Geekbrains

**Разработка web-приложения для работы**

**с клиентами фитнес клуба**

Программа:

Java-разработчик. Специалист

Бродовикова Диана Викторовна



Югорск

2024 г.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc169211701)

[1. АНАЛИТИЧЕСКАЯ ЧАСТЬ 7](#_Toc169211702)

[1.1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 7](#_Toc169211703)

[1.1.1 Словарь предметной области 7](#_Toc169211704)

[1.1.2 Анализ системы работы с клиентами объекта исследования 8](#_Toc169211705)

[1.2 АНАЛИЗ ИСПОЛЬЗУЕМОГО ПРОГРАМНОГО ОБЕСПЕЧЕНИЯ 11](#_Toc169211706)

[2. ПРОЕКТНАЯ ЧАСТЬ 16](#_Toc169211707)

[2.1 ТЕХНИЧЕСКОЕ ЗАДАНИЕ 16](#_Toc169211708)

[2.2 РАЗРАБОТКА ПРИЛОЖЕНИЯ 17](#_Toc169211709)

[2.2.1 Моделирование схемы данных 17](#_Toc169211710)

[2.2.2 Структура исследуемого проекта 19](#_Toc169211711)

[2.2.3 Идентификация, аутентификация, авторизация и регистрация пользователей 26](#_Toc169211712)

[2.2.4 Главная страница приложения 29](#_Toc169211713)

[2.2.5 Реализация работы с тренировочным планом и отчетами по тренировкам 31](#_Toc169211714)

[2.2.6 Реализация работы с планом питания 36](#_Toc169211715)

[2.3 ОБЕСПЕЧЕНИЕ ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ 38](#_Toc169211716)

[2.4 ТЕСТИРОВАНИЕ И ОЦЕНКА КАЧЕСТВА ИНФОРМАЦИОННОЙ СИСТЕМЫ 39](#_Toc169211717)

[2.5 ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ 47](#_Toc169211718)

[2.6 РАСЧЕТ ЭКОНОМИЧЕСКОЙ ЭФФЕКТИВНОСТИ ПРОЕКТА 48](#_Toc169211719)

[ЗАКЛЮЧЕНИЕ 50](#_Toc169211720)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 51](#_Toc169211721)

[Приложение А 55](#_Toc169211722)

[Приложение Б 56](#_Toc169211723)

[Приложение В 57](#_Toc169211724)

[Приложение Г 58](#_Toc169211725)

[Приложение Д 59](#_Toc169211726)

# ВВЕДЕНИЕ

В современном мире все больше внимания уделяется здоровому образу жизни и физической активности. Тренажерные залы становятся популярным местом для тех, кто стремится поддерживать свою физическую форму и заботится о своем здоровье. Однако, чтобы достичь поставленных целей в тренировочном процессе, необходимо не только правильно и систематически заниматься, но и иметь поддержку и контроль со стороны опытного тренера.

Некоторые тренера используют для ведения клиентов Google таблицы или просто excel таблицы. Что является не очень удобным инструментом для работы с клиентами, так как на каждого клиента нужно иметь свой файл с несколькими таблицами, а когда клиентов становится много, в этих таблицах можно легко запутаться.

В данной работе предлагается разработка веб-приложения, которое предназначено для упрощения взаимодействия между тренером и его клиентами. Приложение предоставляет возможность тренеру индивидуально составлять планы тренировок и питания для каждого клиента, отслеживать их физические параметры, активность, питание и получать отчеты о проведенных тренировках. Что позволяет работать с людьми как оффлайн, так и онлайн.

Актуальность разработки такого веб-приложения заключается в следующих аспектах:

* увеличение числа людей, стремящихся к здоровому образу жизни: современное общество все больше осознает важность заботы о своем здоровье и физической форме. Разработка удобного и эффективного инструмента для тренировок может привлечь больше клиентов в тренажерные залы;
* персонализация тренировочных программ: каждый человек уникален, и ему требуется индивидуальный подход к тренировкам. Веб-приложение позволит тренеру создавать персонализированные планы тренировок и питания для каждого клиента, учитывая его цели, физическую подготовку, здоровье и предпочтения;
* улучшение мотивации и результативности: возможность отслеживания прогресса, получения отчетов о тренировках, обратной связи от тренера поможет ученикам быть более мотивированными и дисциплинированными в достижении своих целей;
* удобство и доступность: веб-приложение позволит тренерам и клиентам взаимодействовать независимо от местоположения, что делает процесс тренировок более удобным и гибким.

Таким образом, разработка веб-приложения для тренажерного зала имеет большую актуальность в современном мире, где все больше людей стремятся к здоровому образу жизни и физической активности.

Цель проекта заключается в улучшении эффективности тренировочного процесса, повышении мотивации клиентов к достижению целей и обеспечении более качественного и персонализированного подхода к тренировкам. Разработка веб-приложения способствует более удобному, эффективному и прозрачному взаимодействию между тренером и клиентами.

Для достижения поставленной цели автор ставит следующие задачи:

* исследование и анализ работы с клиентами в организации: автор может провести анализ того, как именно происходит работа тренера фитнес клуба с клиентом и предложить вариант оптимизации этой работы;
* определение целевой аудитории: важно понять, для кого будет предназначено разрабатываемое приложение, какие потребности оно должно удовлетворять и какие преимущества предложить потенциальным пользователям;
* проектирование функционала приложения: автор должен определить основные возможности и инструменты, которые будут доступны пользователям в приложении, включая планы тренировок, отслеживание прогресса, обратную связь с тренером и другие функции;
* разработка дизайна и пользовательского интерфейса: важно создать удобный и интуитивно понятный дизайн приложения, который будет привлекать пользователей и обеспечивать им комфортное взаимодействие с приложением;
* разработка web-приложения для работы с клиентами фитнес клуба;
* тестирование и оптимизация: после разработки приложения необходимо провести тестирование его работоспособности, исправить возможные ошибки и недочеты, а также оптимизировать производительность и удобство использования.

В ходе исследования были использованы инструменты разработки:

* IntelliJ IDEA – это интегрированная среда разработки программного обеспечения для многих языков программирования;
* язык гипертекстовой разметки HTML;
* язык программирования Java;
* фреймворк Spring;
* язык программирования JavaScript;
* язык описания внешнего представления для содержимого, описанного на HTML-страницах;
* система у правления базами данных PostgreSQL;
* сервер Timeweb Cloud;
* операционная система Ubuntu.

Помимо основных программных обеспечений и технологий использовались такие инструменты, как:

* графический редактор для проектирования макетов Figma;
* веб-сервис для хостинга IT-проектов GitHub;
* интерактивная онлайн-доска Miro для создания блок-схем.

Во введении обосновывается актуальность исследуемой темы дипломного проекта, определяется цель, задачи.

В рамках первой главы будет проведен анализ информационных потребностей пользователей, выявление недостатков в текущих системах работы с клиентами, формулирование целей и задач для разработки веб-приложения.

В рамках второй главы исследовательской работы будут решаться поставленные автором задачи, разработка web-приложения для работы с клиентами фитнесс клуба, а именно проектирование функционала приложения, разработка дизайна и пользовательского интерфейса, тестирование и оптимизация приложения.

В заключении подводятся итоги результата разработки, формируются выводы по написанию дипломной работы.

Список использованных источников содержит перечень литературы и Интернет-ресурсов, к которым обращается автор в ходе написания работы.

# 1. АНАЛИТИЧЕСКАЯ ЧАСТЬ

## 1.1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

### 1.1.1 Словарь предметной области

* Тренер – специалист, оказывающий помощь и консультации по занятиям в тренажерном зале. Человек, который составляет план тренировок и план питания для клиента, а также отслеживает его результаты, если нужно, корректируя интенсивность тренировок и количество калорий, белков, жиров и углеводов (КБЖУ) в плане питания.
* Клиент – человек, который пришел заниматься в фитнес-клуб и пользуется услугами персонального тренера;
* КБЖУ – это аббревиатура, которая расшифровывается как калории, белки, жиры и углеводы. Она используется для определения количества питательных веществ, которые человек поглощает в течении суток и которые составляют суточную калорийность его рациона.
* План тренировки, в свою очередь, – это документ, в котором раскрываются направленность, содержание, порядок, последовательность и сроки осуществлении тренировочных и вне тренировочных заданий, связанных с достижением поставленных тренером и спортсменом целей.
* Отчет о тренировке – это список упражнений из данного плана тренировок, к которым прикреплен список, отображающий выполнение каждого упражнения.
* Индивидуальный план питания – это рациональное распределение продуктов и блюд, подсчет нужного количества питательных веществ для достижения поставленной человеком цели.
* Отчет о плане питания – таблица, которая включает в себя количество КБЖУ, которые получил человек в течении одних суток и дату, соответствующих суток. Далее берутся все значения за конкретную неделю и находится среднее каждого параметра. Важно отслеживать именно среднее значение, потому что невозможно питаться одинаково каждый день. И как раз средние показатели тренер сравнивает с планом, который он задал.
* Активность – это количество пройденных человеком шагов за один день. Ее отслеживание важно для понимания тренером, какой образ жизни ведет его клиент. И отталкиваясь от этого, тренер может подстраивать план тренировок или план питания индивидуально под каждого клиента.

### 1.1.2 Анализ системы работы с клиентами объекта исследования

Так как все люди, которые посещают фитнес-клуб, имеют разный уровень знаний о том, как нужно тренироваться и питаться для достижения своей цели, многие из них предпочитают заниматься с тренером, который, в свою очередь, лучше понимает все физиологические процессы, протекающие в организме, и может составить индивидуальный план тренировок и питания для каждого своего клиента.

Проведя анализ работы тренера с клиентом, был сделан вывод, что за частую их взаимодействие происходит в Google-таблицах. На каждого клиента создается файл, который включат в себя отдельные листы с таблицами:

* план режима питания;
* замеры;
* план тренировок;
* рекомендации по питанию.

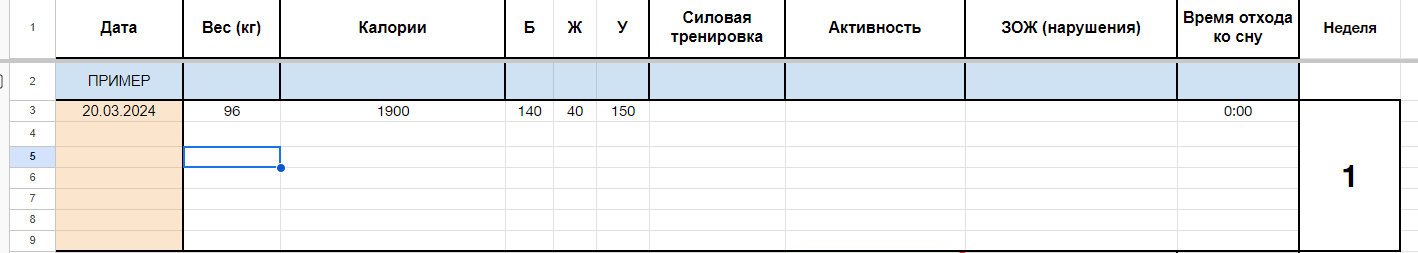


Рисунок 1 – пример плана питания в Google-таблицах[[1]](#footnote-1)

Как видно на рисунке 1, клиенту нужно заполнять каждый день недели свой вес и количество КБЖУ, которые потребляет в день. Также есть колонки, где пользователь может заполнить данные о тренировке, активности, вредные привычки и время отхода ко сну.

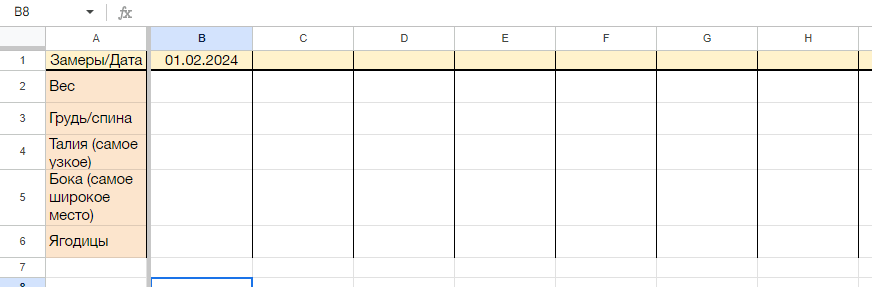


Рисунок 2 – пример листа с замерами в Google-таблицах[[2]](#footnote-2)

На рисунке 2 приведен пример таблицы, в которую клиент должен вносить значения своих физических параметров раз в неделю.

Если проанализировать рисунок 3, можно сделать вывод, что лист сильно перегружен и, используя такой вариант ведения клиента, можно ввести его в заблуждение. Не всё в этом варианте интуитивно для пользователя. Также можно отметить то, Google-таблицы по большей мере рассчитаны на то, что в них будут работать с компьютера. Версия для телефонов есть, но в ней недостаточно удобный интерфейс для таких задач.

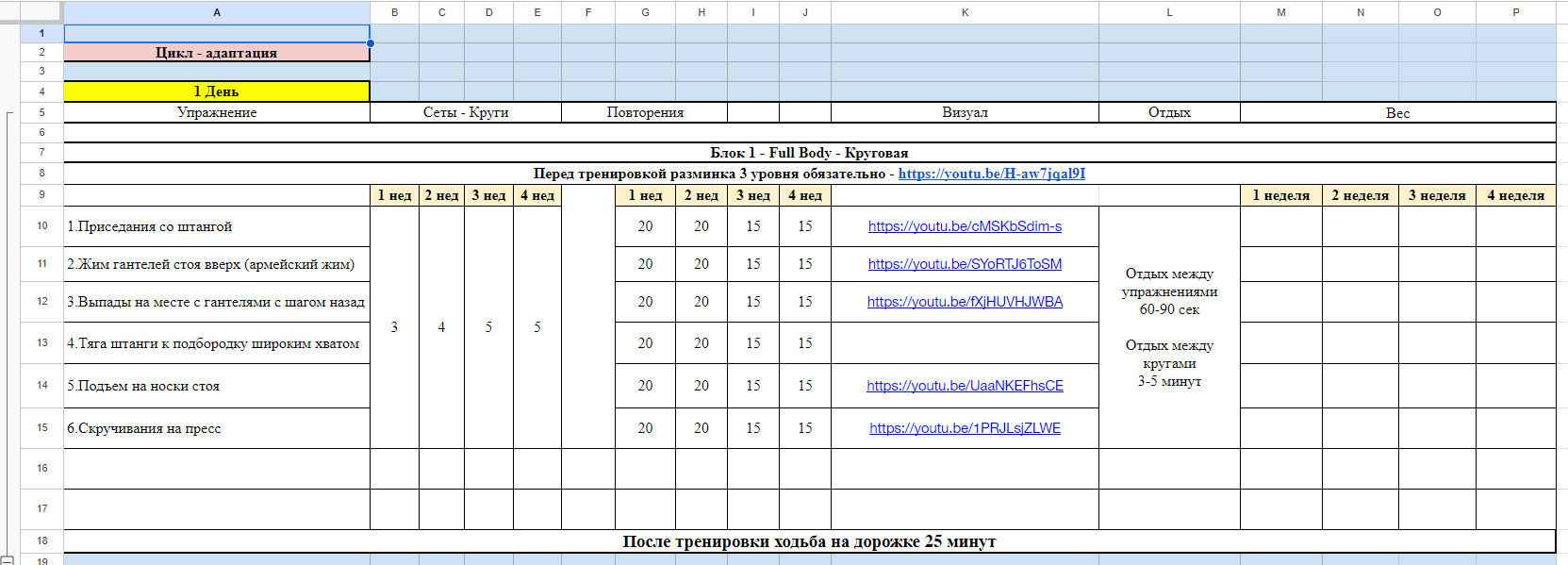


Рисунок 3 – пример листа с планом тренировок в Google-таблицах[[3]](#footnote-3)

Исходя из всего вышеперечисленного, можно сделать вывод, что работа тренера с клиентами фитнес клуба имеет свои недостатки:

* тренеру необходимо создавать отдельные файлы с таблицами для каждого клиента и объяснять, как именно нужно заполнять каждую таблицу:
* нужно тратить много времени для создания таблицы с планом тренировки;
* при заполнении отчетов о тренировке, замерах, питания таблица разрастается в ширь, что делает неудобным анализ информации, которая в ней хранится;
* неадаптивное приложение для телефона, в котором также легко допустить ошибку при внесении данных в таблицу из-за того, что она вся не помещается на экран.

