, вводимых пользователем. Необходимо это для того, чтобы серверная часть смогла получить верные данные и верно их обработать.

Серверная часть приложения обычно относится к коду и процессам, которые выполняются на сервере или в фоновой части приложения. Это может включать такие задачи, как управление аутентификацией пользователей и хранением данных, обработка и выполнение бизнес-логики, а также предоставление интерфейса для клиентов или интерфейсных приложений для запроса и получения данных или услуг. Серверная сторона отвечает за обработку запросов от клиентских устройств, обработку данных и возврат результатов на клиентское устройство. Это также включает обеспечение безопасности, управление ресурсами сервера и выполнение любого необходимого обслуживания или обновлений.

База данных является важнейшим компонентом многих приложений, поскольку она обеспечивает структурированный способ организации, хранения, извлечения данных и эффективного управления ими. Вот некоторые из основных причин, по которым базы данных необходимы для приложений:

- хранение и организация данных. Базы данных позволяют хранить данные в организованном порядке, упрощая доступ к ним и управление ими. Они используют структурированный формат, который позволяет

разработчикам устанавливать отношения между объектами данных и обеспечивать соблюдение правил целостности данных. Без базы данных данные должны были бы храниться в отдельных файлах или документах, что затрудняло бы управление ими и их извлечение;

- поиск данных. Базы данных позволяют пользователям извлекать данные по запросу, используя запросы, которые могут искать определенные данные на основе различных критериев. Это позволяет легко находить и извлекать нужные данные, а не просеивать большие объемы неорганизованной информации;

- безопасность данных. Базы данных предоставляют встроенные функции безопасности, которые могут защитить конфиденциальные данные от несанкционированного доступа и гарантировать, что только авторизованные пользователи могут получать доступ и изменять данные. Это может включать системы аутентификации пользователей, разрешения на управление доступом и методы шифрования;

- масштабируемость. Базы данных могут обрабатывать большие объемы данных и могут увеличиваться или уменьшаться по мере необходимости, что делает их подходящими для приложений, которым необходимо обрабатывать растущие объемы данных с течением времени;

- производительность. Базы данных оптимизированы для быстрого поиска данных и могут быть настроены для получения высокопроизводительных результатов даже при обработке больших объемов данных или сложных запросов.

Таким образом, базы данных являются важным компонентом многих приложений, поскольку они обеспечивают структурированный и эффективный способ хранения, управления и извлечения данных. Они предлагают ряд преимуществ, включая улучшенную организацию данных,

упрощенный поиск данных, повышенную безопасность данных, масштабируемость и высокую производительность.

Системы управления базами данных (СУБД) — это программные приложения, предназначенные для управления и организации данных. Основная цель СУБД — обеспечить надежный, эффективный и безопасный способ хранения и извлечения данных, а также управлять отношениями между различными элементами данных. Некоторые общие задачи, которые может выполнять СУБД, включают:

- создание и ведение баз данных;
- добавление, изменение и удаление данных в базе данных;
- обеспечение целостности и согласованности данных;
- обеспечение безопасности и контроля доступа к базе данных;
- управление одновременным доступом к базе данных несколькими пользователями;
- включение резервного копирования и восстановления данных в случае потери или повреждения данных;
- предоставление инструментов для запросов, отчетов и анализа данных.

СУБД широко используются в бизнесе, правительстве, образовании и других областях, где необходимо эффективно хранить и управлять большими объемами данных.

Структура взаимодействия компонентов представлена на рисунке 1.

Для четкого понимания и отображения всех возможностей сайта, необходимо создать *UML*-диаграмму вариантов использования. Она показана на рисунке 2.

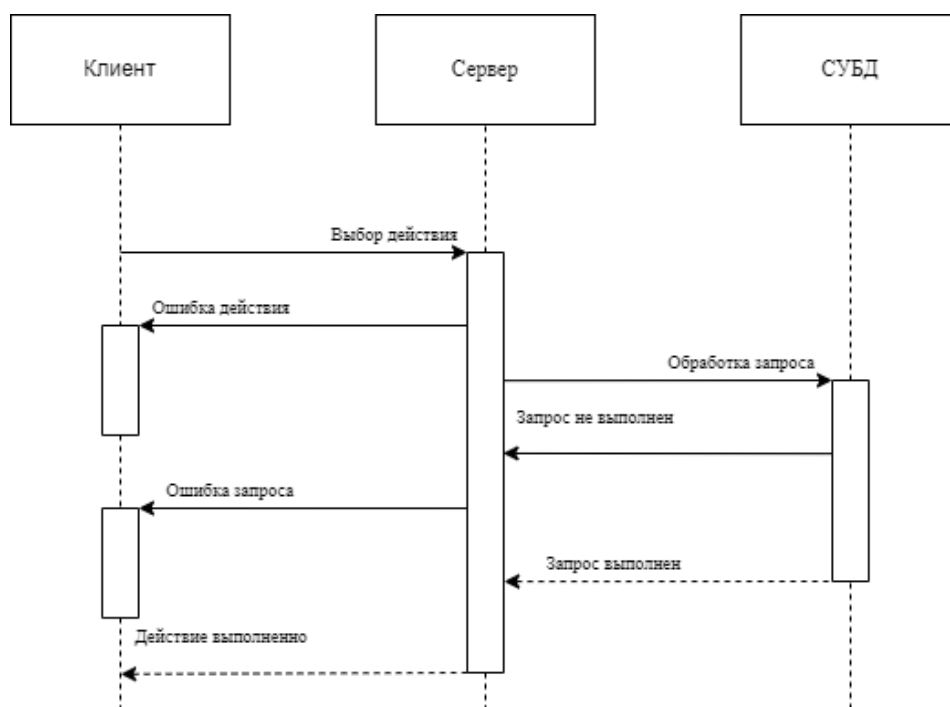


Рисунок 1 – UML-диаграмма последовательности

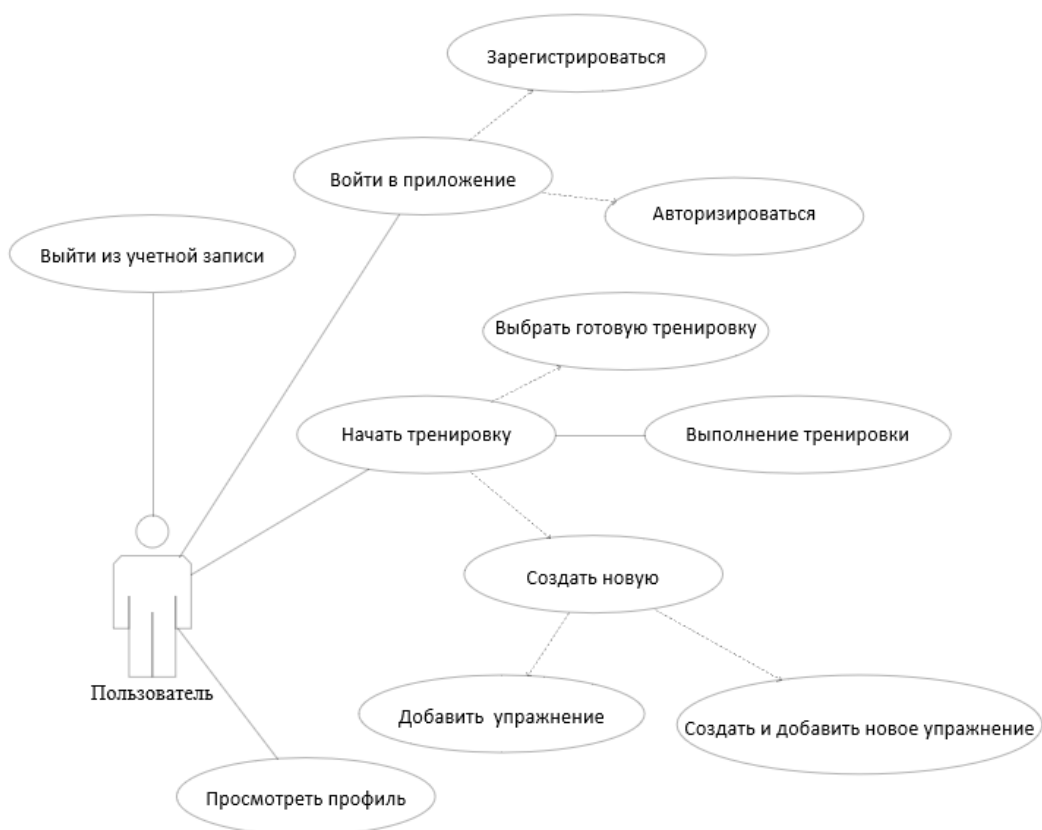


Рисунок 2 – UML-диаграмма вариантов использования

1.1.4 Постановка задачи

Необходимо спроектировать и разработать веб-приложение для сопровождения тренировок для более удобного занятия спортом. Приложение должно предоставлять возможность со стороны пользователя: регистрацию, просмотр профиля, выполнение тренировок, создание новых тренировок, добавления новых упражнений, просмотр истории выполненных упражнений.

1.1.5 Выбор инструментов разработки

1.1.5.1 Выбор платформы

При выборе платформы для создания веб-приложения следует учитывать несколько факторов, в том числе:

- язык программирования. Нужно выбрать платформу, которая поддерживает язык программирования;
- масштабируемость. Необходимо рассмотреть масштабируемость платформы, сможет ли она справиться с ростом вашего приложения с точки зрения трафика и данных;
- безопасность. Требуется выбрать платформу, которая предлагает надежные функции безопасности для защиты вашего приложения от киберугроз;
- интеграция. Важно выбрать платформу, которая может легко интегрироваться с другими инструментами и службами, которые вы планируете использовать;
- поддержка. Также важно убедиться, что у платформы есть активное сообщество и хорошие ресурсы поддержки, которые помогут в случае возникновения каких-либо проблем.

Список популярные платформы для создания веб-приложений:

- *ruby on Rails*;
- джанго (на основе *Python*);

- *node.js*;
- *laravel* (на основе *PHP*);
- *ASP.NET* (на базе *Microsoft*).

Есть несколько преимуществ использования *Node.js* по сравнению с другими платформами, в том числе:

- быстрый и масштабируемый: *Node.js* построен на движке *JavaScript V8*, известном своей скоростью и масштабируемостью. Это обеспечивает быстрое и эффективное выполнение кода, что делает *Node.js* идеальным для создания приложений и сервисов в реальном времени;
- неблокирующий ввод-вывод: *Node.js* использует управляемую событиями неблокирующую модель ввода-вывода, что означает, что он может обрабатывать несколько запросов одновременно, не блокируя другие запросы. Это делает его идеальным для создания масштабируемых приложений;
- большое сообщество: *Node.js* имеет большое и активное сообщество разработчиков, а это означает, что существует множество ресурсов, доступных для обучения и поддержки. Это также означает, что в ваших проектах доступно множество сторонних модулей и библиотек;
- кроссплатформенная поддержка: *Node.js* — это кроссплатформенная среда выполнения, что означает, что она может работать в нескольких операционных системах, включая *Windows*, *Mac OS* и *Linux*. Это упрощает разработку и развертывание приложений на разных платформах;
- простота в освоении: *Node.js* построен на *JavaScript*, который является широко используемым языком программирования. Это означает, что многие разработчики уже немного знакомы с *JavaScript*, что упрощает изучение и начало работы с *Node.js*.

1.1.5.2 Выбор языка программирования

Наше приложение будет состоять как из клиентской части, так и из серверной, поэтому следует внимательно подойти к выбору языка программирования для каждой из сторон. Существует множество различных сред веб-разработки, каждая из которых имеет свои сильные и слабые стороны. В конечном итоге выбор фреймворка будет зависеть от требований проекта, предпочтений языка программирования и личного опыта.

Список языков и библиотек для создания клиентской части:

- *react.js* — это популярная библиотека *JavaScript*, используемая для создания пользовательских интерфейсов. Он известен своей высокой производительностью и возможностью повторного использования компонентов. *React.js* использует декларативный подход к созданию пользовательского интерфейса, что упрощает его понимание и работу. Он поддерживается *Facebook* и имеет большое сообщество разработчиков, что означает, что он постоянно развивается и совершенствуется;
- *angularJS* — это среда *JavaScript*, используемая для создания динамических веб-приложений. Он поддерживается *Google* и имеет большое сообщество разработчиков. Он предоставляет мощный набор функций, таких как двусторонняя привязка данных, архитектура *MVC* и внедрение зависимостей. *AngularJS* идеально подходит для создания сложных веб-приложений, требующих много интерактивности;
- *django* — это платформа *Python*, используемая для быстрого и эффективного создания веб-приложений. Он известен своими функциями безопасности и масштабируемостью. *Django* предоставляет мощный набор функций, таких как *ORM*, маршрутизация *URL*-адресов и механизм шаблонов. Он идеально подходит для создания крупномасштабных веб-приложений, требующих высокой производительности и безопасности;

– *ruby on Rails* — популярная платформа для создания веб-приложений с использованием языка программирования *Ruby*. Он известен своей простотой и удобством использования. *Rails* предоставляет набор соглашений, упрощающих разработчикам быстрое создание приложений. Он также предоставляет мощный набор функций, таких как *ORM*, маршрутизация и механизм шаблонов;

– *laravel* — это *PHP*-фреймворк, используемый для создания веб-приложений, известный своим элегантным синтаксисом и обширной документацией. *Laravel* предоставляет мощный набор функций, таких как *ORM*, маршрутизация и механизм шаблонов. Он идеально подходит для создания веб-приложений, требующих масштабируемости и простоты обслуживания.

Список языков для создания серверной части:

– *node.js* популярен для создания быстрых и масштабируемых веб-приложений. Он использует управляемую событиями неблокирующую модель ввода-вывода, что делает его подходящим для приложений с большими объемами данных, которые необходимо быстро обрабатывать. Его также легко использовать, поскольку он использует *JavaScript*, с которым уже знакомы многие разработчики;

– *ruby on Rails* — это популярная среда веб-приложений, в которой предпочтение отдается соглашениям, а не настройке. Он известен своей простотой и способностью быстро создавать веб-приложения. Он также следует архитектурному шаблону *Model-View-Controller (MVC)*, что упрощает поддержку и обновление кода;

– *java* — это язык программирования общего назначения, который широко используется для приложений корпоративного уровня. Он известен своей надежностью и функциями безопасности, что делает его популярным выбором для создания крупномасштабных приложений. Кроме того, он

кроссплатформенный, что означает, что код *Java* может работать на любой платформе, которая его поддерживает;

– *python* — это язык программирования высокого уровня, известный своей простотой и удобочитаемостью. Он часто используется для веб-разработки, научных вычислений и анализа данных. Его легко изучить и использовать, и у него большое сообщество разработчиков, которые вносят свой вклад в библиотеки и платформы с открытым исходным кодом. Он также известен своей универсальностью, что делает его полезным для широкого круга проектов.

1.1.5.3 Выбор СУБД

Для проектирования базы данных приложения для сопровождения тренировок необходимо выбрать систему управления базами данных. Для решения поставленной задачи, необходимо выбрать такую СУБД, которая будет проста в настройке и развертывании, отказоустойчива, безопасна и стабильна:

– *oracle* — это система управления реляционными базами данных, которая широко используется в корпоративных средах. Он предлагает широкий спектр функций и инструментов для управления данными, таких как хранилище данных, обработка транзакций и интеллектуальный анализ данных. *Oracle* известна своей надежностью, масштабируемостью и безопасностью. Но есть несколько минусов:

1) дороговизна. Система управления базами данных *Oracle* — одна из самых дорогих систем управления базами данных на рынке. Лицензионные сборы и затраты на обслуживание могут быть довольно высокими для малого и среднего бизнеса;

2) сложность. Система управления базами данных *Oracle* — очень сложная система, для управления которой требуются

высококвалифицированные администраторы баз данных. Это также может быть довольно сложно установить и настроить;

3)ресурсоемкость: система управления базами данных *Oracle* требует много ресурсов для эффективной работы. Для оптимальной работы ей требуются высокопроизводительные серверы, системы хранения и другие компоненты инфраструктуры;

4)привязка к поставщику. После того, как вы вложили средства в систему управления базами данных *Oracle*, переход на другую систему может оказаться сложной задачей. Эта привязка к поставщику может ограничить ваши возможности и увеличить ваши расходы в долгосрочной перспективе;

5)проблемы безопасности. Система управления базами данных *Oracle* на протяжении многих лет имела немало уязвимостей в системе безопасности. Это может быть проблемой для предприятий, которым необходимо защитить свои данные и системы от киберугроз.

