МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
Кафедра компьютерных технологий и систем

КЛЫГА Глеб Геннадьевич

РАЗРАБОТКА СИСТЕМЫ МОНИТОРИНГА СОСТОЯНИЯ ЗДОРОВЬЯ С ИСПОЛЬЗОВАНИЕМ НОСИМЫХ УСТРОЙСТВ И МОБИЛЬНЫХ СЕНСОРОВ

Дипломная работа

Научный руководитель:
кандидат технических наук,
доцент кафедры технологий
программирования
И.С. Войтешенко

Допущена к защите
«____» _____ 2025 г.
Заведующий кафедрой компьютерных технологий и сетей, кандидат физико-математических наук, доктор педагогических наук профессор В.В. Казаченок

Минск, 2025

ОГЛАВЛЕНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ

API	программный интерфейс приложения (Application Programming Interface),
EHR	электронная медицинская карта (Electronic Health Record),
FHIR	Стандарт обмена медицинскими данными (Fast Healthcare Interoperability Resources),
IoMT	интернет медицинских вещей (Internet of Medical Things) ,
ML	Машинное обучение (Machine Learning),
PillPack	онлайн-сервис доставки лекарств с предварительной сортировкой,
RPM	удаленный мониторинг пациентов (Remote Patient Monitoring),
TENS	транскутанная электрическая стимуляция нервов,
TF-IDF	частота термина – обратная частота документа,
SQL	язык структурированных запросов (Structured Query Language),
SVD	сингулярное разложение (Singular Value Decomposition),
UML	унифицированный язык моделирования (Unified Modeling Language),
XML	расширяемый язык разметки (Extensible Markup Language).

РЕФЕРАТ

Дипломная работа, 57 с., 20 рис., 1 табл., 5 приложений, 13 источников.

Ключевые слова: МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, IOS, SWIFT, SWIFTUI, HEALTHKIT, НОСИМЫЕ УСТРОЙСТВА, МОНИТОРИНГ ЗДОРОВЬЯ, ДАТЧИКИ, МОБИЛЬНЫЕ СЕНСОРЫ

Объект исследования – методы и технологии разработки приложений для мониторинга здоровья под мобильную платформу с использованием носимых устройств и мобильных сенсоров. Современные подходы к сбору, анализу и визуализации медицинских данных.

Предмет исследования – применение фреймворков SwiftUI и HealthKit для создания системы мониторинга здоровья. Способы интеграции с носимой электроникой (Apple Watch, фитнес-трекеры) и обработки данных с датчиков.

Цели работы – исследовать возможности системы мониторинга для работы с медицинскими данными. Разработать приложение для мониторинга ключевых показателей здоровья (пульс, уровень кислорода, активность, сон). Реализовать визуализацию данных и систему уведомлений о критических состояниях.

Методы исследования – изучение документации Apple, анализ современных методов обработки данных с датчиков, исследование UX/UI-подходов в медицинских приложениях.

Результат – разработана система для мониторинга здоровья с поддержкой Apple Watch. Система сбора и анализа данных с возможностью построения графиков и формирования отчетов. Механизм оповещений при отклонении показателей от нормы.

Область применения – персональный мониторинг здоровья и фитнес-трекинг. Приложение может быть использовано для самостоятельного отслеживания состояния организма, а также как вспомогательный инструмент для врачей.

РЭФЕРАТ

Дыпломная работа, 57 с., 20 мал., 1 табл., 5 дадаткаў, 13 крыніц.

Ключавыя словы: МАБІЛЬНЫ ДАДАТАК, IOS, SWIFT, SWIFTUI, HEALTHKIT, НОСІМЫЯ ПРЫСТРОІ, МАНІТОРЫНГ ЗДАРОЎЯ, ДАТЧЫКІ, МАБІЛЬНЫЯ СЕНСАРЫ

Аб'ект даследавання – метады і тэхналогіі распрацоўкі дадаткаў для маніторынгу здароўя пад мабільную платформу з выкарыстаннем носімых прыстрояў і мабільных сенсараў. Сучасныя падыходы да збору, аналізу і візуалізацыі медыцынскіх даных.

Прадмет даследавання – прымяненне фрэймворкаў SwiftUI і HealthKit для стварэння сістэмы маніторынгу здароўя. Спосабы інтэграцыі з носімай электронікай (Apple Watch, фітнэс-трэкеры) і апрацоўкі даных з датчыкаў.

Мэты працы – даследаваць магчымасці сістэмы маніторынгу для працы з медыцынскімі данымі. Распрацаваць дадатак для маніторынгу ключавых паказчыкаў здароўя (пульс, узровень кіслароду, актыўнасць, сон). Рэалізаваць візуалізацыю даных і сістэму апавяшчэнняў аб крытычных станах.

Метады даследавання – вывучэнне дакументацыі Apple, аналіз сучасных метадаў апрацоўкі даных з датчыкаў, даследаванне UX/UI-падыходаў у медыцынскіх дадатках.

Выніку – распрацавана сістэма для маніторынгу здароўя з падтрымкай Apple Watch. Сістэма збору і аналізу даных з магчымасцю пабудовы графікаў і фарміравання справаздач. Механізм апавяшчэнняў пры адхіленні паказчыкаў ад нормы.

Вобласць прымянення – персанальны маніторынг здароўя і фітнэс-трэкінг. Дадатак можа быць выкарыстаны для самастойнага адсочвання стану арганізма, а таксама як дапаможны інструмент для лекараў.

ABSTRACT

Graduate work, 57 p., 20 images, 1 tables, 5 appendixes, 13 sources.

Keywords: MOBILE APPLICATION, IOS, SWIFT, SWIFTUI, HEALTHKIT, WEARABLE DEVICES, HEALTH MONITORING, SENSORS, MOBILE SENSORS

Research object – methods and technologies for developing health monitoring applications for mobile platforms using wearable devices and mobile sensors. Modern approaches to collecting, analyzing, and visualizing medical data.

Research subject – the use of SwiftUI and HealthKit frameworks to create a health monitoring system. Methods of integration with wearable electronics (Apple Watch, fitness trackers) and processing data from sensors.

Objectives – to explore the capabilities of a monitoring system for working with medical data. Develop an application for tracking key health indicators (heart rate, oxygen level, activity, sleep). Implement data visualization and a notification system for critical conditions.

Research methods – studying Apple's documentation, analyzing modern methods of sensor data processing, researching UX/UI approaches in medical applications.

Results – a health monitoring system with Apple Watch support has been developed. A data collection and analysis system with the ability to generate charts and reports. An alert mechanism for abnormal readings.

Application area – personal health monitoring and fitness tracking. The application can be used for self-tracking of body conditions, as well as an auxiliary tool for doctors.

ВВЕДЕНИЕ

Актуальность темы дипломной работы обусловлена стремительным развитием цифровых технологий в медицине и растущим спросом на персонализированные решения для мониторинга здоровья. В условиях увеличения нагрузки на систему здравоохранения и роста хронических заболеваний особую важность приобретают технологии удаленного контроля физиологического состояния пациентов. Разработка мобильных приложений, интегрированных с носимых устройствами, позволяет не только улучшить качество жизни пользователей, но и способствует раннему выявлению потенциальных угроз для здоровья.

Целью работы является создание системы мониторинга состояния здоровья с использованием носимых устройств и мобильных сенсоров, обеспечивающей:

- непрерывный сбор и анализ медицинских данных;
- обнаружение аномалий в реальном времени;
- прогнозирование потенциальных рисков для здоровья;
- удобный интерфейс для пользователей и медицинских специалистов.

Задачи исследования:

- анализ современных тенденций в области цифрового здравоохранения и персонализированной медицины;
- исследование существующих аппаратно-программных решений для мониторинга здоровья;
- разработка архитектуры мобильного приложения с использованием паттерна MVVM-C;
- реализация алгоритмов анализа медицинских данных;
- тестирование и оценка эффективности разработанного решения.

Объект исследования – процесс разработки программного обеспечения для мониторинга физиологических показателей.

Предмет исследования – методы и алгоритмы обработки медицинских данных в режиме реального времени.

В первой главе работы рассматриваются актуальность и значимость разработки системы мониторинга здоровья, анализируются современные тенденции в области цифровой медицины и удаленного наблюдения за пациентами (RPM). Особое внимание уделяется экономическому эффекту от внедрения подобных систем.

Вторая глава посвящена анализу существующих решений, включая:

- сравнительный анализ носимых устройств для мониторинга здоровья;
- исследование стандартов обмена медицинской информацией;
- выбор оптимальных технических протоколов передачи данных;
- выявление недостатков современных платформ.

Третья глава содержит:

- проектирование архитектуры приложения;
- описание основных классов и их взаимодействия;
- реализацию алгоритмов анализа данных;
- интеграцию машинного обучения для прогнозирования рисков.

В четвертой главе представлены:

- процесс реализации мобильного приложения;
- тестирование функционала;
- оценка эффективности системы.

Научная новизна данной работы состоит в разработке комплексного подхода к анализу медицинских данных, основанного на сочетании статистических методов и алгоритмов машинного обучения. Особое внимание уделено оптимизации вычислительных алгоритмов для эффективной работы на мобильных устройствах, что позволяет использовать систему в условиях ограниченных ресурсов. Кроме того, реализован механизм контекстных уведомлений, формируемых на основе индивидуальных показателей пользователя.

Практическая значимость работы заключается в том, что разработанное приложение может использоваться для персонального мониторинга состояния здоровья, обеспечивая пользователям доступ к актуальной информации о собственных показателях. Помимо этого, система представляет интерес для медицинских учреждений, поскольку может служить эффективным инструментом удалённого наблюдения за пациентами. Предложенные технические решения обладают высокой степенью универсальности и могут быть интегрированы в существующие цифровые платформы здравоохранения, расширяя их функциональность и повышая качество медицинского обслуживания.

Методологическую основу исследования составили:

- анализ научной литературы;
- сравнительный анализ существующих решений;
- методы объектно-ориентированного проектирования;
- технологии машинного обучения;
- принципы юзабилити-тестирования.

ГЛАВА 1 АКТУАЛЬНОСТЬ ПРОБЛЕМЫ РАЗРАБОТКИ СИСТЕМЫ МОНИТОРИНГА СОСТОЯНИЯ ЗДОРОВЬЯ

1.1 Персонализированная медицина как глобальный тренд

1.1.1 Рост спроса на самоконтроль здоровья

Индустрия медицинских технологий – это динамичная сила. Это сфера, где смартфоны могут диагностировать болезни, носимые устройства отслеживают жизненно важные показатели, а телемедицина преодолевает географические разрывы.

Реактивная медицина (традиционная) – подход, при котором лечение начинается после появления симптомов болезни (например, терапия гипертонии только при критических показателях давления).

Превентивная медицина – упор на предупреждение заболеваний через раннюю диагностику, мониторинг рисков и коррекцию образа жизни.

Носимые устройства – например, умные часы и фитнес-браслеты – умеют не только считать шаги. Теперь они круглосуточно следят за здоровьем: измеряют пульс, анализируют сон и находят проблемы. Как будто на руке у вас личный помощник.

Эти устройства работают с **телемедициной**, и врачи могут удаленно наблюдать за пациентами. Это удобно для людей с хроническими болезнями – они получают помощь, не посещая клинику.

Принимать таблетки по расписанию бывает сложно. Но умные дозаторы и браслеты с напоминаниями решают эту проблему. Они подают сигнал, когда нужно выпить лекарство, и снижают риск пропустить прием.

Для борьбы с хронической болью есть носимые устройства **с TENS-технологией**. Их носят под одеждой, они незаметны и помогают без таблеток.

Такие устройства не просто собирают данные, но и дают полезные рекомендации. Специальные приложения обрабатывают информацию и предлагают советы по здоровью.

1.1.2 Тенденции цифрового здравоохранения

Если заглянуть в будущее, то хрустальный шар технологий здравоохранения открывает интригующие возможности (см. рисунок 1.1). К 2030 году мировой рынок цифровых технологий в сфере здравоохранения взлетит до более 585 млрд. долларов США. Телемедицина, которая испытала беспрецедентный рост во время пандемии COVID-19, останется здесь и в дальнейшем. Ожидается, что к 2030 году мировой рынок телемедицины достигнет 459,8 млрд. долларов.

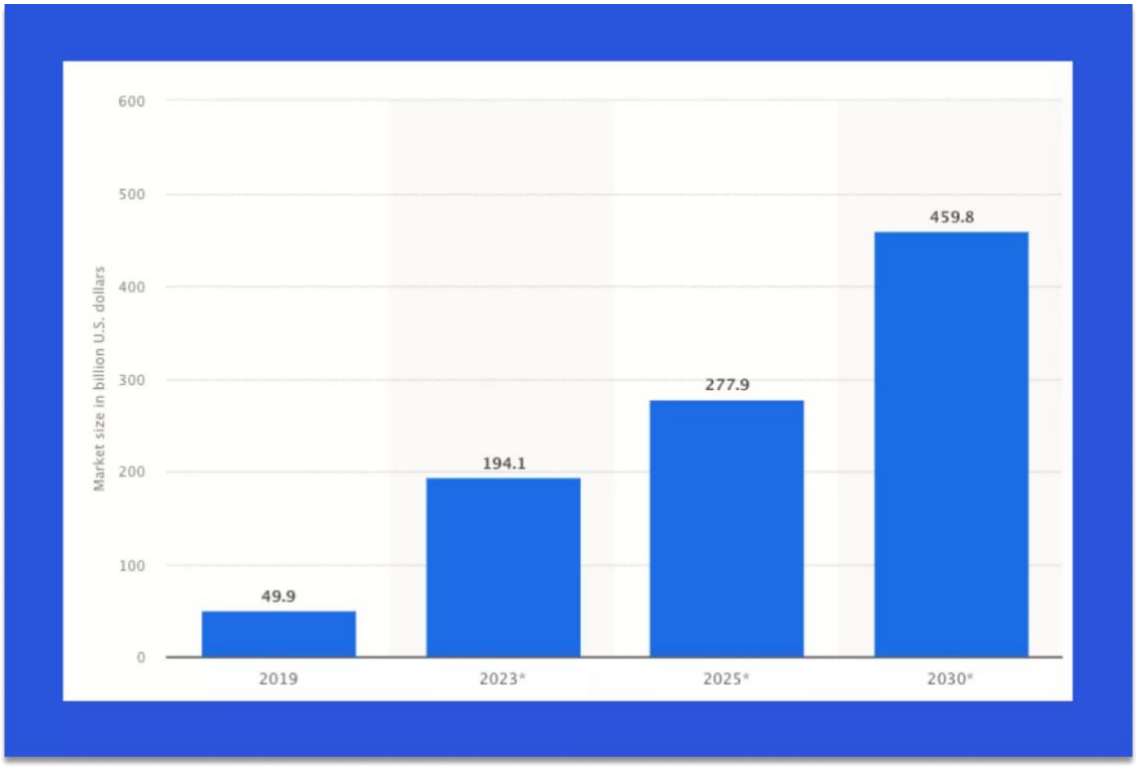

Рисунок 1.1 - График роста телемедицины к 2030 году

1.2 Технологические вызовы и потребности пользователей

1.2.1 Интернет медицинских вещей (IoMT)

IoMT представляет собой сеть взаимосвязанных медицинских устройств, включающих как носимые устройства (например, смарт-часы), так и сложное диагностическое оборудование (МРТ, КТ и др.). Данная система обеспечивает сбор, обработку и обмен медицинскими данными между пациентами и медицинскими специалистами.

Ключевые преимущества IoMT:

1. Дистанционный мониторинг состояния пациентов

Медицинские работники получают возможность непрерывного контроля жизненно важных показателей, динамики хронических заболеваний

и общего состояния пациентов. Система позволяет оперативно выявлять критические изменения показателей, что способствует своевременному медицинскому вмешательству. В результате снижается частота госпитализаций и повышается эффективность лечения.

2. Принятие клинических решений на основе данных IoMT

Генерирует значительные массивы медицинских данных, которые могут подвергаться анализу с применением алгоритмов искусственного интеллекта и методов машинного обучения. Это позволяет медицинскому персоналу принимать более обоснованные решения при разработке тактики лечения и планировании медицинских вмешательств.

3.Оптимизация затрат в системе здравоохранения

Внедрение IoMT способствует снижению экономических нагрузок на медицинские учреждения за счёт:

- сокращения числа повторных госпитализаций;
- рационального распределения медицинских ресурсов;
- уменьшения необходимости проведения очных консультаций благодаря возможностям телемониторинга.

Проблемные аспекты внедрения IoMT

Несмотря на высокую перспективность технологии, её практическая реализация сопровождается рядом серьёзных вызовов. Одним из ключевых является обеспечение конфиденциальности и безопасности медицинских данных, особенно в условиях активного обмена информацией между системами. Кроме того, существует острая необходимость в разработке единых отраслевых стандартов, которые позволили бы медицинским устройствам эффективно взаимодействовать между собой.

Не менее актуальными остаются вопросы кибербезопасности, особенно при передаче чувствительной информации, требующей надёжной защиты от несанкционированного доступа и возможных атак. Указанные аспекты требуют особого внимания при интеграции IoMT в систему здравоохранения.

1.2.2 Удаленный мониторинг пациентов (RPM)

RPM – это сбор и передача поставщикам данных о состоянии здоровья пациента с помощью подключенных устройств вне обычных условий оказания медицинской помощи.

