



Московский государственный университет имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра автоматизации систем вычислительных комплексов

Аникевич Юлия Вадимовна

**Разработка и реализация серверной части платформы
для сбора и обработки медицинской телеметрии**

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Научный руководитель:
к.ф.-м.н., в.н.с.
А.Г. Бахмурев

Научный консультант:
А.А. Любицкий

Москва, 2021

Аннотация

Разработка и реализация серверной части платформы
для сбора и обработки медицинской телеметрии

Аникевич Юлия Вадимовна

На сегодняшний день проблема мониторинга состояния человека чрезвычайно важна и актуальна. В рамках выпускной квалификационной работы была разработана и реализована серверная часть открытой платформы для сбора и обработки физиологических данных о человеке, которая позволяет собирать данные с персональных мониторинговых устройств и тем самым помогает осуществлять наблюдение за состоянием здоровья человеческого организма и влиять на профилактический, лечебный и реабилитационный процессы.

Содержание

1 Введение	5
2 Постановка задачи	7
3 Обзор существующих решений	8
4 Проектирование сервиса	10
4.1 Общая схема	10
4.2 Носимые устройства	10
4.3 Сценарии использования	11
5 Описание реализации	14
5.1 Протокол передачи данных мониторинга	14
5.1.1 MQTT	14
5.1.2 Структура топиков	16
5.1.3 Формат передачи данных	17
5.2 Аутентификация и авторизация	17
5.2.1 JSON Web Token	17
5.2.2 Схема аутентификации	19
5.3 Хранение данных	21
5.3.1 Данные мониторинга	21
5.3.2 Аккаунты пользователей и информация об устройствах	22
5.4 Веб-сервис	24
5.4.1 Описание методов API	26
6 Апробация платформы	29
7 Заключение	32
Список литературы	33
Приложение А. Скриншоты интерфейса оператора	34

1 Введение

Такое направление в здравоохранении, как интернет медицинских вещей (IoMT, Internet of Medical Things), становится с каждым днем все известнее, практичнее и удобнее. Идет активное развитие этого направления: с одной стороны развитие технологий, вовлечение в их создание широкого круга ученых и инженеров, с другой стороны ширится использование этих технологий и их проникновение в повседневность.

Это понятие обозначает концепцию сети, позволяющую объединять определенные устройства и приборы, которые способны осуществлять отслеживание за состоянием человеческого организма, окружающей его среды и влиять на профилактический, лечебный и реабилитационный процессы. Возможность анализировать непрерывный поток данных о здоровье с носимых устройств — это важный аспект, который позволяет изменить взгляд на здравоохранение и процессы внутри него. Поэтому на сегодняшний день проблема мониторинга состояния человека чрезвычайно актуальна.

Это направление существует для того, чтобы быстро получать детальную информацию об организме человека. Сегодня эта концепция позволяет непрерывно контролировать состояние человека, и, с помощью оценки диагностических показателей, выявлять случаи отклонения их от нормы. На основании такой информации специалист сможет принимать настоящие обоснованные решения, с помощью которых будет возможно предотвратить возникновение и развитие заболеваний. Более того, такая система позволит создать более комфортную и оперативную связь между врачом и пациентом, что значительно улучшит качество всех мероприятий, осуществляемых с целью улучшения самочувствия.

Кроме всего прочего, особенность персональных мониторных устройств состоит в том, что с помощью них можно не только выявлять нарушения в работе систем организма на ранней стадии, но и вовремя заметить переход состояния пациента из нейтрального в критическое и как можно раньше обеспечить специалиста информацией для успешного принятия решения по стабилизации состояния пациента.

Сегодня существует широкий выбор на рынке портативных носимых устройств и систем для регистрации физиологических параметров человека. Некоторая часть из них имеет свои решения для обработки и хранения собираемых данных. Однако в большинстве своем они не являются открытыми проектами, и данные из них получить либо

невозможно, либо они выдаются уже в обработанном виде. Таким образом, создание открытой платформы сбора и обработки физиологических данных является актуальной задачей на сегодняшний день.

2 Постановка задачи

Целью данной работы является создание серверной части платформы для сбора и обработки физиологических данных о человеке. Серверная часть представляет собой программное обеспечение, которое предоставляет интерфейс загрузки данных мониторинга состояния человека с мобильного устройства и интерфейс для получения данных, и инструмент для хранения и обработки данных мониторинга. Создаваемое решение может быть впоследствии масштабировано при помощи технологии виртуализации сетевых функций.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Реализация регистрации, аутентификации и авторизации пользователей;
2. Добавление поддержки устройств различных типов;
3. Проектирование и реализация интерфейса оператора для экспорта данных;
4. Развёртывание сервиса на сервере ЛВК;
5. Интеграция с клиентской частью на уровне, допускающем практическое использование;
6. Проведение апробации платформы.

3 Обзор существующих решений

На данный момент существует ряд систем мониторинга здоровья человека, которые широко используются на практике. В их число входят как собственные платформы производителей носимых устройств, такие как Fitbit, Omron, Apple Health и многие другие, так и интеграционные системы, такие как Validic, Seqster, CareSimple, Capsule Technologies.

Платформа Validic [1] предоставляет удобный и легкий доступ к цифровым данным о состоянии здоровья человека с более, чем 400 различных устройств клинического и удаленного мониторинга, фитнес-устройств и приложений для здоровья. При передаче данные проверяются на корректность. Сервис предоставляет большой набор инструментов для администрирования, регистрации и обслуживания пользователей, управления соединениями с устройствами. Также эта платформа предоставляет мобильную технологию, позволяющую собирать и передавать в реальном времени данные с устаревших медицинских устройств, не имеющих возможность подключиться к сети. В число клиентов Validic входят более 223 миллионов человек в 52 странах.

Seqster [2] является безопасной платформой для сбора данных о здоровье, которая предоставляет пользователям возможность сохранять информацию, собранную с носимых устройств и их сенсоров, систем удаленного мониторинга, результатов лабораторных тестов, электронных историй болезни. Также пользователи могут обмениваться данными о своем здоровье с членами семьи, создавая медицинские записи для нескольких поколений.

CareSimple [3] позволяет передавать данные с мобильных медицинских устройств от пациентов в реальном времени на клинический веб-портал и сопровождается дополнительным приложением для пациентов, доступным на любом смартфоне или планшете iOS и Android.

Все перечисленные программные продукты являются проприетарными, из них данные получить невозможно или они выдаются в обработанном виде. Они не являются бесплатными и открытыми проектами и по этой причине нет возможности дальше развивать эти решения в академической среде.

Также существуют IoT платформы для сбора телеметрии с открытым исходным кодом [4], но их использование все равно потребует значительного набора настроек. Дан-

ные решения могут не предоставлять полную свободу в дальнейшем развитии, поэтому было принято решение создавать свой собственный проект, который будет наилучшим образом подходить под требования в использовании и предоставлять возможность свободно развивать полученные результаты.

4 Проектирование сервиса

4.1 Общая схема

Разрабатываемый сервис состоит из клиентской и серверной частей. Клиентская часть реализует пользовательский интерфейс в виде мобильного приложения, организует работу с носимыми устройствами и их датчиками и передачу данных от носимого устройства на сервер, формирует запросы к серверу и обрабатывает ответы от него.

