Оглавление

ВВЕДЕНИЕ

Согласно исследованиям [1], на момент 2019 года 28% сотрудников постоянно или достаточно часто чувствовали себя угнетёнными под давлением рабочих обязанностей, а 48% страдали синдромом эмоционального выгорания время от времени.

На сегодняшний день проблема эмоционального выгорания касается не только самих работников, но и компаний, которые при утрате контроля над ситуацией вынуждены увольнять сотрудников под предлогом неисполнения ими обязанностей. [2]

Причиной синдрома может быть и физическое, и эмоциональное истощение вследствие увеличения нагрузки на работе, а также количества возлагаемых обязанностей на сотрудника. [3]

Синдром хронической усталости также является одним из факторов, понижающим работоспособность сотрудников. Несмотря на то, что истинная этиология данного заболевания до конца не раскрыта, одним из возможных факторов появления данного синдрома приписывают высокой нагрузке как умственной, так и физической. [4]

Усталость негативно влияет на производительность труда, а также на психологическое и физическое состояние человека. В условиях современной цифровизации медицины становится реальным учитывать индивидуальные особенности организма, управление его работоспособностью и проведение профилактики проявления вышеописанных синдромов, приводящих к неутешительным последствиям.

Цель работы – спроектировать и реализовать серверное приложение для доступа к базе данных, предназначенной для хранения информации о действиях и характеристиках, необходимых для определения усталости пользователей автоматизированного рабочего места (АРМ).

Для достижения поставленной цели потребуется:

1) формализовать задачу;
2) определить требуемую функциональность;
3) проанализировать существующие решения;
4) описать протокол взаимодействия клиента и сервера;
5) разработать приложение для решения поставленной задачи.

1. Аналитический раздел

В данном разделе формализуется задача, приводится требуемая функциональность разрабатываемого приложения, проводится анализ существующих решений.

1.1 Формализация задачи

Необходимо реализовать клиент-серверное приложение для доступа к базе данных, предназначенной для хранения информации о действиях и характеристиках, необходимых для определения усталости пользователей АРМ. Данная потребность связана с тем, что при работе с InfluxDB на ЯП Kotlin на текущий момент отсутствуют библиотеки, отвечающие полноте функционала, который может понадобится при проводимых работах.

В качестве решения поставленной задачи поставщики СУБД предлагают разработчику реализовать собственное серверное приложение, которое будет обрабатывать запросы, используя обращения к API развёрнутой СУБД.

К возможностям, которые должен предоставлять сервер, отнесены:

- внесение данных;
- получение данных;
- проверка существования хранилища для пользователя;
- создание хранилищ для новых пользователей.

1.2 Существующие решения

1.2.1 The Kotlin InfluxDB 2.0 Client

The Kotlin InfluxDB 2.0 Client [5] - это клиент, который предоставляет возможность производить запросы и запись в InfluxDB 2.0 с использованием ЯП Kotlin. Данная библиотека поддерживает асинхронные запросы с использованием Kotlin Coroutines.

На данный момент решение поддерживает следующий функционал:

- запись в базу данных;
- чтение базы данных с использованием стандартного языка InfluxQL;
- чтение базы данных с использованием языка Flux.

Данным решением не поддерживается следующий требуемый функционал:

- проверка существования хранилища для пользователя;
- создание хранилищ для новых пользователей.

1.2.2 InfluxDB v2 API

InfluxDB поддерживает обращение к InfluxDB v2 API [6]. InfluxDB API предоставляет способ взаимодействия с базой данных с использованием HTTP-запросов и ответов, включающих в своё тело данные в формате JSON, HTTP аутентификации, а также с поддержкой токенов JWT и базовой аутентификации.

Предоставляемый данным интерфейсом функционал полон и непосредственно используется в реализации Web-клиента данной СУБД. К недостатку использования данного метода взаимодействия относятся формирование множественных HTTP-запросов и потребность в обработке ответов.

Вывод

Среди рассмотренных существующих решений отсутствуют примеры удобной реализации, отвечающей полноте функционала, которую можно было бы использовать при написании приложений на ЯП Kotlin.

1.3 Архитектура клиент-сервер

Программное обеспечение архитектуры ”клиент-сервер”состоит из двух частей: программного обеспечения сервера и программного обеспечения пользователя – клиента. Программа-клиент выполняется на компьютере пользователя и посылает запросы к программе-серверу, которая работает на компьютере общего доступа. Основная обработка данных производится мощным сервером, а на компьютер пользователя возвращаются только результаты выполнения запроса. В такой архитектуре сервер называется сервером баз данных. [7]

Иными словами, данная архитектура определяет общие принципы организации взаимодействия в сети, где имеются серверы, узлы-поставщики некоторых специфичных функций, и клиенты, потребители данных функций.

