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 цели.

1. Аналитический раздел

В данном разделе формализуется задача, приводится требуемая функциональность разрабатываемого приложения и информация об архитектурных решении и технологиях, применимых к разрабатываемому программному обеспечению, а также 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 языка программирования 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 с использованием HTTP аутентификации, поддержкой JWT и базовой аутентификации, а также включающих в своё тело данные в формате JSON.

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. [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 позволяет модифицировать указанные компоненты независимо друг от друга. [9]

Модель предоставляет данные предметной области представлению и реагирует на команды контроллера, изменяя свое состояние. Представление отвечает за отображение данных предметной области (модели) пользователю, реагируя на изменения модели. Контроллер интерпретирует действия пользователя, оповещая модель о необходимости изменений. [9]

Модель обладает следующими признаками [10]:

- содержит бизнес-логику приложения;
- не имеет 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 были получены от модели;
- может включать в себя реализацию некоторой бизнес-логики.

Контроллер обладает следующими признаками [10]:

- 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 чтения и записи 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. Рассмотрена применимые к реализации ”клиент-серверного”приложения шаблоны проектирования и определены части реализуемого программного обеспечения, которые будут их задействовать.

2. Конструкторский раздел

В данном разделе предоставлены диаграммы вариантов использования, 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.

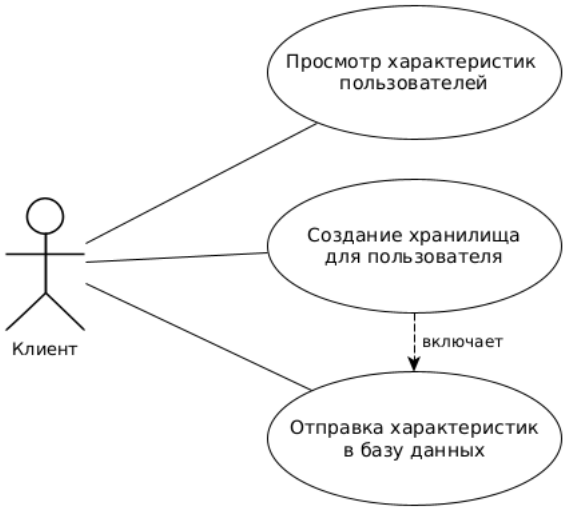

Рис. 2.1: Диаграмма вариантов использования для пользователя.

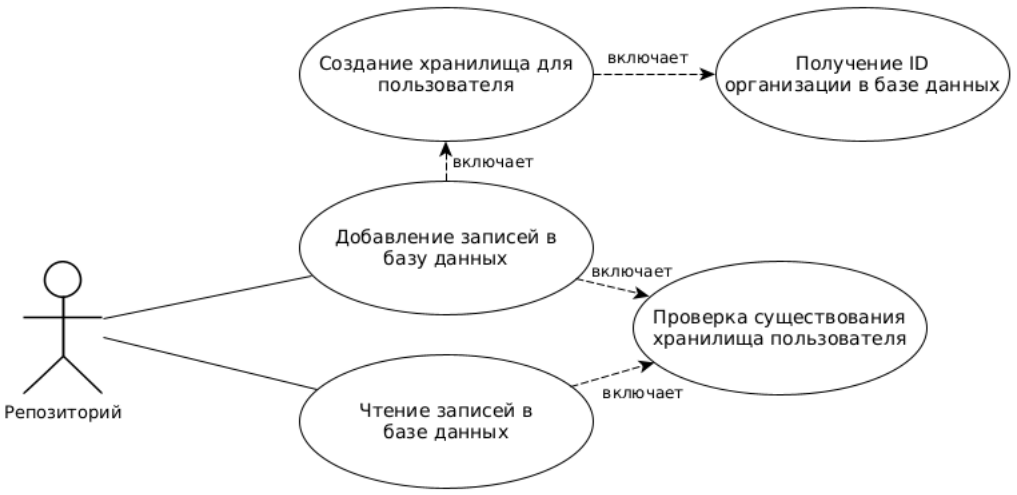

Рис. 2.2: Диаграмма вариантов использования для репозитория.