Серверная часть представляет собой программное обеспечение, которое реализует взаимодействие с клиентом, предоставляет интерфейс для загрузки данных мониторинга, осуществляет организацию, хранение и обработку полученных данных, а также предоставляет интерфейс оператора для их получения.

Упрощенная схема работы разрабатываемого сервиса представлена в следующих пунктах:

1. Физиологические показатели пользователячитываются с помощью датчиков носимого устройства, после чего отправляются на мобильный телефон на операционной системе Android по протоколу Bluetooth Low Energy (BLE).
2. Мобильный телефон принимает считанные показатели и отправляет их на сервер по протоколу MQTT. При разрывах соединения в сети Интернет, мобильное приложение реализует локальное хранение данных до момента восстановления разорванного соединения.
3. Сервер записывает полученные данные в базу данных, откуда они будут доступны через веб-интерфейс для медицинской обработки.

4.2 Носимые устройства

Одной из особенностей разрабатываемой платформы является поддержка нескольких типов носимых устройств. Для поддержки нового типа устройства его нужно зарегистрировать в системе. Устройства нового типа должны иметь открытый программный интерфейс и поддержку BLE.

Примером носимого устройства, используемого в данной работе, является умная одежда Hexoskin [5], которая имеет открытый программный интерфейс и является разработкой компании Carre Technologies Inc. Умная одежда - это одежда, которая может взаимодействовать с окружающей средой, воспринимая сигналы, обрабатывая информацию и запуская ответные реакции. Одежда Hexoskin способна производить мониторинг таких показателей, как ЭКГ и сердцебиение, вариабельность сердечного ритма, события QRS, восстановление сердечного ритма, частот дыхания, минутная вентиляция, интенсивность активности, пиковое ускорение, шаги, ритм, позиции и трекер сна (Рис. 1).

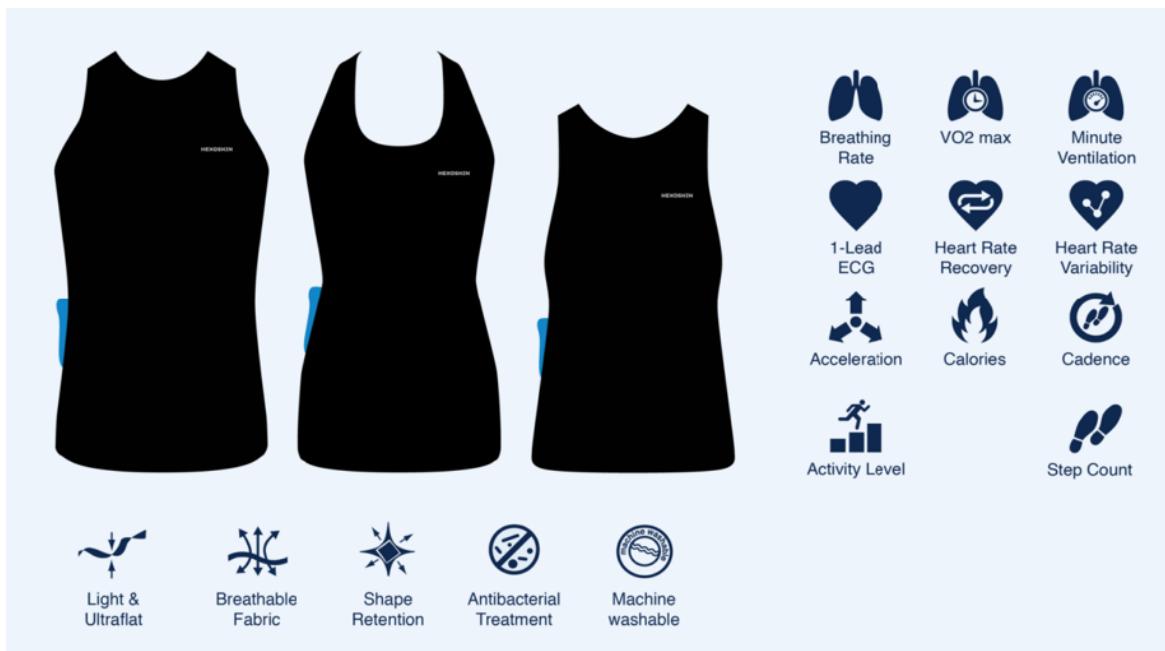


Рис. 1: Умная одежда Hexoskin

4.3 Сценарии использования

Для описания логической модели разрабатываемой системы можно применить визуальное моделирование в UML. Диаграммы вариантов использования описывают взаимоотношения и зависимости между вариантами использования и действующими лицами, называемыми акторами. Такие диаграммы определяют какой функциональностью должна обладать система, не затрагивая ее внутреннее устройство.

На рис. 2 представлена UML-диаграмма вариантов использования разрабатываемой

платформы. С платформой взаимодействуют такие акторы, как неавторизированный пользователь, авторизованный пользователь, оператор и администратор сервиса.

Неавторизированный пользователь имеет возможность авторизоваться или зарегистрироваться в мобильном приложении. Для регистрации необходимо заполнить форму регистрации, указать логин, пароль, ФИО, почтовый адрес, дату рождения и номер телефона. На указанный почтовый адрес отправляется письмо с просьбой подтвердить регистрацию.

Зарегистрированный пользователь может авторизоваться в системе. Далее, в мобильном приложении авторизованный пользователь может выполнить следующие действия:

- Зарегистрировать новое носимое устройство. Пользователь имеет возможность добавить новое устройство из списка доступных для регистрации типов устройств, поддерживаемых платформой. При регистрации, пользователь может для удобства называть свои устройства.
- Выбрать устройство для подключения из списка зарегистрированных устройств и подключиться к нему. После этого пользователь может начать отправку данных с датчиков подключенного устройства на сервер.
- Добавлять или обновлять личную информацию о себе, включая физиологические данные, такие как рост и вес.

Для просмотра данных пользователей оператору необходимо авторизоваться через веб-интерфейс. После этого оператору будут доступны такие действия, как:

- Выгрузка данных мониторинга в файл в формате csv. Для этого необходимо выбрать пользователя, чьи данные необходимо выгрузить, устройство из списка зарегистрированных устройств пользователя, временной интервал и список датчиков.
- Просмотр панелей мониторинга. Панели позволяют визуализировать последние данные, полученные с носимых устройств пользователей.
- Просмотр личной информации пользователей. Таким образом, при необходимости можно связаться с наблюдаемыми или сделать какие-то более обоснованные выводы при медицинской диагностике на основе информации, которую указывают пользователи.

