

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

СИСТЕМА VASCUK ДЛЯ КЛАСТЕРОВ

XCP-NG. СЕРВЕР

Студент _____ Татаршев Андрей Алексеевич
(подпись, дата) 2.06.23 (ФИО полностью)
Руководитель _____ Попов К.В.
(подпись, дата) 9.06.2023 (фамилия, инициалы)
Нормоконтролёр _____ Бычков А.С.
(подпись, дата) 9.06.2023 (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 2.06.2023 г. № 19)

Заведующий кафедрой _____ М.А. Митрохин
(подпись) (фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от
14.06.2023г. № 12)

Секретарь ГЭК _____ Е.И. Гурин
(подпись) (фамилия, инициалы)
Пенза 2023

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«26» апреля 2023 г.

ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Татаршев Андрей Алексеевич гр.19ВВП2 факультета ВТ

2. Тема работы Система Backup для кластеров хср-пг. Сервер.

Тема утверждена приказом ПГУ № 458/о от "26" апреля 2023 г.

3. Руководитель бакалаврской работы Попов К.В.

4. Задание на работу (назначение разработки, исходные данные и т.п.)
Разработать серверную часть системы Backup для кластеров хср-пг. Язык
программирования rust, среда разработки CLion. Сервер должен быть стабильным и не
подверженным к сбоям, чтобы обеспечивать непрерывную работу приложения и
использовать следующие технологии:

1. Rust

2. Actix

3. Асинхронное программирование

4. Сетевое программирование

5. Nginx

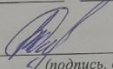
6. Хср-пг

5. Перечень подлежащих разработке вопросов
Требуется произвести анализ предметных областей: изучить существующие системы для
кластера хср-пг, выделить их достоинства и недостатки. Выявить потребности
пользователей. Выбрать язык
программирования, среду разработки, СУБД. Определить
архитектуру приложения и вспомогательные средства разработки. Изучить функционал
библиотек для разработки серверной части системы Backup для кластеров хср-пг.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	32	26.04.2023 – 28.04.2023	
2	Анализ требований	32	29.04.2023 – 1.05.2023	
3	Анализ архитектуры	30	2.05.2023 – 4.05.2023	
4	Выбор технологий, применяемых при проектировании и разработке	40	5.05.2023 – 9.05.2023	
5	Разработка основной части программы	136	10.05.2023 – 23.05.2023	
6	Оформление пояснительной записки	54	24.05.2023 – 31.05.2023	

Дата выдачи задания " 26 " 04 2023 г.

Руководитель бакалаврской работы  Попов К.В.
(подпись, дата)

Задание к исполнению принял студент  Татаршев А.А.
(подпись, дата)

Работу к защите допустить  09.06.2023 Фиопова Л.Р.
Декан факультета (подпись, дата)

Реферат

Бакалаврская работа содержит 95 листа, 25 рисунков, 5 источников, 3 приложения

СЕРВЕР, РЕЗЕРВНОЕ КОПИРОВАНИЕ, БАЗА ДАННЫХ, ФУНКЦИЯ, СЕРВИС, МАШИНА, ПЛАН, КЛАСТЕР, RUST, ACTIX, NGINX, HTTP BACKUP, WEB

Цель работы – разработать сервер, при помощи которого можно управлять резервным копированием машин

Решением поставленной задачи является разработка серверной части системы Backup для кластеров *xcp-ng*, которая будет выполнять следующие функции: подключение к управляющей машине, формирование расписания резервного копирования на управляющей машине, выполнение резервного копирования, восстановление машины по резервной копии, отправка уведомлений о завершении резервного копирования, загрузка скрипта на управляющую машину.

В итоге разработана серверная часть для системы Backup для кластеров *xcp-ng*, которая обеспечивает высокую скорость обработки запросов и передачи данных. Система Backup полностью соответствующий цели данной выпускной квалификационной работы. В ходе достижения результата, были использованы следующие технологии: *rust, actix, CLion, Diesel, Cron, postgresSQL, Nginx, Xcp-ng, serde*. Система Backup протестирована и готова к эксплуатации.

					ПГУ1.090301.35.001 ПЗ		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Татаршин А.А.			Лит.	Лист	Листов
Провер.		Попов К.В.				2	95
И. Контр.		Бычков А.С.			ФВТ, каф. ВТ гр. 19ВВП2		
Утверд.		Митрахин М.А.					
					Система Backup для кластера xcp-ng. Сервер Пояснительная записка		

Содержание

Перечень сокращений и обозначений	4
Введение	5
1 Система <i>Васкир</i> для кластеров <i>хср-ng</i> . Сервер	6
1.1 Анализ предметной области	6
1.2 Анализ актуальности разработки	7
1.3 Анализ аналогов	8
1.4 Анализ архитектуры	10
1.5 Постановка задачи	13
1.6 Выбор инструментов разработки	15
1.7 Реализация	18
1.7.1 Написание <i>web</i> сервера	18
1.7.2 Разработка скрипта	42
1.7.3 Настройки прокси сервера	44
2 Безопасность жизнедеятельности	46
2.1 Правила безопасности при использовании ПК	46
Заключение.	46
Список используемых источников	48
Приложение А – <i>UML</i> диаграммы	49
Приложение Б – Листинг программы	53
Приложение В – Презентация	83

Перечень сокращений и обозначений

В настоящей работе применяют следующие сокращения и обозначения

БД (база данных) - структурированная коллекция данных, организованных и хранимых в электронном виде

ОРСУБД (объектно-реляционная система управления базами данных) – это Реляционная СУБД, поддерживающая некоторые технологии, присущие объектно-ориентированным СУБД и реализующие объектно-ориентированный подход

ПК (персональный компьютер) - электронное устройство, предназначенное для индивидуального использования пользователем

СУБД (система управления базами данных) - программное обеспечение, предназначенное для создания, управления и обработки баз данных

API (*Application Programming Interface*) – описание способов, которыми одна компьютерная программа может взаимодействовать с другой программой

HTTP (*Hypertext Transfer Protocol*) - это протокол передачи гипертекста, который используется для передачи данных между веб-сервером и клиентскими приложениями (например, веб-браузерами)

IDE (*Integrated development environment*) – интегрированная среда разработки

JSON (*JavaScript Object Notation*) - это легкий формат обмена данными, основанный на синтаксисе объектов JavaScript

ORM (*Object-Relational Mapping*) – это Технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования

REST (*Representational State Transfer*) - это архитектурный стиль для создания веб-сервисов

SQL (*Structured Query Language*) – язык структурированных запросов

UML (*Unified Modeling Language*) – унифицированный язык моделирования

					ПГУ1.090301.35.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		4

Введение

В современном мире, когда мы все больше зависим от цифровых технологий и храним наши важные данные в электронном виде, резервное копирование становится необходимостью. Несчастные случаи, такие как сбои в работе жестких дисков, вирусы, кибератаки и другие непредвиденные события могут привести к потере данных, которые мы считали безопасными. Резервное копирование позволяет нам не только сохранить данные, но и быстро восстановить их в случае их потери.

Кроме того, резервное копирование также важно для защиты наших данных от кражи и потери из-за физических бедствий, таких как пожары, наводнения и другие стихийные бедствия. Резервное копирование может защитить наши данные в случае, если наши устройства будут украдены или уничтожены.

В целом, резервное копирование является важной частью нашей стратегии безопасности данных. Оно позволяет нам сохранить и защитить наши данные, и в случае их потери быстро восстановить их. Поэтому, регулярно создавать резервные копии.

В данной работе мы рассмотрим технологию резервного копирования виртуальных машин кластера *XCP-ng* при помощи *Web*-сервера, анализ предметной области, виды приложений, позволяющие выполнять резервное копирование виртуальных машин для кластера *xcp-ng*, их достоинства и недостатки.

					ПГУ1.090301.35.001 ПЗ	Лист
						5
Изм.	Лист	№ докум.	Подпись	Дата		

1 Система Vaskur для кластеров xcp-ng. Сервер

1.1 Анализ предметной области

Резервное копирование — это важный инструмент для обеспечения сохранности данных в различных сферах, включая бизнес, научные исследования и личное использование. Оно позволяет создать резервную копию важных данных, чтобы в случае их потери или повреждения можно было восстановить их.

Один из инструментов, который может быть использован для создания резервных копий — это *xcp-ng*. *Xcp-ng* — это открытая виртуализационная платформа, основанная на проекте *Xen* и поддерживаемая сообществом. Она предоставляет возможность создания кластеров виртуальных машин, которые могут быть использованы для хранения и обработки данных.

Использование *xcp-ng* для создания резервных копий может иметь ряд преимуществ. Во-первых, платформа обеспечивает высокую степень надежности и безопасности данных. Во-вторых, *xcp-ng* предоставляет возможность создания снимков виртуальных машин, что упрощает процесс создания резервных копий.

В целом, использование *xcp-ng* для создания резервных копий может быть эффективным способом обеспечения сохранности важных данных и повышения безопасности в различных сферах деятельности.

Однако, использование *XCP-ng* для резервного копирования может быть не всегда удобным для пользователей, поскольку система не предоставляет удобных средств для ведения планов и учета их выполнения. Для работы с *XCP-ng* необходимо уметь пользоваться командной строкой и знать основы *Linux*. Это может быть неудобно для обычных пользователей.

Для решения этой проблемы может быть полезно разработать сервер, который будет управлять планами резервного копирования. Такой сервер может получать информацию о машинах, планах резервного копирования, записывать ее в базу данных и передавать клиенту. Это позволит обеспечить более удобный

					ПГУ1.090301.35.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		6

доступ к планам резервного копирования. Кроме того, такой сервер может предоставлять более гибкие настройки для управления планами резервного копирования, чем это доступно в стандартной конфигурации *ХСР-ng*.

Важным аспектом работы сервера является учет планов резервного копирования и информации о выполнении их. Для этого сервер должен обеспечивать возможность создания, изменения и удаления планов, а также отслеживать выполнение каждого плана и сохранять информацию о результатах копирования.

Также сервер должен быть надежным и устойчивым к сбоям, чтобы гарантировать сохранность данных в любой ситуации, обеспечивать высокую скорость обработки запросов и передачи данных и обеспечивать безопасность хранения и передачи данных, чтобы избежать несанкционированного доступа к ним.

1.2 Анализ актуальности разработки

Разработка приложения для резервного копирования для кластеров *хср-ng* является чрезвычайно актуальной и значимой задачей в современном мире информационных технологий. В условиях стремительного развития облачных технологий и взрывного прироста объемов данных, которые требуется хранить и обрабатывать, важность резервного копирования становится все более очевидной и неотъемлемой составляющей процесса обеспечения безопасности информации.

Кластеры *хср-ng* представляют собой незаменимый инструмент для виртуализации серверов, позволяющий управлять несколькими виртуальными машинами на одном и том же физическом сервере. Тем не менее, в случае возникновения сбоев в работе кластера или отказа сервера, существует реальная угроза потери всех данных, хранящихся на виртуальных машинах. Для предотвращения таких катастрофических последствий крайне важно регулярно создавать резервные копии всех существенных данных.

Разработка высококачественного приложения для резервного копирования, специально предназначенного для кластеров хср-ng, имеет огромное значение и преимущества. Оно обеспечит полную автоматизацию процесса создания резервных копий, что в значительной степени упростит жизнь системным администраторам и повысит надежность хранения данных. Кроме того, данное приложение может предоставить широкие возможности по управлению резервными копиями, такие как удаление устаревших копий или восстановление данных из выбранной резервной копии.

В целом, разработка приложения для резервного копирования для кластеров хср-ng является важным и перспективным направлением в области информационных технологий. Оно поможет обеспечить надежность и безопасность данных, упростить управление резервными копиями и снизить риск потери информации. Автоматизированный и надежный процесс создания и управления резервными копиями сделает жизнь системных администраторов более эффективной и способствует бесперебойной работе бизнес-процессов, основанных на виртуализации и облачных технологиях.

1.3 Анализ аналогов

Существует множество решений для бекапа виртуальных машин, которые могут помочь вам сохранить данные в безопасности и быстро восстановить их при необходимости. Одними из таких являются приложения *Bacula*, *Duplicati*.

Bacula — это мощное и гибкое приложение для резервного копирования, которое может быть использовано на различных платформах, включая *XCP-ng*. Оно имеет множество функций и возможностей, включая поддержку различных типов носителей, сжатие и шифрование данных, а также поддержку удаленного резервного копирования.

Однако, настройка и управление *Bacula* может быть сложной задачей для новых пользователей, особенно если речь идет о настройке более сложных

					ПГУ1.090301.35.001 ПЗ	Лист
						8
Изм.	Лист	№ докум.	Подпись	Дата		

сценариев резервного копирования. Это может потребовать дополнительного времени и ресурсов для освоения всех функций и возможностей приложения.

Несмотря на это, *Vacula* может быть идеальным решением для средних и крупных предприятий, которые нуждаются в надежной и масштабируемой системе резервного копирования данных. Однако для использования *Vacula* может потребоваться определенный уровень знаний и навыков в области администрирования систем.

Duplicati— это бесплатное и открытое приложение для резервного копирования, которое может использоваться на различных платформах, включая *Windows*, *macOS* и *Linux*. *Duplicati* поддерживает несколько методов резервного копирования, включая локальное хранение, удаленное хранение на облачных сервисах, а также на других удаленных серверах.

Одним из ключевых преимуществ *Duplicati* является его удобный интерфейс, который делает настройку и управление процессом резервного копирования более простым и интуитивно понятным. Приложение также поддерживает шифрование данных, что обеспечивает высокую степень безопасности и защиты от несанкционированного доступа к данным.

Duplicati также имеет множество дополнительных функций, таких как сжатие данных и мультиязычный интерфейс, что делает его более удобным и универсальным в использовании.

Однако, некоторые пользователи отмечают, что процесс резервного копирования может быть несколько медленным и занимать больше времени, чем при использовании других приложений для резервного копирования. Кроме того, некоторые функции, такие как автоматическое обнаружение изменений в файлах, могут работать не всегда точно.

В целом, *Duplicati* представляет собой удобное и бесплатное приложение для резервного копирования, которое поддерживает различные методы хранения данных и обеспечивает высокую степень безопасности и защиты. Однако, он

может несколько отставать от других приложений по скорости и точности в некоторых функциях.

Альтернативное решение в области резервного копирования данных является оправданным из-за недостатков существующих приложений. Например, *Bacula*, хоть и является мощной и гибкой системой резервного копирования, может быть сложной в настройке и управлении для новых пользователей. *Duplicati*, в свою очередь, является бесплатным и открытым приложением с удобным интерфейсом, но некоторые пользователи сообщают о медленной работе и некоторых проблемах с точностью. Создание альтернативного решения может предоставить пользователям больше возможностей выбора наиболее подходящего для их потребностей приложения. Новое решение может быть более удобным, быстрым и точным, а также иметь дополнительные функции и возможности, которых нет в *Bacula* или *Duplicati*. Кроме того, новое решение может быть более доступным для пользователей с разным уровнем знаний в области администрирования систем, что может привлечь новых пользователей и увеличить конкуренцию на рынке резервного копирования данных.

1.4 Анализ архитектуры

Архитектура приложения включает в себя два основных компонента: *Nginx* и *HTTP* сервер. *Nginx* является веб-сервером, который используется как прокси-сервер для перенаправления запросов клиентов на *HTTP* сервер через *RESTAPI*. Это позволяет улучшить производительность приложения, поскольку *Nginx* может кэшировать запросы и ускорять обработку запросов.

HTTP сервер обрабатывает запросы клиентов и возвращает результаты обратно через *Nginx*. Для выполнения операций и доступа к файловой системе *HTTP* сервер подключается к управляющей машине по протоколу *SSH*. Это обеспечивает безопасность и защиту от несанкционированного доступа к файловой системе.

База данных *PostgreSQL* используется для хранения данных приложения. Она обеспечивает надежность и безопасность хранения данных, а также позволяет эффективно работать с большим объемом данных.

Nginx может использоваться как кэширующий прокси-сервер для ускорения обработки запросов. Он может кэшировать ответы от *HTTP* сервера и быстро возвращать их клиентам, что позволяет снизить нагрузку на сервер и улучшить производительность приложения.

В целом, данная архитектура обеспечивает высокую производительность и надежность приложения. Она позволяет эффективно обрабатывать запросы клиентов через *RESTAPI*, обеспечивая безопасность и защиту данных

