«Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И.Ульянова (Ленина)»
(СПбГЭТУ «ЛЭТИ»)

Направление	01.03.02 – Прикладная математика и информатика
Профиль	Без профиля
Факультет	КТИ
Кафедра	МО ЭВМ

К защите допустить

Зав. кафедрой К.В. Кринкин

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Тема: Разработка распределенной отказоустойчивой системы мониторинга доступности веб-сайтов и сетевых сервисов

Студент		_____ *подпись*	М.В. Гаськов
Руководитель	к.т.н.	_____ *подпись*	А.А. Лавров
Консультанты	к.т.н., доцент	_____ *подпись*	В.И. Фомин
	к.т.н., доцент	_____ *подпись*	А.А. Лисс

Санкт-Петербург

2019

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

Утверждаю
Зав. кафедрой МО ЭВМ
_____ К.В. Кринкин
«___»_____2019 г.

Студент Гаськов М.В. Группа 5381

Тема работы: Разработка распределенной отказоустойчивой системы мониторинга доступности веб-сайтов и сетевых сервисов.
Место выполнения ВКР: СПбГЭТУ «ЛЭТИ»

Исходные данные (технические требования):
Реализовать распределенную отказоустойчивую систему мониторинга доступности сетевых сервисов, предназначенную для децентрализованного мониторинга состояния, доступности и работоспособности удаленных сетевых узлов, в том числе размещенных в глобальной сети Интернет, сетевых сервисов (DNS, FTP, SSH и др.) и веб-сайтов.
Содержание ВКР:
Обзор предметной области, разработка архитектуры системы, программная реализация, проектирование пользовательского интерфейса, развертывание и тестирование системы в реальных условиях.

Перечень отчетных материалов: пояснительная записка, иллюстративный материал, код программной реализации.

Дополнительные разделы: разработка и стандартизация программных средств.

Дата выдачи задания «___»_____2019 г.

Дата представления ВКР к защите «___»_____2019 г.

Студент _____ М.В. Гаськов

Руководитель к.т.н. _____ А.А. Лавров

КАЛЕНДАРНЫЙ ПЛАН ВЫПОЛНЕНИЯ
ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

Утверждаю
Зав. кафедрой МО ЭВМ
_____ К.В. Кринкин
«___»_____2019 г.

Студент Гаськов М.В. Группа 5381

Тема работы: Разработка распределенной отказоустойчивой системы мониторинга доступности веб-сайтов и сетевых сервисов.

№ п/п	Наименование работ	Срок выполнения
1	Обзор литературы по теме работы	27.04 – 30.04
2	Разработка архитектуры системы мониторинга	01.05 – 08.05
3	Разработка программной реализации	09.05 – 24.05
4	Тестирование и отладка системы	25.05 – 27.05
5	Оформление пояснительной записки	27.05 – 31.05
6	Оформление иллюстративного материала и доклада	02.06
7	Предварительная защита	10.06

Студент _____ М.В. Гаськов

Руководитель к. т. н. _____ А.А. Лавров

РЕФЕРАТ

Пояснительная записка 113 стр., 14 рис., 2 табл., 16 ист., 6 прил.

Ключевые слова: СЕТЕВОЙ МОНИТОРИНГ, РАСПРЕДЕЛЕННЫЕ СИСТЕМЫ, ОТКАЗОУСТОЙЧИВЫЕ СИСТЕМЫ, JAVA.

Целью работы является реализация распределенной отказоустойчивой системы мониторинга доступности сетевых сервисов.

В рамках выполнения работы на основе поставленных требований была разработана архитектура распределенной системы мониторинга и выбраны инструменты для ее реализации. Для обеспечения отказоустойчивости системы использованы технологии кластеризации серверов и репликации данных. Система мониторинга реализована и протестирована в условиях моделирования реальной глобально распределенной инфраструктуры на базе виртуальных серверов, размещенных в глобальной сети Интернет.

ABSTRACT

Aim of this work is to implement a distributed fault-tolerant monitoring system. Objects of monitoring are network services that work on different net protocols.

Theoretically, the approaches introduced in this work could be used in enterprise companies that need to keep their vast infrastructure highly available.

As a result of the graduation work, the architecture of the distributed monitoring system was developed and tools for its implementation were selected based on the requirements. To provide system fault-tolerance server clustering and data replication technologies were used. The system was implemented and tested in an infrastructure that contained several virtual servers.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В настоящей пояснительной записке использованы следующие термины с соответствующими определениями:

ОС – операционная система;

БД – база данных;

Целевой узел – узел, сервисы которого систематически опрашиваются системой мониторинга с целью получения необходимой информации.

Узел мониторинга – узел, входящий в кластер системы, занимающийся непосредственным опросом сервисов на целевых узлах.

ВВЕДЕНИЕ

На сегодняшний день практически любая коммерческая или государственная организация имеет в своем распоряжении множество различных компьютерных узлов, которые предоставляют те или иные сервисы внутренним и/или внешним пользователям. Естественно, что любая компания заинтересована в бесперебойной работе своих машин, поскольку отказ одной или нескольких из них вероятнее всего приведет к финансовым и репутационным потерям. В некоторых случаях перебои в работе системы ведут даже к более серьезным последствиям, таким как, например, угроза жизни и здоровью людей в случае отказа систем жизнеобеспечения или коммунального обслуживания. Для своевременного устранения неполадок в работе систем, связанных с отказом их узлов, необходимо производить регулярную проверку работоспособности имеющихся машин и сервисов, предоставляемых ими.

Цель работы: Реализация отказоустойчивой системы сетевого мониторинга удаленных узлов и предоставляемых ими сервисов.

В ходе выполнения работы необходимо решить следующие **задачи:**

1. Разработать архитектуру системы мониторинга;
2. Выбрать технологии реализации;
3. Реализовать механизмы обеспечения отказоустойчивости системы;
4. Реализовать модули системы и провести отладку и тестирование разработанного программного обеспечения (ПО).

Объектами исследования являются: организация мониторинга информационных систем и задачи сетевого мониторинга, разработка отказоустойчивых распределенных информационных систем.

Предметом исследования является отказоустойчивая система мониторинга, имеющая распределенную архитектуру.

Практическая значимость проекта состоит в потенциальной возможности использования создаваемой системы мониторинга в организациях, имеющих потребность в постоянном контроле удаленных узлов и их сервисов.

1. ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Понятие распределенных систем

Распределенная система представляет собой совокупность компьютерных узлов, объединенных друг с другом по сети с целью выполнения определенных задач. Зачастую распределенная природа тех или иных систем не видна пользователю.

К распределенным системам предъявляются такие требования как:

- **масштабируемость**: при увеличении нагрузки на систему должна быть предусмотрена возможность добавления в систему новых элементов для увеличения эффективности и/или производительности ее работы. При этом процесс масштабирования не должен нарушать текущей работы распределенной системы и, в идеале, должен проходить незаметно для пользователей. В настоящее время практически любому набирающему популярность веб-приложению или веб-сервису необходима возможность масштабирования [1];
- **отказоустойчивость**: отказ в работе одного или нескольких компонентов не должен привести к сбоям в работе всей распределенной системы в целом. Это требование можно удовлетворить с помощью использования технологий кластеризации и репликации, рассматриваемых далее;
- **прозрачность**: распределенная структура системы должна быть скрыта для пользователя;
- **открытость**: составные части системы и их сообщение друг с другом должны соответствовать общим стандартам, принятым в разработке таких систем;

и другие требования.

Кластеризация – объединение некоторого множества компьютерных узлов для выполнения определенной задачи. Как правило, узлы кластера схожи по своим функциям и взаимозаменяемы.

Существуют различные стратегии построения кластеров:

- Кластеры с избыточностью (узлы кластера выполняют параллельно одинаковые действия, такая стратегия используется, когда простой кластера не допускается);
- пассивные кластеры (некоторые узлы простаивают и включаются в работу только при отказе узла, который они дублируют);
- активные кластеры (каждый узел выполняет некоторую порцию работы всего кластера; когда один из узлов выходит из строя, его часть работы распределяется между оставшимися).

Репликация – поддержание данных в одинаковом состоянии на нескольких узлах системы.

Выделяются синхронная и асинхронная репликации. В первом случае изменения должны распространяться по всем узлам с данными одновременно, во втором – процесс распространения данных может занимать некоторое время.

В контексте темы распределенных систем стоит упомянуть об эвристической CAP теореме (от слов Consistency, Availability, Partition tolerance)[2], которая говорит о возможности одновременного обеспечения не более двух из трех нижеперечисленных свойств распределенной системы:

- Консистентность данных (consistency) – в каждый момент времени данные на различных узлах системы не противоречат друг другу;
- Доступность (availability) – система всегда отвечает корректно;
- Толерантность к разделению (partition tolerance) – разделение системы на отдельные части не ведет к сбоям и некорректной работе всей системы.

В качестве примера можно привести, например, тот факт, что при разработке распределенных систем часто встает выбор между двумя решениями хранения данных, первое из которых предполагает исполь-зование NoSQL баз данных, которые являются устойчивыми к разделению, или некоторые

подходы к организации кластеров реляционных БД, однако это приводит к ухудшению согласованности данных между узлами кластеров баз данных, доступность данных при этом в целом не ухудшается. Альтернативным решением является использование реляционных БД, которые демонстрируют хорошую согласованность данных в небольших кластерах, но базы данных этого типа плохо устойчивы к разделению, доступность данных при этом остается на хорошем уровне. Выбор между вышеперечисленными решениями определяется требованиями к реализуемой системе.

1.2 Мониторинг информационных систем, организация сетевого мониторинга

Под сетевым мониторингом понимается наблюдение за узлами целевой сети. Целями сетевого мониторинга являются выявление неисправных или медленных (в смысле задержки ответов на запросы) узлов, а также своевременное оповещение пользователя об обнаруженных неполадках.

Одна из возможных классификаций [3] методов мониторинга вычислительных сетей (ВС) представлена на рис. 1.1:

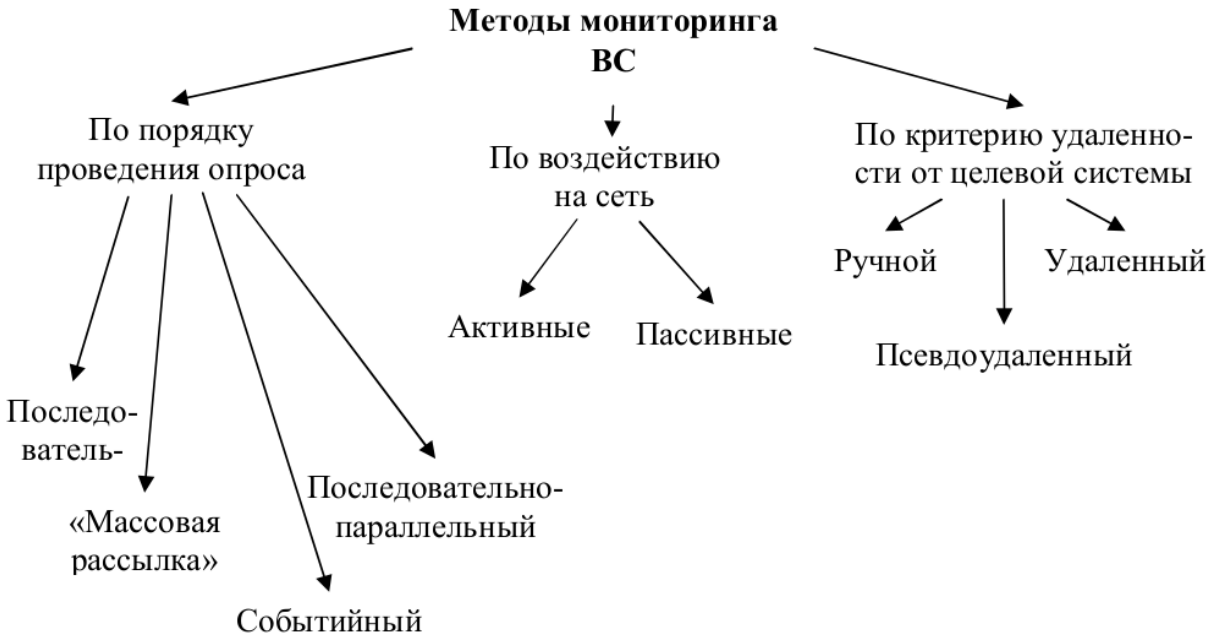

Рисунок 1.1 – Классификация методов мониторинга вычислительных сетей

Таким образом, методы мониторинга можно разделить по следующим критериям:

- Порядку проведения опроса:
  - Последовательный – при таком методе опрос выполняется последовательно через определенные промежутки времени в одном потоке;
  - Последовательно-параллельный – параллельный опрос групп узлов, при котором узлы в каждой группе опрашиваются последовательно;
  - Метод “Массовой рассылки” – запросы отправляются всем отслеживаемым узлам, формируется очередь ответов;
  - Событийный – на отслеживаемых узлах установлены специальные программы (“агенты”), которые отправляют данные о состоянии узлов на сервера мониторинга.
- Воздействию на сеть:
  - Активные – опрос узлов системой мониторинга;
  - Пассивные – прослушивание трафика и его анализ.
- По удаленности от целевой системы:
  - Удаленный – с помощью взаимодействия по сети;
  - Псевдоудаленный – с использованием временных устройств для накопления информации;
  - Ручной – при помощи визуального осмотра отслеживаемых узлов.

При мониторинге распределенных систем, с учетом их специфики, к системам мониторинга предъявляется требование масштабируемости [4].

2 РАЗРАБОТКА АРХИТЕКТУРЫ ОТКАЗОУСТОЙЧИВОЙ СИСТЕМЫ СЕТЕВОГО МОНИТОРИНГА

2.1 Формализация требований к системе

В рамках ВКР требуется реализовать распределенную систему мониторинга, удовлетворяющую требованиям, предъявляемым к распределенным системам в целом, а также:

- Распределенная система должна быть отказоустойчивой и масштабируемой (с точки зрения возможности добавления новых узлов в кластер узлов мониторинга). Под узлом мониторинга понимается узел, осуществляющий мониторинг целевых узлов, т.е. узлов, статус которых необходимо узнавать;
- Мониторинг узлов входящих в кластер мониторинга для контроля состояния самой системы;
- Реализация активного мониторинга;
- Мониторинг широко распространенных сетевых сервисов (HTTP, SMTP, POP3, SSH, FTP);
- В системе должна быть предусмотрена возможность задания пользователем частоты опроса целевых узлов от одной минуты;
- Возможность мониторинга потенциально неограниченного числа целевых узлов;
- Должен быть реализован эффективный механизм хранения исторических данных. Под эффективностью понимается отсутствие не важной и избыточной информации, т.е. эффективное расходование ресурсов при хранении данных с сохранением их информативности.

2.1 Архитектура системы сетевого мониторинга

Реализуемая система должна обеспечивать высокий уровень надежности, чего можно достичь с помощью распределенной архитектуры, обладаю-

щей такими свойствами, как отказоустойчивость, прозрачность, масштабируемость[5].

При разработке системы были использованы принципы т.н. трехслойной архитектуры (*3-tier architecture*), которая схематично изображена на рис. 2.1 и является наиболее распространенным частным случаем N-слойной архитектуры. Под слоем подразумевается некоторая часть программной реализации, нацеленная на выполнение функций определенного типа. Слой может быть представлен в виде отдельного программного модуля.

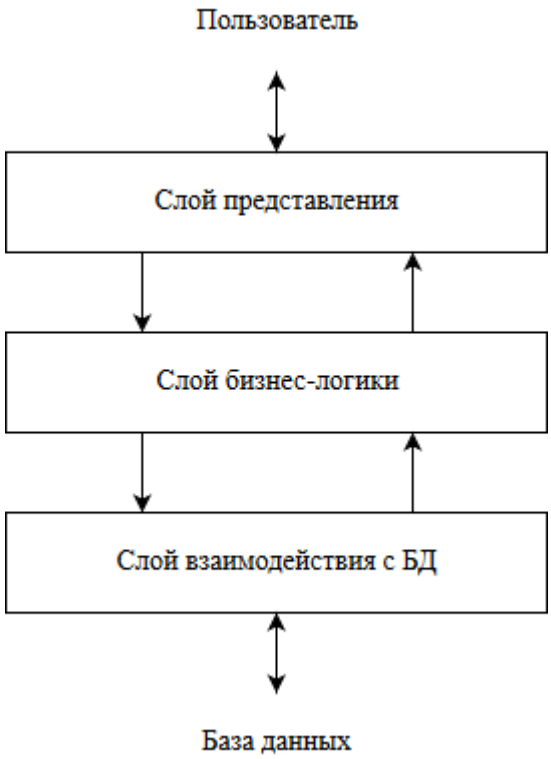

Рисунок 2.1 – трехслойная архитектура

На рисунке изображены три слоя распределенного приложения:

- **Слой представления (Presentation layer)** – данный слой чаще всего представляет собой графический интерфейс в виде web-клиента или

desktop-приложения и служит для непосредственного взаимодействия с пользователем;

- **Слой бизнес-логики (Business Layer)** – данный слой является «краеугольным камнем» всей системы, включает в себя реализацию основной логики системы, взаимодействуя как со слоем представления, так и со слоем доступа к базе данных;
- **Слой взаимодействия с базой данных (Data access layer)** – слой, отвечающий за работу с базой данных, который предоставляет все необходимые интерфейсы и позволяет вынести взаимодействие с базой данных на более высокий (по сравнению с простыми SQL-запросами) уровень, что позволяет абстрагироваться от конкретного поставщика технологий хранения данных.

В рамках данной архитектуры слой представления не должен иметь связи с базой данных напрямую, все операции должны проходить через слой бизнес логики. Отделение слоя представления устраняет связывание бизнес-логики системы с каким-либо конкретным клиентом, например, приложение может иметь одновременно несколько различных способов взаимодействия с пользователем, таких как через Android-приложение, web-сайт и т.д. Также преимуществом трехслойной архитектуры является масштабируемость: при повышении нагрузки на конкретный слой можно селективно масштабировать только его, не затрагивая при этом остальные слои.

На основе принципов трехслойной архитектуры разработана схема архитектуры создаваемой системы мониторинга [6] (рис. 2.2), удовлетворяющая поставленным требованиям.

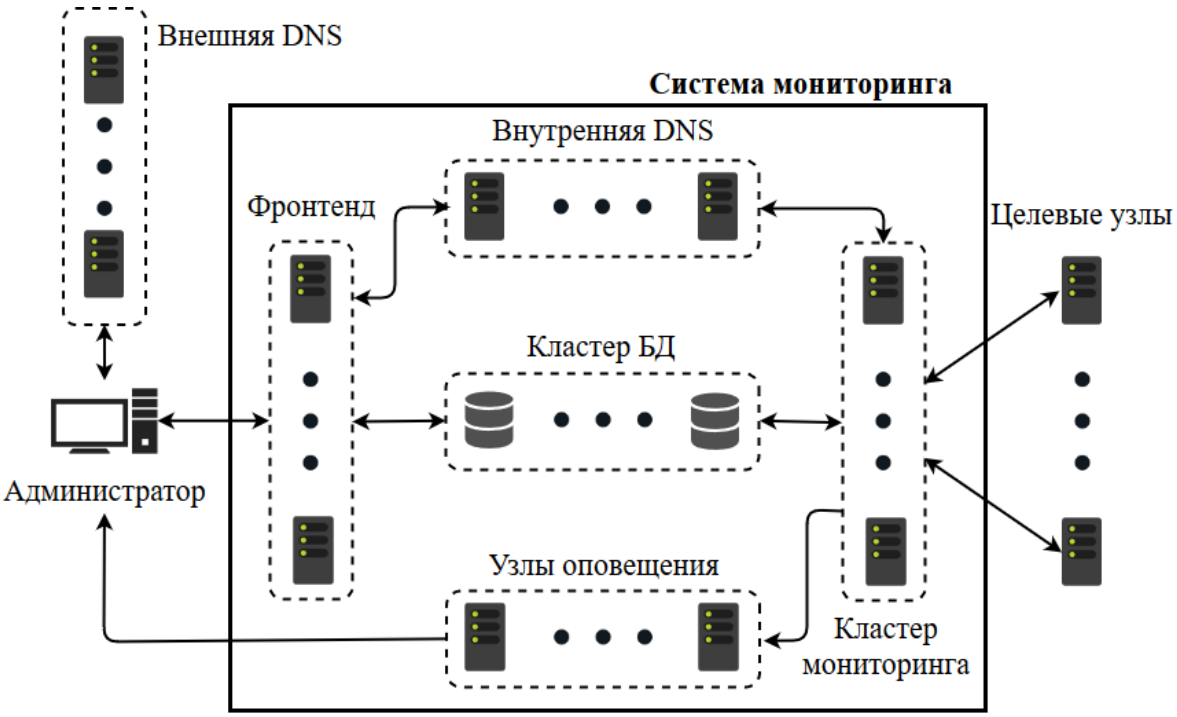

Рисунок 2.2 – архитектура распределенной отказоустойчивой системы мониторинга

Система мониторинга состоит из следующих компонентов:

1. Интерфейс пользователя (Фронтенд) – предоставляет интерфейс, посредством которого пользователь может менять конфигурацию системы и получать необходимую информацию об узлах системы. Интерфейс пользователя является слоем представления;
2. Кластер мониторинга – множество узлов, выполняющих непосредственный мониторинг целевых узлов. Кластер получает необходимую информацию из кластера базы данных, взаимодействуя с ним через слой взаимодействия с базой данных;
3. Кластер базы данных – содержит данные о конфигурации, целевых узлах и пр.;
4. Серверы оповещения – серверы, отвечающие за оповещение администратора о произошедших событиях путем рассылки СМС-сообщений, сообщений электронной почты и т.д.;

5. Внутренняя система DNS – отображает IP-адреса узлов, входящих в систему, на пространство символьных имен.

