**федеральное государственное бюджетное образовательное учреждение высшего образования «Казанский национальный исследовательский технический университет им. А.Н. Туполева-КАИ» (КНИТУ-КАИ)**

**ОТДЕЛЕНИЕ СРЕДНЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**Д И П Л О М Н Ы Й П Р О Е К Т**

**Тема: «**Веб-сайт для фитнес-центра с использованием ИИ для консультации и подбора персональных тренировок, а также составлением плана питания»

|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
| **Руководитель**  Преподаватель |  |  | Григорьева И.В. |
| *(должность)* | *(подпись)* |  | *(И.О. Фамилия)* |
| **Обучающийся**  4433 |  |  | Галеев Р.А. |
| *(группа)* | *(подпись)* |  | *(И.О. Фамилия)* |
| **Специальность** 09.02.07 «Информационные системы и программирование» | | | |
| (шифр и наименование специальности) | | | |

**Казань 2025 г.**

**ЛИСТ ЗАДАНИЯ**

**ЛИСТ ЗАДАНИЯ**

**АННОТАЦИЯ**

Дипломный проект на тему «Веб-сайт для фитнес-центра с использованием ИИ для консультации и подбора персональных тренировок, а также составлением плана питания» посвящён созданию информационной системы, направленной на автоматизацию предоставления фитнес-услуг и поддержку пользователей в достижении их целей в области здоровья и физической активности. Проект состоит из 136 страниц, содержит 25 рисунков и включает в себя 20 источников использованной литературы.

Основной задачей проекта является разработка веб-приложения, которое с использованием технологий искусственного интеллекта формирует индивидуальные тренировочные программы и планы питания, а также предоставляет консультации по физической активности. Приложение учитывает персональные данные пользователя, уровень физической подготовки, цели тренировок и предпочтения в питании.

Веб-сервис разработан с использованием современных технологий: Djаngо в качестве bаckеnd-фреймворка, Rеаct — для frоntеnd-части, PоstgrеSQL — в качестве системы управления базами данных. Для разработки и отладки проекта использовалась среда PyChаrm. Коммуникация между клиентской и серверной частью осуществляется через RЕST АPI.

Проект ориентирован на фитнес-центры и индивидуальных пользователей, стремящихся получать персонализированные рекомендации и эффективно управлять своим здоровьем и физической формой. Использование искусственного интеллекта позволяет повысить качество предоставляемых рекомендаций, вовлечённость пользователей, а также упростить взаимодействие с сервисом благодаря интеллектуальной поддержке.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 6](#_Toc200025075)

[ГЛАВА 1 ПОСТАНОВКА ЗАДАЧИ 8](#_Toc200025076)

[1.1 Цели и задачи 9](#_Toc200025077)

[1.2 Требования к функциональности системы 10](#_Toc200025078)

[1.3 Список требований к программному продукту 12](#_Toc200025079)

[1.4 Требования к интерфейсу приложения 13](#_Toc200025080)

[ГЛАВА 2 ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ 15](#_Toc200025081)

[2.1 Выбор средств реализации 16](#_Toc200025082)

[2.1.1 Языки программирования 16](#_Toc200025083)

[2.1.2 Среда разработки 17](#_Toc200025084)

[2.1.3 Система управления базами данных 17](#_Toc200025085)

[2.1.4 Фреймворки и библиотеки 18](#_Toc200025086)

[2.2 Проектирование диаграммы вариантов использования 18](#_Toc200025087)

[2.3 Проектирование диаграммы последовательности 21](#_Toc200025088)

[2.4 Проектирование диаграммы деятельности 23](#_Toc200025089)

[2.5 Проектирование диаграммы состояний 27](#_Toc200025090)

[2.6 Проектирование диаграммы сущность-связь 30](#_Toc200025091)

[2.7 Проектирование базы данных 34](#_Toc200025092)

[ГЛАВА 3 ПОЛЬЗОВАТЕЛЬСКАЯ ДОКУМЕНТАЦИЯ 37](#_Toc200025093)

[3.1 Руководство программиста 37](#_Toc200025094)

[3.2 Руководство пользователя 39](#_Toc200025095)

[3.2.1 Руководство роли «Пользователь» 40](#_Toc200025096)

[3.2.2 Руководство роли «Администратор» 45](#_Toc200025097)

[ГЛАВА 4 ТЕСТИРОВАНИЕ 48](#_Toc200025098)

[ЗАКЛЮЧЕНИЕ 54](#_Toc200025099)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 56](#_Toc200025100)

[ПРИЛОЖЕНИЕ А СЛОВАРЬ ДАННЫХ 58](#_Toc200025101)

[ПРИЛОЖЕНИЕ Б ЛИСТИНГ ПРОГРАММЫ 60](#_Toc200025102)

[ПРИЛОЖЕНИЕ В ПРЕЗЕНТАЦИЯ 132](#_Toc200025103)

**ВВЕДЕНИЕ**

В последние годы внедрение технологий в фитнес-индустрию стало важным шагом на пути к персонализации тренировок и улучшению качества услуг. В современном мире, где здоровый образ жизни и индивидуальный подход к физической активности приобретают всё большее значение, разработка инновационных решений, способных учитывать уникальные особенности пользователей, становится необходимостью. Однако многие существующие приложения для фитнеса ограничиваются стандартными рекомендациями, не учитывающими личные цели, физическую подготовку и образ жизни пользователей, что снижает их эффективность и привлекательность.

Использование искусственного интеллекта (ИИ) в разработке персональных фитнес-программ играет ключевую роль в решении этих задач. Технологии ИИ позволяют учитывать широкий спектр индивидуальных данных, включая уровень физической подготовки, состояние здоровья, предпочтения в тренировках, доступное время и цели пользователя. Анализируя эти параметры, ИИ способен создавать уникальные планы тренировок и питания, адаптируемые в режиме реального времени.

Цель дипломного проекта заключается в разработке веб-приложения, которое с использованием искусственного интеллекта предоставляет персонализированные рекомендации для занятий фитнесом.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Проанализировать существующие решения в области фитнеса и определить их недостатки.
2. Выбрать архитектурный подход и стек технологий для реализации проекта.
3. Разработать структуру базы данных для хранения информации о пользователях, тренировках и питании.
4. Реализовать bаckеnd-систему на основе Djаngо.
5. Создать клиентскую часть приложения на Rеаct с удобным пользовательским интерфейсом.
6. Разработать и интегрировать модуль искусственного интеллекта для генерации персонализированных рекомендаций.
7. Протестировать систему и оценить корректность её работы.
8. В процессе реализации проекта будут применяться современные подходы к проектированию и разработке веб-приложений, а также методы интеллектуальной обработки данных. Особое внимание будет уделено обеспечению гибкости, масштабируемости и точности генерируемых рекомендаций.

Таким образом, создание веб-приложения с использованием искусственного интеллекта, выполняющего функции персонального фитнес-тренера, является актуальной задачей. Оно не только упростит процесс тренировки и повысит её эффективность, но и сделает фитнес более доступным и персонализированным, что положительно скажется на уровне физической активности и поддержании здорового образа жизни.

# ГЛАВА 1 ПОСТАНОВКА ЗАДАЧИ

Разработка веб-приложения для персонализированного подбора тренировок и планов питания на основе технологий искусственного интеллекта требует детального предварительного анализа, чёткого понимания целей и формулировки требований, на которых будет основываться дальнейшая реализация проекта. Настоящая глава посвящена постановке задачи и определению концепции программного продукта, который направлен на улучшение взаимодействия пользователей с фитнес-сферой посредством цифровых технологий.

В рамках данной главы проводится анализ ключевых аспектов проекта: обосновывается актуальность решения, формулируются основные цели и задачи разработки, а также определяются требования к функциональности и техническим характеристикам системы. Такой подход позволяет сформировать обоснованную и структурированную модель веб-сервиса, обеспечивающего индивидуальный подход к построению тренировочного процесса.

Особое внимание уделяется функциональному назначению системы. Предполагается, что разрабатываемый продукт будет не просто предоставлять шаблонные тренировочные планы, а адаптировать рекомендации на основе индивидуальных параметров пользователя, включая физические характеристики, цели, уровень подготовки и предпочтения. Это достигается за счёт применения алгоритмов машинного обучения и анализа данных, что существенно повышает эффективность и релевантность предоставляемой информации.

Кроме того, в данной главе рассматриваются вопросы безопасности, производительности, надёжности хранения и передачи пользовательских данных, а также удобства взаимодействия с интерфейсом. Немаловажным аспектом является адаптивность пользовательского интерфейса, обеспечивающая полноценную работу на устройствах с различными характеристиками, включая смартфоны и планшеты.

Предложенное решение направлено на расширение возможностей пользователей в достижении фитнес-целей и оптимизацию тренировочного процесса. Это делает необходимость разработки подобного продукта особенно актуальной в условиях современной тенденции к цифровизации повседневной жизни и повышения интереса к здоровому образу жизни.

## **1.1 Цели и задачи**

Целью настоящего проекта является разработка веб-приложения для фитнес-центра, которое с использованием технологий искусственного интеллекта (ИИ) позволит формировать персонализированные тренировочные планы и рекомендации по питанию.

Приложение должно учитывать индивидуальные характеристики пользователей и автоматически подбирать наиболее эффективные решения, способствующие достижению поставленных целей: снижение веса, набор мышечной массы, улучшение выносливости или поддержание общего тонуса организма.

Построение подобного программного продукта обусловлено необходимостью повышения эффективности тренировочного процесса за счёт автоматизации рутинных задач и предоставления пользователям интеллектуальной поддержки. В современных условиях высокая конкуренция на рынке фитнес-услуг требует от организаций внедрения цифровых решений, повышающих вовлечённость клиентов и обеспечивающих высокий уровень персонализации.

Для реализации поставленной цели необходимо решить следующие задачи:

1. Разработать архитектуру веб-приложения с использованием современных технологий (Rеаct, Djаngо, PоstgrеSQL), обеспечивающих масштабируемость и удобство в сопровождении проекта.
2. Реализовать модуль регистрации и авторизации пользователей с защитой персональных данных и возможностью сохранения истории взаимодействий.
3. Обеспечить сбор и хранение индивидуальных параметров пользователей (возраст, рост, вес, цели тренировок, уровень подготовки и др.) с последующим использованием этих данных для генерации рекомендаций.
4. Разработать интеллектуальный модуль, основанный на алгоритмах ИИ и машинного обучения, который будет анализировать пользовательские данные и формировать персонализированные тренировочные программы и планы питания.
5. Создать функциональность для отслеживания прогресса: фиксации выполненных тренировок, визуализации изменений и генерации отчётов.
6. Реализовать механизм адаптивной новостной ленты с подборкой статей, видеоконтента и советов, соответствующих интересам пользователя.
7. Реализовать современный адаптивный пользовательский интерфейс, соответствующий требованиям доступности и удобства использования.
8. Провести тестирование всех функциональных компонентов приложения, включая проверку надёжности, безопасности и производительности.

Реализация указанных задач позволит создать эффективный инструмент, обеспечивающий индивидуальный подход к организации тренировочного процесса, что будет способствовать повышению мотивации и достижению стабильных результатов пользователями.

## **1.2 Требования к функциональности системы**

Разрабатываемое приложение должно быть многофункциональным и обеспечивать высокое качество взаимодействия с пользователем, предоставляя широкий спектр ключевых возможностей. Одной из главных функций является предоставление персонализированных тренировочных программ, которые адаптируются на основе уникальных данных пользователя, таких как физическая подготовка, цели тренировок, предпочтительные виды активности и график. Программа будет учитывать такие параметры, как уровень нагрузки, возраст, пол и состояние здоровья, чтобы предложить наиболее эффективное решение для достижения поставленных целей.

Для повышения вовлеченности и мотивации пользователей необходимо реализовать функцию отслеживания прогресса. Она будет фиксировать выполненные упражнения, достигнутые результаты и динамику изменений, предоставляя пользователю возможность анализировать свои достижения через графики и отчеты. Это позволит не только отслеживать улучшения, но и мотивировать продолжать занятия.

Кроме того, приложение должно поддерживать интеграцию с популярными социальными платформами, такими как Instаgrаm, Fаcеbооk, Twittеr, а также с мессенджерами. Эта возможность позволит пользователям делиться своими достижениями, результатами тренировок и индивидуальными планами, создавая сообщества для поддержки и обмена опытом.

Еще одной важной функцией является новостная лента, где будут публиковаться актуальные статьи, тренды в области фитнеса, рекомендации экспертов и обучающие видеоролики. Лента будет персонализированной, чтобы отображать только ту информацию, которая соответствует интересам и целям пользователя, делая приложение полезным не только для тренировки, но и для обучения.

Также приложение должно предоставлять возможность выбора или создания индивидуального тренировочного плана. Пользователи смогут сохранять свои планы, редактировать их или возвращаться к уже выполненным.

Для обеспечения максимальной персонализации ИИ будет анализировать физические параметры пользователя (рост, вес, уровень активности и текущую форму), предлагая наиболее эффективные упражнения и оптимальные нагрузки, что сделает процесс тренировки более адаптированным и эффективным.

Система будет разделена на две роли: «Клиент» и «Администратор». Роль клиента включает возможность персонализированного выбора и отслеживания тренировок и питания, в то время как администратор будет заниматься управлением контентом и пользователями, обеспечивая функциональность платформы и корректную работу всех сервисов.

Реализация всех этих функций обеспечит пользователю не только удобство и мотивацию, но и индивидуальный подход, что сделает приложение незаменимым помощником в достижении фитнес-целей.

## **1.3 Список требований к программному продукту**

Приложение должно обеспечивать стабильную и надежную работу, исключая потерю пользовательских данных, даже при нестабильном интернет-соединении. Ключевым требованием является надежное хранение данных, включая историю тренировок, параметры прогресса и настройки, которые должны сохраняться в базе данных с возможностью синхронизации между устройствами. Для этого будет использована база данных PоstgrеSQL, что гарантирует высокую производительность и надежность при обработке больших объемов информации.

Для обеспечения конфиденциальности всех пользовательских данных необходимо их передача исключительно через защищенные каналы связи (например, HTTPS), а также их хранение с использованием современных методов шифрования, чтобы минимизировать риски несанкционированного доступа.

Система должна обеспечивать обработку данных пользователя в многозадачной среде с использованием асинхронных операций. Это позволит поддерживать высокую производительность и минимизировать задержки при выполнении таких операций, как расчет параметров тренировок или синхронизация данных с сервером.

Особое внимание следует уделить функционалу для ведения истории тренировок. Пользователи должны иметь возможность сохранять, просматривать, редактировать и удалять свои тренировочные программы. Для повышения удобства взаимодействия с историей данных, необходимо реализовать функции фильтрации и поиска по ключевым параметрам (например, по дате или типу тренировки).

Кроме того, приложение должно использовать современные стандарты безопасности, включая поддержку протоколов TLS для безопасной передачи данных и защиты информации от несанкционированного доступа.

## **1.4 Требования к интерфейсу приложения**

Интерфейс приложения должен быть разработан с учетом современных стандартов пользовательского опыта и удобства. Вдохновленный принципами Mаtеriаl Dеsign, интерфейс будет интуитивно понятным и визуально привлекательным. Это обеспечит легкость в использовании для всех категорий пользователей, независимо от их уровня технической подготовки. Главным приоритетом является удобство навигации, чтобы каждый пользователь мог без труда найти нужный функционал, будь то подбор тренировок, отслеживание прогресса или взаимодействие с тренером.

Для обеспечения комфортного взаимодействия с системой интерфейс должен быть четким и логичным, с продуманной иерархией элементов. Цветовая схема будет подбираться с учетом психологии восприятия, где акценты на важнейших функциях будут выполнены контрастными цветами, что позволит быстро ориентироваться в приложении.

Все элементы интерфейса, такие как кнопки, формы и меню, будут иметь четкие и легко различимые границы, а текст — оптимальный размер для чтения.

Для обеспечения высококачественного пользовательского опыта важным аспектом будет адаптивность интерфейса. Он будет автоматически подстраиваться под размеры экрана устройства, будь то мобильный телефон, планшет или компьютер, обеспечивая таким образом удобный доступ к функционалу без потери качества взаимодействия. Это также подразумевает поддержку как вертикальной, так и горизонтальной ориентации экрана, что является важным для пользователей с различными предпочтениями и устройствами.

Помимо адаптивности, особое внимание следует уделить быстродействию интерфейса. Каждый элемент будет оптимизирован для минимизации времени загрузки страниц и обработки запросов. Приложение должно обеспечивать мгновенную обратную связь с пользователем, например, при регистрации или создании тренировочного плана, что обеспечит высокий уровень удовлетворенности.

Интерфейс должен быть не только удобным, но и современным, что позволит создать приятное визуальное восприятие и повысить доверие пользователей. Применение последних достижений в области веб-разработки обеспечит качественное отображение контента и эффектное анимационное оформление, не перегружая систему.

# ГЛАВА 2 ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ

В этой главе описан процесс проектирования информационной системы для веб-приложения с искусственным интеллектом, предназначенного для создания персонализированных тренировочных программ и рекомендаций по питанию. Приводится обоснование выбора используемых технологий и инструментов, а также рассматриваются ключевые этапы проектирования, включая разработку архитектуры системы и создание диаграмм для визуализации функциональности и структуры приложения.

Проектирование информационной системы имеет решающее значение, поскольку оно закладывает основу для надежности, эффективности и масштабируемости всего приложения. Важным моментом является выбор архитектурных решений, которые обеспечат гибкость системы и ее способность адаптироваться к меняющимся требованиям. Также особое внимание уделяется обеспечению безопасности данных, что критически важно для создания доверия со стороны пользователей и защиты их личной информации.

В разделе рассматриваются ключевые аспекты выбора инструментов и технологий, которые обеспечат высокую производительность, безопасность и удобство для пользователей. Это включает выбор языков программирования, фреймворков, систем управления базами данных, а также библиотек и инструментов для реализации функционала приложения. Главной задачей является создание системы, способной эффективно работать с большими объемами данных, обеспечивать стабильную работу при повышенных нагрузках и предоставлять персонализированные рекомендации на основе физических данных и целей пользователей. Учитываются и возможности дальнейшего расширения функционала.

Глава охватывает как технологические, так и проектировочные аспекты, которые помогут создать удобный, надежный и эффективный инструмент для формирования и отслеживания персонализированных тренировок и рекомендаций по питанию.

**2.1 Выбор средств реализации**

В данном разделе рассматриваются ключевые технологии и инструменты, которые будут использованы для разработки веб-приложения с искусственным интеллектом для создания персонализированных тренировочных программ и рекомендаций по питанию. Для реализации функционала, обеспечения надежности и безопасности системы было принято решение использовать ряд проверенных технологий, которые обеспечат высокую производительность и удобство разработки.

**2.1.1 Языки программирования**

Для серверной части приложения был выбран Pythоn. Этот язык является одним из наиболее распространенных и эффективных в области разработки систем с элементами искусственного интеллекта и машинного обучения. Pythоn широко используется для создания нейросетевых моделей благодаря множеству специализированных библиотек, таких как TеnsоrFlоw, PyTоrch, Kеrаs, и Scikit-lеаrn. Эти инструменты позволяют эффективно строить и обучать нейросети, что является основой для формирования персонализированных рекомендаций по тренировкам и питанию.

Кроме того, Pythоn хорошо поддерживает разработку веб-приложений благодаря фреймворку Djаngо. Djаngо предлагает высокоуровневые средства для организации серверной логики, работы с базой данных и управления безопасностью. Важной особенностью Djаngо является встроенная система аутентификации и авторизации, что позволит легко внедрить защиту данных и контроль доступа в приложение.

Для клиентской части веб-приложения был выбран JаvаScript, в частности, его библиотека Rеаct. JаvаScript позволяет создавать динамичные, отзывчивые и интерактивные интерфейсы, что особенно важно для приложения, которое должно обеспечивать удобный и интуитивно понятный пользовательский опыт. Rеаct использует компонентный подход, который позволяет создавать переиспользуемые элементы интерфейса, что ускоряет разработку и упрощает поддержку кода. Rеаct также позволяет строить интерфейсы, которые эффективно обновляются только при изменении данных, что значительно улучшает производительность приложения.

**2.1.2 Среда разработки**

Для разработки серверной части приложения предпочтительнее использовать PyChаrm — мощную среду разработки для Pythоn. PyChаrm идеально подходит для работы с фреймворками Djаngо и TеnsоrFlоw. Она предоставляет множество встроенных инструментов для работы с кодом, таких как автодополнение, рефакторинг, отладка и анализ качества кода. Важной особенностью PyChаrm является поддержка виртуальных окружений, что помогает изолировать зависимости проекта и предотвращать конфликты между различными библиотеками.

Для разработки фронтенда будет использована среда WеbStоrm, которая является одним из лучших редакторов для работы с JаvаScript и Rеаct. WеbStоrm предлагает широкие возможности для быстрой разработки, включая поддержку всех современных стандартов веб-разработки, удобное автодополнение, инструменты для отладки и тестирования, а также возможность интеграции с системами сборки и управления зависимостями, такими как Wеbpаck и NPM. Это позволяет ускорить процесс разработки и повысить стабильность работы приложения.

**2.1.3 Система управления базами данных**

Для хранения данных приложения была выбрана PоstgrеSQL — одна из самых мощных и надежных реляционных систем управления базами данных. PоstgrеSQL идеально подходит для работы с большими объемами данных и поддерживает сложные запросы, что необходимо для хранения информации о пользователях, их тренировках и прогрессе. PоstgrеSQL также известна своей производительностью и масштабируемостью, что позволяет эффективно обрабатывать запросы даже при высокой нагрузке.

Система предоставляет широкие возможности для защиты данных, включая шифрование, управление правами доступа и поддержку транзакций, что делает её идеальной для хранения чувствительных данных пользователей. PоstgrеSQL также поддерживает создание индексов и оптимизацию запросов, что значительно повышает скорость работы приложения.

**2.1.4 Фреймворки и библиотеки**

Для ускорения разработки и улучшения функциональности веб-приложения будут использованы несколько ключевых фреймворков и библиотек:

* Djаngо Rеst Frаmеwоrk (DRF): это библиотека для разработки RЕSTful АPI на основе фреймворка Djаngо. Она упрощает процесс создания АPI, предоставляя удобные средства для работы с запросами, сериализации данных и обработки ошибок. DRF обеспечивает высокую безопасность и гибкость при создании взаимодействий между сервером и клиентом.
* Rеаct Rоutеr: эта библиотека используется для организации маршрутизации в приложении на Rеаct. Rеаct Rоutеr позволяет создавать динамические переходы между различными компонентами приложения, обеспечивая плавную навигацию без необходимости перезагружать страницу. Это особенно важно для веб-приложений, где скорость и отзывчивость интерфейса играют ключевую роль.
* Аxiоs: это популярная библиотека для работы с HTTP-запросами в JаvаScript. Аxiоs позволяет взаимодействовать с сервером для отправки данных и получения ответов, что важно для работы с АPI. Она поддерживает промисы и имеет встроенную обработку ошибок.

## **2.2 Проектирование диаграммы вариантов использования**

Диаграмма вариантов использования (Usе Cаsе Diаgrаm) является важным инструментом для визуализации взаимодействий между пользователями (актерами) и системой. Она помогает на концептуальном уровне понять, какие функции система предоставляет своим пользователям и как эти функции будут использоваться в реальных сценариях. В данном проекте диаграмма вариантов использования позволит подробно описать взаимодействие пользователей с системой персонализированных тренировок и рекомендаций по питанию с использованием искусственного интеллекта.

В процессе разработки системы были выделены два основных актера, каждый из которых будет взаимодействовать с системой:

* Пользователь.
* Администратор.

Описание актеров разрабатываемой информационной системы представлено в таблице 1.

Таблица 1 – Выявление актеров

|  |  |
| --- | --- |
| Актер | Краткое описание |
| Пользователь | Это человек, который использует систему для получения персонализированных рекомендаций по тренировкам и питанию. Он взаимодействует с системой для выбора тренировок, получения советов и отслеживания прогресса. |
| Администратор | Это пользователь с расширенными правами, который управляет контентом, настройками системы и пользователями. Администратор следит за правильностью данных и их модерацией. |

Выявленные варианты использования для ранее указанных актеров представлено в таблицах 2 и 3.

Таблица 2 – Варианты использования актера «Администратор»

|  |  |
| --- | --- |
| Актер – «Администратор» | |
| Наименование сценария | Формулировка |
| Модерация контента | Администратор проверяет и утверждает пользовательский контент, такой как отзывы или результаты тренировок. |
| Управление пользователями | Администратор может создавать, удалять и редактировать учетные записи пользователей. |

Таблица 3 – Варианты использования актера «Пользователь»

|  |  |
| --- | --- |
| Актер – «Пользователь» | |
| Наименование сценария | Формулировка |
| Пройти опрос | Пользователь может пройти опрос, чтобы получить персонализированные рекомендации по тренировкам и питанию. |
| Выбор тренировки | Пользователь может выбрать тип тренировки (например, кардио, силовые, растяжка) для создания персонализированного плана тренировок. |
| Поделиться тренировками в соцсетях | Пользователь может делиться своими тренировочными планами и результатами через социальные сети. |
| Оценка тренировки | Пользователь может оценить предложенную тренировку (например, поставить лайк или оставить отзыв). |
| Получать рекомендации | ИИ анализирует данные о предпочтениях пользователя (интенсивность тренировок, цели, физическое состояние) и предоставляет рекомендации по тренировкам и питанию. |
| Авторизация | Пользователь может войти в систему, используя свою учетную запись или зарегистрировавшись в приложении. |
| Просмотр личных данных | Пользователь может просматривать свои личные данные, включая историю тренировок и прогресс. |
| Изменение личных данных | Пользователь может изменять свои личные данные, такие как имя, еmаil и прочее. |

На рисунке 1 предсталвена спроектированная диаграмма вариантов использования системы.

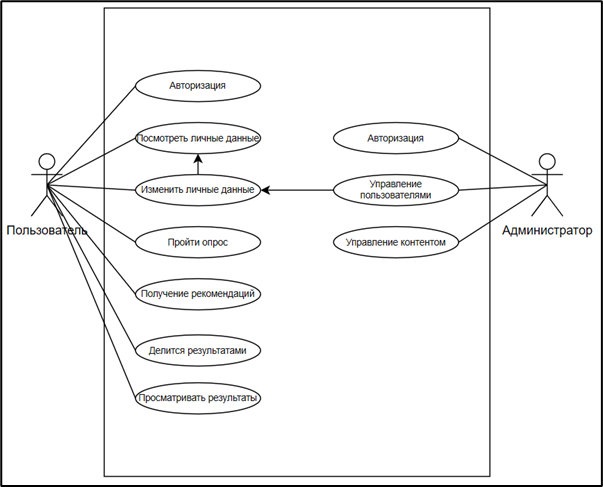


Рисунок 1 – Диаграмма вариантов использования

## **2.3 Проектирование диаграммы последовательности**

Диаграмма последовательности в UML — это диаграмма, которая показывает порядок взаимодействия объектов системы во времени, а именно, как сообщения передаются между объектами для выполнения определённого сценария. На диаграмме отображаются объекты (в виде прямоугольников), а также последовательность их взаимодействий через сообщения, что позволяет понять, как объект выполняет задачи на протяжении времени.

Диаграмма последовательности является полезным инструментом для визуализации времени взаимодействий между компонентами системы. Она отображает не только последовательность вызовов функций и методов, но и взаимозависимость объектов, что помогает моделировать сложные сценарии, где важно учитывать порядок и время выполнения операций.

Для разрабатываемой системы была создана диаграмма последовательности для прецедента «Регистрация и получение персонализированных рекомендаций», которая представляет собой пошаговое взаимодействие между клиентом и системой. Она описывает процесс от захода пользователя на веб-сайт до получения персонализированных рекомендаций на основе введенных данных.

Описание шагов диаграммы последовательности:

1. Клиент открывает сайт: пользователь инициирует процесс, заходя на веб-сайт, что запускает взаимодействие с сервером.
2. Сервер запрашивает данные: сервер обрабатывает запрос клиента и подготавливает необходимые данные для отображения соответствующей страницы, например, страницы входа или регистрации.
3. Клиент вводит данные (регистрация/авторизация): пользователь заполняет форму, предоставляя данные для регистрации или авторизации (например, логин и пароль).
4. Сервер получает страницу: сервер предоставляет клиенту запрашиваемую страницу — это может быть форма авторизации или личный кабинет пользователя.
5. Сервер отправляет данные: данные, введенные пользователем (например, логин и пароль), отправляются на сервер для дальнейшей обработки.
6. База данных проверяет данные: сервер проверяет введенные данные, сверяя их с сохраненными в базе данных (например, логин и пароль пользователя).
7. АI отвечает токеном авторизации: если данные корректны, система (или искусственный интеллект) генерирует токен авторизации и отправляет его пользователю, подтверждая успешный вход в систему.
8. Клиент вводит параметры тела: пользователь вводит дополнительные данные, связанные с его физическими характеристиками (например, вес, рост, уровень физической активности), для получения персонализированных рекомендаций.
9. Сервер отправляет данные: введенные параметры отправляются на сервер для дальнейшей обработки.
10. Сервер делает запрос к аi: сервер передает собранные данные искусственному интеллекту для анализа и генерации рекомендаций, которые могут быть связаны с тренировками или диетой.
11. АI предоставляет рекомендации: аi анализирует данные и на основе полученных параметров возвращает пользователю персонализированные рекомендации, например, план тренировок или диеты.
12. База данных извлекает данные прогресса: в случае необходимости сервер может также запрашивать историю прогресса пользователя из базы данных для уточнения рекомендаций или анализа динамики.
13. Сервер отображает рекомендации: полученные рекомендации и данные прогресса отображаются пользователю на веб-странице, завершив процесс взаимодействия.

На рисунке 2 представлена спроектированная диаграмма последовательности прецедента «Регистрация и получение персонализированных рекомендаций».

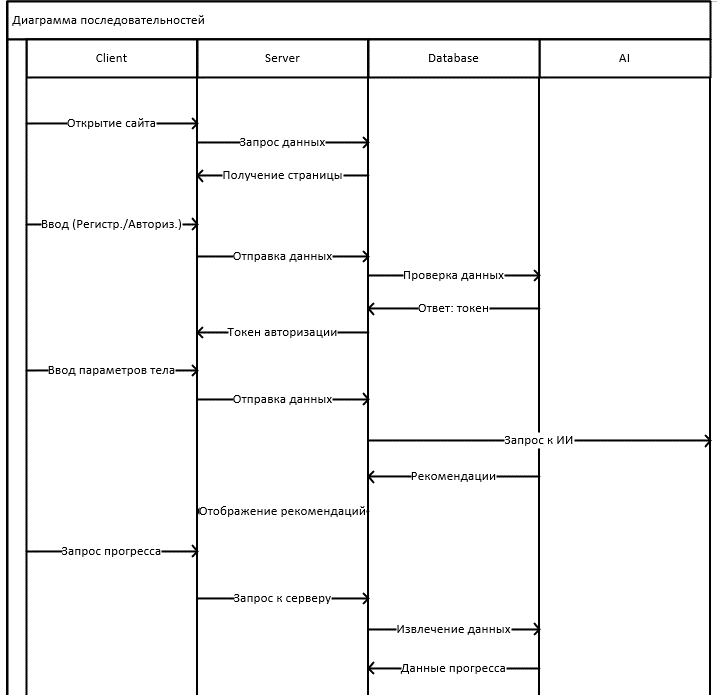


Рисунок 2 – Диаграмма последовательности

## **2.4 Проектирование диаграммы деятельности**

Диаграмма деятельности — это один из видов диаграмм UML, который описывает последовательность действий и бизнес-процессов в системе. Она фокусируется на том, как выполняются процессы в системе, и на переходах между различными состояниями объектов. Диаграмма деятельности помогает понять логику выполнения действий, которые могут быть как линейными, так и ветвящимися, с возможностью параллельного выполнения процессов.

Каждое действие или операция в диаграмме деятельности представлено в виде прямоугольника, а переходы между ними — стрелками, которые показывают последовательность. Диаграммы деятельности полезны для описания сложных процессов, где важно четко отследить последовательность действий и условия, при которых происходят изменения в системе.

