Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «**Информатика и системы управления**»

КАФЕДРА «**Программное обеспечение ЭВМ и информационные технологии**»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ НА ТЕМУ:

«Метод систематического распознавания усталости на автоматизированном рабочем месте»

Студент группы **ИУ7-83Б** _____ (Подпись, дата) **Д.В. Якуба** (И.О. Фамилия)

Руководитель _____ (Подпись, дата) **Ю.В. Строганов** (И.О. Фамилия)

Нормоконтролер _____ (Подпись, дата) _____ (И.О. Фамилия)

Москва — 2022г.

РЕФЕРАТ

Расчетно-пояснительная записка 95 с., 20 рис., 3 табл., 45 ист., 3 прил.

Объектом разработки является метод систематического распознавания усталости на автоматизированном рабочем месте.

Цель работы — разработать и реализовать метод распознавания усталости оператора автоматизированного рабочего места по данным, приходящим с устройств взаимодействия пользователя с системой.

Для достижения поставленной цели необходимо:

- провести анализ существующих методов определения усталости;
- провести анализ действий и характеристик, позволяющих определить усталость пользователя автоматизированного рабочего места;
- определить методы снятия выделенных действий и характеристик;
- разработать метод распознавания усталости оператора;
- реализовать разработанный метод.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

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

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, понижающих работоспособность сотрудников. Несмотря на то, что 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, управлять его работоспособностью и проводить профилактику проявления вышеописанных синдромов, приводящих к неутешительным последствиям.

1 Аналитический раздел

В данном разделе рассмотрены термины предметной области, включающие в себя понятия усталости, хронической усталости, стресса, в том числе профессионального. Также приведена формализация цели разрабатываемого метода, рассмотрены методы анализа клавиатурного почерка, скорости печати и количества ошибок, траекторий перемещения курсора мыши, использования координатного устройства, внешнего состояния пользователя, речевых характеристик и виброакустических шумов при наборе текста или использования мыши. Приведены биофизические факторы, позволяющие определить усталость пользователя. Формализованы требования к разрабатываемому методу.

1.1 Термины предметной области

1.1.1 Усталость

Усталость — ощущение физической усталости и отсутствия энергии, которое нарушает повседневную физическую и социальную жизнь, не связанное с умственным переутомлением, депрессией, сонливостью, нарушением двигательных функций. Выделяют два феномена: физическую усталость, которая заключается в снижении способности поддержания физической активности, и психическую, которая заключается в уменьшении способности выполнения умственных задач. [5]

Диагностирование усталости у пациента затруднительна в силу отсутствия общепринятой симптоматики и определения синдрома. На момент 2022 года объективных критериев усталости не выделено. Также отсутствуют инструментальные методы оценки усталости. [5]

1.1.2 Хроническая усталость

Синдром хронической усталости (СХУ) — это инвалидизирующее клиническое состояние, которое характеризуется стойкой усталостью после физических нагрузок, и сопровождающееся симптомами, связанными с когнитивной, иммунологической, эндокринологической и автономной дисфункцией. [6]

Согласно исследованиям Института медицины США, от 836 тысяч до 2.5 миллионов американцев страдают синдромом хронической усталости. Данный факт приводит к финансовым затратам от 17 до 24 миллиардов долларов в год, каждая семья теряет 20 тысяч долларов. Уровень безработицы среди болеющих

составляет от 35% до 69%. [7]

Наличие синдрома у пациента определяется наличием необъяснимой стойкой хронической усталости. К симптомам заболевания относят [6]:

- нарушение памяти или концентрации внимания;
- фарингит;
- болезненные при пальпации шейные или подмышечные лимфоузлы;
- болезненность или скованность мышц;
- болезненность суставов;
- вновь возникающая головная боль или изменение ее характеристик;
- сон, не приносящий ощущения восстановления;
- усугубление усталости, продолжающееся более 24 часов.

Более трех одновременно проявляющихся симптомов, сохраняющихся в течение более 5 месяцев, позволяют диагностировать синдром хронической ус талости. [6]

1.1.3 Стресс

Стресс — это реакция человеческого организма на неблагоприятные воздействия внешней среды, которая носит психофизиологический характер. К подобным неблагоприятным воздействиям можно отнести: конфликтные ситуации, потеря близких людей, насилие, несправедливость, ограничение потребностей. [8]

Стрессовая реакция заключается в активации парасимпатического и симпатического отдела вегетативной нервной системы, в следствие чего стимулируется одна из трех психофизиологических осей стресса. [8]

Активация симпатической нервной системы заключается в общем возбуждении внутреннего органа, осуществляющего реагирование на стресс, а активация парасимпатического характера проявляется в виде чрезмерного возбуждения внутреннего органа, сменяющегося на торможение, замедление или нормализацию. Проявление симпатической реакции, в случае разрешения стрессовой ситуации, может смениться парасимпатической заторможенностью, возвращающей органы к нормальному функционированию. [8]

При продолжении течения стрессовой реакции задействуется нейроэндокринный процесс, который активирует все ресурсы организма: выделение в кровь адреналина и норадреналина, выброс гормонов. Данная реакция приводит к большему возбуждению, увеличению сердечного выброса, подъему арте-

риального давления, снижению кровотока к почкам, повышению уровня жирных кислот в плазме и увеличению уровня холестерина. [8]

1.1.4 Стадии общего адаптационного синдрома

Общий адаптационный синдром — это сочетание стереотипных реакций, возникающих в организме в ответ на действие стрессоров и обеспечивающих ему устойчивость не только к стрессорному агенту, но и по отношению к другим болезнетворным факторам. [9]

Определены три стадии общего адаптационного синдрома: тревоги, устойчивости и истощения.

Стадия тревоги представляет собой мобилизацию ресурсов организма. Она имеет продолжительность от 6 до 48 часов и включает в себя стадии шока и противошока. Стадия противошока характеризуется мобилизацией основных функциональных систем организма: нервной, симпато-адреналовой, эндокринной и адрено-кортитропной. [9]

При длительном воздействии стрессора на организм наступает стадия устойчивости (адаптации). На данной стадии организм становится более устойчивым к действию раздражителя и другим патогенным факторам, повышается образование и секреция глюкокортикоидов. Поддерживается состояние гомеостаза в присутствии стрессора. [9]

При длительном воздействии стрессора адаптивные механизмы, участвующие в поддержании резистентности, исчерпывают себя, и наступает стадия истощения организма. Данная стадия не является обязательной и стрессовая ситуация может быть окончена до ее наступления. Протекание реакции заключается в активизации эндокринной системы и обеднении коры надпочечников. [9]

В стрессовой ситуации важно выявить наступление стадии истощения организма для его дальнейшего плодотворного функционирования. Подобные действия способны позволить корректировать нагрузку на человека, а также эффективно управлять его активностью.

1.1.5 Профессиональный стресс

Профессиональный стресс возникает при воздействии на работника эмоционально-отрицательных или экстремальных факторов окружающей среды при выполнении профессиональной деятельности. Подобные стрессы влияют на здоровье человека и производительность труда, в особенности на сотруд-

ников организаций, которые работают в условиях систематических рабочих и информационных перегрузок. [10]

Среди причин возникновения профессионального стресса различают следующие организационные факторы в трудовой деятельности человека [10]:

- режим трудовой деятельности (большая рабочая нагрузка, функциональная перегрузка, сроки выполнения);
- нерациональная организация труда и рабочего места (нечеткие ограничения полномочий и обязанностей, неоднозначные требования к выполняемой работе);
- неудовлетворительные условия труда персонала (монотонная работа, отсутствие информационных и материальных ресурсов);
- отсутствие эффективной системы мотивации и стимулирования (недостаточное вознаграждение за труд, риск штрафных санкций противоречие интересов работников и их функциональных обязанностей);
- неэффективный стиль управления,
- отсутствие или недостаток поддержки со стороны начальства;
- неудовлетворительная социально-психологическая атмосфера (неблагоприятный социально-психологический климат в коллективе, отсутствие или недостаток поддержки со стороны коллег, нарушение внутригрупповых норм поведения);
- лишение перспектив карьерного роста, бесперспективность работы.

К последствиям профессионального стресса относят [10]:

- низкую эффективность работы;
- апатичность;
- консерватизм;
- пессимистичность;
- избегание коммуникации на работе;
- физическое истощение, болезненность.

Профилактика профессиональных стрессов может включать в себя [10]:

- рациональное распределение рабочей нагрузки;
- четкое описание должностных обязанностей, полномочий;
- создание комфортной социально-психологической среды в трудовом коллективе и организации, устранение конфликтов;
- предоставление возможности карьерного роста сотрудников в рамках

организации;

- предоставление информации о системе оценки труда и системе мотивации трудовой деятельности, поощрение лучших сотрудников.

Последствия профессиональных стрессов носят как физиологический, так и психологический характер. Предупреждение наступления стадии истощения с использованием информации, поступающей от пользователя, позволит сохранить здоровье и работоспособность трудящихся, а также решить проблему повышения эффективности работы и заболеваемости на высоконагруженных трудовых местах.

1.2 Формализация цели метода систематического распознавания усталости на автоматизированном рабочем месте

В стрессовой ситуации важно выявить наступление стадии истощения организма для его дальнейшего плодотворного функционирования. Подобные действия способны позволить корректировать нагрузку на человека, а также эффективно управлять его активностью.

Для предупреждения наступления стадии истощения требуется определить момент наступления стадии тревоги или стадии тревоги и затем устойчивости. Данные наблюдения позволят определить, продолжается ли воздействие стрессора на организм. В случае возвращения характеристик пользователя к стандартным, построенным на наблюдении вне периода стресса, дальнейших действий не потребуется. Предупреждение о наступлении стадии истощения предполагается в случае, когда характеристики пользователя, находящегося в установленной стадии устойчивости, устремляются к нестандартным значениям, что говорит об активизации эндокринной системы, угнетающей функции внутренних органов.

Согласно исследованиям, проведенными Федеральной службой государственной статистики [11], 71.2% от общей численности населения в городской местности используют персональные компьютеры дома, в то время как 33.9% — на рабочих местах.

Подобная статистика предполагает возможность использования в качестве данных о состоянии пользователя для предупреждения состояния усталости характеристик, получаемых от внешних устройств, как на рабочих местах, так и в домашних условиях.

1.3 Методы определения усталости оператора с использованием устройств автоматизированного рабочего места

Внешние устройства (периферийные устройства) — устройства, предназначенные для внешней машинной обработки информации (в отличие от преобразований информации, осуществляемых центральным процессором) [12]. По роду выполняемых операций данный устройства подразделяются на группы [12]:

- устройства подготовки данных (занесения информации на промежуточные носители данных);
- устройства ввода (считывание информации и её преобразование в кодовую последовательность электрических сигналов, подлежащих передаче в центральный процессор;
- устройства вывода (регистрация результатов обработки информации или их отображения);
- устройства хранения больших объемов информации;
- устройства передачи информации на большие расстояния.

К внешним устройствам, которые являются устройствами ввода, то есть органами управления персональным компьютером, относят:

- клавиатуру;
- мышь;
- графический планшет;
- веб-камеру;
- микрофон;
- игровой манипулятор.

Из указанных выше устройств в рассмотрение не войдут:

- графический планшет;
- игровой манипулятор.

Данное решение связанно с тем, что использование подобного рода устройств указывает на особый род работы. В требованиях к разрабатываемому методу указывается возможность его использования для операторов автоматизированных рабочих мест в наиболее распространенных конфигурациях (например, для офиса), поэтому использование в решении поставленной задачи информации, получаемой с графических планшетов и игровых манипуляторов, не является целесообразным.

Согласно проведенным исследованиям [13] признаки голоса, клавиатурного почерка и характера работы исследуемого или контролируемого субъекта с компьютерной мышью содержат информацию о психофизиологических состояниях оператора: нормальное, усталость, опьянение, возбужденное, расслабленное (сонное).

1.3.1 Метод анализа клавиатурного почерка

Клавиатурный почерк — это подвид поведенческой подгруппы аутентификации по неотчуждаемым признакам. [14]

Клавиатурный почерк определяется по времени между нажатиями клавиш. При снятии биометрического шаблона клавиатурного почерка измеряют время нажатия двух, трех или четырех последовательных клавиш, сохраняют его и на основе полученных значений строят математические модели для сравнения шаблонов нескольких пользователей. [15]

Для системы входными данными будут два биометрических шаблона — эталонного и кандидата. Результат работы системы — рейтинг доверия к биометрическому шаблону кандидата, который является критерием схожести двух переданных шаблонов.

Различают два вида распознавания клавиатурного почерка: распознавание на статическом тексте (пароль или известная кодовая фраза), распознавание при вводе псевдослучайного текста. [14]

Математическая задача распознавания на фиксированном тексте может быть формализована. Для ее выполнения потребуется [14]:

- собрать информацию о времени между нажатиями соседних клавиш в тексте;
- сформировать из полученных данных вектор фиксированной размерности;
- по кластерной модели или другой модели сравнения двух векторов сравнить сформированный вектор и вектор-эталон для этого же текста от этого же пользователя.

Для задачи распознавания клавиатурного почерка при вводе псевдослучайного текста не существует надежных моделей формирования пользовательского шаблона и вычисления рейтинга. Время работы подобных систем не позволяет в реальном времени оценить ситуацию и выдает результат через десятки минут, а память, занимаемая векторами, лишь продолжает расти. [14]

Одним из численных показателей, которые определяют качество биометрической системы, является *ошибка первого рода (FRR, количество ложноотрицательных)* — это вероятность ложного отказа в доступе. Данная ошибка имеет место при возникновении повреждения рук пользователя или ненормального психофизического состояния человека (усталость, алкогольное или наркотическое опьянение, приступ гнева).[14]

Ошибка второго рода (FAR, количество ложноположительных) — это вероятность ложного допуска. Данная ошибка имеет место при ситуации, когда заданные возможные отклонения от допустимых значений при распознавании пользователя были заданы неверно, либо же когда нарушитель сумел скопировать метрики поведения пользователя и обойти систему контроля. [14]

Для проектируемой системы важнейшую роль играет ошибка первого рода. Данная ошибка способна позволить распознать состояние усталости оператора в случае единоличного пользования системой.

1.3.2 Метод анализа скорости печати и количества ошибок

Изменение психофизического состояния пользователя приводит к увеличению времени совершения простейших действий. Контроль времени выполнения данных действий способен помочь определить ухудшение состояния оператора автоматизированного рабочего места и своевременно сообщить пользователю о необходимости соблюдения гигиены труда. К контролю действий оператора относят [16]:

- скорость печати (ввода) — количество символов, введенных пользователем, разделенное на время, за которое данные символы были введены — уменьшение количества символов, введенного в единицу времени, может символизировать ухудшение состояния пользователя;
- динамика печати (ввода) — время между нажатиями клавиш и временем их удержания — увеличение интервала между нажатиями может символизировать ухудшение состояния пользователя;
- частота возникновения ошибок при печати (вводе) — увеличение количества совершаемых ошибок может символизировать ухудшение состояния пользователя.

1.3.3 Метод анализа траекторий перемещения курсора мыши

Компьютерная мышь используется для взаимодействия с оконным интерфейсом операционной системы и программ.

Особенности работы с мышью можно оценить, анализируя траектории передвижения курсора мыши по экрану между элементами интерфейса и среднее время выполнения передвижения. [13]

Оценка среднего времени перемещения курсора мыши между элементами интерфейса может быть выполнена с использованием адаптированным для данной задачи законом Фиттса [17]:

$$T = b \cdot \log_2 \left(\frac{D}{W} + 1 \right), \tag{1}$$

где b — величина, зависящая от типичной скорости движения курсора мыши (отношение средней скорости движения мыши по экрану, осуществляемого субъектом, к установленному в операционной системе коэффициенту чувствительности мыши);

D — дистанция перемещения курсора между элементами интерфейса (в пикселях);

W — ширина элемента интерфейса, к которому направляется курсор (в пикселях).

Адаптированный закон Фиттса связывает время, затраченное на движение к цели, с точностью движения и расстоянием до цели. Причем время перемещения курсора не совпадает с оценкой, выраженной в формуле (1), и отличается на некоторую величину, которую используют в качестве идентифицирующего признака. [18]

В качестве признаков могут быть использованы амплитуды первых десяти низкочастотных гармоник функции скорости перемещения курсора мыши по экрану:

$$V_{xy}(t) = \sqrt{\left(\left(x\left(t_{i+1}\right) - x\left(t_i\right) \right)^2 + \left(y\left(t_{i+1}\right) - y\left(t_i\right) \right)^2 \right)^2}, \tag{2}$$

где x, y — координаты курсора;

t_i — i-ый момент времени регистрации координат курсора (регистрация координат курсора зависит от производительности компьютера).

Разложение функции (2) производится с помощью быстрого преобразования Фурье, тем самым достигается нормирование участков пути курсора по

времени [18]. Каждый участок приводится к длительности в 0.5 секунд, амплитуды нормируются по энергии функции $V_{xy}(t)$, вычисляемой в соответствии с формулой:

$$E_s = \int_{\infty}^{-\infty} A^2(\omega)\, dt, \tag{3}$$

где $A(\omega)$ — амплитуды гармоники с частотой ω функции $V_{xy}(t)$.

Операция нормирования приводит траектории перемещений курсора между элементами интерфейса к единому масштабу. Аналогичные операции осуществляются по отношению к функциям координат курсора $x(t)$ и $y(t)$ с выполнением предварительного перевода данных функций в систему координат, где точкой начала координат является центр элемента интерфейса, к которому двигался курсов, причем ось абсцисс будет расположена в направлении к центру данного элемента. Данные действия обусловлены необходимостью избавиться от наклона линий, связывающих элементы интерфейса, относительно исходной координатной плоскости. [18]

1.3.4 Метод анализа использования координатного устройства мышь с использованием модели искусственной нейронной сети

В качестве главных факторов биометрической информации в данном методе выбираются параметры применения клавиатуры и мыши. [19]

Для сбора и хранения особенностей ввода управляющих последовательностей применяется логирование клавиатурного буфера машины. При любом использовании клавиатуры сохраняются задержки между нажатиями, продолжительности нажима отдельных клавиш, а также продолжительность набора устойчивого сочетания. В качестве сохраняемой информации об использовании мыши и иных координатных устройств фиксируются изменения положений, скорости и времени передвижения курсора. [19]

Анализ сохраненных данных проводится с использованием специальной искусственной нейронной сети. [19]

Основной задачей нейронной сети в данном методе является разбиение поступающих данных на два кластера: случаи санкционированного доступа и несанкционированного доступа. При этом исследуемый показатель не является статическим, в связи с чем нейронная сеть работает в динамическом режи-

ме, переобучаясь по мере изменения поведенческих особенностей пользователя. [19]

После обучения нейронной сети, с ее помощью исследуется поток данных о работе текущего пользователя. Из-за возможности изменения поведения ввода человека, с некоторой долей вероятности модель не может однозначно определить правомерность доступа. В подобных ситуациях требуется принять решение о том, следует ли переобучить модель в связи с изменением анализируемых параметров или запретить доступ к системе. Для решения данной проблемы используется метод нечеткого вывода Мамдани. [19]

1.3.5 Метод анализа внешнего состояния пользователя

Основные исследования в области использования видео-изображений для определения опасных состояний усталости проводятся для реализации систем распознавания усталости водителя.

Признаки состояний ослабленного внимания и усталости у водителя характеризуется следующими наблюдаемыми параметрами [20]:

- поворот головы влево/вправо по отношению к туловищу;
- наклон головы вперед относительно туловища;
- продолжительность моргания век;
- частота моргания век;
- степень открытости рта человека (признаки зевоты).

