Санкт-Петербургский государственный университет

Кафедра системного программирования
Программная инженерия

Бушев Вячеслав Валериевич

Разработка сервера для сбора и анализа биологических маркеров человека

Курсовая работа

Научный руководитель:
ст. преп. Кириленко Я. А.

Консультант:
к. ф.-м. н. Березун Д. А.

Санкт-Петербург
2019

Оглавление

Введение

Одним из самых главных приоритетов в жизни людей всегда являлось поддержание и укрепление здоровья. Для этого ученые, университеты и даже частные компании[1] постоянно проводят различные исследования, а также испытания с целью измерения и отслеживания всевозможных биологических маркеров человека (например, пульса, объема легких, либо же способности человека удерживать равновесие). Исследования человеческого организма как правило проводятся в специальных условиях. Снятие некоторых биологических маркеров[2] является достаточно долгой и сложной процедурой [1]. Для проведения измерений может потребоваться специальное помещение и дорогостоящее оборудование. Кроме того, нужны грамотные специалисты, которые будут следить за процессом измерения и поддерживать необходимые условия для экспериментов, и добровольцы, которые будут выполнять заданные исследователями указания.

Альтернативный подход для измерения биологических маркеров — использование тех инструментов, которые доступны практически каждому, например, мобильных телефонов и фитнес-браслетов. Их датчики, безусловно, не имеют высокой точности лабораторного оборудования, но иногда являются удовлетворительными. На данный момент уже существует огромное множество различных приложений, выполняющих измерения тех или иных биологических маркеров (их даже можно найти в AppStore[3] и GooglePlay[4]).

В СПбГУ и СПбНИИФК[5] регулярно проводятся исследования в области физической культуры и спорта, и есть спрос на подобные при-

[1]Например, компания Nike, которая известна практически во всем мире, имеет свою команду исследователей [5]. Среди актуальных исследований — анализ формы и возможностей сгибов стопы, а также измерение её давления на поверхность [6].

[2]Биологический маркер (биомаркер) — характеристика, которая используется для индикации состояния организма.

[3]Instant Heart Rate: HR Monitor: `https://itunes.apple.com/us/app/instant-heart-rate-hr-monitor/id409625068` (дата обращения: 08.05.2019)

[4]Accurate Heart Rate Monitor: `https://play.google.com/store/apps/details?id=com.repsi.heartrate` (дата обращения: 08.05.2019)

[5]ФГБУ СПбНИИФК. Санкт-Петербургский научно-исследовательский институт физической культуры: http://www.spbniifk.ru/ (дата обращения 10.05.2019)

ложения, причем для проведения испытаний необходимы разработки, зачастую не рассчитанные для широкой аудитории. Например, в исследовательских целях иногда требуется проводить замеры частоты сердечных сокращений сразу после выполнения специальных упражнений. Для этого могут не подойти многие из существующих алгоритмов расчета пульса, так как они делают предположение о том, что в процессе измерения пульс мало изменяется, поэтому в качестве ответа выбирается некоторое среднее значение, которое не является верным [4]. Кроме того, существует потребность в агрегировании данных, поскольку они часто собираются с использованием различных слабосвязанных друг с другом систем [2].

В рамках работы с СПбНИИФК планируется разработка целого ряда клиентских приложений для проведения разнообразных экспериментов в исследовательских целях. Поскольку у каждого приложения есть необходимость в том, чтобы хранить и обрабатывать биологические маркеры, актуальной является задача создания централизованной системы для их сбора с различных клиентских приложений с возможностью сохранения полученных данных и применения к ним заранее заданных в системе алгоритмов для проведения полноценных исследований.

1. Цели и задачи

1.1. Цель работы

Цель курсовой работы — разработка сервера для сбора и анализа биологических маркеров человека. Полученное решение должно иметь возможность легкой интеграции с новыми приложениями, работающими со своими специфическими данными, то есть быть легко *расширяемым*.