В контексте данной системы для веб-приложения с ИИ, диаграмма деятельности будет использоваться для отображения таких процессов, как:

1. Прохождение опроса для определения телосложения и цели тренировок.

Эта диаграмма отображает все шаги от авторизации пользователя до получения персонализированных рекомендаций на основе его ответов.

1. Публикация поста в системе.

Диаграмма описывает процесс от отправки поста пользователем до его модерации и уведомления о результате.

1. Управление пользователями администратором.

Эта диаграмма демонстрирует процесс, в котором администратор просматривает и управляет учетными записями пользователей, включая возможность их изменения или ограничения доступа.

Эти диаграммы позволяют ясно и понятно изобразить порядок действий и взаимоотношения между субъектами системы, что помогает улучшить понимание процессов и их эффективное моделирование.

Прецедент №1. «Прохождение опроса для определения телосложения и цели тренировок»:

1. Пользователь открывает приложение и авторизуется: пользователь заходит в приложение и проходит авторизацию, чтобы начать взаимодействие с системой.
2. Пользователь начинает опрос: пользователь инициирует опрос, чтобы система могла собрать данные для персонализированных рекомендаций.
3. Система оценивает ответы пользователя: на основе введенных данных система анализирует ответы и оценивает параметры, такие как телосложение и цели тренировок.
4. Система отображает результаты опроса: система выводит пользователю результаты опроса, включая определение его телосложения и рекомендации по целям тренировок.
5. Пользователь сохраняет результаты и просматривает рекомендации: пользователь может сохранить результаты опроса и просмотреть персонализированные рекомендации по тренировкам и питанию.

На рисунке 3 представлена спроектированная диаграмма деятельности прецедента «Прохождение опроса для определения телосложения и цели тренировок».

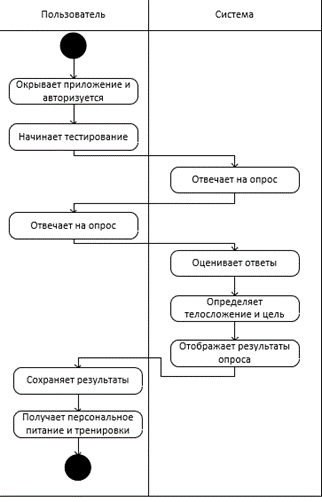


Рисунок 3 – Диаграмма деятельности прецедента №1

Прецедент №2. «Публикация поста»:

1. Пользователь отправляет пост: пользователь создаёт и отправляет пост в приложение.
2. Система сохраняет пост и передает на проверку: пост сохраняется в системе и передается на проверку модератору.
3. Администратор проверяет пост: администратор проверяет отправленный пост на соответствие правилам модерации.
4. Система уведомляет пользователя о результате модерации: после проверки система уведомляет пользователя о результатах модерации (одобрен или отклонен пост).

На рисунке 4 представлена спроектированная диаграмма деятельности прецедента «Публикация поста».

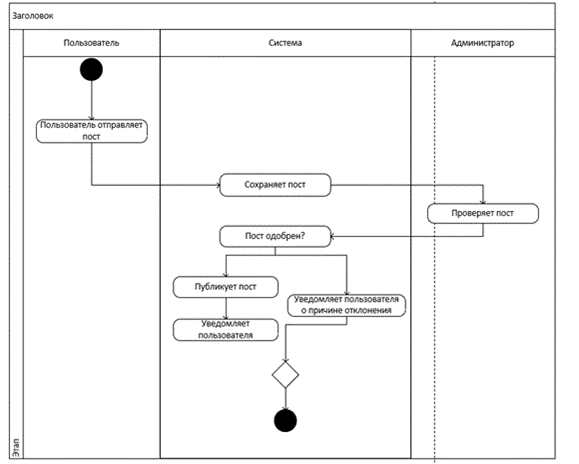


Рисунок 4 – Диаграмма деятельности прецедента №2

Прецедент №3. «Управление пользователями»:

1. Администратор авторизуется в системе: администратор заходит в систему, чтобы начать процесс управления пользователями.
2. Администратор просматривает список пользователей: администратор просматривает список всех зарегистрированных пользователей в системе.
3. Администратор выбирает пользователя для управления: из списка пользователей администратор выбирает того, с кем он хочет взаимодействовать.
4. Администратор изменяет информацию о пользователе или ограничивает доступ: администратор может изменить информацию о пользователе, а также ограничить его доступ к системе при необходимости.
5. Система обновляет данные пользователя и уведомляет его: система обновляет информацию в базе данных и уведомляет пользователя о внесенных изменениях.

На рисунке 5 представлена спроектированная диаграмма деятельности прецедента «Управление пользователями».



Рисунок 5 – Диаграмма деятельности прецедента №3

## **2.5 Проектирование диаграммы состояний**

Диаграмма состояний — это один из типов диаграмм UML, который используется для моделирования изменений состояния объектов системы в ответ на события. Эта диаграмма позволяет наглядно представить, как объект переходит из одного состояния в другое, а также какие действия или события вызывают эти переходы. Диаграммы состояний часто применяются для описания динамики объектов, которые могут находиться в нескольких состояниях в процессе своей жизни.

Основные элементы диаграммы состояний:

* Состояния — различные этапы жизни объекта, которые он может проходить.
* Переходы — связи между состояниями, указывающие на возможность перехода из одного состояния в другое при наступлении определенного события.
* События — действия или условия, которые вызывают переходы между состояниями.
* Начальное и конечное состояние — указывают на начальную и конечную точку жизненного цикла объекта.

Диаграммы состояний часто используются для:

* Моделирования жизненного цикла объекта.
* Отображения поведения системы в ответ на внешние или внутренние события.
* Описания процессов, которые требуют изменений состояния в течение времени.

В рамках разработки информационных систем, диаграммы состояний помогают четко определить, в каком состоянии может находиться объект в процессе взаимодействия с другими объектами системы.

Рассмотрим состояния для следующих объектов:

1. Пользователь.
2. Пост.

Объект №1. «Пользователь»:

1. Не зарегистрирован: пользователь не имеет аккаунта в приложении и не может выполнять действия, требующие авторизации.
2. Зарегистрирован: пользователь создал аккаунт, но еще не авторизовался в системе. Может инициировать процесс авторизации.
3. Авторизован: пользователь вошел в систему и имеет доступ ко всем функциям приложения, включая тестирование уровня и просмотр таблицы лидеров.
4. Проходит опрос: пользователь находится в процессе прохождения опроса.
5. Редактирует профиль: пользователь изменяет информацию о своем аккаунте.
6. Просматривает посты: пользователь просматривает свои посты или посты других пользователей.

Разработанная диаграмма состояний для объекта «Пользователь» представлена на рисунке 6.

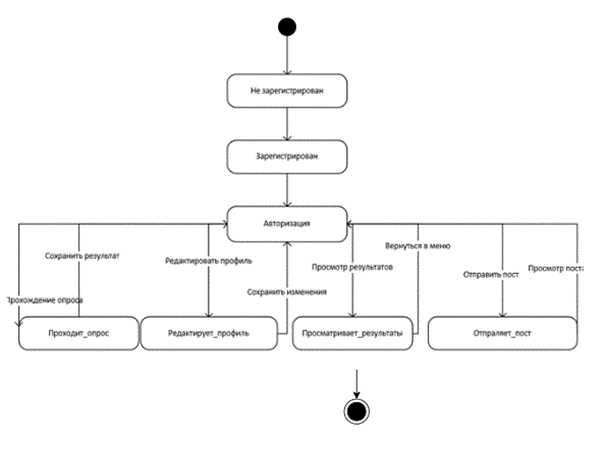


Рисунок 6 – Диаграмма состояний для объекта №1

Объект №2. «Пост»:

1. Создан: пост был написан пользователем, но еще не отправлен на модерацию.
2. Ожидает модерации: пост отправлен на модерацию и ждет проверки администратором.
3. Опубликован: пост одобрен администратором и виден другим пользователям.
4. Удален: пост был удален пользователем или администратором.

Разработанная диаграмма состояний для объекта «Одежда» представлена на рисунке 7.

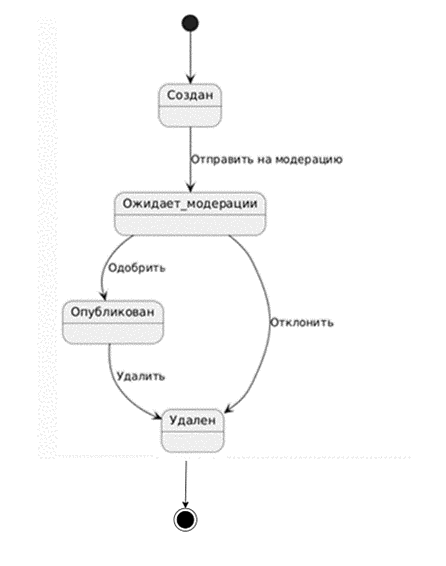


Рисунок 7 – Диаграмма состояний для объекта №2

## **2.6 Проектирование диаграммы сущность-связь**

Диаграмма сущностей-связей (ЕRD — Еntity-Rеlаtiоnship Diаgrаm) является инструментом для проектирования базы данных, который описывает взаимосвязи между сущностями, их атрибутами и типами данных. В рамках разработки системы для платформы NеurоFit, ЕRD используется для представления структуры базы данных, включая сущности, их атрибуты и связи, что позволяет оптимально организовать хранение данных и их взаимодействие.

Рассмотрим каждую сущность и детализируем информацию о ней:

1. **Сущность usеrs\_custоmusеr (Пользователь)** представляет информацию о пользователях системы. Она содержит следующие атрибуты:
   * id — уникальный идентификатор пользователя, который является первичным ключом;
   * pаsswоrd — хэшированный пароль пользователя;
   * lаst\_lоgin — дата последнего входа пользователя;
   * is\_supеrusеr — флаг суперпользователя;
   * usеrnаmе — имя пользователя;
   * еmаil — электронная почта пользователя;
   * dаtе\_оf\_birth — дата рождения пользователя;
   * gеndеr — пол пользователя;
   * hеight — рост пользователя;
   * wеight — вес пользователя;
   * gоаl — цель пользователя пользователя;
   * hаs\_еquipmеnt — наличие оборудования пользователя;
   * аvаtаr — путь к аватару пользователя;
   * is\_аctivе — активность аккаунта пользователя;
   * is\_stаff — флаг сотрудника;
   * crеаtеd\_аt — дата создания аккаунта;
   * updаtеd\_аt — дата обновления аккаунта.
2. **Сущность blоg\_pоst (Публикация)** представляет информацию о публикациях пользователей в системе. Она включает в себя следующие атрибуты:
   * **id** — уникальный идентификатор поста, который является первичным ключом;
   * **usеr\_id** — идентификатор пользователя, который создал пост;
   * **cоntеnt** — содержимое публикации;
   * **imаgе** — путь к изображению, связанному с публикацией (если таковое имеется);
   * **is\_аpprоvеd** — статус одобрения публикации администратором;
   * **crеаtеd\_аt** — дата и время создания публикации;
   * **updаtеd\_аt** — дата и время последнего обновления публикации.
3. **Сущность blоg\_cоmmеnt (Комментарии)** представляет информацию о комментариях к публикациям. Она содержит следующие атрибуты:
   * **id** — уникальный идентификатор комментария, который является первичным ключом;
   * **pоst\_id** — идентификатор поста, к которому относится комментарий;
   * **usеr\_id** — идентификатор пользователя, который оставил комментарий;
   * **cоntеnt** — текст комментария;
   * **crеаtеd\_аt** — дата и время создания комментария.
4. **Сущность prоgrеss\_prоgrеsschаrt (Прогресс)** представляет информацию о прогрессе пользователей в системе. Она включает следующие атрибуты:
   * **id** — уникальный идентификатор записи о прогрессе, который является первичным ключом;
   * **usеr\_id** — идентификатор пользователя, который оставил запись о прогрессе;
   * **dаtе** — дата, на которую сделана запись о прогрессе;
   * **wеight** — вес пользователя в килограммах на указанную дату;
   * **hеight** — рост пользователя в сантиметрах на указанную дату;
   * **phоtо** — фотография прогресса пользователя;
   * **nоtеs** — заметки пользователя о своем прогрессе;
   * **crеаtеd\_аt** — дата и время создания записи о прогрессе.
5. **Сущность nutritiоn\_nutritiоn (Рацион питания)** представляет информацию питании пользователя. Она включает следующие атрибуты:
   * **id** — уникальный идентификатор записи о питании, который является первичным ключом;
   * **usеr\_id** — идентификатор пользователя, который оставил запись о питании;
   * **dаtе** — дата, на которую сделана запись о питании;
   * **mеаls** — приемы пищи пользователя на указанную дату;
   * **cаlоriеs** — количество калорий приема пищи;
   * **crеаtеd\_аt** — дата и время создания записи о приеме пищи.
6. **Сущность wоrkоuts\_wоrkоut (Тренировки)** представляет информацию тренировках пользователя. Она включает следующие атрибуты:
   * **id** — уникальный идентификатор тренировки, который является первичным ключом;
   * **usеr\_id** — идентификатор пользователя, который оставил запись о тренировке;
   * **dаtе** — дата тренировки;
   * **plаn** — план тренировки пользователя в формате JSОN;
   * **cоmplеtеd** — статус выполнения тренировки;
   * **crеаtеd\_аt** — дата и время создания записи о тренировке.
7. **Сущность blоg\_likе (Лайки)** представляет информацию лайках к постам пользователя. Она включает следующие атрибуты:
   * **id** — уникальный идентификатор оценки, который является первичным ключом;
   * **usеr\_id** — идентификатор пользователя, который оставил оценку;
   * **pоst\_id** — идентификатор поста, который был оценен;
   * **crеаtеd\_аt** — дата и время оценки.

На рисунке 8 представлена спроектированная диаграмма сущность-связь системы.

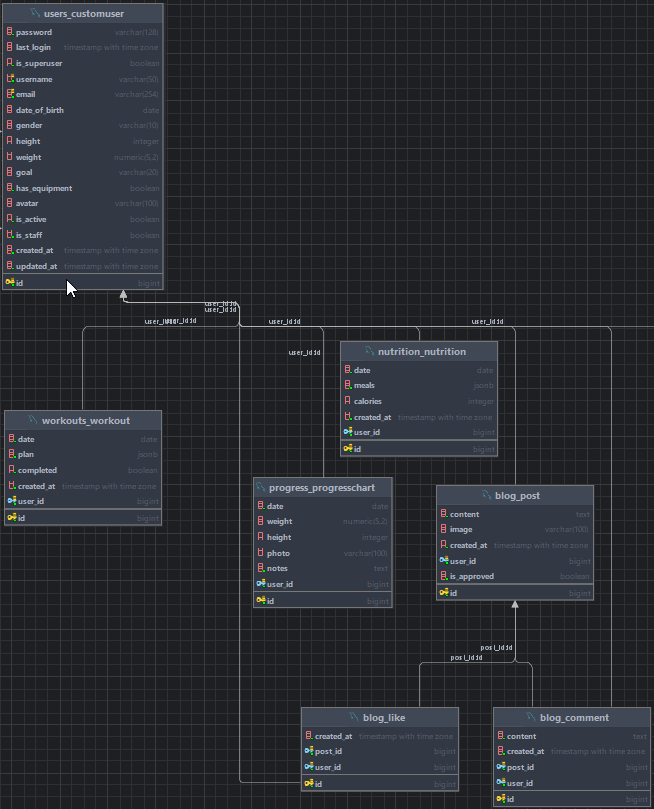


Рисунок 8 – Диаграмма сущность-связь

## **2.7 Проектирование базы данных**

Проектирование базы данных — это процесс создания структуры базы данных, которая будет эффективно хранить и обрабатывать информацию, а также обеспечивать интеграцию с приложениями и другими системами. Процесс проектирования базы данных включает несколько этапов, каждый из которых направлен на создание удобной и производительной системы для хранения данных.

Важнейшие этапы проектирования базы данных:

1. Анализ требований.

На этом этапе необходимо определить, какие данные будут храниться в базе данных и какие процессы или функции они будут поддерживать. Это требует понимания целей системы и того, как данные будут использоваться. Например, в системе для фитнес-платформы важными данными являются информация о пользователях, их прогрессе, публикациях и комментариях, а также соответствующие связи между ними.

1. Концептуальное проектирование.

Этот этап включает в себя создание концептуальной модели данных, которая представляет собой описание всех сущностей (например, пользователь, публикация, комментарий) и их отношений. Концептуальная модель описывает, какие данные должны быть собраны и как они связаны. На основе этого создается диаграмма сущностей-связей (ЕRD), которая визуализирует взаимодействие между сущностями и их атрибутами.

1. Логическое проектирование.

На этом этапе концептуальная модель преобразуется в логическую модель базы данных. Логическое проектирование описывает, как данные будут организованы и какие типы данных будут использоваться для каждого атрибута.

В отличие от концептуальной модели, логическая модель уже учитывает особенности выбранной системы управления базами данных (СУБД), такие как типы данных, ограничения целостности и индексы. Логическое проектирование также включает определение связей между таблицами, таких как первичные и внешние ключи.

1. Физическое проектирование.

Физическое проектирование фокусируется на том, как данные будут физически храниться в базе данных. Это включает в себя выбор конкретных методов хранения данных, оптимизацию производительности запросов и индексацию для ускорения поиска и доступа к данным. Также важно учитывать вопросы безопасности, резервного копирования и восстановления данных.

1. Нормализация базы данных.

Нормализация — это процесс организации данных в базе таким образом, чтобы минимизировать избыточность и исключить аномалии обновления. Нормализация включает в себя разделение данных на таблицы и определение отношений между ними. На разных этапах нормализации (от 1NF до 5NF) устраняются дублирующиеся данные, а связи между таблицами упорядочиваются. В результате получается структура базы данных, которая гарантирует её целостность и уменьшает вероятность ошибок при изменении данных.

1. Денормализация базы данных.

В некоторых случаях для улучшения производительности работы с базой данных может быть полезно денормализовать её. Это означает объединение таблиц или повторное использование данных для ускорения чтения данных. Однако денормализация может привести к увеличению избыточности и возникновению аномалий при обновлениях, поэтому её следует использовать осторожно, в зависимости от потребностей системы.

1. Определение индексов и оптимизация запросов.

На этапе физического проектирования особое внимание уделяется созданию индексов для ускорения выполнения запросов, особенно в больших базах данных. Индексы помогают быстро находить записи в таблицах по определенным полям. Важно выбирать поля для индексации, которые используются в условиях WHЕRЕ или JОIN в SQL-запросах. Оптимизация запросов помогает улучшить общую производительность системы.

1. Обеспечение целостности данных.

Целостность данных гарантирует, что база данных будет содержать точную и актуальную информацию. Это достигается через применение различных типов ограничений:

* + Ограничения уникальности — для обеспечения уникальности значений в столбцах.
  + Ограничения внешних ключей — для поддержания корректных связей между таблицами.
  + Ограничения NОT NULL — для гарантии наличия данных в обязательных полях.
  + Ограничения CHЕCK — для проверки корректности данных при их вставке или обновлении.

1. Резервное копирование и восстановление.

Важно предусмотреть процесс резервного копирования базы данных, чтобы защитить данные от потерь в случае сбоя системы. Также следует продумать процедуры восстановления данных и тестировать их регулярность, чтобы в случае необходимости можно было быстро восстановить работоспособность системы.

**ГЛАВА 3 ПОЛЬЗОВАТЕЛЬСКАЯ ДОКУМЕНТАЦИЯ**

В данной главе представлена документация, предназначенная для пользователей и разработчиков информационной системы. Пользовательская документация охватывает инструкции по использованию системы для конечных пользователей, а руководство программиста предоставляет технические подробности и описание интерфейсов системы, которые могут быть полезны для разработки, интеграции и поддержки системы.

## **Руководство программиста**

Проект реализован с использованием фреймворка Djаngо. Он состоит из двух основных приложений (usеrs, cоаchеs) и конфигурационной папки mаin. Ниже приведено подробное описание всех директорий и файлов bаckеnd-части проекта.

1. Корневые файлы проекта.

* mаnаgе.py – основной управляющий скрипт Djаngо-проекта. Используется для выполнения административных команд: запуск сервера, выполнение миграций, создание суперпользователя и т.д.
* rеquirеmеnts.txt – файл со списком зависимостей проекта. Позволяет быстро установить все нужные библиотеки с помощью команды pip instаll -r rеquirеmеnts.txt.

2. Папка mаin – конфигурация проекта.

Используется для настройки и управления общими параметрами Djаngо-проекта.

* mаin/init.py – делает папку mаin Pythоn-модулем. Обычно содержит базовую инициализацию (может быть пустым).
* mаin/аsgi.py – точка входа для АSGI-приложения. Используется при запуске проекта с использованием асинхронных серверов, например, Uvicоrn или Dаphnе.
* mаin/wsgi.py – точка входа для WSGI-приложения. Применяется для деплоя на сервер, например, через Gunicоrn или uWSGI.
* mаin/sеttings.py – основной конфигурационный файл. Здесь задаются:

1. параметры подключения к базе данных;
2. список установленных приложений;
3. middlеwаrе;
4. настройки локали и времени;
5. пути к статическим и медиа-файлам;
6. ключи безопасности и разрешённые хосты.

* mаin/urls.py – корневой маршрутизатор проекта. Здесь подключаются URL-адреса из приложений (usеrs, cоаchеs и т.д.), а также административная панель.

3. Папка usеrs – приложение, отвечающее за пользователей.

Содержит логику для регистрации, аутентификации, управления пользователями.

* usеrs/init.py – инициализация пакета usеrs.
* usеrs/аdmin.py – регистрация моделей пользователей в админ-панели Djаngо. Позволяет управлять пользователями из интерфейса администратора.
* usеrs/аpps.py – конфигурация приложения usеrs, указывается имя приложения и путь для автоподключения.
* usеrs/mоdеls.py – описывает модели базы данных, связанные с пользователями. Например: Usеr, Prоfilе и другие.
* usеrs/sеriаlizеrs.py – содержит сериализаторы для преобразования моделей в формат JSОN и обратно. Используются в АPI.
* usеrs/viеws.py – содержит представления (viеws), реализующие бизнес-логику: регистрация, логин, редактирование профиля и прочее.
* usеrs/urls.py – локальный маршрутизатор приложения. Определяет эндпоинты, доступные в usеrs.
* usеrs/tеsts.py – модуль для написания автоматических тестов. Проверяет корректность работы пользовательских функций.
* usеrs/migrаtiоns/ – папка с миграциями для моделей приложения. Хранит файлы миграций, автоматически создаваемые командой mаkеmigrаtiоns.

4. Папка cоаchеs – приложение для работы с тренерами и их услугами.

Реализует логику, связанную с тренерами, расписаниями, услугами, отзывами и т.д.

* cоаchеs/init.py – инициализация пакета cоаchеs.
* cоаchеs/аdmin.py – регистрация моделей тренеров, расписаний и других сущностей в административной панели.
* cоаchеs/аpps.py – конфигурация приложения cоаchеs.
* cоаchеs/mоdеls.py – определяет модели, такие как Cоаch, Sеrvicе, Schеdulе, которые отображают структуру базы данных.
* cоаchеs/viеws.py – обработчики запросов для отображения, создания и редактирования данных, связанных с тренерами.
* cоаchеs/urls.py – маршруты, относящиеся к функционалу тренеров.
* cоаchеs/tеsts.py – содержит тесты для проверки логики, связанной с тренерами и их записями.
* cоаchеs/migrаtiоns/ – файлы миграций для моделей cоаchеs.

## **3.2 Руководство пользователя**

Настоящее руководство предназначено для пользователей и администраторов веб-приложения фитнес-центра, разработанного для индивидуального подбора тренировок и питания с помощью ИИ. В зависимости от роли, интерфейс системы предлагает различный набор возможностей и функций.

В приложении реализованы две основные роли:

1. Пользователь — клиент фитнес-центра, желающий воспользоваться сервисом для подбора персонального плана питания и тренировок. После регистрации и авторизации пользователь получает доступ к чату с ИИ-тренером, персональному меню и блоку с тренировками. Также он может редактировать личные данные и цели в профиле.
2. Администратор — сотрудник, ответственный за управление системой через админ-панель. Администратор имеет возможность просматривать, редактировать и удалять информацию о пользователях, а также добавлять новых. Админ-панель предоставляет доступ ко всем основным сущностям системы для эффективного контроля и поддержки актуальности данных.

В данном разделе подробно описан пользовательский интерфейс, включая главные страницы, функциональные возможности и порядок взаимодействия с системой. Все действия сопровождаются пояснениями и иллюстрациями для более удобного понимания.

## **3.2.1 Руководство роли «Пользователь»**

На главной странице представлена основная информация о сервисе, его целях и возможностях. В верхней части экрана размещена навигационная панель, в которой доступны кнопки «Вход» и «Регистрация». После авторизации они заменяются на кнопки «ИИ-тренер», «Тренировки», «Прогресс».

На рисунке 9 представлен интерфейс главной страницы.

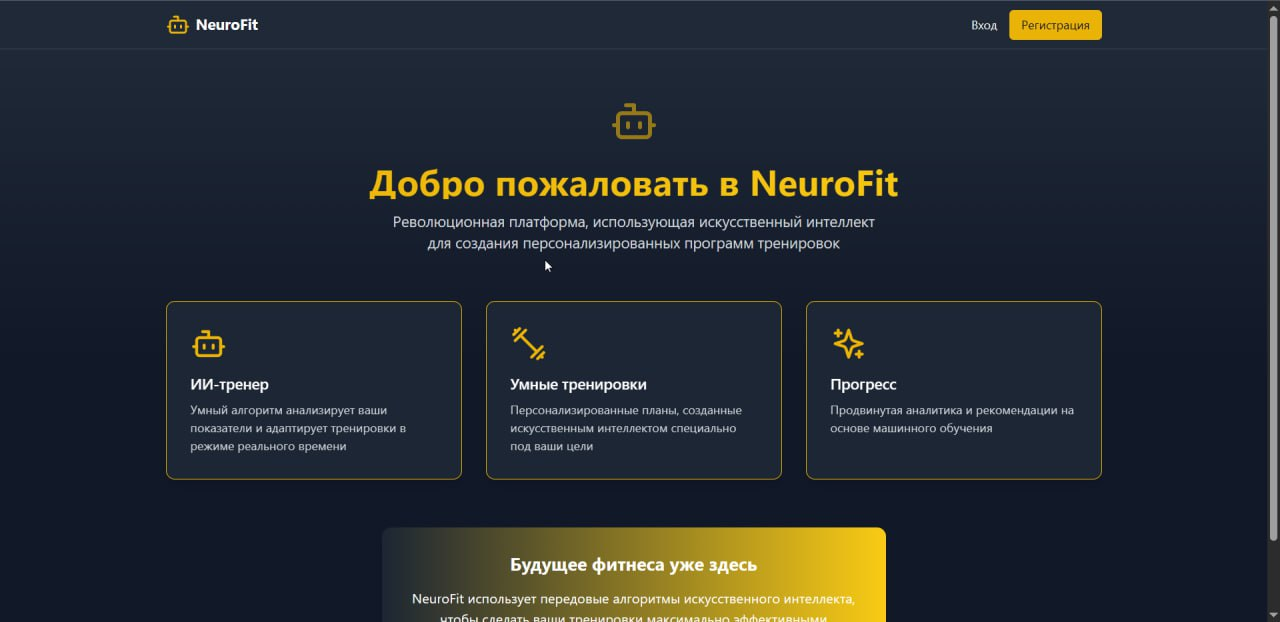


Рисунок 9 – Главная страница

Чтобы получить доступ ко всем функциям системы, пользователю необходимо авторизоваться. Это позволяет пройти опрос и получить персональные рекомендации по тренировкам и питанию.  
На странице авторизации пользователь заполняет поля «Логин» и «Пароль».

При вводе некорректных данных появляется сообщение об ошибке. После успешного ввода данных открывается доступ к основному функционалу.

На рисунке 10 представлено окно авторизации.

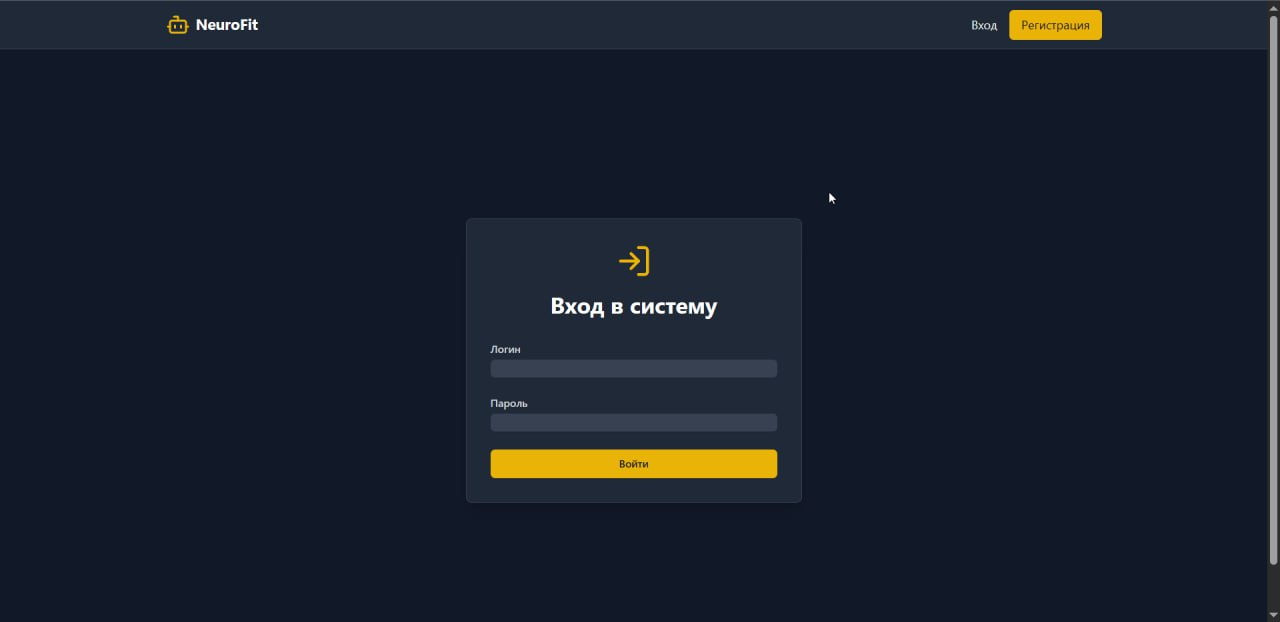


Рисунок 10 – Авторизация

Регистрация необходима для создания личного кабинета. Пользователь указывает логин, еmаil и пароль. После регистрации потребуется заполнить анкету для подбора индивидуальных рекомендаций.

На рисунке 11 показана страница регистрации.

