**Реферат**

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Лисункова А.Д.*

Пров.

*Бернацкий П.В.*

Н. контр.

*Нистюк О.А.*

Утв.

Смелов В.В.

Реферат

Лит.

Листов

1

*74217076, 2023*

Пояснительная записка дипломного проекта содержит 70 страниц пояснительной записки, 10 таблиц, 13 листингов, 19 формул, 39 иллюстраций, 7 источников литературы, 12 приложений.

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, ANDROID, KOTLIN, JETPACK COMPOSE, SPRING FRAMEWORK, JAVA, MS SQL SERVER

Целью дипломного проекта являлось мобильное приложение для оценки физической активности. Пояснительная записка состоит из введения, семи разделов, заключения и источников литературы.

В первой главе проводится аналитический обзор аналогичных решений и постановка задач.

Вторая глава посвящена проектированию системы и содержит структурную схему базы данных, диаграмму вариантов использования, а также схему алгоритма, задействованного в проекте.

В третьей главе описаны процесс разработки, принципы функционирования и назначение созданных компонентов проекта.

В четвертой главе производится анализ информационной безопасности разрабатываемого мобильного приложения.

В пятой главе описано тестирования разработанного мобильного приложения.

В шестой главе приведено руководство программиста, позволяющее понять моменты развертывания частей проекта.

В седьмой главе производится расчет экономических параметров, срока окупаемости мобильного приложения.

Графическая часть включает в себя 4 диаграммы и 2 плаката, указанных в приложениях записки, суммарно имея 6 графических материалов.

**Abstract**

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Лисункова А.Д.*

Пров.

*Бернацкий П.В.*

Н. контр.

*Нистюк О.А.*

Утв.

Смелов В.В.

Abstract

Лит.

Листов

1

*74217076, 2023*

The explanatory note of the diploma project contains 70 pages of an explanatory note, 10 tables, 13 listings, 19 formulas, 39 illustrations, 7 sources of literature, 12 appendices.

MOBILE APPLICATION, ANDROID, KOTLIN, JETPACK COMPOSE, SPRING FRAMEWORK, JAVA, MS SQL SERVER

The purpose of graduation project was a mobile application for accessing physical activity. The explanatory note consists of an introduction, seven sections, a conclusion and references.

In the first chapter, an analytical review of similar solutions and the formation of tasks is carried out.

The second chapter is devoted to the design of the system and contains a structural diagram of the database, a diagram of use cases, as well as a diagram of the algorithm involved in the project.

The third chapter describes the development process, the principles of operation and the purpose of the created components of the project.

The fourth chapter analyzes the information security of the software being developed.

The fifth chapter describes the testing of developed mobile application.

The sixth chapter contains a programmer’s guide that allow to understand the moments of deployment of parts of the project.

In the seventh chapter, the calculation of economic parameters, the payback period of the software tool is carried out.

The graphic part includes 4 diagrams and 2 posters specified in the appendices of the note, having a total of 6 graphic materials.

**Содержание**

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Лисункова А.Д.*

Пров.

*Бернацкий П.В.*

Н. контр.

*Нистюк О.А.*

Утв.

Смелов В.В.

Содержание

Лит.

Листов

2

*74217076, 2023*

[Введение 7](#_Toc137129821)

[1 Постановка задачи и обзор аналогов 8](#_Toc137129822)

[1.1 Актуальность исследуемой области 8](#_Toc137129823)

[1.2 Обзор аналогов 8](#_Toc137129824)

[1.2.1 Приложение «Step Counter» 9](#_Toc137129825)

[1.2.2 Приложение «Stepz – шагомер и счет калорий» 11](#_Toc137129826)

[1.3 Постановка задачи 13](#_Toc137129827)

[1.4 Обзор технических возможностей 14](#_Toc137129828)

[1.5 Выводы по разделу 15](#_Toc137129829)

[2 Проектирование приложения 16](#_Toc137129830)

[2.1 Архитектура приложения 16](#_Toc137129831)

[2.2 Проектирование базы данных 17](#_Toc137129832)

[2.2.1 Таблицы 18](#_Toc137129833)

[2.2.2 Хранимые процедуры 19](#_Toc137129834)

[2.2.3 Триггеры 19](#_Toc137129835)

[2.3 Диаграмма вариантов использования 19](#_Toc137129836)

[2.4 Схема переходов между экранами приложения 21](#_Toc137129837)

[2.5 Блок-схема алгоритма получения достижений 22](#_Toc137129838)

[2.6 Выводы по разделу 23](#_Toc137129839)

[3 Разработка приложения 24](#_Toc137129840)

[3.1 Реализация серверной части 24](#_Toc137129841)

[3.1.1 Уровень серверной модели 25](#_Toc137129842)

[3.1.1.1 База данных на серверной части 25](#_Toc137129843)

[3.1.1.2 Табличные сущности на сервере 25](#_Toc137129844)

[3.1.2 Уровень сетевого взаимодействия серверного приложения 28](#_Toc137129845)

[3.1.3 Конфигурация безопасности приложения 31](#_Toc137129846)

[3.1.4 Технологии, используемые в серверной части приложения 32](#_Toc137129847)

[3.2 Разработка мобильного приложения 32](#_Toc137129848)

[3.2.1 Разработка уровня бизнес-логики мобильного приложения 33](#_Toc137129849)

[3.2.2 Разработка пользовательского интерфейса 34](#_Toc137129850)

[3.3 Выводы по разделу 38](#_Toc137129851)

[4 Анализ информационной безопасности приложения 39](#_Toc137129852)

[4.1 Возможные угрозы безопасности приложения 39](#_Toc137129853)

[4.2 Реализованные методы защиты приложения 39](#_Toc137129854)

[4.2.1 Использование протокола HTTPS 39](#_Toc137129855)

[4.2.2 Механизм аутентификации 39](#_Toc137129856)

[4.2.3 Методы защиты мобильного приложения 41](#_Toc137129857)

[4.3 Выводы по разделу 42](#_Toc137129858)

[5 Тестирование приложения 43](#_Toc137129859)

[5.1 Позитивное тестирование 43](#_Toc137129860)

[5.2 Негативное тестирование 44](#_Toc137129861)

[5.3 Выводы по разделу 46](#_Toc137129862)

[6 Руководство программиста 47](#_Toc137129863)

[6.1 Руководство развертывания приложения 47](#_Toc137129864)

[6.2 Руководство использования мобильного приложения 48](#_Toc137129865)

[6.3 Выводы по разделу 57](#_Toc137129866)

[7 Технико-экономическое обоснование проекта 58](#_Toc137129867)

[7.1 Общая характеристика разрабатываемого программного средства 58](#_Toc137129868)

[7.2 Исходные данные для проведения расчётов 59](#_Toc137129869)

[7.3 Методика обоснования цены 59](#_Toc137129870)

[7.3.1 Объем программного средства 60](#_Toc137129871)

[7.3.2 Основная заработная плата 61](#_Toc137129872)

[7.3.3 Дополнительная заработная плата 62](#_Toc137129873)

[7.3.4 Отчисления в Фонд социальной защиты населения 62](#_Toc137129874)

[7.3.5 Расходы на материалы 63](#_Toc137129875)

[7.3.6 Расходы на оплату машинного времени 63](#_Toc137129876)

[7.3.7 Прочие прямые затраты 64](#_Toc137129877)

[7.3.8 Накладные расходы 64](#_Toc137129878)

[7.3.9 Сумма расходов на разработку программного средства 64](#_Toc137129879)

[7.3.10 Расходы на сопровождение и адаптацию 65](#_Toc137129880)

[7.3.11 Полная себестоимость 65](#_Toc137129881)

[7.3.12 Определение цены, оценка эффективности 65](#_Toc137129882)

[7.4 Выводы по разделу 68](#_Toc137129883)

[Заключение 69](#_Toc137129884)

[Список использованных источников 70](#_Toc137129885)

[ПРИЛОЖЕНИЕ А Диаграмма компонентов приложения 71](#_Toc137129886)

[ПРИЛОЖЕНИЕ Б Логическая схема базы данных 72](#_Toc137129887)

[ПРИЛОЖЕНИЕ В Диаграмма вариантов использования 73](#_Toc137129888)

[ПРИЛОЖЕНИЕ Г Блок-схема алгоритма получения достижений 74](#_Toc137129889)

[ПРИЛОЖЕНИЕ Д Скриншот главного экрана приложения 75](#_Toc137129890)

[ПРИЛОЖЕНИЕ Е Таблица экономических показателей 76](#_Toc137129891)

[ПРИЛОЖЕНИЕ Ж Листинг репозитория AchievementRepository на сервере 77](#_Toc137129892)

[ПРИЛОЖЕНИЕ И Код интерфейса ParticularMapping 78](#_Toc137129893)

[ПРИЛОЖЕНИЕ К Листинг контроллера RelationshipController 79](#_Toc137129894)

[ПРИЛОЖЕНИЕ Л Листинг класса ProfileRepository 80](#_Toc137129895)

[ПРИЛОЖЕНИЕ М Листинг StepRepository 82](#_Toc137129896)

[ПРИЛОЖЕНИЕ Н Листинг класса FriendsViewModel 86](#_Toc137129897)

Введение

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Лисункова А.Д.*

Пров.

*Бернацкий П.В.*

Н. контр.

*Нистюк О.А.*

Утв.

Смелов В.В.

Введение

Лит.

Листов

1

*74217076, 2023*

Ключевым фактором поддержания здоровья является соблюдение определенного режима при двигательной активности, полноценном сне, разборчивости в питании и своевременном употреблении воды. Сейчас процесс заботы о здоровье становится все сложнее из-за стремительности современной жизни, и ввиду этого многие не уделяют должного внимания своему состоянию из-за нехватки времени. Многие люди изо всех сил пытаются найти мотивацию придерживаться постоянного графика, особенно если у них есть другие обязательства или обязанности. Также их стремление ограничивается доступом к подходящим помещениям или оборудованию, что затрудняет регулярное участие людей в этой деятельности. Поэтому такой большой популярностью обуславливаются приложения, которые включают в себя функции мониторинга активности пользователя.

Функциональным назначением разрабатываемого приложения является просмотр статистики, сравнение результатов с пользователями, установка запланированного количества шагов и объема воды.

Целью дипломного проекта является приложение для отслеживания активности пользователей, с учетом получаемых данных, таких как количество шагов, выпитых стаканов и прочей информацией.

Задачи дипломного проекта:

* изучить аналоги разрабатываемого приложения на рынке;
* спроектировать структуру базы данных, разработать сервер;
* разработать мобильное приложение;
* протестировать разработанное приложение;
* разработать руководство программиста по развертыванию приложения;
* проанализировать угрозы информационной безопасности и методы защиты.

# Постановка задачи и обзор аналогов

ФИО

Подпись

Дата

Лист

1

БГТУ 01.00.ПЗ

Разраб.

*Лисункова А.Д.*

Пров.

*Бернацкий П.В.*

Н. контр.

*Нистюк О.А.*

Утв.

Смелов В.В.

1 Постановка задачи и обзор аналогов

Лит.

Листов

8

*74217076, 2023*

* 1. Актуальность исследуемой области

Одним из ключевых факторов в принятии решения о разработке программного продукта является его своевременность с точки зрения закрытия потребностей пользователя. В данный момент наблюдается тенденция здорового образа жизни, которой “заражаются” все больше и больше людей, что ведет к популяризации программных средств, дающих пользователю информацию о здоровье.

Как отмечается в литературном источнике, разработка мобильных приложений в сфере оздоровления и двигательной активности востребованы среди различных категорий пользователей, благодаря своей открытости и доступности [1]. Служат подкреплением актуальности подобных программных средств выводы публицистов о недостатке физической активности у большинства людей за счет малоподвижного образа жизни, что в результате может привести к болезни гипокинезии, в том числе и у молодых людей [1].

То есть, основной идеей является поддержание человека на минимальном уровне положенной активности. Ввиду сидячей работы или отсутствия самодисциплины, обычная физическая нагрузка может показаться непосильной задачей.

Таким образом, была доказана актуальность темы дипломного проекта.

* 1. Обзор аналогов

Для создания мобильного приложения с привлекательным пользовательским интерфейсом и удобной функциональностью необходимо проанализировать аналоги. Анализ конкурентов способствует не только выявлению достоинств и недостатков, но также помогает понять, к чему стоит стремиться при разработке собственного мобильного приложения и выявить основные потребности пользователей. В результате поиска аналогичных решений с похожей тематикой было выбрано два мобильных приложения. Анализ будет осуществляться путем оценки приложений по различным критериям, таким как: интерфейс, функциональность, наличие нестандартных подходов и т.д.

Оптимальным выбором системы в разрабатываемого продукта предстал смартфон системы Android. Этому есть объективные причины, такие как процентное превышение продаж почти в 71% ото всех операционных систем на мобильных устройствах. Здесь также стоит учитывать процентное соотношение использования устройств по странам. Например, в США доля рынка Android составляет 38%, потому что Apple всегда был популярным брендом в этой стране. В то время как для стран СНГ превалирующей на рынке ОС является Android.

На текущий момент около 70 процентов проданных устройств работают на данной системе. Несмотря на тот вывод из статистики о выгоде разработки программного обеспечения на системе Android, в качестве конкурентного средства также было рассмотрено приложение под Apple.

Так как то, что предлагает рынок из схожей тематики, должно быть также тщательно проанализировано для выявления интересных решений.

1.2.1 Приложение «Step Counter»

В качестве одного из аналогов было рассмотрено приложение «Step Counter» [2]. Главная страница приложения представлена на рисунке 1.1.

На главном экране располагаются данные, собранные за сегодняшний день, такие как шаги, сожженные калории, затраченное время и расстояние. В нижней части страницы расположены разделы «Отчет», «Здоровье», «Настройки».

На первом экране находится прогресс за сегодняшний день. Здесь помечена цель, которую стремится достичь пользователь и фактический результат.



Рисунок 1.1 – Главная страница «Step Counter»

Отнести к достоинствам можно удобную отчетную сборку категорий по текущей неделе активности, и самому провести сравнение своих достижений по дням. В нем указано затраченное время, расстояние, калории и прочие параметры. Это можно увидеть на рисунке 1.2.



Рисунок 1.2 – Страница отчета

Также стоит отметить отсутствие GPS, главным недостатком которого является несовершенная точность и объем потребления заряда устройства.

Хоть в мире, особенно технологическом, мало что может обладать достоверной точностью, но стремление к ней является гарантом совершенствования и позволяет не стоять на месте.

Во вкладке «Здоровье» можно указать параметры тела, такие как рост и вес, и отслеживать динамику колебаний значений.

Простой по восприятию график показывает прогресс по временным промежуткам, на скриншоте взята неделя, и таким образом пользователь может явно увидеть в какой момент он был активнее всего и когда сил не хватало до поставленной цели. Награждение происходит за перевыполненный план.

Ниже горизонтальной оси наблюдаются категории, по которым составляются графики за выбранный период времени.

Вместе с тем доступны запись и контроль количества жидкости.

Надо всего лишь надо задать желаемый для потребления объем, и отмечать "постаканно” фактический объем потребленной жидкости.

Это помогает пользователям оставаться гидратированными и удовлетворять свои ежедневные потребности в воде. Демонстрация данных возможностей приложения показаны на рисунке 1.3.

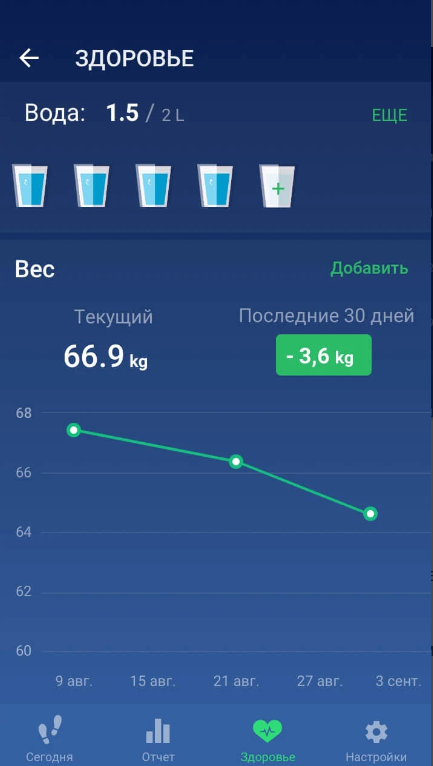


Рисунок 1.3 – Вкладка “Здоровье”

Рассмотренное приложение позволило выявить свои преимущества, но и также заявило о недостатках. Например, к последним можно отнести отсутствие системы получения достижений и скудность параметров.

С другой стороны, минимальность и упор на необходимые функции позволяют разгрузить приложение и дать пользователю получить именно то, что он хочет.

В целом данный аналог является хорошим примером для того, чтобы почерпнуть основную суть подобных приложений, и действовать в том же направлении, что и готовый продукт.

1.2.2 Приложение «Stepz – шагомер и счет калорий»

Данное приложение разработано под iPhone и Apple Watch, в отличие от выше рассмотренного примера, созданного исключительно под Android.

Изучение аналогичных продуктов под разные ОС и виды устройств способствует облегчению поиска новых решений, поэтому не обошло стороной рассмотрение схожей продукции Apple.

На главном экране устройства наблюдаются такие же отслеживаемые параметры, как и в приложении на Android системе, а именно сожженные калории, пройденное расстояние в километрах и количество шагов за день.

Хоть рассматриваемое приложение схоже с предыдущим, но имеет свою особенность. А именно начисление достижений, получаемых при свершенных условий. Демонстрация экрана на рисунке 1.4.



Рисунок 1.4 – Вкладка достижений

Также приложением было предложено просматривать более досконально историю активности на рисунке 1.5. В том числе к положительным сторонам можно причислить распространение на заинтересованную аудиторию достигнутый результат, показанный на рисунке 1.6.



Рисунок 1.5 – История активности

Также к минусам приложения хочется причислить устаревший интерфейс. Несмотря на то, что приложение является по сей день используемым, ему явно не хватает поддержки и современного подхода к визуализации. Как и в прошлом рассмотренном аналоге здесь нет возможности провести сравнение результатов с чужими, только вариант его распространения на другие соцсети, что не является в контексте приложения наилучшим решением.



Рисунок 1.6 – Возможность поделиться в других сетях

Таким образом аналог [3] включил в себя все тот же функционал, что и прошлый, но с дополнением в виде системы начисления достижений и возможности поделиться результатами в социальных сетях.

Минусы были учтены и в дальнейшем проработаны в дипломной разработке.

1.3 Постановка задачи

Первым шагом в разработке дипломного проекта является определение масштаба самого проекта. Например, будет ли оцениваться физическая активность определенной группы населения, например спортсменов, пожилых людей или детей? Или же выбор падет на разработку инструмента, несущего общий характер, который может использовать каждый? Определение целевой аудитории сужает круг задач и показывает, насколько выполнима поставленная цель.

Было принято решение сделать упор на аудиторию без учета возраста, пола и специфики физической нагрузки для большего охвата заинтересованных лиц.

Далее необходимо определиться с технологическим аспектом дипломного проекта, который будет представляться основой реализации оценки физической активности. Доступно множество вариантов, включая носимые устройства, приложения для смартфонов и датчики. Каждый вариант имеет свои преимущества и недостатки, поэтому важно выбрать тот, который лучше всего подходит для проекта. Про это более подробно в подразделе 1.4.

Касательно функционала, необходимым в реализации будет включение возможности сравнения результатов с пользователями, добавление в друзья, просмотр статистики прогресса, а также сам подсчёт шагов и калорий, и наблюдение за выпитым объемом жидкости. Получение достижений и отслеживание не только своих результатов, но и результатов подписчиков, должно представляться мотивационным стимулом для повышения и улучшения уровня активности.

Таким образом, целью дипломного проекта является разработка мобильного приложения для оценки пользовательской активности. Для достижения данной цели необходимо решить последующие задачи:

* разработать базу данных для хранения информации;
* разработать сервер для работы с базой данных и мобильным приложением;
* разработать мобильное приложение под Android.

1.4 Обзор технических возможностей

Немаловажным будет рассмотрение и обнаружение наилучшего способа вычисления динамики ходьбы. Важно понимать, что даже комбинация датчиков не может дать стопроцентный гарант безукоризненного расчета амплитуды шага.

Мало найдется людей, особенно тех, что в критический момент, готовы пожертвовать зарядом в угоду работы приложения. Поэтому, учитывая базовую потребность пользователя обладать максимально доступным зарядом устройства, то хорошим решением будет отказаться от подключения GPS, так как он охотно потребляет заряд мобильного телефона.

Одним из часто используемых датчиков является акселерометр, который измеряет ускорение телефона в трех направлениях: вверх и вниз, вправо и влево, а также вперед и назад. Этот датчик обнаруживает движение телефона и, следовательно, движение человека, который его носит.

Другим датчиком, обычно используемым в шагомерах, является гироскоп. Этот датчик измеряет ориентацию телефона и вращение вокруг своей оси. Это может помочь определить изменения направления и скорости, которые являются важными факторами при подсчете количества пройденных шагов.

Как одним из возможных датчиков, используемых в шагомерах, также используется барометр для измерения атмосферного давления. Этот датчик может обнаруживать изменение высоты, что может быть важным при подсчете количества пройденных ступеней. Но также как и GPS быстро съедает заряд, что будет приводить к скорому разряду устройства.