С учетом ограничений в ресурсах вышеописанная архитектура в полной мере не может быть реализована в рамках ВКР, однако, возможно реализовать упрощенный вариант архитектуры (рис. 2.3), который при наличии необходимых ресурсов может быть легко масштабирован до системы с архитектурой, изображенной на рис. 2.2.

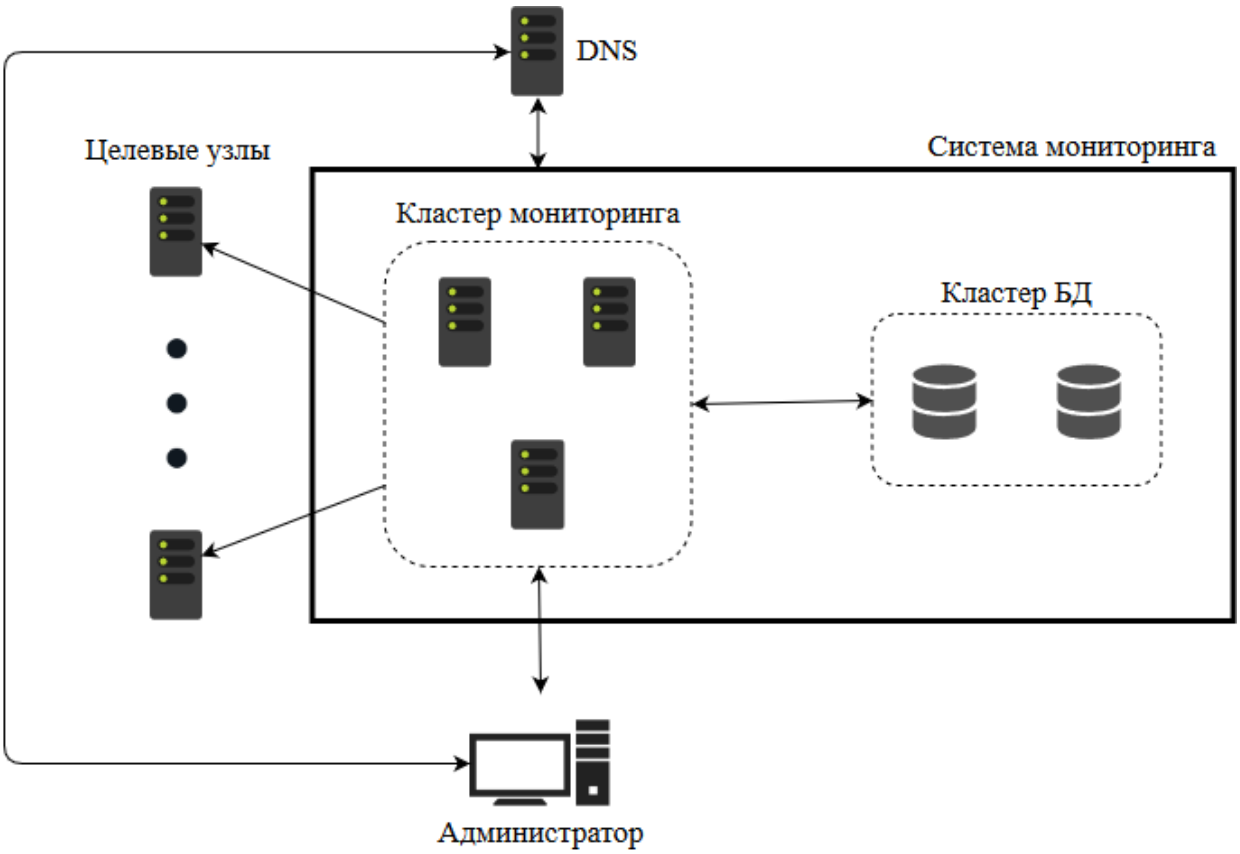

Рисунок 2.3 – Упрощенная архитектура распределенной отказоустойчивой системы мониторинга

В распределенной системе с упрощенной архитектурой слои представления и взаимодействия с базой данных перенесены в кластер мониторинга. Структура узла кластера мониторинга представлена на рис. 2.4.

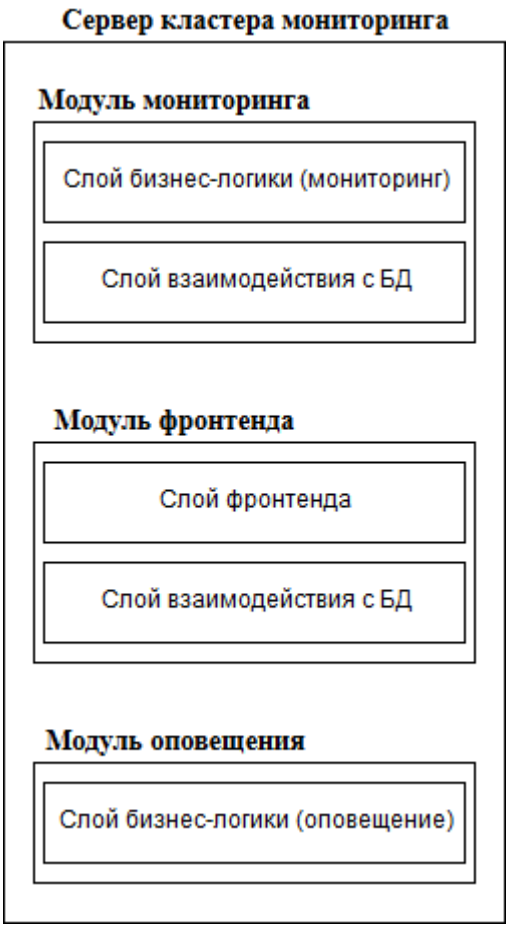

Рисунок 2.4. – Структура сервера кластера мониторинга

За счет того, что модули мониторинга, интерфейса пользователя и оповещения разделены программно (см. рисунок 2.4) и могут запускаться на серверах независимо друг от друга, можно образовывать сервера, предназначенные исключительно для мониторинга целевых узлов, предоставляющие только функции интерфейса пользователя или же занимающиеся только оповещением администратора. Следует также отметить, что в упрощенной архитектуре внутренняя и внешняя системы DNS объединены в одну, занимающуюся разрешением доменных имен как внутренних, так и внешних серверов. При необходимости возможно её расширение до двух DNS-кластеров, как изображено в схеме на рис. 2.2, что потребует незначительных изменений сетевых параметров на узлах системы. Таким образом, распределенная система, реализованная по упрощенной архитектуре, предусматривает воз-

можности масштабирования до системы с архитектурой, изображенной на рис.2.2.

2.2 Методы обеспечения отказоустойчивости

Одним из основных требований к распределенной системе мониторинга является отказоустойчивость. Выполнение этого требования достигнуто с помощью объединения серверов в кластеры, а также технологии репликации баз данных.

2.2.1 Кластер мониторинга

В реализованной системе три сервера мониторинга образуют кластер мониторинга. Отказоустойчивость интерфейса пользователя гарантируется тем, что каждый сервер предоставляет пользователю веб-интерфейс, при взаимодействии с системой пользователь переходит на сайт с установленным доменным именем и подключается к интерфейсу пользователя одного из серверов мониторинга, то, какой именно сервер будет отвечать на запросы пользователя, определяется алгоритмом round robin DNS. Суть алгоритма round robin DNS состоит в том, что некоторому доменному имени ставится в соответствие некоторое множество IP-адресов, затем, при обращении на данное доменное имя, запрос будет направлен на первый IP-адрес из списка, при последующем обращении – на второй и т.д. по замкнутому кругу[7]. Таким образом, при выходе из строя одного или двух из трех серверов пользователь все также сможет взаимодействовать с системой через браузер.

Отказоустойчивость системы оповещения достигается тем, что каждый сервер имеет модуль оповещения, который отправляет информацию о событиях на сервисах целевых узлов, опрашиваемых этим конкретным сервером, на электронную почту администратору.

При выходе из строя или удалении узла из кластера мониторинга целевые узлы, опрос которых производил данный узел, перераспределяются меж-

ду остальными узлами кластера поровну, при добавлении нового узла в кластер мониторинга или при возвращении к работе ранее вышедшего из строя, все целевые узлы перераспределяются аналогичным образом. Перераспределение целевых узлов, а также алгоритм внутреннего мониторинга, о котором будет сказано ниже, реализованы на стороне базы данных с использованием триггерных функций.

Внутри кластера реализован алгоритм внутреннего мониторинга: за каждым узлом в кластере закрепляется другой узел из кластера с целью регулярного опроса, каждый узел из кластера может опрашиваться только одним узлом – так формируется «замкнутый круг» мониторинга кластером самого себя, узлы могут быть добавлены или удалены из «круга». Предположим, узел N осуществляет мониторинг узла N+1, а узел N+1 опрашивает узел N+2, тогда при выходе из строя или удалении узла N+1 мониторинг закрепленного за ним узла N+2 возьмет на себя узел N, т.е. тот, который осуществлял мониторинг удаленного или вышедшего из строя узла N+1. Алгоритм проиллюстрирован на рис. 2.5:

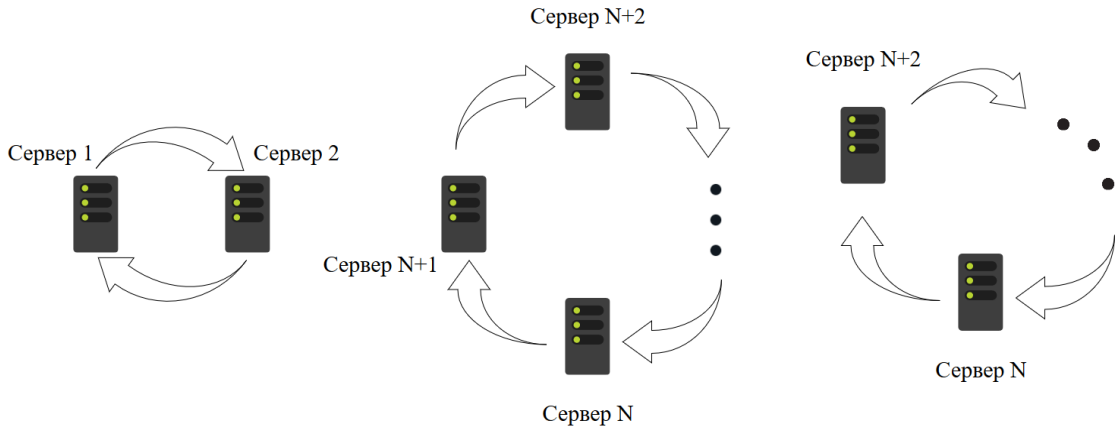

Рисунок 2.5 – Алгоритм внутреннего мониторинг кластера узлов мониторинга

2.2.2 Кластер БД

При выборе технологии и стратегии хранения данных были учтены ограничения, накладываемые CAP-теоремой, в соответствии с которой при

хранении данных в распределенных системах необходимо выбрать два из трех свойств: согласованность, доступность и устойчивость к разделению.

В системе реализован кластер из двух серверов по стратегии multi-master, когда каждый сервер имеет роль master, т.е. предоставляет пользователю возможность чтения, записи, удаления и обновления данных. Данные асинхронно реплицируются между серверами кластера. Такое решение было выбрано с учетом специфики системы мониторинга: асинхронная репликация предоставляет «согласованность в конечном счете» (eventually consistent), т.е. данные на серверах кластера придут в одинаковое состояние, но с некоторой задержкой, что допустимо в конкретной системе, так как между опросами целевых узлов проходит существенный промежуток времени от минуты до часа. Стратегия multi-master позволяет хранить данные на географически удаленных друг от друга серверах кластера, что гарантирует отказоустойчивость в случае стихийных бедствий и других непредвиденных критических ситуаций, которые могут привести к выходу из строя узла кластера, находящегося в конкретном регионе. Таким образом, опираясь на CAP-теорему, при выборе реализации кластера БД был сделан выбор в пользу высокой доступности и устойчивости к разделению, жертвуя при этом строгой согласованностью данных.

По аналогии с кластером мониторинга узлы кластера БД имеют общее доменное имя, и запросы к серверам кластера разрешаются по алгоритму Round Robin.

3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ СИСТЕМЫ СЕТЕВОГО МОНИТОРИНГА

3.1 Используемые технологии и программные средства

Программные модули узлов кластера мониторинга реализованы на языке java из соображений платформонезависимости и простоты поддержки кода. Система разрабатывалась и тестировалась на машинах с ОС Linux Ubuntu, однако все программные модули, выполняющие функции мониторинга, оповещения и пользовательского интерфейса, представляют собой jar архивы, поэтому могут быть запущены на любой ОС с установленной средой выполнения JRE (Java Runtime Environment) версии 8 и выше, JRE включает в себя виртуальную машину JVM (Java Virtual Machine), которая имеет реализацию практически для все распространенных ОС и позволяет запускать один и тот же java код на различных ОС[8]. Для установки СУБД потребуется система Linux. Сопутствующие технологии, применявшиеся при разработке, перечислены ниже:

- **Maven** – инструмент для сборки проектов, который отвечает за управление зависимостями, создание jar архива и дистрибутива программы, а также генерацию документации т.д. На сегодняшний день Maven является наиболее распространенной системой сборки и управления проектами на java;
- **Git** – система контроля версий. В данной системе контроля версий используется принцип центрального репозитория, что позволяет сохранять изменения локально при отсутствии интернет-соединения, а потом отправлять изменения на сервер. Также Git была выбрана из-за наличия опыта работы с ней у автора ВКР;
- **Spring** – семейство фреймворков для разработки программ на java. В основе Spring Framework лежит концепция DI (Dependency Injection - внедрение зависимостей). Суть DI состоит в ослаблении зави-

симостей объектов внутри кода за счет взаимодействия с зависимостями через программные интерфейсы [9]. Данное семейство фреймворков значительно ускоряет разработку и является наиболее удобным решением в сфере enterprise-разработки. Были использованы следующие Spring фреймворки:

  - **Spring boot** – упрощение запуска и первоначальной настройки проекта;
  - **Spring data** – взаимодействие со слоем базы данных (database layer);
  - **Spring security** – авторизация пользователя;
  - **Spring session** – хранение и работа с пользовательскими сессиями.

- **Lombok project** – инструменты для автоматической генерации стандартных методов класса (“сеттеры”, “геттеры”, “конструкторы” и пр.) с помощью java-аннотаций. Lombook делает программный код «чище» и на сегодняшний не имеет надежных альтернатив;
- **JQuery** – библиотека javascript для упрощения создания интерактивности веб-страниц, а также взаимодействия с серверной частью (backend’ом). Jquery имеет весьма низкий порог вхождения по сравнению с другими библиотеками и фреймворками, имеющими подобное предназначение, поэтому и был выбран в качестве используемого инструмента;
- **Bootstrap** – набор инструментов для верстки веб-страниц.

3.1.1 База данных

В качестве базы данных использовалась PostgreSQL, имеющая встроенный механизм репликации, а также поддерживающая концепцию multi-master. Данная СУБД была использована при разработке системы монито-

ринга, поскольку обладает качественной документацией, а также является надежным и проверенным решением и предоставляет сравнительно простую реализацию multi-master кластера. База данных системы содержит две схемы: monitoring_schema, содержание которой представлено на рис. 3.1, а также configuration_and_log_schema, описание которой представлено ниже.

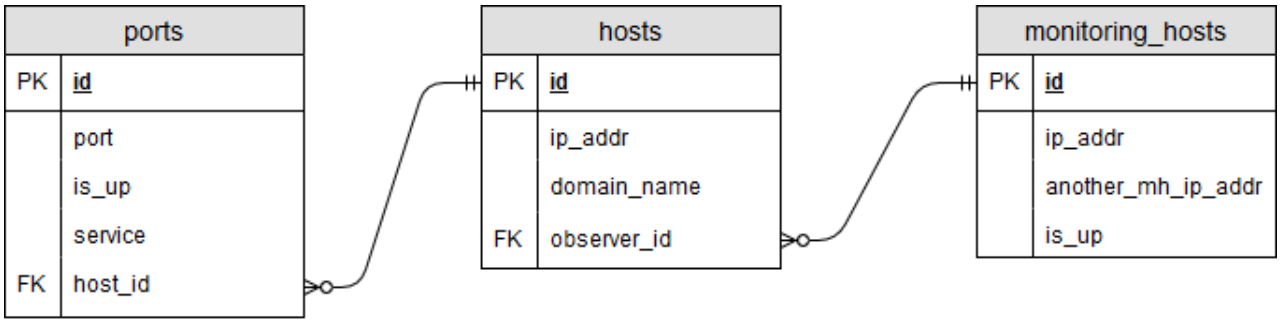

Рисунок 3.1 – Схема monitoring_schema

- **Отношение hosts** содержит информацию о целевых узлах:
  - **id** – целочисленный уникальный суррогатный ключ;
  - **ip_addr** – IP-адрес целевого узла. На уровне базы данных имеет строковый тип, валидация формата IP-адреса происходит на стороне кластера мониторинга;
  - **domain_name** – строковое доменное имя целевого узла (по умолчанию принимает «неизвестно»);
  - **observer_id** – внешний целочисленный ключ, связывающий отношения hosts и monitoring_hosts связью многие к одному, т.е. одному кортежу из отношения monitoring_hosts может соответствовать множество кортежей из отношения hosts.

- **Отношение monitoting_hosts** содержит данные об узлах, входящих в кластер мониторинга:
  - **id** – целочисленный уникальный суррогатный ключ;

- **ip_addr** – IP-адрес узла кластера мониторинга;
- **another_mh_ip_addr** – IP-адрес узла из кластера мониторинга, который опрашивается данным узлом кластера мониторинга. Аналогично с IP-адресом на имеет строковый тип на уровне базы данных, валидация формата адреса происходит на стороне кластера мониторинга;
- **is_up** – поле логического типа, информирующее о рабочем или нерабочем состоянии узла кластера мониторинга.

- **Отношение ports** содержит информацию о портах целевых узлов:
  - **id** – целочисленный уникальный суррогатный ключ;
  - **port** – целочисленный номер порта;
  - **is_up** – поле логического типа, информирующее о рабочем или нерабочем состоянии порта целевого узла;
  - **service** – строковое поле, хранящее информацию о протоколе, по которому взаимодействует сервис на данном порте;
  - **host_id** – внешний целочисленный ключ, связывающий отношения ports и hosts связью многие к одному, т.е. одному кортежу из отношения hosts может соответствовать множество кортежей из отношения ports.

Все сущности находятся в 1,2 и 3 нормальных формах.