– *mySQL* — это система управления реляционными базами данных с открытым исходным кодом, которая обычно используется для веб-приложений. Он известен своей простотой использования, масштабируемостью и скоростью. *MySQL* поддерживает широкий спектр языков программирования и предлагает множество функций, таких как хранимые процедуры, триггеры и представления;

– *microsoft SQL Server* — это система управления реляционными базами данных, которая чаще всего используется с приложениями на базе *Windows*. Он предлагает ряд функций для управления данными, таких как хранилище данных, бизнес-аналитика и отчетность. *Microsoft SQL Server* известен своей масштабируемостью, безопасностью и интеграцией с другими продуктами *Microsoft*;

– *postgresql* — это система управления реляционными базами данных с открытым исходным кодом, известная своей надежностью, масштабируемостью и расширяемостью. Он предлагает широкий спектр функций, таких как поддержка *JSON*, *XML* и пространственных данных, и обычно используется для веб-приложений. *PostgreSQL* также известен своей разработкой и поддержкой сообщества;

– *mongodb* — это документно-ориентированная система управления базами данных *NoSQL*, которая обычно используется для веб-приложений. Он предлагает высокую масштабируемость и гибкость и известен своей способностью обрабатывать неструктурированные данные. *MongoDB* использует гибкую схему, которая позволяет вносить изменения в структуру данных, не затрагивая приложение;

– *cassandra* — это распределенная система управления базами данных *NoSQL*, которая обычно используется для хранения больших объемов данных на нескольких серверах. Он обеспечивает высокую масштабируемость, доступность и отказоустойчивость и предназначен для обработки больших объемов данных с высокой пропускной способностью при записи. *Cassandra* обычно используется в средах, где важны высокая доступность и масштабируемость, например, в социальных сетях и на платформах электронной коммерции.

1.1.6 Выбранные технологии

В качестве языка и библиотеки для создания клиентской части было решено использовать *React.js*. Он имеет ряд преимуществ:

– *react* использует декларативный подход к программированию, упрощая разработчикам описание того, что они хотят, чтобы приложение делало, а не процесс того, как это сделать;

Изм.	Лист	№ докум.	Подпись	Дата

- *react* использует архитектуру на основе компонентов, которая обеспечивает лучшее повторное использование кода, обслуживание и организацию;
- *react* использует виртуальный *DOM*, что позволяет быстрее отображать и обновлять пользовательский интерфейс;
- *react* использует одностороннюю привязку данных, что упрощает управление и поддержку данных во всем приложении;
- *react* имеет большое и активное сообщество разработчиков, а это означает, что существует множество ресурсов и инструментов, доступных для помощи в разработке;
- *react* можно использовать для разработки приложений как для веб-платформ, так и для мобильных платформ, что делает его универсальным выбором для разработчиков.

В качестве языка для создания серверной части было решено использовать *node.js*. Существует множество преимуществ использования *Node.js* для создания серверного приложения, в том числе:

- *node.js* построен на движке *JavaScript V8*, который невероятно быстр и эффективен. В результате приложения *Node.js* могут легко обрабатывать большие объемы данных и выполнять сложные операции;
- *node.js* обладает высокой масштабируемостью, что позволяет разработчикам легко добавлять или удалять серверы по мере необходимости для удовлетворения меняющихся потребностей.
- *node.js* позволяет разработчикам использовать *JavaScript* как во внешнем, так и во внутреннем интерфейсе, что упрощает разработку и поддержку приложений;
- *node.js* имеет большое и активное сообщество разработчиков, которые вносят свой вклад в его развитие и предоставляют поддержку и ресурсы для других разработчиков. Поскольку *Node.js* основан на *JavaScript*,

разработчики, уже знакомые с *JavaScript*, могут быстро изучить и начать использовать *Node.js* для серверной разработки;

– *node.js* может работать в нескольких операционных системах, включая *Windows*, *Linux* и *MacOS*, что делает его более гибким для разработчиков.

В качестве СУБД было решено использовать *PostgreSQL*. Основным преимуществом *PostgreSQL* являются:

– открытый исходный код: *PostgreSQL* — это бесплатная система управления базами данных с открытым исходным кодом, что означает, что ее можно свободно загружать, использовать и модифицировать;

– высокая производительность. *PostgreSQL* известен своей высокой производительностью и масштабируемостью. Он может эффективно обрабатывать большие объемы данных и сложные запросы;

– расширяемость: *PostgreSQL* обладает высокой расширяемостью и поддерживает широкий спектр типов данных, включая *JSON*, *XML* и пространственные данные;

– безопасность: *PostgreSQL* имеет надежную модель безопасности, которая включает в себя такие функции, как шифрование *SSL*, аутентификация пользователей и управление разрешениями;

– совместимость: *PostgreSQL* совместим с широким спектром языков программирования и сред, включая *Python*, *Java* и *Ruby*;

– поддержка сообщества: *PostgreSQL* имеет большое и активное сообщество разработчиков и пользователей, которые вносят свой вклад в его разработку, документацию и поддержку в устранении неполадок.

1.2 Реализация

1.2.1 Настройка окружения

Перед тем, как начать разрабатывать наше веб-приложение необходимо настроить окружение. При создании серверной части будет использоваться *NodeJS*, а для клиентской было решено использовать фреймворк *React*. Для корректной работы нужно установить программную платформу *node.js*. Для этого необходимо открыть и перейти на сайт <https://nodejs.org/ru> и загрузить последнюю или рекомендованную версию. Это показано на рисунке 4.

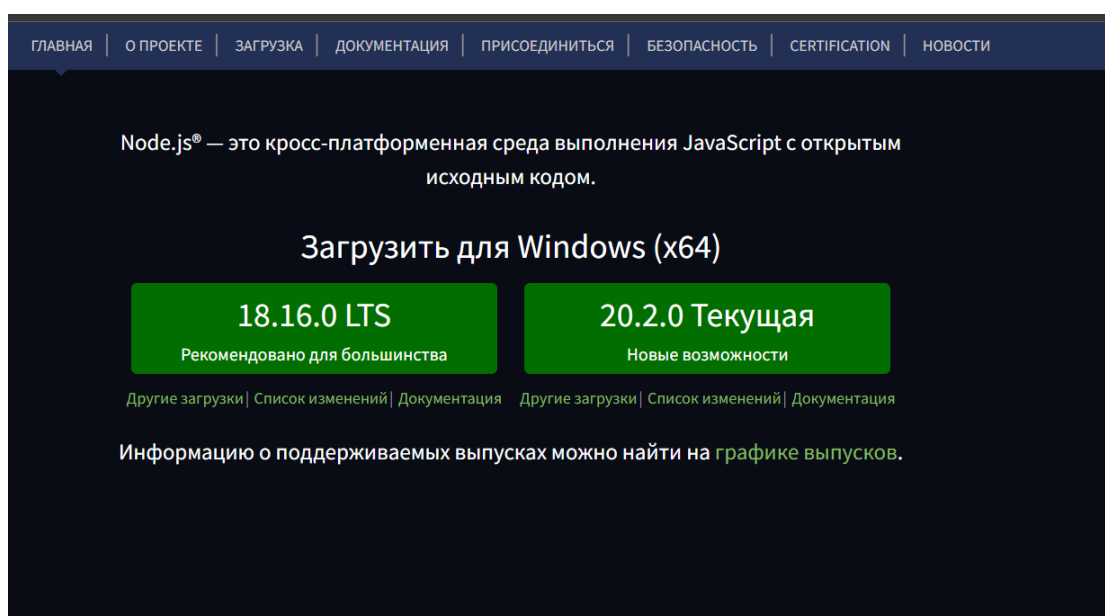


Рисунок 4 – Установка *node.js*

На рисунке 5 показано, что после установки необходимо проверить установилась ли среда. Для это нужно открыть терминал и прописать команду *node -v*. В случае успешной установки в терминале должна высветится текущая версия *Node.js*.

```

MINGW64:/
gusev@vitaliy MINGW64 /
$ node -v
v18.15.0

gusev@vitaliy MINGW64 /
$ |

```

Рисунок 5 – Успешная установка *node.js*

После это можем переходить к созданию *React* приложения. Для этого необходимо ввести в терминале команду *npx create-react-app Test1*. Благодаря данной команде у нас получится создать базовый скелет *react* приложения. На рисунке 6 показан данный процесс.

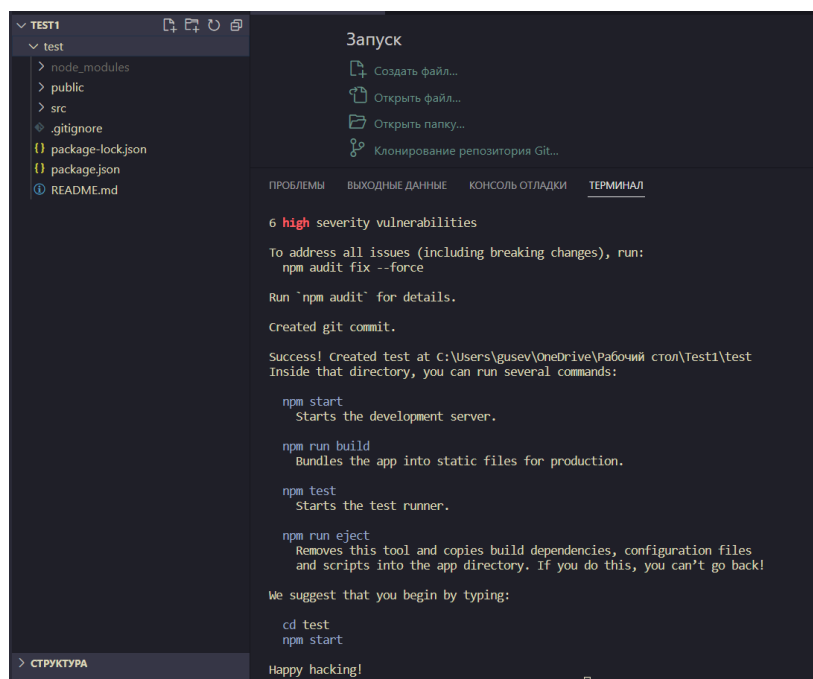


Рисунок 6 – Создание сайта

После предварительных настроек можно приступить к разработке функционала и графической части приложения, а также можно приступать к работе с БД.

1.2.2 Работа с БД

Для хранения информации необходимо реализовать базу данных со следующими таблицами:

- *user* - таблица, содержащая информацию о клиенте;
- *workout* – таблица, содержащая информацию о тренировках;
- *exercise* - таблица, содержащая информацию о упражнениях;
- *exerciseLog* - таблица, содержащая информацию о истории упражнений;
- *workoutLog* - таблица, содержащая информацию о истории тренировок;
- *exerciseTime* – таблица содержащая информацию о количестве подходов.

Описание атрибутов сущности *user* представлено в таблице 1.

Таблица 1 — Описание атрибутов сущности *user*.

Атрибуты	Назначение	Ключевое поле	Нулевое значение	Тип данных
<i>id</i>	Идентификатор пользователя	<i>PK</i>	Недопустимо	<i>INTEGER</i>
<i>email</i>	Логин пользователя		Недопустимо	<i>STRING</i>
<i>name</i>	Имя пользователя		Недопустимо	<i>STRING</i>
<i>password</i>	Пароль пользователя		Недопустимо	<i>STRING</i>

Атрибуты	Назначение	Ключевое поле	Нулевое значение	Тип данных
<i>images</i>	Фотографии пользователя до\после		Недопустимо	<i>STRING</i>
<i>createdAt</i>	Дата создания		Недопустимо	<i>TIMESSTAMP</i>
<i>updatedAt</i>	Дата обновления		Недопустимо	<i>TIMESSTAMP</i>

Описание атрибутов сущности *workout* представлено в таблице 2.

Таблица 2 — Описание атрибутов сущности *workout*.

Атрибуты	Назначение	Ключевое поле	Нулевое значение	Тип данных
<i>id</i>	Идентификатор тренировки	PK	Недопустимо	<i>INTEGER</i>
<i>name</i>	Имя тренировки		Недопустимо	<i>STRING</i>
<i>createdAt</i>	Дата создания		Недопустимо	<i>TIMESSTAMP</i>
<i>updatedAt</i>	Дата обновления		Недопустимо	<i>TIMESSTAMP</i>

Описание атрибутов сущности *exercise* представлено в таблице 3.

Таблица 3 — Описание атрибутов сущности *exercise*.

Атрибуты	Назначение	Ключевое	Нулевое	Тип
----------	------------	----------	---------	-----

		поле	значение	данных
id	Идентификатор упражнения	PK	Недопустимо	<i>INTEGER</i>
name	Название упражнения		Недопустимо	<i>STRING</i>
times	Количество подходов		Недопустимо	<i>STRING</i>
iconPath	Путь к иконке упражнения		Недопустимо	<i>STRING</i>
<i>createdAt</i>	Дата создания		Недопустимо	<i>TIMESTAMP</i>
<i>updatedAt</i>	Дата обновления		Недопустимо	<i>TIMESTAMP</i>

Описание атрибутов сущности *exerciseLog* представлено в таблице 4.

Таблица 4 — Описание атрибутов сущности *exerciseLog*.

Атрибуты	Назначение	Ключевое поле	Нулевое значение	Тип данных
<i>id</i>	Идентификатор истории упражнения	PK	Недопустимо	<i>INTEGER</i>
<i>isCompleted</i>	Состояние выполнения		Недопустимо	<i>BOOLEAN</i>
<i>times</i>	Количество подходов		Недопустимо	<i>INTEGER</i>
<i>userId</i>	Идентификатор		Недопустимо	<i>INTEGER</i>

Атрибуты	Назначение	Ключевое поле	Нулевое значение	Тип данных
	пользователя		о	ER
<i>workoutLog_id</i>	Идентификатор лога тренировки		Недопустим о	INTEGR
<i>exercise_id</i>	Идентификатор упражнения		Недопустим о	INTEGR
<i>createdAt</i>	Дата создания		Недопустим о	TIMESTAMP
<i>updatedAt</i>	Дата обновления		Недопустим о	TIMESTAMP

Описание атрибутов сущности *workoutLog* представлено в таблице 5.

Таблица 5 — Описание атрибутов сущности *workoutLog*.

Атрибуты	Назначение	Ключевое поле	Нулевое значение	Тип данных
<i>id</i>	Идентификатор записи лога	PK	Недопустимо	INTEGER
<i>isCompleted</i>	Состояние выполнения		Недопустимо	BOLLEAN
<i>userId</i>	Идентификатор пользователя		Недопустимо	INTEGER
<i>workout_id</i>	Идентификатор тренировки		Недопустимо	INTEGER
<i>createdAt</i>	Дата создания		Недопустимо	TIMESTAMP

Атрибуты	Назначение	Ключевое поле	Нулевое значение	Тип данных
<i>updatedAt</i>	Дата обновления		Недопустимо	<i>TIMESSTAMP</i>

Описание атрибутов сущности *exerciseTime* представлено в таблице 6.

Таблица 6 — Описание атрибутов сущности *exerciseTime*.

Атрибуты	Назначение	Ключевое поле	Нулевое значение	Тип данных
<i>id</i>	Идентификатор клиента	PK	Недопустимо	<i>INTEGER</i>
<i>weight</i>	Вес		Недопустимо	<i>INTEGER</i>
<i>repeat</i>	Количество подходов		Недопустимо	<i>INTEGER</i>
<i>isCompleted</i>	Состояние выполнения		Недопустимо	<i>BOLLEAN</i>
<i>exerciseLog_id</i>	Идентификатор лога упражнения		Недопустимо	<i>INTEGER</i>
<i>createdAt</i>	Дата создания		Недопустимо	<i>TIMESSTAMP</i>
<i>updatedAt</i>	Дата обновления		Недопустимо	<i>TIMESSTAMP</i>

Схема таблиц БД приведена на рисунке 8:

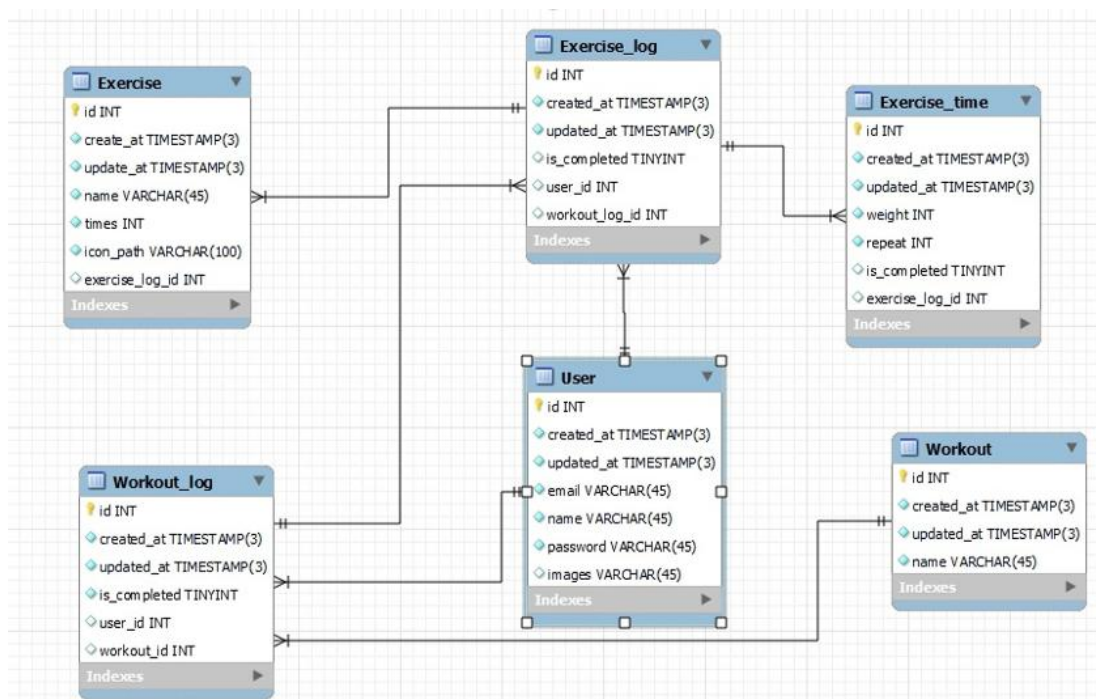


Рисунок 8 - Схема таблиц базы данных.