Сама по себе точность зависит от чувствительности датчика, встроенного в смартфон. Погрешность составляет до 30 процентов, но искажение данных будет меньше у дорогих девайсов, но опять же погрешность не будет чересчур критичной у простых, нефлагманских девайсов.

Большая точность, которая определит колебание именно как шаг, а не какую-то произвольную ротацию устройства, достигается, если расположить телефон в нижней части тела, потому что в таком случае погрешность будет составлять 5-10%.

Свой вклад вносит и тип поверхности. Чем она ровнее, тем точнее будут показатели шагомера. Погрешность при ходьбе по пересечённой местности выше, но всё равно сравнительно небольшая – до 5%. Добиться идеальной точности показаний не позволяют даже профессиональные приборы.

Поэтому из вышеприведенного набора датчиков было принято решение сделать упор на акселерометр и гироскоп, потому что работа вкупе даст счет колебаний и изменение скорости движения.

В дальнейшем развитии приложения, можно будет применить некоторые вспомогательные датчики, но с возможностью платной подписки, к примеру, если пользователь вдруг захочет знать количество пройденных ступеней.

1.5 Выводы по разделу

В данном разделе проведен аналитический обзор существующих приложений, позволяющие организовать процесс оценки активности, выявлены их достоинства и недостатки, а также изучены их специальные возможности.

В результате тщательного рассмотрения аналогичных решений были обнаружены следующие недостатки: сильно порезанный функционал без платной подписки, устаревший дизайн, отсутствие возможности сравнения результатов с пользователями и соответственно, поддержания здоровой конкуренции.

Все вышеупомянутое должно быть учтено в написании дипломного проекта для создания исключительного приложения со схожей тематикой, как у аналогов.

# 2 Проектирование приложения

ФИО

Подпись

Дата

Лист

1

БГТУ 02.00.ПЗ

Разраб.

*Лисункова А.Д.*

Пров.

*Бернацкий П.В.*

Н. контр.

*Нистюк О.А.*

Утв.

Смелов В.В.

2 Проектирование приложения

Лит.

Листов

8

*74217076, 2023*

**2.1 Архитектура приложения**

Основная задача дипломного проекта – создание клиент-серверного приложения, предоставляющего возможности отслеживания состояния физической активности пользователя.

Мобильное приложение представляется клиентской стороной, делегирующей серверу основное выполнение функционала. Таким образом, делая систему разбитой на логический блоки, тем самым упрощая формирование точек связи. Сервером является приложение, расположенное на хостинге для круглосуточного доступа пользователей к данным. Сами данные хранятся в СУБД, с которым связывается сервер для работы с информацией о пользователях. Диаграмма компонентов приложения представлены на рисунке 2.1 и приложении А.

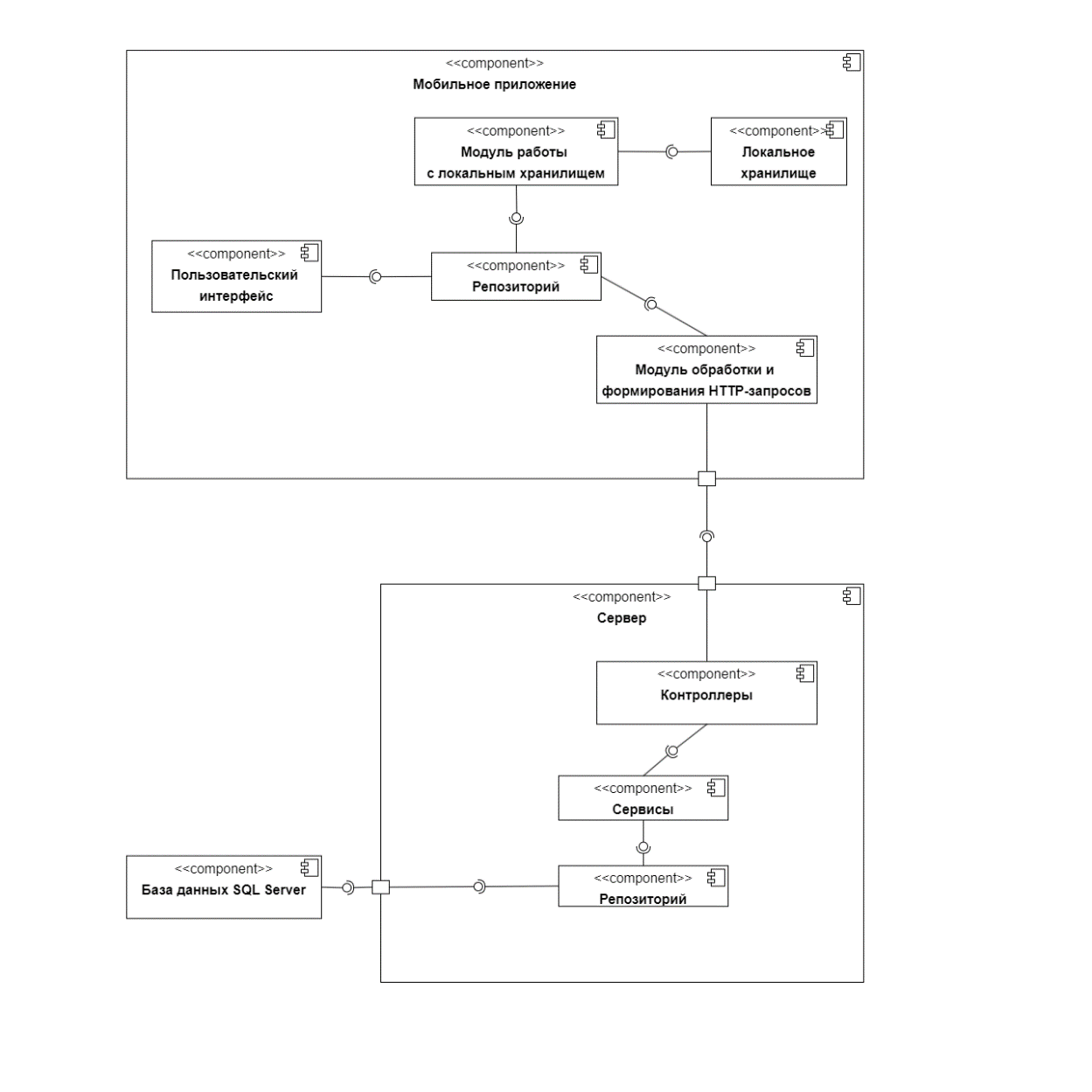


Рисунок 2.1 – Диаграмма компонентов приложения

## 2.2 Проектирование базы данных

В качестве системы управления базами данных была выбрана Microsoft SQL Server 2017. SQL Server – это система управления реляционными базами данных, разработанная корпорацией Microsoft. В СУБД используется язык запросов Transact SQL, являющийся реализацией стандарта ANSI/ISO по структурированному языку запросов SQL с расширениями. Microsoft SQL Server используется для работы как с персональными базами данных, так и с базами данных крупных предприятий.

Перед началом разработки приложения была спроектирована база данных для сервера с 8 таблицами. Список таблиц с их кратким описанием приведен в таблице 2.1. Логическая схема базы данных приведена ниже на рисунке 2.2.

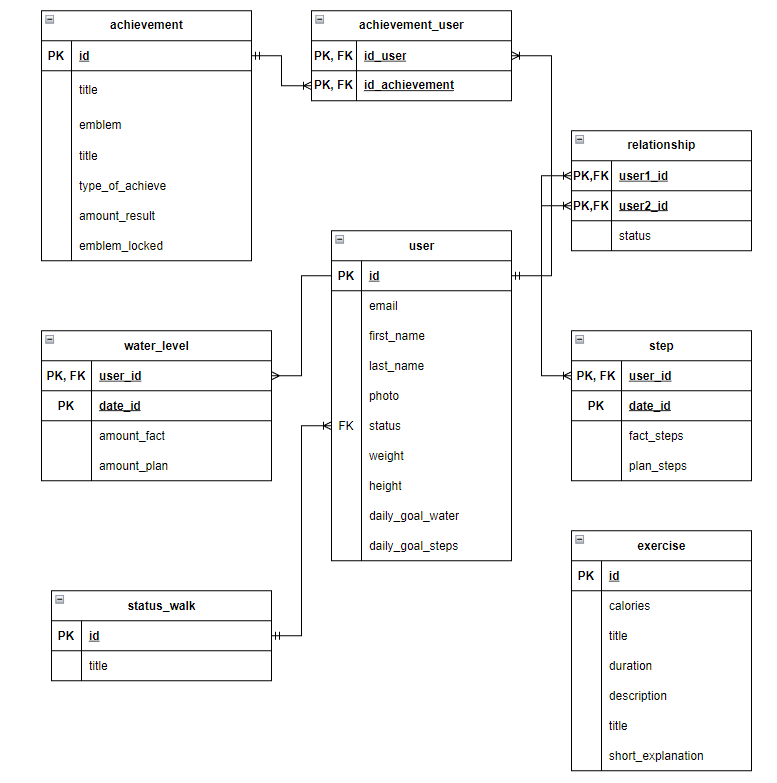


Рисунок 2.2 – Логическая схема базы данных

Обобщенное описание направленности таблиц дает первичное представление о созданной системе связи данных и является мини-презентацией особенностей структуры. Также данная структура приведена в приложении Б.

Структура представлена восьмью таблицами, представленными в таблице 2.1.

Таблица 2.1 – Описание таблиц

|  |  |
| --- | --- |
| Таблица | Описание |
| achievement | Содержит информацию о достижениях |
| achievement\_user | Связывает таблицы Achievement и User\_profile |
| step | Информация о проделанных шагах пользователя |
| user\_profile | Содержит информацию пользователя |
| status\_walk | Определяет статус пользователя по ходьбе |
| water\_level | Содержит информацию об объеме выпитой воды |
| relationship | Устанавливает связи между пользователями |
| exercise | Содержит информацию об упражнениях |

Подробное описание содержимого некоторых основных таблиц будут рассмотрены в последующих пунктах данной записки.

2.2.1 Таблицы

Таблица user\_profile содержит в себе всех пользователей, зарегистрированных в системе, и состоит из полей, приведенных в таблице 2.2.

Таблица 2.2 – Поля таблицы user\_profile

|  |  |
| --- | --- |
| Поле | Описание |
| id | Идентификатор пользователя |
| weight | Вес пользователя |
| height | Рост пользователя |
| daily\_goal\_water | Цель ежедневного уровня потребления жидкости |
| daily\_goal\_steps | Цель ежедневной ходьбы |
| email | Почта gmail пользователя |

Идентификатор пользователя генерируется автоматически при авторизации через аккаунт Google посредством передачи и выявления действительности токена.

Поля step представлены в таблице 2.3.

Таблица 2.3 – Поля таблицы step

|  |  |
| --- | --- |
| Поле | Описание |
| user\_id | Идентификатор пользователя |
| date\_id | Идентификатор даты |
| fact\_steps | Фактические шаги пользователя за обозначенную дату |
| plan\_steps | Планируемые шаги пользователя за обозначенную дату |

Поле daily\_goal\_steps напрямую связано с аналогичным ему полем plan\_steps в таблице step, в которое значение поступает при изменении в профиле пользователя информации о цели на день по количеству шагов.

Здесь поля user\_id и date\_id представляют собой составной первичный ключ, так как один и тот же пользователь может иметь результат по нескольким датам, так

и для одной и той же даты могут сопоставиться несколько пользователей.

Такой же принцип находит место и в таблице water\_level с составным первичным ключом, фактическим и запланированным объемом жидкости, поэтому нет смысла в описании повтора однотипной структуры.

Как было показано на рисунке 2.2 таблицы состоят в различных связях друг с другом. Основная связь представляется зависимостью один-ко-многим, которую можно наблюдать между таблицей water\_level и user\_profile. Данная связка обусловлена исключительностью поля в water\_level в отношении одного пользователя, поскольку несколько дат могут быть зафиксированы у одного человека.

Другая связь в схеме – многие-ко-многим, которая по классике обязует создать новую таблицу, объединяющая две другие. Примером связующей таблицы стала achievement\_user, состоящая из двух полей – идентификаторов таблиц.

2.2.2 Хранимые процедуры

Хранимые процедуры используются в базе данных разрабатываемого приложения для упрощения и ускорения логики обработки и проверки данных, запрашиваемых сервером. Неполный набор хранимых процедур приложения:

– adding\_achievement\_to\_user – записывает достижение пользователю при выполнении условий;

– set\_relationship – возвращает статус отношений между пользователями при выполнении различных действий, влияющих на смену статуса;

– compare\_results\_between\_users – возвращает по заданному промежутку времени результаты двух юзеров по шагам.

2.2.3 Триггеры

В базе данных используются триггеры для таблиц user\_profile, step, water\_level:

– user\_update\_goals – задает логику, затрагивающую значения в таблицах step и water\_level, соответствующих полям целей пользователя;

– daily\_steps\_into\_goal – задает логику первичной установки значения в столбец plan\_steps по показателям из столбца таблицы user\_profile;

– daily\_water\_into\_goal – задает логику первичной установки значения в столбец amount\_plan таблицы water\_level по показателям из столбца таблицы user\_profile.

2.3 Диаграмма вариантов использования

Диаграмма вариантов использования – это поведенческая UML-диаграмма, описывающая набор возможностей системы (преценденты), которые могут быть совершены одним или несколькими внешними пользователями (актерами).

Диаграммы UML (унифицированный язык моделирования) используются для визуального представления архитектуры и дизайна программной системы.

Одним из назначений является предоставление разработчикам, заинтересованным сторонам и бизнес-пользователям общий язык для общения со структурой и понимания поведения системы.

Диаграммы UML помогают понять и проанализировать программные системы,

выявить потенциальные проблемы и проверить проектные решения, принятые в процессе разработки.

Они также помогают облегчить общение между членами команды, документировать систему для использования в будущем и улучшать общее качество программного обеспечения. В создаваемом приложении предусмотрено рассмотрение следующих актеров:

– пользователь;

– мобильное приложение;

– Google поставщик аутентификации.

В ходе разработки мобильного приложения будет задействован один вид пользователя, а именно прошедший этап подтверждения Google-аккаунта, либо же регистрацию, и в последствии согласованности доступа с Google-провайдером.

Основные действия реализует сервер, который сообщается с мобильным приложением по протоколу HTTP. Соответственно, когда посредством мобильного приложения происходит попытка входа в Google-аккаунт, генерируется токен при первом входе либо же происходит извлечение имеющегося токена и отправка его на сервер для проверки его действительности.

В противном случае, происходит обновление токена, который в последствии высылается обратно на мобильное устройство для финального входа в приложение.

Разделение ролей на мобильное приложение и обычного пользователя позволяет более подробно разобрать влияние того или иного актера на функциональные особенности системы. Все роли и их возможности приведены на рисунке 2.3.

К примеру, будучи автономным в своих действиях, на диаграмме мобильное приложение показывается как своего рода источник для пользовательского простора действий, таким образом облегчая первоначальную задачу пользователя в виде контроля за своим образом жизни. Пользователь просто пользуется предоставляемым функционалом без вникания в особенности работы приложения.

В то же самое время пользователю доступен дополнительный спектр действий на основе уже высчитанного мобильной системой характеристик.

Пользователю предоставлена база для мониторинга уровня жизни, также немаловажным является возможность конкурировать с другими пользователями, таким образом повышая градус вовлеченности в повышении физической активности.

Мобильное приложение предоставляет такие функции как подсчет дистанции, шагов, калорий и ИМТ, по заданным параметрам в пользовательском профиле.

Также, в нем можно увидеть сборник упражнений, которыми можно заменить недожженные шагами калории. Упражнения подбираются с учетом недостающих по целеполаганию пользователя шагов и ввиду этого предстает заменой недовыполненной ежедневной миссии.

По итогу, мы имеем мобильное приложение, берущее на себя основную роль пользовательского взаимодействия, а значит часть данных будет смещена на эту систему. Соответственно, частичная обработка и подгрузка данных ляжет посильным грузом на данную совокупность возможностей.

Диаграмма вариантом использования приведена в приложении В.

Было принято решение не выводить на показ в диаграмме роль сервера в этой системе взаимодействий. Поскольку в данном случае сервер является автономной единицей сбора и обработки информации.

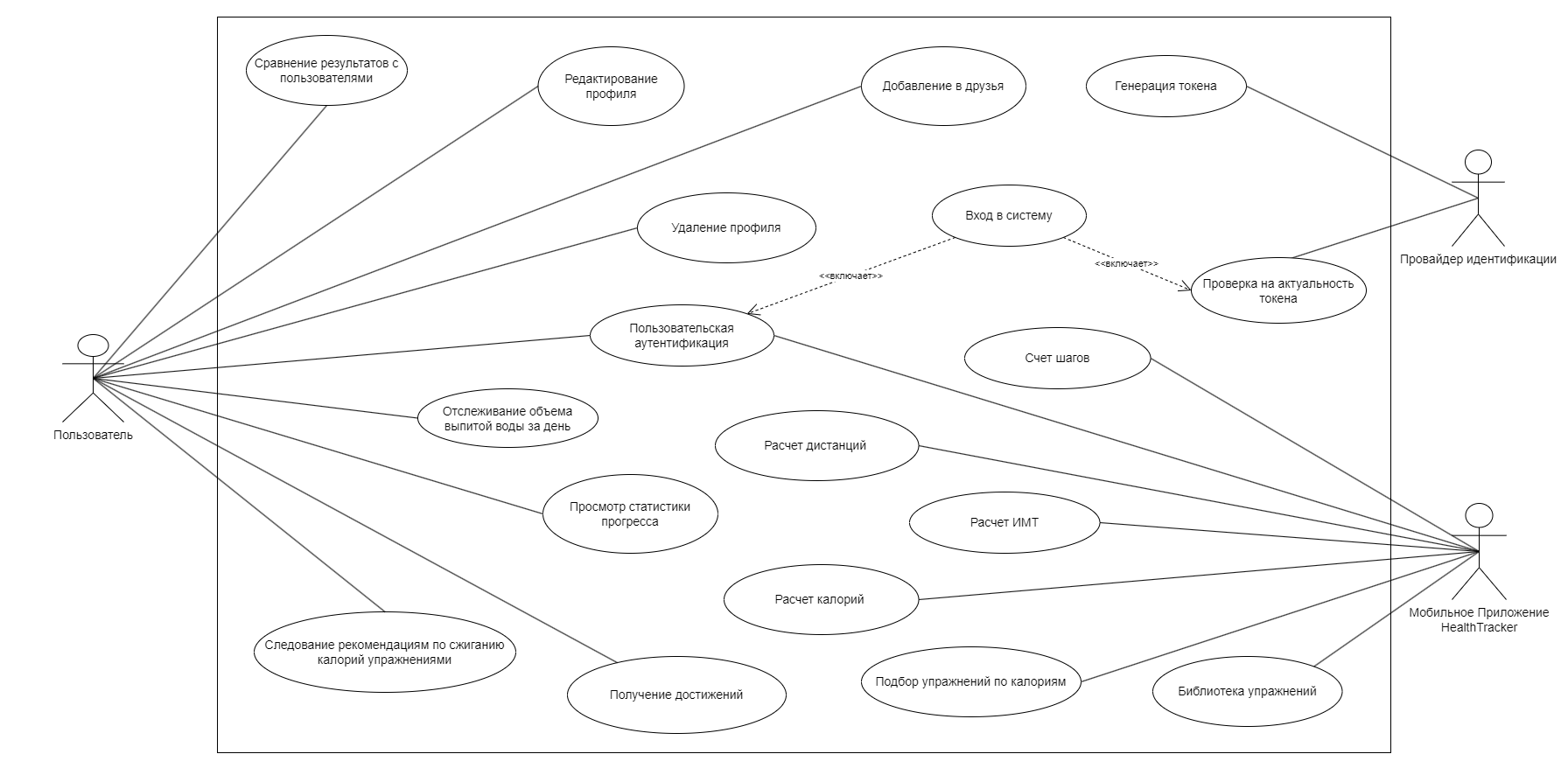


Рисунок 2.3 – Диаграмма вариантов использования

Таким образом данная диаграмма демонстрирует прямую взаимосвязь всех трех компонентов, где без проявления одного не может идти выполнение другого.

2.4 Схема переходов между экранами приложения

На этапе проектирования было решено создать примерную схему, которая покажет логику доступа к информации и функционалу приложения. Главной задачей является обеспечение интуитивно понятного доступа к искомому содержимому приложения. Целью создания наброска переходов по экранам является поиск наиболее оптимального способа отображения элементов интерфейса.

Спроектированная схема переходов между экранами мобильного приложения представлена на рисунке 2.4.

На схеме не показаны элементы, при взаимодействии с которыми не создается новый экран, поскольку это не является постраничной навигации.

Из схемы можно понять, что в первый раз экран, встречающий пользователя, является экраном входа, на котором будет изображен логотип и кнопка для перехода на следующую страницу авторизации, после чего идет переход на главный экран приложения со всем заверенным функционалом.

С главной страницы мобильного приложения можно перейти на такие экраны как «Профиль», «Достижения», «Друзья», «Прогресс сегодня» и «Шаги». Расположение доступа к ним является интуитивно понятным и не поставит пользователя в тупик при первом использовании. Каждая страница ведет к логическому продолжению того, что от нее ожидается.