В схему configuration_and_log_schema входят две не связанных друг с другом таблицы, одна из которых является вспомогательной и хранит в себе лишь одно значение – частоту мониторинга целевых узлов, а во второй таблице хранятся исторические данные в виде кортежей из трех строковых полей: дата, порт и событие. Таблица исторических данных представлена на рис. 3.2

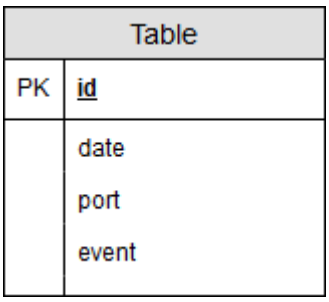

Рисунок 3.2 – Таблица исторических данных

Отношение ports содержит информацию о портах целевых узлов:

- **id** – целочисленный уникальный суррогатный ключ;
- **date –** дата и время события;
- **port** – IP адрес и номер порта;
- **event –** информация о случившемся событии.

Перераспределение целевых узлов при выходе из строя или удалении, а также при добавлении или возвращении к работе узлов кластера мониторинга происходит с помощью триггерных функций написанных на языке plpgql – расширении языка SQL, предоставляемом СУБД PostgreSQL. Выбор в пользу применения триггерных функций сделан из соображений скорости, т.к. выполнение алгоритма на узлах мониторинга потребует дополнительных запросов к БД. Исходный код триггерных функций представлен в приложении В.

Кластеризация по концепции multi-master реализована с помощью инструмента BDR(Bi-Directional Replication for postgresql). В основе реализации BDR лежит логическая асинхронная репликация [10].

Далее подробно описана программная реализация модулей, входящих в состав узла кластера мониторинга. Основные классы и взаимодействие между ними в системе, а также их ключевые атрибуты отражены на рисунке Б.1 приложения Б. В иллюстрации из приложения Б опущены некоторые вспомогательные классы, а также слой взаимодействия с базой данных.

3.1.2 Модуль мониторинга

Модуль мониторинга включает в себя слой бизнес-логики, отвечающий за опрос портов целевых узлов и имеет стандартную структуру Maven проекта (рис. 3.3) – в корне находится pom файл, который содержит в себе информацию о включаемых в проект зависимостях в виде различных библиотек и фремворков; интерфейсы и классы хранятся в папке src/main/java, а в папке resources находится файл настроек проекта application.properties. Исходный код модуля представлен в приложении Г.

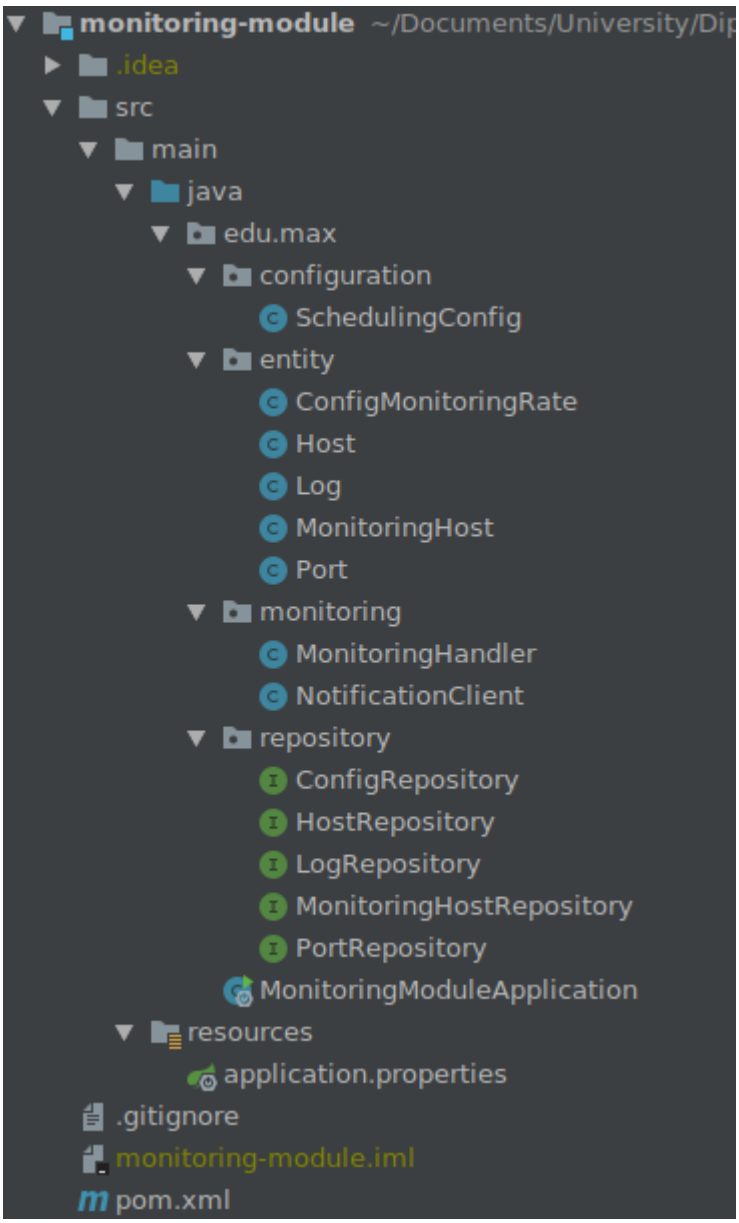

Рисунок 3.3 – Структура модуля мониторинга

Папки entity и repository формируют слой взаимодействия с базой данных. Для работы с БД используется концепция ORM (Объектно-реляционное отображение). ORM отображает таблицы реляционной базы данных и отношения между ними на java-классы, сформированные по определенным правилам – такие классы называются сущностями (Entity) [11]. Таким образом, появляется возможность работать с сущностями базы данных в объектно-ориентированной парадигме, те или иные действия сущностями будут автоматически преобразовываться ORM в SQL-запросы к базе данных. Необходимые инструменты ORM предоставляются фреймворком Spring Data.

Каждому классу из папки entity соответствует таблица в БД:

- Host – соответствует таблице целевых узлов;
- Port – таблице опрашиваемых портов целевых узлов;
- MonitoringHost – таблице узлов кластера мониторинга;
- Log – таблице с историческими данными;
- ConfigMonitoringRate – вспомогательной таблице с частотой мониторинга целевых узлов.

В папке repository находятся интерфейсы, которые содержат методы, используемые для взаимодействия с базой данных (удаление, добавление, обновление, поиск по конкретным полям и т.д.). Для каждой сущности есть свой интерфейс, унаследованный от интерфейса JpaRepository, предоставляемого фреймворком Spring Data. Наследование JpaRepository позволяет не создавать конкретной реализации интерфейса взаимодействия с БД, необходимые методы генерируются автоматически, основываясь на сигнатуре метода интерфейса, написанной по установленным правилам. Некоторые сложные запросы не могут быть сгенерированы таким образом, но в конкретной реализации возможностей Spring Data вполне достаточно для того, чтобы не реализовывать интерфейсы самостоятельно.

Прежде чем рассмотреть механизм мониторинга следует ознакомиться с алгоритмом записи исторических данных. Записи с информацией о доступности порта/сервиса сохраняются в таблицу исторических данных при первом опросе, а в дальнейшем только после того, как недоступный порт вновь станет доступным. Записи с информацией о недоступности портов записываются однократно только после того, как доступный порт становится недоступным. Записи сохраняются как для событий на целевых узлах, так для событий на узлах, входящих в состав кластера мониторинга.

За процесс мониторинга целевых узлов и самой системы отвечают два класса: MonitoringHandler и SchedulingConfig. SchedulingConfig при запуске модуля мониторинга обращается к базе данных с целью получения частоты опроса узлов и вызывает метод класса MonitoringHandler, если частота не установлена, то она принимается равной одной минуте, через промежуток времени равный частоте опроса SchedulingConfig снова обращается к базе данных и устанавливает при необходимости новую частоту мониторинга, данный процесс повторяется до выключения или выхода из строя модуля мониторинга. По умолчанию частота опроса мониторинга устанавливается в одну минуту. MonitoringHandler осуществляет следующую последовательность действий:

- Запрашивает у базы данных из таблицы monitoring_hosts IP-адрес другого узла кластера мониторинга (поле ip_addr), закрепленного за этим узлом для опроса и, если такой адрес есть в БД, пытается установить соединение с опрашиваемым узлом. При удачном или неудачном соединении с узлом кластера в БД соответствующим образом обновляется поле is_up таблицы monitoring_hosts для записи с IP-адресом опрашиваемого узла кластера мониторинга. При необходимости будет добавлена запись в таблицу исторических данных;
- Запрашивает у БД IP-адреса целевых узлов, закрепленных за данным узлом мониторинга и номера их портов;

- Для каждого IP-адреса целевого узла выполняются действия:
    - Для каждого порта целевого узла MonitoringHandler последовательно пытается установить соединение по протоколам HTTP, SMTP, POP3, SSH, FTP, для чего используются библиотека org.apache.commons.net, предоставляющая интерфейсы для взаимодействия по вышеперечисленным протоколам;
    - Если порт целевого узла корректным образом отвечает по одному из протоколов, в таблице ports обновляются поля is_up и service для записи соответствующей данному порту. При необходимости будет добавлена запись в таблицу исторических данных;
    - Если не удалось установить соединения ни по одному из протоколов, в БД соответствующим образом обновляется запись для данного порта целевого узла. При необходимости будет добавлена запись в таблицу исторических данных. Также будет отправлено соответствующее сообщение на электронную почту администратору с помощью модуля оповещения.

Связь с модулем оповещения осуществляется с помощью класса NotificationClinet, который является реализацией клиента для модуля оповещения. С помощью данного класса можно отправлять текст сообщений, которые будет доставлены администратору на электронную почту, подробнее этот процесс раскрыт ниже, в описании реализации модуля оповещения.

В файле application.properties хранится информация, необходимая для подключения к серверу БД, адрес узла и прочая служебная информация. Класс MonitoringModuleApplication является точкой входа программы и связующим звеном для ее компонентов с использованием фреймворка Spring-Boot.

3.1.3 Модуль оповещения

Модуль оповещения включает в себя слой бизнес-логики, отвечающей за оповещение администратора о событиях в системе по электронной почте. Реализация конечного функционала модуля оповещения зависит от требований, предъявляемых к функциям оповещения: способ и частота оповещений, каналы оповещений (электронная почта, SMS, звонок и т.п.). В рамках ВКР реализована отправка уведомлений по электронной почте. Модуль имеет стандартную структуру Maven проекта (рис. 3.4). Исходный код модуля представлен в приложении Д.

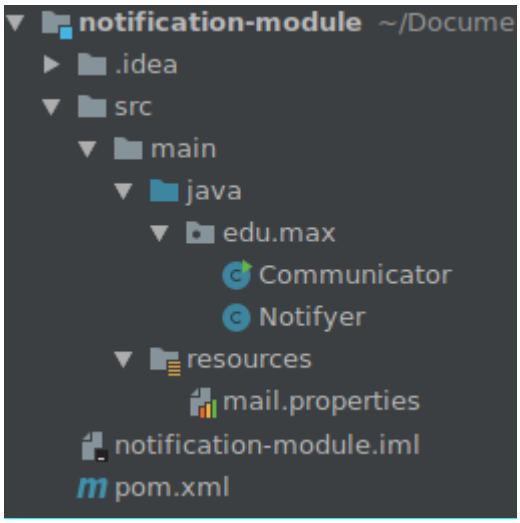

Рисунок 3.4 – Структура модуля оповещения

Класс Communicator является точкой входа модуля и выполняет функции сервера, который прослушивает указанный в конфигурационном файле порт по протоколу TCP. При установлении соединения с сервером оповещения клиент должен отправить серверу приветствие определенного формата, если приветствие будет иметь формат отличный от установленного, сервер прервет соединение с клиентом, иначе – клиент получит ответное приветственное сообщение от сервера и сможет отправить текст сообщения, которое должно быть доставлено на электронную почту администратора. Непосредственной доставкой сообщения на электронную почту занимается класс Notifyer, используется java библиотека javax.mail.

Все необходимые настройки, включающие в себя номер порта сервера оповещения, электронную почту администратора и пр. указываются в файле mail.properties, находящемся в папке resources.

3.1.4 Web-модуль (Модуль пользовательского интерфейса)

Web-модуль соответствует концепции REST (REpresentational State Transfer) архитектуры. REST – это архитектурный стиль для создания сетевого приложения, в котором ключевую роль играют ресурсы. В качестве ресурсов могут выступать HTML-документы, изображения, файлы и т.д. Каждый ресурс имеет свой адрес, например, URI - унифицированный идентификатор ресурса, определяющий имя и адрес ресурса в сети Internet. В данной архитектуре подразумевается использование «клиент-серверной» модели, где сервер хранит ресурсы, а клиент запрашивает их получение, делает запросы на удаление или изменение ресурсов, а также на добавление новых. Взаимодействие между клиентом и сервером или промежуточными компонентами должны происходить через унифицированный интерфейс, при этом серверу не требуется запоминать состояние клиента, а сообщения клиента должны быть самодостаточными, т.е. давать исчерпывающую информацию для формирования сервером ответа на запрос клиента [12].

Так же как и модуль мониторинга web-модуль имеет стандартную структуру Maven проекта (рис. 3.5 и 3.6). Исходный код модуля представлен в приложении Е.

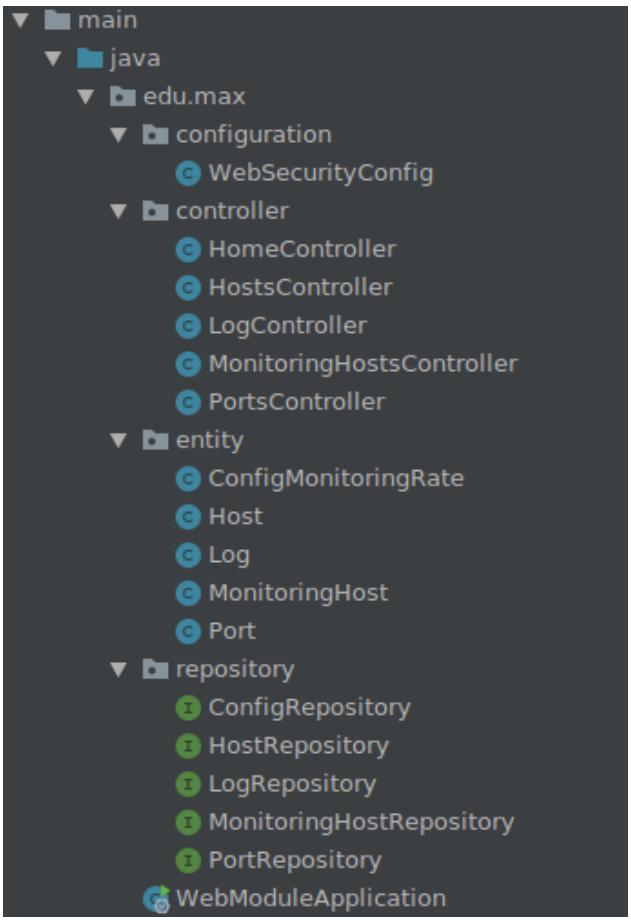

Рисунок 3.5 – Структура исходного каталога модуля

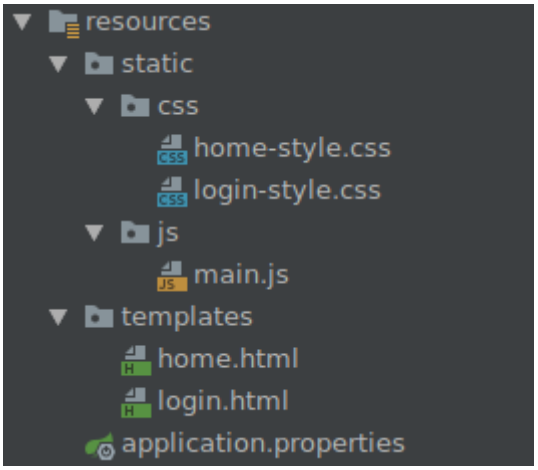

Рисунок 3.6 – Структура каталога ресурсов

Папки entity и repository дублируют папки из модуля мониторинга и представляют собой слой взаимодействия с БД. В папке controller располо-

жены REST-контроллеры, которые принимают запросы по конкретным URI и возвращают запрошенные ресурсы или ответы, информирующие об успехе или неудаче выполнения запроса клиента, например, при отправке клиентом GET-запроса по протоколу HTTP по адресу «/hosts», будет вызван соответствующий метод контроллера HostsController, который обратится к базе данных с запросом на получение информации обо всех целевых узлах и вернет полученную информацию пользователю. Различные контроллеры отвечают за разные адреса и разделены по зоне ответственности: HostsController обрабатывает запросы связанные с целевыми узлами, PortsController – запросы связанные с портами целевых узлов, MonitoringHostsController отвечает за все запросы связанные с узлами кластера мониторинга , LogController предназначен для получения и удаления исторических данных, HomeController возвращает главную страницу.

Класс WebSecurityConfig наследуется от класса WebSecurityConfigurerAdapter, который предоставляется фремворком Spring Security. Данный класс задает полномочия пользователя и путь к HTML-странице входа в систему. Имя пользователя и пароль хранятся в файле application.properies и могут задаваться при установки модуля на узел системы, при этом на каждом узле должны быть установлены одинаковые логин и пароль.

В папке resources хранятся статические HTML-страницы, а также их CSS-стили и скрипты, написанные на языке javascript, скрипты отвечают за динамику страницы, а также реализуют Ajax-запросы (асинхронные запросы) к серверу для получения необходимой информации или внесения данных на сервер.

3.2 Разработка пользовательского интерфейса и сценарии использования

3.2.1 Пользовательский интерфейс

Пользовательский интерфейс представляет собой web-сайт, доступный по некоторому доменному имени http://domain.name. При входе в систему пользователю будет предложено ввести имя пользователя и пароль на отдельной странице с формой для входа. После успешного ввода данных администратор попадает в панель управления, изображенную на рис. 3.7.

