Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт
Кафедра вычислительной техники
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой
_____О.В.Непомнящий
подпись инициалы, фамилия
«___» _____2022г.

БАКАЛАВРСКАЯ РАБОТА

09.03.01 – Информатика и вычислительная техника
код – наименование направления

Система уведомлений для модемов компании “Систематикс”
тема

Руководитель	_____	_____	Л. И. Покидышева
	подпись, дата	должность, ученая степень	инициалы, фамилия
Студент	_____		Д. А. Веселков
	подпись, дата		инициалы, фамилия

Красноярск 2022

РЕФЕРАТ

Выпускная квалификационная работа по теме “Система уведомления для модемов компании Систематикс” содержит 40 страниц текстового документа, 1 приложение, 11 использованных источников, 19 рисунков.

МОДЕМ, ALERTD, СИСТЕМАТИКС, ДЕМОН, УВЕДОМЛЕНИЯ, API.

Цель работы: разработка системы уведомлений для своевременного информирования пользователей об изменении состояний параметров на устройствах компании “Систематикс”.

В результате выполнения ВКР было разработано приложение «Alertd».

В выпускную квалификационную работу входит введение, 3 главы и заключение.

Во введении ставится цель и выполняется ее декомпозиция на задачи.

В первой главе рассматриваются аналоги, формулируются основные требования к разрабатываемому приложению и выбираются инструменты разработки.

Во второй главе разрабатываются: архитектура системы, структура базы данных, структуры данных и алгоритм сбора сконфигурированных уведомлений в память приложения.

В третьей главе описана реализация основных элементов приложения.

В заключении подводятся итоги по выполненной работе.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Интернет в современном мире позволяет каждому желающему в любой момент обратиться к неограниченной базе знаний, чтобы совершенствоваться и открывать для себя что-то новое. По мере роста пропускающих способностей, стали появляться ресурсы иного характера, такие как: youtube.com, twitch.tv и т. д., где мы можем потреблять как развлекательный, так и познавательный контент в превосходном качестве и без задержек.

Но для того, чтобы пользоваться этими ресурсами, нужны некоторые устройства. Это может быть стационарный роутер (маршрутизатор) или телефон с GSM (Global System for Mobile Communications) модулем, который может выходить в сеть посредством сотовой связи (Cellular Network).

Проблема заключается в том, что роутер нельзя взять с собой, так как он соединен с сетью провайдера и, за счет этого, маршрутизируется в глобальную сеть. Телефон же, с одним GSM модулем, редко может дать желаемую скорость интернет-подключения, а где-то, и вовсе, какую бы то ни было скорость. Это связано с тем, что разным операторам предоставлены разные базовые станции, и зоны покрытия сотовой связи отличаются от провайдера к провайдеру.

Эту проблему и решает продукт компании «Систематикс» - Мульти Модем (Мульти Роутер), далее ММ. ММ объединяет в себе всю вариативность настроек обычного домашнего роутера (DHCP, бриджевание интерфейсов, NAT, Firewall и др.) и 4 LTE (Long-Term Evolution) модуля, которые устанавливаются в современные мобильные устройства.

Благодаря тому, что есть возможность использовать до 4 SIM-карт, зона доступности сотовой связи увеличивается, а скорость достигается за счет агрегации каналов связи нескольких SIM-карт.

Покупателями подобных устройств зачастую являются юридические лица, такие как владельцы заправочных станций или передвижных медицинских центров.

Так как устройства позволяют пользователям выходить в интернет практически из любой местности, где ММ попадает в зону покрытия как минимум одного из операторов, за ними требуется постоянный мониторинг. В случае сбоев или полного выхода ММ из строя, заказчик, по понятным причинам, будет недоволен.

Для того, чтобы минимизировать подобные случаи, на помощь приходят системы мониторинга, которые, с некоторой периодичностью проверяют состояние устройств, и в случае, если, какой-нибудь из параметров достигнет критического значения, отправляют уведомление техподдержке, или, непосредственно, пользователю, чтобы, после согласования, оперативно устранить неполадку.

Целью работы является разработка ПО, которое позволит следить за состояниями ММ и уведомлять пользователя или техподдержку о сбоях в работе устройств. Приложение представляет собой демона, задача которого заключается в постоянном мониторинге базы данных на предмет каких-либо изменений в состоянии Модема или группы Модемов, в зависимости от выбора пользователя. Для достижения цели в работе решаются следующие **задачи**:

- проанализировать техническое задание на разработку приложения «Alertd», основываясь на прецедентах. Выполнить проектирование и разработку ПО.

1 Анализ задания

Необходимо разработать приложение для уведомления пользователей о состоянии устройств, при помощи конфигурации объектов уведомлений на сайте мониторинга [1]. Для обеспечения конкурентоспособности приложения, выполнен анализ сильных и слабых сторон аналогичных приложений. С их учетом составлены спецификации требований к серверной части на разработку. Также, проанализирован ряд проектов с открытым исходным кодом на предмет возможности их доработки.

1.1 Анализ существующих аналогов

Прямым конкурентом разрабатываемого ПО Alertd, является сервис Zabbix [2]. Zabbix — свободная система мониторинга и отслеживания статусов разнообразных сервисов компьютерной сети, серверов и сетевого оборудования. Zabbix имеет масштабируемый функционал на более высоком уровне абстракции, нежели Alertd, но между ними все же можно провести параллель.

Как и Alertd, Zabbix умеет анализировать состояние узлов сети и уведомлять пользователей о каких-либо изменениях в указанные каналы связи, будь то канал в социальной сети «Телеграм» или электронная почта. Данные о состоянии объекта Zabbix получает посредством централизованного узла сети, где находится серверная сторона сервиса, и агента, то есть клиентской части на устройствах, за которыми необходимо следить. Полезные данные, а именно текущие состояния выбранных для отслеживания параметров, передаются между клиентом и сервером по протоколу SNMP (Simple Network Management Protocol) [3].

Проблема заключается в том, что на ММ имеется собственное ПО, а именно его клиентская часть, которая передает зашифрованные MD5 методом шифрования, состояния устройства на сервер, которые в дальнейшем записываются в базу данных MariaDB. Так как передавать идентичные данные

разными методами неэффективно, а избавляться от хорошо отлаженного кода не хотелось, было принято решение отказаться от сервиса Zabbix в пользу разработки собственной системы уведомлений – Alertd.

1.2 Спецификация требований на приложение

1.2.1 Функциональные требования

Приложение Alertd является составной частью целой платформы для мониторинга параметров Мульти Модемов, который базируется на физическом сервере внутри виртуальной машины под управлением системы виртуализации ProxMox и KVM гипервизора.

Для корректного функционирования приложения, требуется реализовать API запросы на сайте мониторинга, чтобы пользователи могли создавать, просматривать и удалять уведомления.

API запросы предоставляют следующий функционал:

- Получение из БД списка уведомлений для конкретного пользователя;
- Изменение статуса отдельно взятого уведомления (следить или не следить за выбранными, в параметрах уведомления, устройствами);
- Удаление уведомления;
- Получение из БД каналов связи, для отправки уведомления пользователю
- Сохранение или изменение параметров уведомления (реализуется одним API запросом);
- Добавление новых каналов связи;
- Удаление каналов связи;
- Получение списка триггеров (параметров устройства, за которыми необходимо следить).

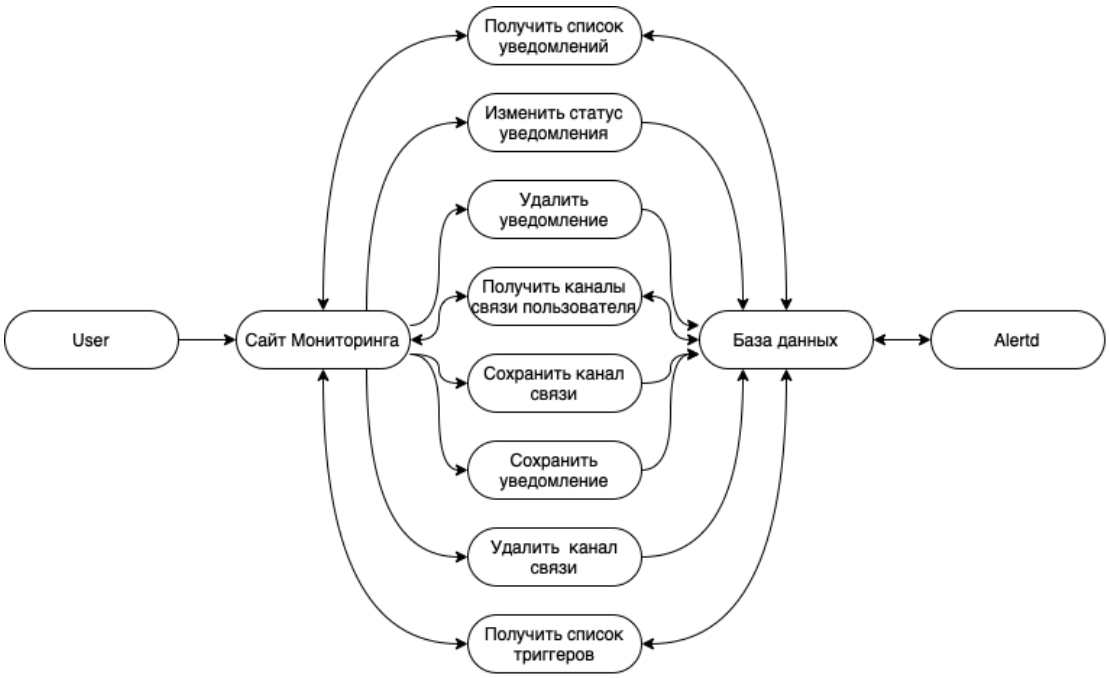

Рисунок 1 — Диаграмма API вызовов для взаимодействия пользователя с «Alertd»