Разработка web-приложения для работы с клиентами фитнес-клуба является неотъемлемой потребностью как фитнес-клуба, так и его тренеров. Это позволит:

* повысить эффективность работы тренера с каждым клиентом;
* вносить и извлекать необходимые данные с любого устройства;
* отслеживать прогресс и достижения клиентов;
* повысить удовлетворенность клиентов;
* привлечь новых клиентов;
* работать с клиентами в онлайн формате.

В конечном итоге, фитнес-клуб сможет достичь более высокого уровня эффективности и удовлетворенности клиентов, а также улучшить свою конкурентоспособность на рынке.

## 1.2 АНАЛИЗ ИСПОЛЬЗУЕМОГО ПРОГРАМНОГО ОБЕСПЕЧЕНИЯ

При разработке web-приложения для фитнес-клуба, автор выделяет языки программирования и инструменты, которыми будет пользоваться в процессе разработки:

* Java [1] – один из наиболее используемых языков программирования. Он активно применяется в разработке корпоративных, веб- и мобильных приложений. Java имеет применение в ведущих технических отраслях, от Data Science [2] до AR/VR. Несмотря на то, что корпоративная разработка на Java, –наиболее популярный вариант использования этого языка, веб- и мобильные приложения также считаются одной из сфер применения Java. Одна из основных причин использовать Java для веба и мобильных устройств состоит в том, что это высокопроизводительный язык программирования. Он гарантирует, что приложения смогут работать быстро и без каких-либо ошибок обработки;
* HTML [3] (от англ. HyperText Markup Language – «язык гипертекстовой разметки») – стандартизированный язык гипертекстовой разметки документов для просмотра веб-страниц в браузере. Веб-браузеры получают HTML документ от сервера по протоколам HTTP/HTTPS или открывают с локального диска, далее интерпретируют код в интерфейс, который будет отображаться на экране монитора;
* PostgreSQL [4], также известная как Postgres, и является бесплатным и открытым исходным кодом система управления реляционными базами данных (РСУБД), подчеркивая, расширяемости и соответствия SQL. PostgreSQL поддерживает транзакции со свойствами атомарности, согласованности, изоляции, долговечности (ACID), автоматически обновляемыми представлениями, материализованными представлениями, триггерами, внешними ключами и хранимыми процедурами. Он поддерживается всеми основными операционными системами, включая Linux, FreeBSD, OpenBSD, macOS и Windows, и обрабатывает широкий спектр рабочих нагрузок - от отдельных компьютеров до хранилищ данных или веб-служб с большим количеством одновременных пользователей;
* Платформа Spring Framework [5] предоставляет комплексную модель программирования и настройки для современных корпоративных приложений на базе Java на любой платформе развертывания. Ключевым элементом Spring является инфраструктурная поддержка на уровне приложений: Spring фокусируется на "настройке" корпоративных приложений, чтобы команды могли сосредоточиться на бизнес-логике на уровне приложений без ненужной привязки к конкретным средам развертывания. Spring Framework представляет собой контейнер внедрения зависимостей, с несколькими удобными слоями (например, доступ к базе данных, прокси, аспектно-ориентированное программирование, RPC, веб-инфраструктура MVC). Это все позволяет вам быстрее и удобнее создавать Java-приложения;
* Apache Tomcat [6] – это контейнер сервлетов с открытым исходным кодом, разрабатываемый Apache Software Foundation. Он реализует спецификации сервлетов, JavaServer Pages (JSP) и JavaServer Faces (JSF) и написан на языке Java. Tomcat позволяет запускать веб-приложения и содержит ряд программ для самоконфигурирования;
* Каскадные таблицы стилей (CSS) [7] – это язык таблиц стилей, используемый для определения представления и стиля документа, написанного на языке разметки, таком как HTML или XML (включая диалекты XML, такие как SVG, MathML или XHTML). CSS является краеугольной технологией Всемирной паутины, наряду с HTML и JavaScript;
* JavaScript [8] – это язык программирования, изначально разработанный для взаимодействия с элементами веб-страниц. JavaScript позволяет изменять содержимое, структуру и стили HTML-элементов на странице. Это позволяет динамически обновлять содержимое страницы без перезагрузки. Также JavaScript используется для проверки данных, введенных пользователем в формы на веб-странице. Это позволяет предотвращать отправку некорректных данных на сервер;
* Timeweb [9] – это российский хостинг, отличающийся стабильным функционированием, безопасностью, быстродействием и множеством предложений для размещений сайтов. Он имеет идеально структурированную панель администрирования и обеспечивает моментальный доступ к интернет-площадке.

При выборе редактора кода автор определенно останавливается на программном обеспечении от компании JetBrains [10] – IntelliJ IDEA. IntelliJ IDEA – это кроссплатформенная среда разработки, созданная компанией JetBrains в 2001 году. «Кроссплатформенная» означает, что ей можно пользоваться на разных операционных системах: Windows, macOS и системах семейства Linux. Изначально в IntelliJ можно было писать код лишь на Java, но со временем разработчики добавили поддержку Kotlin, Python, JavaScript, языков разметки (HTML и XML), фреймворков Spring Boot [11], Hibernate [12] и JUnit. Помимо этого, за 20 лет IDE обросла внушительным набором плагинов и инструментов, упрощающих написание кода.

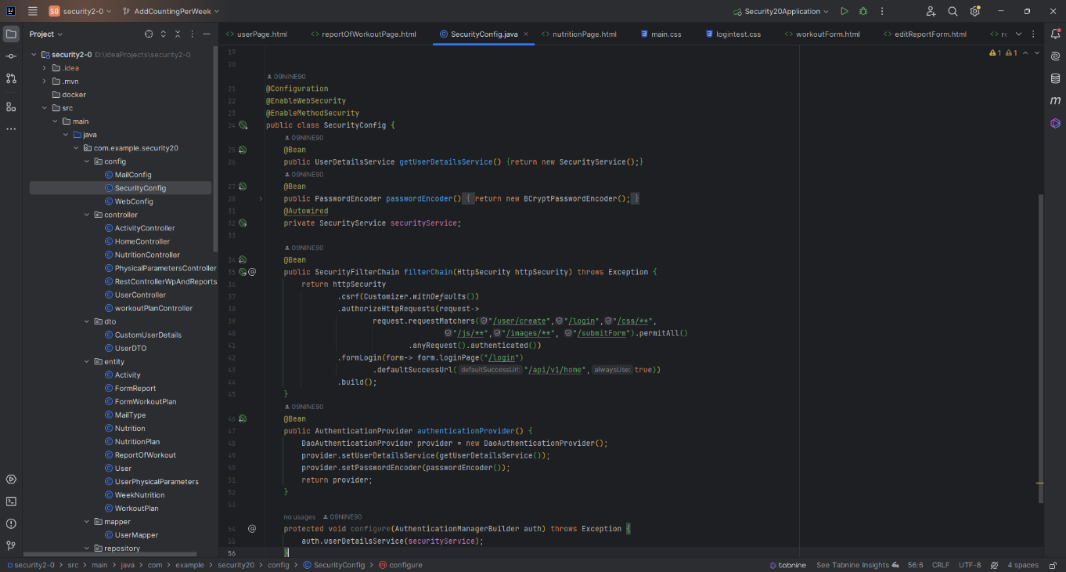


Рисунок 4 – Интерфейс среды разработки IntelliJ IDEA[[4]](#footnote-4)

На рисунке 4 отображен режим подсветки в стиле «Тёмная тема», что позволяет разработчику создать ночную атмосферу и освободить глаза от усталости.

Также для разработки приложения необходимы зависимости(dependency), которые располагаются в POM файле.

POM [13] (Project Object Model) является базовым модулем Maven. Это специальный XML-файл, который всегда хранится в базовой директории проекта и называется pom.xml. Файл POM содержит информацию о проекте и различных деталях конфигурации, которые используются Maven для создания проекта.

Maven [14] — инструмент для управления и сборки проектов. Он помогает разработчикам правильно подключить библиотеки и фреймворки, управлять их версиями, выстроить структуру проекта и составить к нему документацию. Maven — не единственный сборщик проектов. Некоторые разработчики используют его аналоги — Gradle и Ant. Но именно Maven сегодня — золотой стандарт в индустрии.

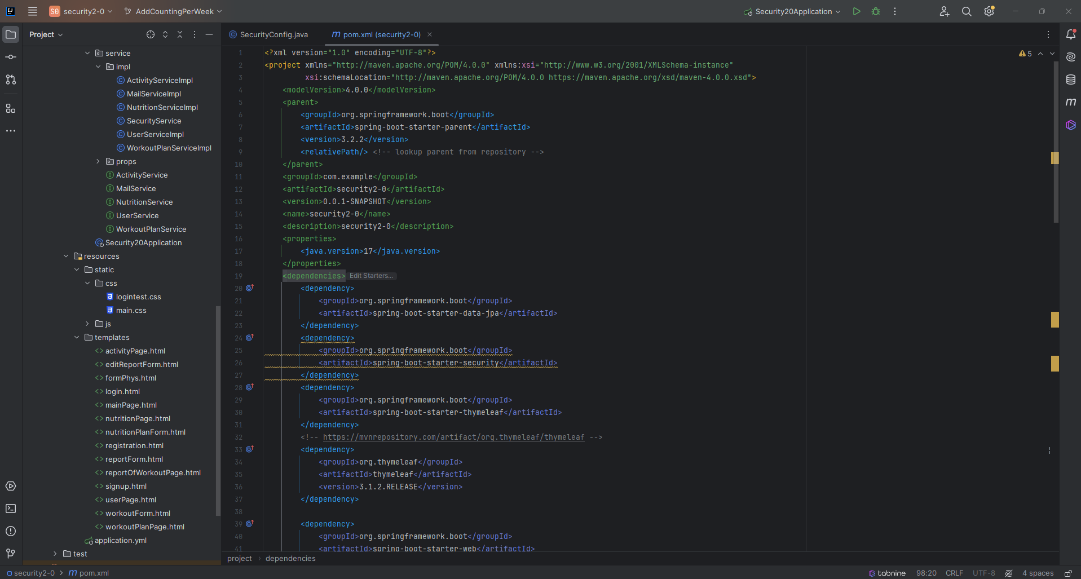


Рисунок 5 – pom.xml[[5]](#footnote-5)

Зависимости (Dependency) [15] — это сторонние библиотеки и модули, которые влияют на работоспособность программы. Maven обеспечивает автоматическое управление зависимостями: он загружает модули и генерирует их описание в объектной модели проекта.

# 2. ПРОЕКТНАЯ ЧАСТЬ

## 2.1 ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Основное требование – разработка web-приложения для работы тренера с клиентами. В приложение должен быть реализован функционал со стороны тренера:

* добавление и удаление пользователей;
* доступ ко всем данным, которые предоставляют клиенты (ФИО, номер телефона, почта, возраст, вес, замеры);
* составление планов тренировок и планов питания индивидуально для каждого пользователя;
* возможность редактирования планов тренировок и питания.

Со стороны клиента:

* возможность вносить данные о своих физических параметрах;
* создание отчетов о тренировках и заполнение дневника питания;
* возможность интуитивно ориентироваться в приложении (понятный и отзывчивый интерфейс).

Общие требования к дизайну:

* дизайн должен быть современным, привлекательным и соответствовать бренду компании;
* цветовая палитра должна быть гармоничной и соответствовать корпоративным цветам;
* дизайн должен быть адаптивным для различных устройств (мобильные устройства, планшеты, десктопы);
* использование анимации и интерактивных элементов для улучшения пользовательского опыта.

## 2.2 РАЗРАБОТКА ПРИЛОЖЕНИЯ

### 2.2.1 Моделирование схемы данных

В начале разработки нужно спроектировать схему базы данных для того, чтобы иметь представление о том, как будет храниться информация о пользователях в исследуемом приложении.

База данных [16] – это организованный набор структурированной информации или данных, обычно хранящихся в электронном виде в компьютерной системе. База данных обычно управляется системой управления базами данных (СУБД). Вместе данные и СУБД, а также связанные с ними приложения называются системой баз данных, часто сокращаемой до просто базы данных.

Данные в базах данных наиболее распространенных типов, используемых в настоящее время, обычно моделируются в виде строк и столбцов в виде ряда таблиц, что повышает эффективность обработки и запроса данных. Затем к данным можно легко получить доступ, управлять ими, изменять, обновлять, контролировать и упорядочивать. Большинство баз данных используют язык структурированных запросов (SQL) [17] для записи и запроса данных.

В исследуемом проекте будет схема данных, изображенная на рисунке 6. Подробнее о каждой таблице:

* security\_user – таблица, в которой будет храниться вся основная информация о каждом пользователе (id, имя, фамилия, номер телефона, email, роль, пароль от учетной записи и id тренера, у которого занимается клиент);
* user\_phys\_parameters – таблица, в которой хранятся физические параметры пользователя (возраст, вес, обхваты руки, ноги, грудной клетки, бедер, талии, также дата замеров и id пользователя, которому принадлежат данные);
* workout\_plan – таблица, которая содержит данные о тренировочном плане для конкретного пользователя (строка с упражнениями, строка с количеством подходов и повторений, id пользователя, которому принадлежат данные);
* report\_of\_workout – таблица, в которой содержатся данные об отчете пользователя о тренировке (строка с упражнениями, строка с выполненными упражнениями, дата тренировки, id пользователя, которому принадлежат данные);
* nutrition – это таблица, которая хранит в себе питательные вещества, потребляемы пользователем за одни (калории, белки, жиры, углеводы, дата, id пользователя, которому принадлежат данные);
* week\_nutrition – таблица, в которой считаются средние значения количества потребляемых питательных веществ в день в течение недели (начало недели(дата), конец недели(дата), последний заполненный день(дата), сумма КБЖУ за все заполненные дни недели, количество заполненных дней, id пользователя, которому принадлежат данные).