Например, на экране «Профиль» пользователь сможет просмотреть не только информацию об аккаунте, такую как почта, заставка и т.д., но, а также воспользоваться настройками для установки целей и прочих характеристик.

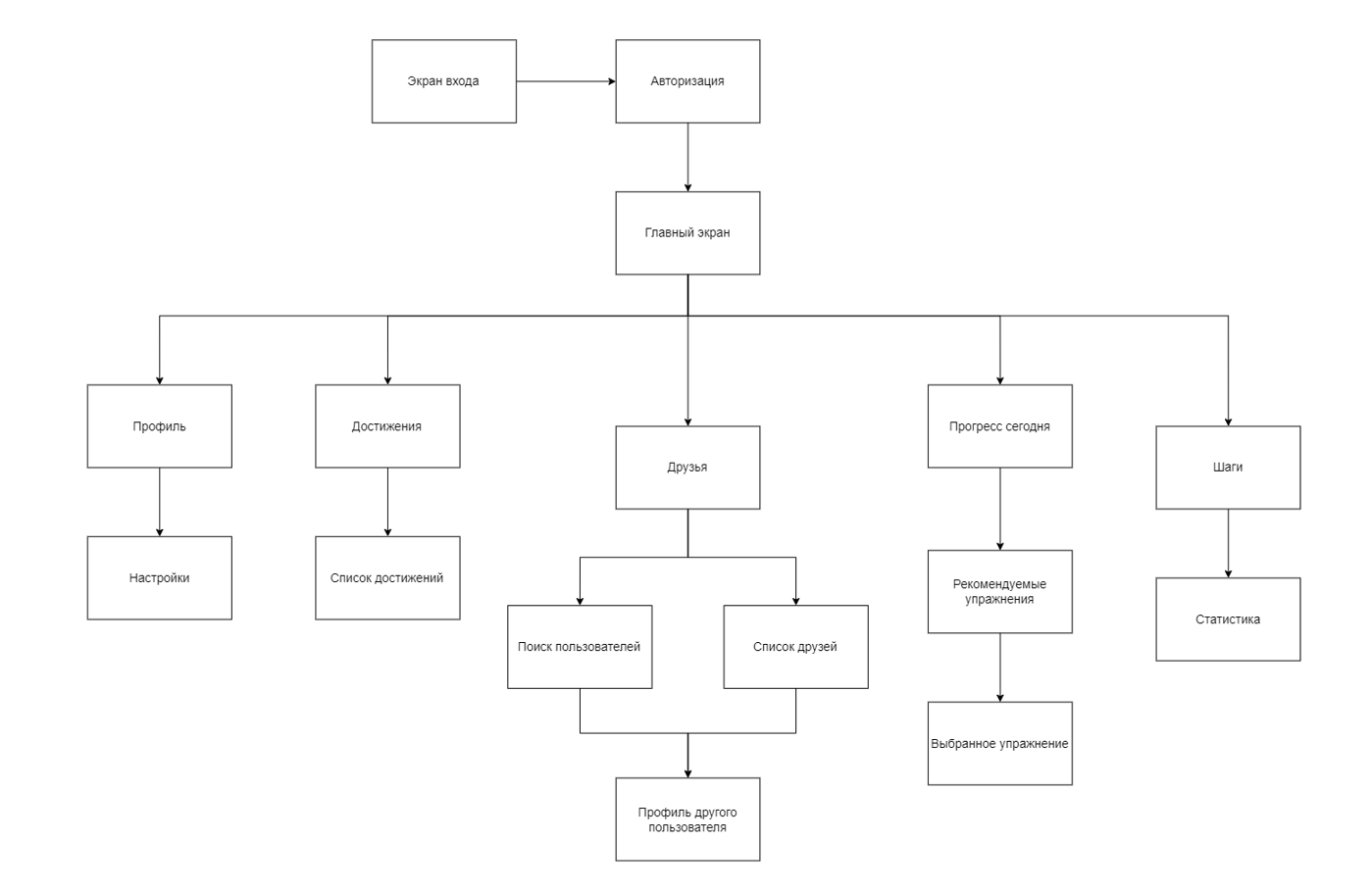


Рисунок 2.4 – Схема переходов между экранами приложения

«Достижения» ведут к спискам достижений пользователя. «Друзья» ведут к двум экранам поиска и списка имеющихся друзей, после взаимодействия с которыми, откроется их профиля. При нажатии на «Прогресс сегодня» пользователю будет предоставлен список рекомендуемых упражнений, при выборе одного из которых, откроется новый экран с описанием упражнения. «Шаги» же приведут к подробной статистике за различные промежутки времени.

Таким образом, схема дает упрощенное понимание действий и делает ожидаемым поведение приложения при взаимодействии с ним.

## 2.5 Блок-схема алгоритма получения достижений

Блок-схема алгоритмов – это визуальное представление шагов, которые необходимо выполнить для решения проблемы. Она используется для планирования, проектирования, документирования и передачи логики программы или процесса. Это помогает разбить процесс на управляемые этапы и дает четкое понимание того, что необходимо сделать на каждом этапе.

В качестве такого представление было решено показать алгоритм получения достижений, который можно увидеть в приложении Г.

Данный алгоритм работает благодаря триггерам на таблицу step, поскольку основные достижения будут связаны с результатами по пройденным шагам. Из-за бесконечности изменяющихся значений простым решением было построить алгоритм на триггерах, облегчающих своим механизмом работу системы.

На рисунке 2.5 изображена часть полного алгоритма, показывающая как учитываются данные по шагам, дням и дистанции для получения достижения.

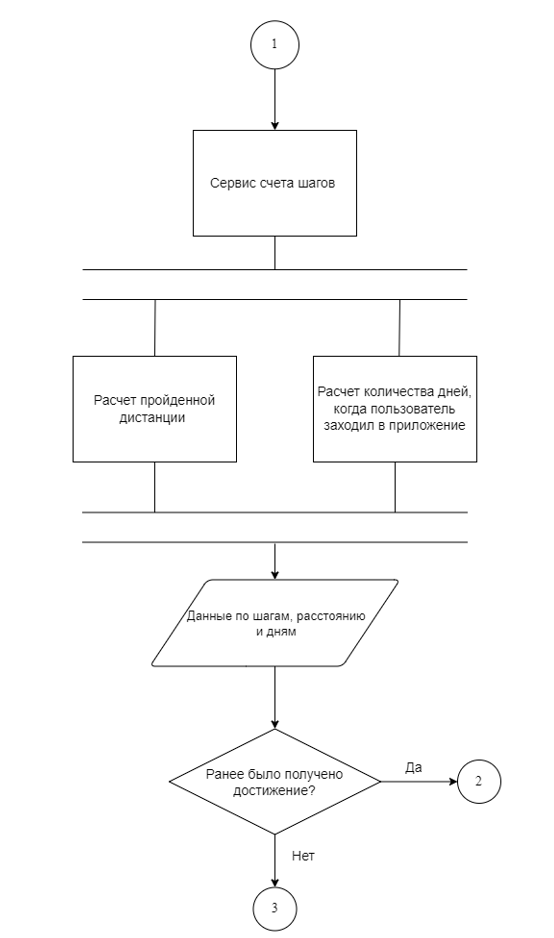


Рисунок 2.5 – Часть блок-схемы алгоритма получения достижений

Для работы алгоритма необходимо иметь данные о шагах пользователя, значения которых влияли бы на запуск методики сравнения полученных результатов с тем, что лежит в значениях достижений.

Если же данные совпадают, то в итоге пользователю начисляется заслуженное достижение. Иначе стабильный подсчет шагов продолжит свою работу без задействования метода.

## 2.6 Выводы по разделу

В данной главе была разработана диаграмма вариантов использования мобильного приложения, которая позволила составить более целостное понимание всех функциональных возможностей проекта.

Была спроектирована база данных, где определены необходимые таблицы, поля для каждой из них, типы данных и ограничения целостности. Также были установлены связи между таблицами, которые описаны на схеме базы данных.

Помимо этого, были представлены основные диаграммы, полно определяющие задумку проекта, что позволяет быстро и наглядно вникнуть в суть продукта.

3 Разработка приложения

ФИО

Подпись

Дата

Лист

1

БГТУ 03.00.ПЗ

Разраб.

*Лисункова А.Д.*

Пров.

*Бернацкий П.В.*

Н. контр.

*Нистюк О.А.*

Утв.

Смелов В.В.

3 Разработка приложения

Лит.

Листов

15

*74217076, 2023*

3.1 Реализация серверной части

В процессе разработки серверной части было решено разгруппировать по пакетам классы схожего назначения для удобной навигации в проекте, что привело бы к быстрому изменению или добавлению новых компонентов приложения.

Пакетная структура представлена на рисунке 3.1.

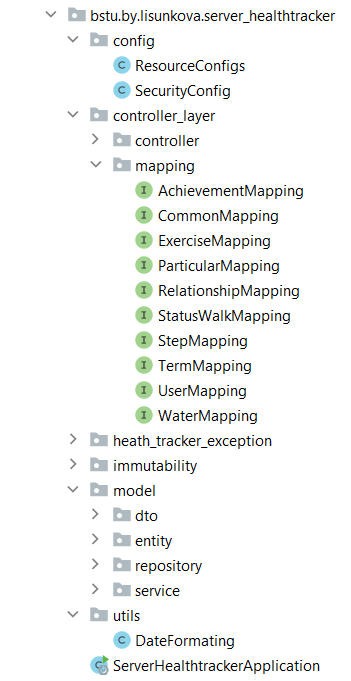


Рисунок 3.1 – Пакетная структура сервера

Обособленно расположенный класс ServerHealthtrackerApplication является

точкой входа приложения Spring Boot и содержит метод main, благодаря такой исключительности вынесен отдельно ото всех логичных блоков объединения.

В пакете config расположены классы, предназначенные для настройки приложения, такие как SecurityConfig и ResourceConfig. Класс SecurityConfig содержит логику аутентификации и авторизации пользователей. В классе ResourceConfig указан список статических ресурсов, к которому имеет доступ мобильное приложение.

В пакете controller\_layer реализована логика маршрутизации запросов по сети, а также преобразование данных в вид, удобоваримый для передачи. Эти обязанности разделены по двум пакетам – controller и mapping. Соответственно, в controller располагаются контроллеры сервера, к конечным точкам которых пользователь будет делать запросы для получения данных. Пакет mapper содержит интерфейсы с описанием логики преобразования объектов сущности в объекты передачи данных (Data Transfer Object) при помощи библиотеки mapstruct.

Пакет model содержит классы, относящиеся к уровню модели приложения. В пакете dto расположены классы передачи данных. Пакет entity содержит классы, отображающие данные в СУБД. Пакет repository содержит интерфейсы, описывающие запросы к базе данных. В пакете service находятся классы-сервисы, в которых содержится бизнес-логика приложения.

3.1.1 Уровень серверной модели

3.1.1.1 База данных на серверной части

Как оговаривалось в разделе с проектированием, СУБД является сторонним компонентом разработки целой системы. Поскольку применялся подход, который по коду создает структуру в самой базе данных, то описание разработки классов сущностей будет рассмотрено в следующем подпункте.

3.1.1.2 Табличные сущности на сервере

В пакете entity находятся классы, описывающие структуру таблиц в серверной базе данных, а именно MS SQL Server. Данные классы помечены специальными аннотациями библиотеки Spring Data JPA, которые позволяют определить какую табличную сущность описывает класс, в том числе устанавливают соответствие между полями таблицы и класса, указывают связь между сущностями.

Основной аннотацией, которой были помечены классы, определяющие табличную сущность, является @Entity. Она задает связь с нужной таблицей с помощью аргумента name. Аннотация @Column использовалась для пометки поля и указания имени столбца таблицы, с которым будет ассоциироваться поле.

Аннотацией @Id помечалось поле или поля, как в случае с составными ключами, в качестве первичного ключа сущности. В ситуации, когда таблица имеет составной первичный ключ описание частей ключа выносились в отдельный класс, становившимся аргументом аннотации @IdClass, которой пометилась цельная таблица.

Аннотация @Basic была у тех полей, которым соответствуют базовые типы данных. Это дает понять Hibernate (библиотека объектно-ориентированного отображения), что следует использовать стандартный маппинг для их представления. Аннотации @OneToMany, @ManyToOne, @ManyToMany описывали отношения между сущностями, вместе с ними шла аннотация @JoinColumn, присоединяющая по имени поля, ту сущность, на которую ссылается внешний ключ.

Кроме аннотаций Spring Data JPA для классов хранения данных использовались аннотации библиотеки Lombok, которые за счет автогенерации методов и полей позволяют сократить код. К таким относятся @Data, которая объединяет в себе такие аннотации как @Getter, @Setter, @ToString, @EqualsAndHashCode и т.д. таким образом, она позволяет сгенерировать шаблонный код для конструктора как со всеми аргументами, так и без аргументов, с геттерами, сеттерами всех полей класса и также методами toString(), equals(), hashCode().

Фрагмент кода класса сущности WaterLevel приведен в листинге 3.1

|  |
| --- |
| @Entity  @Data @IdClass(WaterLevelPK.class) public class WaterLevel extends CommonEntity{  @Id  @Column(name = "data\_id")  private Date dataId;   @Id  @Column(name = "user\_id")  private String userId;   @Basic  @Column(name = "amount\_plan")  private Integer amountPlan;   @Basic  @Column(name = "amount\_fact")  private Integer amountFact;   @Basic  @Column(name = "measure")  private String measure;   @ManyToOne  @JoinColumn(name = "user\_id", referencedColumnName = "id",  insertable = false,updatable = false,nullable = false)  private User user; |

Листинг 3.1 – Сущность WaterLevel

В пакете repository располагаются интерфейсы, описывающие запросы к базе данных. Классы помечены аннотацией @Repository. Такая аннотация является частным случаем аннотации @Component, используемой для создания “бина” – объекта, образуемого контейнером Spring-а. Благодаря данной технологии заниматься такими вещами, как создание экземпляра, его инициализация, внедрение зависимостей и генерацией оберток над бином, будет не код, а контейнер Spring-a.

Сами интерфейсы позволяют уйти от деталей реализации доступа к базе данных. Была задействована имплементация интерфейса JpaRepository, принимающего два параметра – класс сущности и тип данных первичного ключа. Данный интерфейс имеет особенность строить запросы к сущности прямо из имени метода.

Для этого используется механизм аффиксов «find…By...», «read…By…», «query…By…». Первый By действует как разделитель, чтобы указать начало фактических критериев. Таких разделителей может быть несколько.

Пример автоматического создания запроса по имени метода с помощью механизма аффиксов показан в листинге 3.2.

|  |
| --- |
| @NoRepositoryBean public interface IRepository<E extends CommonEntity> extends JpaRepository<E,String> {  @Override  List<E>findAll(); } |

Листинг 3.2 – Создание запроса по наименованию метода

Здесь аннотация @NoRepositoryBean является пометкой к интерфейсу, бин или, иначе говоря, его экземпляр не будет генерироваться Spring, так как данный интерфейс объединяет в себе общие характеристики для других интерфейсов. Аннотация @Query применялась для создания JPA и SQL запросов. В качестве дополнения к ней использовались аннотации @Transactional и @Modifying, если в запросе идет обращение к процедуре, которая подразумевает обработку нескольких запросов, от чего данные аннотации становятся незаменимыми в коде проекта. Пример использования приведен на листинге 3.3 и полная версия репозитория приведена в приложении Ж.

|  |
| --- |
| @Modifying @Transactional @Query (nativeQuery = true, value = "exec achievement\_for\_user\_add :user\_id,:achievement\_id")  void addAchievementToUser(@Param("user\_id") String user,@Param("achievement\_id") String achievement); } |

Листинг 3.3 – Метод добавления достижения пользователю

В пакете dto находятся классы, задача которых быть конечным отображением сущностей при передаче данных между мобильным клиентом и сервером. В них отсутствуют зависимости, лишь содержится необходимая информация для отправки клиенту. Логика трансформации объектов сущностей в объекты передачи данных, и обратно, описана в интерфейсах пакета mapper.

В пакете service содержатся классы с бизнес-логикой серверной части приложения. Они имеют ссылки на необходимые им репозитории и других нужных компонентов, получаемых с помощью DI (Dependency Injection), реализованной путем применения аннотации @Autowired. Она указывает куда непосредственно нужно вложить сгенерируемый бин. В листинге 3.4 представлен код метода обновление данных в таблице шагов в базе данных.

|  |
| --- |
| public Step updateStep(Step step) { List<Step> steps = repository.findStepsByUserIdAndAndDateId(step.getUserId(), step.getDateId().toString());  Step stepFromDb = steps.stream().findFirst().orElse(null);  if (stepFromDb != null) {  if(step.getFactSteps() != null)  stepFromDb.setFactSteps(step.getFactSteps());  if(step.getPlanSteps() != null)  stepFromDb.setPlanSteps(step.getPlanSteps());  stepFromDb.setTerm(step.getTerm());  } else {  stepFromDb = new Step();  stepFromDb.setUserId(step.getUserId());  stepFromDb.setDateId(step.getDateId());  stepFromDb.setUser(step.getUser());  stepFromDb.setFactSteps(step.getFactSteps());  stepFromDb.setPlanSteps(step.getPlanSteps());  stepFromDb.setTerm(step.getTerm());  }  return repository.save(stepFromDb); } |

Листинг 3.4 – Метод обновления данных по шагам

Метод принимает аргументом сущность класса Step, которая после передается в метод контроллера. Для того, чтобы получить сущность из таблицы базы данных, потребовалось вызвать метод из репозитория, помеченный аннотацией @Autowired. Из-за чего подставился экземпляр репозитория без задействования конструктора для его создания. После некоторых манипуляций с объектом типа данных Step, он возвращается в контроллер.

3.1.2 Уровень сетевого взаимодействия серверного приложения

В подкаталоге mapper пакета controller\_layer находятся интерфейсы с аннотацией @Mapper библиотеки mapstruct. Они предназначены для преобразования объектов табличных сущностей в объекты передачи данных, и наоборот.

В качестве примера на рассмотрение возьмем интерфейс StepMapping, который преобразует сущность Step в объект StepDto, и обратно. В интерфейсе содержится два основных метода dtoToEntity и entityToDto. В процессе компиляции сгенерируются классы, которые реализуют методы интерфейса, таким образом позволив уменьшить строки однотипного кода, если бы тот писался вручную.

С помощью параметра uses аннотации @Mapper ссылаемся на интерфейс для маппинга другого класса – UserMapping, который задействуется в преобразованиях текущего интерфейса, ввиду установленной связи между сущностями.

Применение данной аннотации позволяет задавать особые правила преобразований объектов. Например, к тем методам, которые после компиляции будут сгенерированы, добавить аннотацию @Mapping, в которой нужно указать источник данных, получателя и параметр qualifiedByName, что определяет имя метода, дополнительно преобразующего поля конвертируемых классов.

Такое применение возможности mapper-а необходимо в текущем случае, где класс сущности и класс преобразования не копируют досконально поля друг друга. Клиенту не во всех ситуациях нужны в полном объеме данные, как правило ему из сервера отправляют измененную урезанную информацию, с которой он дальше работает без прибегания к ее модификации.

Пример интерфейса маппинга шагов приведен на листинге 3.5

|  |
| --- |
| @Mapper(componentModel = "spring", uses = {UserMapping.class}) public interface StepMapping extends ParticularMapping<Step, StepDto> {  @Mapping(target = "user", source = "dto.user", qualifiedByName = "toUser")  Step dtoToEntity(Step dto);   @Mapping(target = "user", source = "entity.user", qualifiedByName = "toUserId")  Step entityToDto(Step entity);  @Named("toUser")  default User toEntityUser(String userId){  User user = new User();  user.setId(userId);  return user;  } @Named("toUserId")  default String toDtoUser(User user){  return user.getId();  }  } |

Листинг 3.5 – интерфейс StepMapping

Интерфейс расширяется кастомным интерфейсом PatricularMapping, в котором приведена дефолтная реализация методов преобразования. Аннотацией @Named помечены методы, имена которых упоминались в параметре qualifiedByName. Они занимаются трансформацией несовпадающих полей двух классов.

Код интерфейса ParticularMapping продемонстрирован в приложении И.

В папке controller\_layer.controller содержатся REST-контроллеры, помеченные соответствующей аннотацией.

Рассмотрим класс RelationshipController, предназначенный для работы с созданием и изменением отношений между пользователями.

Он помечается двумя аннотациями, первая @RestController для обозначения класса, использующего REST-методы, вторая - @RequestMapping, определяющая базовый адрес контроллера. Значения для четырех помеченных аннотацией @Autowired полей внедряются с помощью Dependency Injection.