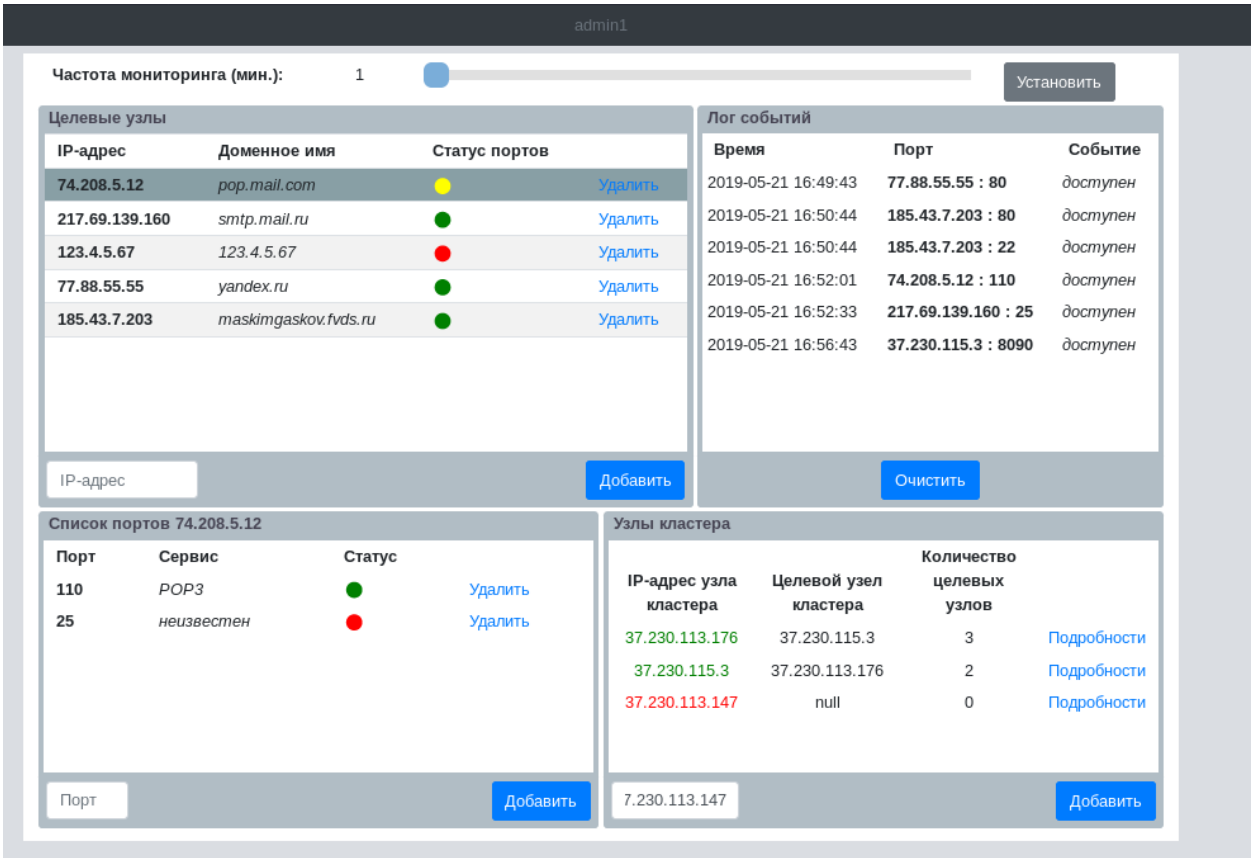

Рисунок 3.7 – Панель управления системой мониторинга

Экран системы мониторинга включает в себя слайдер, который позволяет устанавливать частоту опроса целевых узлов в диапазоне от 1 минуты до 1 часа. Также экран содержит четыре панели:

- «Целевые узлы» – панель, в которой отображаются текущие целевые узлы в виде таблицы, в каждой строке которой находится информация об IP-адресе целевого узла, его доменном имени и статусе портов. Статус портов может принимать одно из трех значений и представляет собой индикатор красного, зеленого или желтого цвета, где красный цвет сигнализирует о том, что все порты узла недоступны, желтый говорит о том, что узел доступен, но какие-то из опрашиваемых портов недоступны, зеленый индикатор сигнализирует о том, что узел доступен и все его опрашиваемые порты работают также доступны. Также данная панель позволяет добавлять новые целевые узлы по IP-адресу, а также удалять узлы путем нажатия на кнопку «удалить» в строке с информацией о целевом узле;
- «Лог событий» – данная панель отображает данные о произошедших событиях: время события, порт, на котором произошло событие и описание произошедшего события, например «1998-25-05 16:00 12.34.56.78:8080 недоступен», что говорит о том, что порт 8080 данного узла перестал быть доступным в указанное время, если данный порт станет снова доступным, то об этом выведется соответствующее сообщение в таблицу. Также в панели существует возможность очистки памяти исторических данных;
- «Список портов» – данная панель отображает список портов целевого узла, выбранного путем нажатия на соответствующую строку в панели «Целевые узлы» (данная строка будет выделена более темным цветом относительно остальных). Для каждого порта выводится его номер, протокол по которому работает сервис на данном порту (один из списка: HTTP, SMTP, POP3, SSH, FTP) и статус порта, изображаемый, по аналогии с панелью «Целевыеузлы» красным либо зеленым индикатором, который обозначает доступность или недоступность порта соответственно. Также панель позволяет добав-

лять новые порты, которые необходимо опрашивать, по их номеру и удалять порты путем нажатия соответствующей кнопки;

- «Узлы кластера» – панель, отображающая узлы кластера мониторинга и их состояние. Строка из таблицы данной панели содержит следующую информацию: IP-адрес узла кластера, который выделен зеленым или красным цветом в зависимости от его доступности; IP-адрес другого узла из кластера мониторинга, опрос которого осуществляется с целью внутреннего мониторинга кластера (в таблице – столбец «Целевой узел кластера); количество целевых узлов мониторинга (учитываются только целевые узлы), кнопку «Подробности», нажав на которую, в сплывающем модальном окне можно получить информацию об IP-адресах выбранного узла мониторинга, а также удалить выбранный узел кластера из кластера мониторинга.

3.2.2 Сценарии использования

Сценарии использования представлены в виде UML (Unified Modeling Language – унифицированный язык моделирования) диаграммы (рис. 3.8), которая включает в себя одно действующее лицо – администратора системы и прецеденты использования, некоторые из которых объединены отношением включения. Прецеденты использования – это конкретные способы взаимодействия с системой. Отношение включения («include») между прецедентами использования означает, что при выполнении прецедента, из которого исходит отношение включения, обязательно должен быть выполнен включаемый прецедент [13].

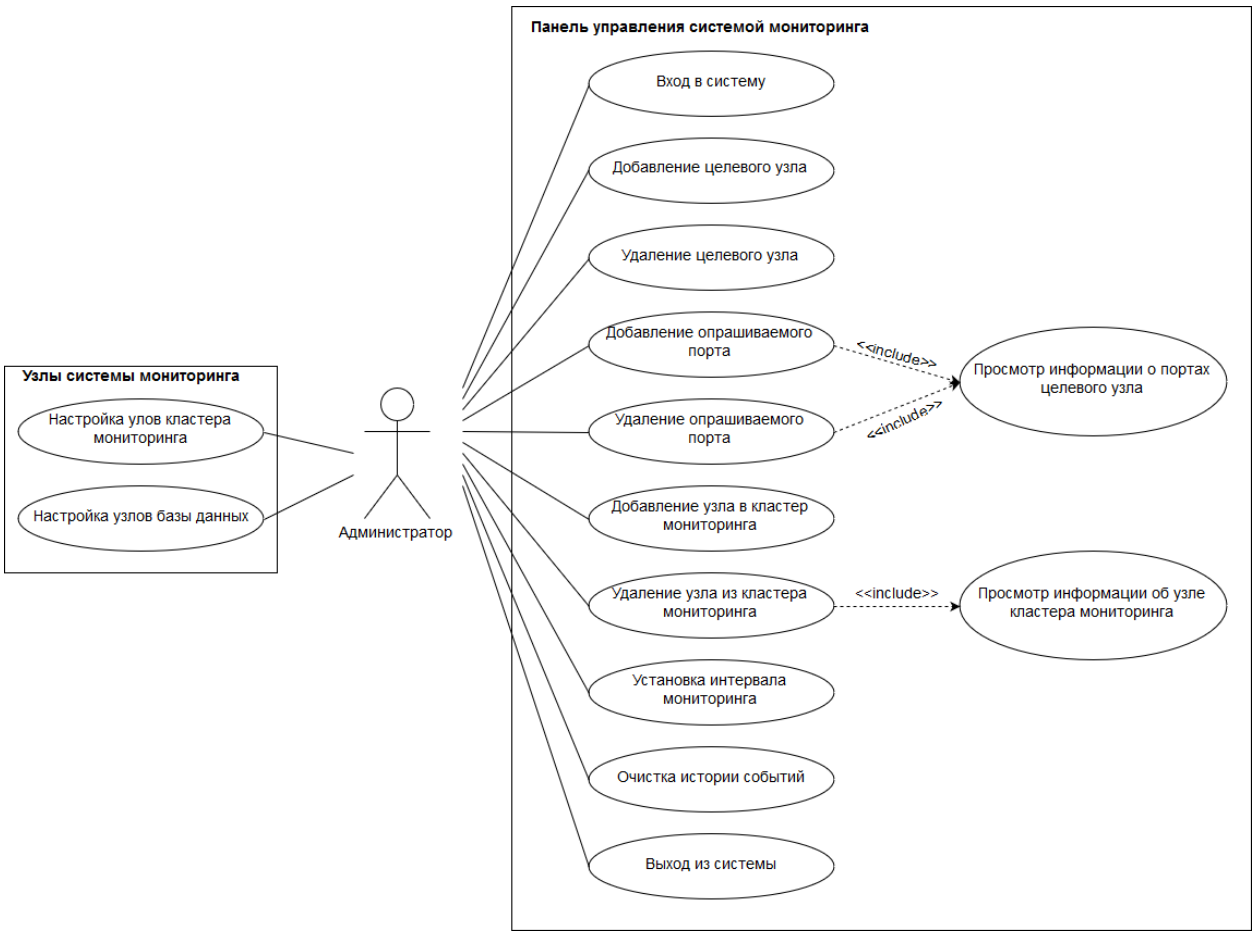

Рисунок 3.8 – UML диаграмма прецедентов

Прецеденты описаны в приложении А. Во всех случаях действующим лицом является администратор. Понятия «пользователь» и «администратор» в контексте описания сценариев взаимозаменяемы.

3.3 Требования к окружению для развертывания системы, процесс инсталляции

3.3.1 Кластер БД

Релиз для ОС Linux Ubuntu, отвечающий за установку и настройку базы данных, представлен по ссылке:

https://github.com/MaximGaskov/distributed_monitoring_system/releases/tag/db1.0

Требования: Linux Ubuntu версии 16 и выше. Установка может работать корректно и на более ранних версиях Linux Ubuntu, однако такое тестирование не проводилось.

Для установки кластера базы данных в системе необходимо скачать архив по ссылке, распаковать его в любую директорию, после чего необходимо:

- Создать пользователя без пароля с именем «postgres» в системе;
- Запустить с правами администратора скрипт «start_setup.sh», находящийся в распакованном архиве в папке «commands». Данный скрипт выполняет установку и сборку программного обеспечения, необходимого для установки и работы кластера БД, а также выполняет некоторые служебные операции;
- Выполнить скрипт «configure_postgresql.sh», который внесет необходимые изменения в конфигурационные файлы СУБД;
- Далее следует разветвление в возможном сценарии установки:
    - Если пользователь хочет создать кластер «с нуля» ему необходимо выполнить скрипт «setup_initial_node.sh», который принимает два параметра: первым аргументом передается IP-адрес узла, который станет первым в кластере БД, вторым аргументом указывается уникальное имя этого узла в кластере. Выполнение данного скрипта создаст кластер базы данных, а также инициализируют саму базу данных с необходимыми схемами и отношениями и установит триггерные функции, реализующие алгоритмы перераспределение целевых узлов между узлами кластера мониторинга;

- Если же в кластере уже имеются сервера, пользователь должен запустить скрипт под названием «setup_secondary_node.sh» и передать в качестве аргументов: IP-адрес узла, на котором запущен скрипт; Уникальное название узла в кластере; IP-адрес одного из узлов, уже входящих в кластер БД.

Все узлы, на которых будет произведена установка, образуют кластер по стратегии multi-master с настроенной асинхронной репликацией между узлами. Теоретически возможно неограниченное масштабирование кластера, однако это приведет к падению уровня согласованности данных между узлами БД.

3.3.2 Кластер мониторинга

Релиз для ОС Linux, отвечающий за установку и настройку узлов кластера мониторинга, представлен по ссылке:

https://github.com/MaximGaskov/distributed_monitoring_system/releases/tag/Monsys1.1

Требования:

- ОС Linux. Тестирования проводились на ОС Linux Ubuntu 16.04, однако реализованные скрипты для автоматизации установки не содержат комманд специфических для конкретного дистрибутива Linux;
- JDK(Java Development Kit) версии 8 и выше, необходимый для работы скрипта остановки системы, однкао достаточно наличия в системе JRE(Java Runtime Environment) версии 8 и выше, но в таком случае завершение работы модулей придется производить вручную.

Для установки модулей на узел кластера мониторинга необходимо:

- скачать архив по ссылке указанной выше;

- После распаковывания архива перейти в папку, в которую распакован архив. В данной папке находятся три jar-архива – модули мониторинга, оповещения и пользовательского интерфейса, также в папке находится файл «properties» с конфигурацией узла кластера мониторинга;
- Изменить конфигурационный файл (указать IP-адрес узла, URL базы данных, имя пользователя БД и пароль и другую необходимую информацию);
- После сохранения изменений в конфигурационном файле можно выполнить скрипт «run.sh», который запускает модули, передавая в них необходимые настройки из файла конфигурации. Ход процесса запуска модулей выводится в, созданные после запуска скрипта, файлы с именами «log-monitoring-module.txt», «log-notification-module.txt» и «log-web-module.txt»;
- Для остановки работы всех модулей нужно запустить скрипт «stop.sh».

4 ТЕСТИРОВАНИЕ РАЗРАБОТАННЫХ ПРОГРАММНЫХ СРЕДАСТВ

4.1 Организация тестового стенда

Инфраструктура, на которой производилось тестирование системы, включает в себя 5 виртуальных серверов, арендованных у одного из провайдеров. Три сервера были выделены под кластер мониторинга, на них были запущены модули мониторинга, пользовательского интерфейса и оповещения. Каждый сервер имеет свое доменное имя, для простоты в данном разделе сервера кластера мониторинга будут иметь имена «serv1», «serv2», «serv3». Два оставшихся сервера «dbserv1» и «dbserv2», имеющие общее доменное имя, образуют кластер базы данных с настроенными механизмом multi-master и необходимыми триггерными функциями.

Все сервера имеют одинаковые характеристики:

- ОС: Linux Ubuntu 16.04-x86_64;
- Процессор: Intel Xeon 2,4 ГГц;
- Объем оперативной памяти: 1 ГБ;
- Объем жесткого диска HDD: 30 ГБ.

В качестве целевых узлов были использованы общедоступные сервера с доменными именами:

- «yandex.ru» – несколько серверов под общим доменным именем, которые предоставляют web-сервис по протоколу HTTP на порту под номером 80;

- «smtp.mail.ru» – несколько серверов под общим доменным именем, которые предоставляют почтовый сервис по протоколу SMTP на порту под номером 25;
- «pop.mail.com» – сервер, предоставляющий почтовый сервис по протоколу POP3 на порту под номером 110;
- «ftp.mccme.ru» – сервер, предоставляющий сервис обмена файлами по протоколу FTP на порту под номером 21;
- «monsysdb1.tavg.net» – сервер dbsrerv1 с сервисом удаленного доступа по протоколу SSH на порту 22.

4.2 Разработка тестовых сценариев

Под тестовым сценарием (тест-кейсом) понимается некоторый набор входных данных или описание шагов, предусловия и результаты выполнения [14]. Все представленные в данном подразделе тестовые сценарии прошли проверку в тестовой системе. т.е. привели к ожидаемым результатам при заданных входных данных и предусловиях. При тестировании была использована стратегия «черного ящика», при такой стратегии программная реализация не рассматривается, и ключевую роль играют входные данные [15]. Такая стратегия позволяет имитировать работу пользователя с системой, поэтому более наглядна для читателя. В списке тестовых сценариев ниже не описаны простейшие сценарии использования пользовательского интерфейса, которые были описаны в подразделе “Сценарии использования” раздела 3, и их ожидаемые результаты.

1) Перераспределение целевых узлов при добавлении нового узла в кластер мониторинга:

Изначально в кластере мониторинга 1 узел

Предусловия:

- В кластере мониторинга находится один узел serv1 в рабочем состоянии;
- В списке целевых узлов находятся 7 серверов с доменными указанными в списке общедоступных серверов выше;
- Все целевые узлы опрашиваются узлом serv1;

Шаги:

- Запустить модуль мониторинга на узле serv2;
- Добавить serv2 по IP-адресу в список узлов кластера мониторинга.

Ожидаемые результаты:

- За узлом serv1 закрепляются 3 целевых узла, а за узлом serv2 - 4 (Возможен и обратный вариант).

Изначально в кластере мониторинга 2 узла

Предусловия:

- В кластере мониторинга находятся узлы serv1 и serv2 в рабочем состоянии;
- В списке целевых узлов находятся 7 серверов с доменными указанными в списке общедоступных серверов выше;
- Распределение целевых узлов:
    - serv1 - 3
    - serv2 - 4

Шаги:

- Запустить модуль мониторинга на узле serv3;
- Добавить serv3 по IP-адресу в список узлов кластера мониторинга.

Ожидаемые результаты:

- Распределение целевых узлов:
    - serv1 - 2
    - serv2 - 3
    - serv3 - 3

(Возможны варианты: 3, 2, 3 и 3, 3, 2)

2) Перераспределение целевых узлов при удалении узла из кластера мониторинга:

Изначально в кластере мониторинга 2 узла

Предусловия:

- В кластере мониторинга находятся узлы serv1 и serv2 в рабочем состоянии;
- Распределение целевых узлов:
  - serv1 - 3
  - serv2 - 4

Шаги:

- Удалить serv1 из списка узлов кластера мониторинга (или симулировать выход узла из строя, прекратив работу модуля мониторинга на узле serv1).

Ожидаемые результаты:

- За узлом serv2 закрепляются все 7 целевых узлов.
- Администратор получает сообщение на почту о выходе из строя узла serv1;
- В панели «Лог событий» появляется запись о том, что serv1 вышел из строя.

Изначально в кластере мониторинга 3 узла

Предусловия:

- В кластере мониторинга находятся узлы serv1, serv2 и serv3 в рабочем состоянии;
- Распределение целевых узлов:
  - serv1 - 2
  - serv2 - 3

- serv3 - 3

Шаги:

- Удалить serv1 из списка узлов кластера мониторинга (или симулировать выход узла из строя, прекратив работу модуля мониторинга на узле serv1).

Ожидаемые результаты:

- Распределение целевых узлов:
  - serv2 - 3
  - serv3 - 4

  (Возможен вариант: 4, 3)
- Администратор получает сообщение на почту о выходе из строя узла serv1;
- В панели «Лог событий» появляется запись о том, что serv1 вышел из строя.

3) Внутренний мониторинг системы. Удаление узлов из кластера мониторинга:

Изначально в кластере мониторинга 3 узла

Предусловия:

- В кластере мониторинга находятся узлы serv1, serv2 и serv3 в рабочем состоянии;
- Внутренний мониторинг:
  - serv1 ведет наблюдение за serv2;
  - serv2 ведет наблюдение за serv3;
  - serv3 ведет наблюдение за serv1.

Шаги:

- Остановить модуль мониторинга на узле serv2;

Ожидаемые результаты:

- Внутренний мониторинг:
    - serv1 ведет наблюдение за serv3;
    - serv3 ведет наблюдение за serv1.
- Администратор получает сообщение на почту о выходе из строя узла serv2;
- В панели «Лог событий» появляется запись о том, что serv2 вышел из строя.

Изначально в кластере мониторинга 2 узла

Предусловия:

- В кластере мониторинга находятся узлы serv1 и serv2 в рабочем состоянии;
- Внутренний мониторинг:
    - serv1 ведет наблюдение за serv2;
    - serv2 ведет наблюдение за serv1.

Шаги:

- Остановить модуль мониторинга на узле serv1.

Ожидаемые результаты:

- Узел мониторинга не ведет наблюдение за serv1.
- Администратор получает сообщение на почту о выходе из строя узла serv1;
- В панели «Лог событий» появляется запись о том, что serv1 вышел из строя.

4) Внутренний мониторинг системы. Добавление узлов в кластер мониторинга:

Изначально в кластере мониторинга 2 узла

Предусловия:

- В кластере мониторинга находятся узлы serv1 и serv2 в рабочем состоянии;
- Внутренний мониторинг:
    - serv1 ведет наблюдение за serv2;
    - serv2 ведет наблюдение за serv1.

Шаги:

- Внести узел serv3 в список узлов кластера мониторина;
- Запустить модуль мониторинга на узле serv3.

Ожидаемые результаты:

- Внутренний мониторинг:
    - serv1 ведет наблюдение за serv2;
    - serv2 ведет наблюдение за serv3;
    - serv3 ведет наблюдение за serv1.

Изначально в кластере мониторинга 1 узел

Предусловия:

- В кластере мониторинга находится узел serv1 в рабочем состоянии;

Шаги:

- Добавить узел мониторинга serv2 в список узлов кластера мониторинга;
- Запустить модуль мониторинга на узле serv2.

Ожидаемые результаты:

- Внутренний мониторинг:
    - serv1 ведет наблюдение за serv2;
    - serv2 ведет наблюдение за serv1.

5) Выход из строя единственного узла мониторинга. Восстановление системы:

Предусловия:

- В кластере мониторинга находится узел serv1 в рабочем состоянии;
- serv1 опрашивает 7 целевых узлов.

Шаги:

- Приостановить работу модуля мониторинга на узле serv1;
- Добавить узел serv2 в список узлов кластера мониторинга;
- Запустить узел мониторинга на узле serv2.

Ожидаемые результаты:

- serv2 опрашивает 7 целевых узлов;
- Администратор получает сообщение на почту о выходе из строя узла serv1;
- В панели «Лог событий» появляется запись о том, что serv1 вышел из строя.

6) Тестирование опроса доступности сервисов:

Сервис, работающий по протоколу HTTP:

Предусловия:

- В кластере мониторинга находится узел serv1 в рабочем состоянии;

Шаг:

- Добавить в список целевых узлов один из узлов с доменным именем «yandex.ru». Выбран целевой узел с IP-адресом 77.88.55.55.
- Добавить в список опрашиваемых портов целевого узла порт под номером 80.