Каждое приложение, опирающееся на архитектуру ”клиент-сервер определяет собственные или использует имеющиеся правила взаимодействия между клиентами и сервером, которые называются протоколом обмена или протоколом взаимодействия. [8]

1.4 Шаблон проектирования Model-View-Controller

Шаблон проектирования MVC предполагает разделение данных приложения, пользовательского интерфейса и управляющей логики на три от-

дельных компонента: модель, представление и контроллер, что приводит к возможности модификации каждого из упомянутых компонентов независимо друг от друга. [9]

Model предоставляет данные предметной области представлению и реагирует на команды контроллера, изменяя свое состояние. В это премя View отвечает за отображение данных предментной области (модели) пользователю, реагируя на изменения модели. Controller интерпретирует действия пользователя, оповещая модель о необходимости изменений. [9]

Модель обладает следующими признаками [10]:

- содержание бизнес-логики приложения;
- отсутствие связей с контроллером и представлением;
- представляет собой слой данных, менеджер базы данных или набор объектов.

Представление обладает следующими признаками [10]:

- включает в себя реализацию отображения данных, которые получаются от модели любым способом;
- в некоторых случаях может включать в себя реализацию некоторой бизнес-логики.

Контроллер обладает следующими признаками [10]:

- объект определяет, какое представление должно быть отображено в данный момент;
- зависим от событий представления.

Таким образом, для проектируемого программного обеспечения можно определить, что при использовании паттерна проектирования MVC, в качестве модели будут использоваться компоненты доступа к данным. Сервер будет использовать контроллеры для получения необходимой клиенту информации, причём представление будет определяться пользователем удалённо, так как предоставляемый сервис не предусматривает пользовательского интерфейса для взаимодействия с конкретными функциями.

1.5 Протокол HTTP

Протокол HTTP реализует клиент-серверную технологию, которая предполагает наличие множества клиентов, инициирующих соединение и посылающих запрос, а также множества серверов, получающих запросы, выполняющих требуемые действия и возвращающих клиентам результат. [11]

В данном протоколе главным объектом обработки является ресурс, который в клиентском запросе записан в URI (Uniform Resource Identifier). В качестве ресурса выступают файлы, хранящиеся на сервере. HTTP позволяет определить в запросе и ответе способ представления ресурса по различным параметрам. [11]

К преимуществам протокола HTTP относят [11]:

- простоту;
- расширяемость;
- распространённость;
- документация на различных языках.

К недостаткам данного протокола относят [11]:

- отсутствие ”навигации”;
- отсутствие поддержки распределенных действий.

Каждое HTTP-сообщение состоит из трех частей [12]:

- стартовой строки (определяющей тип сообщения);
- заголовков (характеризующих тело сообщения, параметры передачи и т.д.);
- тела (данные сообщения).

Версия протокола 1.1 включает в себя требование наличия заголовка Host. Сами заголовки представляют собой строки, содержащие разделенную двоеточием пару параметра и значения. [12]

Стартовая строка запроса включает в себя [12]:

- метод (тип запроса, одно слово заглавными буквами);
- путь к запрашиваемому ресурсу;
- версию используемого протокола.

Метод указывает на основную операцию над ресурсом. Среди наиболее часто используемых методов можно выделить [12]:

- GET – запрос содержимого указанного ресурса;
- POST – передача пользовательских данных заданному ресурсу;
- PUT – загрузка содержимого запроса по указанному пути;
- PATCH – PUT, применимый к фрагменту ресурса;
- DELETE – удаление указанного ресурса.

В качестве ответа клиенту сервер также направляет код состояния, по которому тот узнает о результатах выполнения запроса. Выделяют 5 классов

состояний [12]:

- 1xx – информационный (информирование о процессе передачи);
- 2xx – успех (информирование о случаях успешного принятия и обработки запроса);
- 3xx – перенаправление (сообщение о том, что для успешного выполнения операции потребуется выполнить другой запрос);
- 4xx – ошибка клиента (указание на ошибки со стороны клиента);
- 5xx – ошибка сервера (информирование о неудачном выполнении операции на стороне сервера).

Вывод

В разделе была предоставлена формализация задачи: реализация клиент-серверного приложения для доступа к базе данных InfluxDB с предоставлением возможности внесения и получения данных, проверки существования хранилища для пользователя и создания хранилищ для новых пользователей. Была предоставлена информация о существующих решениях, которые могут использоваться при решении задачи. Сделаны выводы о том, что среди рассмотренных продуктов отсутствуют примеры удобной в использовании реализации, отвечающей полноте функционала. Также было предоставлено определение понятия ”клиент-серверного”приложения и приведена информация о наиболее часто используемом протоколе взаимодействия, используемого в подобных приложениях.