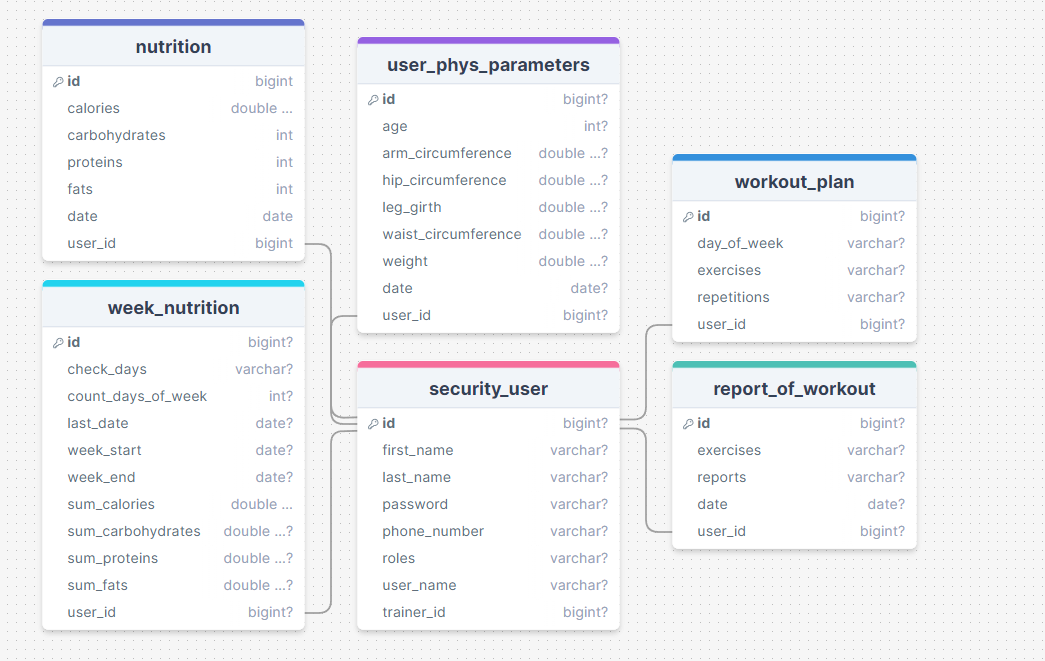


Рисунок 6 – схема базы данных приложения[[6]](#footnote-6)

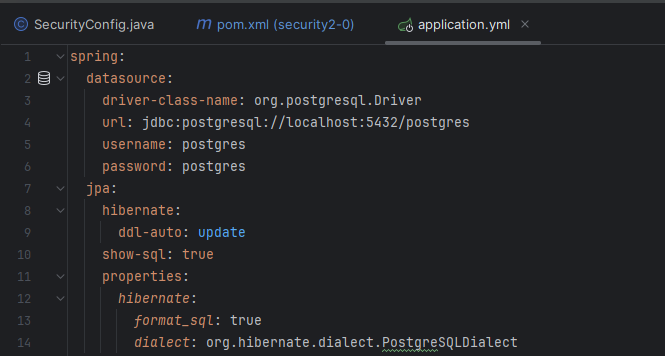


Рисунок 7 – подключение к базе данных[[7]](#footnote-7)

В качестве системы управления данными выступит PostgreSQL. Подключение к базе данных прописано в свойствах приложения – файле application.yml рисунок 7.

### 2.2.2 Структура исследуемого проекта

Фреймворк Spring MVC [18] обеспечивает архитектуру паттерна Model –View – Controller (Модель – Отображение (далее – Вид) – Контроллер) при помощи слабо связанных готовых компонентов. Паттерн MVC разделяет аспекты приложения (логику ввода, бизнес-логику и логику UI), обеспечивая при этом свободную связь между ними (Рисунок 8).

* Model (Модель) инкапсулирует (объединяет) данные приложения, в целом они будут состоять из POJO («Старых добрых Java-объектов», или бинов);
* View (Отображение, Вид) отвечает за отображение данных Модели, – как правило, генерируя HTML, которые мы видим в своём браузере;
* Controller (Контроллер) обрабатывает запрос пользователя, создаёт соответствующую Модель и передаёт её для отображения в Вид.

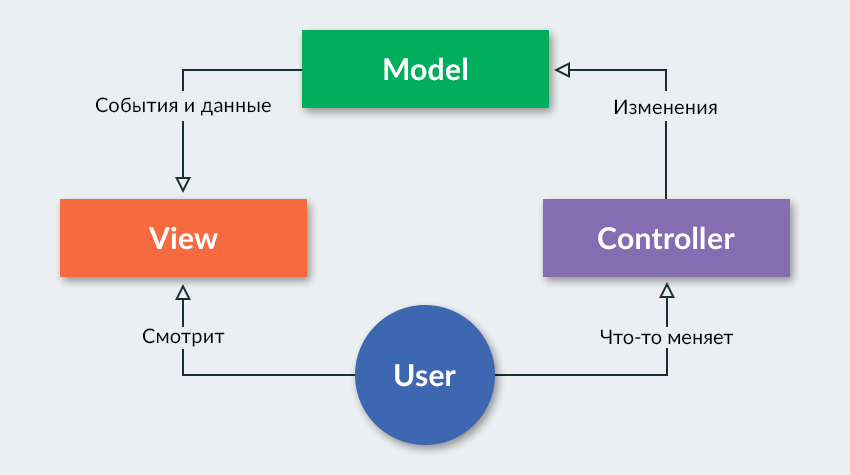


Рисунок 8 – архитектура Spring MVC[[8]](#footnote-8)

Для того, чтобы программа функционировала, зависимости внедрялись, контроллеры поддерживались, определялись конечные точки приложения(endpoints) используется Spring Boot. Spring Boot является расширением Spring Framework, которое построено поверх сервлета. Spring Framework в основном используется для создания слабосвязанных, надежных и автономных корпоративных приложений на базе Java. Spring Boot предлагает несколько универсальных функций, таких как автоконфигурация, настраиваемые значения по умолчанию, встроенный сервер, обширная поддержка аннотаций и т.д.

Аннотации Spring Boot [19] являются наиболее распространенной функциональностью, которая эффективно используется для создания и разработки программной системы. Эти аннотации содержат ряд функций, таких как внедрение зависимостей, сканирование компонентов, поддержка контроллера, определение конечных точек REST и др.

Основные аннотации и их определения:

* @SpringBootApplication – эта аннотация используется для начальной загрузки приложения Spring Boot и запуска его с помощью метода Java main. Она инициализирует контекст приложения из classpath и запускает приложение. Это позволяет разработчику избежать напряженной работы, связанной с настройкой, за счет автоматической настройки, сканирования компонентов и т.д;
* @Service. Компоненты, помеченные данной, аннотации содержат бизнес-логику. Помимо использования на уровне сервиса, для этой аннотации нет другого специального применения;
* @Configuration – это аннотация Spring Core, которая, будучи аннотированной поверх класса, означает, что этот класс содержит определения для различных Java-компонентов, аннотированные с помощью @Bean которые будут управляться с помощью Контейнера Spring IoC. Эта аннотация также содержит семантику параметра @Component, который сообщает spring Framework, что этот класс должен быть обнаружен при сканировании компонентов;
* @Bean – Это самая простая аннотация, которая помещается поверх метода, сигнализирующего Spring Framework, что этот метод содержит определение Java Bean и вернет Java Bean, который должен управляться контейнером Spring IoC;
* @Autowired – единственная цель этой аннотации – упростить работу с основной функцией Spring Framework - «Внедрение зависимостей». Внедрение зависимостей — это процедура присоединения одного объекта к другому для обмена данными и доступа к функциональности друг друга;
* @Controller – это аннотация на основе класса, в основном она помечает класс как «Контроллер». Контроллер – это важнейший компонент Spring Framework, который знает, как обрабатывать входящие запросы и предоставлять индивидуальный ответ на каждый запрос. Эта аннотация используется с spring MVC вместе с @RequestMapping;
* @RestController – эта аннотация используется для создания веб-сервисов RESTful и представляет собой специализированную версию @Controller, о которой написано выше. Это также аннотация уровня класса, которая имеет дело с различными REST API – GET, PUT, POST, DELETE и т.д. Считается, что класс определяет различные конечные точки REST, если он помечен @RestController. Он состоит из двух вложенных аннотаций – @Controller и @ResponseBod;
* @RequestMapping – эту аннотацию можно использовать как на уровне класса, так и на уровне метода для определения пути URL, который является конечными точками. Основная аннотация определяет путь, который вернет некоторый ответ, содержащий ресурс, запрошенный клиентом. Хотя это предпочтительнее только на уровне класса, потому что предполагается, что методы выполняют одну-единственную задачу и ее следует аннотировать чем-то более содержательным, поэтому часто используются @GetMapping, @PostMapping для таких целей;
* @PathVariable – это аннотация уровня метода, которая позволяет быть более конкретными в зависимости от контекста и извлекать более конкретный ресурс. Например, пользователю могут понадобиться данные клиента с id = 5. Затем можно указать то же самое в URL-адресе при попытке извлечения ресурса. @PathVariable используется для передачи и извлечения параметров из конечной точки в параметры метода, а затем выполнения операций над ним.

В исследуемом проекте присутствуют контроллеры для изменения информации, которую увидит пользователь:

* ActivityController – обработка данных со страницы активности пользователя;
* HomeController – обработка данных главной страницы;
* NutritionController – обработка данных со страницы, на которой отображается план питания пользователя;
* workoutPlanController – обработка данных со страницы на которой отображается план тренировок;
* PhysicalParametersController – отвечает за обработку данных о физических параметрах пользователя;
* UserController – аутенфикация и регистрация пользователей;
* RestControllerWpAndReports – отвечает за обработку данных из форм в которых составляется план тренировок, создается и редактируется отчет о тренировках.

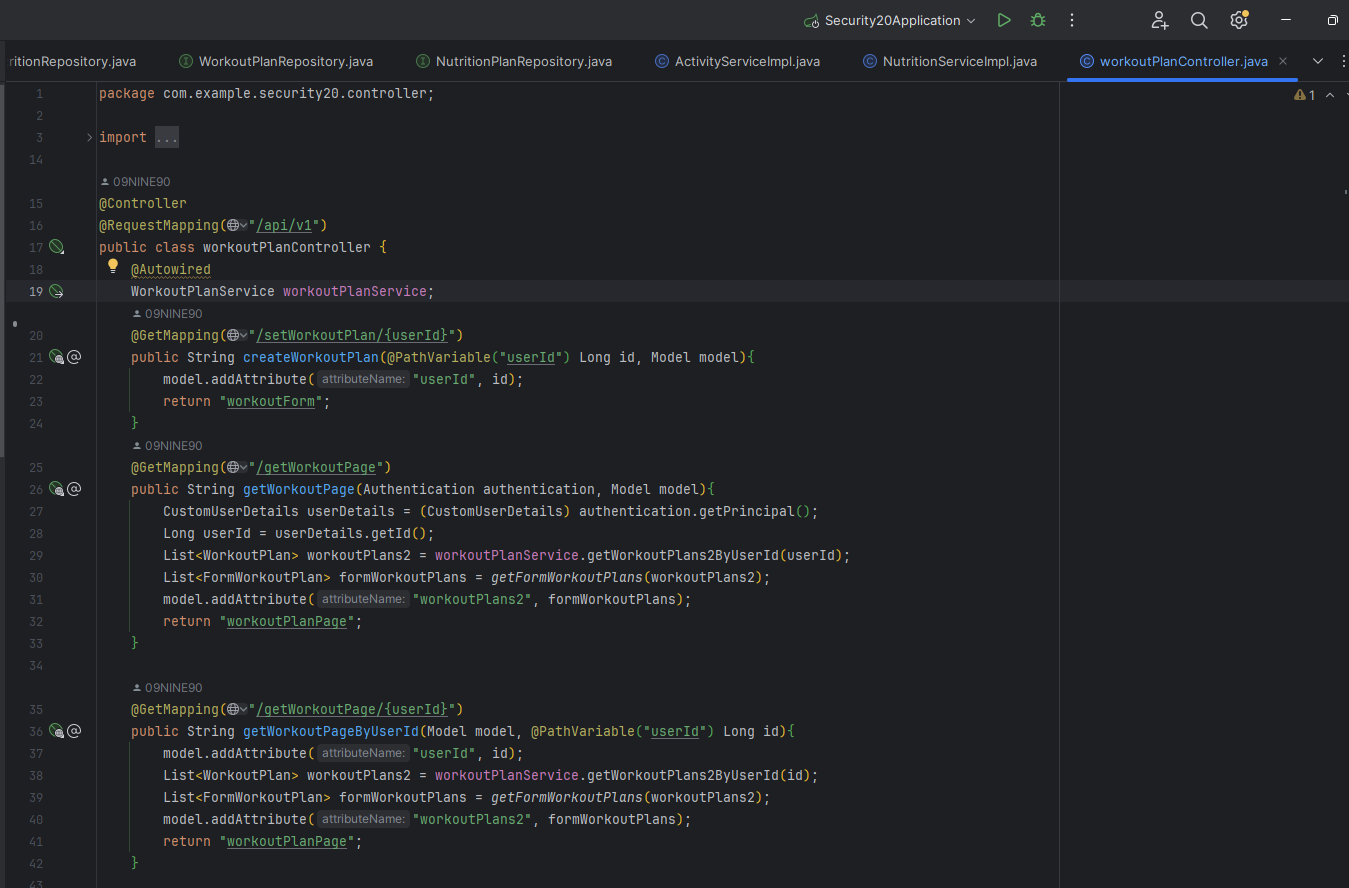


Рисунок 9 – пример контроллера из исследуемого проекта[[9]](#footnote-9)

Для работы с базой данных в проекте используется Spring Data [20]. Spring Data – дополнительный удобный механизм для взаимодействия с сущностями базы данных, организации их в репозитории, извлечение данных, изменение, в каких-то случаях для этого будет достаточно объявить интерфейс и метод в нем, без имплементации.

Основное понятие в Spring Data — это репозиторий. Это несколько интерфейсов которые используют JPA Entity для взаимодействия с ней. Так, например, интерфейс:

Листинг 1 – Пример интерфейса репозитория

public interface CrudRepository<T, ID extends Serializable> extends Repository<T, ID>

Он обеспечивает основные операции по поиску, сохранения, удалению данных (CRUD операции).

Листинг 2 – Пример интерфейса репозитория из исследуемого проекта

@Repository  
public interface NutritionRepository extends JpaRepository<Nutrition,Long> {

//запрос к базе данных  
 @Query(value = "SELECT \* FROM nutrition n WHERE user\_id = :userId ORDER BY n.date DESC", nativeQuery = true)  
 List<Nutrition> getNutritionByUserId(@Param("userId") Long userId);  
}

На листинге 2 можно увидеть пример репозитория, который используется в разработке дипломного проекта. Он имеет один метод, помеченный аннотацией @Query, в которой прописан запрос к базе данных. Запрос можно объяснить, как: выбрать все из таблицы nutrition, где id пользователя совпадает с id, который передали в параметры метода getNutritionByUserId, он указан в аннотации @Param("userId"). В конце запроса прописана сортировка всех полученных данных по дате от большей к меньшей.

Полученные данные из базы данных нужно где-то хранить и каким-то образом структурировать. Для это в проекте созданы сущности объектов (entity). Класс-сущность это обыкновенный POJO класс, с приватными полями и геттерами и сеттерами для них. У него обязательно должен быть не приватный конструктор без параметров (или конструктор по умолчанию), и он должен иметь первичный ключ, т.е. то, что будет однозначно идентифицировать каждую запись этого класса в БД.

Листинг 3 – Пример класса-сущности

@Entity

@AllArgsConstructor

@NoArgsConstructor

@Data

@Table(name = "report\_of\_workout")

public class ReportOfWorkout {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private Long userId;

private String exercises;

private String reports;

@Column(columnDefinition = "TIMESTAMP")

@DateTimeFormat(pattern = "yyyy-MM-dd")

private Date date;

}

В листинге 3 приведен пример сущности, которая привязана к таблице report\_of\_workout, это прописано в аннотации @Table. Также в этом примере присутствуют аннотации @Entity, которая указывает на то, что класс является сущностью, @Data, которая автоматически создает геттеры и сеттеры для всех полей класса, @AllArgsConstructor и @NoArgsConstructor, которые создают конструкторы со всеми полями класса и без всего соответственно. Строка @GeneratedValue(strategy = GenerationType.IDENTITY) также является аннотацией, которая указывает на то, что, при добавлении новой записи в таблицу, id будет генерироваться автоматически. Аннотация @DateTimeFormat(pattern = "yyyy-MM-dd") указывает на то, каким образом хранить дату в базе данных.