В классе присутствуют четыре метода, два из которых будут вызываться с запросом http-метода GET, один – методом POST, и еще один с методом DELETE. Все это также помечено соответствующими аннотациями, значением которых является адрес для вызова метода. @RequestParam применяется для помечания переменной, в которую будет извлекаться параметр запроса. Часть кода класса RelationshipController содержится в листинге 3.6, полная версия в приложении К.

|  |
| --- |
| @RestController @RequestMapping("relationship") public class RelationshipController {  @Autowired  RelationshipMapping relationshipMapping;  @Autowired  RelationService relationshipService;  @Autowired  UserMapping userMapping;  @Autowired  UserService userService;  @GetMapping("/user\_friends")  public List<UserDto> getFriendsUser(){  String currentUserId = SecurityContextHolder.getContext().getAuthentication().getName();  User user = userService.findById(currentUserId).get();  List<User> users = relationshipS ervice.friendsAtUser(user.getId());  return userMapping.entitiesToDtoes(users);  }  @PostMapping("/add\_friend")  public String addFriend(@RequestParam String friendId){  String currentUserId = SecurityContextHolder.getContext().getAuthentication().getName();  User user = userService.findById(currentUserId).get();  String statusResponse = relationshipService.addingShipBetweenUsers(user.getId(), friendId);  return statusResponse;  } } |

Листинг 3.6 – Класс RelationshipController

Здесь внедряются помимо непосредственных компонентов, относящихся к сущности Relationship, но также зависимые User-компоненты.

Методы работы с информацией помечены аннотациями маппинга по методу взаимодействия с данными. Все это сопровождается помечанием аннотацией, создающей бин данных классов и интерфейсов. С помощью класса SecurityContextHolder получаем контекст, через который связываемся с аутентифицированным профилем и достаем из него личную информацию.

Таким образом, получив текущего пользователя из запроса, контроллер передает данные на обработку сервисам в формате, создаваемом интерфейсами маппинга.

Всего было разработано 7 REST-контроллеров, в которые суммарно входят 21 метод. Методы и их описания приведены в таблице 3.1.

Таблица 3.1 – Описание методов REST API

|  |  |  |
| --- | --- | --- |
| Адрес | Метод | Описание |
| /authcode | POST | Проверить/сгенерировать токена для входа пользователя |
| /achieve/activating\_achieves | GET | Получить список активированных достижений у пользователя |
| /achieve/add\_achieves | POST | Добавить достижение |
| /achieve/all\_achievments\_at\_user | GET | Получить все достижения вне зависимости от активации |
| /exercises | GET | Получить упражнения |
| /relationship/user\_friends | GET | Получить список друзей |
| /relationship/waiting\_users | GET | Получить список заявок на дружбу |
| /relationship/add\_friend | POST | Добавить друга |
| /relationship/delete\_friend | DELETE | Удалить друга |
| /steps | POST | Добавить запись о шагах |
| /steps/comparison | GET | Сравнить результаты с пользователями |
| /steps/fact\_steps/{period} | GET | Получить фактические шаги за определенный период |
| /steps/statistic/{period} | GET | Получить статистику за период времени |
| /steps/from\_step | PUT | Обновить данные по шагам |
| /user | GET | Получить данные о пользователе |
| /user/view | GET | Получить чужую страницу |
| /user/searching | GET | Искать пользователей |
| /user/update | PUT | Обновить данные о пользователе |
| /user/deactivate | DELETE | Удалить аккаунт пользователя |
| /water | POST | Добавить информацию о выпитой воде за текущий день |
| /water/update | PUT | Обновить данные о количестве выпитой воды |

Таким образом, клиент зная об адресах запросов может получить доступ к любой информации на сервере, будучи авторизованным.

3.1.3 Конфигурация безопасности приложения

Класс SecurityConfig пакета config содержит в себе настройки сообщения с сетью и описывает дозволенные адреса, с которыми может взаимодействовать пользователь. Данный класс наследуется от WebSecurityConfigurerAdapter – класс библиотеки Spring Security, предназначенный для настройки безопасности приложения.

С помощью аннотации @EnableGlobalMethodSecurity применяются настройки на весь проект. В переопределенном методе configure задаются настройки безопасности с помощью паттерна строитель, в котором задаются адреса и методы запросов, доступных при определенных условиях, настройка аутентификации и т.д.

Метод antMatchers() задает паттерн адресов, доступных для определенных ролей, а также для состояний, в которых пребывают роли, например, существует метод проверки как fullyAuthenticated(), дозволяющий тем, кто прошел проверку подлинности, использовать предоставленные сервером методы.

Также работа с сессией происходит с помощью метода sessionManagement(), настраивающего ее режим создания. В довершении конфигурации задается протокол OAuth 2, позволяющий выдать одному сервису(приложению) права на доступ к ресурсам. Дальше протокол настраивается на использование JWT токенов.

3.1.4 Технологии, используемые в серверной части приложения

Для реализации серверной части программного средства были использованы технологии, представленные в таблице 3.2.

Таблица 3.2 – Технологии серверной части

|  |  |
| --- | --- |
| Название | Назначение |
| Spring Framework | Фреймворк для облегчения написания кода в основном на языке Java. |
| Spring Boot | Фреймворк, созданный поверх Spring, предназначенный для упрощения и оптимизирования создания веб-приложений. |
| Spring Security | Среда безопасности для приложений на основе Java. |
| MapStruct | Библиотека, упрощающая сопоставление между различными компонентами Java и POJO. |
| Язык Java | Объектно-ориентированный язык программирования. |
| Spring Security OAuth 2.0 | Библиотека для реализации аутентификации и авторизации на основе OAuth 2.0. |

Совокупная работа всех этих технологий привело к желаемому результату. Большая часть из них встраивалась в проект посредством внедрения зависимостей в конфигурационный файл pom.xml.

3.2 Разработка мобильного приложения

Файловая структура распределена по пакетам на основе их назначения для облегчения ориентирования в каталогах. В корне каталогов расположены папки data, service, ui и класс с точкой входа в приложение MainActivity. Структура пакетов проекта приведена на рисунке 3.2.

Библиотека Jetpack Compose была применена в разработке пользовательского интерфейса, позволив заменить доселе привычную xml-разметку на универсальный декларативный стиль. В качестве средства, связывающего данные, хранимые на клиентском приложении, и на удаленном сервере, выступает библиотека Retrofit. Данные мобильного устройства хранятся в SharedPreferences в JSON-формате. Часть информации хранится в кеше для доступа к ней при отсутствии подключения к сети.

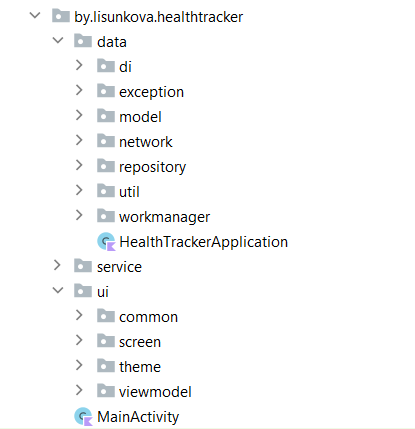


Рисунок 3.2 – Структура пакетов мобильного приложения

Содержимое каждого пакета подробно рассмотрено в последующих пунктах данного раздела.

3.2.1 Разработка уровня бизнес-логики мобильного приложения

Пакет data содержит каталоги для работы с данными и их отображением на ui, видоизменение при передаче по сети. Так в пакете model содержатся данные, полученные по запросам Retrofit из серверного приложения. Они содержат в себе лишь конструктор с полями помеченные аннотацией @SerializedName, значение которой несет в себе название полученного результата запроса.

Пример класса модели данных User в листинге 3.7.

|  |
| --- |
| @Stable data class User(  @SerializedName("id") val id: String,  @SerializedName("email") val email: String,  @SerializedName("firstName") val firstName: String,  @SerializedName("lastName") val lastName: String,  @SerializedName("photo") val photoUrl: String,  @SerializedName("height") val height: Float,  @SerializedName("weight") val weight: Float,  @SerializedName("dailyGoalSteps") val stepsGoal: Int,  @SerializedName("dailyGoalWater") val waterGoal: Float, ) {  val stepLength: Float  {  \*\*\*  } } |

Листинг 3.7 – модель данных User

Здесь имеется дополнительный метод для вычисления длины шага исходя из поступивших значений параметров, что в дальнейшем будет учитываться в функции определения дистанции. Аннотация @Stable использовалась для передачи компилятору Compose некоторых гарантий относительно того, как будет вести себя определенная функция. В данном случае класс содержит метод, логику которого будет учитывать при создании интерфейса Compose.

В папке network находятся интерфейсы с суффиксом Service, выполняющие запросы на сервер. В них задействована библиотека Retrofit, поэтому в интерфейсах методы помечаются аннотациями соответствующих REST-методов. Все эти методы являются suspend-функциями, так как запросы в сети являются относительно долгими операциями, прерыванию, которых способствует suspend.

Пример части интерфейса действий с профилем приведен на листинге 3.8

|  |
| --- |
| interface ProfileService {  @POST(“authcode”)  suspend fun signIn(  @Body idToken: RequestBody,  @Header(NO\_AUTH\_HEADER\_KEY) noAuth: String = NO\_AUTH\_HEADER\_KEY  ): Response<User>   @GET(“user”)  suspend fun getCurrentUser(): Response<User>   @POST(“user/partly\_update”)  suspend fun updateUser(  @Query(“weight”) weight: Float,  @Query(“height”) height: Float,  @Query(“plan\_step”) stepsGoal: Int,  @Query(“plan\_water”) waterGoal: Float,  ): Response<Unit> } |

Листинг 3.8 – интерфейс ProfileService

Каталог repository включает в себя классы, содержащие основную бизнес-логику мобильного приложения и соответствующие компоненту Repository уровня Model архитектуры MVVM приложений Android.

Пример ProfileRepository и StepRepository приведены в приложениях Л и М.

3.2.2 Разработка пользовательского интерфейса

Для разработки пользовательского интерфейса применялась современная библиотека Jetpack Compose. Она является на данный момент весомой библиотекой в Android-разработке, которой отдают предпочтение при написании кода в декларативном стиле. Она представляет собой набор инструментов для создания нативного пользовательского интерфейса, упрощающего и ускоряющего разработку под Android систему. В проекте использована версия 1.4.0 библиотеки, поскольку на момент разработки она являлась самой обновленной стабильной версией.

В классе MainActivity при запуске приложения в переопределенном методе onCreate() происходит обращение к compose-функции, создающую начальный экран. В корневом подкаталоге ui находятся пакеты common, screen, theme и viewmodel, каждый из которых в зависимости от назначения содержат экраны, модели представления, темы и другие дополняющие компоненты.

Пакет common содержит файлы с общими composable-функциями, которые часто используются на разных экранах. В пакете util располагаются файлы со вспомогательными классами и функциями для упрощения и некоторой оптимизации кода в пользовательском интерфейсе.

Следующим рассмотрим метод onCreate в классе MainActivity. Код данного метода приведен в листинге 3.9.

|  |
| --- |
| override fun onCreate(savedInstanceState: Bundle?) {  super.onCreate(savedInstanceState)  lifecycleScope.launch {  profileViewModel.currentUser.collect {  \*\*\*  }  }   setContent {  HealthTrackerTheme {  MainScreen(profileViewModel)  }  } } |

Листинг 3.9 – Метод onCreate в MainActivity

Тут можно найти примечательное отличие от содержимого метода onCreate Android-приложения, основанного на системе View, то есть вместо вызова метода setContentView с передачей идентификатора ресурса xml-файла разметки в данном случае вызывается setContent. Здесь можно также наблюдать одну из особенностей Kotlin, а именно вынос последнего аргумента – лямбда-функцию за круглые скобки. То есть вид вызова метода setContent{…} соответствует вызову setContent({…}).

Метод setContent предназначен для передачи разметки в активность и имеет параметр @Composable -> Unit, что означает функцию, не принимающую аргументов и ничего не возвращающая, и в том числе помеченная аннотацией @Composable. Данная аннотация является принципиально важной для библиотеки Jetpack Compose, так как ей помечаются функции, в которых описывается декларативно пользовательских интерфейс. Такие функции не могут быть вызваны из обычных методов, только из таких же composable-функций.

В setContent передается функция HealthTrackerTheme, задающая цветовую тему приложения с помощью composable-функции Material-Theme. Она реализует механизм, позволяющий обращаться к теме из любого места. Поскольку тема в HealthTrackerTheme вызывается из MainScreen, содержащий весь пользовательский интерфейс приложения, то тема применится ко всему приложению.

Далее приведем пример composable-функции, задающей бизнес-логику начисления достижений пользователю. Фрагмент кода composable-функции AchievementList из файла AchievementScreen представлен в листинге 3.10.

|  |
| --- |
| @Composable private fun AchievementList(  achievements: List<Achievement>,  label: String,  token: String?,  isExpandedInitial: Boolean = false ) {  var isExpanded by remember { mutableStateOf(isExpandedInitial) }   HealthTrackerCard {  Row(  horizontalArrangement = Arrangement.SpaceBetween,  verticalAlignment = Alignment.CenterVertically,  modifier = Modifier.fillMaxWidth()  ) {  Text(  text = label,  style = MaterialTheme.typography.h4  )   IconButton(onClick = { isExpanded = !isExpanded }) {  Icon(  painter = painterResource(id = R.drawable.ic\_chevrone),  contentDescription = "edit",  modifier = Modifier.rotate(if (isExpanded) 180f else 0f)  )  }  }  \*\*\*  } |

Листинг 3.10 – Фрагмент кода из метода AchievementList

Здесь, как и все composable-функции AchievementList помечается аннотацией @Composable, вдобавок принимает 4 параметра: лист достижений, название достижения, токен активации текущего юзера и значение булевой переменной, которая показывает был ли раскрыт список. Далее в теле самой функции создаем переменную isExpanded, в которую передаем значение аргумента isExpandedInitial через делегат by remember. Он позволяет создать объект один раз при первом запуске composable-функции и получать его экземпляр во всех последующих перезапусках. Помимо этого, вызывается функция mutableStateOf. Она создает неявные геттер и сеттер, позволяющие уведомлять Compose об обновлении значения, но взаимодействовать с полем мы не сможем как с обычной переменной указанного типа.

Переходим к визуализации интерфейса. Далее можно увидеть функции, имя которых являются отображением элементов экрана, также применяющихся и в xml-разметке. Но в данном варианте, благодаря вложенности функций и описанию определенного поведения, можно построить структуру для отображения пользовательского интерфейса. Обратим внимание, что данные функции также являются composable. Так, например, composable-функция Row создана для отображения заключенных в ней также же как она функций в виде строки.

Рассмотрим фрагмент класса модели представления FriendsViewModel, отделяющей бизнес-логику от представления, и наоборот (листинг 3.11).

|  |
| --- |
| @HiltViewModel class FriendsViewModel @Inject constructor(  private val friendsRepository: FriendsRepository ): ViewModel() {  val friends: MutableStateFlow<Status<List<User>>> = MutableStateFlow(Status.Success(emptyList()))  val incomingRequests: MutableStateFlow<Status<List<User>>> = MutableStateFlow(Status.Success(emptyList()))  val friendsFound: MutableStateFlow<Status<List<User>>> = MutableStateFlow(Status.Success(emptyList()))   fun getFriends() {  viewModelScope.launch(Dispatchers.IO) {  val result = friendsRepository.getFriends()  if (result is Status.Success)  friends.emit(result)  }  suspend fun getUserAndComparison(id: String): Pair<Status<User?>, Status<List<List<Steps>>>> = withContext(Dispatchers.IO) {  val user = async { friendsRepository.getUserById(id) }  val comparisons = async { friendsRepository.compareStepsById(id) }  user.await() to comparisons.await()  }  } |

Листинг 3.11 – Фрагмент класса FriendsViewModel

Аннотация @HiltViewModel является частью Hilt – библиотеки под Android и Compose, позволяющая внедрять зависимости. В месте создания модели представления не передается репозиторий, а просто прописывается функция hiltViewModel() в файлах каталога screen. Полный код класса в приложении Н.

В папке data.di класса AppModules описан граф зависимостей, это в будущем позволило определить жизненный цикл модели представления, которые по дефолту уничтожаются вместе с экраном, к которому привязаны. Но в текущем проекте такое поведение не подходило – надо, чтобы модели представления не пересоздавались. И поэтому в классе MainScreen определен NavGraph, как вложенный граф, в котором есть все файлы каталога screen, за исключением пары штук. Таким образом, модели представления пересоздаются только при повторном входе в аккаунт.

Что касается непосредственно примера в листинге 3.11 в класс внедряется репозиторий с друзьями. А после определяются переменные определенного назначения, и содержащие ту информацию, которая удовлетворяет смыслу их создания.

В итоге мы получаем списки с данными о поступающих заявках на дружбу, о результатах поиска пользователей и список имеющихся друзей.

Во фрагменте задействованы две функции: getFriends и getUserAndComparison. Последняя помечена словом suspend, что означает возможность запуска такой функции либо из такой же suspend-функции, либо внутри CoroutineScope.

Здесь применяется supend-функция withContext, которая вызывает блок кода с заданным контекстом, и приостанавливает до тех пор, пока он не завершится, и после возвращает результат.

Переданный в параметр Dispatchers.IO указывает на то, что внутренний блок кода будет запущен в потоке ввода/вывода, так как подобные функции зачастую являются долгоиграющими, за счет чего сильно перегружают вычислительные возможности системы. Внутри выполняется асинхронный код, конца работы которого дожидается withContext, а после возвращает результат.

В функции getFriends запускается с помощью вызова функции launch на представлении CoroutineScope – viewModelScope. Это значит, что выполняемые в этом контексте suspend-функции будут отменены при разрушении модели представления, что позволяет избежать утечек памяти.

3.3 Выводы по разделу

В соответствии со спроектированной ранее архитектурой, разработаны:

– серверная часть проекта, взаимодействующая с мобильным приложением;

– клиентская часть (мобильное приложение для устройств, работающих под управлением операционной системы Android).

Были подробно описаны используемые компоненты, необходимые для работы приложения, а также особенности их реализации. Итогом выполнения разработки стал программный продукт для оценки уровня активности. Пользовательский интерфейс разработанного мобильного приложения приведен в приложении Д.

Также приведён этап разработки серверного приложения, была рассмотрена структура проекта, использование технологий Spring Framework, Boot, Security и библиотека MapStruct. Рассмотрена структура проекта, а также описаны методы контроллеров. Приведена разработка клиентской части, где была рассмотрена архитектура приложения, этапы создания, обращение к серверной части, а также был представлен потенциал Android приложения на Kotlin.

4 Анализ информационной безопасности приложения

ФИО

Подпись

Дата

Лист

1

БГТУ 04.00.ПЗ

Разраб.

*Лисункова А.Д.*

Пров.

*Бернацкий П.В.*

Н. контр.

*Нистюк О.А.*

Утв.

Смелов В.В.

4 Анализ информационной безопасности приложения

Лит.

Листов

4

*74217076, 2023*

Информационная безопасность системы – это свойство информационной системы или реализуемого в ней процесса, характеризующее способность обеспечить необходимый уровень своей защиты [4].

Разработанное приложение состоит из следующих компонентов: мобильный клиент, сервер, база данных сервера. Каждый из компонентов приложения может иметь уязвимости, которые необходимо ликвидировать.

4.1 Возможные угрозы безопасности приложения

К уязвимостям серверной части можно отнести отсутствие какого-либо метода авторизации, у злоумышленников появляется доступ к данным, которые хранятся в базе данных через общедоступное API.

Потенциальной угрозой также может являться перехват трафика по незащищенному соединению.

Со стороны мобильного приложения существует угроза получения доступа к данным, а также получения исходного кода путем декомпиляции байткода dex-файлов собранного apk приложения.

4.2 Реализованные методы защиты приложения

4.2.1 Использование протокола HTTPS

Для защиты передаваемых данных в сети между клиентом и сервером, в приложении используется протокол HTTPS. Благодаря использованию протокола TLS исключается возможность получения передаваемого трафика третьими лицами. TLS (Transport Layer Security) – это протокол транспортного уровня, который обеспечивает взаимную аутентификацию сторон протокола, конфиденциальность и контроль целостности данных, передаваемых между сторонами на транспортном коммуникационном уровне. TLS использует асимметричное шифрование для аутентификации и симметричное шифрование для конфиденциальности [5].

Данный тип защиты предоставляется в полной мере облачной платформой Heroku, сертифицирующей развернутое приложение.

4.2.2 Механизм аутентификации

Для входа в приложение используется учетная запись Google. Google API использует протокол OAuth 2.0 для аутентификации и авторизации.

Сперва получаются данные об учётной записи пользователя OAuth 2.0 из

файла на Google API Console. А после клиентское приложение запрашивает токен доступа с сервера ресурсов, который в свою очередь перенаправляет запрос на сервер авторизации Google. Данное разделение серверов на узкую направленность действий позволяет разграничить работу, не захламляя один ресурс несколькими обширными задачами.

Также важными составляющими в логике являются использующиеся JWT (JSON Web Token) и непрозрачный токен. Первый компонент является способом безопасной передачи засекреченной информации в формате JSON. Такая информация может содержать в себе пользовательские данные или данные о самом токене. Непрозрачный токен имеет формат, не предназначенный для чтения. По сути, он является своеобразным первичным ключом, указывающим на сущность с данными вместо непосредственного хранения информации о пользователе.

Таким образом, с помощью OAuth токенов происходит защита данных на сервере ресурсов. Задача же самого сервера ресурсов состоит в проверке токена перед отправкой ресурса клиенту. Отображением логики взаимодействия различных компонентов служит схема на рисунке 4.1.