2. Конструкторский раздел

В данном разделе предоставлены диаграмма вариантов использования, блок-схема работы сервера и описание собственного протокола взаимодействия клиента и сервера, основанного на спецификации протокола HTTP.

2.1 Диаграмма вариантов использования

На рисунках 2.1–2.2 предоставлены диаграммы вариантов использования.

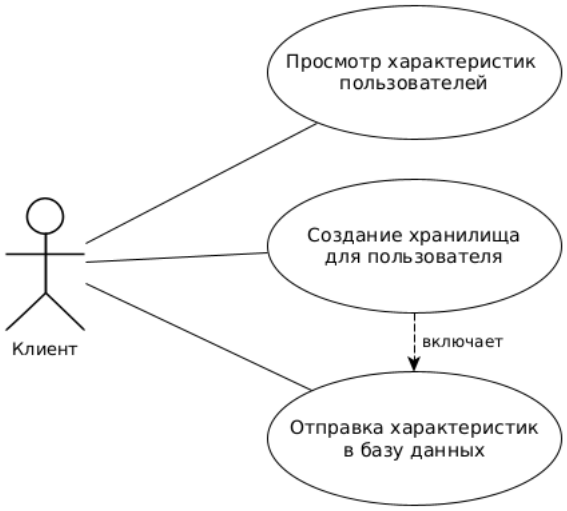

Рис. 2.1: Диаграмма вариантов использования для пользователя.

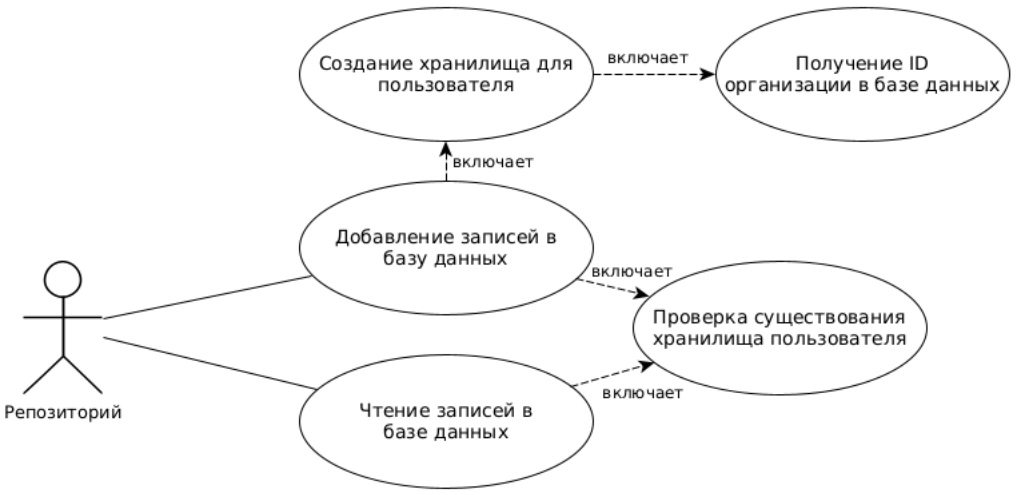

Рис. 2.2: Диаграмма вариантов использования для репозитория.

2.2 Блок-схема работы сервера

На рисунке 2.3 предоставлена блок-схема начала работы сервера и алгоритма обработки запросов пользователей.