Ожидаемые результаты:

- По прошествии времени равному установленному промежутку между опросами узлов порт 80 целевого узла помечается как доступный;
- В поле информации о протоколе сервиса порта значение «неизвестно» заменяется на «HTTP»;

- В лог событий добавляется запись о доступности сервиса на порту 80 по IP-адресу целевого узла.

Сервис, работающий по протоколу SMTP:

Предусловия:

- В кластере мониторинга находится узел serv1 в рабочем состоянии;

Шаг:

- Добавить в список целевых узлов один из узлов с доменным именем «smtp.mail.ru». Выбран целевой узел с IP-адресом 217.69.139.160.
- Добавить в список опрашиваемых портов целевого узла порт под номером 25.

Ожидаемые результаты:

- По прошествии времени равному установленному промежутку между опросами узлов порт 25 целевого узла помечается как доступный;
- В поле информации о протоколе сервиса порта значение «неизвестно» заменяется на «SMTP»;
- В лог событий добавляется запись о доступности сервиса на порту 25 по IP-адресу целевого узла.

Сервис, работающий по протоколу POP3:

Предусловия:

- В кластере мониторинга находится узел serv1 в рабочем состоянии;

Шаг:

- Добавить в список целевых узлов узел с доменным именем «pop.mail.com» и IP-адресом 74.208.5.12.
- Добавить в список опрашиваемых портов целевого узла порт под номером 110.

Ожидаемые результаты:

- По прошествии времени равному установленному промежутку между опросами узлов порт 110 целевого узла помечается как доступный;
- В поле информации о протоколе сервиса порта значение «неизвестно» заменяется на «POP3»;
- В лог событий добавляется запись о доступности сервиса на порту 110 по IP-адресу целевого узла.

Сервис, работающий по протоколу FTP:

Предусловия:

- В кластере мониторинга находится узел serv1 в рабочем состоянии;

Шаг:

- Добавить в список целевых узлов узел с доменным именем «ftp.mccme.ru» и IP-адресом 185.54.136.70.
- Добавить в список опрашиваемых портов целевого узла порт под номером 21.

Ожидаемые результаты:

- По прошествии времени равному установленному промежутку между опросами узлов порт 21 целевого узла помечается как доступный;
- В поле информации о протоколе сервиса порта значение «неизвестно» заменяется на «FTP»;
- В лог событий добавляется запись о доступности сервиса на порту 21 по IP-адресу целевого узла.

Сервис, работающий по протоколу SSH:

Предусловия:

- В кластере мониторинга находится узел serv1 в рабочем состоянии;

Шаг:

- Добавить в список целевых узлов узел с доменным именем «monsysdb1.tavg.net» и IP-адресом 185.43.7.203.

- Добавить в список опрашиваемых портов целевого узла порт под номером 22.

Ожидаемые результаты:

- По прошествии времени равному установленному промежутку между опросами узлов порт 22 целевого узла помечается как доступный;
- В поле информации о протоколе сервиса порта значение «неизвестно» заменяется на «SSH»;
- В лог событий добавляется запись о доступности сервиса на порту 22 по IP-адресу целевого узла.

Сервис, работающий по протоколу SSH. Симуляция выхода из строя:

Предусловия:

- В кластере мониторинга находится узел serv1 в рабочем состоянии;
- serv1 опрашивает сервис, работающий по протоколу SSH на узле с доменным именем «monsysdb1.tavg.net» и IP-адресом 185.43.7.203.

Шаг:

- Установить минимальную частоту мониторинга;
- Запустить перезагрузку операционной системы на целевом узле.

Ожидаемые результаты:

- Во время перезагрузки сервера, сервис на порту 22 перестанет быть доступным, что будет отмечено в системе;
- В лог событий добавляется запись о доступности сервиса на порту 22 по IP-адресу целевого узла.

При реализации тестовых сценариев полученные результаты соответствуют ожидаемым, таким образом, можно сделать вывод о работоспособности и корректности функционирования разработанных программных средств.

5 РАЗРАБОТКА И СТАНДАРТИЗАЦИЯ ПРОГРАММНЫХ СРЕДСТВ

5.1 Планирование работ проекта

На практике применяются несколько способов формализованного представления совокупности планируемых работ, основанных на графическом отображении процессов и позволяющих в той или иной степени контролировать ход работ и вносить необходимые изменения в их организацию. Наиболее часто для целей планирования и управления проектными работами используют линейные графики (диаграммы Ганта), операграммы, а также сетевые графики. В качестве целей планирования и управления проектными работами была выбрана диаграмма Ганта.

Диаграмма Ганта (называемая также линейным или ленточным графиком) представляет собой отображение работ в виде протяженных во времени отрезков. Помимо изображаемой в том или ином масштабе общей длительности каждая работа может характеризоваться датами её начала и завершения, исполнителем, а некоторых случаях и иным параметрами. Диаграмма Ганта для разрабатываемого приложения калькулятора с рукописным вводом представлена в табл. 4.1. На представленной диаграмме Ганта введены обозначения для исполнителей различных видов работ: разработчик и руководитель обозначены буквами А и В, соответственно.

Подобная диаграмма позволяет наглядно представить количество работ для завершения проекта, процесс выполнения этих работ во времени и показать ответственных за каждую работу. В то же время, взаимосвязи между отдельными работами в явном виде на диаграмме не отображаются, что создает трудности при необходимости корректировки общего графика работ из-за изменения сроков выполнения какой-либо работы. Не отражает график и потребные для исполнения работ ресурсы, а также взаимосвязи между соисполнителями одной и той же работы.

Таблица 5.1 – Диаграмма Ганта выполнения комплекса проектных работ

Работы	Начало	Конец	Временные периоды (в неделях)						Исполнители
			I	II	III	IV	V	VI	
Постановка задачи и ее описание	27.04	28.04	–						A, B
Изучение материала по теме	29.04	04.05	—						A
Разработка архитектуры распределенной системы	05.05	06.05		-					A
Реализация кластера мониторинга	07.05	14.05		—	—				A
Установка и настройка серверов баз данных	15.05	18.05			—	—			A
Установка и настройка серверов кластера монитроинга	19.05	21.05				–			A

Продолжение таблицы 5.1

Установка программных модулей кластера мониторинга на сервера	22.05	23.05				-			А
Тестирование распределенной системы	24.05	25.05					-		А, В
Оформление пояснительной записки ВКР	26.05	31.05							А
Подготовка к защите ВКР	01.06	7.06							А, В

5.2 Классификация разрабатываемого продукта

ОКПД 2 – Общероссийский Классификатор Продукции по видам экономической Деятельности [16].

Коды ОКПД 2 это классификаторы продукции, работ и услуг. Такая классификация конкретизирует работы, товары и услуги, которые необходимо получить от поставщика при проведении различного рода государственных торгов, формировании заказов на выполнение задач, необходимых для удовлетворения государственных и муниципальных нужд.

Согласно приказу Росстандарта от 31.01.2014 №14-ст действует классификатор ОКПД 2 (ОК 034-2014 (КПЕС 2008)). ОКПД 2 дополнил преды-

дущую версию – ОКПД. С 1 января 2017 года стал обязательным условием использование только классификатора ОКПД 2. До этого момента шел переходный период, когда возможным считалось использование и старой версии (т.е. ОКПД).

К заказчику, при составлении плана-графика закупок, предъявляется требование указания кода ОКПД 2 с указанием класса, подкласса, группы, подгруппы и вида продукта закупки. ОКПД 2 имеет иерархический вид представленный в табл. 5.2.

Таблица 5.2 – Структура классификации ОКПД 2

XX	класс
XX.X	подкласс
XX.XX	группа
XX.XX.X	подгруппа
XX.XX.XX	вид
XX.XX.XX.XX	категория
XX.XX.XX.XXX	подкатегория

Классификатор ОКПД 2 Российской Федерации не должен противоречить данным КПЕС 2008 (Классификатору продукции Европейского союза). И для их соответствия в классификаторе присутствует разделение точкой между знаками кода. Самые крупные разделы имеют буквенное обозначение на латинице. Если в ОКПД 2 присутствует большее количество уровней вложенности по сравнению с Европейским классификатором, то точкой разделяются также шестой и седьмой знаки. В этом случае, если вид имеет деление на категории и должна быть указана детализация, то указывается обозначение 7-9 знаков отличное от нуля. Если такая подробная детализация отсутствует, то 7-9 знаки имеют значение ноль. Такое разделение характерно в тех

случаях, когда категория продукции (работ или услуг) имеет несколько более мелких и узких подкатегорий.

Распределенная система мониторинга имеет код 62.01.12.000:

- Класс 62 – Продукты программные и услуги по разработке программного обеспечения; консультационные и аналогичные услуги в области информационных технологий;
- Подкласс 62.0 – Продукты программные и услуги по разработке программного обеспечения; консультационные и аналогичные услуги в области информационных технологий;
- Группа 62.01 – Продукты программные и услуги по разработке и тестированию программного обеспечения;
- Подгруппа 62.01.1 – Услуги по проектированию, разработке информационных технологий для прикладных задач и тестированию программного обеспечения;
- Вид 62.01.12 – Услуги по проектированию и разработке информационных технологий для сетей и систем;
- Категория и подкатегория 62.01.12.000 – Услуги по проектированию и разработке информационных технологий для сетей и систем.

5.3 Определение затрат на выполнение и внедрение проекта

Затраты на проектирование (АИС или ее части комплекса задач, программного модуля, программного изделия и т.п.) рассчитываются на стадии завершения инициативной разработки АИС или ее части с целью определения рыночной цены созданного продукта (с учетом планируемых объемов ее тиражирования). Состав работ проекта оказывает влияние лишь на общую трудоемкость проекта, тогда как порядок расчетов остается в своей основе неизменным. Экономические расчеты выполняются после завершения проектирования АИС. В данном случае капитальные затраты на создание и внед-

рение АИС являются фактическими. При этом возможно проведения расчетов как до, так и после проведения опытной эксплуатации, затраты на которую также должны быть учтены либо в качестве планируемых, либо в качестве фактических затрат по этой статье расходов. Затраты на проектирование АИС (или ее части) $К_{ПР}$ в общем случае рассчитываются в соответствии с (4.1).

$$К_{ПР} = К_{ПЕРС} + К_{СВТ} + К_{ИПС} + К_{ПРОЧ} \qquad (4.1)$$

где:

$К_{ПЕРС}$ – затраты на оплату труда проектировщиков (персонала) и связанные с этим выплаты;

$К_{СВТ}$ – затраты на средства вычислительной техники, используемые для проектирования;

$К_{ИПС}$ – затраты на инструментальные программные средства;

$К_{ПРОЧ}$ – прочие расходы на проектирование.

При неоднородном составе команды разработчиков затраты на оплату труда проектировщиков КПЕРС могут быть рассчитаны по формуле (4.2):

$$К_{ПЕРС} = \sum_{i=1}^{m_n} Z_{ЗПi} * (1 + Н + Ф) * d_{загрi} * n_{Пi} \qquad (4.2)$$

где индексом i помечены значения $Z_{ЗП}$, $d_{загр}$, $n_{П}$ для i-го участника команды разработчиков.

$Z_{ЗП}$ – заработная плата одного проектировщика за месяц (неделю, день – если оплата назначается за неделю или день);

H – процент (доля) накладных расходов, исчисляемых к сумме зарплаты проектировщиков (определяется по данным бухгалтерии на каждый учетный период времени, чаще всего на квартал);

$Ф$ – процент (доля) отчислений в фонды (пенсионный, медицинского страхования и др.), относимых к единому социальному налогу (утверждается ежегодно Государственной думой РФ);

$d_{загр}$ – доля загрузки проектировщика работой по проекту АИС (если проектировщик загружен работами по проекту не полный рабочий день, выполняя параллельно какие-либо иные обязанности);

$n_{П}$ – количество месяцев (недель, дней) в течение которых проектировщик был занят нашим проектом;

Расчет $К_{ПЕРС}$ с учетом работы одного разработчика и одного руководителя:

$$К_{ПЕРС} = (10\,000 * (1 + 0{,}4 + 0{,}3) * 1 * 1{,}25) + (19\,000 * (1 + 0{,}4 + 0{,}3) * 0{,}05 * 1{,}25) = 23\,268 \text{ руб.} \quad (4.3)$$

Средства вычислительной техники (СВТ) при выполнении проекта информационной системы в общем случае необходимы:

– для ввода и отладки прикладного программного обеспечения, разрабатываемого в рамках проекта;

– для комплексной отладки прикладного программного обеспечения, разрабатываемого в рамках проектах;

– для оформления документации по проекту (договоров, инструкций, спецификаций, пояснительных записок и т.п.);

– для имитации управляемых объектов в процессе отладки задач, связанных со взаимодействием с каким-либо оборудованием (расчетные узлы в торговом центре, электронные турникеты для учета времени работы сотрудников, технологическое оборудование в интегрированных АИС и т.п.).

Необходимые для выполнения проектирования СВТ взяты в аренду на период выполнения проекта – в этом случае арендодателю выплачивается арендная плата за все время аренды вне зависимости от степени реального использования техники (при этом неизбежные затраты по обслуживанию техники несет арендатор):

$$K_{CBT} = \sum_{j=1}^{k} T_{apj} \cdot (C_{apj} + C_{opj}) \qquad (4.5)$$

где:

T_{apj} – время аренды j-ого СВТ, обычно совпадающие с периодом выполнения проекта n (дней);

C_{apj} – арендная плата за j-е СВТ (руб./день);

C_{opj} – затраты на обслуживание и ремонт j-ого СВТ (руб./день);

k – общее число арендуемых СВТ.

В нашем случае было арендовано 5 виртуальных серверов с одинаковыми характеристиками на месяц каждый, при фиксированной плате за месяц равной 159 руб, при этом все затраты на обслуживание и ремонт взял на себя арендодатель, поэтому сумма аренды необходимых ресурсов может быть вычислена по формуле:

$$К_{СВТ} = 159 * 5 = 795 \text{ руб.} \qquad (4.6)$$

Затраты $К_{ИПС}$ на инструментальные программные средства отсутствуют, т.к. были использованы программные средства, доступные для бесплатного использования.

$К_{ПРОЧ}$ также равны нулю.

Таким образом, $К_{ПР}$ будет представлять из себя сумму $К_{ПЕРС}$ и $К_{СВТ}$:

$$К_{ПР} = К_{ПЕРС} + К_{СВТ} = 24\ 063 \text{ руб.} \quad (4.7)$$

При выполнении проекта по договору с заказчиком цена договора Цдог рассчитывается по формуле:

$$Ц_{ДОГ} = К_{ПР} \cdot (1+П) = 24\ 857 * (1 + 0{,}25) = 30\ 079 \text{ руб.} \quad (4.8)$$

где П – доля прибыли, закладываемая в договорную цену.

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы были выполнены все изначальное поставленные задачи:

- Разработана архитектура распределенной системы мониторинга, а также ее упрощенная схема (с возможностью масштабирования до первоначальной архитектуры), используемая при реализации системы;
- Обеспечена отказоустойчивость системы за счет технологий кластеризации и репликации данных;
- Реализованы программные модули системы (модуль мониторинга, модуль оповещения, модуль пользовательского интерфейса);
- Произведены отладка и тестирование, разработаны тестовые сценарии, которые были успешно выполнены в тестовой системе.

Таким образом, была достигнута цель работы – реализована распределенная система мониторинга сетевых сервисов, которая отвечает требованиям отказоустойчивости, а также иным требованиям, предъявляемым к распределенным системам в целом (прозрачность, масштабируемость, открытость и т.д.).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Б. Бёрнс. Распределенные системы. Паттерны проектирования. СПб.: Издательство «Питер», 2019. 222 с.

2. М. Клеппман. Высоконагруженные приложения. Программирование, масштабирование, поддержка. СПб.: Издательство «Питер», 2018. 628 с.

3. Лавров А.А. Методы и алгоритмы мониторинга вычислительных сетей на основе совместного анализа временных и функциональных характеристик стека протоколов TCP/IP: дисс. канд. техн. наук / СПбГЭТУ «ЛЭТИ», СПБ, 2013.

4. Мониторинг и администрирование в корпоративных вычислительных сетях: монография / С. А. Ивановский, А. А. Лавров, В. В. Яновский. СПб.: Издательство СПбГЭТУ «ЛЭТИ», 2013. 160 с.

5. Э. Таненбаум, М. ван Стеен. Распределенные системы. Принципы и парадигмы. СПб.: Издательство «Питер», 2003. 878 с.

6. А.А. Лавров, М.В. Гаськов. Архитектура отказоустойчивой распределенной системы мониторинга информационных ресурсов // Управление в современных системах: сборник трудов VIII Всероссийской научно-практической конференции научных, научно-педагогических работников и аспирантов, 2018. с. 277–283.

7. R. Aitchison. Pro DNS and BIND 10. Издательство «Apress», 2018. 679 с.

8. Schildt H. Java the complete reference eleventh edition. Издательство «Oracle Press», 2019. 1882 с.

9. Уоллс К. Spring в действии. М.: Издательство «ДМК Пресс», 2013. 752 с.

10. Документация и описание технологии BDR PostgreSQL // 2ndQuadrant PostgreSQL

.URL:https://www.2ndquadrant.com/en/resources/postgres-bdr-2ndquadrant (дата обращения 19.05.2019).

11. Java persistence with Hibernate second edition / C. Bauer, G. King, G. Gregory. Издательство «Manning Publications», 2017. 608 с.

12. B. Varanasi, S. Belida. Spring REST. Издательство «Apress», 2015. 195 с.

13. Объектно-ориентированный анализ и проектирование с примерами приложений, 3-е изд. / Буч Гредди, Максимчук Роберт А., Энгл Майкл У., Янг Бобби Дж., Коналлен Джим, Хьюстон Келли А.: Пер с англ. – М.: ООО «И.Д. Вильямс», 2010. 720 с.

14. Куликов С. Тестирование программного обеспечения. Базовый курс. 2-е издание. Спб.: Издательство «Epam Systems», 2019. 298 с.

15. Glenford J. Mayers, T. Budgett, C. Sandler. The art of software testing. Third edition. Нью-Джерси: Издательство «John Wiley & Sons, Inc», 2013. 240 с.

16. ОКПД-2 // Общероссийские классификаторы [Электронный ресурс]. URL: http://classifikators.ru/okpd (дата обращения: 27.05.2019).

ПРИЛОЖЕНИЕ А

Сценарии использования

Сценарии использования панели управления системой мониторинга:

Сценарий 1

Название прецедента: Вход в систему

Главная последовательность:

- Администратор переходит на сайт системы по ссылке http://domain.name;
- Пользователь попадает на страницу входа в систему, где от него требуется ввести логин и пароль;
- После ввода правильных данных администратор попадает на страницу с панелью управления.

Альтернативная последовательность:

- Администратор переходит на сайт системы по ссылке http://domain.name;
- Пользователь попадает на страницу входа в систему, где от него требуется ввести логин и пароль.
- При вводе неверных данных в форме для входа выводится сообщение об ошибке. Пользователь остается на странице входа и может совершить повторную попытку входа.

Сценарий 2

Название прецедента: Добавление целевого узла

Главная последовательность:

- На панели «Целевые узлы» панели управления пользователь вводит IP-адрес целевого узла;
- Пользователь нажимает кнопку добавить;

- В таблицу с целевыми узлами добавляется строка соответствующая добавленному целевому узлу с необходимой информацией о его состоянии.

Альтернативная последовательность:

- На панели «Целевые узлы» панели управления пользователь вводит IP-адрес целевого узла в формате недопустимом для IP-адреса;
- Пользователь нажимает кнопку добавить;
- Рядом с полем ввода появляется сообщение об ошибке, информирующее о некорректном формате IP-адреса.

Альтернативная последовательность:

- На панели «Целевые узлы» панели управления пользователь вводит IP-адрес целевого узла, уже имеющегося в таблице отслеживаемых узлов;
- Пользователь нажимает кнопку добавить;
- Рядом с полем ввода появляется сообщение об ошибке, информирующее о том, что добавляемый целевой узел уже есть в списке.

Альтернативная последовательность:

- На панели «Целевые узлы» панели управления пользователь вводит IP-адрес целевого узла, при условии, что все узлы мониторинга находятся в нерабочем состоянии или отсутствуют в системе;
- Пользователь нажимает кнопку добавить;
- Рядом с полем ввода появляется сообщение об ошибке, информирующее о том, что необходимо добавить рабочий узел в кластер мониторинга.