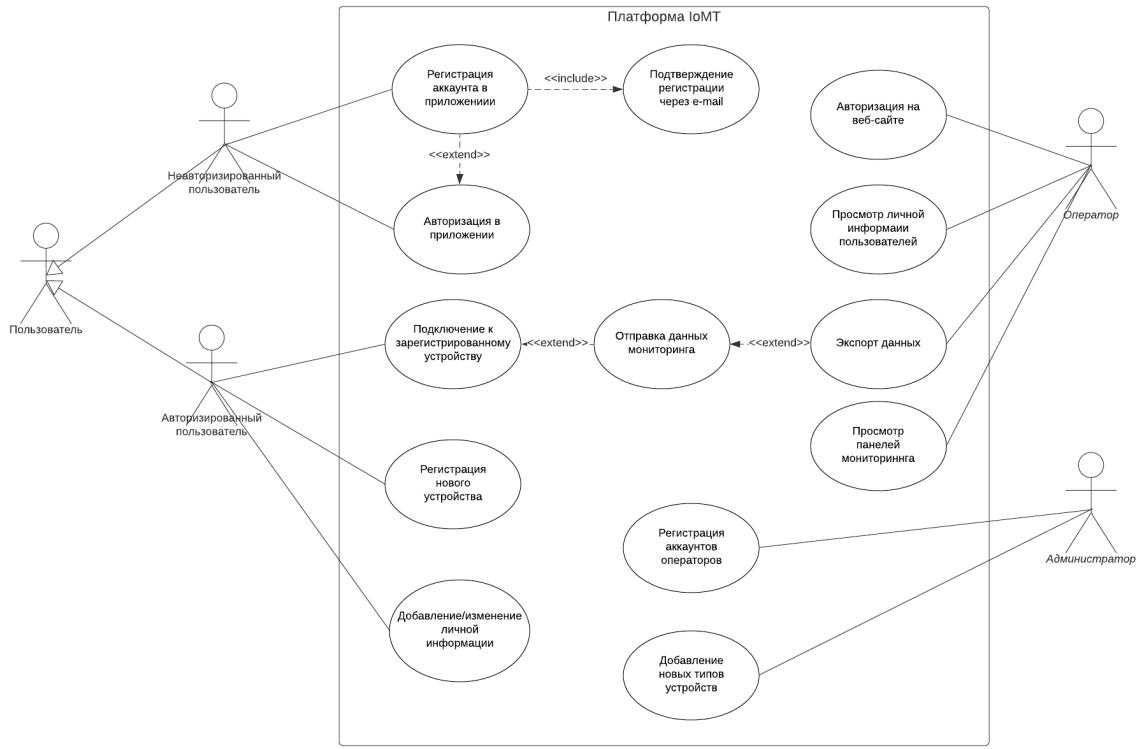


Рис. 2: Сценарии использования сервиса

Администратор сервиса выполняет такие функции, как добавление нового типа устройства, которое будет доступно для регистрации пользователями, а также создание аккаунтов для операторов.

5 Описание реализации

На рисунке 3 представлена архитектура серверной части платформы. Более детальное описание каждого элемента и их интерфейсов приводится в следующих параграфах.

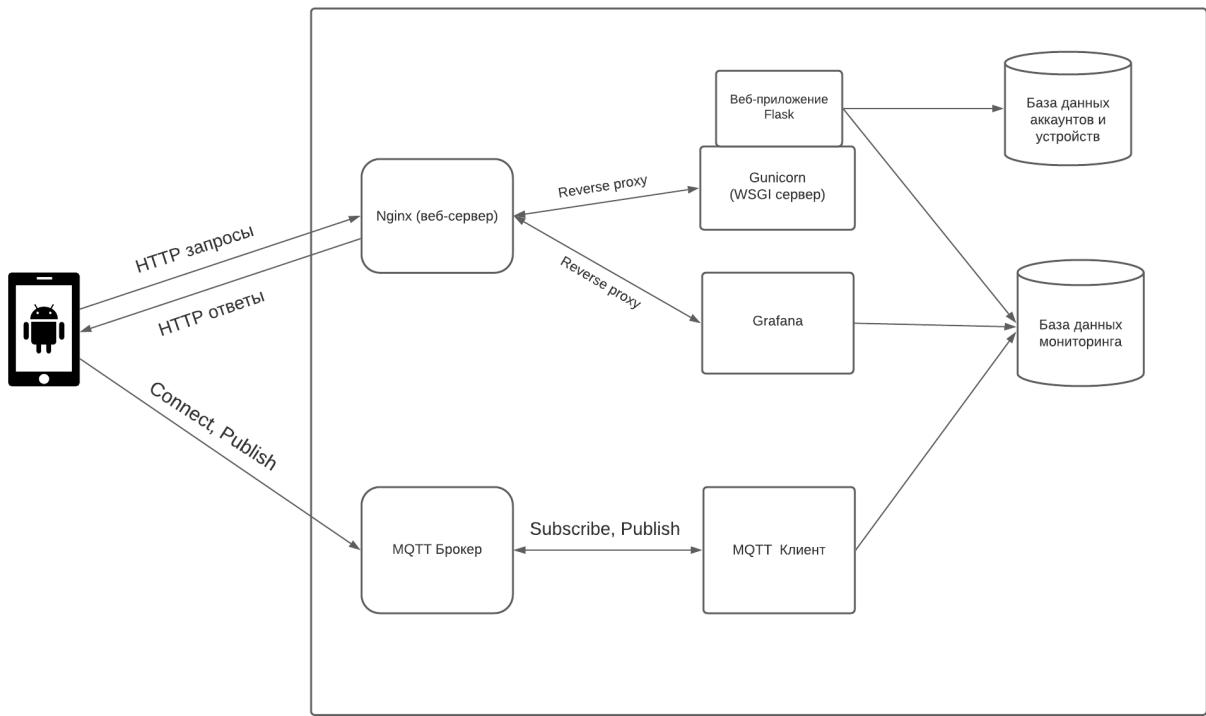


Рис. 3: Архитектура серверной части платформы

5.1 Протокол передачи данных мониторинга

5.1.1 MQTT

В качестве протокола передачи данных мониторинга используется протокол MQTT [6]. Это сетевой протокол, работающий поверх TCP/IP, ориентированный для обмена сообщениями между устройствами по принципу издаватель-подписчик. Весь обмен данными происходит через центральный компонент, называемый брокером. На него ложатся функции по доставке сообщений, их маршрутизации и обеспечению качества обслуживания QoS (Quality of Service), что позволяет клиентам оставаться легковесными, используя возможности брокера. Клиент может быть издателем, подписчиком или иметь

обе роли сразу. Издатель не требует никаких предварительных настроек по количеству подключаемых клиентов и их месторасположению. TCP-порт 1883 зарезервирован для протокола MQTT. Если задействуется TLS для защиты связи между клиентом и брокером, то используется порт 8883. Все клиенты должны однозначно идентифицировать себя при подключении к брокеру.

На рисунке 4 изображена простая схема взаимодействия между брокером и двумя клиентами. Один из клиентов является издателем и публикует данные, а другой – совершает подписку на топик, в который публикует сообщения первый клиент.