RPM позволяет медицинским работникам отслеживать информацию о состоянии здоровья и жизненно важных показателях пациента на расстоянии. Пациенты используют нательные датчики, манжеты для измерения артериального давления, глюкометры или даже приложения для здоровья на своих телефонах, чтобы измерять и регистрировать показатели

своего здоровья. Эти устройства отправляют данные через интернет в центральную систему мониторинга или облачную платформу. Медицинские работники имеют доступ к этим данным на платформе. Они используют эту информацию для мониторинга здоровья пациента в режиме реального времени и анализа тенденций с течением времени.

RPM имеет много преимуществ, среди которых:

- **улучшенный мониторинг**. RPM позволяет осуществлять непрерывный мониторинг пациентов. Это дает более полное представление о состоянии их здоровья по сравнению с визитами в клинику;
- **своевременное вмешательство**. Это позволяет медицинским работникам выявлять проблемы со здоровьем или изменения на ранних стадиях. Это приводит к более быстрому вмешательству и лучшим результатам;
- **лучшее распределение ресурсов**. На основе данных RPM медицинский работник может определить приоритетность пациентов, требующих немедленного внимания.
- **данные для исследований**. Собранные данные можно использовать в исследовательских целях. Это помогает совершенствовать медицинские знания и методы лечения;
- **снижение расходов**. Уменьшение количества госпитализаций и повторных госпитализаций – это один из способов, как RPM может уменьшить расходы на здравоохранение.

1.3 Социально-экономическая значимость

1.3.1 Инструменты автоматизации

Автоматизация имеет ряд преимуществ, такие как:

Повышение эффективности работы медицинского персонала. Автоматизация таких задач, как ведение электронных медицинских карт (EHR), назначение лекарств, обработка лабораторных анализов и напоминания о приёме препаратов, значительно сокращает время, которое врачи и медсёстры тратят на административную работу. Например, системы автоматического документирования, такие как Nuance Dragon Medical, используют технологии распознавания речи, чтобы врачи могли быстро заполнять истории болезни, экономя до 30% рабочего времени.

Снижение количества медицинских ошибок. По данным ВОЗ, медицинские ошибки являются одной из ведущих причин смертности в мире. Автоматизированные системы минимизируют человеческий фактор, который часто приводит к неправильному назначению дозировок, ошибочным

диагнозам или несвоевременному оказанию помощи. Например, автоматические дозаторы лекарств (такие как PillPack) исключают риск передозировки, а системы поддержки принятия клинических решений (CDSS) анализируют данные пациента и предлагают врачу оптимальные варианты лечения на основе актуальных медицинских протоколов.

Улучшение координации между специалистами. Интегрированные платформы, такие как Epic Systems или Cerner, позволяют разным медицинским учреждениям обмениваться данными в режиме реального времени. Это особенно важно для пациентов с хроническими заболеваниями, которые наблюдаются у нескольких врачей. Автоматизация обеспечивает целостность и актуальность информации, предотвращая дублирование анализов и процедур.

Персонализация лечению. Современные алгоритмы машинного обучения анализируют огромные массивы данных (от генетических тестов до показаний носимых устройств) и помогают врачам подбирать индивидуальные схемы терапии. Например, платформы IBM Watson Health используют ИИ для анализа медицинской литературы и рекомендации персонализированных методов лечения онкологических заболеваний.

1.3.2 Экономический эффект автоматизации в здравоохранении

Внедрение автоматизированных систем в медицину не только улучшает качество помощи, но и оказывает значительное влияние на экономику здравоохранения. Далее перечислены ключевые аспекты экономического эффекта.

Сокращение затрат на лечение достигается за счёт внедрения автоматизированных систем, которые позволяют значительно снизить расходы. Это становится возможным благодаря снижению числа ошибочных диагнозов и избыточных процедур, более эффективному использованию ресурсов — например, автоматизированное планирование операций помогает сократить простои операционных. Кроме того, удалённый мониторинг состояния пациентов (RPM) позволяет своевременно выявлять осложнения и тем самым предотвращать дорогостоящие госпитализации.

Исследования показывают, что внедрение EHR-систем может сократить затраты на 10–20%, а телемедицинские платформы – на 30% за счёт снижения количества повторных госпитализаций.

Улучшение коммуникации и скорости оказания помощи. Интегрированные системы электронных сообщений (например, TigerConnect) ускоряют взаимодействие между врачами, лабораториями и аптеками. Это особенно критично в экстренных случаях, где каждая минута влияет на исход

лечения. Автоматические уведомления о критических показателях (например, аномальных результатах анализов) позволяют врачам оперативно реагировать.

Повышение удовлетворённости пациентов и персонала. Пациенты получают более быстрый доступ к результатам анализов, онлайн-консультациям и персонализированным рекомендациям. Врачи меньше времени тратят на бумажную работу и больше – на непосредственное общение с пациентами, что снижает профессиональное выгорание.

Поддержка медицинских исследований. Автоматизированные системы собирают и структурируют огромные объёмы данных, которые можно использовать для клинических исследований. Например, платформы на основе ИИ помогают идентифицировать группы риска для новых заболеваний или анализировать эффективность лекарств в реальном времени.

Пример успешного внедрения. Больница Mayo Clinic внедрила систему автоматической расшифровки рентгеновских снимков с помощью ИИ (Lunit INSIGHT)[2]. Это сократило время диагностики на 40% и повысило точность выявления патологий на 15%.

ВЫВОДЫ

Проведенный анализ актуальности разработки системы мониторинга состояния здоровья позволил выявить несколько ключевых тенденций, подтверждающих востребованность и перспективность данного направления:

Рост персонализированного подхода в медицине стал глобальным трендом, смещающим фокус с лечения заболеваний на их профилактику. Это требует новых технологических решений для постоянного контроля показателей здоровья, что подтверждается растущим спросом на средства самоконтроля среди населения.

Развитие цифрового здравоохранения демонстрирует устойчивую динамику, где системы удаленного мониторинга пациентов (RPM) занимают центральное место. Современные технологии позволяют не только собирать данные, но и анализировать их с помощью алгоритмов ИИ, обеспечивая раннюю диагностику потенциальных проблем.

Экономическая эффективность автоматизированных систем мониторинга доказана многочисленными исследованиями. Внедрение подобных решений снижает нагрузку на медицинские учреждения за счет предотвращения критических состояний и оптимизации процессов наблюдения за пациентами.

Проведенный анализ подтверждает, что разработка системы мониторинга здоровья соответствует современным запросам как со стороны пользователей, заинтересованных в сохранении своего здоровья, так и со стороны медицинских организаций, нуждающихся в эффективных инструментах профилактики. Это создает прочную основу для дальнейшего проектирования и реализации предложенного решения.

ГЛАВА 2 АНАЛИЗ СОСТОЯНИЯ ПРОБЛЕМЫ И СУЩЕСТВУЮЩИХ РАЗРАБОТОК

2.1 Программные платформы

2.1.1 Сравнительный анализ носимых устройств для мониторинга здоровья

Для сравнительного анализа использовалась линейка смарт-часов трех видов это Apple Watch Series, Fitbit, Withings. В первую очередь проводилось сравнение не только между самими устройствами, которые мы носим в повседневной жизни, но и с профессиональными решениями, такими как глюкометры, ЭКГ – датчики, пульсоксиметры.

После длительного исследования учебного материала [6] были выявлены особенности в линейке смарт-часов и в тоже время проведен обзор профессиональных решений. Составлена таблица где описаны ключевые особенности каждого предлагаемого устройства (см. риунок 2.1),.

Рисунок 2.1 - Сравнительный анализ носимых устройств для мониторинга здоровья

2.1.2 Стандарты обмена медицинской информации

HL7 (Health Level 7 - "Седьмой уровень") представляет собой международный стандарт, регламентирующий обмен, управление и интеграцию электронной медицинской информации. Наиболее современной версией данного стандарта является FHIR (Fast Healthcare Interoperability Resources), разработанный для решения проблем взаимодействия медицинских информационных систем.

Основная цель FHIR заключается в упрощении реализации механизмов обмена данными между медицинскими приложениями при сохранении целостности информации. Стандарт использует существующие логические и теоретические модели, обеспечивая последовательный и строгий подход к интеграции систем здравоохранения.

Актуальная версия стандарта – FHIR 4.0.1 была выпущена в октябре 2019г. и в настоящее время активно внедряется в медицинских информационных системах США, Европы и России.

Ключевые принципы FHIR включают:

- Ориентацию на современные веб-технологии.
- Практическую реализуемость для разработчиков.
- Фокус на наиболее значимых концептах (принцип 80/20).
- Открытость стандарта и развитие в парадигме Open Source.

Архитектура и основные компоненты FHIR

Фундаментальной составляющей стандарта FHIR является ресурс — это структурированная единица, предназначенная для представления и обмена медицинской информацией. Каждый ресурс отражает конкретную реальную сущность, такую как пациент, визит или результат медицинского обследования, и служит цифровым аналогом этих объектов в информационной системе. В зависимости от своего назначения и содержимого, ресурсы классифицируются на клинические, административные, финансовые и технические.

Каждый ресурс характеризуется:

- стандартизированным набором атрибутов;
- механизмом расширений для специализированного контекста;
- возможностью установления связей между ресурсами.

Механизмы взаимодействия и обмена данными. Спецификация FHIR предусматривает несколько подходов к организации обмена данными:

- обмен отдельными ресурсами;
- групповая передача связанных ресурсов;
- поддержка различных протоколов взаимодействия.

Основные достоинства FHIR заключаются в его открытой лицензионной политике и наличии развитой экосистемы инструментов, что облегчает его внедрение и адаптацию. Стандарт основан на современных веб-технологиях, таких как XML, JSON, REST и HTTP, что делает его совместимым с существующими ИТ-инфраструктурами. Важным преимуществом является гибкость системы — разработчики могут расширять ресурсы в соответствии с конкретными потребностями. FHIR предлагает унифицированный подход к работе со словарями и справочниками, а также предоставляет возможность профилирования как ресурсов, так и операций. Дополнительно реализована поддержка истории изменений, расширенные механизмы поиска и активное профессиональное сообщество, регулярно организующее образовательные мероприятия и обмен опытом.

Ограничения и проблемы внедрения. Несмотря на широкие возможности стандарта, его внедрение сопряжено с рядом трудностей. Одним из основных недостатков является ограниченность обязательного набора элементов в ресурсах, что может затруднять полное представление медицинской информации. Кроме того, стандарт предъявляет жёсткие требования к использованию словарей и терминологических систем, что не всегда удобно при интеграции с уже существующими решениями. Существенные сложности также вызывает адаптация FHIR к национальным особенностям систем здравоохранения.

2.1.3 Выбор технического протокола передачи данных

BLE (Bluetooth Low Energy) – это технология беспроводной связи с низким энергопотреблением для обмена данными на небольших расстояниях. Она была создана для устройств, которым необходимо длительный период времени поддерживать между собой связь.

NFC (Near Field Communication) – это технология беспроводной передачи данных с устройства на устройство в радиусе до 10 см. Если оба гаджета оснащены чипами NFC, то, чтобы подключить их друг к другу, не потребуется скачивать никаких приложений. Для обмена информацией их нужно просто близко поднести друг к другу для синхронизации.

Для определения нужного протокола данному проекту ориентировались на несколько ключевых аспектов, таких как:

Энергопотребление. BLE разработан для минимального расхода энергии, что делает его идеальным для носимых устройств, таких как фитнес-браслеты и умные часы. NFC, в свою очередь, бывает двух типов: пассивные метки почти не тратят энергию, но активные устройства (например, смартфоны) потребляют больше, чем BLE при постоянной работе.

Скорость передачи данных. BLE, особенно в версии Bluetooth 5.0 и выше, обеспечивает скорость до 1–2 Мбит/с, что удобно для частой передачи небольших данных, например показаний пульса или температуры. NFC работает медленнее – до 424 кбит/с, зато лучше подходит для быстрого обмена информацией, как при считывании данных с медицинской карты.

Дальность связи. BLE может работать на расстоянии до 100 метров в идеальных условиях, хотя на практике это чаще 10–30 метров, что удобно для непрерывного мониторинга. NFC же требует близкого контакта – максимум 10 см, а обычно 1 – 4 см, что ограничивает его применение, но повышает безопасность.

Безопасность. NFC считается более защищённым из-за крайне малой дистанции работы – перехватить сигнал сложнее. Кроме того, он поддерживает шифрование. BLE тоже может быть безопасным, но требует дополнительных настроек, таких как защищённое сопряжение, и при неправильной конфигурации уязвим к атакам.

Совместимость и удобство. BLE поддерживается большинством современных смартфонов, планшетов и носимых устройств, обеспечивая гибкость в использовании. NFC тоже есть во многих телефонах, но из-за необходимости подносить устройства вплотную его применение ограничено.

NFC лучше всего подходит для мгновенного считывания данных (медкарты, одноразовые датчики) и для систем, где важна безопасность и минимальное энергопотребление. BLE можно использовать в качестве непрерывного мониторинга (ЭКГ, уровень кислорода, активность), а также с устройствами с автономным питанием таких как часа, датчики.

В нашем случае сделан выбор в сторону BLE для подключения к носимым устройствам и мобильным сенсором для непрерывного мониторинга и возможности собственного шифрования информации для сохранности данных.

2.1.4 Недостатки существующих систем

Первое, что сразу хочется выделить это закрытость API и Apple HealthKit и Google Fit предоставляют доступ к данным здоровья, но с ограничениями (например, нельзя считать сырые данные ЭКГ на Apple Watch без одобрения Apple). Пользователи разных платформ не могут обмениваться данными. Пользователи Android и iOS часто не могут обмениваться данными между устройствами (например, данные с Huawei Watch сложно передать в Apple Health).

Ограниченная аналитика данных, а также отсутствие уведомлений при превышении минимально допустимого порога показателей здоровья. Невозможность настройки уведомлений под свои нужды. Нельзя создавать свои собственные правила например: "Если пульс > 120 более 5 минут и уровень кислорода < 90% – отправить SMS врачу". Нет ML – аналитики в реальном времени, то есть пропускают аномалии (например, предынфарктное состояние), так как алгоритмы слишком простые.

2.1.5 Выбор технического протокола передачи данных

Проведенное исследование нормативных требований к обработке медицинских данных выявило критическую важность соблюдения международных стандартов информационной безопасности. Соответствие современным регуляторным нормам представляет собой не только обязательное юридическое требование, но и ключевой элемент стратегии управления рисками, позволяющий избежать существенных финансовых потерь и репутационного ущерба.

HIPAA — американский стандарт, устанавливающий строгие требования к обеспечению безопасности медицинских данных. Он предусматривает обязательное шифрование информации как при передаче, так и при хранении, наличие эффективной системы контроля и разграничения доступа, ведение подробных журналов аудита, а также организацию механизмов аварийного восстановления. Для практической реализации этих требований применяются регулярные оценки рисков, внедрение современных криптографических протоколов, многоуровневая авторизация и мониторинг активности доступа к данным.

HITRUST CSF представляет собой более комплексную систему, которая объединяет в себе усовершенствованные механизмы защиты информации, непрерывное управление рисками и интеграцию с другими международными стандартами. Процесс внедрения включает этап первичной самооценки соответствия, разработку плана сертификации, постоянный мониторинг уровня безопасности, а также использование многофакторной аутентификации и end-to-end шифрования.

GDPR — европейский регламент, основное внимание в котором уделяется защите персональных данных пользователей. Среди его ключевых положений — необходимость получения явного согласия на обработку данных, соблюдение принципа минимально необходимого сбора информации, обеспечение возможности переноса данных между системами и предоставление права на удаление персональной информации («право на забвение»).

Был выбран американский стандарт HIPAA, который покрывает все необходимые задачи данного проекта по шифрованию данных и имеет легкость внедрения в создаваемую систему мониторинга состояния здоровья. Так как имеет уже готовую реализацию с помощью встроенной библиотеки от **Google Tink**, которая поддерживает как раз экосистему Android/iOS и содержит готовые криптоалгоритмы шифрования данных.

ВЫВОДЫ

Проведенный анализ существующих решений и технологических аспектов разработки системы мониторинга здоровья позволил сделать следующие ключевые выводы.

Технологический выбор протоколов передачи данных был обоснован комплексным сравнением характеристик:

- BLE доказал свое преимущество для непрерывного мониторинга благодаря оптимальному балансу энергопотребления, дальности связи и скорости передачи данных;
- выбранный протокол полностью соответствует требованиям носимых устройств с батарейным питанием;
- реализованные механизмы шифрования данных обеспечивают необходимый уровень безопасности.

Критический анализ существующих платформ выявил существенные ограничения:

- проблемы кросс-платформенной совместимости данных;
- недостаточная гибкость систем оповещения;
- примитивные алгоритмы анализа без использования современных ML-технологий;
- жесткие ограничения доступа к "сырым" медицинским данным.

Сравнительный анализ носимых устройств показал:

- существенный разрыв между потребительскими и профессиональными медицинскими устройствами;
- необходимость разработки унифицированного интерфейса для работы с разными типами датчиков;
- важность поддержки открытых стандартов обмена медицинской информацией.