На рисунке 1.1 представлена диаграмма развертывания

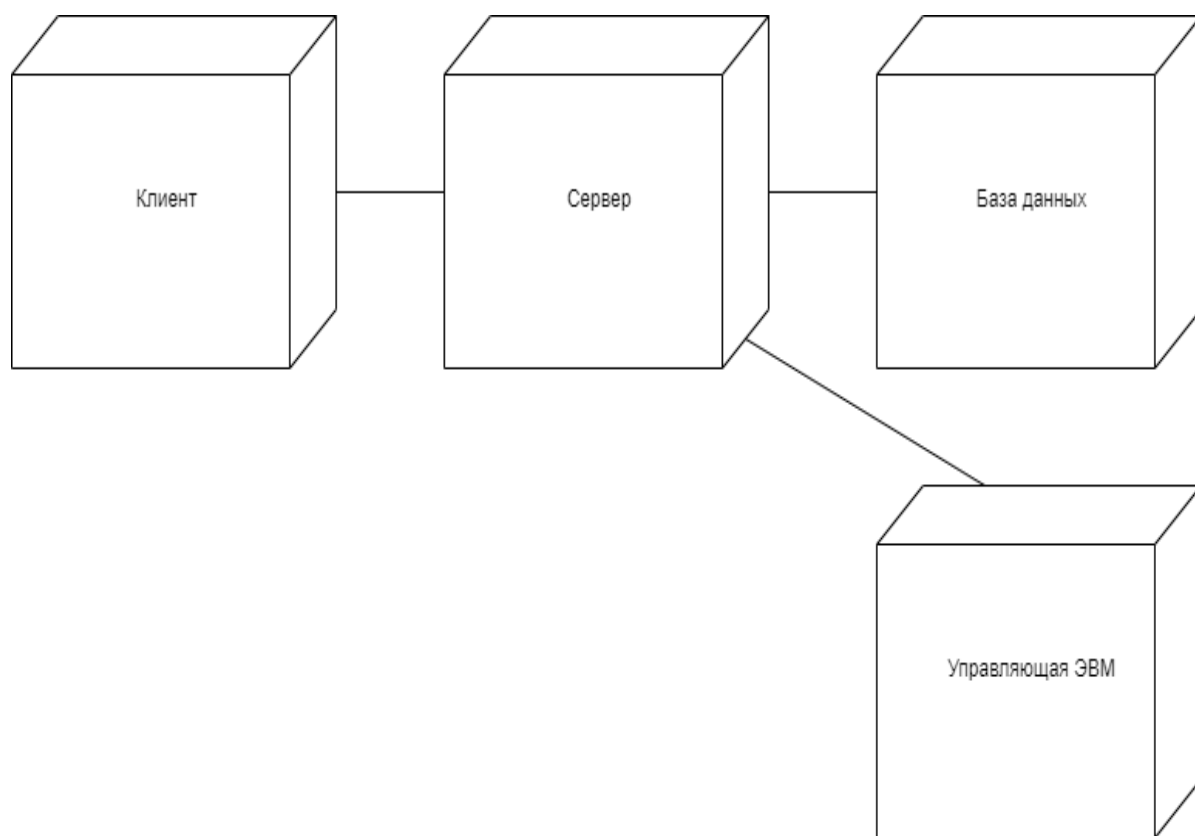


Рисунок 1.1 - *UML* диаграмма развертывания

На рисунке 1.2 представлена диаграмма прецедентов

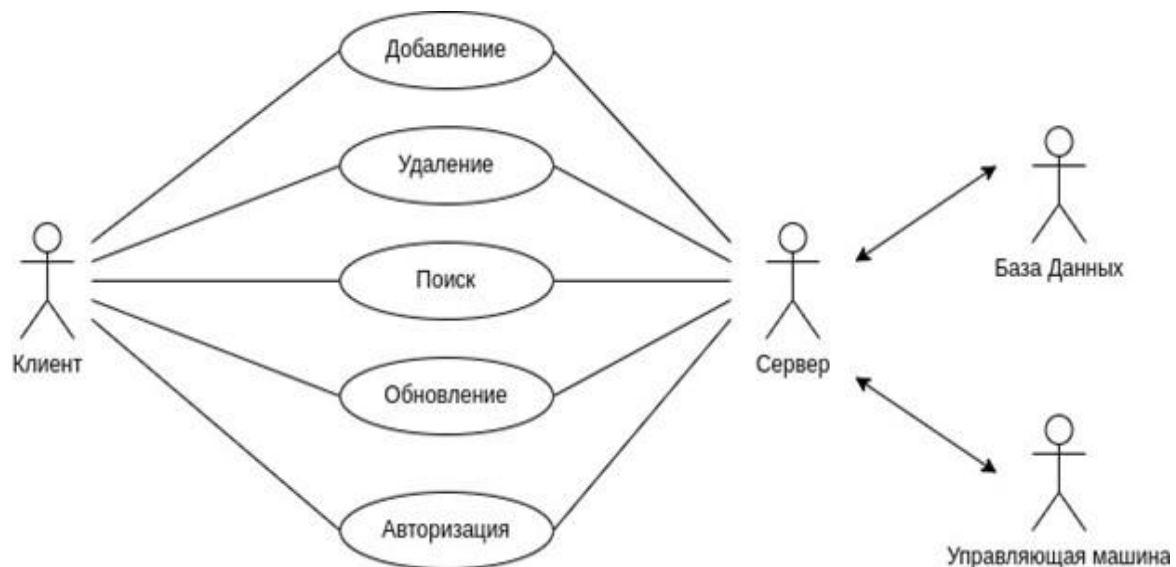


Рисунок 1.2 -UML диаграмма прецедентов

На рисунке 1.3 представлена диаграмма последовательности

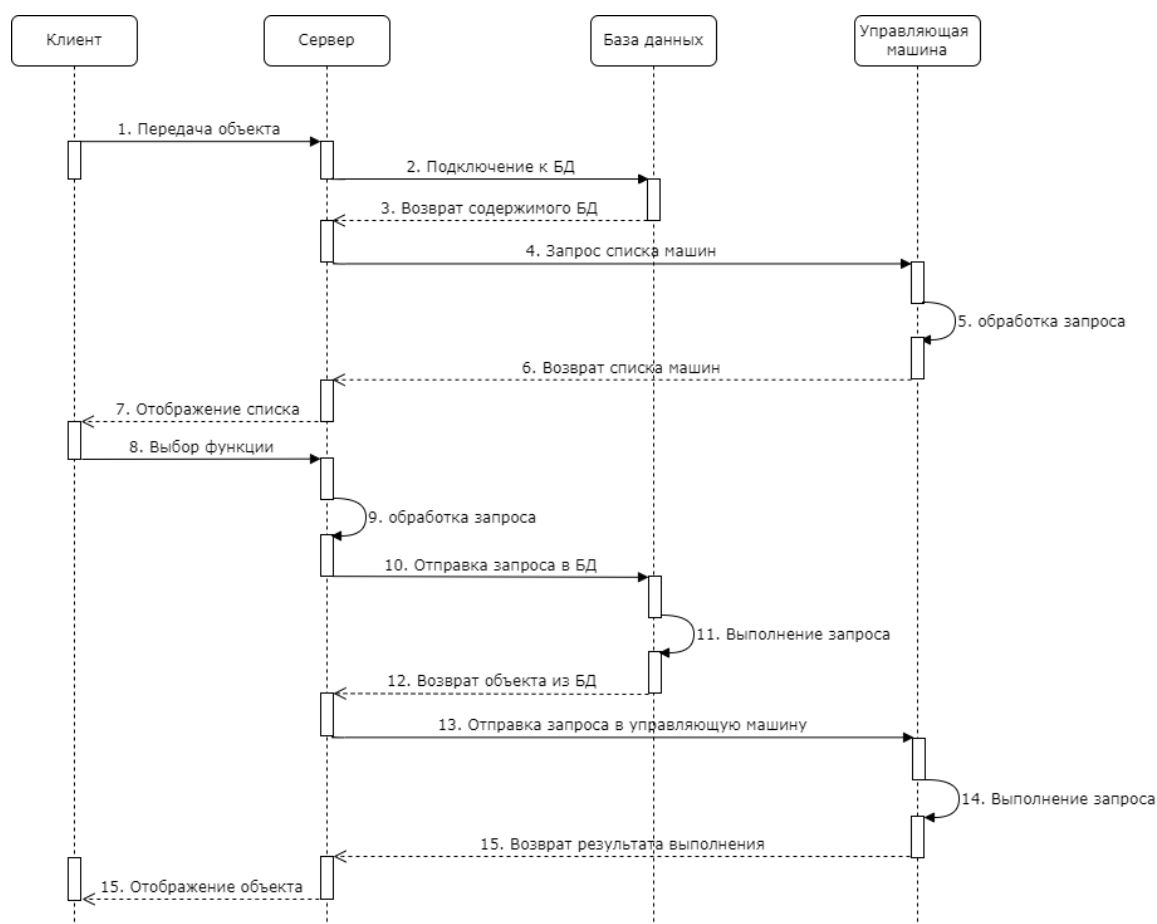


Рисунок 1.3– UML диаграмма последовательности

На рисунке 1.4 представлена диаграмма деятельности

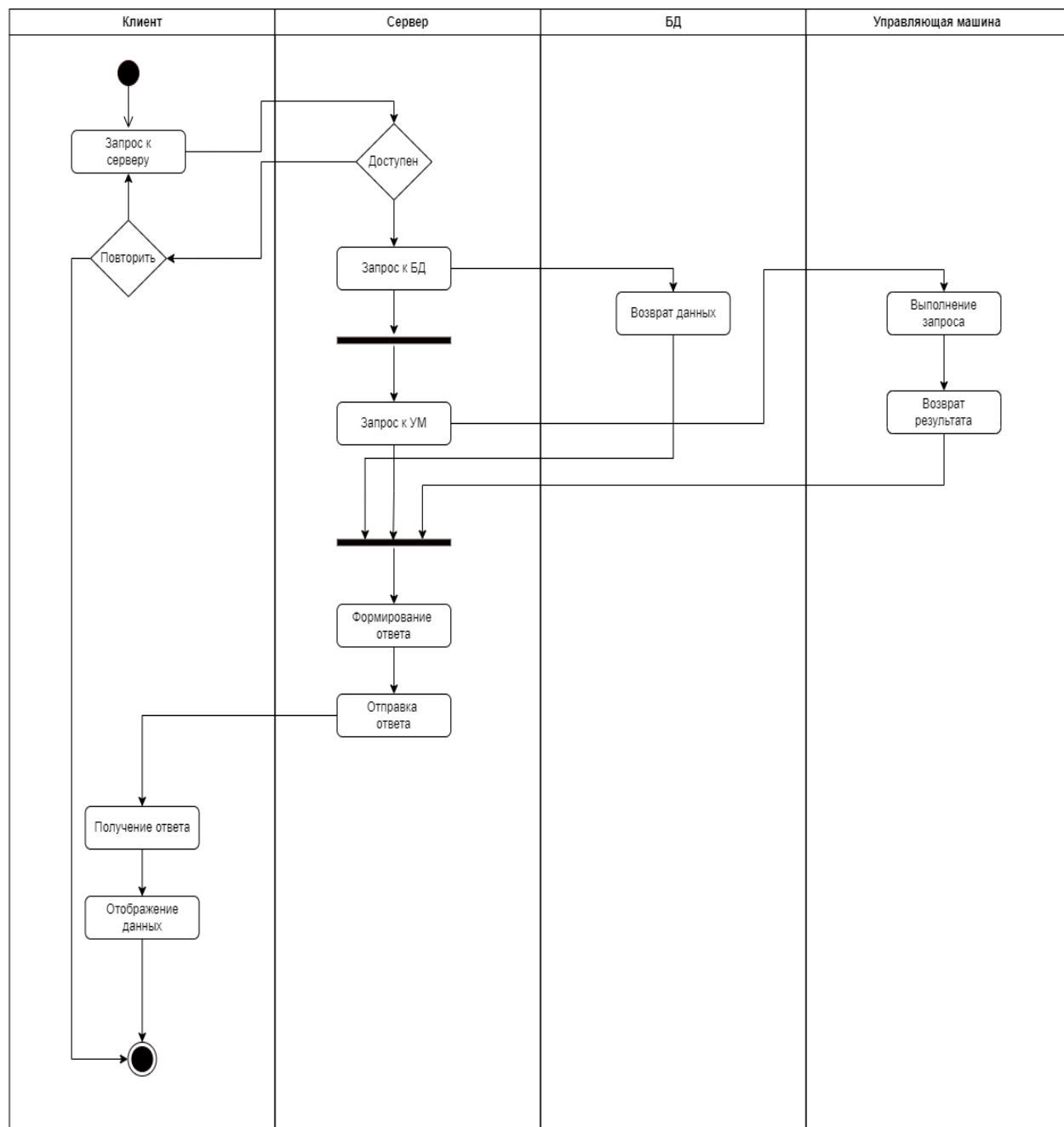


Рисунок 1.4 - UML диаграмма деятельности.

1.5 Постановка задачи

Целью данной работы является разработка серверной части приложения для резервного копирования виртуальных машин кластера XCP-ng. Сервер должен задавать планы резервного копирования на управляющей машине,

сохранять данные о машинах, их состоянии и состоянии резервного копирования в базу данных, а также обеспечивать стабильный обмен данными с клиентом

Возможности и функционал программного обеспечения:

- подключение к управляющей машине. Сервер получает данные в формате *json* и подключается к управляющей машине по *ssh*;
- получение списка машин. Сервер обращается к управляющей машине и получает список машин находящихся в кластере, преобразует полученную информацию в формат *json* и отправляет её клиенту;
- формирование расписания резервного копирования на управляющей машине. Сервер получает данные в формате *json*, преобразует их в формат *cron* и создаёт план на заданную машину;
- выполнение резервного копирования. Сервер получает *id* машины и выполняет скрипт резервного копирования для данной машины;
- восстановление машины по резервной копии. Сервер получает имя резервной копии машины и выполняет скрипт восстановления машины для машины;
- отправка уведомлений о завершении резервного копирования и восстановлении машины или отправка уведомления об ошибке в процессе восстановления или резервного копирования;
- получения списка резервных копий заданных машин;
- загрузка скрипта на управляющую машину. При подключении к управляющей машине приложение проверяет существует ли скрипт на управляющей машине и если его не находит, то предлагает произвести настройку и загружает скрипт;
- обновление *ip*. При подключении к управляющей машине сервер получает *ip* машины на которой он установлен и отправляет его управляющей машине, это нужно для обмена данными между управляющей машиной и сервером.

1.6 Выбор инструментов разработки

Выбор инструментов разработки является важным этапом в создании любого программного продукта. Выбор правильных инструментов и технологий может существенно повысить качество и производительность приложения, снизить время разработки.

Для реализации программного обеспечения для резервного копирования виртуальных машин кластера *XCP-ng* при помощи *Web*-сервера были рассмотрены следующие технологии и языки:

CLion - это интегрированная среда разработки (*IDE*) от *JetBrains*, которая предназначена для работы с языками программирования. Она имеет множество функций, таких как автодополнение кода, отладчик, поддержку системы контроля версий и множество других инструментов для удобной и эффективной разработки

Rust - это системный язык программирования с открытым исходным кодом, который разработан для создания быстрых и надежных приложений. Он был создан компанией *Mozilla* и выпущен в 2010 году. Это сказано в [2]. Некоторые из его особенностей включают в себя:

- безопасность. *Rust* предоставляет механизмы безопасности на уровне компиляции, которые позволяют избежать ошибок типа *null* и буферного переполнения;

- высокая производительность: *Rust* имеет низкий уровень абстракции и позволяет более эффективно использовать ресурсы системы;

- многопоточность: *Rust* предоставляет механизмы многопоточности, которые позволяют создавать параллельные приложения.

Actix— это библиотека для создания веб-приложений на *Rust*. Она предоставляет механизмы для обработки *HTTP*-запросов и управления состоянием приложения. Некоторые из ее особенностей включают в себя:

– асинхронность: *Actix* предоставляет механизмы асинхронной обработки запросов, что позволяет увеличить производительность приложения;

– многопоточность. *Actix* позволяет создавать многопоточные приложения, что позволяет более эффективно использовать ресурсы системы;

– *webSocket*. *Actix* предоставляет механизмы для работы с протоколом *WebSocket*, что позволяет создавать приложения в режиме реального времени.

Diesel - это библиотека для работы с базами данных на *Rust*. Она предоставляет механизмы для создания запросов к базе данных и управления ее состоянием. Некоторые из ее особенностей включают в себя.

– безопасность. *Diesel* предоставляет механизмы безопасности на уровне компиляции, что позволяет избежать ошибок при работе с базой данных;

– абстракция. *Diesel* предоставляет абстракцию над различными типами баз данных, что позволяет легко переключаться между ними;

– миграции. *Diesel* предоставляет механизмы для создания миграций базы данных, что позволяет легко изменять ее структуру.

Serde— это библиотека для сериализации и десериализации данных на *Rust*. Она предоставляет механизмы для преобразования данных из одного формата в другой. Некоторые из ее особенностей включают в себя:

– поддержка различных форматов. *Serde* поддерживает множество форматов, таких как *JSON*, *TOML*, *YAML* и другие;

– безопасность. *Serde* предоставляет механизмы безопасности на уровне компиляции, что позволяет избежать ошибок при работе с данными;

– гибкость. *Serde* предоставляет механизмы для настройки процесса сериализации и десериализации данных, что позволяет легко адаптировать его под конкретные нужды.

SSH2— это библиотека для работы с протоколом *SSH* на *Rust*. Она предоставляет механизмы для установления защищенного соединения с удаленным сервером и выполнения команд на нем. Некоторые из ее особенностей включают в себя:

– безопасность. *SSH2* предоставляет механизмы безопасности на уровне протокола, что позволяет защитить передаваемые данные;

– асинхронность. *SSH2* предоставляет механизмы асинхронной работы, что позволяет увеличить производительность приложения;

– гибкость. *SSH2* предоставляет механизмы для настройки процесса установления соединения и выполнения команд на удаленном сервере, что позволяет легко адаптировать его под конкретные нужды.

XCP-ng - это свободная платформа виртуализации, основанная на открытом исходном коде Xen. Он предоставляет возможность создания виртуальных машин на основе различных операционных систем, таких как *Linux*, *Windows* и другие.

Cron — это стандартный инструмент в Unix-подобных операционных системах, который позволяет запускать задачи по расписанию. Об этом говорится в [5] Он может использоваться для автоматического выполнения резервного копирования данных на определенной частоте, например ежедневно, еженедельно или ежемесячно. Cron может быть настроен для запуска скриптов или команд на определенное время и дату, что делает его полезным инструментом для автоматизации повторяющихся задач. Например, он может быть использован для автоматического обновления системы или регулярного очищения временных файлов.

Nginx— это веб-сервер и прокси-сервер с открытым исходным кодом, который разработан для обработки большого количества запросов и обеспечения высокой производительности. Некоторые из его особенностей включают в себя:

– проксирование. *Nginx* может использоваться в качестве прокси-сервера для балансировки нагрузки и распределения запросов между несколькими серверами;

– статический контент. *Nginx* может использоваться для обслуживания статического контента, такого как *HTML*, *CSS* и *JavaScript*;

– архитектура. *Nginx* имеет асинхронную архитектуру, которая позволяет обрабатывать большое количество запросов одновременно.

PostgreSQL - это объектно-реляционная система управления базами данных (ОРСУБД) с открытым исходным кодом, которая разработана для хранения и управления большим объемом структурированных данных. Об этом говорится в [4]. Некоторые из ее особенностей включают в себя:

– расширяемость. *PostgreSQL* имеет множество расширений, которые позволяют добавлять новые функциональные возможности;

– безопасность. *PostgreSQL* предоставляет механизмы безопасности на уровне базы данных, которые позволяют защитить данные от несанкционированного доступа;

– *acid*-совместимость. *PostgreSQL* гарантирует соблюдение принципов *ACID* (атомарность, согласованность, изолированность, долговечность) при выполнении транзакций;

– масштабируемость. *PostgreSQL* может масштабироваться горизонтально и вертикально, что позволяет обрабатывать большой объем данных.

1.7 Реализация

1.7.1 Написание *web* сервера

Для написания *web* сервера был выбран фреймворк *actix*.

Actix— это фреймворк для создания высокопроизводительных и масштабируемых веб-приложений на языке программирования *Rust*. Он основан на модели актёров, которая позволяет создавать легковесные и независимые компоненты приложения, каждый из которых может быть запущен в отдельном потоке.

Actix предоставляет множество инструментов и функциональных возможностей, таких как:

- обработка запросов *HTTP* с использованием асинхронных методов и мультиплексирование соединений *TCP*;

- поддержка *WebSocket* соединений и обработка событий в режиме реального времени;
- интеграция с базами данных и *ORM (Object-Relational Mapping)* библиотеками;
- поддержка многопоточности и распределенных систем;
- различные инструменты для отладки и тестирования приложений.

Actix имеет высокую производительность благодаря использованию асинхронной обработки запросов и оптимизации работы с памятью. Он также обладает высокой степенью безопасности благодаря статической типизации *Rust* и проверкам времени компиляции. Об это говориться в [3].

В целом, *Actix* - это мощный и гибкий фреймворк для создания высокопроизводительных веб-приложений на *Rust*. Он подходит для различных задач, от простых *API* до сложных микросервисов и распределенных систем.

Исходя из плюсов фреймворка *actix* была написана функция *main()*, которая запускает веб-сервер на основе фреймворка *Actix-web*. В начале функции происходит загрузка переменных окружения из файла *.env* и создание объекта конфигурации, используя структуру *Config* из модуля *config*. Затем создается объект *ConnectionManager* для управления соединениями с базой данных *Postgres*, а также создается пул соединений с помощью библиотеки *r2d2*.

Далее происходит настройка сервера: устанавливается *CORS*-политика, создается объект *AppData* для передачи данных между обработчиками запросов, устанавливается ограничение на размер передаваемых файлов, вызывается функция *app_config* для настройки обработчиков запросов и устанавливается ограничение на размер *JSON*-объектов.

Наконец, создается объект *HttpServer* и запускается на указанном порте и хосте. В случае успешного запуска сервера функция возвращает *Ok()*, в противном случае возвращается ошибка *std::io::Result*. Код функции представлен ниже.

```
#[actix_web::main]
async fn main() -> std::io::Result<()> {
```

```

dotenv().ok();
let config = crate::config::Config::from_env().unwrap();
let main_config =
crate::config::Config::from_env().unwrap();
let manager =
ConnectionManager::<PgConnection>::new(main_config.database.url);
let pool = r2d2::Pool::builder()
    .build(manager)
    .expect("Failed to create postgres pool.");

info!(
    "Starting server at http://{host}:{port}",
    main_config.server.host, main_config.server.port
);
HttpServer::new( move|| {
    let cors = Cors::permissive();
    App::new()
        .data(AppData::new(pool.clone(), config.clone()))
        .wrap(cors)
        .app_data(actix_web::web::Bytes::configure(|cfg| {
            cfg.limit(1024 * 1024 * 250) // Set max file
size to 6 Mib
        })))
        .configure(app_config)
        .data(web::JsonConfig::default().limit(1024 * 1024
* 250))

    })
    .bind(format!(
        "{host}:{port}",
        main_config.server.host, main_config.server.port
    ))?
    .run()
    .await
}

```

Архитектура приложения состоит из нескольких компонентов:

Описание структур базы данных, которая находится в файле *schema.rs*.