2.2 Схема работы сервера

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

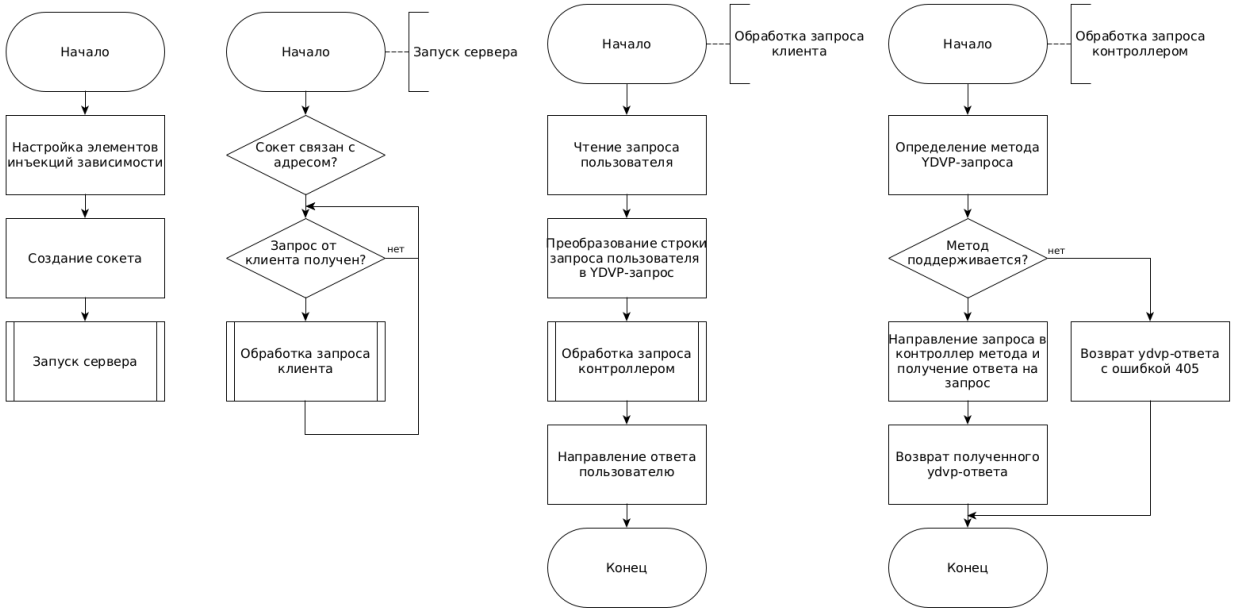

Рис. 2.3: Схема работы сервера приложения.

2.3 Описание YDVP-протокола

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

YDVP-запрос и YDVP-ответ включают 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.

Стартовая строка ответа на запрос включает в себя:

- версию используемого протокола;
- код состояния (определяющее дальнейшее содержимое сообщения и поведение клиента);

- пояснение к коду состояния.

Код состояния, определяется по спецификации HTTP.

Вывод

В разделе была предоставлена диаграмма вариантов использования, позволившая выделить операции, доступные клиенту и серверу. Также была предоставлена схема работы сервера, что позволило выделить основные компоненты, действующие в системе. Была описана версия 0.1 собственного протокола YDVP для взаимодействия клиента и сервера реализуемого приложения.

3. Технологический раздел

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

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 шаблоне проектирования 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, 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. Данный пакет 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.