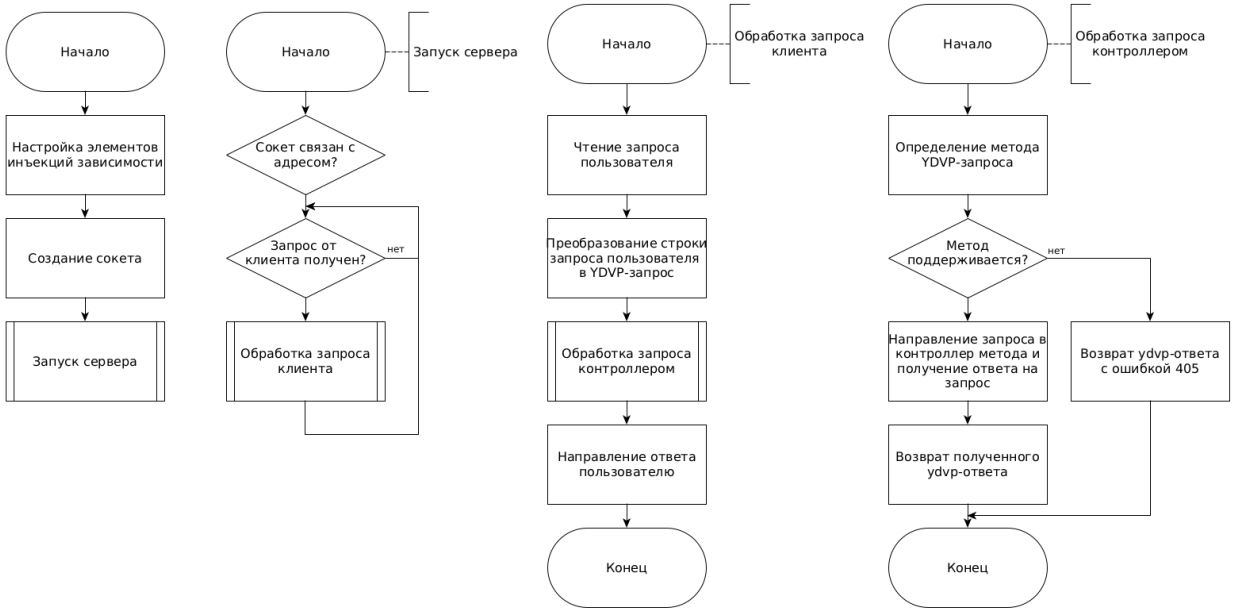

Рис. 2.3: Блок-схема работы сервера приложения.

2.3 Описание YDVP-протокола

На основе существующего стандарта HTTP в данном подразделе описывается структура реализуемого YDVP-протокола.

YDVP-запрос включает в себя три компонента:

- стартовую строку (определяющей тип сообщения);
- заголовки (характеризующих тело сообщения, параметры передачи и т.д.);
- тело (данные сообщения).

Стартовая строка запроса включает в себя:

- метод (тип запроса);
- путь к запрашиваемому ресурсу;
- версию используемого протокола.

Метод указывает на основную операцию над ресурсом. При этом в качестве используемых в версии 0.1 могут быть задействованы лишь методы GET и POST, отвечающие требованиям, определенным в формализации задачи. В последующих версиях список доступных методов может быть расширен.

В качестве ответа клиенту сервер направляет код состояния, который определяется по стандарту HTTP.

Вывод

В разделе была предоставлена диаграмма вариантов использования, позволившая выделить две основные операции, доступные клиенту. Также была предоставлена схема работы сервера, которая позволила выделить основные компоненты, действующие в системе. Была описана версия 0.1 собственного протокола для взаимодействия клиента и сервера.

3. Технологический раздел

В данном разделе предоставлено обоснование используемых языка программирования и среды разработки. Также приведены сведения о модулях и диаграмма классов приложения.

3.1 Выбор и обоснование языка программирования и среды разработки

При написании программного продукта был использован язык программирования Kotlin [13].

Данный выбор обусловлен следующими факторами:

- задача подразумевает под собой разработку способа взаимодействия с базой данной InfluxDB с расширением функционала библиотеки The Kotlin InfluxDB 2.0 Client,
- возможность запуска программного кода на любом устройстве, поддерживающем Java,
- большое количество литературы, связанной с ЯП Java.

При разработке использовалась среда IntelliJ IDEA. Данный выбор обусловлен тем, что Kotlin является продуктом компании JetBrains, поставляющей данную среду.

3.2 Сведения о модулях

Приложение логически разделено на следующие части:

- модуль доступа к данным,
- модуль бизнес-логики,
- модуль реализации протокола,
- модуль клиента,
- модуль сервера.

Модуль доступа к данным включает в себя два класса, основанных на паттернах репозиторий (CharRepositoryImpl) и объекта доступа к данным (InfluxDAO). В данной реализации объект доступа к данным позволяет сделать репозиторий независимым от реализации исполнения запросов к базе данных. Данный объект использует InfluxDB-client-kotlin для запросов на внесение и чтение записей в базу данных, однако отдельный функционал (например, создание пользовательского хранилища) производится через HTTP-запросы напрямую к серверу InfluxDB с использование OkHttp3.

Модуль бизнес-логики включает в себя множество сущностей, фигурирующих между слоями клиент-серверной архитектуры.

Модуль реализации протокола включает в себя класс YDVP, который может представлять собой YDVP-запрос или YDVP-ответ, единственным различием для них будет интерпретация абстрактного класса StartingLine, от которого наследуются классы YdvpStartingLineRequest и YdvpStartingLineResponse. Данная часть также содержит класс YdvpParser, который предоставляет услуги парсинга приходящих YDVP-запросов.

Модуль клиента включает в себя класс InfluxServiceClient, который позволяет подключиться к удалённому серверу, отправлять на него YDVP-запросы и получать YDVP-ответы.

Модуль сервера включает в себя всё необходимое для запуска сервера на заданном порту устройства.