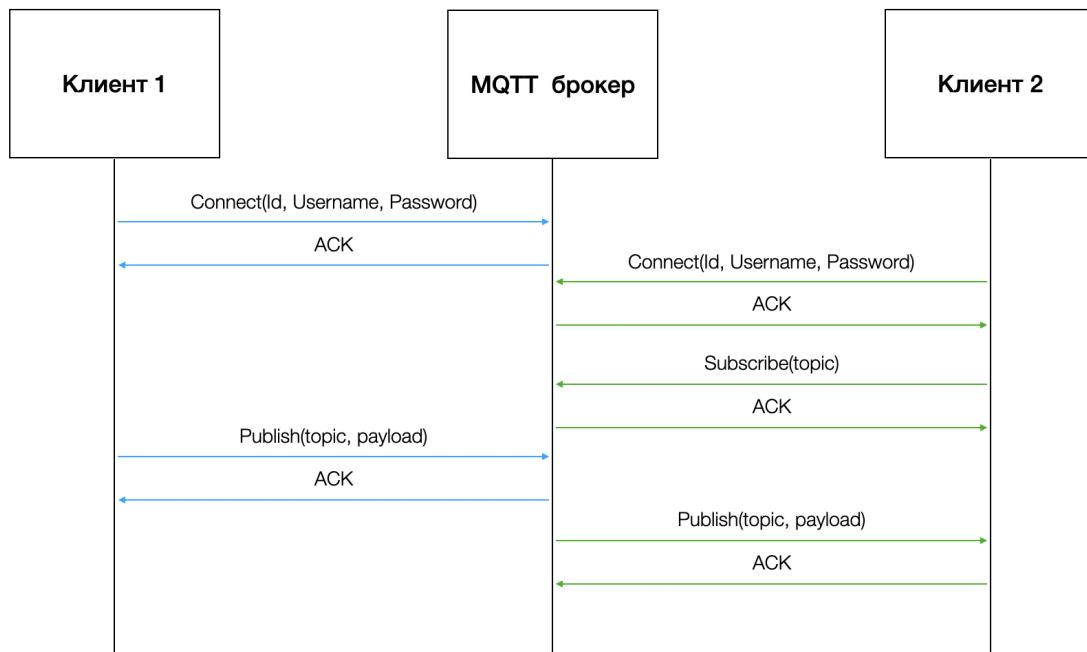


Рис. 4: Схема взаимодействия посредством протокола MQTT

Для использования в реализации был выбран брокер Eclipse Mosquitto [7]. Eclipse Mosquitto – брокер с открытым исходным кодом (лицензии EPL/EDL), который реализует протоколы MQTT различных версий.

5.1.2 Структура топиков

Каждое публикуемое сообщение содержит поле для имени топика, которое идентифицируют публикуемые данные. Аналогичным образом, клиент имеет возможность подписаться на определенные топики, если заинтересован в получении конкретных его данных. Топики имеют иерархическую структуру в формате “дерева”, что упрощает их организацию и доступ к данным.

При передаче данных между клиентом и сервером в разрабатываемом сервисе топики имеют следующую структуру:

/from/user_id/device_id/action

, где:

- from – часть, которая показывает кто публикует данное сообщение – клиент или сервер. Данный элемент позволяет читать серверу сообщения от клиентов, а клиентам – исключительно от сервера.
- user_id - уникальный идентификатор пользователя, который выдается каждому пользователю при регистрации. Данный элемент позволяет публиковать сообщения в топик, предназначенный для отправки данных только этого пользователя, а так же позволяет отправлять сообщения от сервера конкретному пользователю.
- device_id - уникальный идентификатор устройства, с которого отправляются данные. Данный идентификатор представляет собой MAC-адрес устройства.
- action – действие, показывающее цель отправки сообщения.

Древовидная структура помогает организовать клиентам получение данных сразу с нескольких топиков. Для этого существуют "wildcard" символы двух типов: одноуровневые ('+') и многоуровневые ('\#'). Например, таким образом клиент может подписаться на все топики, адресованные ему со стороны сервера, с помощью следующего варианта подписки: /s/{user_id}/#.

5.1.3 Формат передачи данных

В качестве формата обмена данными между клиентом и сервером был выбран формат JavaScript Object Notation (JSON). Это известный текстовый формат, полностью независимый от языка реализации, но также он использует соглашения, знакомые программистам С-подобных языков, таких как Java, Python и многих других. Так как при разработке в клиентской и в серверной частях используются различные языки программирования, кроссплатформенность и поддерживаемость различными языками программирования делает JSON удобным форматом передачи данных. При необходимости, JSON можно сжать такими кодеками, как Deflate или gzip, а также использовать использовать msgpack для его кодирования в бинарный формат или декодирования.

Пример единицы данных, полученной с умного жилета Hexoskin, которую клиент отправляет на сервер в формате JSON:

```
{  
    "Clitime": 2020-04-08 21:07:21,  
    "Millisec": 263,  
    "Steps":45,  
    "RespRate":2,  
    "Cadence":229,  
    "Insp":1914.375,  
    "Exp": 0.0,  
    "Activity":5.296875,  
    "HeartRate":70  
}
```

5.2 Аутентификация и авторизация

5.2.1 JSON Web Token

В основу разработанной схемы аутентификации и авторизации лег такой механизм, как JWT (JSON Web Token). JWT представляет собой JSON объект, который определен в открытом стандарте RFC 7519. Применение токена широко распространено для аутентификации и безопасного обмена информацией. Токен JWT состоит из трех ча-

стей: заголовка (header), полезной нагрузки (payload) и подписи (signature). Все три элемента кодируются с помощью base64 и соединяются точкой.

Заголовок токена состоит из двух полей: тип (на данный момент поддерживается только значение JWT) и алгоритм шифрования (например, HS256). В части полезной нагрузки содержатся заявки (claims) - это поля с информацией, которую необходимо передать, и дополнительными метаданными. Существует несколько типов заявок. Зарезервированные заявки (Reserved claims) являются необязательные заявками с предопределенными именами, например, iss (эмитент), exp (срок действия), sub (тема), aud (получатели). Публичные заявки (Public claims) могут быть определены по желанию пользователями JWT.

Для создания подписи необходимо взять закодированные заголовок и полезную нагрузку, секрет, хранимый только на сервере и с помощью алгоритма, указанного в заголовке, сформировать подпись. Пример создания подписи при помощи алгоритма HMAC SHA256:

```
HMACSHA256(  
    base64UrlEncode(header) + "." +  
    base64UrlEncode(payload),  
    secret)
```

JWT обладает такими преимуществами, как:

- Стандартизованность - обмен информацией выполняется при помощи широко распространенного формата JSON. Как следствие реализация существует для большого количества языков программирования, например, Python, Java, JavaScript, Node.js, PHP и так далее.
- Простота передачи и использования - поскольку при передаче JWT используется всего одна структура в формате JSON, его можно вставлять в заголовок HTTP при аутентификации или передавать при помощи URL. Для генерации или дешифрации токена достаточно использовать готовую библиотеку.
- Самодостаточность - вся информация, необходимая для обработки JWT содержится в нем самом. JWT включает в себя информацию для обработки, записанную в заголовке, пользовательскую информацию и подпись.

- Удобство - при каждом обращении к серверу нет необходимости передавать данные пользователя (например, логин и пароль). Пользователю достаточно аутентифицироваться один раз, чтобы получить доступ к сервису на определенное время. При этом не нужно хранить информацию о сеансе на сервере.

В статье [8] описан механизм аутентификации с помощью JWT при использовании протокола MQTT. Клиент MQTT отправляет брокеру токен в поле пароля сообщения подключения к брокеру. JWT, содержащийся в поле пароля, содержит утверждения, такие как время выдачи, дату истечения срока действия и любые другие утверждения, определенные службой, а также подпись. Подпись проверяется с помощью секрета, хранимого на сервере. При успешной аутентификации, клиент подключается к брокеру. Такой механизм аутентификации используется, например, на платформе Google Cloud IoT Core [9].