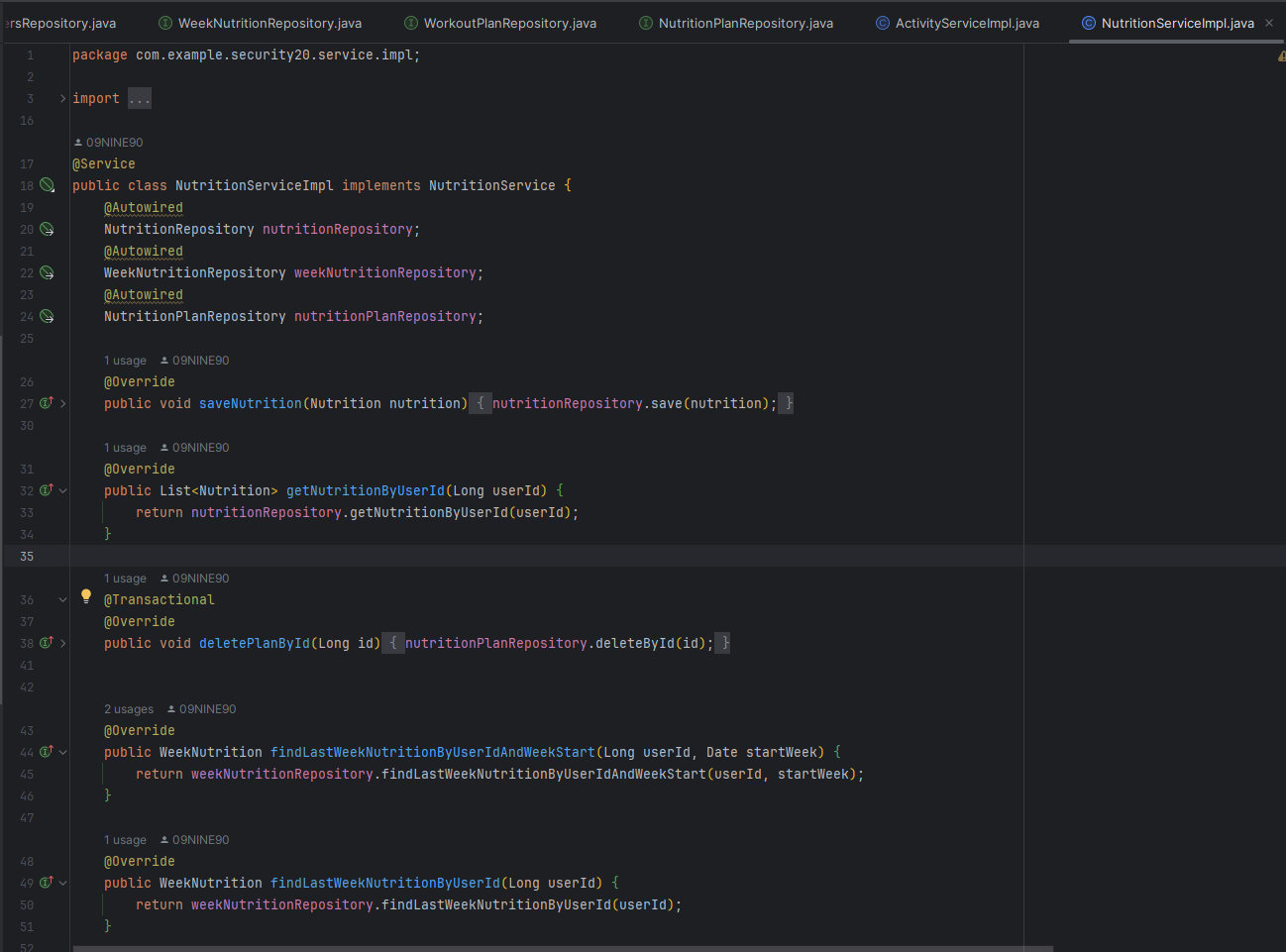


Рисунок 10 – пример сервиса из исследуемого проекта[[10]](#footnote-10)

Для того, чтобы контроллеры могли получать и отправлять данные из базы данных используются сервисы (классы с аннотацией @Service). В них приписывается вся логика взаимодействия с базой данным с помощью репозиориев. Пример сервиса приведен на рисунке 10.

### 2.2.3 Идентификация, аутентификация, авторизация и регистрация пользователей

В исследуемом проекте у каждого пользователя приложения будет свой аккаунт. Для того, чтобы получить доступ к своей странице, пользователю нужно пройти идентификацию, аутентификацию и авторизацию [21] (Рисунок 11). Для реализации аутентификации пользователя и используется Spring Security [22]. Spring Security – это фреймворк обеспечения безопасности, предоставляющий возможность декларативного управления безопасностью приложений на основе фреймворка Spring. Добавим класс SecurityConfig. Также укажем аннотации @Configuration (это означает, что данный класс является конфигурационным) и @EnableWebSecurity (это означает, что данный класс является содержит настройки для защиты веб-приложения).

В листинге 4 показана настройка безопасности веб-приложения. Настраивается csrf токен [23] – уникальное, секретное и непредсказуемое значение, генерируемое приложением на стороне сервера и передающееся клиенту для дальнейшей обработки запросов пользователя. Также настраивается доступ к страницам приложения. Указывается форма для авторизации в приложении. Если ее не указывать, будет использоваться форма по умолчанию. После этого указан путь до страницы, на которую попадет пользователь после успешной авторизации. Код класса SecurityConfig находится в приложении А.

Листинг 4 – Метод SecurityFilterChain класса SecurityConfig

@Bean

public SecurityFilterChain filterChain(HttpSecurity httpSecurity) throws Exception {

return httpSecurity

//настрока csrf токена

.csrf(Customizer.withDefaults())

//настройка доступа к страницам приложения

.authorizeHttpRequests(request->

request.requestMatchers("/user/create","/login","/css/\*\*",

"/js/\*\*","/images/\*\*", "/submitForm").permitAll()

.anyRequest().authenticated())

//перенаправление на форму входа и указание страницы, на которую пользователь попадет при успешной аутентификации

.formLogin(form-> form.loginPage("/login")

.defaultSuccessUrl("/api/v1/home",true))

.build();

}

Подробнее про каждый этап:

* цель идентификации – понять, существует ли пользователь с таким логином в нашей базе данных. В качестве логина в приложении выступает email;
* цель аутентификации – проверить, является ли тот, кто использует данный логин, обладателем пароля, который закреплен за этим логином;
* завершающий этап – авторизация. Это наделение пользователя определенными правами с определенным набором возможностей. Далее система показывает пользователю ту информацию, к которой у него есть доступ.

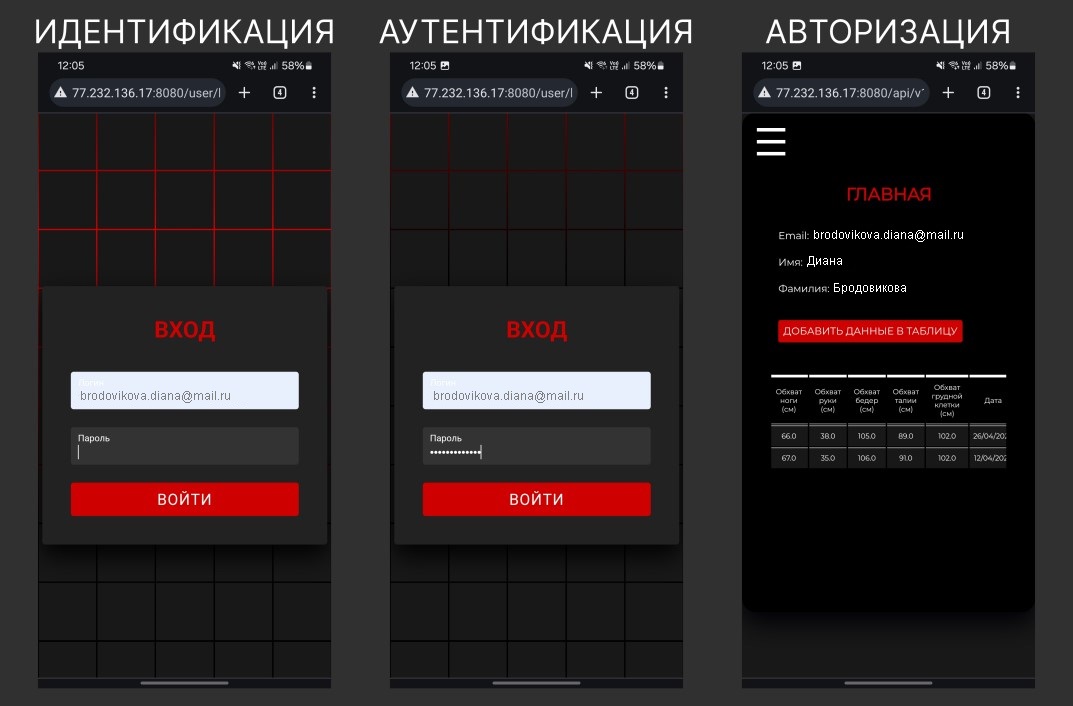


Рисунок 11 – идентификация, аутентификация и авторизация в дипломной работе[[11]](#footnote-11)

В данной работе каждый пользователь имеет роль. Всего роли три:

* администратор (ROLE\_ADMIN) – пользователь, которой регистрирует тренеров в системе, при этом тренеру автоматически присваивается его роль;
* тренер (ROLE\_COACH) – пользователь, которой регистрирует клиентов в системе, при этом клиенту автоматически присваивается его роль. Также тренер имеет доступ ко всем данным каждого клиента, которого он зарегистрировал, он может задавать планы тренировок и питания, проверять отчеты пользователя;
* клиент (ROLE\_USER) – пользователь, который имеет доступ к планам тренировок и питания, составленным для него. Также, имеет возможность создавать и редактировать отчеты о питании и тренировках.

Сам клиент зарегистрироваться не может, только со стороны тренера. После того, как тренер регистрирует нового клиента, клиенту на указанную почту приходит письмо с его логином, паролем и ссылкой для входа.

Листинг 5 – Метод для отправки сообщений на почту

@SneakyThrows

private void sendRegistrationEmail(User user, Properties properties){

//создание объекта сообщения

MimeMessage mimeMessage = javaMailSender.createMimeMessage();

MimeMessageHelper helper = new MimeMessageHelper(mimeMessage, false, "UTF-8");

helper.setSubject("Аккаунт зарегистрирован! " + user.getFirstName());

helper.setTo(user.getUserName());

String emailContent = getRegistrationEmailContent(user);

helper.setText(emailContent, true);

javaMailSender.send(mimeMessage);

}

В листинге 5 представлена реализация метода по формированию и отправке сообщения пользователю. В этот метод передается объект пользователя, которому нужно отправить сообщение. Далее с помощью классов MimeMessage и MimeMessageHelper сообщение формируется и отправляется пользователю на почту, которая содержится в параметрах объекта user. Код всего сервиса по отправке сообщений представлен в приложении Б.

### 2.2.4 Главная страница приложения

После авторизации пользователь получает доступ к функционалу приложения, который соответствует его роли.

У каждого тега на HTML странице есть параметр, который проверяет роль пользователя, чтобы скрывать кнопки, таблицы, навигацию в меню или текст от пользователя, который не должен иметь доступ к этой информации.

Листинг 6 – Пример ограничения доступа к элементам на HTML странице

<div th:if="${#authorization.expression('hasRole(''ROLE\_COACH'')')}" class="menu"></div>

В листинге 6 приведен пример, по которому можно понять, что к элементу div с классом menu есть доступ только у пользователя с ролью ROLE\_COACH, то есть только у тренера.

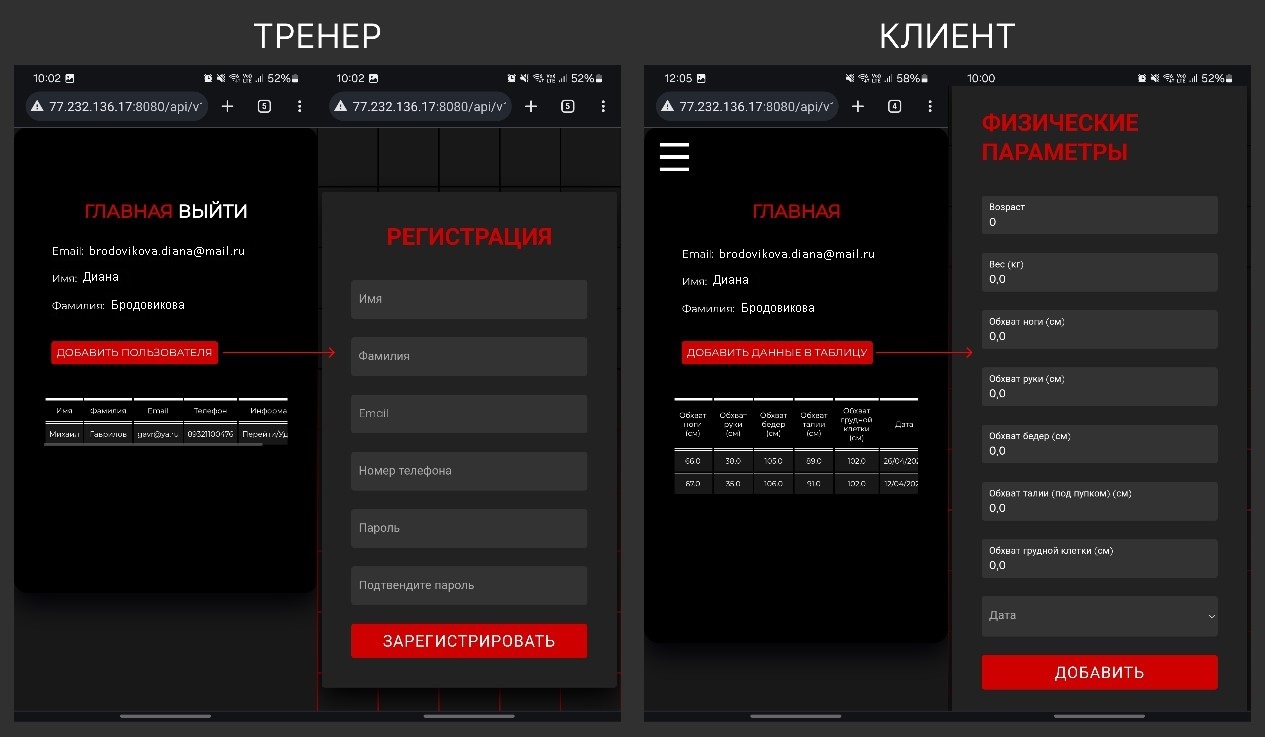


Рисунок 12 – главная страница приложения (мобильная версия)[[12]](#footnote-12)

В приложении у каждого пользователя должна быть главная страница, на которой будут отображаться его контактные данные. Также дополнительно на это странице у тренера будет список всех его клиентов в виде таблицы и кнопка «добавить пользователя», которая должна переводить на форму для заполнения данных нового клиента. У клиентов на главной странице их физические параметры, тоже в виде таблицы и кнопка для их добавления (Рисунок 12).

При отправке обоих форм, данные сохраняются в базе данных, после чего система возвращает пользователей на главную страницу, где отображаются уже обновленные данные.

За отображение информации на страницах и обработку данных отвечают следующие классы программы:

* UserController – класс, помеченный аннотациями @Controller и @RequestMapping("/user"). Обработка и отображение данных из формы регистрации пользователя;
* PhysicalParametersController – класс помеченный аннотациями @Controller и @RequestMapping("/api/v1"). Отвечает за отображение и обновление данных на главной странице и форме заполнения физических параметров;
* UserServiceImpl – класс, реализующий интерфейс UserService, является посредником между контроллером и репозиторием. Отвечает за добавление информации о новых пользователях в базу данных, обращаясь к репозиторию;
* UserPhysicalParametersRepository – интерфейс, который отвечает за взаимодействие с базой данных, а именно с таблицей user\_phys\_parameters;
* UserRepository – интерфейс, который отвечает за взаимодействие с базой данных, а именно с таблицей security\_user;
* User – класс-сущность, который представляет данные из таблицы security\_user в виде объекта;
* UserPhysicalParameters – класс-сущность, который представляет данные из таблицы user\_phys\_parameters в виде объекта.

### 2.2.5 Реализация работы с тренировочным планом и отчетами по тренировкам

Основная цель при разработке системы работы с тренировочными планами и отчётами — создать удобный и интуитивно понятный интерфейс.

Чтобы достичь этой цели, тренировочные планы и отчёты были разделены на разные страницы. Это позволяет пользователям не путаться и легко находить нужную информацию.