Предложенные технические решения включают:

- использование открытых стандартов FHIR для хранения и обмена данными;
- применение TinyML-моделей для обработки данных на edge-устройствах;
- гибкую систему правил для генерации уведомлений;
- альтернативные каналы синхронизации данных (NFC для офлайн-сценариев).

Полученные результаты легли в основу технических требований к разрабатываемой системе и определили ключевые направления для ее реализации. Выявленные ограничения существующих решений будут учтены при проектировании архитектуры системы в следующих главах работы.

ГЛАВА 3. ПРОЕКТИРОВАНИЕ МОБИЛЬНОГО ПРИЛОЖЕНИЯ

3.1 Анализ и сравнение прототипа приложения и аналогов

Тематика приложения является крайне актуальной в современном мире, однако исследование существующих приложений показало, что некоторые приложения перестали поддерживаться разработчиками, другие же недоступны в нашем регионе, а среди оставшихся доступных нет таких, которые могли бы полноценно раскрыть возможности мониторинга состояния здоровья.

Apple Health

Рисунок 3.1 - Демонстрация работы с приложением Apple Health

В стандартном приложении от Apple можно собирать персональную медицинскую карту, отслеживать свою ежедневную активность, питание, сон, менструальный цикл, громкость окружающей среды, наушников и многое другое. В Apple Health можно подтягивать данные из других приложений для здоровья и собирать единую базу (см. рисунок 3.1). Однако доступен только для Apple техники.

FatSecret

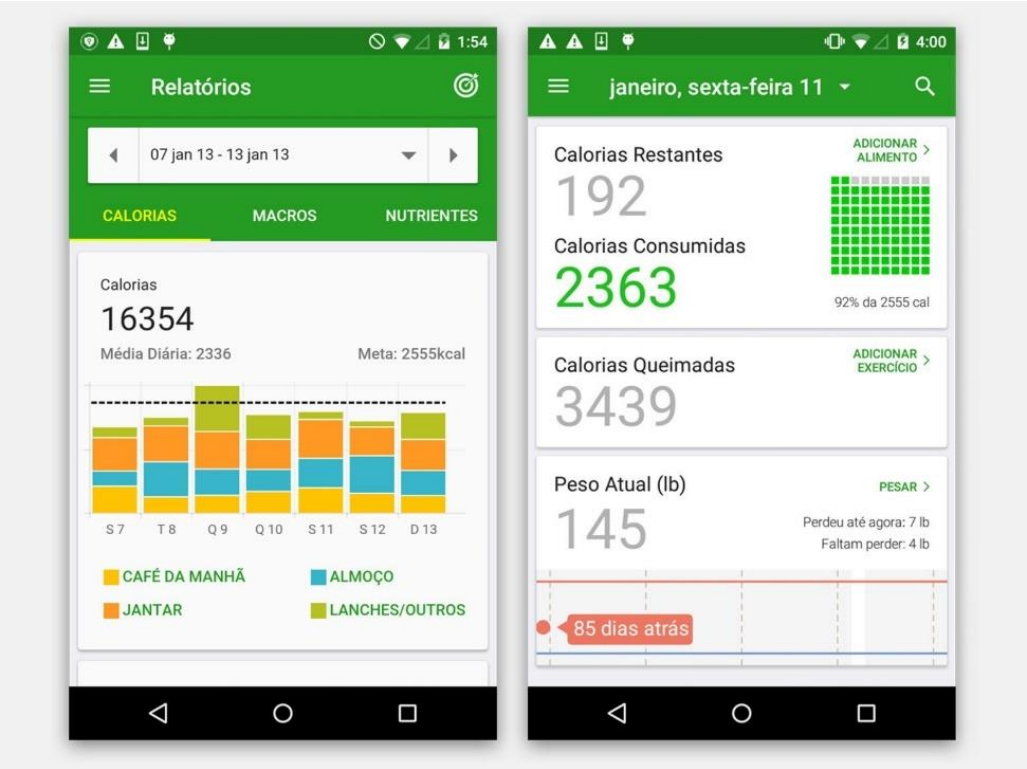

Рисунок 3.2 - Демонстрация работы с приложением FatSecret

Приложение для ведения дневника питания FatSecret (см. рисунок 3.2). FatSecret умеет распознавать еду по фото и считывать штрих-коды. Помимо белков, жиров и углеводов, программа помогает рассчитывать, сколько вы употребили сахара, соли, клетчатки и холестерина. Приложение формирует удобную отчётность о питании и активности за день, за текущую и прошлую недели.

Большинство функций приложения бесплатны. Пользователям с Premium-аккаунтами доступны индивидуальные планы питания, разработанные диетологами.

MyTherapy

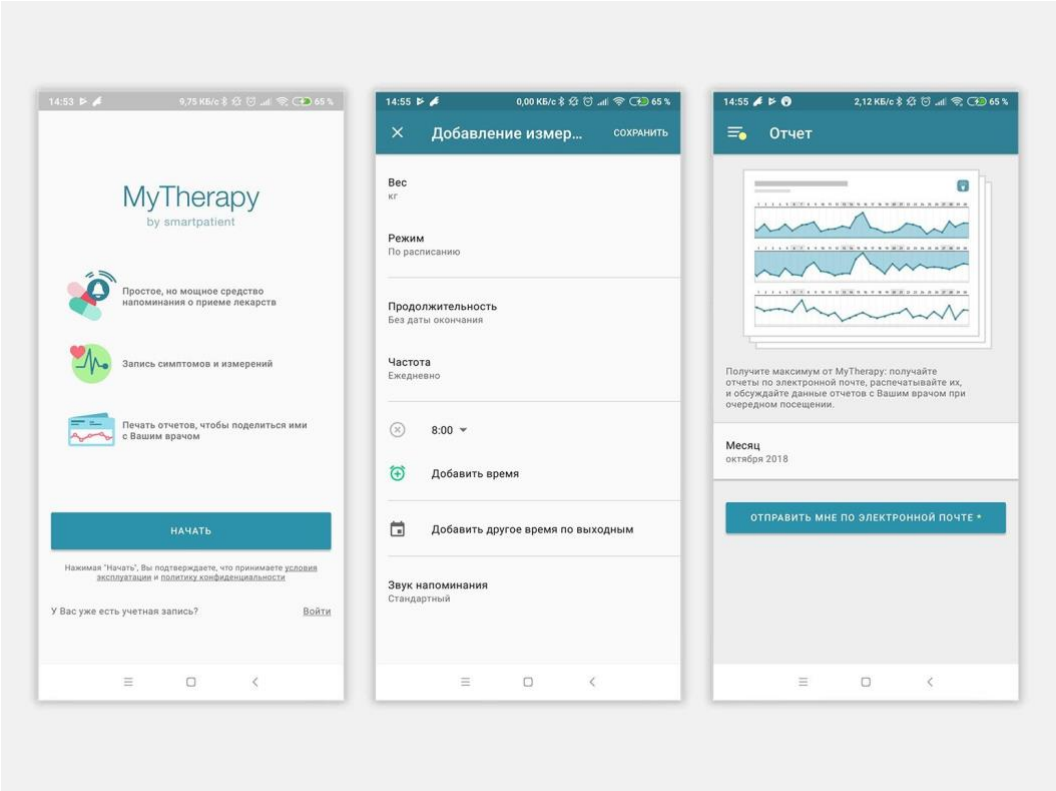

Рисунок 3.3 - Демонстрация работы с приложением MyTherapy

Приложение, которое напомнит, что нужно принять лекарства, измерить давление или уровень сахара в крови, сделать упражнение. MyTherapy отслеживает количество оставшихся таблеток и при необходимости напоминает, что пора пополнить аптечку (см. рисунок 3.3).

Итоговый обзор доступных приложений показал, что приложение, поддерживаемое не только одной версией iOS/Android будет более удобным и привлекательным для пользователей.

В итоге разрабатываемое приложение действительно может оказаться полезным для пользователей за счет следующих факторов:

- Поддержка нескольких устройств.
- Локализация на 6 языках.
- Бесплатный доступ к приложению
- Мониторинг давления с помощью нейросети, считывающая данные с тонометра.
- Система уведомлений пользователя под его конкретные нужды.

3.2 Функциональные и нефункциональные требования

Сбор и обработка биометрических данных осуществляется автоматически с использованием носимых устройств, обеспечивающих регулярное получение ключевых физиологических показателей. Частота измерений варьируется в зависимости от типа данных и уровня активности пользователя. Так, пульс фиксируется каждые 5 минут в состоянии покоя и каждые 30 секунд при физической активности. Измерение кровяного давления производится 2–3 раза в час при наличии подключённого тонометра. Уровень насыщения крови кислородом (SpO_2) определяется с интервалом в 10 минут, тогда как показатели активности, такие как количество шагов и потраченные калории, отслеживаются непрерывно.

Поддержка разнородных датчиков:

- фитнес-трекеры (Mi Band, Fitbit);
- умные часы (Apple Watch, Samsung Galaxy Watch);
- специализированные медицинские устройства (тонометры, глюкометры).

Анализ данных включает в себя выявление аномалий в режиме реального времени, основанное на отклонении текущих показателей от индивидуальной нормы пользователя. Кроме того, система формирует тренды и позволяет строить долгосрочные прогнозы, способствуя более глубокой оценке состояния здоровья.

Механизм уведомлений и рекомендаций реализован в виде трёхуровневой системы оповещений: от информационных сообщений до предупреждающих и экстренных сигналов, требующих немедленного реагирования. В дополнение к этому пользователю предоставляются персонализированные рекомендации, направленные на улучшение физиологических показателей и общее укрепление здоровья.

Локальное хранение данных. Ведение дневника здоровья с возможностью ручного добавления записей.

Нефункциональные требования:

Производительность:

- задержка при обработке данных: не более 2 секунд;
- автономная работа (без облачных серверов) при базовом функционале.

Безопасность:

- шифрование всех медицинских данных (алгоритм AES-256);
- поддержка двухфакторной аутентификации для критических операций.

Совместимость:

- iOS 15+ (поддержка HealthKit);
- Android 10+ (поддержка Google Fit).

Технологический стек:

- iOS: Swift, SwiftUI, HealthKit;
- Android: Kotlin, Jetpack Compose;
- Аналитика: Python (TensorFlow Lite для on-device ML).

Пользовательский интерфейс:

- поддержка динамических шрифтов и тем (для слабовидящих);
- минимальное время отклика на действия (менее 0,5 сек).

Надежность:

- автоматическое резервное копирование данных раз в сутки;
- восстановление после сбоев без потери информации.

Диаграммы вариантов использования. Диаграмма вариантов использования описывает, какой функционал разрабатываемой системы доступен каждой группе пользователей.

Диаграммы деятельности. Диаграммы деятельности представляет собой блок-схему, которая наглядно показывает, как поток управления переходит от одной деятельности к другой.

Диаграммы классов. Диаграмма классов определяет типы классов системы и различного рода статические связи, которые существуют между ними. На диаграммах классов изображаются также атрибуты классов, операции классов и ограничения, которые накладываются на связи между классами.

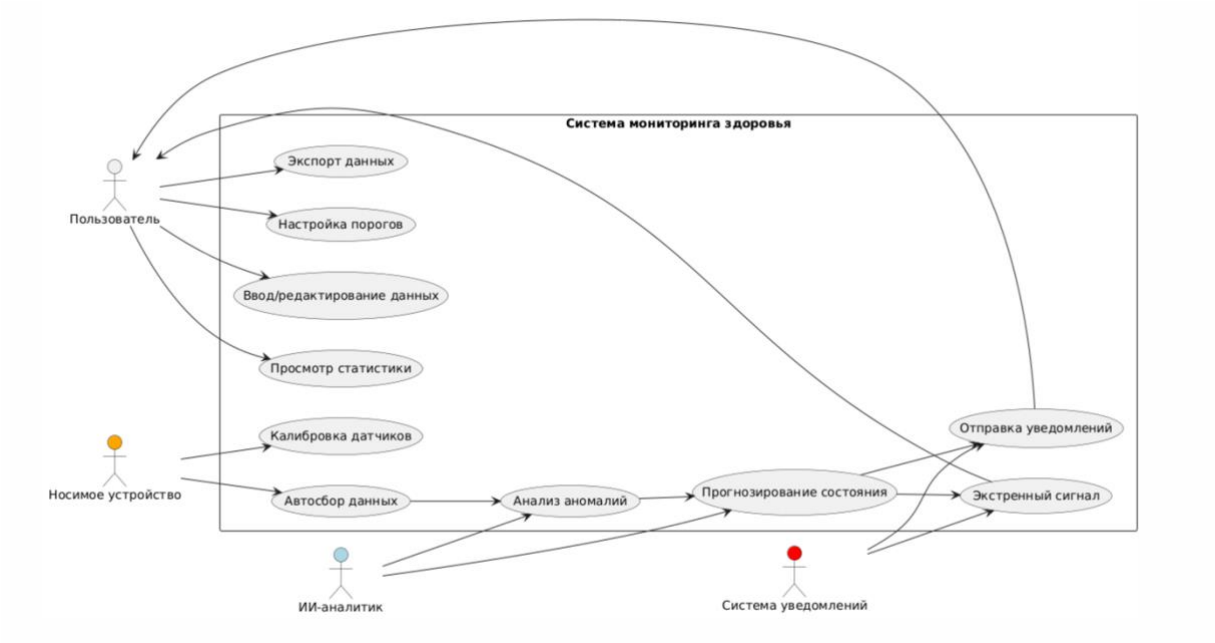

Рисунок 3.4 - Диаграмма вариантов использования процесса работы с основной частью приложения

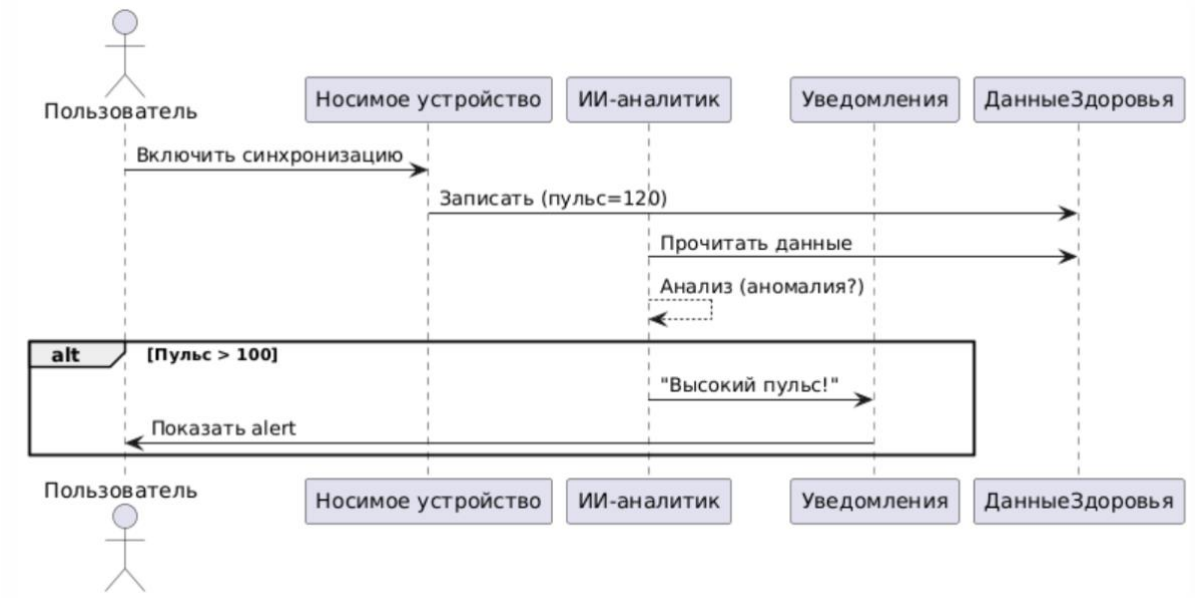

Рисунок 3.5 - Диаграмма последовательностей приложения

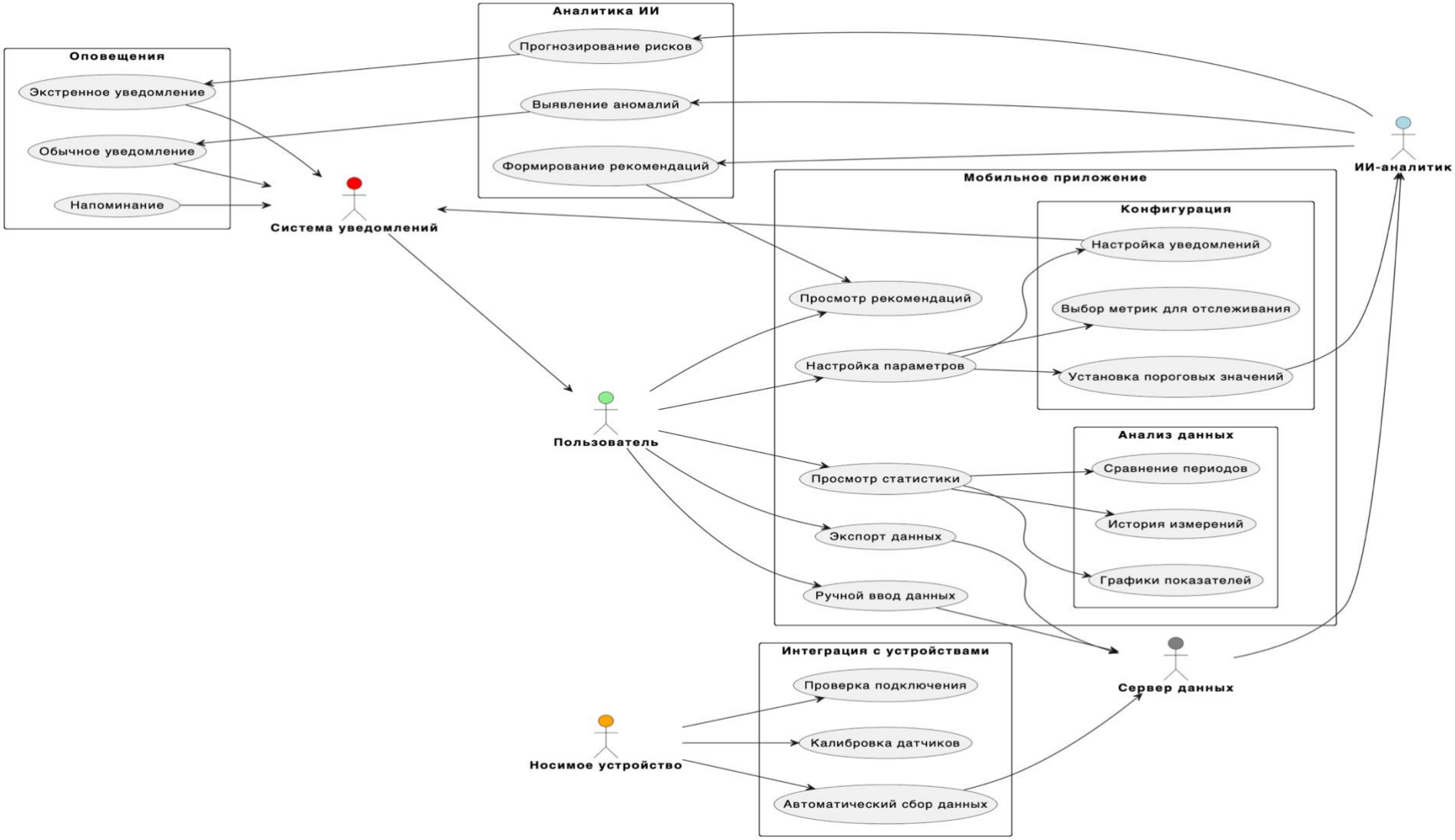

Рисунок 3.6 - Диаграмма вариантов использования процесса работы с основной частью приложения