5.2.2 Схема аутентификации

Схема аутентификации выглядит следующим образом: клиент посылает запрос, содержащий идентификационные данные пользователя, на сервер, где данный запрос принимает сервис аутентификации. Далее полученные логин и пароль сверяются с данными, лежащими в базе данных аккаунтов, и в случае успешной аутентификации, генерируется токен и отправляется клиенту вместе с идентификационным номером пользователя. При всех последующих обращениях к серверу и подключениях к брокеру MQTT клиент обязан использовать этот токен. Если же аутентификация не прошла, то клиенту отправляется статус ошибки и сообщение. На рис. изображена предложенная схема аутентификации с последующим подключением к броекру MQTT.

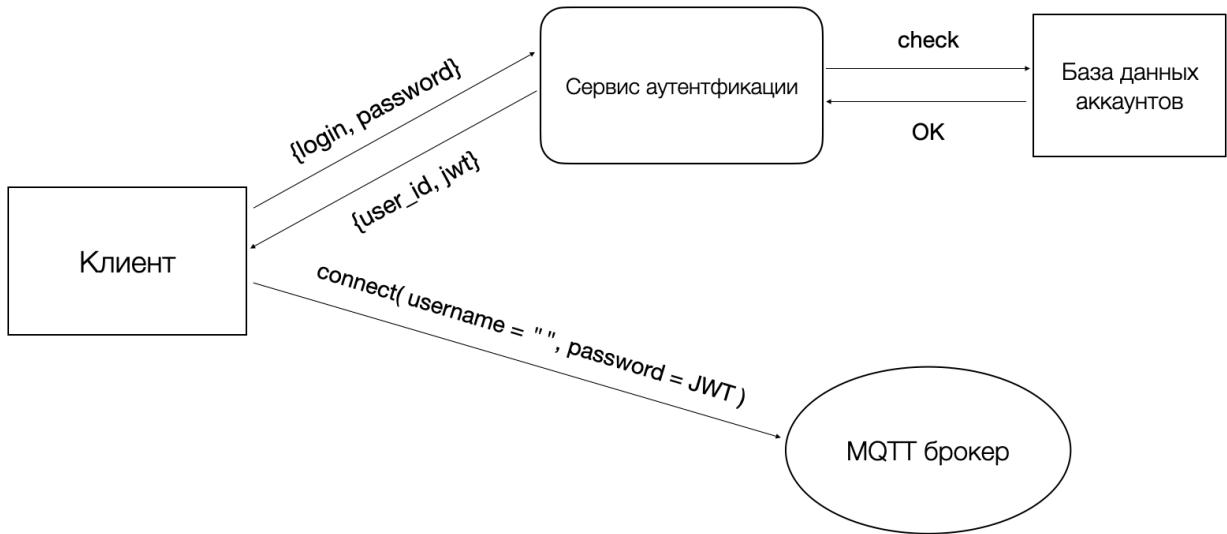


Рис. 5: Схема аутентификации с использованием технологии JWT

Для реализации предложенной схемы был взят плагин mosquitto-jwt-auth, который обеспечивает аутентификацию через JWT и авторизацию через ACL, хранящихся в заявках JWT. Полезная нагрузка используемых токенов имеет такие заявки, как тема токена, время создания токена и время окончания действия токена, а также указываются ACL, то есть списки контроля доступа, определяющие, на какие топики клиент может подписываться, и в какие публиковать сообщения. Ниже указан пример полезной нагрузки генерируемого токена:

```
{
    "sub": "mqttUser",
    "iat": 1516239022,
    "exp": 1616239022,
    "publ": ["/c/#"],
    "subs": ["/s/<user_id>/#"]
}
```

5.3 Хранение данных

5.3.1 Данные мониторинга

На сегодняшний день существует ряд прикладных областей, в которых широко используется такое понятие, как временной ряд. Известный пример - интернет вещей, а также ответвление данной концепции - интернет медицинских вещей. В подавляющем большинстве случаев данные, получаемые с датчиков умных устройств, являются временными рядами. Под временным рядом понимается совокупность значений какого-либо показателя, собранного в различные моменты времени. Например, данные, получаемые с акселерометра, или показатели ЭКГ можно рассматривать как временные ряды. Измерения, составляющие временной ряд, упорядочены по временной шкале, что показывает историю изменения данных во времени и может быть очень полезным для медицинской диагностики.

Возможные ограничения, связанные со скоростью поступления новых данных, большими объемами, а также особенностями обработки временных рядов приводят к необходимости использовать специализированные системы для хранения временных рядов. В курсовой работе автора за 3 курс [10] были выполнены обзор и экспериментальное исследование решений, оптимизированных под хранение и обработку временных рядов. В качестве СУБД для использования в реализации, подходящей под все критерии обзора, а также показавшей лучшую производительность, была выбрана СУБД ClickHouse.

ClickHouse [11] – это аналитическая колоночная база данных с открытым исходным кодом, разработанная компанией Яндекс для OLAP сценариев работы. Язык запросов ClickHouse представляет собой dialect SQL. Колоночные базы данных хранят записи в блоках, сгруппированных по столбцам, а не по строкам. Поскольку такая БД не загружает данные для столбцов, которых нет в запросе, она тратит меньше времени на чтение данных при выполнении запросов. Потому колоночные базы данных могут вычислять и возвращать результаты для определенных рабочих нагрузок, таких как OLAP, намного быстрее, чем традиционные строчные системы.

Каждая таблица в базе данных соответствует паре "пользователь-устройство". Столбцы согласованы с параметрами, получаемыми с датчиков соответствующего устройства, также присутствуют столбцы для меток времени. ClickHouse не поддерживает тип данных для времени с разрешением более одной секунды, поэтому

используется решение хранить время с разрешением до секунд в одном столбце типа DateTime, а миллисекунды в другом. Таблица создается сразу же при регистрации пользователем нового устройства и ее имя соответствует шаблону {id пользователя}_{id устройства}. Ниже показан пример создания таблицы ClickHouse, где носимым устройством выступает жилет Hexoskin:

```
CREATE TABLE {user_id}_{device_id} (
    Clitime DateTime,
    Millisec UInt16,
    HeartRate Nullable(UInt16),
    RespRate Nullable(UInt16),
    Insp Nullable(Float64),
    Exp Nullable(Float64),
    Steps Nullable(UInt32),
    Activity Nullable(Float64),
    Cadence Nullable(UInt32)
)
ENGINE = MergeTree()
PARTITION BY toYYYYMM(Clitime)
ORDER BY (Clitime, Millisec)
```

Для работы с ClickHouse на сервере используется клинетская библиотека clickhouse-driver.

5.3.2 Аккаунты пользователей и информация об устройствах

Для реализации хранилища аккаунтов и информации об устройствах была взята NoSQL СУБД MongoDB [12]. MongoDB — документоориентированная система управления базами данных с открытым исходным кодом, не требующая описания схемы таблиц. Для хранения данных в MongoDB применяется формат, который называется BSON (binary JSON). Данный формат позволяет ускорить выполнение поиска и обработки данных, но в целом данные в JSON-формате занимают меньше места, чем в формате BSON.