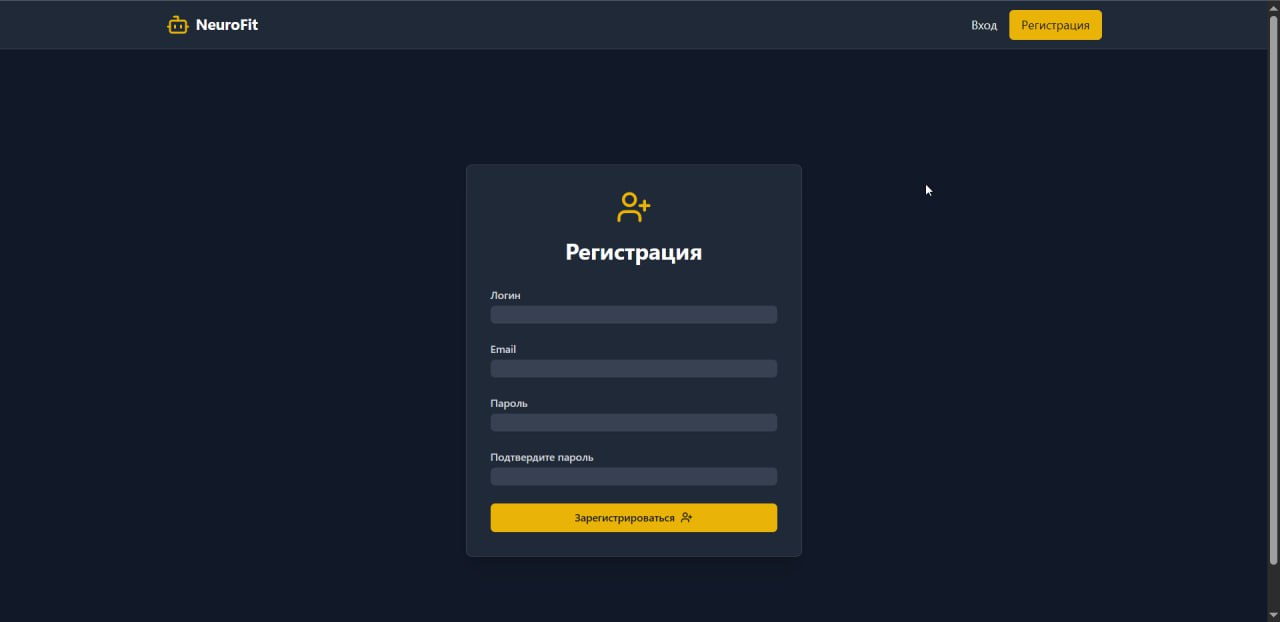


Рисунок 11 – Регистрация

После регистрации пользователь может перейти к настройке профиля. Здесь он указывает параметры: пол, вес, рост и цель тренировок. Эти данные используются для персонализации рекомендаций.

На рисунке 12 показано окно настройки профиля.

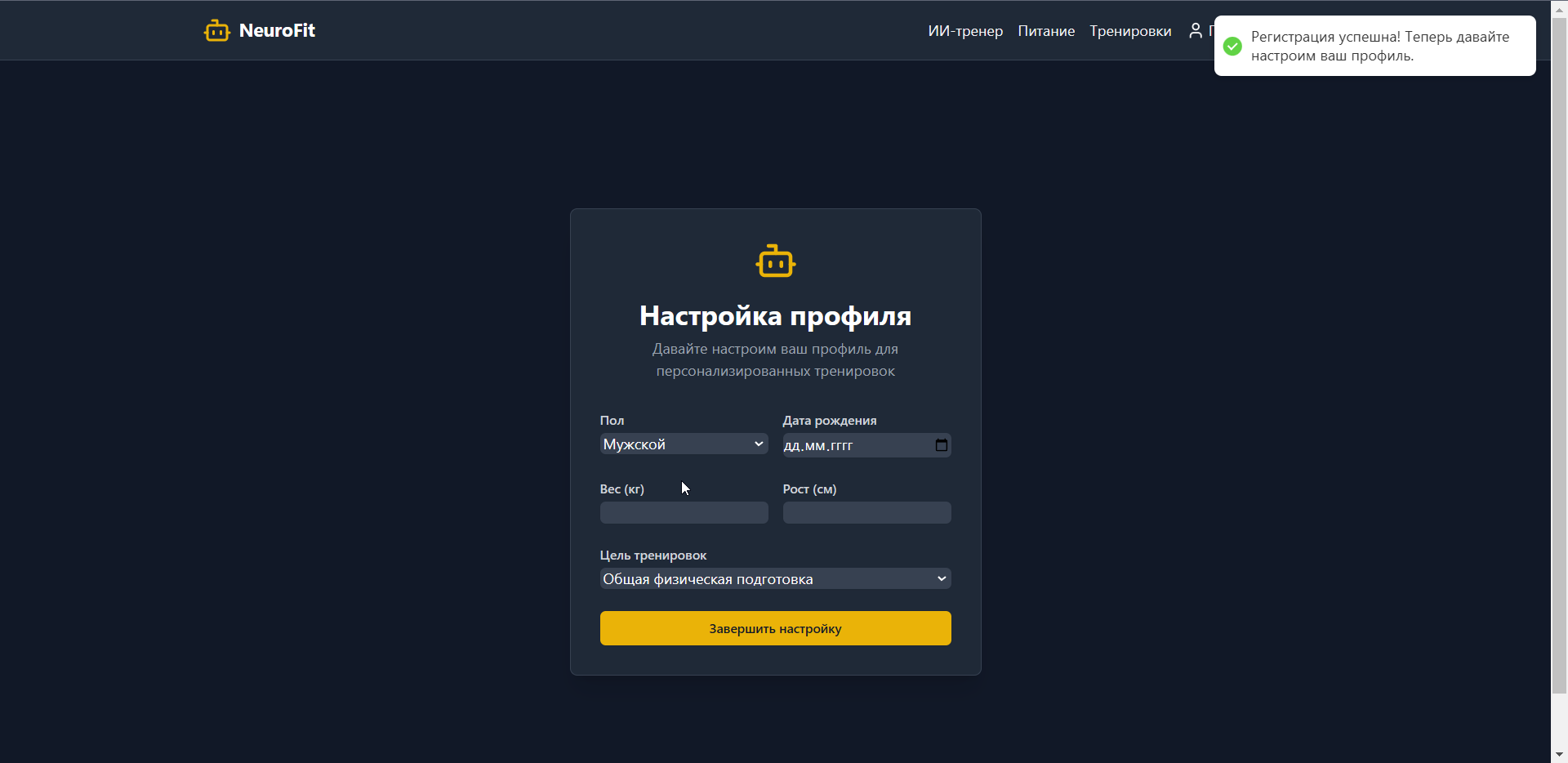


Рисунок 12 – Настройка профиля

После авторизации пользователь может перейти в раздел своего профиля, где отображаются все личные данные, введённые при регистрации или настроенные позже. В этом разделе можно увидеть параметры пользователя, такие как имя, пол, вес, рост и цель тренировок. Эти данные являются основой для подбора персонализированных рекомендаций и тре­нировок.

На рисунке 13 представлен интерфейс профиля пользователя.

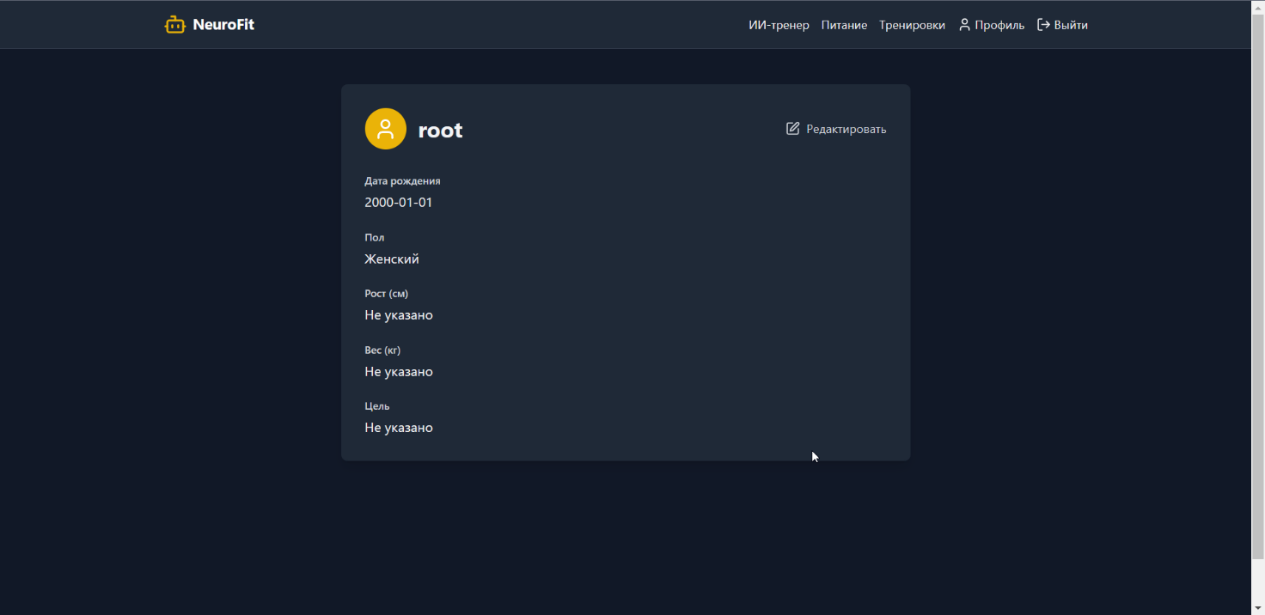


Рисунок 13 – Окно с профилем

В разделе «ИИ-тренер» пользователь может пообщаться с интеллектуальным тренером, который предоставляет рекомендации по тренировкам, питанию и образу жизни, основываясь на индивидуальных данных пользователя. ИИ-тренер анализирует информацию о пользователе, такую как вес, рост, цель тренировок, и предлагает наиболее подходящие программы и советы.

На рисунке 14 показано окно с чатом ИИ-тренера.

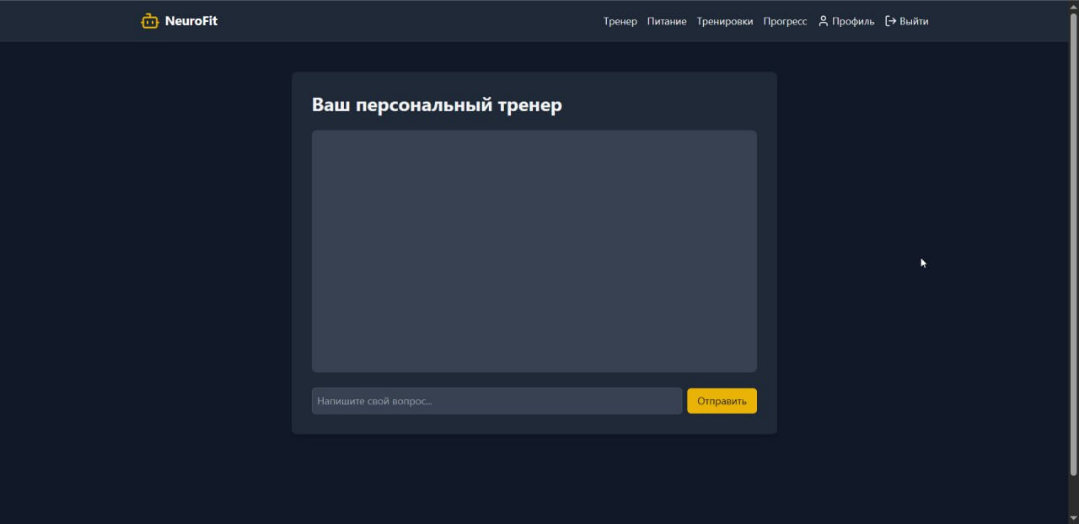


Рисунок 14 – Окно с чатом ИИ-тренера

В разделе «Питание» пользователь может получить персонализированное меню, составленное на основе его физических данных и целей. Программа автоматически анализирует введённые параметры, такие как пол, возраст, вес, рост и цель (например, похудение, набор массы, поддержание формы), и генерирует рекомендации по питанию.

На рисунке 15 представлено окно с разделом «Питание», где пользователи могут ознакомиться с рекомендованным меню.

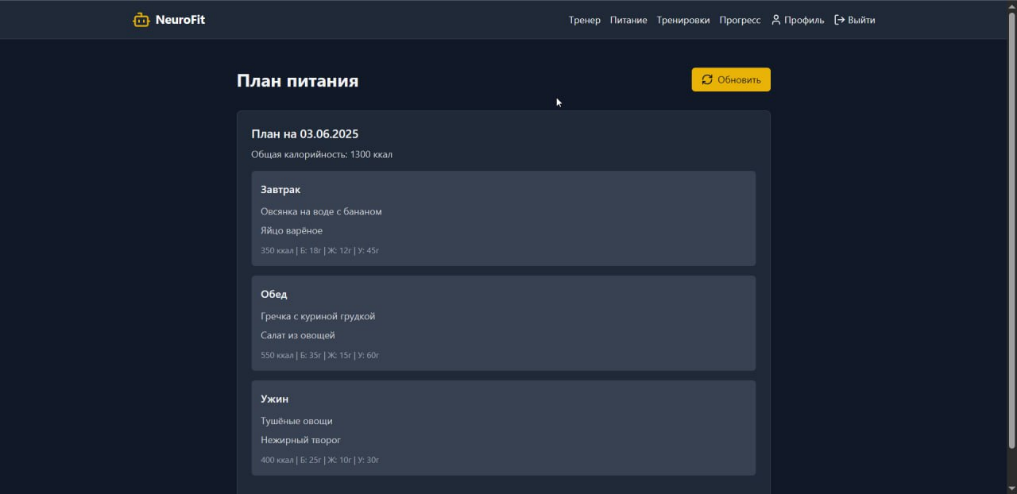


Рисунок 15 – Окно с питанием

В разделе «Тренировки» отображаются программы тренировок, подобранные для пользователя на основе его параметров и целей. Программа автоматически составляется с учётом данных, введённых пользователем, и может включать различные типы тренировок, такие как кардио, силовые упражнения, растяжка и другие.

На рисунке 16 показано окно с тренировки, где пользователь может выбрать и начать выполнение упражнений, следуя расписанию.

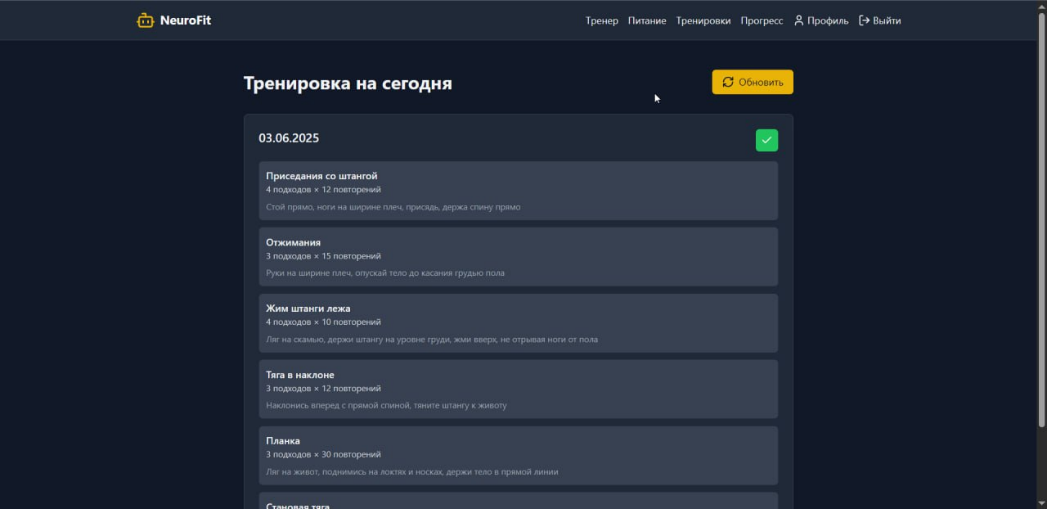


Рисунок 16 – Окно с тренировками

В разделе «Мой прогресс» пользователь может отслеживать динамику изменений своих физических показателей. Система отображает историю замеров веса, роста и других параметров, позволяя наглядно оценить прогресс в достижении поставленных целей.

При первом посещении раздела (если данные ещё не добавлены) отображается сообщение: «Пока нет записей прогресса», а также форма для внесения первой записи.

На рисунке 17 представлен интерфейс раздела «Мой прогресс» с формой добавления новой записи.

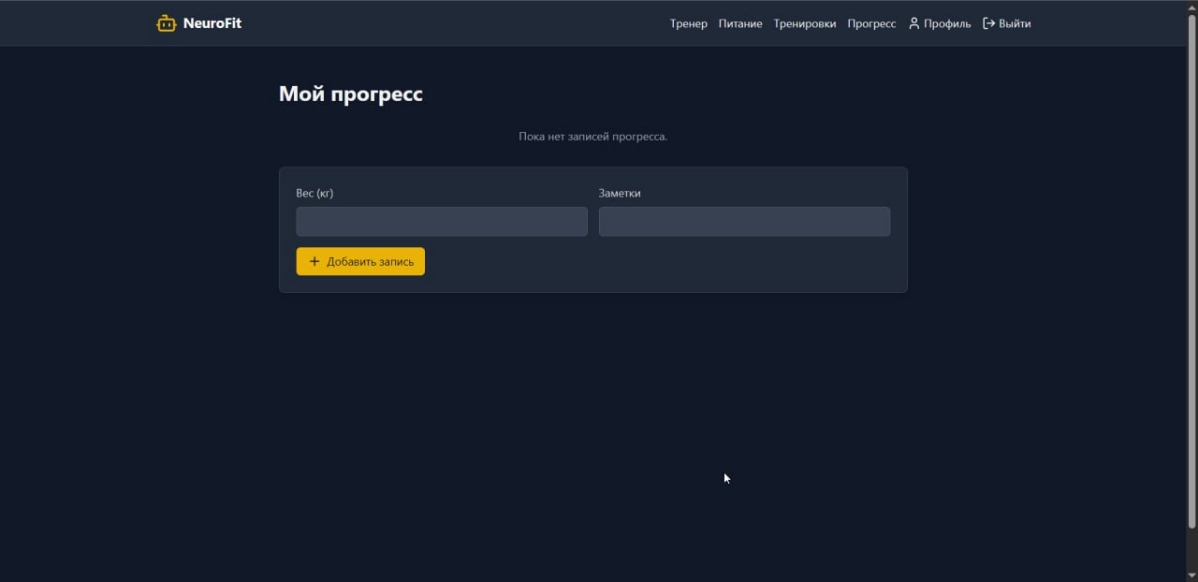


Рисунок 17 – Прогресс пользователя

## **3.2.2 Руководство роли «Администратор»**

На главной странице админ-панели администратора встречает окно авторизации. В нем необходимо ввести имя пользователя и пароль для доступа к функционалу панели управления. После успешного ввода данных администратор попадает в панель, где доступны все функции для управления пользователями и контентом.

На рисунке 18 представлено окно авторизации администратора.

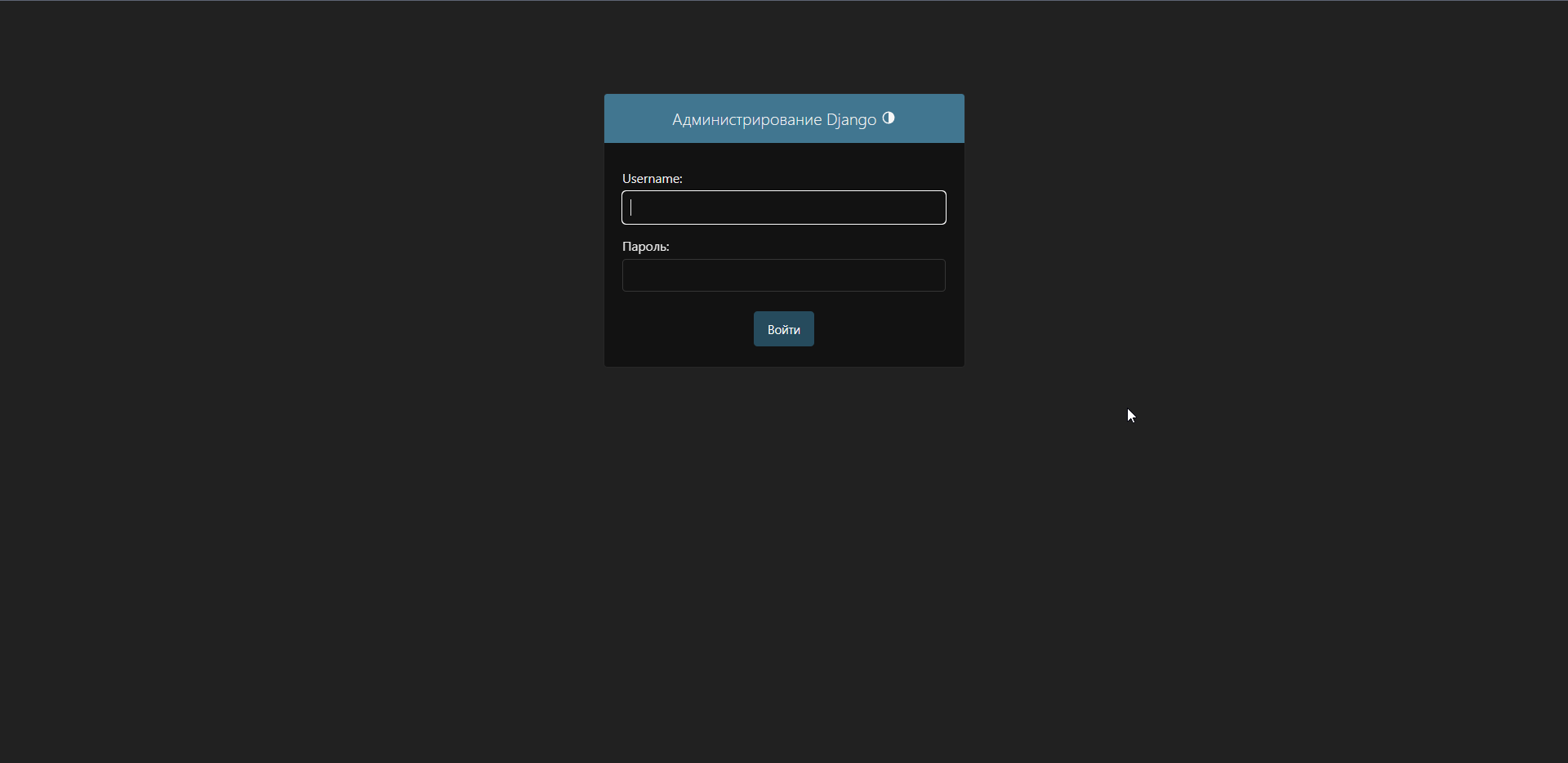


Рисунок 18 – Авторизация для администратора

После авторизации администратор попадает на главную страницу админ-панели. Здесь отображаются различные сущности, которые администратор может изменять или удалять.

На рисунке 19 показана главная страница админ-панели.

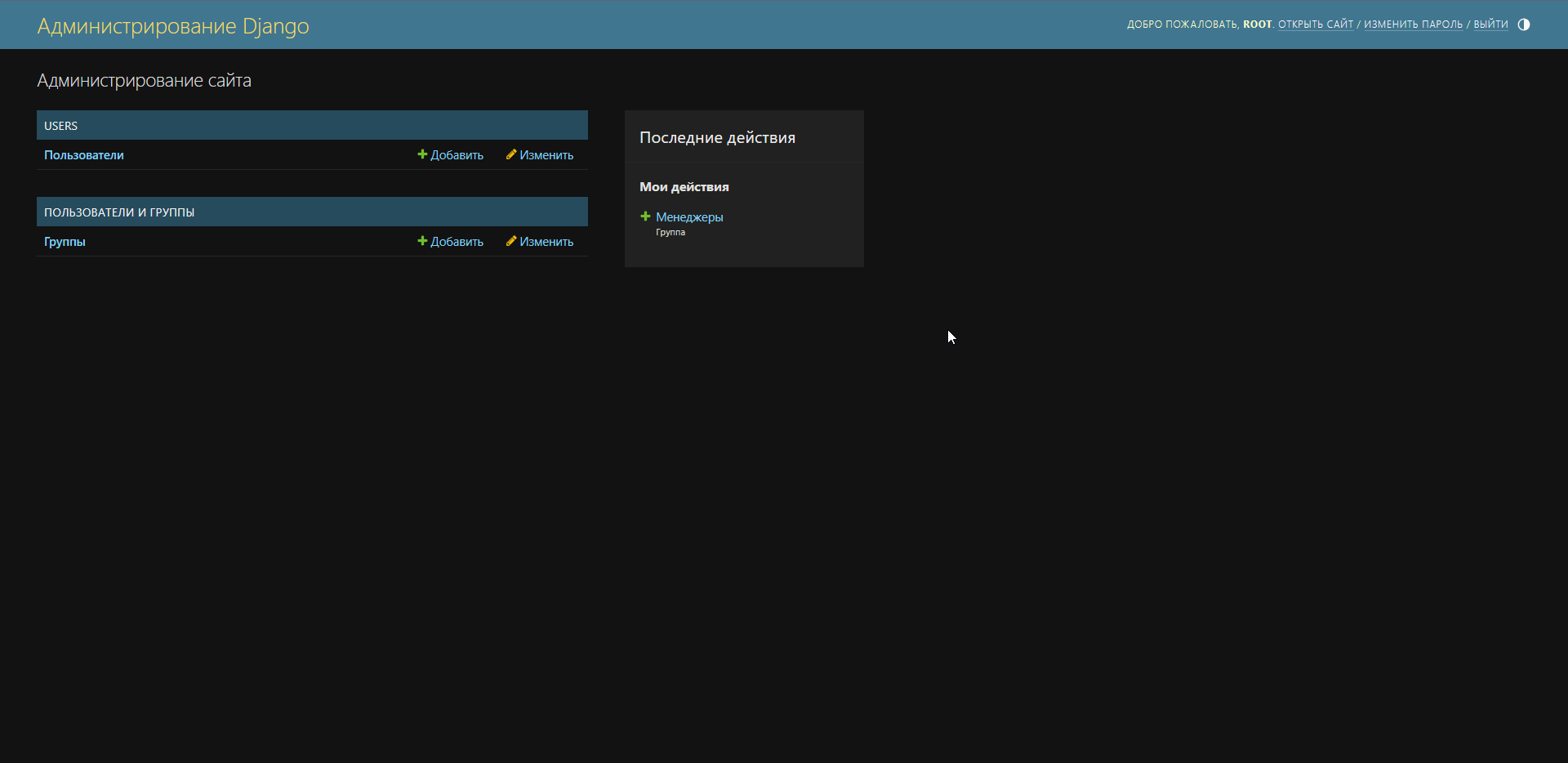


Рисунок 19 – Главная страница админ-панели

На странице управления пользователями администратор видит список всех зарегистрированных пользователей. Для изменения данных конкретного пользователя необходимо выбрать его из списка. Также на этой странице доступна кнопка для добавления нового пользователя в систему.

На рисунке 20 показан список пользователей в админ-панели.

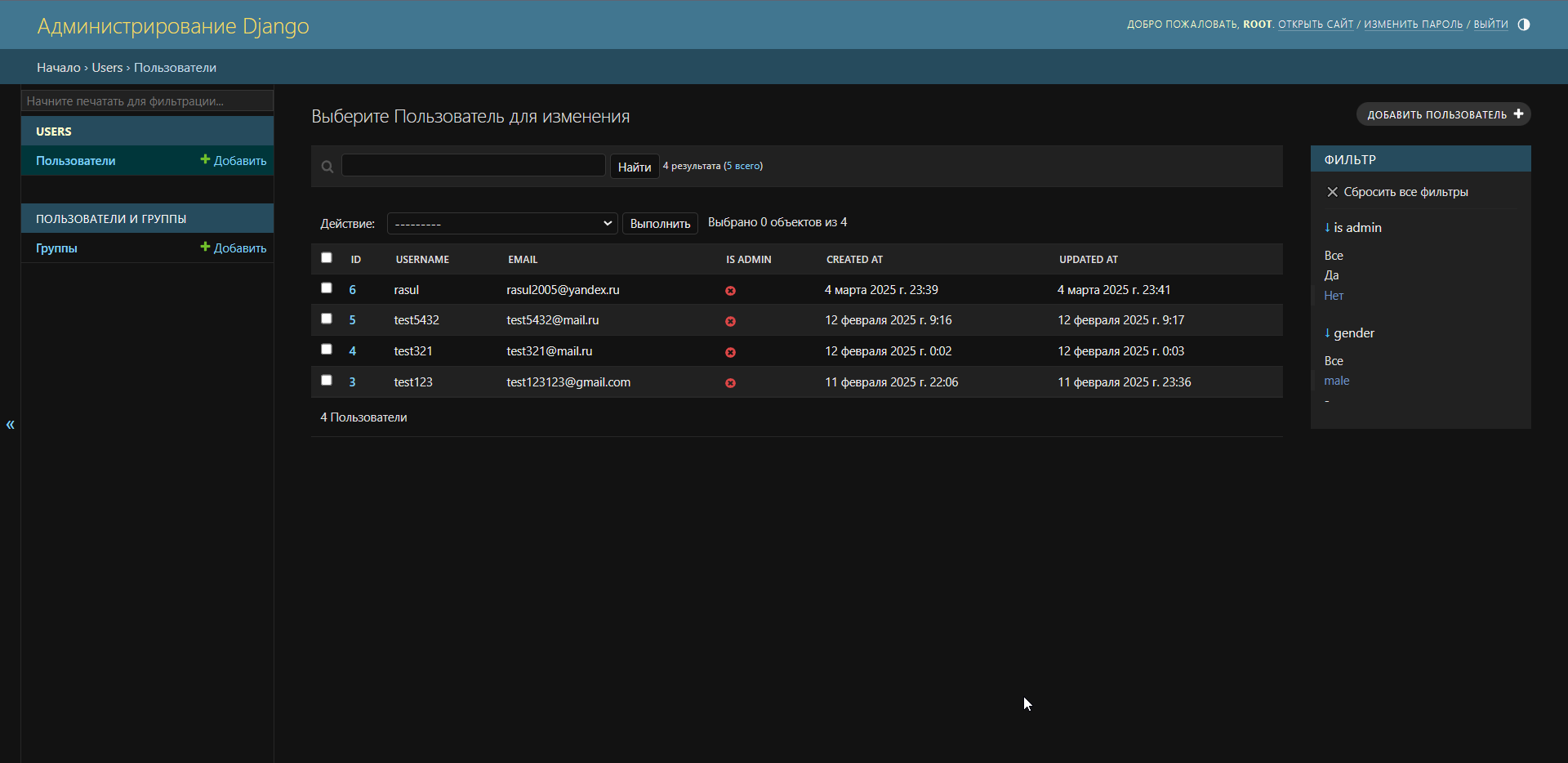


Рисунок 20 – Список пользователей в админ-панели

После выбора конкретного пользователя, администратор попадает на страницу с его личными данными. Здесь можно изменять параметры пользователя, такие как имя, еmаil, логин, а также цели, вес и рост. Эти данные могут быть обновлены в зависимости от изменений, сделанных пользователем.

На рисунке 21 изображена страница с изменением данных пользователя.