Работа системы строится на использовании технологий машинного обучения и предварительно обученных на наборах данных моделей. Детектирование опасных состояний в поведении повышает точность и полноту определения ослабленного внимания и усталости — накопление, анализ и обработка информации, поступающей в облачный сервис и формирующей историю вождения, приводят к созданию новых моделей поведения того или иного пользователя, вычислять характерные параметры, оказывающие влияние на эффективность работы схем распознавания опасных состояний. Таким образом высчитываются не только пороговые параметры для каждого водителя индивидуально, но и поддерживается надежность обновляемых параметров для новых пользователей, чьи параметры пока не известны. [20]

Применение рассматриваемого метода, в комплексе с уже рассмотренными, для определения состояния усталости оператора автоматизированного рабочего места потребует больших вычислительных ресурсов серверной части

программного комплекса. Кроме того, со стороны клиента потребуется беспрерывное соединение с сетью Интернет, увеличение ресурса оперативной памяти и современная веб-камера с приемлемым качеством фото- и видео-съемки.

1.3.6 Метод анализа речевых характеристик пользователя

Микрофон позволяет регистрировать аудиопоток, исходящий от пользователя и его окружения.

Исследования показали, что признаки голоса наилучшим образом позволяют распознавать усталость или расслабленное (сонное) состояние диктора. [13]

Симптоматика усталости заключается в проявлениях слабости, раздражительности, расстройствах сна и вегетативных нарушениях. [21]

Степень раздражительности может быть оценена по шкале Рэймонда Новако. Тест Новако содержит в себе 25 вопросов, ответы на которые позволяют определить показатель раздражительности. При интерпретации результатов показатель в 0–45 баллов определяет низкую степень раздражительности, 46–55 баллов — раздражительность ниже средней, 56–75 баллов — среднюю степень раздражительности, 76–85 баллов — раздражительность выше средней и 86–100 баллов — высокую степень раздражительности. [22]

Проявления слабости, расстройства сна и вегетативные нарушения могут быть установлены с использованием клинических методов.

Наиболее полную информацию о внутреннем психоэмоциональном состоянии человека может дать анализ его связной речи [23]:

- расстановка логических ударений;
- скорость произнесения слов;
- конструкция фразы;
- неуверенный или неверный подбор слов;
- обрывание фраз на полуслове;
- изменение слов;
- появление слов-паразитов;
- исчезновение пауз.

Разработки в сфере анализа напряжения голоса тестируются в судебной психологии для выявления лжи и обмана посредством обнаружения микротремора. Однако целесообразность и надежность использования данного метода до сих пор является предметом дискуссий в силу того, что успех в выявлении

лжи зависит от опыта эксперта. Кроме того, было доказано, что невиновные люди, чья невиновность оспаривается, проявляют не меньше стресса, чем виновные, увеличивая риск появления ложных срабатываний. Данные проблемы в практике привели к тому, что на сегодняшний день данные разработки рассматриваются как многообещающий инструмент обнаружения стресса, однако проблема межиндивидуальных различий пока еще остается нерешенной. [24]

1.3.7 Метод анализа виброакустических шумов при наборе текста или использовании мыши

Данный метод предполагает использование виброакустических датчиков, установленных непосредственно на рабочем столе пользователя, данные с которых вводятся в звуковую карту. [25]

Для идентификации оператора из записей виброакустического сигнала при наборе произвольного текста удаляются паузы между нажатиями и отпусканиями клавиш клавиатуры. Затем очищенный исследуемый сигнал разбивается на перекрывающиеся интервалы, длины которых определяются идентификацией оператора с минимальной погрешностью. Основное требование для векторов идентификации — устойчивость для всех пользователей в системе распознавания. Таким образом формируется база виброакустических сигналов, возникающих при наборе каждым из пользователей системы. [25]

Для идентификации оператора с использованием манипулятора мышь в качестве векторов признаков используются относительные длительности периодов активности пользователя, которые вычисляются на участках виброакустического сигнала, сопоставленных с записями в журнале событий мыши. [26]

Для каждого оператора обучается нейронная сеть для его идентификации и затем используется при работе системы. [25]

В качестве оптимизации формирования обучающей выборки предлагается использование алгоритма выделения фрагментов, которые содержат виброакустический сигнал нажатия или отпускания клавиш на основе ошибки линейного предсказания и последующего удаления единичных максимумов. [25]

1.4 Биофизические факторы, позволяющие определить усталость

1.4.1 Частота пульса и возраст сосудистой системы

Согласно исследованиям [27], при кратковременном стрессовом воздействии было выявлено три показателя, которые подвергаются изменениям: частота пульса, возраст сосудистой системы и индекс стресса.

Частота пульса, в среднем, до стрессового воздействия составляла 77.42 $\pm$ 4.12 ударов в минуту, после — 99.67 $\pm$ 5.54 ударов в минуту с уровнем достоверности $p = 0.004$. [27]

Возраст сосудистой системы — это параметр, определяющий биологический возраст индивида, то есть изношенность его организма. [28]

Данный параметр, в среднем, до стрессового воздействия был равен 29 $\pm$ 1.9 лет, после — 34.5 $\pm$ 1.69 лет с уровнем достоверности $p = 0.041$. [27]

1.4.2 Индекс Баевского

Индекс стресса (индекс Баевского) характеризует вариабельность сердечного ритма и состояние центров регуляции сердечно-сосудистой системы. Норма данного индекса для человека находится в диапазоне от 50 до 150 единиц. Увеличение данного показателя до значений от 150 до 500 может говорить о наличии физических нагрузок или хронической усталости. Увеличение индекса до значений в диапазоне от 500 до 900 говорит о наличии существенного психологического и эмоционального стресса, либо о психофизиологическом переутомлении или стенокардии. Превышение индексом значения 900 единиц свидетельствует о нарушении регуляторных механизмов или предынфарктном состоянии. [29]

Индекс Баевского может быть рассчитан по следующей формуле [30]:

$$I = \frac{AM_o}{2M_o\Delta X}, \tag{4}$$

где M_o — это наиболее часто встречающееся в динамическом ряде значение кардиоинтервала (мода);

AM_o — это число кардиоинтервалов, соответствующих значению моды (амплитуда моды), в процентах от объему выборки;

ΔX — вариационный размах среднего значения продолжительности сердечного цикла, показатель деятельности контура автономной регуляции ритма сердца, который целиком связан с дыхательными колебаниями тонуса блуждающих нервов, в секундах.

$M_o\Delta X$ — отражает степень вариативности значений кардиоинтервалов в исследуемом динамическом ряду.

В исследовании данный параметр, в среднем, до стрессового воздействия был равен 75.27 ± 11.84 единиц, после — 334.55 единиц с уровнем достоверно-

сти $p = 0.001$.

Частота сердечных сокращений является включением для индекса стресса, из чего следует факт того, что в системе отсутствует потребность в анализе данной характеристики отдельно. Возраст сосудистой системы — параметр, определяемый сложной медицинской техникой. В силу изложенных фактов, в дальнейшем в систему в качестве характеристики для определения стресса войдет лишь индекс Баевского.

В качестве устройства для снятия индекса, основанного на вариабельности сердечного ритма, могут быть предложены смарт-часы с поддержкой данной функции. Модуль программного обеспечения, сохраняющий данные характеристики, должен записывать ее значения в базу данных с использованием интерфейса, предоставляемым выбранной моделью.

1.5 Формализация требований к методу систематического распознавания усталости на автоматизированном рабочем месте

На текущий момент не каждое автоматизированное рабочее место может включать в себя все рассмотренные внешние устройства взаимодействия пользователя с системой. Данное обстоятельство указывает на требование возможности работы системы в условиях отсутствия тех или иных периферийных устройств.

К требованиям также отнесена потребность в сохранении ресурсов персонального компьютера автоматизированного рабочего места в силу отсутствия стандартизации конфигурации технической составляющей компьютера.

Важно отметить, что в системе требуется разграничение личностей пользователей в силу индивидуальности характеристик каждого.

Вывод

Были рассмотрены понятия усталости, хронической усталости, стресса и профессионального стресса.

Была формализована цель разрабатываемого метода: предупреждение состояния усталости пользователя на основании данных, получаемых от внешних устройств, как на рабочих местах, так и в домашних условиях.

Были рассмотрены методы анализа:

- клавиатурного почерка;
- скорости печати и количества ошибок;
- использования координатного устройства мышь с использованием мо-

дели искусственной нейронной сети;

- внешнего состояния пользователя;
- речевых характеристик пользователя;
- виброакустических шумов при наборе текста или использовании мыши.

Среди биофизических факторов, позволяющих определить усталость оператора, были выделены: частота пульса и возраст сосудистой системы и индекс Баевского.

Формализация требований к методу систематического распознавания усталости на автоматизированном рабочем месте определила требование возможности работы системы в условиях отсутствия тех или иных периферийных устройств. Также в качестве требования была отнесена потребность в сохранении ресурсов персонального компьютера в силу отсутствия стандартизации конфигурации технической составляющей на рабочем месте. Была указана важность разграничения личностей пользователей.

2 Конструкторский раздел

В данном разделе определяются включаемые в метод характеристики, описывается метод систематического распознавания усталости на автоматизированном рабочем месте.

Рассматривается формат и метод сбора данных, предоставляемых пользователем, приведены диаграммы вариантов использования системы и сервера хранилища данных, схема работы сервера. Также представлены IDEF0 диаграмма задачи определения усталости оператора автоматизированного рабочего места, диаграмма “сущность-связь” в нотации Чена.

Приведены описания алгоритмов кластеризации.

2.1 Включаемые в метод характеристики, получаемые с использованием периферийных устройств

При проведении анализа метода распознавания усталости с использованием веб-камеры было указано, что применение данного метода может потребовать больших вычислительных ресурсов серверной части программного комплекса, а также постоянное соединение с сетью Интернет со стороны клиента. Данные факторы указывают на то, что метод не отвечает формализованным требованиям к методу систематического распознавания усталости на автоматизированном рабочем месте.

При проведении анализа метода распознавания усталости и стресса с использованием микрофона стало известно, что данный метод позволяет определить лишь проявления слабости, которые не могут однозначно указывать на наступление стадии истощения, адаптации или тревоги.

Для определения усталости будут использоваться устройства клавиатура (скорость печати) и мышь (скорость передвижения курсора между двумя нажатиями).

2.2 Метод систематического распознавания усталости на автоматизированном рабочем месте

Метод систематического распознавания усталости на автоматизированном рабочем месте включает в себя:

- хранение и анализ данных, получаемых от клавиатуры — для определения усталости используется нечеткая кластеризация;
- хранение и анализ данных, получаемых от мыши — для определения

усталости используется нечеткая кластеризация.

Определение нечетких кластеров происходит на этапе синхронизации поступающих данных о действиях пользователя с данными о скорости его реакции в отдельные моменты времени. В дальнейшем данная модель используется до ее актуализации. Система актуализирует центры кластеров в случае, если будет определено, что имеют место ложные срабатывания и “эталонные” патерны поведения объекта изменились.

2.3 Формат и метод сбора данных, предоставляемых оператором автоматизированного рабочего места

В качестве данных, характеризующих действия оператора с клавиатурой и мышью, выступают нажатия клавиш данных устройств.

Каждое нажатие на клавишу клавиатуры характеризуется двумя полями:

- наименование нажатой клавиши (в случае литеры — литера, в случае специальных клавиш – полное наименование);
- временная метка.

Каждое нажатие на клавишу мыши характеризуется четырьмя полями:

- координата X, в которой было совершено действие;
- координата Y, в которой было совершено действие;
- номер нажатой кнопки мыши;
- временная метка.

Для определения центров нечетких кластеров используются результаты теста на реакцию. Данный тест включает в себя появление некоторого элемента на экране и регистрацию времени, за которое оператор среагировал на его появление и нажал на него. При следующем появлении элемент не изменяет своей формы или положения, время его появления определяется случайным образом в интервале от двух до десяти секунд. Объекту предоставляется десять попыток, в результате чего среднее время реакции принимается за текущий показатель времени реакции. Тест проводится каждые 10 минут.

Каждый тест на реакцию характеризуется двумя полями:

- значение времени реакции;
- временная метка.

Может быть использован как локальный, так и удаленный метод хранения данных. Отличие заключается лишь в расположении хранилища данных. Первично данные должны записываться в логирующие файлы, которые в даль-

нейшем могут быть направлены в базу данных и удалены из локального хранилища.

Метод включает в себя логирование прерываний, поступающих с клавиатуры и мыши, включающих необходимую информацию о действиях пользователя, а также результатов тестов на реакцию.

2.4 Декомпозиция системы

На рисунке 2.1 представлена диаграмма вариантов использования системы распознавания усталости.

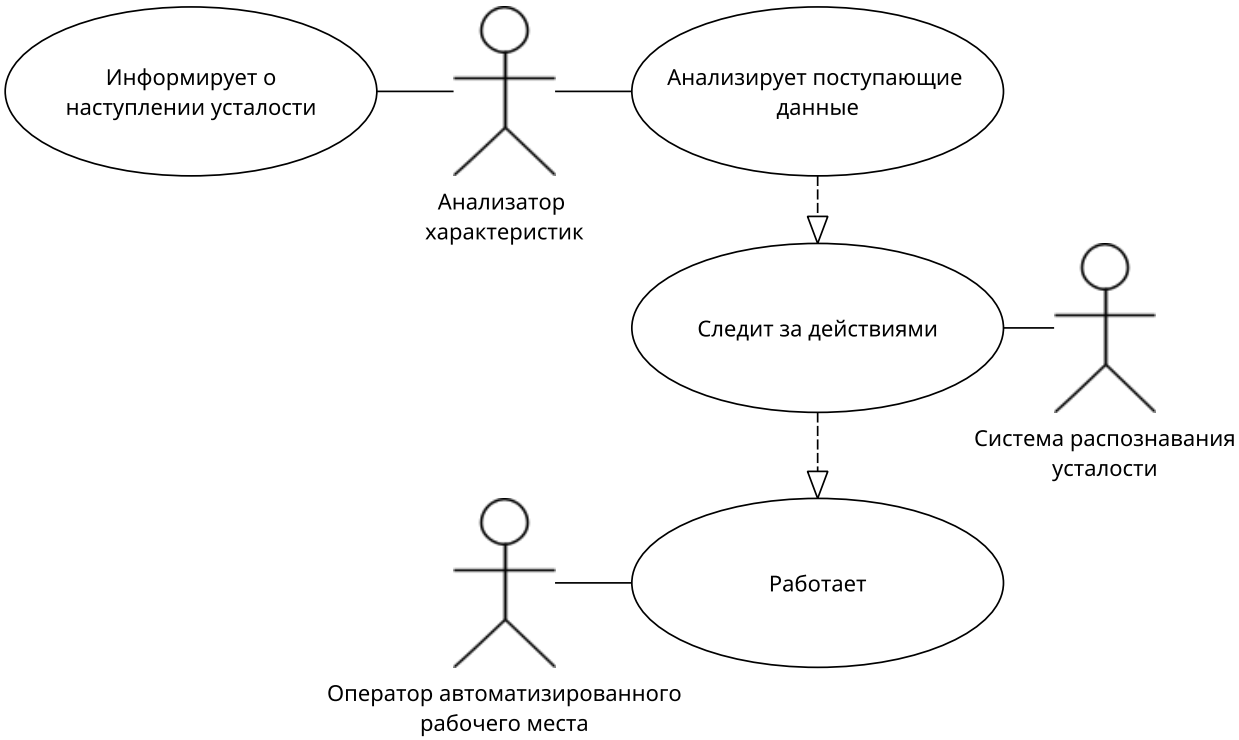

Рисунок 2.1 – Диаграмма вариантов использования системы распознавания усталости.

В системе определены 3 роли: система распознавания усталости, анализатор характеристик и оператор автоматизированного рабочего места.

Роль системы распознавания усталости заключается в сборе и систематизации поступающей информации от оператора. Данная часть системы не анализирует информацию и определяется в качестве медиатора для базы данных и программного обеспечения снятия данных.

Анализатор поступающих характеристик строит нечеткие кластеры по первичным данным, принимаемым за эталонные, а также отвечает за их актуализацию. Также данная часть системы отвечает за информирование о наступ-

лении усталости.

На рисунке 2.2 представлена диаграмма вариантов использования сервера для пользователя.

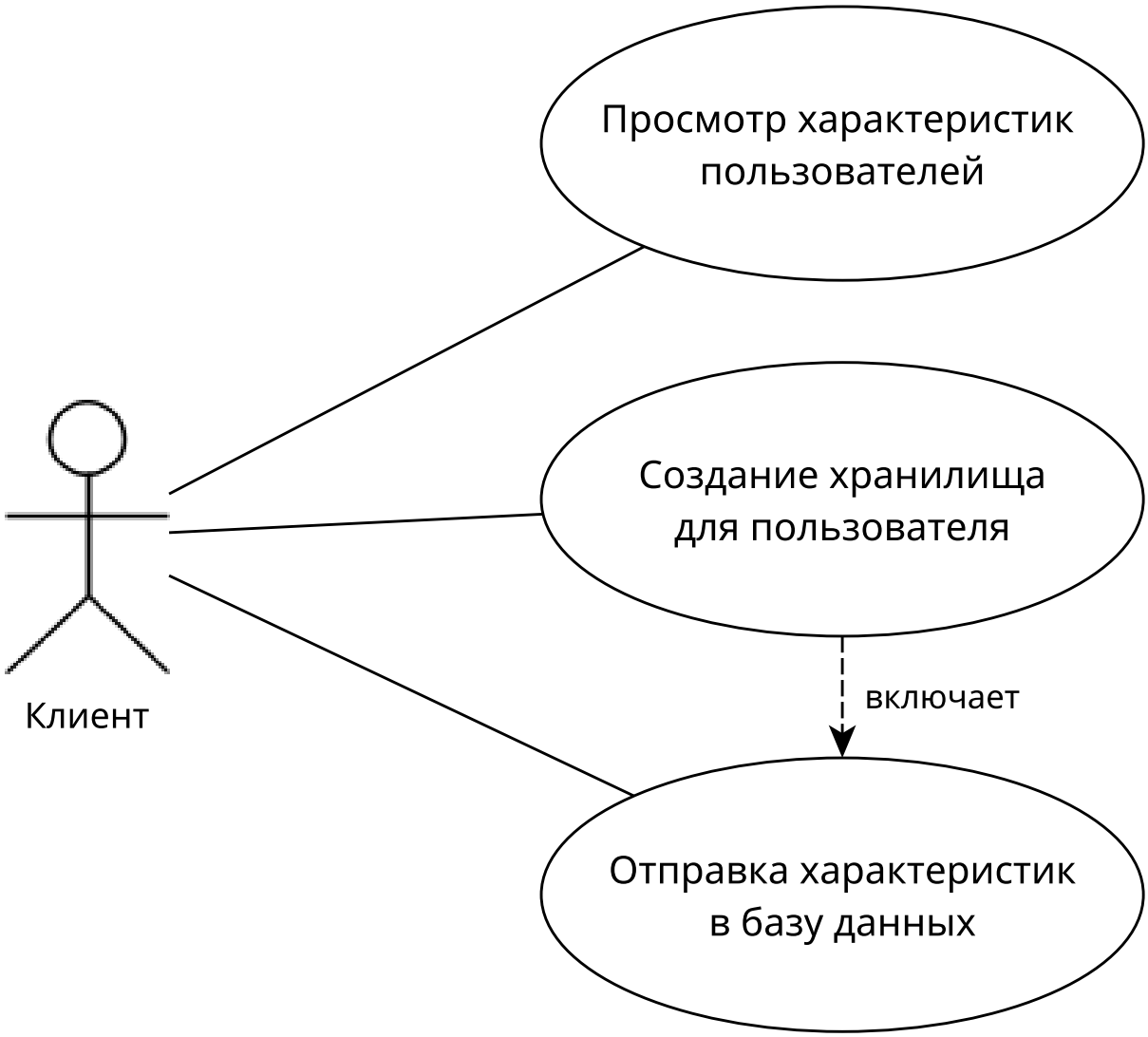

Рисунок 2.2 – Диаграмма вариантов использования сервера для пользователя.