Аналогом строк и таблиц реляционных баз данных в MongoDB являются документы и коллекции. Базы данных состоят из коллекций, коллекция представляет собой группу документов. В отличие от строк документы могут хранить сложную по структуре информацию. Документ можно представить как хранилище ключей и значений. Коллекции могут содержать различные объекты, имеющие различную структуру и различный набор свойств.

База данных содержит пять коллекций. Ниже приводится их описание.

1. Accounts: коллекция с аккаунтами пользователей. Документ коллекции содержит уникальный идентификатор, логин и хешированный пароль пользователя.

```
accounts = {  
    user_id: String,  
    login: String,  
    password: String  
}
```

2. Users: коллекция с личной информацией пользователей. Документ коллекции содержит уникальный идентификатор, имя, фамилию, отчество, дату рождения, номер телефон, почтовый адрес, вес и рост пользователя.

```
users = {  
    user_id : String,  
    name : String,  
    surname : String,  
    patronymic : String,  
    email : String,  
    birth_date : String,  
    phone : String,  
    weight : Float,  
    height : Int  
}
```

3. Operators: коллекция с аккаунтами операторов. Документ коллекции содержит логин и хешированный пароль оператора.

```
operators = {
    login: String,
    password: String
}
```

4. Devices: Коллекция с типами устройств, поддерживаемых на платформе. Документ коллекции содержит уникальный идентификатор типа устройства и список его датчиков.

```
devices = {
    device_type: String,
    sensors: Array
}
```

5. UserDevices: Коллекция с устройствами пользователей. Документ коллекции содержит уникальный идентификатор устройства, типа устройства, поользователя и имя, которое пользователь дал своему устройству.

```
devices = {
    user_id: String,
    device_id: String,
    device_name: String,
    device_type: String
}
```

5.4 Веб-сервис

Для реализации серверной части веб-сервиса был выбран микрофреймворк Flask на языке программирования Python [13]. Ядро данного веб-фреймфорка достаточно легковесно и имеет минимальное количество надстроек, но существует множество библиотек, с помощью которых ядро Flask можно с легкостью расширить и настроить под свои нужды. Flask имеет зависимость от двух сторонних библиотек: Jinja2 и Werkzeug WSGI. Jinja2 – это шаблонизатор на языке Python, служащий для динамической генерации html разметки. Werkzeug WSGI предоставляет инструментарий для стандартно-

го интерфейса взаимодействия между веб-приложением и сервером WSGI (Web Server Gateway Interface).

В качестве веб-сервера используется nginx [14], который позиционируется производителем как простой, быстрый и надёжный веб-сервер, не перегруженный функциями. Он широко используется как для небольших проектов, так и крупными компаниями для высоконагруженных сервисов. Применяется он в качестве обратного прокси-сервера в связке с Gunicorn. При обратном проксировании сервер, получающий запрос от клиента, не обрабатывает его полностью самостоятельно, а отправляет этот запрос для обработки другим серверам. Gunicorn [15] - это относительно быстрый и ресурсоемкий WSGI-сервер, созданный для использования в UNIX-системах. Он работает с большим количеством веб-фреймворков, в том числе с фреймворком Flask. Он выполняет такие функции, как запуск пула рабочих процессов, перевод запросов, поступающих от nginx, для совместимости с WSGI, перевод ответов WSGI веб-приложения в правильные ответы HTTP.

Для того, чтобы настроить nginx для использования в качестве обратного прокси-сервера, необходимо создать виртуальный хост. Виртуальный хост - это секция конфигурации веб-сервера, отвечающая за обслуживание определенного домена. Пример настройки виртуального хоста:

```
server {  
    listen 80;  
    server_name iomt.lvk.cs.msu.su iomt.lvk.cs.msu.ru;  
  
    location /grafana/ {  
        proxy_pass http://localhost:3000/;  
        rewrite ^/grafana/(.*) $1 break;  
        proxy_set_header Host $host;  
    }  
  
    location / {  
        include proxy_params;  
        proxy_pass  
            → http://unix:/home/anikevich.yu/iomt-project/web/iomt.sock;  
    }  
}
```

```
    }  
}  
}
```

Такая конфигурация позволяет проксировать запросы, в которых URL начинается на `/grafana/` на соответствующий веб-интерфейс сервиса Grafana, запущенный на порту 3000, а остальные запросы проксируются на Gunicorn.

5.4.1 Описание методов API

Взаимодействие мобильного приложения в роли клиента и веб-сервиса строится по принципу запрос-ответ. Запрос с необходимыми параметрами формируется на стороне клиента и передается методом POST в формате JSON или методом GET. Далее приводится описание реализованных методов API. В описываемых запросах используются значения только двух параметров: `token` (токен, выданный при аутентификации) и `id` (уникальный идентификатор пользователя). Данные в теле запроса и теле ответа передаются в формате JSON.

- **POST /users/register/**

Запрос добавит аккаунт нового пользователя в базу данных и отправит письмо на указанный электронный адрес с ссылкой для подтверждения регистрации. Тело запроса содержит информацию, указанную пользователем при регистрации.

- **GET /confirm_email/{user_id}/{token}**

Запрос будет сделан при переходе по ссылке в письме о подтверждении регистрации пользователя. Регистрация пользователя `user_id` будет завершена.

- **POST /auth/**

Запрос обращается к сервису аутентификации, который проверяет корректность введенных данных, и при успешной аутентификации возвращает токен. Тело запроса содержит логин и пароль пользователя.

- **GET /users/info/?token=<token>&id=<user_id>**

Запрос возвращает информацию о пользователе, необходимую для организации пользовательского интерфейса в мобильном приложении.

- **POST /users/info/?token=<token>&id=<user_id>**

Запрос фиксирует изменения в личных данных пользователя, которые были сделаны в мобильном приложении. Тело запроса содержит измененные данные.

- **GET /devices/types/?token=<token>&id=<user_id>**

Запрос возвращает список доступных для регистрации типов устройств, поддерживаемых платформой.

- **GET /devices/get/?token=<token>&id=<user_id>**

Запрос возвращает список устройств пользователя.

- **POST /devices/register/?token=<token>&id=<user_id>**

Запрос регистрирует новое устройство пользователя, делая соответствующую запись в базу данных. Тело запроса содержит информацию о новом устройстве, такую как уникальный идентификатор, тип и имя, указанное при регистрации.

Для реализации взаимодействия оператора с сервисом был создан веб-сайт со структурой, которая изображена на рис. 6. В приложении А приведены скриншоты веб-страниц данного сайта.