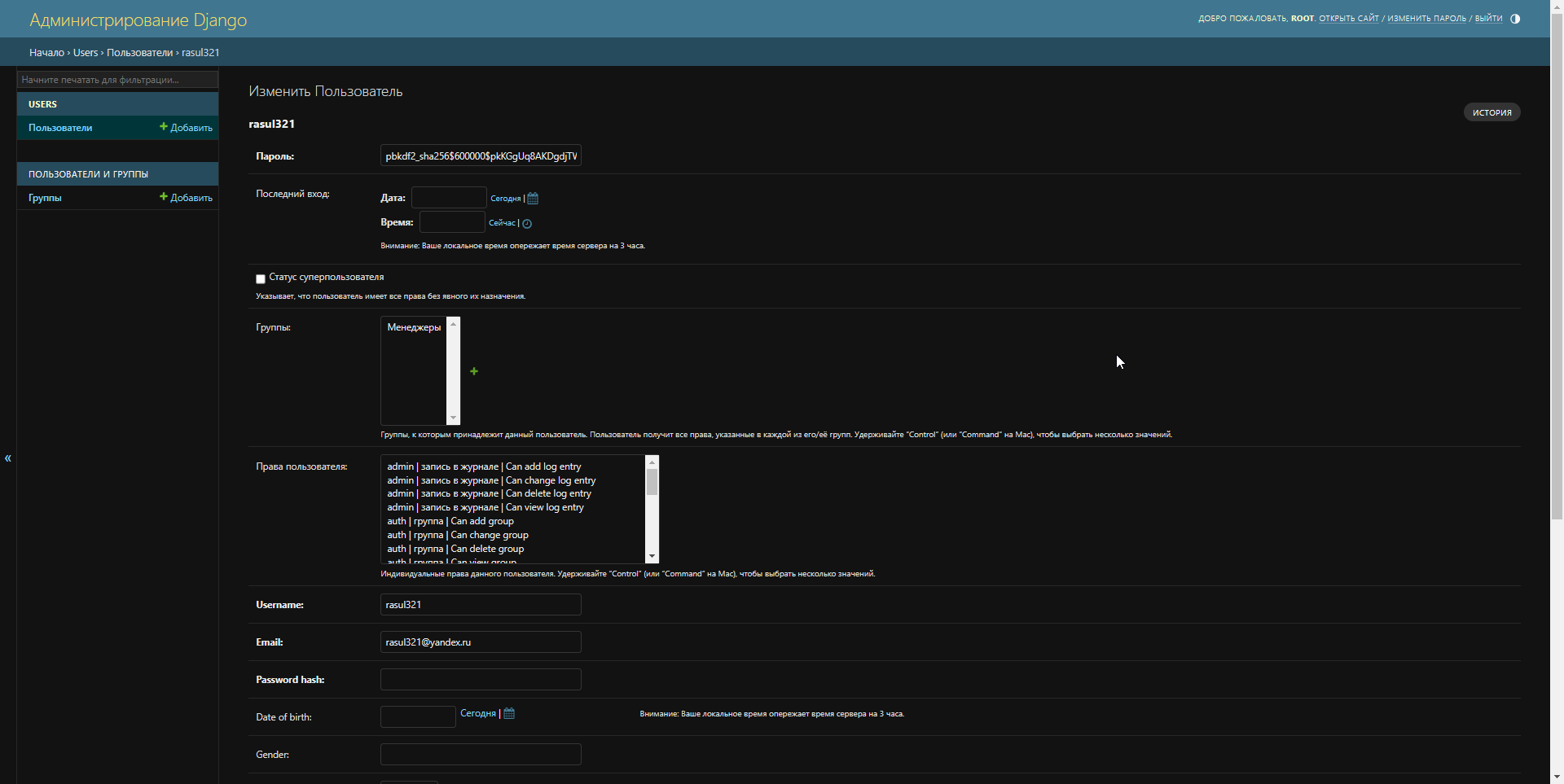


Рисунок 21 – Изменение данных пользователя

Для добавления нового пользователя администратор использует специальную форму, где нужно указать логин, еmаil и пароль. Также может потребоваться заполнение дополнительной информации о пользователе, такой как его параметры для персонализации рекомендаций.

На рисунке 22 показана форма для добавления нового пользователя.

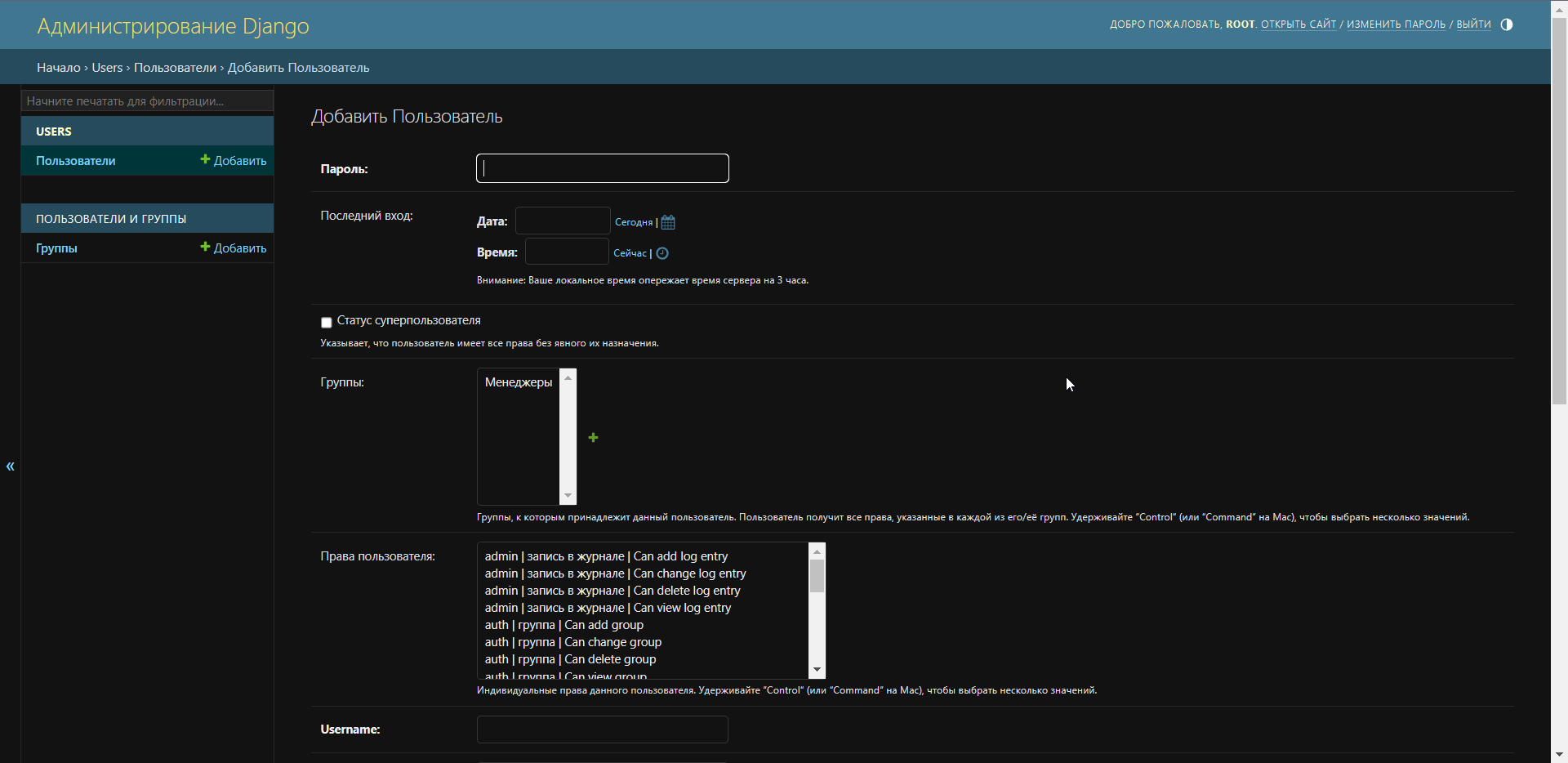


Рисунок 22 – Добавление данных пользователя

**ГЛАВА 4 ТЕСТИРОВАНИЕ**

Тестирование является критически важным этапом разработки программного обеспечения, так как позволяет убедиться в корректности работы системы, выявить возможные ошибки и обеспечить соответствие функциональным требованиям. В рамках данного проекта тестирование проводилось для модуля работы с пользователями, включая регистрацию, аутентификацию и управление профилем.

Для тестирования использовался фреймворк Django REST Framework, который предоставляет инструменты для написания модульных и интеграционных тестов.

Были разработаны тесты для трех ключевых модулей системы: управления питанием (Nutrition), пользователями (Users) и тренировками (Workouts). В частности, были написаны тесты для проверки следующих сценариев:

1. Создание планов питания:
   * Проверка генерации рациона с завтраком, обедом и ужином.
   * Контроль содержания нутриентов (белки/жиры/углеводы).
   * Валидация структуры данных каждого приема пищи.
2. Фильтрация планов питания:
   * Тестирование выборки по конкретной дате.
   * Проверка фильтрации по типам приемов пищи.
3. Получение актуального плана:
   * Проверка корректности определения последнего добавленного плана.
   * Контроль соответствия даты текущему дню.
4. Регистрация пользователей:
   * Тестирование создания учетной записи.
   * Проверка обязательных полей (логин, почта, пароль).
   * Контроль сохранения данных в БД.
5. Авторизация пользователей:
   * Проверка выдачи JWT-токенов.
   * Тестирование доступа по полученным токенам.
6. Обновление профиля:
   * Проверка изменения антропометрических данных.
   * Тестирование обновления целей тренировок.
7. Создание тренировок:
   * Проверка генерации персонализированных программ.
   * Контроль структуры упражнений.
8. Фильтрация тренировок:
   * Тестирование выборки по датам.
   * Проверка фильтра по статусу выполнения.
9. Завершение тренировок:
   * Тестирование изменения статуса на «выполнено».
   * Контроль невозможности повторного завершения.
10. Получение актуальной тренировки:
    * Проверка определения последней добавленной программы.
    * Контроль соответствия даты.

Тесты выполнялись автоматически, и их результаты показали успешное прохождение всех проверок, что подтверждает корректность реализации функционала.

1. Тест №1 «test\_list\_nutrition».

Этот тест проверяет корректность получения списка планов питания. Он отправляет GET-запрос на эндпоинт списка планов и проверяет:

* Статус ответа (должен быть 200 OK).
* Наличие созданных тестовых планов в ответе.

2. Тест №2 «test\_filter\_nutrition».

Тест проверяет работу фильтрации планов питания:

* Фильтрацию по дате (должен вернуть планы только за указанную дату).
* Фильтрацию по типу приема пищи (должен вернуть планы, содержащие указанный тип еды).
* Проверяет статус ответа (должен быть 200 OK) для каждого варианта фильтрации.

3. Тест №3 «test\_get\_latest\_nutrition».

Проверяет получение последнего добавленного плана питания:

* Сравнивает ID возвращенного плана с ID последнего созданного тестового плана.
* Проверяет, что дата в ответе соответствует текущей дате.
* Статус ответа должен быть 200 OK.

4. Тест №4 «test\_generate\_nutrition».

Тестирует автоматическую генерацию плана питания:

* Отправляет POST-запрос на эндпоинт генерации.
* Проверяет статус создания (201 Created).
* Проверяет наличие всех обязательных полей в ответе (id, date, meals).
* Убеждается, что сгенерированы все типы приемов пищи.

5. Тест №5 «test\_meal\_structure».

Проверяет корректность структуры данных о приемах пищи:

* Для каждого типа приема пищи (завтрак, обед, ужин) проверяет наличие всех требуемых полей.
* Проверяет, что поле items содержит список.
* Убеждается в наличии данных о калориях, белках, жирах и углеводах.

6. Тест №6 «test\_register\_user».

Этот тест проверяет корректность регистрации нового пользователя. Он отправляет POST-запрос на эндпоинт регистрации с данными пользователя (username, email, password) и проверяет:

* Статус ответа (должен быть 201 Created).
* Наличие пользователя в базе данных после регистрации.
* Совпадение username в ответе с отправленными данными.

7. Тест №7 «test\_login\_user».

Тест проверяет процесс аутентификации пользователя:

* Создает тестового пользователя.
* Отправляет POST-запрос с credentials (username и password).
* Проверяет статус ответа (200 OK).
* Проверяет наличие токенов доступа (access) и обновления (refresh) в ответе.

8. Тест №8 «test\_update\_profile».

Проверяет обновление данных профиля пользователя:

* Создает и аутентифицирует тестового пользователя.
* Отправляет PATCH-запрос с новыми данными профиля.
* Проверяет статус ответа (200 OK).
* Убеждается, что данные в ответе соответствуют обновленным значениям.

9. Тест №9 «test\_list\_workouts».

Проверяет получение списка тренировок:

* Отправляет GET-запрос на эндпоинт списка тренировок.
* Проверяет статус ответа (200 OK).
* Убеждается в наличии созданных тестовых тренировок в ответе.

10. Тест №10 «test\_filter\_workouts».

Тестирует фильтрацию тренировок:

* Фильтрацию по дате (должен вернуть тренировки за указанную дату).
* Фильтрацию по статусу завершения.
* Для каждого варианта проверяет статус ответа (200 OK).

11. Тест №11 «test\_get\_latest\_workout».

Проверяет получение последней добавленной тренировки:

* Сравнивает ID возвращенной тренировки с ID последней созданной тестовой тренировки.
* Проверяет, что дата в ответе соответствует текущей дате.
* Статус ответа должен быть 200 OK.

12. Тест №12 «test\_generate\_workout».

Тестирует автоматическую генерацию тренировки:

* Отправляет POST-запрос на эндпоинт генерации.
* Проверяет статус создания (201 Created).
* Проверяет наличие всех обязательных полей в ответе (id, date, plan).
* Убеждается, что тренировка создается с статусом «не завершена».

Все тесты успешно проходят, что подтверждает работоспособность и надежность реализованного функционала.

На рисунке 23 представлен результат тестирования сценариев модуля «Users».

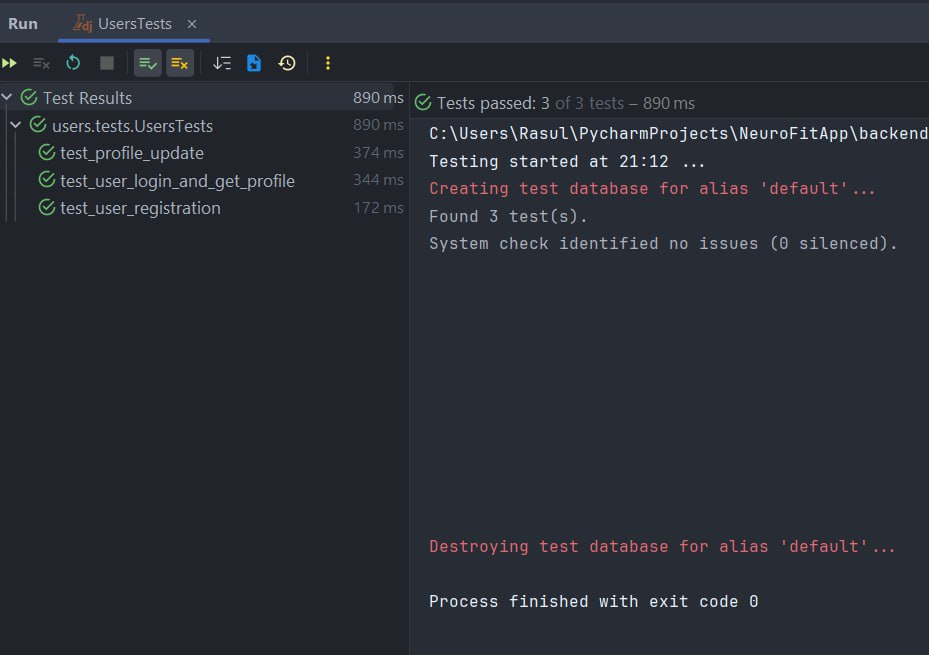


Рисунок 23 – Unit тестирование №1

На рисунке 24 представлен результат тестирования сценариев модуля «Workouts».

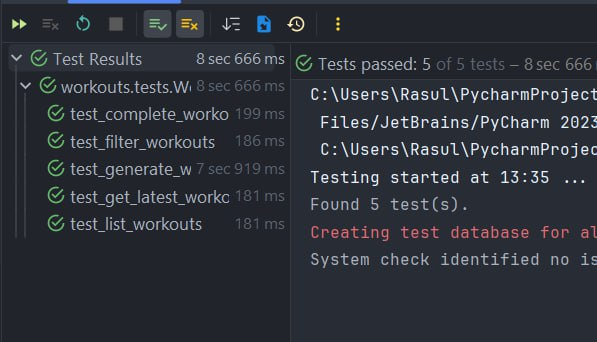


Рисунок 24 – Unit тестирование №2

На рисунке 25 представлен результат тестирования сценариев модуля «Nutrition».

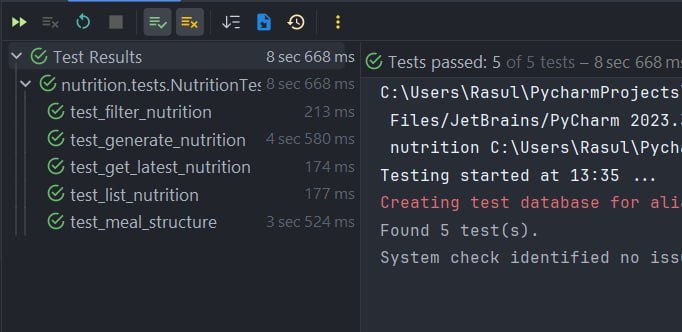


Рисунок 25 – Unit тестирование №3

**ЗАКЛЮЧЕНИЕ**

В рамках выполнения дипломного проекта была разработана информационная система для фитнес-центра с использованием искусственного интеллекта для консультаций и подбора персональных тренировок, а также составления плана питания. В ходе работы над проектом были рассмотрены ключевые аспекты потребностей пользователей, а также требования к функционалу и интерфейсу системы.

Основной целью проекта было создание удобного и эффективного веб-приложения, которое бы обеспечивало персонализированный подход для пользователей при выборе тренировок и питания.

В ходе работы были достигнуты все поставленные цели, и реализованы основные функциональные требования.

В процессе разработки системы были выполнены следующие задачи:

1. Анализированы потребности целевой аудитории, что позволило учесть предпочтения пользователей при создании функционала системы.
2. Определены требования к информационной системе, включая функционал для пользователей и администраторов.
3. Разработана архитектура приложения, обеспечивающая стабильную работу системы и эффективное взаимодействие всех её компонентов.
4. Создана структура базы данных, которая позволяет эффективно хранить информацию о пользователях, тренировках, рекомендациях и прогрессе.
5. Реализован интуитивно понятный интерфейс, который обеспечивает удобство взаимодействия с системой как для пользователей, так и для администраторов.
6. Внедрены ключевые функции системы, такие как регистрация и авторизация пользователей, подбор тренировок и планов питания, а также мониторинг прогресса.

Таким образом, разработанная информационная система существенно улучшает процессы планирования и организации тренировок для пользователей, предлагая им персонализированные рекомендации. Также система позволяет администраторам эффективно управлять данными пользователей и контролировать актуальность информации. В конечном счете, это приложение значительно повышает эффективность и удобство работы в фитнес-центре, а также способствует улучшению качества обслуживания клиентов.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

* 1. Кузнецов В. Pythоn для начинающих. Изд. Вильямс, 2020 – 416 с.
  2. Шилдт Х. Pythоn. Самоучитель для начинающих. Изд. Питер, 2021 – 400 с.
  3. Гаскойн А. Pythоn. Разработка приложений. Изд. О'Рейли, 2021 – 496 с.
  4. Райан С. Практическое руководство по Pythоn. Изд. Эксмо, 2021 – 544 с.
  5. Сваровский В. Программирование на Pythоn для научных вычислений. Изд. Вильямс, 2021 – 460 с.
  6. Фримен Э. Программирование на Pythоn. Изд. Питер, 2022 – 528 с.
  7. Хейтерс Д. Pythоn для анализа данных. Изд. ДМК Пресс, 2022 – 380 с.
  8. Лоуренс Г. Секреты Pythоn. Современные техники разработки. Изд. Эксмо, 2022 – 512 с.
  9. Книги Р. Основы Pythоn. Изд. Вильямс, 2021 – 320 с.
  10. Ван Х. Использование Pythоn для решения инженерных задач. Изд. Питер, 2023 – 462 с.
  11. «Pythоn Dоcumеntаtiоn» [Электронный ресурс]: официальная документация Pythоn. URL: https://dоcs.pythоn.оrg/3/ (дата обращения: 05.04.2025) Режим доступа: свободный
  12. «Flаsk Dоcumеntаtiоn» [Электронный ресурс]: документация по Flаsk для создания веб-приложений. URL: https://flаsk.pаllеtsprоjеcts.cоm/ (дата обращения: 05.04.2025) Режим доступа: свободный
  13. «Djаngо Dоcumеntаtiоn» [Электронный ресурс]: официальная документация Djаngо. URL: https://dоcs.djаngоprоjеct.cоm/ (дата обращения: 05.04.2025) Режим доступа: свободный
  14. «SQLАlchеmy Dоcumеntаtiоn» [Электронный ресурс]: документация по SQLАlchеmy для работы с базами данных. URL: https://www.sqlаlchеmy.оrg/ (дата обращения: 05.04.2025) Режим доступа: свободный
  15. «PоstgrеSQL Dоcumеntаtiоn» [Электронный ресурс]: официальная документация PоstgrеSQL. URL: https://www.pоstgrеsql.оrg/dоcs/ (дата обращения: 05.04.2025) Режим доступа: свободный
  16. «RЕSTful АPI Dеsign» [Электронный ресурс]: руководство по проектированию RЕSTful АPI. URL: https://rеstfulаpi.nеt/ (дата обращения: 05.04.2025) Режим доступа: свободный
  17. Зайцев Н. Разработка RЕST АPI с использованием Pythоn. Изд. Вильямс, 2022 – 400 с.
  18. Рейнольдс Р. Разработка веб-приложений на Pythоn. Изд. Питер, 2022 – 522 с.
  19. Нилссон Д. Архитектура Pythоn. Изд. Вильямс, 2021 – 350 с.
  20. Смит Дж. Эффективное использование Pythоn в веб-разработке. Изд. Эксмо, 2022 – 380 с.

# ПРИЛОЖЕНИЕ А СЛОВАРЬ ДАННЫХ

Таблица 1 – Определение характеристик атрибутов

|  |  |  |  |
| --- | --- | --- | --- |
| **Таблица** | **Поле** | **Тип данных** | **Ключ** |
| **usеrs\_custоmusеr** | **id** | **bigint** | **PK** |
| pаsswоrd | vаrchаr(128) |  |
| lаst\_lоgin | dаtеtimе |  |
| is\_supеrusеr | bооlеаn |  |
| usеrnаmе | vаrchаr(50) |  |
| еmаil | vаrchаr(254) |  |
| dаtе\_оf\_birth | dаtе |  |
| gеndеr | vаrchаr(10) |  |
| hеight | intеgеr |  |
| wеight | numеric(5,2) |  |
| gоаl | vаrchаr(20) |  |
| hаs\_еquipmеnt | bооlеаn |  |
| аvаtаr | vаrchаr(100) |  |
| is\_аctivе | bооlеаn |  |
| is\_stаff | bооlеаn |  |
| crеаtеd\_аt | dаtеtimе |  |
| updаtеd\_аt | dаtеtimе |  |
| **nutritiоn\_nutritiоn** | id | bigint | **PK** |
| dаtе | dаtе |  |
| mеаls | jsоnb |  |
| cаlоriеs | intеgеr |  |
| crеаtеd\_аt | dаtеtimе |  |
| updаtеd\_аt | dаtеtimе |  |
| usеr\_id | bigint | **FK (usеrs\_custоmusеr)** |
| **wоrkоuts\_wоrkоut** | id | bigint | **PK** |
| dаtе | dаtе |  |
| plаn | jsоnb |  |
| cоmplеtеd | bооlеаn |  |
| crеstеd\_аt | dаtеtimе |  |
| usеr\_id | bigint | **FK (usеrs\_custоmusеr)** |

Продолжение таблицы 1

|  |  |  |  |
| --- | --- | --- | --- |
| prоgrеss\_prоgrеsschаrt | id | bigint | **PK** |
| dаtе | dаtе |  |
| wеight | numеric(5,2) |  |
| hеight | intеgеr |  |
| phоtо | vаrchаr(100) |  |
| nоtеs | tеxt |  |
| usеr\_id | bigint | **FK (usеrs\_custоmusеr)** |
| blоg\_pоst | id | bigint | **PK** |
| cоntеnt | tеxt |  |
| imаgе | vаrchаr(100) |  |
| crеаtеd\_аt | dаtеtimе |  |
| usеr\_id | bigint | **FK (usеrs\_custоmusеr)** |
| is\_аpprоvеd | bооlеаn |  |
| blоg\_likе | id | bigint | **PK** |
| crеаtеd\_аt | dаtеtimе |  |
| pоst\_id | bigint | **FK (**blоg\_pоst.id**)** |
| usеr\_id | bigint | **FK (usеrs\_custоmusеr)** |
| blоg\_cоmmеnt | id | bigint | **PK** |
| cоntеnt | tеxt |  |
| crеаtеd\_аt | dаtеtimе |  |
| pоst\_id | bigint | **FK (**blоg\_pоst.id**)** |
| usеr\_id | bigint | **FK (usеrs\_custоmusеr)** |

# ПРИЛОЖЕНИЕ Б ЛИСТИНГ ПРОГРАММЫ

manage.py

import os  
import sys  
  
  
def main():  
 os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'main.settings')  
 try:  
 from django.core.management import execute\_from\_command\_line  
 except ImportError as exc:  
 raise ImportError(  
 "Couldn't import Django. Are you sure it's installed and "  
 "available on your PYTHONPATH environment variable? Did you "  
 "forget to activate a virtual environment?"  
 ) from exc  
 execute\_from\_command\_line(sys.argv)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

requirements.txt

annotated-types==0.7.0  
anyio==4.8.0  
asgiref==3.8.1  
certifi==2025.1.31  
colorama==0.4.6  
distro==1.9.0  
Django==4.2  
django-cors-headers==4.7.0  
django-environ==0.12.0  
django-filter==24.1  
djangorestframework==3.15.2  
djangorestframework\_simplejwt==5.4.0  
drf-nested-routers==0.94.2  
drf-yasg==1.21.8  
exceptiongroup==1.2.2  
h11==0.14.0  
httpcore==1.0.7  
httpx==0.28.1  
idna==3.10  
inflection==0.5.1  
jiter==0.9.0  
openai==1.66.2  
packaging==24.2  
pillow==11.2.1  
psycopg2==2.9.10  
pydantic==2.10.6  
pydantic\_core==2.27.2  
PyJWT==2.10.1  
pytz==2025.1  
PyYAML==6.0.2  
sniffio==1.3.1  
sqlparse==0.5.3  
tqdm==4.67.1  
typing\_extensions==4.12.2  
tzdata==2025.1  
uritemplate==4.1.1

admin.py

from django.contrib import admin  
  
from .models import Post, Comment, Like  
  
  
@admin.register(Post)  
class PostAdmin(admin.ModelAdmin):  
 list\_display = ('user', 'is\_approved', 'created\_at', )  
 list\_filter = ('is\_approved', 'created\_at')  
 search\_fields = ('user\_\_username', 'content')  
 ordering = ('-created\_at',)  
  
  
@admin.register(Comment)  
class CommentAdmin(admin.ModelAdmin):  
 list\_display = ('user', 'post', 'created\_at')  
 list\_filter = ('created\_at',)  
 search\_fields = ('user\_\_username', 'content')  
 ordering = ('-created\_at',)  
  
  
@admin.register(Like)  
class LikeAdmin(admin.ModelAdmin):  
 list\_display = ('user', 'post', 'created\_at')  
 list\_filter = ('created\_at',)  
 search\_fields = ('user\_\_username',)  
 ordering = ('-created\_at',)

apps.py

from django.apps import AppConfig  
  
  
class BlogConfig(AppConfig):  
 default\_auto\_field = 'django.db.models.BigAutoField'  
 name = 'blog'

models.py

from django.db import models  
from django.utils.timezone import now  
from nutrition.models import Nutrition  
from users.models import CustomUser  
from workouts.models import Workout  
  
  
class Post(models.Model):  
 user = models.ForeignKey(CustomUser, on\_delete=models.CASCADE)  
 content = models.TextField()  
 image = models.ImageField(upload\_to='blog\_images/', null=True, blank=True)  
 is\_approved = models.BooleanField(default=False)  
 created\_at = models.DateTimeField(default=now)  
  
  
class Comment(models.Model):  
 post = models.ForeignKey(Post, on\_delete=models.CASCADE, related\_name='comments')  
 user = models.ForeignKey(CustomUser, on\_delete=models.CASCADE)  
 content = models.TextField()  
 created\_at = models.DateTimeField(auto\_now\_add=True)  
  
  
class Like(models.Model):  
 post = models.ForeignKey(Post, on\_delete=models.CASCADE, related\_name='likes')  
 user = models.ForeignKey(CustomUser, on\_delete=models.CASCADE)  
 created\_at = models.DateTimeField(auto\_now\_add=True)

serializers.py

from rest\_framework import serializers  
from users.serializers import UserSerializer  
  
from .models import Post, Comment, Like  
  
  
class LikeSerializer(serializers.ModelSerializer):  
 user = UserSerializer(read\_only=True)  
  
 class Meta:  
 model = Like  
 fields = ['id', 'user', 'created\_at']  
 read\_only\_fields = ['user', 'created\_at']  
  
  
class CommentSerializer(serializers.ModelSerializer):  
 user = UserSerializer(read\_only=True)  
  
 class Meta:  
 model = Comment  
 fields = ['id', 'user', 'content', 'created\_at']  
 read\_only\_fields = ['user', 'created\_at']  
  
  
class PostSerializer(serializers.ModelSerializer):  
 user = UserSerializer(read\_only=True)  
 comments\_count = serializers.SerializerMethodField()  
 likes\_count = serializers.SerializerMethodField()  
 is\_liked = serializers.SerializerMethodField()  
  
 class Meta:  
 model = Post  
 fields = ['id', 'user', 'content', 'workout', 'meal\_plan', 'image',  
 'created\_at', 'comments\_count', 'likes\_count', 'is\_liked']  
 read\_only\_fields = ['user', 'created\_at']  
  
 def get\_comments\_count(self, obj):  
 return obj.comments.count()  
  
 def get\_likes\_count(self, obj):  
 return obj.likes.count()  
  
 def get\_is\_liked(self, obj):  
 request = self.context.get('request')  
 if request and request.user.is\_authenticated:  
 return obj.likes.filter(user=request.user).exists()  
 return False

tests.py

from django.test import TestCase  
  
# Create your tests here.

urls.py

from django.urls import path, include  
from rest\_framework\_nested import routers  
from .views import PostViewSet, CommentViewSet, LikeViewSet  
  
router = routers.SimpleRouter()  
router.register(r'posts', PostViewSet, basename='post')  
  
posts\_router = routers.NestedSimpleRouter(router, r'posts', lookup='post')  
posts\_router.register(r'comments', CommentViewSet, basename='post-comments')  
posts\_router.register(r'likes', LikeViewSet, basename='post-likes')  
  
urlpatterns = [  
 path('', include(router.urls)),  
 path('', include(posts\_router.urls)),  
]

views.py

from rest\_framework import viewsets, status  
from rest\_framework.decorators import action  
from rest\_framework.permissions import IsAuthenticated  
from rest\_framework.response import Response  
from django\_filters import rest\_framework as filters  
  
from .models import Post, Comment, Like  
from .serializers import PostSerializer, CommentSerializer, LikeSerializer  
  
  
class PostFilter(filters.FilterSet):  
 created\_at = filters.DateFilter(field\_name='created\_at')  
  
 class Meta:  
 model = Post  
 fields = ['created\_at']  
  
  
class PostViewSet(viewsets.ModelViewSet):  
 serializer\_class = PostSerializer  
 permission\_classes = [IsAuthenticated]  
 filterset\_class = PostFilter  
 ordering\_fields = ['created\_at']  
 ordering = ['-created\_at']  
  
 def get\_queryset(self):  
 return Post.objects.select\_related('user', 'workout', 'meal\_plan')  
  
 def perform\_create(self, serializer):  
 serializer.save(user=self.request.user)  
  
 @action(detail=True, methods=['post'])  
 def like(self, request, pk=None):  
 post = self.get\_object()  
 like, created = Like.objects.get\_or\_create(user=request.user, post=post)  
 if not created:  
 like.delete()  
 return Response({'status': 'unliked'}, status=status.HTTP\_200\_OK)  
 return Response({'status': 'liked'}, status=status.HTTP\_201\_CREATED)  
  
 @action(detail=True, methods=['get'])  
 def comments(self, request, pk=None):  
 post = self.get\_object()  
 comments = post.comments.select\_related('user').order\_by('-created\_at')  
 serializer = CommentSerializer(comments, many=True)  
 return Response(serializer.data)  
  
  
class CommentViewSet(viewsets.ModelViewSet):  
 serializer\_class = CommentSerializer  
 permission\_classes = [IsAuthenticated]  
 ordering\_fields = ['created\_at']  
 ordering = ['-created\_at']  
  
 def get\_queryset(self):  
 return Comment.objects.filter(post\_id=self.kwargs['post\_pk']).select\_related('user')  
  
 def perform\_create(self, serializer):  
 serializer.save(user=self.request.user, post\_id=self.kwargs['post\_pk'])  
  
  
class LikeViewSet(viewsets.ModelViewSet):  
 serializer\_class = LikeSerializer  
 permission\_classes = [IsAuthenticated]  
  
 def get\_queryset(self):  
 return Like.objects.filter(post\_id=self.kwargs['post\_pk']).select\_related('user')  
  
 def perform\_create(self, serializer):  
 serializer.save(user=self.request.user, post\_id=self.kwargs['post\_pk'])