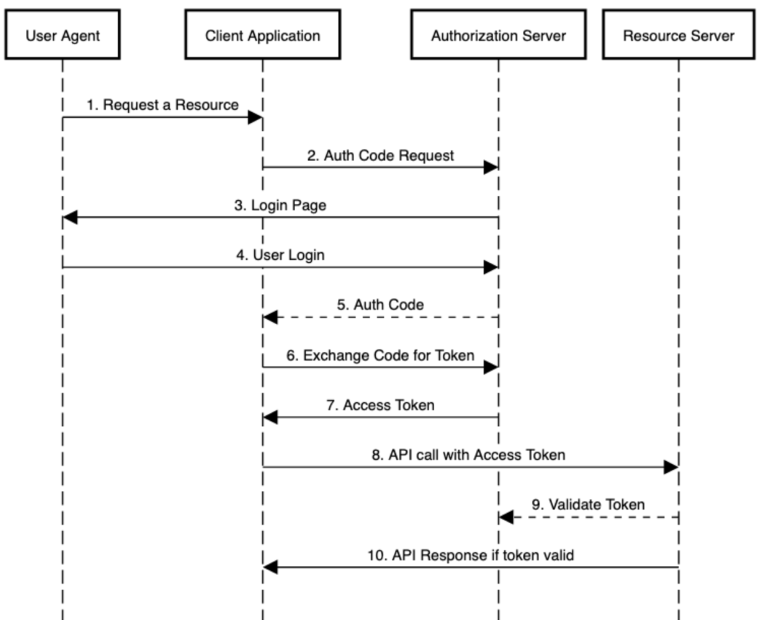


Рисунок 4.1 – Механизм OAuth 2.0

Как мы можем видеть на шаге 8, когда клиентское приложение вызывает API сервера ресурсов для доступа к защищенному ресурсу, оно сначала отправляется на сервер авторизации для проверки токена (шаг 9), содержащегося в заголовке запроса Authorization: Bearer, а затем отвечает клиенту (шаг 10).

Также легкая интеграция Oauth 2.0 в Spring Security, о котором рассказывалось в третьей главе, дает большую гарантию безопасности ресурсов сервера, поскольку Spring Security предоставляет набор функции безопасности веб-приложений, в числе которых есть методом, вызывающий на различных этапах проверки безопасности механизм Oauth 2.0.

4.2.3 Методы защиты мобильного приложения

Все данные приложения хранятся во внутреннем хранилище, что исключает возможность доступа к ним пользователя, а также других приложений.

Доступ к внутреннему хранилищу доступен только при наличии ROOT-прав у пользователя смартфона, однако приложение хранит токен, необходимый для входа в приложение, в файле SharedPreferences.

Также в мобильном приложении используется инструмент ProGuard, который необходим для оптимизации сборки Release-версии приложения, а также для обфускации исходного кода. Настройка работы ProGuard описывается в файле проекта proguardrules.pro и приведена в листинге 4.1

|  |
| --- |
| -keep class by.lisunkova.healthtracker.data.model.\*\* { \*; }  -keep class by.lisunkova.healthtracker.repository.\*\* { \*; }  -keep class com.google.android.gms.auth.api.signin.GoogleSignIn.\*\* { \*; }  -keep com.google.android.gms.auth.api.signin.GoogleSignInAccount.\*\* { \*; }  -keep com.google.android.gms.auth.api.signin.GoogleSignInClient.\*\* { \*; } |

Листинг 4.1- Файл proguardrules.pro

Такой оптимизацией сборки считается обфускация. Обфускация – процесс изменения кода программы, в результате которого он приобретает вид, трудный для понимания – при этом программа сохраняет свои функции.

Этот процесс представлен на листинге 4.1. Данные настройки необходимы для сохранения исходного содержимого классов, используемых в библиотеках, связанных с генерацией кода, таких как repository, google auth и т.д. Поскольку даже изменение названий таких классов может привести к фатальности запуска приложения. Но для остальных классов ProGuard заменяет имена классов, методов, полей, переменных и т.д. на хаотичный набор букв и цифр.

Помимо обфускации, ProGuard позволяет уменьшить размер исходных файлов, что в случае разработанного мобильного приложения снизило размер собранного apk-файла. Уменьшение размеров исходных файлов проекта производится путем удаления неиспользуемых конструкций, заменой вызова функции на ее код в случае, если она вызывается из одного места, слияния родительского и дочернего класса, если родительский класс имеет лишь одного наследника.

Также помимо удаления излишних частей неиспользуемого кода, ProGuard очищает приложение от таких же излишних ресурсов. Это работает в сочетании с сокращением кода таким образом, что после удаления неиспользуемого кода любые ресурсы, на которые больше нет ссылок, также могут быть безопасно удалены.

ProGuard позволяет оптимизировать ресурсы путем удаления неиспользуемых ресурсов в соответствии с конфигурацией проекта.

Пример байт-кода одного из классов приведен на листинге 4.2.

|  |
| --- |
| .class public abstract Lc0/e;  .super Ljava/lang/Object;  .source ""  # annotations  .annotation system Ldalvik/annotation/Signature;  value = {  "<T:",  "Lc0/e<", "TT;>;>",  "Ljava/lang/Object;"  }  .end annotation # instance fields  .field public final a:Lr1/a;  .field public final b:J  .field public final c:Lr1/p;  .field public final d:Lw1/o;  .field public final e:Lc0/e0;  .field public f:J  .field public g:Lr1/a; |

Листинг 4.2 – Байт-код проектного класса после обфускации

Как можно видеть из приведенного выше листинга, имя класса, а также его полей были заменены на случайные символы, что значительно усложняет копирование бизнес-логики проекта при декомпиляции файлов из apk.

4.3 Выводы по разделу

В данном разделе были рассмотрены потенциальные угрозы информационной безопасности разработанного программного средства, а также описаны реализованные методы защиты информации.

К ним можно отнести использование защищенного протокола HTTPS, аутентификацию пользователей при использовании приложения, а также обфускацию исходного кода мобильного приложения.

5 Тестирование приложения

ФИО

Подпись

Дата

Лист

1

БГТУ 05.00.ПЗ

Разраб.

*Лисункова А.Д.*

Пров.

*Бернацкий П.В.*

Н. контр.

*Нистюк О.А.*

Утв.

Смелов В.В.

5 Тестирование приложения

Лит.

Листов

4

*74217076, 2023*

Целью тестирования приложения является выявление и устранение любых проблем или дефектов в программном обеспечении до того, как оно будет выпущено для конечных пользователей. Тестируя приложение, мы можем убедиться, что оно соответствует функциональным и нефункциональным требованиям, удобно для пользователя и обеспечивает положительный пользовательский опыт.

Здесь будут рассмотрены два сценария: позитивного и негативного исхода тестирования. Положительное тестирование гарантирует, что система может обрабатывать ожидаемые входные данные, в то время как отрицательное тестирование гарантирует, что система может обрабатывать неожиданные, неправильные входные данные.

5.1 Позитивное тестирование

Позитивное тестирование позволяет проверить систему на корректное поведение. По ходу такого тестирования можно выявить исправность функционирования целой системы, либо ее части. Рассмотрим сценарий первого входа в аккаунт- Google, которого нет в списке (рисунок 5.1).

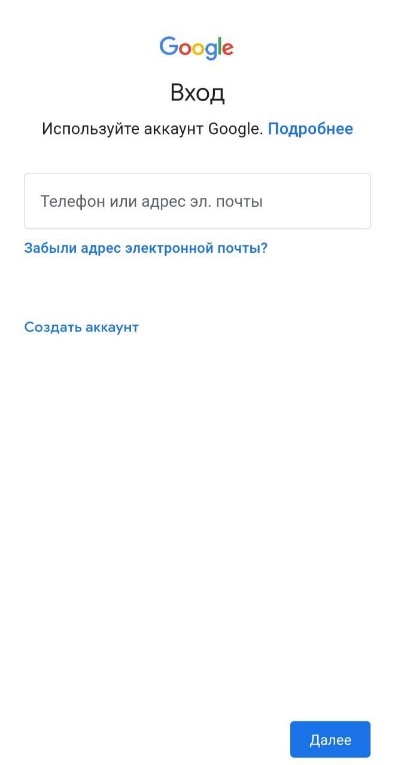
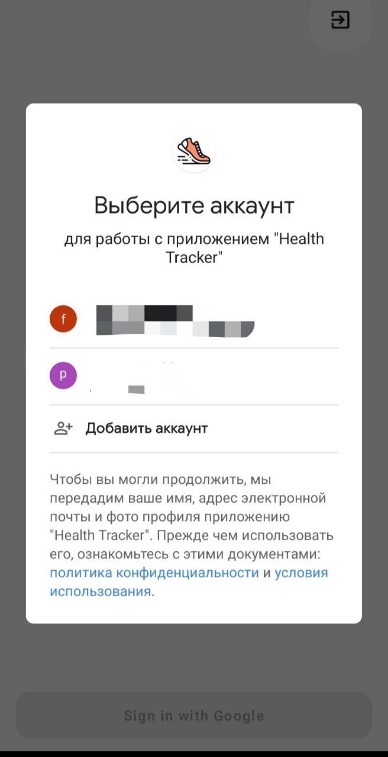


Рисунок 5.1 – Переход на форму регистрации Google

На первом скриншоте демонстрируется появление дефолтного окна входа в Google-аккаунт. Как правило из списка можно выбрать существующие на телефоне

учетные записи, но в случае, если нужного аккаунта нет, то можно войти, создав с помощью стандартных Google-форм новую запись, и в дальнейшем ее использовать в мобильном приложении.

Таким образом, сдерживающим фактором от ввода неправильного формата почты является форма входа Google, не пропускающая что-то отличное от Gmail.

Дальнейшим к рассмотрению сценарием будет проверка на работоспособность шагомера (рисунок 5.2).



Рисунок 5.2 – Работоспособность шагомера

Здесь пользователь только включил приложение, делает пробных шесть шагов, потому что счетчик временно не реагирует на шесть первых колебаний из-за того, что могла бы произойти случайна тряска, которая бы не являлась шагом и поэтому не фиксируется в момент первого колебания. Но если колебания имеют продолжительное действие, то счет шагомера начинает свою работу.

Таким образом, видим, что счетчик шагов улавливает колебания при перемещении устройства в пространстве.

В ходе позитивного тестирования было рассмотрено два сценария поведения разработанной системы: вход в приложение и работа шагомера. Проверка выполнения сценариев показала, что приложение выполняет функции исправно и соответствует заявленным критериям.

5.2 Негативное тестирование

Негативное тестирование – вид тестирования, позволяющий проверить систему на некорректное поведение. В процессе можно узнать, что происходит с системой в непредвиденных ситуациях и какое решение она может предложить.

В качестве негативного исхода рассмотрим сценарий первого входа в приложение. Пользователь получает сообщение о разрешение доступа к данным физической активности. В случае, если пользователь отказывается давать права на доступ, то он не сможет, как использовать, так и войти в приложение (рисунок 5.3).

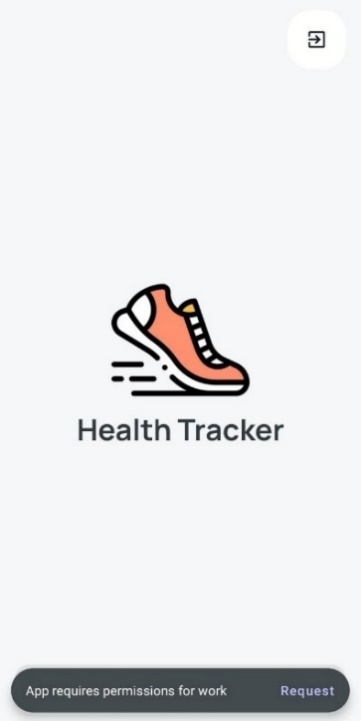
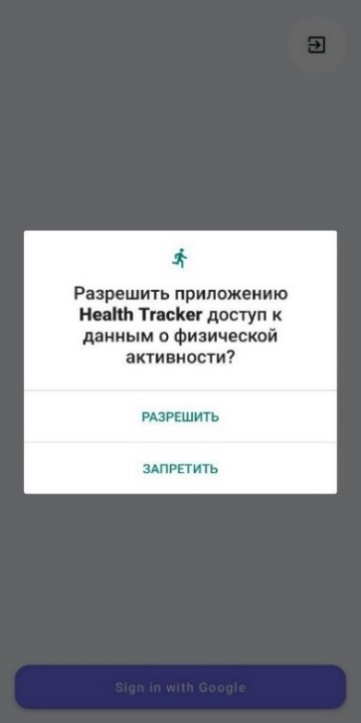


Рисунок 5.3 – Доступ к данным

Также рассмотрим еще один негативный сценарий, где пользователь вводит некорректные данные в форму физических показателей (рисунок 5.4).

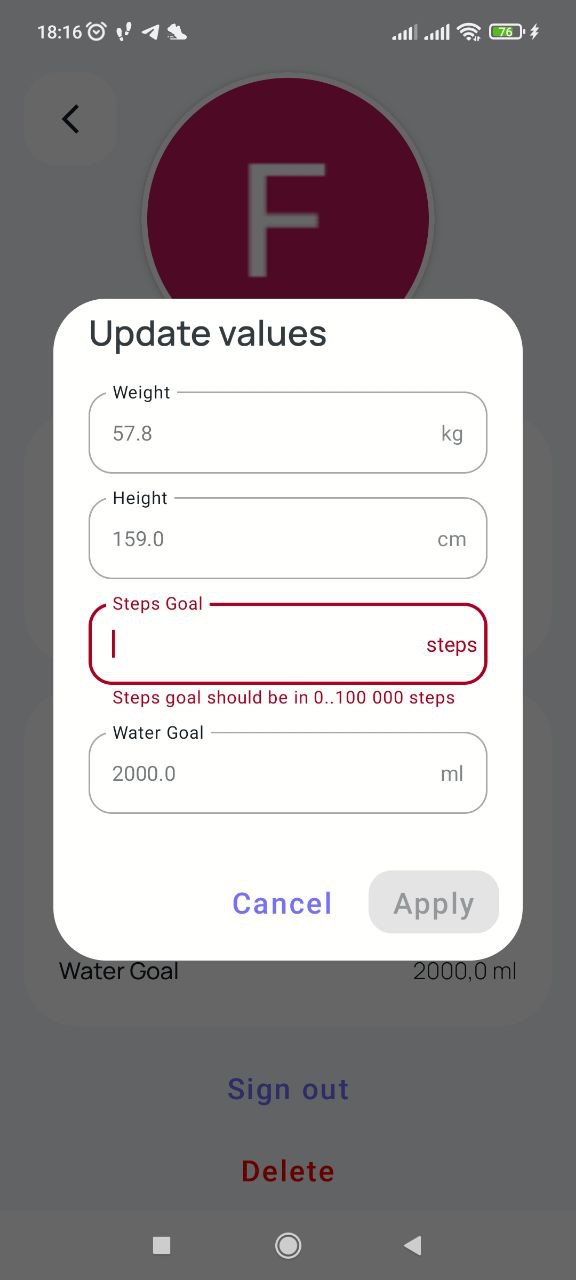


Рисунок 5.4 – Ввод некорректных данных

В поле ввода целевых шагов значение не может быть отсутствующим или не входить в заданный диапазон. Пользователь будет осведомлен о неправильности ввода ярко красной рамкой и таким же текстом, указывающим на возможные значения ввода. И в таком случае не сможет утвердить изменения.

Такого характера предупреждения будут получены во всех остальных полях ввода при неправильных значениях.

В третьем сценарии будет рассмотрена способность работы приложения как с подключением к сети, так и без него. Областью проверки была выбрана часть интерфейса, отображающая рекомендованный список упражнений пользователю. Поскольку данные берутся с помощью запросов на интернет-ресурс, то без подключения пользователь получит сообщение о возникшей ошибки.

На рисунке 5.5 продемонстрировано поведение с подключением и без.

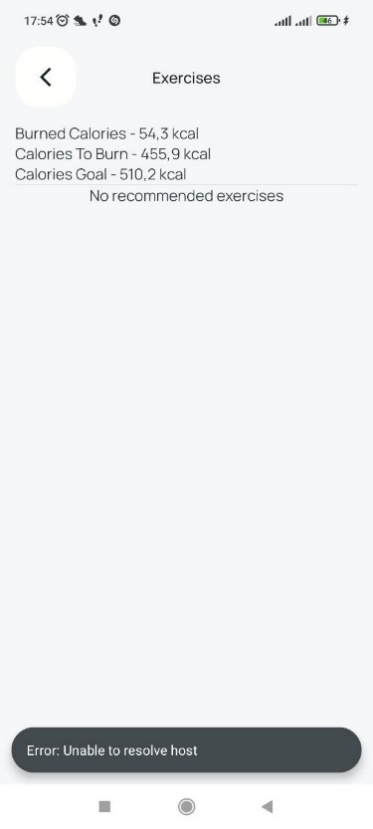
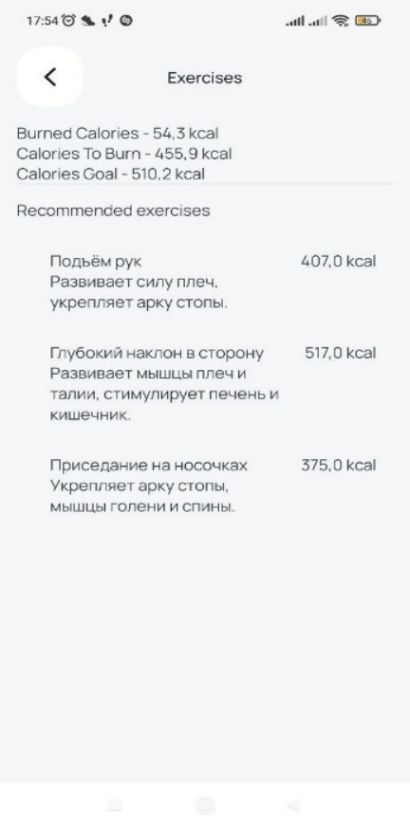


Рисунок 5.5 – Экран упражнений с/без подключения к Интернету

В ходе негативного тестирования были проработаны 2 негативных сценария. Проверка выполнения сценариев показала, что приложение корректно обрабатывает нестандартные ситуации

5.3 Выводы по разделу

В данном разделе предоставлены результаты тестирования приложения, разработанного для отслеживания уровня активности.

Использовалось позитивное и негативное тестирования, были осуществлены проверки на ввод некорректных и неверных данных.

Полученные результаты при ручном тестировании показали, что программное средство работает правильно и в соответствии с требованиями.

6 Руководство программиста

ФИО

Подпись

Дата

Лист

1

БГТУ 06.00.ПЗ

Разраб.

*Лисункова А.Д.*

Пров.

*Бернацкий П.В.*

Н. контр.

*Нистюк О.А.*

Утв.

Смелов В.В.

6 Руководство программиста

Лит.

Листов

11

*74217076, 2023*

6.1 Руководство развертывания приложения

Для развертывания серверной части необходимо иметь:

– IntelliJ IDEA Ultimate Edition 2020.3;

– Java 1.8;

– Tomcat 9.0;

– операционную систему Windows 10 версии;

– Intel(R) Core(TM) i7-9750H;

– CPU @ 2.60GHz, 2592 МГц.

Нет особой необходимости запускать серверную часть, так как она расположена на хостинге Heroku [6] и работает непрерывно. Но если есть заинтересованность в запуске на локальном устройстве для отправки запросов по стандартному адресу 8080, то в таком случае для развертывания локального сервера в первую очередь понадобится сама среда разработки, а именно IntelliJ IDEA. Загружаем серверную часть в данную среду разработки, ожидаем момента пока все библиотеки и модули подгрузятся, а затем запускаем Spring Boot приложение, как обычное Java приложение в подобной среде.

Если приложение не удалось запустить, то необходимо очистить кэш, поскольку IntelliJ IDEA может хранить подгруженные иные варианты и версии используемых в проекте библиотек, старых скомпилированных частей кода, ресурсы аннотаци1, которые могли бы вступить в конфликт в момент первой компиляции на новом устройстве. После очищения и моментальной перезагрузки, пробуем запустить проект снова. Успешным завершением компиляции будет вывод на экран консоли информации о работе приложения.

Удаленная база данных доступна также из серверного приложения, для этого из Tool Windows необходимо выбрать Database, в которую загрузятся все таблицы. Через этот интерфейс можно получить лежащие в них данные, а также производить все возможные CRUD-операции с ними.

Главное понимать, что при локальном запуске сервера мобильное приложение сможет работать только на эмуляторе этого же устройства, на котором был запущен сервер. Поскольку локальный адрес машины сможет запустится на том же адресе.

Для развертывания мобильной части проекта необходимо следующее:

– Android Studio Flamingo | 2022.2.1 Patch 1;

– Java 1.8;

– Gradle JDK corretto-11 Amazon Corretto version 11.04.14.

Загружаем проект в Android Studio, ждем загрузки библиотек. Но на этом моменте может произойти казус: из-за неверной версии JDK среда выдаст сообщение

об ошибке и даст возможность установить нужную версию Gradle JDK, а именно corretto-11 от Amazon. Kotlin установится по умолчанию вместе со средой, поэтому не был указан в качестве необходимого компонента развертывания мобильной части. В случае несовпадения встроенной версией Kotlin с библиотекой Jetpack Compose, в процессе компиляции будет указана версия языка, которая корректно взаимодействует с библиотекой, и которую нужно будет указать в настройках Gradle проекта. Язык Java нужен из-за того, что Android-приложения выполняются с помощью JVM. Благодаря особенности Java поддерживать старые версии новыми JDK, то переменную среды JAVA HOME нет нужды менять на более старую версию.