При переходе на страницу пользователя, тренер открывает вкладку с планом тренировок, где отображается кнопка для добавления тренировки, кнопка для перехода на страницу с отчетами о тренировках выбранного пользователя и все созданные им тренировки в виде таблицы.

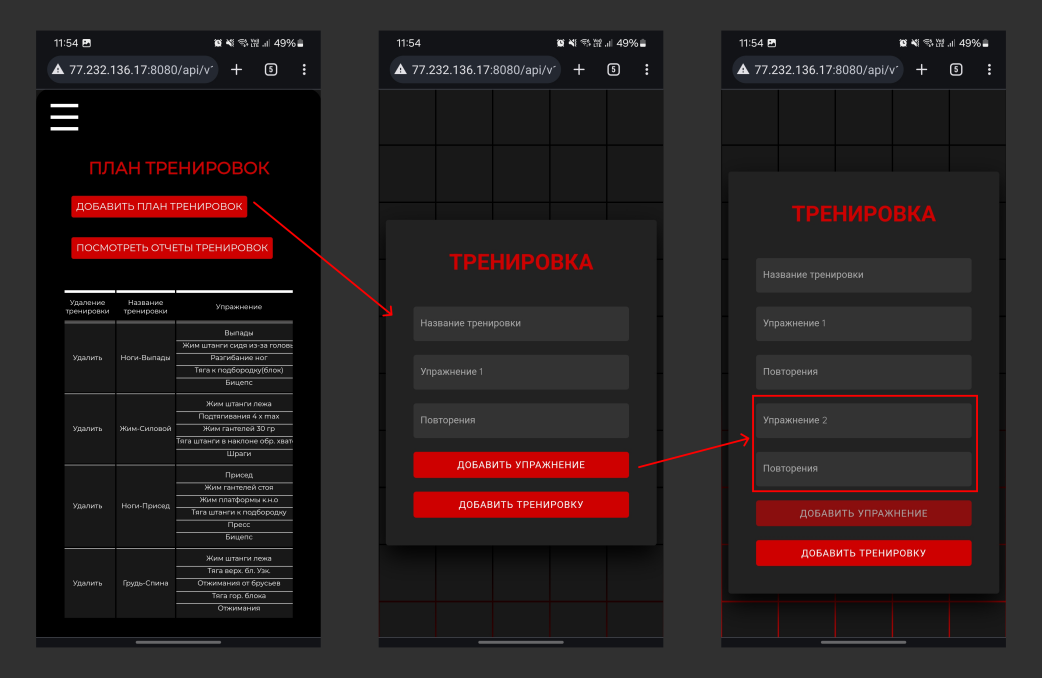


Рисунок 13 – страница с планом тренировок[[13]](#footnote-13)

На форме для создания тренировки есть поля, в которых тренер может указать название тренировки, затем добавлять упражнения и соответствующее им количество подходов и повторений. Изначально есть возможность добавления только одного упражнения. Для того, чтобы добавить поля для других упражнения нужно нажать на кнопку «добавить упражнения». После чего добавится еще два поля для названия упражнения и количества повторений. Так тренер может создавать план тренировок с любым количеством упражнений (Рисунок 13).

Реализация добавления полня ввода упражнения на форму прописана на языке JavaScript в файле workoutForm. При нажатии на кнопку, скрипт создает объекты полей ввода (Листинг 7), настраивает их, затем добавляет в форму. Код всего скрипта, отвечающего за обработку нажатия кнопки добавления упражнения расположен в приложении В.

Листинг 7 – Создание поля ввода на JavaScript

const newInput1 = document.createElement('input');

//настройка типа вводимых данных  
newInput1.type = 'text';

//указание имени поля

newInput1.name = "exercises" + counter;

//указание id поля  
newInput1.id = "exercises" + counter; newInput1.required = true;

//добавление в div  
newFormGroup.appendChild(newInput1);

//добавление в форму

form.insertBefore(newFormGroup, addInputBtn);

Реализация отправки данных из формы также сделана с помощью JavaScript. После нажатия на кнопку «добавить тренировку» значения из всех полей ввода собираются и формируются в JSON файл [24] (Приложение Г). После чего отправляется POST запрос [25] на RestController [26], где данные обрабатываются и записываются в базу данных (Листинг 8). В заголовке передается csrf токен, чтобы запрос корректно сработал и не выдал ошибку.

Листинг 8 – Обработка данных с формы

@PostMapping("/submitForm")  
public String processDataWorkoutPlan(@RequestBody FormWorkoutPlan data) {  
 // Обработка данных  
 WorkoutPlan workoutPlan = new WorkoutPlan();  
 workoutPlan.setDayOfWeek(data.getDayOfWeek());  
 workoutPlan.setUserId(data.getUserId());  
 workoutPlan.setExercises(Arrays.toString(data.getExercises())

.replace("[", "").replace("]", ""));  
workoutPlan.setRepetitions(Arrays.toString(data.getRepetitions()).replace("[", "").replace("]", ""));

//сохранение в базу данных  
 workoutPlanService.saveWorkoutPlan2(workoutPlan);  
 return "redirect:/api/v1/getWorkoutPage/" + data.getUserId();  
}

Со стороны пользователя эта страница выглядит также, но на ней отображается только таблица с планом тренировки. Также в этой таблице есть возможность сделать отчет по каждой тренировке отдельно. Для этого нужно нажать на ссылку «Сделать отчет» у нужной тренировки. После этого откроется форма, в которой можно указать объем выполненной работы (Рисунок 18).

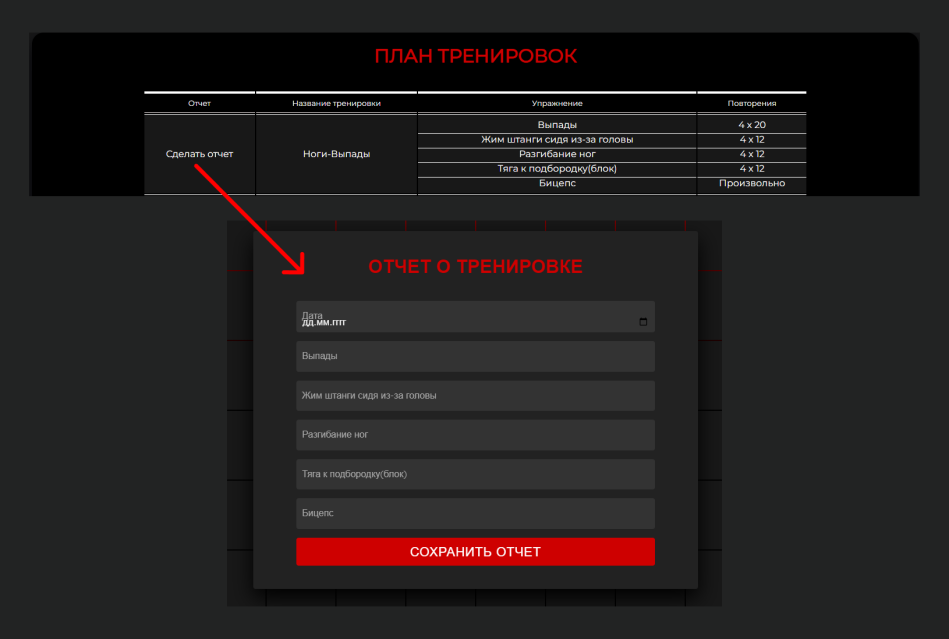


Рисунок 14 – форма для отчетов по тренировке[[14]](#footnote-14)

В форме, изображенной на рисунке 14 необязательно заполнять все поля сразу, главное указать дату тренировки. После нажатия на кнопку «сохранить отчет» вся информация сохраняется в базу данных. Незаполненные поля с упражнениями помечаются как «не сделано», это означает, что эти упражнения пока не выполнены. После этого пользователя переводит на страницу, где отображаются все его отчеты о тренировках. На это странице у пользователя есть возможность отредактировать любой отчет. Например, исправить ошибки, допущенные при заполнении отчета или добавить информацию о тех упражнениях, которые не были заполнены.

За сбор информации с формы для формирования отчета о тренировке, а также за передачу данных на нее тоже отвечает JavaScript. После нажатия кнопки сохранения отчета, данные собираются в JSON файл. После чего отправляется POST запрос на RestController, где данные обрабатываются и записываются в базу данных (Приложение Д).

За отображение информации и обработку данных на страницах с планами тренировок и отчетов о тренировках отвечают следующие классы программы:

* WorkoutPlanController – класс, помеченный аннотациями @Controller и @RequestMapping("/user"). Отвечает за отображение информации на странице с планом тренировок;
* RestControllerWpAndReports – класс, помеченный аннотацией @RestController. Отвечает за отображение и сбор информации с форм для создания плана тренировок, создания отчета о тренировке, редактирование отчета о тренировке;
* WorkoutPlanServiceImpl – класс, реализующий интерфейс WorkoutPlanService, является посредником между контроллером и репозиторием. Отвечает за сохранение информации о планах тренировок и отчетов о тренировках, обращаясь к репозиторию;
* WorkoutPlanRepository – интерфейс, который отвечает за взаимодействие с базой данных, а именно с таблицей workout\_plan;
* ReportOfWorkoutRepository – интерфейс, который отвечает за взаимодействие с базой данных, а именно с таблицей report\_of\_workout;
* WorkoutPlan – класс-сущность, который представляет данные из таблицы workout\_plan в виде объекта;
* ReportOfWorkout – класс-сущность, который представляет данные из таблицы report\_of\_workout в виде объекта.

Таким образом, тренер получает возможность создавать тренировки через форму, в которой можно изменять количество упражнений, при этом задавать план тренировок индивидуально для каждого клиента. В то же время клиент может четко разделять план тренировок и отчеты по ним, которые он делает. При создании и редактировании отчета в каждом поле для заполнения формы прописано название упражнения, как показано на рисунке 14.

### 2.2.6 Реализация работы с планом питания

Работа с планом питания должна быть также реализована таким образом, чтобы клиенту и тренеру нужно было произвести минимальное количество действий для заполнения и проверки плана питания и отчетов по нему.

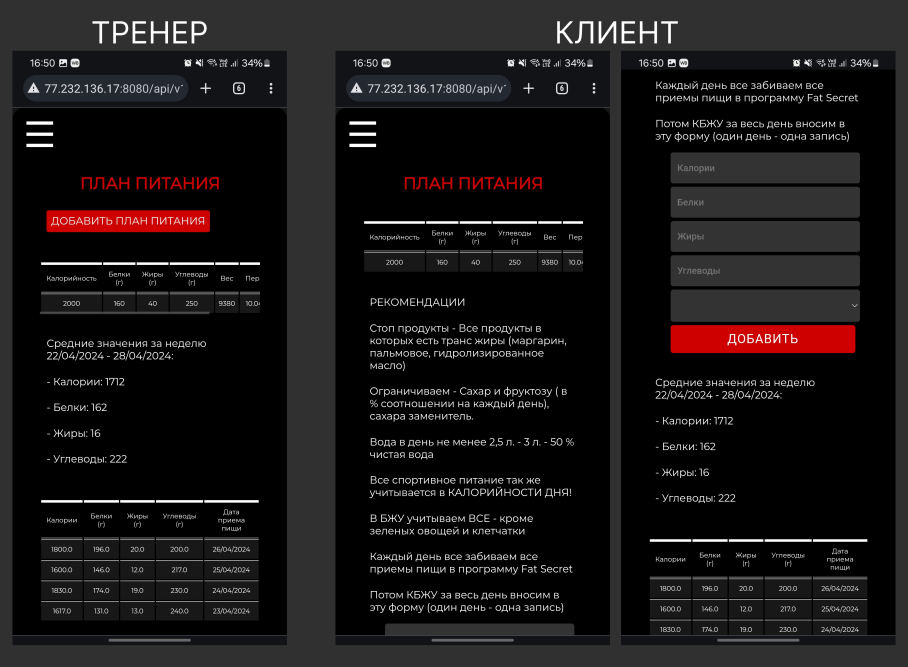


Рисунок 15 – страница плана питания[[15]](#footnote-15)

Страница с планом питания отделена от остальных разделов. Чтобы её открыть, нужно воспользоваться навигационным меню. Тренер, зайдя на страницу пользователя с планом питания, может установить план питания и видеть, сколько питательных веществ употребляет его клиент (Рисунок 15).

Информация представлена в виде таблицы, в которой указано количество калорий, белков, жиров и углеводов, потребляемых в день, а также дата заполнения. Отчёты заполняются пользователем ежедневно.

Кроме того, на этой странице перед таблицей указано среднее количество КБЖУ, которое пользователь вносит за неделю.

Со стороны пользователя страница с планом питания показывает план питания, составленный тренером. В форме пользователь вносит данные в таблицу. Затем система рассчитывает среднее количество КБЖУ, которое пользователь потребляет за неделю. После этого появляется таблица, в которой каждая строка показывает количество КБЖУ, потребляемых в определённый день. Также на странице прописаны рекомендации по питанию и правила заполнения таблицы.

За отображение информации и обработку данных на странице с планом питания и отчетами по нему отвечают следующие классы программы:

* NutritionController – класс, помеченный аннотациями @Controller и @RequestMapping("/user"). Отвечает за отображение информации на странице с планом питания и сбор данных из формы для заполнения отчета по плану питания;
* NutritionServiceImpl – класс, реализующий интерфейс NutritionService, является посредником между контроллером и репозиторием. Отвечает за сохранение информации об отчетах о питани, обращаясь к репозиторию;
* NutritionRepository – интерфейс, который отвечает за взаимодействие с базой данных, а именно с таблицей nutrition;
* NutritionPlanRepository – интерфейс, который отвечает за взаимодействие с базой данных, а именно с таблицей nutrition\_plan;
* WeekNutritionRepository– интерфейс, который отвечает за взаимодействие с базой данных, а именно с таблицей week\_nutrition;
* Nutrition – класс-сущность, который представляет данные из таблицы nutrition в виде объекта;
* WeekNutrition – класс-сущность, который представляет данные из таблицы week\_nutrition в виде объекта;
* NutritionPlan – класс-сущность, который представляет данные из таблицы nutrition\_plan в виде объекта.

Таким образом, тренер может составить индивидуальный план питания для каждого пользователя и проверить, следует ли он ему. Клиенты могут заполнять отчёты о своём питании через специальную форму, которая находится на той же странице, где выводится таблица со всеми отчётами. В итоге и тренер, и клиент могут видеть все данные о питании на одной странице и контролировать его соблюдение.

## 2.3 ОБЕСПЕЧЕНИЕ ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

Информационная безопасность – это сохранение и защита данных, а также ее основных составляющих, в том числе системы и оборудование, предназначенные для использования, сбережения и передачи этой информации.

Обеспечение информационной безопасности — это комплекс мер для обеспечения бесперебойной работы сайта, секретности, целостности и доступности пользователям хранящейся информации;

* протокол SSL [27] (HTTPS). HTTPS соединение шифрует трафик между браузером и сервером. Это усложняет для злоумышленника получение данных в промежуточных узлах (например, извлечение пароля пользователя при входе). Также SSL дает некую гарантию пользователю, что он взаимодействует именно с нужным сайтом (в браузере видна информация чей это сертификат);
* обновление ПО [28]. Периодически разработчики находят уязвимости в ПО и выпускают обновления. Если не делать обновления, то есть риск, что этой уязвимостью воспользуются злоумышленники. Необходимо проводить профилактику сервера и ПО на нем;
* правильная структура данных в базе данных. Если структура базы данных имеет проблемы, то при частичном обновлении данных может нарушиться целостность данных. Необходимо использовать нормализацию базы данных достаточного уровня, вводить ограничения, устанавливать внешние ключи и т.д;
* защита от DOS [29]. DOS атака — это организация множества запросов на сайт с целью его перегрузки. Происходит отказ в обслуживании — сервер просто начинает не справляться с возникшей нагрузкой. Защита от DOS атаки лежит в первую очередь на хостере;
* защита от CSRF атаки [30]. CSRF (Cross-Site Request Forgery, также XSRF) – опаснейшая атака, которая приводит к тому, что хакер может выполнить на неподготовленном сайте массу различных действий от имени других, зарегистрированных посетителей. Какие это действия – отправка ли сообщений, перевод денег со счёта на счёт или смена паролей – зависят от сайта, но в любом случае эта атака входит в образовательный минимум веб-разработчика.
* анонимность информационной системы [31]. Система должна быть внутренним продуктом организации, доступ к которой осуществляется по ссылке. Эта ссылка не должна быть видна в поисковых системах и не появляться на просторах форумов. Каждые полгода система должна менять доменное имя для полноценного сохранения анонимности.
* повышенное внимание к безопасности – это прежде всего вклад в успешное функционирование и развитие бизнеса.