На рисунке 2.3 представлена диаграмма вариантов использования сервера для репозитория.

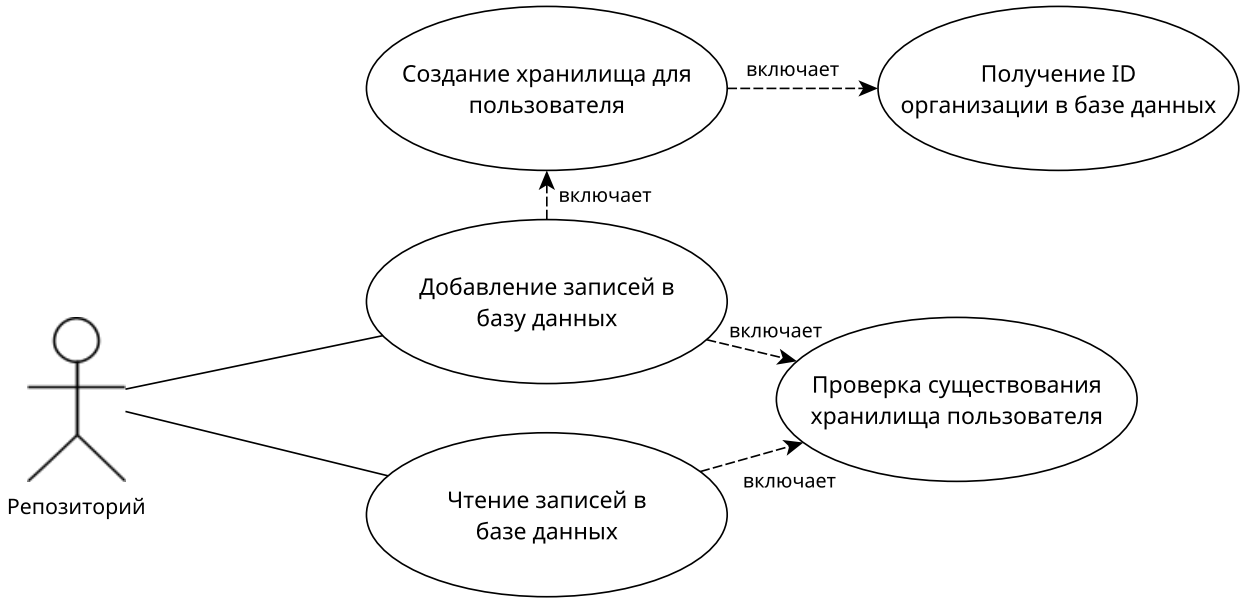

Рисунок 2.3 – Диаграмма вариантов использования сервера для репозитория.

2.5 Схема работы сервера

На рисунке 2.4 представлена схема работы сервера и алгоритма обработки запросов пользователей.

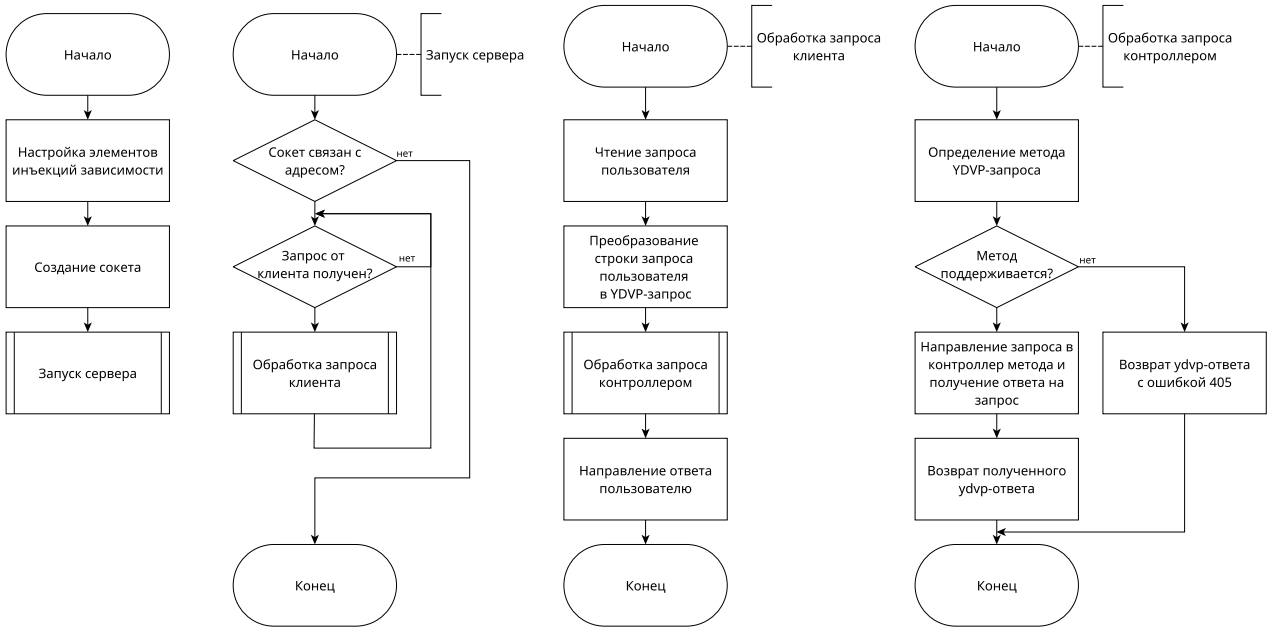

Рисунок 2.4 – Схема работы сервера.

На рисунках 2.5–2.6 предоставлена диаграмма IDEF0 задачи определения усталости оператора автоматизированного рабочего места.

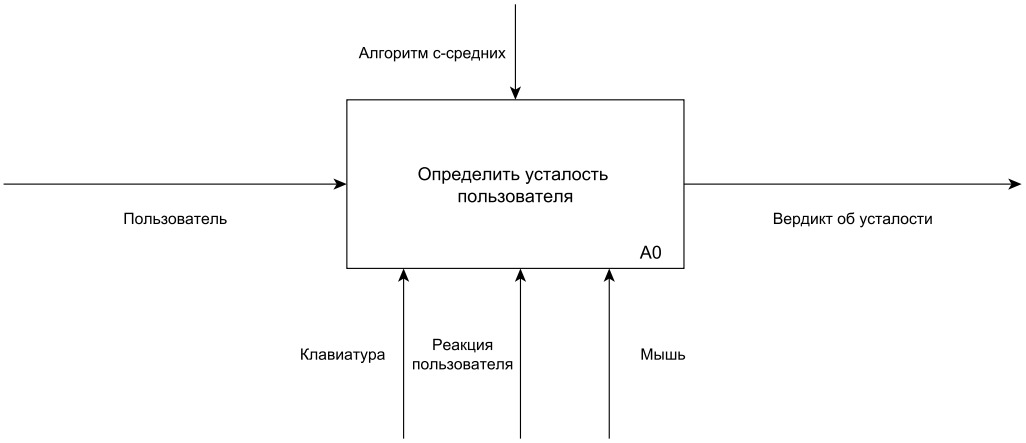

Рисунок 2.5 – IDEF0 диаграмма уровня A0.

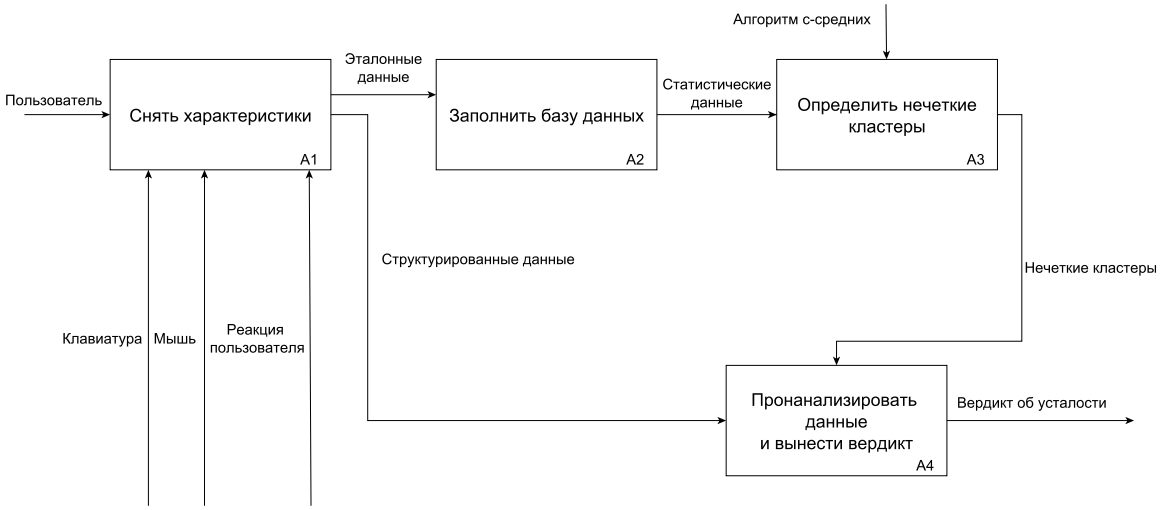

Рисунок 2.6 – IDEF0 диаграмма уровня A1-A3.

На рисунке 2.7 предоставлена диаграмма “сущность-связь” в нотации Чена.

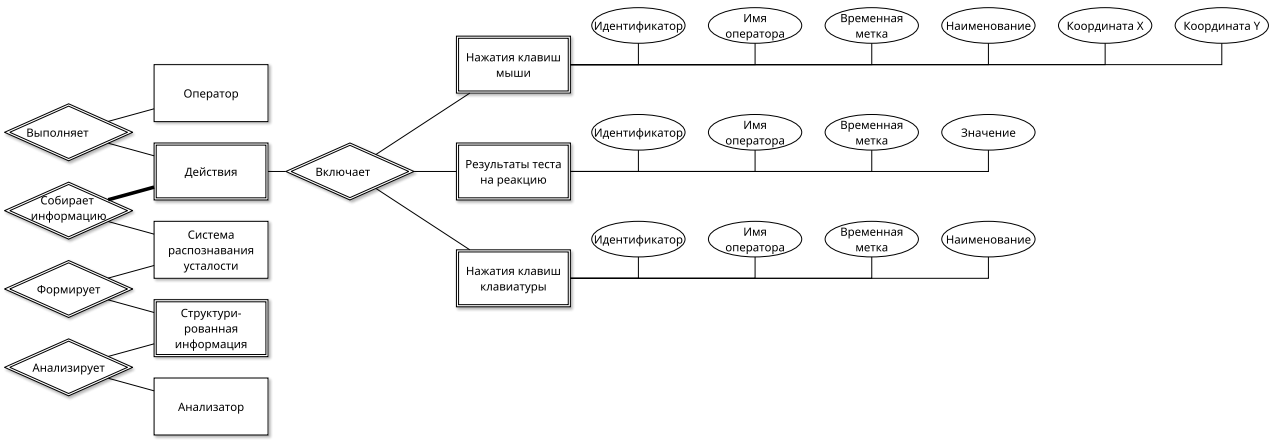

Рисунок 2.7 – Диаграмма “сущность-связь” в нотации Чена.

2.6 Алгоритмы кластеризации

Задачей кластеризации является объединение в группы объектов, схожих по некоторому признаку. Данная задача включает в себя разбиение множества объектов, называемых кластерами. [31]

Одним из главных отличий кластеризации от классификации является факт того, что перечень групп четко не задан и определяется в процессе работы алгоритма. [31]

Этапы кластерного анализа включают в себя [31]:

1) Отбор выборки объектов;
2) Определение множества переменных, по которым будут оцениваться объекты в выборке;
3) Вычисление значений меры сходства между объектами;
4) Применение метода кластерного анализа;
5) Представление результатов анализа.

Выделяют две основные классификации алгоритмов кластеризации: иерархические и плоские и четкие и нечеткие. [31]

Иерархические алгоритмы строят не одно разбиение выборки на непересекающиеся кластеры, а систему вложенных разбиений. Плоские алгоритмы Строят разбиение объектов на кластеры. [31]

Четкие алгоритмы каждому объекту выборки ставят в соответствие номер кластера, то есть каждый объект определенно относится к одному из выделенных кластеров. Нечеткие алгоритмы каждому объекту ставят в соответствие

набор вещественных значений, показывающих степень отношения объекта к кластерам, каждый объект относится к каждому кластеру с некоторой вероятностью. [31]

2.6.1 Алгоритмы иерархической кластеризации

Среди алгоритмов иерархической кластеризации выделяются два типа: нисходящие и восходящие алгоритмы. В первом типе в начале все объекты помещаются в один кластер, который затем разбивается на все более мелкие кластеры. Восходящие алгоритмы в начале работы помещают каждый объект в отдельный кластер, а затем объединяют кластеры во все более крупные, пока все объекты выборки не будут содержаться в одно кластере. [31]

2.6.2 Алгоритмы квадратичной ошибки

Задачу кластеризации можно рассматривать как построение оптимального разбиения объектов на группы. При этом оптимальность может быть определена как требование минимизации среднеквадратической ошибки разбиения [31]:

$$e^2(X, L) = \sum_{j=1}^{K} \sum_{i=1}^{n_j} \left\| x_i^{(j)} - c_j \right\|^2, \tag{5}$$

где c_j — “центр масс” кластера j (точка со средними значениями характеристик для данного кластера),

K — количество кластеров,

n — количество точек.

Алгоритмы квадратичной ошибки относятся к типу плоских алгоритмов. Самые распространенным алгоритмом этой категории является метод k средних. Этот алгоритм строит заданное число кластеров, расположенных как можно дальше друг от друга. Алгоритм описывается следующими действиями [31]:

1) Случайно выбрать k точек, являющихся начальными “центрами масс” кластеров;
2) Отнести каждый объект к кластеру с ближайшим “центром масс”;
3) Пересчитать “центры масс” кластеров согласно их текущему составу;
4) Если критерий остановки алгоритма не удовлетворен, вернуться к п. 2.

В качестве критерия остановки алгоритма выбирают минимальное изменение среднеквадратической ошибки. Также возможность останавливаться, ес-

ли на шаге 2 не было объектов, переместившихся из кластера в кластер. [31]

Недостаток данного алгоритма заключается в том, что необходимо задавать количество кластеров для разбиения. [31]

2.6.3 Нечеткие алгоритмы

Наиболее популярным алгоритмом нечеткой кластеризации является алгоритм c-средних. Он представляет собой модификацию метода k-средних. [31]

Алгоритм может быть описан следующей последовательностью действий [31]:

1) Выбрать начальное нечеткое разбиение n объектов на k кластеров путем выбора матрицы принадлежности U размера (n, k);
2) Используя матрицу U, найти значение критерия нечеткой ошибки:

$$E^2(X, L) - \sum_{i=1}^{N} \sum_{j=1}^{K} U_{ik} \left\| x_i^{(k))} - c_k \right\|^2, \tag{6}$$

где c_k — "центр масс" нечеткого кластера k: $c_k = \sum_{i=1}^{N} U_{ik} x_i$,
K — количество кластеров,
N — количество точек.

3) Перегруппировать объекты с целью уменьшения этого значения критерия нечеткой ошибки;
4) Возвращаться в п. 2 до тех пор, пока изменения матрицы U не станут незначительными.

Данный алгоритм может не подойти, если заранее неизвестно число кластеров, либо необходимо однозначно отнести каждый объект к одному кластеру. [31]

Вычислительная сложность для алгоритмов k-средних и c-средних одинакова и равна $O(n \cdot k \cdot l)$, где k — число кластеров, l — число итераций, а n — количество точек для кластеризации. [31]

Вывод

Были определены включаемые в метод характеристики: скорость печати и скорость перемещения курсора между двумя нажатиями.

Был описан метод систематического распознавания усталости на автоматизированном рабочем месте, он включил в себя следующие задачи хранения и анализа данных, получаемых от клавиатуры и мыши. Было определено, что вычисление центров нечетких кластеров происходит на этапе синхронизации

поступающих данных о действиях пользователя с данными о скорости его реакции в отдельные моменты времени. Определенная модель используется до ее актуализации, которая происходит в случае, если будет определено, что имеют место ложные срабатывания и “эталонные” паттерны поведения объекта изменились.

Был описан формат и метод сбора данных, предоставляемых оператором.

Было определено, что каждое нажатие на клавишу клавиатуры характеризуется наименованием нажатой клавиши и временной меткой. Также было определено, что каждое нажатие на клавишу мыши характеризуется координатами действия, номером клавиши и временной меткой.

Были приведены диаграммы вариантов использования. Для системы распознавания усталости было определено три действующих лица: анализатор характеристик, система распознавания усталости и оператор. Для сервера было определено два действующих лица: клиент и репозиторий.

Схема работы сервера позволила определить особенности удаленной системы хранения данных.

Была приведена IDEF0 диаграмма, декомпозирована главная задача метода — определение усталости пользователя.

Диаграмма “сущность-связь” в нотации Чена позволила на абстрактном уровне описать систему распознавания, а также описать поля снимаемых характеристик в базе данных.

Были рассмотрены алгоритмы иерархической кластеризации, а также алгоритмы квадратичной ошибки, и нечеткие алгоритмы.

3 Технологический раздел

В данном разделе определяются средства реализации программного обеспечения, аргументируется выбор используемой СУБД и алгоритма кластеризации. Описываются данные для проведения кластеризации и приводятся сведения о модулях.

3.1 Средства реализации программного обеспечения

При написании программного продукта был использован язык программирования Kotlin [32].

Данный выбор обусловлен следующими факторами:

- возможность запуска программного кода на любом устройстве, поддерживающем Java Virtual Machine;
- большое количество актуализируемой справочной литературы, связанной как я языком программирования Java, так и Kotlin;
- возможность интеграции программного кода в приложения для операционной системы Android.

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

3.2 Выбор СУБД

3.2.1 Базы данных временных рядов

Базы данных временных рядов отличаются от статических баз данных тем, что содержат записи, в которых некоторые из атрибутов ассоциируются с временными метками. В качестве таких записей могут выступать данные мониторинга, биржевые данные о торгах или транзакции продаж. [33]

InfluxDB InfluxDB - это база данных временных рядов, предназначенная для обработки высокой нагрузки записи и запросов.

Основным назначением является хранение больших объемов данных с метками времени. Например, данные мониторинга, метрики приложений и данные датчиков интернета вещей.

В традиционной реляционной базе данных данные хранятся до тех пор, пока не будет принято решение об их удалении. Учитывая сценарии использования баз данных временных рядов, можно не хранить данные слишком долго:

это или дорого, или они со временем теряют актуальность. [34]

Системы, подобные InfluxDB, могут удалять данные спустя определенное время, используя концепцию, называемую политикой хранения. Также поддерживается функционал отправки непрерывных запросов к оперативным данным для выполнения определенных операций. [34]

OpenTSDB OpenTSDB включает в себя “демона” временных рядов, а также набор утилит командной строки. Взаимодействие с OpenTSDB в первую очередь достигается путем запуска одного или нескольких независимых “демонов”.

“Демон” использует базу данных с открытым исходным кодом HBase или службу Google Bigtable для хранения и получения данных временных рядов. Схема данных высоко оптимизирована для быстрого объединения аналогичных временных рядов, чтобы минимизировать пространство хранения. Пользователям никогда не требуется прямой доступ к базовому хранилищу. Можно общаться с “демоном” через протокол telnet, HTTP API или простой встроенный графический интерфейс.