При дискретизации задачи по разработке системы уведомлений «Alertd» были выделены следующие подзадачи:

- Демон должен получать значения проверяемых параметров из БД;

- Демон должен отправлять уведомление пользователю в канал связи в случае, если значение параметра достигло критической отметки или вернулось в норму;

- Демон должен уметь оповещать пользователя сразу в несколько каналов связи, указанных в одном уведомлении;

- Должен быть, как минимум, один тип канала связи (Телеграм);

- Приложение должно разворачиваться внутри Docker контейнера [4];

- Количество копий, запущенных экземпляров(инстансов) «Alertd», может быть больше одного;

- Количество уведомлений, обрабатываемых одной копией приложения ограничено;

- Демон, с некоторой периодичностью, должен проверять базу данных на наличие новых, созданных пользователем, уведомлений или изменение настроек старых;

- Демон должен, в зависимости от проверяемых параметров, создавать обрабатывающий контроллер, задача которого заключается в получении значения параметра и вынесении вердикта, перешел ли триггер в фазу критического состояния, или, наоборот, вернулся в нормальное состояние;

- Приложение должно быть расширяемым. Это значит, что сценариев отправки уведомления может быть множество, в зависимости от реализации контроллера и какого рода параметры он обрабатывает.

1.2.2 Структуры данных приложения

Приложение должно состоять из следующих структур.

- Уведомление (**Notification**) – непосредственно главный класс, объекты которого хранят в себе **ID**, для идентификации уведомления. **Status**, который определяет, стоит ли следить за этим уведомлением или нет. **Name** – имя уведомления, конфигурируемое пользователем. **Daemon_ID** – идентификатор копии запущенного демона, который обрабатывает данное уведомление. **Triggers** – параметры, которые требуется отслеживать на устройствах, указанных в уведомлении, **Devices** – устройства, об изменении состояния которых следует уведомлять пользователя.

- Триггер (**NotificationTrigger**) – объекты этого класса служат для того, чтобы стать своего рода промежуточным звеном между объектом уведомления и объектом контроллера. Состоит эта структура из поля **Notif_ID**, которое позволяет демону понимать, что некоторый объект триггера привязан к конкретному уведомлению. **Wait_Timer** – свойство, обозначающее, через какое время следует отправить уведомление в случае; если в течение этого времени значение параметра вернулось в исходное, отправка уведомления отменяется, это сделано для того, чтобы избежать ложных срабатываний. **Crit** и **Normal** –

значения, которые и определяют что считать критическим состоянием, а что нормальным. **ID** – идентификатор триггера. **Definition_ID** – идентификатор описания триггера, именно он указывает на то, где в БД искать параметры для слежения, какого типа будут отслеживаемые данные и какой формат будет принимать сообщение для отправки уведомления. **TriggerDefinition** – сам объект описания триггера, который несет все вышеописанные данные. **Controller** – объект абстрактной структуры данных, благодаря которой и достигается расширяемость приложения, за счет возможности обрабатывать абсолютно разные типы данных в различных БД. **Channels** – каналы связи, в которые нужно отправить уведомление, в случае изменения состояния.

- Контроллер (**Controller**) – абстрактный класс, в подчинении у которого лишь одна функция получения текущего значения параметра для каждого из устройств, реализация которой выбирается, в зависимости от описания триггера, его типа данных или особенности хранения параметров устройства в БД.

На Рис. 2 можно увидеть алгоритм работы программы. Инициализация, в данном случае, подразумевает получение данных для работы демона из виртуального окружения внутри контейнера. Таких, как имя хостов баз данных для подключения, пороли и порты.

Таймаут предусмотрен для того, чтобы приложение не съедало все процессорное время гипервизора.

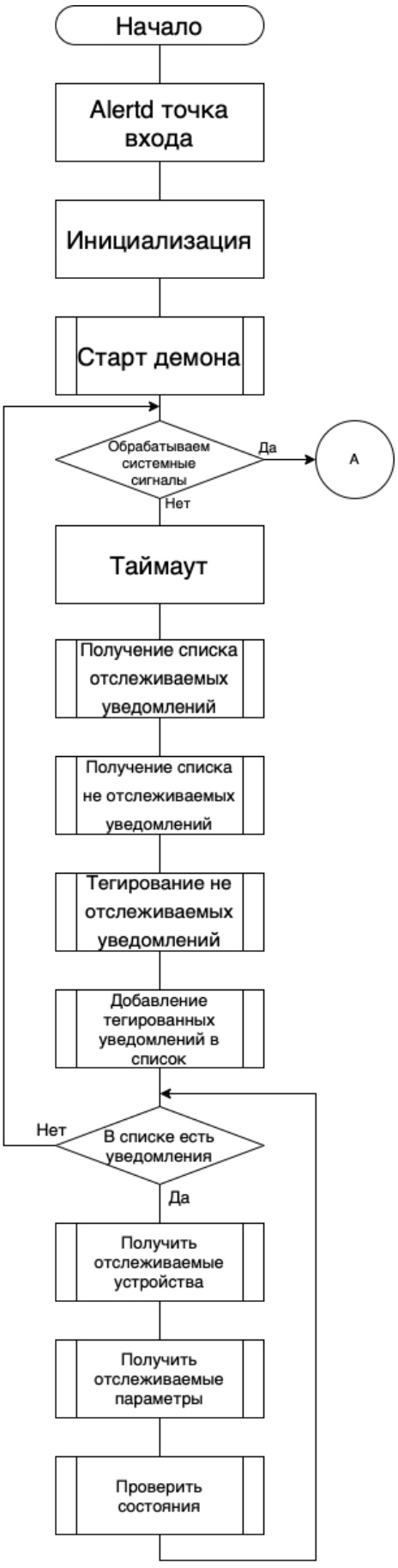

Рисунок 2 — Диаграмма алгоритма приложения «Alertd»

Завершение приложения предусмотрено за счет обработчика системных сигналов, который приведен на Рис. 2 (продолжение)

Рисунок 2 (продолжение) – Алгоритм обработки системных сигналов

В алгоритме обработки системных сигналов отлавливается лишь один сигнал SIGTERM, так как, в отличие от SIGKILL, программное обеспечение может его обработать или проигнорировать. После получения иных сигналов такой возможности нет. Данный сигнал можно послать в “Alertd” при помощи Docker агента, так как ПО запускается внутри Docker контейнера.

1.4 Выбор инструментов

Приложение разрабатывается на языке GoLang, выбор обусловлен тем, что данный язык является компилируемым и хорошо поддерживаемым сообществом, в связи с чем можно прибегать к помощи сторонних библиотек, таких, например, как pprof [5], для оптимизации ПО.

Приложение разворачивается внутри виртуального контейнера на базе Docker, что обеспечивает отказоустойчивость хостовой системы и безопасность, передаваемых между контейнерами данных.

В качестве баз данных используется MariaDB (необъемные данные) и Clickhouse (объемные данные).

1.5 Выводы по главе

В ходе написания главы были проанализированы аналоги на рынке, сформулирована цель работы, разработана спецификация требований для приложения «Alertd».

2 Этапы проектирования

2.1 Прецеденты

Прецедент 1. Получить список уведомлений (Рис. 3).

Цель сценария: Увидеть список сконфигурированных ранее уведомлений.

Предусловия: Пользователь открыл вкладку “Notification”.

Основной сценарий: Отправляется запрос на сервер, возвращаются объекты уведомлений с информацией о выбранных устройствах, триггерах и каналах связи.

Условия ввода в действие альтернативных сценариев

Условие 1. Сервер вернул ошибку. Отображается всплывающая плашка с описанием ошибки.

Условие 2. Уведомления не найдены. Отображается кнопка создания уведомления (Рис. 4).