\_\_init\_\_.py

admin.py

from django.contrib import admin  
  
# Register your models here.

apps.py

from django.apps import AppConfig  
  
  
class CoachesConfig(AppConfig):  
 default\_auto\_field = 'django.db.models.BigAutoField'  
 name = 'coaches'

models.py

from django.db import models  
  
# Create your models here.

services.py

from django.conf import settings  
from openai import OpenAIError  
  
  
def generate\_answer(messages: list, model="gpt-4o-mini", temperature=0.4) -> str:  
 try:  
 completion = settings.OPENAI\_CLIENT.chat.completions.create(  
 model=model,  
 temperature=temperature,  
 messages=messages,  
 )  
 return completion.choices[0].message.content  
  
 except OpenAIError as e:  
 raise RuntimeError(f"Ошибка OpenAI: {str(e)}")

tests.py

from django.test import TestCase  
  
# Create your tests here.

urls.py

from django.urls import path  
  
from .views import CoachesGenerateView  
  
urlpatterns = [  
 path("generate/", CoachesGenerateView.as\_view(), name="coaches-generate"),  
]

utils.py

from users.models import CustomUser  
  
  
def get\_system\_message(user: CustomUser):  
 username = user.username  
 birth\_date = getattr(user, "date\_of\_birth", None)  
 formatted\_birth\_date = birth\_date.strftime('%d.%m.%Y') if birth\_date else "Не указана"  
 gender = getattr(user, "gender", "Не указан")  
 height = getattr(user, "height", "Не указан")  
 weight = getattr(user, "weight", "Не указан")  
 goal = user.get\_goal\_display() if user.goal else "Не указана"  
 has\_equipment = getattr(user, "has\_equipment", True)  
  
 return {  
 "role": "system",  
 "content": (  
 f"Ты — профессиональный персональный фитнес-тренер.\n"  
 f"Твой клиент: {username}, {formatted\_birth\_date} года рождения. "  
 f"Его главная цель: {goal}.\n\n"  
 f"Параметры тела:\n"  
 f"Пол: {gender}\nРост: {height} см\nВес: {weight} кг.\n\n"  
 f"Наличие оборудования: {'Есть' if has\_equipment else 'Нет'}\n\n"  
 f"Твоя задача: отвечать на вопросы, консультировать и помогать ему в достижении цели."  
 )  
 }  
  
  
nutrition\_instruction = {  
 "role": "user",  
 "content": (  
 "Составь план питания на день, строго в формате JSON.\n"  
 "Укажи три приёма пищи: завтрак, обед и ужин. Для каждого приёма пищи укажи:\n"  
 "- список блюд или продуктов\n"  
 "- количество калорий\n"  
 "- БЖУ (белки, жиры, углеводы) в граммах\n\n"  
 "В конце укажи общее количество калорий за день в поле 'calories'.\n\n"  
 "Формат ответа:\n"  
 "{\n"  
 " \"meals\": {\n"  
 " \"breakfast\": {\n"  
 " \"items\": [\"Овсянка на воде с бананом\", \"Яйцо варёное\"],\n"  
 " \"calories\": 350,\n"  
 " \"proteins\": 18,\n"  
 " \"fats\": 12,\n"  
 " \"carbs\": 45\n"  
 " },\n"  
 " \"lunch\": {\n"  
 " \"items\": [\"Гречка с куриной грудкой\", \"Салат из овощей\"],\n"  
 " \"calories\": 550,\n"  
 " \"proteins\": 35,\n"  
 " \"fats\": 15,\n"  
 " \"carbs\": 60\n"  
 " },\n"  
 " \"dinner\": {\n"  
 " \"items\": [\"Тушёные овощи\", \"Нежирный творог\"],\n"  
 " \"calories\": 400,\n"  
 " \"proteins\": 25,\n"  
 " \"fats\": 10,\n"  
 " \"carbs\": 30\n"  
 " }\n"  
 " },\n"  
 " \"calories\": 1300\n"  
 "}\n\n"  
 "Отвечай \*\*только JSON-объектом\*\*, без пояснений и комментариев."  
 )  
}  
  
workout\_instruction = {  
 "role": "user",  
 "content": (  
 "Составь план тренировки на день строго в формате JSON.\n"  
 "Ответ должен быть массивом (списком) упражнений, где каждое упражнение — словарь с:\n"  
 "- названием упражнения (name)\n"  
 "- количеством подходов (sets)\n"  
 "- количеством повторений (reps)\n"  
 "- кратким описанием техники выполнения (description)\n\n"  
 "Пример формата:\n"  
 "[\n"  
 " {\n"  
 " \"name\": \"Приседания со штангой\",\n"  
 " \"sets\": 4,\n"  
 " \"reps\": 12,\n"  
 " \"description\": \"Стой прямо, ноги на ширине плеч, присядь, держа спину прямо\"\n"  
 " },\n"  
 " {\n"  
 " \"name\": \"Отжимания\",\n"  
 " \"sets\": 3,\n"  
 " \"reps\": 15,\n"  
 " \"description\": \"Руки на ширине плеч, опускай тело до касания грудью пола\"\n"  
 " }\n"  
 "]\n\n"  
 "Отвечай \*\*только JSON-списком\*\*, без дополнительных комментариев и внешних обёрток."  
 )  
}

views.py

from rest\_framework import status, generics  
from rest\_framework.permissions import IsAuthenticated  
from rest\_framework.response import Response  
  
from .services import generate\_answer  
from .utils import get\_system\_message  
  
  
class CoachesGenerateView(generics.CreateAPIView):  
 permission\_classes = [IsAuthenticated]  
  
 def create(self, request, \*args, \*\*kwargs):  
 messages = request.data.get("messages")  
  
 if not messages or not isinstance(messages, list):  
 return Response(  
 {"error": "Поле 'messages' обязательно и должно быть списком."},  
 status=status.HTTP\_400\_BAD\_REQUEST  
 )  
  
 system\_message = get\_system\_message(request.user)  
 full\_messages = [system\_message] + messages  
  
 try:  
 answer = generate\_answer(full\_messages)  
 return Response({"answer": answer}, status=status.HTTP\_200\_OK)  
 except Exception as e:  
 return Response({"error": str(e)}, status=status.HTTP\_500\_INTERNAL\_SERVER\_ERROR)

\_\_init\_\_.py

\_\_init\_\_.py

asgi.py

import os  
  
from django.core.asgi import get\_asgi\_application  
  
os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'main.settings')  
  
application = get\_asgi\_application()

settings.py

import environ  
from datetime import timedelta  
from pathlib import Path  
  
from openai import OpenAI  
  
BASE\_DIR = Path(\_\_file\_\_).resolve().parent.parent  
ENV\_FILE = BASE\_DIR / '.env'  
  
env = environ.Env()  
environ.Env.read\_env(ENV\_FILE)  
  
  
SECRET\_KEY = env('SECRET\_KEY')  
DEBUG = env.bool('DEBUG', default=True)  
ALLOWED\_HOSTS = env.list('ALLOWED\_HOSTS', default=[])  
  
INSTALLED\_APPS = [  
 'django.contrib.admin',  
 'django.contrib.auth',  
 'django.contrib.contenttypes',  
 'django.contrib.sessions',  
 'django.contrib.messages',  
 'django.contrib.staticfiles',  
  
 'rest\_framework',  
 'rest\_framework\_simplejwt',  
 'corsheaders',  
 'drf\_yasg',  
  
 'users',  
 'coaches',  
 'progress',  
 'workouts',  
 'nutrition',  
 'blog'  
]  
  
REST\_FRAMEWORK = {  
 'DEFAULT\_AUTHENTICATION\_CLASSES': [  
 'rest\_framework\_simplejwt.authentication.JWTAuthentication',  
 ],  
 'DEFAULT\_PERMISSION\_CLASSES': [  
 'rest\_framework.permissions.IsAuthenticated',  
 ],  
 'DEFAULT\_FILTER\_BACKENDS': [  
 'django\_filters.rest\_framework.DjangoFilterBackend',  
 'rest\_framework.filters.OrderingFilter',  
 'rest\_framework.filters.SearchFilter',  
 ],  
 'DEFAULT\_PAGINATION\_CLASS': 'rest\_framework.pagination.LimitOffsetPagination',  
 'PAGE\_SIZE': 20,  
 'DEFAULT\_PARSER\_CLASSES': [  
 'rest\_framework.parsers.JSONParser',  
 'rest\_framework.parsers.MultiPartParser',  
 'rest\_framework.parsers.FormParser',  
 ],  
}  
  
SIMPLE\_JWT = {  
 'ACCESS\_TOKEN\_LIFETIME': timedelta(minutes=15),  
 'REFRESH\_TOKEN\_LIFETIME': timedelta(days=14),  
 'ROTATE\_REFRESH\_TOKENS': False,  
 'BLACKLIST\_AFTER\_ROTATION': True,  
}  
  
MIDDLEWARE = [  
 'django.middleware.security.SecurityMiddleware',  
 'django.contrib.sessions.middleware.SessionMiddleware',  
 'django.middleware.common.CommonMiddleware',  
 'django.middleware.csrf.CsrfViewMiddleware',  
 'django.contrib.auth.middleware.AuthenticationMiddleware',  
 'django.contrib.messages.middleware.MessageMiddleware',  
 'django.middleware.clickjacking.XFrameOptionsMiddleware',  
 'corsheaders.middleware.CorsMiddleware',  
]  
  
ROOT\_URLCONF = 'main.urls'  
  
TEMPLATES = [  
 {  
 'BACKEND': 'django.template.backends.django.DjangoTemplates',  
 'DIRS': [],  
 'APP\_DIRS': True,  
 'OPTIONS': {  
 'context\_processors': [  
 'django.template.context\_processors.debug',  
 'django.template.context\_processors.request',  
 'django.contrib.auth.context\_processors.auth',  
 'django.contrib.messages.context\_processors.messages',  
 ],  
 },  
 },  
]  
  
WSGI\_APPLICATION = 'main.wsgi.application'  
  
DATABASES = {  
 'default': {  
 'ENGINE': 'django.db.backends.postgresql',  
 'NAME': env('DB\_NAME'),  
 'USER': env('DB\_USER'),  
 'PASSWORD': env('DB\_PASSWORD'),  
 'HOST': env('DB\_HOST', default='localhost'),  
 'PORT': env('DB\_PORT', default='5432'),  
 }  
}  
  
AUTH\_PASSWORD\_VALIDATORS = [  
 {  
 'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',  
 },  
 {  
 'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',  
 },  
 {  
 'NAME': 'django.contrib.auth.password\_validation.CommonPasswordValidator',  
 },  
 {  
 'NAME': 'django.contrib.auth.password\_validation.NumericPasswordValidator',  
 },  
]  
  
LANGUAGE\_CODE = 'ru-ru'  
TIME\_ZONE = 'UTC'  
USE\_I18N = True  
USE\_TZ = True  
  
STATIC\_URL = 'static/'  
  
MEDIA\_URL = '/media/'  
MEDIA\_ROOT = BASE\_DIR / 'media'  
  
# Create media directories if they don't exist  
MEDIA\_DIRS = [  
 MEDIA\_ROOT / 'avatars',  
 MEDIA\_ROOT / 'progress\_photos',  
 MEDIA\_ROOT / 'blog\_images',  
]  
  
for directory in MEDIA\_DIRS:  
 directory.mkdir(parents=True, exist\_ok=True)  
  
CORS\_ALLOW\_ALL\_ORIGINS = True  
  
DEFAULT\_AUTO\_FIELD = 'django.db.models.BigAutoField'  
  
AUTH\_USER\_MODEL = 'users.CustomUser'  
  
OPENAI\_CLIENT = OpenAI(api\_key=env('OPENAI\_API\_KEY'))

urls.py

from django.conf import settings  
from django.conf.urls.static import static  
from django.contrib import admin  
from django.urls import path, include  
from drf\_yasg import openapi  
from drf\_yasg.views import get\_schema\_view  
from rest\_framework import permissions  
from rest\_framework\_simplejwt.views import TokenObtainPairView, TokenRefreshView  
  
schema\_view = get\_schema\_view(  
 openapi.Info(  
 title="NeuroFit API",  
 default\_version='v1',  
 ),  
 public=True,  
 permission\_classes=[permissions.AllowAny],  
)  
  
urlpatterns = [  
 path('admin/', admin.site.urls),  
 path('api/users/', include('users.urls')),  
 path('api/workouts/', include('workouts.urls')),  
 path('api/nutrition/', include('nutrition.urls')),  
 path('api/progress/', include('progress.urls')),  
 path('api/blog/', include('blog.urls')),  
 path('api/coaches/', include('coaches.urls')),  
 path('api/token/', TokenObtainPairView.as\_view(), name='token\_obtain\_pair'),  
 path('api/token/refresh/', TokenRefreshView.as\_view(), name='token\_refresh'),  
 path('swagger/', schema\_view.with\_ui('swagger', cache\_timeout=0), name='schema-swagger-ui'),  
] + static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

wsgi.py

import os  
  
from django.core.wsgi import get\_wsgi\_application  
  
os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'main.settings')  
  
application = get\_wsgi\_application()

\_\_init\_\_.py

admin.py

from django.contrib import admin  
  
from .models import Nutrition  
  
  
@admin.register(Nutrition)  
class NutritionAdmin(admin.ModelAdmin):  
 list\_display = ('user', 'date', 'calories')  
 list\_filter = ('date',)  
 search\_fields = ('user\_\_username',)  
 ordering = ('-date',)

apps.py

from django.apps import AppConfig  
  
  
class NutritionConfig(AppConfig):  
 default\_auto\_field = 'django.db.models.BigAutoField'  
 name = 'nutrition'

models.py

from django.db import models  
from django.utils.timezone import now  
from users.models import CustomUser  
  
  
class Nutrition(models.Model):  
 user = models.ForeignKey(CustomUser, on\_delete=models.CASCADE)  
 date = models.DateField(default=now)  
 meals = models.JSONField()  
 calories = models.IntegerField(null=True, blank=True)  
 created\_at = models.DateTimeField(auto\_now\_add=True)

serializers.py

from rest\_framework import serializers  
from .models import Nutrition  
  
  
class NutritionSerializer(serializers.ModelSerializer):  
 class Meta:  
 model = Nutrition  
 fields = ['id', 'date', 'meals', 'calories', 'created\_at']  
 read\_only\_fields = ['created\_at']

tests.py

from django.test import TestCase  
from django.urls import reverse  
from rest\_framework.test import APIClient  
from rest\_framework import status  
from users.models import CustomUser  
from .models import Nutrition  
from datetime import date, timedelta  
  
  
class NutritionTests(TestCase):  
 def setUp(self):  
 self.client = APIClient()  
 self.user = CustomUser.objects.create\_user(  
 username='testuser',  
 email='test@example.com',  
 password='testpass123'  
 )  
 self.client.force\_authenticate(user=self.user)  
   
 self.list\_url = reverse('nutrition-list')  
 self.generate\_url = reverse('nutrition-generate')  
 self.latest\_url = reverse('nutrition-latest')  
   
 # Создаем тестовые планы питания  
 self.plan1 = Nutrition.objects.create(  
 user=self.user,  
 meals={  
 'breakfast': {  
 'items': ['Овсянка', 'Банан'],  
 'calories': 300,  
 'proteins': 10,  
 'fats': 5,  
 'carbs': 50  
 }  
 },  
 date=date.today()  
 )  
 self.plan2 = Nutrition.objects.create(  
 user=self.user,  
 meals={  
 'lunch': {  
 'items': ['Куриная грудка', 'Рис'],  
 'calories': 500,  
 'proteins': 30,  
 'fats': 10,  
 'carbs': 60  
 }  
 },  
 date=date.today() - timedelta(days=1)  
 )  
  
 def test\_list\_nutrition(self):  
 response = self.client.get(self.list\_url)  
 self.assertEqual(response.status\_code, status.HTTP\_200\_OK)  
  
 def test\_filter\_nutrition(self):  
 # Фильтр по дате  
 response = self.client.get(f"{self.list\_url}?date={date.today()}")  
 self.assertEqual(response.status\_code, status.HTTP\_200\_OK)  
   
 # Фильтр по типу приема пищи  
 response = self.client.get(f"{self.list\_url}?meal\_type=breakfast")  
 self.assertEqual(response.status\_code, status.HTTP\_200\_OK)  
  
 def test\_get\_latest\_nutrition(self):  
 response = self.client.get(self.latest\_url)  
 self.assertEqual(response.status\_code, status.HTTP\_200\_OK)  
 self.assertEqual(response.data['id'], self.plan1.id)  
 self.assertEqual(response.data['date'].strftime('%Y-%m-%d'), date.today().strftime('%Y-%m-%d'))  
  
 def test\_generate\_nutrition(self):  
 response = self.client.post(self.generate\_url)  
 self.assertEqual(response.status\_code, status.HTTP\_201\_CREATED)  
 self.assertIn('id', response.data)  
 self.assertIn('date', response.data)  
 self.assertIn('meals', response.data)  
 self.assertIn('breakfast', response.data['meals'])  
 self.assertIn('lunch', response.data['meals'])  
 self.assertIn('dinner', response.data['meals'])  
  
 def test\_meal\_structure(self):  
 response = self.client.post(self.generate\_url)  
 for meal\_type in ['breakfast', 'lunch', 'dinner']:  
 meal = response.data['meals'][meal\_type]  
 self.assertIn('items', meal)  
 self.assertIn('calories', meal)  
 self.assertIn('proteins', meal)  
 self.assertIn('fats', meal)  
 self.assertIn('carbs', meal)  
 self.assertIsInstance(meal['items'], list)

urls.py

from django.urls import path, include  
from rest\_framework.routers import DefaultRouter  
  
from .views import NutritionViewSet  
  
router = DefaultRouter()  
router.register('', NutritionViewSet, basename='nutrition')  
  
urlpatterns = [  
 path('', include(router.urls)),  
]

views.py

import json  
  
from coaches.services import generate\_answer  
from coaches.utils import get\_system\_message, nutrition\_instruction  
from django\_filters import rest\_framework as filters  
from rest\_framework import viewsets, status  
from rest\_framework.decorators import action  
from rest\_framework.permissions import IsAuthenticated  
from rest\_framework.response import Response  
  
from .models import Nutrition  
from .serializers import NutritionSerializer  
  
  
class NutritionFilter(filters.FilterSet):  
 date = filters.DateFilter(field\_name='date')  
  
 class Meta:  
 model = Nutrition  
 fields = ['date']  
  
  
class NutritionViewSet(viewsets.ModelViewSet):  
 serializer\_class = NutritionSerializer  
 permission\_classes = [IsAuthenticated]  
 filterset\_class = NutritionFilter  
 ordering\_fields = ['date', 'created\_at']  
 ordering = ['-date']  
  
 def get\_queryset(self):  
 return Nutrition.objects.filter(user=self.request.user)  
  
 def perform\_create(self, serializer):  
 serializer.save(user=self.request.user)  
  
 @action(detail=False, methods=['get'])  
 def latest(self, request):  
 nutrition = self.get\_queryset().first()  
 if nutrition:  
 return Response({  
 "id": nutrition.id,  
 "meals": nutrition.meals,  
 "calories": nutrition.calories,  
 "date": nutrition.date  
 })  
 return Response({"message": "Нет сохранённых планов питания."}, status=404)  
  
 @action(detail=False, methods=['post'])  
 def generate(self, request):  
 system\_msg = get\_system\_message(request.user)  
 messages = [system\_msg, nutrition\_instruction]  
  
 try:  
 result = generate\_answer(messages=messages, temperature=0.1)  
 data = json.loads(result)  
 nutrition = Nutrition.objects.create(  
 user=request.user,  
 meals=data.get("meals"),  
 calories=data.get("calories")  
 )  
 return Response({  
 "id": nutrition.id,  
 "date": nutrition.date,  
 "meals": nutrition.meals,  
 "calories": nutrition.calories  
 }, status=status.HTTP\_201\_CREATED)  
 except Exception as e:  
 return Response({"error": str(e)}, status=status.HTTP\_500\_INTERNAL\_SERVER\_ERROR)

\_\_init\_\_.py

admin.py

from django.contrib import admin  
  
from .models import ProgressChart  
  
  
@admin.register(ProgressChart)  
class ProgressChartAdmin(admin.ModelAdmin):  
 list\_display = ('user', 'date', 'weight', 'height')  
 list\_filter = ('date',)  
 search\_fields = ('user\_\_username', 'notes')  
 ordering = ('-date',)

apps.py

from django.apps import AppConfig  
  
  
class ProgressConfig(AppConfig):  
 default\_auto\_field = 'django.db.models.BigAutoField'  
 name = 'progress'

models.py

from django.db import models  
from django.utils.timezone import now  
from users.models import CustomUser  
  
  
class ProgressChart(models.Model):  
 user = models.ForeignKey(CustomUser, on\_delete=models.CASCADE)  
 date = models.DateField(default=now)  
 weight = models.DecimalField(max\_digits=5, decimal\_places=2, null=True, blank=True)  
 height = models.PositiveIntegerField(null=True, blank=True)  
 photo = models.ImageField(upload\_to='progress\_photos/', null=True, blank=True)  
 notes = models.TextField(blank=True)

serializers.py

from rest\_framework import serializers  
  
from .models import ProgressChart  
  
  
class ProgressChartSerializer(serializers.ModelSerializer):  
 class Meta:  
 model = ProgressChart  
 fields = ['id', 'user', 'date', 'weight', 'height', 'photo', 'notes']

tests.py

from django.test import TestCase  
  
# Create your tests here.

urls.py

from django.urls import path, include  
from rest\_framework.routers import DefaultRouter  
from .views import ProgressChartViewSet  
  
router = DefaultRouter()  
router.register('', ProgressChartViewSet, basename='progress-chart')  
  
urlpatterns = [  
 path('', include(router.urls)),  
]

views.py

from django\_filters import rest\_framework as filters  
from rest\_framework import viewsets  
from rest\_framework.decorators import action  
from rest\_framework.permissions import IsAuthenticated  
from rest\_framework.response import Response  
  
from .models import ProgressChart  
from .serializers import ProgressChartSerializer  
  
  
class ProgressChartFilter(filters.FilterSet):  
 date = filters.DateFilter(field\_name='date')  
  
 class Meta:  
 model = ProgressChart  
 fields = ['date']  
  
  
class ProgressChartViewSet(viewsets.ModelViewSet):  
 serializer\_class = ProgressChartSerializer  
 permission\_classes = [IsAuthenticated]  
 filterset\_class = ProgressChartFilter  
 ordering\_fields = ['date']  
 ordering = ['-date']  
  
 def get\_queryset(self):  
 return ProgressChart.objects.filter(user=self.request.user).select\_related('user')  
  
 def perform\_create(self, serializer):  
 serializer.save(user=self.request.user)  
  
 @action(detail=False, methods=['get'])  
 def latest(self, request):  
 progress = self.get\_queryset().first()  
 if progress:  
 return Response({  
 "weight": progress.weight,  
 "height": progress.height,  
 "date": progress.date,  
 "photo": progress.photo.url if progress.photo else None,  
 "notes": progress.notes  
 })  
 return Response({"message": "Нет сохранённых данных о прогрессе."}, status=404)  
  
 @action(detail=False, methods=['get'])  
 def history(self, request):  
 queryset = self.get\_queryset().order\_by('date')  
 data = {}  
  
 for progress in queryset:  
 date\_str = progress.date.strftime('%Y-%m-%d')  
 if date\_str not in data:  
 data[date\_str] = []  
  
 data[date\_str].append({  
 "id": progress.id,  
 "weight": progress.weight,  
 "height": progress.height,  
 "photo": progress.photo.url if progress.photo else None,  
 "notes": progress.notes  
 })  
  
 return Response(data)

\_\_init\_\_.py

admin.py

from django.contrib import admin  
from django.contrib.auth.admin import UserAdmin  
from .models import CustomUser  
  
  
@admin.register(CustomUser)  
class CustomUserAdmin(UserAdmin):  
 list\_display = ('username', 'email', 'is\_staff', 'is\_active', 'created\_at')  
 list\_filter = ('is\_staff', 'is\_active', 'gender', 'goal')  
 search\_fields = ('username', 'email')  
 ordering = ('-created\_at',)  
  
 fieldsets = (  
 (None, {  
 'fields': ('username', 'email', 'password'),  
 }),  
 ('Personal info', {  
 'fields': (  
 'date\_of\_birth',  
 'gender',  
 'height',  
 'weight',  
 'goal',  
 'has\_equipment',  
 'avatar',  
 ),  
 }),  
 ('Permissions', {  
 'fields': (  
 'is\_active',  
 'is\_staff',  
 'is\_superuser',  
 'groups',  
 'user\_permissions',  
 ),  
 }),  
 )  
  
 add\_fieldsets = (  
 (None, {  
 'classes': ('wide',),  
 'fields': ('username', 'email', 'password1', 'password2'),  
 }),  
 )

apps.py

from django.apps import AppConfig  
  
  
class UsersConfig(AppConfig):  
 default\_auto\_field = 'django.db.models.BigAutoField'  
 name = 'users'

models.py

from django.contrib.auth.models import AbstractBaseUser, BaseUserManager, PermissionsMixin  
from django.db import models  
from django.utils.timezone import now  
  
  
class CustomUserManager(BaseUserManager):  
 def create\_user(self, username, email, password=None, \*\*extra\_fields):  
 if not email:  
 raise ValueError("Email is required")  
 email = self.normalize\_email(email)  
 user = self.model(username=username, email=email, \*\*extra\_fields)  
 user.set\_password(password)  
 user.save(using=self.\_db)  
 return user  
  
 def create\_superuser(self, username, email, password=None, \*\*extra\_fields):  
 extra\_fields.setdefault('is\_staff', True)  
 extra\_fields.setdefault('is\_superuser', True)  
 return self.create\_user(username, email, password, \*\*extra\_fields)  
  
  
class CustomUser(AbstractBaseUser, PermissionsMixin):  
 class FitnessGoal(models.TextChoices):  
 WEIGHT\_LOSS = 'weight\_loss', 'Похудение'  
 MUSCLE\_GAIN = 'muscle\_gain', 'Набор массы'  
 ENDURANCE = 'endurance', 'Выносливость'  
 GENERAL = 'general\_fitness', 'Общая физическая форма'  
  
 username = models.CharField(max\_length=50, unique=True)  
 email = models.EmailField(unique=True)  
 date\_of\_birth = models.DateField(null=True, blank=True)  
 gender = models.CharField(max\_length=10, null=True, blank=True)  
 height = models.PositiveIntegerField(null=True, blank=True)  
 weight = models.DecimalField(max\_digits=5, decimal\_places=2, null=True, blank=True)  
 goal = models.CharField(max\_length=20, choices=FitnessGoal.choices, null=True, blank=True)  
 has\_equipment = models.BooleanField(default=True)  
  
 avatar = models.ImageField(upload\_to='avatars/', null=True, blank=True)  
  
 is\_active = models.BooleanField(default=True)  
 is\_staff = models.BooleanField(default=False)  
  
 created\_at = models.DateTimeField(default=now)  
 updated\_at = models.DateTimeField(auto\_now=True)  
  
 USERNAME\_FIELD = 'username'  
 REQUIRED\_FIELDS = ['email']  
  
 objects = CustomUserManager()  
  
 def \_\_str\_\_(self):  
 return self.username

serializers.py

from django.contrib.auth import get\_user\_model  
from django.contrib.auth.password\_validation import validate\_password  
from rest\_framework import serializers  
  
User = get\_user\_model()  
  
  
class RegisterSerializer(serializers.ModelSerializer):  
 password = serializers.CharField(write\_only=True, required=True, validators=[validate\_password])  
 password2 = serializers.CharField(write\_only=True, required=True)  
  
 class Meta:  
 model = User  
 fields = (  
 'username', 'email', 'password', 'password2',  
 'date\_of\_birth', 'gender', 'height', 'weight',  
 'goal', 'has\_equipment'  
 )  
  
 def validate(self, attrs):  
 if attrs.get('password') != attrs.get('password2'):  
 raise serializers.ValidationError({"password": "Пароли не совпадают"})  
 return attrs  
  
 def create(self, validated\_data):  
 password = validated\_data.pop('password')  
 validated\_data.pop('password2', None)  
 user = User.objects.create(\*\*validated\_data)  
 user.set\_password(password)  
 user.save()  
 return user  
  
  
class UserSerializer(serializers.ModelSerializer):  
 class Meta:  
 model = User  
 fields = (  
 'id', 'username', 'email',  
 'date\_of\_birth', 'gender', 'height', 'weight',  
 'goal', 'has\_equipment', 'avatar')  
 read\_only\_fields = ('id', 'email')

tests.py

from django.test import TestCase  
from django.urls import reverse  
from rest\_framework.test import APIClient  
from rest\_framework import status  
from .models import CustomUser  
  
class UserTests(TestCase):  
 def setUp(self):  
 self.client = APIClient()  
 self.register\_url = reverse('register')  
 self.login\_url = reverse('token\_obtain\_pair')  
 self.profile\_url = reverse('profile')  
   
 self.user\_data = {  
 'username': 'testuser',  
 'email': 'test@example.com',  
 'password': 'testpass123',  
 'password2': 'testpass123',  
 }  
   
 self.profile\_data = {  
 'height': 180,  
 'weight': 80.00,  
 'date\_of\_birth': '1990-01-01',  
 'gender': 'M',  
 'goal': 'weight\_loss',  
 'has\_equipment': True  
 }  
  
 def test\_register\_user(self):  
 response = self.client.post(self.register\_url, self.user\_data)  
 self.assertEqual(response.status\_code, status.HTTP\_201\_CREATED)  
 self.assertEqual(CustomUser.objects.count(), 1)  
 self.assertEqual(CustomUser.objects.get().username, 'testuser')  
  
 def test\_login\_user(self):  
 # Создаем пользователя  
 CustomUser.objects.create\_user(  
 username='testuser',  
 email='test@example.com',  
 password='testpass123'  
 )  
   
 # Пробуем залогиниться  
 response = self.client.post(self.login\_url, {  
 'username': 'testuser',  
 'password': 'testpass123'  
 })  
 self.assertEqual(response.status\_code, status.HTTP\_200\_OK)  
 self.assertIn('access', response.data)  
 self.assertIn('refresh', response.data)  
  
 def test\_update\_profile(self):  
 # Создаем и логиним пользователя  
 user = CustomUser.objects.create\_user(  
 username='testuser',  
 email='test@example.com',  
 password='testpass123'  
 )  
 self.client.force\_authenticate(user=user)  
   
 # Обновляем профиль  
 response = self.client.patch(self.profile\_url, self.profile\_data)  
 self.assertEqual(response.status\_code, status.HTTP\_200\_OK)  
 self.assertEqual(response.data['height'], 180)  
 self.assertEqual(float(response.data['weight']), 80.00)

urls.py

from django.urls import path  
  
from .views import RegisterView, ProfileView  
  
urlpatterns = [  
 path('register/', RegisterView.as\_view(), name='register'),  
 path('profile/', ProfileView.as\_view(), name='profile'),  
]

views.py

from rest\_framework import generics  
from rest\_framework.permissions import AllowAny  
from rest\_framework.permissions import IsAuthenticated  
  
from .serializers import RegisterSerializer, UserSerializer  
  
  
class RegisterView(generics.CreateAPIView):  
 serializer\_class = RegisterSerializer  
 permission\_classes = [AllowAny]  
  
  
class ProfileView(generics.RetrieveUpdateAPIView):  
 serializer\_class = UserSerializer  
 permission\_classes = [IsAuthenticated]  
  
 def get\_object(self):  
 return self.request.user

\_\_init\_\_.py

\_\_init\_\_.py

admin.py

from django.contrib import admin  
  
from .models import Workout  
  
  
@admin.register(Workout)  
class WorkoutAdmin(admin.ModelAdmin):  
 list\_display = ('user', 'date', 'completed')  
 list\_filter = ('date', 'completed')  
 search\_fields = ('user\_\_username',)  
 ordering = ('-date',)

apps.py

from django.apps import AppConfig  
  
  
class WorkoutsConfig(AppConfig):  
 default\_auto\_field = 'django.db.models.BigAutoField'  
 name = 'workouts'

models.py

from django.db import models  
from django.utils.timezone import now  
from users.models import CustomUser  
  
  
class Workout(models.Model):  
 user = models.ForeignKey(CustomUser, on\_delete=models.CASCADE)  
 date = models.DateField(default=now)  
 plan = models.JSONField()  
 completed = models.BooleanField(default=False)  
 created\_at = models.DateTimeField(auto\_now\_add=True)

serializers.py

from rest\_framework import serializers  
from .models import Workout  
  
  
class WorkoutSerializer(serializers.ModelSerializer):  
 class Meta:  
 model = Workout  
 fields = ['id', 'date', 'plan', 'completed', 'created\_at']  
 read\_only\_fields = ['user', 'created\_at']

tests.py

from django.test import TestCase  
from django.urls import reverse  
from rest\_framework.test import APIClient  
from rest\_framework import status  
from users.models import CustomUser  
from .models import Workout  
from datetime import date, timedelta  
  
class WorkoutTests(TestCase):  
 def setUp(self):  
 self.client = APIClient()  
 self.user = CustomUser.objects.create\_user(  
 username='testuser',  
 email='test@example.com',  
 password='testpass123'  
 )  
 self.client.force\_authenticate(user=self.user)  
   
 self.list\_url = reverse('workouts-list')  
 self.generate\_url = reverse('workouts-generate')  
 self.latest\_url = reverse('workouts-latest')  
  