3.3 Проектирование архитектуры приложения и основных классов

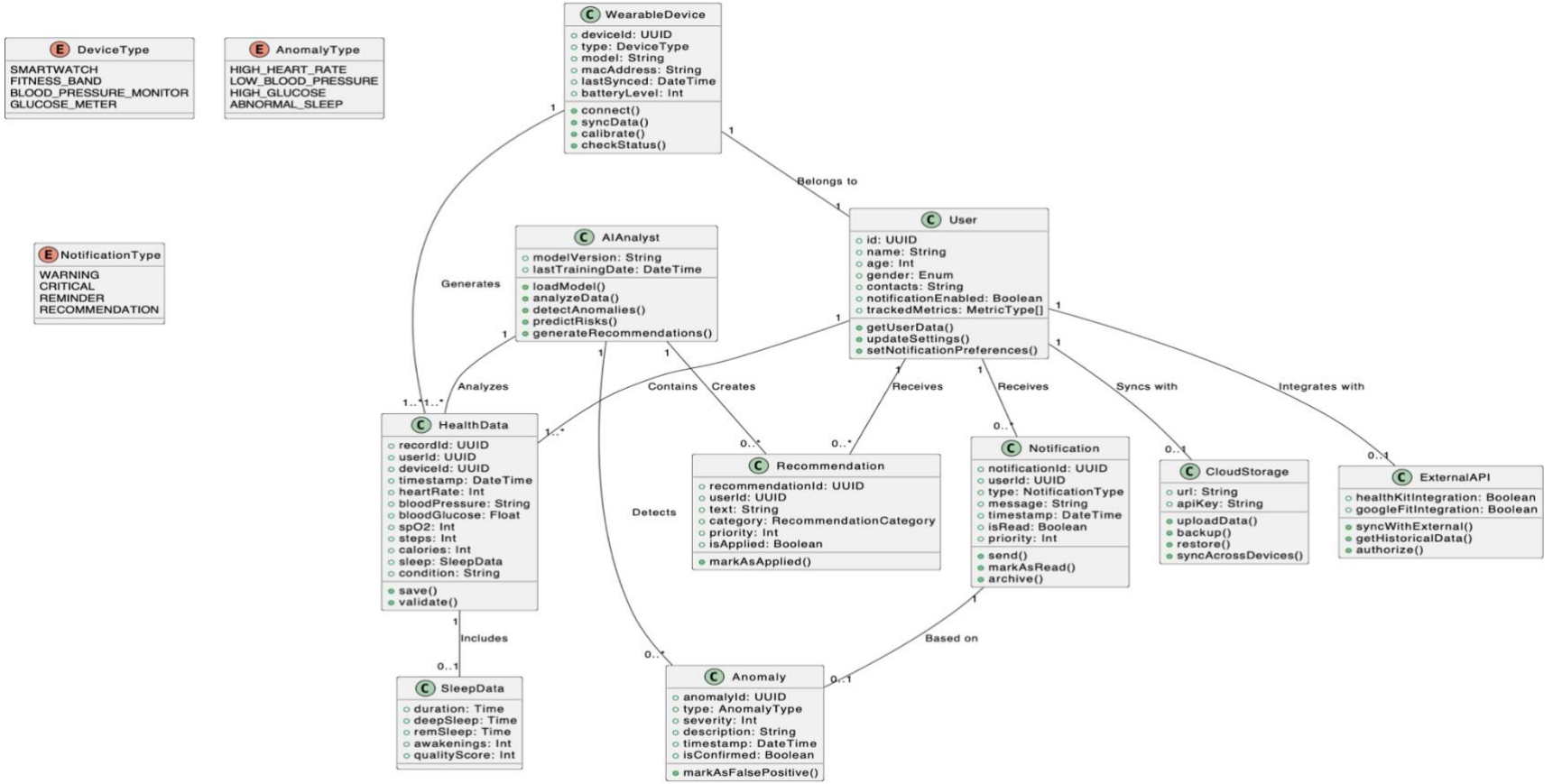

Рисунок 3.7 - Диаграмма классов

Приложение для мониторинга здоровья представляет собой интеллектуальную систему, которая работает как персональный помощник, незаметно следящий за состоянием пользователя через подключенные носимые устройства. Оно не требует сложных настроек – после простого сопряжения со смарт-часами или фитнес-трекером система начинает непрерывно анализировать ключевые показатели: сердечный ритм, уровень кислорода в крови, качество сна и физическую активность. Встроенный искусственный интеллект изучает индивидуальные особенности организма, формируя персональную базу нормальных значений для каждого пользователя.

Когда система замечает отклонения от привычных параметров, она оперативно оценивает степень опасности. Незначительные колебания просто фиксируются в дневнике здоровья, а вот тревожные изменения – например, длительное повышение пульса в состоянии покоя или резкое падение уровня кислорода – вызывают мгновенную реакцию. В таких случаях приложение не просто показывает предупреждение, а предлагает конкретные действия: от рекомендации выпить воды до совета немедленно обратиться к врачу. Особые алгоритмы отличают реальные угрозы от ложных тревог, учитывая физическую активность и другие контекстные факторы.

Все данные хранятся локально на устройстве пользователя, обеспечивая полную конфиденциальность. Интерфейс выполнен в минималистичном стиле: на главном экране отображаются только ключевые показатели и их динамика, а более детальная статистика доступна в соответствующих разделах. Система постоянно обучается, подстраиваясь под изменения в организме пользователя и со временем предлагая все более точные рекомендации. Это не просто трекер показателей, а настоящий цифровой компаньон, который помогает не только отслеживать, но и по-настоящему понимать состояние своего здоровья.

GRASP – это набор принципов проектирования в объектно-ориентированном программировании, которые помогают правильно распределять ответственности между классами и объектами, делая систему более понятной, гибкой и поддерживаемой. Вот краткое описание каждого принципа GRASP:

Информационный эксперт (Information Expert) является ключевым принципом, согласно которому ответственность за выполнение операции должна возлагаться на класс, обладающий всей необходимой информацией. Такой подход естественным образом приводит к концентрации данных и методов их обработки в одном месте, что делает систему более логичной и предсказуемой. Когда класс содержит данные, необходимые для определенных вычислений, именно он должен реализовывать

соответствующую функциональность, что соответствует принципу инкапсуляции.

Создатель (Creator) определяет, какой класс должен отвечать за создание объектов другого класса. Это решение принимается на основе анализа отношений между классами - если один класс содержит или активно использует объекты другого класса, он и должен брать на себя ответственность за их создание. Такой подход помогает поддерживать порядок в системе и избегать хаотичного создания объектов в разных частях программы.

Контроллер (Controller) играет важную роль в архитектуре приложения, выступая посредником между пользовательским интерфейсом и бизнес-логикой. Этот принцип гласит, что обработка системных событий и пользовательских запросов должна выполняться специальными классами, а не интерфейсом напрямую. Такое разделение позволяет четко разграничить ответственность между представлением данных и их обработкой.

Слабая связность (Low Coupling) является фундаментальным принципом, способствующим созданию гибких систем. Он предполагает минимизацию зависимостей между классами, что делает систему более устойчивой к изменениям. Когда классы слабо связаны, модификация одного из них оказывает минимальное влияние на другие компоненты системы.

Высокая зацепленность (High Cohesion) дополняет принцип слабой связности, фокусируясь на внутренней организации классов. Согласно этому принципу, каждый класс должен решать четко определенную задачу или группу тесно связанных задач. Это делает код более понятным и облегчает его сопровождение.

Полиморфизм (Polymorphism) в контексте GRASP предлагает использовать механизмы наследования и переопределения методов вместо сложных условных конструкций. Когда поведение системы зависит от типа объекта, полиморфный подход позволяет сделать код более элегантным и расширяемым.

Чистая выдумка (Pure Fabrication) признает, что не все классы в системе должны соответствовать объектам предметной области. Иногда для решения технических задач необходимо создавать специальные служебные классы, которые не имеют аналогов в реальном мире, но важны для работы приложения.

Перенаправление (Indirection) предлагает вводить промежуточные объекты для уменьшения связности между компонентами системы. Такой подход особенно полезен в сложных системах, где прямое взаимодействие между классами может создать запутанную сеть зависимостей.

Устойчивость к изменениям (Protected Variations) завершает список принципов GRASP, предлагая защищать критические точки системы через абстракции. Этот принцип особенно важен при работе с внешними сервисами и компонентами, которые могут изменяться независимо от основной системы.

При разработке приложения все принципы будут соблюдаться, позволяя делать код более гибким и эффективным.

3.4 MVVM-C в контексте разработки приложения

MVVM-C (Model-View-ViewModel-Coordinator) – это современная архитектурная паттерн-комбинация, расширяющая классический MVVM за счёт введения координаторов для управления навигацией и потоком данных. В вашем приложении для мониторинга здоровья она реализуется следующим образом:

Model (Модель данных) отвечает за хранение и валидацию медицинских показателей, таких как пульс, уровень кислорода в крови (SpO_2) и артериальное давление. Она обеспечивает интеграцию с носимыми устройствами с использованием технологий BLE или NFC, а также гарантирует соответствие требованиям стандартов конфиденциальности и безопасности, включая HIPAA и GDPR.

View (Пользовательский интерфейс) выполняет роль пассивного отображения данных и автоматически обновляется при изменении состояния данных с помощью механизмов биндинга. Интерфейс адаптирован под требования доступности, включая поддержку динамических шрифтов и VoiceOver, что делает приложение удобным для широкого круга пользователей.

ViewModel (Бизнес-логика) служит посредником между моделью и представлением. Она обрабатывает сырые данные, поступающие с датчиков, и преобразует их в формат, удобный для отображения в интерфейсе. Здесь же реализуются правила формирования уведомлений, например, при превышении пульса заданного порога. Кроме того, ViewModel интегрируется с ИИ-модулем, предназначенным для выявления аномалий и предиктивного анализа состояния здоровья пользователя.

Coordinator (Управление навигацией) инкапсулирует всю логику переходов между экранами, обеспечивая централизованное и чистое управление маршрутизацией. Он также реализует поддержку deep linking — переходов по уведомлениям, ведущих напрямую к конкретному экрану, и управляет зависимостями компонентов приложения.

Сравнение с GRASP-принципами в таблице 3.1

Таблица 3.1 – Пояснение применений GRASP и MVVM-C

Принцип GRASP	**Реализация в MVVM-C**	**Пример из проекта**
Информационный эксперт	ViewModel содержит логику обработки данных	Анализ пульса в DashboardViewModel
Слабая связность	Координатор изолирует навигацию	AppCoordinator не зависит от View
Защита изменений	Абстракция HealthDataService	Возможность замены BLE на NFC
Чистая выдумка	Coordinator как искусственная сущность	Управление стеком навигации

Преимущества данного проекта:

- Гибкость. Легкая замена датчиков (например, Withings → Fitbit) без необходимости изменять View. Возможность проведения изолированных тестов ViewModel с использованием моков сервиса HealthDataService.
- Безопасность. Инкапсуляция медицинской логики внутри ViewModel, исключающая доступ к чувствительным данным извне. Контроль доступа к экранам через Coordinator, включая возможность карантина при обнаружении критических показателей.
- Масштабируемость. Возможность добавления новых медицинских метрик (например, глюкозы или ЭКГ) путём расширения слоя Model без затрагивания остальных компонентов.

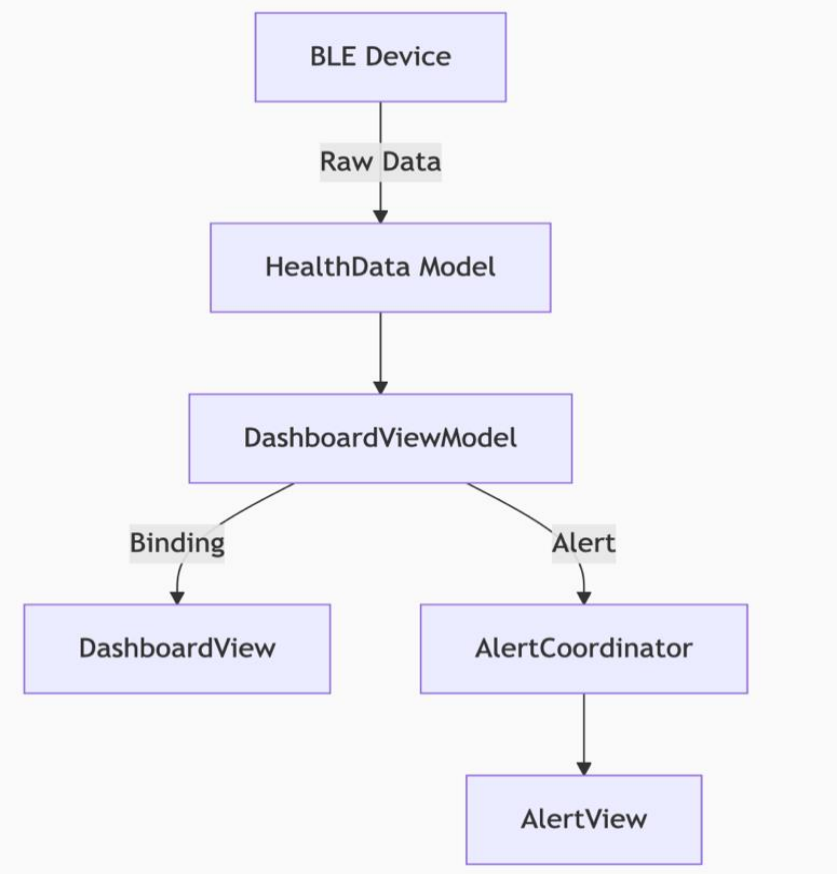

Рисунок 3.8 – UML-диаграмма последовательности

3.5 Алгоритмы анализа медицинских данных

Для эффективной работы сложных технологических систем необходимы не только мониторинг, но и различные методы аналитики. Это позволяет контролировать процессы, управлять ими и своевременно корректировать параметры. Обычно мониторинг реализуется с помощью стандартных инструментов, которые обеспечивают надёжный сбор и визуализацию данных. Однако для построения эффективных аналитических решений требуются дополнительные исследования, эксперименты и глубокое знание предметной области.

Принято выделять четыре основных типа аналитики данных [1]:

Описательная аналитика — визуализирует собранные данные, часто преобразуя их для большей наглядности и удобства интерпретации. Это самый базовый, но необходимый этап, который служит основой для последующего анализа.

Диагностическая аналитика — помогает установить причины событий, произошедших в прошлом. На этом этапе выявляются тренды, аномалии, характерные особенности процессов, а также ищутся взаимосвязи и причины произошедшего.

Прогнозная аналитика — позволяет предсказывать вероятные исходы, используя обнаруженные тенденции и статистические модели, построенные на исторических данных.

Предписывающая аналитика — на основе результатов прогнозирования предлагает оптимальные решения для производственных задач. Например, это может быть оптимизация параметров работы оборудования, совершенствование бизнес-процессов или рекомендации по предотвращению аварий.