3.3 Структура и состав классов

На рисунках 3.1 – 3.4 предоставлены диаграммы классов компонентов приложения.

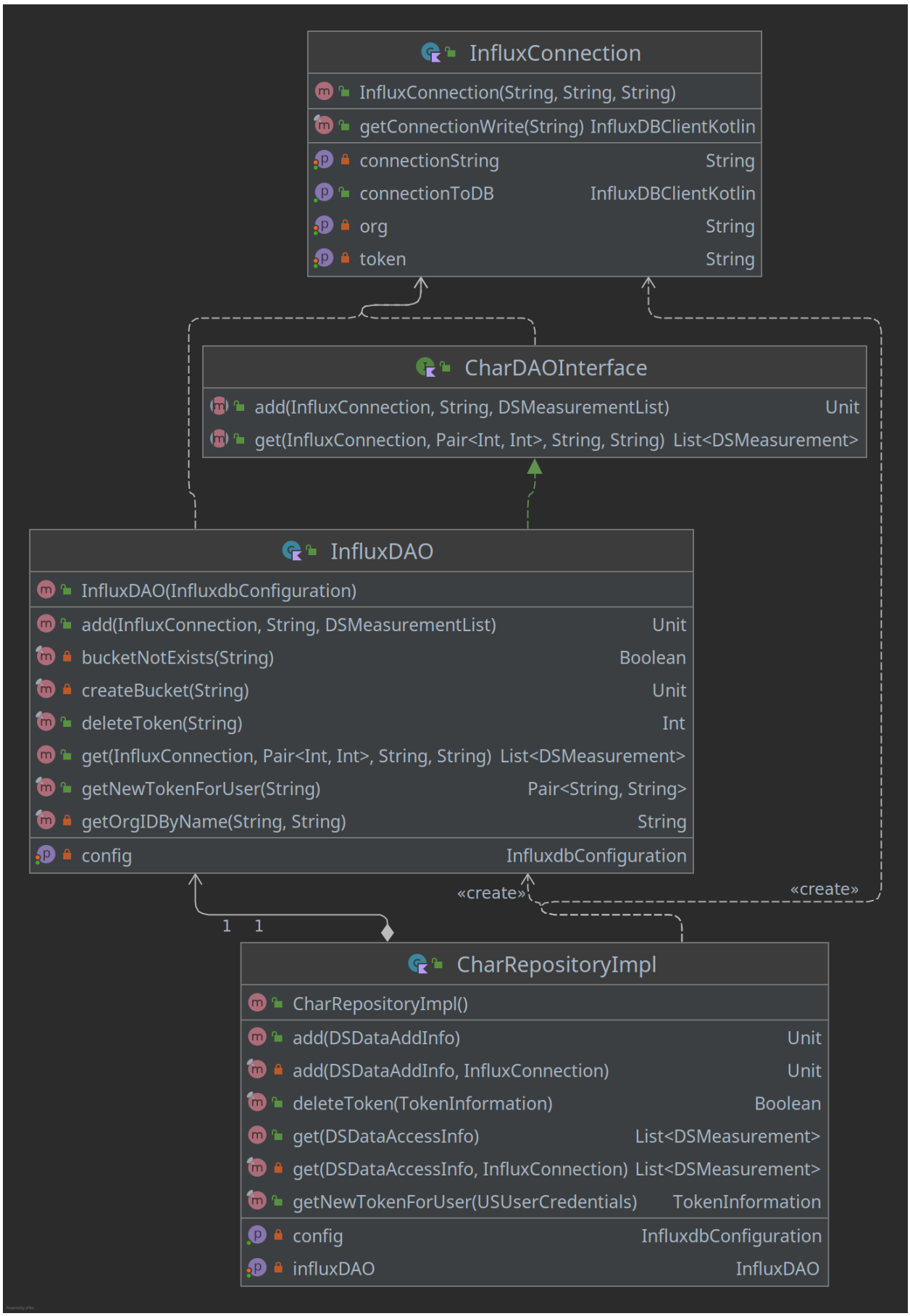

Рис. 3.1: Диаграмма классов слоя доступа к данным приложения.