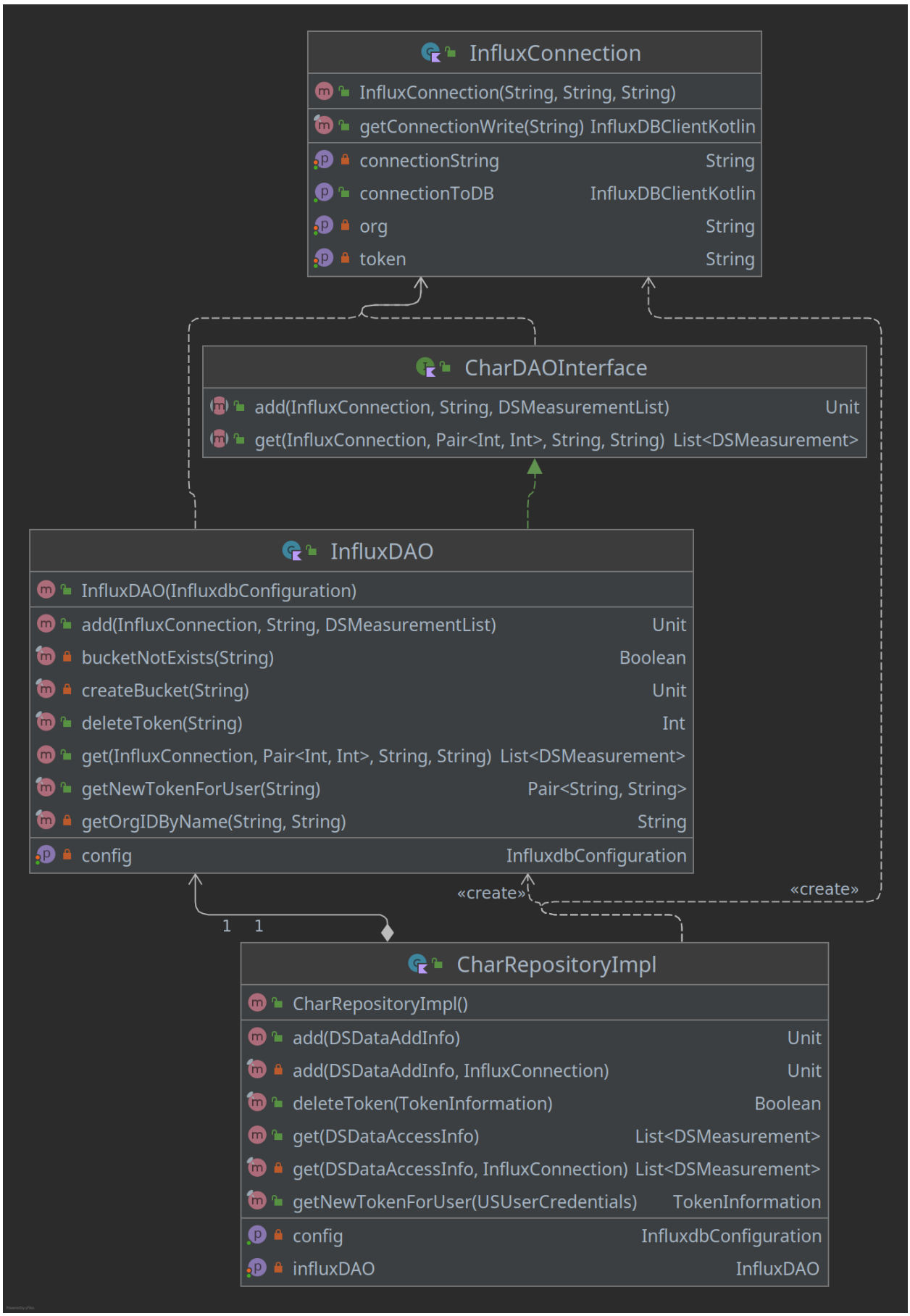

Рис. 3.1: Диаграмма классов слоя доступа к данным приложения.