Для прогнозной и предписывающей аналитики часто применяются методы моделирования, включая машинное обучение. Эффективность этих подходов во многом зависит от правильной организации сбора, обработки и предварительного анализа данных. Приведённые виды аналитики различаются по сложности используемых моделей и степени вовлечённости человека в процесс анализа.

Многие технологические системы представляют данные мониторинга в виде временных рядов [2]. Ключевые свойства временных рядов включают:

- Фиксацию момента времени для каждого измерения (сэмпла,
- дискрета);
- Равные временные интервалы между измерениями;
- Возможность прогнозирования текущего и будущего поведения процесса на основе исторических данных.

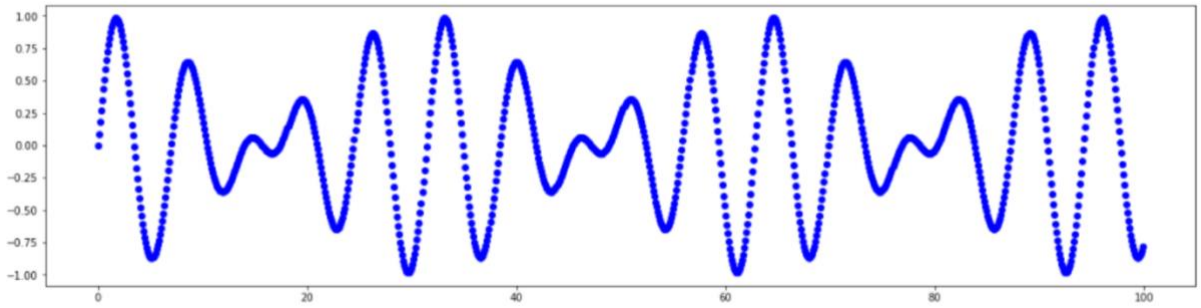

Рисунок 3.9 – Временной ряд

Характеристики временных рядов:

- **Период** – фиксированный временной интервал, на границах которого значения ряда схожи.
- **Сезонность** – повторяемость паттернов с определённой периодичностью (сезон = период).
- **Цикл** – изменения, вызванные глобальными факторами (например, экономическими циклами), без строгой периодичности.
- **Тренд** – устойчивая долгосрочная тенденция к росту или снижению значений.

Аномалии во временных рядах

Аномалия – это отклонение от нормального поведения процесса. Алгоритмы их обнаружения анализируют датасеты, причём в разных областях аномалии могут отличаться. Основные типы:

Точечные аномалии – резкие отклонения в отдельных точках (например, выбросы). Обнаруживаются по пороговым значениям и сильно влияют на статистику.

Групповые аномалии – совокупность точек, каждая из которых в отдельности нормальна, но вместе формирует аномальное поведение (например, изменение формы сигнала или статистических показателей).

Контекстные аномалии – отклонения, связанные с внешними факторами (например, аномально низкая температура летом).

Точечные аномалии выявляются проще всего, тогда как групповые требуют более сложных методов, так как стандартные статистические подходы могут их не обнаружить.

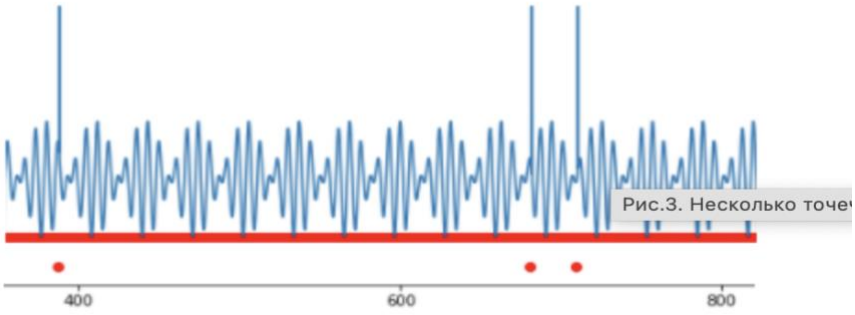

Рисунок 3.10 – Несколько точечных аномалий

Сложнее обнаружить аномалию в ситуации, когда в каждой точке процесс ведет себя «нормально», но в совокупности значения в нескольких точках ведут себя «странно». К такому аномальному поведению можно отнести, например, изменение формы сигнала, изменение статистических показателей (среднее значение, мода, медиана, дисперсия), появление взаимной корреляции между двумя параметрами, небольшие или краткосрочные аномальные изменения амплитуды и так далее. И в этом случае задача заключается в распознавании аномального поведения параметров, которое нельзя выявить обычными статистическими методами.

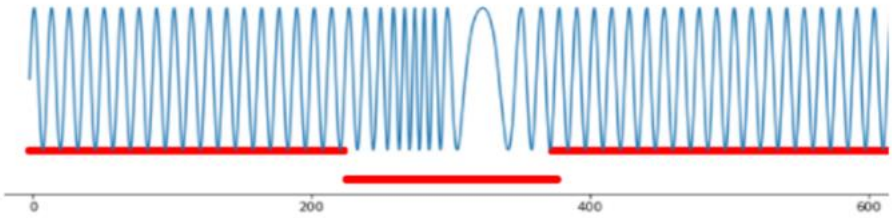

Рисунок 3.11 – Груповая аномалия, изменение частоты

Значение поиска аномалий

Обнаружение аномалий играет ключевую роль в анализе данных. В одних случаях их устранение помогает получить более достоверную картину процессов, в других – аномалии служат индикаторами потенциальных сбоев, предупреждая о возможных аварийных ситуациях.

Сложности автоматизированного поиска аномалий во временных рядах обусловлены рядом факторов. Прежде всего, определение отклонений нередко носит субъективный характер, поскольку границы «нормы» могут существенно различаться в зависимости от индивидуальных особенностей пользователя. Дополнительную сложность представляет отсутствие размеченных данных, необходимых для обучения и валидации алгоритмов. Также важно учитывать наличие скрытых зависимостей в данных, которые не всегда поддаются выявлению стандартными методами. Даже современные алгоритмы, относящиеся к уровню state-of-the-art, подвержены ошибкам и нередко формируют ложные срабатывания (False Positive), что снижает надёжность системы.**Ограничения ручного анализа**

Визуальный анализ позволяет обнаружить лишь простые точечные аномалии. Однако групповые аномалии, особенно в больших объёмах данных или при анализе множества устройств, требуют автоматизации. Особую сложность представляют временные аномалии – ситуации, когда корректные по значению данные появляются в неверный момент времени.

Проблема отсутствия разметки

В реальных данных аномалии часто не имеют чётких критериев, что исключает применение методов обучения с учителем (supervised learning). В таких случаях используются unsupervised-методы, которые самостоятельно выявляют закономерности и отклонения.

Proximity-based (на основе близости) методы опираются на принцип поиска точек, которые значительно отклоняются от остальных в пространстве признаков. Эти подходы хорошо подходят для выявления выбросов и точечных аномалий, поскольку позволяют зафиксировать отдельные значения, не вписывающиеся в общую структуру данных. Примером может служить простой пороговый контроль, при котором значения, выходящие за пределы допустимого диапазона, считаются аномальными. Однако основной недостаток данного подхода — неспособность распознавать изменения формы сигнала или групповые отклонения, где отклонение выражается не в отдельных точках, а в их последовательности.

Prediction-based (прогнозные методы) основаны на сравнении фактических данных с предсказанными значениями, которые генерирует обученная модель. Если наблюдаемое значение существенно отличается от прогноза, оно рассматривается как потенциальная аномалия. Эти методы

особенно эффективны для временных рядов с выраженной сезонностью, повторяющимися циклами и устойчивыми закономерностями. В качестве критерия отклонения выступает величина ошибки прогноза. Подобные подходы широко применяются в задачах мониторинга, где ожидается стабильная динамика показателей, например, в анализе сердечного ритма или уровня активности.

Reconstruction-based (методы реконструкции) используют подход, при котором обученная модель пытается восстановить исходный фрагмент данных, и если ошибка реконструкции превышает допустимый уровень, фиксируется аномалия. Эти методы эффективны как для точечных, так и для групповых аномалий, включая структурные изменения сигнала. Такой подход позволяет фиксировать нарушения, которые невозможно обнаружить с помощью простого анализа значений. Благодаря гибкости и глубокой обработке структуры сигнала, методы реконструкции находят широкое применение в задачах медицинского мониторинга и анализа биометрических данных.

Для системы мониторинга здоровья с помощью носимых устройств и мобильных сенсоров наиболее подходящим подходом является Proximity-based метод обнаружения аномалий. Это связано с тем, что биометрические данные (такие как пульс, уровень кислорода в крови, температура тела и физическая активность) часто имеют четкие медицинские границы нормы, выход за которые можно легко определить с помощью пороговых значений. Например, частота сердечных сокращений ниже 40 или выше 150 ударов в минуту, уровень SpO_2 ниже 90%, или резкий скачок температуры выше 38°C – все эти случаи являются явными аномалиями, которые можно выявить простыми правилами.

Проблема отсутствия разметки в медицинских данных делает Proximity-based методы особенно актуальными, так как они не требуют предварительно размеченных данных о аномалиях. В отличие от supervised-подходов, где нужно точно знать, какие именно точки являются аномальными, proximity-based алгоритмы работают на основе статистических характеристик данных или заданных медицинских норм. Это делает их идеальными для начального этапа мониторинга, когда аномалии еще не классифицированы, но необходимо быстро отсекать заведомо некорректные или опасные значения.

Основные преимущества Proximity-based методов для мониторинга здоровья:

- Простота реализации – можно использовать элементарные пороговые проверки (например, "если ЧСС > 150 → тревога").
- Интерпретируемость – врачи и пользователи легко понимают, почему то или иное значение было помечено как аномальное.

- Низкие вычислительные затраты – критично для носимых устройств с ограниченными ресурсами.
- Высокая скорость работы – позволяет обнаруживать аномалии в реальном времени.

Примеры применения:

- Пульс: аномально высокий или низкий ЧСС (например, <40 или >150 уд./мин);
- SpO_2: падение сатурации ниже 90% (гипоксия);
- Температура: резкий подъем выше 38°C (возможная инфекция);
- Активность: внезапное падение движения (риск падения или потери сознания).

Ограничения метода:

Не может обнаруживать сложные аномалии, такие как изменения формы ЭКГ-сигнала или кратковременные колебания, которые по отдельности находятся в норме, но в совокупности опасны.

Требует настройки порогов (слишком строгие – будут ложные срабатывания, слишком мягкие – пропуск аномалий).

ВЫВОДЫ

Проделанная работа позволила создать целостную концепцию системы мониторинга здоровья, объединяющую передовые технологии носимых устройств и искусственного интеллекта. На основании анализа предметной области были определены ключевые функциональные возможности приложения: автоматизированный сбор биометрических данных, интеллектуальный анализ показателей в реальном времени, система многоуровневых уведомлений и персонализированные рекомендации для пользователя.

Разработанная архитектура системы отражает современные подходы к проектированию медицинских IT-решений. Четкое разделение на модули (сбор данных, аналитическая обработка, интерфейс взаимодействия) обеспечивает гибкость разработки и возможность дальнейшего масштабирования. Особое внимание уделено безопасности и конфиденциальности – выбранная модель хранения и обработки данных соответствует строгим требованиям защиты персональной медицинской информации.

Созданные диаграммы вариантов использования и классов наглядно демонстрируют логику взаимодействия всех компонентов системы. Проработанные сценарии работы приложения подтверждают его практическую ценность для различных категорий пользователей – от людей,

следящих за своим здоровьем, до пациентов, нуждающихся в постоянном медицинском контроле.

Результаты проектирования создают прочный фундамент для последующей реализации системы. Следующим этапом станет разработка прототипа с базовым функционалом, который позволит проверить работоспособность ключевых алгоритмов и получить первые отзывы от потенциальных пользователей. Дальнейшее развитие проекта будет включать совершенствование нейросетевых моделей анализа данных и расширение списка поддерживаемых носимых устройств.

Проведенная работа доказала жизнеспособность концепции и продемонстрировала значительный потенциал приложения в области персонального мониторинга здоровья. Реализация этого проекта открывает новые возможности для профилактики заболеваний и повышения качества жизни пользователей за счет своевременного выявления рисков для здоровья.

ГЛАВА 4 РЕАЛИЗАЦИЯ МОБИЛЬНОГО ПРИЛОЖЕНИЯ

4.1 Экраны приложения

Экраны, кратко описывающие суть приложения. При открытии приложения пользователь увидит приветствующие экраны (см. рисунок 4.1). Эти экраны показывают кратко дизайн приложения и какая функциональность доступна.

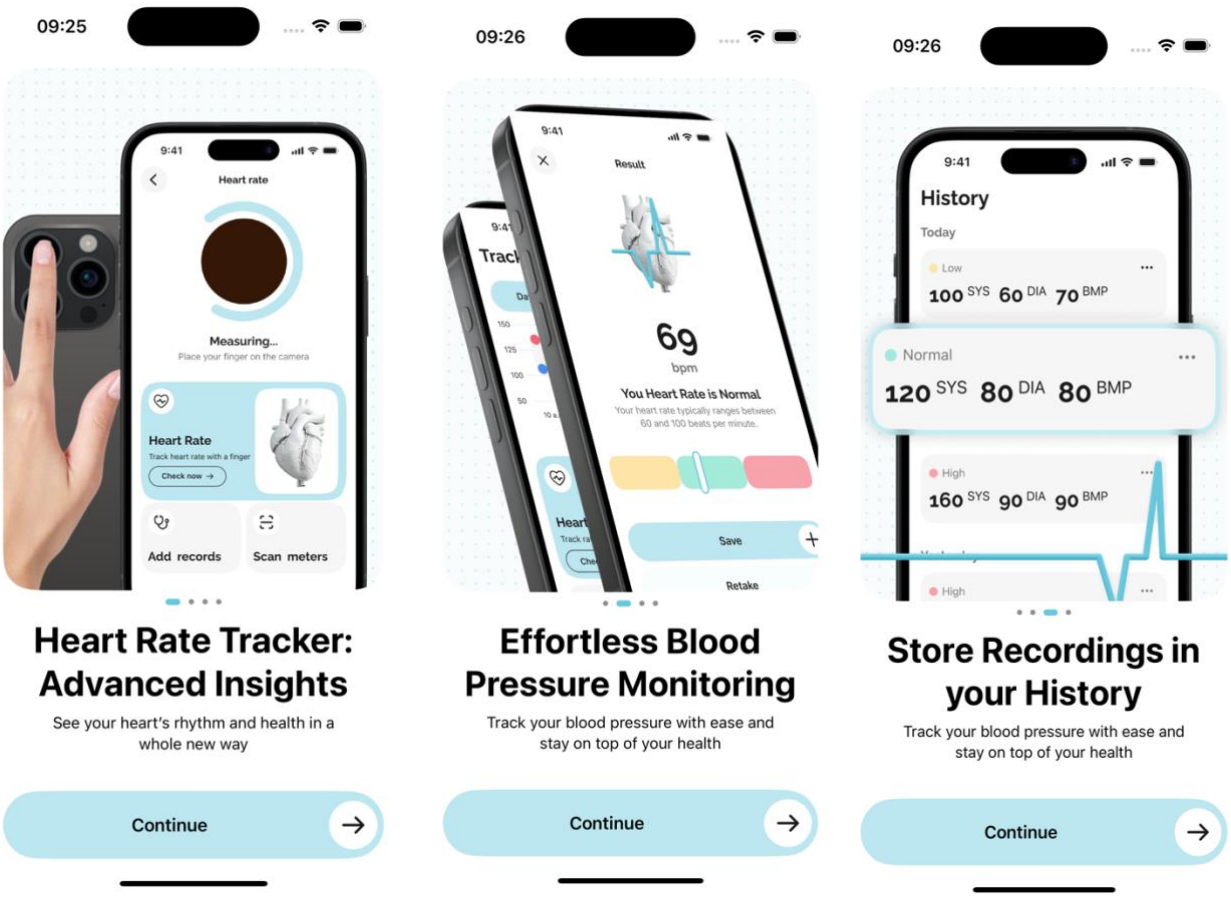

Рисунок 4.1 - Приветственные экраны

Главный экран. На главном экране (см. рисунок 4.2) сразу видно несколько возможностей приложения. Во-первых, это измерение пульса, а также кислорода в крови (Health Rate). Во-вторых возможность записать данные о здоровье (AddRecords), такие как систолическое и диастолическое давление с пульсом. Третья кнопка предлагает нам с помощью камеры и встроенной нейросети сканировать данные с тонометра и автоматически записать их в приложение

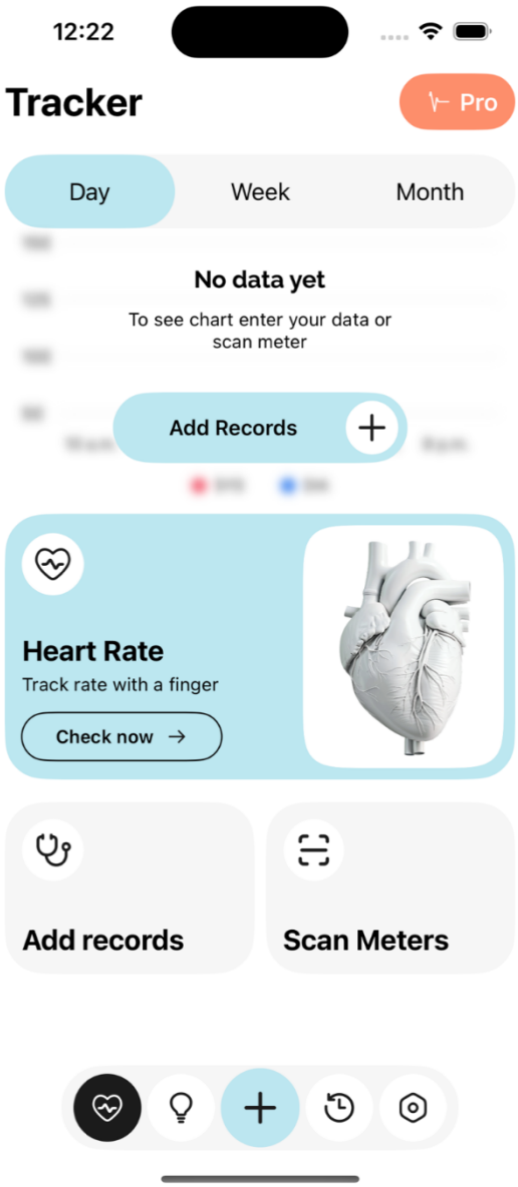

Рисунок 4.2 - Главный экран

Экран с информацией об использовании приложения. Далее пользователь может нажать на иконку лампочки и перейти на экран информации (см. рисунок 4.3). Где есть список возможностей приложения, а также пояснение как им пользоваться. Connecting to Apple Health, там описано как подключаться к здоровью Apple и сохранять данные. How to scan meters описано как сканировать данные с помощью камеры наводя ее на тонометр. How to take heart rate описано измерение пульса и кислорода с помощью приложения. В Recommendations описано, почему нужно вести здоровый образ жизни и как ему следовать.