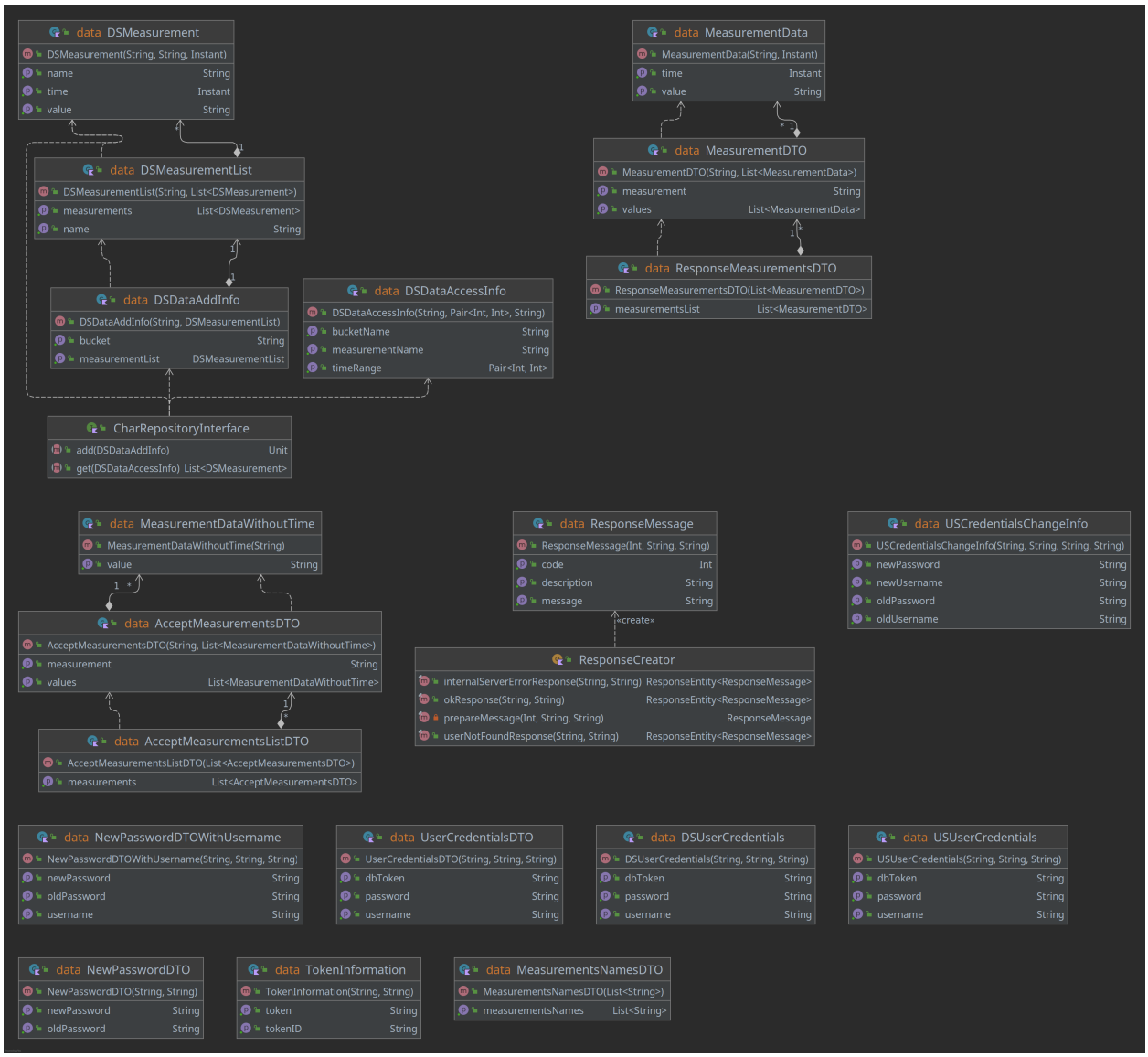

Рис. 3.2: Диаграмма классов бизнес-логики приложения.