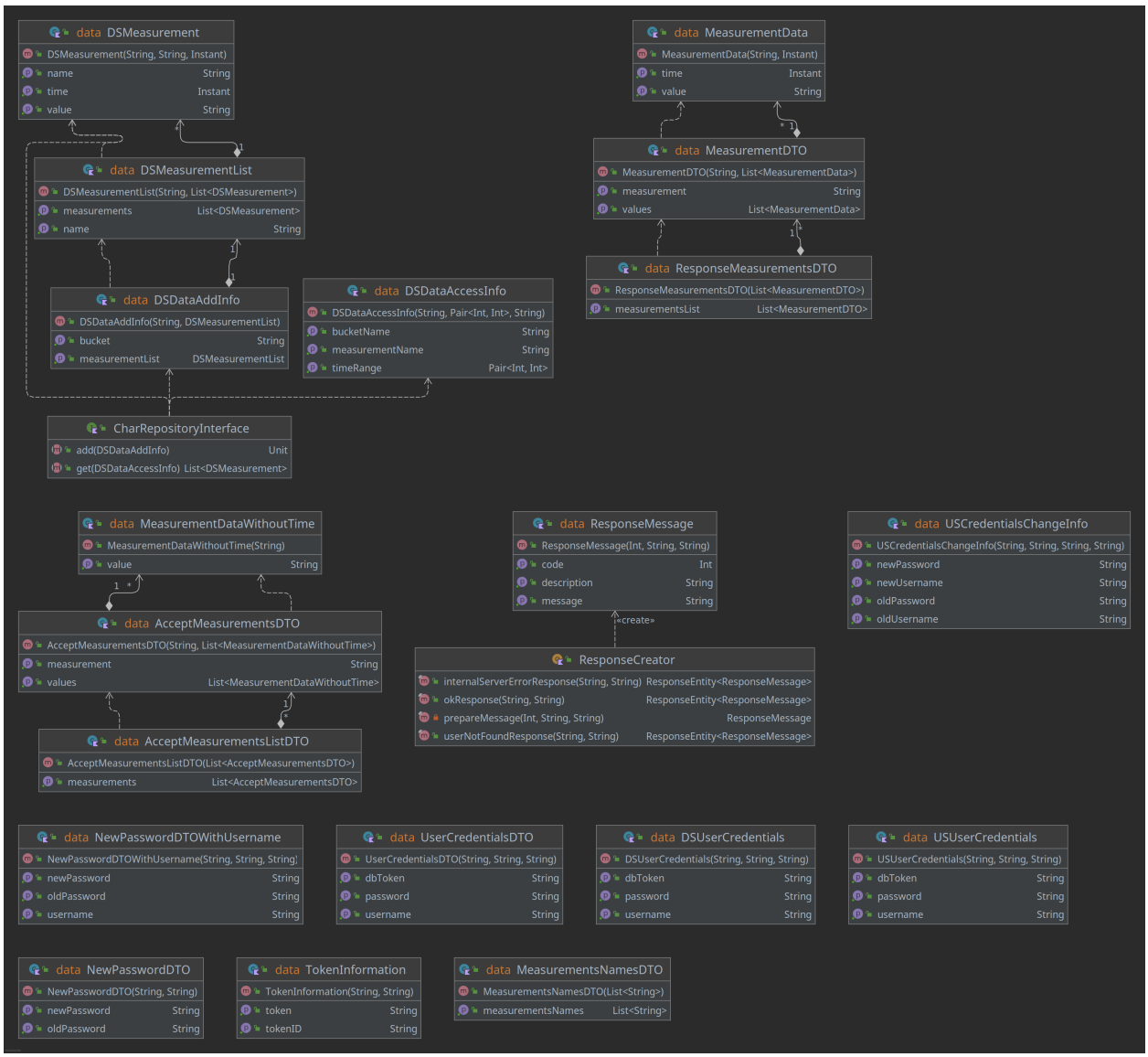

Рис. 3.2: Диаграмма классов бизнес-логики приложения.