Рекомендуется запускать на настоящем мобильном устройстве, путем подключения его USB-кабелем к устройству, на котором находится проект. Поскольку приложение содержит шагомер, то запуск на эмуляторе может затруднить его работу. Либо другим вариантом является запуск приложения на мобильном устройстве по Wi-fi. Настройку того или иного способа подскажет Android Studio. Но в случае, если сервер работает локально, то запускать исключительно на эмуляторе, поскольку его адрес будет совпадать с адресом всей машины.

При запуске проекта на мобильное устройстве устанавливается приложение, причем некоторые устройства требуют непосредственного согласия на установку. В таком случае нажимаем на кнопку «Согласиться».

Таким образом, были описаны нюансы запуска частей проекта, учет которых способствует корректному запуску конечного продукта.

6.2 Руководство использования мобильного приложения

Мобильное приложение поддерживает 2 языка локализации: английский и русский. Язык приложения выбирается в соответствии с системными настройками устройства. Английский язык используется по умолчанию, если язык системы не поддерживается. Пользователь нажимает на иконку приложения и при первой установке ему выведется сообщение о необходимости дать разрешение к физической активности. Оно необходимо для работы приложения, в ином случае, если пользователь не соглашается, то в приложение он не сможет попасть (рисунок 6.1).

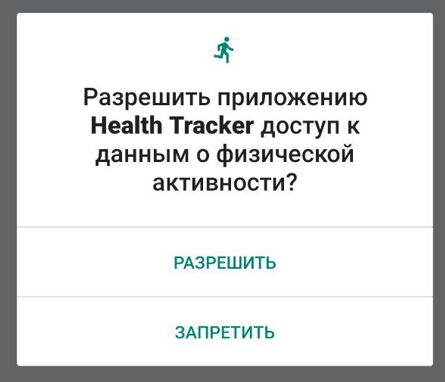


Рисунок 6.1 – Разрешение на доступ к данным

Если это первый вход пользователя, то на экране представлен центрированный логотип, верхняя боковая кнопка выхода из приложения и широкая прижатая книзу кнопка входа, нажав на которую выведется дефолтный экран Google-формы для выбора или добавления аккаунта.

На рисунке 6.2 продемонстрирован экран входа в мобильное приложение.



Рисунок 6.2 – Экран входа в приложение

Приложение дает возможность авторизации только посредством Google-аккаунта. Препятствует вводу неверного формата почты непосредственно внутренние настройки и алгоритмы аутентификации Google. Выбор аккаунта изображен на рисунке 6.3.

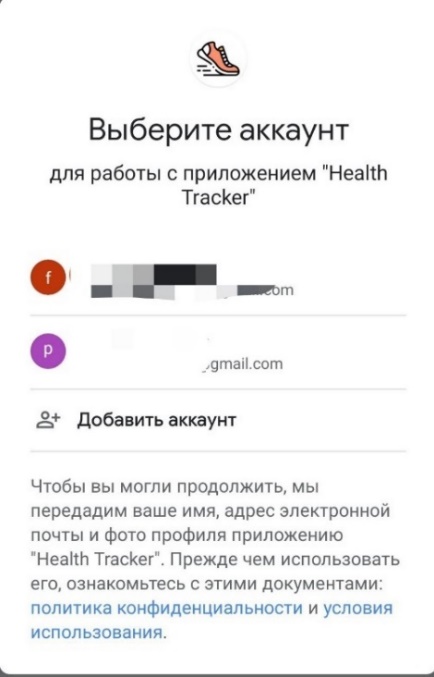


Рисунок 6.3 – Выбор аккаунта

Определившись на прошлом этапе с аккаунтом, пользователь попадает на главный экран приложения, где информация разбита по блокам в зависимости от назначения. Такая простота интерфейса дает пользователю мгновенную ориентацию в незнакомом приложении, интуитивное понимание как им пользоваться и что ожидать от взаимодействия с его элементами. Скриншот главного экрана приложения приведен в приложении Д.

Следующим сценарием будет открытие профиля пользователя. Для этого в верхнем левом углу главного экрана выбираем иконку с именем авторизовавшегося пользователя. Рассмотрим часть данного экрана, которая выводит в виде отдельного блока информацию, берущуюся из аккаунта Google (рисунок 6.3).

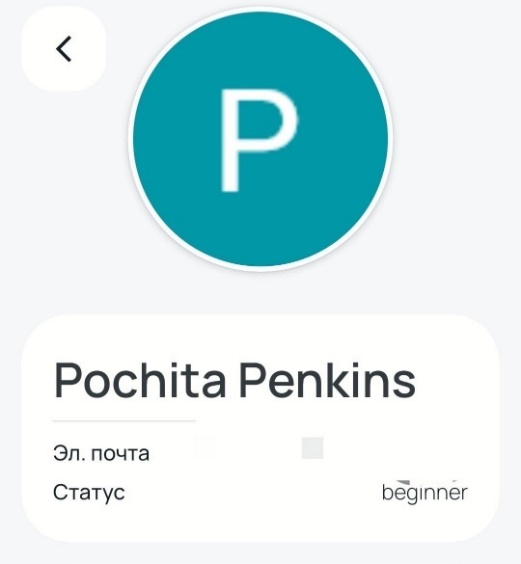


Рисунок 6.3 – Фрагмент страницы пользовательского аккаунта

Здесь отображается изображение профиля, электронная почта и статус, который будет в дальнейшем меняться при открытии новых достижений. Эти данные, за исключением статуса, пользователь сможет поменять, зайдя в свой Google-профиль.

На этом же экране блоком ниже находятся настройки, устанавливающие такие параметры, как рост, вес, цель по шагам и цель по воде.

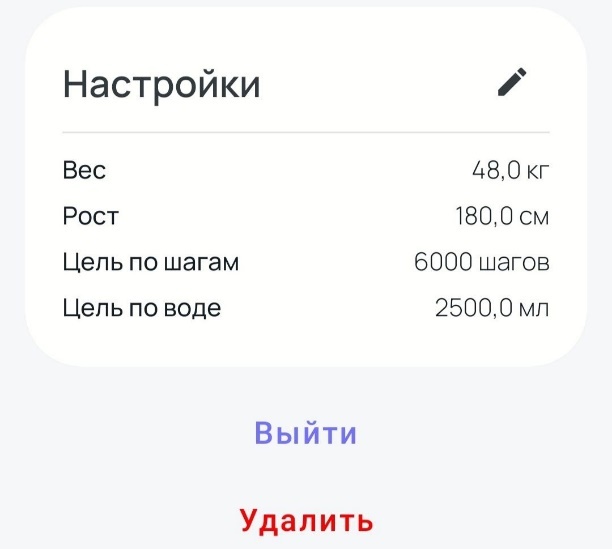


Рисунок 6.4 – Фрагмент экрана с настройками

Нажав на значок «Карандаша» можно поменять значения, выставленные по умолчанию или ранее вводимые пользователем. Кнопка «Выйти» осуществляет выход пользователя из учетной записи, а удалить – стирает все данные с устройства и сервера. На рисунке 6.5 показан процесс изменения значений параметров.

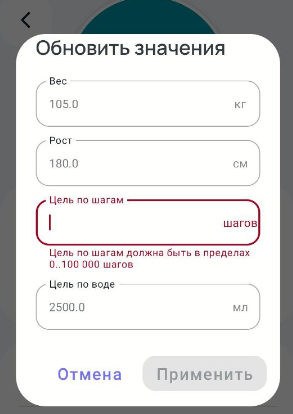


Рисунок 6.5 – Изменение значений параметра настроек

Колебания веса и роста отображаются на полоске индекса массы тела (ИМТ), расположенного на главном экране приложения. На рисунке 6.6 представлен ИМТ.



Рисунок 6.6 – Показатель ИМТ

По цветовому градиенту можно определить степень избыточности, так и недостаточности веса по отношению к росту. Зеленый цвет считается цветом баланса и равновесия, что на подсознательном уровне трактуется человеком как норма. В дополнение к этому, центрирование зеленого также закладывает мысль о том, что в данном случае этот участок является «золотой» серединой индекса. Соответственно уклоны в большую и меньшую сторону помечены цветами, которые распространенно считаются индикаторами нижнего и высокого уровня измеряемой величины. Таким образом, пользователь может присмотреться к своему здоровью на основании краткого анализа по его показателям.

Основной функцией приложения является счет шагов и вытекающие из них расчеты других показателей, таких как калории и расстояние.

На рисунке 6.7 показан фрагмент отслеживания шагов.



Рисунок 6.7 – Отслеживание шагов

В вытянутом синем блоке выводятся текущие пройденные шаги за день и количество, к которому нужно стремиться. Два соседних блока показывают суммарную цель за месяц и неделю. При нажатии на любой из этих блоков осуществляется переход на экран со статистикой за выбранный отрезок времени.

Для просмотра сожженных калорий и пройденной дистанции надо прокрутить до конца главный экран и в последнем блоке найти данные показатели (рисунок 6.8).



Рисунок 6.8 – Прогресс за сегодня

Вернувшись на главный экран, пользователь может добавить то количество выпитых стаканов, который он сделал за день, опираясь на заданную цель по водному балансу. То есть суммарное количество стаканов формируется по тому какой литраж пользователь задал в настройках, приведенных на рисунке 6.4.

На рисунке 6.9 демонстрируется возможность добавлять выпитую воду.

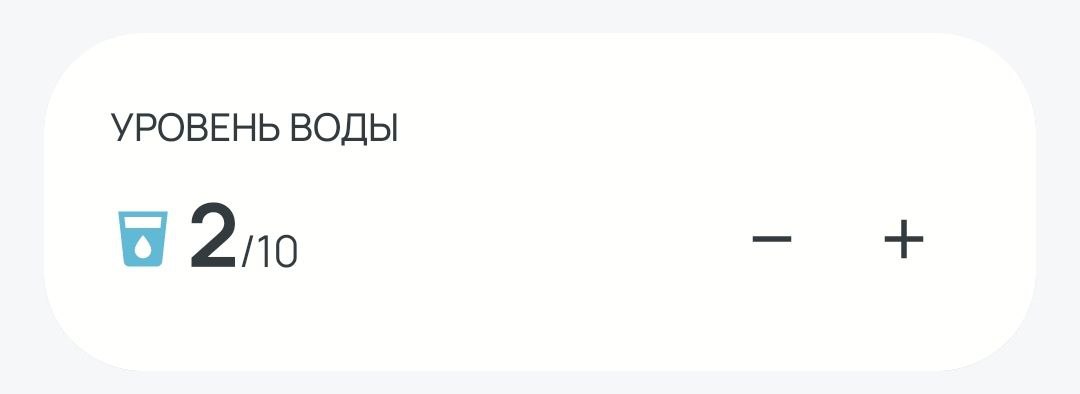


Рисунок 6.9 – Добавление выпитой воды в стаканах

Пользователь, нажимая на «плюс», добавляет то количество жидкости в стакане, которое выпил. «Минус» предусмотрен для случайного добавления результата, который не соответствует действительности, благодаря чему пользователь убавляет неверное количество стаканов.

Также пользователю приходят напоминания о том, чтобы тот не забывал придерживаться плану по водному балансу за день (рисунок 6.10).

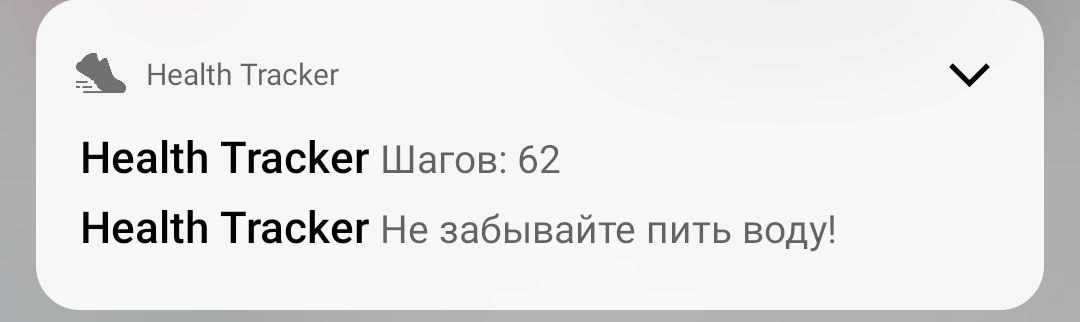


Рисунок 6.10 – Напоминание выпить воду

На этом рисунке изображено начисление шагов во время ходьбы. Это делает удобным для пользователя отслеживание шагов без непосредственного входа в приложение. В том числе для того, чтобы открыть приложение достаточно нажать на данные уведомления на панели уведомлений системы Android.

После нажатия одного из элементов на рисунке 6.7 пользователь перейдет на экран статистики по шагам за разные периоды времени (рисунок 6.11).



Рисунок 6.11 – Статистика по шагам

График показывает динамику активности по дням, где идет зависимость намеченной цели от фактического результата. Также детально по дням отображены проценты того, насколько успешно справился пользователь.

На главной странице размещен блок с добавленными друзьями и поиском новых контактов (рисунок 6.12).

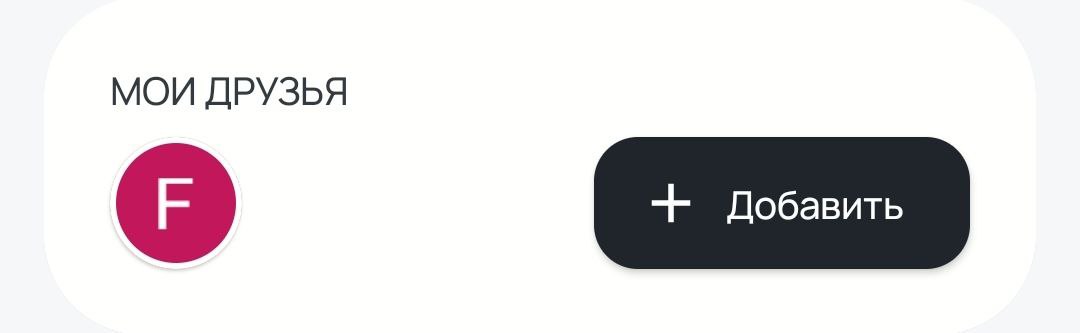


Рисунок 6.12 – Друзья пользователя

Нажав на профиль одного из друзей, осуществляется переход на страницу с полным списком всех установленных контактов (рисунок 6.13).

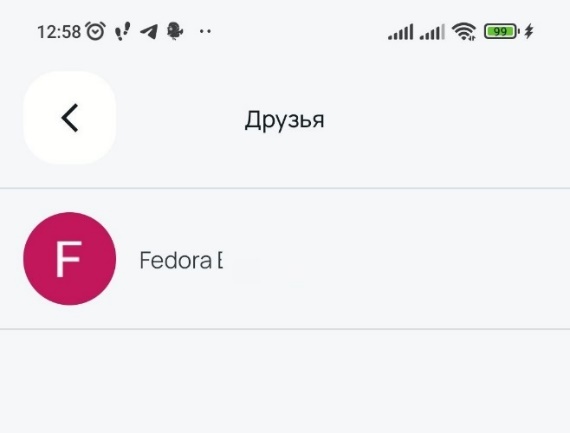


Рисунок 6.13 – Друзья пользователя

Перейти на чужой профиль можно нажатием на иконку пользователя. На рисунке 6.14 изображена часть вида чужого профиля.



Рисунок 6.14 – Часть чужого профиля

На странице приводится сравнение результатов по шагам с авторизовавшимся профилем, где за 100 %-ое значение берется максимальный показатель среди пользователей в тот день. В правом верхнем углу целой страницы расположена иконка «пользователь минус», позволяющая удалить из друзей данный профиль. Кнопка с аналогичным расположением, но на профиле, с которым не было установлено дружеской связи выглядит как пользователь рядом со знаком «Плюс».

При нажатии на кнопку добавить на рисунке 6.12 появляется экран поиска, продемонстрированный на рисунке 6.15.

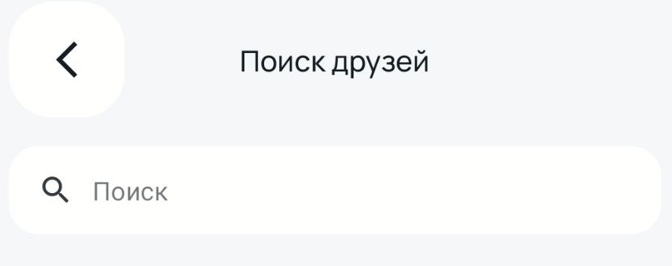


Рисунок 6.15 – Поиск друзей

Поле для ввода принимает имя и фамилию, прописанные в Google-аккаунте. Принимает как вместе, так и по отдельности, искомого пользователя.

Возвратившись на главный экран, в правом верхнем углу расположена иконка «Звезды», нажав на которую, открывается страница с достижениями пользователя. На рисунке 6.16 демонстрируется экран с разблокированными и заблокированными достижениями в виде списков.

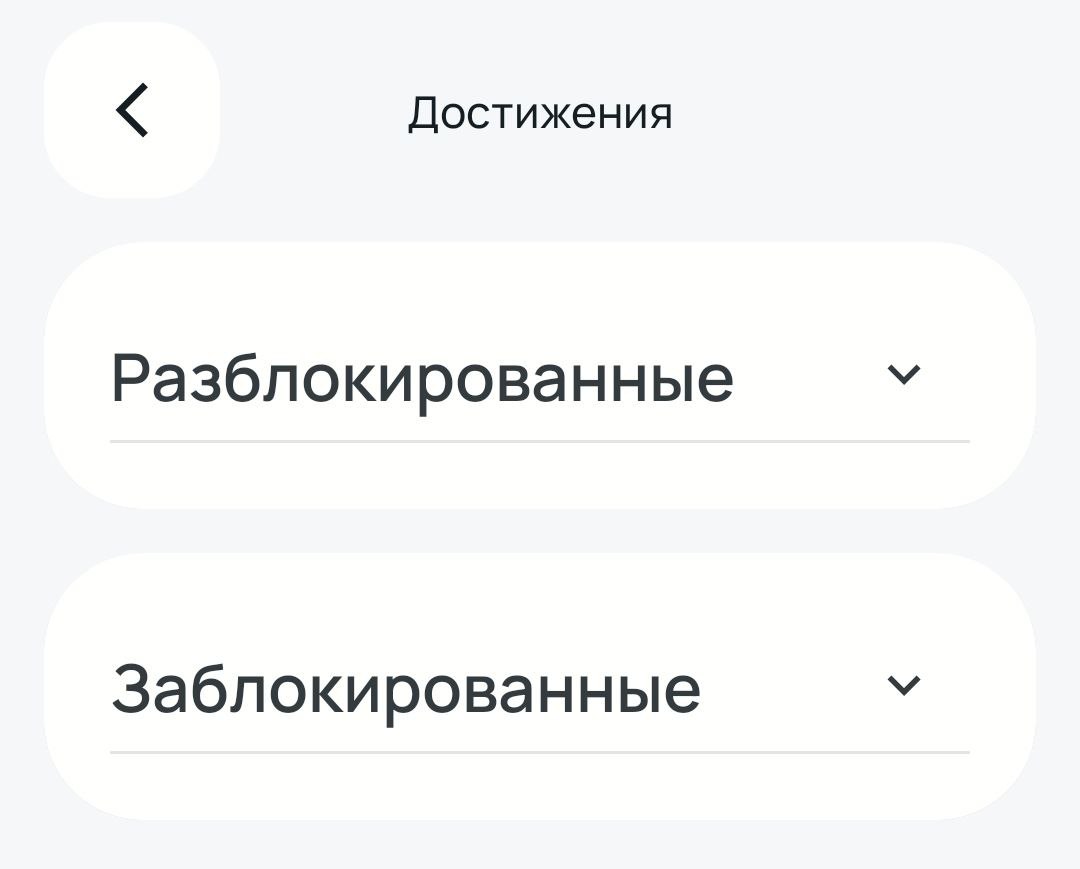


Рисунок 6.16 – Достижения пользователя

Достижения двух категорий представлены раскрывающимися списками, в которых достижения выводятся в виде небольшой иконки с описанием за что она начисляется. На рисунке 6.17 показан развернутый список достижений.

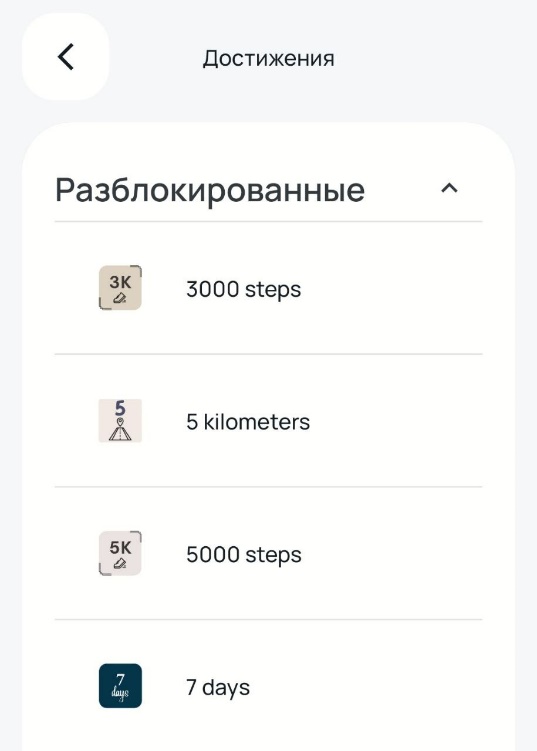


Рисунок 6.17 – Список разблокированных достижений

Также в случае получения достижения отобразится уведомление в панели оповещений Android системы.