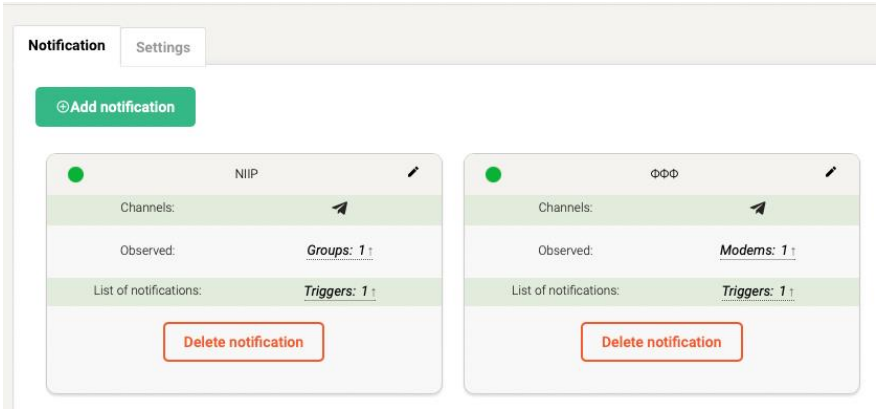

Рисунок 3 – Ответ с созданными ранее уведомлениями

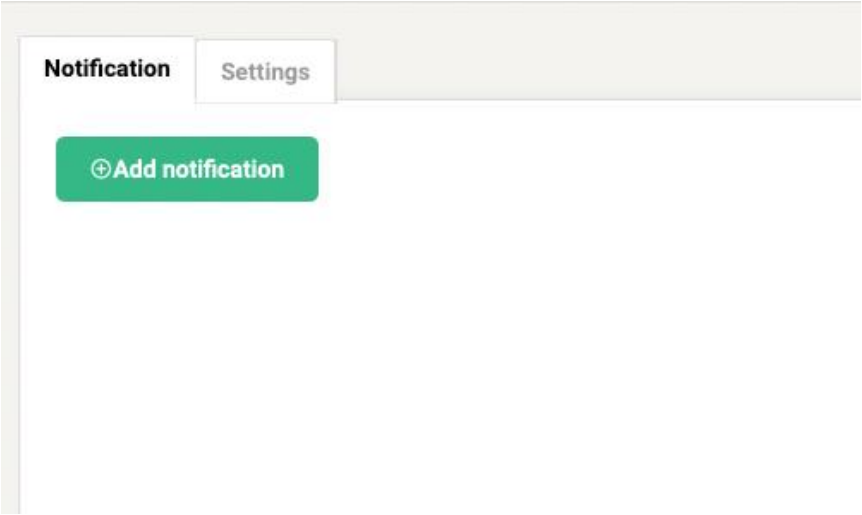

Рисунок 4 – Ответ, в котором не содержится уведомлений

Прецедент 2. Изменить статус уведомления.

Цель сценария: Активировать или деактивировать отправку уведомления.

Предусловия: Пользователь открыл окно редактирования уведомлений.

Основной сценарий: Отправляется запрос на сервер со значением статуса уведомления.

Условия ввода в действие альтернативных сценариев

Условие 1. Сервер вернул ошибку. Отображается всплывающая плашка с описанием ошибки.

Прецедент 3. Удалить уведомление.

Цель сценария: Удалить существующее уведомление.

Предусловия: Пользователь нажал кнопку “Delete notification”.

Основной сценарий: Отправляется запрос на сервер с идентификатором уведомления и идентификатором пользователя.

Условия ввода в действие альтернативных сценариев

Условие 1. Сервер вернул ошибку. Отображается всплывающая плашка с описанием ошибки.

Прецедент 4. Получить каналы связи пользователя (Рис. 5).

Цель сценария: Отобразить пользователю сконфигурированные им каналы связи.

Предусловия: Пользователь открыл вкладку “Settings”.

Основной сценарий: Отправляется запрос на сервер, возвращаются объекты каналов.

Условия ввода в действие альтернативных сценариев

Условие 1. Сервер вернул ошибку. Отображается всплывающая плашка с описанием ошибки.

Условие 2. Каналы не найдены. Отображается кнопка создания канала связи (Рис. 6).

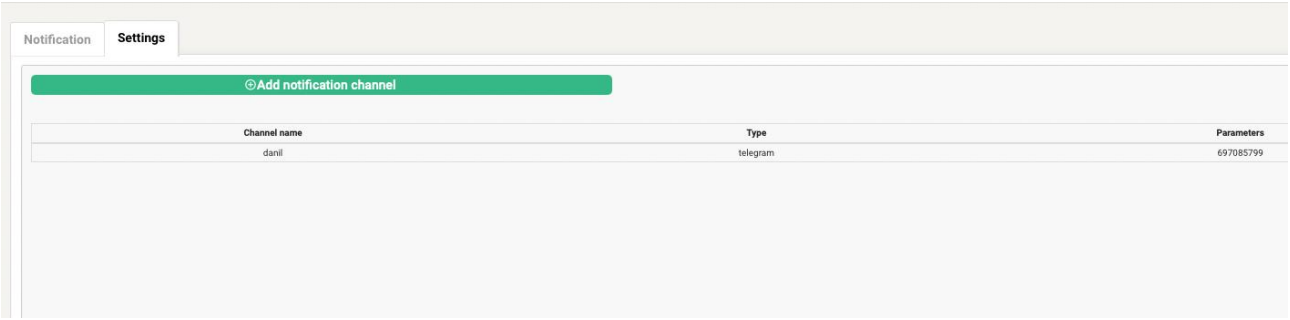

Рисунок 5 – Ответ с созданным ранее каналом связи

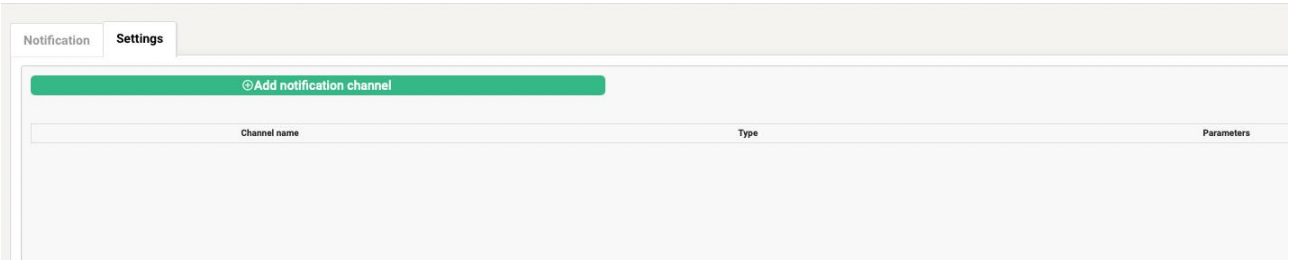

Рисунок 6 – Ответ, в котором не содержатся каналы связи

Прецедент 5. Сохранить канал связи.

Цель сценария: Сохранить канал связи пользователя.

Предусловия: Пользователь сконфигурировал канал связи и нажал кнопку “Save”.

Основной сценарий: Отправляется запрос на сервер с данными о канале связи и идентификатором пользователя.

Условия ввода в действие альтернативных сценариев

Условие 1. Сервер вернул ошибку. Отображается всплывающая плашка с описанием ошибки.

Прецедент 6. Сохранить уведомление.

Цель сценария: Сохранить уведомление пользователя.

Предусловия: Пользователь сконфигурировал уведомление и нажал кнопку “Save”.

Основной сценарий: Отправляется запрос на сервер с данными об уведомлении и идентификатором пользователя.

Условия ввода в действие альтернативных сценариев

Условие 1. Сервер вернул ошибку. Отображается всплывающая плашка с описанием ошибки.

Прецедент 7. Удалить канал связи.

Цель сценария: Удалить канал связи пользователя.

Предусловия: Пользователь открыл окно редактирования канала связи и нажал кнопку “Delete”.

Основной сценарий: Отправляется запрос на сервер с идентификатором канала связи и идентификатором пользователя.

Условия ввода в действие альтернативных сценариев

Условие 1. Сервер вернул ошибку. Отображается всплывающая плашка с описанием ошибки.

Прецедент 8. Получить список триггеров.

Цель сценария: Отобразить список триггеров для выбора при конфигурировании/изменении уведомления.

Предусловия: Пользователь открыл вкладку “Notification”.

Основной сценарий: Отправляется запрос на сервер, возвращаются доступные объекты триггеров.

Условия ввода в действие альтернативных сценариев

Условие 1. Сервер вернул ошибку. Отображается всплывающая плашка с описанием ошибки.

2.2 База данных

Таблица **notification**:

- id – идентификатор уведомления;
- uid – идентификатор пользователя на сайте мониторинга;
- status – статус уведомления (активное или нет);
- name – имя уведомления;
- daemon_id – идентификатор инстанса “Alertd”, который обрабатывает указанное уведомление.

Данная таблица описывает объекты уведомлений. При помощи таблицы **notification,** front-end часть получает информацию, какие уведомления сконфигурированы для того или иного пользователя.

Таблица **notification_channels**:

- id – идентификатор канала связи;
- name – человекочитаемое имя канала связи;
- user_id – идентификатор пользователя, которому принадлежит канал;
- type – тип канала связи (telegram, email …);

- params – необходимые параметры для отправки уведомления (идентификатор telegram чата, email …).