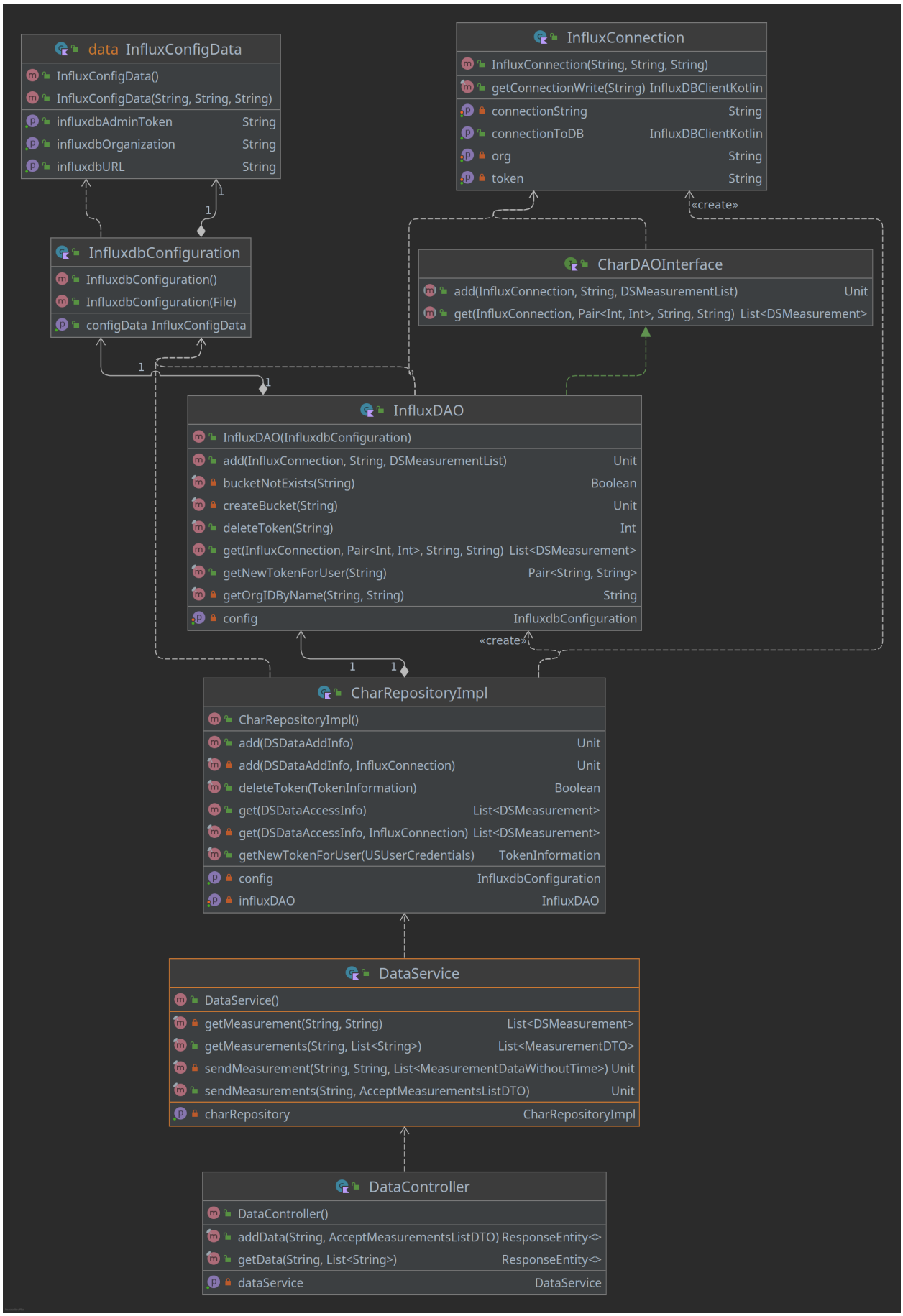

Рис. 3.3: Диаграмма классов, показывающая связь контроллера и слоя доступа к данным.

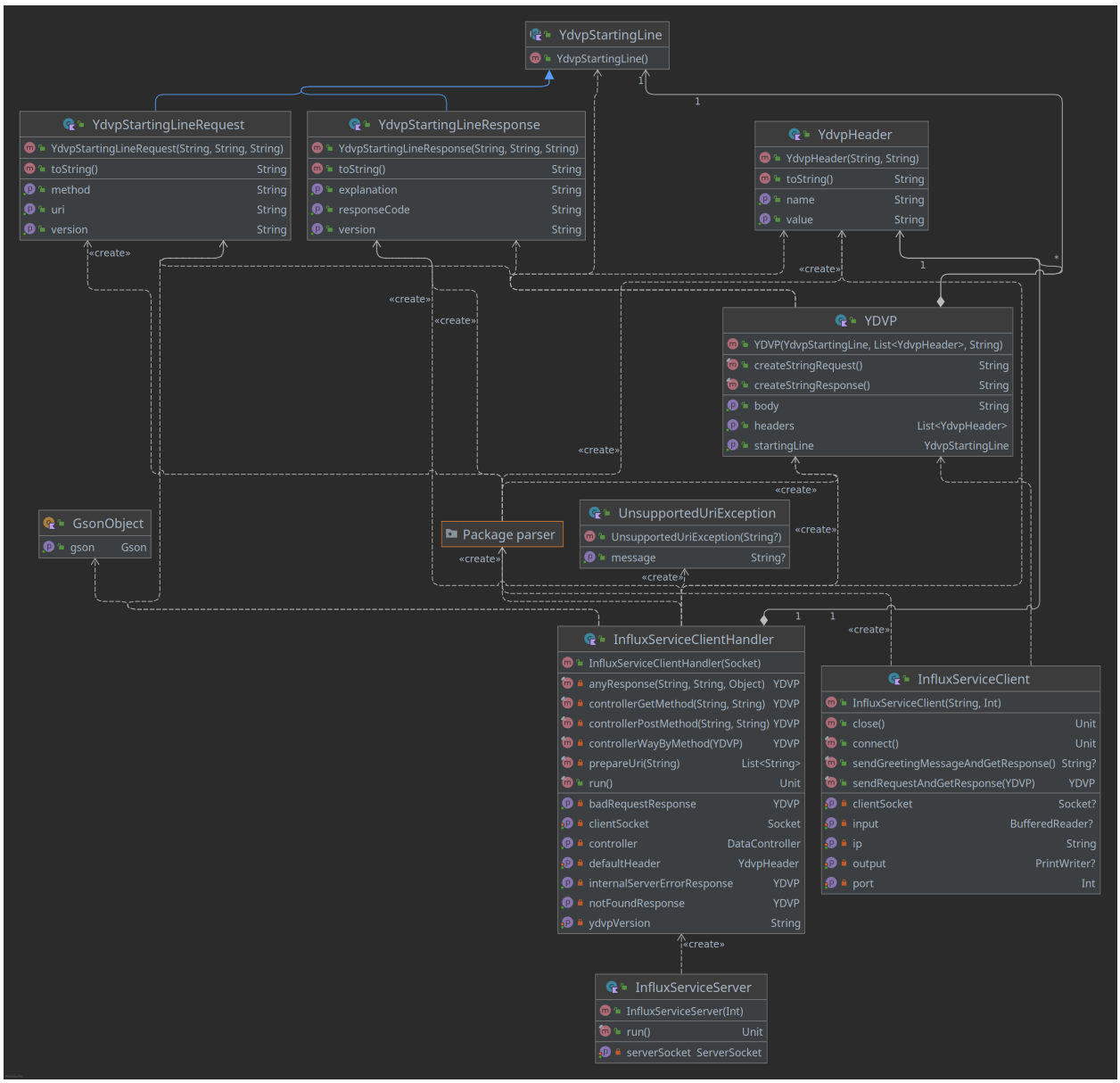

Рис. 3.4: Диаграмма классов серверной части приложения.