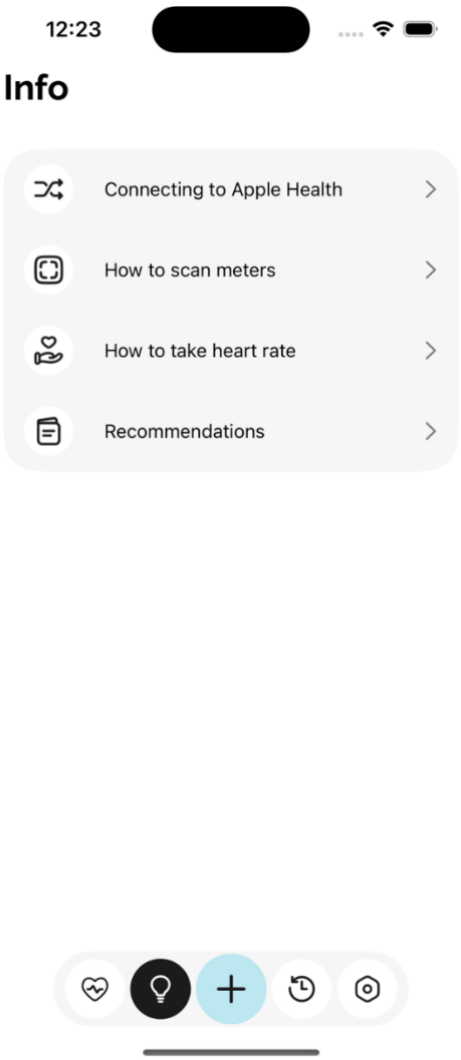

Рисунок 4.3 - Экран с информацией

Экран ручного добавления данных. Нажимая на кнопку плюс, мы переходим на экран ручного добавления, где можем добавить наши данные и поставить время для данных (см. рисунок 4.4). Так же можно отменить данное действие с помощью нажатия верхней левой кнопки и вернуться на главный экран или нажав кнопку Cancel.

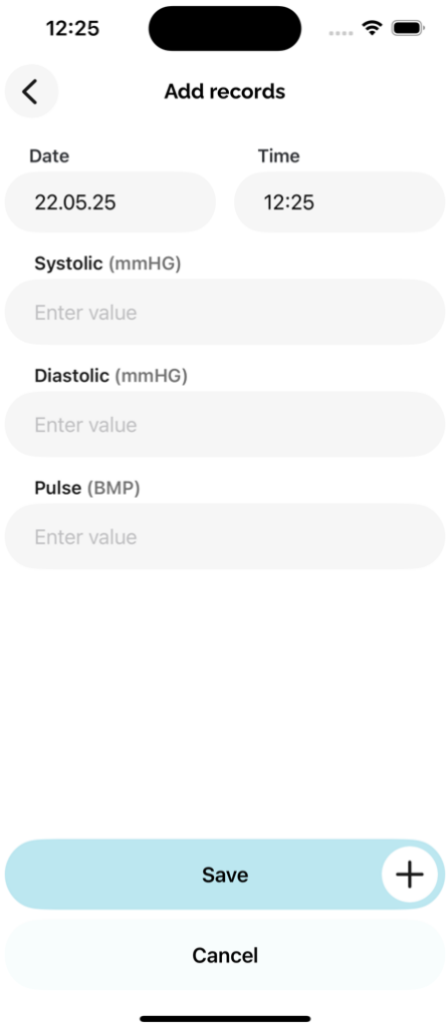

Рисунок 4.4 - Экран ручного добавления данных.

Экран истории. Для отслеживания своих медицинских данных есть экран истории (см. рисунок 4.5). В нем каждый баннер хранит показатели данных. Слева вверху дата их добавления. Нажав на многоточие, которое находится в правом углу баннера можно изменить свои показатели или удалить их. При нажатии на Edit вы можете отредактировать данные, сама дата хранится в локальном кэше телефона.

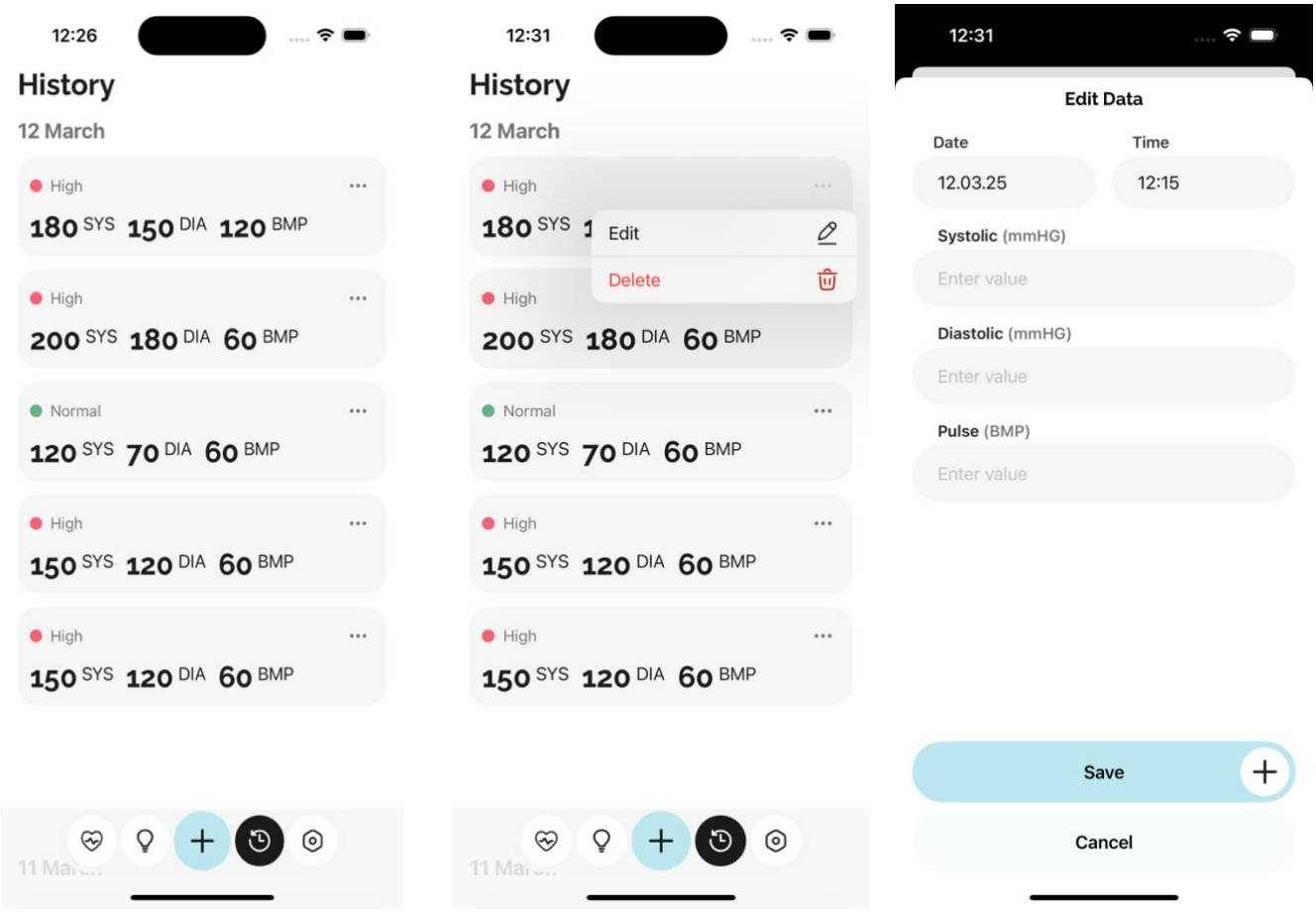

Рисунок 4.5 - Экран истории

Экран подписок предоставляет пользователю доступ к расширенному функционалу приложения, который недоступен в базовой версии (см. рисунок 4.6). После оформления подписки становятся активными такие возможности, как сканирование данных с тонометра, измерение пульса и уровня кислорода в крови с использованием сенсоров смартфона, а также настройка персонализированных уведомлений и отслеживание состояния здоровья с помощью встроенного ИИ-ассистента. Кроме того, при оформлении подписки на длительный срок предоставляются бонусные условия, делающие использование приложения более выгодным и удобным.

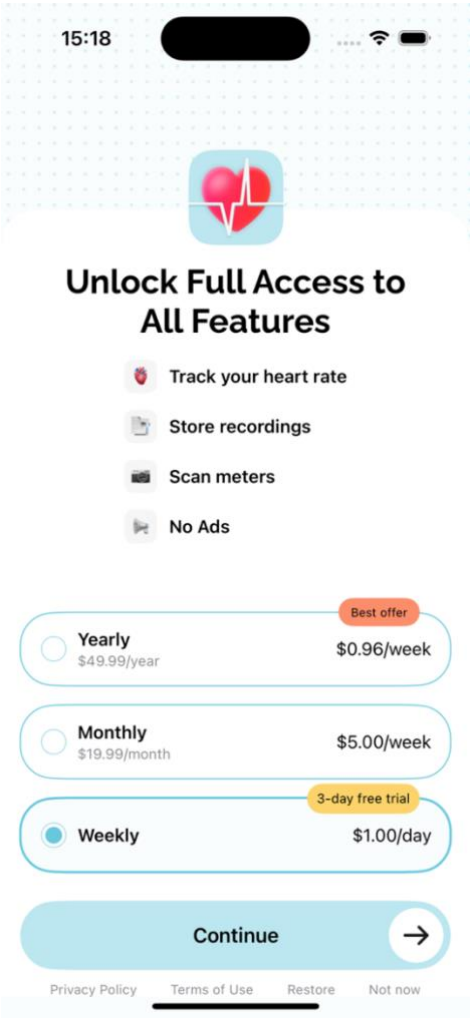

Рисунок 4.6 - Экран подписок

Экран настроек предоставляет пользователю доступ к дополнительным функциям и возможностям управления приложением (см. рисунок 4.7). В верхней части экрана расположен баннер Unlock your health, нажав на который можно приобрести подписку и получить доступ к расширенным функциям. Ниже располагаются различные сервисные опции: возможность связаться с разработчиками через раздел Contact us для отправки отзывов, рекомендаций или жалоб; функция Share App позволяет поделиться приложением с другими пользователями; через пункт Restore можно восстановить ранее приобретённую подписку. Также пользователь может оценить приложение в магазине с помощью Rate us, ознакомиться с Политикой конфиденциальности и Условиями использования, регулирующими работу с персональными данными и правилами использования приложения.

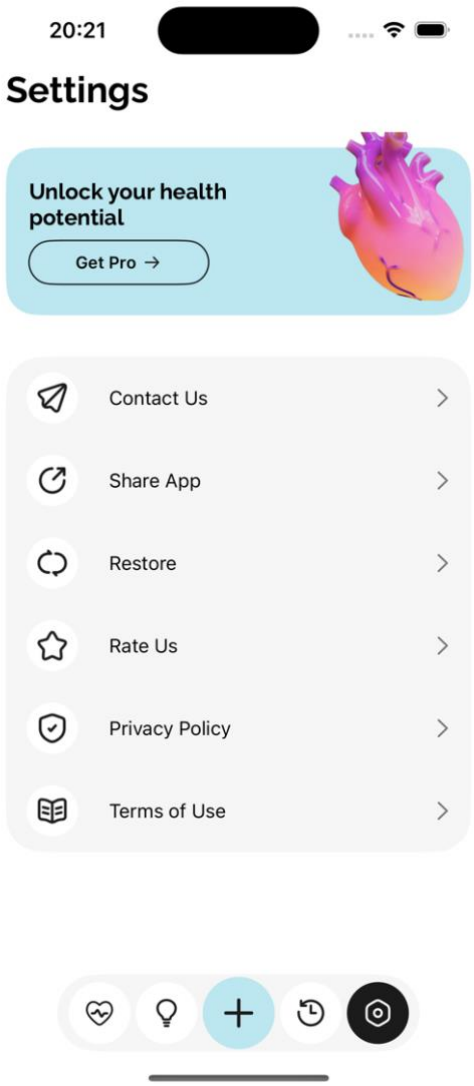

Рисунок 4.7 – Экран настроек

4.2 Регистрация пользователей с помощью Firebase

Firebase представляет собой облачную платформу от компании Google, предназначенную для разработки мобильных и веб-приложений. Данное решение позволяет эффективно организовать серверную часть приложения, включая работу с базами данных и систему аутентификации пользователей. Особенно востребованной платформа становится при создании прототипов и реализации проектов с ограниченными техническими возможностями.

Будучи облачным сервисом, Firebase обеспечивает хранение всех компонентов приложения на распределенных серверах Google. Платформа предлагает широкий спектр функциональных возможностей, среди которых стоит отметить механизмы авторизации, базы данных с синхронизацией в реальном времени, облачное хранение файлов, систему push-уведомлений и аналитические инструменты.

В процессе разработки приложения "BloodApp" были успешно реализованы различные компоненты Firebase. Для организации работы с пользователями применялась система аутентификации, хранение данных осуществлялось в NoSQL-базе Cloud Firestore, а для мониторинга стабильности работы использовались инструменты Crash Analytics и App Metrics. Дополнительно были задействованы функции удаленной конфигурации и система мгновенных сообщений.

Для работы с сервисами Google Firebase следует создать проект в Firebase, зарегистрироваться в нём приложение, скачать и добавить в приложение файл конфигурации google-service.info.plist и, следовательно, добавить сам FirebaseSDK.