1.2. Поставленные задачи

Для выполнения обозначенной цели были выделены следующие задачи:

- Исследовать предметную область: провести обзор существующих методов обработки и хранения данных, а также изучение языков, технологий и средств для создания и развертывания современных клиент-серверных приложений.
- Разработать расширяемый сервер для обслуживания приложений, осуществляющих сбор биологических маркеров человека (далее соответствующую такому приложению серверную часть будем называть модулем[6]).
- Реализовать тестовый модуль для работы с приложением, осуществляющим сбор данных для измерения частоты сердечных сокращений. Реализовать алгоритм вычисления пульса.

[6]Тогда расширяемость подразумевает легкость добавления новых модулей к уже работающему с другими модулями серверу.

2. Возможные решения

В данной главе рассмотрены возможные варианты достижения поставленной цели, и обоснован выбор разработки собственного сервера.

2.1. Удаленная база данных

В рамках курсовой работы поставлена задача создания сервера для сбора биологических маркеров. Одним из альтернативных решений является использование удаленной базы данных. Например, известным продуктом в этой области является Google Firebase[7]. Это решение предоставляет нереляционную базу данных для хранения и считывания произвольной информации.

Использование одной лишь облачной базы данных имеет ряд недостатков. Алгоритмы обработки биологических маркеров при таком подходе придется хранить на клиентском приложении, а это также значит, что их содержимое будет общедоступно. *Декомпиляторы* (программы, транслирующие исполняемый файл в высокоуровневый исходный код) иногда способны получить достаточную часть исходного кода для восстановления общей логики работы алгоритмов. Некоторые из них могут быть не защищены патентом, поэтому давать к ним доступ было бы нелогичным и рискованным решением.

Другая проблема связана с тем, что произведение всех расчетов на клиенте означает, что если в алгоритме обработки данных нашлась серьезная ошибка, нужно будет в срочном порядке обновлять все клиентские приложения для её полного исправления. Если клиентов много, это может быть проблематично, поскольку пользователи не всегда придают должное внимание обновлениям, в связи с чем редко устанавливают их самостоятельно. При этом, даже уведомления о срочной необходимости установки новой версии зачастую игнорируются [7].

[7] `https://firebase.google.com/` (дата обращения: 10.05.2019)

2.2. Облачные вычисления

Хорошим дополнением к удаленной базе данных могли бы служить облачные вычисления. Тогда алгоритмы обработки биомаркеров не приходилось бы хранить на клиентском приложении, и проблема обновления клиентов при обнаружении критических ошибок стала бы неактуальной. Разработки, использующие облачные вычисления, ещё называют *бессерверными*, поскольку все операции производятся на серверах компаний, предоставляющих своим клиентам вычислительные мощности. Стратегия работы с облаком выглядит следующим образом: данные собираются с устройства и отправляются на заданный URL, после этого вызываются известные заранее пользовательские функции, которые производят обработку данных и, если необходимо, сохраняют какой-то результат в облачное хранилище. Такой подход избавляет от необходимости развертывания сервера и поддержки его работоспособности, позволяет легко масштабировать проект при необходимости и платить только за используемые ресурсы. Среди известных провайдеров облачных вычислений числятся крупные компании с высоким коэффициентом доверия: Amazon (AWS Lambda[8]), Google (Google Cloud Platform[9]), Microsoft (Azure[10]), IBM (IBM Cloud[11]) и другие.

Несмотря на очевидные плюсы, облачные вычисления также имеют существенные недостатки, из-за которых приходится отказаться от них в рамках курсовой работы. В проекте планируется собирать информацию[12], которая в некоторых странах попадает под закон о защите персональных данных. В этом случае, хранение и обработка данных в облаке на серверах другой страны может быть недопустима. Кроме того, незапатентованные алгоритмы работы с биомаркерами по-прежнему будут располагаться у третьих лиц.