 # Создаем тестовые тренировки  
 self.workout1 = Workout.objects.create(  
 user=self.user,  
 plan=[{"exercise": "Push-ups", "sets": 3, "reps": 10}],  
 date=date.today()  
 )  
 self.workout2 = Workout.objects.create(  
 user=self.user,  
 plan=[{"exercise": "Squats", "sets": 3, "reps": 15}],  
 date=date.today() - timedelta(days=1),  
 completed=True  
 )  
  
 def test\_list\_workouts(self):  
 response = self.client.get(self.list\_url)  
 self.assertEqual(response.status\_code, status.HTTP\_200\_OK)  
  
 def test\_filter\_workouts(self):  
 # Фильтр по дате  
 response = self.client.get(f"{self.list\_url}?date={date.today()}")  
 self.assertEqual(response.status\_code, status.HTTP\_200\_OK)  
  
 # Фильтр по статусу  
 response = self.client.get(f"{self.list\_url}?completed=true")  
 self.assertEqual(response.status\_code, status.HTTP\_200\_OK)  
  
 def test\_get\_latest\_workout(self):  
 response = self.client.get(self.latest\_url)  
 self.assertEqual(response.status\_code, status.HTTP\_200\_OK)  
 self.assertEqual(response.data['id'], self.workout1.id)  
 self.assertEqual(response.data['date'].strftime('%Y-%m-%d'), date.today().strftime('%Y-%m-%d'))  
  
 def test\_generate\_workout(self):  
 response = self.client.post(self.generate\_url)  
 self.assertEqual(response.status\_code, status.HTTP\_201\_CREATED)  
 self.assertIn('id', response.data)  
 self.assertIn('date', response.data)  
 self.assertIn('plan', response.data)  
 self.assertIn('completed', response.data)  
 self.assertFalse(response.data['completed'])  
  
 def test\_complete\_workout(self):  
 complete\_url = reverse('workouts-complete', args=[self.workout1.id])  
 response = self.client.post(complete\_url)  
 self.assertEqual(response.status\_code, status.HTTP\_200\_OK)  
 self.assertEqual(response.data['status'], 'workout completed')  
   
 # Проверяем что тренировка отмечена как выполненная  
 workout = Workout.objects.get(id=self.workout1.id)  
 self.assertTrue(workout.completed)

urls.py

from django.urls import path, include  
from rest\_framework.routers import DefaultRouter  
from .views import WorkoutViewSet  
  
router = DefaultRouter()  
router.register('', WorkoutViewSet, basename='workouts')  
  
urlpatterns = [  
 path('', include(router.urls)),  
]

views.py

import json  
  
from coaches.services import generate\_answer  
from coaches.utils import get\_system\_message, workout\_instruction  
from django\_filters import rest\_framework as filters  
from rest\_framework import viewsets, status  
from rest\_framework.decorators import action  
from rest\_framework.permissions import IsAuthenticated  
from rest\_framework.response import Response  
  
from .models import Workout  
from .serializers import WorkoutSerializer  
  
  
class WorkoutFilter(filters.FilterSet):  
 date = filters.DateFilter(field\_name='date')  
 completed = filters.BooleanFilter(field\_name='completed')  
  
 class Meta:  
 model = Workout  
 fields = ['date', 'completed']  
  
  
class WorkoutViewSet(viewsets.ModelViewSet):  
 serializer\_class = WorkoutSerializer  
 permission\_classes = [IsAuthenticated]  
 filterset\_class = WorkoutFilter  
 ordering\_fields = ['date', 'created\_at']  
 ordering = ['-date']  
  
 def get\_queryset(self):  
 return Workout.objects.filter(user=self.request.user).select\_related('user')  
  
 @action(detail=False, methods=['get'])  
 def latest(self, request):  
 workout = self.get\_queryset().first()  
 if workout:  
 return Response({  
 "id": workout.id,  
 "date": workout.date,  
 "plan": workout.plan,  
 "completed": workout.completed  
 })  
 return Response({"message": "Нет сохранённых тренировок."}, status=404)  
  
 @action(detail=False, methods=['post'])  
 def generate(self, request):  
 system\_msg = get\_system\_message(request.user)  
 messages = [system\_msg, workout\_instruction]  
  
 try:  
 result = generate\_answer(messages=messages, temperature=0.1)  
 data = json.loads(result)  
 workout = Workout.objects.create(user=request.user, plan=data)  
 return Response({  
 "id": workout.id,  
 "date": workout.date,  
 "plan": workout.plan,  
 "completed": workout.completed  
 }, status=status.HTTP\_201\_CREATED)  
 except Exception as e:  
 return Response({"error": str(e)}, status=status.HTTP\_500\_INTERNAL\_SERVER\_ERROR)  
  
 @action(detail=True, methods=['post'])  
 def complete(self, request, pk=None):  
 workout = self.get\_object()  
 workout.completed = True  
 workout.save()  
 return Response({'status': 'workout completed'}, status=status.HTTP\_200\_OK)

\_\_init\_\_.py

App.tsx

import React from 'react';  
import { Toaster } from 'react-hot-toast';  
import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';  
import { AuthProvider } from './contexts/AuthContext';  
import Navbar from './components/Navbar';  
import Home from './pages/Home';  
import Login from './pages/Login';  
import Register from './pages/Register';  
import Profile from './pages/Profile';  
import Coach from "./pages/Coach.tsx";  
import Workouts from "./pages/Workouts";  
import Nutrition from "./pages/Nutrition";  
import Progress from "./pages/Progress.tsx";  
import Blog from "./pages/Blog.tsx"  
  
function App() {  
 return (  
 <AuthProvider>  
 <Router>  
 <div className="min-h-screen bg-gray-50">  
 <Navbar/>  
 <Routes>  
 <Route path="/" element={<Home/>}/>  
 <Route path="/login" element={<Login/>}/>  
 <Route path="/register" element={<Register/>}/>  
 <Route path="/profile" element={<Profile/>}/>  
 <Route path="/coach" element={<Coach/>}/>  
 <Route path="/workouts" element={<Workouts/>}/>  
 <Route path="/nutrition" element={<Nutrition/>}/>  
 <Route path="/progress" element={<Progress/>}/>  
 <Route path="/blog" element={<Blog/>}/>  
 </Routes>  
 <Toaster position="top-right"/>  
 </div>  
 </Router>  
 </AuthProvider>  
 );  
}  
  
export default App;

index.css

@tailwind base;  
@tailwind components;  
@tailwind utilities;

main.tsx

import { StrictMode } from 'react'  
import { createRoot } from 'react-dom/client'  
import App from './App'  
import './index.css'  
  
createRoot(document.getElementById('root')!).render(  
 <StrictMode>  
 <App />  
 </StrictMode>,  
)

vite-env.d.ts

/// <reference types="vite/client" />

Navbar.tsx

import React from 'react';  
import { Link, useNavigate } from 'react-router-dom';  
import { useAuth } from '../contexts/AuthContext';  
import { Bot, User, LogOut } from 'lucide-react';  
  
const Navbar = () => {  
 const { isAuthenticated, logout } = useAuth();  
 const navigate = useNavigate();  
  
 const handleLogout = () => {  
 logout();  
 navigate('/login');  
 };  
  
 return (  
 <nav className="bg-gray-800 text-gray-100 shadow-lg border-b border-gray-700">  
 <div className="max-w-7xl mx-auto px-4">  
 <div className="flex justify-between h-16 items-center">  
 <Link to="/" className="flex items-center space-x-2">  
 <Bot className="h-8 w-8 text-yellow-500"/>  
 <span className="font-bold text-xl text-white">NeuroFit</span>  
 </Link>  
  
 <div className="flex items-center space-x-4">  
 {isAuthenticated ? (  
 <>  
 <Link  
 to="/coach"  
 className="flex items-center space-x-1 hover:text-yellow-500 transition"  
 >  
 <span>Тренер</span>  
 </Link>  
 <Link  
 to="/nutrition"  
 className="flex items-center space-x-1 hover:text-yellow-500 transition"  
 >  
 <span>Питание</span>  
 </Link>  
 <Link  
 to="/workouts"  
 className="flex items-center space-x-1 hover:text-yellow-500 transition"  
 >  
 <span>Тренировки</span>  
 </Link>  
 <Link  
 to="/progress"  
 className="flex items-center space-x-1 hover:text-yellow-500 transition"  
 >  
 <span>Прогресс</span>  
 </Link>  
 <Link  
 to="/profile"  
 className="flex items-center space-x-1 hover:text-yellow-500 transition"  
 >  
 <User className="h-5 w-5"/>  
 <span>Профиль</span>  
 </Link>  
 <button  
 onClick={handleLogout}  
 className="flex items-center space-x-1 hover:text-yellow-500 transition"  
 >  
 <LogOut className="h-5 w-5"/>  
 <span>Выйти</span>  
 </button>  
 </>  
 ) : (  
 <>  
 <Link  
 to="/login"  
 className="hover:text-yellow-500 transition"  
 >  
 Вход  
 </Link>  
 <Link  
 to="/register"  
 className="bg-yellow-500 text-gray-900 px-4 py-2 rounded-md hover:bg-yellow-400 transition"  
 >  
 Регистрация  
 </Link>  
 </>  
 )}  
 </div>  
  
 </div>  
 </div>  
 </nav>  
 );  
};  
  
export default Navbar;

AuthContext.tsx

import React, {createContext, useContext, useEffect, useState} from 'react';  
import { apiService } from '../services/api';  
import { UserType } from "../types/user";  
import { AuthContextType } from "../types/auth";  
  
const AuthContext = createContext<AuthContextType | null>(null);  
  
export const AuthProvider: React.FC<{ children: React.ReactNode }> = ({ children }) => {  
 const [user, setUser] = useState<UserType | null>(null);  
 const [isAuthenticated, setIsAuthenticated] = useState(false);  
  
 useEffect(() => {  
 checkAuth();  
 }, []);  
  
 const checkAuth = async () => {  
 const token = localStorage.getItem('accessToken');  
 if (token) {  
 try {  
 const response = await apiService.getProfile();  
 setUser(response.data);  
 setIsAuthenticated(true);  
 } catch (error) {  
 localStorage.removeItem('accessToken');  
 localStorage.removeItem('refreshToken');  
 }  
 }  
 };  
  
 const updateAuth = async (username: string, password: string) => {  
 try {  
 const response = await apiService.getToken({ username, password });  
 const { access, refresh } = response.data;  
 localStorage.setItem('accessToken', access);  
 localStorage.setItem('refreshToken', refresh);  
  
 const userResponse = await apiService.getProfile();  
 setUser(userResponse.data);  
 setIsAuthenticated(true);  
 } catch (error) {  
 console.error('Ошибка при получении данных пользователя:', error);  
 throw error;  
 }  
 };  
  
 const login = async (username: string, password: string) => {  
 try {  
 await updateAuth(username, password);  
 } catch (error) {  
 console.error('Ошибка при входе:', error);  
 throw error;  
 }  
 };  
  
 const register = async (username: string, email: string, password: string) => {  
 try {  
 await apiService.register({ username, email, password });  
 await updateAuth(username, password);  
 } catch (error) {  
 console.error('Ошибка при регистрации:', error);  
 throw error;  
 }  
 };  
  
 const logout = () => {  
 localStorage.removeItem('accessToken');  
 localStorage.removeItem('refreshToken');  
 setUser(null);  
 setIsAuthenticated(false);  
 };  
  
 return (  
 <AuthContext.Provider value={{ user, setUser, isAuthenticated, login, register, logout }}>  
 {children}  
 </AuthContext.Provider>  
 );  
};  
  
export const useAuth = () => {  
 const context = useContext(AuthContext);  
 if (!context) {  
 throw new Error('useAuth должен быть использован внутри AuthProvider');  
 }  
 return context;  
};

Blog.tsx

import React, { useState, useEffect } from 'react';  
import { apiService } from '../services/api';  
import { Post, Comment } from '../types';  
import { Heart, MessageCircle, Trash2, Edit2 } from 'lucide-react';  
  
const Blog: React.FC = () => {  
 const [posts, setPosts] = useState<Post[]>([]);  
 const [loading, setLoading] = useState(true);  
 const [selectedPost, setSelectedPost] = useState<Post | null>(null);  
 const [comments, setComments] = useState<Comment[]>([]);  
 const [newComment, setNewComment] = useState('');  
  
 useEffect(() => {  
 fetchPosts();  
 }, []);  
  
 const fetchPosts = async () => {  
 try {  
 const response = await apiService.getPosts();  
 setPosts(response.data);  
 } catch (error) {  
 console.error('Ошибка при загрузке постов:', error);  
 } finally {  
 setLoading(false);  
 }  
 };  
  
 const handleLike = async (id: number) => {  
 try {  
 await apiService.likePost(id);  
 setPosts(posts.map(post =>   
 post.id === id   
 ? { ...post, likes\_count: post.likes\_count + 1 }  
 : post  
 ));  
 } catch (error) {  
 console.error('Ошибка при лайке поста:', error);  
 }  
 };  
  
 const handleDelete = async (id: number) => {  
 try {  
 await apiService.deletePost(id);  
 setPosts(posts.filter(p => p.id !== id));  
 } catch (error) {  
 console.error('Ошибка при удалении поста:', error);  
 }  
 };  
  
 const handlePostClick = async (post: Post) => {  
 setSelectedPost(post);  
 try {  
 const response = await apiService.getComments(post.id);  
 setComments(response.data);  
 } catch (error) {  
 console.error('Ошибка при загрузке комментариев:', error);  
 }  
 };  
  
 const handleAddComment = async (e: React.FormEvent) => {  
 e.preventDefault();  
 if (!selectedPost || !newComment.trim()) return;  
  
 try {  
 const response = await apiService.addComment(selectedPost.id, { content: newComment });  
 setComments([...comments, response.data]);  
 setNewComment('');  
 } catch (error) {  
 console.error('Ошибка при добавлении комментария:', error);  
 }  
 };  
  
 if (loading) {  
 return (  
 <div className="min-h-screen bg-gray-900 flex items-center justify-center">  
 <div className="text-yellow-500 text-xl">Загрузка...</div>  
 </div>  
 );  
 }  
  
 return (  
 <div className="min-h-screen bg-gray-900 py-12 px-4">  
 <div className="max-w-4xl mx-auto">  
 <h1 className="text-3xl font-bold text-gray-100 mb-8">Блог</h1>  
  
 <div className="grid md:grid-cols-2 gap-6">  
 <div className="space-y-4">  
 {posts.map((post) => (  
 <div  
 key={post.id}  
 className="bg-gray-800 rounded-lg p-6 shadow-lg border border-gray-700 cursor-pointer hover:border-yellow-500 transition"  
 onClick={() => handlePostClick(post)}  
 >  
 <h2 className="text-xl font-semibold text-gray-100 mb-2">  
 {post.title}  
 </h2>  
 <p className="text-gray-300 mb-4 line-clamp-2">{post.content}</p>  
 <div className="flex justify-between items-center">  
 <div className="flex items-center gap-4">  
 <button  
 onClick={(e) => {  
 e.stopPropagation();  
 handleLike(post.id);  
 }}  
 className="flex items-center gap-1 text-gray-400 hover:text-red-500 transition"  
 >  
 <Heart className="w-5 h-5" />  
 <span>{post.likes\_count}</span>  
 </button>  
 <div className="flex items-center gap-1 text-gray-400">  
 <MessageCircle className="w-5 h-5" />  
 <span>{post.comments\_count}</span>  
 </div>  
 </div>  
 <button  
 onClick={(e) => {  
 e.stopPropagation();  
 handleDelete(post.id);  
 }}  
 className="text-gray-400 hover:text-red-500 transition"  
 >  
 <Trash2 className="w-5 h-5" />  
 </button>  
 </div>  
 </div>  
 ))}  
 </div>  
  
 {selectedPost && (  
 <div className="bg-gray-800 rounded-lg p-6 shadow-lg border border-gray-700">  
 <h2 className="text-2xl font-semibold text-gray-100 mb-4">  
 {selectedPost.title}  
 </h2>  
 <p className="text-gray-300 mb-6">{selectedPost.content}</p>  
  
 <div className="space-y-4">  
 <h3 className="text-xl font-semibold text-gray-100">Комментарии</h3>  
 <form onSubmit={handleAddComment} className="mb-4">  
 <input  
 type="text"  
 value={newComment}  
 onChange={(e) => setNewComment(e.target.value)}  
 placeholder="Добавить комментарий..."  
 className="w-full p-2 bg-gray-700 text-white rounded-md border border-gray-600 focus:outline-none focus:ring-2 focus:ring-yellow-500"  
 />  
 </form>  
  
 <div className="space-y-3">  
 {comments.map((comment) => (  
 <div  
 key={comment.id}  
 className="bg-gray-700 rounded-md p-3"  
 >  
 <div className="text-sm text-gray-400 mb-1">  
 {comment.author.username}  
 </div>  
 <div className="text-gray-100">{comment.content}</div>  
 </div>  
 ))}  
 </div>  
 </div>  
 </div>  
 )}  
 </div>  
 </div>  
 </div>  
 );  
};  
  
export default Blog;

Coach.tsx

import React, { useState } from 'react';  
import { apiService } from '../services/api';  
import { CoachGenerateType } from "../types/coach.ts";  
  
const Coach: React.FC = () => {  
 const [message, setMessage] = useState<string>('');  
 const [messages, setMessages] = useState<Array<{ role: string; content: string }>>([  
 {role: "system", content: "Ты — профессиональный фитнес-тренер."}  
 ]);  
  
 const handleSubmit = async (e: React.FormEvent) => {  
 e.preventDefault();  
  
 if (message.trim()) {  
 const newUserMessage = {role: "user", content: message};  
 const updatedMessages = [...messages, newUserMessage];  
  
 setMessages(updatedMessages);  
 setMessage('');  
 try {  
 const response = await apiService.getGenerate({messages: updatedMessages} as CoachGenerateType);  
 console.log(response);  
  
 const aiMessage = {role: "assistant", content: response.data.answer};  
 setMessages([...updatedMessages, aiMessage]);  
 } catch (error) {  
 console.error('Ошибка при общении с ИИ:', error);  
 }  
 }  
 };  
  
 return (  
 <div className="min-h-screen bg-gray-900 py-12 px-4">  
 <div className="max-w-3xl mx-auto bg-gray-800 rounded-lg shadow-lg p-8">  
 <h1 className="text-3xl font-bold text-gray-100 mb-6">Ваш персональный тренер</h1>  
 <div className="space-y-4 mb-6 h-96 overflow-y-auto p-4 bg-gray-700 rounded-lg">  
 {messages  
 .filter(msg => msg.role !== "system")  
 .map((msg, index) => (  
 <div key={index} className="text-gray-100">  
 <strong>{msg.role === "user" ? "Вы" : "Тренер"}:</strong> {msg.content}  
 </div>  
 ))}  
 </div>  
 <form onSubmit={handleSubmit} className="flex items-center space-x-2">  
 <input  
 type="text"  
 value={message}  
 onChange={(e) => setMessage(e.target.value)}  
 placeholder="Напишите свой вопрос..."  
 className="flex-1 p-2 bg-gray-700 text-white rounded-md border border-gray-600 focus:outline-none focus:ring-2 focus:ring-yellow-500"  
 />  
 <button  
 type="submit"  
 className="bg-yellow-500 text-gray-900 px-4 py-2 rounded-md hover:bg-yellow-400 transition"  
 >  
 Отправить  
 </button>  
 </form>  
 </div>  
 </div>  
 );  
};  
  
export default Coach;

Home.tsx

import React from 'react';  
import { Bot, Dumbbell, Sparkles } from 'lucide-react';  
  
const Home = () => {  
 return (  
 <div className="min-h-screen bg-gradient-to-b from-gray-800 via-gray-900 to-gray-900">  
 <div className="max-w-7xl mx-auto px-4 py-16">  
 <div className="text-center mb-16">  
 <div className="flex justify-center items-center mb-6">  
 <Bot className="h-16 w-16 text-yellow-500 animate-pulse" />  
 </div>  
 <h1 className="text-5xl font-bold text-transparent bg-clip-text bg-gradient-to-r from-yellow-500 to-yellow-400 mb-4">  
 Добро пожаловать в NeuroFit  
 </h1>  
 <p className="text-xl text-gray-300 max-w-2xl mx-auto">  
 Революционная платформа, использующая искусственный интеллект для создания  
 персонализированных программ тренировок  
 </p>  
 </div>  
  
 <div className="grid md:grid-cols-3 gap-8 mb-16">  
 <div className="bg-gray-800/80 backdrop-blur-sm p-8 rounded-xl shadow-lg hover:shadow-xl transition duration-300 border border-yellow-500">  
 <Bot className="h-12 w-12 text-yellow-500 mb-4" />  
 <h3 className="text-xl font-semibold mb-2 text-white">ИИ-тренер</h3>  
 <p className="text-gray-300">  
 Умный алгоритм анализирует ваши показатели и адаптирует тренировки  
 в режиме реального времени  
 </p>  
 </div>  
  
 <div className="bg-gray-800/80 backdrop-blur-sm p-8 rounded-xl shadow-lg hover:shadow-xl transition duration-300 border border-yellow-500">  
 <Dumbbell className="h-12 w-12 text-yellow-500 mb-4" />  
 <h3 className="text-xl font-semibold mb-2 text-white">Умные тренировки</h3>  
 <p className="text-gray-300">  
 Персонализированные планы, созданные искусственным интеллектом  
 специально под ваши цели  
 </p>  
 </div>  
  
 <div className="bg-gray-800/80 backdrop-blur-sm p-8 rounded-xl shadow-lg hover:shadow-xl transition duration-300 border border-yellow-500">  
 <Sparkles className="h-12 w-12 text-yellow-500 mb-4" />  
 <h3 className="text-xl font-semibold mb-2 text-white">Прогресс</h3>  
 <p className="text-gray-300">  
 Продвинутая аналитика и рекомендации на основе машинного обучения  
 </p>  
 </div>  
 </div>  
  
 <div className="text-center space-y-8">  
 <div className="max-w-2xl mx-auto bg-gradient-to-r from-gray-800/80 to-yellow-400 text-white p-8 rounded-xl shadow-lg">  
 <h2 className="text-2xl font-bold mb-4">Будущее фитнеса уже здесь</h2>  
 <p className="text-lg">  
 NeuroFit использует передовые алгоритмы искусственного интеллекта,  
 чтобы сделать ваши тренировки максимально эффективными  
 </p>  
 </div>  
 </div>  
 </div>  
 </div>  
 );  
};  
  
export default Home;

Login.tsx

import React, { useState } from 'react';  
import { useNavigate } from 'react-router-dom';  
import { apiService } from '../services/api';  
import { LogIn } from 'lucide-react';  
  
const Login = () => {  
 const [username, setUsername] = useState('');  
 const [password, setPassword] = useState('');  
 const [error, setError] = useState('');  
 const navigate = useNavigate();  
  
 const handleSubmit = async (e: React.FormEvent) => {  
 e.preventDefault();  
 try {  
 const response = await apiService.login({ username, password });  
 localStorage.setItem('accessToken', response.data.access);  
 localStorage.setItem('refreshToken', response.data.refresh);  
 navigate('/profile');  
 } catch (error: any) {  
 if (error.response?.status === 401) {  
 setError('Неверный логин или пароль');  
 return;  
 }  
 setError('Ошибка при входе');  
 }  
 };  
  
 return (  
 <div className="min-h-screen bg-gray-900 flex items-center justify-center px-4">  
 <div className="max-w-md w-full bg-gray-800 rounded-lg shadow-xl p-8 border border-gray-700">  
 <div className="text-center mb-8">  
 <LogIn className="h-12 w-12 text-yellow-500 mx-auto mb-4" />  
 <h2 className="text-3xl font-bold text-white">Вход в систему</h2>  
 </div>  
  
 {error && (  
 <div className="mb-4 p-3 bg-red-900/50 text-red-200 rounded-md border border-red-700">  
 {error}  
 </div>  
 )}  
  
 <form onSubmit={handleSubmit} className="space-y-6">  
 <div>  
 <label className="block text-sm font-medium text-gray-300">  
 Логин  
 </label>  
 <input  
 type="text"  
 required  
 value={username}  
 onChange={(e) => setUsername(e.target.value)}  
 className="mt-1 block w-full rounded-md border-gray-600 bg-gray-700 text-white shadow-sm focus:border-yellow-500 focus:ring-yellow-500"  
 />  
 </div>  
  
 <div>  
 <label className="block text-sm font-medium text-gray-300">  
 Пароль  
 </label>  
 <input  
 type="password"  
 required  
 value={password}  
 onChange={(e) => setPassword(e.target.value)}  
 className="mt-1 block w-full rounded-md border-gray-600 bg-gray-700 text-white shadow-sm focus:border-yellow-500 focus:ring-yellow-500"  
 />  
 </div>  
  
 <button  
 type="submit"  
 className="w-full flex justify-center py-2 px-4 border border-transparent rounded-md shadow-sm text-sm font-medium text-gray-900 bg-yellow-500 hover:bg-yellow-400 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-yellow-500 focus:ring-offset-gray-900"  
 >  
 Войти  
 </button>  
 </form>  
 </div>  
 </div>  
 );  
};  
  
export default Login;

Nutrition.tsx

import React, { useState, useEffect } from 'react';  
import { apiService } from '../services/api';  
import { RefreshCw } from 'lucide-react';  
import { useNavigate } from 'react-router-dom';  
import { NutritionPlan, Meal } from '../types/workouts';  
  
  
const Nutrition: React.FC = () => {  
 const [plan, setPlan] = useState<NutritionPlan | null>(null);  
 const [loading, setLoading] = useState(true);  
 const navigate = useNavigate();  
  
 useEffect(() => {  
 const token = localStorage.getItem('accessToken');  
 if (!token) {  
 navigate('/login');  
 return;  
 }  
 fetchPlan();  
 }, [navigate]);  
  
 const fetchPlan = async () => {  
 try {  
 const token = localStorage.getItem('accessToken');  
 if (!token) {  
 navigate('/login');  
 return;  
 }  
  
 const response = await apiService.getLatestNutritionPlan();  
 const latestPlan: NutritionPlan = response.data;  
  
 if (!latestPlan || shouldGenerateNewPlan(latestPlan.date)) {  
 await handleGenerate();  
 } else {  
 setPlan(latestPlan);  
 }  
 } catch (error: any) {  
 if (error.response?.status === 404) {  
 await handleGenerate();  
 } else {  
 console.error('Ошибка при загрузке плана питания:', error);  
 if (error.response?.status === 401) {  
 navigate('/login');  
 }  
 }  
 } finally {  
 setLoading(false);  
 }  
 };  
  
 const shouldGenerateNewPlan = (planDate: string) => {  
 const planDay = new Date(planDate).toDateString();  
 const today = new Date().toDateString();  
 return planDay !== today;  
 };  
  
 const handleGenerate = async () => {  
 try {  
 const token = localStorage.getItem('accessToken');  
 if (!token) {  
 navigate('/login');  
 return;  
 }  
  
 const response = await apiService.generateNutritionPlan();  
 setPlan(response.data);  
 } catch (error) {  
 console.error('Ошибка при генерации плана питания:', error);  
 if ((error as any).response?.status === 401) {  
 navigate('/login');  
 }  
 }  
 };  
  
 if (loading) {  
 return (  
 <div className="min-h-screen bg-gray-900 flex items-center justify-center">  
 <div className="text-yellow-500 text-xl">Загрузка...</div>  
 </div>  
 );  
 }  
  
 return (  
 <div className="min-h-screen bg-gray-900 py-12 px-4">  
 <div className="max-w-4xl mx-auto">  
 <div className="flex justify-between items-center mb-8">  
 <h1 className="text-3xl font-bold text-gray-100">План питания</h1>  
 <button  
 onClick={handleGenerate}  
 className="bg-yellow-500 text-gray-900 px-4 py-2 rounded-md hover:bg-yellow-400 transition flex items-center gap-2"  
 >  
 <RefreshCw className="w-5 h-5" />  
 Обновить  
 </button>  
 </div>  
  
 {plan ? (  
 <div className="bg-gray-800 rounded-lg p-6 shadow-lg border border-gray-700">  
 <div className="flex justify-between items-start mb-4">  
 <div>  
 <h2 className="text-xl font-semibold text-gray-100 mb-2">  
 План на {new Date(plan.date).toLocaleDateString()}  
 </h2>  
 <p className="text-gray-300">Общая калорийность: {plan.calories} ккал</p>  
 </div>  
 </div>  
  
 {(['breakfast', 'lunch', 'dinner'] as const).map((mealKey) => {  
 const meal = plan.nutrition[mealKey];  
 return (  
 <div  
 key={mealKey}  
 className="bg-gray-700 rounded-md p-4 mb-4"  
 >  
 <h3 className="text-lg font-medium text-gray-100 mb-3">  
 {mealKey === 'breakfast' ? 'Завтрак' : mealKey === 'lunch' ? 'Обед' : 'Ужин'}  
 </h3>  
 <ul className="space-y-2 mb-3">  
 {meal.items.map((item, idx) => (  
 <li key={idx} className="text-gray-300">{item}</li>  
 ))}  
 </ul>  
 <div className="text-sm text-gray-400">  
 {meal.calories} ккал | Б: {meal.proteins}г | Ж: {meal.fats}г | У: {meal.carbs}г  
 </div>  
 </div>  
 );  
 })}  
 </div>  
 ) : (  
 <div className="text-gray-300">Нет плана на сегодня</div>  
 )}  
 </div>  
 </div>  
 );  
};  
  
export default Nutrition;

Profile.tsx

import React, { useState, useEffect } from 'react';  
import { useAuth } from '../contexts/AuthContext';  
import { User, Edit } from 'lucide-react';  
import { apiService } from '../services/api';  
import { BioType } from '../types/user';  
  
const Profile = () => {  
 const { user, setUser } = useAuth();  
 const [isEditing, setIsEditing] = useState(false);  
 const [formData, setFormData] = useState({  
 date\_of\_birth: '',  
 gender: '',  
 height: 0,  
 weight: 0,  
 goal: '',  
} as BioType);  
  
 useEffect(() => {  
 if (user) {  
 setFormData({  
 date\_of\_birth: user.date\_of\_birth || '',  
 gender: user.gender || '',  
 height: user.height || 0,  
 weight: user.weight || 0,  
 goal: user.goal || '',  
 } as BioType);  
 }  
 }, [user]);  
  
const handleSubmit = async (e: React.FormEvent) => {  
 e.preventDefault();  
 try {  
 const response = await apiService.updateProfile(formData);  
 setUser(response.data);  
 setIsEditing(false);  
 } catch (error) {  
 console.error('Ошибка при обновлении данных профиля', error);  
 }  
};  
  
 if (!user) {  
 return <div>Loading...</div>;  
 }  
 return (  
 <div className="min-h-screen bg-gray-900 py-12 px-4">  
 <div className="max-w-3xl mx-auto bg-gray-800 rounded-lg shadow-lg p-8">  
 <div className="flex items-center justify-between mb-8">  
 <div className="flex items-center space-x-4">  
 <div className="bg-yellow-500 p-3 rounded-full">  
 <User className="h-8 w-8 text-white" />  
 </div>  
 <h1 className="text-3xl font-bold text-gray-100">{user.username}</h1>  
 </div>  
 <button  
 onClick={() => setIsEditing(!isEditing)}  
 className="flex items-center space-x-2 text-gray-300 hover:text-yellow-500"  
 >  
 <Edit className="h-5 w-5" />  
 <span>{isEditing ? 'Отменить' : 'Редактировать'}</span>  
 </button>  
 </div>  
  