1.2.3 Описание доступных функций

Приложение должно реализовывать такие функции как:

- регистрацию пользователя;
- вход пользователя по логину и паролю;
- создание упражнений;
- просмотр списка упражнений;
- возможность завершить упражнение;
- наличие истории упражнения;
- создание тренировки;
- добавление в тренировку готовых упражнений;
- возможность завершения тренировки;
- возможность просмотра списка готовых тренировок;
- обновление статистики о выполненных тренировках.

Для того чтобы начать пользоваться данным приложением пользователю необходимо зарегистрироваться. Для это нужно ввести адрес своей

электронной почты и придумать пароль, после чего нажать на кнопку Войти. Это показано на рисунке 9.

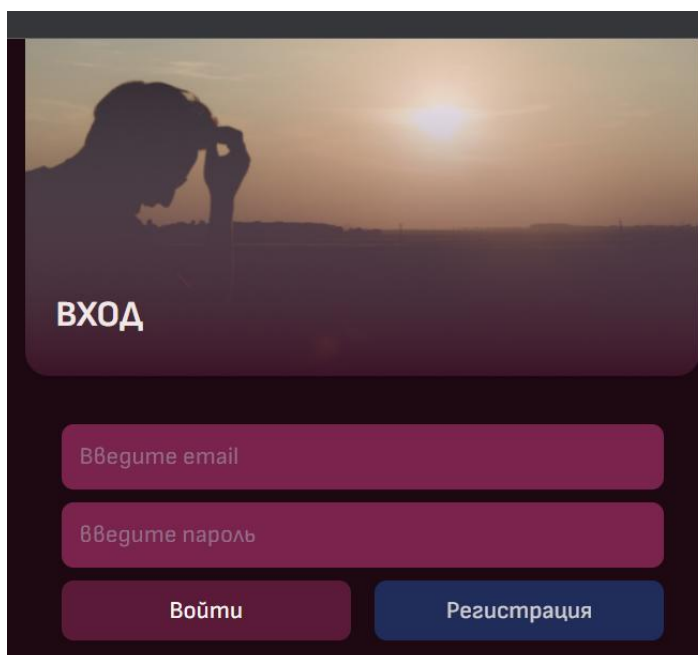


Рисунок 9 – Начало работы приложения

После регистрации пользователь попадает на главную страницу. На ней есть кнопка с меню, в котором мы можем выйти из учетной записи, создать новую тренировку или посмотреть список имеющихся, а также перейти на страницу профиля, где показана статистика пользователя. Можем видеть это на рисунках 10-12.

Прямо со стартовой страницы пользователь может создать новую тренировку нажав на кнопку Новая тренировка. После чего его перенаправит на новую страницу (рисунок 13). Тут он пользователь может ввести имя тренировки и выбрать упражнение из списка имеющихся, либо же создать новое, нажав на кнопку Создать новое упражнение. Это показано на рисунке 14 и рисунке 15.

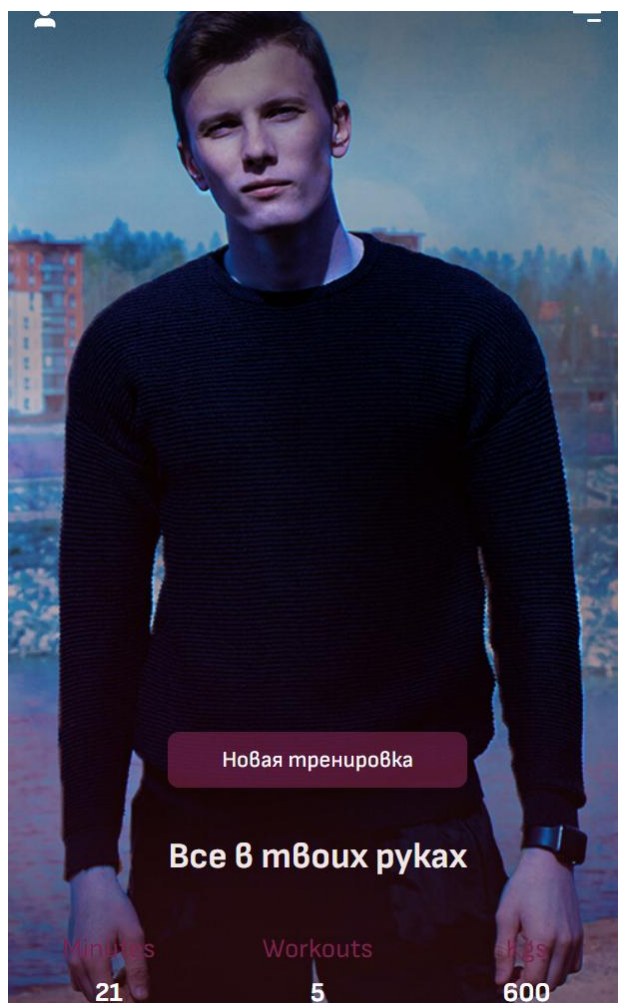


Рисунок 10 – Главная страница

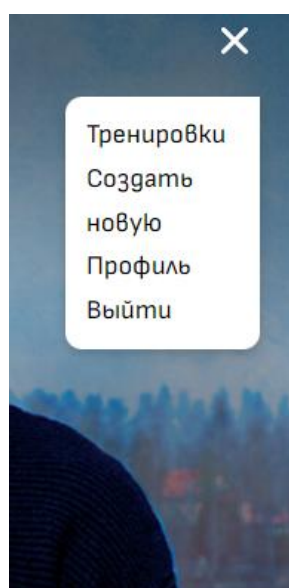


Рисунок 11 – Меню

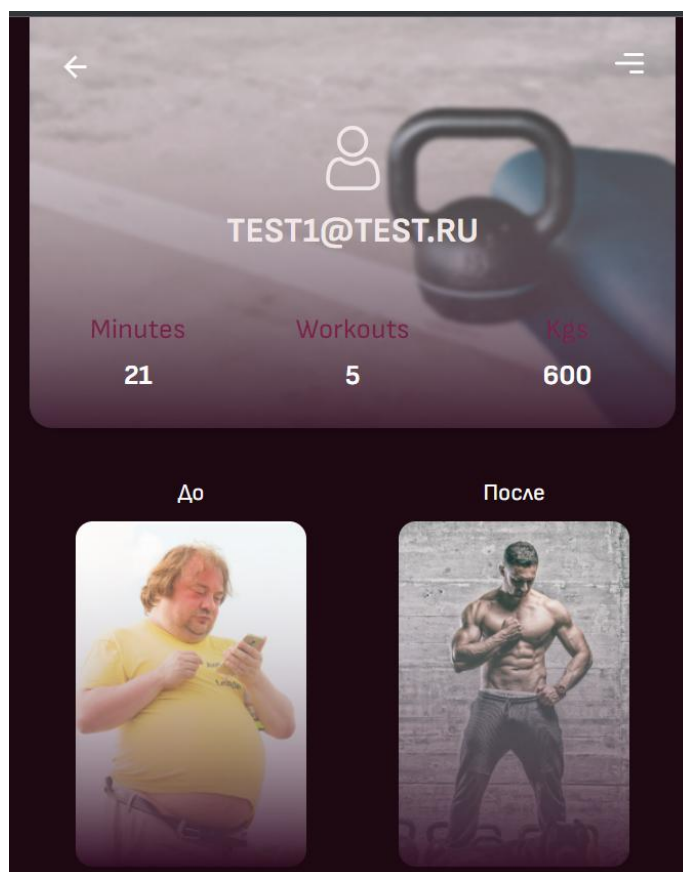


Рисунок 12 – Страница пользователя

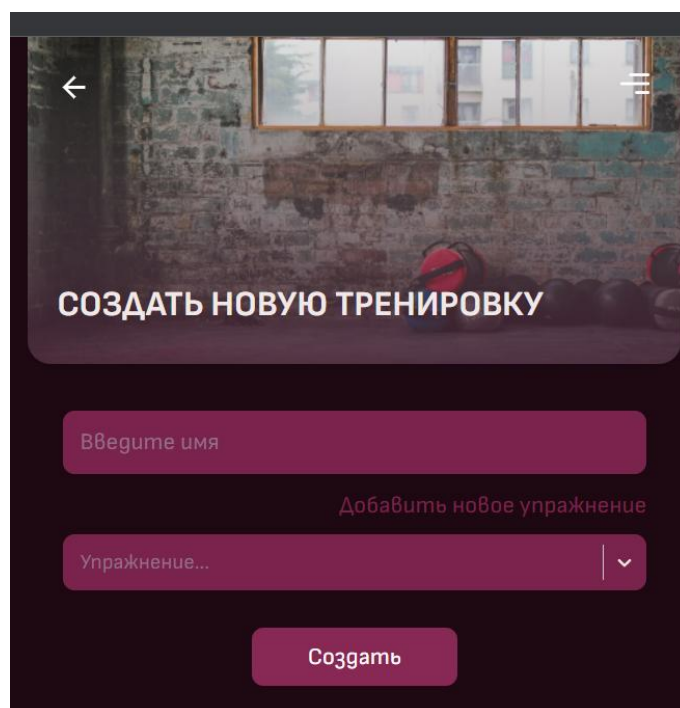


Рисунок 13 – Страница создания новой тренировки

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.6.001 ПЗ

Лист

33

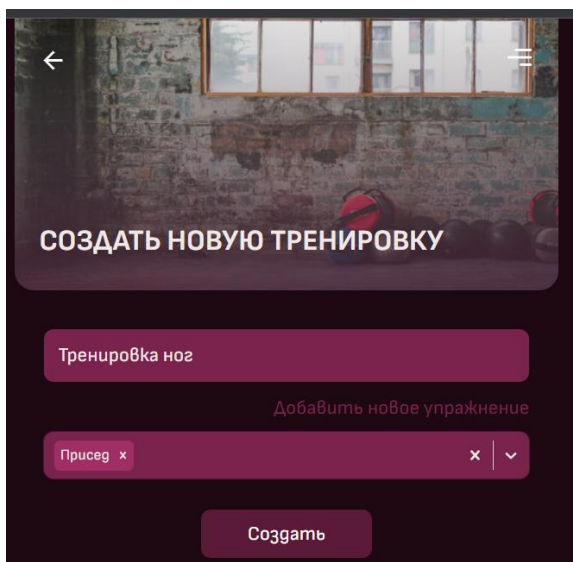


Рисунок 14 – Создание тренировки

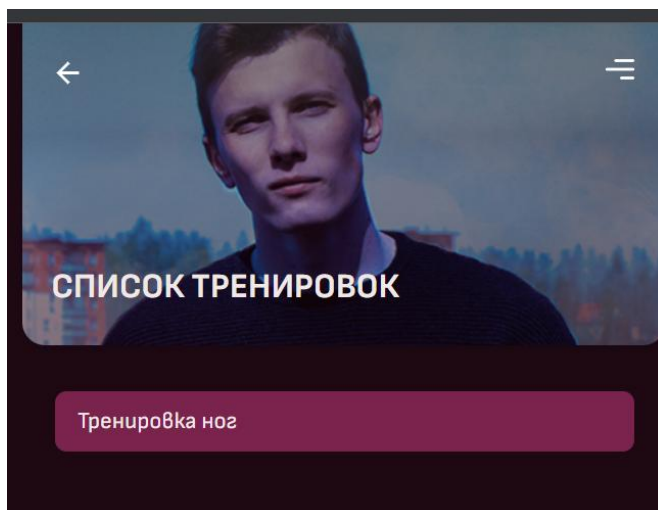


Рисунок 15 – Успешное добавление тренировок в список

Если же пользователь не нашел нужное ему упражнение, то он может создать его сам. Для этого нужно перейти на страницу создания тренировки, после чего нажать на кнопку Добавить новое упражнение (рисунок 16). Данный компонент нашего приложения позволяет нам создать упражнение, выбрать иконку, благодаря которой мы будем визуальным образом определять на какую группу мышц данное упражнение. Так же мы должны указать количество подходов. Это показано на рисунке 17.

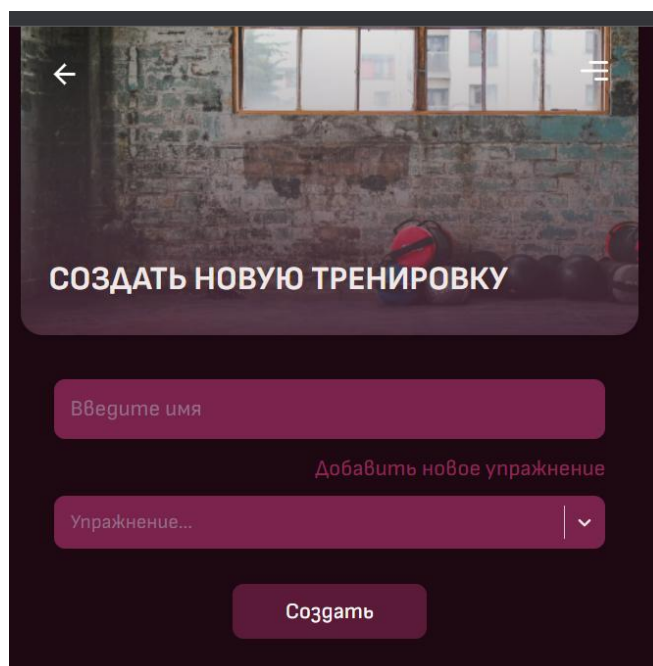


Рисунок 16 – кнопка создания нового упражнения

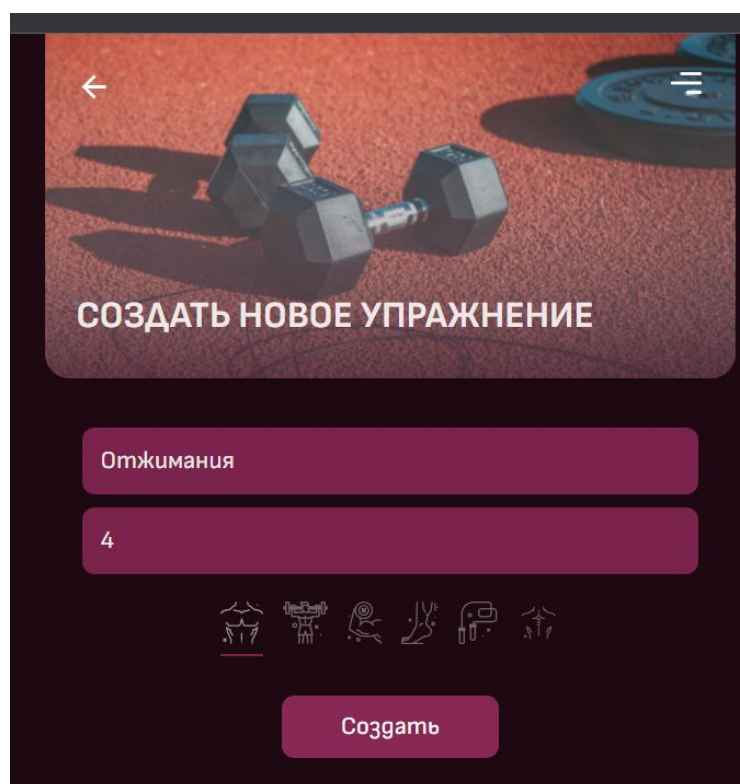


Рисунок 17 – Добавление нового упражнения

После того как мы создали упражнение можем посмотреть его в списке доступных упражнений для. Это продемонстрировано на рисунке 18.

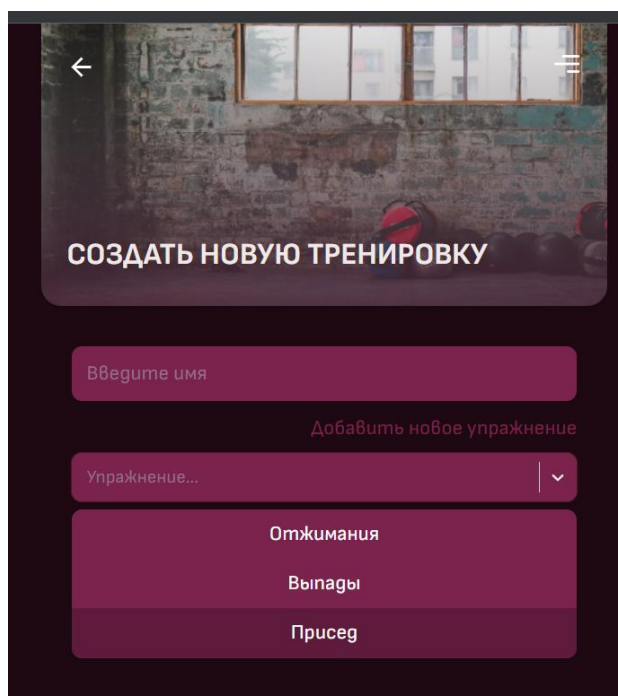


Рисунок 18 – Добавление нового упражнения в список

Выполнение тренировки. Для того чтобы пользователю начать выполнять тренировку ее нужно выбрать на странице *Тренировки*, после чего нажать на нее. Откроется страница выбранной тренировки, в которой будут упражнения, из которых она состоит. Нажимаем на упражнение. Это показано на рисунке 19.