## 2.4 ТЕСТИРОВАНИЕ И ОЦЕНКА КАЧЕСТВА ИНФОРМАЦИОННОЙ СИСТЕМЫ

В ходе тестирования web-приложения для работы с клиентами фитнес-клуба был разработан и выполнен обширный план тестирования, включающий в себя тестирование всего функционала приложения с разных устройств и учетных записей с разным уровнем доступа.

Методы тестирования включали ручное тестирование разработчика и некоторых тренеров фитнес клуба, а также была произведена тестовая работа с клиентами. Результаты тестирования выявили несколько ошибок, влияющих на работу сотрудников в системе: неправильно отображались и заполнялись отчеты по тренировкам. Результаты тестирования были документированы и учтены автором для дальнейшей оптимизации. Все критические ошибки были исправлены.

Оценка качества системы проводилась на основе критериев надежности, функциональности и производительности. Показатели надежности системы превысили установленные требования, что подтверждает ее стабильность и отказоустойчивость.

В целом, тестирование и оценка качества информационной системы позволили выявить ее сильные и слабые стороны, а также определить направления для дальнейшего улучшения и совершенствования системы.

В процессе стандартного тестирования проект успешно проявил себя в:

* кроссбраузерности;
* удобстве пользования (usability) [32];
* тестирование интерфейса пользователя.

Тестирование usability – это процесс оценки и проверки удобства использования информационной системы или веб-сайта для конечных пользователей. Целью тестирования usability является выявление проблем, с которыми пользователи могут столкнуться при взаимодействии с системой, а также сбор обратной связи и предложений по улучшению пользовательского опыта.

В ходе тестирования usability пользователи выполняют различные задачи или сценарии использования системы, в то время как исследователи наблюдают и анализируют их взаимодействие и отзывы. Оценка проводится на основе таких критериев, как простота навигации, понятность интерфейса, эффективность выполнения задач, удовлетворенность пользователей и другие.

Результаты тестирования usability помогают выявить слабые места в системе и принять меры по их исправлению. Это может включать переработку интерфейса, оптимизацию навигации, улучшение понятности инструкций и т. д. Путем повторных тестирований и итераций можно достичь лучшего usability и обеспечить удобство использования системы для пользователей.

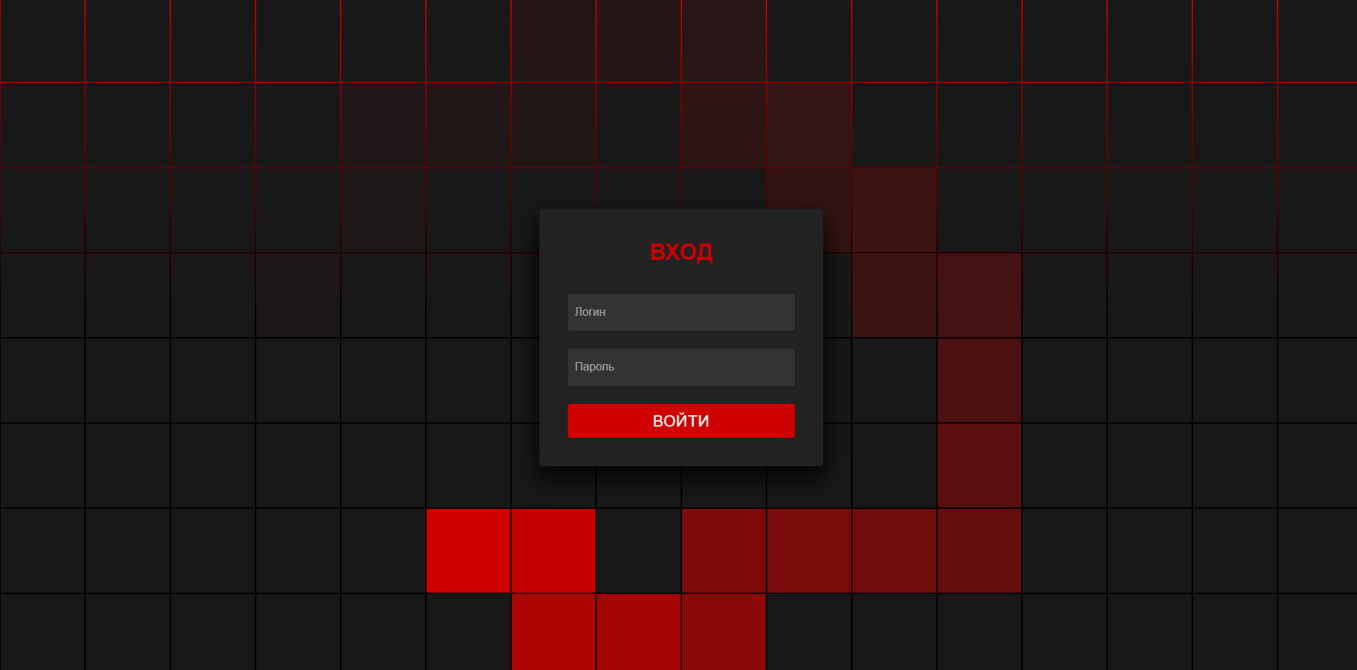


Рисунок 16 - Окно авторизации в приложении[[16]](#footnote-16)

Тестирование интерфейса – это тестирование графического интерфейса пользователя, которое предполагает проверку сайта на соответствие требованиям к графическому интерфейсу, требованиям из ТЗ. Во время данного тестирования ошибок не выявлено.

Тестирование на кроссбраузерность проводилось в популярных браузерах: Google Chrome, Yandex Browser, Opera. Замечаний по кроссбраузерности не выявлено. На рисунке 16 отображен общий вид окна авторизации во всех вышеперечисленных браузерах.

После ввода логина и пароля перед пользователем появляется главная страница с его данными. Вход осуществлен с учетной записи клиента (Рисунок 17).

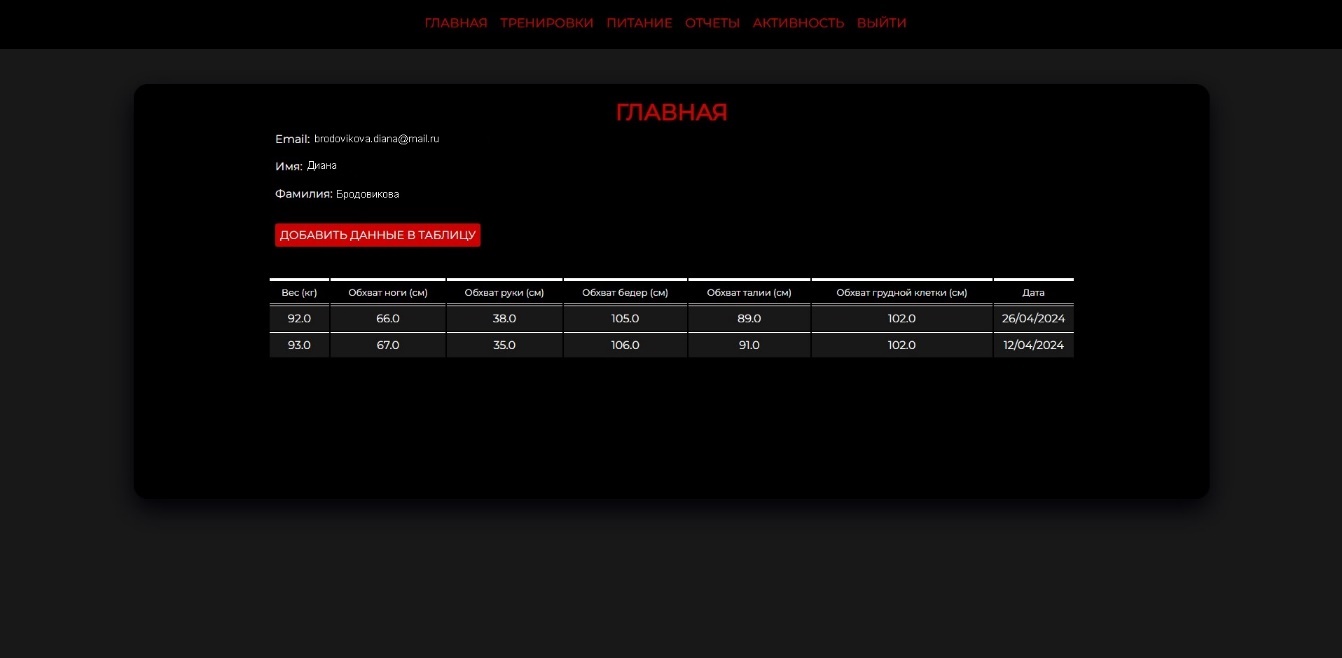


Рисунок 17 – Главная страница клиента в приложении[[17]](#footnote-17)

На главной странице представлено навигационное меню, которое включает в себя все разделы приложения: «тренировки», «питание», «отчеты», «активность», а также кнопку выхода из аккаунта. На главное странице пользователь может видеть свои физические параметры, которые представлены в виде таблицы, а также имеет возможность добавлять новые значения в эту таблицу с помощью специальной формы (Рисунок 18), которая открывается при нажатии на кнопку «добавить данные в таблицу».

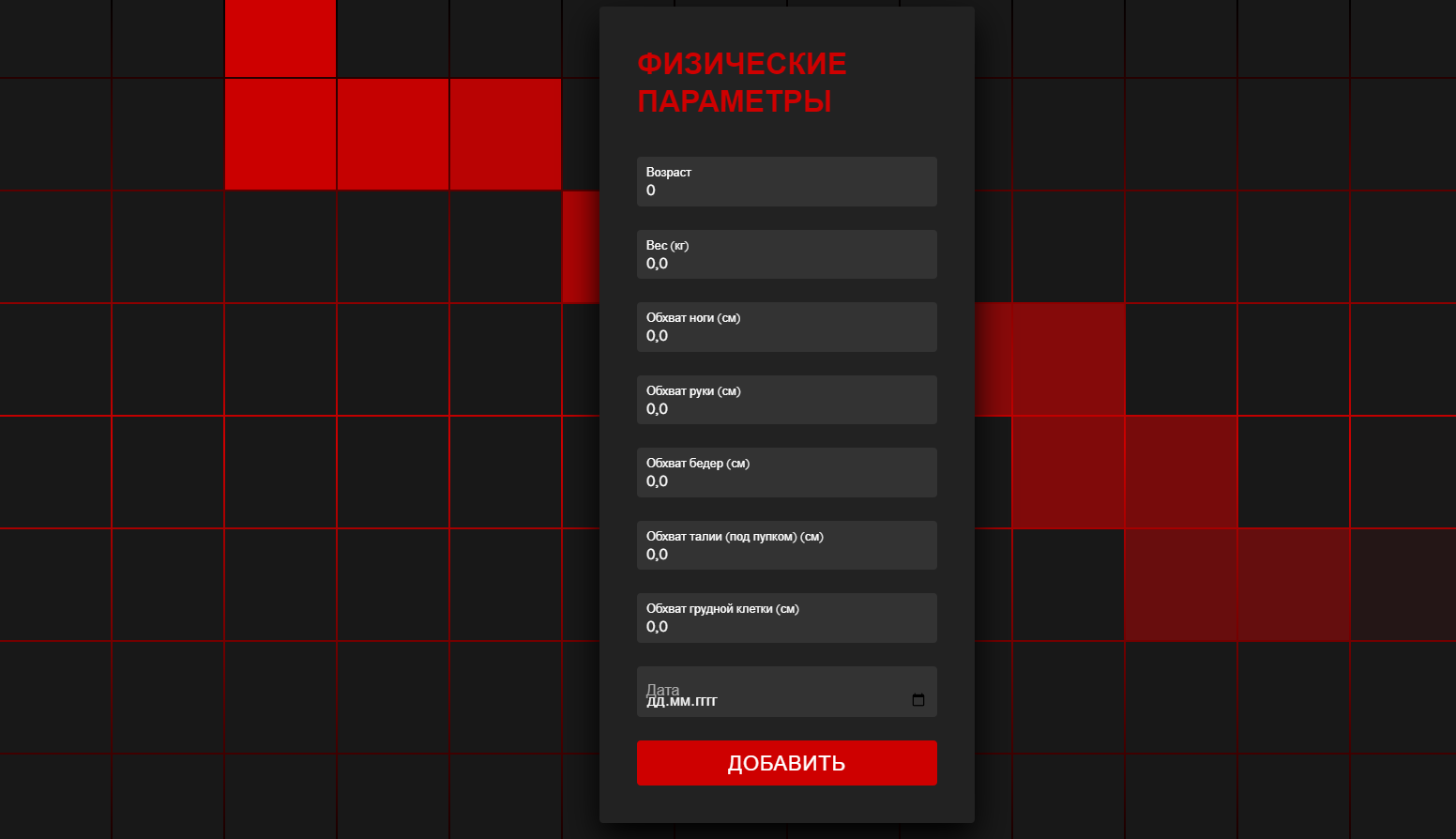


Рисунок 18 – Форма добавления физических параметров[[18]](#footnote-18)

В данной форме пользователь заполняет свой вес, обхват ноги, обхват бедер, обхват талии, обхват грудной клетки и дату дня, в который были сделаны замеры. После нажатия на кнопку «добавить» данные добавляются в базу данных, после чего система возвращает пользователя на главную страницу, где он уже увидит обновленные данные.

Говоря о главной странице и окне авторизации, следует отметить успешное функционирование системы на мобильных устройствах, что поможет пользователю совершать действия в web-приложении с любого удобного для него устройства. Фрагменты адаптивного представления представлены на рисунках 19-21.