3.2.2 Реляционные базы данных

Реляционная база данных — это организованный по реляционной модели набор таблиц, в которых каждая ячейка этих таблиц имеет некоторое соответствующее описание. [35]

Использование реляционной модели предполагает возможность идентификации элементов по совокупности уникальных идентификаторов: имя столбца, первичный ключ. Для построения логической связи между строками и ячейками разных таблиц используются внешние ключи. [35]

Среди подобных СУБД, основанных на реляционных базах данных, пользуются популярностью Oracle и PostgreSQL.

Каждая из указанных СУБД имеет некоторые отличительные особенности. Так, например, PostgreSQL поддерживает вставки кода, написанного на языке программирования Python, в тело процедуры. Однако выделить среди данных особенностей важных для данной работы не представляется возможным.

Вывод

Базы данных временных рядов являются наиболее подходящими для решаемой задачи, так как они нацелены на хранение, извлечение и анализ большого количества статистических данных, в которых имеются временные метки.

Для организации хранения данных будет использоваться СУБД InfluxDB, так как она является одной из самых популярных среди известных баз данных временных рядов, а также по той причине, что поддержка данной СУБД все еще не прекращена на сегодняшний день.

3.3 Алгоритм и данные для кластеризации

В качестве используемого алгоритма кластеризации был выбран метод с-средних в силу того, что число кластеров заранее известно, а также задача рассматривает установку соответствия некоторого объекта (например, значения скорости печати) набору вещественных значений, показывающих степень отношения объекта к кластерам.

В качестве данных для кластеризации используются действия оператора автоматизированного рабочего места, производимые с использованием клавиатуры и мыши. Данные действия логируются, а затем направляются в базу данных для возможности переноса определенной модели поведения на другое автоматизированное место.

Действия пользователя, участвующие в построении модели, соотносятся с временем реакции, которое фиксируется каждые 10 минут, причем по времени реакции определяется, в каком состоянии в текущий момент времени находится организм оператора.

3.4 Сведения о модулях

Программное обеспечение состоит из модулей логирования действий оператора, обработки и анализа данных. Также определен модуль, который позволяет развернуть сервер для хранения данных пользователей с использованием базы данных InfluxDB.

3.4.1 Модуль логирования действий оператора

Данный модуль предназначен для записи информации о действиях оператора.

Библиотеки, используемые в модуле:

- Java Swing [36] — библиотека легковесных компонентов для реализации оконного интерфейса приложения;

- JNativeHook [37] — библиотека, предоставляющая средства перехвата прерываний, поступающих от клавиатуры и мыши.

Модуль состоит из четырех пакетов.

Пакет window

Данный пакет включает в себя абстрактный класс Window, предоставляющий родительский класс с определенными свойствами для всех окон реализуемого программного обеспечения. Реализация приведена в листинге 2 (приложение А, с. 62).

Пакет bigBrother

Данный пакет включает в себя класс BigBrotherWindow, который является реализацией класса Window и определяет функционал главного экрана приложения. Реализация приведена в листинге 3 (приложение А, с. 62).

Пакет loggers

Данный пакет включает в себя пакеты реализаций логирующих классов: KeyLogger (нажатия на клавиши клавиатуры), MouseLogger (нажатия на клавиши и движения), ReactionLogger (результаты пройденных тестов на реакцию). Реализация классов приведена в листингах 5 – 8 (приложение А, с. 62).

В пакете реализован класс ReactionTestWindow, предоставляющий интерфейс и логику определения реакции пользователя по нажатию на кнопку, появляющуюся в случайные моменты времени (от 2 до 10 секунд). Код приведен в листинге 9 (приложение А, с. 62).

Каждый логирующий класс локально создает текстовый файл, в который записывает в определенном формате собранные данные. Для исключения попыток изменения файла конкурирующими потоками в каждом из них представлена реализация очереди записи, которая переносится в файл при достижения размера в сотню записей либо по завершению работы приложения.

Пакет random

Данный пакет включает в себя класс, реализованный по шаблону “Одиночка”, предоставляющий доступ к классу Random, инициализированного отложено. Данный класс используется для получения данных о реакции пользователя. Реализация класса представлена в листинге 4 (приложение А, с. 62).

Также в модуле определен файл main.kt, который является точкой входа в приложения. Код файла представлен в листинге 1 (приложение А, с. 62).

На рисунке 3.1 представлена диаграмма классов модуля.

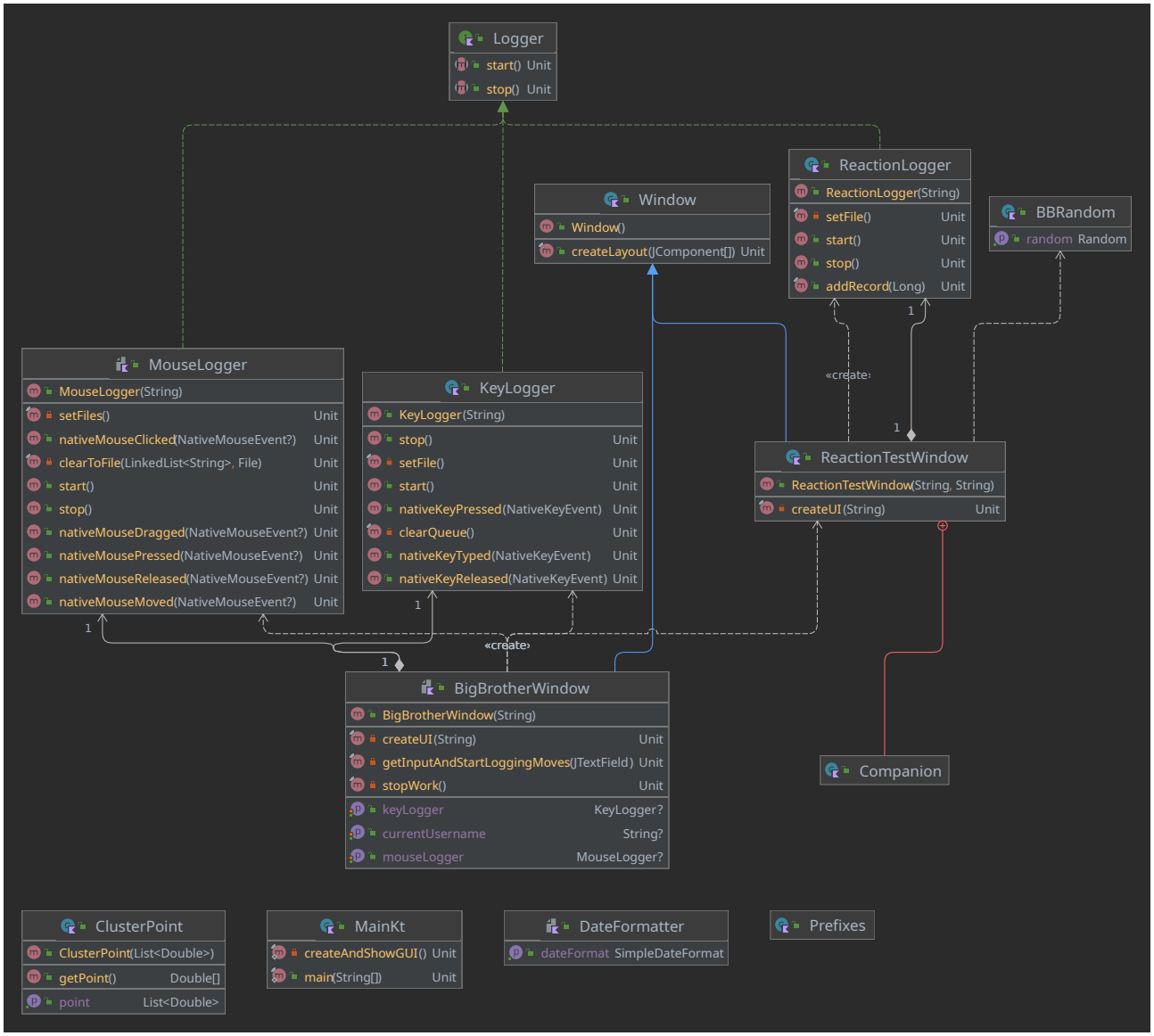

Рисунок 3.1 – Диаграмма классов модуля логирования.

Пакет window

Данный пакет включает в себя абстрактный класс Window, представляющий родительский класс с определенными свойствами для всех окон реализуемого программного обеспечения. Реализация приведена в листинге 2 (приложение А, с. 62).

3.4.2 Модуль обработки данных

Модуль синтаксического анализа данных предоставляет функции приведения записанных текстовых данных к определенным классам моделей данных.

Модуль обработки данных включает в себя два пакета: синтаксического анализа и приведения.

Пакет bbParser.models

Данный пакет включает в себя модели данных.

В листинге 10 (приложение А, с. 62) представлена реализация абстрактного класса модели, в листингах 11 – 13 (приложение А, с. 62) представлены примеры конкретных моделей.

Пакет bbParser.parsers

Данный пакет включает в себя синтаксические анализаторы для получаемых текстовых данных.

В листинге 14 (приложение А, с. 62) представлена реализация абстрактного класса синтаксического анализатора. В листинге 15 (приложение А, с. 62) представлен пример конкретного анализатора.

Пакет bbConverter

Данный пакет отвечает за получение требуемых характеристик по полученным данных, например, скорости печати.

В листинге 18 (приложение А, с. 62) представлена реализация абстрактного класса. В листинге 20 (приложение А, с. 62) и 19 (приложение А, с. 62) — примеры преобразователей.

На рисунке 3.2 представлена диаграмма классов модуля.

3.4.3 Модуль анализа данных

Данный модуль предназначен для анализа поступающей от оператора информации: кластеризации данных для модели и непосредственно текущей оценки усталости оператора.

Единственной библиотекой, используемой в модуле является The Commons Mathematics Library [38], из которой была взята реализация алгоритма кластеризации с-средних.

Пакет analyze.clusterization

Данный пакет включает в себя утилиты для кластеризации данных, в нем представлен абстрактный класс Clusterer и его реализация BbClusterer, предоставляющий функционал определения нечетких кластеров по переданным

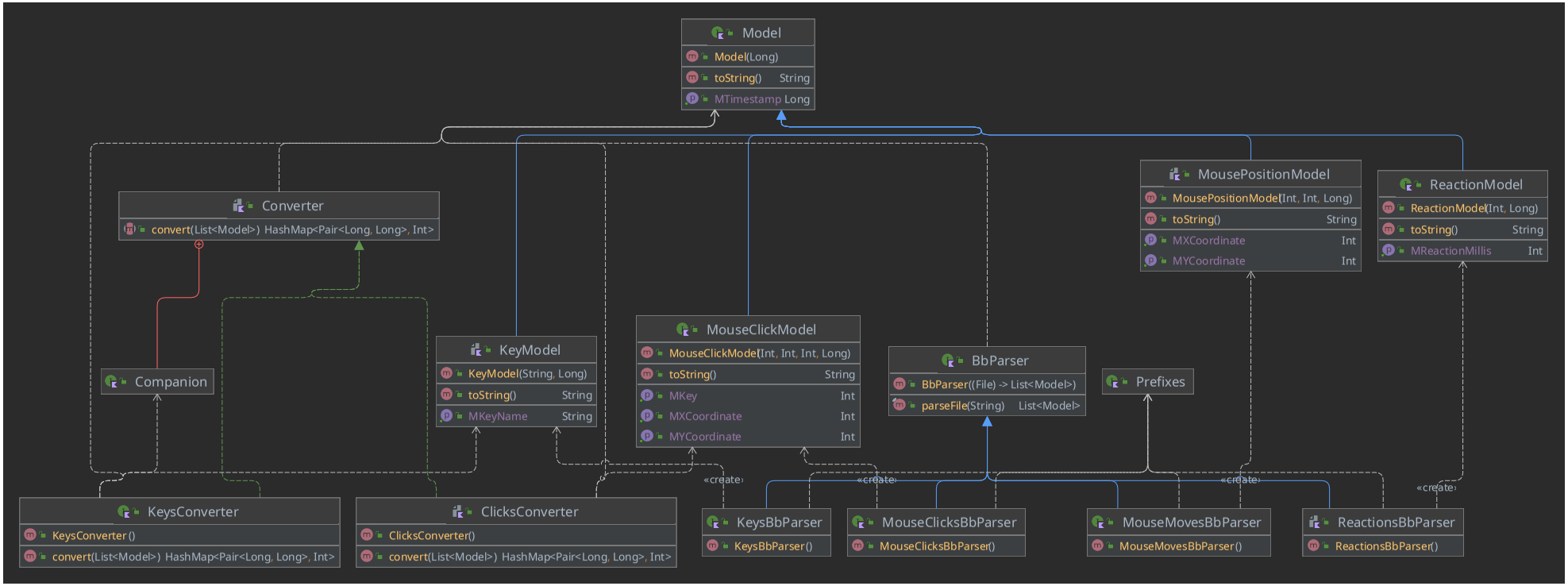

Рисунок 3.2 – Диаграмма классов модуля обработки данных.

данным. Также в пакете приведен класс точки для кластеризации, используемая библиотекой The Commons Mathematics Library. Реализация приведена в листингах 22 — 24 (приложение А, с. 62).

Пакет analyze.analyzers Данный пакет включает в себя анализаторы, позволяющие построить модель и оценивать состояние оператора с использованием различных представлений данных и их методов обработки. В листингах 25 (приложение А, с. 62) и 26 (приложение А, с. 62) представлены абстрактный класс анализатора и его реализация для работы с файлами.

Пакет analyze.fileAnalyzer Данный пакет предоставляет реализацию класса, позволяющего полноценно создать модель по заданным файлам и определить состояния оператора. Реализация представлена в листинге 27 (приложение А, с. 62).

На рисунке 3.3 представлена диаграмма классов модуля.

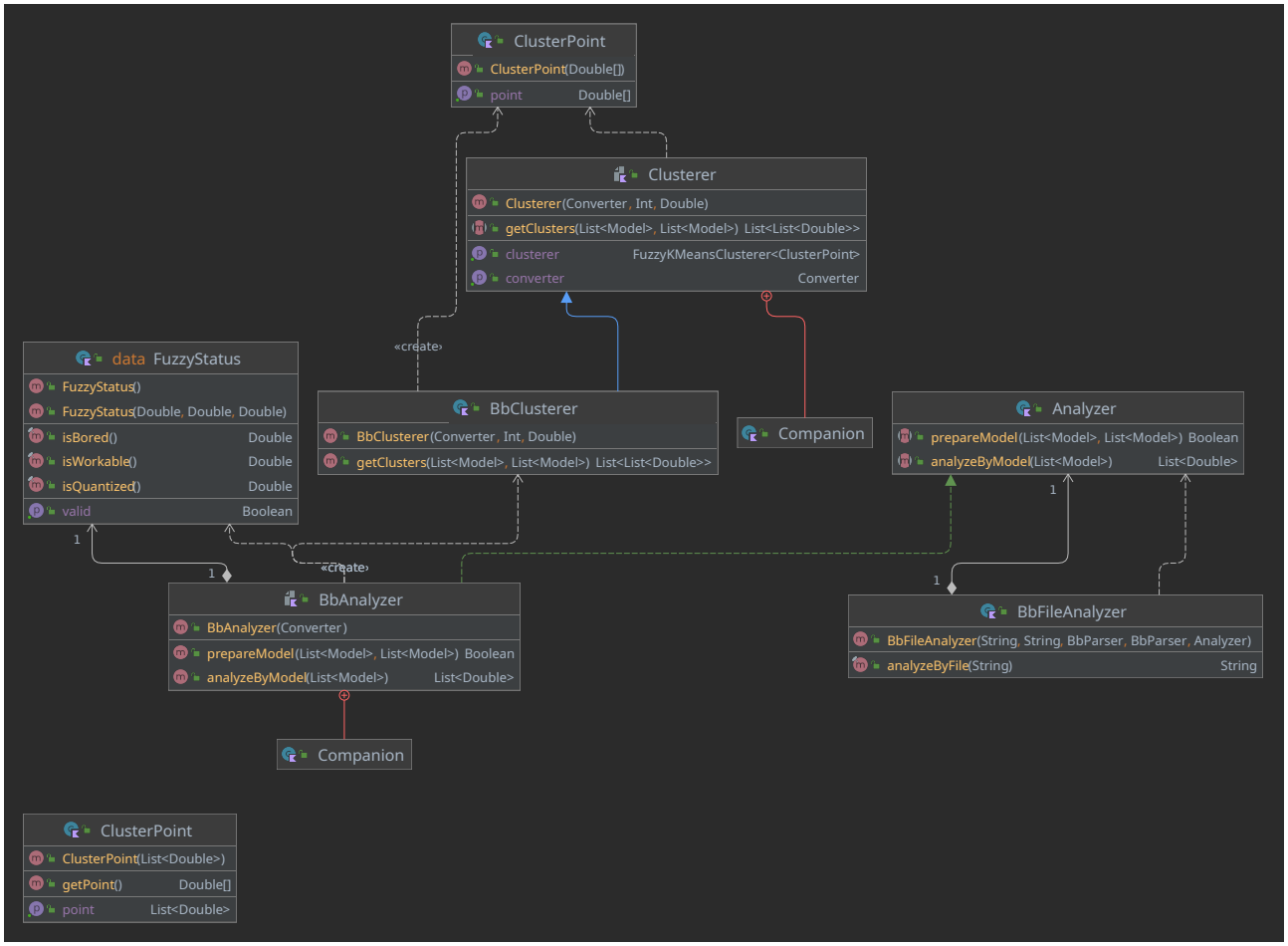

Рисунок 3.3 – Диаграмма классов модуля анализа данных.

3.4.4 Модуль серверного приложения

В данном модуле описана серверная составляющая хранилища данных, полученных от оператора. Главным назначением является хранение данных, позволяющих построить модель для пользователя в случае их утраты в локальном хранилище.

Данный модуль логически разделен на следующие части:

- пакет доступа к данным;
- пакет бизнес-логики;
- пакет реализации протокола;
- пакет клиента;
- пакет сервера.

Пакет доступа к данным

Данный пакет включает в себя реализацию двух классов: CharRepositoryImpl, основанного на шаблоне проектирования “Репозиторий”, и InfluxDAO, основаного на шаблоне проектирования “Объект доступа к данным”. Объект доступа к данным позволяет сделать репозиторий независимым от реализации исполнения запросов к базе данных. Данный объект использует InfluxDB-client-kotlin [39] для запросов на внесение и чтение записей, однако отдельный функционал реализован через отправку HTTP-запросов напрямую к серверу InfluxDB с использованием OkHttp3 [40].

Пакет бизнес-логики

Данный пакет включает в себя множество сущностей, фигурирующих между слоями клиент-серверной архитектуры.

Пакет реализации протокола

Данный пакет включает в себя класс YDVP, который может представлять собой YDVP-запрос или YDVP-ответ, единственным отличием для них будет интерпретация абстрактного класса YdvvpStartingLine, от которого наследуются классы YdvpStartingLineRequest и YdvpStartingLineRespone. Данный пакет также содержит класс YdvpParser, который предоставляет функционал обработки приходящих YDVP-запросов.

YDVP — собственный протокол приложения, основанный на версии HTTP 1.1.

Пакет клиента

Данный пакет включает в себя класс InfluxServiceClient, который позволяет подключиться к удаленному серверу, направлять ему YDVP-запросы и получать ответы.

Пакет сервера

Данный пакет включает в себя все необходимое для запуска сервера на заданном порту устройства.