[8]AWS Lambda: https://aws.amazon.com/ru/lambda/

[9]Google Cloud Platform: https://cloud.google.com/products/

[10]Microsoft Azure: https://azure.microsoft.com/ru-ru/

[11]IBM Cloud: https://www.ibm.com/cloud/

[12]Например, согласно федеральному закону Российской федерации от 27.07.2006 N 152-ФЗ "О персональных данных" информация, относящаяся к состоянию здоровья человека, а также сведения, которые характеризуют его физиологические и биологические особенности, на основании которых можно установить его личность, попадают под определение "*персональных данных*" и для их сбора, хранения и обработки требуют выполнения специальных, описанных в законе условий.

Облачные вычисления иногда усложняют разработку, поскольку их может быть сложно отлаживать и переносить (если речь идет о переходе с одного сервиса на другой, это может потребовать переписывания большей части существующей функциональности). Полноценное локальное тестирование облачных приложений невозможно из-за наличия жестких ограничений на серверах провайдеров (по памяти или времени выполнения). Эти же ограничения создают такую ситуацию, в которой результаты вычислений могут потеряться из-за превышения тех или иных лимитов.

2.3. Сервер

При использовании своего сервера, любые алгоритмы можно будет обновлять в одностороннем порядке, поэтому проблема с обновлением клиентов неактуальна. Также правильно выбранный стек технологий решает проблему переносимости. Но самое главное, что наличие своего сервера исключает любые вопросы, связанные с общедоступностью исходного кода или разглашением персональных данных третьим лицам.

Таким образом, по совокупности преимуществ последнего подхода было решено создать свой сервер для работы с приложениями, собирающими биологические маркеры.

3. Стек технологий

Сервер написан на языке Python (интерпретируемый, объектно-ориентированный язык высокого уровня), который был выбран в силу своей простоты, а также благодаря наличию множества готовых веб-фреймворков и библиотек, облегчающих разработку. Используемая база данных — Postgres — является объектно-реляционной и умеет работать с пользовательскими типами и сложными структурами. База данных в курсовой работе выполняет роль *микросервиса*. Она слабо связана с самим сервером, поскольку обновляется и поддерживается независимо от него. Для использования базы данных в таком контексте хорошо подходит Docker [3]. Решение поместить её в изолированный Docker-контейнер также удобно тем, что это позволяет один раз настроить конфигурационный файл, а затем использовать базу данных на различных устройствах с различными операционными системами без необходимости установки дополнительных пакетов и их настройки. Эта особенность имеет важную роль, поскольку работа над проектом велась с разных операционных систем. Для сервера также были использованы возможности веб-фреймворка Flask, который, по сравнению со своим известным аналогом Django, является более низкоуровневым и предоставляет больше возможностей для настройки проекта под свои нужды. Для работы с базой данных используется библиотека SQLAlchemy. Она позволяет единым образом описывать запросы к разным базам данных, а также предоставляет более удобный интерфейс в сравнении с использованием в коде чистого языка SQL, благодаря чему запросы к базе легче отслеживать и изменять.

4. Архитектура сервера

В этой главе представлено описание разработанного сервера, процесса добавления и принципов работы его модулей, а также взаимодействия сервера и модулей с базой данных.

4.1. Модульность

Основная цель при проектировании сервера заключалась в том, чтобы позволить легко встраивать новые модули с различной функциональностью. Для её достижения в корне сервера была отдельно создана специальная директория. Предполагается, что каждый новый модуль будет создаваться в ней. Любой новый модуль должен в одном из файлов содержать класс (*менеджер*), который наследует *BaseAppManager*, и, возможно, переопределяет некоторые его поля, среди которых ссылка на *обработчик запросов* и ссылка на класс, осуществляющий подготовку и фильтрацию данных для сохранения в базу данных (далее — *валидатор*). Любой обработчик запросов для модуля должен унаследовать *BaseRequestHandler*. Аналогично, валидатор наследуется от *BaseDataValidator*. Минимальный рабочий модуль может содержать только менеджер.

Рабочий процесс выстроен следующим образом:

- На выделенный для модуля URL приходит запрос в формате JSON (см. листинг 1).
- Данные проходят через обработчик запросов.
- Затем сохраняются согласно правилам, заданным в валидаторе.
- В качестве ответа, сервер возвращает вычисленный результат от обработчика.