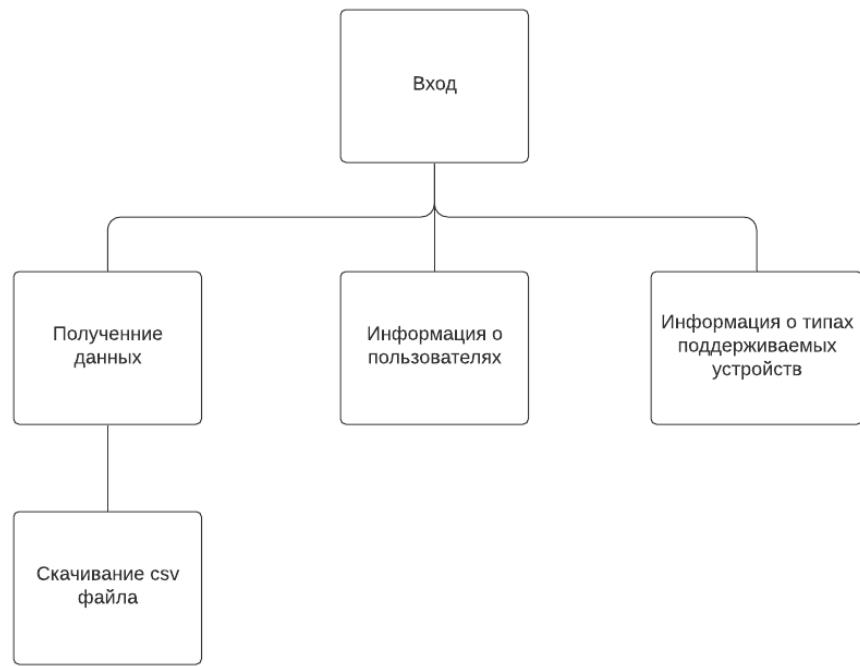


Рис. 6: Структура веб-сайта

6 Апробация платформы

Для апробирования разработанного решения был проведен эксперимент, суть которого заключалась в проведении стандартного сценария использования платформы. Пользователь заводит аккаунт и регистрирует свое умное устройство в мобильном приложении на Android, после чего он может подключиться к зарегистрированному устройству и начать отправку данных, полученных с датчиков устройства. В данном эксперименте в качестве носимого устройства был использован умный жилет Hexoskin, датчики которого способны считывать такие показатели, как интенсивность активности, частота сердечных сокращений, частота дыхания, объем вдыхаемого и выдыхаемого воздуха, ритм, и число шагов. Эксперимент по отправке и сбору данных состоял из трех этапов, которые соответствовали разным уровням активности: покой, ходьба и бег. Полученные данные были визуализированы при помощи сервиса Grafana. Ниже представлены графики таких показателей, как ритм, частота сердечных сокращений и частота дыхания, на каждом из которых отмечен соответствующий уровень активности. Остальные показатели представлены в Приложении Б.

Частота сердечных сокращений (рис. 7) в состоянии покоя лежит в пределах нормы 60-90 уд/мин. При ходьбе этот показатель становится на 10-20 ударов чаще, а при беге достигает своего пикового значения - около 180 уд/мин.



Рис. 7: Частота сердечных сокращений

Такой показатель, как ритм (рис. 8), представляет собой количество шагов в минуту. Идеальный ритм при беге составляет от 175 до 185. По результатам эксперимента эта величина колеблется от 100 до 120 во время ходьбы и 160 до 180 во время бега.



Рис. 8: Ритм

Частота дыхания (рис. 9) представляет собой скорость, с которой выполняется один полный цикл дыхания (вдох и выдох). Обычно в состоянии покоя частота дыхания колеблется от 12 до 16 об/мин и, как видно из графика, данная величина лежит в этом диапазоне в состоянии покоя. Во время упражнений высокой интенсивности эта величина может достигать до 70 об/мин, в данном эксперименте максимальное значение частоты дыхания - 40 об/мин.

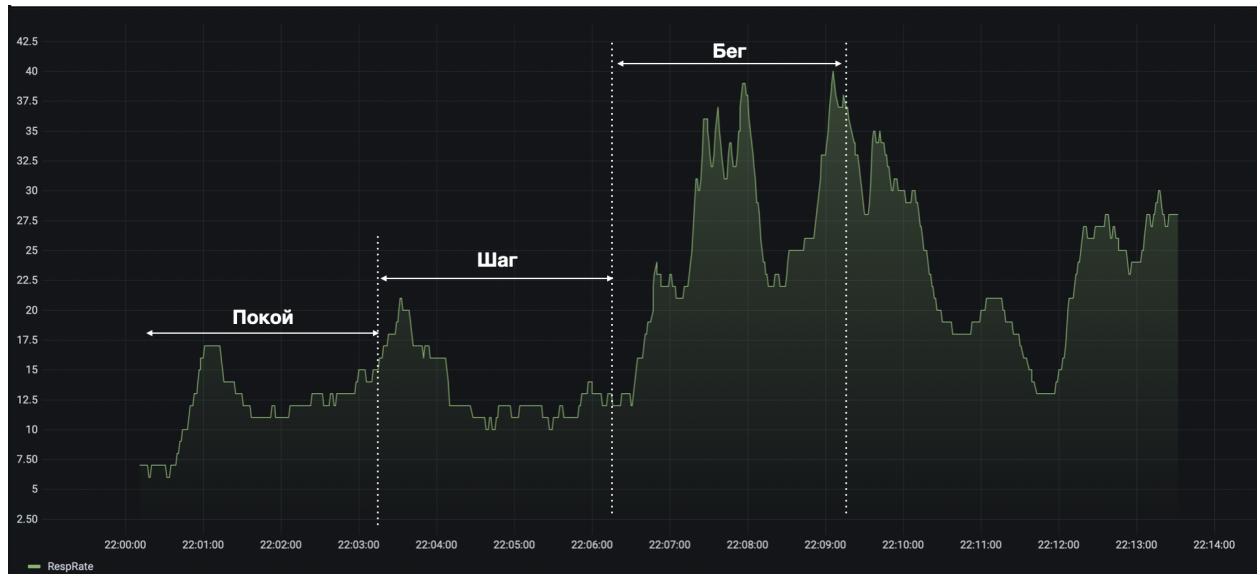


Рис. 9: Частота дыхания

7 Заключение

Таким образом, в ходе результате работы была разработана и реализована серверная часть открытой платформы для сбора и обработки физиологических данных о человеке.

В ходе работы был разработан и реализован механизм аутентификации и авторизации пользователей, спроектировано и реализовано веб-приложение, предоставляющее API для мобильного приложения и интерфейс оператора для экспорта данных и получения информации о пользователях, добавлена поддержка устройств нескольких типов, выполнено развертывание сервиса на сервере лаборатории вычислительных комплексов, а также произведена апробация платформы.

Список литературы

- [1] Validic. — <https://validic.com>.
- [2] Seqster. — <https://seqster.com>.
- [3] CareSimple. — <https://caresimple.com>.
- [4] 12 Open Source Internet of Things (IoT) Platforms and Tools. — <https://geekflare.com/iot-platform-tools/>, 2020.
- [5] Hexoskin. — <https://www.hexoskin.com>.
- [6] MQTT Documentation v3.1.1. — <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf>.
- [7] Eclipse Mosquitto. — <https://mosquitto.org>.
- [8] *Shingala, Krishna.* JSON Web Token (JWT) based client authentication in Message Queuing Telemetry Transport (MQTT) / Krishna Shingala. — 2019.
- [9] Cloud IoT Core,
<https://cloud.google.com/iot/docs/how-tos/credentials/jwts>. — Using JSON Web Tokens (JWTs).
- [10] *Аникиевич, Юлия.* — Разработка и реализация серверной части платформы для сбора и обработки физиологических данных о человеке. — https://drive.google.com/file/d/1L4YhNNT0rhu27o775aA2RqR_1dbgoB6v/view?usp=sharing, 2020.
- [11] ClickHouse Documentation. — <https://clickhouse.tech/docs/ru/>.
- [12] MongoDB Documentation. — <https://docs.mongodb.com>.
- [13] Flask Documentation (2.0.x). — <https://flask.palletsprojects.com/en/2.0.x/>.
- [14] nginx documentation. — <https://nginx.org/en/docs/>.
- [15] Gunicorn. — <https://gunicorn.org>.