В листинге 29 (приложение Б, с. 85) представлено описание класса сервера приложения. Данный класс инициализирует классы для инъекции зависимостей в модули, используемые сервисами и контроллерами, а также передает приходящих клиентов обработчик в отдельном потоке, таким образом достигается возможность одновременной обработки запросов от нескольких клиентов.

В листинге 30 (приложение Б, с. 85) представлено 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.

В листингах 31 и 32 (приложение Б, с. 85) представлен пример реализации взаимодействия пользователя с сервером.

На рисунках 3.4 — 3.7 представлены диаграммы классов компонентов модуля.

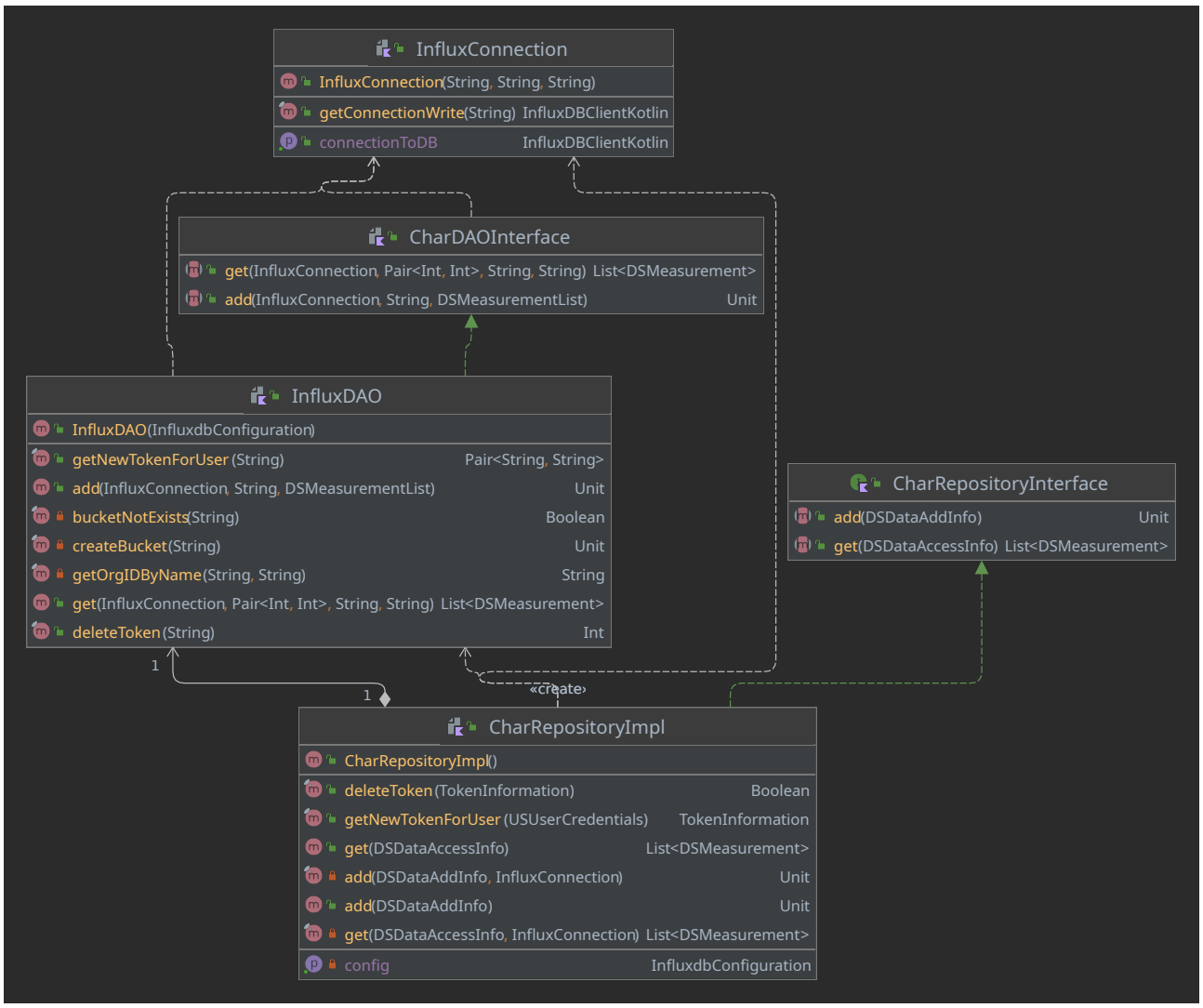

Рисунок 3.4 – Диаграмма классов слоя доступа к данным приложения.

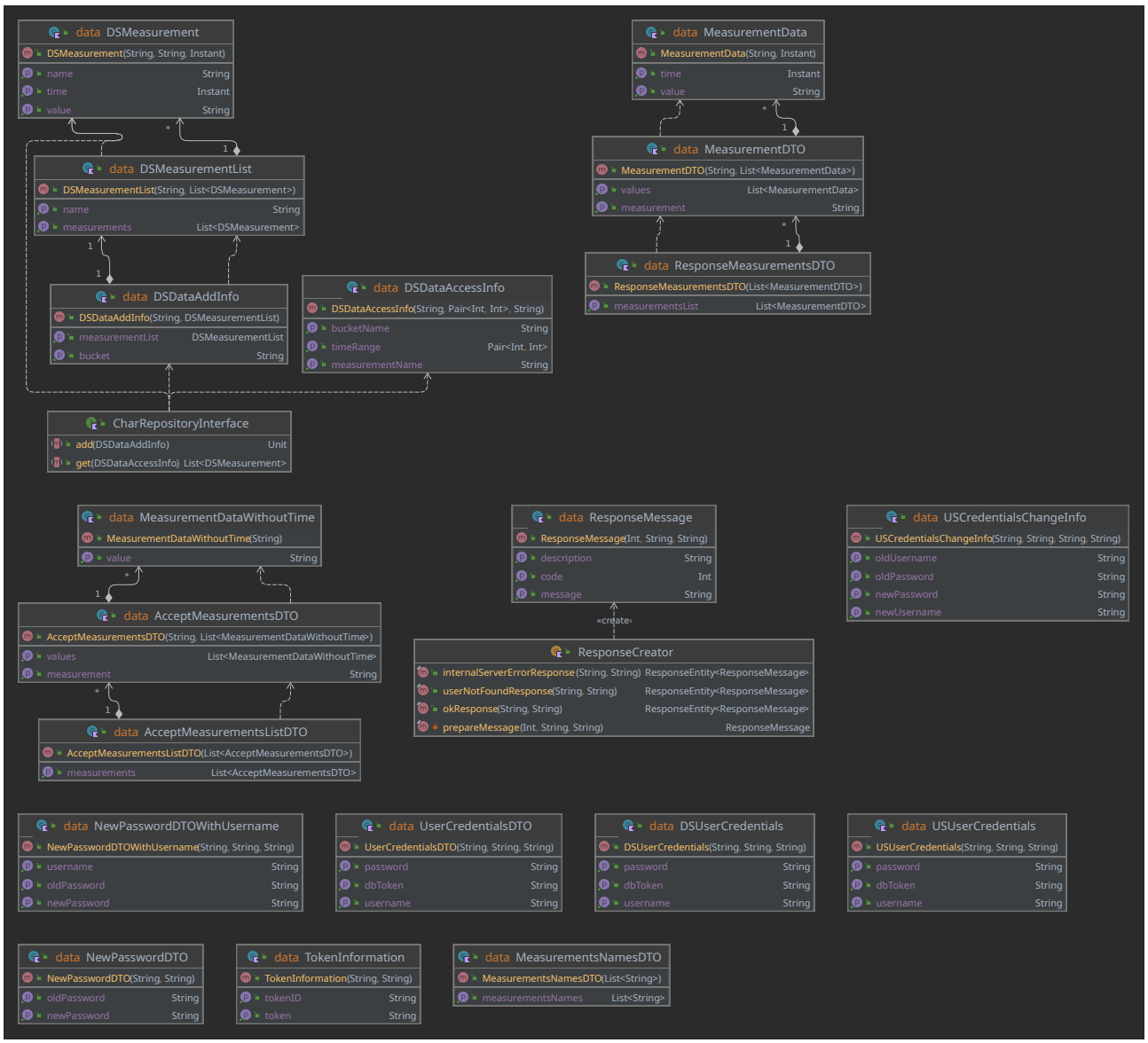

Рисунок 3.5 – Диаграмма классов бизнес-логики приложения.

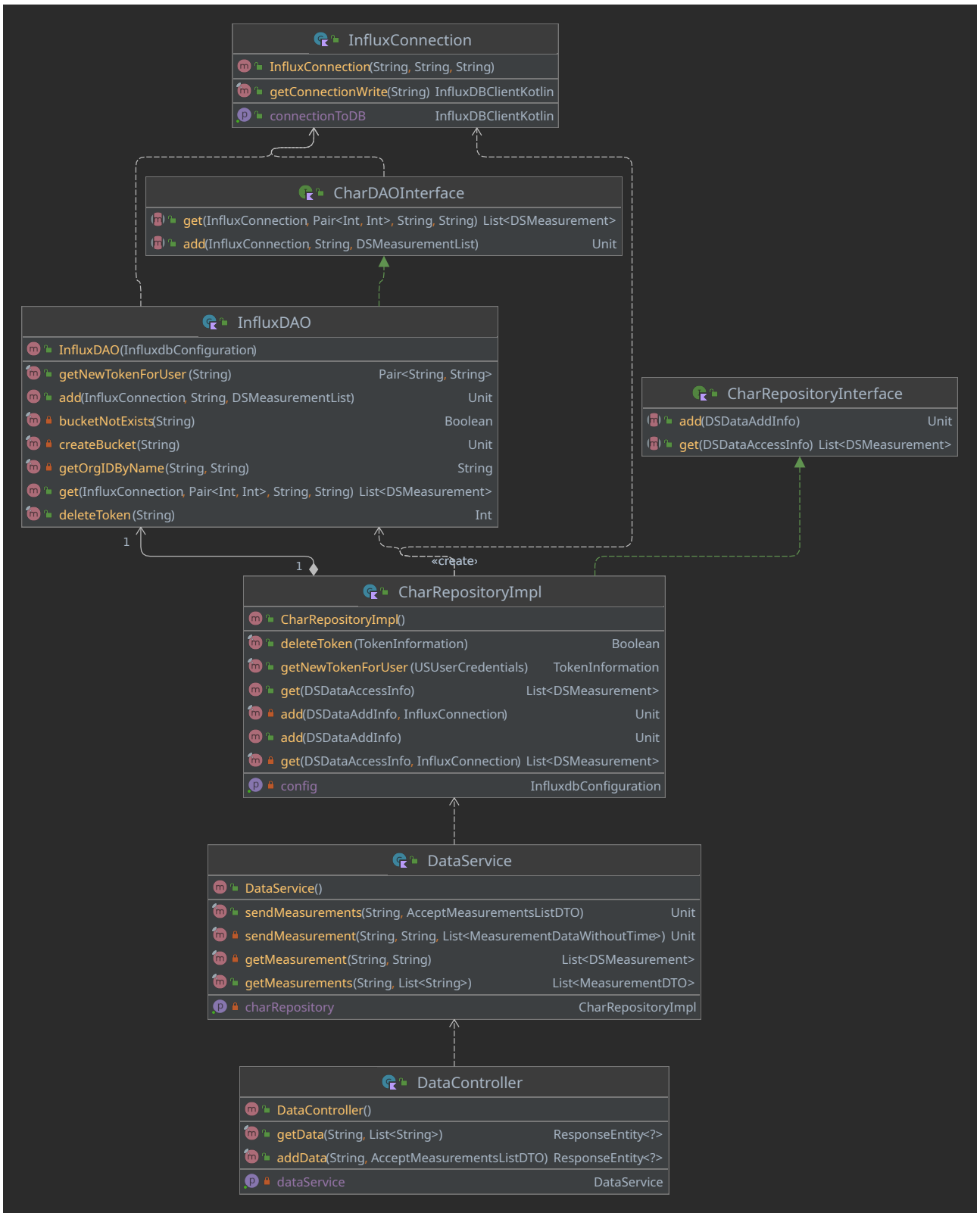

Рисунок 3.6 – Диаграмма классов, показывающая связь контроллера и слоя доступа к данным.

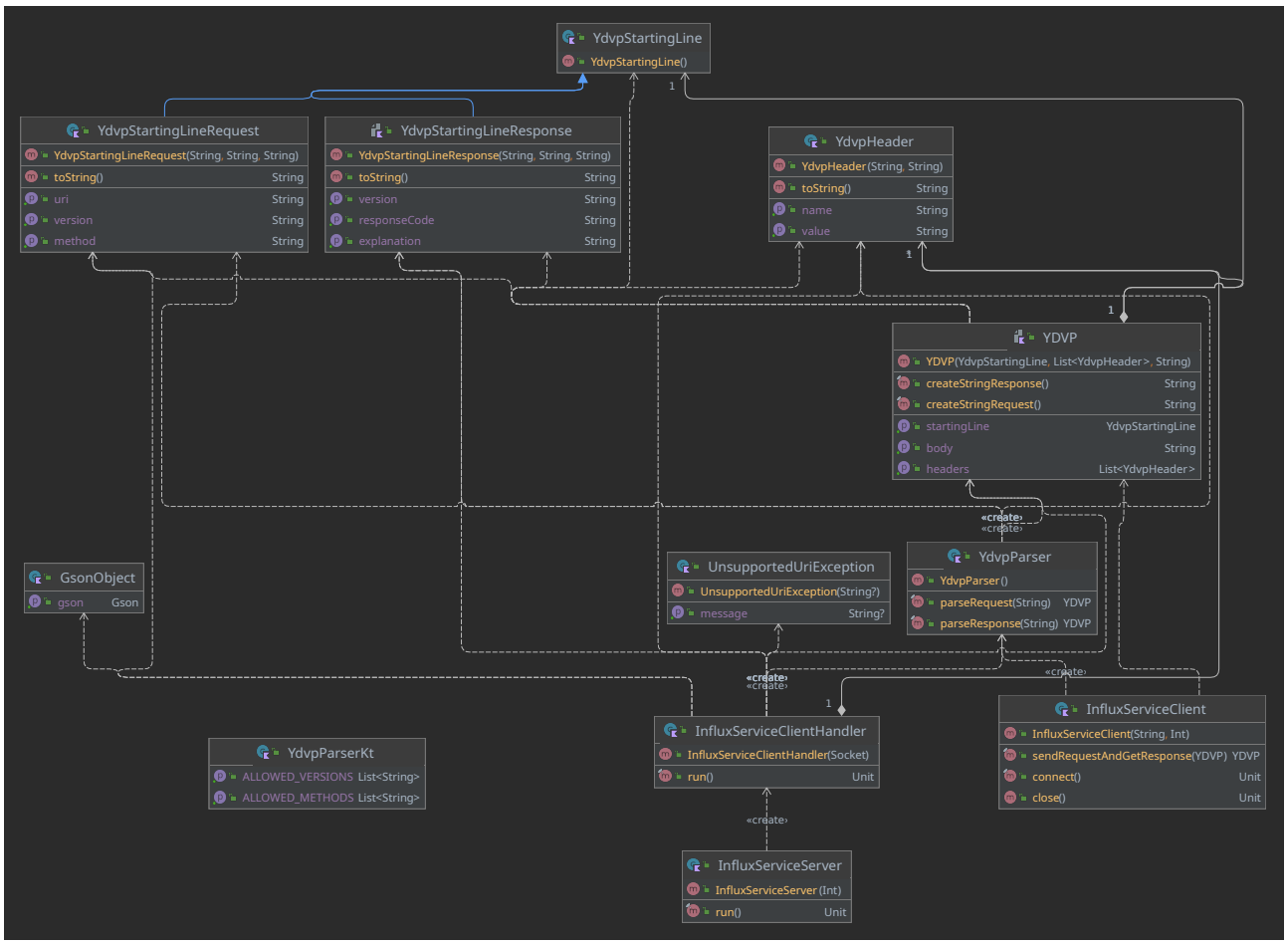

Рисунок 3.7 – Диаграмма классов серверной части приложения.

3.4.5 Пример использования модулей обработки и анализа данных

В листинге 28 представлен пример определения текущего состояния пользователя.

На рисунках 3.8 – 3.10 представлены все возможные результаты выполнения программы.

```
/home/trvehazzk3r/.jdks/corretto-15.0.2/bin/java ...
Quantized


Process finished with exit code 0
```

Рисунок 3.8 – Результат выполнения, сообщающий о том, что состояние пользователя не установлено.

```
/home/trvehazzk3r/.jdks/corretto-15.0.2/bin/java ...
Bored


Process finished with exit code 0
```

Рисунок 3.9 – Результат выполнения, сообщающий о том, что пользователь устал.

```
/home/trvehazzk3r/.jdks/corretto-15.0.2/bin/java ...
Workable


Process finished with exit code 0
```

Рисунок 3.10 – Результат выполнения, сообщающий о том, что пользователь работоспособен.

3.4.6 Пример использования серверной части приложения

В листингах 31 (приложение А, с. 62) и 32 (приложение А, с. 62) представлен пример реализации взаимодействия пользователя с сервером.

На рисунках 3.11 и 3.12

```
/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"}
```

Рисунок 3.11 – Результат выполнения запроса на стороне сервера.

Рисунок 3.12 – Результат выполнения запроса на стороне клиента.

Вывод

В качестве средства реализации был выбран язык программирования Kotlin, использовалась среда разработки IntelliJ IDEA.

В качестве используемой базы данных была выбрана база данных временных рядов, так как она нацелена на хранение, извлечение и анализ большого количества статистических данных. В качестве СУБД было решено использовать InfluxDB в силу отсутствия аналогов, а также по причине того, что поддержка данной СУБД все еще не прекращена на сегодняшний день.

В качестве алгоритма был выбран метод с-средних в силу того, что число кластеров заранее известно, а также задача рассматривает установку соответствия некоторого объекта набора вещественных значений, показывающих степень отношения объекта к кластерам.

Было определено, что в качестве данных для кластеризации используются действия оператора автоматизированного рабочего места, производимые с использованием клавиатуры и мыши.

Были приведены сведения и особенности модулей логирования действий оператора, анализа данных, серверного приложения.

Также были приведены примеры, которые показали возможные пути использования реализованных компонентов.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики ЭВМ, на котором выполнялись исследования:

- операционная система: Manjaro Linux (5.13.19-2);
- оперативная память: 16 гигабайт;
- процессор: Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz.

4.2 Сравнение времени исполнения запросов в базы данных Postgres и InfluxDB с использованием языка программирования Kotlin

4.2.1 Средства проведения стресс-тестирования баз данных

InfluxDB inch tool [41] — это средство симуляции потоковой передачи данных в InfluxDB для определения производительности системы.

Интерфейс приложения предполагает выбор количества отправляемых записей в базу данных, задание количества клиента, количества записей в серии отправки и длительности стресс-тестирования.

Данный проект предназначен для тестирования InlfluxDB версии 1.8, в связи с чем использование данного средства для версии, используемой в проекте (2.0), невозможна.

Apache JMeter [42] — это средство, разработанное для нагрузочного тестирования функционального поведения и измерения производительности с открытым исходным кодом.

В текущей версии проекта доступен Backend Listener, сконфигурированный специально для прослушивания порта InfluxDB и получения данных о выполняемых действиях.

Данный проект предполагает направления POST-запросов на адрес тестируемой базы данных, и изъятие данных по обработке данных запросов. При попытке реализации данного метода в документации проекта не было обнаружено конечного адреса отправки POST-запросов. В связи с данной проблемой данный метод тестирования был исключен из рассмотрения.