Файл *schema.rs* в библиотеке *Diesel* используется для определения структуры базы данных и ее таблиц. Он содержит определения таблиц и их полей, а также связей между таблицами.

Этот файл является основой для генерации кода на языке *Rust*, который позволяет взаимодействовать с базой данных. *Diesel* использует макросы для генерации этого кода, основываясь на определениях таблиц и полей в файле *schema.rs*.

В общем случае файл *schema.rs* необходим для создания и поддержки базы данных при использовании библиотеки *Diesel*. Он является основным источником информации о структуре базы данных и используется для генерации соответствующего кода на *Rust*.

Структуры *Rust*.

В директории *entity* содержатся файлы с функциями, преобразующие логику базы данных в структуры. Это необходимо из-за специфики языка программирования. Также эти структуры реализуют сериализацию и десериализацию

Структуры запросов.

Каталог *dto* хранит модели запросов и функции для их соответствующего преобразования в структуры из каталога *entity*, а также модели и функции по преобразованию ответов из структур *entity* в структуру ответа.

Работа с базой данных.

В каталоге *dao* содержатся файлы с функциями для формирования *SQL*-запросов для работы с базой данных.

Создание планов

В каталоге *cron* содержатся файлы с функциями для создания планов на управляющей машине.

Сервисы

Web сервер реализует компоненты, которые предоставляют определенный функционал для обработки запросов, называемые сервисами. Об этом говорится в [1] Они могут обрабатывать запросы на авторизацию, работу с базой данных или отправку электронной почты. Каждый сервис реализован как отдельный компонент приложения, который может быть легко масштабирован и переиспользован в других приложениях. В веб-серверах используется модульная архитектура, где каждый сервис представляет собой независимую часть приложения, которая может быть легко добавлена или удалена в зависимости от потребностей

Теперь, когда мы рассмотрели основные компоненты веб-приложения, давайте перейдем к описанию нашей программы.

Начнём с запроса на соединения с управляющей машиной.

Функция *connect()* является обработчиком *POST*-запроса на подключение к удаленному хосту. Она принимает два параметра:

- *app_data web::Data<AppData>* - объект, содержащий данные приложения, такие как пул соединений с базой данных;
- *connect Json<ConnectRequest>* - JSON-объект, содержащий данные для подключения к удаленному хосту.

Функция начинается с попытки установить соединение с удаленным хостом с помощью метода *TcpStream::connect()*. Если соединение не установлено, функция возвращает *HTTP*-ответ со статусом *BadRequest* и сообщением об ошибке.

Затем создается новая сессия *SSH* с помощью метода *Session::new()*. Если создание сессии не удалось, функция возвращает *HTTP*-ответ со статусом *BadRequest* и сообщением об ошибке.

Далее происходит авторизация на удаленном хосте с помощью метода *userauth_password()*. Если авторизация не удалась, функция возвращает *HTTP*-ответ со статусом *BadRequest* и сообщением об ошибке.

Затем открывается новый канал *SSH* с помощью метода *channel_session()*. На этом канале выполняется команда "*ls test/backupsript.sh*" с помощью метода *exec()*. Результат выполнения команды читается с помощью метода *read_to_string()* и сохраняется в переменной *s*.

Если в результате выполнения команды был найден файл "*test/backupsript.sh*", то вызывается метод *update_ip()* для обновления *IP*-адреса удаленного хоста в скрипте. Это сделано для того, чтобы управляющая машина всегда знала *ip* сервера и могла обмениваться данными. Затем функция возвращает *HTTP*-ответ со статусом *Ok* и сообщением об успешном подключении.

Если файл *"test/backupscript.sh"* не был найден, то функция возвращает *HTTP*-ответ со статусом *Forbidden* и сообщением об ошибке. Нижепредставленкод.

```
#[post("")]
pub async fn connect(
    app_data: web::Data<AppData>,
    connect: Json<ConnectRequest>,
) -> HttpResponse {
    let tcp = TcpStream::connect(connect.host.clone());
    if tcp.is_err() {
        return HttpResponse::BadRequest().json("Failed
connection to host");
    }
    let mut sess = Session::new();
    if sess.is_err() {
        return HttpResponse::BadRequest().json("Failed
connection to host");
    }
    sess.as_mut().unwrap().set_tcp_stream(tcp.unwrap());
    if sess.as_mut().unwrap().handshake().is_err() {
        return HttpResponse::BadRequest().json("Failed
connection to host");
    }

    if
sess.as_mut().unwrap().userauth_password(&*connect.login,
&*connect.password).is_err() {
        return HttpResponse::BadRequest().json("Incorrect
password or login");
    }
    let mut channel =
sess.as_mut().unwrap().channel_session().unwrap();
    channel.exec("ls test/backupscript.sh");
    let mut s = String::new();
    channel.read_to_string(&mut s).unwrap();
    println!("{}", s);
    channel.wait_close();
    if s.find("test/backupscript.sh").is_some() {
        FileLogic::update_ip(connect.clone());
        return HttpResponse::Ok().json("Connect success");
    }else {
```

```

        return    HttpResponseMessage::Forbidden().json("Agent not
found");
    }
}

```

Запрос на соединение с управляющей машиной показан на рисунке 1.5.

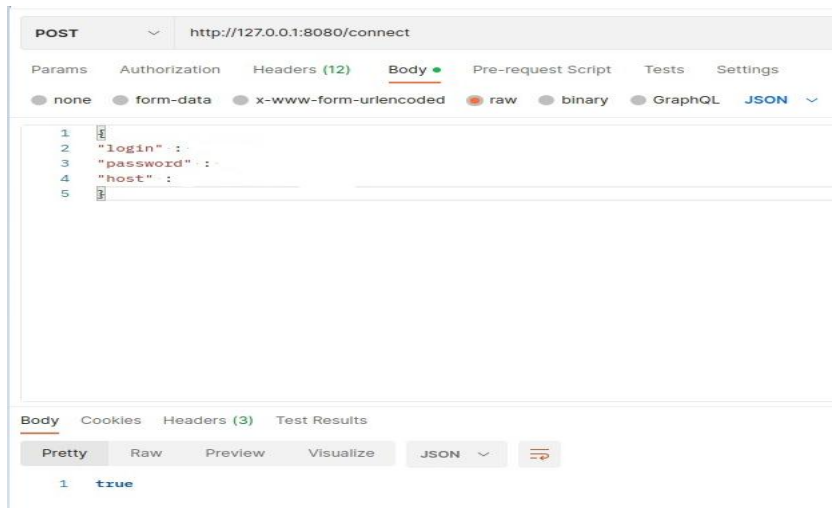


Рисунок 1.5 – Соединение с управляющей машиной

Далее рассмотрим функцию создания плана

Функция является обработчиком *POST*-запроса на создание нового плана. Она принимает два параметра:

- *app_data web::Data<AppData>* - объект, содержащий данные приложения, такие как пул соединений с базой данных;
- *plan1 Json<PlanRequest>* - *JSON*-объект, содержащий данные о новом плане.

Функция начинается с преобразования *JSON*-объекта в объект типа *Plan* с помощью метода *to_plan()*. Затем вызывается метод *create()* для сохранения нового плана в базе данных. После этого вызывается метод *create()* для создания задачи *cron*, которая будет выполняться в соответствии с расписанием нового плана.

Затем вызывается метод *find()* для поиска только что созданного плана в базе данных. Результат поиска преобразуется в объект типа *PlanResponse* и возвращается клиенту в виде *JSON*-объекта с помощью метода *HttpResponse::Ok().json()*. Код функции представлен ниже

```
#[post("")]
pub async fn create(
    app_data: web::Data<AppData>,
    plan1: Json<PlanRequest>, ) -> HttpResponse {
    let new_plan1 = plan1.to_plan();
    PlanDao::create(app_data.pool.clone(), new_plan1.clone());
    PlanCron::create(plan1.0);
    let res = PlanDao::find(app_data.pool.clone(),
new_plan1.uuid);
    HttpResponse::Ok().json(PlanResponse::from_plan(res[0].clone()))
}
```

Запрос на добавление времени в расписание представлен на рисунке 1.6.

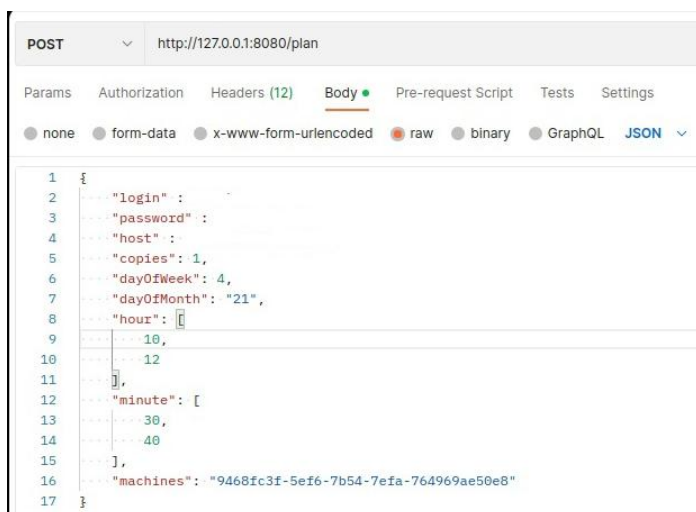


Рисунок 1.6— Добавление времени в расписание

Ответ на добавление времени в расписание представлен на рисунке 1.7.

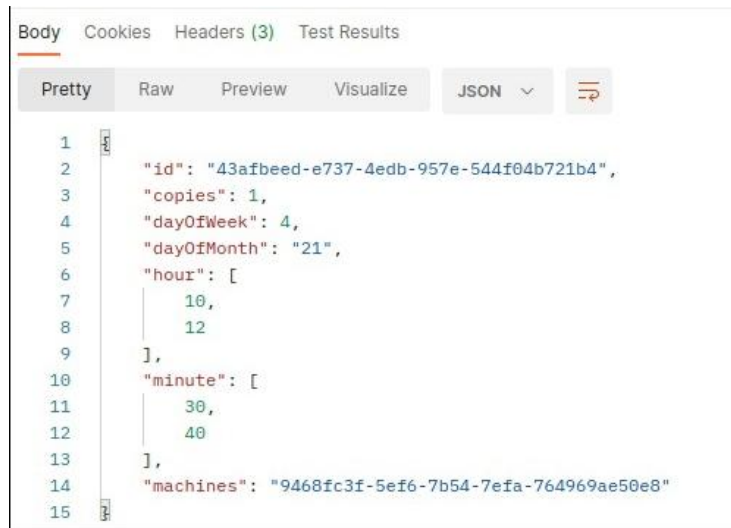


Рисунок 1.7– Ответ на добавление времени

Результаты выполнения запроса представлены на рисунке 1.8 и рисунке 1.9.

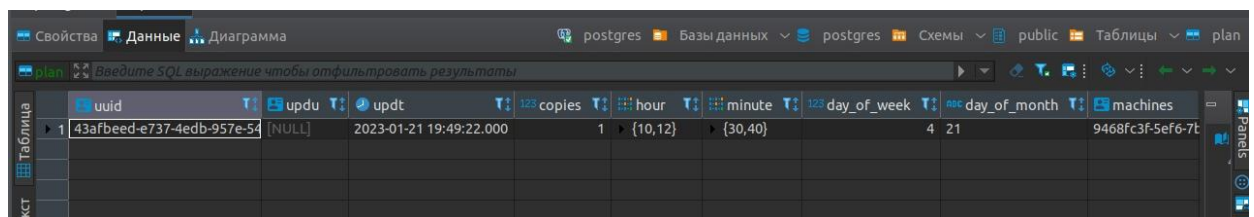


Рисунок 1.8– Запись в базе данных

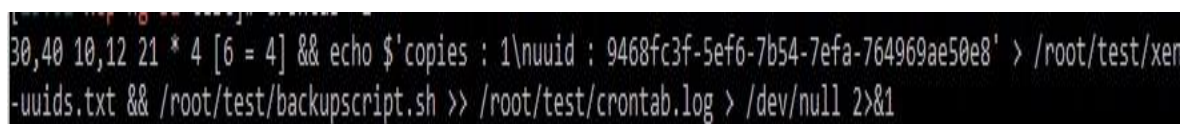


Рисунок 1.9– Запись в crontаблице

Далее рассмотрим функцию изменения плана резервного копирования для машины

Функция `update()` является обработчиком *PUT*-запроса для обновления информации о плане. Она принимает два параметра:

- `app_data web::Data<AppData>` - объект, содержащий данные приложения, такие как пул соединений с базой данных;

– *updated_plan1Json<PlanRequest>* – JSON-объект, содержащий обновленные данные для плана.

Функция начинается с проверки наличия *id* в JSON-объекте. Если *id* отсутствует, функция возвращает *HTTP*-ответ со статусом *BadRequest* и сообщением об ошибке.

Затем создается новый объект плана на основе обновленных данных с помощью метода *to_plan()* типа *PlanRequest*. Далее происходит поиск реального плана в базе данных с помощью метода *find()* типа *PlanDao*.

Затем создается объект *ConnectRequest* на основе данных из JSON-объекта *updated_plan1*. Этот объект будет использоваться для подключения к удаленному хосту.

Вызывается метод *delete()* типа *PlanCron* для удаления задачи *cron* для старого плана. Затем вызывается метод *create()* типа *PlanCron* для создания новой задачи *cron* для обновленного плана.

Если план не был найден в базе данных, функция возвращает *HTTP*-ответ со статусом *NotFound* и сообщением об ошибке.

Затем вызывается метод *update()* типа *PlanDao* для обновления информации о плане в базе данных.

Затем происходит поиск обновленного плана в базе данных с помощью метода *find()* класса *PlanDao*. Результат поиска преобразуется в объект *PlanResponse* с помощью метода *from_plan()* типа *PlanResponse*.

Функция возвращает *HTTP*-ответ со статусом *Ok* и объектом *PlanResponse*.

Код и результат представлен ниже

```
#[put("")]
pub async fn update(
    app_data: web::Data<AppData>,
    updated_plan1: Json<PlanRequest>,
) -> HttpResponse {
    if updated_plan1.id.is_none() {
        return HttpResponse::BadRequest().json(
            "field id shouldn't be null!"
        );
    }
    let plan1_to_update = updated_plan1.to_plan();
```

```

        let real_plan1 = PlanDao::find(app_data.pool.clone(),
updated_plan1.id.unwrap());
        let conn = ConnectRequest {
            login: updated_plan1.login.clone(),
            password: updated_plan1.password.clone(),
            host: updated_plan1.host.clone()
        };
        PlanCron::delete(real_plan1[0].clone(), conn);
        PlanCron::create(updated_plan1.0);
        if real_plan1.len() < 1 {
            return HttpResponse::NotFound()
                .json("Plan not found!");
        }

        PlanDao::update(
            app_data.pool.clone(),
            real_plan1[0].clone(),
            plan1_to_update.clone(),
        );
        let res = PlanDao::find(app_data.pool.clone(),
plan1_to_update.uuid);

HttpResponse::Ok().json(PlanResponse::from_plan(res[0].clone()))
    }

```

Запрос на изменение количества копий в расписании представлен на рисунке 1.10.

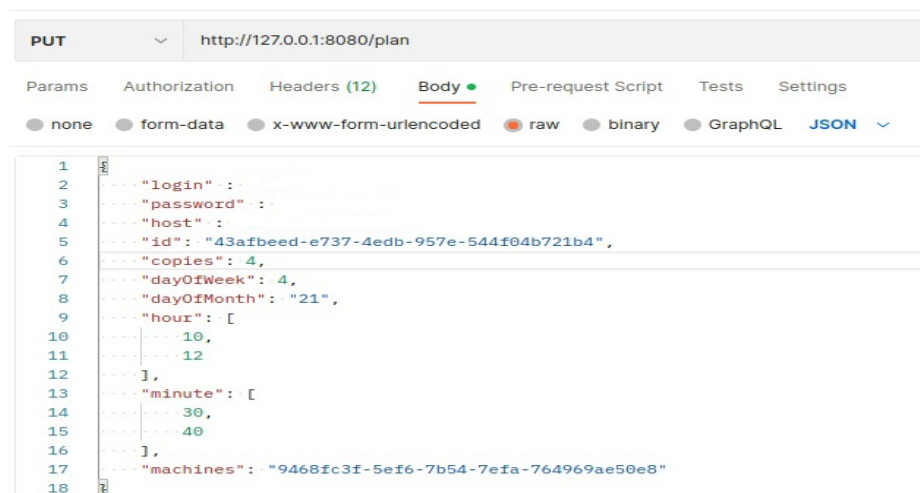


Рисунок 1.10– Изменение количества копий

Ответ на запрос об количества копий в расписании представлен на рисунке 1.11

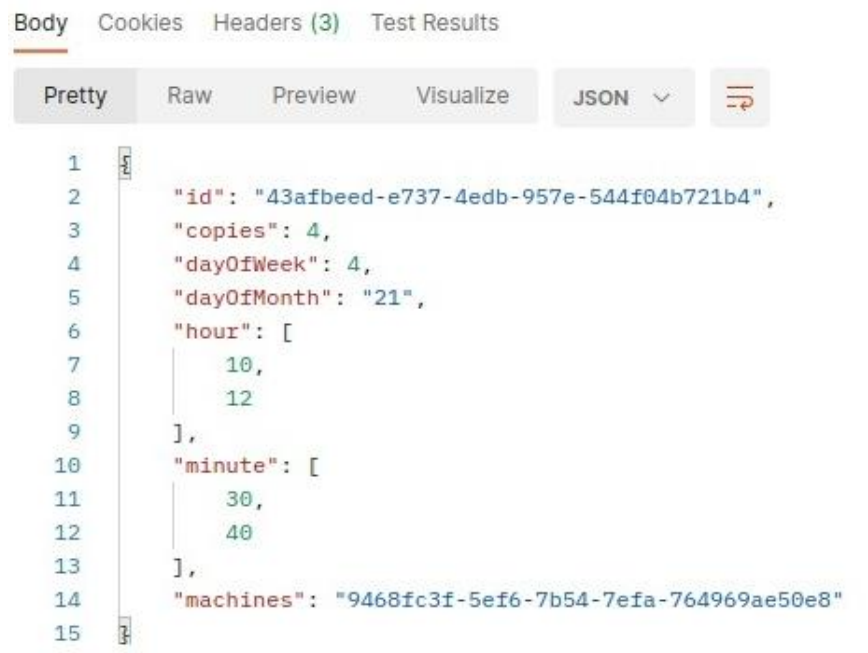


Рисунок 1.11– Ответ на запрос об изменении количества копий

Результаты изменения количества копий в *cron* в базе данных представлен на рисунке 1.12 и рисунке 1.13

```
30,40 10,12 21 * 4 [6 = 4] && echo $'copies : 4\nuuid : 9468fc3f-5ef6-7b54-7efa-764969ae50e8' > /root/test/xen-uids.txt && /root/test/backupscrip.sh >> /root/test/crontab.log > /dev/null 2>&1
```

Рисунок 1.12– Изменение количества копий в *cron*

uuid	updu	updt	copies	hour	minute	day_of_week	day_of_month	machines
43afbeed-e737-4edb-957e-54	[NULL]	2023-01-21 19:59:11.000	4	{10,12}	{30,40}	4	21	9468fc3f-5ef6-7t

Рисунок 1.13– Изменение количества копий в базе данных

Рассмотрим функцию удаления плана

Функция *delete()* является обработчиком *HTTPDELETE*-запроса, который принимает параметры *app_data*, *req* и *connect*. Параметр *app_data* содержит данные приложения, такие как пул подключений к базе данных. Параметр *req* содержит информацию о запросе, а параметр *connect* содержит данные для подключения к базе данных.

Внутри функции извлекается *id* из параметра *req* с помощью метода *match_info().get("id")*, который возвращает значение из маршрута запроса. Затем

вызывается метод *PlanDao::find()*, который находит план по *id* в базе данных и возвращает его. Полученный план передается в метод *PlanCron::delete()*, который удаляет связанные задачи *cron*. Затем вызывается метод *PlanDao::delete()*, который удаляет план из базы данных по *id*.