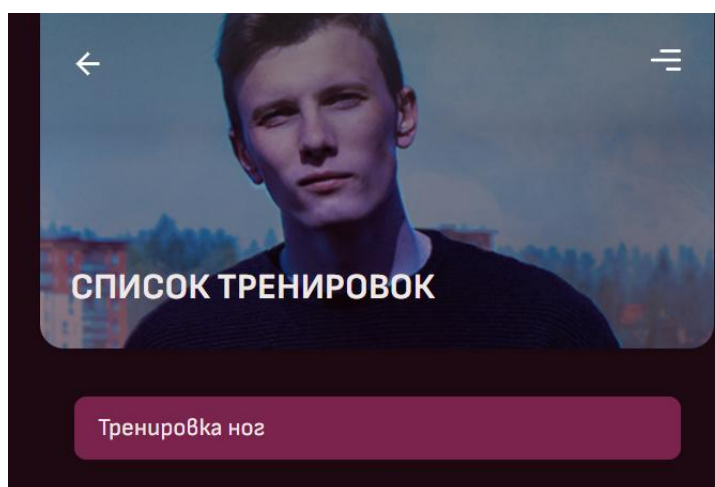


Рисунок 19 – Список упражнений, из которых состоит тренировка

После нажатия нам откроется страница упражнения. Она состоит из трех колонок:

- вес / повторения упражнений, выполняемых на предыдущих, тренировка;
- текущий вес / повторения;
- состояние выполнения. После того как пользователь выполнил нужное количество повторений с определенным весом в подходе он ставит галочку в столбце Выполнено. Это показано на рисунке 20.

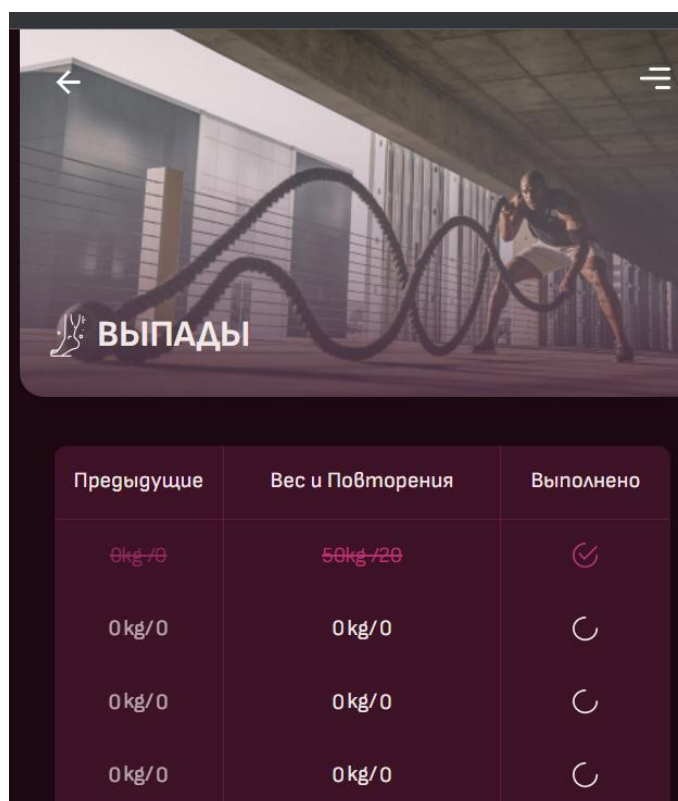


Рисунок 20 – таблица подходов, состояния выполнения и предыдущих результатов

Для того чтобы завершить выполнять данное упражнение в тренировки необходимо закончить все подходы, после чего упражнение пометится выполненным. Это показано на рисунке 21.

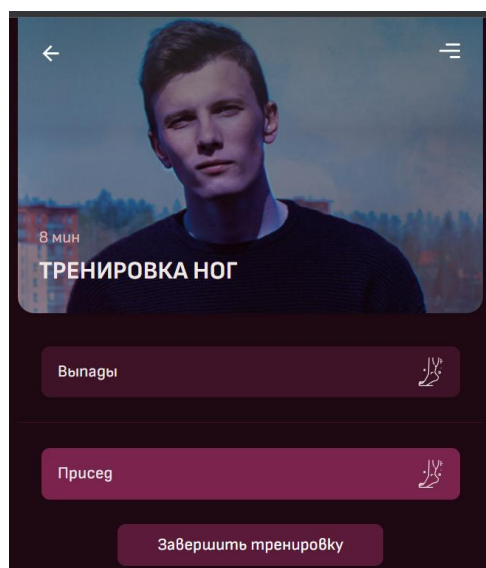


Рисунок 21 – вид выполненного упражнения

Чтобы завершить тренировку необходимо выполнить все упражнения, после чего нажать на кнопку Завершить тренировку (рисунок 22).

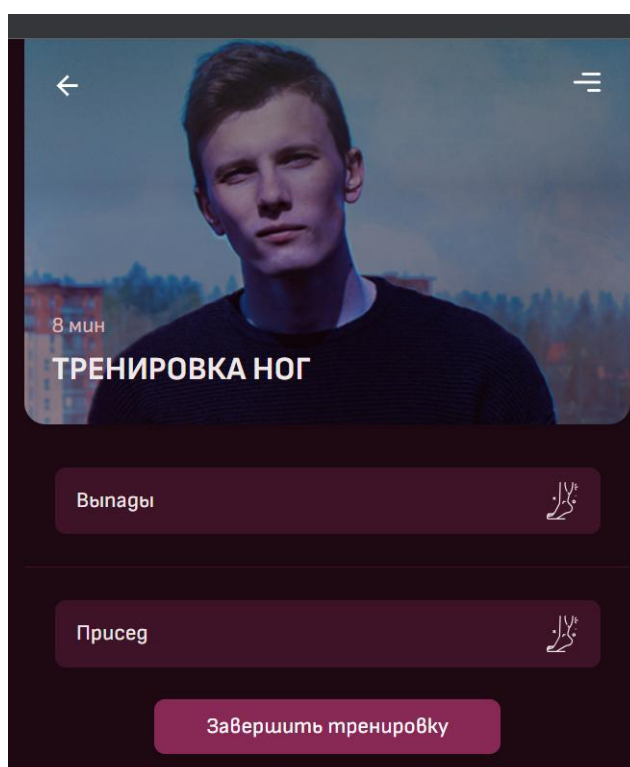


Рисунок 22 – выполнение тренировки

После того как тренировка была выполнена у нас обновилась статистика в профиле. Поднятые килограммы засчитались в общий тоннаж, количество выполненных тренировок увеличилось. Это видно на рисунке 23.

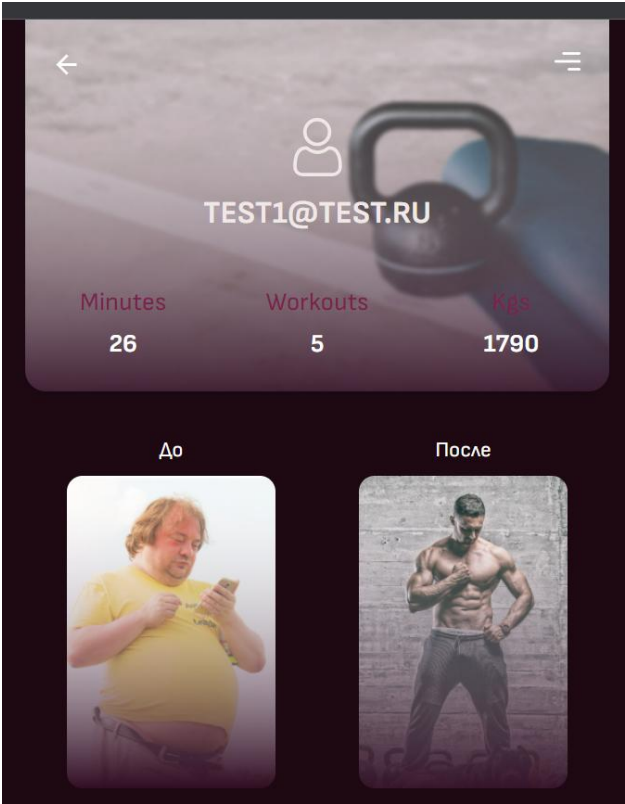


Рисунок 23 – обновление статистики

1.2.4 Описание программы

Диаграмма развертывания, приведенная на рисунке 24, отображает все возможное физическое развертывание артефактов на узлах.

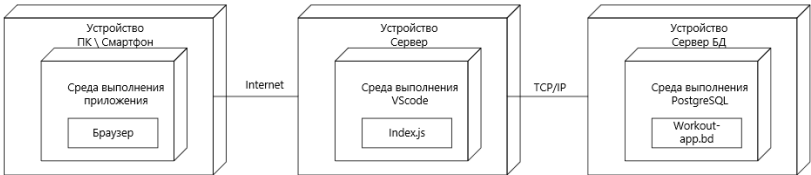


Рисунок 24 - UML-диаграмма развертывания

Ниже приведено описание файлов, составляющих важную часть программы:

- файл *package.json* - файл, с различными зависимостями. библиотеками необходимыми для разработки нашего приложения;
- файл *.gitignore* - данный файл ограничивает попадание в репозиторий файлов с системными переменными, папкой *node_modules*, различных кэшей и т.д.;
- файл *.env* - файл с настройками приложения, который при начале работы получает данные из переменных, а именно режим разработки, путь к базе данных и секретный ключ для шифрования паролей;
- файл *server.js* - файл, в который мы импортируем все функции для запроса к базе данных, после чего вызываем асинхронную функцию *main*. После чего идет подключение всех обработчиков из вышеупомянутых импортов. С помощью функции *listen* сервер запускается и начинает работу.;
- файл *schema.prisma* - файл с описанием таблиц базы данных.

Рассмотрим на примере таблицы *User* поля в базе данных: *id Int @id @default(autoincrement())* (идентификатор пользователя), *createdAt DateTime @default(now())* (дата создания пользователя, *updatedAt DateTime @updatedAt* (дата обновления данных в таблице), *email String @unique*(электронная почта пользователя, должна быть уникальна), *name String*(имя пользователя), *password String*(пароль);

– файл *auth.controller* - в данном файле описан механизм регистрации и авторизации пользователя. Регистрация выполняется функцией *registerUser*. Изначально мы получаем данные из запроса и проверяем существует ли пользователь с такой электронной почтой. В случае если данный пользователь существует, то мы выводим ошибку *User already exists*. Если же такого пользователя нет, то обращаемся к базе данных, в ней создаем пользователя. Из полученного запроса достаем электронную почту,

пароль, генерируем имя и создаем уникальный токен для пользователя, по которому будет осуществляться авторизация. Функция *authUser* отвечает за авторизацию уже созданного пользователя. Из запроса мы получаем введенный пользователем *email* и пароль, после чего обращаемся к базе данных, где ищем пользователя по электронной почте. Если такая почта существует, то мы сравниваем пароль из базы данных с паролем из запроса. Если он совпадает, то авторизация успешна, иначе программа выдает ошибку *Email or password are not correct*;

- файл *exercise.controller* -файл, где функция *createNewExercise* получает из запроса имя упражнения, количество подходов и путь к иконке упражнения. После чего создает в базе данных данное упражнение;

- файл *workout.controller* – данный файл отвечает за создание тренировки, за получение всех тренировок, а также за обновление и удаление тренировок;

- файл *user.controller* - файл, отображает статистику пользователя;

- файл *auth.middlwere* - этот код в файле определяет *middleware*-функцию *protect*, которая может использоваться в *Express.js* приложениях для защиты определенных маршрутов или конечных точек от неавторизованного доступа. Цель этого промежуточного программного обеспечения - проверить *подлинность JSON Web Token (JWT)*, который должен быть включен в заголовок *Authorization* входящего запроса. Функция сначала проверяет, есть ли заголовок *Authorization* и начинается ли он со слова "*Bearer*". Если да, то она извлекает токен из заголовка и пытается декодировать его с помощью библиотеки *jsonwebtoken* и секретного ключа, хранящегося в переменной окружения *process.env.JWT_SECRET*. Если токен успешно декодирован, функция использует *ORM prisma* для поиска пользователя с идентификатором, извлеченным из полезной нагрузки токена. Если такой пользователь найден, его данные добавляются в объект *req* в

качестве нового свойства с именем *user*, а *middleware* вызывает функцию *next()* для передачи управления следующему *middleware* в цепочке. Если декодированный токен не соответствует действительному пользователю, *middleware* возвращает *HTTP*-статус 401 (*Unauthorized*) и выбрасывает сообщение об ошибке, указывающее, что токен не удалось авторизовать запрос. Если заголовок *Authorization* отсутствует или он не начинается со слова "*Bearer*", *middleware* также возвращает *HTTP*-статус 401 и выбрасывает сообщение об ошибке, указывающее на отсутствие токена;

- файл *workout.routes.js* - Этот код создает *Express* роутер, который определяет *API*-точки доступа для приложения по логированию тренировок.

- В частности, он включает в себя маршруты для создания, чтения, обновления и удаления тренировок, а также создания и чтения журналов тренировок и завершения журналов тренировок. Функция *protect* используется как промежуточное ПО для аутентификации запросов пользователей перед предоставлением доступа к защищенным маршрутам. Функции *createNewWorkoutLog*, *getWorkoutLog* и *updateCompleteWorkoutLog* являются контроллерами, которые обрабатывают соответствующие *HTTP*-запросы на создание новых журналов тренировок, получение существующих журналов тренировок и обновление завершенных журналов тренировок соответственно. Функции *createNewWorkout*, *getWorkouts*, *getWorkout*, *updateWorkout* и *deleteWorkout* также являются контроллерами, которые обрабатывают соответствующие *HTTP*-запросы для создания новых тренировок, получения всех тренировок, получения конкретной тренировки, обновления конкретной тренировки и удаления конкретной тренировки соответственно. Общая задача этого роутера - определить необходимые конечные точки *API* для полного управления и логирования тренировок в приложении.

2 Безопасность жизнедеятельности

2.1 Правила безопасности при использовании ПК

Правила выполнения работ с ПК обязаны обеспечивать минимизацию основных рисков, перечисленных в ГОСТ 12.0.003-2015. В первую очередь к ним относятся:

- повышенная температура отдельных элементов персонального компьютера;
- монотонность трудового процесса;
- высокий уровень зрительных нагрузок;
- вероятность поражения статическим электричеством, генерируемым техникой;
- недостаточная освещенность в зоне выполнения операций у компьютера;
- высокий уровень напряженности электрического и магнитного полей в зоне установки компьютера;
- другие факторы.

Полностью избавиться от их действия, конечно, не получится, однако работодатель обязан стремиться к тому, чтобы сократить воздействие этих факторов на сотрудников, обеспечивая возможность восстановления нормального самочувствия и работоспособности уже к началу следующей рабочей смены без применения дополнительных средств, например, массажа или медикаментозной терапии.

2.2 Правила безопасности при использовании смартфона

Для того, чтобы предотвратить возможные поломки и повреждения смартфона и сопутствующего оборудования, и уменьшить потенциальные риски возгорания или удара током, необходимо серьезно отнестись к условиям, представленным ниже:

- запрещено самостоятельно открыть смартфон, зарядное устройство и инструмент;
- не сдавливайте ваш смартфон, зарядное устройство и инструмент и не допускайте возможного поражения, а также колющих и режущих повреждений;
- не закрывайте зону внешних коммуникаций и электроснабжения, держите смартфон подальше от открытого огня и горящего табака, избегайте попадания воды или огня на них;
- не используйте телефон и чашу рядом с водой или жидкостью, не прикасайтесь к ним мокрыми руками;
- если телефон или зарядное устройство намокли, во избежание повреждений не пытайтесь сушить их в духовке, микроволновой печи или феном;
- не роняйте, не бросайте и не сгибайте свой мобильный телефон и посуду;
- не используйте телефон и тарелку, если они неисправны, а также после повреждения или падения;
- используйте сертифицированные источники питания класса 2 неограниченной мощности: 5 В постоянного тока (не более 3) или 9 В постоянного тока (не более 2 А);

Расстояние между корпусом и телефоном должно составлять не менее 1 см.

Чтобы предотвратить риски (а также потому, что это является нарушением закона в большинстве стран), не используйте телефон во время вождения, во время работы с производственным инструментом, а также при выполнении других потенциально опасных действий.

Соблюдайте правила авиакомпании об использовании мобильных телефонов во время полета и перед входом. Использование мобильных устройств на борту воздушного судна может быть запрещено законом, а также препятствовать деятельности беспроводных сетей и может быть опасным для воздушного судна. Чтобы избежать этого, переведите телефон в режим полета.