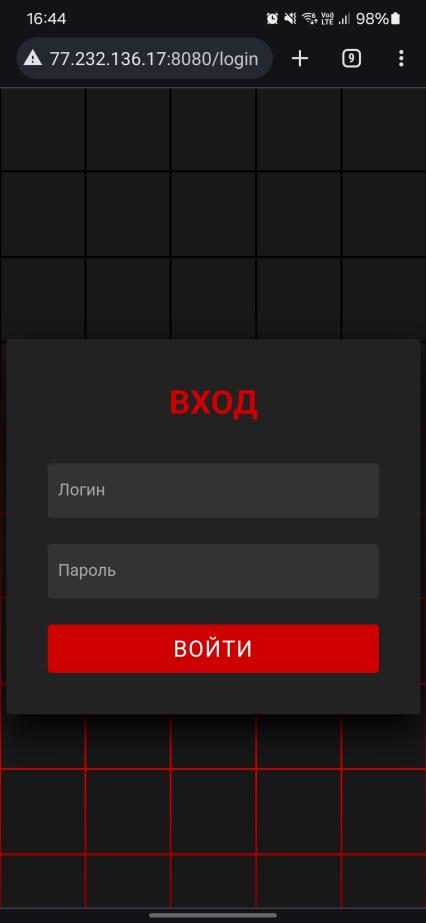


Рисунок 19 - Адаптивное представление приложения на мобильном устройстве[[19]](#footnote-19)



Рисунок 20 - Адаптивное представление приложения на мобильном устройстве[[20]](#footnote-20)

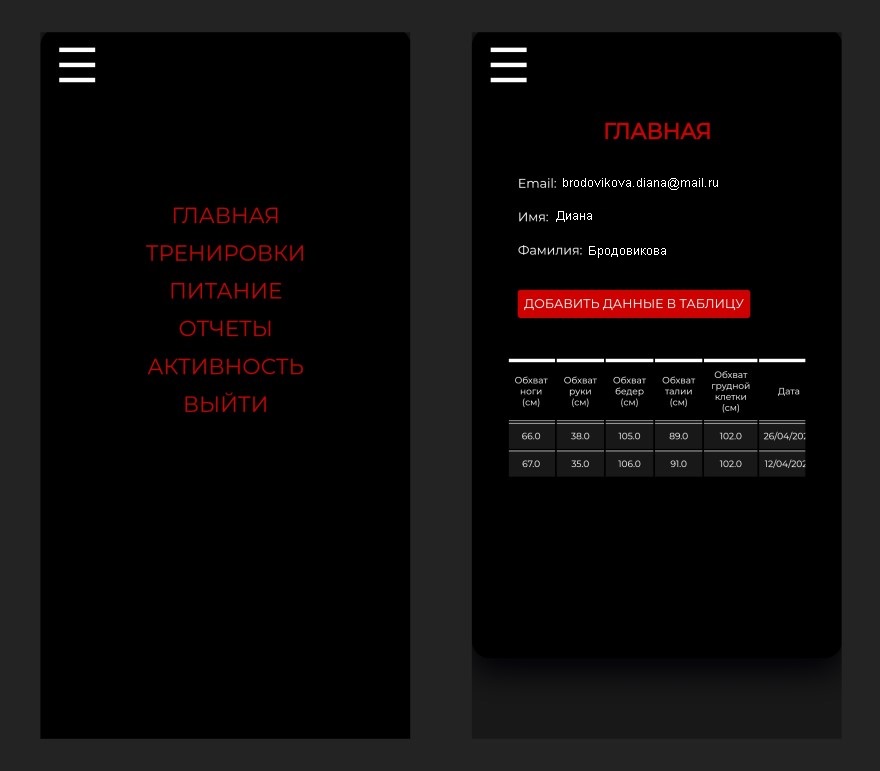


Рисунок 21 – Адаптивное представление меню на мобильном устройстве[[21]](#footnote-21)

При переходе в раздел «тренировки» через навигационное меню, пользователя встречает страница, на которой представлен его тренировочный план (Рисунок 22).

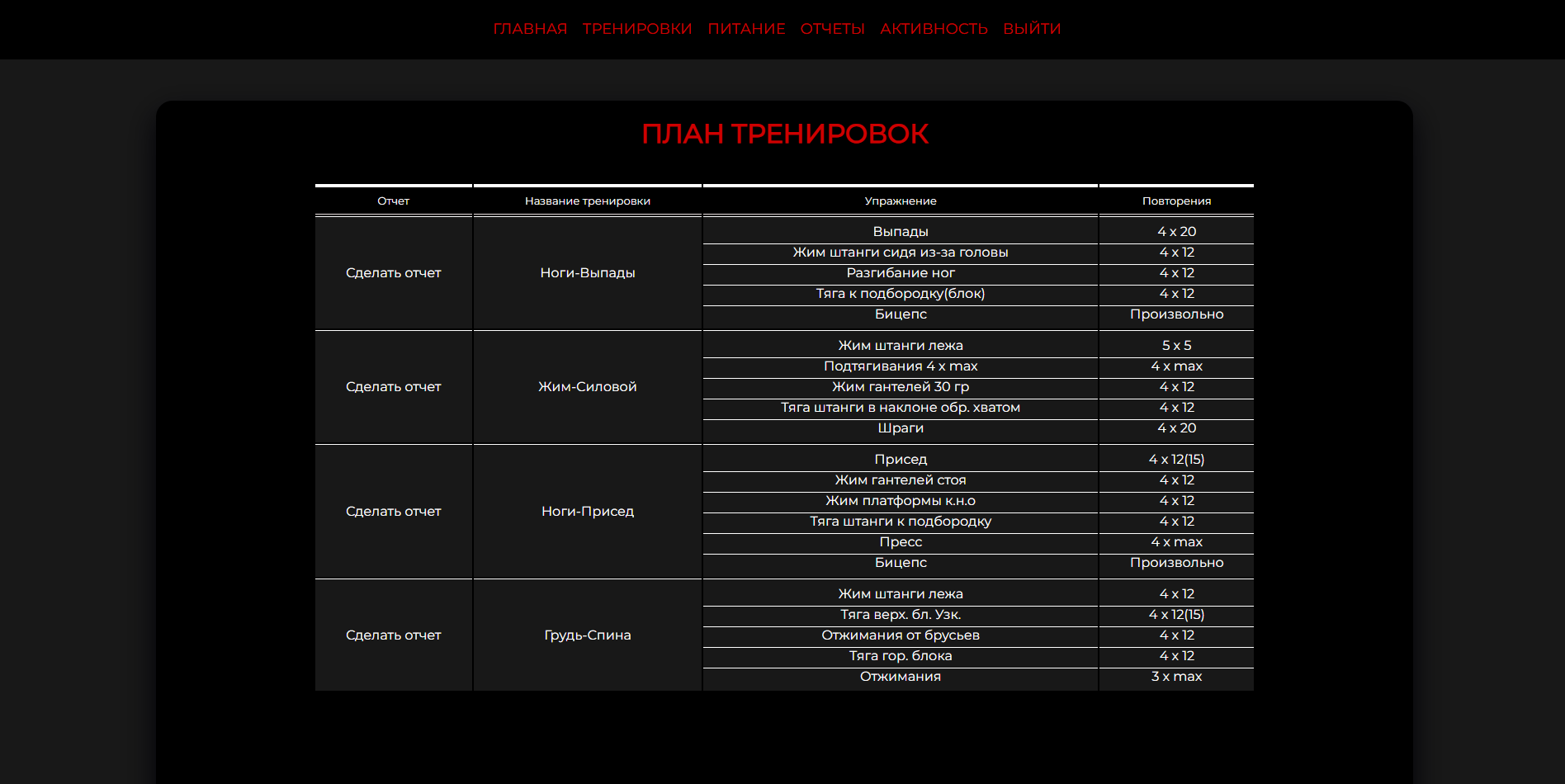


Рисунок 22 – Раздел «тренировки»[[22]](#footnote-22)

Для раздела с тренировочным планом также сделана адаптация под мобильные устройства.

При тестировании приложения, клиенты заполняли отчеты о своих тренировках, которые отображаются на отдельной странице, пример которой представлен на рисунке 23.



Рисунок 23 – Раздел «отчеты»[[23]](#footnote-23)

В данном разделе клиент может наблюдать за прогрессом в своих тренировках. Отчеты в таблицу отсортированы по дате, это дает возможность пользователю внести данные за любой день, при этом структура останется. Также пользователь имеет возможность редактировать и удалять отчеты о тренировках при необходимости.

Следующий раздел – питание. На странице с питанием пользователь видит план питания, составленные для него тренером, план содержит в себе количество КБЖУ, которое необходимо потреблять пользователю и рекомендации по питанию. После чего идет форма для заполнения отчета о питании, в которой пользователь указывает КБЖУ, потребляемые в день и дату. После формы представлены поля, которые выводят средние значения питательных веществ, потребляемых клиентом в день в течении одной календарной недели. Дальше таблица, в которой отображаются все отчеты по питанию клиента (Рисунок 24).

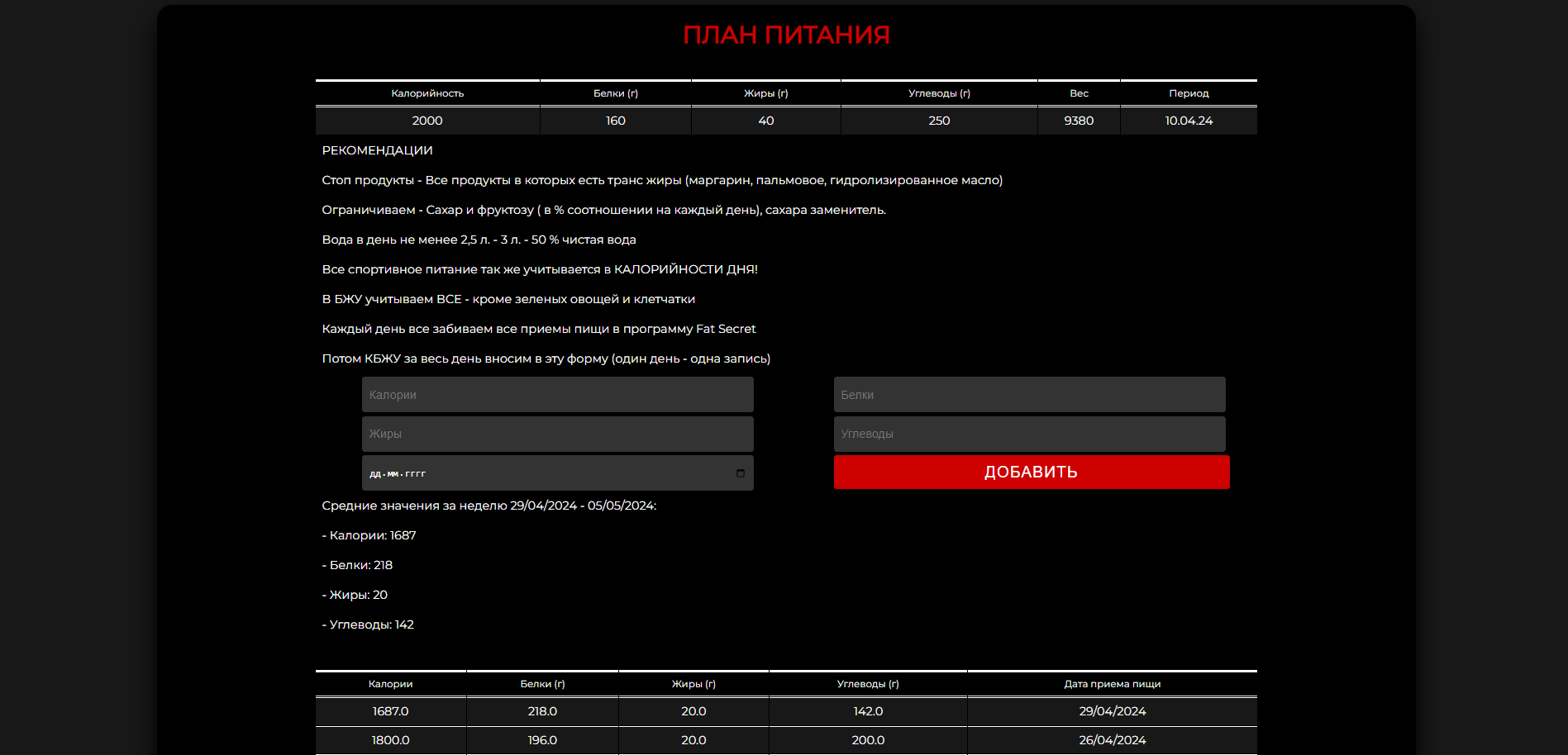


Рисунок 24 – Раздел «питание»[[24]](#footnote-24)

Тестирование разработанного приложения прошло успешно: все идеи и задачи заложенные на этапе проектирования – функционируют, анимации – отыгрываются, данные из базы штатно функционируют с frontend-частью. В дальнейшем автор планирует модернизировать проект в более масштабные объёмы, связанные с расширением функционала.

## 2.5 ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ

При развертывании созданного веб-приложения необходимо убедиться в его надлежащем функционировании. Ответственность за уровень надежности и качество ложится на разработчика, но фактическую реализацию обеспечивает аппаратное обеспечение. Любая информационная система должна иметь достаточное количество ресурсов. Если ресурсы недостаточны, то программа может потерять определенную часть своих функциональных возможностей. В случае критического дефицита ресурсов, приложение может аварийно остановиться. Для предотвращения ситуаций, когда аппаратное обеспечение неспособно справиться с поставленными задачами, определим системные требования для бесперебойной работы сервиса. Такие требования могут быть разделены на две категории:

* минимальные системные требования - это набор характеристик, достаточных для выполнения всех функций приложения без сбоев;
* рекомендуемые системные требования - это набор оптимальных характеристик, обеспечивающих полноценную работу всех функций системы.

На момент разработки проектируемая система должна находиться на серверах хотя бы с минимальными системными требованиями, что в будущем можно усовершенствовать.

Таким образом, минимальными требованиями, предъявляемые пользователю для работы в разработанном приложении являются:

* наличие на устройстве браузера, версия которого не ниже, чем на 10 от самой свежей версии. Иначе, контент на странице может отображаться не корректно, и скрипты выдавать ошибки при обработке информации из форм;
* скорость Интернет-соединения – 5 Мбит/сек.

Рекомендуемыми требованиями, предъявляемыми пользователю для исполнения функций в проектируемой ИС являются:

* браузер последней версии;
* скорость Интернет-соединения – 10 Мбит/сек.

В этом параграфе были выявлены минимальные и рекомендуемые системные требования для использования разработанного приложения.

## 2.6 РАСЧЕТ ЭКОНОМИЧЕСКОЙ ЭФФЕКТИВНОСТИ ПРОЕКТА

При создании проекта всегда нужно учитывать его эффективность. В случае данного проекта, можно оценить то, как именно повлияет появление приложения для работы с клиентами на дальнейшее развитие в этой сфере.

Как уже обозначалось в начале работы, взаимодействие тренера и клиента происходило по средствам таблиц. Работа с каждым клиентом, при использовании такого варианта, занимает достаточно много времени, так как нужно обрабатывать много информации и следить за всеми. Один клиент – один файл. Один файл – пять таблиц. Также играет роль взаимодействия с этими файлами с разных устройств. Приложение Google-таблицы не подходит для полноценной работы на телефоне.

В итоге, имеется неэффективный и перегруженный метод взаимодействия тренера и клиента, который в какой-то степени не удобен обоим сторонам.

Решением выступает отдельное приложение, которое решает все вышеперечисленные проблемы. Суть заключается в том, что все нужное структурировано и находится в одном месте. Тренеру больше не нужно создавать файлы с таблицами на каждого клиента, а потом следить за всем этим. Теперь тренер регистрирует пользователя, которому сразу после регистрации приходит письмо на почту с ссылкой на вход в приложение. То, что приложение является по сути сайтом в интернете и адаптировано под все браузеры и устройства, также является плюсом, так как не нужно ничего устанавливать на свое устройство. А для доступа нужна только ссылка и данные аккаунта.

Также такой формат тренировок можно применять и онлайн. У клиента всегда будут прописаны индивидуальные планы питания и тренировок, которые, в свою очередь, будут обновляться тренером в зависимости от результатов.

Исходя из всего вышеперечисленного, можно сделать вывод, что эффективность приложения обусловлена тем, что тренер может следить за большим количеством клиентов, затрачивая на это меньше времени. Клиенты, в то же время, получают инструмент, который помогает им добиваться поставленных целей и следить за результатами тренировок. К этому можно также добавить тот факт, что приложение будет привлекать новых клиентов от тех, кто уже им пользуется.

# ЗАКЛЮЧЕНИЕ

Результатом выпускной квалификационной работы стала разработка web-приложения для работы с клиентами фитнес клуба. В данной работе были рассмотрены актуальные вопросы создания платформы, с помощью которой тренер может производить работу с клиентом.

В ходе выполнения работы была достигнута поставленная цель и выполнены следующие задачи:

* анализ работы тренера с клиентом;
* выявление потребностей пользователей;
* проектирование функционала приложения;
* разработка дизайна и пользовательского интерфейса;
* разработка web-приложения для работы с клиентами фитнес клуба;
* тестирование и оптимизация разработанного приложения.