Сценарий 3

Название прецедента: Удаление целевого узла

Главная последовательность:

- На панели «Целевые узлы» панели управления пользователь нажимает на кнопку «Удалить», находящуюся в строке с информацией о целевом узле, который необходимо удалить;
- Целевой узел удаляется из списка на панели «Целевые узлы», также на панели «Узлы кластера» значение «Количество целевых узлов» узла кластера, за которым был закреплен удаляемый узел, уменьшается на единицу, а информация о целевом узле удаляется из списка опрашиваемых узлов данного узла кластера мониторинга.

Сценарий 4

Название прецедента: Просмотр списка опрашиваемых портов

Главная последовательность:

- На панели «Целевые узлы» панели управления пользователь кликает мышью на строку с информацией о целевом узле. Строка выделяется цветом отличным от остальных в таблице;
- В панели «Список портов» появляется информация об опрашиваемых портах выбранного целевого узла.

Сценарий 5

Название прецедента: Добавление опрашиваемого порта

Главная последовательность:

- Выполняется главная последовательность из прецедента «Просмотр списка опрашиваемых портов» при условии, что целевой узел не выбран;
- Пользователь вводит номер порта в соответствующее поле ввода на панели «Список портов»;
- Пользователь нажимает кнопку добавить;

- В таблицу со списком портов добавляется строка с необходимой информацией.

Альтернативная последовательность:

- Выполняется главная последовательность из прецедента «Просмотр списка опрашиваемых портов» при условии, что целевой узел не выбран;
- Пользователь вводит номер порта недопустимого формата в соответствующее поле ввода на панели «Список портов»;
- Пользователь нажимает кнопку добавить;
- Рядом с полем ввода появляется сообщение об ошибке, информирующее о том, что формат номера порта недопустим.

Альтернативная последовательность:

- Выполняется главная последовательность из прецедента «Просмотр списка опрашиваемых портов» при условии, что целевой узел не выбран;
- Пользователь вводит номер порта в соответствующее поле ввода на панели «Список портов», при условии, что порт с таким номером уже есть в списки опрашиваемых для данного целевого узла;
- Пользователь нажимает кнопку добавить;
- Рядом с полем ввода появляется сообщение об ошибке, информирующее о том, что порт с таким номер уже есть в списке опрашиваемых.

Альтернативная последовательность:

- Пользователь вводит номер порта в соответствующее поле ввода на панели «Список портов», при условии, что целевой узел не выбран;
- Пользователь нажимает кнопку добавить;
- Рядом с полем ввода появляется сообщение об ошибке, информирующее о том, что целевой узел не выбран.

Сценарий 6

Название прецедента: Удаление опрашиваемого порта

Главная последовательность:

- Выполняется главная последовательность из прецедента «Просмотр списка опрашиваемых портов» при условии, что целевой узел не выбран;
- На панели «Список портов» пользователь нажимает на кнопку «Удалить» в строке, соответствующей удаляемому порту целевого узла.

Сценарий 7

Название прецедента: Добавление узла в кластер мониторинга

Главная последовательность:

- На панели «Узлы кластера» пользователь вводит IP-адрес узла мониторинга в соответствующее поле ввода;
- Пользователь нажимает кнопку «Добавить»;
- В список узлов кластера мониторинга вносится добавленный узел.

Альтернативная последовательность:

- На панели «Узлы кластера» пользователь вводит IP-адрес узла мониторинга в соответствующее поле ввода, при условии, что узел с таким адресом уже имеется в списке;
- Пользователь нажимает кнопку «Добавить»;
- Рядом с полем ввода появляется сообщение об ошибке, информирующее о том, что узел с таким IP-адресом уже есть в списке.

Альтернативная последовательность:

- На панели «Узлы кластера» пользователь вводит IP-адрес узла мониторинга недопустимого формата в соответствующее поле ввода;
- Пользователь нажимает кнопку «Добавить»;

- Рядом с полем ввода появляется сообщение об ошибке, информирующее о том, что введенный IP-адрес имеет недопустимый формат.

Сценарий 8

Название прецедента: Просмотр информации об узле кластера мониторинга

Главная последовательность:

- На панели «Узлы кластера» пользователь нажимает на кнопку «Подробности» в строке с информацией об интересующем администратора узле мониторинга;
- На экране появляется всплывающее модальное окно со списком опрашиваемых целевых узлов, закрепленных за выбранным узлом кластера, и кнопкой удаления узла из кластера.

Сценарий 9

Название прецедента: Удаление узла из кластера мониторинга

Главная последовательность:

- Выполняется главная последовательность прецедента «Просмотр информации об узле кластера мониторинга»;
- Пользователь нажимает на кнопку «Удалить узел» во всплывающем модальном окне под списком целевых узлов выбранного узла кластера;
- Строка с информацией об удаленном узле кластера исчезает из списка на панели «Узлы кластера».

Сценарий 10

Название прецедента: Установка интервала мониторинга

Главная последовательность:

- Пользователь выставляет значение от 1 минуты до часа на слайдере вверху панели управления;

- Пользователь нажимает на кнопку «Установить», после чего в системе мониторинга устанавливается выбранный интервал между опросами узлов.

Сценарий 11

Название прецедента: Очистка истории событий

Главная последовательность:

- На панели «Лог событий» пользователь нажимает на кнопку «Очистить»;
- Данные таблицы «Лог событий» стираются.

Сценарий 12

Название прецедента: Выход из системы

Главная последовательность:

- Пользователь нажимает на кнопку «Выйти» в правом верхнем углу страницы с панелью системы;
- Пользователь попадает на страницу входа, на которой находится форма для ввода логина и пароля, при этом выведено сообщение о том, что пользователь вышел из системы.

Сценарии использования при настройке узлов системы мониторинга:

Сценарий 13

Название прецедента: Настройка узлов кластера мониторинга

Главная последовательность:

- Пользователь скачивает релиз на узел, который будет выполнять роль узла кластера мониторинга;
- Пользователь следует руководству по установке, указанному в описании к релизу кластера мониторинга;

Сценарий 14

Название прецедента: Настройка узлов кластера мониторинга

Главная последовательность:

- Администратор скачивает релиз для кластера БД и следует руководству по установке из описания релиза.

ПРИЛОЖЕНИЕ Б

Основные классы модулей кластера мониторинга и взаимодействие между ними.

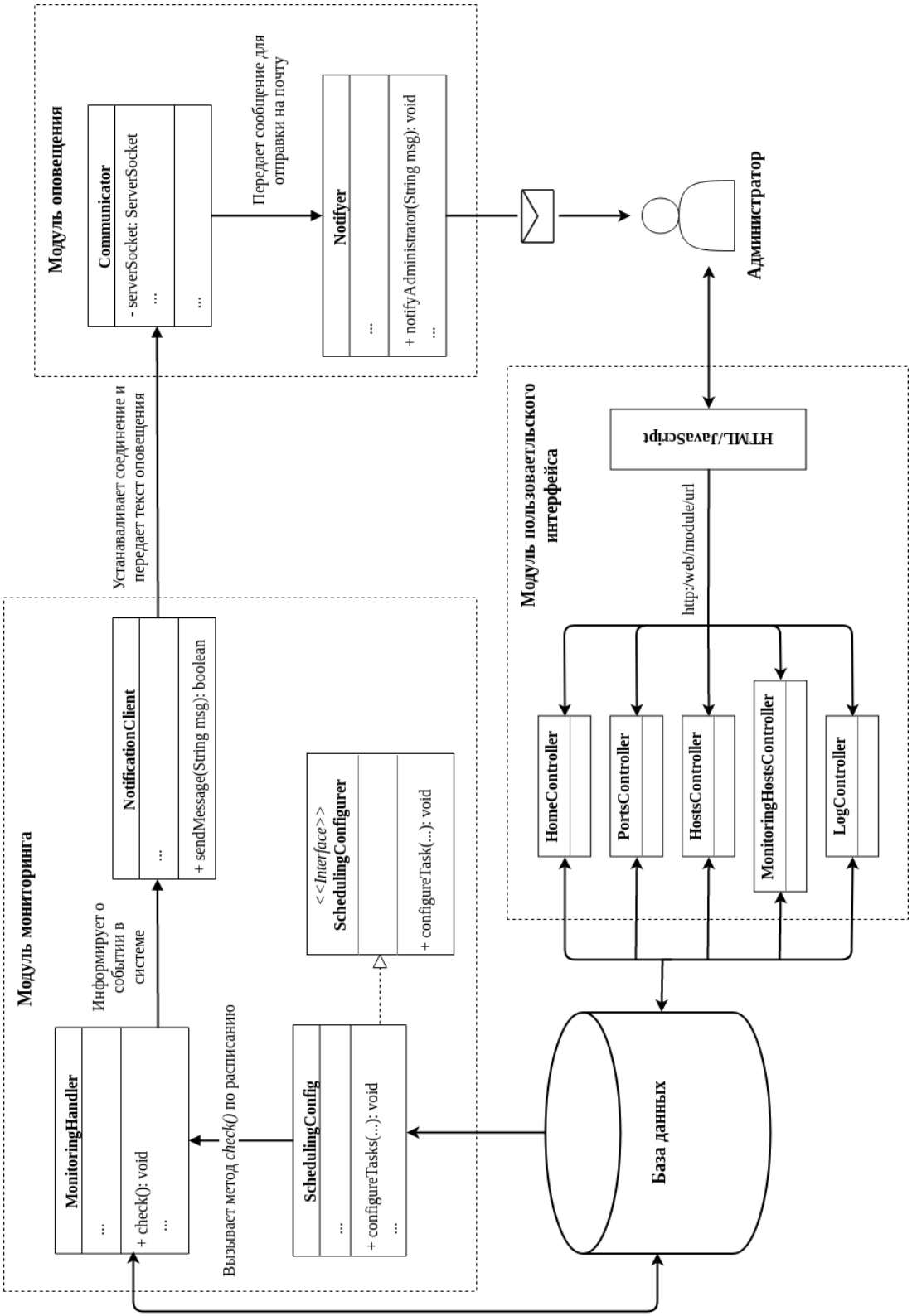

Рисунок Б.1 – Классы модулей кластера мониторинга

ПРИЛОЖЕНИЕ В

Триггерные функции PostgreSQL

createAddNewHostTrigger

```
CREATE OR REPLACE FUNCTION
        monitoring_schema.add_host_to_laziest_func()
        RETURNS trigger AS
$$
BEGIN
NEW.observer_id = (SELECT id FROM monitoring_schema.monitoring_hosts
                        WHERE NOT EXISTS (SELECT observer_id from mon-
itoring_schema.hosts
                                                        WHERE  observ-
er_id=monitoring_hosts.id) AND is_up <> FALSE
                    FETCH FIRST 1 ROWS ONLY);
IF NEW.observer_id IS NULL THEN
NEW.observer_id = (SELECT id FROM ( SELECT COUNT(*), monitor-
ing_hosts.id
                        FROM monitoring_schema.hosts INNER JOIN moni-
toring_schema.monitoring_hosts
                        ON hosts.observer_id = monitoring_hosts.id
                        GROUP BY monitoring_hosts.id
                        ORDER BY count) AS foo
                  FETCH FIRST 1 ROWS ONLY);
END IF;


RETURN NEW;
END;
$$
LANGUAGE 'plpgsql';

CREATE TRIGGER insertToHosts
  BEFORE INSERT
  ON monitoring_schema.hosts
  FOR EACH ROW
  EXECUTE PROCEDURE monitoring schema.add host to laziest func();
```

CreateNewMonitoringHostTrigger

```
CREATE OR REPLACE FUNCTION
        monitoring_schema.check_self_monitoring()
        RETURNS trigger AS
$$
DECLARE
randomHostId integer;
workingMHostsNumber integer;
hostWithNullAnotherMhIP integer;
BEGIN

      IF NEW.is_up = FALSE THEN
```

```
           IF EXISTS (SELECT * FROM monitoring_schema.monitoring_hosts
WHERE another_mh_ip_addr IS NULL AND is_up = TRUE) THEN
                hostWithNullAnotherMhIP = (SELECT id FROM monitor-
ing_schema.monitoring_hosts WHERE another_mh_ip_addr IS NULL AND is_up
= TRUE);
                UPDATE monitoring_schema.monitoring_hosts SET anoth-
er_mh_ip_addr = NEW.ip_addr WHERE id = hostWithNullAnotherMhIP;
           END IF;
     ELSE

           workingMHostsNumber := COUNT(*) FROM monitor-
ing_schema.monitoring_hosts WHERE is_up = TRUE AND id <> NEW.id;
              randomHostId := (SELECT id FROM monitor-
ing_schema.monitoring_hosts WHERE is_up = TRUE AND id <> NEW.id LIMIT
1);

           IF workingMHostsNumber > 1 THEN
                NEW.another_mh_ip_addr = (SELECT another_mh_ip_addr
FROM monitoring_schema.monitoring_hosts
                           WHERE id = randomHostId);

                     UPDATE monitoring_schema.monitoring_hosts SET an-
other_mh_ip_addr = NEW.ip_addr
                           WHERE id = randomHostId;
             ELSE
                  NEW.another_mh_ip_addr = (SELECT ip_addr FROM moni-
toring_schema.monitoring_hosts
                       WHERE id = randomHostId);
                     UPDATE monitoring_schema.monitoring_hosts SET an-
other_mh_ip_addr = NEW.ip_addr
                           WHERE id = randomHostId;
             END IF;

     END IF;


RETURN NEW;
END;
$$
LANGUAGE 'plpgsql';

CREATE TRIGGER insertToMonitoringHosts
  BEFORE INSERT
  ON monitoring_schema.monitoring_hosts
  FOR EACH ROW
  EXECUTE PROCEDURE monitoring_schema.check_self_monitoring();
```

createRedistributionTriggerOnMonitoringHostDelete

```
CREATE OR REPLACE FUNCTION
        monitoring_schema.redistribute_target_host_on_mHost_deletion
()
        RETURNS trigger AS
$$
```

```
DECLARE
mhID integer;
updChunkSize integer;
numberOfTargets integer;
numberOfMHostsLeft integer;

BEGIN

-- Redistribute targets

UPDATE monitoring_schema.hosts SET observer_id = NULL WHERE observ-
er_id = OLD.id;

numberOfTargets := COUNT(*) FROM monitoring_schema.hosts WHERE observ-
er_id IS NULL;
numberOfMHostsLeft := COUNT(*) FROM monitoring_schema.monitoring_hosts
WHERE id <> OLD.id AND is_up = TRUE;

     -- Reassign self monitoring
     IF numberOfMHostsLeft = 1 THEN
               UPDATE monitoring_schema.monitoring_hosts SET anoth-
er_mh_ip_addr = NULL
                         WHERE another_mh_ip_addr = OLD.ip_addr;
     ELSE
          UPDATE monitoring_schema.monitoring_hosts SET anoth-
er_mh_ip_addr = OLD.another_mh_ip_addr
               WHERE another_mh_ip_addr = OLD.ip_addr;
     END IF;
     ------------

IF numberOfMHostsLeft = 0 THEN
     RETURN OLD;
ELSE
     updChunkSize = numberOfTargets / numberOfMHostsLeft;

     FOR mhID IN (SELECT id FROM monitoring_schema.monitoring_hosts
          WHERE id <> OLD.id)
     LOOP
          UPDATE monitoring_schema.hosts SET observer_id = mhID WHERE
id IN
                    (SELECT id FROM monitoring_schema.hosts WHERE ob-
server_id IS NULL LIMIT updChunkSize);
     END LOOP;

-- update left hosts
     UPDATE monitoring_schema.hosts SET observer_id =
           (SELECT id FROM monitoring_schema.monitoring_hosts WHERE id
<> OLD.id AND is_up = TRUE LIMIT 1)
          WHERE observer_id IS NULL;

RETURN OLD;
END IF;
END;
$$
LANGUAGE 'plpgsql';
```

```
CREATE TRIGGER redistribute_targets_on_MH_delete_trigger
   BEFORE DELETE ON monitoring_schema.monitoring_hosts FOR EACH ROW
   EXECUTE PROCEDURE monitor-
ing_schema.redistribute_target_host_on_mHost_deletion();
```

createRedistributionTriggerOnMonitoringHostUpdate

```
CREATE OR REPLACE FUNCTION
        monitoring_schema.redistribute_target_host_on_mHost_update ()
        RETURNS trigger AS
$$
DECLARE
mhID integer;
busiestID integer;
updChunkSize integer;
randomHostId integer;
numberOfTargets integer;
busiestTargetNum integer;
numberOfMHostsLeft integer;
workingMHostsNumber integer;
BEGIN

IF (OLD.is_up = TRUE) AND (NEW.is_up = FALSE)
    THEN

--Redistributr targets
     UPDATE monitoring_schema.hosts SET observer_id = NULL WHERE ob-
server_id = OLD.id;

        numberOfTargets := COUNT(*) FROM monitoring_schema.hosts WHERE
observer_id IS NULL;
        numberOfMHostsLeft := COUNT(*) FROM monitor-
ing_schema.monitoring_hosts WHERE id <> OLD.id AND is_up = TRUE;

     -- Reassign self monitoring
        IF numberOfMHostsLeft = 1 THEN
                NEW.another_mh_ip_addr = NULL;
           UPDATE monitoring_schema.monitoring_hosts SET anoth-
er_mh_ip_addr = NULL
                        WHERE another_mh_ip_addr = OLD.ip_addr;
        ELSE
           NEW.another_mh_ip_addr = NULL;
           UPDATE monitoring_schema.monitoring_hosts SET anoth-
er_mh_ip_addr = OLD.another_mh_ip_addr
                WHERE another_mh_ip_addr = OLD.ip_addr;
        END IF;
        ------------




        IF numberOfMHostsLeft = 0 THEN
```

```
        RETURN NEW;
        ELSE

        updChunkSize = numberOfTargets / numberOfMHostsLeft;

        FOR mhID IN (SELECT id FROM monitoring_schema.monitoring_hosts
            WHERE id <> OLD.id)
        LOOP
            UPDATE monitoring_schema.hosts SET observer_id = mhID
WHERE id IN
                (SELECT id FROM monitoring_schema.hosts WHERE observ-
er_id IS NULL LIMIT updChunkSize);
        END LOOP;

        -- update left hosts
        UPDATE monitoring_schema.hosts SET observer_id =
                (SELECT id FROM monitoring_schema.monitoring_hosts
WHERE id <> OLD.id AND is_up = TRUE LIMIT 1)
                WHERE observer_id IS NULL;
        END IF;


ELSE IF (OLD.is_up = FALSE) AND (NEW.is_up = TRUE)
    THEN


--Reassign self monitoring

     workingMHostsNumber := COUNT(*) FROM monitor-
ing_schema.monitoring_hosts WHERE is_up = TRUE AND id <> NEW.id;
     randomHostId := (SELECT id FROM monitor-
ing_schema.monitoring_hosts WHERE is_up = TRUE AND id <> NEW.id LIMIT
1);

     IF workingMHostsNumber > 1 THEN
          NEW.another_mh_ip_addr = (SELECT another_mh_ip_addr FROM
monitoring_schema.monitoring_hosts
                WHERE id = randomHostId);

          UPDATE monitoring_schema.monitoring_hosts SET anoth-
er_mh_ip_addr = NEW.ip_addr
                WHERE id = randomHostId;
     ELSE
          NEW.another_mh_ip_addr = (SELECT ip_addr FROM monitor-
ing_schema.monitoring_hosts
                WHERE id = randomHostId);
          UPDATE monitoring_schema.monitoring_hosts SET anoth-
er_mh_ip_addr = NEW.ip_addr
                        WHERE id = randomHostId;
     END IF;

--Redistribute targets

        busiestID := (SELECT id FROM ( SELECT COUNT(*), monitor-
ing_schema.monitoring_hosts.id
                        FROM monitoring_schema.hosts INNER JOIN moni-
```

```
toring_schema.monitoring_hosts
                        ON hosts.observer_id = monitoring_hosts.id
                        GROUP BY monitoring_hosts.id
                        ORDER BY count DESC) AS foo
                  FETCH FIRST 1 ROWS ONLY);

        busiestTargetNum := COUNT(*) FROM monitoring_schema.hosts
WHERE observer_id = busiestID;

        IF busiestTargetNum > 1 THEN
            UPDATE monitoring_schema.hosts SET observer_id = NEW.id
WHERE id IN
                (SELECT id FROM monitoring_schema.hosts WHERE observ-
er_id = busiestID LIMIT (busiestTargetNum/2));
        END IF;
    END IF;
END IF;
RETURN NEW;
END;
$$
LANGUAGE 'plpgsql';

CREATE TRIGGER redistribute_targets_on_MH_update_trigger
   BEFORE UPDATE
   ON monitoring_schema.monitoring_hosts
   FOR EACH ROW
   EXECUTE PROCEDURE monitor-
ing_schema.redistribute_target_host_on_mHost_update();
```