Не используйте телефон и зарядное устройство при температуре окружающей среды ниже 0°C и выше 40°C. Оптимальный температурный диапазон для хранения: от 0°C до 40°C.

Отключайте телефоны и тарелку от сети перед чисткой, а также при сильном грохоте и, если вы планируете не использовать устройство в течение длительного периода времени.

Для чистки используйте чистую, сухую, мягкую ткань.

Не используйте моющие средства, порошки или другие химические вещества (например, спирт или бензол) для очистки телефона и устройства.

Во время нормальной работы телефон и лоток могут нагреваться. Вскоре (в том числе косвенно) при контакте устройств с кожей – это может привести к неприятным ощущениям и может.

Меры предосторожности при использовании кардиостимулятора и других медицинских устройств

Телефоны могут передавать и принимать сигналы *LTE*, *WCDMA*, *GSM*, *Wifi* и *Bluetooth*, а также могут содержать информацию об электромагнитных

					ПГУ1.090301.6.001 ПЗ	Лист
						45
Изм.	Лист	№ докум.	Подпись	Дата		

полях, которые могут мешать работе кардиостимуляторов и других имплантированных медицинских устройств. Владелец должен соблюдать следующие условия:

- следите за тем, чтобы телефон и тарелка всегда находились на расстоянии более 16 см от поверхности кожи над кардиостимулятором или другим медицинским устройством.
- если кардиостимулятор или другие материалы помещены в нужную часть тела, держите телефон слева, а если в левой – справа;
- не кладите мобильный телефон в карманный карман (или другие вещи), который находится рядом с кардиостимулятором или другим имплантированным медицинским устройством;
- если у вас есть какие-либо вопросы о совместимости вашего телефона с таким устройством, проконсультируйтесь с врачом;
- если вы считаете, что мобильное устройство мешает работе обычного кардиостимулятора или другого медицинского устройства, выключите телефон и обратитесь к врачу.

Меры предосторожности при использовании зарядного устройства:

- при отсоединении пластины от розетки тяните непосредственно к дому, а не по кабелю;
- когда вы отвечаете на телефонный звонок, убедитесь, что к адаптеру питания легко получить доступ и он подключен к розетке рядом с телефоном;
- подключайте к телефону только те устройства, которые имеют логотип *USB-IF* или прошли тестирование на соответствие этому стандарту.
- меры предосторожности при использовании и утилизации аккумулятора;

– чтобы заменить аккумулятор, обратитесь в сервисный центр. Использование батареи, не соответствующей стандарту, может привести к возгоранию, взрыву, утечке воды и другим опасным побочным эффектам.

– если из аккумулятора вытекает вода, не допускайте ее попадания в глаза, на кожу и одежду. Если жидкость попала вам в глаза, не трите их, а промойте чистой водой и немедленно обратитесь к врачу;

– держите протекающую батарею подальше от огня, чтобы избежать возгорания или взрыва;

– замена телефона и аккумулятора устройства должна производиться в соответствии с местным законодательством. Не выбрасывайте электронные устройства вместе с бытовыми отходами. Неправильная утилизация может привести к пожару, взрыву и другим опасным побочным эффектам.

Не используйте и не храните мобильный телефон и устройство в пыльных, прокуренных, влажных и загрязненных помещениях, а также в зоне воздействия магнитного поля – это может привести к повреждению или короткому замыканию внутри помещения.

Держите телефон подальше от источников тепла и не подвергайте его воздействию прямых солнечных лучей. Не оставляйте телефон в автомобиле и других местах, где температура может превышать 60 °C, например, на приборной панели автомобиля, на подоконнике, рядом с вентиляционными отверстиями системы отопления, а также за стеклом, которое подвергается длительному воздействию прямых солнечных лучей или сильное ультрафиолетовое излучение. Это приводит к поломке телефона, перегреву аккумулятора, а также может привести к взрыву или возгоранию.

Не используйте телефон и не храните его вблизи легковоспламеняющихся и взрывающихся вещей (например, на заправочных станциях, складах горючего или химических заводах). Не используйте телефон во время взрывных работ и в потенциально взрывоопасном месте,

например, на нижней палубе судна, вблизи горючих и химических соединений в условиях хранения или транспортировки, а также в атмосфере присутствуют частицы химических веществ или пыли (в том числе риса и железа). В подобных случаях одна искра может привести к взрыву или пожару и, как следствие, к травмам и даже смерти.

					ПГУ1.090301.6.001 ПЗ	Лист
						48
Изм.	Лист	№ докум.	Подпись	Дата		

Заключение

Результатом выполнения данной выпускной квалификационной работы можно считать успешно реализованное в соответствии с заданием веб-приложение для сопровождения тренировок.

В результате был проведен глубокий анализ процессов тренировок. Были изучены проблемы тренировок в классическом понимании. Перед началом разработки, также был проведен анализ всех возможных инструментов, а именно: ЯП, СУБД, библиотек для разработки систем веб-приложений.

В итоге было принято решение о разработке приложения на базе *node.js* на языке программирования *JavaScript*, с использованием в разработке системы управления базами данных *postgreSql*.

На основе приведенных выше технологий, удалось спроектировать, разработать и протестировать приложение, со следующим функционалом:

- регистрацию пользователя;
- вход пользователя по логину и паролю;
- создание упражнений;
- просмотр списка упражнений;
- возможность завершить упражнение;
- наличие истории упражнения;
- создание тренировки;
- добавление в тренировку готовых упражнений;
- возможность завершения тренировки;
- возможность просмотра списка готовых тренировок;
- обновление статистики о выполненных тренировках.

В своей перспективе данное веб-приложение может быть расширено до полноценного мобильного приложения, также можно добавить программу

питания, подсчет БЖУ, усовершенствовать блок статистики, а также добавить внутренний таймер.

Список использованных источников

1. Зеланд Ю.В. *PostgreSQL: подробное руководство* / Ю.В. Зеланд. - М.: Диалектика / Вильямс, 2015.
2. *JavaScript and jQuery: Interactive Front-End Web Development* (*JavaScript и jQuery: разработка интерактивных сайтов*), Джон Дакетт.
3. *Eloquent JavaScript 3rd Edition* («Выразительный JavaScript»), Марейн Хавербек.
4. *React: Up & Running: Building Web Applications* by Stoyan Stefanov.
5. *Node.js Web Development* by David Herron.
6. *PostgreSQL: Up and Running* by Regina O. Obe and Leo S. Hsu

Приложение А

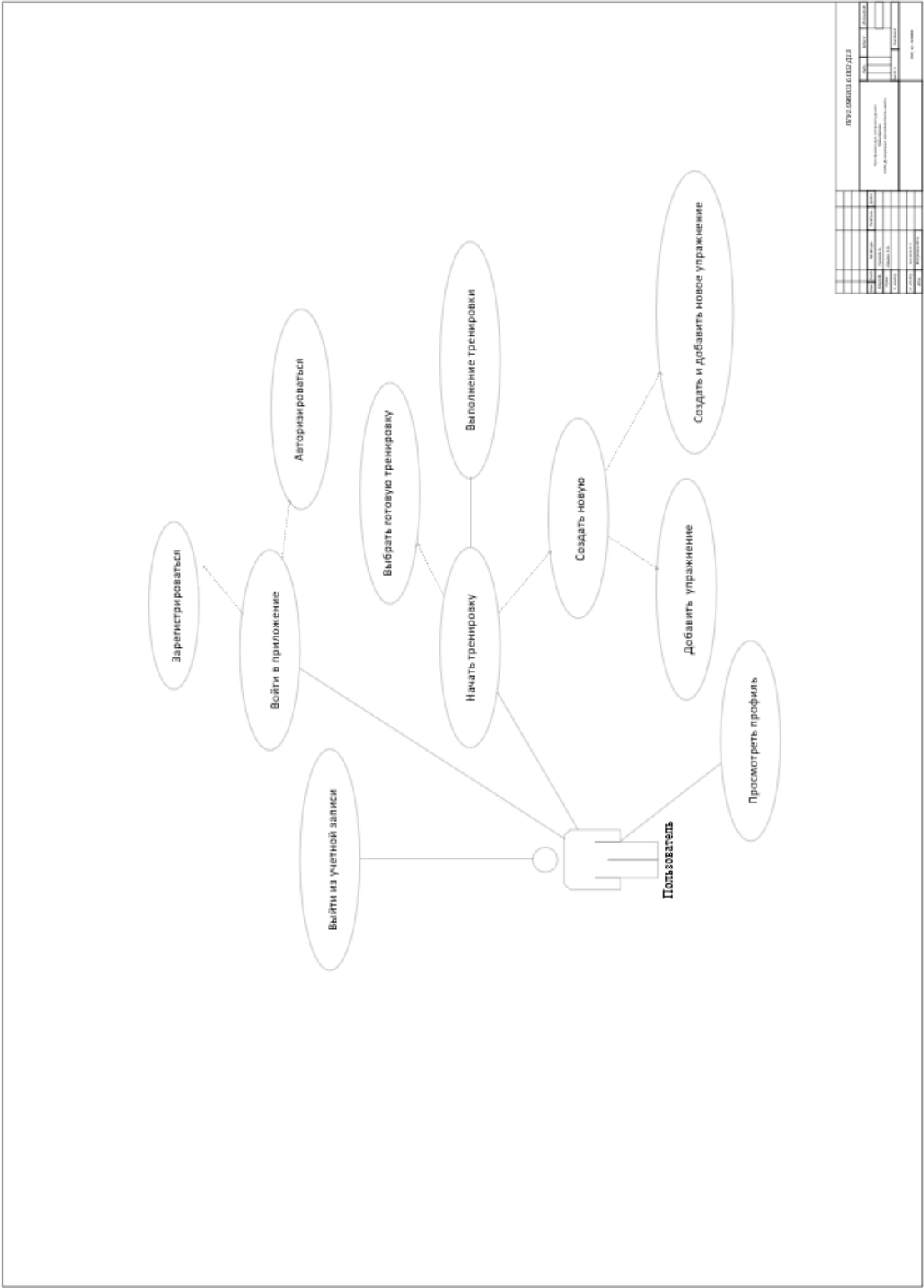
(обязательное)

Разработка веб-приложения для сопровождения тренировок

UML-диаграммы

					ПГУ1.090301.6.001 ПЗ	Лист
						52
Изм.	Лист	№ докум.	Подпись	Дата		

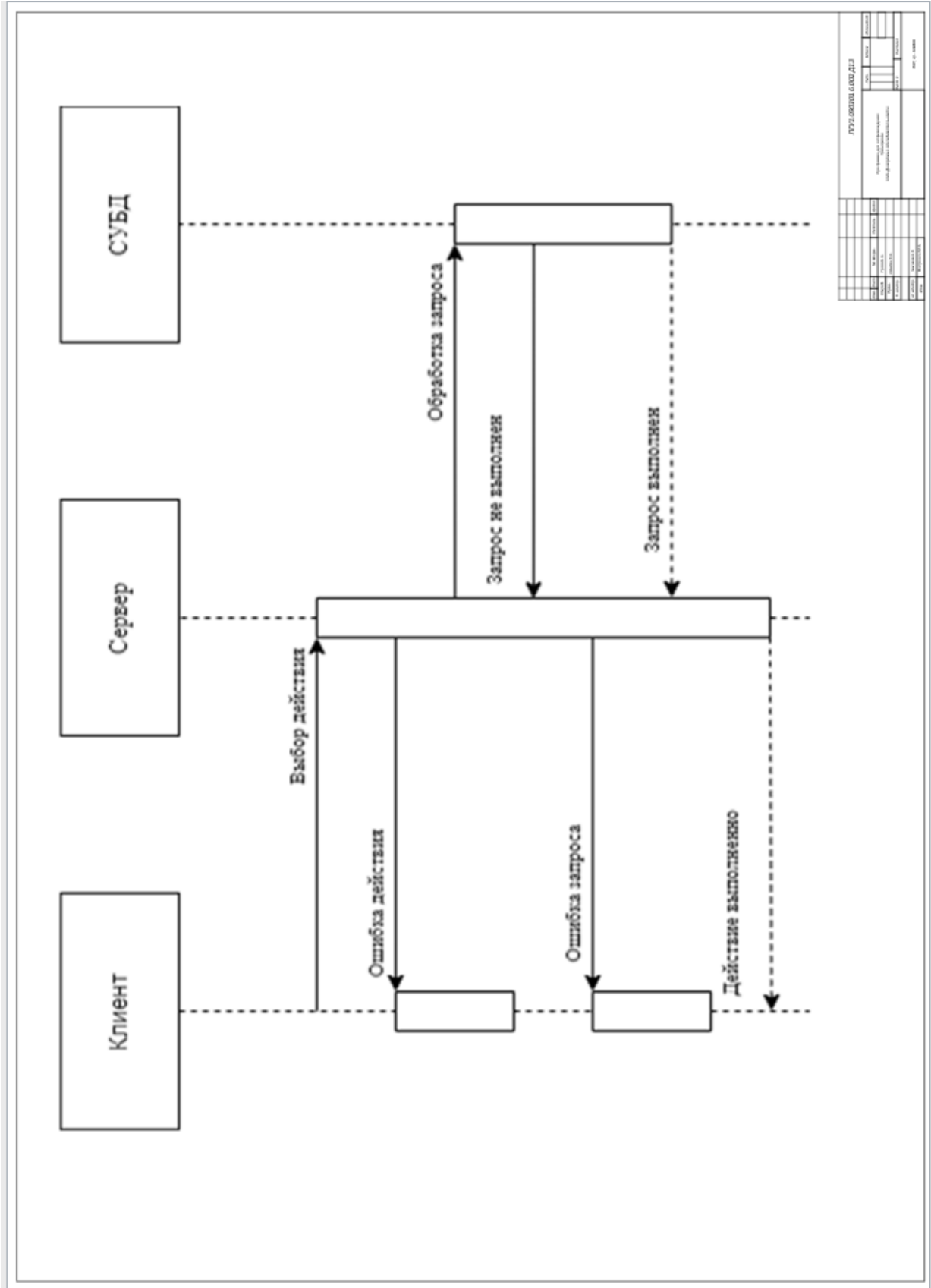
Приложение А.1 - Диаграмма вариантов использования



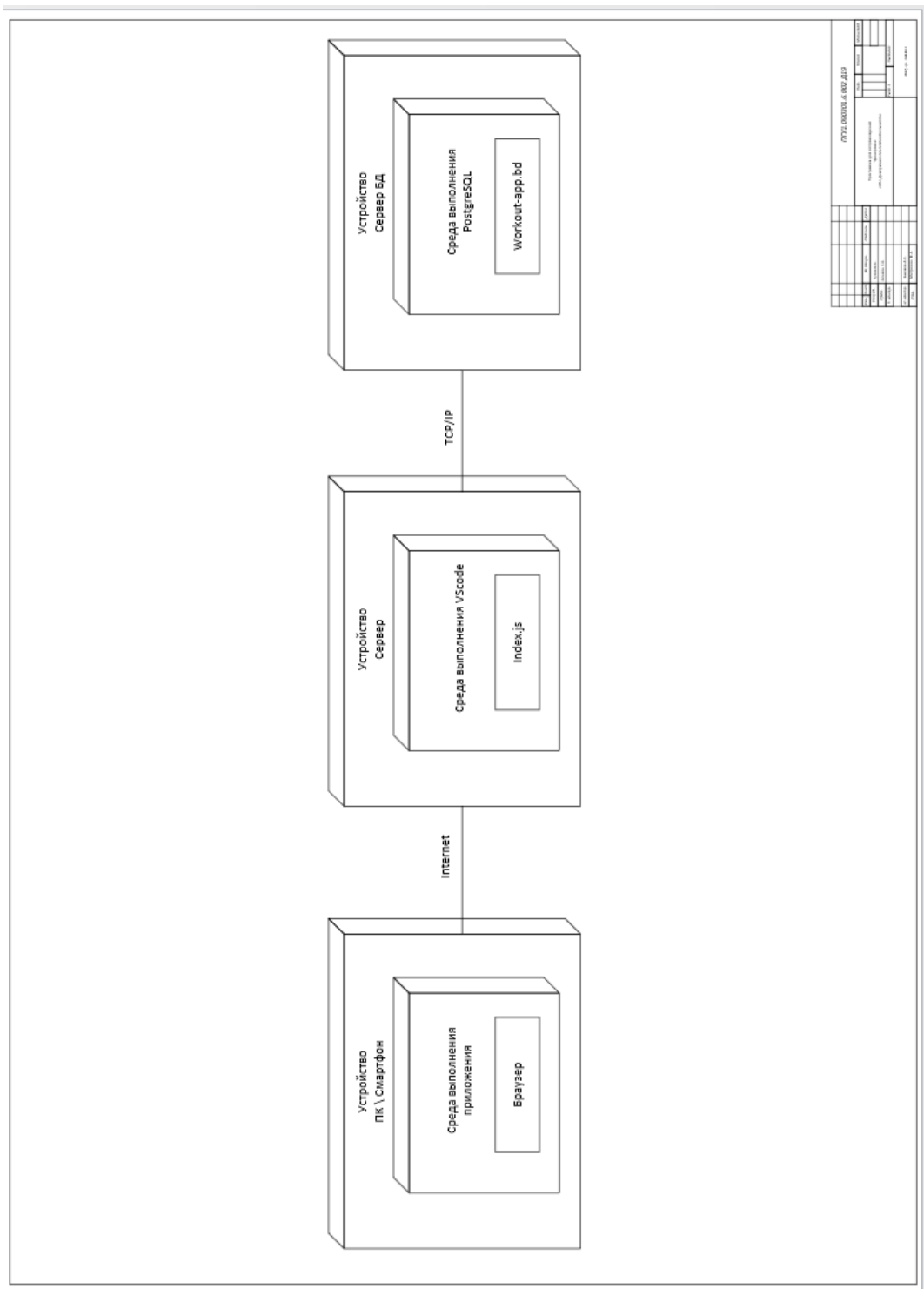
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.6.001 ПЗ