Если удаление прошло успешно, функция возвращает статус 200 и сообщение "200". Если удаление не удалось, функция возвращает статус 400 и сообщение "*Temporary exception! Please, try later*". Код и результат представлен ниже.

```
#[delete("{id}")]
pub async fn delete(app_data: web::Data<AppData>, req:
HttpRequest, connect : Json<ConnectRequest>) -> HttpResponse {
    let id: Uuid =
req.match_info().get("id").unwrap().parse().unwrap();
    let planls = PlanDao::find(app_data.pool.clone(), id);
    PlanCron::delete(planls[0].clone(), connect.0);
    let res = PlanDao::delete(app_data.pool.clone(), id);
    if res == 0 {
        return HttpResponse::BadRequest().json(
            "Temporary exception! Please, try later",
        );
    }
    HttpResponse::Ok().json("200")
}
```

Запрос на удаление времени из расписания и положительный ответ представлен на рисунке 1.14

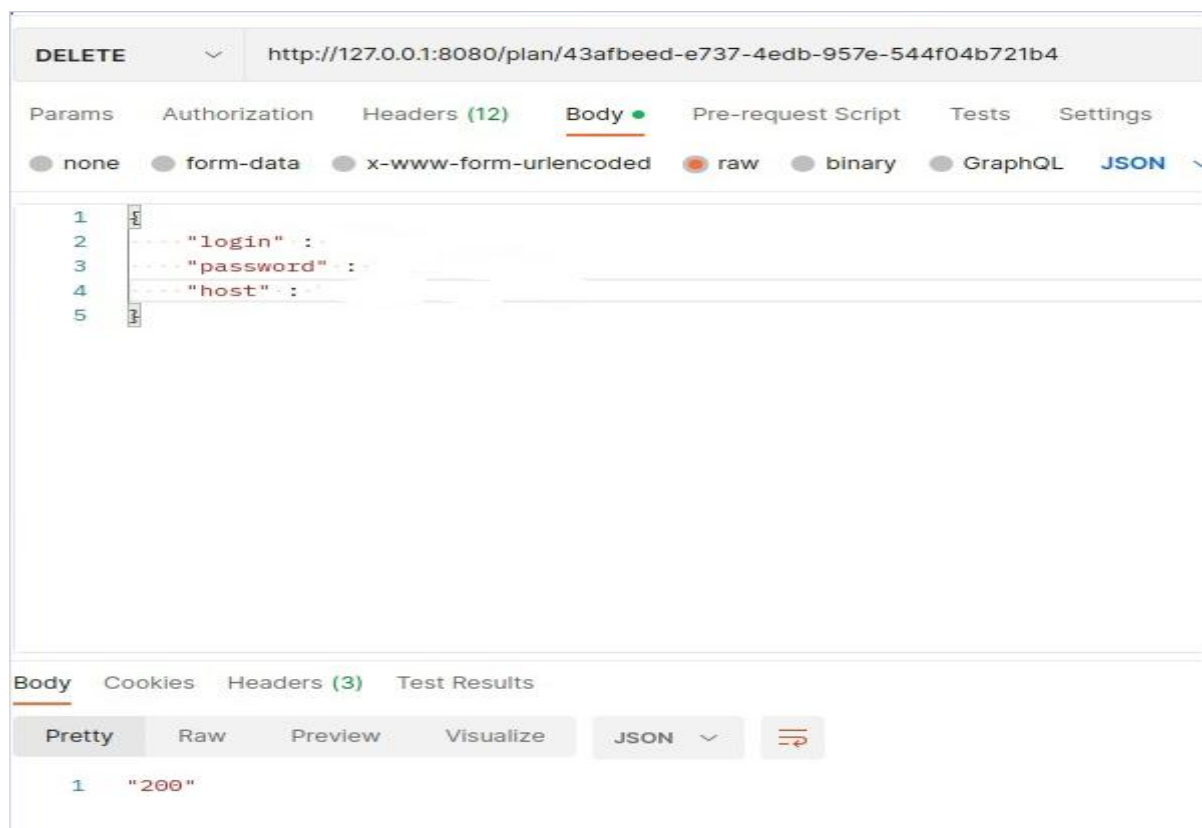


Рисунок 1.14–Запрос удаления времени из расписания

Результаты удаления времени из расписания представлен на рисунке 1.15 и рисунке 1.16.

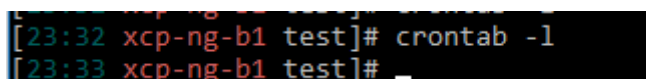


Рисунок 1.15– Результат удаления времени из cron

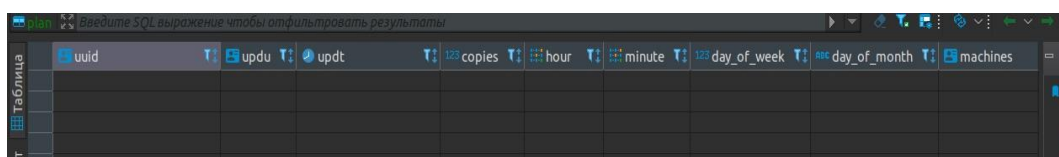


Рисунок 1.16– Результат удаления времени из базы данных

Рассмотрим функцию поиска плана

Функция *find()* является обработчиком *HTTP GET*-запроса, который принимает параметры *app_data* и *req*. Параметр *app_data* содержит данные приложения, такие как пул подключений к базе данных. Параметр *req* содержит информацию о запросе.

Внутри функции извлекается *id* из параметра *req* с помощью метода *match_info().get("id")*, который возвращает значение из маршрута запроса. Затем вызывается метод *PlanDao::find()*, который находит план по *id* в базе данных и возвращает его.

Полученный план передается в метод *PlanResponse::from_plan()*, который преобразует план в объект *PlanResponse*, содержащий информацию о плане.

Функция возвращает статус 200 и объект *PlanResponse* в формате *JSON* с помощью метода *HttpResponse::Ok().json()*. Если план не найден, функция возвращает статус 404 и сообщение "*Plan not found*". Код и результат представлен ниже.

```
#[get("{id}")]
pub async fn find(app_data: web::Data<AppData>, req: HttpRequest)
-> HttpResponse {
    let id: Uuid =
req.match_info().get("id").unwrap().parse().unwrap();

    let plans = PlanDao::find(app_data.pool.clone(), id);

    HttpResponse::Ok().json(PlanResponse::from_plan(plans[0].clone
()))
}
```

Запрос на получение времени из расписания из базы данных представлен на рисунке 1.17

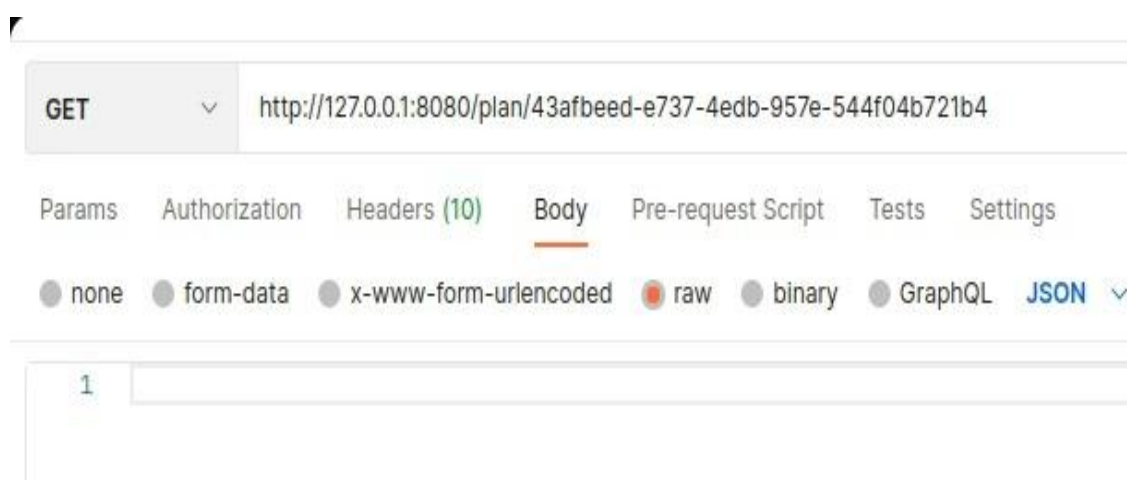


Рисунок 1.17– Запрос времени из базы данных

Ответ на запрос о получении времени из расписания представлен на рисунке 1.18.

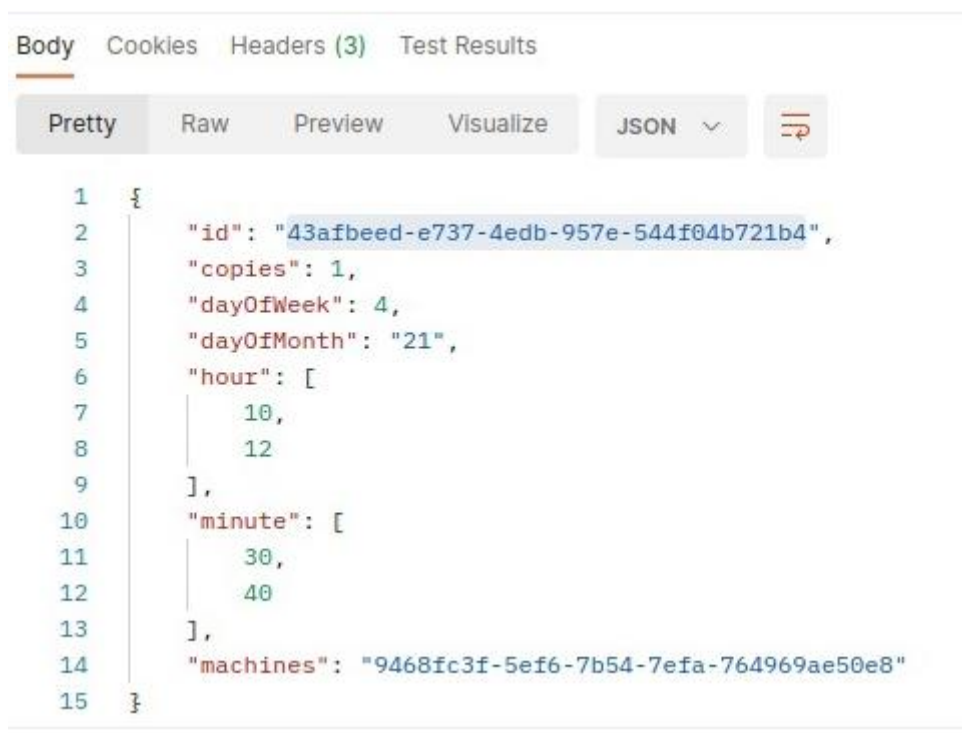


Рисунок 1.18— Ответ на запрос о получении времени из базы данных

Рассмотрим функцию запуска резервного копирования

Функция представляет собой обработчик *POST* запроса по определенному маршруту, содержащему идентификатор. Функция принимает два параметра: *connect* - объект типа *Json*, содержащий данные для подключения к удаленному хосту, и *req* - объект типа *HttpRequest*, содержащий информацию о запросе.

Внутри функции происходит установка соединения с удаленным хостом, проверка возможности установления соединения и аутентификации пользователя. После успешной аутентификации выполняются команды на удаленном хосте для создания резервной копии данных.

Функция возвращает объект типа *HttpResponse*, который содержит ответ сервера на запрос. В случае успешного выполнения операции возвращается статус 200 *OK* и сообщение "*Backup run*". В случае ошибки возвращается статус 400 *Bad Request* и соответствующее сообщение об ошибке.

Коди результат выполнения представлены ниже.

```
#[post("{id}")]
pub async fn backup(
    connect: Json<ConnectRequest>,
    req: HttpRequest
) -> HttpResponse {
    let id: Uuid =
req.match_info().get("id").unwrap().parse().unwrap();
    let tcp = TcpStream::connect(connect.host.clone());
    if tcp.is_err() {
        return HttpResponse::BadRequest().json("Failed
connection to host");
    }
    let mut sess = Session::new();
    if sess.is_err() {
        return HttpResponse::BadRequest().json("Failed
connection to host");
    }
    sess.as_mut().unwrap().set_tcp_stream(tcp.unwrap());
    if sess.as_mut().unwrap().handshake().is_err() {
        return HttpResponse::BadRequest().json("Failed
connection to host");
    }

    if
sess.as_mut().unwrap().userauth_password(&*connect.login,
&*connect.password).is_err() {
        return HttpResponse::BadRequest().json("Incorrect
password or login");
    }
    let mut channel =
sess.as_mut().unwrap().channel_session().unwrap();
    channel.exec(&*format!("echo $('copies : 0\\nuuid : {}' >
/root/test/xen-uuids.txt", id));
    channel.wait_close();
    let mut channel1 =
sess.as_mut().unwrap().channel_session().unwrap();
    channel1.exec("/root/test/backupscrip.sh 1").unwrap();
    sleep(Duration::new(10,0));
    channel1.close();
    HttpResponse::Ok().json("Backup run")
}
```

Ответ на запрос о запуске резервного копирования представлен на рисунке 1.19

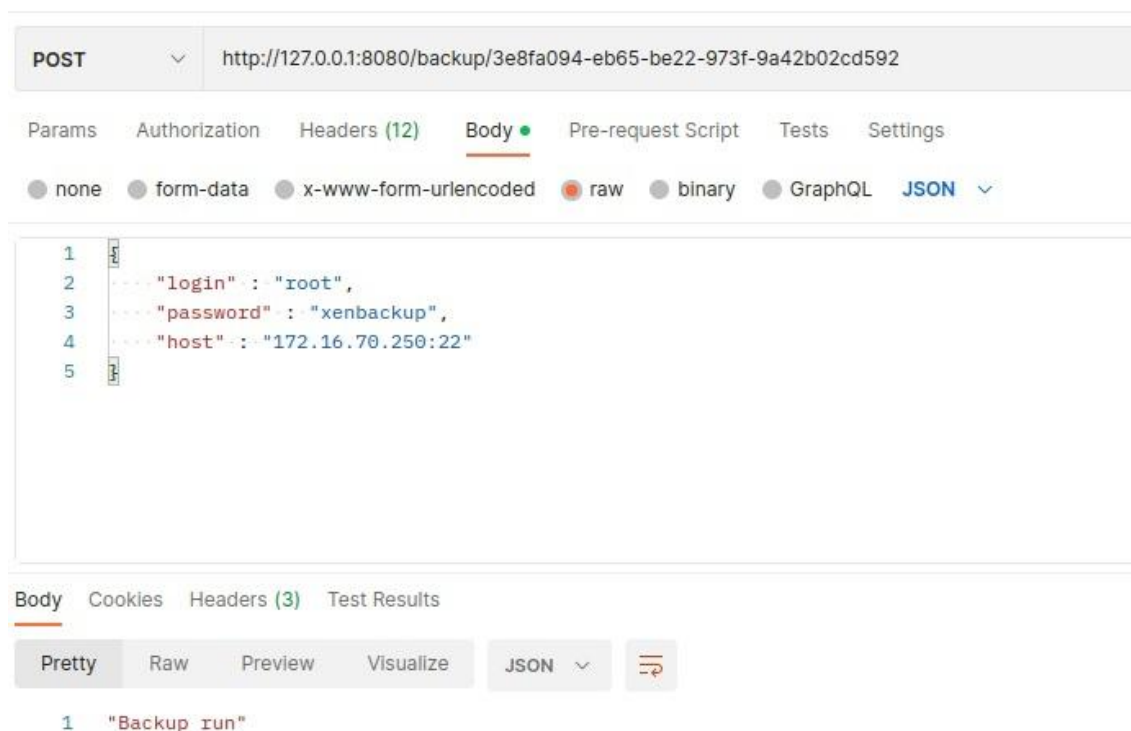


Рисунок 1.19 – Запуск резервного копирования

Рассмотрим функцию восстановление машины

Функция является обработчиком *POST*-запроса и принимает два параметра *app_data* и *connect*.

Первым делом функция пытается установить соединение с машиной по протоколу *TCP*, используя данные, переданные в параметре *connect*. Если соединение не установлено, то функция возвращает *HTTP*-ответ со статусом ошибки 400 и сообщением "*Failed connection to host*".

Далее выполняется команда на удаленной машине для восстановления данных. Команда формируется на основе данных, переданных в параметре *connect*. Если выполнение команды не удалось, то функция возвращает *HTTP*-ответ со статусом ошибки 500 и сообщением "*Machine not recovery*".

После выполнения команды происходит обработка полученного результата. Если результат содержит идентификатор восстановленной машины,

то происходит обновление информации о плане резервного копирования. Если идентификатор не найден, то функция возвращает *HTTP*-ответ со статусом ошибки 500 и сообщением "*Machine not recovery*".

В конце функция возвращает *HTTP*-ответ со статусом 200 и сообщением "200".

Код функции и результат выполнения запроса представлены ниже.

```
#[post("recovery")]
pub async fn recovery_machine(
    app_data: web::Data<AppData>,
    connect: Json<RecoveryRequest>,
) -> HttpResponse {
    // let res = MachineDao::get_machine(connect.0);
    let tcp = TcpStream::connect(connect.host.clone());
    if tcp.is_err() {
        return HttpResponse::BadRequest().json("Failed
connection to host");
    }
    let mut sess = Session::new();
    if sess.is_err() {
        return HttpResponse::BadRequest().json("Failed
connection to host");
    }
    sess.as_mut().unwrap().set_tcp_stream(tcp.unwrap());
    if sess.as_mut().unwrap().handshake().is_err() {
        return HttpResponse::BadRequest().json("Failed
connection to host");
    }

    if
sess.as_mut().unwrap().userauth_password(&*connect.login,
&*connect.password).is_err() {
        return HttpResponse::BadRequest().json("Incorrect
password or login");
    }
    let mut channel =
sess.as_mut().unwrap().channel_session().unwrap();
    channel.exec(&*format!("/root/test/backupscript.sh 0 {} {}
{}", connect.id.clone(), connect.path.clone(), connect.name.clone())
).unwrap();
    let mut s = String::new();
```

```

        channel.read_to_string(&mut s).unwrap();
        channel.wait_close();
        let mut line = s.lines();
        if line.clone().last().is_some() {
            let id = Uuid::parse_str(line.last().unwrap());
            if id.is_ok() {
                let plan =
PlanDao::find_by_machine(app_data.pool.clone(), connect.id.clone());
                if plan.len() > 0 {
                    let mut plan1_to_update = plan[0].clone();
                    plan1_to_update.machines = id.unwrap();
                    let conn = ConnectRequest {
                        login: connect.login.clone(),
                        password: connect.password.clone(),
                        host: connect.host.clone()
                    };
                    PlanCron::delete(plan[0].clone(), conn);
                    PlanCron::create(PlanRequest {
                        id:
Option::from(plan1_to_update.uuid.clone()),
                        copies: plan1_to_update.copies.clone(),
                        day_of_week:
plan1_to_update.day_of_week.clone(),
                        day_of_month:
plan1_to_update.day_of_month.clone(),
                        hour: plan1_to_update.hour.clone(),
                        minute: plan1_to_update.minute.clone(),
                        machines: connect.id.clone(),
                        login: connect.login.clone(),
                        password: connect.password.clone(),
                        host: connect.host.clone()
                    });
                    PlanDao::update(
                        app_data.pool.clone(),
                        plan[0].clone(),
                        plan1_to_update.clone(),
                    );
                }
            } else {
                return
HttpResponse::InternalServerError().json("Machine not recovery")
            }
        }
    }
}

```

```

    }else {
        return HttpResponseMessage::InternalServerError().json("Machine
not recovery")
    }
    // let plan =
    HttpResponseMessage::Ok().json("200")
}

```

Ответ на запрос о запуске восстановление машины представлен на рисунке 1.20

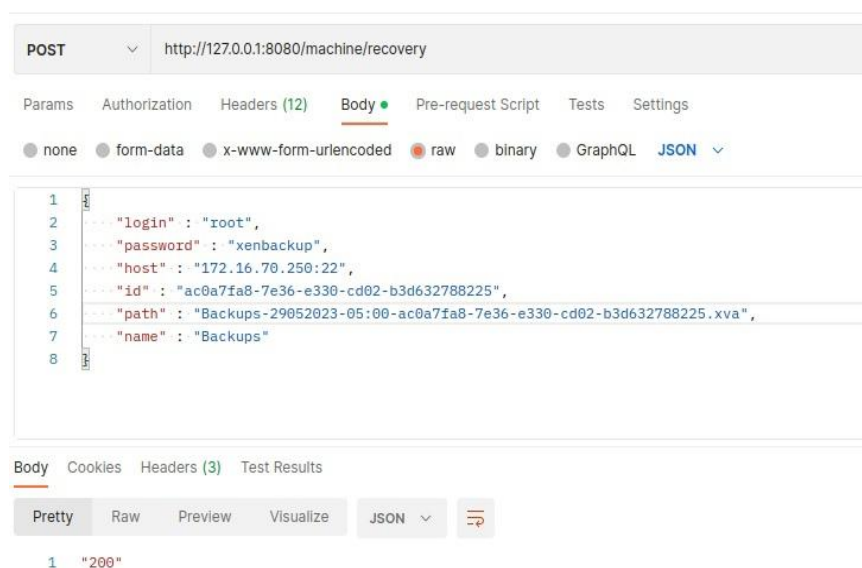


Рисунок 1.20 – Восстановление машины

Рассмотрим функцию запуска машины

Функция `start_machine` является обработчиком POST-запроса по адресу `/start/{id}`. Она принимает два параметра: `connect` и `req`.