ПРИЛОЖЕНИЕ Г

Исходный код программной реализации модуля мониторинга

SchedulingConfig.java

```
package edu.max.monsys.configuration;

import edu.max.monsys.monitoring.MonitoringHandler;
import edu.max.monsys.repository.ConfigRepository;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.scheduling.annotation.EnableScheduling;
import org.springframework.scheduling.annotation.SchedulingConfigurer;
import org.springframework.scheduling.config.ScheduledTaskRegistrar;

import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.concurrent.Executor;
import java.util.concurrent.Executors;

@Configuration
@EnableScheduling
public class SchedulingConfig implements SchedulingConfigurer {

    private final ConfigRepository configRepository;

    public SchedulingConfig(ConfigRepository configRepository) {
        this.configRepository = configRepository;
    }

    @Bean
    public MonitoringHandler monitoringHandler() {
        return new MonitoringHandler();
    }

    @Bean
    public Executor taskExecutor() {
        return Executors.newScheduledThreadPool(100);
    }

    @Override
    public void configureTasks(ScheduledTaskRegistrar taskRegistrar) {
        taskRegistrar.setScheduler(taskExecutor());
        taskRegistrar.addTriggerTask(
                () -> monitoringHandler().check(),
                triggerContext -> {
                    Calendar nextExecutionTime =  new GregorianCalen-
dar();
                    Date lastActualExecutionTime = triggerCon-
text.lastActualExecutionTime();
                    nextExecutionTime.setTime(lastActualExecutionTime
!= null ? lastActualExecutionTime : new Date());
                    if (configRepository.count() == 0)
```

```
                        nextExecutionTime.add(Calendar.MILLISECOND,
15000);
                    else
                        nextExecutionTime.add(Calendar.MILLISECOND,
                                configReposito-
ry.findById(0).get().getRateSeconds() * 1000);
                    return nextExecutionTime.getTime();
                }
        );
    }
}
```

ConfigMonitoringRate.java

```
package edu.max.entity;

import lombok.Getter;
import lombok.Setter;

import javax.persistence.*;

@Entity
@Getter
@Setter
@Table(name = "rate_config", schema = "config_and_log_schema")
public class ConfigMonitoringRate {

    @Id
    @Column(name = "id", columnDefinition = "integer default 0", up-
datable = false, unique = true)
    private Integer id;

    @Column(name = "rate", nullable = false, columnDefinition = "inte-
ger default 60")
    private int rateSeconds;

    public ConfigMonitoringRate() {
       id= 0;
       rateSeconds = 60;
    }
}
```

Host.java

```
package edu.max.entity;

import lombok.AccessLevel;
import lombok.Getter;
```

```
import lombok.NoArgsConstructor;
import lombok.Setter;

import javax.persistence.*;
import java.util.HashSet;
import java.util.Set;

@Entity
@NoArgsConstructor
@Getter
@Setter
@Table(name = "hosts", schema = "monitoring_schema")
@org.hibernate.annotations.Entity(dynamicInsert = true)
public class Host {

    @Id
    @GeneratedValue
    @Setter(AccessLevel.NONE)
    private Integer id;

    @Column(name = "ip_addr", nullable = false, unique = true)
    private String ipAddress;

    @Column(name = "domain", columnDefinition = "varchar(255) default
'PSPµPëP·PIPµCЃC,PSPs'")
    private String domainName;

    @OneToMany(cascade = CascadeType.ALL, orphanRemoval = true, fetch
= FetchType.EAGER, mappedBy = "host")
    private Set<Port> ports = new HashSet<>();


    public Host(String ipAddress) {
        this.ipAddress = ipAddress;
    }


    public void addPort(Port port) {
        this.ports.add(port);
        port.setHost(this);
    }

    @Override
    public String toString() {
        return "Host{" +
                "id=" + id +
                ", ipAddress='" + ipAddress + '\'' +
                ", domainName='" + domainName + '\'' +
                '}';
    }
}
```

Log.java

```
package edu.max.entity;

import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

import javax.persistence.*;

@Entity
@NoArgsConstructor
@Getter
@Setter
@Table(name = "logs", schema = "config_and_log_schema")
public class Log {

    @Id
    @GeneratedValue
    private Integer id;

    private String date;

    private String host;

    private int port;

    private String event;

    public Log(String date, String host, int port, String event) {
        this.date = date;
        this.host = host;
        this.port = port;
        this.event = event;
    }

}
```

MonitoringHost.java

```
package edu.max.entity;

import lombok.AccessLevel;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

import javax.persistence.*;
import java.util.HashSet;
import java.util.Set;

@Entity
@NoArgsConstructor
@Getter
```

```
@Setter
@Table(name = "monitoring_hosts", schema = "monitoring_schema")
public class MonitoringHost {

    @Id
    @GeneratedValue
    @Setter(AccessLevel.NONE)
    Integer id;

    @Column(name = "ip_addr", nullable = false, unique = true)
    private String ipAddress;

    @Column(name = "another_mh_ip_addr")
    private String anotherMHIpAdress;

    @Column(name = "is_up", columnDefinition = "boolean default
false")
    private boolean up;

    @OneToMany
    @JoinColumn(name = "observer_id")
    private Set<Host> targets = new HashSet<>();

    public MonitoringHost(String ipAddress) {
        this.ipAddress = ipAddress;
    }


}
```

Port.java

```
package edu.max.entity;

import com.fasterxml.jackson.annotation.JsonIgnore;
import lombok.AccessLevel;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

import javax.persistence.*;

@Entity
@NoArgsConstructor
@Getter
@Setter
@Table(name = "ports", schema = "monitoring_schema")
@org.hibernate.annotations.Entity(dynamicInsert = true)
public class Port {

    @Id
    @GeneratedValue
    @Setter(AccessLevel.NONE)
    private Integer id;
```

```
    @Column(nullable = false)
    private int number;
    @Column(name = "port_is_up")
    private boolean up;
    @Column(name = "service", columnDefinition = "varchar(255) default
'PSPµPëP·PIPµCЃC,PµPS'")
    private String service;
    @ManyToOne
    @JoinColumn(name="host_id")
    @JsonIgnore
    private Host host;

    public Port(int number) {
        this.number = number;
    }

    @Override
    public String toString() {
        return "Port{" +
                "id=" + id +
                ", portNum=" + number +
                ", up=" + up +
                ", service='" + service + '\'' +
                '}';
    }
}
```

ConfigRepository.java

```
package edu.max.repository;


import edu.max.entity.ConfigMonitoringRate;
import org.springframework.data.jpa.repository.JpaRepository;

public interface ConfigRepository extends JpaReposito-
ry<ConfigMonitoringRate, Integer> {
}
```

HostRepository.java

```
package edu.max.repository;


import edu.max.entity.Host;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.Optional;

public interface HostRepository extends JpaRepository<Host, Integer> {
    Optional<Host> findById(Integer id);
    Optional<Host> findHostByIpAddress(String ip);}
```

LogRepository.java

```
package edu.max.repository;

import edu.max.entity.Log;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.Optional;

public interface LogRepository extends JpaRepository<Log, Integer> {

    Optional<Log> findLogByHostAndPort(String host, int port);
}
```

MonitoringHostRepository.java

```
package edu.max.repository;

import edu.max.entity.MonitoringHost;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.Optional;

public interface MonitoringHostRepository extends JpaReposito-
ry<MonitoringHost, Integer> {
    Optional<MonitoringHost> findMonitoringHostByIpAddress(String ip);
    Optional<MonitoringHost> findMonitoringHostByAnotherMHIpA-
dress(String ip);

}
```

PortRepository

```
package edu.max.repository;

import edu.max.entity.Port;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.Optional;

public interface PortRepository extends JpaRepository<Port, Integer> {

    Optional<Port> findPortByNumber(int num);//and host id
}
```

MonitoringHandler.java

```
package edu.max.monitoring;


import edu.max.entity.Host;
import edu.max.entity.Log;
import edu.max.entity.MonitoringHost;
import edu.max.entity.Port;
import edu.max.repository.HostRepository;
import edu.max.repository.LogRepository;
import edu.max.repository.MonitoringHostRepository;
import org.apache.commons.net.ftp.FTPClient;
import org.apache.commons.net.ftp.FTPCmd;
import org.apache.commons.net.ftp.FTPReply;
import org.apache.commons.net.pop3.POP3Client;
import org.apache.commons.net.smtp.SMTPClient;
import org.apache.commons.net.smtp.SMTPReply;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;

import javax.transaction.Transactional;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.*;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Optional;
import java.util.TimeZone;


public class MonitoringHandler {

    private final int CONNECTION_TIMEOUT = 1000; //ms

    @Autowired
    private HostRepository hostRepository;

    @Autowired
    private MonitoringHostRepository monitoringHostRepository;

    @Autowired
    private LogRepository logRepository;

    @Autowired
    private NotificationClient notificationClient;


    @Value("${server.address}")
    private String myIP;

    @Value("${server.port}")
    private Integer sysPort;
```

```
    @Transactional
    public void check() {

        if (monitoringHostReposito-
ry.findMonitoringHostByIpAddress(myIP).isPresent()) {
            System.out.println("check");
            monitoringHostReposito-
ry.findMonitoringHostByIpAddress(myIP).get().setUp(true);
            String targetIP = monitoringHostReposito-
ry.findMonitoringHostByIpAddress(myIP).get().getAnotherMHIpAdress();
            Optional<MonitoringHost> targetMH = monitoringHostReposi-
tory.findMonitoringHostByIpAddress(targetIP);

            if (targetIP != null && httpPortCheck(targetIP, sysPort)
&& targetMH.isPresent()) {
                if (!targetMH.get().isUp()) {
                    Date date = new Date();
                    DateFormat df = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss");

df.setTimeZone(TimeZone.getTimeZone("Europe/Moscow"));
                    logRepository.save(
                            new Log(df.format(date), tar-
getMH.get().getIpAddress(), sysPort, "PґPsCЃC,CґPїPµPS"));
                    logRepository.flush();
                }
                targetMH.get().setUp(true);
            }
            else if (targetIP != null && targetMH.isPresent()) {
                if (targetMH.get().isUp()) {
                    Date date = new Date();
                    DateFormat df = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss");

df.setTimeZone(TimeZone.getTimeZone("Europe/Moscow"));
                    logRepository.save(
                            new Log(df.format(date), tar-
getMH.get().getIpAddress(), sysPort, "PSPµPґPsCЃC,CґPїPµPS"));
                    logRepository.flush();
                    notificationClient.sendMessage("P’PSPëPjP°PSPëPµ,
PSP° PICЂPµPjCЏ " + df.format(date)
                            + " CґP·PµP» PєP»P°CЃC,CЂPµP°
PjPsPSPëC,PsCЂPëPSPiP°  "
                            + targetMH.get().getIpAddress()
                            + "PїPµCЂPµCЃC,P°P» P±C‹C,CЊ
PґPsCЃC,CґPїPSC‹Pj.");
                }
                targetMH.get().setUp(false);
            }

        } else {
            System.out.println("no");
            return;
        }
```

```
        for (Host host : monitoringHostReposito-
ry.findMonitoringHostByIpAddress(myIP).get().getTargets()) {



            for (Port port : host.getPorts()) {

                if (ftpPortCheck(host.getIpAddress(),
port.getNumber())) {
                    logPortIsUp(host, port);
                    port.setService("FTP");
                } else if (httpPortCheck(host.getIpAddress(),
port.getNumber())) {
                    logPortIsUp(host, port);
                    port.setService("HTTP");
                    System.out.println("http check");
                } else if (smtpPortCheck(host.getIpAddress(),
port.getNumber())) {
                    logPortIsUp(host, port);
                    port.setService("SMTP");
                } else if (pop3PortCheck(host.getIpAddress(),
port.getNumber())) {
                    logPortIsUp(host, port);
                    port.setService("POP3");
                } else if (sshCheck(host.getIpAddress(),
port.getNumber())) {
                    logPortIsUp(host, port);
                    port.setService("SSH");
                } else {
                    System.out.println("port doesnt work");
                    if (port.isUp()) {
                        Date date = new Date();
                        DateFormat df = new SimpleDateFormat("yyyy-MM-
dd HH:mm:ss");

df.setTimeZone(TimeZone.getTimeZone("Europe/Moscow"));

                        logRepository.save(
                                new Log(df.format(date),
host.getIpAddress(), port.getNumber(), "PSPµPrPsCЃC,CґPїPµPS"));
                        logRepository.flush();


notificationClient.sendMessage("P’PSPëPjP°PSPëPµ, PSP° PICЂPµPjCЏ " +
df.format(date)
                                + " PЎPµCЂPIPëCЃ,
CЂP°P±PsC,P°CЂC‰PëP№ PїPs PїCЂPsC,PsPєPsP»Cґ "
                                + port.getService()
                                + " PSP° PїPsCЂC,Cґ "
                                + host.getIpAddress() + ":"
                                + port.getNumber()
                                + " PїPµCЂPµCЃC,P°P» P±C‹C,CЊ
PrPsCЃC,CґPїPSC‹Pj.");
                    }
                    port.setUp(false);
```

```
                }

            }

        }

        this.hostRepository.flush();
    }


    private boolean ftpPortCheck(String hostname, int port) {


        InetAddress host;
        try {
            host = InetAddress.getByName(hostname);
        } catch (UnknownHostException e) {
            return false;
        }

        FTPClient ftp = new FTPClient();
        try {
            ftp.setConnectTimeout(CONNECTION_TIMEOUT);
            ftp.connect(host, port);
            int reply = ftp.getReplyCode();

            if (!FTPReply.isPositiveCompletion(reply)) {
                return false;
            } else {
                ftp.sendCommand(FTPCmd.USER, "user");
                return ftp.getReplyCode() == 331 || ftp.getReplyCode()
== 332;
            }
        } catch (Exception e) {
            return false;
        } finally {
            try {
                ftp.disconnect();
            } catch (Exception e) {
                return false;
            }
        }
    }

    private boolean httpPortCheck(String hostname, int port) {

        HttpURLConnection con = null;
        try {
            URL url = new URL("http://" + hostname + ":" + port);
            con = (HttpURLConnection) url.openConnection();
            con.setRequestMethod("GET");
            con.setConnectTimeout(CONNECTION_TIMEOUT);
            con.setRequestProperty("User-Agent", "Mozilla 5.0 (Win-
dows; U; "
                    + "Windows NT 5.1; en-US; rv:1.8.0.11) ");
            int status = con.getResponseCode();
```

```
            return status != -1;

        } catch (Exception e) {
            return false;
        } finally {
            if (con != null)
                con.disconnect();
        }
    }

    private boolean smtpPortCheck(String hostname, int port) {

        InetAddress host;
        try {
            host = InetAddress.getByName(hostname);
        } catch (UnknownHostException e) {
            return false;
        }

        SMTPClient smtp = new SMTPClient();
        try {
            smtp.setConnectTimeout(CONNECTION_TIMEOUT);
            smtp.connect(host, port);
            int reply = smtp.getReplyCode();

            if (!SMTPReply.isPositiveCompletion(reply)) {
                return false;
            } else {
                smtp.sendCommand("HELO ME");
                return smtp.getReplyCode() == 250;
            }
        } catch (Exception e) {
            return false;
        } finally {
            try {
                smtp.disconnect();
            } catch (Exception e) {
                return false;
            }
        }

    }

    private boolean pop3PortCheck(String hostname, int port) {

        InetAddress host;
        try {
            host = InetAddress.getByName(hostname);
        } catch (UnknownHostException e) {
            return false;
        }

        POP3Client pop3 = new POP3Client();
        try {
            pop3.setConnectTimeout(CONNECTION_TIMEOUT);
            pop3.connect(host, port);
```

```
            return true;

        } catch (Exception e) {
            return false;
        } finally {
            try {
                pop3.disconnect();
            } catch (IOException e) {
                return false;
            }
        }

    }

    private boolean sshCheck(String hostname, int port) {

        Socket socket = null;
        try {
            socket = new Socket();
            socket.connect(new InetSocketAddress(hostname,port), CON-
NECTION_TIMEOUT);
            InputStreamReader streamReader= new InputStreamRead-
er(socket.getInputStream());
            BufferedReader reader= new BufferedReader(streamReader);

            String greeting= reader.readLine();

            boolean isSSH;
            isSSH = greeting.contains("SSH");

            reader.close();

            return isSSH;

        } catch(Exception e) {
            return false;
        }finally{
            if(socket != null){
                try {
                    socket.close();
                }catch(IOException e){
                    return false;
                }
            }
        }

    }

    private void logPortIsUp (Host host, Port port) {
        if (!port.isUp()) {
            Date date = new Date();
            DateFormat df = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss");
            df.setTimeZone(TimeZone.getTimeZone("Europe/Moscow"));
            logRepository.save(
```

```
                    new Log(df.format(date), host.getIpAddress(),
port.getNumber(), "PґPsCЃC,CґPїPµPS"));
            logRepository.flush();
        }
        port.setUp(true);
    }

}
```

NotificationClient.java

```
package edu.max.monitoring;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;


@Component
public class NotificationClient {


    @Value("${notification.module.domain}")
    private String notificationModuleDomain;

    @Value("${notification.module.port}")
    private int notificationModulePort;

    public boolean sendMessage(String msg) {

        try {

            Socket clientSocket = new Socket(notificationModuleDomain,
notificationModulePort);
            PrintWriter out = new PrintWrit-
er(clientSocket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(new In-
putStreamReader(clientSocket.getInputStream()));
            out.println(msg);
            String resp = in.readLine();

            in.close();
            out.close();
            clientSocket.close();
        }
        catch (IOException e) {
            e.printStackTrace();
            return false;
        }
```

```
        return true;
    }
}
```

MonitoringModuleApplication.java

```
package edu.max;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import
org.springframework.data.jpa.repository.config.EnableJpaRepositories;
import org.springframework.scheduling.annotation.EnableScheduling;
import
org.springframework.transaction.annotation.EnableTransactionManagement
;

@SpringBootApplication
@EnableScheduling
@EnableJpaRepositories
@EnableTransactionManagement
public class MonitoringModuleApplication {

    public static void main(String[] args) {
        SpringApplication.run(MonitoringModuleApplication.class,
args);
    }

}
```

ПРИЛОЖЕНИЕ Д

Исходный код программной реализации модуля оповещения

Communicator.java

```
package edu.max;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;

public class Communicator {

    private ServerSocket serverSocket;

    private void start(int port) throws IOException {
        serverSocket = new ServerSocket(port);
        Notifyer notifyer = new Notifyer();
        while (true)
            new CommunicatorClientHandler(serverSocket.accept(), noti-
fyer).start();
    }

    public void stop() throws IOException {
        serverSocket.close();
    }

    private static class CommunicatorClientHandler extends Thread {

        private Socket clientSocket;
        private PrintWriter out;
        private BufferedReader in;
        private Notifyer notifyer;

        public CommunicatorClientHandler(Socket socket, Notifyer noti-
fyer) {
            this.clientSocket = socket;
            this.notifyer = notifyer;
        }

        public void run()  {

            try {
                out = new PrintWriter(clientSocket.getOutputStream(),
true);
                in = new BufferedReader(
                        new InputStreamRead-
er(clientSocket.getInputStream()));

                String greeting = in.readLine();
```

```
                if ("monsys n-module hello".equals(greeting))
                    out.println("hello monsys m-module");
                else {
                    out.println("unrecognised greeting");
                    in.close();
                    out.close();
                    clientSocket.close();
                }

                String event = in.readLine();
                notifyer.notifyAdministrator(event);
                in.close();
                out.close();
                clientSocket.close();
            } catch (IOException e) {
                e.printStackTrace();
            }

        }
    }


    public static void main(String[] args) throws IOException {


        Communicator server = new Communicator();
        server.start(6666);
        server.stop();

    }
}
```