3.4 Особенности реализации

В листинге 1 предоставлено описание класса сервера приложения.

Данный класс инициализирует классы для инъекции зависимостей в модули, используемые сервисами и контроллерами, а также передаёт приходящих клиентов обработчику (классу InfluxServiceClientHandler) в отдельном потоке, таким образом достигается возможность одновременной обработки запросов от нескольких клиентов.

Листинг 1: Реализация класса InfluxServiceServer

```
185  class InfluxServiceServer(socketPort: Int) {
186      init {
```

```
187             startKoin {
188                 modules(module {
189                     single { InfluxdbConfiguration() }
190 
191                     single { CharRepositoryImpl() }
192 
193                     single { DataService() }
194 
195                     single { DataController() }
196                 })
197             }
198         }
199 
200         private val serverSocket = ServerSocket(socketPort)
201 
202         fun run() {
203             getRuntime().addShutdownHook(Thread {
204                 println("Server on port
                    ↪  ${serverSocket.localPort} stopped")
205             })
206 
207             if (!serverSocket.isBound ||
                ↪  serverSocket.isClosed) {
208                 throw SocketException("Server socket is
                    ↪  already in use")
209             }
210 
211             println("Server started on port
                ↪  ${serverSocket.localPort}")
212 
213             while (true) {
214                 val clientSocket = serverSocket.accept()
215                 thread {
216                     InfluxServiceClientHandler(clientSocket)
217                         .run()
218                 }
219             }
220         }
221     }
```

В листинге 2 предоставлено описание класса обработчика запроса клиента.

Данный класс включает в себя необходимый для навигации по ресурсам контроллер, а также типовые объекты YDVP-ответов и методы обработки запроса клиента.

Листинг 2: Реализация класса InfluxServiceClientHadler

```
26  class InfluxServiceClientHandler(private val clientSocket:
    ↪  Socket) {
27      private val ydvpVersion = "0.1"
28      private val defaultHeader = YdvpHeader("Server",
        ↪  "127.0.0.1")
29
30      private val controller by
        ↪  inject<DataController>(DataController::class.java)
31
32      private fun anyResponse(code: String, explanation:
        ↪  String, body: Any): YDVP {
33          return YDVP(
34              YdvpStartingLineResponse(
35                  ydvpVersion,
36                  code,
37                  explanation
38              ),
39              listOf(defaultHeader),
40              GsonObject.gson.toJson(body)
41          )
42      }
43
44      private val badRequestResponse by lazy {
45          YDVP(
46              YdvpStartingLineResponse(ydvpVersion, "400",
                ↪  "BAD REQUEST"),
47              listOf(defaultHeader)
48          )
49      }
50      private val internalServerErrorResponse by lazy {
51          YDVP(
52              YdvpStartingLineResponse(ydvpVersion, "500",
                ↪  "INTERNAL SERVER ERROR"),
```

```
53                  listOf(defaultHeader)
54              )
55          }
56
57          private val methodNotAllowed by lazy {
58              YDVP(
59                  YdvpStartingLineResponse(ydvpVersion, "405",
                     ↪  "METHOD NOT ALLOWED"),
60                  listOf(defaultHeader)
61              )
62          }
63
64          private val notFoundResponse by lazy {
65              YDVP(
66                  YdvpStartingLineResponse(ydvpVersion, "404",
                     ↪  "NOT FOUND"),
67                  listOf(defaultHeader)
68              )
69          }
70
71          private fun prepareUri(uri: String): List<String> {
72              val parsedUri = uri.split("/").toMutableList()
73
74              if (parsedUri[0] != "")
75                  throw UnsupportedUriException("URI format
                     ↪  error")
76
77              parsedUri.removeAt(0)
78              return parsedUri
79          }
80
81          private fun controllerPostMethod(uri: String, body:
             ↪  String): YDVP {
82              val parsedUri = prepareUri(uri)
83
84              println("Body in controller:")
85              println(body)
86
87              return when (parsedUri.first()) {
88                  "data" -> {
89                      if (parsedUri.size < 2)
```

```
90                      throw UnsupportedUriException("Not
                         ↪  enough inline arguments")
91                  val response = controller.addData(
92                      parsedUri[1],
93                      GsonObject.gson.fromJson(
94                          body,
95                          AcceptMeasurementsListDTO::class
96                              .java
97                      )
98                  )
99
100                 anyResponse(response.statusCodeValue
101                                         .toString(),
102                             response.statusCode.name,
103                             response.body)
104             }
105             else -> throw
                 ↪  UnsupportedUriException("Unsupported URI")
106         }
107     }
108
109     private fun controllerGetMethod(uri: String, body:
         ↪  String): YDVP {
110         val parsedUri = prepareUri(uri)
111
112         return when (parsedUri.first()) {
113             "data" -> {
114                 if (parsedUri.size < 2)
115                     throw UnsupportedUriException("Not
                         ↪  enough inline arguments")
116                 val response =
117                     controller.getData(parsedUri[1],
                         ↪  GsonObject.gson.fromJson(body,
                         ↪  Array<String>::class.java).toList())
118
119                 anyResponse(response.statusCodeValue
120                                         .toString(),
121                             response.statusCode.name,
122                             response.body)
123             }
124             else -> throw UnsupportedUriException("Way not
                 ↪  found")
```

```
125             }
126         }
127
128         private fun controllerWayByMethod(ydvpRequest: YDVP):
            ↪ YDVP {
129             ydvpRequest.startingLine as
                ↪ YdvpStartingLineRequest
130             val method = ydvpRequest.startingLine.method
131             val uri = ydvpRequest.startingLine.uri
132
133             val body = ydvpRequest.body
134
135             return try {
136                 when (method) {
137                     "GET" -> controllerGetMethod(uri, body)
138                     "POST" -> controllerPostMethod(uri, body)
139                     else -> methodNotAllowed
140                 }
141             } catch (exc: NullPointerException) {
142                 badRequestResponse
143             } catch (exc: UnsupportedUriException) {
144                 notFoundResponse
145             } catch (exc: Exception) {
146                 println("EXCEPTION")
147                 println(exc.javaClass)
148                 println(exc.localizedMessage)
149                 internalServerErrorResponse
150             }
151         }
152
153         fun run() {
154             println("Accepted client on
                    ↪ ${clientSocket.localSocketAddress} from
                    ↪ ${clientSocket.remoteSocketAddress}")
155
156             val bufferedReader = BufferedReader(
157                                      InputStreamReader(
158                                          clientSocket
159                                              .getInputStream()
160                                      )
161                                  )
```

```
162
163             var gotRequest = bufferedReader.readLine() + "\n"
164             while (bufferedReader.ready())
165                 gotRequest += bufferedReader.readLine() + "\n"
166
167             println("Got lines are: \n$gotRequest")
168             val clientOut =
                 ↪  PrintWriter(clientSocket.getOutputStream(),
                 ↪  true)
169
170             val ydvpRequest = try {
171                 YdvpParser().parseRequest(gotRequest)
172             } catch (exc: Exception) {
173                 clientOut.println(badRequestResponse)
174
175                 return
176             }
177
178             val response = controllerWayByMethod(ydvpRequest)
179                                 .createStringResponse()
180             println("Returned to client:\n$response")
181             clientOut.println(response)
182         }
183     }
```