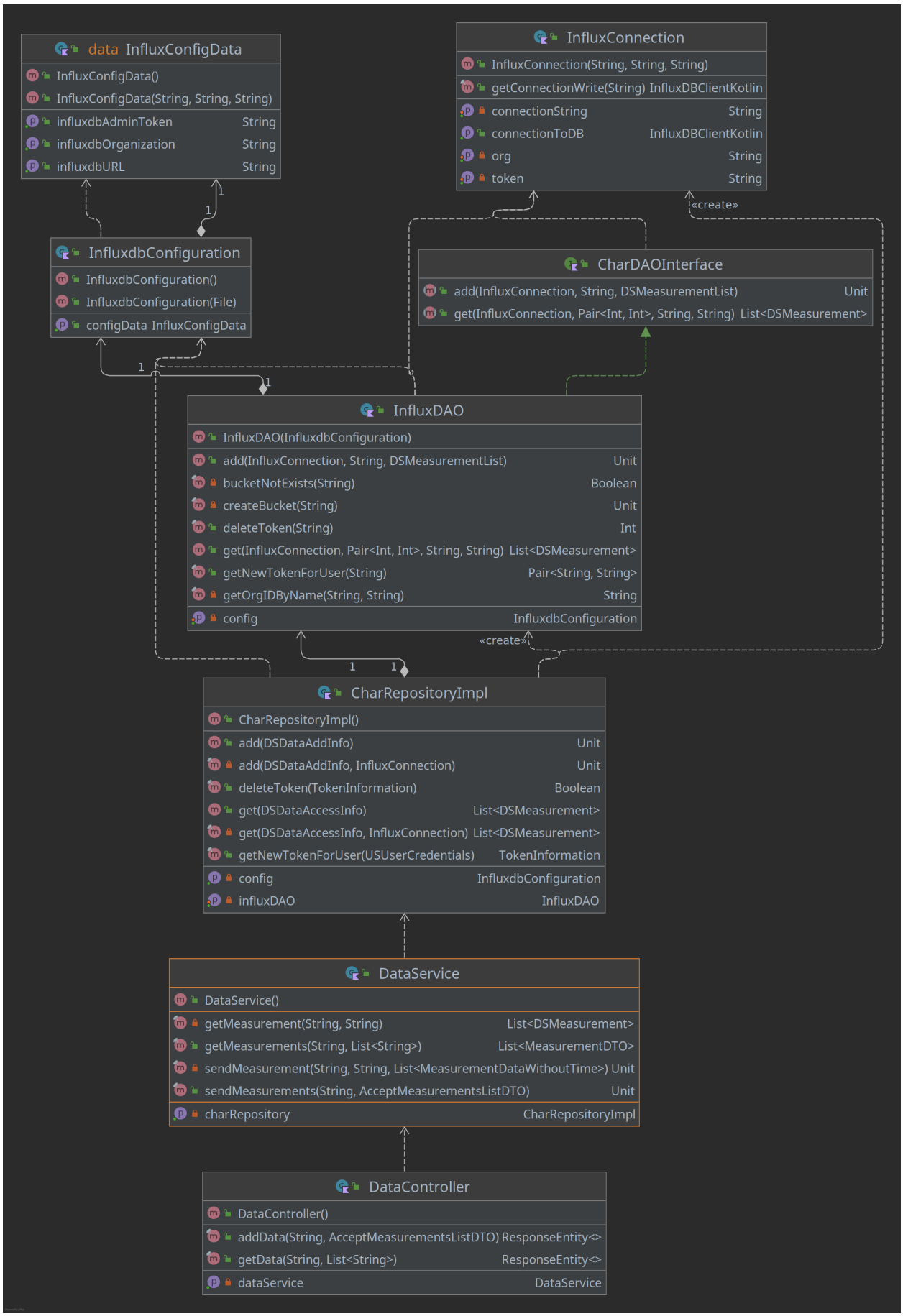

Рис. 3.3: Диаграмма классов, показывающая связь контроллера и слоя доступа к данным.