Иногда приложения могут не предполагать сохранения данных, либо исключать их обработку. Для обозначения таких специальных случаев внутри запроса приходят опции (*options*). Булевый параметр *calc*

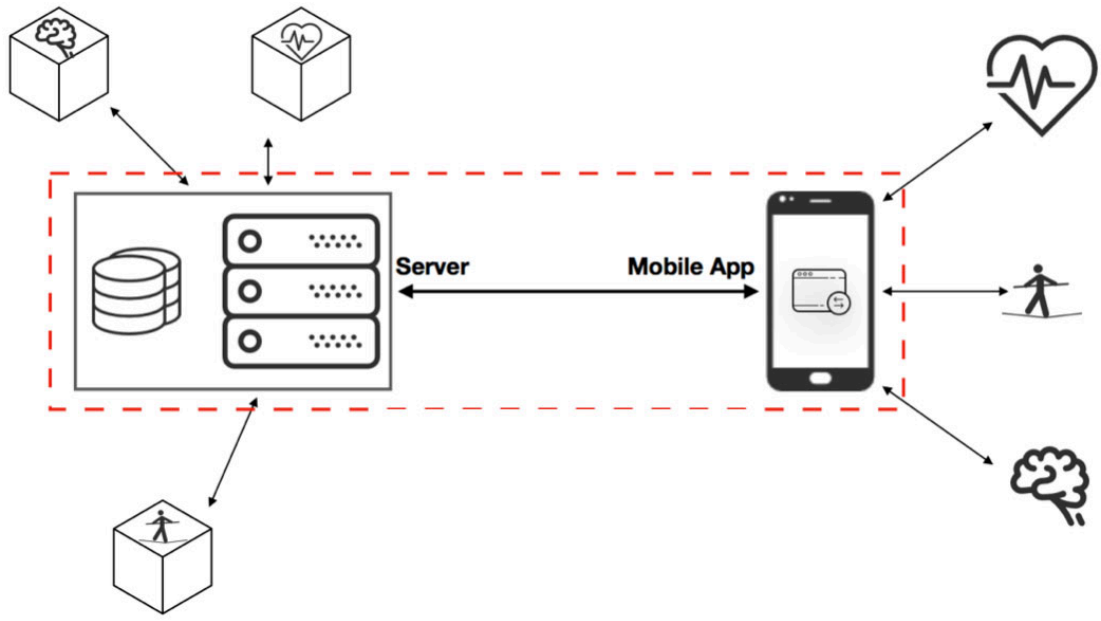

Рис. 1: Концепция взаимодействия сервера и мобильного клиента

отвечает за передачу входящих данных в обработчик, а *store* регулирует вызов валидатора. По умолчанию обе опции включены.

На рисунке 1 изображена концепция взаимодействия приложений, собирающих биологические маркеры, с сервером, который обрабатывает запросы и сохраняет вычисленные данные в базу. Стоит отметить, что обработчики запросов каждого из модулей могут перенаправлять информацию на другой удаленный ресурс, где будут производиться иные вычисления, используемые для построения своего собственного рабочего процесса в рамках конкретного модуля.

4.2. Работа с базой данных

С самого начала работы над курсовой предполагалось, что архитектура сервера и структура базы данных могут меняться в процессе разработки. Так появилась потребность в наличии на сервере механизма *миграций* (в данном контексте — обновления структуры) базы данных, который, если это необходимо, вызывался бы автоматически при каждом запуске сервера. Чтобы осуществить поддержку автомати-

ческих миграций, были изучены различные стандартные библиотеки, работающие вместе с SQLAlchemy. Одним из самых популярных решений в этой области является библиотека от авторов SQLAlchemy, которая называется Alembic[1]. Стандартный рабочий процесс использования Alembic подразумевает, что пользователь будет взаимодействовать с ней через *CLI* (интерфейс командной строки). Обновления базы данных время от времени придется производить вручную, выполняя накопленные миграционные скрипты. Иными словами, при администрировании сервера необходимо будет совершать дополнительные действия. Так было принято решение автоматизировать процесс миграций.