3.5 Пример использования приложения

В листингах 3–4 предоставлены пример реализации взаимодействия пользователя с сервером.

Листинг 3: Пример организации серверной части.

```
1   package examples.helloexample
2
3   import server.InfluxServiceServer
4
5   fun main() {
6       val server = InfluxServiceServer(6666)
7
8       server.run()
```

```
}
```

Листинг 4: Пример вызовов со стороны клиента.

```
package examples.helloexample

import client.InfluxServiceClient
import domain.dtos.AcceptMeasurementsDTO
import domain.dtos.AcceptMeasurementsListDTO
import domain.dtos.MeasurementDataWithoutTime
import domain.dtos.ResponseMeasurementsDTO
import gson.GsonObject
import protocol.YDVP
import protocol.YdvpHeader
import protocol.YdvpStartingLineRequest
import protocol.YdvpStartingLineResponse
import java.net.ConnectException

val measurementsToSend = AcceptMeasurementsListDTO(
    listOf(
        AcceptMeasurementsDTO(
            "pulse", listOf(
                MeasurementDataWithoutTime("30"),
                MeasurementDataWithoutTime("40")
            )
        ),
        AcceptMeasurementsDTO(
            "botArterialPressure", listOf(
                MeasurementDataWithoutTime("40"),
                MeasurementDataWithoutTime("50")
            )
        ),
        AcceptMeasurementsDTO(
            "topArterialPressure", listOf(
                MeasurementDataWithoutTime("80"),
                MeasurementDataWithoutTime("90")
            )
        )
    )
```

```
36      )
37
38      fun sendTest() {
39          val client = InfluxServiceClient("localhost", 6666)
40
41          client.use {
42              try {
43                  client.connect()
44              } catch (exc: ConnectException) {
45                  println("Server is dead")
46                  return
47              }
48
49              client.sendRequestAndGetResponse(
50                  YDVP(
51                      YdvpStartingLineRequest("POST",
                        ↪  "/data/TestUser", "0.1"),
52                      listOf(YdvpHeader("Host", "127.0.0.1")),
53                      GsonObject.gson.toJson(measurementsToSend)
54                  )
55              )
56          }
57      }
58
59      fun getTest() {
60          val client = InfluxServiceClient("localhost", 6666)
61
62          client.use {
63              try {
64                  client.connect()
65              } catch (exc: ConnectException) {
66                  println("Server is dead")
67                  return
68              }
69
70              client.sendRequestAndGetResponse(
71                  YDVP(
72                      YdvpStartingLineRequest("GET",
                        ↪  "/data/TestUser", "0.1"),
73                      listOf(YdvpHeader("Host", "127.0.0.1")),
74                      GsonObject.gson.toJson(listOf("pulse",
                        ↪  "botArterialPressure"))
```

```
                )
            )
        }
    }

    fun main() {
        sendTest()
    //    getTest()
    }
```