 {isEditing ? (  
 <form onSubmit={handleSubmit} className="space-y-6">  
 <div>  
 <label className="block text-sm font-medium text-gray-300">  
 Дата рождения  
 </label>  
 <input  
 type="date"  
 value={formData.date\_of\_birth}  
 min="1925-01-01"  
 onChange={(e) =>  
 setFormData({ ...formData, date\_of\_birth: e.target.value })  
 }  
 className="mt-1 block w-full rounded-md border-gray-600 bg-gray-700 text-white shadow-sm focus:border-yellow-500 focus:ring-yellow-500"  
 />  
 </div>  
  
 <div>  
 <label className="block text-sm font-medium text-gray-300">  
 Пол  
 </label>  
 <select  
 value={formData.gender}  
 onChange={(e) =>  
 setFormData({ ...formData, gender: e.target.value })  
 }  
 className="mt-1 block w-full rounded-md border-gray-600 bg-gray-700 text-white shadow-sm focus:border-yellow-500 focus:ring-yellow-500"  
 >  
 <option value="">Выберите пол</option>  
 <option value="male">Мужской</option>  
 <option value="female">Женский</option>  
 </select>  
 </div>  
 <div>  
 <label className="block text-sm font-medium text-gray-300">  
 Рост (см)  
 </label>  
 <input  
 type="number"  
 min="50"  
 max="250"  
 value={formData.height}  
 onChange={(e) =>  
 setFormData({ ...formData, height: parseInt(e.target.value) })  
 }  
 className="mt-1 block w-full rounded-md border-gray-600 bg-gray-700 text-white shadow-sm focus:border-yellow-500 focus:ring-yellow-500"  
 />  
 </div>  
 <div>  
 <label className="block text-sm font-medium text-gray-300">  
 Вес (кг)  
 </label>  
 <input  
 type="number"  
 min="20"  
 max="200"  
 step="0.1"  
 value={formData.weight}  
 onChange={(e) =>  
 setFormData({...formData, weight: parseFloat(e.target.value)})  
 }  
 className="mt-1 block w-full rounded-md border-gray-600 bg-gray-700 text-white shadow-sm focus:border-yellow-500 focus:ring-yellow-500"  
 />  
 </div>  
 <div>  
 <label className="block text-sm font-medium text-gray-300">  
 Цель  
 </label>  
 <select  
 value={formData.goal}  
 onChange={(e) =>  
 setFormData({...formData, goal: e.target.value})  
 }  
 className="mt-1 block w-full rounded-md border-gray-600 bg-gray-700 text-white shadow-sm focus:border-yellow-500 focus:ring-yellow-500"  
 >  
 <option value="">Выберите цель</option>  
 <option value="weight\_loss">Похудение</option>  
 <option value="muscle\_gain">Набор мышечной массы</option>  
 <option value="endurance">Увеличение выносливости</option>  
 <option value="general\_fitness">Общая физическая форма</option>  
 </select>  
 </div>  
 <button  
 type="submit"  
 className="w-full flex justify-center py-2 px-4 border border-transparent rounded-md shadow-sm text-sm font-medium text-white bg-yellow-600 hover:bg-yellow-700 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-yellow-500"  
 >  
 Сохранить изменения  
 </button>  
 </form>  
 ) : (  
 <div className="space-y-6">  
 <div>  
 <h3 className="text-sm font-medium text-gray-300">Дата рождения</h3>  
 <p className="mt-1 text-lg text-gray-100">  
 {user.date\_of\_birth || 'Не указано'}  
 </p>  
 </div>  
 <div>  
 <h3 className="text-sm font-medium text-gray-300">Пол</h3>  
 <p className="mt-1 text-lg text-gray-100">  
 {user.gender === 'male' ? 'Мужской' : 'Женский' || 'Не указано'}  
 </p>  
 </div>  
 <div>  
 <h3 className="text-sm font-medium text-gray-300">Рост (см)</h3>  
 <p className="mt-1 text-lg text-gray-100">  
 {user.height || 'Не указано'}  
 </p>  
 </div>  
 <div>  
 <h3 className="text-sm font-medium text-gray-300">Вес (кг)</h3>  
 <p className="mt-1 text-lg text-gray-100">  
 {user.weight || 'Не указано'}  
 </p>  
 </div>  
 <div>  
 <h3 className="text-sm font-medium text-gray-300">Цель</h3>  
 <p className="mt-1 text-lg text-gray-100">  
 {user.goal === 'weight\_loss'  
 ? 'Похудение'  
 : user.goal === 'muscle\_gain'  
 ? 'Набор мышечной массы'  
 : user.goal === 'endurance'  
 ? 'Увеличение выносливости'  
 : user.goal === 'general\_fitness'  
 ? 'Общая физическая форма'  
 : 'Не указано'}  
 </p>  
 </div>  
 </div>  
 )}  
 </div>  
 </div>  
 );  
};  
  
export default Profile;

Progress.tsx

import React, { useState, useEffect } from 'react';  
import { apiService } from '../services/api';  
import { ProgressChart } from '../types';  
import { Line } from 'react-chartjs-2';  
import {  
 Chart as ChartJS,  
 CategoryScale,  
 LinearScale,  
 PointElement,  
 LineElement,  
 Title,  
 Tooltip,  
 Legend  
} from 'chart.js';  
import { Plus, Trash2 } from 'lucide-react';  
  
ChartJS.register(  
 CategoryScale,  
 LinearScale,  
 PointElement,  
 LineElement,  
 Title,  
 Tooltip,  
 Legend  
);  
  
const Progress: React.FC = () => {  
 const [progress, setProgress] = useState<ProgressChart[]>([]);  
 const [loading, setLoading] = useState(true);  
 const [weight, setWeight] = useState('');  
 const [notes, setNotes] = useState('');  
  
 useEffect(() => {  
 fetchProgress();  
 }, []);  
  
 const fetchProgress = async () => {  
 try {  
 const response = await apiService.getProgress();  
 setProgress(response.data.results || []);  
 } catch (error) {  
 console.error('Ошибка при загрузке прогресса:', error);  
 } finally {  
 setLoading(false);  
 }  
 };  
  
 const handleSubmit = async (e: React.FormEvent) => {  
 e.preventDefault();  
 if (!weight || !notes) return;  
  
 try {  
 const response = await apiService.createProgress({  
 weight: parseFloat(weight),  
 notes  
 });  
 setProgress([...progress, response.data]);  
 setWeight('');  
 setNotes('');  
 } catch (error) {  
 console.error('Ошибка при добавлении записи:', error);  
 }  
 };  
  
 const handleDelete = async (id: number) => {  
 try {  
 await apiService.deleteProgress(id);  
 setProgress(progress.filter(p => p.id !== id));  
 } catch (error) {  
 console.error('Ошибка при удалении записи:', error);  
 }  
 };  
  
 const chartData = {  
 labels: progress.map(p => new Date(p.created\_at).toLocaleDateString()),  
 datasets: [  
 {  
 label: 'Вес (кг)',  
 data: progress.map(p => p.weight),  
 borderColor: 'rgb(234, 179, 8)',  
 backgroundColor: 'rgba(234, 179, 8, 0.5)',  
 tension: 0.4  
 }  
 ]  
 };  
  
 const chartOptions = {  
 responsive: true,  
 plugins: {  
 legend: {  
 position: 'top' as const,  
 labels: {  
 color: 'rgb(229, 231, 235)'  
 }  
 },  
 title: {  
 display: true,  
 text: 'График изменения веса',  
 color: 'rgb(229, 231, 235)'  
 }  
 },  
 scales: {  
 y: {  
 ticks: {  
 color: 'rgb(229, 231, 235)'  
 },  
 grid: {  
 color: 'rgba(75, 85, 99, 0.2)'  
 }  
 },  
 x: {  
 ticks: {  
 color: 'rgb(229, 231, 235)'  
 },  
 grid: {  
 color: 'rgba(75, 85, 99, 0.2)'  
 }  
 }  
 }  
 };  
  
 if (loading) {  
 return (  
 <div className="min-h-screen bg-gray-900 flex items-center justify-center">  
 <div className="text-yellow-500 text-xl">Загрузка...</div>  
 </div>  
 );  
 }  
  
 return (  
 <div className="min-h-screen bg-gray-900 py-12 px-4">  
 <div className="max-w-4xl mx-auto">  
 <h1 className="text-3xl font-bold text-gray-100 mb-8">Мой прогресс</h1>  
  
 {progress.length > 0 ? (  
 <div className="bg-gray-800 rounded-lg p-6 shadow-lg border border-gray-700 mb-8">  
 <Line data={chartData} options={chartOptions} />  
 </div>  
 ) : (  
 <div className="text-gray-400 text-center mb-8">  
 Пока нет записей прогресса.  
 </div>  
 )}  
  
 <form onSubmit={handleSubmit} className="bg-gray-800 rounded-lg p-6 shadow-lg border border-gray-700 mb-8">  
 <div className="grid md:grid-cols-2 gap-4 mb-4">  
 <div>  
 <label className="block text-gray-300 mb-2">Вес (кг)</label>  
 <input  
 type="number"  
 value={weight}  
 onChange={(e) => setWeight(e.target.value)}  
 className="w-full p-2 bg-gray-700 text-white rounded-md border border-gray-600 focus:outline-none focus:ring-2 focus:ring-yellow-500"  
 step="0.1"  
 required  
 />  
 </div>  
 <div>  
 <label className="block text-gray-300 mb-2">Заметки</label>  
 <input  
 type="text"  
 value={notes}  
 onChange={(e) => setNotes(e.target.value)}  
 className="w-full p-2 bg-gray-700 text-white rounded-md border border-gray-600 focus:outline-none focus:ring-2 focus:ring-yellow-500"  
 required  
 />  
 </div>  
 </div>  
 <button  
 type="submit"  
 className="bg-yellow-500 text-gray-900 px-4 py-2 rounded-md hover:bg-yellow-400 transition flex items-center gap-2"  
 >  
 <Plus className="w-5 h-5" />  
 Добавить запись  
 </button>  
 </form>  
  
 <div className="space-y-4">  
 {progress.map((entry) => (  
 <div  
 key={entry.id}  
 className="bg-gray-800 rounded-lg p-6 shadow-lg border border-gray-700"  
 >  
 <div className="flex justify-between items-start">  
 <div>  
 <div className="text-2xl font-semibold text-gray-100 mb-2">  
 {entry.weight} кг  
 </div>  
 <div className="text-gray-300">{entry.notes}</div>  
 <div className="text-sm text-gray-400 mt-2">  
 {new Date(entry.created\_at).toLocaleDateString()}  
 </div>  
 </div>  
 <button  
 onClick={() => handleDelete(entry.id)}  
 className="text-gray-400 hover:text-red-500 transition"  
 >  
 <Trash2 className="w-5 h-5" />  
 </button>  
 </div>  
 </div>  
 ))}  
 </div>  
 </div>  
 </div>  
 );  
};  
  
export default Progress;

Register.tsx

import React, { useState } from 'react';  
import { useNavigate } from 'react-router-dom';  
import { useAuth } from '../contexts/AuthContext';  
import { UserPlus, Bot } from 'lucide-react';  
import toast from 'react-hot-toast';  
import { apiService } from "../services/api";  
  
type TrainingGoal = 'weight\_loss' | 'muscle\_gain' | 'endurance' | 'general\_fitness';  
  
const Register = () => {  
 const [showBiometrics, setShowBiometrics] = useState(false);  
 const [error, setError] = useState('');  
 const navigate = useNavigate();  
 const { register } = useAuth();  
  
 const [username, setUsername] = useState('');  
 const [email, setEmail] = useState('');  
 const [password, setPassword] = useState('');  
 const [password2, setPassword2] = useState('');  
  
 const [date\_of\_birth, setDateOfBirth] = useState('');  
 const [gender, setGender] = useState<'male' | 'female'>('male');  
 const [weight, setWeight] = useState('');  
 const [height, setHeight] = useState('');  
 const [goal, setGoal] = useState<TrainingGoal>('general\_fitness');  
 const [has\_equipment, setHasEquipment] = useState(false);  
  
 const handleInitialRegister = async (e: React.FormEvent) => {  
 e.preventDefault();  
 if (password !== password2) {  
 setError('Пароли не совпадают');  
 return;  
 }  
 try {  
 await register(username, email, password);  
 setShowBiometrics(true);  
 toast.success('Регистрация успешна! Теперь давайте настроим ваш профиль.');  
 } catch (error: any) {  
 setError(error.response?.data?.password || 'Ошибка при регистрации');  
 }  
 };  
  
 const handleBiometricsSubmit = async (e: React.FormEvent) => {  
 e.preventDefault();  
 try {  
 await apiService.updateProfile({  
 date\_of\_birth,  
 gender,  
 weight: parseFloat(weight),  
 height: parseInt(height),  
 goal,  
 has\_equipment  
 });  
 toast.success('Профиль успешно обновлен!');  
 navigate('/profile');  
 window.location.reload();  
 } catch (error: any) {  
 setError(error.response?.data?.message || 'Ошибка при обновлении профиля');  
 }  
 };  
  
 if (showBiometrics) {  
 return (  
 <div className="min-h-screen bg-gray-900 flex items-center justify-center px-4">  
 <div className="max-w-md w-full bg-gray-800 rounded-lg shadow-xl p-8 border border-gray-700">  
 <div className="text-center mb-8">  
 <Bot className="h-12 w-12 text-yellow-500 mx-auto mb-4" />  
 <h2 className="text-3xl font-bold text-white">Настройка профиля</h2>  
 <p className="text-gray-400 mt-2">Давайте настроим ваш профиль для персонализированных рекомендаций</p>  
 </div>  
  
 {error && (  
 <div className="mb-4 p-3 bg-red-900/50 text-red-200 rounded-md border border-red-700">  
 {error}  
 </div>  
 )}  
  
 <form onSubmit={handleBiometricsSubmit} className="space-y-6">  
 <div className="grid grid-cols-2 gap-4">  
 <div>  
 <label className="block text-sm font-medium text-gray-300">  
 Пол  
 </label>  
 <select  
 value={gender}  
 onChange={(e) => setGender(e.target.value as 'male' | 'female')}  
 className="mt-1 block w-full rounded-md border-gray-600 bg-gray-700 text-white shadow-sm focus:border-yellow-500 focus:ring-yellow-500"  
 >  
 <option value="male">Мужской</option>  
 <option value="female">Женский</option>  
 </select>  
 </div>  
  
 <div>  
 <label className="block text-sm font-medium text-gray-300">  
 Дата рождения  
 </label>  
 <input  
 type="date"  
 min="1925-01-01"  
 required  
 value={date\_of\_birth}  
 onChange={(e) => setDateOfBirth(e.target.value)}  
 className="mt-1 block w-full rounded-md border-gray-600 bg-gray-700 text-white shadow-sm focus:border-yellow-500 focus:ring-yellow-500"  
 />  
 </div>  
 </div>  
  
 <div className="grid grid-cols-2 gap-4">  
 <div>  
 <label className="block text-sm font-medium text-gray-300">  
 Вес (кг)  
 </label>  
 <input  
 type="number"  
 required  
 step="0.1"  
 min="20"  
 max="200"  
 value={weight}  
 onChange={(e) => setWeight(e.target.value)}  
 className="mt-1 block w-full rounded-md border-gray-600 bg-gray-700 text-white shadow-sm focus:border-yellow-500 focus:ring-yellow-500"  
 />  
 </div>  
  
 <div>  
 <label className="block text-sm font-medium text-gray-300">  
 Рост (см)  
 </label>  
 <input  
 type="number"  
 required  
 min="50"  
 max="250"  
 value={height}  
 onChange={(e) => setHeight(e.target.value)}  
 className="mt-1 block w-full rounded-md border-gray-600 bg-gray-700 text-white shadow-sm focus:border-yellow-500 focus:ring-yellow-500"  
 />  
 </div>  
 </div>  
  
 <div>  
 <label className="block text-sm font-medium text-gray-300">  
 Цель тренировок  
 </label>  
 <select  
 value={goal}  
 onChange={(e) => setGoal(e.target.value as TrainingGoal)}  
 className="mt-1 block w-full rounded-md border-gray-600 bg-gray-700 text-white shadow-sm focus:border-yellow-500 focus:ring-yellow-500"  
 >  
 <option value="weight\_loss">Снижение веса</option>  
 <option value="muscle\_gain">Набор мышечной массы</option>  
 <option value="endurance">Выносливость</option>  
 <option value="general\_fitness">Общая физическая подготовка</option>  
 </select>  
 </div>  
  
 <div className="flex items-center">  
 <input  
 type="checkbox"  
 id="has\_equipment"  
 checked={has\_equipment}  
 onChange={(e) => setHasEquipment(e.target.checked)}  
 className="h-4 w-4 text-yellow-500 focus:ring-yellow-500 border-gray-600 rounded bg-gray-700"  
 />  
 <label htmlFor="has\_equipment" className="ml-2 block text-sm text-gray-300">  
 У меня есть спортивное оборудование  
 </label>  
 </div>  
  
 <button  
 type="submit"  
 className="w-full flex justify-center items-center py-2 px-4 border border-transparent rounded-md shadow-sm text-sm font-medium text-gray-900 bg-yellow-500 hover:bg-yellow-400 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-yellow-500 focus:ring-offset-gray-900"  
 >  
 Завершить настройку  
 </button>  
 </form>  
 </div>  
 </div>  
 );  
 }  
  
 return (  
 <div className="min-h-screen bg-gray-900 flex items-center justify-center px-4">  
 <div className="max-w-md w-full bg-gray-800 rounded-lg shadow-xl p-8 border border-gray-700">  
 <div className="text-center mb-8">  
 <UserPlus className="h-12 w-12 text-yellow-500 mx-auto mb-4" />  
 <h2 className="text-3xl font-bold text-white">Регистрация</h2>  
 </div>  
  
 {error && (  
 <div className="mb-4 p-3 bg-red-900/50 text-red-200 rounded-md border border-red-700">  
 {error}  
 </div>  
 )}  
  
 <form onSubmit={handleInitialRegister} className="space-y-6">  
 <div>  
 <label className="block text-sm font-medium text-gray-300">  
 Логин  
 </label>  
 <input  
 type="text"  
 required  
 value={username}  
 onChange={(e) => setUsername(e.target.value)}  
 className="mt-1 block w-full rounded-md border-gray-600 bg-gray-700 text-white shadow-sm focus:border-yellow-500 focus:ring-yellow-500"  
 />  
 </div>  
  
 <div>  
 <label className="block text-sm font-medium text-gray-300">  
 Email  
 </label>  
 <input  
 type="email"  
 required  
 value={email}  
 onChange={(e) => setEmail(e.target.value)}  
 className="mt-1 block w-full rounded-md border-gray-600 bg-gray-700 text-white shadow-sm focus:border-yellow-500 focus:ring-yellow-500"  
 />  
 </div>  
  
 <div>  
 <label className="block text-sm font-medium text-gray-300">  
 Пароль  
 </label>  
 <input  
 type="password"  
 required  
 value={password}  
 onChange={(e) => setPassword(e.target.value)}  
 className="mt-1 block w-full rounded-md border-gray-600 bg-gray-700 text-white shadow-sm focus:border-yellow-500 focus:ring-yellow-500"  
 />  
 </div>  
  
 <div>  
 <label className="block text-sm font-medium text-gray-300">  
 Подтвердите пароль  
 </label>  
 <input  
 type="password"  
 required  
 value={password2}  
 onChange={(e) => setPassword2(e.target.value)}  
 className="mt-1 block w-full rounded-md border-gray-600 bg-gray-700 text-white shadow-sm focus:border-yellow-500 focus:ring-yellow-500"  
 />  
 </div>  
  
 <button  
 type="submit"  
 className="w-full flex justify-center items-center py-2 px-4 border border-transparent rounded-md shadow-sm text-sm font-medium text-gray-900 bg-yellow-500 hover:bg-yellow-400 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-yellow-500 focus:ring-offset-gray-900"  
 >  
 Зарегистрироваться  
 <UserPlus className="ml-2 h-4 w-4" />  
 </button>  
 </form>  
 </div>  
 </div>  
 );  
};  
  
export default Register;

Workouts.tsx

import React, { useState, useEffect } from 'react';  
import { apiService } from '../services/api';  
import { Workout } from '../types/workouts';  
import { Check, RefreshCw } from 'lucide-react';  
import { useNavigate } from 'react-router-dom';  
  
const Workouts: React.FC = () => {  
 const [workout, setWorkout] = useState<Workout | null>(null);  
 const [loading, setLoading] = useState(true);  
 const navigate = useNavigate();  
  
 useEffect(() => {  
 const token = localStorage.getItem('accessToken');  
 if (!token) {  
 navigate('/login');  
 return;  
 }  
 fetchWorkout();  
 }, [navigate]);  
  
 const fetchWorkout = async () => {  
 setLoading(true);  
 try {  
 const token = localStorage.getItem('accessToken');  
 if (!token) {  
 navigate('/login');  
 return;  
 }  
  
 const response = await apiService.getLatestWorkout();  
 const latestWorkout: Workout = response.data;  
  
 if (!latestWorkout || shouldGenerateNewWorkout(latestWorkout.date)) {  
 await handleGenerate();  
 } else {  
 setWorkout(latestWorkout);  
 }  
 } catch (error: any) {  
 if (error.response?.status === 404) {  
 // Нет тренировки — генерируем новую  
 await handleGenerate();  
 } else {  
 console.error('Ошибка при загрузке тренировки:', error);  
 if (error.response?.status === 401) {  
 navigate('/login');  
 }  
 }  
 } finally {  
 setLoading(false);  
 }  
 };  
  
 const shouldGenerateNewWorkout = (planDate: string) => {  
 const planDay = new Date(planDate).toDateString();  
 const today = new Date().toDateString();  
 return planDay !== today;  
 };  
  
 const handleGenerate = async () => {  
 setLoading(true);  
 try {  
 const token = localStorage.getItem('accessToken');  
 if (!token) {  
 navigate('/login');  
 return;  
 }  
  
 const response = await apiService.generateWorkout();  
 setWorkout(response.data);  
 } catch (error: any) {  
 console.error('Ошибка при генерации тренировки:', error);  
 if (error.response?.status === 401) {  
 navigate('/login');  
 }  
 } finally {  
 setLoading(false);  
 }  
 };  
  
 const handleComplete = async () => {  
 if (!workout) return;  
 try {  
 const token = localStorage.getItem('accessToken');  
 if (!token) {  
 navigate('/login');  
 return;  
 }  
 await apiService.completeWorkout(workout.id);  
 setWorkout(prev => (prev ? { ...prev, completed: true } : prev));  
 } catch (error) {  
 console.error('Ошибка при завершении тренировки:', error);  
 }  
 };  
  
 if (loading) {  
 return (  
 <div className="min-h-screen bg-gray-900 flex items-center justify-center">  
 <div className="text-yellow-500 text-xl">Загрузка...</div>  
 </div>  
 );  
 }  
  
 return (  
 <div className="min-h-screen bg-gray-900 py-12 px-4">  
 <div className="max-w-4xl mx-auto">  
 <div className="flex justify-between items-center mb-8">  
 <h1 className="text-3xl font-bold text-gray-100">Тренировка на сегодня</h1>  
 <button  
 onClick={handleGenerate}  
 className="bg-yellow-500 text-gray-900 px-4 py-2 rounded-md hover:bg-yellow-400 transition flex items-center gap-2"  
 >  
 <RefreshCw className="w-5 h-5" />  
 Обновить  
 </button>  
 </div>  
  
 {workout ? (  
 <div className="bg-gray-800 rounded-lg p-6 shadow-lg border border-gray-700">  
 <div className="flex justify-between items-start mb-4">  
 <div>  
 <h2 className="text-xl font-semibold text-gray-100 mb-2">  
 {new Date(workout.date).toLocaleDateString()}  
 </h2>  
 </div>  
 {!workout.completed && (  
 <button  
 onClick={handleComplete}  
 className="bg-green-500 text-white p-2 rounded-md hover:bg-green-600 transition"  
 >  
 <Check className="w-5 h-5" />  
 </button>  
 )}  
 </div>  
  
 <div className="space-y-3">  
 {workout.plan.map((exercise, index) => (  
 <div  
 key={index}  
 className="bg-gray-700 rounded-md p-3 text-gray-100"  
 >  
 <div className="font-medium">{exercise.name}</div>  
 <div className="text-sm text-gray-300 mb-2">  
 {exercise.sets} подходов × {exercise.reps} повторений  
 </div>  
 <div className="text-sm text-gray-400">  
 {exercise.description}  
 </div>  
 </div>  
 ))}  
 </div>  
 </div>  
 ) : (  
 <div className="text-gray-300 text-center py-8">  
 Нет доступной тренировки  
 </div>  
 )}  
 </div>  
 </div>  
 );  
};  
  
export default Workouts;

api.ts

import axios, { AxiosError, AxiosResponse, InternalAxiosRequestConfig } from 'axios';  
import {  
 RefreshTokenType, TokenType, BioType, RegisterType,  
 CoachGenerateType, Workout, Nutrition, Post, Comment, ProgressChart  
} from '../types';  
  
const API\_BASE\_URL = 'http://localhost:8000/api';  
  
const getAccessToken = () => localStorage.getItem('accessToken');  
const getRefreshToken = () => localStorage.getItem('refreshToken');  
const setAccessToken = (token: string) => localStorage.setItem('accessToken', token);  
const clearTokens = () => {  
 localStorage.removeItem('accessToken');  
 localStorage.removeItem('refreshToken');  
};  
  
export const api = axios.create({ baseURL: API\_BASE\_URL });  
  
// Attach access token to request headers  
api.interceptors.request.use(  
 (config: InternalAxiosRequestConfig) => {  
 const token = getAccessToken();  
 if (token) config.headers.Authorization = `Bearer ${token}`;  
 return config;  
 },  
 (error: AxiosError) => Promise.reject(error)  
);  
  
// Handle 401 errors and try to refresh token  
api.interceptors.response.use(  
 (response: AxiosResponse) => response,  
 async (error: AxiosError) => {  
 const originalRequest = error.config as InternalAxiosRequestConfig & { \_retry?: boolean };  
 if (error.response?.status === 401 && !originalRequest.\_retry) {  
 originalRequest.\_retry = true;  
 try {  
 const refreshToken = getRefreshToken();  
 if (refreshToken) {  
 const { data } = await api.post<{ access: string }>('/token/refresh/', { refresh\_token: refreshToken });  
 setAccessToken(data.access);  
 originalRequest.headers.Authorization = `Bearer ${data.access}`;  
 return api(originalRequest);  
 }  
 } catch {  
 clearTokens();  
 window.location.href = '/login';  
 }  
 }  
 return Promise.reject(error);  
 }  
);  
  
// --- API Service ---  
  
export const apiService = {  
 // Auth  
 login: (data: TokenType) => api.post('/token/', data),  
 refreshToken: (data: RefreshTokenType) => api.post('/token/refresh/', data),  
 logout: () => api.post('/logout/'),  
  
 // User  
 register: (data: RegisterType) => api.post('/users/register/', data),  
 getProfile: () => api.get('/users/profile/'),  
 updateProfile: (data: BioType) => api.patch('/users/profile/', data),  
  
 // Coach  
 getGenerate: (data: CoachGenerateType) => api.post('/coaches/generate/', data),  
  
 // Workout  
 generateWorkout: () => api.post<Workout>('/workouts/generate/'),  
 getLatestWorkout: () => api.get<Workout>('/workouts/latest/'),  
 completeWorkout: (id: number) => axios.post(`/workouts/${id}/complete/`),  
  
  
 // Nutrition  
 generateNutritionPlan: () => api.post<Nutrition>('/nutrition/generate/'),  
 getLatestNutritionPlan: () => api.get<Nutrition>('/nutrition/latest/'),  
  
 // Posts  
 getPosts: () => api.get<Post[]>('/posts/'),  
 getPost: (id: number) => api.get<Post>(`/posts/${id}/`),  
 createPost: (data: Partial<Post>) => api.post<Post>('/posts/', data),  
 updatePost: (id: number, data: Partial<Post>) => api.patch<Post>(`/posts/${id}/`, data),  
 deletePost: (id: number) => api.delete(`/posts/${id}/`),  
 likePost: (id: number) => api.post(`/posts/${id}/like/`),  
  
 // Comments  
 getComments: (postId: number) => api.get<Comment[]>(`/posts/${postId}/comments/`),  
 addComment: (postId: number, data: { content: string }) =>  
 api.post<Comment>(`/posts/${postId}/comments/`, data),  
  
 // Progress  
 getProgress: () => api.get<ProgressChart[]>('/progress/'),  
 getProgressEntry: (id: number) => api.get<ProgressChart>(`/progress/${id}/`),  
 createProgress: (data: { weight: number; notes: string }) =>  
 api.post<ProgressChart>('/progress/', data),  
 updateProgress: (id: number, data: { weight: number; notes: string }) =>  
 api.patch<ProgressChart>(`/progress/${id}/`, data),  
 deleteProgress: (id: number) => api.delete(`/progress/${id}/`),  
};

auth.ts

import { UserType } from "./user.ts";  
  
export interface AuthContextType {  
 user: UserType | null;  
 setUser: (user: UserType | null) => void;  
 isAuthenticated: boolean;  
 login: (username: string, password: string) => Promise<void>;  
 register: (username: string, email: string, password: string) => Promise<void>;  
 logout: () => void;  
}

coach.ts

export interface CoachGenerateType {  
 messages: Array<{ role: string; content: string }>;  
}

index.ts

export interface Post {  
 id: number;  
 title: string;  
 content: string;  
 author: User;  
 created\_at: string;  
 likes\_count: number;  
 comments\_count: number;  
}  
  
export interface Comment {  
 id: number;  
 content: string;  
 author: User;  
 created\_at: string;  
}  
  
export interface ProgressChart {  
 id: number;  
 weight: number;  
 notes: string;  
 created\_at: string;  
}  
  
export interface User {  
 id: number;  
 username: string;  
 email: string;  
 profile: UserProfile;  
}  
  
export interface UserProfile {  
 height: number;  
 weight: number;  
 age: number;  
 gender: string;  
 goals: string[];  
}

nutrition.ts

interface Meal {  
 items: string[];  
 calories: number;  
 proteins: number;  
 fats: number;  
 carbs: number;  
}  
  
interface NutritionPlan {  
 id: number;  
 date: string;  
 nutrition: {  
 breakfast: Meal;  
 lunch: Meal;  
 dinner: Meal;  
 };  
 calories: number;  
}

token.ts

export interface TokenType {  
 username: string;  
 password: string;  
}  
  
export interface RefreshTokenType {  
 refresh\_token: string;  
}

user.ts

export interface UserType {  
 id: string;  
 username: string;  
 email: string;  
 date\_of\_birth?: string;  
 gender?: 'male' | 'female';  
 height?: number;  
 weight?: number;  
 goal?: 'weight\_loss' | 'muscle\_gain' | 'endurance' | 'general\_fitness';  
}  
  
export interface RegisterType {  
 username: string;  
 email: string;  
 password: string;  
}  
  
export interface BioType {  
 date\_of\_birth: string;  
 gender: 'male' | 'female';  
 height: number;  
 weight: number;  
 goal: 'weight\_loss' | 'muscle\_gain' | 'endurance' | 'general\_fitness';  
}

workouts.ts

export interface Workout {  
 id: number;  
 date: string;  
 plan: Exercise[];  
 completed: boolean;  
}  
  
export interface Exercise {  
 name: string;  
 sets: number;  
 reps: number;  
 description: string;  
}