Для реализации регистрации нам понадобится создать модуль аутентификации (AuthService), сервис валидации (ValidationService), менеджер сессий (SessionManager). Но так же для использования всех функций FireBase нужно прописать в AppDelegate.swift:

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
        // Инициализация Firebase
        FirebaseApp.configure()
        // Настройка Auth
       let auth = Auth.auth()
       auth.useAppLanguage()
       auth.settings?.isAppVerificationDisabledForTesting = false
       return true
}
```

Листинг 4.1 – Внедрение Firebase сервисов

```
// AuthService.swift
class AuthService {
    static let shared = AuthService()
    private init() {}

    func registerUser(email: String,
                      password: String,
                      completion: @escaping (Result<User, Error>) -> Void) {

        Auth.auth().createUser(withEmail: email,
                               password: password) { authResult,

            guard let user = authResult?.user else {
                completion(.failure(AuthError.unknownError))
                return
            }

            let changeRequest = user.createProfileChangeRequest()
            changeRequest.commitChanges { error in
                if let error = error {
                    completion(.failure(error))
                } else {
                    completion(.success(user))
                }
            }
        }
    }
}
```

Листинг 4.2 – Регистрация пользователя

Более подробное пояснение можно увидеть на рисунке 4.8 на нем изображено UML-диаграмма последовательности регистрации пользователя.

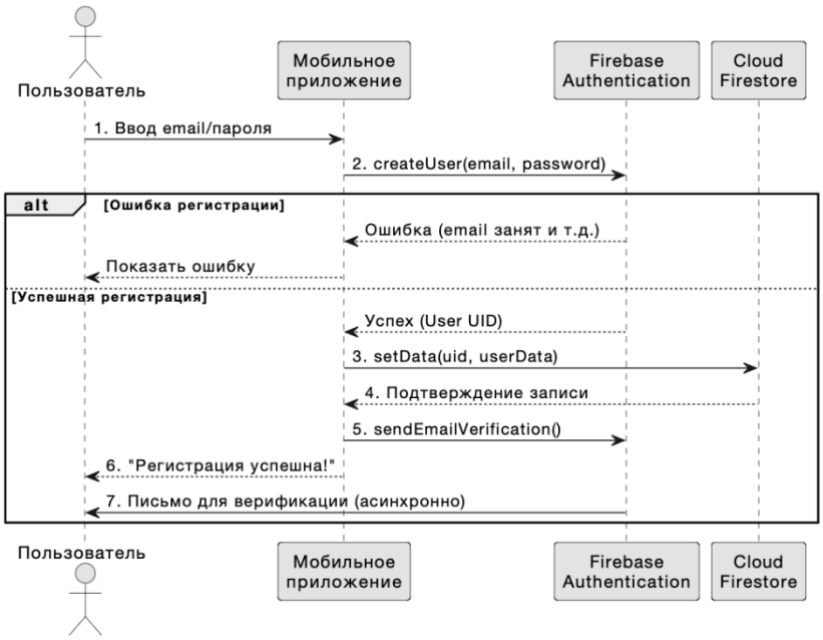

Рисунок 4.8 – UML-диаграмма последовательности регистрации

Firebase Auth можно просматриривать зарегистрированных пользователей и так же их уникальные USER UID (см. рисунок 4.9)

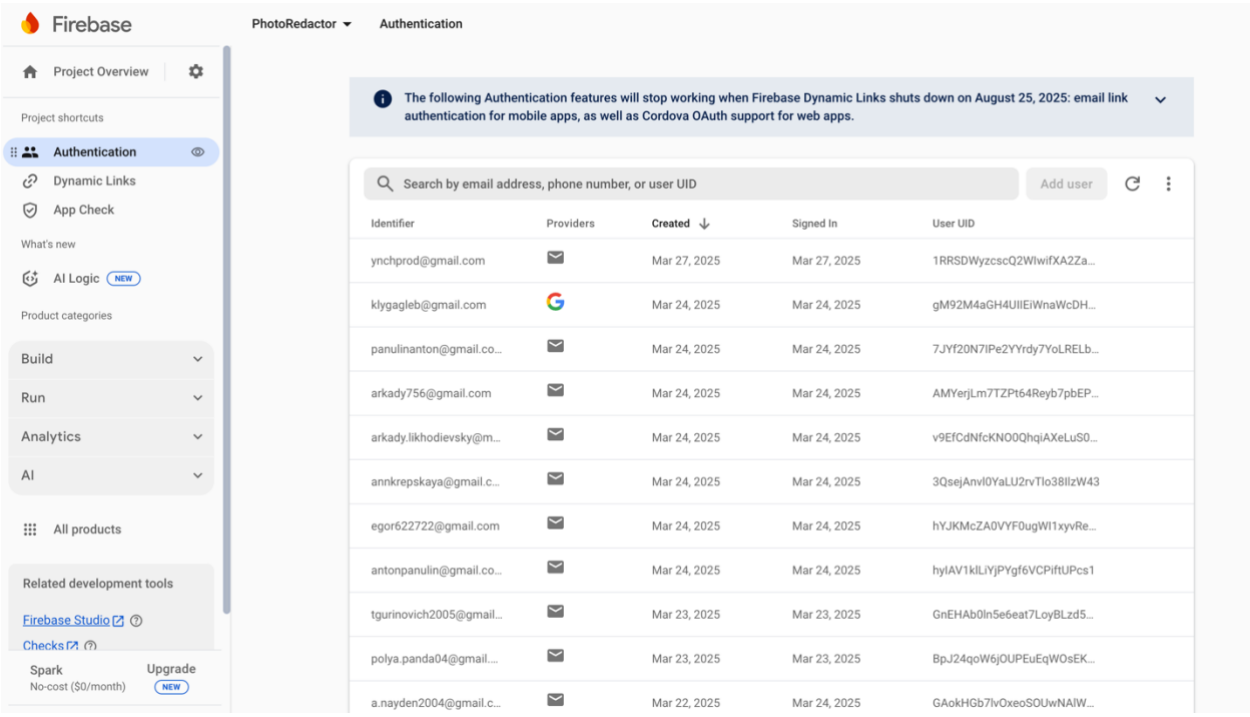

Рисунок 4.9 – Информация зарегистрированных пользователей

4.3 Организация хранения и управления данными

Для обеспечения надежного хранения и эффективного доступа к медицинским данным в приложении была разработана многоуровневая система работы с информацией. В качестве основного решения для локального хранения данных была выбрана встроенная база данных SQLite, обладающая рядом ключевых преимуществ для медицинских приложений.

SQLite представляет собой компактную, но мощную реляционную систему управления базами данных, которая идеально подходит для мобильных приложений. Основные преимущества данного решения включают:

Кроссплатформенную совместимость - единая структура базы данных работает как на Android, так и на iOS, что значительно сокращает время разработки и исключает проблемы с миграцией данных между платформами.

Высокую производительность - оптимизированные алгоритмы работы обеспечивают быстрый доступ к данным даже при работе с большими объемами медицинских показателей.

Надежность хранения - поддержка ACID-транзакций гарантирует целостность данных при любых сценариях работы приложения.

Для обеспечения безопасности конфиденциальной медицинской информации реализовано шифрование данных с использованием алгоритма TENT, который обеспечивает:

- Защиту данных на уровне всего файла базы данных;
- Высокую скорость шифрования/дешифрования;
- Минимальное влияние на производительность приложения.

Структура базы данных была тщательно продумана для эффективного хранения всех типов медицинской информации - от базовых показателей жизнедеятельности до сложных данных о циклах сна пользователя. На рисунке 4.10 представлена UML-диаграмма классов, наглядно демонстрирующая организацию таблиц и связей между ними в разработанной базе данных SQLite.

Данное решение обеспечивает стабильную работу приложения как при наличии интернет-соединения, так и в автономном режиме, что особенно важно для медицинских систем, где непрерывность мониторинга является критически важным требованием.

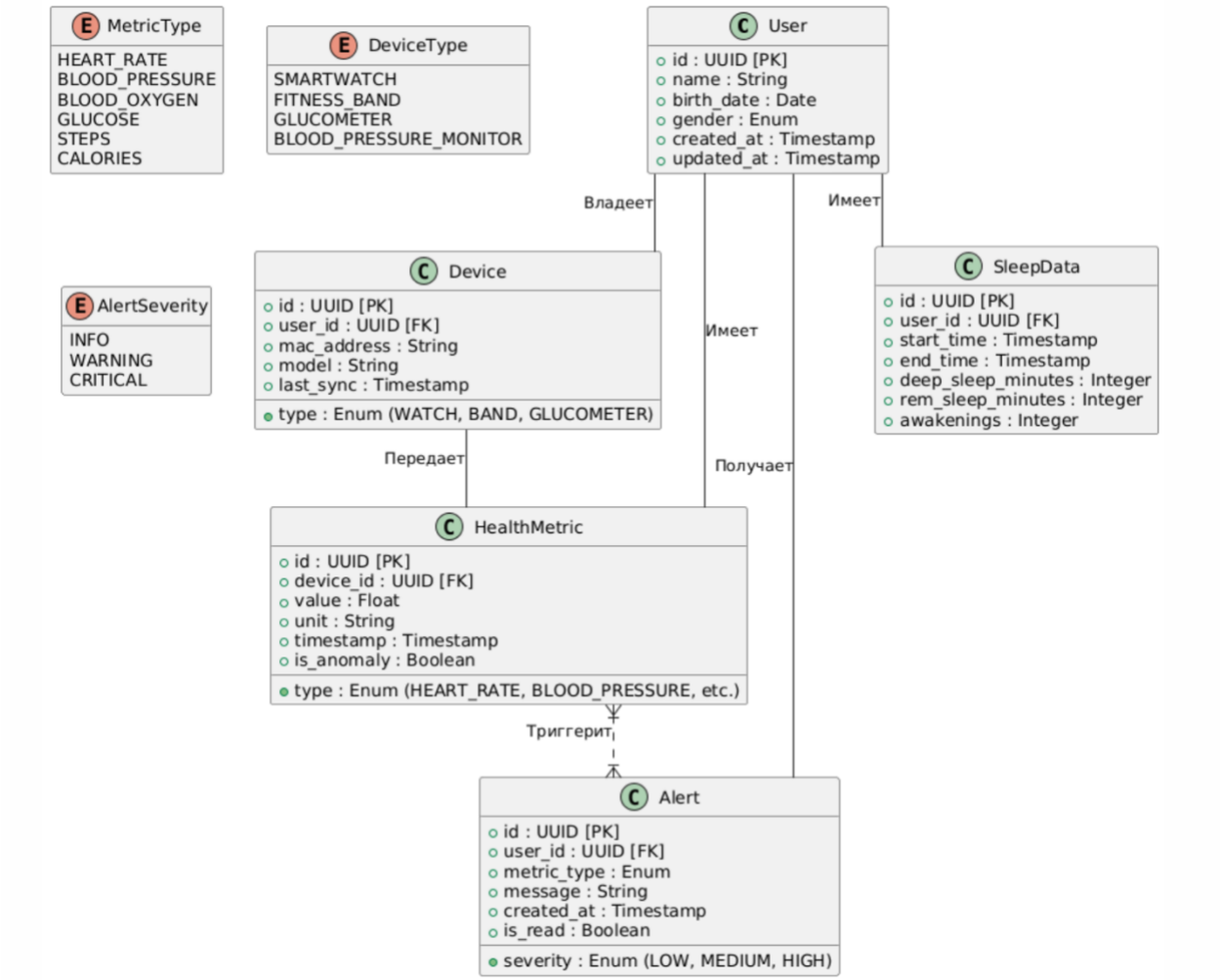

Рисунок 4.10 – UML-диаграмма классов SQLite

ВЫВОДЫ

В рамках данной главы была успешно реализована мобильная система мониторинга здоровья с использованием носимых устройств и мобильных сенсоров. Ключевые резул ьтаты разработки включают:

В рамках разработки пользовательского интерфейса были реализованы решения, направленные на обеспечение удобства и информативности отображения медицинских данных. Созданы интуитивно понятные экраны, позволяющие пользователю в реальном времени отслеживать такие показатели, как пульс, уровень кислорода в крови (SpO_2), артериальное давление и физическая активность.

Интерфейс поддерживает адаптивную визуализацию, включая графики и диаграммы, отображающие историю измерений и выявленные тренды. Это даёт пользователю возможность отслеживать динамику своего состояния в удобной форме.

Особое внимание уделено юзабилити: дизайн учитывает потребности различных возрастных групп, включая пожилых пользователей. Использованы крупные элементы управления, понятные пиктограммы и возможность увеличения шрифта, что повышает доступность приложения и снижает барьер использования для пользователей с ограниченным опытом работы с цифровыми технологиями.

Регистрация и аутентификация пользователей

Внедрена безопасная система входа через Firebase Authentication с поддержкой:

- Email/пароля;
- Социальных сетей (Google, Apple);
- Номера телефона (SMS-верификация).

Обеспечено соответствие требованиям GDPR и HIPAA за счет:

- Шифрования персональных данных;
- Возможности полного удаления аккаунта;
- Двухфакторной аутентификации для медицинского персонала.

Организация хранения данных

Для локального кэширования выбрана SQLite с шифрованием (SQLCipher), что позволяет:

- Работать оффлайн при отсутствии интернета;
- Обеспечивать высокую скорость доступа к истории измерений;
- Защищать конфиденциальные медицинские данные.

Для облачной синхронизации используется Cloud Firestore, обеспечивающий:

- Автоматическую репликацию данных между устройствами;
- Резервное копирование;
- Гибкие правила доступа;
- Интеграция с носимыми устройствами.

Реализована поддержка BLE (Bluetooth Low Energy), что позволяет подключать внешние устройства для мониторинга состояния здоровья пользователя. В частности, обеспечена совместимость с умными часами, такими как Apple Watch и устройства на базе Wear OS**,** а также с популярными фитнес-браслетами, включая Mi Band и Fitbit. Это расширяет функциональность приложения, обеспечивая более точный и регулярный сбор биометрических данных в режиме реального времени.

Реализованное мобильное приложение полностью соответствует поставленным задачам:

- Обеспечивает непрерывный мониторинг здоровья с помощью носимых устройств.
- Предоставляет аналитику в реальном времени на основе алгоритмов машинного обучения.
- Гарантирует безопасность и конфиденциальность данных.
- Поддерживает кроссплатформенность (iOS/Android) и оффлайн-работу.

ЗАКЛЮЧЕНИЕ

В данной работе была разработана система персонального мониторинга здоровья с использованием носимых устройств и мобильных сенсоров. Исследование подтвердило актуальность решений для удалённого контроля состояния пациентов и востребованность таких систем в цифровом здравоохранении.

На основе анализа технологий были выбраны оптимальные протоколы передачи данных, стандарты обмена медицинской информацией и архитектурные решения. Реализованное приложение обеспечивает сбор, хранение и анализ медицинских показателей в реальном времени, с поддержкой персонализированных норм и интеллектуальной аналитики.

Ключевые преимущества системы — адаптация под пользователя, комплексный мониторинг, прогнозирование рисков и удобство использования. Работа продемонстрировала техническую реализуемость и практическую значимость проекта, а также открыла перспективы для интеграции с EHR-системами и расширения функционала.

Разработанное решение может быть полезно для специалистов, занимающихся цифровой медициной, телемедициной и разработкой смарт-устройств.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Бледжянц, Г. А. Ключевые технологии формирования искусственного интеллекта в медицине / Г. А. Бледжянц, М. А. Саркисян, Ю. А. Исакова [и др.] // Ремедиум. Журнал о российском рынке лекарств и медицинской технике. – 2015. – № 12. – С. 10–15.
2. Гусев, А. В. Перспективы нейронных сетей и глубокого машинного обучения в создании решений для здравоохранения / А. В. Гусев // Врач и информационные технологии. – 2017. – № 3. – С. 92–105.
3. Жариков, О. Г. Нейросетевые технологии в медицине / О. Г. Жариков, Ю. В. Мещеряков, А. А. Литвин // Вопросы организации и информатизации здравоохранения. – 2007. – № 4 (53). – С. 59–63.
4. Костюк, К. И. Международный стандарт HL7 FHIR как основа создания единого информационного пространства здравоохранения Республики Беларусь / К. И. Костюк, А. В. Браницкий, В. В. Роубо, И. М. Нестерович – 2020. -№ 4 (83). – С. 83–91.
5. Машинное обучение поможет врачам принимать более информированные решения [Электронный ресурс] // Телемедицина.ru. – 2017. – Режим доступа: https://telemedicina.ru/news/equip/mashinnoe-obuchenie-pomojet-vracham-prinimat-bolee-informirovannyie-resheniya. – Дата доступа: 23.05.2025. – Заглавие с экрана.
6. Удаленный мониторинг пациентов в мире, управляемом искусственным интеллектом [Электронный ресурс] // Evercare. – Режим доступа: https://evercare.ru/news/udalennyy-monitoring-pacientov-v-mire-upravlyaemom-iskusstvennym-intellektom. – Дата доступа: 23.05.2025. – Заглавие с экрана.
7. Что такое NFC и как это работает? [Электронный ресурс] // Sravni.ru. – Режим доступа: https://www.sravni.ru/text/nfc/?upd=true. – Дата доступа: 23.05.2025. – Заглавие с экрана.
8. Bock, C. Healthcare Strategic Focus Area: Clinical Informatics / C. Bock [et al.]. – Gaithersburg: National Institute of Standards and Technology, 2005. – P. 1–33.
9. Dynamic Time Warping [Electronic resource]. – Mode of access: https://towardsdatascience.com/dynamic-time-warping-3933f25fcdd. – Date of access: 23.05.2025. – Title from the screen.
10. Fundamentals of RNN [Electronic resource]. – Mode of access: https://arxiv.org/abs/1808.03314. – Date of access: 23.05.2025. – Title from the screen.

11. Geiger, A. TadGAN: Time Series Anomaly Detection Using Generative Adversarial Networks [Electronic resource] / A. Geiger [et al.]. – 2020. – Mode of access: https://arxiv.org/abs/2009.07769. – Date of access: 23.05.2025. – Title from the screen.

12. Kononenko, I. Machine learning for medical diagnosis: history, state of the art and perspective / I. Kononenko // Artificial Intelligence in Medicine. – 2001. – Vol. 23, no. 1. – P. 89–109.

13. Orion: A Library for Anomaly Detection in Signals [Electronic resource]. – Mode of access: https://github.com/signals-dev/Orion. – Date of access: 23.05.2025. – Title from the screen.

ПРИЛОЖЕНИЕ А
Модель данных «Здоровья»

```
import Foundation
import HealthKit

class HealthKitManager: ObservableObject {
   fileprivate let healthKitStore = HKHealthStore()

   @Published var readings: [Reading] = []

   @MainActor
   func loadReadings() {
      readSampleByBloodPressure { newReadings in
         self.readings = newReadings
      }
   }

   private let systolicType = HKQuantityType.quantityType(forIdentifier:
.bloodPressureSystolic)!
   private let diastolicType = HKQuantityType.quantityType(forIdentifier:
.bloodPressureDiastolic)!
   private let heartRateType = HKQuantityType.quantityType(forIdentifier:
.heartRate)!