Кроме того, что Alembic опирается на CLI, он имеет огромную внутреннюю реализацию, малая часть из которой документирована. Поэтому сама библиотека практически не использовалась, вместо этого была написана своя облегченная реализация для работы с миграционными скриптами, которая, тем не менее, удовлетворяет всем потребностям сервера. При каждом его запуске происходит извлечение идентификатора текущей миграции из версионной таблицы; если обнаруживается, что существуют дальнешие миграции, то они выполняются.

Модули могут хранить результаты своих вычислений, однако они взаимодействуют с базой данной не напрямую, а через валидатор, который может только выбирать, что требуется сохранить, но не каким образом. Под каждый модуль в базе выделена отдельная таблица, которая создается автоматически и поддерживается без участия разработчиков. Структура таблиц унифицирована и может быть изменена только с помощью миграционных скриптов, относящихся не к модулям по отдельности, а к серверу в целом. Всё это в совокупности обеспечивает защиту от ошибок разработчиков, а также создает дополнительный уровень изоляции модулей от сервера и друг от друга.

Таким образом, разработанный сервер полностью удовлетворяет заявленным функциональным требованиям.

[1] Alembic 1.0.10: `https://pypi.org/project/alembic` (дата обращения: 08.05.2019)

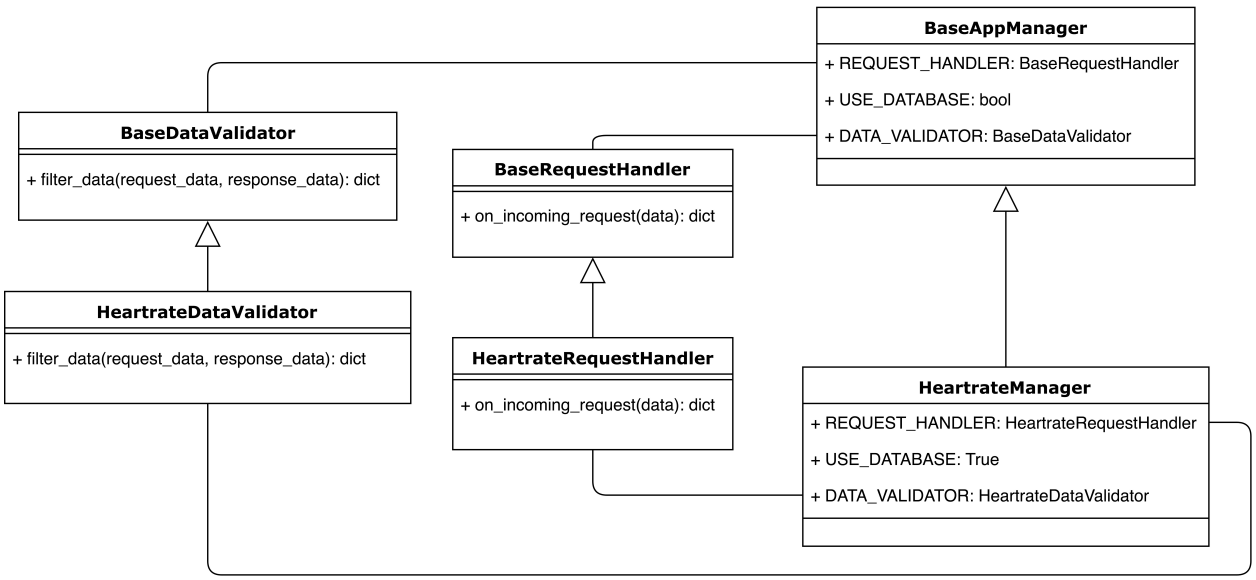

Рис. 2: Диаграмма классов модуля расчета ЧСС

5. Модуль измерения частоты сердечных сокращений

Рассмотрим тестовый модуль для интеграции сервера с приложением, которое собирает данные о притоках крови к пальцу для расчёта пульса с помощью камеры мобильного телефона.