Вернувшись на главный экран и нажав на блок, показанный на рисунке 6.8 осуществится переход на страницу с подробным описанием текущих калорий и подбором упражнений по данным о калориях (рисунок 6.18).

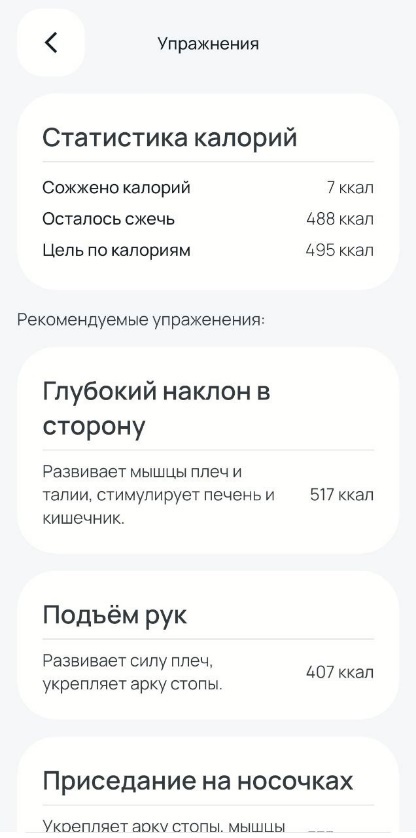


Рисунок 6.18 – Экран с упражнениями

Здесь показано количество сожженных калорий, сколько осталось сжечь – данное количество вычисляется автоматически за счет поставленной цели по шагам. Подбираются 3 упражнения, которые способны сжечь примерно то желаемое количество калорий. Нажав на одно из них, открывается экран с подробным описанием выполнения и картинками, на которых изображены правильные позы. Демонстрация тому представлена на рисунке 6.19.



Рисунок 6.19 – Описание упражнения

Таким образом, пользователь может, ориентируясь на подсказки, выполнить упражнение и сжечь калории.

6.3 Выводы по разделу

В данном разделе представлено пошаговое руководство программиста, описывающее основные этапы развертывания частей проекта, а также работу с мобильным приложением «HealthTracker» с помощью скриншотов интерфейса приложения.

Прояснились моменты запуска серверной и клиентской частей, представлены основные характеристики устройств для работы с проектами.

Интерфейс разработанного приложения является современным, а также он интуитивно понятный и простой в эксплуатации, поэтому у пользователя не должно возникнуть трудностей при работе с приложением.

После завершения разработки руководства необходимо перейти к технико-экономическому обоснованию, связанного с разработкой проекта.

7 Технико-экономическое обоснование проекта

ФИО

Подпись

Дата

Лист

1

*58*

*БГТУ 07.00.ПЗ*

Разраб.

*Лисункова А.Д.*

Провер.

*.*

*Бернацкий П.В.*

Консульт.

*Соболевский А.С. .*

Н. контр.

Нистюк О.А.

Утв.

*Смелов В.В.*

7 Технико-экономическое обоснование проекта

Лит.

Листов

11

*74217076, 2023*

Основной целью экономического раздела является экономическое обоснование целесообразности разработки программного средства, представленного в дипломном проекте. В рамках данного раздела необходимо определить затраты, произведенные на всех стадиях разработки описанного программного модуля, а также полную стоимость программного средства.

7.1 Общая характеристика разрабатываемого программного средства

При выполнении дипломного проекта было разработано мобильное приложение, предназначенное для оценки уровня активности пользователя.

Цель приложения – предоставить пользователю возможность отслеживать свою физическую активность. Любой пользователь может войти в приложении, для этого надо лишь зайти в Google-аккаунт. Облегчить эту задачу сможет уже задействованные на мобильном устройстве аккаунты, которые можно будет выбрать в окне авторизации. После того, как пользователь зашел в свою учетную запись посредством Google- авторизации, он попадает на главную страницу и ему становятся доступны все имеющиеся функции приложения.

Мобильный продукт позволяет пользователю считать сожженные во время ходьбы калории, пройденные шаги, отслеживать уровень индекса массы тела при изменении физических параметров, добавлять количество выпитой жидкости для поддержания водного баланса в организме. Пользователю будут предложены упражнения, количество калорий которых может покрыть недожженные калории во время ходьбы. Так же имеется возможность просматривать статистику по своим шагам за определенные интервалы времени, получать достижения за количество пройденных шагов, дней посещения приложения и километров пути.

Во время разработки приложения использовались технологии Spring Framework, JPA, Compose, языки программирования Java, Kotlin, SQL, хостинг Heroku. Непосредственно в разработке мобильного продукта использовались библиотека Jetpack Compose для создания собственного пользовательского интерфейса, а также язык программирования Kotlin под разработку мобильного приложения на Android.

Стратегия монетизации предполагает платную месячную подписку на дополнительный функционал в приложении. В рамках данного раздела необходимо определить затраты на всех стадиях разработки программного средства.

Необходимо выполнить расчет экономии основных видов ресурсов в связи с использованием разработанного программного средства на основе успеха и монетизации приложений аналогов.

**7.2 Исходные данные для проведения расчётов**

Исходные данные для расчета приведены в таблице 7.1.

Таблица 7.1 – Исходные данные для расчета

|  |  |  |
| --- | --- | --- |
| Наименование показателя | Условные обозначения | Норматив |
| Численность разработчиков, чел | Чр | 1 |
| Коэффициент изменения скорости обработки информации, ед. | Кск | 0,6 |
| Норматив дополнительной заработной платы, % | Ндз | 14 |
| Норматив общепроизводственных и общехозяйственных расходов, % | Нобп, обх | 20 |
| Ставка отчислений в Фонд социальной защиты населения, % | Нфсзн | 34 |
| Ставка отчислений по обязательному страхованию в БРУСП «Белгосстрах», % | Нбгс | 0,6 |
| Цена одного машино–часа, руб. | Смч | 0,06 |
| Норматив прочих прямых затрат, % | Нпз | 20 |
| Норматив расходов на сопровождение и адаптацию, % | Нрса | 17,0 |
| Ставка НДС, % | ННДС | 0 |
| Налог на прибыль, % | Нп | 0 |

Источниками исходных данных для данных расчетов выступают действующие законы и нормативно-правовые акты. Для ПВТ НДС и налог на прибыль 0%.

**7.3 Методика обоснования цены**

В современных рыночных экономических условиях программное средство (ПС) выступает преимущественно в виде продукции организаций, представляющей собой функционально завершенные и имеющие товарный вид, реализуемые покупателям по рыночным отпускным ценам. Все завершенные разработки являются научно-технической продукцией.

Широкое применение вычислительных технологий требует постоянного обновления и совершенствования программных средств. Выбор эффективных проектов программных средств связан с их экономической оценкой и расчетом экономического эффекта, который может определяться как у разработчика, так и у обычного пользователя системы.

У разработчика экономический эффект выступает в виде чистой прибыли от реализации программных средств, остающейся в распоряжении организации, а у пользователя – в виде экономии трудовых, материальных и финансовых ресурсов, получаемой за счет:

* снижения трудоемкости расчетов и алгоритмизации программирования и отладки программ;
* сокращения расходов на оплату машинного времени и других ресурсов на отладку программ;
* снижения расходов на материалы;
* ускорение ввода в эксплуатацию новых систем;
* улучшения показателей основной деятельности в результате использования передовых программных средств.

Стоимостная оценка программных средств у разработчиков предполагает определение затрат, что включает следующие статьи:

* заработная плата исполнителей – основная и дополнительная;
* отчисления в фонд социальной защиты населения;
* отчисления по обязательному страхованию от несчастных случаев на производстве и профессиональных заболеваний;
* расходы на оплату машинного времени;
* прочие прямые затраты;
* накладные расходы.

На основании затрат рассчитывается себестоимость и отпускная цена конечного программного средства [7].

**7.3.1 Объем программного средства**

В таблице 7.2 указаны в укрупнённом виде все работы, реально выполненные для создания, указанного в дипломной работе программного средства и количество рабочих дней, реально потраченных для выполнения этих работ.

Таблица 7.2 – Затраты рабочего времени на разработку ПС

|  |  |
| --- | --- |
| Содержание работ | Затраты рабочего времени, дней |
| Построение диаграмм для проектирования дипломного проекта | 6 |
| Создание базы данных | 3 |
| Разработка серверной части программного средства | 15 |
| Развертывание сервера | 1 |
| Разработка клиентской части программного средства | 21 |
| Тестирование работоспособности мобильного средства | 5 |
| Написание руководства пользователя | 2 |
| Всего | 53 |

Результат по данной таблице будет использовано далее для расчётов.

Для оценки объёма программного средства, все его функции классифицируются с использованием специального каталога функций, который определяет их объем. Общий объем программного средства *V*o, вычисляется как сумма объёмов *Vi* каждой из *n* его функций по формуле 7.1.

|  |  |  |
| --- | --- | --- |
|  | , | (7.1) |

где – объем *i*-ой функции ПС, условных машинных команд;

*n* – общее число функций.

В таблице 7.3 представлены функции, присутствующие в рассматриваемом программном средстве и соответствующий им объем в условных машино–командах.

Таблица 7.3 – Содержание и объем функций в программном средстве

|  |  |  |
| --- | --- | --- |
| Номер функции | Содержание функции | Объем, условных  машино-команд |
| 101 | Организация ввода информации | 150 |
| 102 | Контроль, предварительная обработка | 450 |
| 111 | Управление вводом/выводом | 2400 |
| 204 | Обработка наборов и записей базы данных | 2670 |
| 207 | Манипулирование данными | 9550 |
| 504 | Обработка прерываний | 540 |
| 506 | Обработка ошибочных и сбойных ситуаций | 410 |
| 703 | Расчет показателей | 460 |
| 707 | Графический вывод результатов | 480 |

Опираясь на данные таблицы 7.3, можно определить объем программного средства, разработанного в ходе дипломного проектирования:

*Vo* = 150 + 450 + 2400 + 2670 + 9550 + 540 + 410 + 460 + 480 = 17110 (маш. команд).

Уточнённый объем программного средства *V*o*/* вычисляется по формуле 7.2.

|  |  |  |
| --- | --- | --- |
|  | , | (7.2) |

где *V*o – объем программного средства, усл. машино-команд;

Кск – коэффициент изменения скорости обработки информации.

Исходя из вычисленного объёма программного средства, можно определить уточненный объем программного средства:

*V*o*/* = 17110 ⋅ 0,6 = 10266 (условных машино–команд).

**7.3.2 Основная заработная плата**

Для определения величины основной заработной платы, было проведено исследование величин заработных плат для специалистов в программирования на Kotlin Android. В итоге было установлено, что средняя месячная заработная плата на позиции junior составляет 1600 рублей.

Согласно таблице 7.1, проект разрабатывался одним человеком на протяжении 53 дней, что соответствует 2,5 месяца, поскольку в среднем рабочих дней в месяце 21, отсюда суммарное количество дней разработки делим на усредненное количество рабочих дней в месяце. Таким образом, основная заработная плата будет рассчитываться по формуле 7.3

|  |  |  |
| --- | --- | --- |
|  | , | (7.3) |

где Соз– основная заработная плата, руб.;

Траз1 – время раработки , месяцев;

Сзп1 – средняя месячная заработная плата Kotlin Android разработчика.

|  |  |  |
| --- | --- | --- |
|  | руб. |  |

В дальнейшем для других расчётов используется основная заработная плата, рассчитанная по указанной выше методике.

**7.3.3 Дополнительная заработная плата**

Дополнительная заработная плата на конкретное программное средство включает выплаты, предусмотренные законодательством о труде, и определяется по нормативу в процентах к основной заработной плате по формуле (7.4)

|  |  |  |
| --- | --- | --- |
|  | , | (7.4) |

где Соз– основная заработная плата, руб.;

Ндз – норматив дополнительной заработной платы, %.

руб.

**7.3.4 Отчисления в Фонд социальной защиты населения**

Отчисления в Фонд социальной защиты населения (ФСЗН) и по обязательному страхованию от несчастных случаем на производстве и профессиональных заболеваний определяются в соответствии с действующими законодательными актами по нормативу в процентном отношении к фонду основной и дополнительной зарплаты исполнителей.

Отчисления в Фонд социальной защиты населения вычисляются по формуле 7.5

|  |  |  |
| --- | --- | --- |
|  |  | (7.5) |

где – основная заработная плата, руб.;

– дополнительная заработная плата на конкретное ПС, руб.;

– норматив отчислений в Фонд социальной защиты населения, %.

Отчисления в БРУСП «Белгосстрах» вычисляются по формуле 7.6

|  |  |  |
| --- | --- | --- |
|  |  | (7.6) |

Таким образом, общие отчисления в БРУСП «Белгосстрах» составили руб., а в фонд социальной защиты населения – руб.

**7.3.5 Расходы на материалы**

Сумма расходов на материалы СМ определяется как произведение нормы расхода материалов в расчете на сто строк исходного кода НМ на уточненный объем программного средства *Vo/* , по формуле 7.7

|  |  |  |
| --- | --- | --- |
|  | . | (7.7) |

Учитывая, что норма расхода материалов в расчете на сто строк исходного кода равен 0,46 руб. (по данным, приведенным в приложении 2 таблице П 2.10 «Оценка значений среднего расхода материалов на разработку и отладку 100 строк кода применения ПС» методического пособия), можно определить сумму расходов на материалы.

СМ = 0,46 ⋅ 10266 / 100 = 47,22 руб.

Сумма расходов на материалы была вычислена на основе данных, приведенных в таблице 7.1 данного дипломного проектирования.

**7.3.6 Расходы на оплату машинного времени**

Сумма расходов на оплату машинного времени Смв определяется как произведение стоимости одного машино-часа Смч на уточненный объем программного средства *Vo/* и на норматив расхода машинного времени на отладку ста строк исходного кода НМВ, по формуле 7.8

|  |  |  |
| --- | --- | --- |
|  | . | (7.8) |

Учитывая, что норматив машинного времени на отладку ста строк исходного кода равен 12 (по данным, приведенным в приложении 2 таблице П 2.11 «Оценка значений среднего машинного времени на отладку 100 строк исходного кода без применения ПС» методического пособия), можно определить сумму расходов на оплату машинного времени.

Смв = 0,06 ⋅ 10266 ⋅ 12 / 100 = 73,92 руб.

Сумма расходов на оплату машинного времени была вычислена на основе данных, приведенных в таблице 7.1 данного дипломного проектирования.

**7.3.7 Прочие прямые затраты**

Сумма прочих затрат Спз определяется как произведение основной заработной платы исполнителей на конкретное программное средство Соз на норматив прочих затрат в целом по организации Нпз, и находится по формуле 7.9.

|  |  |  |
| --- | --- | --- |
|  |  | (7.9) |

Все данные необходимые для вычисления есть, поэтому можно определить сумму прочих затрат.

Спз = ⋅ 20/ 100 = 800 руб.

**7.3.8 Накладные расходы**

Сумма накладныхрасходовСобп,обх – произведение основной заработной платы исполнителей на конкретное программное средство Соз на норматив накладных расходов в целом по организации Нобп,обх, по формуле 7.10.

|  |  |  |
| --- | --- | --- |
|  |  | (7.10) |

Все данные необходимые для вычисления есть, поэтому можно определить сумму накладных расходов.

= ⋅ 20 / 100 = 800 руб.

**7.3.9 Сумма расходов на разработку программного средства**

Сумма расходов на разработку программного средства Ср определяется как сумма основной и дополнительной заработных плат исполнителей на конкретное программное средство, отчислений на социальные нужды, расходов на материалы, расходов на оплату машинного времени, суммы прочих затрат и суммы накладных расходов, по формуле 7.11.

|  |  |  |
| --- | --- | --- |
|  | Ср = Соз + Сдз + Сфсзн + Сбгс + См + Смв + Спз + Собп,обх. | (7.11) |

Все данные необходимые для вычисления есть, поэтому можно определить сумму расходов на разработку программного средства.

Ср = +++ 27,36 + 47,22 + 73,92 + 800 + 800 = 7858,90 руб.

Сумма расходов на разработку программного средства была вычислена на основе данных, рассчитанных ранее в данном разделе, и составила 7858,90 рублей.

**7.3.10 Расходы на сопровождение и адаптацию**

Сумма расходов на сопровождение и адаптацию программного средства Срса определяется как произведение суммы расходов на разработки на норматив расходов на сопровождение и адаптацию Нрса, и находится по формуле 7.12.

|  |  |  |
| --- | --- | --- |
|  |  | (7.12) |

= 7858,90 ⋅ 17 / 100 = 1336,01 руб.

Сумма расходов на сопровождение и адаптацию была вычислена на основе данных, рассчитанных ранее в данном разделе.

Все проведенные выше расчеты необходимы для вычисления полной себестоимости проекта.

**7.3.11 Полная себестоимость**

Полная себестоимость Сп определяется как сумма двух элементов: суммы расходов на разработку Ср и суммы расходов на сопровождение и адаптацию программного средства Срса.

Полная себестоимость Сп вычисляется по формуле 7.13

|  |  |  |
| --- | --- | --- |
|  |  | (7.13) |

Сп = 7858,90 + 1336,01 = 9194,91 руб.

Полная себестоимость программного средства была вычислена на основе данных, рассчитанных ранее в данном разделе.

**7.3.12 Определение цены, оценка эффективности**

Рассматриваемое программное средство разрабатывается для потребления другими лицами. После анализа способа монетизации продуктов-аналогов, был выбран способ монетизации «расширенная подписка на месяц».

Продукты-аналоги, наиболее близкие теме дипломного проекта:

* «Step Counter» – мобильное приложение на Android систему, в большинстве своем выполняет функцию шагомера со встроенными возможностями, как счет калорий, расстояния и отслеживанием принятого уровня жидкости за день. [2]
* «Stepz – шагомер и счет калорий» – мобильное приложение, перенявшее характеристики предыдущего, но с возможностью получения достижений. [3]

По балловому методу цены рассчитываются на основе оценок значимости различных параметров качества программного продукта для потребителей. Сначала каждому из тех параметров, по которым судят о качестве продукции, присваивается значение удельного веса, которое характеризует, насколько данный параметр важен для потребителя по сравнению с другими. Далее каждому параметру базового программного продукта присваивается определенное число баллов, суммирование которых с учетом весовых коэффициентов дает интегральный показатель качества продукции конкурента ИК:

ИК = Σ (Кi ПКi) / Σ Кi при Σ Кi = 1, (7.14)

где К*i* − весовой коэффициент, отражающий значимость *i-*го показателя;

ПК*i* − число баллов, присвоенное *i-*му показателю продукта конкурента.

Показатель качества рассматриваемого продукта ИР рассчитывается как:

ИР = Σ (Кi ПРi) / Σ Кi при Σ Кi = 1, (7.15)

где ПР*i* − число баллов, присвоенное *i-*му показателю качества рассматриваемого программного продукта.

Были выбраны следующие характеристики для показателей качества рассматриваемого программного продукта и программного продукта конкурента:

* Дизайн – то, как приложение выглядит, очень важно для любого современного приложения.
* Юзабилити – насколько приложение удобно в использовании.
* Функциональность – количество инноваций, внедряемых в приложение.
* Отсутствие багов – наличие несущественных ошибок в работе.

Расчет показателей качества базового и нового продуктов, согласно балловому методу, приводится в таблицах 7.4

Таблица 7.4 – Показатели качества рассматриваемого программного продукта и программного продукта конкурента

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Показатель качества | Весовой коэффициент | Моё приложение | Step Counter | Stepz – шагомер и счет калорий |
| Дизайн | 0,3 | 8 | 6 | 4 |
| Юзабилити | 0,4 | 7 | 7 | 6 |
| Функциональность | 0,2 | 7 | 7 | 5 |
| Отсутствие багов | 0,1 | 8 | 7 | 7 |
| Всего | 1 | 7,4 | 6,7 | 5,3 |

Расчёт прогнозного количества установок программного средства Кi при монетизации методом расширенной месячной подписки на дополнительный функционал приложения, рассчитывается по формуле 7.16:

|  |  |
| --- | --- |
|  | (7.16) |

где К0 − количество установок ПС конкурента;

T0 – количество лет существования приложения;

ИР – показатель рассматриваемого программного продукта;

ИК – показатель программного продукта конкурента.

К1 = (500 000 / 6 7,4) / 6,7 = 92 039,80 (установок в год),

К2 = (200 000 / 4 7,4) / 5,3 = 69 811,32 (установок в год),

К = (92 039,80 + 69 811,32) / 2 = 80 925,56 (установок в год).

Определение цены месячной расширенной подписки нового продукта Ц1 будет осуществляться по следующей формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.17) |

где Ц0 − цена подписки программного продукта конкурента,

ИР – показатель рассматриваемого программного продукта,

ИК – показатель программного продукта конкурента.