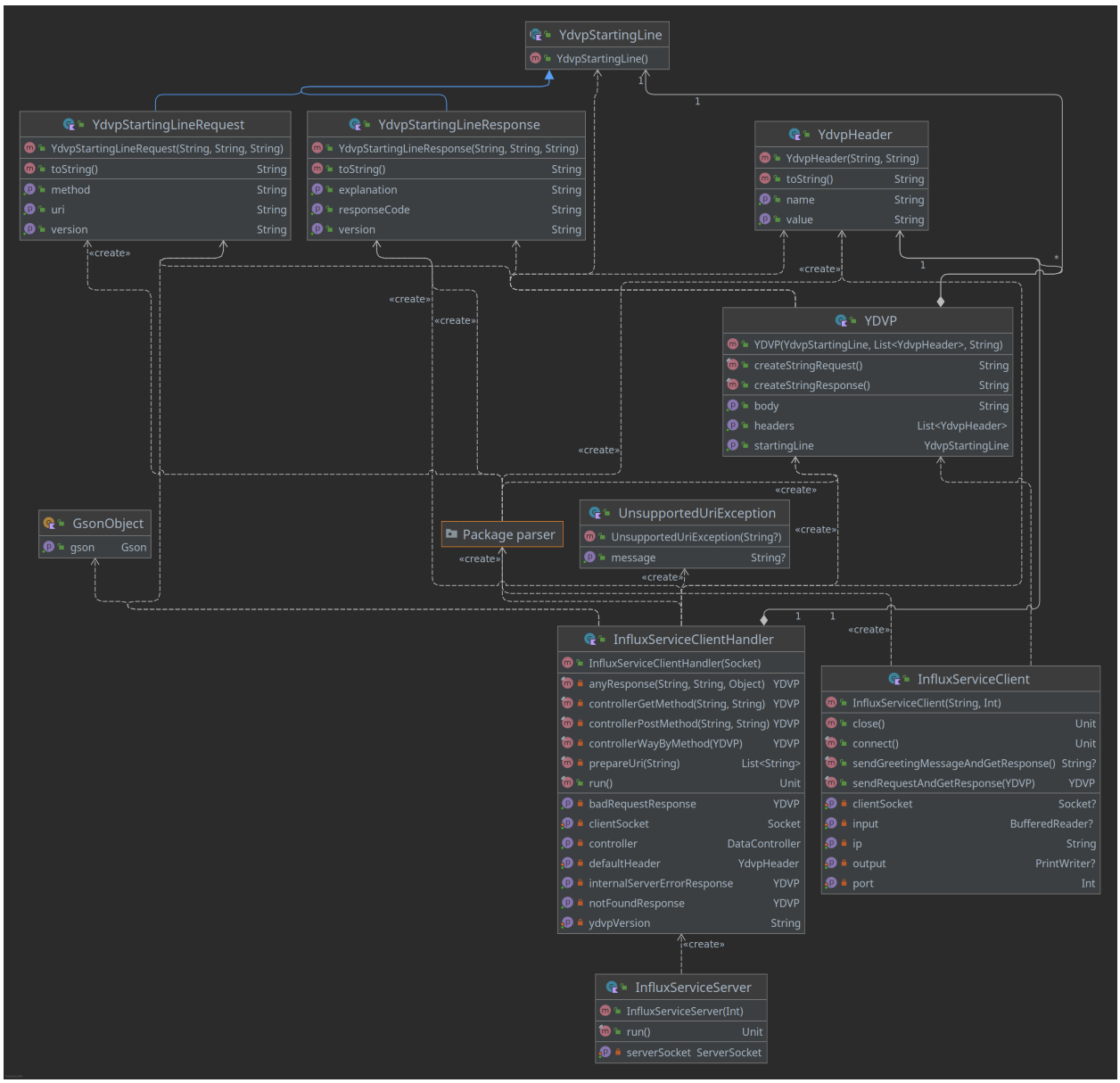

Рис. 3.4: Диаграмма классов серверной части приложения.

3.4 Пример использования приложения

Вывод

В качестве средств реализации были выбраны язык программирования Kotlin и среда разработки Intellij IDEA.

В разделе были предоставлены краткие сведения о модулях программы, в которых была предоставлена информация об используемых паттернах проектирования, а также системе взаимодействии классов в приложении.

Также были рассмотрены структура и состав классов, представленных в форме UML-диаграмм.

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

Список литературы

1. Gallup. Employee Burnout: Causes and Cures // Gallup. 2020. p. 32.

2. Moss J. Burnout Is About Your Workplace, Not Your People [Электронный ресурс]. Режим доступа: https://hbr.org/2019/12/burnout-is-about-your-workplace-not-your-people (дата обращения 27.03.2021).

3. Г.А. Макарова. Синдром эмоционального выгорания. Просвящение, 2009. с. 432.

4. Е.А. Пигарова А.В. Плещева. Синдром хронической усталости: современные представления об этиологии // Ожирение и метаболизм. 2010. с. 13.

5. The Kotlin InfluxDB 2.0 Client Github [Электронный ресурс]. Режим доступа: https://github.com/influxdata/influxdb-client-java/tree/master/client-kotlin (дата обращения 13.12.2021).

6. Influx Data: InfluxDB v2 API [Электронный ресурс]. Режим доступа: https://docs.influxdata.com/influxdb/v2.1/reference/api/ (дата обращения 13.12.2021).

7. Гайнанова Р.Ш. Широкова О.А. Создание клиент-серверных приложений // Вестник Казанского технологического университета. 2017. № 9. С. 79–84.

8. Учебно-методические материалы для судентов кафедры АСОИУ: архитектура клиент-сервер [Электронный ресурс]. Режим доступа: https://www.4stud.info/networking/lecture5.html (дата обращения 13.12.2021).

9. Обобщенный Model-View-Controller (Сергей Рогачёв) [Электронный ресурс]. Режим доступа: http://rsdn.org/article/patterns/generic-mvc.xml (дата обращения 14.12.2021).

10. Паттерны для новичков: MVC vs MVP vs MVVM [Электронный ресурс]. Режим доступа: https://habr.com/ru/post/215605/ (дата обращения 14.12.2021).

11. Бондаренко Т.В. Федотов Е.А. Бондаренко А.В. Разработка http сервера // ИВД. 2018. Т. 49, № 2.

12. Официальная документация HTTP/1.1 [Электронный ресурс]. Режим доступа: https://datatracker.ietf.org/doc/html/rfc2616 (дата обращения 13.12.2021).

13. Kotlin language specification [Электронный ресурс]. Режим доступа: https://kotlinlang.org/spec/introduction.html (дата обращения 09.10.2020).