Данная таблица описывает канал связи, по которому будут отправлены соответствующие уведомления.

Таблица **notification_device**:

- notif_id – идентификатор уведомления;
- device_id – идентификатор устройства, за состоянием которого необходимо следить.

Данная таблица описывает отношения уведомлений и устройств, сообщение об изменении состояния которых требуется отправить.

Таблица **notification_group**:

- notif_id – идентификатор уведомления;
- group_id – идентификатор группы устройств, за состояниями которых необходимо следить.

Данная таблица описывает отношения уведомлений и групп устройств, сообщение об изменении состояния которых требуется отправлять. “Alertd” разворачивает указанный **group_id** в массив **device_id** и следит за каждым устройством в отдельности.

Таблица **notification_queue**:

- id – идентификатор уведомления в очереди на отправку;
- type – тип канала связи (telegram, email …);
- sent – флаг, указывающий, было ли отправлено сообщение, или нет;
- text – текст сообщения;

- receiver – получатель сообщения (дублирует поле **params** в таблице **notification_channels**);
- send_ts – временная метка, обозначающая время отправки сообщения.

Данная таблица описывает очередь сообщений для отправки. После удачной отправки уведомления флаг **sent** становится равен 1.

Таблица **notification_trigger**:

- id – идентификатор триггера;
- notif_id – идентификатор уведомления;
- definition_id – идентификатор описания триггера;
- params – параметры, которые необходимы для корректного поведения триггера.

Данная таблица описывает объекты триггеров.

Таблица **notification_trigger_definition**:

- id – идентификатор описания триггера;
- params – данные, которые описывают местоположения параметра устройства, за которым требуется следить и тип контроллера, который будет производить обработку этих данных;
- ui_params – параметры для отрисовки front-end части приложения;
- valid_for – параметр, определяющий, к каким объектам может быть применен триггер (device, group. all);
- text – сырое сообщение уведомления с плейсхолдерами, которые будут заменены на валидные данные к моменту попадания сообщения в очередь (Например: “Значение температуры мат. платы на модеме $fid достигло критического: $value”).

Данная таблица включает в себя объекты определения триггеров, их поведение и пути к значениям параметров устройства.

Таблица **notification_trigger_state**:

- trigger_id – идентификатор триггера, который отвечает за обработку параметра устройства;
- device_id – идентификатор устройство, чье состояние проверяется;
- state – состояние триггера (отработал или нет);
- queue_id – идентификатор положения в очереди (NULL, если записей в очереди нет или все сообщения отправлены);
- value – поле, обозначающие количество записей (нужно для контроллера, который уведомляет о новых записях в указанной таблице БД).

Данная таблица описывает состояния триггеров Alertd периодически, через некоторый промежуток времени, проверяет эту таблицу, на предмет несоответствия значения поля **state,** и состояние параметра устройства. Если такое несоответствие было найдено, демон поставит уведомление в очередь на отправку.

Исходя из имеющихся таблиц, первичных и внешних ключей, была составлена ER диаграмма, на которой изображены отношения между моделями БД (Рис. 7).

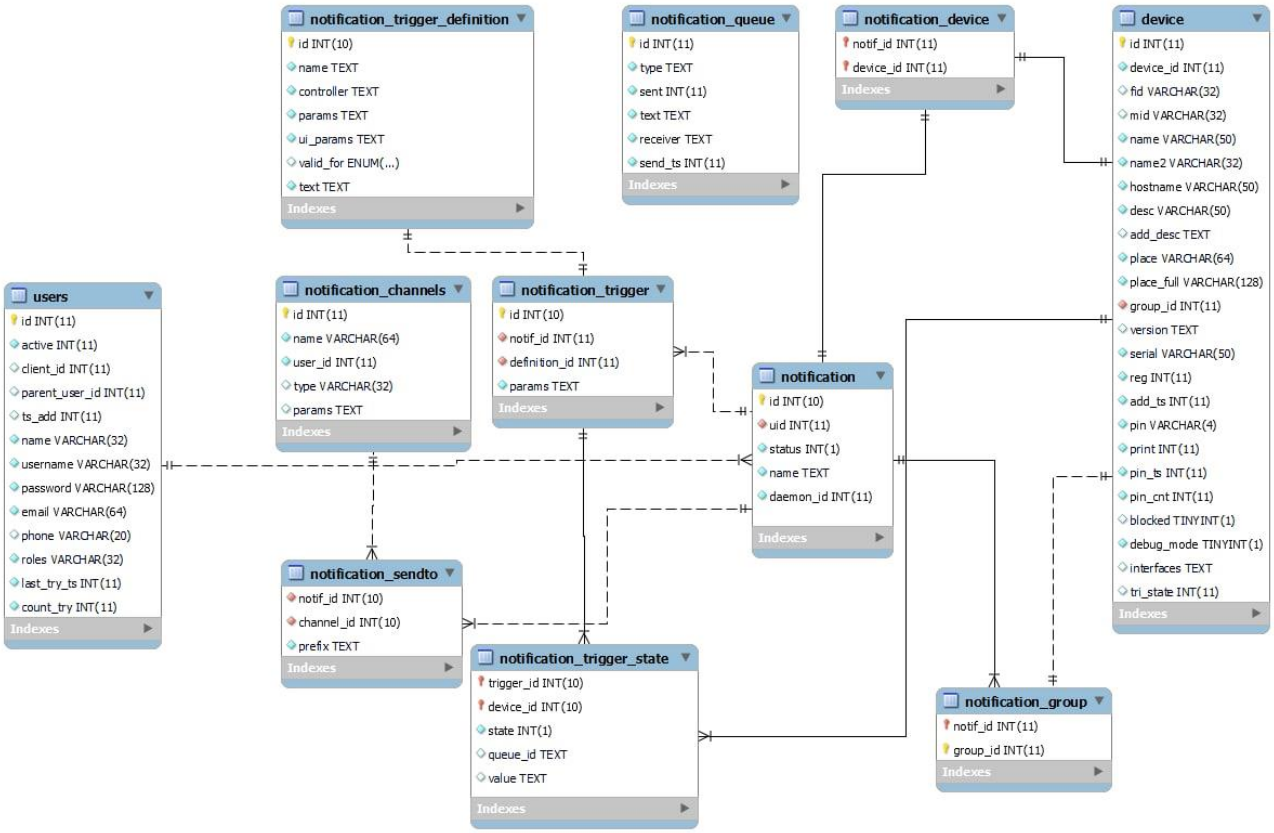

Рисунок 7 – ER Диаграмма приложения Alertd

2.3 Алгоритмы приложения

2.3.1 Алгоритм получения текущего состояния триггеров

Каждая запись состояния триггера включает в себя идентификатор обрабатывающего триггера, устройство, параметр которого он обрабатывает и состояние – сработал триггер или нет. Количество таких записей будет равно количеству устройств, сконфигурированных пользователем при создании уведомления (Рис. 8).