4.2.2 Среднее количество принимаемых от клиента данных

Согласно исследованиям 2014 года средняя продолжительность рабочего дня у российских организаций — 9 часов 50 минут. В Москве средняя продол-

жительность составляет 10 часов 31 минуту, а в Калининградской области — 9 часов 34 минуты. [43]

Количество поступающих данных от одного сотрудника за день может быть определено по формуле:

$$N_R = H \cdot M_R \cdot 60 + M \cdot M_R, \tag{7}$$

где H — количество часов в рабочем дне пользователя;

M_R — количество записей, поступающих от пользователя в минуту;

M — количество минут в рабочем дне пользователя.

С использованием (7) можно определить, что для Москвы среднее количество записей по единственной характеристике, учитывая обеденное время (предположительно час), будет равным 571. Для Калининградской области данное значение составит 514 записей.

Принимая за среднюю скорость печати 200 символов в минуту и за среднее количество кликов мыши 100 нажатий в минуту, максимальное количество записей, направленное пользователем за рабочий день, в Москве составит 114 тысяч, однако, учитывая перерывы и мыслительную работу, значение в 28 тысяч записей за день принимается в качестве среднего количества записей от пользователя в день.

4.2.3 Сравнение времени исполнения запросов в базы данных

В таблице 1 представлены результаты выполнения замеров скорости исполнения запросов на изъятие данных из сравниваемых баз данных. На рисунке 4.1 представлены данные из таблицы 1 в виде графика.

Таблица 1 – Замеры времени получения требуемого набора данных по пользователю в базах данных.

Общее количество записей в таблице (Postgres)	Количество записей, принадлежащих пользователю	Время исполнения запроса (Postgres), мс	Время исполнения запроса (InfluxDB), мс
30000	10000	188	64
36000	12000	248	65
42000	14000	274	60
48000	16000	261	60
54000	18000	266	61
60000	20000	284	68
66000	22000	316	76
72000	24000	340	65
78000	26000	363	65
84000	28000	410	67

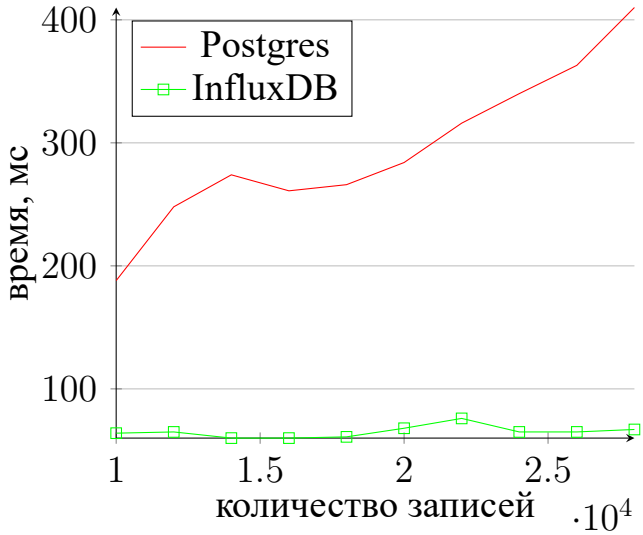

Рисунок 4.1 – Зависимость времени исполнения запроса от количества записей в таблице по пользователю.

Из данных, представленных на рисунке 4.1 видно, что время исполнения команды с использованием СУБД InfluxDB практически константно на рассмотренных значениях количества записей. Время исполнения запросов в СУБД Postgres было более долгим и при увеличении количества записей росло.

По таблице 1 были получены следующие результаты:

- пик превосходства в скорости исполнения был достигнут InfluxDB на количестве записей равным 28 тысяч — запрос был выполнен быстрее в 6 раз;
- в среднем InfluxDB исполняет запрос в ≈ 4.53 раза быстрее своего конкурента;
- минимальная разница в скорости исполнения запроса составила 124 миллисекунды, максимальная — 343 миллисекунды.

Для исследования в базе данных Postgres были созданы таблица пользователей и таблица, содержащая данные о нажатых клавишах клавиатуры.

В качестве средств доступа к базам данных использовались решения: PostgreSQL JDBC Driver [44] и InfluxDB Client Kotlin. Замеры производились с использованием утилиты measureTimeMillis [45].

Вывод

Поставленный эксперимент показал преимущество скорости исполнения запросов в базу данных InfluxDB над скоростью исполнения запросов в базу данных Postgres с использованием языка программирования Kotlin.

Исследования также показали, что время исполнения запросов в InfluxDB при количестве записей в таблице от 10 тысяч до 28 тысяч можно определить как константное. При этом в среднем InfluxDB позволяет получить ответ в $\approx$ 4.53 раза быстрее, чем Postrgres.

4.3 Сравнение количества успешных определений работоспособности пользователя путем варьирования фактора нечеткости

4.3.1 Предварительные условия

В данном исследовании используется набор данных, полученных от студента, выполняющего письменную работу. Во время сбора данных объекту исследования было разрешено отвлекаться на отдых, выполняя действия развлекательного характера в сети Интернет.

Полные данные, в силу их объема, приведены быть не могут. В листингах 33 (приложение В, с. 93) и 34 (приложение В, с. 93) приведена часть полученных по пользователю данных.

В качестве варьируемого параметра принимается критерий нечеткости при проведении кластеризации методом с-средних.

По тесту на реакцию было определено, что пользователю требуется отдых по данным, направляемым анализатору.

4.3.2 Сравнение количества успешных определений работоспособности по клавиатуре

В таблице 2 представлены результаты определений работоспособности пользователя по заданной выборке с использованием различных факторов не четкости. Для каждого значения фактора проводилось 10 определений состояния.

Таблица 2 – Количество определений состояния системой в зависимости от значения критерия нечеткости.

Значение критерия нечеткости	Количество определений состояния		
	Неопределен	Устал	Работоспособен
1.5	10	0	0
2.0	10	0	0
2.5	10	0	0
3.0	10	0	0
3.5	10	0	0
4.0	10	0	0
4.5	0	10	0
5.0	0	10	0
5.5	0	10	0
6.0	0	10	0
6.5	0	10	0
7.0	0	10	0
7.5	0	10	0
8.0	0	10	0
8.5	0	10	0
9.0	0	10	0
9.5	0	10	0
10.0	0	10	0

Полученные данные позволяют определить, что наибольшей точности можно добиться при использовании значения критерия нечеткости больше или равного 4.5, так как в таком случае шанс успешного определения состояния пользователя составляет 100%.

Вывод

В результате проведенного исследования было определено, что на заданной выборке наиболее точными являются факторы нечеткости, лежащие на отрезке от 4.5 до 10.0, так как при их использовании шанс распознать истинное

состояние пользователя составляет 100%.

4.3.3 Сравнение количества успешных определений работоспособности по мыши

В таблице 3 представлены результаты определений работоспособности пользователя по заданной выборке с использованием различных факторов не четкости. Для каждого значения фактора проводилось 10 определений состояния.

Таблица 3 – Количество определений состояния системой в зависимости от значения критерия нечеткости.

Значение критерия нечеткости	Количество определений состояния		
	Неопределен	Устал	Работоспособен
1.5	0	10	0
2.0	0	10	0
2.5	0	10	0
3.0	0	10	0
3.5	0	10	0
4.0	10	0	0
4.5	8	2	0
5.0	6	4	0
5.5	9	1	0
6.0	9	1	0
6.5	10	0	0
7.0	10	0	0
7.5	10	0	0
8.0	0	10	0
8.5	10	0	0
9.0	0	10	0
9.5	0	10	0
10.0	0	10	0

Полученные данные позволяют определить, что наибольшей точности можно добиться при использовании значений критерия нечеткости 1.5 – 3.5, 8.0, 9.0 – 10.0, так как при их использовании шанс распознать истинное состояние пользователя составляет 100%.

Вывод

В результате проведенного исследования было определено, что на заданной выборке наиболее точными являются факторы нечеткости 1.5 – 3.5, 8.0, 9.0

– 10.0, так как при их использовании шанс распознать истинное состояние пользователя составляет 100%. Причем критерий нечеткости 5.0 позволил с шансом в 40% распознать опасное состояние оператора, в то время как оставшиеся критерии имели шанс в 20% и 10%.

Вывод

Исследование в области сравнения времени исполнения запросов в базы данных показало преимущество скорости исполнения запросов в базу данных InfluxDB над скоростью исполнения запросов в базу данных Postgres с использованием языка программирования Kotlin. Результаты показали, что время исполнения запросов в InfluxDB при количестве записей в таблице от 10 тысяч до 28 тысяч практически можно приравнять к константе. При этом в среднем InfluxDB позволяет получить ответ в ≈ 4.53 раза быстрее, чем Postgres.

Исследования в области сравнения количества успешных определений работоспособности пользователя показали, что при распознавании усталости с использованием клавиатуры при варьировании фактора нечеткости было получено $\approx 67\%$ верных результатов, что на 13% больше, чем при использовании мыши с точностью 54%. Таким образом, на заданной выборке более эффективным устройством для распознавания усталости оказалась клавиатура. При этом для клавиатуры наиболее точными факторами нечеткости являются значения, лежащие на отрезке от 4.5 до 10.0, которые позволили с точностью в 100% определить истинное состояние оператора. Для мыши наиболее точными факторами нечеткости являются 1.5–3.5, 8.0, 9.0–10.0.

ЗАКЛЮЧЕНИЕ

Были рассмотрены понятия усталости, хронической усталости, стресса и профессионального стресса, формализована цель разработанного метода.

Был проведен анализ существующих методов определения усталости:

- анализ клавиатурного почерка;
- анализ скорости печати и количества ошибок;
- анализ использования координатного устройства мышь с использованием модели искусственной нейронной сети;
- анализ внешнего состояния пользователя;
- анализ речевых характеристик пользователя;
- анализ виброакустических шумов при наборе текста или использовании мыши.

Был проведен анализ действий и характеристик, позволяющих определить усталость пользователя автоматизированного рабочего места. Выделены биофизические факторы, позволяющие определить усталость оператора: частота пульса и возраст сосудистой системы, индекс Баевского. Также рассмотренные существующие методы позволили в качестве рассматриваемых действий пользователя выбрать нажатия на клавиши клавиатуры и мыши. Было определено, что система будет проводить кластеризацию методом с-средних по значениям скорости печати, скорости передвижения курсора между двумя нажатиями, а также по значениям времени реакции пользователя.

Были определены методы снятия выделенных действий и характеристик. Каждое нажатие на клавишу клавиатуры характеризуется наименованием нажатой клавиши и временной меткой. Каждое нажатие на клавишу мыши характеризуется координатами экрана, в которых действие было совершено, номером нажатой клавиши и временной меткой. Тест на реакцию характеризуется определенным временем реакции и временной меткой, когда значение было получено.

Был разработан метод распознавания усталости оператора, который включил в себя хранение и анализ данных, получаемых от клавиатуры и мыши. Было определено, что формирование нечетких кластеров происходит на этапе синхронизации поступающих данных о действиях пользователя с данными о скорости его реакции в отдельные моменты времени. Отмечено, что в дальней-

шем полученная модель используется до ее актуализации, которая происходит в случае появления ложных срабатываний или радикального изменения поведения объекта.

Разработанный метод был реализован. Реализация включила в себя три модуля: логирования действий оператора, анализа данных и серверного приложения. Были приведены особенности реализации каждого модуля, диаграммы классов.

Результаты исследования позволили определить, что выбор в пользу InfluxDB имеет и экспериментальные подтверждения. Поставленный эксперимент показал преимущество скорости исполнения запросов в базу данных InfluxDB над скоростью исполнения запросов в базу данных Postgres с использованием языка программирования данных. В среднем InfluxDB позволил получить ответ в ≈ 4.53 раза быстрее.

Также было проведено исследование в области сравнения количества успешных определения работоспособности пользователя путем варьирования фактора нечеткости. В результате было определено, что при определении усталости с использованием клавиатуры на заданной выборке при варьировании фактора нечеткости было получено $\approx$ 67% верных результатов, что на 13% больше, чем при использовании мыши. Эффективным устройством для распознавания усталости в данных испытаниях была признана клавиатура.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

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. М.: 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. Т.В. Байдина Т.И. Колесова Б.В. Малинина. Усталость как симптом неврологических заболеваний // Пермский медицинский журнал. 2021. Т. 38, № 2. С. 37–44.

6. Н.В. Пизова А.В. Пизов. Когнитивные нарушения и синдром хронической усталости // Нервные болезни. 2021. № 3. С. 10–16.

7. Committee on the Diagnostic Criteria for Myalgic Encephalomyelitis/Chronic Fatigue Syndrome; Board on the Health of Select Populations; Institute of Medicine // The National Academies Collection: Reports funded by National Institutes of Health. 2015.

8. Долбышев А.В. Нейрофизиологические механизмы стресса // StudNet. 2020. № 7. С. 163–167.

9. Г.В. Порядина. Стресс и патология: учеб. пособие. М.: РГМУ, 2009. с. 23.

10. Деева О.С. Причины профессионального стресса и методы его профилактики // Ученые записки Тамбовского отделения РоСМУ. 2019. Т. 38, № 14. С. 140–146.

11. Информационное общество в Российской Федерации, 2020. Статистический сборник [Электронный ресурс]. Режим доступа:

https://rosstat.gov.ru/storage/mediabank/lqv3T0Rk/info-ob2020.pdf (дата обращения 08.12.2021). 2020.

12. Большой энциклопедический политехнический словарь [Электронный ресурс]. Режим доступа: https://rus-big-polyheh-dict.slovaronline.com/ (дата обращения 22.12.2021).

13. В.И. Васильев А.Е. Сулавко Р.В. Борисов. Распознавание психофизиологических состояний пользователей на основе скрытого мониторинга действий в компьютерных системах // ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ И ПРИНЯТИЕ РЕШЕНИЙ. 2017. С. 21–37.

14. Е. В. Шкляр Е. Г. Воробьев М. Ф. Савельев. Распознавание клавиатурного почерка в браузере // Известия СПбГЭТУ «ЛЭТИ». 2019. С. 58–63.

15. Axelsson S. The Base-Rate Fallacy and its Implications for the Difficulty of Intrusion Detection. 1999. 12. P. 1–10.

16. Попов А.Ю. Макушкина Л.А. ИССЛЕДОВАНИЕ ПСИХОФИЗИЧЕСКОГО СОСТОЯНИЯ ПОЛЬЗОВАТЕЛЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ // Вестник магистратуры. 2015. Т. 41, № 2-2.

17. Раскин Д. Интерфейс: новые направления в проектировании компьютерных систем. Символ-плюс, 2010. с. 272.

18. Р.В. Борисов Д.Н. Зверев А.Е. Сулавко. Оценка идентификационных возможностей особенностей работы пользователя с компьютерной мышью // Вестник СибАДИ. 2015. С. 106–113.

19. Савинова В.М. Бесхмельницкий А.А. Бибина Е.С. Идентификация пользователей корпоративной системы с помощью поведенческого анализа с использованием модели искусственной нейронной сети // ТДР. 2017. № 5.

20. И.Б. Лашков А.М. Кашевник. Определение опасных состояний водителя на основе мобильных видеоизмерений его лицевых характеристик // ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ И ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ. 2019. С. 84–96.

21. М.А. Лебедев С.Ю. Палатов Г.В. Ковров. Усталость и ее проявления // Медицинское обозрение. 2014. С. 282–288.

22. Д. Бернс. Терапия настроения: Клинически доказанный способ победить депрессию без таблеток. М.: Альпина Паблишер, 2019. с. 271.

23. В.В. Киселев. Автоматическое определение эмоций по речи // Технологии и практика обучения. 2012. С. 85–89.

24. Voice Stress Analysis: A New Framework for Voice and Effort in Human Performance / Martine Van Puyvelde, Xavier Neyt, Francis McGlone [и др.] // Frontiers in Psychology. 2018. Т. 9, № 14.

25. Федеров В.М. Реблев Д.П. Панченко Е.М. Идентификация пользователя по виброакустическим шумам // Известия ЮФУ. Технические науки. 2013. Т. 149, № 12.

26. Рублев Д.П. Федоров В.М. Индетификация пользователя по динамическим характеристикам работы с манипулятором “мышь” с использованием нейронных сетей // Известия ЮФУ. Технические науки. 2017. Т. 190, № 5.

27. Кропачев И.Г. Архипова Е.И. Нора С.А. Показатели сердечно-сосудистой системы и гемодинамики в условиях стрессового воздействия как фактор развития острых респираторных инфекций // Вестник НовГУ. 2020. Т. 119, № 3.

28. А.В. Капустин. Изменение показателей биологического возраста при патологии сосудов // Бюллетень Северного Государственного медицинского Университета. 2018. № 1.

29. Э.Э. Ибрагимова. Мониторинг уровня стресса обучающихся как подход профилактики нарушения регуляторных механизмов // Ученые записки Крымского федерального университета имени В. И. Вернадского. Социология. Педагогика. Психология. 2019. № 2.

30. Р.М. Баевский. Прогнозирование состояний на грани нормы и патологии. М.: Медицина, 1979. с. 298.

31. Ершов К.С. Романова Т.Н. Анализ и классификация алгоритмов кластеризации // Новые ифномрационные технологии в автоматизированных системах. 2016. № 19.

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

33. Шабельников А.Н. Шабельников В.А. Поиск аномалий в технических базах данных временных рядов // Известия ЮФУ. Технические науки. 2008. № 4.

34. TProger : Знакомство с InfluxDB и базами данных временных рядов [Электронный ресурс]. Режим доступа: https://tproger.ru/translations/influxdb-guide/ (дата обращения 01.04.2020).

35. Н.Р. Булахов. Основы реляционных баз данных // Вестник науки и образования. 2019. Т. 76, № 22-2.

36. Документация Oracle - Java Swing [Электронный ресурс]. Режим доступа: https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html (дата обращения 16.05.2022).

37. Официальный репозиторий GitHub проекта JNativeHook [Электронный ресурс]. Режим доступа: https://github.com/kwhat/jnativehook (дата обращения 16.05.2022).

38. Apache Commons: официальный сайт [Электронный ресурс]. Режим доступа: https://commons.apache.org/proper/commons-math/ (дата обращения 20.05.2022).

39. InfluxDB Client Kotlin: официальный репозиторий [Электронный ресурс]. Режим доступа: https://github.com/influxdata/influxdb-client-java/tree/master/client-kotlin (дата обращения 20.05.2022).

40. OkHttp: официальная страница [Электронный ресурс]. Режим доступа: https://square.github.io/okhttp/ (дата обращения 20.05.2022).

41. Infludata: InfluxDB inch tool [Электронный ресурс]. Режим доступа: https://docs.influxdata.com/influxdb/v1.8/tools/inch/ (дата обращения 01.04.2020).

42. Apache Jmeter [Электронный ресурс]. Режим доступа: https://jmeter.apache.org/usermanual/realtime-results.html (дата обращения 01.04.2020).

43. Исследование Яндекс: Время работы организаций [Электронный ресурс]. Режим доступа: https://yandex.ru/company/researches/2014/ya_time_regions. (дата обращения 01.04.2020).

44. Oracle: Develop Java applications with Oracle Database [Электронный ресурс]. Режим доступа: https://www.oracle.com/ru/database/technologies/appdev/jdbc.html (дата обращения 01.04.2020).

45. Kotlin Documentation: measureTimeMillis [Электронный ресурс]. Режим доступа: https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.system/measure-time-millis.html (дата обращения 01.04.2020).