Приложение А.2 - Диаграмма последовательностей



Приложение А.3 - Диаграмма развертывания



Приложение Б

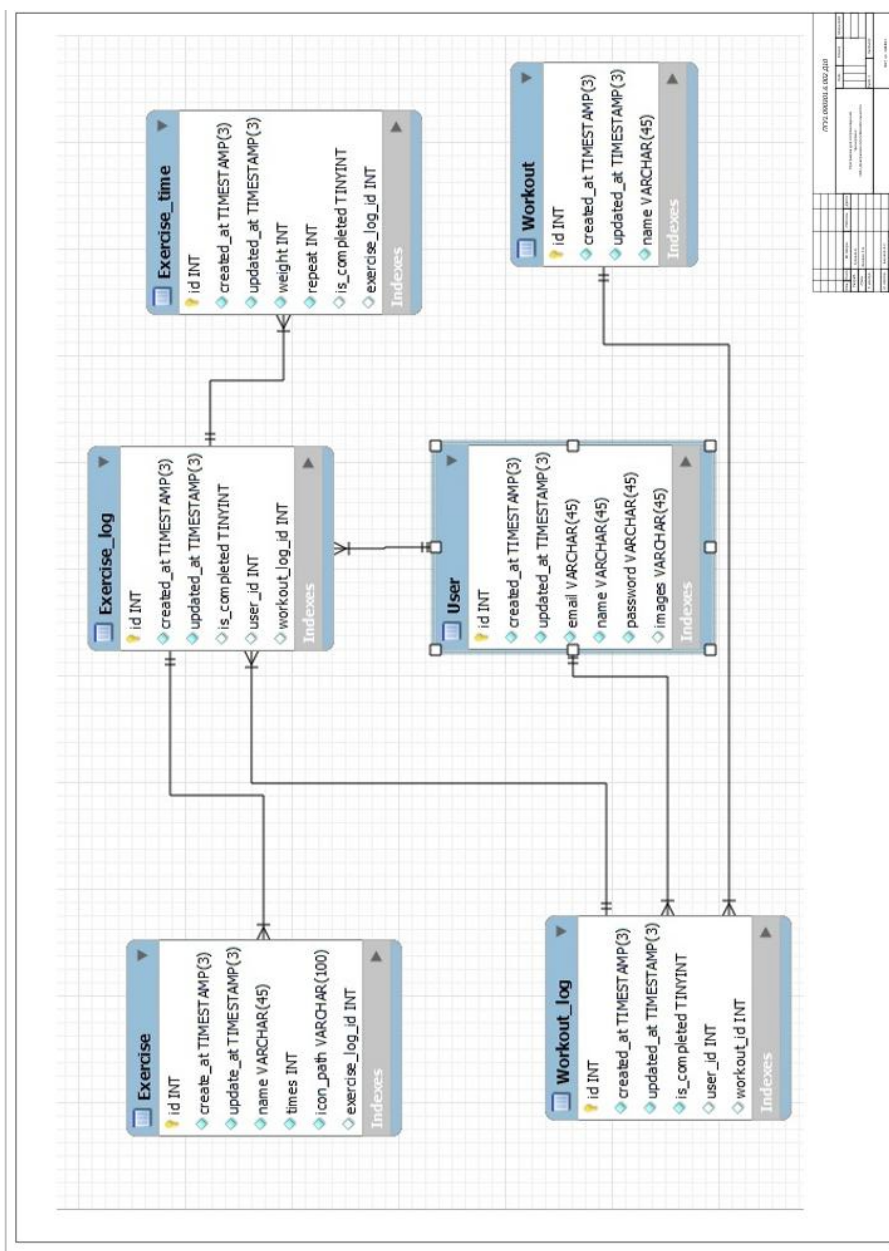
(Обязательное)

Разработка веб-приложения для сопровождения тренировок

Физическая модель базы данных

					ПГУ1.090301.6.001 ПЗ	Лист
						56
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Б.1 – Физическая модель базы данных



Приложение В

(Обязательное)

Разработка веб-приложения для сопровождения тренировок

Листинг программы

					ПГУ1.090301.6.001 ПЗ	Лист
						58
Изм.	Лист	№ докум.	Подпись	Дата		

Файл env.example

```
NODE_ENV = development
DATABASE_URL =
postgres://postgres:12345sqlp@localhost:5432/workout-v-
2?schema=public
JWT_SECRET = wefwe23F
```

Файл shema.prisma

```
// This is your Prisma schema file,
// learn more about it in the docs: https://pris.ly/d/prisma-schema

generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

model User {
  id          Int          @id @default(autoincrement())
  createdAt   DateTime      @default(now()) @map("created_at")
  updatedAt   DateTime      @updatedAt @map("updated_at")

  email       String        @unique
  name        String
  password    String
  images      String[]

  exerciseLogs ExerciseLog[]
  workoutLogs  WorkoutLog[]
}

model Workout {
  id          Int          @id @default(autoincrement())
  createdAt   DateTime      @default(now()) @map("created_at")
  updatedAt   DateTime      @updatedAt @map("updated_at")

  name        String
  exercises   Exercise[]
  workoutLogs WorkoutLog[]
}

model Exercise {
  id          Int          @id @default(autoincrement())
  createdAt   DateTime      @default(now()) @map("created_at")
```

```

    updatedAt DateTime @updatedAt @map("updated_at")

    name      String
    times     Int
    iconPath  String @map("icon_path")

    workouts Workout[]

    exerciseLogs ExerciseLog[]
}

model ExerciseLog {
  id          Int          @id @default(autoincrement())
  createdAt   DateTime @default(now()) @map("created_at")
  updatedAt   DateTime @updatedAt @map("updated_at")

  isCompleted Boolean @default(false)
  @map("is_completed")
  times        ExerciseTime[]

  user      User? @relation(fields: [userId], references: [id])
  userId    Int?  @map("user_id")

  workoutLog WorkoutLog? @relation(fields: [workoutLogId],
  references: [id])
  workoutLogId Int?      @map("workout_log_id")

  exercise    Exercise? @relation(fields: [exerciseId],
  references: [id])
  exerciseId  Int?      @map("exercise_id")

  @@map("Exercise_log")
}

model ExerciseTime {
  id          Int          @id @default(autoincrement())
  createdAt   DateTime @default(now()) @map("created_at")
  updatedAt   DateTime @updatedAt @map("updated_at")

  weight      Int          @default(0)
  repeat      Int          @default(0)
  isCompleted Boolean @default(false) @map("is_completed")

  exerciseLog ExerciseLog? @relation(fields: [exerciseLogId],
  references: [id])
  exerciseLogId Int?      @map("exercise_log_id")

  @@map("Exercise_time")
}

```

```

model WorkoutLog {
  id          Int          @id @default(autoincrement())
  createdAt   DateTime @default(now()) @map("created_at")
  updatedAt   DateTime @updatedAt @map("updated_at")

  isCompleted Boolean @default(false) @map("is_completed")

  exerciseLogs ExerciseLog[]

  user      User? @relation(fields: [userId], references: [id])
  userId    Int?  @map("user_id")

  workout    Workout? @relation(fields: [workoutId], references:
[id])
  workoutId  Int?      @map("workout_id")

  @@map("Workout_log")
}

```

Файл workout.router.js

```

import express from 'express'

import { protect } from '../middleware/auth.middleware.js'

import { getWorkoutLog } from './log/get-workout-
log.controller.js'
import { updateCompleteWorkoutLog } from './log/update-workout-
log.controller.js'
import { createNewWorkoutLog } from './log/workout-
log.controller.js'
import {
  createNewWorkout,
  deleteWorkout,
  getWorkout,
  getWorkouts,
  updateWorkout
} from './workout.controller.js'

const router = express.Router()

router.route('/').post(protect, createNewWorkout).get(protect,
getWorkouts)

router
  .route('/:id')
  .get(protect, getWorkout)
  .put(protect, updateWorkout)
  .delete(protect, deleteWorkout)

router

```

```

    .route('/log/:id')
    .post(protect, createNewWorkoutLog)
    .get(protect, getWorkoutLog)

router.route('/log/complete/:id').patch(protect,
updateCompleteWorkoutLog)

export default router

```

файл workout.controller.js

```

import asyncHandler from 'express-async-handler'

import { prisma } from '../prisma.js'

import { calculateMinute } from './calculate-minute.js'

// @desc    Get workouts
// @route    GET /api/workouts
// @access   Private
export const getWorkouts = asyncHandler(async (req, res) => {
  const workouts = await prisma.workout.findMany({
    orderBy: {
      createdAt: 'desc'
    },
    include: {
      exercises: true
    }
  })

  res.json(workouts)
})

// @desc    Get workout
// @route    GET /api/workouts/:id
// @access   Private
export const getWorkout = asyncHandler(async (req, res) => {
  const workout = await prisma.workout.findUnique({
    where: { id: +req.params.id },
    include: {
      exercises: true
    }
  })

  if (!workout) {
    res.status(404)
    throw new Error('Workout not found!')
  }

  const minutes = calculateMinute(workout.exercises.length)

```

					ПГУ1.090301.6.001 ПЗ	Лист
						62
Изм.	Лист	№ докум.	Подпись	Дата		

```

        res.json({ ...workout, minutes })
    })

    // @desc    Create new workout
    // @route    POST /api/workouts
    // @access    Private
    export const createNewWorkout = asyncHandler(async (req, res) => {
        const { name, exerciseIds } = req.body

        const workout = await prisma.workout.create({
            data: {
                name,
                exercises: {
                    connect: exerciseIds.map(id => ({ id: +id }))
                }
            }
        })

        res.json(workout)
    })

    // @desc    Update workout
    // @route    PUT /api/workouts/:id
    // @access    Private
    export const updateWorkout = asyncHandler(async (req, res) => {
        const { name, exerciseIds } = req.body

        try {
            const workout = await prisma.workout.update({
                where: {
                    id: +req.params.id
                },
                data: {
                    name,
                    exercises: {
                        set: exerciseIds.map(id => ({ id: +id }))
                    }
                }
            })

            res.json(workout)
        } catch (error) {
            res.status(404)
            throw new Error('Workout not found!')
        }
    })

    // @desc    Delete workout
    // @route    DELETE /api/workouts/:id

```

```
// @access Private
export const deleteWorkout = asyncHandler(async (req, res) => {
  try {
    const workout = await prisma.workout.delete({
      where: {
        id: +req.params.id
      }
    })

    res.json({ message: 'Workout deleted!' })
  } catch (error) {
    res.status(404)
    throw new Error('Workout not found!')
  }
})
```

файл workoutlog-controller.js

```
import asyncHandler from 'express-async-handler'

import { prisma } from '../prisma.js'

// @desc    Create new workout log
// @route    POST /api/workouts/log/:id
// @access   Private
export const createNewWorkoutLog = asyncHandler(async (req, res)
=> {
  const workoutId = +req.params.id

  const workout = await prisma.workout.findUnique({
    where: {
      id: workoutId
    },

    include: {
      exercises: true
    }
  })

  if (!workout) {
    res.status(404)
    throw new Error('Workout not found!')
  }

  const workoutLog = await prisma.workoutLog.create({
    data: {
      user: {
        connect: {
          id: req.user.id
        }
      },
    },
  })
```

```

        workout: {
          connect: {
            id: workoutId
          }
        },
        exerciseLogs: {
          create: workout.exercises.map(exercise => ({
            user: {
              connect: {
                id: req.user.id
              }
            },
            exercise: {
              connect: {
                id: exercise.id
              }
            },
            times: {
              create: Array.from({ length:
exercise.times }, () => ({
              weight: 0,
              repeat: 0
            })))
          }
        })))
      },
      include: {
        exerciseLogs: {
          include: {
            times: true
          }
        }
      }
    })

    res.json(workoutLog)
  })

```

Файл user.controller.js

```

import asyncHandler from 'express-async-handler'

import { prisma } from '../prisma.js'
import { UserFields } from '../utils/user.utils.js'

// @desc    Get user profile
// @route    GET /api/users/profile
// @access   Private
export const getUserProfile = asyncHandler(async (req, res) => {
  const user = await prisma.user.findUnique({

```

```

        where: {
          id: req.user.id
        },
        select: UserFields
      })

      const countExerciseTimesCompleted = await
prisma.exerciseLog.count({
      where: {
        userId: req.user.id,
        isCompleted: true
      }
    })

const kgs = await prisma.exerciseTime.aggregate({
      where: {
        exerciseLog: {
          userId: req.user.id
        },
        isCompleted: true
      },

      _sum: {
        weight: true
      }
    })

const workouts = await prisma.workoutLog.count({
      where: {
        userId: user.id,
        isCompleted: true
      }
    })

res.json({
  ...user,
  statistics: [
    {
      label: 'Минуты',
      value: Math.ceil(countExerciseTimesCompleted *
2.3) || 0
    },
    {
      label: 'Тренировки',
      value: workouts
    },
    {
      label: 'Кг',
      value: kgs._sum.weight || 0
    }
  ]
})

```

```

    ]
  })
})

```

Файл auth.middleware.js

```

import asyncHandler from 'express-async-handler'
import jwt from 'jsonwebtoken'

import { prisma } from '../prisma.js'
import { UserFields } from '../utils/user.utils.js'

export const protect = asyncHandler(async (req, res, next) => {
  let token

  if (req.headers.authorization?.startsWith('Bearer')) {
    token = req.headers.authorization.split(' ')[1]

    const decoded = jwt.verify(token,
process.env.JWT_SECRET)

    const userFound = await prisma.user.findUnique({
      where: {
        id: decoded.userId
      },
      select: UserFields
    })

    if (userFound) {
      req.user = userFound
      next()
    } else {
      res.status(401)
      throw new Error('Not authorized, token failed')
    }
  }

  if (!token) {
    res.status(401)
    throw new Error('Not authorized, I do not have a token')
  }
})

```

Файл exercise.controller.js

```

import asyncHandler from 'express-async-handler'

import { prisma } from '../prisma.js'

// @desc    Create new exercise
// @route    POST /api/exercises

```

					ПГУ1.090301.6.001 ПЗ	Лист
						67
Изм.	Лист	№ докум.	Подпись	Дата		

```

// @access Private
export const createNewExercise = asyncHandler(async (req, res)
=> {
    const { name, times, iconPath } = req.body

    const exercise = await prisma.exercise.create({
        data: {
            name,
            times,
            iconPath
        }
    })

    res.json(exercise)
})

// @desc    Update exercise
// @route    PUT /api/exercises/:id
// @access   Private
export const updateExercise = asyncHandler(async (req, res) => {
    const { name, times, iconPath } = req.body

    try {
        const exercise = await prisma.exercise.update({
            where: {
                id: +req.params.id
            },
            data: {
                name,
                times,
                iconPath
            }
        })

        res.json(exercise)
    } catch (error) {
        res.status(404)
        throw new Error('Exercise not found!')
    }
})

// @desc    Delete exercise
// @route    DELETE /api/exercises/:id
// @access   Private
export const deleteExercise = asyncHandler(async (req, res) => {
    try {
        const exercise = await prisma.exercise.delete({
            where: {
                id: +req.params.id
            }
        })
    }
})

```

```

    })

    res.json({ message: 'Exercise deleted!' })
  } catch (error) {
    res.status(404)
    throw new Error('Exercise not found!')
  }
})

// @desc    Get exercises
// @route    GET /api/exercises
// @access   Private
export const getExercises = asyncHandler(async (req, res) => {
  const exercises = await prisma.exercise.findMany({
    orderBy: {
      createdAt: 'desc'
    }
  })
  res.json(exercises)
})

```

Файл update-exercise-log.controller.js

```

import asyncHandler from 'express-async-handler'

import { prisma } from '../prisma.js'

// @desc    Update exercise log time
// @route    PUT /api/exercises/log/time/:id
// @access   Private
export const updateExerciseLogTime = asyncHandler(async (req, res) => {
  const { weight, repeat, isCompleted } = req.body

  try {
    const exerciseLogTime = await
prisma.exerciseTime.update({
      where: {
        id: +req.params.id
      },
      data: {
        weight,
        repeat,
        isCompleted
      }
    })

    res.json(exerciseLogTime)
  } catch (error) {
    res.status(404)
    throw new Error('Exercise log time not found!')
  }
})