На рисунках 3.5–3.6 предоставлен результат выполнения клиентской и серверной частей предоставленного примера.

```
/home/trvehazzk3r/Downloads/jdk-11.0.12/bin/java ...
Server started on port 6666
Accepted client on /127.0.0.1:6666 from /127.0.0.1:55082
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by retrofit2.Platform (file:/home/trvehazzk3r/.gradle/caches/modules-2/files-2.1/
WARNING: Please consider reporting this to the maintainers of retrofit2.Platform
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
Returned to client:
YDVP/0.1 200 OK
Server: 127.0.0.1

{"code":200,"message":"Measurements were carefully sent","description":"We know all about you now \u003e:c"}
```

Рис. 3.5: Результат выполнения запроса на стороне сервера.

```
/home/trvehazzk3r/Downloads/jdk-11.0.12/bin/java ...
Client connected from /127.0.0.1:55082 to localhost/127.0.0.1:6666
Formed request in client:
POST /data/TestUser YDVP/0.1
Host: 127.0.0.1

{"measurements":[{"measurement":"pulse","values":[{"value":"30"},{"value":"40"}]},{"measurement":"botArterialPressu
Response in client:
YDVP/0.1 200 OK
Server: 127.0.0.1

{"code":200,"message":"Measurements were carefully sent","description":"We know all about you now \u003e:c"}



Process finished with exit code 0
```

Рис. 3.6: Результат выполнения запроса на клиентской стороне.

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 шаблонах проектирования, а также системе взаимодействии классов в приложении.

Была рассмотрена структура и состав классов, представленных в форме UML-диаграмм, и приведены особенности реализации, указывающие на возможность одновременной работы реализованного сервера с несколькими клиентами.

Пример, предоставленный в разделе, показал возможные пути использования реализованных компонентов.

ЗАКЛЮЧЕНИЕ

Во время выполнения курсового проекта были достигнуты поставленные задачи:

- формализована задача,
- определена требуемая функциональность,
- проанализированы существующие решения,
- описан протокол взаимодействия клиента и сервера,
- разработано приложения для решения поставленной цели.

Проведённая аналитическая работа позволила формализовать задачу и определить требуемую функциональность, реализованную на сервере. Также были проанализированы существующие решения The Kotlin InfluxDB 2.0 Client и InfluxDB v2 API, которые в дальнейшем позволили реализовать приложение с использованием возможностей, предоставляемых данными решениями. Были рассмотрены понятия клиент-серверной архитектуры, шаблон проектирования Model-View-Controller и протокол HTTP, на основе которого был реализован собственный протокол взаимодействия сервера и пользователя YDVP.

В результате работы, проведенной в конструкторском разделе, были приведены диаграммы вариантов использования для клиента и репозитория. Также была определена схема работы сервера и описание протокола YDVP версии 0.1.

Для реализации в качестве используемого языка программирования был выбран ЯП Kotlin, а в качестве среды разработки – IntelliJ IDEA.

В результате работы было реализовано приложение, состоящее из 5 модулей, структура и состав классов которых были представлены в графической интерпретации диаграммы классов. Приведены особенности реализации и пример использования приложения.

В ходе выполнения поставленных задач были получены знания в области компьютерных сетей, а также изучены возможности ЯП Kotlin, библиотеки OkHttp3, а также особенности работы сокетов на Java Virtual Machine.

В качестве дальнейшего развития проекта могут быть предприняты следующие действия:

- расширение стандарта YDVP и переход на новую версию,

- увеличение доступных на сервере ресурсов,
- переход на использование фреймворка SpringBoot,
- добавление системы аутентификации,
- добавление поддержки JWT-авторизации.

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).