У приложения Step Counter ежемесячная подписка – 7$ (20,51),

У Stepz – шагомер и счет калорий – 5$ (14,65)

Ц1 = (20,51 7,4) / 6,7 = 22,65 рублей

Ц2 = ( 7,4) / 5,3 = 20,4 рублей

Ц = (22,65 + 20,45) / 2 = 21,55 рублей

Пользователи получают базовый функционал бесплатно, а за дополнительные опции платят. При таком раскладе обычно только 0,5 процент пользователей покупает расширенную подписку. В таком случае при среднем количестве установок в год, равном 80 925,56, денежные поступления от покупки подписки будут составлять Пост.в год = 8 719,73 рублей за год.

Количество покупателей продукта необходимых для окупаемости расширения Пп вычисляется по формуле 7.18:

|  |  |  |
| --- | --- | --- |
|  |  | (7.18) |

где – полная себестоимость, руб.;

Пост.в год – денежные поступления от продажи подписки за год, руб.;

Ткп = 9194,9 / 21,55 = 426 покупателя.

Срок окупаемости приложения Tок вычисляется по формуле 7.19

|  |  |  |
| --- | --- | --- |
|  |  | (7.19) |

где – полная себестоимость, руб.;

Пост.в год – денежные поступления от месячной подписки на дополнительный функционал приложения за год, руб.

Ток = 9194,9 / 8719,73 = 1,05 года.

**7.4 Выводы по разделу**

В таблице 7.5 представлены результаты расчётов для основных показателей данной главы в краткой форме. Данные по экономическому обоснованию приведены в приложении Е.

Таблица 7.5 – Таблица экономических показателей

|  |  |
| --- | --- |
| Наименование показателя | Значение |
| Время разработки, мес. | 2,5 |
| Количество программистов, чел. | 1 |
| Основная заработная плата, руб. | 4000 |
| Дополнительная заработная плата, руб. |  |
| Отчисления в Фонд социальной защиты населения и по обязательному страхованию от несчастных случаев на производстве и профессиональных заболеваний, руб. |  |
| Расходы на материалы, руб. | 47,22 |
| Расходы на оплату машинного времени, руб. | 73,92 |
| Прочие прямые затраты, руб. | 800 |
| Накладные расходы, руб. | 800 |
| Себестоимость разработки программного средства, руб. | 7 858,90 |
| Расходы на сопровождение и адаптацию, руб. | 1 336,01 |
| Полная себестоимость, руб. | 9 194,91 |
| Годовые денежные поступления от продажи подписки, руб. | 8 719,73 |
| Срок окупаемости, лет | 1,05 |
| Количество покупателей для окупаемости, чел. | 426 |

Разработка программного средства, осуществляемая одним программистом в течение 2,5 месяцев, при заданных условиях обойдется в 9194,91 руб. Реализации данного программного средства будет приносить годовые денежные поступления в размере 8 719,73 рублей и окупиться при покупке 426 подписок через 1,05 года.

Заключение

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Лисункова А.Д.*

Пров.

*Бернацкий П.В.*

Н. контр.

*Нистюк О.А.*

Утв.

Смелов В.В.

Заключение

Лит.

Листов

1

*74217076, 2023*

В результате выполнения дипломного проектирования было разработано мобильное приложение для оценки активности.

В рамках работы было рассмотрено 2 аналогичных приложения, проанализированы их преимущества и недостатки, изучены возможности рассматриваемых программных продуктов.

Были спроектированы структура базы данных, диаграмма вариантов использования, диаграмма компонентов приложения, а также алгоритм получения достижений. Описаны механизмы построения сущностей базы данных, был выбран инструментарий для разработки приложения.

Были рассмотрены технологии, применяемые в северной и клиентской частях, определен стиль взаимодействия между севером и клиентом (REST), вместе с тем описаны методы взаимодействия с сервером. В клиентской части применена современная технология создания интерфейса (Jetpack Compose), показаны механизмы фоновой работы приложения.

Проводилось тестирование, которое показало, что разработанное мобильное приложение соответствует заявленным требованиям.

Также были рассмотрены возможные угрозы безопасности, методы защиты данных на стороне клиента и сервера.

Составлено руководство, подробно описывающее шаги развертывания серверной и клиентской частей, каким образом взаимодействовать с интерфейсом мобильного приложения.

Приведена рентабельность разработки проекта, а также рассчитана себестоимость и прибыль от реализации.

В соответствии с поставленными задачами и достигнутым результатом, можно сделать вывод, что дипломный проект выполнен и поставленная цель достигнута. В рамках дипломного проекта было разработано мобильное приложение.

Список использованных источников

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Лисункова А.Д.*

Пров.

*Бернацкий П.В.*

Н. контр.

*Нистюк О.А.*

Утв.

Смелов В.В.

Список использованных источников

Лит.

Листов

1

*74217076, 2023*

1 Кондаков В.Л., Шепляков А.С. Анализ мобильных приложений для повышения уровня двигательной активности студенческой молодежи [Электронный ресурс]. - Режим доступа: https://cyberleninka.ru/article/n/analiz-mobilnyh-prilozheniy-dlya-povysheniya-urovnya-dvigatelnoy-aktivnosti-studencheskoy-molodezhi/viewer – Дата доступа: 29.03.2023.

2 Приложение Step Counter. [Электронный ресурс]. – Режим доступа: https://play.google.com/store/apps/details?id=pedometer.steptracker.calorieburner.stepcounter – Дата доступа: 29.03.2023.

3. Приложение Stepz – шагомер и счет калорий [Электронный ресурс] – Режим доступа: https://apps.apple.com/ru/app/stepz – Дата доступа: 29.03.2023.

4 Урбанович, П. П. Защита информации методами криптографии, стеганографии и обфускации: учеб.-метод. пособие / П.П. Урбанович. – Минск : БГТУ, 2016. – 22с. – Дата доступа: 22.04.2023.

5 Информационные технологии и безопасность. Протокол защиты транспортного уровня (TLS): СТБ 34.101.65-2014. – Введ. 22.05.2014. – Минск: Госстандарт Республики Беларусь, 2004. – 56с. – Дата доступа: 22.04.2023.

6 Heroku Cloud Platform MSSQL Server [Электронный ресурс]– Режим доступа: https://devcenter.heroku.com/articles/mssql – Дата доступа: 04.05.2023.

7 Экономическое обоснование дипломных проектов [Электронный ресурс]. – Режим доступа: <https://elib.belstu.by/handle/123456789/14185>. ­– Дата доступа: 15.05.2023.

ПРИЛОЖЕНИЕ А Диаграмма компонентов приложения

ПРИЛОЖЕНИЕ Б Логическая схема базы данных

ПРИЛОЖЕНИЕ В Диаграмма вариантов использования

ПРИЛОЖЕНИЕ Г Блок-схема алгоритма получения достижений

ПРИЛОЖЕНИЕ Д Скриншот главного экрана приложения

ПРИЛОЖЕНИЕ Е Таблица экономических показателей

# ПРИЛОЖЕНИЕ Ж Листинг репозитория AchievementRepository на сервере

|  |
| --- |
| @Repository public interface AchievementRepository extends IRepository<Achievement>{   @Query(nativeQuery = true,value = "select \* from achievement" +  " join achievement\_user au on au.id\_achievement = achievement.id" +  " join user\_profile up on au.id\_user = up.id" +  " where up.id = :userId")  List<Achievement> getAchievementByUserId(String userId);   Achievement getAchievementsById(String achievementId);   @Query(nativeQuery = true, value = "select \* from achievement order by type\_of\_achieve")  List<Achievement> getAllAchievements();   @Modifying  @Transactional  @Query(nativeQuery = true, value = "exec achievement\_for\_user\_add :user\_id,:achievement\_id")  void addAchievementToUser(@Param("user\_id") String user,@Param("achievement\_id") String achievement);   @Query(nativeQuery = true, value = "select \* from achievement\n" +  " join achievement\_user au on au.id\_achievement = achievement.id\n" +  " join user\_profile up on au.id\_user = up.id \n" +  " where au.id\_achievement = :achievementId and up.id = :userId")  Achievement getOneAchievementByUserId(String userId, String achievementId);   @Query(nativeQuery = true, value = "select emblem from achievement\n" +  " join achievement\_user au on au.id\_achievement = achievement.id\n" +  " join user\_profile up on au.id\_user = up.id \n" +  " where au.id\_achievement = :achievementId and up.id = :userId")  String getAchievementEmblem(String userId, String achievementId);  } |

ПРИЛОЖЕНИЕ И Код интерфейса ParticularMapping

|  |
| --- |
| public interface ParticularMapping <E extends CommonEntity, D extends ParticularDto>{   E dtoToEntity(D dto);  D entityToDto(E entity);   default List<E> dtoesToEntities(List<D> dtoes) {  ArrayList<E> entities = new ArrayList<>();  for (D dto: dtoes) {  entities.add(dtoToEntity(dto));  }  return entities;  }   default List<D> entitiesToDtoes(List<E> entities){  ArrayList<D> dtoes = new ArrayList<>();  for(E entity: entities){  dtoes.add(entityToDto(entity));  }  return dtoes;  } } |

# ПРИЛОЖЕНИЕ К Листинг контроллера RelationshipController

|  |
| --- |
| @RestController @RequestMapping("relationship") public class RelationshipController {  @Autowired  RelationshipMapping relationshipMapping;  @Autowired  RelationService relationshipService;  @Autowired  UserMapping userMapping;  @Autowired  UserService userService;  @GetMapping("/user\_friends")  public List<UserDto> getFriendsUser(){  String currentUserId = SecurityContextHolder.getContext().getAuthentication().getName();  User user = userService.findById(currentUserId).get(); // todo exception handling  List<User> users = relationshipService.friendsAtUser(user.getId());  return userMapping.entitiesToDtoes(users);}  @GetMapping("/waiting\_users")  public List<UserDto> viewAllWaiting(){  String currentUserId = SecurityContextHolder.getContext().getAuthentication().getName();  User user = userService.findById(currentUserId).get(); // todo exception handling  List<User> users = relationshipService.waitingConfirmationAtUser(user.getId());  return userMapping.entitiesToDtoes(users);  }  @PostMapping("/add\_friend")  public String addFriend(@RequestParam String friendId){  String currentUserId = SecurityContextHolder.getContext().getAuthentication().getName();  User user = userService.findById(currentUserId).get(); // todo exception handling  String statusResponse = relationshipService.addingShipBetweenUsers(user.getId(), friendId);  return statusResponse;  }  @DeleteMapping("/delete\_friend")  public String deleteFriend(@RequestParam String friendId){  String currentUserId = SecurityContextHolder.getContext().getAuthentication().getName();  User user = userService.findById(currentUserId).get(); // todo exception handling  String statusResponse = relationshipService.removingShipBetweenUsers(user.getId(), friendId);  return statusResponse;  } } |

ПРИЛОЖЕНИЕ Л Листинг класса ProfileRepository

|  |
| --- |
| class ProfileRepository(  private val service: ProfileService,  private val sharedPreferences: SharedPreferences,  private val authInterceptor: AuthInterceptor ) {  init {  authInterceptor.retrieveToken = ::retrieveToken  }   suspend fun signIn(  idToken: String,  isExpiredCheck: () -> Boolean,  signInAction: suspend () -> String?  ): Status<User?> = handleRequest {  authInterceptor.silentSignIn = {  val token = signInAction()  if (token != null)  saveToken(token)   token  }  authInterceptor.isExpired = isExpiredCheck  val response = service.signIn(idToken.toRequestBody("text/plain".toMediaTypeOrNull()))  if (response.isSuccessful) {  val savingSuccessful = saveToken(idToken)  if (savingSuccessful)  Status.Success(response.body())  else  Status.Error(R.string.error\_shared\_prefs\_write)  } else  Status.Error(R.string.error\_unexpected)  }   fun signOut(): Status<Unit> {  val clearingSuccessful = sharedPreferences.edit().clear().commit()  return if (clearingSuccessful)  Status.Success(Unit)  else  Status.Error(R.string.error\_shared\_prefs\_clear)  }  fun isTokenExists() = retrieveToken() != null   suspend fun delete(): Status<Unit> = handleRequest {  val response = service.deactivateUser()  if (response.isSuccessful) {  val clearingSuccessful = sharedPreferences.edit().clear().commit()  if (clearingSuccessful)  Status.Success(Unit)  else  Status.Error(R.string.error\_shared\_prefs\_clear)  } else  Status.Error(R.string.error\_unexpected)  }  suspend fun getCurrentUser(  isExpiredCheck: () -> Boolean,  signInAction: suspend () -> String?,  overrideInterceptor: Boolean = true  ): Status<User?> = handleRequest(sharedPreferences, "getCurrentUser") {  if (overrideInterceptor) {  authInterceptor.silentSignIn = {  val token = signInAction()  if (token != null)  saveToken(token)   token  }  authInterceptor.isExpired = isExpiredCheck  }  val response = service.getCurrentUser()   if (response.isSuccessful)  Status.Success(response.body())  else  Status.Error(R.string.error\_unexpected)  }  suspend fun updateUser(  weight: Float,  height: Float,  stepsGoal: Int,  waterGoal: Float,  ): Status<User?> = handleRequest {  val response = service.updateUser(  weight = weight,  height = height,  stepsGoal = stepsGoal,  waterGoal = waterGoal  )   if (response.isSuccessful)  Status.Success(service.getCurrentUser().body())  else  Status.Error(R.string.error\_unexpected)  }  fun retrieveToken(): String? =  sharedPreferences.getString(TOKEN\_SHARED\_PREFERENCE, null)  private fun saveToken(idToken: String): Boolean = sharedPreferences  .edit()  .putString(TOKEN\_SHARED\_PREFERENCE, idToken)  .commit() } |

ПРИЛОЖЕНИЕ М Листинг StepRepository

|  |
| --- |
| private const val TAG = "StepsRepository"  class StepsRepository(  private val application: Context,  private val sharedPreferences: SharedPreferences,  private val stepsService: StepsService ) {  fun getSteps(): Flow<Int> {  val sensorManager: SensorManager =  application.getSystemService(Context.SENSOR\_SERVICE) as SensorManager  val sensor = sensorManager.getDefaultSensor(Sensor.TYPE\_STEP\_COUNTER)   val steps = callbackFlow {  val listener = object : SensorEventListener {  override fun onSensorChanged(event: SensorEvent?) {  event ?: return  event.values.firstOrNull()?.let {  runBlocking(Dispatchers.IO) {  val initialValue = getInitialValueSteps(it)  trySend(it.toInt() - initialValue)  .onFailure { /\*todo exception handling\*/ }  }  }  }   override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int) {}  }  sensorManager.registerListener(  /\* listener = \*/ listener,  /\* sensor = \*/ sensor,  /\* samplingPeriodUs = \*/ SensorManager.SENSOR\_DELAY\_NORMAL  )   awaitClose {  sensorManager.unregisterListener(listener)  }  }   return steps  }   //todo flow success failure  suspend fun sendCurrentSteps(): Status<Unit> = suspendCancellableCoroutine { continuation ->  val sensorManager: SensorManager =  application.getSystemService(Context.SENSOR\_SERVICE) as SensorManager  val sensor = sensorManager.getDefaultSensor(Sensor.TYPE\_STEP\_COUNTER)   val listener = object : SensorEventListener {  override fun onSensorChanged(event: SensorEvent?) {  event ?: return  val steps = event.values.firstOrNull()  if (steps != null) {  val result = runBlocking(Dispatchers.IO) {  val initialValue = getInitialValueSteps(steps)  val currentSteps = steps.toInt() - initialValue  handleRequest {  val response = stepsService.updateSteps(currentSteps)  if (response.isSuccessful)  Status.Success(Unit)  else  Status.Error(R.string.error\_unexpected)  }  }  sensorManager.unregisterListener(this)  continuation.resume(result)  }  }   override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int) {}  }  sensorManager.registerListener(  /\* listener = \*/ listener,  /\* sensor = \*/ sensor,  /\* samplingPeriodUs = \*/ SensorManager.SENSOR\_DELAY\_NORMAL  )  continuation.invokeOnCancellation { sensorManager.unregisterListener(listener) }  }   suspend fun sendEachStep(steps: Int): Status<Unit> = handleRequest {  val response = stepsService.updateSteps(steps)  Log.d(TAG, "sending $steps steps to server completed successfully: ${response.isSuccessful}")  if (response.isSuccessful)  Status.Success(Unit)  else  Status.Error(R.string.error\_unexpected)  }   private suspend fun getInitialValueSteps(sensorData: Float): Int {  val currentDateString = getDateString()  val entry = sharedPreferences.getString(INITIAL\_STEPS\_SHARED\_PREFERENCE, null)  return if (entry == null) {  val todayStepsFromServer = handleRequest {  Status.Success(stepsService.getStepsByDate(currentDateString).body()?.steps ?: 0)  }  val initialValue = sensorData.toInt() - ((todayStepsFromServer as? Status.Success)?.value ?: 0)  sharedPreferences  .edit()  .putString(INITIAL\_STEPS\_SHARED\_PREFERENCE, "$currentDateString $initialValue")  .apply()  initialValue  } else {  val pair = entry.split(" ")  val entryDate = pair[0]  val entryValue = pair[1].toInt()   if (entryDate != currentDateString) {  val initialValue = sensorData.toInt()  sharedPreferences  .edit()  .putString(INITIAL\_STEPS\_SHARED\_PREFERENCE, "$currentDateString $initialValue")  .apply()  initialValue  } else {  entryValue  }  }  }   suspend fun getStepsByType(type: StatsType): Status<Int?> = handleRequest(sharedPreferences, "getStepsByType\_${type.type}") {  val (isSuccessful, steps) = when (type) {  DAY -> {  val response = stepsService.getStepsByDate(getDateString())  response.isSuccessful to response.body()?.steps  }  WEEK -> {  val response = stepsService.getWeekSteps()  response.isSuccessful to response.body()  }  MONTH -> {  val response = stepsService.getMonthSteps()  response.isSuccessful to response.body()  }  HALF\_YEAR -> {  val response = stepsService.getHalfYearSteps()  response.isSuccessful to response.body()  }  }  if (isSuccessful)  Status.Success(steps)  else  Status.Error(R.string.error\_unexpected)  }   suspend fun getStepsListByType(type: StatsType): Status<List<Steps>> = handleRequest(sharedPreferences, "getStepsListByType\_${type.type}") {  val (isSuccessful, list) = when (type) {  DAY -> {  val response = stepsService.getStepsByDate(getDateString())  val isSuccessful = response.isSuccessful  val list = if (isSuccessful) {  val steps = response.body()  if (steps == null)  emptyList()  else  listOf(steps)  } else  emptyList()  isSuccessful to list  }  WEEK -> {  val response = stepsService.getWeekStepsList()  response.isSuccessful to (response.body() ?: emptyList())  }  MONTH -> {  val response = stepsService.getMonthStepsList()  response.isSuccessful to (response.body() ?: emptyList())  }  HALF\_YEAR -> {  val response = stepsService.getHalfYearStepsList()  response.isSuccessful to (response.body() ?: emptyList())  }  }  if (isSuccessful)  Status.Success(list.reversed())  else  Status.Error(R.string.error\_unexpected)  }   private fun getDateString(): String = SimpleDateFormat("yyyy-MM-dd", Locale.US).format(Date()) } |

# ПРИЛОЖЕНИЕ Н Листинг класса FriendsViewModel

|  |
| --- |
| @HiltViewModel class FriendsViewModel @Inject constructor(  private val friendsRepository: FriendsRepository ): ViewModel() {  val friends: MutableStateFlow<Status<List<User>>> = MutableStateFlow(Status.Success(emptyList()))  val incomingRequests: MutableStateFlow<Status<List<User>>> = MutableStateFlow(Status.Success(emptyList()))  val friendsFound: MutableStateFlow<Status<List<User>>> = MutableStateFlow(Status.Success(emptyList()))  fun getFriends() {  viewModelScope.launch(Dispatchers.IO) {  val result = friendsRepository.getFriends()  if (result is Status.Success)  friends.emit(result)}  }  fun searchFriends(query: String) {  viewModelScope.launch(Dispatchers.IO) {  val result = friendsRepository.searchFriends(query)  friendsFound.emit(result)}  }  fun getIncomingRequests() {  viewModelScope.launch(Dispatchers.IO) {  val result = friendsRepository.getIncomingRequests()  incomingRequests.emit(result)}  }  suspend fun addFriendById(id: String): Status<Unit> {  val result = friendsRepository.addFriendById(id)  if (result is Status.Success) {  getFriends()  friendsFound.emit(Status.Success(emptyList()))  getIncomingRequests()}  return result  }  suspend fun deleteFriendById(id: String): Status<Unit> {  val result = friendsRepository.deleteFriendById(id)  if (result is Status.Success) {  getFriends()  friendsFound.emit(Status.Success(emptyList()))  getIncomingRequests()}  return result  }  suspend fun getUserAndComparison(id: String): Pair<Status<User?>, Status<List<List<Steps>>>> = withContext(Dispatchers.IO) {  val user = async { friendsRepository.getUserById(id) }  val comparisons = async { friendsRepository.compareStepsById(id) }  user.await() to comparisons.await()} } |