```

```

    }
  })

  // @desc    Update status of complete exercise log
  // @route    PATCH /api/exercises/log/complete/:id
  // @access   Private
  export const completeExerciseLog = asyncHandler(async (req, res)
=> {
    const { isCompleted } = req.body

    try {
      const exerciseLog = await prisma.exerciseLog.update({
        where: {
          id: +req.params.id
        },
        data: {
          isCompleted
        },
        include: { exercise: true, workoutLog: true }
      })

      res.json(exerciseLog)
    } catch (error) {
      res.status(404)
      throw new Error('Exercise log not found!')
    }
  })
})

```

Файл get-exercise-log.controller.js

```

import asyncHandler from 'express-async-handler'

import { prisma } from '../../prisma.js'

import { addPrevValues } from './add-prev-values.util.js'

// @desc    Get exerciseLog
// @route    GET /api/exercises/log/:id
// @access   Private
export const getExerciseLog = asyncHandler(async (req, res) => {
  const exerciseLog = await prisma.exerciseLog.findUnique({
    where: {
      id: +req.params.id
    },
    include: {
      exercise: true,
      times: {
        orderBy: {
          id: 'asc'
        }
      }
    }
  })
})

```

					ПГУ1.090301.6.001 ПЗ	Лист
						70
Изм.	Лист	№ докум.	Подпись	Дата		

```

    }
  })

  if (!exerciseLog) {
    res.status(404)
    throw new Error('Exercise Log not found!')
  }

  const prevExerciseLog = await prisma.exerciseLog.findFirst({
    where: {
      exerciseId: exerciseLog.exerciseId,
      userId: req.user.id,
      isCompleted: true
    },
    orderBy: {
      createdAt: 'desc'
    },
    include: {
      times: true
    }
  })

  res.json({
    ...exerciseLog,
    times: addPrevValues(exerciseLog, prevExerciseLog)
  })
})

```

Файл exercise-log.controller.js

```

import asyncHandler from 'express-async-handler'

import { prisma } from '../prisma.js'

// @desc    Create new exerciseLog
// @route    POST /api/exercises/log/:exerciseId
// @access   Private
export const createNewExerciseLog = asyncHandler(async (req, res) => {
  const exerciseId = +req.params.id

  const exercise = await prisma.exercise.findUnique({
    where: {
      id: exerciseId
    }
  })

  if (!exercise) {
    res.status(404)
  }

```

					ПГУ1.090301.6.001 ПЗ	Лист
						71
Изм.	Лист	№ докум.	Подпись	Дата		

```

        throw new Error('Exercise not found!')
    }

    let timesDefault = []

    for (let i = 0; i < exercise.times; i++) {
        timesDefault.push({
            weight: 0,
            repeat: 0
        })
    }

    const exerciseLog = await prisma.exerciseLog.create({
        data: {
            user: {
                connect: {
                    id: req.user.id
                }
            },
            exercise: {
                connect: {
                    id: exerciseId
                }
            },
            times: {
                createMany: {
                    data: timesDefault
                }
            }
        },
        include: {
            times: true
        }
    })

    res.json(exerciseLog)
})

```

Файл auth.controller.js

```

import { faker } from '@faker-js/faker'
import { hash, verify } from 'argon2'
import asyncHandler from 'express-async-handler'

import { prisma } from '../prisma.js'
import { UserFields } from '../utils/user.utils.js'

import { generateToken } from './generate-token.js'

// @desc    Auth user
// @route    POST /api/auth/login

```

					ПГУ1.090301.6.001 ПЗ	Лист
						72
Изм.	Лист	№ докум.	Подпись	Дата		

```

// @access Public
export const authUser = asyncHandler(async (req, res) => {
  const { email, password } = req.body

  const user = await prisma.user.findUnique({
    where: {
      email
    }
  })

  const isValidPassword = await verify(user.password,
password)

  if (user && isValidPassword) {
    const token = generateToken(user.id)
    res.json({ user, token })
  } else {
    res.status(401)
    throw new Error('Email and password are not correct')
  }
})

// @desc Register user
// @route POST /api/auth/register
// @access Public
export const registerUser = asyncHandler(async (req, res) => {
  const { email, password } = req.body

  const isHaveUser = await prisma.user.findUnique({
    where: {
      email
    }
  })

  if (isHaveUser) {
    res.status(400)
    throw new Error('User already exists')
  }

  const user = await prisma.user.create({
    data: {
      email,
      password: await hash(password),
      name: faker.name.fullName(),
      images: ['/images/before.jpg', '/images/after.jpg']
    },
    select: UserFields
  })

  const token = generateToken(user.id)

```

```

    res.json({ user, token })
  })
}

```

Файл server.js

```

import 'colors'
import cors from 'cors'
import dotenv from 'dotenv'
import express from 'express'
import morgan from 'morgan'
import path from 'path'

import { errorHandler, notFound } from
'./app/middleware/error.middleware.js'

import authRoutes from './app/auth/auth.routes.js'
import exerciseRoutes from './app/exercise/exercise.routes.js'
import { prisma } from './app/prisma.js'
import userRoutes from './app/user/user.routes.js'
import workoutRoutes from './app/workout/workout.routes.js'

// [] - Add cors (app.use(cors()) and $ npm install cors)
// getUserProfile
// get Workout log and get exercise log exercise

dotenv.config()

const app = express()

async function main() {
  if (process.env.NODE_ENV === 'development')
    app.use(morgan('dev'))

  app.use(cors())
  app.use(express.json())

  const __dirname = path.resolve()

  app.use('/uploads', express.static(path.join(__dirname,
'uploads/')))

  app.use('/api/auth', authRoutes)
  app.use('/api/users', userRoutes)
  app.use('/api/exercises', exerciseRoutes)
  app.use('/api/workouts', workoutRoutes)

  app.use(notFound)
  app.use(errorHandler)
}

```

```

const PORT = process.env.PORT || 5000

app.listen(
  PORT,
  console.log(
    `🚀 Server running in ${process.env.NODE_ENV} mode
on port ${PORT}`.blue
    .bold
  )
)
}

main()
  .then(async () => {
    await prisma.$disconnect()
  })
  .catch(async e => {
    console.error(e)
    await prisma.$disconnect()
    process.exit(1)
  })

```

Файл Auth.jsx

```

import Layout from "../../layout/Layout";
import Button from "../../ui/button/Button";
import Field from "../../ui/field/Field";
import Loader from "../../ui/Loader";
import styles from "./Auth.module.scss";
import { useAuthPage } from "./useAuthPage";

const Auth = () => {
  const { setType, register, handleSubmit, errors, isLoading,
onSubmit } =
    useAuthPage();

  return (
    <>
      <Layout heading="Вход" backgroundImage="/images/auth-bg.png" />
      <div className="wrapper-inner-page">
        {isLoading && <Loader />}
        <form onSubmit={handleSubmit(onSubmit)}>
          <Field
            error={errors?.email?.message}
            name="email"
            register={register}
            options={{
              required: "Email обязательный",

```

```

    }}
    type="text"
    placeholder="Введите email"
  />
  <Field
    error={errors?.password?.message}
    name="password"
    register={register}
    options={{
      required: "пароль обязательный",
    }}
    type="password"
    placeholder="введите пароль"
  />
  <div className={styles.wrapperButtons}>
    <Button clickHandler={() =>
setType("login")}>Войти</Button>
    <Button clickHandler={() => setType("register")}>
      Регистрация
    </Button>
  </div>
</form>
</div>
</>
);
};
export default Auth;

```

файл useAuthPage.js

```

import { useMemo } from "react";
import { useMutation } from "@tanstack/react-query";
import AuthService from "../../services/auth.service";
import { useAuth } from "../../Hooks/useAuth";
import { useNavigate } from "react-router-dom";
import { useForm } from "react-hook-form";
import { useEffect, useState } from "react";
export const useAuthPage = () => {
  const [type, setType] = useState("login");
  const {
    register,
    handleSubmit,
    formState: { errors },
    reset,
  } = useForm({
    mode: "onChange",
  });

  const { isAuth, setIsAuth } = useAuth();

```

```

const navigate = useNavigate();

useEffect(() => {
  if (isAuth) navigate("/");
}, [isAuth]);

const { mutate, isLoading } = useMutation(
  ["auth"],
  ({ email, password }) => AuthService.main(email, password,
type),
  {
    onSuccess: () => {
      setIsAuth(true);
      reset();
    },
  }
);

const onSubmit = (data) => {
  mutate(data);
};

return useMemo(
  () => ({
    setType,
    register,
    handleSubmit,
    errors,
    isLoading,
    onSubmit,
  })),
  [errors, isLoading]
);
};

```

Файл useExerciseLog.js

```

import { useQuery } from "@tanstack/react-query";
import { useParams } from "react-router-dom";
import ExerciseLogService from
"../../../../../services/exercise/exercise-log.service";
import { useState } from "react";
import { useUpdateLogTime } from "../useUpdateLogTime";
export const useExerciseLog = () => {
  const { id } = useParams();

  const [times, setTimes] = useState([]);

  const {
    data: exerciseLog,
    isSuccess,

```

					ПГУ1.090301.6.001 ПЗ	Лист
						77
Изм.	Лист	№ докум.	Подпись	Дата		

```

        isLoading,
        } = useQuery(["get exercise log", id], () =>
ExerciseLogService.getById(id), {
    select: ({ data }) => data,
    onSuccess(data) {
        if (data?.times?.length) setTimes(data.times);
    },
});

const { error, updateTime } =
useUpdateLogTime(exerciseLog?.times);

const onChangeState = (timeId, key, value) => {
    const newTimes = times.map((time) => {
        if (time.id === timeId) {
            return { ...time, [key]: value };
        }

        return time;
    });

    setTimes(newTimes);
};

const getTime = (timeId) => {
    return times.find((time) => time.id === timeId);
};

const getState = (timeId, key) => {
    const time = getTime(timeId);
    return time ? time[key] : key === "isCompleted" ? false : 0;
};

const toggleTime = (timeId, isCompleted) => {
    const time = getTime(timeId);
    updateTime({
        timeId,
        body: {
            isCompleted,
            repeat: +time.repeat,
            weight: +time.weight,
        },
    });
};

return {
    exerciseLog,
    isSuccess,
    isLoading,
    toggleTime,

```

```

    error,
    onChangeState,
    getState,
  };
};

```

Файл useUpdateLogTime.js

```

import { useMutation, useQueryClient } from "@tanstack/react-
query";
import ExerciseLogService from
"../../../../../services/exercise/exercise-log.service";
import { useParams } from "react-router-dom";
import { useCompleteLog } from "../useCompleteLog";

export const useUpdateLogTime = (times) => {
  const { id } = useParams();

  const queryClient = useQueryClient();

  const { completeLog, errorCompleted } = useCompleteLog();

  const { mutate, error: errorChange } = useMutation(
    ["update log time"],
    ({ timeId, body }) => ExerciseLogService.updateTime(timeId,
body),
    {
      onSuccess: () => {
        queryClient.invalidateQueries(["get exercise log",
id]).then(() => {
          const filteredTimes = times.filter((time) =>
time.isCompleted);

          if (filteredTimes.length === times.length - 1) {
            completeLog({ isCompleted: true });
          }
        });
      },
    }
  );

  return { updateTime: mutate, error: errorChange ||
errorCompleted };
};

```

Файл TableRow.jsx

```

import cn from "clsx";
import styles from "../../ExerciseLog.module.scss";

const TableRow = ({ item, getState, onChangeState, toggleTime })
=> {

```

```

return (
  <div
    className={cn(styles.row, {
      [styles.completed]: item.isCompleted,
    })}
    key={`time${item.id}`}
  >
    <div
      className={styles.opacity}
      key={`Prev ${item.id}/${item.prevWeight}`}
    >
      <input type="number" defaultValue={item.prevWeight}
disabled />
      <i>kg{item.isCompleted && " "}/</i>
      <input type="number" defaultValue={item.prevRepeat}
disabled />
    </div>

    <div key={`RepeatWeight ${item.id}/${item.weight}`}>
      <input
        type="tel"
        pattern="[0-9]*"
        value={getState(item.id, "weight")}
        disabled={item.isCompleted}
        onChange={(e) => onChangeState(item.id, "weight",
e.target.value)}
      />
      <i>kg{item.isCompleted && " "}/</i>
      <input
        type="number"
        value={getState(item.id, "repeat")}
        disabled={item.isCompleted}
        onChange={(e) => onChangeState(item.id, "repeat",
e.target.value)}
      />
    </div>

    <div key={`Completed ${item.id}/${item.isCompleted}`}>
      <img
        className={styles.checkbox}
        src={
          getState(item.id, "isCompleted")
            ? "/images/exercises/check-completed.svg"
            : "/images/exercises/check.svg"
        }
        alt=""
        onClick={() => {
          toggleTime(item.id, !getState(item.id,
"isCompleted"))};
        }}
      />
    </div>
  </div>
)

```

```

        />
      </div>
    </div>
  );
};

export default TableRow;

```

Файл ExerciseLog.jsx

```

import styles from "../ExerciseLog.module.scss";
import ExerciseError from "../ExerciseError";

import HeaderExerciseLog from "../HeaderExerciseLog";
import Alert from "../../ui/alert/Alert";
import Loader from "../../ui/Loader";
import { useExerciseLog } from "../hooks/useExerciseLog";

import TableHeader from "../table/TableHeader";
import TableRow from "../table/TableRow";

const ExerciseLog = () => {
  const {
    exerciseLog,
    isSuccess,
    isLoading,
    getState,
    onChangeState,
    toggleTime,
    error,
  } = useExerciseLog();

  return (
    <>
      <HeaderExerciseLog exerciseLog={exerciseLog}
        isSuccess={isSuccess} />

      <div
        className="wrapper-inner-page"
        style={{ paddingLeft: 0, paddingRight: 0 }}
      >
        <ExerciseError errors={[error]} />
        {isLoading ? (
          <Loader />
        ) : (
          <div className={styles.wrapper}>
            <div className={styles.row}>
              <TableHeader />
            </div>
            {exerciseLog?.times.map((item) => (
              <TableRow

```

```

        getState={getState}
        onChangeState={onChangeState}
        toggleTime={toggleTime}
        item={item}
        key={item.id}
      />
    )))
  </div>
)}

{isSuccess && exerciseLog?.times?.length === 0 && (
  <Alert type="warning" text={"Times not found"} />
)}
</div>
</>
);
};

export default ExerciseLog;

```

файл HeaderExerciseLog.jsx

```

import styles from "../ExerciseLog.module.scss";
import cn from "clsx";
import Header from "../../layout/header/Header";
import stylesLayout from "../../layout/Layout.module.scss";

const HeaderExerciseLog = ({ isSuccess, exerciseLog }) => {
  return (
    <div
      className={cn(stylesLayout.wrapper,
        stylesLayout.otherPage)}
      style={{
        backgroundImage: `url('/images/ex-bg-1.jpg')`,
        height: 300,
      }}
    >
      <Header
        backLink={
          isSuccess ? `/workout/${exerciseLog.workoutLogId}` :
          "/workouts"
        }
      />

      {isSuccess && (
        <div className={styles.heading}>
          <img

src={`http://localhost:5000${exerciseLog.exercise.iconPath}`}
          height="34"
          draggable={false}

```

```

        alt=""
      />
      <h1
className={stylesLayout.heading}>{exerciseLog.exercise.name}</h1
>
    </div>
  )}
</div>
);
};

export default HeaderExerciseLog;

```

файл NewExercise.jsx

```

import { useMutation } from "@tanstack/react-query";
import cn from "clsx";
import ExerciseService from
"../../services/exercise/exercise.service";
import Layout from "../../layout/Layout";
import { useForm, Controller } from "react-hook-form";
import Loader from "../../ui/Loader";
import styles from "./NewExercise.module.scss";
import Button from "../../ui/button/Button";
import { getImagePath } from "./image-path.utils";
import Field from "../../ui/field/Field";
import Alert from "../../ui/alert/Alert";

const data = ["chest", "shoulders", "biceps", "legs", "hit",
"back"];

const NewExercise = () => {
  const {
    register,
    handleSubmit,
    formState: { errors },
    reset,
    control,
  } = useForm({
    mode: "onChange",
  });

  const { isSuccess, error, isLoading, mutate } = useMutation(
    ["create exercise"],
    (body) => ExerciseService.create(body),
    {
      onSuccess: () => {
        reset();
      },
    },
  );
};

```

```

const onSubmit = (data) => {
  mutate(data);
};

return (
  <>
    <Layout
      bgImage="/images/new-exercise-bg.jpg"
      heading="Создать новое упражнение"
      backLink="/new-workout"
    />
    <div className="wrapper-inner-page">
      {error && <Alert type="error" text={error} />}
      {isSuccess && <Alert text="Exercise created" />}
      {isLoading && <Loader />}
      <form onSubmit={handleSubmit(onSubmit)}>
        <Field
          error={errors?.name?.message}
          name="name"
          register={register}
          options={{
            required: "Имя обязательно",
          }}
          type="text"
          placeholder="Введите имя"
        />

        <Field
          error={errors?.times?.message}
          name="times"
          register={register}
          options={{
            valueAsNumber: true,
            validate: (value) => value > 0 || "Times must be
number",
            required: "Подходы обязательны",
          }}
          placeholder="Введите количество подходов"
        />

        <Controller
          name="iconPath"
          control={control}
          render={({ field: { value, onChange } }) => (
            <div className={styles.images}>
              {data.map((name) => (
                <img
                  key={`ex img ${name}`}