5.1. Реализация модуля

В качестве примера серверного модуля был разработан модуль для измерения частоты сердечных сокращений (далее — *ЧСС*). Диаграмма классов данного модуля изображена на рисунке 2. Все вычисления происходит в рамках обработчика запросов – *HeartrateRequestHandler*. Поддерживается два типа измерения ЧСС: в состоянии покоя и в состоянии релаксации. Под состоянием *релаксации* понимается состояние восстановления обычной функциональности человеческого организма после выполнения физических упражнений.

Все существующие алгоритмы расчета ЧСС как правило состоят из двух этапов. Первый — преобразование кадров с камеры в какую-либо числовую характеристику. Второй — вычисление пульса в зависимости от изменения данной характеристики со временем [8]. Реализованный

на сервере алгоритм работает аналогично.

Используя клиентское приложение под ANDROID[13], испытуемый в течение некоторого времени при включенном фонарике снимает данные с камеры мобильного телефона, приложив к ней указательный палец. Затем они конвертируются из формата YUV в RGB представление, из которого извлекаются значения красного канала. С их помощью можно явно различить моменты притока крови к пальцу. Собранные данные отправляются на сервер (описание формата см. в листингах 2 и 3), где происходит вычисление пульса в соответствующем обработчике входящих запросов.

Задача алгоритма вычисления ЧСС в состоянии покоя состоит в том, чтобы рассчитать результат, имея частоту снятия кадров с камеры и сигнал красного канала RGB. Для определения доминирующей частоты сигнала применяется метод, использующий быстрое преобразование Фурье, как наиболее точный и устойчивый к помехам [8]. Кроме того, для улучшения точности измерения в состоянии покоя также используется метод пересекающихся отрезков, суть которого состоит в том, что исходные данные разбиваются на такие отрезки, у которых начало каждого следующего является серединой предыдущего. После разбиения доминирующая частота рассчитывается на каждом отрезке, а также на всех данных целиком. В качестве ответа принимается некоторое усредненное число с поправкой на доминирующую частоту.

В состоянии релаксации пульс вычисляется методом скользящего окна. В исходных данных дополнительно присутствует его ширина и параметр сдвига (см. листинг 3). Метод пересекающихся отрезков не применяется, поскольку результаты на отрезках будут отличаться, и их будет сложно интерпретировать из-за постоянного снижения пульса. Быстрое преобразование Фурье может давать ошибку, выбирая неверную частоту в качестве доминирующей, поэтому из всего измерения удаляются ЧСС, сильно отклоняющиеся от среднего значения. Имеющиеся данные затем приближаются экспоненциальной функцией[14], за-

[13]Приложение разработано в рамках связанной курсовой работы.

[14]После проведения экспериментов такая функция выбрана как обеспечивающая наибольшую точ-

висящей от времени. Совокупность значений данной функции за время измерения является ответом.

После расчёта пульса в обработчике данные попадают в валидатор, который сохраняет как исходные данные, так и вычисленный результат.

Таким образом, при разработке тестового модуля удалось организовать расчет пульса и сохранение необходимых результатов, задействуя предназначенную для модулей функциональность.

5.2. Тестирование модуля

В рамках курсовой работы проводилось как тестирование реализованных алгоритмов расчета пульса, так и тестирование работы приложения с сервером. Кроме того, при разработке сервера было налажено непрерывное тестирование.

Проверка алгоритмов расчета ЧСС велась коллегами из СПБНИИФК. Мобильное приложение было установлено на устройстве Sony Xperia M4 Aqua Dual (E2333). В качестве контрольного прибора использовался профессиональный пульсометр Polar M600. Во время тестирования было выявлено, что в начале результаты измерений были удовлетворительными и слабо отклонялись от показаний контрольного прибора, но с течением времени показания телефона ухудшились. Было сделано предположение, что неточности могут быть связаны с дискомфортными ощущениями спортсменов из-за сильного нагревания устройства. Кроме того, в процессе проведения экспериментов некоторые измерения признавались некачественными (сообщалось о невозможности вычислить пульс). В состоянии покоя это значит, что значения на пересекающихся отрезках сильно отличаются. Происходит это, потому что в приложении заданы строгие критерии совпадения значений. Сложившуюся ситуацию можно трактовать по-разному. С одной стороны, чем больше совпадений на отрезках, тем выше вероятность корректности результата. С другой же, слишком строгие критерии могут привести к тому, что большинство измерений будет счи-

ность.