Первым делом функция получает идентификатор машины из параметра `id`, который передается в адресной строке запроса. Затем функция устанавливает соединение с машиной по протоколу TCP, используя данные, переданные в параметре `connect`.

Затем функция выполняет команду на удаленной машине для запуска виртуальной машины с указанным идентификатором. Команда формируется на основе полученного идентификатора. Если выполнение команды не удалось, то функция выбрасывает исключение.

После выполнения команды функция получает результат выполнения и выводит его в консоль. Затем сессия закрывается.

В конце функция возвращает HTTP-ответ со статусом 200 и сообщением "VM started".

Если в процессе выполнения какой-либо из этих операций происходит ошибка, то функция выбрасывает исключение, которое обрабатывается внешним кодом и преобразуется в HTTP-ответ со статусом ошибки и соответствующим сообщением.

```
#[post("start/{id}")]  
pub async fn start_machine(  
    connect: Json<ConnectRequest>,  
    req: HttpRequest,  
    ) -> HttpResponse {  
    let id: Uuid =  
req.match_info().get("id").unwrap().parse().unwrap();  
    let conn = connect.0;  
  
    let tcp = TcpStream::connect(conn.host.clone()).unwrap();  
  
    let mut sess = Session::new().unwrap();  
    sess.set_tcp_stream(tcp);  
    sess.handshake().unwrap();  
    sess.userauth_password(&*conn.login, &*conn.password);  
    let mut channel = sess.channel_session().unwrap();  
    channel.exec(&*format!("xe vm-start uuid={}", id)).unwrap()  
    let mut s = String::new();  
    channel.read_to_string(&mut s).unwrap();  
    println!("{}", s);  
    channel.wait_close();  
  
    HttpResponse::Ok().json("VM started")  
}
```

На рисунке 1.21 представлено состояние запущенной машины

```
params= power-  
[18:05 xcp-ng-b1 ~]# xe vm-list uuid=bba326df-a7c9-8b30-72b7-315d2530182e params=power-state  
power-state ( R0) : running  
  
[18:05 xcp-ng-b1 ~]#
```

Рисунок 1.21 – Состояние машины

На рисунке 1.22 представлен ответ на запрос о запуске машины

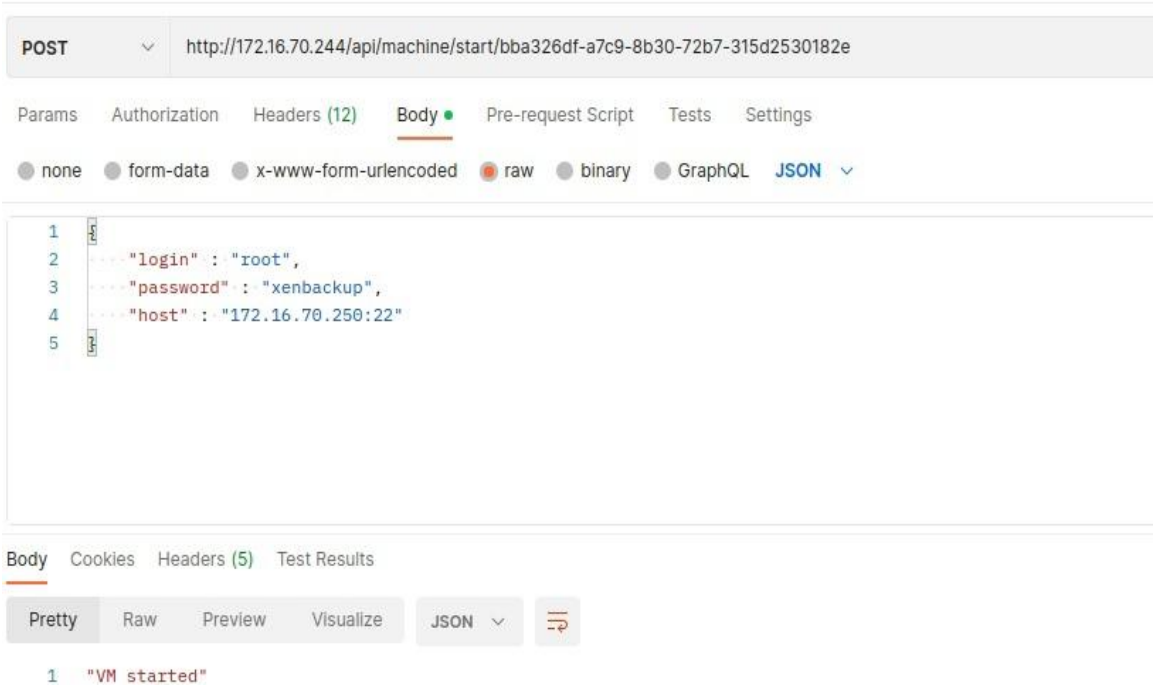


Рисунок 1.22 – Запуск машины

На рисунке 1.23 представлен ответ на запрос об остановки машины

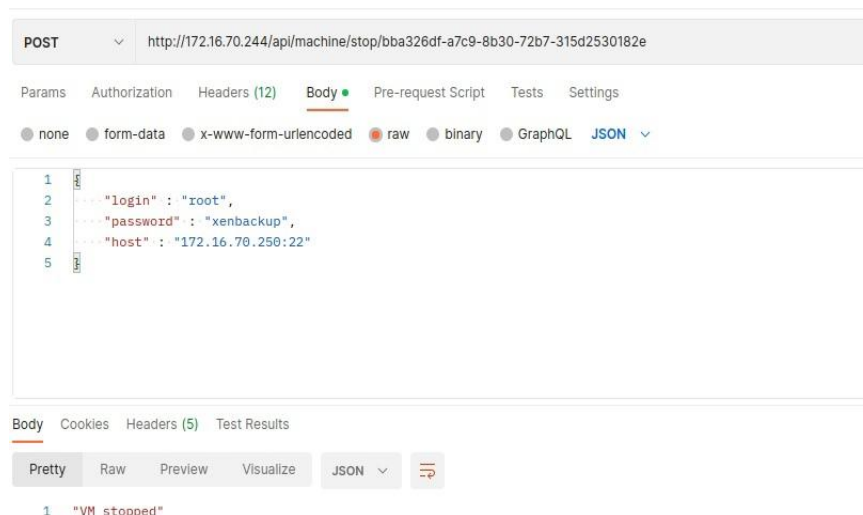


Рисунок 1.23 – Остановка машины

На рисунке 1.24 представлено состояние запущенной машины

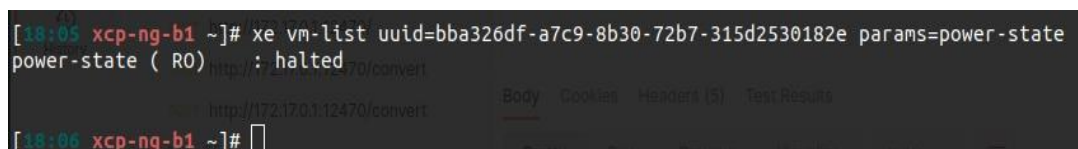


Рисунок 1.24 – Состояние машины

На рисунке 1.25 представлен ответ на запрос о перезагрузки машины

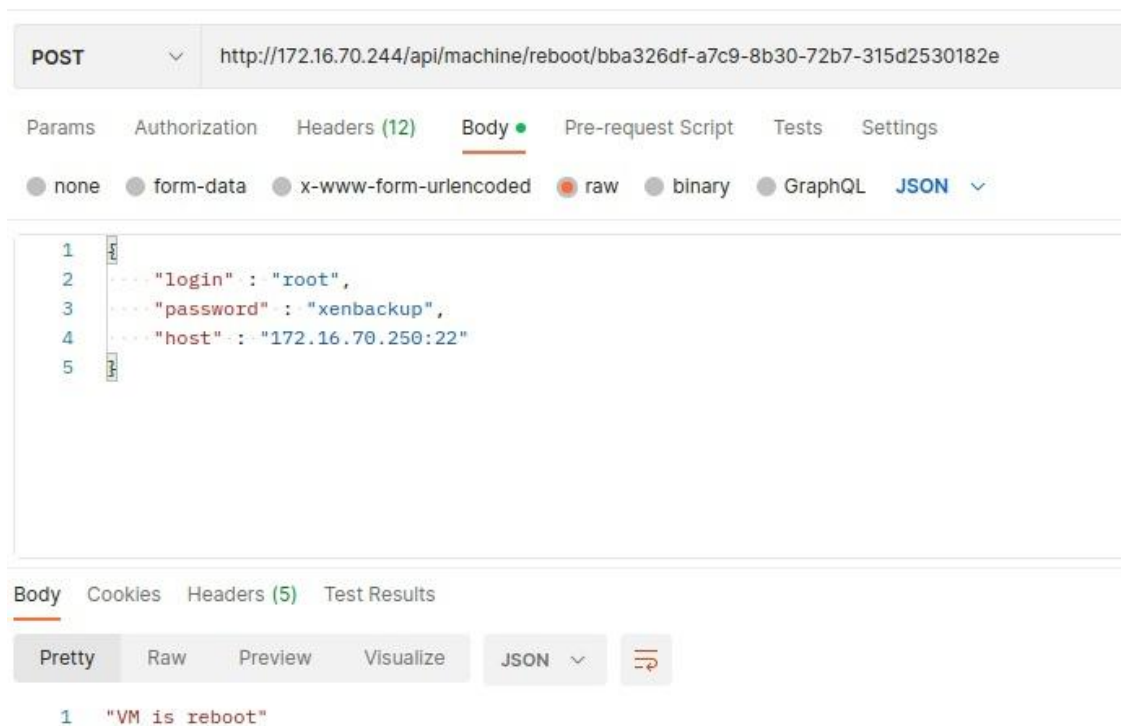


Рисунок 1.25 – Перезагрузка машины

1.7.2 Разработка скрипта

Для удобного построения планов резервного копирования был написан скрипт, для выполнения бэкапа достаточно выполнить этот скрипт, описание представлено ниже.

При передаче в скрипт аргумента 0 выполняется скрипт восстановления машины.

Скрипт создает точку монтирования, монтирует удаленный *cifs*-шар на локальную машину и проверяет, существует ли точка монтирования. Если точка монтирования не найдена, скрипт выдает ошибку и завершает работу.

Затем скрипт останавливает виртуальную машину с указанным *UUID* и удаляет ее.

Далее скрипт импортирует резервную копию виртуальной машины из указанного файла и присваивает ей новый *UUID*. Если операция импорта завершается успешно, то скрипт изменяет имя виртуальной машины на указанное в параметре *name-label* и переименовывает файлы с резервной копией, заменяя старый *UUID* на новый. Если же возникает ошибка, то скрипт отправляет сообщение об ошибке и завершает работу.

Код скрипта представлен ниже.

```
mkdir -p ${MOUNTPOINT}
[ ! -d ${MOUNTPOINT} ] && echo "No mount point found, kindly
check!": exit 1
mount -t cifs "//172.16.254.190/xcp" -o
username=${LOGIN},password=${PASSWD} ${MOUNTPOINT}
xe vm-shutdown uuid=$2
xe vm-destroy uuid=$2
UUIDNEW=$(xe vm-import filename="/mnt/backup/${XSNAME}/${3}" | tr -
d ' ')
if [ $? -ne 0 ]; then
echo "Error"
else
xe vm-param-set name-label=$4 uuid=${UUIDNEW}
for file in $(find /mnt/backup/xcp-ng-bl/ -type f -name
"*$2*"); do
mv "$file" "$(echo "$file" | sed "s/$2/$UUIDNEW/g")"
done
echo "${UUIDNEW}"
fi
umount ${MOUNTPOINT}
```

При передаче в скрипт аргумента 1 выполняется резервное копирование машины

Скрипт предназначен для создания резервной копии виртуальной машины в *XenServer* и ее экспорта на удаленный *cifs*-шар.

Первая часть скрипта проверяет, существует ли точка монтирования для удаленного *cifs*-шара. Если точка монтирования не найдена, то скрипт выдает ошибку и завершает работу. Затем скрипт монтирует удаленный *cifs*-шар на локальную машину.

Затем скрипт получает *UUID* виртуальной машины из файла *UUIDFILE* и ее имя. С помощью команды *xe vm-snapshot* создается снимок виртуальной машины.

Далее скрипт экспортирует виртуальную машину в файл формата *xva* на удаленный *cifs*-шар. Если операция экспорта завершается успешно, то скрипт отправляет сообщение об успешном выполнении операции на сервер. Если же возникает ошибка, то скрипт отправляет сообщение об ошибке.

Затем скрипт проверяет количество копий резервных копий виртуальной машины, указанных в файле *UUIDFILE*, и удаляет старые резервные копии, если их количество превышает указанное значение.

После завершения операции резервного копирования скрипт размонтирует удаленный *cifs*-шар и завершает работу. Кодскриптапредставленниже

```
[ ! -d ${MOUNTPOINT} ] && echo "No mount point found, kindly
check!": exit 1
mount -t cifs "//172.16.254.190/xcp" -o
username=${LOGIN},password=${PASSWD} ${MOUNTPOINT}
BACKUPPATH=${MOUNTPOINT}/${XSNAME}
mkdir -p ${BACKUPPATH}
[ ! -d ${BACKUPPATH} ] && echo "No backup directory found!":
exit 1
[ ! -f ${UUIDFILE} ] && echo "No UUID list file found!": exit 1
VMUUID=`cat $UUIDFILE | grep uuid | cut -d":" -f2 | sed 's/^
*//g'`
VMNAME=`xe vm-list uuid=$VMUUID | grep name-label | cut -d":" -
f2 | sed 's/^ *//g'`
echo "Backup vm ${VMNAME}..." >> ${LOG}
echo "Backup vm ${VMNAME}..."
```

```

        SNAPUUID=`xe vm-snapshot uuid=$VMUUID new-name-
label="$VMNAME-$BACKUPTIME"`
        echo "Snapshot vm ${VMNAME}..." >> ${LOG}
        echo "Snapshot vm ${VMNAME}..."
        xe template-param-set is-a-template=false ha-always-
run=false uuid=${SNAPUUID}
        echo "Export vm ${VMNAME}..." >> ${LOG}
        echo "Export vm ${VMNAME}..."
        TASKEEXEC=$(xe task-list)
        if [[ ! ${TASKEEXEC} ]]; then
            ERRORMSG=$(xe --noss1 vm-export vm=${SNAPUUID}
filename="$BACKUPPATH/$VMNAME-$BACKUPTIME-$VMUUID.xva" 2>&1)
            if [ $? -ne 0 ]; then
                echo "${VMNAME}-${ERRORMSG}" >> ${IMVMFILE}
                curl -X POST $BACKENDURL -H "Cache-Control: no-cache" -H
"Content-Type: application/json" -d '{ "machine" : "'$VMUUID'" ,
"message" : "'$ERRORMSG'" , "typeMessage" : "Error"}'
            else
                curl -X POST $BACKENDURL -H "Cache-Control: no-cache" -H
"Content-Type: application/json" -d '{ "machine" : "'$VMUUID'" ,
"message" : "Backup success" , "typeMessage" : "Success"}'
                echo "$VMNAME-$BACKUPTIME.xva" >> ${IMVMFILE}
                echo "Backup vm ${VMNAME} done." >> ${LOG}
                echo "Backup vm ${VMNAME} done."
                NUMCOPIES=`cat $UUIDFILE | grep copies | cut -d":" -f2 |
sed 's/^ *//g'`
                if [ $NUMCOPIES -ne 0 ]; then
                    NUMBACKUPS=`ls -Art $BACKUPPATH | grep "${VMNAME}-" | wc -
1`
                    for((i=0;i<(($NUMBACKUPS-$NUMCOPIES));i++)); do
                        DELFILE=`ls -Art $BACKUPPATH | grep "${VMNAME}-" |
head -n 1`
                        rm -f $BACKUPPATH/$DELFILE
                    done
                fi
            else
                curl -X POST $BACKENDURL -H "Cache-Control: no-cache" -H
"Content-Type: application/json" -d '{ "machine" : "'$VMUUID'" ,
"message" : "Another task executable!" , "typeMessage" : "Error"}'
                echo "Backup vm ${VMNAME} error." >> ${LOG}
            fi
            xe vm-uninstall uuid=${SNAPUUID} force=true
            ls $BACKUPPATH > /root/test/vmcopies.txt
            umount ${MOUNTPOINT}

```

1.7.3 Настройки прокси сервера

Для проксирования запросов был написан настройка для *nginx*, которая определяет прокси-сервер для приложения, работающего на порту 8080 на

локальном хосте. Все запросы, начинающиеся с */api/*, будут перенаправляться на этот прокси-сервер. Остальные запросы будут обрабатываться локальным сервером, который будет искать соответствующие файлы в корневой директории и, если не найдет, вернет страницу *index.html*. Эта настройка также указывает на отсутствие имени сервера (`_`), что означает, что эта конфигурация будет использоваться для любого имени сервера, которое не было явно указано в других настройках. Код представлен ниже

```
upstream apps_server01 {
    server 127.0.0.1:8080 fail_timeout=0;
}
server {
    listen 80 default_server;
    listen [::]:80 default_server;
    location /api/ {
        rewrite ^/api(.*)$ $1 break;
        proxy_pass http://apps_server01;
    }
    location / {
        try_files $uri /index.html;
    }
}
```

2 Безопасность жизнедеятельности

2.1 Правила безопасности при использовании ПК

Правила безопасности при использовании ПК должны опираться не только на защиту компьютера от вирусов и хакеров, но и на здоровье человека. Вот несколько правил, которые помогут сохранить здоровье при работе за компьютером:

- регулярно делайте перерывы. Длительная работа за компьютером может привести к усталости глаз, шеи, спины и запястий. Рекомендуется делать перерывы каждый час на 10-15 минут, чтобы размять мышцы и глаза;

- используйте эргономичное рабочее место. Клавиатура и мышь должны быть расположены на уровне локтей, монитор - на уровне глаз. Стул должен быть удобным и поддерживать правильную осанку;

- не работайте за компьютером в темноте. Освещение должно быть достаточным, чтобы не напрягать глаза;

- избегайте слишком ярких или мерцания экрана монитора. Это может вызвать головную боль и усталость глаз;

- используйте специальные программы для защиты глаз от усталости, например, *f.lux*;

- пейте достаточное количество воды. Работа за компьютером может привести к обезвоживанию, поэтому не забывайте пить воду;

- избегайте длительной работы за компьютером в одной и той же позе. Рекомендуется менять положение тела и делать упражнения для разминки мышц;

- не работайте за компьютером перед сном. Это может нарушить ваш сон и привести к бессоннице.