```

```

src={`http://localhost:5000${getImagePath(name)}`}
      alt={name}
      className={cn({
        [styles.active]: value ===
getImagePath(name),
      })}
      onClick={() => onChange(getImagePath(name))}
      draggable={false}
      height="45"
    />
  ))}
</div>
)}
/>

{errors?.iconPath && (
  <div
className="error">{errors?.iconPath?.message}</div>
)}

  <Button>Создать</Button>
</form>
</div>
</>
);
};

export default NewExercise;

```

Файл NewWorkout.jsx

```

import Layout from "../../layout/Layout";
import Loader from "../../ui/Loader";
import Button from "../../ui/button/Button";
import Field from "../../ui/field/Field";
import Alert from "../../ui/alert/Alert";
import { useNewWorkout } from "./useNewWorkout";
import { Link } from "react-router-dom";
import SelectExercises from "./SelectExercises";

const NewWorkouts = () => {
  const {
    handleSubmit,
    errors,
    control,
    isSuccess,
    error,
    isLoading,
    onSubmit,
    register,

```

					ПГУ1.090301.6.001 ПЗ	Лист
						85
Изм.	Лист	№ докум.	Подпись	Дата		

```

    } = useNewWorkout();
    return (
      <>
        <Layout
          backgroundImage="/images/new-workout-bg.jpg"
          heading="Создать новую тренировку"
        />
        <div className="wrapper-inner-page">
          {error && <Alert type="error" text={error} />}
          {isSuccess && <Alert text="Тренировка создана успешно"
        />}

          {isLoading && <Loader />}
          <form onSubmit={handleSubmit(onSubmit)}>
            <Field
              error={errors?.name?.message}
              name="name"
              register={register}
              options={{
                required: "Имя обязательно",
              }}
              type="text"
              placeholder="Введите имя"
            />

            <Link to="/new-exercise" className="dark-link">
              Добавить новое упражнение
            </Link>

            <SelectExercises control={control} />

            {errors?.iconPath && (
              <div
                className="error">{errors?.iconPath?.message}</div>
            )}

            <Button>Создать</Button>
          </form>
        </div>
      </>
    );
  };
};

export default NewWorkouts;

```

Файл useNewWorkout.js

```

import { useMemo } from "react";
import WorkoutService from
  "../.../services/workout/workout.service";
import { useForm } from "react-hook-form";
import { useMutation } from "@tanstack/react-query";

```

```

export const useNewWorkout = () => {
  const {
    register,
    handleSubmit,
    formState: { errors },
    reset,
    control,
  } = useForm({
    mode: "onChange",
  });

  const { isSuccess, error, isLoading, mutate } = useMutation(
    ["create workout "],
    (body) => WorkoutService.create(body),
    {
      onSuccess: () => {
        reset({
          name: "",
          exerciseIds: [],
        });
      },
    }
  );

  const onSubmit = (data) => {
    mutate({
      name: data.name,
      exerciseIds: data.exerciseIds.map((ex) => ex.value),
    });
  };

  return useMemo(
    () => ({
      handleSubmit,
      errors,
      control,
      isSuccess,
      error,
      isLoading,
      onSubmit,
      register,
    }),
    [errors, isSuccess, error, isLoading]
  );
};

```

Файл Profile.jsx

```

import Layout from "../../layout/Layout";
import { useProfile } from "./useProfile";

```

					ПГУ1.090301.6.001 ПЗ	Лист
						87
Изм.	Лист	№ докум.	Подпись	Дата		

```

import stylesLayout from "../../layout/Layout.module.scss";
import styles from "../Profile.module.scss";
import Header from "../../layout/header/Header";
import Loader from "../../ui/Loader";
import Counter from "../statistics/Counter";

const Profile = () => {
  const { data, isLoading } = useProfile();
  return (
    <>
      <div
        className={` ${stylesLayout.wrapper}
        ${stylesLayout.otherPage} `}
        style={{
          backgroundImage: `url(../images/profile-bg.jpg)`,
          height: 356,
        }}
      >
        <Header />

        <div className={styles.center}>
          {isLoading ? (
            <Loader />
          ) : (
            <>
              

              <h1
                className={stylesLayout.heading}>{data?.email}</h1>
            </>
          )}
        </div>

        <Counter />
      </div>

      <div
        className="wrapper-inner-page"
        style={{ paddingLeft: 0, paddingRight: 0 }}
      >
        <div className={styles.before_after}>
          {data?.images?.map((image, index) => (
            <div key={image}>
              <div className={styles.heading}>
                {index === 1 ? "После" : "До"}
            </div>
          )}
        </div>
      </div>
    </>
  );
};

```

```

        </div>
        <img src={image} alt="" style={{ borderRadius: 15
    }} />
    </div>
  )}
</div>
</div>
</>
);
};
export default Profile;

```

Файл Workout.jsx

```

import { useMutation, useQuery } from "@tanstack/react-query";
import { useNavigate, useParams } from "react-router-dom";
import { Fragment } from "react";
import WorkoutLogService from
"../../../../../services/workout/workout-log.service.";
import HeaderWorkout from "../HeaderWorkout";
import ExerciseItem from "../ExerciseItem";
import Loader from "../../../../../ui/Loader";
import styles from "../Workout.module.scss";
import Button from "../../../../../ui/button/Button";

const Workout = () => {
  const { id } = useParams();
  const navigate = useNavigate();
  const { mutate } = useMutation(
    ["complete workout"],
    () => WorkoutLogService.complete(id),
    {
      onSuccess() {
        navigate("/workouts");
      },
    }
  );

  const {
    data: workoutLog,
    isSuccess,
    isLoading,
  } = useQuery(["get workout", id], () =>
    WorkoutLogService.getById(id), {
      select: ({ data }) => data,
    });

  return (
    <>
      <HeaderWorkout isSuccess={isSuccess}
        workoutLog={workoutLog} />
    </>
  );
};

```

```

    <div
      className="wrapper-inner-page"
      style={{ paddingLeft: 0, paddingRight: 0 }}
    >
      <div style={{ width: "90%", margin: "0 auto" }}>
        {/* {errorCompleted} && <Alert type='error' text=
{errorCompleted}/>} */}
      </div>
      {isLoading ? (
        <Loader />
      ) : (
        <div className={styles.wrapper}>
          {workoutLog?.exerciseLogs?.map((exerciseLog, index)
=> (
            <Fragment key={exerciseLog.id}>
              <ExerciseItem exerciseLog={exerciseLog} />
              {index % 2 === 0 &&
                index !== workoutLog.exerciseLogs.length - 1
&& (
                  <div className={styles.line}></div>
                )}
            </Fragment>
          )}}
        </div>
      )}

      <Button clickHandler={() => mutate()}> Завершить
тренировку</Button>
    </div>
  </>
);
};

export default Workout;

```

Файл useWorkouts.jsx

```

import { useMutation, useQuery } from "@tanstack/react-query";
import WorkoutService from
"../../../../../services/workout/workout.service";
import workoutLogService from
"../../../../../services/workout/workout-log.service.";
import { useNavigate } from "react-router-dom";

export const useWorkouts = () => {
  const { data, isSuccess } = useQuery(
    ["get workouts"],
    () => WorkoutService.getAll(),
    {
      select: ({ data }) => data,
    }
  );

```

```

);

const navigate = useNavigate();

const {
  mutate,
  isLoading,
  isSuccess: isSuccessMutate,
  error,
} = useMutation(
  ["create new workout log"],
  (workoutId) => workoutLogService.create(workoutId),
  {
    onSuccess({ data }) {
      navigate(`/workout/${data.id}`);
    },
  }
);

return {
  data,
  isSuccess,
  mutate,
  isLoading,
  isSuccessMutate,
  error,
};
};

```

Файл exercise-log-service.js

```

import { $axios } from "../api";
import { EXERCISES } from "../exercise.service";

const LOG = `${EXERCISES}/log`;

class ExerciseLogService {
  async getById(id) {
    return $axios.get(`${LOG}/${id}`);
  }

  async create(exerciseId) {
    return $axios.post(`${LOG}/${exerciseId}`);
  }

  //weight
  //repeat
  //isCompleted

  async updateTime(timeId, body) {

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.6.001 ПЗ

Лист

91

```

        return $axios.put(`${LOG}/time/${timeId}`, body);
    }

    //isCompleted : true/false
    async complete(id, body) {
        return $axios.patch(`${LOG}/complete/${id}`, body);
    }
}

export default new ExerciseLogService();

```

Файл workout-service.js

```

import { $axios } from "../api";

export const WORKOUTS = "/workouts";

class WorkoutsService {
    async getAll() {
        return $axios.get(WORKOUTS);
    }

    async getById(id) {
        return $axios.get(`${WORKOUTS}/${id}`);
    }

    //name, exerciseids
    async create(body) {
        return $axios.post(WORKOUTS, body);
    }

    async update(id, body) {
        return $axios.put(`${WORKOUTS}/${id}`, body);
    }

    async delete(id) {
        return $axios.delete(`${WORKOUTS}/${id}`);
    }
}

export default new WorkoutsService();

```

Файл auth-service.js

```

import Cookies from "js-cookie";

import { $axios } from "../api";
import { TOKEN } from "../app.constants";

class AuthService {
    async main(email, password, type) {

```

					ПГУ1.090301.6.001 ПЗ	Лист
						92
Изм.	Лист	№ докум.	Подпись	Дата		

```

try {
  const { data } = await $axios.post(`/auth/${type}`, {
    email,
    password,
  });

  if (data.token) Cookies.set(TOKEN, data.token);

  return data;
} catch (error) {
  throw new Error(error);
}
}

export default new AuthService();

```

					ПГУ1.090301.6.001 ПЗ	Лист
						93
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Г

(обязательное)

Приложение для сопровождения тренировок

Презентация

					ПГУ1.090301.6.001 ПЗ	Лист
						94
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение для сопровождения тренировок

Выполнил:

студент группы 19BBB1 Гусев Виталий

Научный руководитель:

Зинкин С.А.



Цель работы

Разработка веб-приложения для сопровождения тренировок

Назначение разработки

Разрабатываемое приложение должно быть предназначено для сопровождения тренировок, в качестве удобного места для хранения результата и создания тренировок.



					ПГУ1.090301.6.001 ПЗ	Лист
						95
Изм.	Лист	№ докум.	Подпись	Дата		

Функции системы

Приложение должно реализовывать такие функции как:

- регистрацию пользователя;
- вход пользователя по логину и паролю;
- создание упражнений;
- просмотр списка упражнений;
- возможность завершить упражнение;
- наличие истории упражнения;
- создание тренировки;
- добавление в тренировку готовых упражнений;
- возможность завершения тренировки;
- возможность просмотра списка готовых тренировок;
- обновление статистики о выполненных тренировках.



Схема базы данных

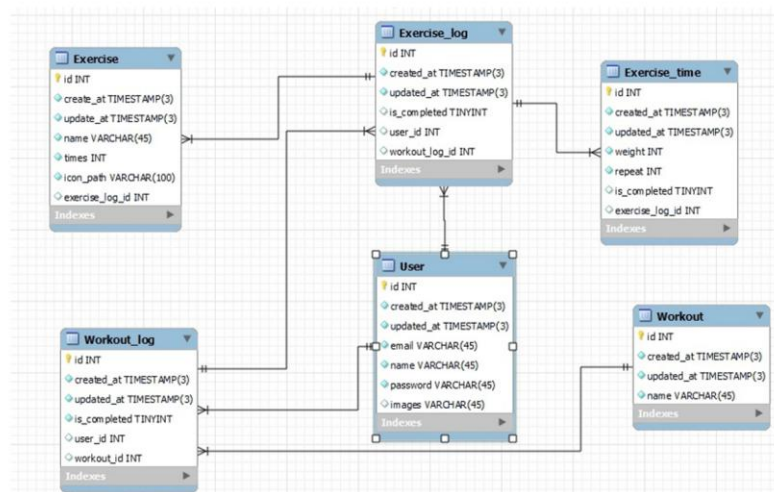
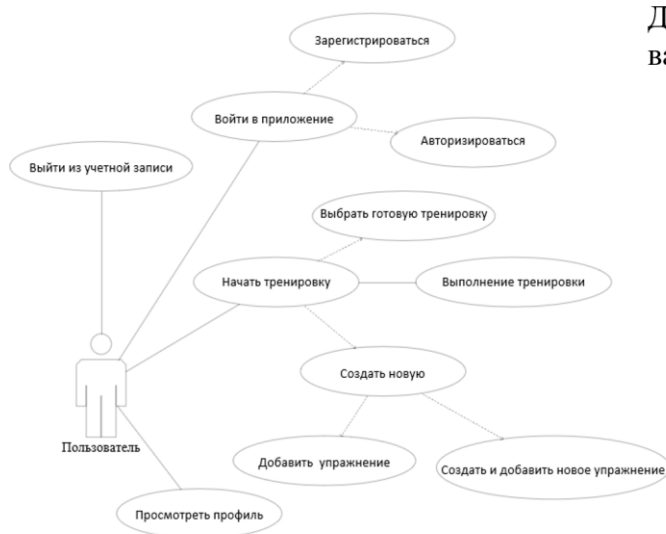


Диаграмма
вариантов использования



Основные
страницы сайта

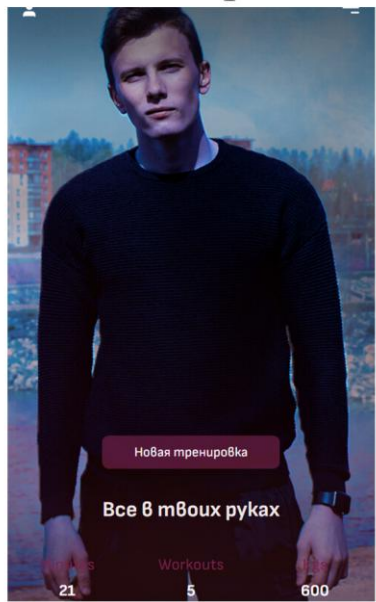
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.6.001 ПЗ

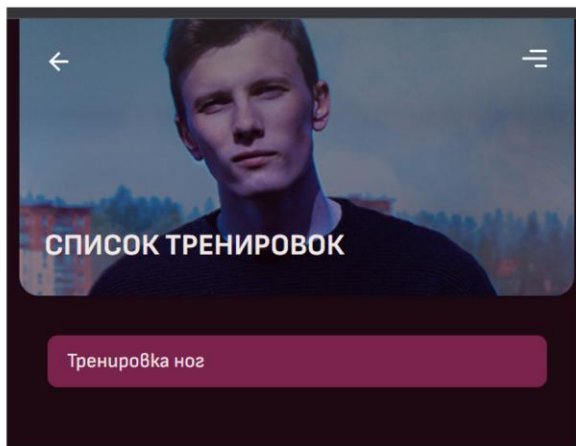
Лист

97

Главная страница



Просмотр тренировок



Создание тренировки



<

≡

СОЗДАТЬ НОВУЮ ТРЕНИРОВКУ

Тренировка ног

Добавить новое упражнение

Присед × × ▾

Создать



Создание нового упражнения



<

≡

СОЗДАТЬ НОВОЕ УПРАЖНЕНИЕ

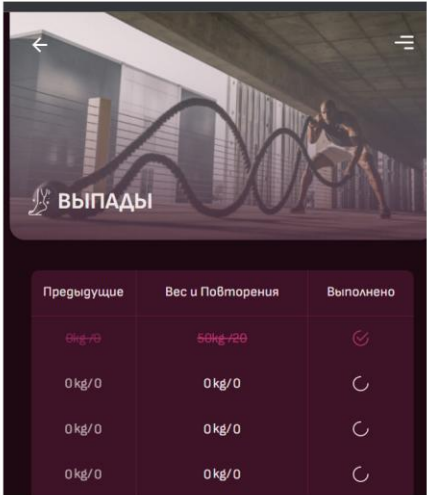
Отжимания

4

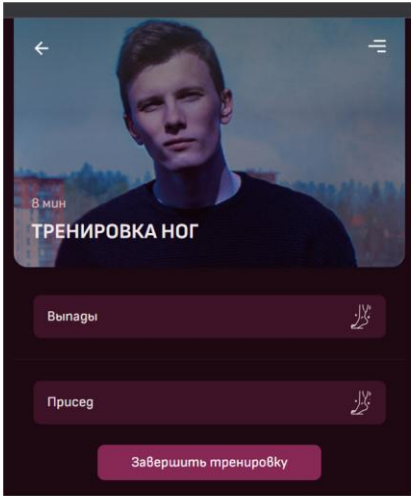
Создать



Страница упражнения с историей прошлого занятия



Страница завершения тренировки



Заклучение



- ✓ Выполнено проектирование
- ✓ Разработана база данных
- ✓ Разработано веб-приложение для сопровождения тренировок

В результате выполнения дипломного проекта было разработано приложение для сопровождения тренировок.

Было проведено тестирование приложения, в результате которого было выявлено, что сайт работает верно, разработано руководство пользователя.



Спасибо за внимание!



					ПГУ1.090301.6.001 ПЗ	Лист
						101
Изм.	Лист	№ докум.	Подпись	Дата		