Приложение А. Скриншоты интерфейса оператора



Рис. 10: Скриншот веб-страницы входа на веб-сайт

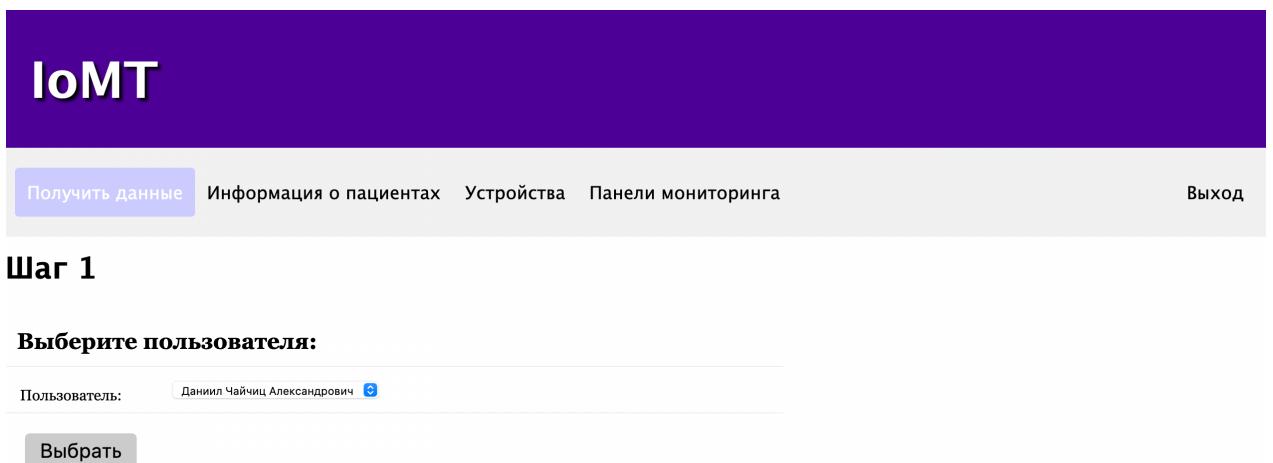


Рис. 11: Скриншот веб-страницы выбора пользователя

Приложение Б. Графики физических показателей

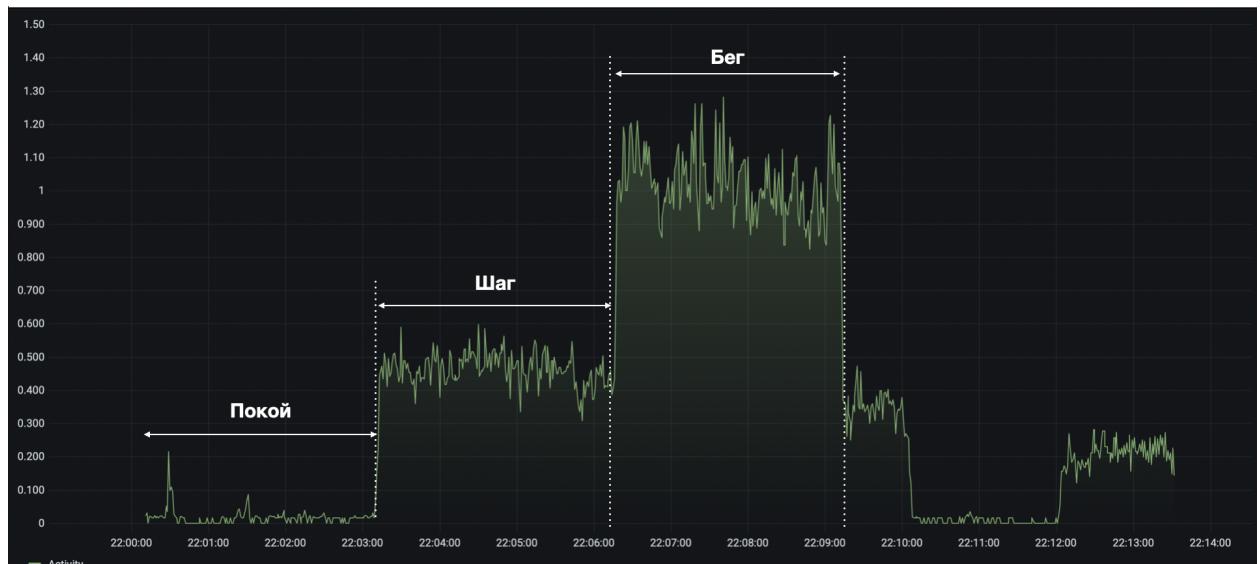


Рис. 12: Интенсивность активности

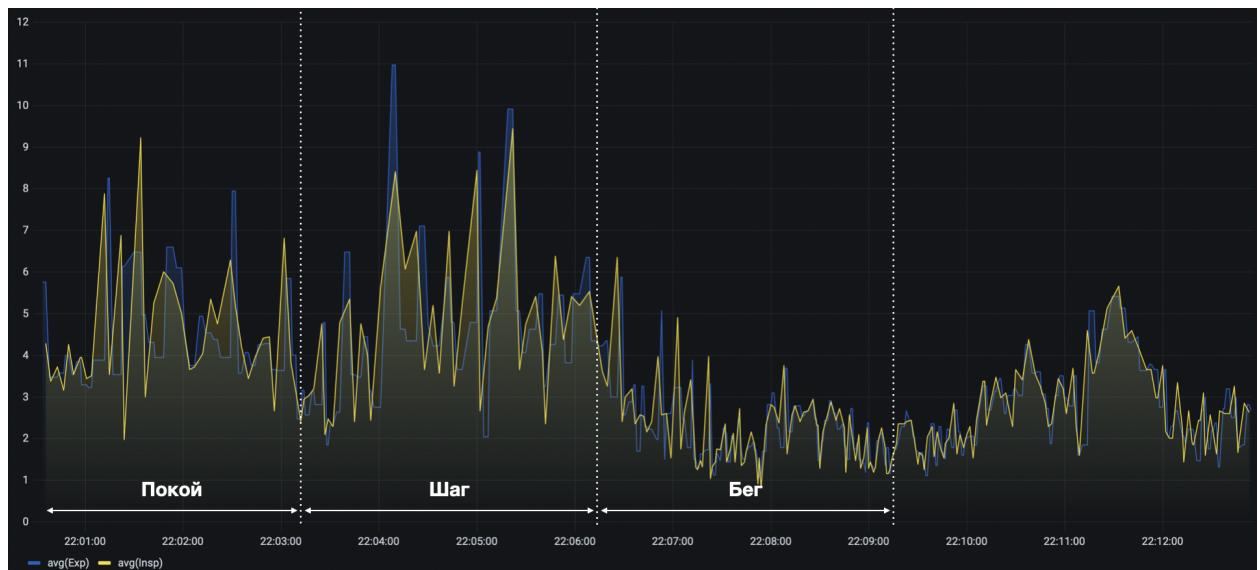


Рис. 13: Объем вдыхаемого и выдыхаемого воздуха



Рис. 14: Число шагов