Соблюдение этих правил поможет сохранить здоровье при работе за компьютером и избежать неприятных последствий для здоровья.

Заключение

Результат выполнения данной выпускной квалификационной работы можно считать успешным, серверная часть системы Backup для кластеров *xrpg* реализована в соответствии с заданием.

Были выполнены следующие задачи:

- анализ аналогов. Были исследованы аналоги приложения. Это позволило определить функциональность и основные модули сервера;

- выбор языка программирования. Был выбран *Rust* в качестве языка программирования для разработки *web*сервера. *Rust* является безопасным языком с высокой надёжностью;

- выбор системы управления базами данных. Была выбрана *PostgreSQL* в качестве СУБД для хранения и управления данными. *PostgreSQL* является мощной и надёжной системой, обладающей широкими возможностями, включая поддержку сложных запросов и транзакций;

- разработка функционала системы. Были реализованы основные модули системы, такие как: создание планов резервного копирования, восстановление, остановка, старт и перезагрузка машины, отправка уведомлений о завершение резервного копирования, настройка точки монтирования.

В результате работы была создана сервер, который обеспечивает непрерывную работу приложения за счёт своей надёжности и стабильности. Также сервер обеспечивает высокую скорость обработки запросов и минимальное время ожидания ответа.

Перспективы развития сервера включают расширение функционала, добавление новых модулей. Разработанная система *Vaskir* для кластера *xsr-ng* готова к использованию и отвечает всем требованиям

Список используемых источников

1. Б. Кришнамурти, Дж. Рексфорд. Web-протоколы. Теория и практика. — М.: ЗАМО «Издательство БИНОМ», 2002г. — 592 с.
2. Язык программирования Rust [Электронный ресурс]. — URL: <https://doc.rust-lang.ru/book/>.
3. Actix web [Электронный ресурс]. — URL: <https://actix.rs/docs/>.
4. Дэн Бирк. "PostgreSQL: Современный язык запросов и администрирование". 2021.
5. Баррет Д. Linux - основные команды. Карманный справочник — М.; Кудицобраз, 2005

Приложение А

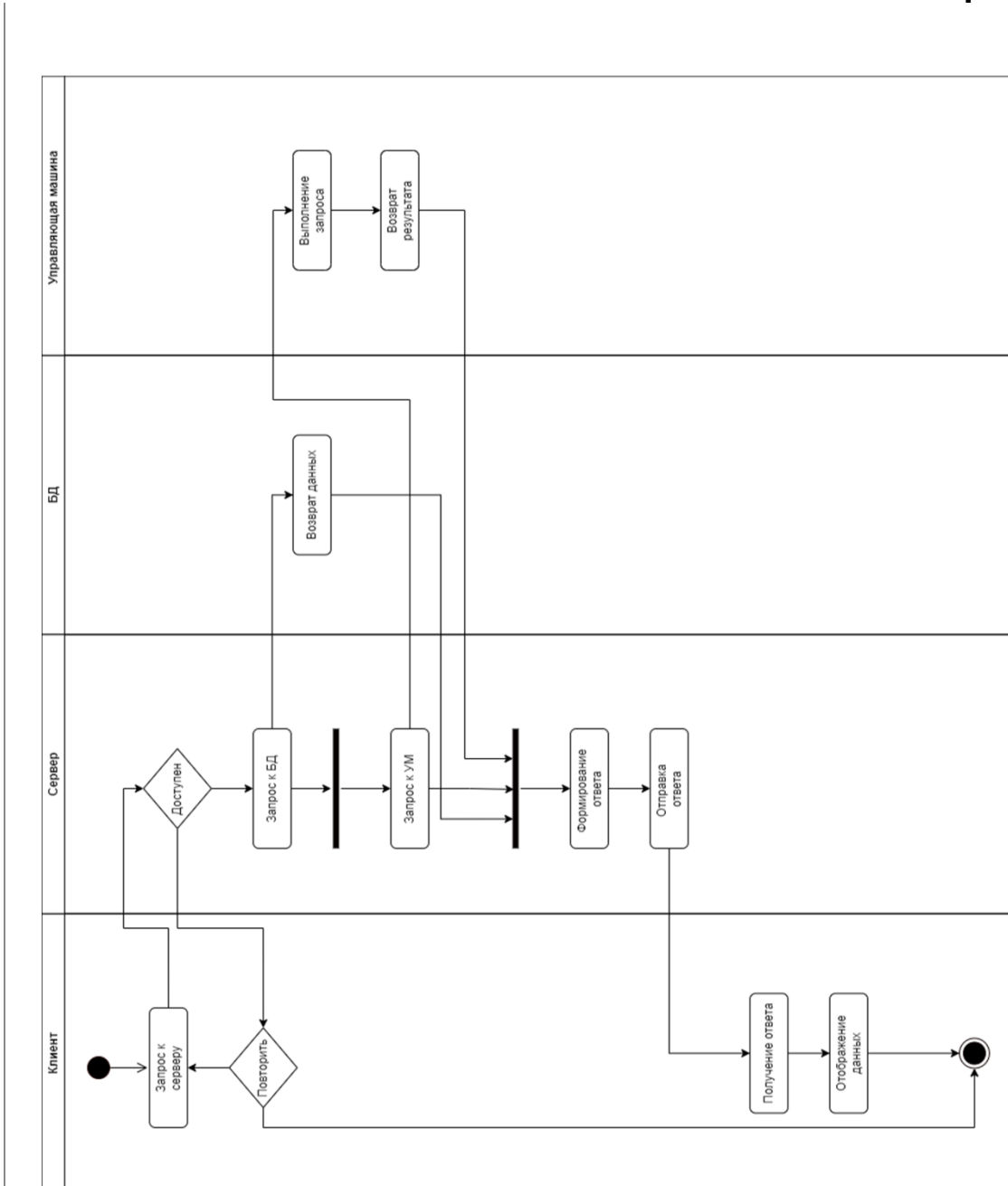
(Обязательное)

Система Ваксирдля кластеров хср-ng. Сервер.

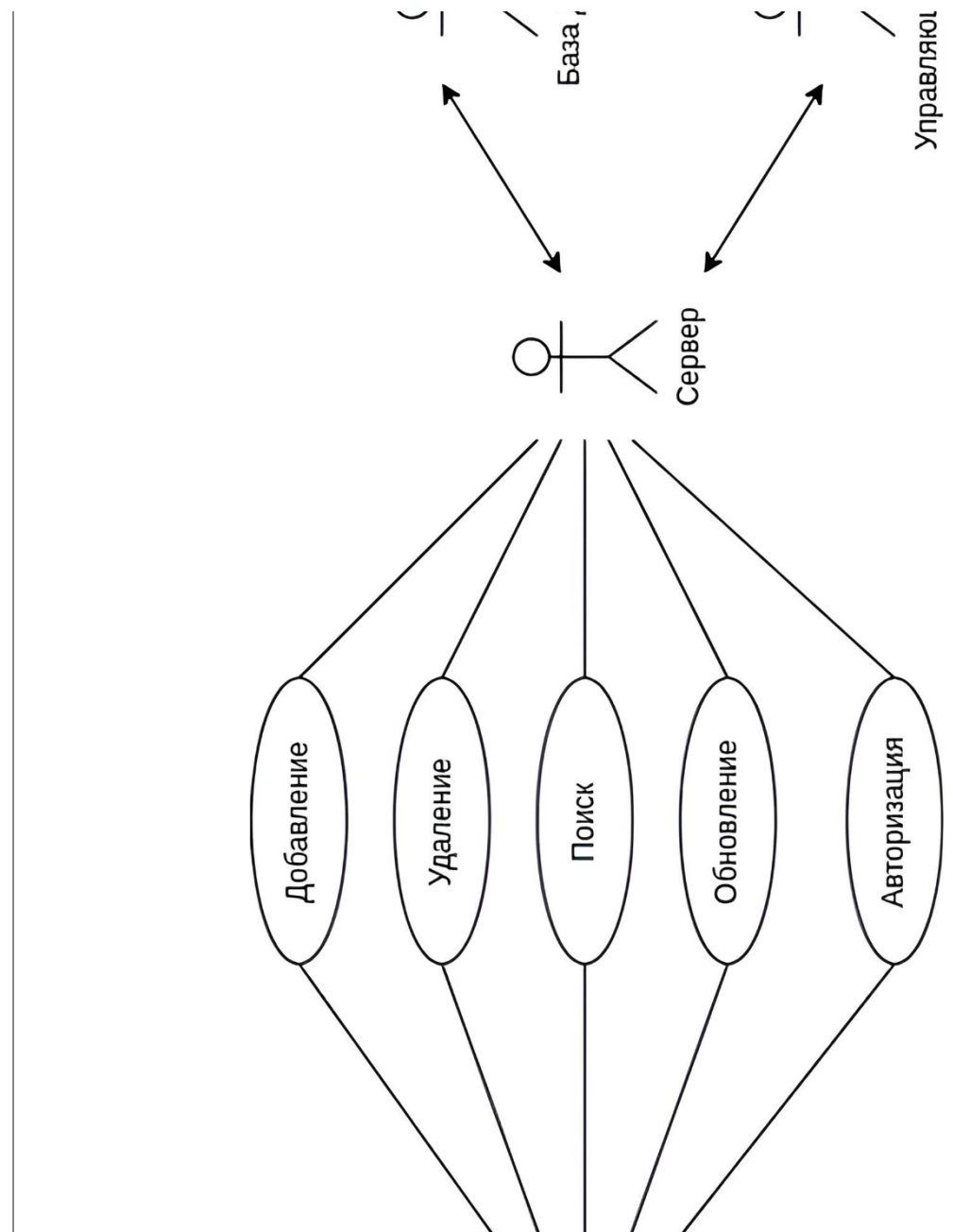
UML диаграммы

					ПГУ1.090301.35.001 ПЗ	Лист
						49
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А.1 – UML–диаграмма деятельности



Приложение А.2 – UML–диаграмма прецедентов



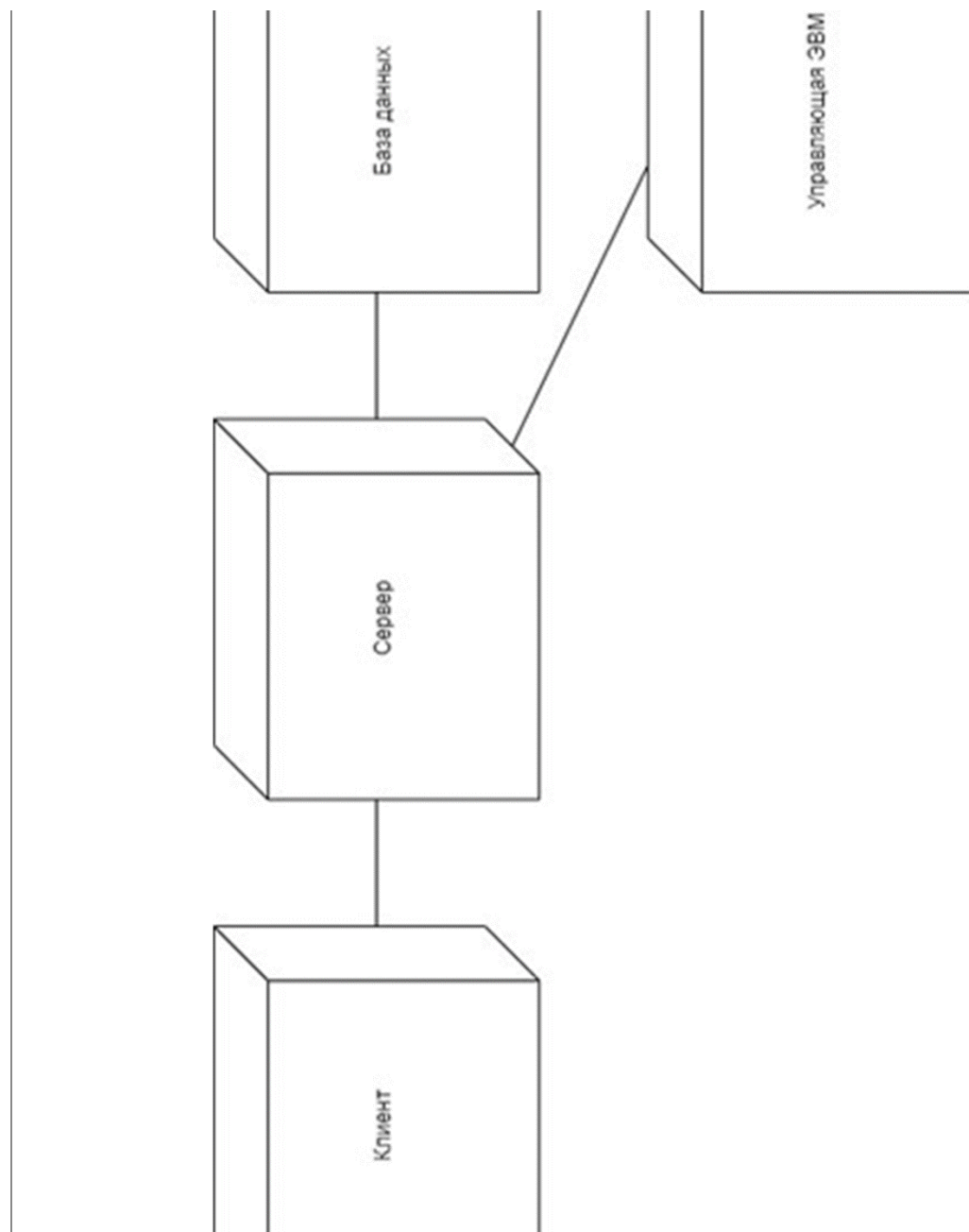
Приложение А.3 – UML–диаграммаразвёртывания

Изм.	Лист	№ докум.	Подпись	Дата

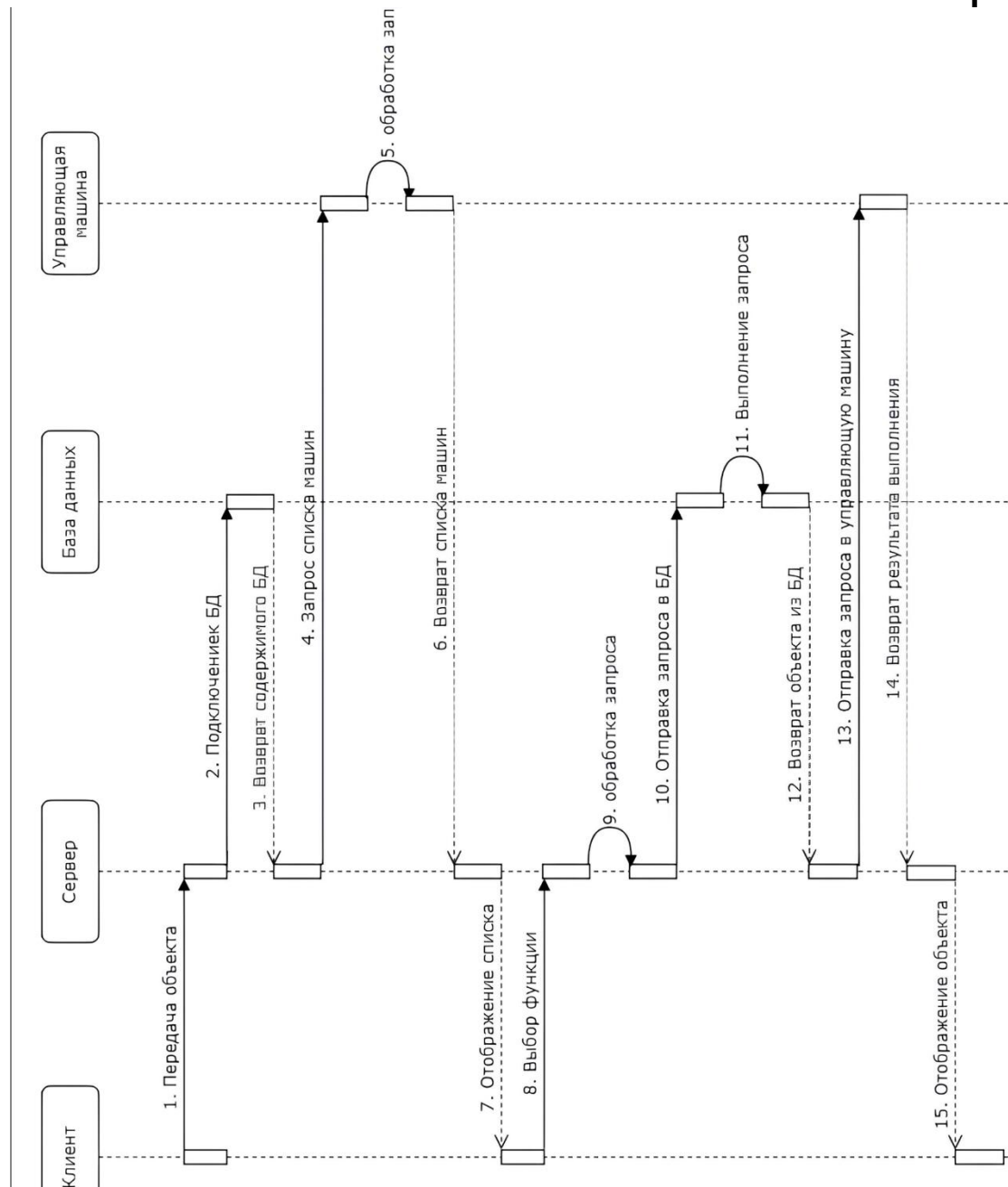
ПГУ1.090301.35.001 ПЗ

Лист

51



Приложение А.4 – UML–диаграмма последовательности



Приложение Б

(Обязательное)

Система Васкурд для кластеров хср-ng. Сервер.