Notifyer.java

```
package edu.max;

import javax.mail.*;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;
import java.util.Properties;

public class Notifyer {


    public void notifyAdministrator(String eventMessage) {


        try(InputStream input = new FileIn-
putStream(this.getFileFromResources("mail.properties"))) {
```

```
            Properties props = new Properties();

            props.load(input);

            String reciever = props.getProperty("reciever");
            final String password =
props.getProperty("sender_password");
            final String user = props.getProperty("sender");


            Session session = Session.getDefaultInstance(props,
                    new javax.mail.Authenticator() {
                        protected PasswordAuthentication getPass-
wordAuthentication() {
                            return new PasswordAuthentication(user,
password);
                        }
                    });

            try {
                MimeMessage message = new MimeMessage(session);
                message.setFrom(new InternetAddress(user));
                message.addRecipient(Message.RecipientType.TO, new In-
ternetAddress(reciever));
                message.setSubject("PµCЂPsPëP·PsC€P»Ps CЃPsP±C‹C,PëPµ
PI CЃPëCЃC,PµPjPµ PjPsPSPëC,PsCЂPëPSPiP°!");
                message.setText(eventMessage);

                Transport.send(message);

            } catch (MessagingException e) {
                e.printStackTrace();
            }


        } catch(
        IOException ex)

        {
            ex.printStackTrace();
        }

    }

    private File getFileFromResources(String fileName) {

        ClassLoader classLoader = getClass().getClassLoader();

        URL resource = classLoader.getResource(fileName);
        if (resource == null) {
            throw new IllegalArgumentException("file is not found!");
        } else {
            return new File(resource.getFile());
        }
```

```
        }
}
```

ПРИЛОЖЕНИЕ Е

Исходный код программной реализации модуля пользовательского интерфейса

WebSecurityConfig.java

```
package edu.max.configuration;

import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.web.builders.HttpSecuri
ty;
import
org.springframework.security.config.annotation.web.configuration.Enabl
eWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSe
curityConfigurerAdapter;

@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
                .authorizeRequests()
                    //.antMatchers("/", "/home").permitAll()
                    .antMatchers("/resources/**", "/webjars/**",
"/css/**").permitAll()
                    .anyRequest().authenticated()
                        .and()
                .formLogin()
                    .loginPage("/login")
                    .permitAll()
                        .and()
                .logout()
                    .permitAll()
                        .and()
                .csrf().disable();
    }

}
```

HomeController.java

```
package edu.max.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class HomeController {
```

```
    @GetMapping("/")
    public String getHomePage() {
        return "home";
    }

    @GetMapping("/login")
    public String getLogInPage() {
        return "login";
    }
}
```

HostsController.java

```
package edu.max.controller;

import edu.max.entity.Host;
import edu.max.entity.Port;
import edu.max.repository.HostRepository;
import edu.max.repository.MonitoringHostRepository;
import edu.max.repository.PortRepository;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.net.InetAddress;
import java.net.UnknownHostException;

@RestController
@RequestMapping("/hosts")
class HostsController {

    private final HostRepository hostRepository;

    private final PortRepository portRepository;

    private final MonitoringHostRepository monitoringHostRepository;

    public HostsController(HostRepository hostRepository, PortReposi-
tory portRepository, MonitoringHostRepository monitoringHostReposito-
ry) {
        this.hostRepository = hostRepository;
        this.portRepository = portRepository;
        this.monitoringHostRepository = monitoringHostRepository;
    }

    @GetMapping
    public Iterable<Host> findAll() {
        return hostRepository.findAll();
    }

    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public ResponseEntity<String> create(@RequestParam String ip) {
```

```
        if (!ip.matches(
                "^([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\." +
                    "([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\." +
                    "([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\." +
                    "([01]?\\d\\d?|2[0-4]\\d|25[0-5])$"))
                return new ResponseEntity<>("РќРµРІРµСЂРЅС‹Р№
С„РѕСЂРјР°С‚ IP-Р°РґСЂРµСЃР°", HttpStatus.BAD_REQUEST);
        else if (hostRepository.findHostByIpAddress(ip).isPresent())
            return new ResponseEntity<>("РЈР·РµР» СѓР¶Рµ РµСЃС‚СЊ РІ
СѓРїРёСЃРєРµ", HttpStatus.BAD_REQUEST);


        if (monitoringHostRepository.count() == 0)
            return new ResponseEntity<>("РЎРїРёСЃРѕРє РґРѕР±Р°РІР»РµРЅРЅС‹С…,Рµ
СѓР·Р»РµР» РјРѕРЅРёС‚РѕСЂРёРЅРіР°", HttpStatus.BAD_REQUEST);
        else {

            Host h = new Host(ip);

            InetAddress addr = null;
            try {
                addr = InetAddress.getByName(ip);
            } catch (UnknownHostException e) {
                e.printStackTrace();
            }
            if (addr != null) {
                String hostname = addr.getCanonicalHostName();
                h.setDomainName(hostname);
            }

            hostRepository.save(h);
        }

        return ResponseEntity.ok("");
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<?> delete(@PathVariable("id") Integer id) {
        for (Port p : hostRepository.findById(id).get().getPorts())
            portRepository.deleteById(p.getId());
        hostRepository.deleteById(id);
        return ResponseEntity.noContent().build();
    }

}
```

LogController.java

```
package edu.max.controller;

import edu.max.entity.Log;
import edu.max.repository.LogRepository;
import org.springframework.http.ResponseEntity;
```

```
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/logs")
public class LogController {

    private final LogRepository logRepository;

    public LogController(LogRepository logRepository) {
        this.logRepository = logRepository;
    }

    @GetMapping
    public Iterable<Log> getLogs() {
        return logRepository.findAll();
    }

    @DeleteMapping
    public ResponseEntity<?> delete() {

        logRepository.deleteAll();

        return ResponseEntity.ok("");
    }
}
```

MonitoringHostsController.java

```
package edu.max.controller;

import edu.max.entity.ConfigMonitoringRate;
import edu.max.entity.MonitoringHost;
import edu.max.repository.ConfigRepository;
import edu.max.repository.MonitoringHostRepository;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.Optional;

@RestController
@RequestMapping("/hosts/monitoring")
public class MonitoringHostsController {


    private final ConfigRepository configRepository;

    private final MonitoringHostRepository monitoringHostRepository;

    public MonitoringHostsController(ConfigRepository configReposito-
ry, MonitoringHostRepository monitoringHostRepository) {
        this.configRepository = configRepository;
        this.monitoringHostRepository = monitoringHostRepository;
    }
```

```
    @GetMapping
    public Iterable<MonitoringHost> findAll() {
        return monitoringHostRepository.findAll();
    }

    @GetMapping(value = "/{id}")
    public Optional<MonitoringHost> findById(@PathVariable("id") Inte-
ger id) {
        return monitoringHostRepository.findById(id);
    }

    @PostMapping
    public ResponseEntity<String> create(@RequestParam String ip) {

        if (!ip.matches(
                "^([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\." +
                        "([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\." +
                        "([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\." +
                        "([01]?\\d\\d?|2[0-4]\\d|25[0-5])$"))
            return new ResponseEntity<>("РќРµРІРµСЂРЅС‹Р№
С„РѕСЂРјР°С, IP-Р°РґСЂРµСЃР°", HttpStatus.BAD_REQUEST);
        else if (monitoringHostReposito-
ry.findMonitoringHostByIpAddress(ip).isPresent())
            return new ResponseEntity<>("РЈР·РµР» СѓР¶Рµ РµСЃС‚СЊ РІ
СЃРїРёСЃРєРµ", HttpStatus.BAD_REQUEST);

        MonitoringHost addedMHost = new MonitoringHost(ip);

        monitoringHostRepository.save(addedMHost);


        monitoringHostRepository.flush();

        return ResponseEntity.ok("");
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<?> delete(@PathVariable("id") Integer id) {
        if (monitoringHostRepository.count() == 1)
            return new ResponseEntity<>("РЎРїРµСЂРІР° РґРѕР±Р°РІСЊС,Рµ
РґСЂСѓРіРёРѕР№ С…РѕСЃС, РјРѕРЅРёС‚РѕСЂР°",
HttpStatus.BAD_REQUEST);
        else {
            monitoringHostRepository.deleteById(id);
        }
        return ResponseEntity.ok("");
    }

    @PostMapping("/rate")
    public void setMonitoringRate(@RequestParam int rateVal) {

        if (configRepository.count() == 0)
            configRepository.save(new ConfigMonitoringRate());
```

```
        else  {
            Optional<ConfigMonitoringRate> cfmr = configReposito-
ry.findById(0);
            if (cfmr.isPresent()) {
                cfmr.get().setRateSeconds(rateVal * 60);
                configRepository.flush();
            }
        }
    }
}
```

PortsController.java

```
package edu.max.controller;

import edu.max.entity.Host;
import edu.max.entity.Port;
import edu.max.repository.HostRepository;
import edu.max.repository.PortRepository;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.Optional;

@RestController
@RequestMapping("/ports")
class PortsController {

    private final PortRepository portRepository;
    private final HostRepository hostRepository;

    public PortsController(PortRepository portRepository, HostReposi-
tory hostRepository) {
        this.portRepository = portRepository;
        this.hostRepository = hostRepository;
    }

    @GetMapping(value = "/{host_id}")
    public Iterable<Port> findByHostId(@PathVariable("host_id") Inte-
ger id) {

        Optional<Host> h = hostRepository.findById(id);

        return h.<Iterable<Port>>map(Host::getPorts).orElse(null);
    }

    @PostMapping
    public ResponseEntity<String> addByHostId(@RequestParam("port")
String portNumber, @RequestParam("hostId") String hostId) {

        if (!portNumber.matches("\\d+"))
            return new ResponseEntity<>("РќРµРґРѕСЃС‚Р°С‚РѕС‡РЅРѕ Р№
С„РѕСЂРјР°С‚Р°", HttpStatus.BAD_REQUEST);
```

```
        if (!hostId.matches("\\d+"))
            return new ResponseEntity<>("PJP·PµP» PSPµ PIC‹P±CЂP°PS",
HttpStatus.BAD_REQUEST);

        long port = Long.valueOf(portNumber);
        int hId = Integer.valueOf(hostId);


        if (port < 0 || port > 65535)
            return new ResponseEntity<>("PќPµPґPsPїCґCЃC,PëPjC‹P№
PSPsPjPµCЂ PїPsCЂC,P°", HttpStatus.BAD_REQUEST);

        else if (hostReposito-
ry.findById(hId).get().getPorts().stream()
                .anyMatch(o -> o.getNumber() == port))
            return new ResponseEntity<>("PµPsCЂC, CґP¶Pµ
PґPsP±P°PIP»PµPS", HttpStatus.BAD_REQUEST);

        Port p = new Port((int)port);

        hostRepository.findById(hId).ifPresent(x->x.addPort(p));
        if (hostRepository.findById(hId).isPresent())
            portRepository.save(p);
        else
            return new ResponseEntity<>("PJP·PµP»
PsC,CЃCґC,CЃC,PICґPµC,", HttpStatus.BAD_REQUEST);


        return ResponseEntity.ok("");
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<?> delete(@PathVariable("id") Integer id) {
        portRepository.findById(id).ifPresent(x-
>x.getHost().getPorts().removeIf(y -> y.getId().equals(id)));
        portRepository.deleteById(id);
        return ResponseEntity.noContent().build();
    }
}
```

WebModuleApplication.java

```
package edu.max;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class WebModuleApplication {

    public static void main(String[] args) {
        SpringApplication.run(WebModuleApplication.class, args);
    }}
```

main.js

```
function updateHostTable() {
    $.getJSON('/hosts', function (data) {

        var rowData = [];

        jQuery(data).each(function(i, hostEntity){

            var statusColor;
            var portStatus = [];
            var downPortsCounter = 0;


            jQuery(hostEntity.ports).each(function (i, portEntity) {
                portStatus.push(portEntity.up);
                if (portEntity.up === false)
                    downPortsCounter += 1;
            });


            if (downPortsCounter === portStatus.length)
                statusColor = "red";
            else if (downPortsCounter > 0 && downPortsCounter <
portStatus.length)
                statusColor = "yellow";
            else
                statusColor = "green";


            if ($("#hTable #" + hostEntity.id).hasClass("activeRow")){
                showPortsForHost(hostEntity.id);
                addRow("activeRow");
            }
            else addRow("");

            function addRow(activeRowClass) {
                rowData.push("<tr id='" + hostEntity.id + "' class='"
+ activeRowClass + "' >" +
                    "<th scope='row'>" + hostEntity.ipAddress +
"</th>" +
                    "<td><i>" + hostEntity.domainName + "</i></td>" +
                    "<td>\n" +
                    "<svg height=\"20\" width=\"20\">\n" +
                    "<circle cx=\"10\" cy=\"10\" r=\"8\" stroke-
width=\"1\" fill=\"" + statusColor + "\" />\n" +
                    "</svg>\n" +
                    "</td>" +
                    "<td><a href=\"#\">PJPrP°P»PëC,CЊ</a></td>" +
                    "</tr>")
            }


        });
```

```
        $("#hTable tbody tr").remove();
        $("#hTable tbody").append(rowData.join(""))
    });
}

function updateMonitoringHostTable() {
    $.getJSON('/hosts/monitoring', function (data) {

        var rowData = [];

        jQuery(data).each(function(i, mHostEntity){
            if (mHostEntity.up === true) {
                rowData.push("<tr>" +
                    "<td><div class='up'>" + mHostEntity.ipAddress +
"</div></td>");
            } else {
                rowData.push("<tr>" +
                    "<td><div class='down'>" + mHostEntity.ipAddress +
"</div></td>");
            }


            rowData.push(
                "<td>" + mHostEntity.anotherMHIpAdress + "</td>" +
                "<td>" + mHostEntity.targets.length + "</td>" +
                "<td><a id='" + mHostEntity.id + "' data-
toggle=\"modal\" " +
                "data-target=\"#monitoringSettings\"
href=\"#\">РџPsPґCЂPsP±PSPsCЃC,Pё</a></td>" +
                "</tr>")

        });

        $("#mTable tbody tr").remove();
        $("#mTable tbody").append(rowData.join(""));
    });
}

function updateLogsTable() {
    $.getJSON('/logs', function (data) {

        var rowData = [];

        jQuery(data).each(function(i, logEntity){

            rowData.push("<tr>" +
                "<td>" + logEntity.date + "</td>" +
                "<td><b>" + logEntity.host + " : " + logEntity.port +
"</b></td>" +
                "<td><i>" + logEntity.event + "</i></td>"+
                "</tr>")
        });

        $("#lTable tbody tr").remove();
        $("#lTable tbody").append(rowData.join(""));
    });
```

```
}

function showPortsForHost(id) {
    $.getJSON('/ports/' + id, function (data) {

        var portRow = [];

        jQuery(data).each(function (i, portEntity) {
            var statusColor;

            if (portEntity.up)
                statusColor = "green";
            else
                statusColor = "red";

            portRow.push("<tr id='" + portEntity.id + "'>" +
                "<th scope='row'>" + portEntity.number + "</th>" +
                "<td><i>" + portEntity.service + "</i></td>" +
                "<td>" +
                "<svg height=\"20\" width=\"20\">\n" +
                "<circle cx=\"10\" cy=\"10\" r=\"8\" stroke-
width=\"1\" fill=\"" + statusColor + "\" />\n" +
                "</svg>" +
                "</td>" +
                "<td><a href='#'>PJPґP°P»PëC,CЊ</a></td>" +
                "</tr>"
            );

        });

        $("#pTable tbody tr").remove();
        $("#pTable tbody").append(portRow.join(""));

    });
}

function updateModalContent(monitoringHostId) {

    $.getJSON('/hosts/monitoring/' + monitoringHostId, function (data)
{

        var hostRow = [];

        jQuery(data.targets).each(function (i, hostEntity) {

            hostRow.push("<tr>" +
                "<td>" + hostEntity.ipAddress + "</td>" +
                "</tr>"
            );
        });


        $("#modalTable tbody tr").remove();
        $("#modalTable tbody").append(hostRow.join(""));

    });
```

```
}

$(document).on("click", "#hTable tbody tr", function () {
    if ($(this).hasClass("activeRow")) {
        $(this).removeClass("activeRow");
        $("#pTable tbody tr").remove();
        $("#portsTableLabel").html("РЎРїРёСЃРѕРє РїРѕСЂС‚РѕРІ");
        chosenHostId = 0;
    }
    else {
        $("#hTable tbody tr").removeClass("activeRow");
        $(this).addClass("activeRow");
        showPortsForHost( $(this).attr('id'));
        $("#portsTableLabel").html("РЎРїРёСЃРѕРє РїРѕСЂС‚РѕРІ " +
$(this).find("th").text());
        chosenHostId = $(this).attr('id');
    }
});


var activeMonitoringHost;
$(document).on("click", "#mTable tbody a", function () {
    $("#modalLabel").html("Р¦РµР»Рё СѓР·Р»Р° РєР»Р°СЃС‚РµСЂР°<i> " +
$(this).parent().parent().find("th").text() + "</i>");
    updateModalContent( $(this).attr('id'));
    activeMonitoringHost = $(this).attr('id');
});

$('#hostIpForm').submit(function(e){
    e.preventDefault();
    $.ajax({
        url:'/hosts',
        type:'post',
        data:$('#hostIpForm').serialize(),
        success:function() {
            updateHostTable();
            updateMonitoringHostTable();
            $("#hostIpValidationMsg").text('');
        },
        error: function (data) {
            $("#hostIpValidationMsg").text(data.responseText);
        }
    });
});

var chosenHostId;

$('#portForm').submit(function(e){
    e.preventDefault();
    var dataToSend = $('#portForm').serialize() + "&hostId=" + chosen-
HostId;
    if(chosenHostId !== 0) {
        $.ajax({
            url:'/ports',
            type:'post',
            data: dataToSend,
```

```
            success:function() {
                showPortsForHost(chosenHostId);
                $("#hostIpValidationMsg").text('');
            },
            error: function (data) {
                $("#portValidationMsg").text(data.responseText);
            }
        });
    }
});

$('#monitoringForm').submit(function(e){
    e.preventDefault();
    $.ajax({
        url:'/hosts/monitoring',
        type:'post',
        data:$(this).serialize(),
        success:function() {
            updateMonitoringHostTable();
            $("#monitoringValidationMsg").text('');
        },
        error: function (data) {
            $("#monitoringValidationMsg").text(data.responseText);
        }
    });
});

$('#logForm').submit(function(e){
    e.preventDefault();
    $.ajax({
        url:'/logs',
        type:'delete',
        success:function() {
            updateLogsTable()
        }
    });
});

$(document).on("click", "#hTable tbody a", function (e) {

    e.preventDefault();
    $.ajax({
        url:'/hosts/' + $(this).parent().parent().attr('id'),
        type: 'delete',
        success: function () {
            updateHostTable();
            showPortsForHost('');
            $("#portsTableLabel").html("РЎРїРёСЃРѕРє РїРѕСЂС‚РѕРІ");

            updateMonitoringHostTable();
        }
    })

});

$(document).on("click", "#pTable tbody a", function (e) {
```

```
    e.preventDefault();
    $.ajax({
        url:'/ports/' + $(this).parent().parent().attr('id'),
        type: 'delete',
        success: function () {
            showPortsForHost(chosenHostId);
        }
    })

});

$("#mhostDelete").click(function () {
    $.ajax({
        url:'/hosts/monitoring/' + activeMonitoringHost,
        type: 'delete',
        success: function () {
            updateMonitoringHostTable();
            $("#monitoringValidationMsg").text('');
        },
        error: function (data) {
            $("#monitoringValidationMsg").text(data.responseText);
        }
    })
});

$("#idRangeSlider").on("input", function() {

    $("#rangeVal").text($(this).val());

    $("#rateButton").removeClass("btn-secondary");
    $("#rateButton").addClass("btn-primary");
    $("#rateButton").prop("disabled", false);
});

$("#rateForm").submit(function(e) {
   e.preventDefault();
    $.ajax({
        url:'/hosts/monitoring/rate/',
        type:'post',
        data:$(this).serialize()
    });
});

$("#rateButton").on("click", function () {

    $(this).removeClass("btn-primary");
    $(this).addClass("btn-secondary");
    $(this).prop("disabled", true);
});

intervalId = setInterval(updateHostTable, 3000);
intervalId = setInterval(updateMonitoringHostTable, 3000);
intervalId = setInterval(updateLogsTable, 3000);
```