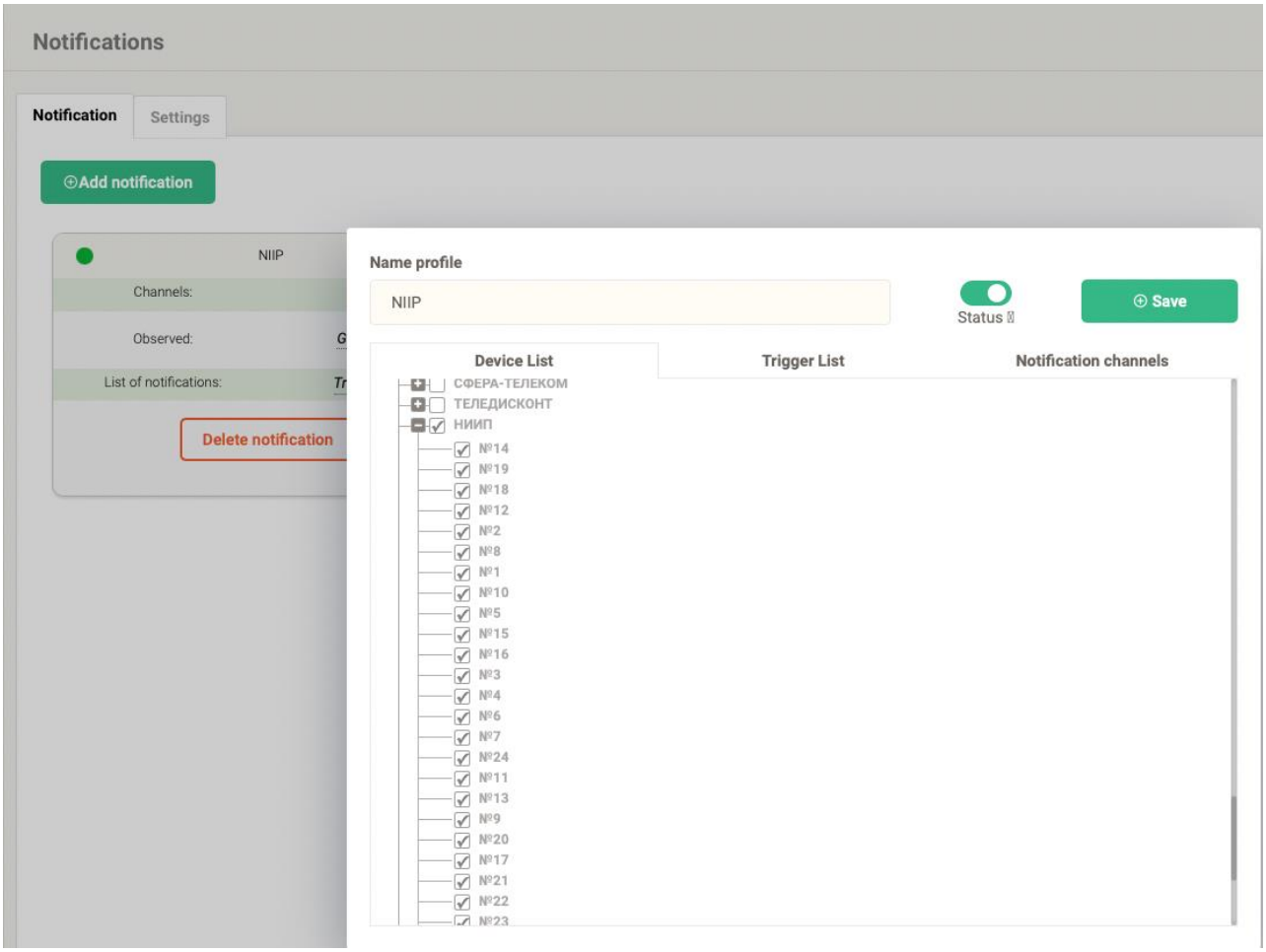

Рисунок 8 – Вкладка выбора устройств при создании уведомления

На Рис. 9 можно видеть пример конфигурации триггера и его критического значения. В данном случае триггер отвечает за проверку канала связи между устройством и сайтом мониторинга – “Modem On/Off”.

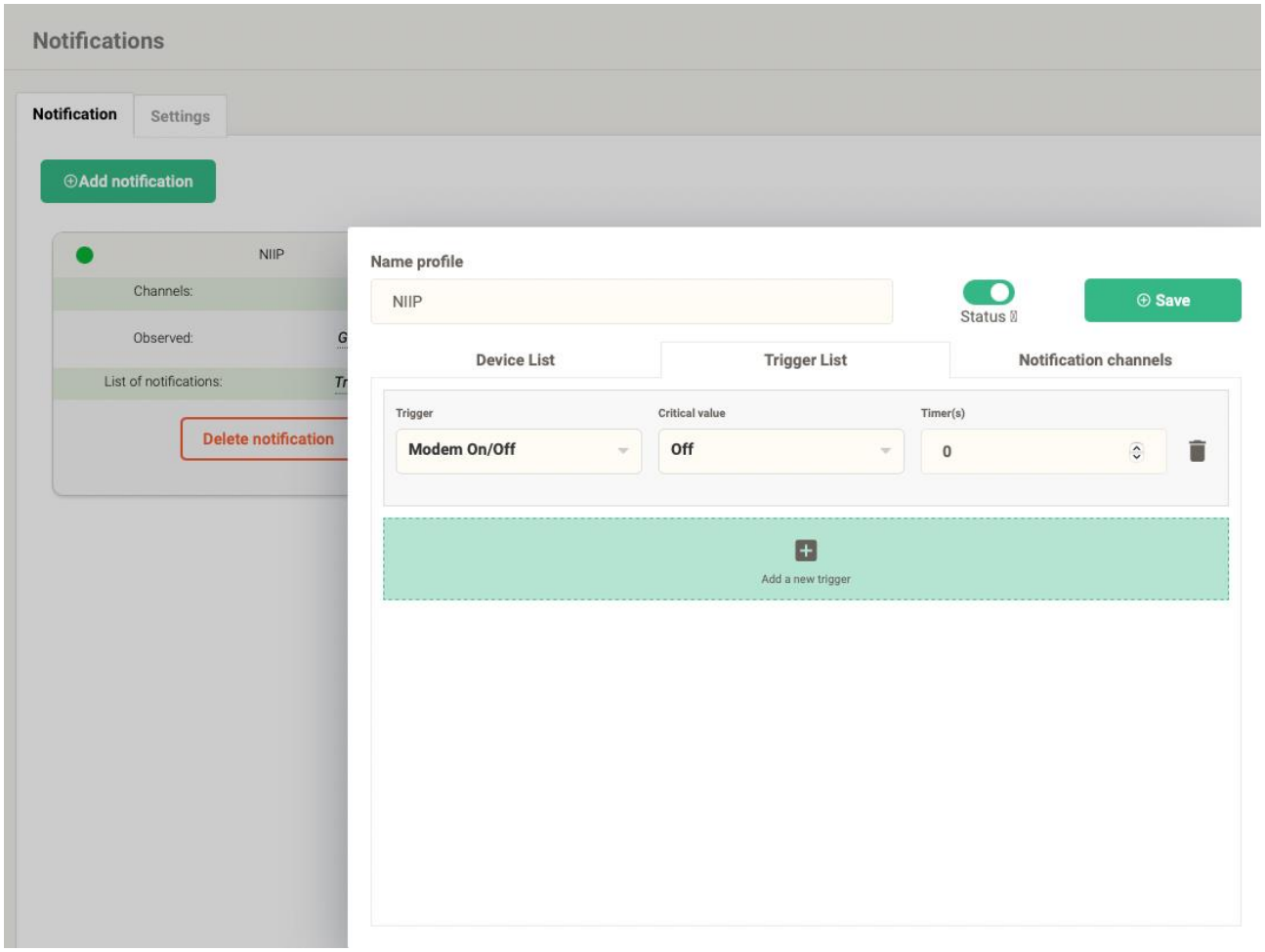

Рисунок 9 – Вкладка создания параметров для отслеживания (триггеров)

Одним из главных процессов, реализованных в ПО “Alertd”, является алгоритм получения текущих состояний триггеров (Рис. 10). Принцип данного алгоритма заключается в том, что демон, с некоторой периодичностью, получает из базы данных список, зафиксированных на последней итерации состояний, чтобы в дальнейшем, после сравнения параметров, выбранных пользователем, проверить поле **state** соответствующего триггера, если оно было равным 0 и значение параметра стало критическим – требуется поменять значение на 1 и отправить уведомление, иначе, если значение было равным 1 и параметр устройства вернулся в норму – следует перевести значение в 0 и отправить уведомление, но уже с другим текстом, обозначающим нормализацию состояния.

На Рис. 11, обозначен принцип определения “Alertd” изменение некоторого параметра устройства. Из схемы можно заметить, что для “Alertd”

нормальным значение считается до того момента, пока не пересечет точку “Critical”. В этом случае ПО меняет значение поля **state** таблицы **notification_trigger_state** на 1 и ставит уведомление в очередь на отправку. Иначе, значение будет считаться как критическое, пока значение не пересечет отметку “Normal”. В таком случае значение поля **state** станет равным 0 и уведомление также встанет в очередь.

Рисунок 10 — Алгоритм получения состояний устройств

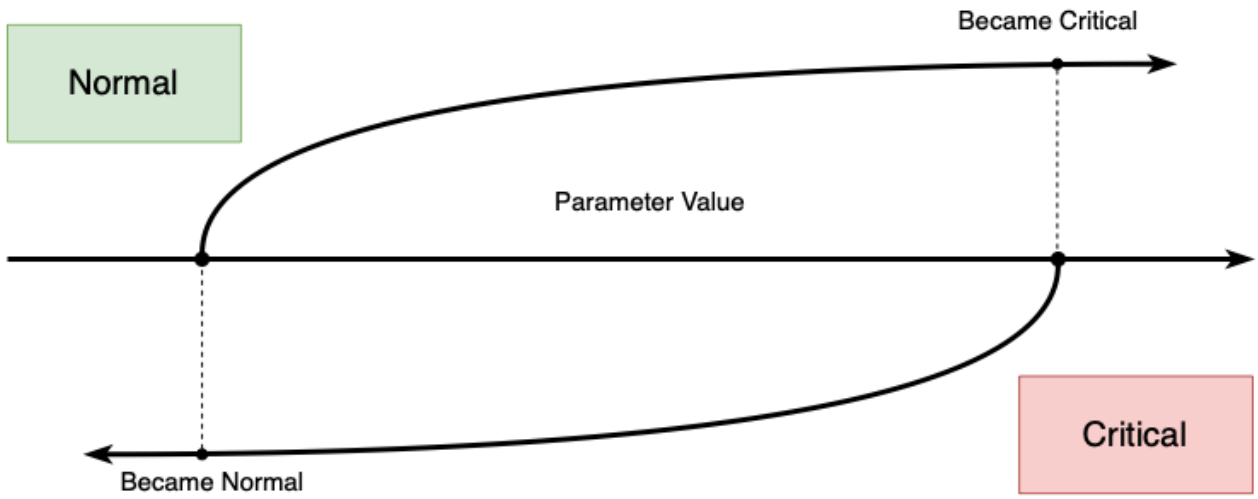

Рисунок 11 — Нормальное и критическое значение в понимании “Alertd”