Листинг программы

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.35.001 ПЗ

Лист

53

Файл «*notification.rs*»

```
use serde_json;
use serde::{Serialize, Deserialize};
use chrono::{NaiveDateTime, Utc};
use uuid::Uuid;
use crate::schema::notification;
#[derive(
Clone,
Serialize,
Deserialize,
Debug,
PartialEq,
Queryable,
Insertable,
```

					ПГУ1.090301.35.001 ПЗ	Лист
						54
Изм.	Лист	№ докум.	Подпись	Дата		

```

AsChangeset,
Identifiable,
QueryableByName,
Default,
)]
#[primary_key(uuid)]
#[table_name = "notification"]
pub struct Notification {
    pub uuid: Uuid,
    pub updt: Option<NaiveDateTime>,
    pub updu: Option<Uuid>,
    pub machine : Option<Uuid>,
    pub viewed : Option<bool>,
    pub message : Option<String>,
    pub type_message : Option<String>
}

```

Файл «*plan.rs*»

```

use serde_json;
use serde::{Serialize, Deserialize};
use chrono::{NaiveDateTime, Utc};
use uuid::Uuid;
use crate::schema::plan;
#[derive(
Clone,
Serialize,
Deserialize,
Debug,
PartialEq,
Queryable,
Insertable,
AsChangeset,
Identifiable,
QueryableByName,
Default,
)]
#[primary_key(uuid)]
#[table_name = "plan"]
pub struct Plan {
    pub uuid: Uuid,
    pub updt: Option<NaiveDateTime>,
    pub updu: Option<Uuid>,
    pub copies : i32,
    pub day_of_week : Option<i32>,
    pub day_of_month : Option<String>,
    pub hour : Vec<i32>,
    pub minute : Vec<i32>,
    pub machines : Uuid,
}

```

Файл «*configure_file_request.rs*»

```
use uuid::Uuid;
use serde_json;
use serde::{Serialize, Deserialize};
#[serde(rename_all = "camelCase")]
#[derive(Serialize, Deserialize, Debug, Clone)]
pub struct ConfigureFileRequest{
    pub login: String,
    pub password : String,
    pub host : String,
    pub mount_ip : Option<String>,
    pub mount_login : Option<String>,
    pub mount_password : Option<String>,
    pub server_ip : Option<String>,
}
```

Файл «*connect_request.rs*»

```
use uuid::Uuid;
use serde_json;
use serde::{Serialize, Deserialize};
#[serde(rename_all = "camelCase")]
#[derive(Serialize, Deserialize, Debug, Clone)]
pub struct ConnectRequest{
    pub login: String,
    pub password : String,
    pub host : String
}
```

Файл «*machine_request.rs*»

```
use uuid::Uuid;
use serde_json;
use serde::{Serialize, Deserialize};
#[serde(rename_all = "camelCase")]
#[derive(Serialize, Deserialize, Debug, Clone, Default)]
pub struct MachineRequest{
    pub uuid : String,
    pub name : String,
    pub state: String,
}
```

Файл «*notification_request.rs*»

```
use chrono::Utc;
use diesel::sql_types::Binary;
use uuid::Uuid;
use serde_json;
use serde::{Serialize, Deserialize};
use super::super::entity::notification::Notification;
use crate::schema::notification;

#[serde(rename_all = "camelCase")]
#[derive(Serialize, Deserialize, Debug, Clone)]
```

```

pub struct NotificationRequest {
    pub id: Option<Uuid>,
    pub machine: Option<Uuid>,
    pub message: Option<String>,
    pub viewed : Option<bool>,
    pub type_message : Option<String>,
}
impl NotificationRequest {
    pub fn to_notification(&self) -> Notification{
        Notification{
            uuid : if self.id.is_none() { Uuid::new_v4() } else
{ self.id.unwrap() },
            updt : Option::from(Utc::now().naive_local()),
            updu : None,
            machine: self.machine.clone(),
            viewed : if self.viewed.is_none() {
Option::from(false)} else { self.viewed},
            message : self.message.clone(),
            type_message : self.type_message.clone(),

        }
    }
}

```

Файл «notification_response.rs»

```

use chrono::Utc;
use uuid::Uuid;
use serde_json;
use serde::{Serialize, Deserialize};
use chrono::NaiveDateTime;
use diesel::sql_types::Binary;
use super::super::entity::notification::Notification;
use crate::schema::notification;
#[serde(rename_all = "camelCase")]
#[derive(Serialize, Deserialize, Debug, Clone)]
pub struct NotificationResponse{

    pub id: Uuid,
    #[serde(skip_serializing_if = "Option::is_none")]
    pub data : Option<NaiveDateTime>,
    #[serde(skip_serializing_if = "Option::is_none")]
    pub machine : Option<Uuid>,
    #[serde(skip_serializing_if = "Option::is_none")]
    pub viewed : Option<bool>,
    #[serde(skip_serializing_if = "Option::is_none")]
    pub message : Option<String>,
    #[serde(skip_serializing_if = "Option::is_none")]
    pub type_message : Option<String>

}
impl NotificationResponse{

```

```

    pub fn from_notification(plan : Notification) ->
NotificationResponse{
    NotificationResponse{
        id : plan.uuid,
        data : plan.updt,
        machine : plan.machine,
        viewed : plan.viewed,
        message : plan.message,
        type_message : plan.type_message
    }
}
    pub fn from_notification_list(notifications :
Vec<Notification>) -> Vec<NotificationResponse>{
    let mut res: Vec<NotificationResponse> = vec![];
    if notifications.is_empty() {
        return res;
    }
    for notification in notifications.iter() {

res.push(NotificationResponse::from_notification(notification.cl
one()));
    }
    res
}
}

```

Файл «*plan_request.rs*»

```

use chrono::Utc;
use diesel::sql_types::Binary;
use uuid::Uuid;
use serde_json;
use serde::{Serialize, Deserialize};
use super::super::entity::plan::Plan;
use crate::schema::plan;

#[serde(rename_all = "camelCase")]
#[derive(Serialize, Deserialize, Debug, Clone)]
pub struct PlanRequest{
    pub id: Option<Uuid>,
    pub copies : i32,
    pub day_of_week : Option<i32>,
    pub day_of_month : Option<String>,
    pub hour : Vec<i32>,
    pub minute : Vec<i32>,
    pub machines : Uuid,
    pub login: String,
    pub password : String,
    pub host : String
}

impl PlanRequest {
    pub fn to_plan(&self) -> Plan{

```

```

        Plan{
            uuid : if self.id.is_none() { Uuid::new_v4() } else
{ self.id.unwrap() },
            updt : Option::from(Utc::now().naive_local()),
            updu : None,
            copies: self.copies.clone(),
            day_of_week : self.day_of_week.clone(),
            day_of_month : self.day_of_month.clone(),
            hour: self.hour.clone(),
            minute: self.minute.clone(),
            machines: self.machines.clone()
        }
    }
}

```

Файл «*plan_response.rs*»

```

use diesel::sql_types::Binary;
use super::super::entity::plan::Plan;
use crate::schema::plan;
#[serde(rename_all = "camelCase")]
#[derive(Serialize, Deserialize, Debug, Clone)]
pub struct PlanResponse{

    pub id: Uuid,
    pub copies : i32,
    #[serde(skip_serializing_if = "Option::is_none")]
    pub day_of_week : Option<i32>,
    #[serde(skip_serializing_if = "Option::is_none")]
    pub day_of_month : Option<String>,
    pub hour : Vec<i32>,
    pub minute : Vec<i32>,
    pub machines : Uuid,

}

impl PlanResponse{
    pub fn from_plan(plan : Plan) -> PlanResponse{
        PlanResponse{
            id : plan.uuid,
            copies: plan.copies,
            day_of_week: plan.day_of_week,
            day_of_month : plan.day_of_month,
            hour: plan.hour,
            minute: plan.minute,
            machines: plan.machines
        }
    }

    pub fn from_plan_list(plans : Vec<Plan>) ->
Vec<PlanResponse>{
        let mut res: Vec<PlanResponse> = vec![];
        if plans.is_empty() {
            return res;
        }
    }
}

```

```

        for plan in plans.iter() {
            res.push(PlanResponse::from_plan(plan.clone()));
        }
    }
    res
}

```

Файл «recovery_request.rs»

```

use uuid::Uuid;
use serde_json;
use serde::{Serialize, Deserialize};
#[serde(rename_all = "camelCase")]
#[derive(Serialize, Deserialize, Debug, Clone)]
pub struct RecoveryRequest{
    pub login: String,
    pub password : String,
    pub host : String,
    pub id : Uuid,
    pub path : String,
    pub name : String
}

```

Файл «notification_dao.rs»

```

use r2d2::Pool;
use r2d2_diesel::ConnectionManager;
use diesel::{RunQueryDsl, QueryDsl, ExpressionMethods, PgConnection};
use crate::entity::notification::Notification;
use crate::schema::notification::dsl::notification;
use crate::schema;
use uuid::Uuid;
pub struct NotificationDao {}
impl NotificationDao{
    pub fn find_all(conn: &PgConnection) ->
    Result<Vec<Notification>, diesel::result::Error>{
        Ok(notification
            .load:::<Notification>(conn)
            .expect("Product not found!"))
    }
    pub fn find(pool: Pool<ConnectionManager<PgConnection>>, id:
    Uuid) -> Vec<Notification> {
        let connection = pool.get().unwrap();
        notification
            .filter(schema::notification::uuid.eq(id))
            .load:::<Notification>(&*connection)
            .expect("Product not found!")
        }
    pub fn create(pool: Pool<ConnectionManager<PgConnection>>,
    product1: Notification) -> use {
        let connection = pool.get().unwrap();

```

```

        diesel::insert_into(notification)
            .values(product1)
            .execute(&*connection)
            .unwrap()
    }
    pub fn update(
        pool: Pool<ConnectionManager<PgConnection>>,
        product1: Notification,
        updated_product1: Notification,
    ) -> usize {
        let connection = pool.get().unwrap();
        diesel::update(&product1)
            .set(updated_product1)
            .execute(&*connection)
            .unwrap()
    }

    pub fn delete(pool: Pool<ConnectionManager<PgConnection>>,
        product1_id: Uuid) -> usize {
        let connection = pool.get().unwrap();

        diesel::delete(notification.filter(schema::notification::uuid.eq
            (product1_id)))
            .execute(&*connection)
            .unwrap()
    }
}

```

Файл «*plan_dao.rs*»

```

use r2d2::Pool;
use r2d2_diesel::ConnectionManager;
use diesel::{RunQueryDsl, QueryDsl, ExpressionMethods,
PgConnection};
use crate::entity::plan::Plan;
use crate::schema::plan::dsl::plan;
use crate::schema;
use uuid::Uuid;
use crate::schema::plan::{day_of_month, day_of_week};

pub struct PlanDao {}
impl PlanDao{
    pub fn find_all(conn: &PgConnection) -> Result<Vec<Plan>,
diesel::result::Error>{

        Ok(plan
            .load:::<Plan>(conn)
            .expect("Product not found!"))
    }

    pub fn find(pool: Pool<ConnectionManager<PgConnection>>, id:
Uuid) -> Vec<Plan> {

```

```

        let connection = pool.get().unwrap();
        plan
            .filter(schema::plan::uuid.eq(id))
            .load::<Plan>(&*connection)
            .expect("Product not found!")
    }
    pub fn find_by_machine(pool:
Pool<ConnectionManager<PgConnection>>, machine: Uuid) ->
Vec<Plan> {
        let connection = pool.get().unwrap();
        plan
            .filter(schema::plan::machines.eq(machine))
            .load::<Plan>(&*connection)
            .expect("Product not found!")
    }
    pub fn create(pool: Pool<ConnectionManager<PgConnection>>,
product1: Plan) -> use {
        let connection = pool.get().unwrap();
        diesel::insert_into(plan)
            .values(product1)
            .execute(&*connection)
            .unwrap()
    }
    pub fn update(
        pool: Pool<ConnectionManager<PgConnection>>,
        product1: Plan,
        updated_product1: Plan,
    ) -> use {
        let connection = pool.get().unwrap();
        if updated_product1.day_of_month.is_none() {
            diesel::update(&product1).
                set(day_of_month.eq(None::<String>))
                    .execute(&*connection)
                    .unwrap();
        }
        if updated_product1.day_of_week.is_none() {
            diesel::update(&product1).
                set(day_of_week.eq(None::<i32>))
                    .execute(&*connection)
                    .unwrap();
        }
        diesel::update(&product1)
            .set(updated_product1)
            .execute(&*connection)
            .unwrap()
    }

    pub fn delete(pool: Pool<ConnectionManager<PgConnection>>,
product1_id: Uuid) -> use {
        let connection = pool.get().unwrap();

```

```

diesel::delete(plan.filter(schema::plan::uuid.eq(product1_id)))
    .execute(&*connection)
    .unwrap()
}
}

```

Файл «*plan_cron.rs*»

```

use std::net::TcpStream;
use ssh2::Session;
use crate::dto::connect_request::ConnectRequest;
use crate::dto::plan_request::PlanRequest;
use crate::entity::plan::Plan;

pub struct PlanCron {}

impl PlanCron {
    pub fn create(plan: PlanRequest) {
        let tcp =
        TcpStream::connect(plan.host.clone()).unwrap();

        let mut sess = Session::new().unwrap();
        sess.set_tcp_stream(tcp);
        sess.handshake().unwrap();

        sess.userauth_password(&*plan.login, &*plan.password);

        let mut channel = sess.channel_session().unwrap();
        let mut min = String::new();
        for (b, i) in plan.minute.iter().enumerate() {
            if b != plan.minute.len()-1 {
                min.push_str(&*format!("{}", i));
            } else {
                min.push_str(&*format!("{}", i));
            }
        }
        let mut h = String::new();
        for (b, i) in plan.hour.iter().enumerate() {
            if b != plan.hour.len()-1 {
                h.push_str(&*format!("{}", i));
            } else {
                h.push_str(&*format!("{}", i));
            }
        }
        let mut day_of_month = String::new();
        let mut day_of_week = String::new();

        if !plan.day_of_week.is_none() {
            day_of_week = plan.day_of_week.unwrap().to_string();
        } else {
            day_of_week = "*".parse().unwrap();
        }
    }
}

```

```

        if !plan.day_of_month.is_none() &&
!plan.day_of_week.is_none(){
            day_of_month = plan.day_of_month.unwrap();
            day_of_week = format!("{}", [\'$(date \'+%u\')\'\' =
\'{}\'] &&', plan.day_of_week.unwrap(),
plan.day_of_week.unwrap())
        } else {
            day_of_month = "*".parse().unwrap();
            //day_of_week = "*".parse().unwrap();
        }
        let ss = format!("(crontab -l ; echo {} {} {} * {}
echo $'copies : {}\\nuuid : {}' > /root/test/xen-uuids.txt &&
/root/test/backupscrip.sh 1 >> /root/test/crontab.log >
/dev/null 2>&1\\") | crontab -", min, h, day_of_month,
day_of_week,plan.copies, plan.machines);
        println!("{}", ss);
        channel.exec(&*ss);
        channel.wait_close();
        println!("{}", channel.exit_status().unwrap());
        sess.disconnect(None, "", None);
    }
    pub fn delete(plan: Plan, conn: ConnectRequest) {
        let tcp =
TcpStream::connect(conn.host.clone()).unwrap();

        let mut sess = Session::new().unwrap();
        sess.set_tcp_stream(tcp);
        sess.handshake().unwrap();

        sess.userauth_password(&*conn.login, &*conn.password);

        let mut channel = sess.channel_session().unwrap();
        let mut min = String::new();
        for (b, i) in plan.minute.iter().enumerate() {
            if b != plan.minute.len()-1 {
                min.push_str(&*format!("{}",",", i));
            } else {
                min.push_str(&*format!("{}",",", i));
            }
        }
        let mut h = String::new();
        for (b, i) in plan.hour.iter().enumerate() {
            if b != plan.hour.len()-1 {
                h.push_str(&*format!("{}",",", i));
            } else {
                h.push_str(&*format!("{}",",", i));
            }
        }
        let mut day_of_month = String::new();
        let mut day_of_week = String::new();

        if !plan.day_of_week.is_none() {

```



```

        let mut s = String::new();
        channel.read_to_string(&mut s).unwrap();
        channel.wait_close();
        let mut line = s.lines();
        for x in line {
            let id : Uuid =
x.chars().rev().collect::<String>()[4..40].to_string().chars().r
ev().collect::<String>().parse().unwrap();
            let str : String =
x.chars().rev().collect::<String>()[41..55].to_string().chars().
rev().collect::<String>();
            let date =
NaiveDateTime::parse_from_str(str.as_str(), "%d%m%Y-
%H:%M").unwrap();
            let name =
x.chars().rev().collect::<String>()[56..x.chars().count()].to_st
ring().chars().rev().collect::<String>();
            // let mut id : Uuid = x.chars().rev().collect();
            res.push( BackupResponse {
                id : id.clone(),
                date: date.clone(),
                name : name.clone()
            })
        }
        res
    }
}

```

Файл «file_logic.rs»

```

use std::io::{Read, Write};
use std::net::TcpStream;
use std::path::Path;
use std::str::from_utf8;
use local_ip_address::local_ip;
use regex::Regex;
use ssh2::Session;
use crate::dto::connect_request::ConnectRequest;

pub struct FileLogic {}

impl FileLogic {
    pub fn update_ip (connect : ConnectRequest){
        let tcp =
TcpStream::connect(connect.host.clone()).unwrap();

        let mut sess = Session::new().unwrap();
        sess.set_tcp_stream(tcp);
        sess.handshake().unwrap();

        sess.userauth_password(&connect.login,
&connect.password);
    }
}

```

```

        let b= sess.authenticated();
        let my_local_ip = local_ip().unwrap();
        println!("This is my local IP address: {:?}",
my_local_ip);
        let (mut remote_file, stat) =
sess.scp_recv(Path::new("test/backupsript.sh")).unwrap();
        println!("remote file size: {}", stat.size());
        let mut contents = Vec::new();
        remote_file.read_to_end(&mut contents).unwrap();

// Close the channel and wait for the whole content to be
tranferred
        remote_file.send_eof().unwrap();
        remote_file.wait_eof().unwrap();
        remote_file.close().unwrap();
        remote_file.wait_close().unwrap();
        let s = from_utf8(&contents).unwrap().to_string();
        println!("{}", s);
        let re =
Regex::new(r#"BACKENDULR="http://([^\"]+)/api"#).unwrap();
        let captures = re.captures(&s).unwrap();
        let match_str = captures.get(1).unwrap().as_str();
        let out = s.replace(match_str, &format!("{}",80",
my_local_ip.to_string()));
        // println!("{}", out);

        let mut remote_file1 =
sess.scp_send(Path::new("test/backupsript.sh"), 0o644,
out.len() as u64, None).unwrap();
        //println!("{}", str.clone());
        remote_file1.write_all(out.as_ref()).unwrap();
// Close the channel and wait for the whole content to be
transferred
        remote_file1.send_eof().unwrap();
        remote_file1.wait_eof().unwrap();
        remote_file1.close().unwrap();
        remote_file1.wait_close().unwrap();
    }}

```

Файл «*machine_logic.rs*»

```

use std::io::Read;
use std::net::TcpStream;
use ssh2::Session;
use crate::dto::connect_request::ConnectRequest;
use crate::dto::machine_request::MachineRequest;

pub struct MachineLogic {}

impl MachineLogic {
    pub fn get_machine(conn : ConnectRequest)->
Vec<MachineRequest> {

```

```

let tcp =
TcpStream::connect(conn.host.clone()).unwrap();

let mut sess = Session::new().unwrap();
sess.set_tcp_stream(tcp);
sess.handshake().unwrap();

sess.userauth_password(&*conn.login,
&*conn.password).unwrap();
sess.authenticated();
let mut channel = sess.channel_session().unwrap();
channel.exec("xe vm-list is-control-domain=false is-a-
snapshot=false").unwrap();
let mut s = String::new();
channel.read_to_string(&mut s).unwrap();
println!("{}", s);
// channel.wait_close();
println!("{}", channel.exit_status().unwrap());

let mut data : Vec<MachineRequest> = vec![];
let mut line = s.lines();
let mut id = "".to_string();
let mut name = "".to_string();
for x in line {

    if !x.find("uuid").is_none() {
        let start_bytes = s.find(": ").unwrap_or(0);
        // let end_bytes =
s.find("\n").unwrap_or(s.len());
        id = x[start_bytes+2..x.len()].parse().unwrap();
    }
    if !x.find("name").is_none() {
        let start_bytes = s.find(": ").unwrap_or(0);
        // let end_bytes =
s.find("\n").unwrap_or(s.len());
        name =
x[start_bytes+2..x.len()].parse().unwrap();
    }
    if !x.find("power").is_none() {
        let start_bytes = s.find(": ").unwrap_or(0);
        // let end_bytes =
s.find("\n").unwrap_or(s.len());
        // info.state =
x[start_bytes+2..x.len()].parse().unwrap();
        // info.state =
x[start_bytes+2..x.len()].parse().unwrap();
        data.push(MachineRequest {
            uuid: id.clone(),
            name: name.clone(),
            state:
x[start_bytes+2..x.len()].parse().unwrap()

```

					ПГУ1.090301.35.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		68

```

        });
    }
}
data

}
}