ПРИЛОЖЕНИЕ А

Иходный код модулей обработки и анализа данных

Листинг 1: Файл main.kt

```
 1 import bigBrother.BigBrotherWindow
 2 import java.awt.EventQueue
 3
 4 private fun createAndShowGUI() {
 5     val frame = BigBrotherWindow("BB Неактивен()")
 6     frame.isVisible = true
 7 }
 8
 9 fun main(args: Array<String>) {
10     EventQueue.invokeLater(::createAndShowGUI)
11 }
```

Листинг 2: Файл Window.kt

```
 1 package window
 2
 3 import javax.swing.GroupLayout
 4 import javax.swing.ImageIcon
 5 import javax.swing.JComponent
 6 import javax.swing.JFrame
 7
 8 open class Window: JFrame() {
 9
10     init {
11         iconImage = ImageIcon(this.javaClass.getResource("/bb.
               png")).image
12     }
13
14     fun createLayout(vararg components: JComponent) {
15         val gl = GroupLayout(contentPane)
16         contentPane.layout = gl
17
18         gl.autoCreateContainerGaps = true
19
20         val horizontalGroup = gl.createParallelGroup(
```

```
            GroupLayout.Alignment.CENTER)
21          components.forEach { horizontalGroup.addComponent(it) }
22
23          gl.setHorizontalGroup(horizontalGroup)
24
25
26          val verticalGroup = gl.createSequentialGroup()
27          components.forEach { verticalGroup.addComponent(it) }
28
29          gl.setVerticalGroup(verticalGroup)
30
31          pack()
32      }
33 }
```

Листинг 3: Файл BigBrotherWindow.kt

```
 1 package bigBrother
 2
 3 import loggers.keyLogger.KeyLogger
 4 import loggers.mouseLogger.MouseLogger
 5 import loggers.reactionTest.ReactionTestWindow
 6 import window.Window
 7 import java.io.File
 8 import java.lang.Exception
 9 import javax.swing.*
10
11 class BigBrotherWindow(title: String) : Window() {
12
13     var currentUsername: String? = null
14
15     var mouseLogger: MouseLogger? = null
16     var keyLogger: KeyLogger? = null
17
18     init {
19         try {
20             UIManager.setLookAndFeel(UIManager.
                   getSystemLookAndFeelClassName())
21         } catch (ex: Exception) {
22         }
23
24         File("${System.getProperty("user.dir")}/data").mkdir()
```

```
25
26          createUI(title)
27      }
28
29      private fun getInputAndStartLoggingMoves(textField:
            JTextField) {
30          currentUsername = textField.text.trim()
31
32          mouseLogger = MouseLogger(currentUsername!!)
33          keyLogger = KeyLogger(currentUsername!!)
34
35          mouseLogger?.start()
36          keyLogger?.start()
37      }
38
39      private fun stopWork() {
40          mouseLogger!!.stop()
41          keyLogger!!.stop()
42      }
43
44      private fun createUI(title: String) {
45
46          setTitle(title)
47
48          val input = JTextField("ФамилияИмя__Ваш( факультет)xxx-
                ")
49          input.horizontalAlignment = JTextField.CENTER
50
51          val goButton = JButton("Начать слежку")
52          goButton.addActionListener {
53              getInputAndStartLoggingMoves(input)
54              JOptionPane.showMessageDialog(
55                  this,
56                  "Наблюдаю зажизнедеятельностью ",
57                  "Изменение статуса",
58                  JOptionPane.INFORMATION_MESSAGE
59              )
60              setTitle("BB активен()")
61          }
62
63          val stopButton = JButton("Остановить слежку")
```

```
64          stopButton.addActionListener {
65              stopWork()
66              JOptionPane.showMessageDialog(
67                  this,
68                  "Закончил наблюдениезажизнедеятельностью  ",
69                  "Изменение статуса",
70                  JOptionPane.INFORMATION_MESSAGE
71              )
72              setTitle("BB неактивен()")
73          }
74
75          val checkReactionButton = JButton("Проверить реакцию")
76          checkReactionButton.addActionListener {
77              val frame = ReactionTestWindow("Тест реакции",
                    currentUsername ?: input.text.trim())
78              frame.isVisible = true
79          }
80
81          createLayout(input, goButton, stopButton,
                checkReactionButton)
82
83          defaultCloseOperation = JFrame.EXIT_ON_CLOSE
84          setSize(400, 200)
85          setLocationRelativeTo(null)
86      }
87 }
```

Листинг 4: Файл BBRandom.kt

```
1 package random
2
3 import java.util.*
4
5 object BBRandom {
6     val random by lazy { Random() }
7 }
```

Листинг 5: Файл Logger.kt

```
1 package loggers
2
3 interface Logger {
4     fun start()
```

```
5
6     fun stop()
7 }
```

Листинг 6: Файл KeyLogger.kt

```
1 package loggers.keyLogger
2
3 import com.github.kwhat.jnativehook.GlobalScreen
4 import com.github.kwhat.jnativehook.NativeHookException
5 import com.github.kwhat.jnativehook.keyboard.NativeKeyEvent
6 import loggers.Logger
7 import com.github.kwhat.jnativehook.keyboard.NativeKeyListener
8 import java.io.File
9 import java.util.LinkedList
10 import kotlin.system.exitProcess
11
12 class KeyLogger(username: String) : Logger, NativeKeyListener {
13     private val mPathToFile = "${System.getProperty("user.dir")
          }/data/${username}_Keys.txt"
14
15     private var mFile = File(mPathToFile)
16
17     @Volatile
18     private var queueToWrite = LinkedList<String>()
19
20     private fun setFile() {
21         if (!mFile.exists()) mFile.createNewFile()
22     }
23
24     override fun start() {
25         try {
26             if (!GlobalScreen.isNativeHookRegistered())
                  GlobalScreen.registerNativeHook()
27         } catch (ex: NativeHookException) {
28             System.err.println("There was a problem registering
                   the native hook.")
29             exitProcess(1)
30         }
31
32         setFile()
33
```

```
34          GlobalScreen.addNativeKeyListener(this)
35      }
36
37      private fun clearQueue() {
38          queueToWrite.forEach {
39              mFile.appendText(it)
40          }
41          queueToWrite.clear()
42      }
43
44      override fun stop() {
45          clearQueue()
46
47          try {
48              GlobalScreen.removeNativeKeyListener(this)
49              GlobalScreen.unregisterNativeHook()
50          } catch (e: Exception) {
51          }
52      }
53
54      override fun nativeKeyPressed(e: NativeKeyEvent) {
55          if (!e.isActionKey) {
56              queueToWrite.add(
57                  "key=${NativeKeyEvent.getKeyText(e.keyCode)},"
                        + " timestamp=${System.currentTimeMillis()}\
                        n"
58              )
59              if (queueToWrite.size == 100) {
60                  clearQueue()
61              }
62          }
63      }
64
65      override fun nativeKeyReleased(e: NativeKeyEvent) {
66      }
67
68      override fun nativeKeyTyped(e: NativeKeyEvent) {
69      }
70  }
```

Листинг 7: Файл MouseLogger.kt

```
 1 package loggers.mouseLogger
 2
 3 import com.github.kwhat.jnativehook.GlobalScreen
 4 import com.github.kwhat.jnativehook.NativeHookException
 5 import com.github.kwhat.jnativehook.mouse.NativeMouseEvent
 6 import com.github.kwhat.jnativehook.mouse.
     NativeMouseInputListener
 7 import loggers.Logger
 8 import java.io.File
 9 import java.util.*
10 import kotlin.system.exitProcess
11
12 class MouseLogger(username: String) : Logger,
     NativeMouseInputListener {
13
14     private val mPathToFileForMoves = "${System.getProperty("
         user.dir")}/data/${username}_Mouse_Moves.txt"
15     private val mPathToFileForClicks = "${System.getProperty("
         user.dir")}/data/${username}_Mouse_Clicks.txt"
16
17     @Volatile
18     private var mFileForMoves = File(mPathToFileForMoves)
19
20     @Volatile
21     private var mFileForClicks = File(mPathToFileForClicks)
22
23     @Volatile
24     private var queueOfMoves = LinkedList<String>()
25
26     @Volatile
27     private var queueOfClicks = LinkedList<String>()
28
29     private fun setFiles() {
30         if (!mFileForMoves.exists()) mFileForMoves.
             createNewFile()
31         if (!mFileForClicks.exists()) mFileForClicks.
             createNewFile()
32     }
33
34     private fun clearToFile(queue: LinkedList<String>, file:
         File) {
```

```
35          queue.forEach {
36              file.appendText(it)
37          }
38          queue.clear()
39      }
40
41      override fun start() {
42          try {
43              if (!GlobalScreen.isNativeHookRegistered())
44                  GlobalScreen.registerNativeHook()
45          } catch (ex: NativeHookException) {
46              System.err.println("There was a problem registering
                    the native hook.")
47              exitProcess(1)
48          }
49
50          setFiles()
51
52          GlobalScreen.addNativeMouseListener(this)
53          GlobalScreen.addNativeMouseMotionListener(this)
54      }
55
56      override fun stop() {
57          clearToFile(queueOfClicks, mFileForClicks)
58          clearToFile(queueOfMoves, mFileForMoves)
59
60          try {
61              GlobalScreen.removeNativeMouseListener(this)
62              GlobalScreen.removeNativeMouseMotionListener(this)
63              GlobalScreen.unregisterNativeHook()
64          } catch (e: Exception) {
65          }
66      }
67
68      override fun nativeMouseClicked(e: NativeMouseEvent?) {
69          queueOfClicks.add(
70              "x=${e!!.x}, y=${e.y}, key=${e.button}," +
71                      " timestamp=${System.currentTimeMillis()}\n
                           "
72          )
73
```

```
 74         if (queueOfClicks.size == 100) {
 75             clearToFile(queueOfClicks, mFileForClicks)
 76         }
 77     }
 78
 79     override fun nativeMousePressed(p0: NativeMouseEvent?) {
 80
 81     }
 82
 83     override fun nativeMouseReleased(p0: NativeMouseEvent?) {
 84
 85     }
 86
 87     override fun nativeMouseMoved(e: NativeMouseEvent?) {
 88         queueOfMoves.add(
 89             "x=${e!!.x}, y=${e.y}," +
 90                     " timestamp=${System.currentTimeMillis()}\n
                         "
 91         )
 92
 93         if (queueOfMoves.size == 100) {
 94             clearToFile(queueOfMoves, mFileForMoves)
 95         }
 96     }
 97
 98     override fun nativeMouseDragged(p0: NativeMouseEvent?) {
 99
100     }
101
102 }
```

Листинг 8: Файл ReactionLogger.kt

```
 1 package loggers.reactionTest
 2
 3 import loggers.Logger
 4 import java.io.File
 5
 6 class ReactionLogger(username: String) : Logger {
 7
 8     private val mPathToFile = "${System.getProperty("user.dir")
          }/data/${username}_Reactions.txt"
```

```
 9
10      private val mFile = File(mPathToFile)
11
12      private fun setFile() {
13          if (!mFile.exists()) mFile.createNewFile()
14      }
15
16      override fun start() {
17          setFile()
18      }
19
20      fun addRecord(resultInMillis: Long) {
21          mFile.appendText(
22              "reaction_time=${resultInMillis}, " +
23                      "timestamp=${System.currentTimeMillis()}\n"
24          )
25      }
26
27      override fun stop() {}
28  }
```

Листинг 9: Файл ReactionTestWindow.kt

```
 1 package loggers.reactionTest
 2
 3 import random.BBRandom
 4 import window.Window
 5 import java.awt.Font
 6 import java.lang.Thread.sleep
 7 import javax.swing.*
 8 import kotlin.concurrent.thread
 9
10 class ReactionTestWindow(title: String, username: String) :
       Window() {
11
12      @Volatile
13      private var testButtonPressed = false
14
15      @Volatile
16      private var reactionTimestamp = 0L
17
18      private var reactionsTotal = 0L
```

```
19
20      private var reactionLogger: ReactionLogger
21
22      init {
23          reactionLogger = ReactionLogger(username)
24
25          reactionLogger.start()
26
27          createUI(title)
28      }
29
30      private fun createUI(title: String) {
31
32          setTitle(title)
33
34          val testButton = JButton("Жать сюда!")
35          testButton.font = Font("Arial", Font.PLAIN, 50)
36          testButton.addActionListener {
37              reactionTimestamp = System.currentTimeMillis()
38              testButtonPressed = true
39          }
40          testButton.isVisible = true
41
42          val spacer = JLabel(" ")
43          spacer.font = Font("Arial", Font.PLAIN, 50)
44
45          val startButton = JButton("Начать тест")
46          startButton.addActionListener {
47              startButton.isVisible = false
48              thread {
49                  var startTime: Long
50                  for (i in 0 until NUMBER_OF_TESTS) {
51                      sleep((3 + BBRandom.random.nextInt(8).
                            toLong()) * 1000)
52                      testButton.isVisible = true
53                      startTime = System.currentTimeMillis()
54                      while (!testButtonPressed) {
55                          sleep(20)
56                      }
57                      reactionsTotal += reactionTimestamp -
                            startTime
```

```
58                              testButton.isVisible = false
59                          testButtonPressed = false
60                  }
61
62                  this.isVisible = false
63
64                  reactionLogger.addRecord(reactionsTotal /
                        NUMBER_OF_TESTS)
65              }
66          }
67
68
69          createLayout(testButton, spacer, startButton)
70
71          testButton.isVisible = false
72
73          setLocationRelativeTo(null)
74      }
75
76      companion object {
77          private const val NUMBER_OF_TESTS = 10
78      }
79 }
```

Листинг 10: Файл Model.kt

```
 1 package bbParser.models
 2
 3 import java.util.*
 4
 5 abstract class Model(val mTimestamp: Long) {
 6
 7     override fun toString(): String {
 8         return "timestamp=$mTimestamp]"
 9     }
10 }
```

Листинг 11: Файл KeyModel.kt

```
 1 package bbParser.models
 2
 3 import dateFormat.DateFormatter
 4
```

```
 5 class KeyModel(
 6     val mKeyName: String,
 7     timestamp: Long
 8 ): Model(timestamp) {
 9
10     override fun toString(): String {
11         return "[key=$mKeyName; " + super.toString()
12     }
13 }
```

Листинг 12: Файл MouseClickModel.kt

```
 1 package bbParser.models
 2
 3 class MouseClickModel(
 4     val mXCoordinate: Int,
 5     val mYCoordinate: Int,
 6     val mKey: Int,
 7     timestamp: Long
 8 ): Model(timestamp) {
 9
10     override fun toString(): String {
11         return "[x=$mXCoordinate; y=$mYCoordinate; key=$mKey; "
               + super.toString()
12     }
13 }
```

Листинг 13: Файл ReactionModel.kt

```
 1 package bbParser.models
 2
 3 class ReactionModel(
 4     val mReactionMillis: Int,
 5     timestamp: Long
 6 ): Model(timestamp) {
 7
 8     override fun toString(): String {
 9         return "[reaction_time=$mReactionMillis; " + super.
              toString()
10     }
11 }
```

Листинг 14: Файл BbParser.kt

```
1 package bbParser.parsers
2
3 import bbParser.models.Model
4 import java.io.File
5
6 abstract class BbParser(private val parseFun: (File) -> List<
    Model>) {
7
8     fun parseFile(path: String): List<Model> {
9         val file = File(path)
10        return if (file.exists() && file.canRead()) parseFun(
              file) else listOf()
11    }
12 }
```

Листинг 15: Файл KeysBbParser.kt

```
1 package bbParser.parsers
2
3 import bbParser.models.KeyModel
4 import bbParser.prefixes.Prefixes
5
6 class KeysBbParser : BbParser(
7     { file ->
8         file.readLines().map { line ->
9             val strValues = line.split(',')
10            KeyModel(
11                strValues[0].trim().removePrefix(Prefixes.KEY),
12                strValues[1].trim().removePrefix(Prefixes.
                      TIMESTAMP).toLong()
13            )
14        }
15    }
16 )
```

Листинг 16: Файл MouseClicksBbParser.kt

```
1 package bbParser.parsers
2
3 import bbParser.models.MouseClickModel
4 import bbParser.prefixes.Prefixes
5
```

```
 6 class MouseClicksBbParser : BbParser(
 7     { file ->
 8         file.readLines().map { line ->
 9             val strValues = line.split(',')
10             MouseClickModel(
11                 strValues[0].trim().removePrefix(Prefixes.
                       X_COORDINATE).toInt(),
12                 strValues[1].trim().removePrefix(Prefixes.
                       Y_COORDINATE).toInt(),
13                 strValues[2].trim().removePrefix(Prefixes.KEY).
                       toInt(),
14                 strValues[3].trim().removePrefix(Prefixes.
                       TIMESTAMP).toLong()
15             )
16         }
17     }
18 )
```

Листинг 17: Файл ReactionsBbParser.kt

```
 1 package bbParser.parsers
 2
 3 import bbParser.models.ReactionModel
 4 import bbParser.prefixes.Prefixes
 5
 6 class ReactionsBbParser : BbParser(
 7     { file ->
 8         file.readLines().map { line ->
 9             val strValues = line.split(',')
10             ReactionModel(
11                 strValues[0].trim().removePrefix(Prefixes.
                       REACTION).toInt(),
12                 strValues[1].trim().removePrefix(Prefixes.
                       TIMESTAMP).toLong()
13             )
14         }
15     }
16 )
```

Листинг 18: Файл Converter.kt

```
 1 package bbConverter
 2
```

```
 3 import bbParser.models.Model
 4 import java.util.Date
 5 import kotlin.collections.HashMap
 6
 7 interface Converter {
 8     fun convert(models: List<Model>): HashMap<Pair<Long, Long>,
         Int>
 9
10     companion object {
11         const val MILLIS_IN_MINUTE = 1000 * 60
12     }
13 }
```

Листинг 19: Файл ClicksConverter.kt

```
 1 package bbConverter
 2
 3 import bbParser.models.Model
 4 import bbParser.models.MouseClickModel
 5 import kotlin.math.sqrt
 6
 7 @Suppress("UNCHECKED_CAST")
 8 class ClicksConverter : Converter {
 9
10     /**
11      * Возвращаетзначенияввиде    :
12      * время "( первогоклика  - времяпоследнегоклика  ) -
          пройденноерасстояниевпикселях   "
13      */
14     override fun convert(clicks: List<Model>): HashMap<Pair<
         Long, Long>, Int> {
15         val out = HashMap<Pair<Long, Long>, Int>()
16
17         val sortedClicks = (clicks as List<MouseClickModel>).
             sortedBy { it.mTimestamp }
18
19         var prevClick: MouseClickModel
20         var curClick: MouseClickModel
21         for (i in 1 until sortedClicks.size) {
22             prevClick = sortedClicks[i - 1]
23             curClick = sortedClicks[i]
24             out[Pair(prevClick.mTimestamp, curClick.mTimestamp)
```

```
                ] =
25               sqrt(
26                   ((curClick.mXCoordinate - prevClick.
                        mXCoordinate) *
27                           (curClick.mXCoordinate - prevClick.
                              mXCoordinate) +
28                           (curClick.mYCoordinate - prevClick.
                              mYCoordinate) *
29                           (curClick.mYCoordinate - prevClick.
                              mYCoordinate)).toDouble()
30               ).toInt()
31         }
32
33         return out
34     }
35 }
```