На Рис. 12 можно заметить, что все состояния имеют значение 1; это обозначает, что некоторый параметр, перечисленных устройств, перешел в критическую зону, уведомление об этом было поставлено в очередь на отправку и успешно отправлено.

	trigger_id id триггера	device_id id девайса	state статус	queue_id id очереди	value значение
Изменить Копировать Удалить	1	2514	1	NULL	NULL
Изменить Копировать Удалить	1	2515	1	NULL	NULL
Изменить Копировать Удалить	1	2516	1	NULL	NULL
Изменить Копировать Удалить	1	2723	1	NULL	NULL
Изменить Копировать Удалить	1	2724	1	NULL	NULL
Изменить Копировать Удалить	1	2725	1	NULL	NULL

Рисунок 12 — Пример данных в таблице notification_trigger_state

2.3.2 Алгоритм добавления новых уведомлений

После того, как новое уведомление было создано пользователем посредством API на сайте мониторинга, оно попадает в базу данных, где его, в свою очередь, должен подхватить “Alertd” для последующей обработки (Рис. 13).

Изначально, вновь созданные уведомления помечаются как untracked при помощи поля **daemon_id** = 0 таблицы **notification**. Это означает, что уведомление не обрабатывает ни одна из запущенных в данный момент копий “Alertd”.

На каждой итерации “Alertd” опрашивает базу данных на предмет записей, у которых поле **daemon_id** равно идентификатору опрашивающей копии. Количество таких записей может быть равно нулю, если, например, демон с таким идентификатором был запущен впервые. После получения “своих” уведомлений “Alertd” запрашивает из БД **max_notifications** - len(**available_notifications**), которые не привязаны ни к одной копии демона (**daemon_id** = 0) и закрепляет их за собой: **daemon_id** = идентификатор копии демона.

На каждой итерации работы “Alertd” информация с предыдущей итерации стирается. Это сделано с той целью, что информация об уведомлении может измениться в любой момент, будь то статус уведомления, его имя, критические и нормальные значения, параметры, за которыми требуется следить и т. д.

Система профилирования “pprof” показала, что подход, при котором мы перезаписываем данные итерацию за итерацией, является менее затратным по потреблению ресурсов, таких как оперативная память и процессорное время, которые уходят на кеширование и постоянную проверку каждого уведомления на какое-либо изменение.

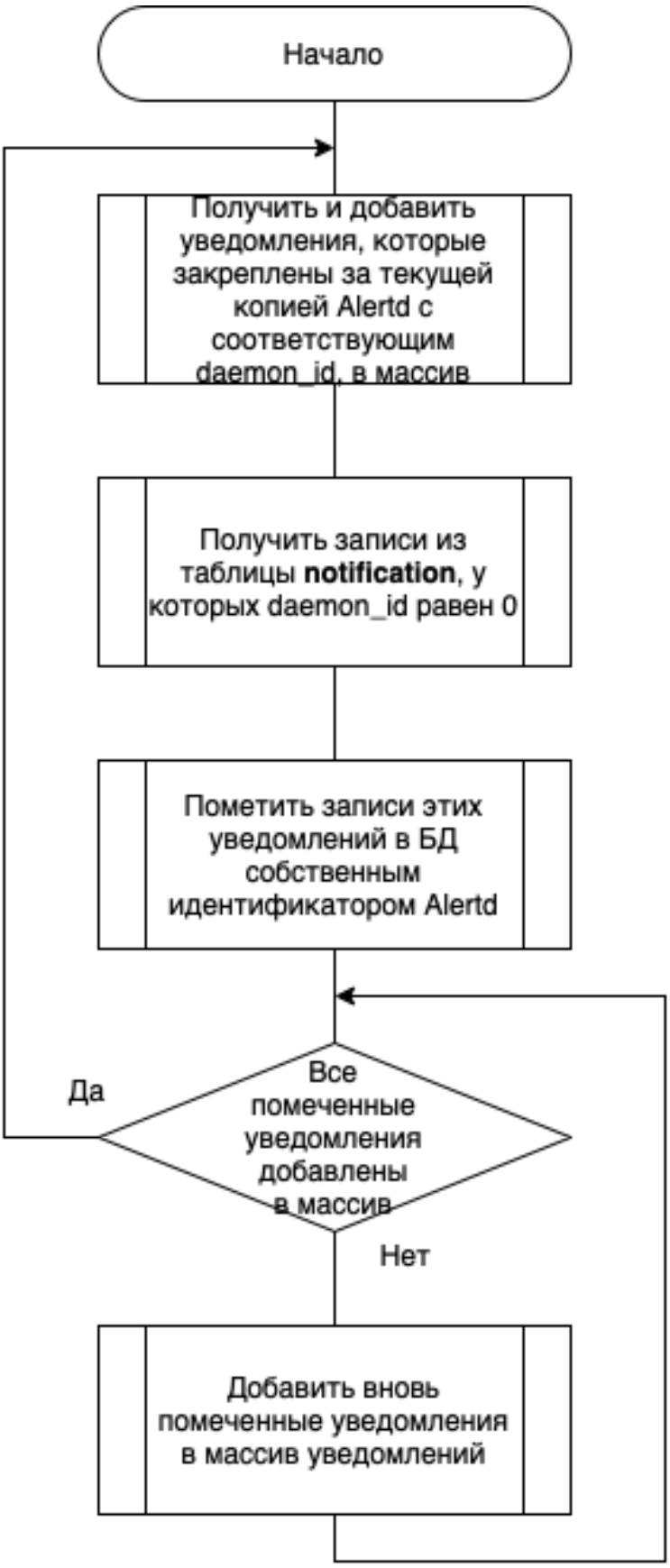

Рисунок 13 – Добавление уведомлений для последующей обработки

2.4 Диаграмма классов

На Рис. 14 видно, что большая часть логики сводится к интерфейсу **Controller**. Это связано с тем, что именно объект контроллера реализует логику проверки, изменилось ли состояние параметра некоторого устройства или нет.

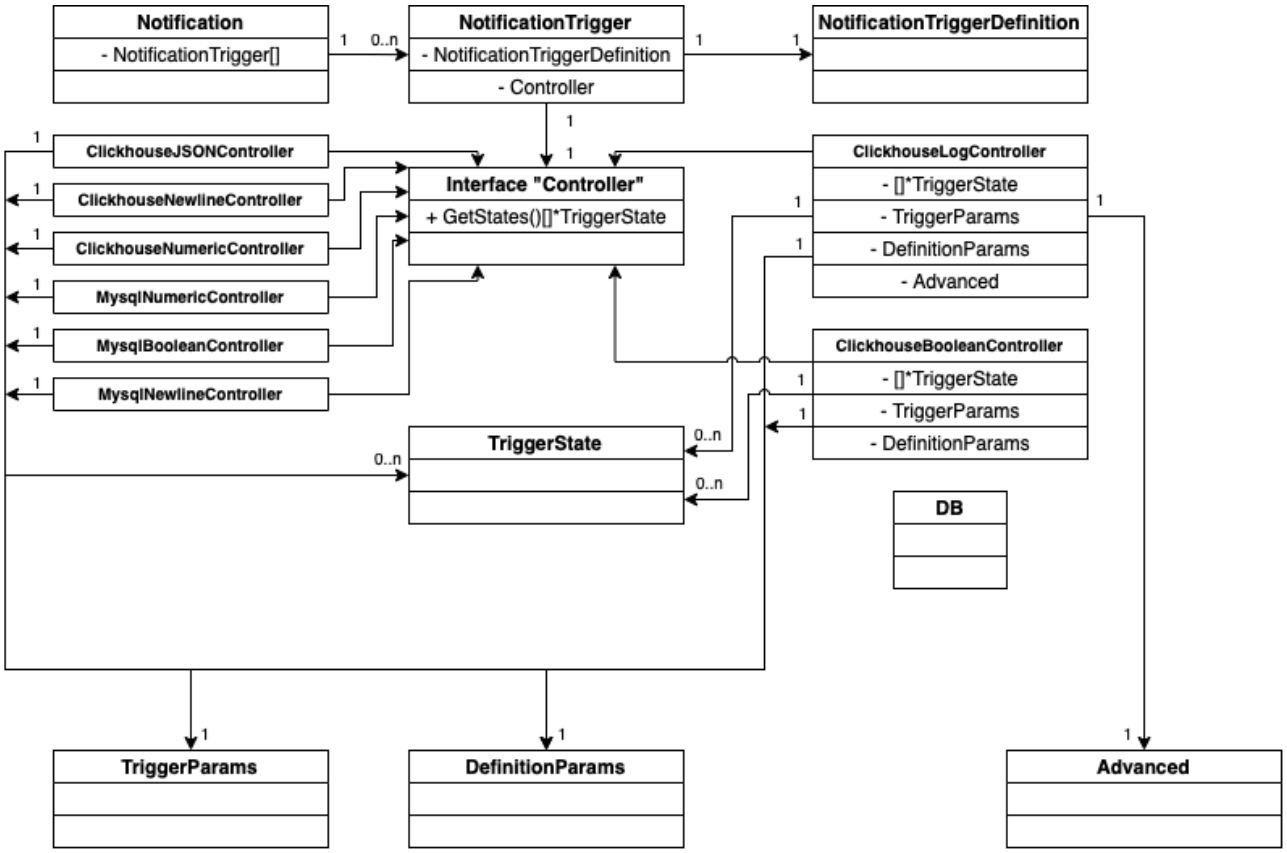

Рисунок 14 – Диаграмма классов “Alertd”

2.5 Выводы по главе

В соответствии с техническим заданием предложена структура взаимодействия классов и структура базы данных; с помощью блок – схем задокументированы наиболее сложные отношения в системе, более детально проработано взаимодействие объектов; разработаны алгоритмы получения состояний триггеров и получения уведомлений.