   func isSharingAuthorized() -> Bool {
      let systolicStatus = healthKitStore.authorizationStatus(for: systolicType)
      let diastolicStatus = healthKitStore.authorizationStatus(for: diastolicType)
      let heartRateStatus = healthKitStore.authorizationStatus(for: heartRateType)
      return systolicStatus == HKAuthorizationStatus.sharingAuthorized &&
diastolicStatus == HKAuthorizationStatus.sharingAuthorized && heartRateStatus
== HKAuthorizationStatus.sharingAuthorized
   }

   func authorizationRequestHealthKit(completion: @escaping (Bool, Error?) ->
Void) {
      guard HKHealthStore.isHealthDataAvailable() else {
         let error = NSError(domain: "com.zsoltkebel.mobile", code: 999,
                     userInfo: [NSLocalizedDescriptionKey : "Healthkit not
available on this device"])
         completion(false, error)
```

```
            print("HealthKit not available on this device")
            return
        }
        let types: Set<HKSampleType> = [systolicType, diastolicType,
heartRateType]
        healthKitStore.requestAuthorization(toShare: types, read: types) { (success:
Bool, error: Error?) in
            completion(success, error)
        }
    }

    func saveBloodPressureMeasurement(date startDate: Date = Date(), systolic:
Int, diastolic: Int, heartRate: Int, completion: @escaping (Bool, Error?) -> Void) {
        let endDate = startDate
        let systolicQuantity = HKQuantity(unit: HKUnit.millimeterOfMercury(),
doubleValue: Double(systolic))
        let systolicSample = HKQuantitySample(type: systolicType, quantity:
systolicQuantity, start: startDate, end: endDate)
        let diastolicQuantity = HKQuantity(unit: HKUnit.millimeterOfMercury(),
doubleValue: Double(diastolic))
        let diastolicSample = HKQuantitySample(type: diastolicType, quantity:
diastolicQuantity, start: startDate, end: endDate)
        let heartRateQuantity = HKQuantity(unit: HKUnit.count().unitDivided(by:
HKUnit.minute()), doubleValue: Double(heartRate))
        let heartRateSample = HKQuantitySample(type: heartRateType, quantity:
heartRateQuantity, start: startDate, end: endDate)
        let bpCorrelationType = HKCorrelationType.correlationType(forIdentifier:
.bloodPressure)!
        let bpCorrelation = HKCorrelation(type: bpCorrelationType, start: startDate,
end: endDate, objects: [systolicSample, diastolicSample, heartRateSample])
        healthKitStore.save(bpCorrelation) { success, error in
            if !success {
                print("Ошибка сохранения: \(error?.localizedDescription ??
"Неизвестная ошибка")")
            }
            completion(success, error)
        }
    }
```

```
    func deleteReading(_ reading: Reading, completion: @escaping (Bool) -> Void)
{
        healthKitStore.delete(reading.correlation) { success, error in
            if success {
                DispatchQueue.main.async {
                    self.readings.removeAll { $0.id == reading.id }
                    completion(true)
                }
            } else {
                print("Ошибка удаления: \(error?.localizedDescription ?? "Unknown
error")")
                completion(false)
            }
        }
    }


    func readSampleByBloodPressure(completion: @escaping ([Reading]) -> Void)
{
        let predicate = HKQuery.predicateForSamples(withStart: nil, end: nil,
options: .strictStartDate)
        let type = HKCorrelationType(.bloodPressure)
        let sampleQuery = HKSampleQuery(sampleType: type, predicate: predicate,
limit: HKObjectQueryNoLimit, sortDescriptors: nil)
        { (sampleQuery, results, error) -> Void in
            if let error = error {
                print("Error: \(error.localizedDescription)")
                return
            }

            guard let samples = results as? [HKCorrelation] else {
                return
            }

            var readings: [Reading] = []
            for sample in samples {
                if let sys = sample.objects(for:
HKQuantityType(.bloodPressureSystolic)).first as? HKQuantitySample,
                    let dia = sample.objects(for:
HKQuantityType(.bloodPressureDiastolic)).first as? HKQuantitySample {
```

```
                let value1 = sys.quantity.doubleValue(for:
HKUnit.millimeterOfMercury())
                let value2 = dia.quantity.doubleValue(for:
HKUnit.millimeterOfMercury())
                let heartRateSample = sample.objects(for:
HKQuantityType(.heartRate)).first as? HKQuantitySample
                let heartRate = heartRateSample?.quantity.doubleValue(for:
HKUnit.count().unitDivided(by: HKUnit.minute())) ?? 0

                readings.append(Reading(sys: value1, dia: value2, heartRate:
heartRate, date: sample.startDate, correlation: sample))
            }
        }
        DispatchQueue.main.async {
            completion(readings)
        }
    }
    self.healthKitStore.execute(sampleQuery)
  }
}
```

ПРИЛОЖЕНИЕ Б
Интерфейс главного экрана

```
import SwiftUI

struct HeartView: View {
   @EnvironmentObject private var coordinator: AppCoordinator

   var body: some View {
      ScrollView(.vertical, showsIndicators: false) {
         VStack {
            GraphicView()
            HeartRateBannerView()
            HStack {
               addRecordsView(onTap: { coordinator.navigate(to: .addData) })
               scanMetersView(onTap: { coordinator.navigate(to: .scan) })
            }
            .padding(.top, 8)
            .padding(.horizontal, 16)
         }
      }
      .navigationBarTitleDisplayMode(.inline)
      .navigationBarBackButtonHidden(true)
      .toolbar {
         ToolbarItem(placement: .topBarLeading) {
            Text("Tracker")
               .font(.title2)
               .fontWeight(.bold)
         }
         ToolbarItem(placement: .topBarTrailing) {
            ProView()
               .frame(width: 82, height: 40)
         }
      }
   }
}
```

ПРИЛОЖЕНИЕ В
Детекция данных с тонометра

```
import SwiftUI
import Vision
import VisionKit
import CoreImage

struct ScannerView: UIViewControllerRepresentable {
   @EnvironmentObject private var coordinator: AppCoordinator

   func makeCoordinator() -> Coordinator {
      Coordinator(coordinator: coordinator)
   }

   func makeUIViewController(context: Context) ->
VNDocumentCameraViewController {
      let scanner = VNDocumentCameraViewController()
      scanner.delegate = context.coordinator
      return scanner
   }

   func updateUIViewController(_ uiViewController:
VNDocumentCameraViewController, context: Context) {}

   class Coordinator: NSObject, VNDocumentCameraViewControllerDelegate {
      private var coordinator: AppCoordinator

      init(coordinator: AppCoordinator) {
         self.coordinator = coordinator
      }

      func documentCameraViewController(_ controller:
VNDocumentCameraViewController, didFinishWith scan:
VNDocumentCameraScan) {
         controller.dismiss(animated: true)
         let images = (0..<scan.pageCount).map { scan.imageOfPage(at: $0) }
         recognizeText(from: images)
      }
```

```
    private func recognizeText(from images: [UIImage]) {
        let request = VNRecognizeTextRequest { [weak self] request, error in
            guard let self = self,
                  let observations = request.results as?
[VNRecognizedTextObservation],
                  error == nil
            else { return }

            let text = observations.compactMap { $0.topCandidates(1).first?.string
}.joined(separator: " ")
            self.extractValues(from: text)

            DispatchQueue.main.async {
                self.coordinator.navigate(to: .addData)
            }
        }
        request.recognitionLevel = .accurate
        request.usesLanguageCorrection = true
        request.customWords = ["SYS", "DIA", "PULSE"]
        request.revision = VNRecognizeTextRequestRevision2

        if let cgImage = preprocessImage(images.first!) {
            let handler = VNImageRequestHandler(cgImage: cgImage, options: [:])
            DispatchQueue.global(qos: .userInitiated).async {
                try? handler.perform([request])
            }
        }
    }

    private func extractValues(from text: String) {
        let pattern = "(\\d{2,3})[^\\d]+(\\d{2,3})[^\\d]+(\\d{2,3})"
        guard let regex = try? NSRegularExpression(pattern: pattern),
              let match = regex.firstMatch(in: text, range: NSRange(text.startIndex...,
in: text))
        else { return }

        let nsText = text as NSString
        let numbers = (1...3).compactMap {
            let range = match.range(at: $0)
```

```
            return range.location != NSNotFound ? nsText.substring(with: range) :
nil
        }

        guard numbers.count == 3 else { return }

        DispatchQueue.main.async { [weak self] in
            self?.coordinator.savedSystolic = Int(numbers[0])
            self?.coordinator.savedDiastolic = Int(numbers[1])
            self?.coordinator.savedPulse = Int(numbers[2])
        }
    }

    private func preprocessImage(_ image: UIImage) -> CGImage? {
        guard let ciImage = CIImage(image: image) else { return nil }
        let filter = CIFilter(name: "CIColorControls")
        filter?.setValue(ciImage, forKey: kCIInputImageKey)
        filter?.setValue(1.2, forKey: kCIInputContrastKey)
        let context = CIContext()
        return filter?.outputImage.flatMap { context.createCGImage($0, from:
$0.extent) }
    }
  }
}
```

ПРИЛОЖЕНИЕ Г
Построение графиков для отслеживания здоровья

```
import SwiftUI
import Charts

struct GraphicView: View {
    @EnvironmentObject var healthKitManager: HealthKitManager
    @State private var selectedInterval: TimeInterval = .day

    private var dateFormatter: DateFormatter {
        let formatter = DateFormatter()
        switch selectedInterval {
        case .day:
            formatter.dateFormat = "h a"
        case .week, .month:
            formatter.dateFormat = "dd MMM"
        }
        return formatter
    }

    private var filteredReadings: [Reading] {
        let calendar = Calendar.current
        let now = Date()
        let startDate: Date

        switch selectedInterval {
        case .day:
            startDate = calendar.date(byAdding: .day, value: -1, to: now)!
        case .week:
            startDate = calendar.date(byAdding: .weekOfYear, value: -1, to: now)!
        case .month:
            startDate = calendar.date(byAdding: .month, value: -1, to: now)!
        }

        return healthKitManager.readings.filter { $0.date >= startDate }
    }

    var body: some View {
        VStack {
```

```
Picker("Период", selection: $selectedInterval) {
   ForEach(TimeInterval.allCases) { interval in
      Text(interval.rawValue).tag(interval)
   }
}
.pickerStyle(.segmented)
.padding()

if filteredReadings.isEmpty {
   Text("Нет данных")
      .padding()
} else {
   Chart {
      ForEach(filteredReadings) { reading in
         RectangleMark(
            x: .value("Время", reading.date),
            yStart: .value("Диастолическое", reading.dia - 6),
            yEnd: .value("Систолическое", reading.sys + 6),
            width: 20
         )
         .cornerRadius(20)
         .foregroundStyle(.gray.opacity(0.5))

         PointMark(
            x: .value("Время", reading.date),
            y: .value("Систолическое", reading.sys)
         )
         .symbolSize(160)
         .foregroundStyle(.red)

         PointMark(
            x: .value("Время", reading.date),
            y: .value("Диастолическое", reading.dia)
         )
         .symbolSize(160)
         .foregroundStyle(.blue)
      }
   }
   .chartXAxis {
      AxisMarks(position: .bottom) { value in
```

```
                AxisGridLine()
                AxisTick()
                if let date = value.as(Date.self) {
                    AxisValueLabel(dateFormatter.string(from: date))
                }
            }
        }
        .chartYAxis {
            AxisMarks(position: .leading) { value in
                AxisGridLine()
                AxisTick()
                AxisValueLabel()
            }
        }
        .frame(height: 300)
        .padding(.horizontal, 16)
        .padding(.vertical, 8)
        HStack {
            Circle().frame(width: 10, height: 10).foregroundStyle(.red)
            Text("SYS")
            HStack {
                Circle().frame(width: 10, height: 10).foregroundStyle(.blue)
                Text("DIA")
            }
            .padding(.leading)
        }
      }
    }
    .onAppear {
      healthKitManager.loadReadings()
    }
  }
}

enum TimeInterval: String, CaseIterable, Identifiable {
  case day = "Day"
  case week = "Week"
  case month = "Month"

  var id: String { rawValue }
```

```
}
```

ПРИЛОЖЕНИЕ Д
Построение графиков для отслеживания здоровья

```
import SwiftUI
import AVFoundation
import Accelerate

class PulseMonitor: NSObject, ObservableObject {
    @Published var isMeasuring = false
    @Published var progress: CGFloat = 0.0
    @Published var pulse: Int?
    @Published var showError = false
    @Published var errorMessage = ""

    var captureDevice: AVCaptureDevice?
    private var lastBrightnessValues: [Double] = []
    private let requiredSamples = 256
    private let fps: Double = 30.0
    private let minValidBPM: Double = 40.0
    private let maxValidBPM: Double = 200.0
    private let bandpassFilter = BandpassFilter()


    func startMeasurement() {
        reset()
        DispatchQueue.main.async {
            self.isMeasuring = true
            self.toggleFlash(on: true)
        }
    }

    func stopMeasurement() {
        DispatchQueue.main.async {
            self.isMeasuring = false
            self.toggleFlash(on: false)
        }
        calculatePulse()
    }
```

```
    func toggleFlash(on: Bool) {
        guard let device = captureDevice, device.hasTorch else { return }
        do {
            try device.lockForConfiguration()
            device.torchMode = on ? .on : .off
            device.unlockForConfiguration()
        } catch {
            print("Flash error: \(error)")
        }
    }

    func processBuffer(_ buffer: CMSampleBuffer) {
        guard isMeasuring else { return }

        let brightness = getBrightness(from: buffer)
        lastBrightnessValues.append(brightness)

        DispatchQueue.main.async {
            self.progress = CGFloat(self.lastBrightnessValues.count) /
CGFloat(self.requiredSamples)
        }

        if lastBrightnessValues.count >= requiredSamples {
            stopMeasurement()
        }
    }

    func requestCameraPermission() {
        AVCaptureDevice.requestAccess(for: .video) { granted in
            if granted {
                print("Доступ к камере разрешен")
            } else {
                print("Доступ к камере запрещен")
            }
        }
    }

    private func calculatePulse() {
        guard lastBrightnessValues.count >= requiredSamples else { return }
```

```
        let normalized = normalize(lastBrightnessValues)
        guard let filtered = bandpassFilter.process(data: normalized) else {
            showError(message: "Низкое качество сигнала")
            return
        }
        let peaks = findSignificantPeaks(in: filtered)
        guard peaks.count >= 3 else {
            showError(message: "Недостаточно данных")
            return
        }
        let intervals = calculatePeakIntervals(peaks: peaks)
        guard let averageInterval = intervals.isEmpty ? nil : intervals.reduce(0, +) /
Double(intervals.count), averageInterval > 0 else {
            showError(message: "Некорректные интервалы")
            return
        }
        let bpm = 60.0 / (averageInterval / fps)
        guard (minValidBPM...maxValidBPM).contains(bpm) else {
            showError(message: "Некорректное значение: \(Int(bpm))")
            return
        }

        DispatchQueue.main.async {
            self.pulse = Int(bpm.rounded())
        }
    }

    private func findSignificantPeaks(in data: [Double]) -> [Int] {
        guard !data.isEmpty else { return [] }
        let sortedData = data.sorted(by: >)
        let thresholdIndex = min(data.count / 10, sortedData.count - 1)
        let threshold = 0.6 * sortedData[thresholdIndex]

        var peaks = [Int]()
        for i in 2..<data.count-2 {
            let window = data[(i-2)...(i+2)]
            if data[i] == window.max() && data[i] > threshold {
                peaks.append(i)
            }
        }
```

```
        return peaks
    }

    private func calculatePeakIntervals(peaks: [Int]) -> [Double] {
        var intervals = [Double]()
        for i in 1..<peaks.count {
            intervals.append(Double(peaks[i] - peaks[i-1]))
        }
        return intervals
    }

    private func showError(message: String) {
        DispatchQueue.main.async {
            self.pulse = nil
            self.errorMessage = message
            self.showError = true
        }
    }

    private func getBrightness(from buffer: CMSampleBuffer) -> Double {
        guard let imageBuffer = CMSampleBufferGetImageBuffer(buffer) else {
return 0 }
        CVPixelBufferLockBaseAddress(imageBuffer, .readOnly)

        let width = CVPixelBufferGetWidth(imageBuffer)
        let height = CVPixelBufferGetHeight(imageBuffer)
        let bytesPerRow = CVPixelBufferGetBytesPerRow(imageBuffer)
        guard let baseAddress = CVPixelBufferGetBaseAddress(imageBuffer) else {
return 0 }

        var totalBrightness: Double = 0
        let buffer = baseAddress.assumingMemoryBound(to: UInt8.self)

        for y in 0..<height {
            for x in 0..<width {
                let pixel = buffer[y * bytesPerRow + x * 4]
                totalBrightness += Double(pixel)
            }
        }
```

```
        CVPixelBufferUnlockBaseAddress(imageBuffer, .readOnly)
        return totalBrightness / Double(width * height)
    }

    private func normalize(_ data: [Double]) -> [Double] {
        guard !data.isEmpty else { return [] }
        let mean = data.reduce(0, +) / Double(data.count)
        return data.map { $0 - mean }
    }

    private func reset() {
        lastBrightnessValues.removeAll()
        pulse = nil
        progress = 0.0
        bandpassFilter.reset()
    }
}
```