Листинг 20: Файл KeysConverter.kt

```
 1 package bbConverter
 2
 3 import bbParser.models.KeyModel
 4 import bbParser.models.Model
 5 import kotlin.collections.HashMap
 6
 7 @Suppress("UNCHECKED_CAST")
 8 class KeysConverter : Converter {
 9
10     /**
11      * Возвращаетзначениявввиде    :
12      * время "( первогоклика  - времяпоследнегоклика  ) -
           количествовведённыхсимволов  "
13      */
14     override fun convert(keys: List<Model>): HashMap<Pair<Long,
           Long>, Int> {
15         val out = hashMapOf<Pair<Long, Long>, Int>()
16
17         val sortedKeys = (keys as List<KeyModel>).sortedBy { it
              .mTimestamp }
18
19         var currentTimestamp = sortedKeys.first().mTimestamp
20         var passedKeys = 1
```

```
21          for (i in 1 until sortedKeys.size) {
22              passedKeys++
23              if (sortedKeys[i].mTimestamp - currentTimestamp >
                    Converter.MILLIS_IN_MINUTE) {
24                  out[Pair(currentTimestamp, sortedKeys[i - 1].
                        mTimestamp)] = passedKeys
25                  passedKeys = 1
26                  currentTimestamp = sortedKeys[i].mTimestamp
27              }
28          }
29
30          out[Pair(currentTimestamp, sortedKeys.last().mTimestamp
                )] = passedKeys
31
32          return out
33      }
34  }
```

Листинг 21: Файл FuzzyStatus.kt

```
 1 package analyze.models
 2
 3
 4 data class FuzzyStatus(
 5     val isQuantized: Double = 0.0,
 6     val isBored: Double = 0.0,
 7     val isWorkable: Double = 0.0
 8 ) {
 9
10     fun isValid(): Boolean {
11         return (isQuantized > 0.0 || isBored > 0.0 ||
                isWorkable > 0.0) &&
12                 (isQuantized != isBored && isBored !=
                       isWorkable && isQuantized != isWorkable)
13     }
14 }
```

Листинг 22: Файл Clusterer.kt

```
1 package analyze.clusterization
2
3 import bbConverter.Converter
4 import bbParser.models.Model
```

```
 5 import org.apache.commons.math3.ml.clustering.
     FuzzyKMeansClusterer
 6
 7 abstract class Clusterer(val converter: Converter, k: Int = 3,
     fuzziness: Double = 5.0) {
 8
 9     val clusterer = FuzzyKMeansClusterer<ClusterPoint>(k,
         fuzziness)
10
11     abstract fun getClusters(fDimension: List<Model>,
         sDimension: List<Model>): List<List<Double>>
12
13     companion object {
14         const val MILLIS_IN_MINUTE = 1000 * 60
15     }
16 }
```

Листинг 23: Файл ClusterPoint.kt

```
 1 package analyze.clusterization
 2
 3 import org.apache.commons.math3.ml.clustering.Clusterable
 4
 5 class ClusterPoint(private val point: DoubleArray) :
     Clusterable {
 6
 7     override fun getPoint(): DoubleArray {
 8         return point
 9     }
10 }
```

Листинг 24: Файл BbClusterer.kt

```
 1 package analyze.clusterization
 2
 3 import bbConverter.Converter
 4 import bbParser.models.Model
 5 import bbParser.models.ReactionModel
 6
 7 class BbClusterer(converter: Converter, k: Int = 3, fuzziness:
     Double = 5.0) : Clusterer(converter, k, fuzziness) {
 8
 9     override fun getClusters(fDimension: List<Model>, reactions
```

```
        : List<Model>): List<List<Double>> {
10          val converted = converter.convert(fDimension)
11          reactions as List<ReactionModel>
12
13          val clusterable = mutableListOf<ClusterPoint>().apply {
14              reactions.forEach { reaction ->
15                  converted.entries.filter { filIt ->
16                      reaction.mTimestamp - filIt.key.first in
                            (0..(10 * MILLIS_IN_MINUTE))
17                  }.sortedBy { it.key.first }.forEach { innerIt
                        ->
18                      add(
19                          ClusterPoint(
20                              doubleArrayOf(
21                                  reaction.mReactionMillis.
                                        toDouble(),
22                                  (innerIt.value / (innerIt.key.
                                        second - innerIt.key.first).
                                        toDouble() * 1000 * 60)
23                              )
24                          )
25                      )
26                  }
27              }
28          }
29
30          clusterer.cluster(clusterable)
31          return clusterer.clusters.map { listOf(it.center.point
                [0], it.center.point[1]) }
32      }
33  }
```

Листинг 25: Файл Analyzer.kt

```
1  package analyze.analyzers
2
3  import bbParser.models.Model
4
5  interface Analyzer {
6
7      fun prepareModel(fDim: List<Model>, sDim: List<Model>):
           Boolean
```

```
 8
 9     fun analyzeByModel(values: List<Model>): List<Double>
10 }
```

Листинг 26: Файл BbAnalyzer.kt

```
 1 package analyze.analyzers
 2
 3 import analyze.clusterization.BbClusterer
 4 import analyze.models.FuzzyStatus
 5 import bbConverter.Converter
 6 import bbParser.models.Model
 7 import kotlin.math.abs
 8
 9 class BbAnalyzer(private val converter: Converter) : Analyzer {
10
11     private var fuzzyStatus = FuzzyStatus()
12
13     override fun prepareModel(fDim: List<Model>, reactions:
           List<Model>): Boolean {
14         val gotCenters = BbClusterer(converter).getClusters(
               fDim, reactions)
15
16         val centers = gotCenters.sortedBy { it.first() }.map {
               it[1] }
17
18         fuzzyStatus = FuzzyStatus(centers[0], centers[1],
               centers[2])
19
20         return true
21     }
22
23     override fun analyzeByModel(values: List<Model>): List<
           Double> {
24         if (!fuzzyStatus.isValid()) return emptyList()
25
26         var average = 0.0
27         converter.convert(values).entries.sortedBy { it.key.
               first }
28             .takeLast(NUMBER_OF_RECORDS_TO_BE_TAKEN)
29             .forEach { average += it.value }
30
```

```
31          average /= NUMBER_OF_RECORDS_TO_BE_TAKEN
32
33          return listOf(
34              abs(average - fuzzyStatus.isQuantized),
35              abs(average - fuzzyStatus.isBored),
36              abs(average - fuzzyStatus.isWorkable)
37          )
38      }
39
40      companion object {
41          const val NUMBER_OF_RECORDS_TO_BE_TAKEN = 5
42      }
43  }
```

Листинг 27: Файл BbFileAnalyzer.kt

```
 1 package analyze.fileAnalyzer
 2
 3 import analyze.analyzers.Analyzer
 4 import bbParser.parsers.BbParser
 5
 6 class BbFileAnalyzer(
 7     modelPathFileFDimension: String,
 8     modelPathFileSDimension: String,
 9     private val mMainParser: BbParser,
10     slaveParser: BbParser,
11     private val mAnalyzer: Analyzer
12 ) {
13
14     init {
15         val fDim = mMainParser.parseFile(
              modelPathFileFDimension)
16         val sDim = slaveParser.parseFile(
              modelPathFileSDimension)
17
18         mAnalyzer.prepareModel(fDim, sDim)
19     }
20
21     fun analyzeByFile(filepath: String): String {
22         val dim = mMainParser.parseFile(filepath)
23
24         val verdictList = mAnalyzer.analyzeByModel(dim)
```

```
25
26          return when (verdictList.indexOf(verdictList.minOrNull
                ())) {
27              0 -> "Quantized"
28              1 -> "Bored"
29              else -> "Workable"
30          }
31      }
32 }
```

Листинг 28: Файл analyze/main.kt

```
 1 package analyze
 2
 3 import analyze.analyzers.BbAnalyzer
 4 import bbConverter.KeysConverter
 5 import bbParser.parsers.KeysBbParser
 6 import bbParser.parsers.ReactionsBbParser
 7 import analyze.fileAnalyzer.BbFileAnalyzer
 8
 9 fun main() {
10
11     val fileAnalyzer = BbFileAnalyzer(
12         "${System.getProperty("user.dir")}/dataExample/
              test_Keys.txt",
13         "${System.getProperty("user.dir")}/dataExample/
              test_Reactions.txt",
14         KeysBbParser(), ReactionsBbParser(), BbAnalyzer(
              KeysConverter())
15     )
16
17     println(fileAnalyzer.analyzeByFile("${System.getProperty("
           user.dir")}/dataExample/test_Keys.txt"))
18 }
```

ПРИЛОЖЕНИЕ Б

Исходный код серверной части

Листинг 29: Реализация класса InluxServiceServer

```
        startKoin {
            modules(module {
                single { InfluxdbConfiguration() }

                single { CharRepositoryImpl() }

                single { DataService() }

                single { DataController() }
            })
        }
    }

    private val serverSocket = ServerSocket(socketPort)

    fun run() {
        getRuntime().addShutdownHook(Thread {
            println("Server on port ${serverSocket.localPort}
               stopped")
        })

        if (!serverSocket.isBound || serverSocket.isClosed) {
            throw SocketException("Server socket is already in
               use")
        }

        println("Server started on port ${serverSocket.
           localPort}")

        while (true) {
            val clientSocket = serverSocket.accept()
            thread {
                InfluxServiceClientHandler(clientSocket).run()
            }
        }
```

```
33     }
34 }
```

Листинг 30: Реализация класса InfluxServiceClientHadler

```
 1 class InfluxServiceClientHandler(private val clientSocket:
     Socket) {
 2     private val ydvpVersion = "0.1"
 3     private val defaultHeader = YdvpHeader("Server", "127.0.0.1
         ")
 4
 5     private val controller by inject<DataController>(
         DataController::class.java)
 6
 7     private fun anyResponse(code: String, explanation: String,
         body: Any): YDVP {
 8         return YDVP(
 9             YdvpStartingLineResponse(
10                 ydvpVersion,
11                 code,
12                 explanation
13             ),
14             listOf(defaultHeader),
15             GsonObject.gson.toJson(body)
16         )
17     }
18
19     private val badRequestResponse by lazy {
20         YDVP(
21             YdvpStartingLineResponse(ydvpVersion, "400", "BAD
                 REQUEST"),
22             listOf(defaultHeader)
23         )
24     }
25     private val internalServerErrorResponse by lazy {
26         YDVP(
27             YdvpStartingLineResponse(ydvpVersion, "500", "
                 INTERNAL SERVER ERROR"),
28             listOf(defaultHeader)
29         )
30     }
31
```

```
32      private val methodNotAllowed by lazy {
33          YDVP(
34              YdvpStartingLineResponse(ydvpVersion, "405", "
                    METHOD NOT ALLOWED"),
35              listOf(defaultHeader)
36          )
37      }
38
39      private val notFoundResponse by lazy {
40          YDVP(
41              YdvpStartingLineResponse(ydvpVersion, "404", "NOT
                    FOUND"),
42              listOf(defaultHeader)
43          )
44      }
45
46      private fun prepareUri(uri: String): List<String> {
47          val parsedUri = uri.split("/").toMutableList()
48
49          if (parsedUri[0] != "")
50              throw UnsupportedUriException("URI format error")
51
52          parsedUri.removeAt(0)
53          return parsedUri
54      }
55
56      private fun controllerPostMethod(uri: String, body: String)
            : YDVP {
57          val parsedUri = prepareUri(uri)
58
59          return when (parsedUri.first()) {
60              "data" -> {
61                  if (parsedUri.size < 2)
62                      throw UnsupportedUriException("Not enough
                            inline arguments")
63                  val response = controller.addData(
64                      parsedUri[1],
65                      GsonObject.gson.fromJson(body,
                            AcceptMeasurementsListDTO::class.java)
66                  )
67
```

```
68                anyResponse(response.statusCodeValue.toString()
                      , response.statusCode.name, response.body)
69            }
70            else -> throw UnsupportedUriException("Unsupported
                  URI")
71        }
72    }
73
74    private fun controllerGetMethod(uri: String, body: String):
          YDVP {
75        val parsedUri = prepareUri(uri)
76
77        return when (parsedUri.first()) {
78            "data" -> {
79                if (parsedUri.size < 2)
80                    throw UnsupportedUriException("Not enough
                          inline arguments")
81                val response =
82                    controller.getData(parsedUri[1], GsonObject
                          .gson.fromJson(body, Array<String>::
                          class.java).toList())
83
84                anyResponse(response.statusCodeValue.toString()
                      , response.statusCode.name, response.body)
85            }
86            else -> throw UnsupportedUriException("Way not
                  found")
87        }
88    }
89
90    private fun controllerWayByMethod(ydvpRequest: YDVP): YDVP
          {
91        ydvpRequest.startingLine as YdvpStartingLineRequest
92        val method = ydvpRequest.startingLine.method
93        val uri = ydvpRequest.startingLine.uri
94
95        val body = ydvpRequest.body
96
97        return try {
98            when (method) {
99                "GET" -> controllerGetMethod(uri, body)
```

```
100                 "POST" -> controllerPostMethod(uri, body)
101                 else -> methodNotAllowed
102             }
103         } catch (exc: NullPointerException) {
104             badRequestResponse
105         } catch (exc: UnsupportedUriException) {
106             notFoundResponse
107         } catch (exc: Exception) {
108             println("EXCEPTION")
109             println(exc.javaClass)
110             println(exc.localizedMessage)
111             internalServerErrorResponse
112         }
113     }
114
115     fun run() {
116         println("Accepted client on ${clientSocket.
                localSocketAddress} from ${clientSocket.
                remoteSocketAddress}")
117
118         val bufferedReader = BufferedReader(InputStreamReader(
                clientSocket.getInputStream()))
119
120         var gotRequest = bufferedReader.readLine() + "\n"
121         while (bufferedReader.ready())
122             gotRequest += bufferedReader.readLine() + "\n"
123
124         val clientOut = PrintWriter(clientSocket.
                getOutputStream(), true)
125
126         val ydvpRequest = try {
127             YdvpParser().parseRequest(gotRequest)
128         } catch (exc: Exception) {
129             clientOut.println(badRequestResponse)
130
131             return
132         }
133
134         val response = controllerWayByMethod(ydvpRequest).
                createStringResponse()
135         println("Returned to client:\n$response")
```

```
136         clientOut.println(response)
137     }
138 }
139
140 class InfluxServiceServer(socketPort: Int) {
141     init {
```

Листинг 31: 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()
 9 }
```

Листинг 32: Пример организации клиентской части

```
 1 package examples.helloexample
 2
 3 import client.InfluxServiceClient
 4 import domain.dtos.AcceptMeasurementsDTO
 5 import domain.dtos.AcceptMeasurementsListDTO
 6 import domain.dtos.MeasurementDataWithoutTime
 7 import domain.dtos.ResponseMeasurementsDTO
 8 import gson.GsonObject
 9 import protocol.YDVP
10 import protocol.YdvpHeader
11 import protocol.YdvpStartingLineRequest
12 import protocol.YdvpStartingLineResponse
13 import java.net.ConnectException
14
15 val measurementsToSend = AcceptMeasurementsListDTO(
16     listOf(
17         AcceptMeasurementsDTO(
18             "pulse", listOf(
19                 MeasurementDataWithoutTime("30"),
20                 MeasurementDataWithoutTime("40")
21             )
22         ),
```

```
23          AcceptMeasurementsDTO(
24              "botArterialPressure", listOf(
25                  MeasurementDataWithoutTime("40"),
26                  MeasurementDataWithoutTime("50")
27              )
28          ),
29          AcceptMeasurementsDTO(
30              "topArterialPressure", listOf(
31                  MeasurementDataWithoutTime("80"),
32                  MeasurementDataWithoutTime("90")
33              )
34          )
35      )
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"))
75              )
76          )
77      }
78 }
79
80 fun main() {
81      sendTest()
82 //     getTest()
83 }
```

ПРИЛОЖЕНИЕ В

Примеры полученных от пользователя данных

Листинг 33: Первые 40 строк данных анализа действий с клавиатурой

```
 1 key=G, timestamp=1652607722497
 2 key=T, timestamp=1652607722677
 3 key=H, timestamp=1652607722806
 4 key=D, timestamp=1652607722933
 5 key=F, timestamp=1652607723101
 6 key=Z, timestamp=1652607723242
 7 key=Slash, timestamp=1652607723511
 8 key=Space, timestamp=1652607723627
 9 key=F, timestamp=1652607723781
10 key=D, timestamp=1652607724434
11 key=F, timestamp=1652607724589
12 key=Y, timestamp=1652607724757
13 key=U, timestamp=1652607725089
14 key=F, timestamp=1652607725218
15 key=H, timestamp=1652607725448
16 key=L, timestamp=1652607725629
17 key=Slash, timestamp=1652607725834
18 key=Space, timestamp=1652607725925
19 key=E, timestamp=1652607726066
20 key=Semicolon, timestamp=1652607726156
21 key=T, timestamp=1652607726260
22 key=Space, timestamp=1652607726361
23 key=D, timestamp=1652607726631
24 key=S, timestamp=1652607726734
25 key=H, timestamp=1652607726876
26 key=B, timestamp=1652607727017
27 key=C, timestamp=1652607727094
28 key=J, timestamp=1652607727210
29 key=D, timestamp=1652607727326
30 key=S, timestamp=1652607727466
31 key=D, timestamp=1652607727554
32 key=F, timestamp=1652607727747
33 key=K, timestamp=1652607727888
34 key=F, timestamp=1652607727978
35 key=C, timestamp=1652607728131
```

```
36 key=M, timestamp=1652607728209
37 key=Space, timestamp=1652607728310
38 key=Y, timestamp=1652607728477
39 key=F, timestamp=1652607728541
40 key=Space, timestamp=1652607728632
```

Листинг 34: Первые 40 строк данных анализа действий с мышью

```
 1 x=994, y=570, key=1, timestamp=1652607697196
 2 x=993, y=618, key=1, timestamp=1652607698841
 3 x=884, y=553, key=1, timestamp=1652607700152
 4 x=1005, y=503, key=1, timestamp=1652607707841
 5 x=1150, y=454, key=1, timestamp=1652607708860
 6 x=2848, y=1, key=1, timestamp=1652607710663
 7 x=3165, y=129, key=1, timestamp=1652607712493
 8 x=3063, y=478, key=1, timestamp=1652607714309
 9 x=1990, y=692, key=1, timestamp=1652607717712
10 x=2665, y=596, key=1, timestamp=1652607718976
11 x=2834, y=294, key=1, timestamp=1652607744520
12 x=2094, y=198, key=1, timestamp=1652607745173
13 x=3076, y=472, key=1, timestamp=1652607746519
14 x=1980, y=676, key=1, timestamp=1652607756266
15 x=2540, y=607, key=1, timestamp=1652607761777
16 x=2789, y=392, key=1, timestamp=1652607767999
17 x=2511, y=379, key=1, timestamp=1652607775627
18 x=2772, y=475, key=1, timestamp=1652607836448
19 x=870, y=340, key=1, timestamp=1652607842495
20 x=13, y=549, key=1, timestamp=1652607845704
21 x=2658, y=483, key=1, timestamp=1652607848005
22 x=2744, y=586, key=1, timestamp=1652607905944
23 x=2710, y=604, key=1, timestamp=1652607907428
24 x=2657, y=425, key=1, timestamp=1652607908124
25 x=2094, y=288, key=1, timestamp=1652607914433
26 x=3696, y=461, key=1, timestamp=1652607918059
27 x=2142, y=431, key=1, timestamp=1652607919348
28 x=3388, y=546, key=1, timestamp=1652607920515
29 x=3381, y=563, key=1, timestamp=1652607921828
30 x=3575, y=444, key=1, timestamp=1652607923673
31 x=3575, y=444, key=1, timestamp=1652607925103
32 x=3385, y=741, key=1, timestamp=1652607927127
33 x=2656, y=417, key=1, timestamp=1652607928988
34 x=2642, y=593, key=1, timestamp=1652607929821
```

```
35 x=2116, y=117, key=1, timestamp=1652607991163
36 x=1950, y=51, key=1, timestamp=1652607994926
37 x=2004, y=268, key=1, timestamp=1652608003053
38 x=3724, y=463, key=1, timestamp=1652608011495
39 x=893, y=572, key=1, timestamp=1652608014592
40 x=2627, y=520, key=1, timestamp=1652608016682
```