3 Реализация

3.1 API на сайте мониторинга

Как уже было сказано, пользователь должен иметь возможность конфигурировать уведомления на сайте мониторинга. Для этого мною была реализована Backend часть API.

Для его написания был использован язык PHP 7 и Yii Framework. Yii, в данном случае, выполняет роль маршрутизатора, службы миграции и ORM системы.

Правила маршрутизация URL адресов описывается в файле url_manager_rules.php. Пример маршрутов для конфигурации уведомлений приведены на Рис. 15.

```
19    // универсальные роуты для API
20    'API/<controller:[\wd-]+>/<id:\d+>' => 'API/<controller>/index',
21    'API/<controller:[\wd-]+>/<action:[\wd-]+>/<id:\d+>' => 'API/<controller>/<action>',
22    'API/<controller:[\wd-]+>/<action:[\wd-]+>/<page:\d+>' => 'API/<controller>/<action>',
23    'API/<controller:[\wd-]+>/<action:[\wd-]+>' => 'API/<controller>/<action>',
```

Рисунок 15– Пример универсальных маршрутизаторов

Все конечные точки, которые будут описаны ниже, обрабатываются маршрутом, который описан в 23 строке (Рис. 15).

3.1.1 Конечные точки

Конечные точки предоставляют возможность сохранения, изменения и просмотр данных об уведомлениях:

- **/API/notice/list** – содержит в теле ответа уведомления пользователя, его каналы связи и описания всех доступных параметров для отслеживания на устройствах;

- **/API/notice/status** – позволяет поменять статус уведомления (отдавать его в обработку экземпляром Alertd или нет);
- **/API/notice/delete** – позволяет удалить уведомление из базы данных;
- **/API/notice/channels** – содержит в теле ответа каналы связи пользователя;
- **/API/notice/save** – позволяет сохранить или изменить уведомление пользователя;
- **/API/notice/savechannel** – позволяет сохранить или изменить канал связи пользователя;
- **/API/notice/deletechannel** – позволяет удалить канал связи пользователя;
- **/API/notice/triggers** – возвращает в теле ответа список параметров устройств, которые можно отслеживать.

Функция сохранения(изменения) уведомления, является самой объемной и, чтобы избежать некорректных вставок в базу данных, было принято использовать механизм транзакции. Это значит, что в случае, если какое - то из обращений к базе данных вызвало исключение, то все изменения, прошедшие без ошибок, будут возвращены в первоначальное состояние (Рис. 16).

```
266         } catch (Throwable $e) {
267             Yii::error($e->getTraceAsString());
268             return ['status' => 'Error', 'code' => self::INVALID_PARAMETER];
269             $transaction->rollBack();
270         }
```

Рисунок 16 – Откат изменений в случае возникновения ошибки

3.2 Окружение

Сайт мониторинга разворачивается при помощи утилиты Docker Compose. Он включает в себя множество контейнеров, в числе которых базы данных MariaDB и Clickhouse, принимающая сторона данных с модема – Modem Data Receiver, серверная сторона API Backend, клиентская сторона Frontend и так далее. Эта инфраструктура разворачивается с использованием системы виртуализации ProxMox под управлением гипервизора KVM внутри виртуальной машины.

Демон Alertd контейнеризируется, встраивается в это инфраструктуру и функционирует наравне с остальными контейнерами, благодаря чему достигается минимальная задержка при общении его с базами данных.

В то же время приложение может работать и с удаленными базами данных, если это потребуется.

Данные для подключения к базам данных, а также параметры запуска экземпляра, ПО берет из переменных окружения, которые прописываются в системе контейнера в момент его развертывания.

Пример развернутой инфраструктуры представлен на Рис. 17.

```
root@monitoringsvc:~# docker ps -a
CONTAINER ID   IMAGE                                    COMMAND                  CREATED       STATUS      PORTS
                          NAMES
b81fb1ee93ff   cr.sis.ski/monitoring-services:1.6.8     "docker-php-entrypoi…"   9 days ago    Up 9 days
                          monitoring-services
7ace11d46a31   cr.sis.ski/monitoring-node:1.6.8         "docker-entrypoint.s…"   9 days ago    Up 9 days   0.0.0.0:13674->13674/udp, :::13674->13674/udp
                          monitoring-node
da3765e2cd43   cr.sis.ski/monitoring-php:1.6.8          "docker-php-entrypoi…"   9 days ago    Up 9 days   9000/tcp
                          monitoring-php
63260b9546f8   phpmyadmin/phpmyadmin:4.8.5              "/run.sh supervisord…"   9 days ago    Up 6 days   9000/tcp, 0.0.0.0:8080->80/tcp, :::8080->80/tcp
                          monitoring-pma
4a2d9dc60006   cr.sis.ski/monitoring-nginx:1.6.8        "/docker-entrypoint.…"   9 days ago    Up 9 days   80/tcp
                          monitoring-nginx
e158a03ffef0   cr.sis.ski/monitoring-alertd-sender:1.6.8 "alertd_sender"          9 days ago    Up 9 days
                          monitoring-alertd-sender
28938e8ae62c   cr.sis.ski/monitoring-clickhouse:1.6.8   "/entrypoint.sh"         9 days ago    Up 9 days   0.0.0.0:8123->8123/tcp, :::8123->8123/tcp, 0.0.0.0:9000->9000/tcp,
:::9000->9000/tcp, 9009/tcp   monitoring-clickhouse
d09d6ac5142d   cr.sis.ski/monitoring-db:1.6.8           "docker-entrypoint.s…"   9 days ago    Up 9 days   0.0.0.0:3306->3306/tcp, :::3306->3306/tcp
                          monitoring-db
de00be066d45   cr.sis.ski/monitoring-alertd:1.6.8       "alertd"                 9 days ago    Up 9 days
                          monitoring-alertd
b12ac8935991   redis:6-alpine3.15                       "docker-entrypoint.s…"   2 months ago  Up 9 days   0.0.0.0:6379->6379/tcp, :::6379->6379/tcp
                          monitoring-redis
```

Рисунок 17 – Пример развернутого сайта мониторинга

3.3 Alertd

3.3.1 Инициализация

Первоначальный запуск приложения, как и у большинства других ПО, начинается с инициализации. При реализации Alertd было принято решение брать параметры запуска для настройки и подключения к БД из переменных окружения. Значения и сами переменные попадают в контейнер с приложением из хостовой системы, где они описаны в файле .env, этот функционал предусмотрен в Docker по умолчанию.

3.3.2 Формирование списка уведомлений

Формирование списка уведомлений происходит в главном цикле приложения.

Принцип работы алгоритма выглядит следующим образом:

- на первом этапе из базы данных собираются все уведомления, статус которых равен 1, и имеется привязка к обрабатывающему экземпляру;
- после этого за данным экземпляром закрепляются все уведомления, которые еще не обрабатываются ни одной копией Alertd, до того момента, пока не будет достигнуто максимально возможное количество уведомлений;
- далее, для каждого уведомления из сформированного списка собираются параметры, которые необходимо обрабатывать и устройства, параметры на которых необходимо проверять;
- последним этапом данного алгоритма является запуск алгоритма проверки состояний параметров для каждого из уведомлений.

3.3.3 Проверка состояний

Проверка состояний параметров проводится в 4 этапа итеративно, по каждому из сконфигурированных пользователем параметров:

- для параметра получаем текущие состояния по каждому из устройствю. При первом запуске, мы предполагаем, что состояние параметра для всех устройств является нормальным;
- создаем обрабатывающий контроллер для этого параметра и передаем ему текущие состояния параметра. Именно контроллер будет получать текущее значение параметра для устройств и выносить вердикт, изменилось состояние или нет;
- далее контроллер, исходя из своей логики, проверяет текущие значение параметра устройства и сверяет его с критическими и нормальными значениями, если состояние не сходится со значением, которое получил контроллер, то есть, состояние изменилось – контроллер помечает его измененным, тем самым формирует новый список состояний для всех переданных ему устройств;
- в последнюю очередь в работу вступает функция, которая сверяет текущие состояния и состояния, которые были на предыдущей итерации. Для тех устройств, состояние параметра у которых изменилось, формируется сообщение и ставится в очередь на отправку.