В ходе написания выпускной квалификационной работы автор изучил основы работы с Spring Framework, практиковал программирование на языке Java, создал схему данных для приложения. Также автором получен большой опыт в области backend и frontend разработки. Впервые был произведен деплой проекта – выгрузка и запуск приложения на удаленном сервере.

Были отмечены основные направления необходимого профессионального роста, совершенствования навыков программирования и расширения научных знаний. Итогом работы стало внедрение приложения в организацию.

В дальнейшем планируется развитие приложения, добавление больше функционала и больше возможностей для пользователей. Также планируется совершенствование системы до уровня, который мог бы быть равен профессиональной разработке.

В итоге, в ходе проделанной работы было разработано web-приложение, которое соответствует всем требованиям заказчика и может являться конкурентоспособным на рынке современных приложений.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Кэти Сиерра и Берт Бейтс Head First Java. - 2-е изд. - Москва: Эксмо, 2021. - 720 с. – ISBN 978-5-699-54574-2.
2. Все что вы (не) хотели знать о Data Science // habr URL: https://habr.com/ru/companies/citymobil/articles/557416/ (дата обращения: 29.04.2024).
3. HTML для начинающих: вопросы и ответы // habr URL: https://habr.com/ru/companies/vdsina/articles/500190/ (дата обращения: 29.04.2024).
4. Documentation // PostgreSQL URL: https://www.postgresql.org/docs/ (дата обращения: 29.04.2024).
5. Spring Framework // Spring Framework URL: https://spring.io/projects/spring-framework (дата обращения: 29.04.2024).
6. Основные функции Apache Tomcat // habr URL: https://habr.com/ru/companies/otus/articles/800377/ (дата обращения: 29.04.2024).
7. Знакомство с CSS // htmlacademy URL: https://htmlacademy.ru/blog/css/introduction-to-css (дата обращения: 29.04.2024).
8. Современный учебник JavaScript // Современный учебник JavaScript URL: https://learn.javascript.ru/ (дата обращения: 29.04.2024).
9. Timeweb Cloud // habr URL: https://habr.com/ru/companies/timeweb/profile/ (дата обращения: 29.04.2024).
10. Make Your Software Vision a Reality // jetbrains URL: https://www.jetbrains.com/ (дата обращения: 29.04.2024).
11. Spring Boot // Spring URL: https://spring.io/projects/spring-boot (дата обращения: 30.04.2024).
12. Как фреймворк Hibernate помогает Java‑разработчику облегчить работу с базами данных // Яндекс Практикум URL: https://practicum.yandex.ru/blog/hibernate-java/ (дата обращения: 30.04.2024).
13. Руководство по Maven. POM // proselyte URL: https://proselyte.net/tutorials/maven/pom/ (дата обращения: 29.04.2024).
14. Welcome to Apache Maven // Apache Maven URL: https://maven.apache.org/ (дата обращения: 29.04.2024).
15. Maven dependencies // jetbrains URL: https://www.jetbrains.com/help/idea/work-with-maven-dependencies.html (дата обращения: 29.04.2024).
16. Что такое база данных? // oracle URL: https://www.oracle.com/cis/database/what-is-database/ (дата обращения: 29.04.2024).
17. Алан Бьюли Изучаем SQL. - СанктПетербург–Москва: Символ, 2007. - 308 с. – ISBN 0­-596-­00727-­2.
18. Spring MVC — основные принципы // habr URL: https://habr.com/ru/articles/336816/ (дата обращения: 29.04.2024).
19. Spring Boot Annotations // baeldung URL: https://www.baeldung.com/spring-boot-annotations (дата обращения: 29.04.2024).
20. Spring Data // spring.io URL: https://spring.io/projects/spring-data (дата обращения: 29.04.2024).
21. Идентификация, Аутентификация, Авторизация. В чем же разница? // habr URL: https://habr.com/ru/articles/720842/ (дата обращения: 29.04.2024).
22. Spring Security // Spring URL: https://spring.io/projects/spring-security (дата обращения: 30.04.2024).
23. Что такое CSRF токен // dev-notes URL: https://www.dev-notes.ru/articles/security/csrf-bypassing-token-validation/ (дата обращения: 30.04.2024).
24. Что такое формат JSON // TimeWeb Cloud URL: https://timeweb.com/ru/community/articles/format-json (дата обращения: 30.04.2024).
25. Posting with Java HttpClient // Baeldung URL: https://www.baeldung.com/java-httpclient-post (дата обращения: 30.04.2024).
26. Annotation Interface RestController // Документация по Spring URL: https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/bind/annotation/RestController.html (дата обращения: 30.04.2024).
27. Бирюков Андрей Александрович Информационная безопасность: защита и нападение. - Москва: ДМК-Пресс, 2017. - 434 с. – ISBN 978-5-97060-435-9.
28. Обновление программ: зачем нужно и как автоматизировать? // IT аутсорсинг URL: https://upsystems.ru/article9 (дата обращения: 30.04.2024).
29. DDoS-атаки — что это: происхождение, виды и способы защиты // Академия Selectel URL: https://selectel.ru/blog/ddos-attacks/ (дата обращения: 30.04.2024).
30. Атака CSRF // Современный учебник JavaScript URL: https://learn.javascript.ru/csrf (дата обращения: 30.04.2024).
31. Карпов Дмитрий Сергеевич, Ибрагимова Заира Анзоровна СПОСОБЫ И СРЕДСТВА ОБЕСПЕЧЕНИЯ АНОНИМНОСТИ В ГЛОБАЛЬНОЙ СЕТИ ИНТЕРНЕТ // [Правовая информатика](https://cyberleninka.ru/journal/n/pravovaya-informatika). - 2021. - С. 8.
32. Как провести юзабилити-тестирование с респондентом и не провалить его // Хабр URL: https://habr.com/ru/companies/lamoda/articles/673884/ (дата обращения: 30.04.2024).
33. Spring для ленивых. Основы, базовые концепции и примеры с кодом. Часть 1 // Java Rush URL: https://javarush.com/groups/posts/spring-framework-java-1 (дата обращения: 30.04.2024).
34. Введение в Java EE // Java Rush URL: https://javarush.com/groups/posts/2637-vvedenie-v-java-ee (дата обращения: 30.04.2024).
35. Clean Architecture with Spring Boot // Baeldung URL: https://www.baeldung.com/spring-boot-clean-architecture (дата обращения: 30.04.2024).
36. Сакович В. В. Кожомбердиева Г. И. Бураков Д. П. Использование фреймворков семейства Spring Projects для разработки веб-приложений на платформе Java // Интеллектуальные технологии на транспорте. - 2023. - С. 9.
37. Знакомство с JavaScript // Html Academy URL: https://htmlacademy.ru/blog/js/introduction-to-javascript (дата обращения: 30.04.2024).
38. Java Security Overview // Oracle URL: https://docs.oracle.com/en/java/javase/21/security/java-security-overview1.html#GUID-69EE84E6-E0BD-48B2-B3F2-200D9A5FCF93 (дата обращения: 30.04.2024).
39. Lesson: Object-Oriented Programming Concepts // Oracle URL: https://docs.oracle.com/javase/tutorial/java/concepts/ (дата обращения: 30.04.2024).
40. Принципы ООП // Java Rush URL: https://javarush.com/groups/posts/principy-oop (дата обращения: 30.04.2024).
41. Getting Started with Java Properties // Baeldung URL: https://www.baeldung.com/java-properties (дата обращения: 30.04.2024).
42. What is jQuery? // jQuery URL: https://jquery.com/ (дата обращения: 30.04.2024).
43. Lesson: JDBC Introduction // Oracle URL: https://docs.oracle.com/javase/tutorial/jdbc/overview/index.html (дата обращения: 30.04.2024).
44. Learn JPA & Hibernate // Baeldung URL: https://www.baeldung.com/learn-jpa-hibernate (дата обращения: 30.04.2024).

# Приложение А

@Configuration  
@EnableWebSecurity  
@EnableMethodSecurity  
public class SecurityConfig {  
 @Bean  
 public UserDetailsService getUserDetailsService() {return new SecurityService();}  
 @Bean  
 public PasswordEncoder passwordEncoder(){  
 return new BCryptPasswordEncoder();  
 }  
 @Autowired  
 private SecurityService securityService;  
  
 @Bean  
 public SecurityFilterChain filterChain(HttpSecurity httpSecurity) throws Exception {  
 return httpSecurity  
 .csrf(Customizer.*withDefaults*())  
 .authorizeHttpRequests(request->  
 request.requestMatchers("/user/create","/login","/css/\*\*",  
 "/js/\*\*","/images/\*\*", "/submitForm").permitAll()  
 .anyRequest().authenticated())  
 .formLogin(form-> form.loginPage("/login")  
 .defaultSuccessUrl("/api/v1/home",true))  
 .build();  
 }  
 @Bean  
 public AuthenticationProvider authenticationProvider() {  
 DaoAuthenticationProvider provider = new DaoAuthenticationProvider();  
 provider.setUserDetailsService(getUserDetailsService());  
 provider.setPasswordEncoder(passwordEncoder());  
 return provider;  
 }  
  
 protected void configure(AuthenticationManagerBuilder auth) throws Exception {  
 auth.userDetailsService(securityService);  
 }  
}

# Приложение Б

@Service  
@RequiredArgsConstructor  
public class MailServiceImpl implements MailService {  
  
 private final Configuration configuration;  
 private final JavaMailSender javaMailSender;  
  
 @Override  
 public void sendMail(User user, MailType type, Properties properties) {  
 switch (type) {  
 case *REGISTRATION* -> sendRegistrationEmail(user, properties);  
 default -> {}  
 }  
 }  
 @SneakyThrows  
 private void sendRegistrationEmail(User user, Properties properties){  
 MimeMessage mimeMessage = javaMailSender.createMimeMessage();  
 MimeMessageHelper helper = new MimeMessageHelper(mimeMessage, false, "UTF-8");  
 helper.setSubject("Аккаунт зарегистрирован! " + user.getFirstName());  
 helper.setTo(user.getUserName());  
 String emailContent = getRegistrationEmailContent(user);  
 helper.setText(emailContent, true);  
 javaMailSender.send(mimeMessage);  
 }  
 @SneakyThrows  
 private String getRegistrationEmailContent(User user){  
 StringWriter stringWriter = new StringWriter();  
 Map<String, Object> model = new HashMap<>();  
 model.put("name", user.getFirstName());  
 model.put("email", user.getUserName());  
 model.put("password", user.getPassword());  
 configuration.getTemplate("registration.html")  
 .process(model,stringWriter);  
 return stringWriter.getBuffer().toString();  
 }  
}

# Приложение В

const *form* = *document*.getElementById('dynamic-form');  
const *addInputBtn* = *document*.getElementById('add-btn');  
const *submitBtn* = *document*.getElementById('submit-btn');  
let *counter* = 1;

addInputBtn.addEventListener('click', function(event) {  
 event.preventDefault();  
 counter++;  
  
 const newFormGroup = document.createElement('div');  
 newFormGroup.className = 'inputBox';  
 // newFormGroup.classList.add('form-group');  
  
 const newInput1 = document.createElement('input');  
 newInput1.type = 'text';  
 newInput1.name = "exercises" + counter;  
 newInput1.id = "exercises" + counter;  
 newInput1.required = true;  
 newFormGroup.appendChild(newInput1);  
  
 const newI1 = document.createElement('i');  
 newI1.textContent = 'Упражнение ' + counter;  
 newFormGroup.appendChild(newI1);  
  
 const newFormGroup2 = document.createElement('div');  
 newFormGroup2.className = 'inputBox';  
 newFormGroup2.classList.add('form-group');  
  
 const newInput2 = document.createElement('input');  
 newInput2.type = 'text';  
 newInput2.name = "repetitions" + counter;  
 newInput2.id = "repetitions" + counter;  
 newInput2.required = true;  
 newFormGroup2.appendChild(newInput2);  
  
 const newI2 = document.createElement('i');  
 newI2.textContent = 'Повторения';  
 newFormGroup2.appendChild(newI2);  
  
 form.insertBefore(newFormGroup, addInputBtn);  
 form.insertBefore(newFormGroup2, addInputBtn);  
  
});

# Приложение Г

submitBtn.addEventListener('click', function(event) {  
 event.preventDefault();  
  
 const formData = {};  
 const inputs = document.querySelectorAll('.form input');  
  
 // Создаем списки для упражнений и повторений  
 const exercisesList = [];  
 const repetitionsList = [];  
  
 inputs.forEach(function(input) {  
 // Проверяем имя input и добавляем данные в соответствующий список  
 if (input.name.includes('exercises')) {  
 exercisesList.push(input.value);  
 } else if (input.name.includes('repetitions')) {  
 repetitionsList.push(input.value);  
 } else if (input.type !== 'submit') {  
 formData[input.name] = input.value;  
 }  
 });  
 formData['exercises'] = exercisesList;  
 formData['repetitions'] = repetitionsList;  
 formData['userId'] = userId;  
 const token = $("meta[name='\_csrf']").attr("content");  
 // const header = $("meta[name='\_csrf\_header']").attr("content");  
  
 fetch('/submitForm', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'X-CSRF-TOKEN': token  
 },  
 body: JSON.stringify(formData)  
 })  
 .then(response => {  
 console.log(response)  
 window.location.href = '/api/v1/getWorkoutPage/' + userId  
 })  
 .catch(error => {  
 console.error('Ошибка при отправке формы:', error);  
 });  
  
});

# Приложение Д

*addButton*.addEventListener('click', function() {  
 const formData = {};  
 const inputs = *document*.querySelectorAll('.form input');  
 const i = *document*.querySelectorAll('.form i')  
  
 let exercisesList = [];  
 let reportsList = [];  
 i.forEach(function(ii) {  
 if (ii.textContent !== 'Дата'){  
 exercisesList.push(ii.textContent);  
 }  
 });  
 inputs.forEach(function(input) {  
 // Проверяем имя input и добавляем данные в соответствующий список  
 if (input.name.includes('exercise')) {  
 if(input.value === ''){  
 reportsList.push('Не сделано');  
 }else {  
 reportsList.push(input.value);  
 }  
  
 } else if (input.name.includes('date')) {  
 formData[input.name] = input.value;  
 }  
 });  
 formData['exercises'] = exercisesList;  
 formData['reports'] = reportsList;  
  
 const token = $("meta[name='\_csrf']").attr("content");  
 // const header = $("meta[name='\_csrf\_header']").attr("content");  
 fetch('/submitReportForm', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'X-CSRF-TOKEN': token  
 },  
 body: *JSON*.stringify(formData)  
 })  
 .then(response => {  
 *console*.log(response)  
 })  
 .catch(error => {  
 *console*.error('Ошибка при отправке формы:', error);  
 });  
});

1. -2 Составлено автором [↑](#footnote-ref-1)
2. [↑](#footnote-ref-2)
3. Составлено автором [↑](#footnote-ref-3)
4. Составлено автором [↑](#footnote-ref-4)
5. Составлено автором [↑](#footnote-ref-5)
6. Составлено автором [↑](#footnote-ref-6)
7. Составлено автором [↑](#footnote-ref-7)
8. Составлено автором [↑](#footnote-ref-8)
9. Составлено автором [↑](#footnote-ref-9)
10. Составлено автором [↑](#footnote-ref-10)
11. Составлено автором [↑](#footnote-ref-11)
12. Составлено автором [↑](#footnote-ref-12)
13. Составлено автором [↑](#footnote-ref-13)
14. Составлено автором [↑](#footnote-ref-14)
15. Составлено автором [↑](#footnote-ref-15)
16. Составлено автором [↑](#footnote-ref-16)
17. Составлено автором [↑](#footnote-ref-17)
18. Составлено автором [↑](#footnote-ref-18)
19. -20 Составлено автором [↑](#footnote-ref-19)
20. [↑](#footnote-ref-20)
21. -22 Составлено автором [↑](#footnote-ref-21)
22. [↑](#footnote-ref-22)
23. Составлено автором [↑](#footnote-ref-23)
24. Составлено автором [↑](#footnote-ref-24)