таться неверным. Таким образом, можно сделать вывод, что полученный алгоритм не является совершенным и нуждается в доработке. Для улучшения алгоритма планируется провести дальнейшие исследования описаной проблемы.

В конечном итоге, тестовый модуль смог обеспечить приложение по измерению ЧСС необходимой функциональностью расчёта пульса и сохранения вычисленных данных. Модуль может быть переиспользован, поскольку реализованные алгоритмы расчета ЧСС показали неплохие результаты в реальных условиях.

Заключение

В ходе работы были достугнинуты следующие результаты:

– Разработан модульный сервер для обслуживания приложений, измеряющих биологические маркеры.

– Внедрён тестовый модуль через который работает приложение по измерению пульса.

– Реализованы алгоритмы расчета частоты сердечных сокращений в состоянии покоя и релаксации.

Таким образом, поставленные задачи были полностью выполнены. В результате создан расширяемый сервер, который будет использован при проведении реальных исследований.[15]

[15]Исходный код доступен по ссылке: https://gitlab.com/SergeyGorbatyuk171/biometrichub (дата обращения: 12.05.2019)

Листинги

```
{
    ''data'': {
        ...
    },
    ''options'': {
        ''store'': bool,
        ''calc'': bool
    }
}
```

Листинг 1: Общий формат входящих запросов

```
{
    ''data'': {
        ''type'': ''peace'',
        ''measure'': {
            ''time'': int,
            ''redsums'': [int, int, ...]
        }
    }
}
```

Листинг 2: Формат запросов для измерения пульса в состоянии покоя с сохранением результата в базу данных

```
{
    ''data'': {
        ''type'': ''relax'',
        ''measure'': {
            ''time'': int,
            ''windowSize'': int,
            ''shift'': int,
            ''redsums'': [int, int, ...]
        }
    }
}
```

Листинг 3: Формат запросов для измерения пульса в состоянии релаксации с сохранением результата в базу данных

Список литературы

[1] Biomarker detection technologies and future directions / Satish Balasaheb Nimse, Mukesh Digambar Sonawane, Keum-Soo Song, Taisun Kim // Analyst. — 2016. — Vol. 141, no. 3. — P. 740–755.

[2] Connecting information to improve health / W Ed Hammond, Christopher Bailey, Philippe Boucher et al. // Health Affairs. — 2010. — Vol. 29, no. 2. — P. 284–288.

[3] Docker: Microservices. — URL: `https://www.docker.com/solutions/microservices`. Дата обращения: 03.05.2019.

[4] Improved heart rate detection using smart phone / Arpan Pal, Aishwarya Visvanathan, Anirban Dutta Choudhury, Aniruddha Sinha // Proceedings of the 29th Annual ACM Symposium on Applied Computing / ACM. — 2014. — P. 8–13. — Дата обращения: 10.05.2019.

[5] NIKE EXPLORE TEAM SPORT RESEARCH LAB. — URL: `https://about.nike.com/pages/nike-explore-team-sport-research-lab`. Дата обращения: 08.05.2019.

[6] Sport science: The 5 Physical Tests All Nike Athletes Take at the Nike Sport Research Lab. — URL: `https://news.nike.com/news/nike-sport-research-lab-athlete-tests`. Дата обращения: 10.05.2019.

[7] Vaniea Kami, Rashidi Yasmeen. Tales of software updates: The process of updating software // Proceedings of the 2016 chi conference on human factors in computing systems / ACM. — 2016. — P. 3215–3226.

[8] Ташкинов Михаил Ильич. Оценка пульса с помощью камеры мобильного телефона. — URL: `https://se.math.spbu.ru/SE/YearlyProjects/spring-2016/371/Tashkinov_report.pdf`. Дата обращения: 02.04.2019.