3.4 Alertd Sender

На данном этапе уведомление окончательно сформировано и ожидает отправки. Этим занимается приложение Alertd Sender.

Это приложение работает параллельно с Alertd. Оно выбирает из базы данных все уведомления, которые помечены как не отправленные, проверяет временную метку и, если текущее время меньше полученной временной метки, то отправляет уведомление пользователю в сконфигурированный канал связи (Рис. 18).

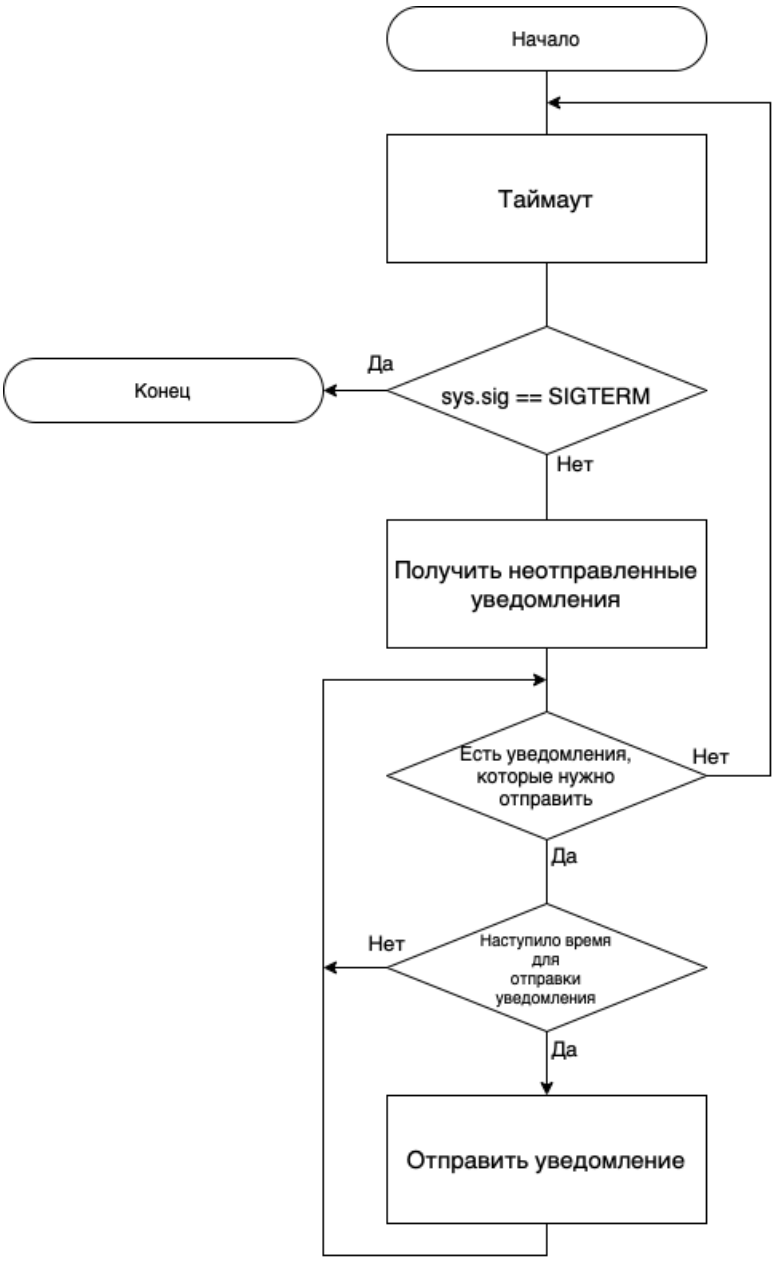

Рисунок 18 – Алгоритм Alertd Sender

3.5 Завершение приложений

Как Alertd, так и Alertd Sender параллельно обрабатывают системные сигналы. Если в приложение пришел системный сигнал SIGTERM, то Alertd снимет со всех уведомлений свой идентификатор экземпляра и завершит работу (Рис. 19), а Alertd Sender без каких - либо действий также завершается.

В данном случае обрабатывается только системный сигнал SIGTERM, так как, в отличие от, например, SIGKILL, SIGTERM можно отловить и обработать или проигнорировать.

```
39      signal.Notify(sigs, syscall.SIGINT, syscall.SIGTERM)
40      go func() {
41          for {
42              sig := <-sigs
43              if sig == syscall.SIGTERM {
44                  ClearDaemonID(db, daemon_id)
45                  os.Exit(0)
46              }
47          }
48      }()
```

Рисунок 19 – Обработка системных сигналов

ЗАКЛЮЧЕНИЕ

В процессе написания выпускной квалификационной работы:

- была разработана архитектура БД, которая позволила бы расширять функционал по мере надобности без непосредственного изменения кода;
- разработано API, которое, вкупе с пользовательским интерфейсом, дает возможность легко конфигурировать объекты уведомлений;
- были описаны структуры, соответствующие моделям в БД;
- был разработан базовый алгоритм, который собирает достаточные данные моделей для проверки состояний выбранных пользователем устройств или групп устройств и добавления записи в очередь на отправку;
- была разработана логика, которая позволяет с легкостью добавлять больше возможных параметров для отслеживания без времени простоя;
- в алгоритм заложена возможность отложенной отправки сообщения благодаря тому, что при добавлении записи в очередь на отправку вместе с данными о получателе и другой необходимой информацией, в базе данных обозначается временная метка, которая описывает время, когда требуется отправить сообщение. Благодаря этому механизму шанс ложной отправки уведомления сводится к минимуму;
- благодаря полю daemon_id в модели уведомления и соответствующей логике, достигается возможность горизонтального масштабирования, то есть распределение обработки уведомлений на несколько экземпляров Alertd;
- был разработан отправитель уведомлений. Он выбирает все записи из очереди на отправку, уведомление по которым еще не было отправлено, получает тип канала связи (telegram, email, … реализован только telegram) и сверяет время, когда требуется отправить уведомление. Если текущее время становится меньше, чем то, что было обозначено в БД, уведомление отправляется и помечается, как отправленное в случае, если не возникло ошибок при отправке;

- приложение было собрано в Docker образ, в основе которого лежит ОС Alpine. Данные о настройке экземпляра, именах хостов БД и пароли записываются внутрь контейнера в виде переменных окружения.

В итоге решены все поставленные задачи, получен Акт о внедрении в производственный процесс компании Систематикс (ПРИЛОЖЕНИЕ А). В настоящий момент, приложение используется компанией в полной мере.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Ресурс мониторинга за Мульти Модемами [Электронный ресурс]: – Режим доступа: https://my.sistematics.ru/#/modem-list
2. Система уведомления Zabbix [Электронный ресурс]: – Режим доступа: https://www.zabbix.com/ru
3. SNMP [Электронный ресурс]: – Режим доступа: https://selectel.ru/blog/snmp/
4. Система контейнерной виртуализации Docker [Электронный ресурс]: – Режим доступа: https://www.docker.com
5. Pprof [Электронный ресурс]: – Режим доступа: https://github.com/google/pprof
6. Сайт компании «Систематикс» [Электронный ресурс]: – Режим доступа: http://sistematics.ru
7. Язык программирования Golang [Электронный ресурс]: – Режим доступа: https://go.dev
8. Язык программирования PHP [Электронный ресурс]: – Режим доступа: https://www.php.net
9. База данных Clickhouse [Электронный ресурс]: – Режим доступа: https://clickhouse.com/docs/en/intro
10. База данных MariaDB [Электронный ресурс]: – Режим доступа: https://mariadb.org/documentation/
11. Обработка системных сигналов с использованием языка Golang [Электронный ресурс]: – Режим доступа: https://stackoverflow.com/questions/18106749/golang-catch-signals

ПРИЛОЖЕНИЕ А

Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт
Кафедра вычислительной техники
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой
_____ О.В.Непомнящий
подпись инициалы, фамилия
«20» 06 2022г.

БАКАЛАВРСКАЯ РАБОТА

09.03.01 – Информатика и вычислительная техника
код – наименование направления

Система уведомления для модемов компании “Систематикс”
тема

Руководитель _____ 20.06.22 доц. каф. _____ Л. И. Покидышева
подпись, дата должность, ученая степень инициалы, фамилия

Студент _____ Д. А. Веселков
подпись, дата инициалы, фамилия

Нормоконтролер _____ 20.06.22 доц. каф. _____ Л. И. Покидышева
подпись, дата должность, ученая степень инициалы, фамилия

Красноярск 2022