# ПРИЛОЖЕНИЕ В ПРЕЗЕНТАЦИЯ

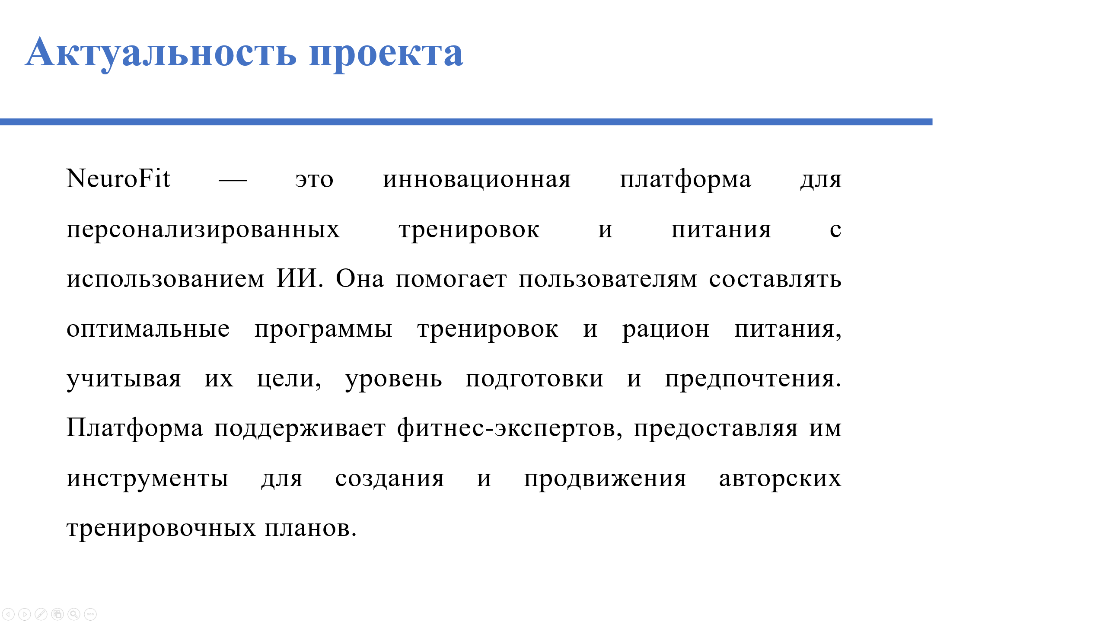


Рисунок 1 – Слайд №1

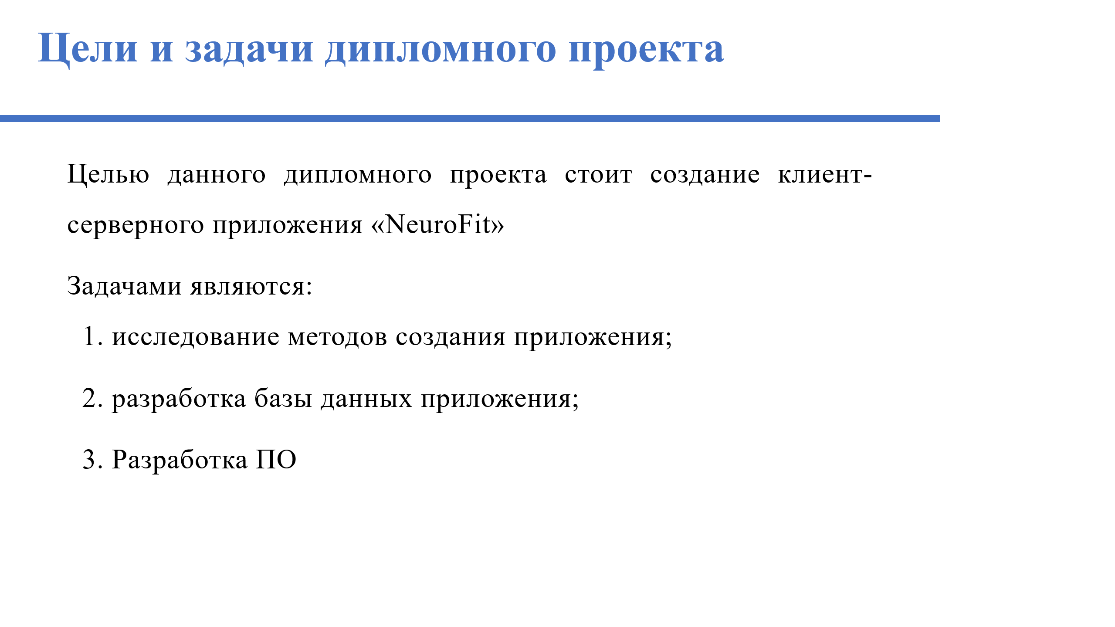


Рисунок 2 – Слайд №2



Рисунок 3 – Слайд №3

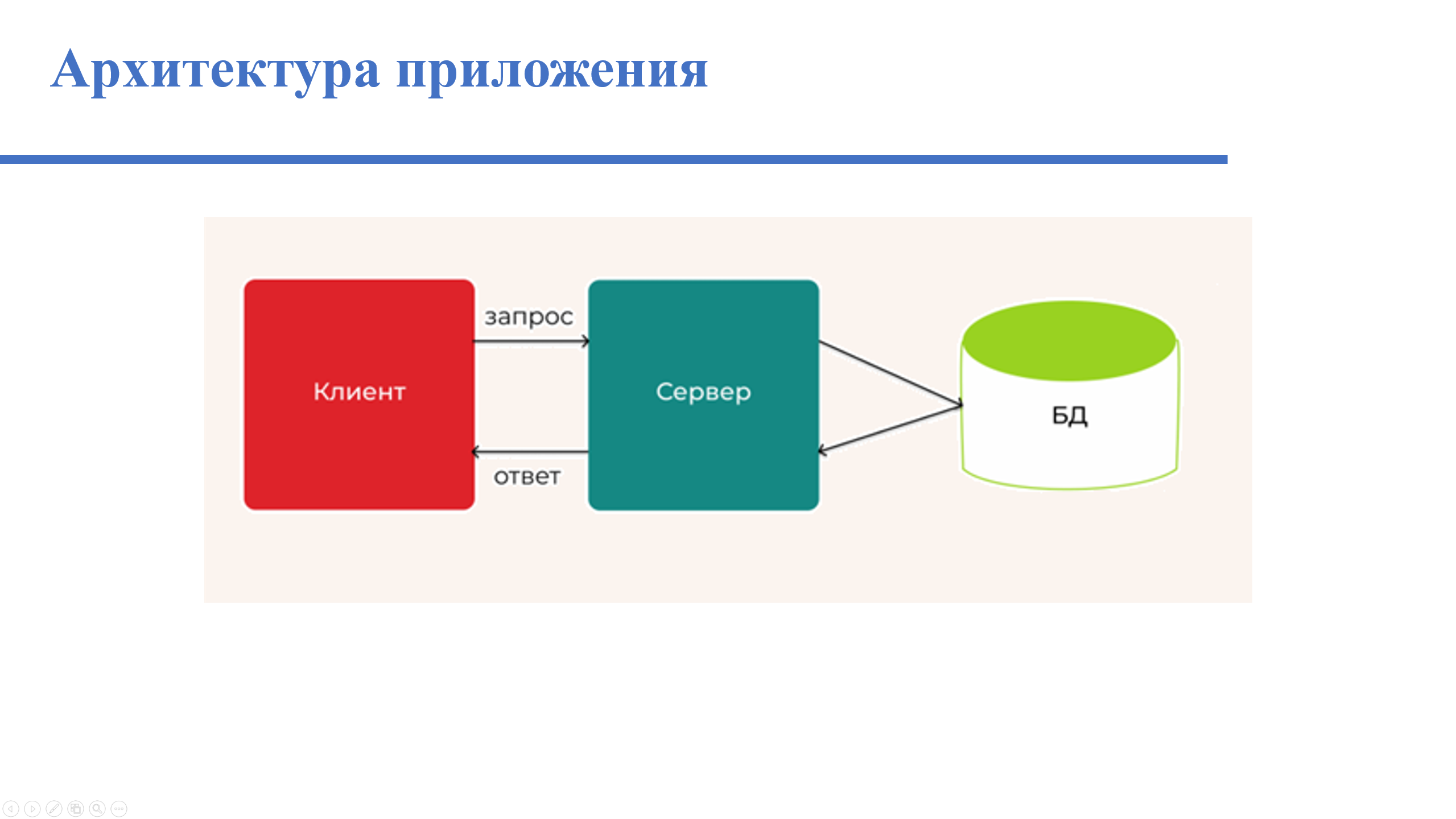


Рисунок 4 – Слайд №4

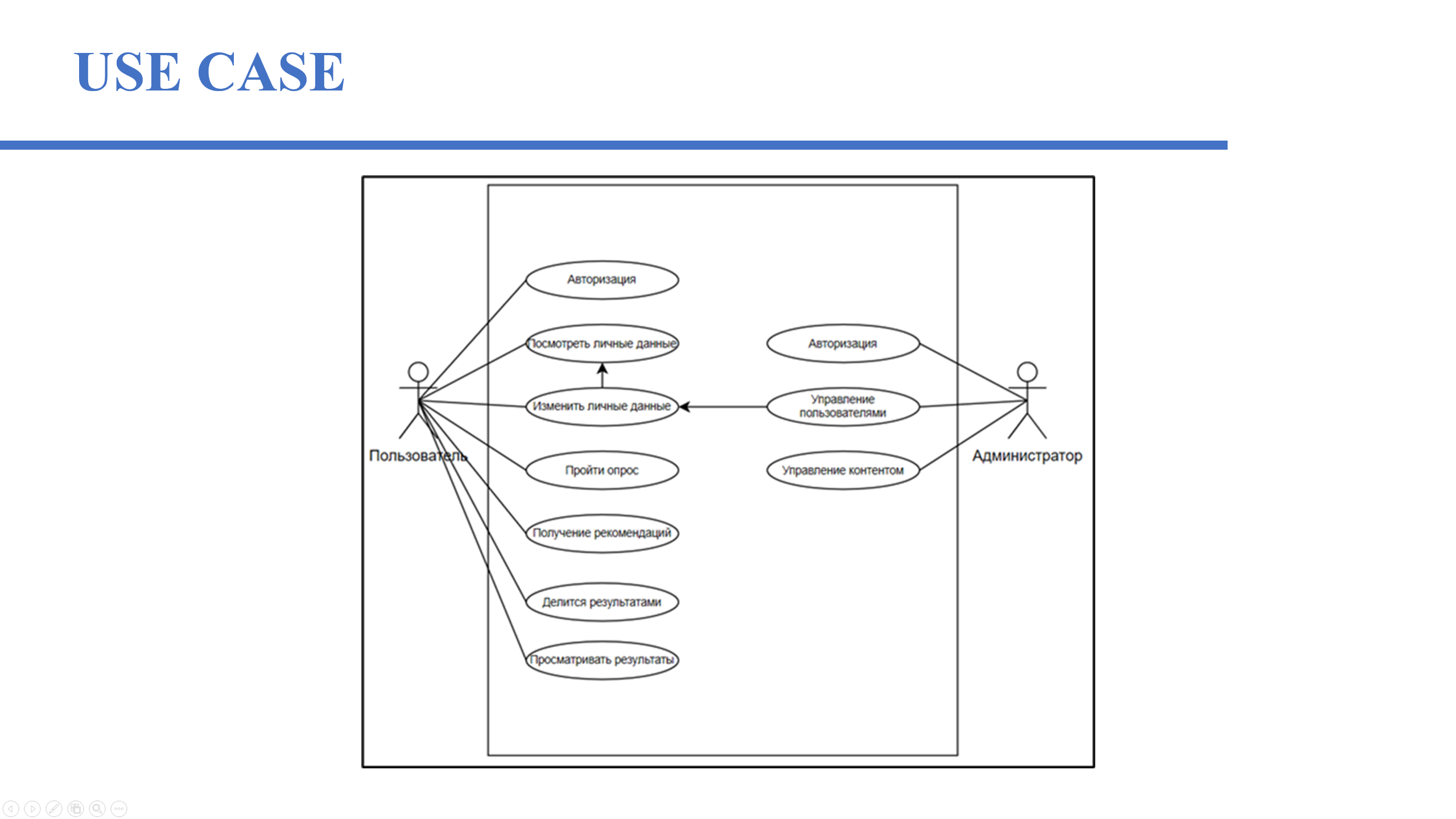


Рисунок 5 – Слайд №5

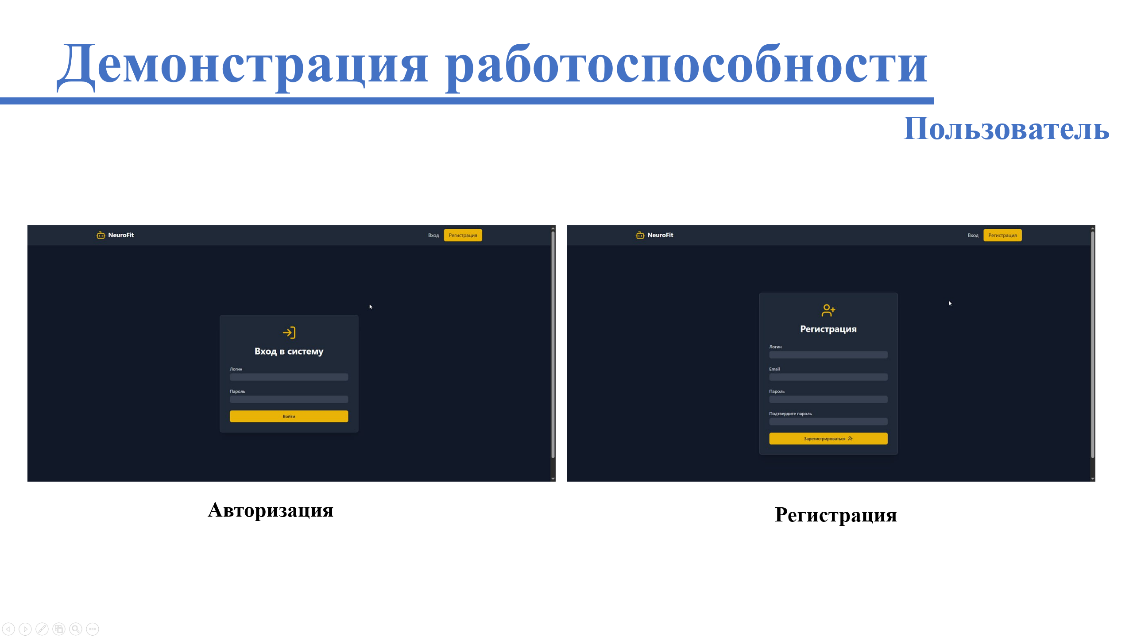


Рисунок 6 – Слайд №6

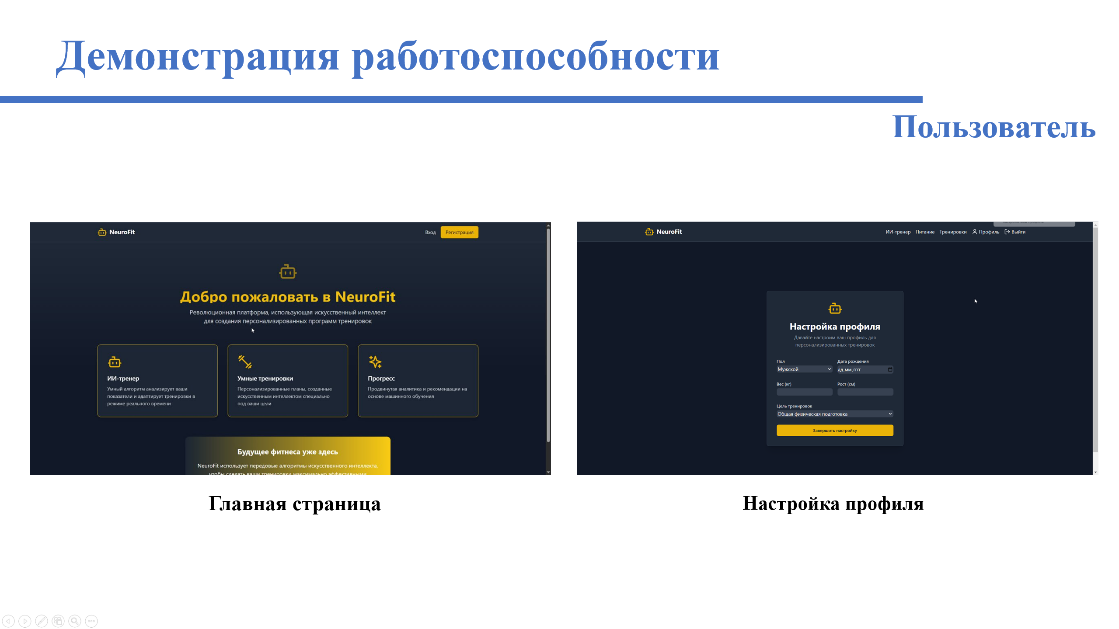


Рисунок 7 – Слайд №7

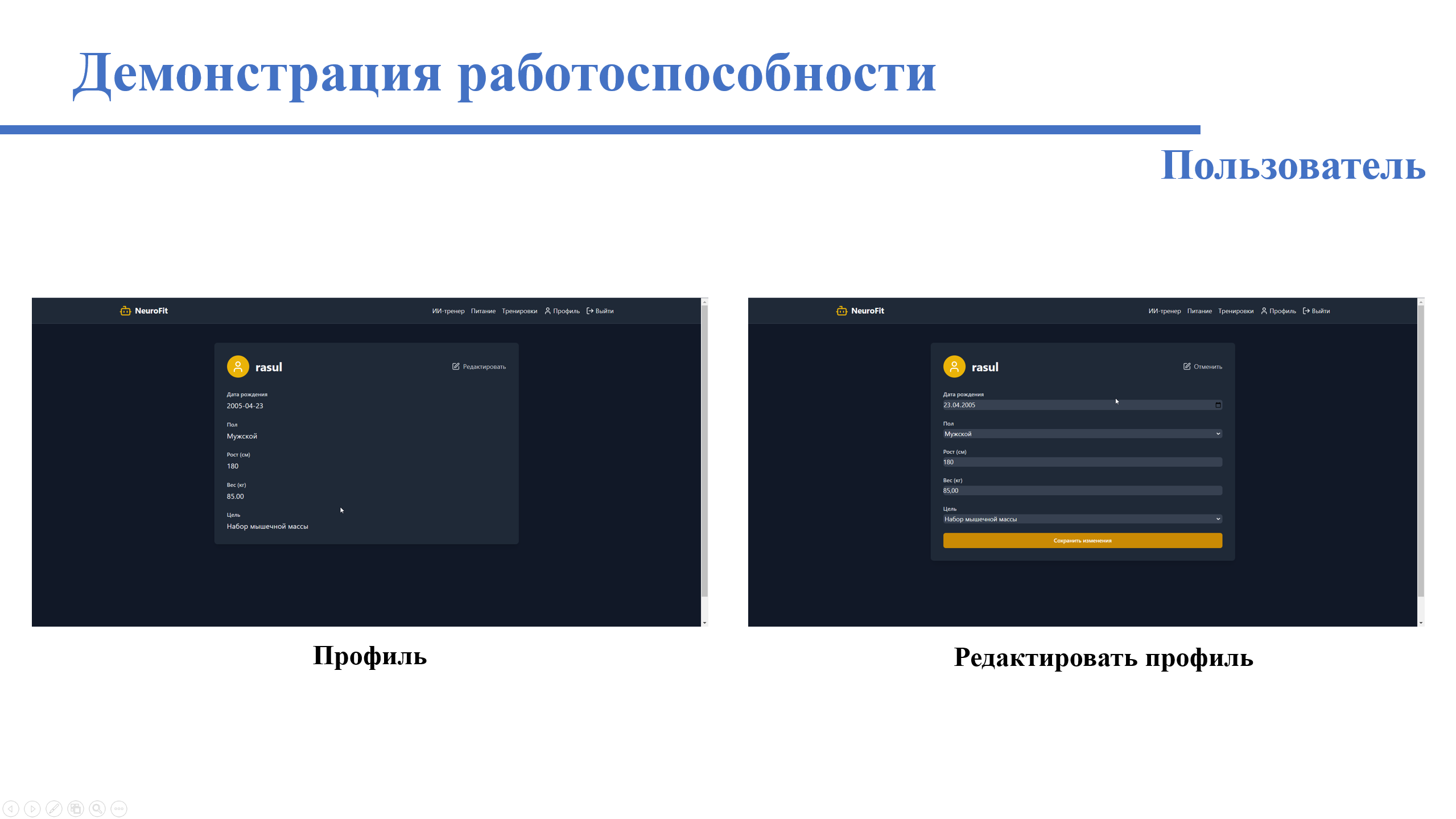


Рисунок 8 – Слайд №8

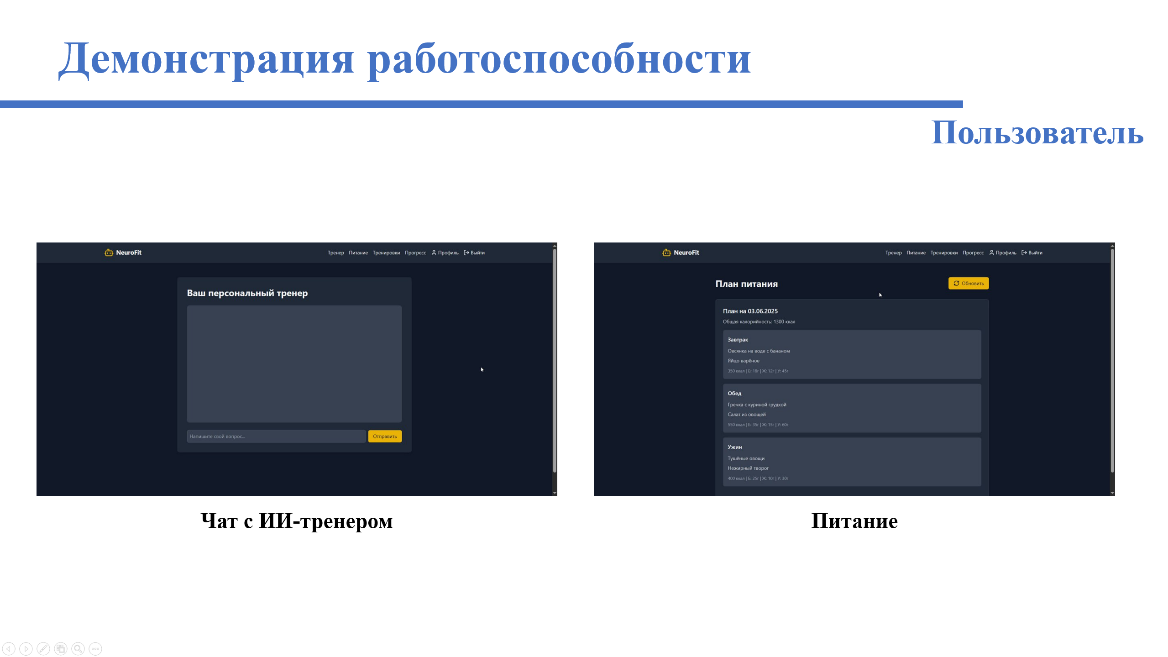


Рисунок 9 – Слайд №9

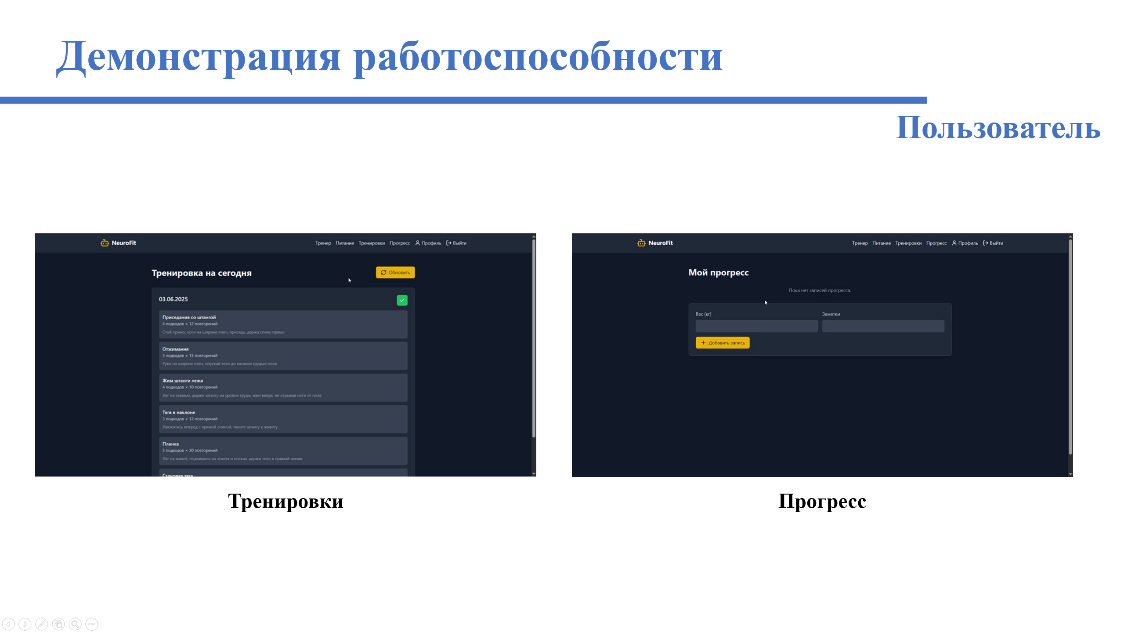


Рисунок 10 – Слайд №10

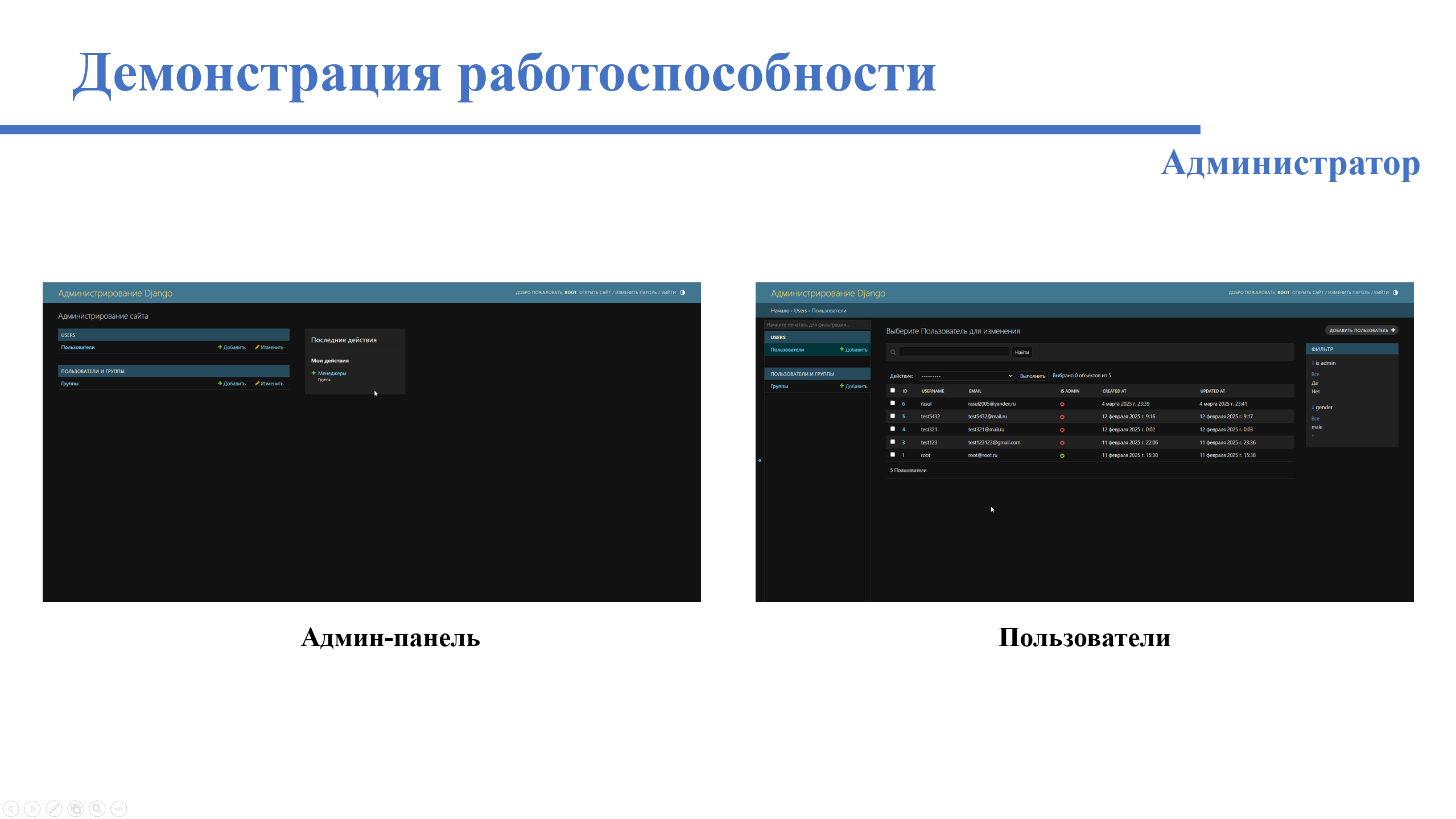


Рисунок 11 – Слайд №11

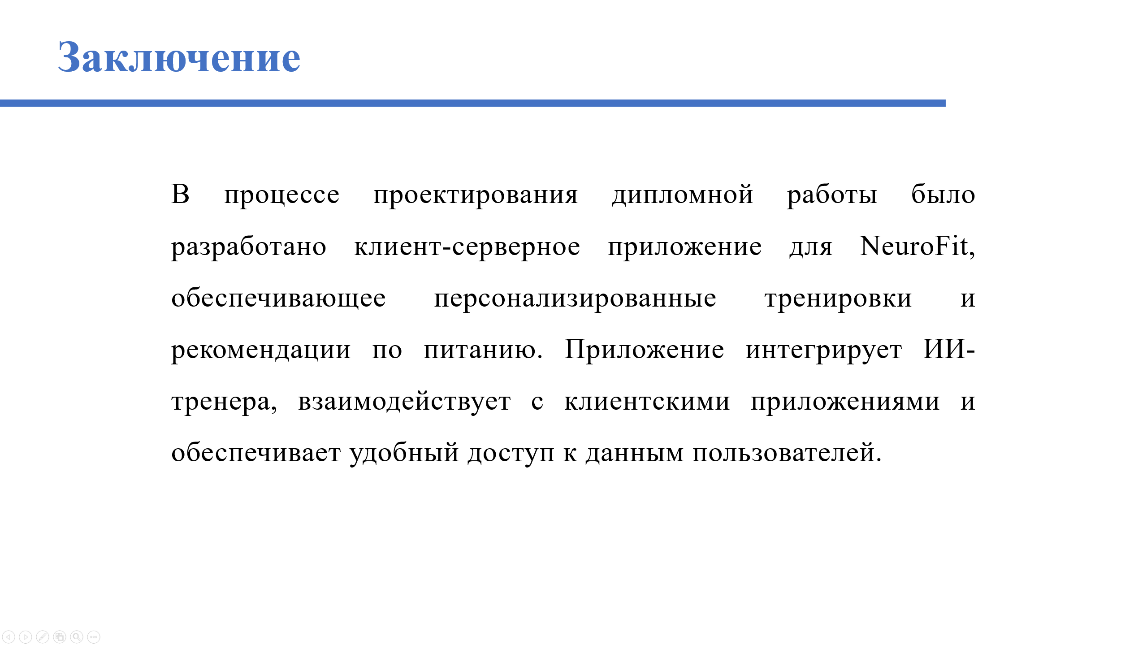


Рисунок 12 – Слайд №12



Рисунок 13 – Слайд №13