```

Файл «*backup_service.rs*»

```

use std::io::Read;
use std::net::TcpStream;
use std::thread::sleep;
use std::time::Duration;
use actix_web::{get, post, put, delete, web, HttpRequest,
HttpResponse};
use actix_web::web::Json;
use ssh2::Session;
use uuid::Uuid;
use crate::config::app_data::AppData;
use crate::dto::connect_request::ConnectRequest;
use crate::logic::backup_logic::BackupLogic;

pub fn init_routes(config: &mut web::ServiceConfig) {
    config.service(
        web::scope("backup")
            .service(backup_list)
            .service(backup));
}

#[post("list")]
pub async fn backup_list(
    connect: Json<ConnectRequest>,
    req: HttpRequest
) -> HttpResponse {
    let res = BackupLogic::backup_list(connect.0);

    HttpResponse::Ok().json(res)
}

#[post("{id}")]
pub async fn backup(
    connect: Json<ConnectRequest>,
    req: HttpRequest
) -> HttpResponse {
    let id: Uuid =
req.match_info().get("id").unwrap().parse().unwrap();
    let tcp = TcpStream::connect(connect.host.clone());
    if tcp.is_err() {
        return HttpResponse::BadRequest().json("Failed
connection to host");
    }
    let mut sess = Session::new();
    if sess.is_err() {

```

```

        return HttpResponse::BadRequest().json("Failed
connection to host");
    }
    sess.as_mut().unwrap().set_tcp_stream(tcp.unwrap());
    if sess.as_mut().unwrap().handshake().is_err() {
        return HttpResponse::BadRequest().json("Failed
connection to host");
    }

    if sess.as_mut().unwrap().userauth_password(&*connect.login,
&*connect.password).is_err() {
        return HttpResponse::BadRequest().json("Incorrect
password or login");
    }
    let mut channel =
sess.as_mut().unwrap().channel_session().unwrap();
    channel.exec(&*format!("echo $('copies : 0\\nuuid : {}' >
/root/test/xen-uuids.txt", id));
    channel.wait_close();
    let mut channel1 =
sess.as_mut().unwrap().channel_session().unwrap();
    channel1.exec("/root/test/backupscript.sh 1").unwrap();
    sleep(Duration::new(10,0));
    channel1.close();
    //let mut s = String::new();
    // channel1.read_to_string(&mut s).unwrap();
    // println!("{}", s);
    // channel1.wait_close();
    HttpResponse::Ok().json("Backup run")
}

```

Файл «connect_service.rs»

```

use std::io::Read;
use std::net::TcpStream;
use actix_web::{get, post, put, delete, web, HttpRequest,
HttpResponse};
use actix_web::web::Json;
use serde_json::json;
use ssh2::Session;
use crate::config::app_data::AppData;
use crate::dto::connect_request::ConnectRequest;
use crate::logic::connect_logic::ConnectLogic;
use crate::logic::file_logic::FileLogic;

pub fn init_routes(config: &mut web::ServiceConfig) {
    config.service(
        web::scope("connect")
            .service(connect));
}
#[post("")]
pub async fn connect(
    app_data: web::Data<AppData>,

```

```

        connect: Json<ConnectRequest>,
    ) -> HttpResponse {
        let tcp = TcpStream::connect(connect.host.clone());
        if tcp.is_err() {
            return HttpResponse::BadRequest().json("Failed
connection to host");
        }
        let mut sess = Session::new();
        if sess.is_err() {
            return HttpResponse::BadRequest().json("Failed
connection to host");
        }
        sess.as_mut().unwrap().set_tcp_stream(tcp.unwrap());
        if sess.as_mut().unwrap().handshake().is_err() {
            return HttpResponse::BadRequest().json("Failed
connection to host");
        }

        if sess.as_mut().unwrap().userauth_password(&*connect.login,
&*connect.password).is_err() {
            return HttpResponse::BadRequest().json("Incorrect
password or login");
        }
        let mut channel =
sess.as_mut().unwrap().channel_session().unwrap();
        channel.exec("ls test/backupscript.sh");
        let mut s = String::new();
        channel.read_to_string(&mut s).unwrap();
        println!("{}", s);
        channel.wait_close();
        if s.find("test/backupscript.sh").is_some() {
            FileLogic::update_ip(connect.clone());
            return HttpResponse::Ok().json("Connect success");
        } else {
            return HttpResponse::Forbidden().json("Agent not
found");
        }
    }

```

Файл «notification_service.rs»

```

use std::io::Read;
use std::net::TcpStream;
use actix_web::{get, post, put, delete, web, HttpRequest,
HttpResponse};
use actix_web::web::Json;
use ssh2::Session;
use uuid::Uuid;
use crate::config::app_data::AppData;
use crate::cron::plan_cron::PlanCron;
use crate::dao::plan_dao::PlanDao;
use crate::dto::connect_request::ConnectRequest;
use crate::dto::plan_request::PlanRequest;

```

```

use crate::dto::recovery_response::RecoveryRequest;
use crate::logic::machine_logic::MachineLogic;

pub fn init_routes(config: &mut web::ServiceConfig) {
    config.service(
        web::scope("machine")
            .service(get_machine)
            .service(recovery_machine)
            .service(start_machine)
            .service(stop_machine)
            .service(reboot_machine)
    );
}

#[post("")]
pub async fn get_machine(
    app_data: web::Data<AppData>,
    connect: Json<ConnectRequest>,
) -> HttpResponse {
    let res = MachineLogic::get_machine(connect.0);
    HttpResponse::Ok().json(res)
}

#[post("start/{id}")]
pub async fn start_machine(
    connect: Json<ConnectRequest>,
    req: HttpRequest,
) -> HttpResponse {
    let id: Uuid =
    req.match_info().get("id").unwrap().parse().unwrap();
    let conn = connect.0;
    let tcp = TcpStream::connect(conn.host.clone()).unwrap();

    let mut sess = Session::new().unwrap();
    sess.set_tcp_stream(tcp);
    sess.handshake().unwrap();
    sess.userauth_password(&conn.login, &conn.password);
    let mut channel = sess.channel_session().unwrap();
    channel.exec(&format!("xe vm-start uuid={}", id)).unwrap();
    let mut s = String::new();
    channel.read_to_string(&mut s).unwrap();
    println!("{}", s);
    channel.wait_close();
    HttpResponse::Ok().json("VM started")
}

#[post("stop/{id}")]
pub async fn stop_machine(
    req: HttpRequest,
    connect: Json<ConnectRequest>,
) -> HttpResponse {
    let id: Uuid =
    req.match_info().get("id").unwrap().parse().unwrap();

```

					ПГУ1.090301.35.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		72

```

let conn = connect.0;
let tcp = TcpStream::connect(conn.host.clone()).unwrap();

let mut sess = Session::new().unwrap();
sess.set_tcp_stream(tcp);
sess.handshake().unwrap();
sess.userauth_password(&*conn.login, &*conn.password);
let mut channel = sess.channel_session().unwrap();
channel.exec(&*format!("xe vm-shutdown uuid={}",
id)).unwrap();
let mut s = String::new();
channel.read_to_string(&mut s).unwrap();
println!("{}", s);
channel.wait_close();
HttpResponse::Ok().json("VM stopped")
}

#[post("reboot/{id}")]
pub async fn reboot_machine(
    req: HttpRequest,
    connect: Json<ConnectRequest>,
) -> HttpResponse {
    let id: Uuid =
req.match_info().get("id").unwrap().parse().unwrap();
    let conn = connect.0;
    let tcp = TcpStream::connect(conn.host.clone()).unwrap();

    let mut sess = Session::new().unwrap();
    sess.set_tcp_stream(tcp);
    sess.handshake().unwrap();
    sess.userauth_password(&*conn.login, &*conn.password);
    let mut channel = sess.channel_session().unwrap();
    channel.exec(&*format!("xe vm-reboot uuid={}",
id)).unwrap();
    let mut s = String::new();
    channel.read_to_string(&mut s).unwrap();
    println!("{}", s);
    channel.wait_close();
    HttpResponse::Ok().json("VM is reboot")
}

#[post("recovery")]
pub async fn recovery_machine(
    app_data: web::Data<AppData>,
    connect: Json<RecoveryRequest>,
) -> HttpResponse {
    // let res = MachineDao::get_machine(connect.0);
    let tcp = TcpStream::connect(connect.host.clone());
    if tcp.is_err() {
        return HttpResponse::BadRequest().json("Failed
connection to host");
    }
}

```

```

    let mut sess = Session::new();
    if sess.is_err() {
        return HttpResponse::BadRequest().json("Failed
connection to host");
    }
    sess.as_mut().unwrap().set_tcp_stream(tcp.unwrap());
    if sess.as_mut().unwrap().handshake().is_err() {
        return HttpResponse::BadRequest().json("Failed
connection to host");
    }

    if sess.as_mut().unwrap().userauth_password(&connect.login,
&connect.password).is_err() {
        return HttpResponse::BadRequest().json("Incorrect
password or login");
    }
    let mut channel =
sess.as_mut().unwrap().channel_session().unwrap();
    channel.exec(&format!("/root/test/backupsript.sh 0 {} {}
{}", connect.id.clone(), connect.path.clone(),
connect.name.clone() )).unwrap();
    let mut s = String::new();
    channel.read_to_string(&mut s).unwrap();
    channel.wait_close();
    let mut line = s.lines();
    if line.clone().last().is_some() {
        let id = Uuid::parse_str(line.last().unwrap());
        if id.is_ok() {
            let plan =
PlanDao::find_by_machine(app_data.pool.clone(),
connect.id.clone());
            if plan.len() > 0 {
                let mut plan1_to_update = plan[0].clone();
                plan1_to_update.machines = id.unwrap();
                let conn = ConnectRequest {
                    login: connect.login.clone(),
                    password: connect.password.clone(),
                    host: connect.host.clone()
                };
                PlanCron::delete(plan[0].clone(), conn);
                PlanCron::create(PlanRequest {
                    id:
Option::from(plan1_to_update.uuid.clone()),
                    copies: plan1_to_update.copies.clone(),
                    day_of_week:
plan1_to_update.day_of_week.clone(),
                    day_of_month:
plan1_to_update.day_of_month.clone(),
                    hour: plan1_to_update.hour.clone(),
                    minute: plan1_to_update.minute.clone(),
                    machines: connect.id.clone(),
                    login: connect.login.clone(),

```

```

        password: connect.password.clone(),
        host: connect.host.clone()
    ));
    PlanDao::update(
        app_data.pool.clone(),
        plan[0].clone(),
        plan1_to_update.clone(),
    );
    }
    }else {
        return
        HttpResponse::InternalServerError().json("Machine not recovery")
    }

    }else {
        return HttpResponse::InternalServerError().json("Machine
not recovery")
    }
    // let plan =
    HttpResponse::Ok().json("200")
}

```

Файл «notification_service.rs»

```

use actix_web::{get, post, put, delete, web, HttpRequest,
HttpResponse};
use actix_web::web::Json;
use uuid::Uuid;

use crate::schema;
use crate::entity::notification::Notification;
use crate::dao::notification_dao::NotificationDao;
use crate::dto::notification_response::NotificationResponse;
use crate::dto::notification_request::NotificationRequest;
use crate::config::app_data::AppData;

pub fn init_routes(config: &mut web::ServiceConfig) {
    config.service(
        web::scope("notification")
            .service(create)
            .service(update)
            .service(find_all)
            .service(find)
            .service(delete),
    );
}

#[put("")]
pub async fn update(
    app_data: web::Data<AppData>,
    updated_product1: Json<NotificationRequest>,
) -> HttpResponse {
    if updated_product1.id.is_none() {
        return HttpResponse::BadRequest().json(

```

					ПГУ1.090301.35.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		75

```

        "field id shouldn't be null!"
    );
}
let real_product1 =
NotificationDao::find(app_data.pool.clone(),
updated_product1.id.unwrap());
if real_product1.len() < 1 {
    return HttpResponse::NotFound()
        .json("Product not found!");
}
let product1_to_update = updated_product1.to_notification();
NotificationDao::update(
    app_data.pool.clone(),
    real_product1[0].clone(),
    product1_to_update.clone(),
);
let res = NotificationDao::find(app_data.pool.clone(),
product1_to_update.uuid);

HttpResponse::Ok().json(NotificationResponse::from_notification(
res[0].clone()))
}
#[post("")]
pub async fn create(
    app_data: web::Data<AppData>,
    product1: Json<NotificationRequest>, ) -> HttpResponse {
    let new_product1 = product1.to_notification();
    NotificationDao::create(app_data.pool.clone(),
new_product1.clone());
    let res = NotificationDao::find(app_data.pool.clone(),
new_product1.uuid);

    HttpResponse::Ok().json(NotificationResponse::from_notification(
res[0].clone()))
}
#[get("all")]
pub async fn find_all(app_data: web::Data<AppData>) ->
Result<HttpResponse, actix_web::Error>{
    let conn = app_data.pool.get().unwrap();
    let product1 = web::block(move ||
NotificationDao::find_all(&conn))
        .await
        .map_err(|e| {
            eprintln!("{:?}", e);
            HttpResponse::InternalServerError().finish()
        })?;

    Ok(HttpResponse::Ok().json(NotificationResponse::from_notificati
on_list(product1)))
}
#[delete("{id}")]

```

```

pub async fn delete(app_data: web::Data<AppData>, req:
HttpRequest) -> HttpResponse {
    let id: Uuid =
req.match_info().get("id").unwrap().parse().unwrap();
    let res = NotificationDao::delete(app_data.pool.clone(),
id);
    if res == 0 {
        return HttpResponse::BadRequest().json(
            "Temporary exception! Please, try later",
        );
    }
    HttpResponse::Ok().json("200")
}
#[get("{id}")]
pub async fn find(app_data: web::Data<AppData>, req:
HttpRequest) -> HttpResponse {
    let id: Uuid =
req.match_info().get("id").unwrap().parse().unwrap();
    let products = NotificationDao::find(app_data.pool.clone(),
id);

    HttpResponse::Ok().json(NotificationResponse::from_notification(
products[0].clone()))
}

```

Файл «*plan_service.rs*»

```

use actix_web::{get, post, put, delete, web, HttpRequest,
HttpResponse};
use actix_web::web::Json;
use uuid::Uuid;

use crate::schema;
use crate::entity::plan::Plan;
use crate::dto::plan_response::PlanResponse;
use crate::dto::plan_request::PlanRequest;
use crate::dao::plan_dao::PlanDao;
use crate::config::app_data::AppData;
use crate::cron::plan_cron::PlanCron;
use crate::dto::connect_request::ConnectRequest;

pub fn init_routes(config: &mut web::ServiceConfig) {
    config.service(
        web::scope("plan")
            .service(create)
            .service(update)
            .service(find_all)
            .service(find)
            .service(delete),
    );
}

```

					ПГУ1.090301.35.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		77

```

#[put("")]
pub async fn update(
    app_data: web::Data<AppData>,
    updated_plan1: Json<PlanRequest>,
) -> HttpResponse {
    if updated_plan1.id.is_none() {
        return HttpResponse::BadRequest().json(
            "field id shouldn't be null!"
        );
    }
    let plan1_to_update = updated_plan1.to_plan();
    let real_plan1 = PlanDao::find(app_data.pool.clone(),
updated_plan1.id.unwrap());
    let conn = ConnectRequest {
        login: updated_plan1.login.clone(),
        password: updated_plan1.password.clone(),
        host: updated_plan1.host.clone()
    };
    PlanCron::delete(real_plan1[0].clone(), conn);
    PlanCron::create(updated_plan1.0);
    if real_plan1.len() < 1 {
        return HttpResponse::NotFound()
            .json("Plan not found!");
    }

    PlanDao::update(
        app_data.pool.clone(),
        real_plan1[0].clone(),
        plan1_to_update.clone(),
    );
    let res = PlanDao::find(app_data.pool.clone(),
plan1_to_update.uuid);

    HttpResponse::Ok().json(PlanResponse::from_plan(res[0].clone()))
}

#[post("")]
pub async fn create(
    app_data: web::Data<AppData>,
    plan1: Json<PlanRequest>, ) -> HttpResponse {
    let new_plan1 = plan1.to_plan();
    PlanDao::create(app_data.pool.clone(), new_plan1.clone());
    PlanCron::create(plan1.0);
    let res = PlanDao::find(app_data.pool.clone(),
new_plan1.uuid);

    HttpResponse::Ok().json(PlanResponse::from_plan(res[0].clone()))
}

#[get("all")]
pub async fn find_all(app_data: web::Data<AppData>) ->
Result<HttpResponse, actix_web::Error>{

```

					ПГУ1.090301.35.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		78

```

        let conn = app_data.pool.get().unwrap();
        let plan1 = web::block(move || PlanDao::find_all(&conn))
            .await
            .map_err(|e| {
                eprintln!("{:?}", e);
                HttpResponse::InternalServerError().finish()
            })?;

Ok(HttpResponse::Ok().json(PlanResponse::from_plan_list(plan1)))
}
#[delete("/{id}")]
pub async fn delete(app_data: web::Data<AppData>, req:
HttpRequest, connect : Json<ConnectRequest>) -> HttpResponse {
    let id: Uuid =
req.match_info().get("id").unwrap().parse().unwrap();
    let plan1s = PlanDao::find(app_data.pool.clone(), id);
    PlanCron::delete(plan1s[0].clone(), connect.0);
    let res = PlanDao::delete(app_data.pool.clone(), id);
    if res == 0 {
        return HttpResponse::BadRequest().json(
            "Temporary exception! Please, try later",
        );
    }
    HttpResponse::Ok().json("200")
}
#[get("/{id}")]
pub async fn find(app_data: web::Data<AppData>, req:
HttpRequest) -> HttpResponse {
    let id: Uuid =
req.match_info().get("id").unwrap().parse().unwrap();
    let plan1s = PlanDao::find(app_data.pool.clone(), id);

    HttpResponse::Ok().json(PlanResponse::from_plan(plan1s[0].clone(
    )
    )
    )
    )
}

```

Файл «main.rs»

```

use std::io::Read;
use std::net::TcpStream;
use actix_web::{web, App, HttpRequest, HttpServer, Responder,
FromRequest};
use r2d2_diesel::ConnectionManager;
use crate::service::app_config;
#[macro_use]
extern crate diesel;
extern crate r2d2;
extern crate r2d2_diesel;
use dotenv::dotenv;
use diesel::pg::PgConnection;

```

```

use log::info;
use crate::config::app_data::AppData;
use actix_cors::Cors;
mod entity;
mod dto;
mod dao;
mod service;
mod config;
mod schema;
mod cron;
mod logic;

use ssh2::Session;

#[actix_web::main]
async fn main() -> std::io::Result<()> {
    dotenv().ok();
    let config = crate::config::Config::from_env().unwrap();
    let main_config =
crate::config::Config::from_env().unwrap();
    let manager =
ConnectionManager::<PgConnection>::new(main_config.database.url)
;
    let pool = r2d2::Pool::builder()
        .build(manager)
        .expect("Failed to create postgres pool.");

    info!(
        "Starting server at http://{}:{})",
        main_config.server.host, main_config.server.port
    );

    HttpServer::new(move || {
        let cors = Cors::permissive();
        App::new()
            .data(AppData::new(pool.clone(), config.clone()))
            .wrap(cors)
            .app_data(actix_web::web::Bytes::configure(|cfg| {
                cfg.limit(1024 * 1024 * 250) // Set max file
size to 6 Mib
            })))
            .configure(app_config)
            .data(web::JsonConfig::default().limit(1024 * 1024 *
250))

        })
        .bind(format!(
            "{}:{})",
            main_config.server.host, main_config.server.port
        ))?
        .run()
        .await

```

}

Файл «*schema.rs*»

```
table! {  
    plan (uuid) {  
        uuid -> Uuid,  
        updt -> Nullable<Timestamp>,  
        updu -> Nullable<Uuid>,  
        copies -> Int4,  
        day_of_week -> Nullable<Int4>,  
        day_of_month -> Nullable<Varchar>,  
        hour -> Array<Int4>,  
        minute -> Array<Int4>,  
        machines -> Uuid,  
    }  
}  
table! {  
    notification (uuid) {  
        uuid -> Uuid,  
        updt -> Nullable<Timestamp>,  
        updu -> Nullable<Uuid>,  
        machine -> Nullable<Uuid>,  
        viewed -> Nullable<Bool>,  
        message -> Nullable<Varchar>,  
        type_message -> Nullable<Varchar>,  
    }  
}
```

ПриложениеВ

(Обязательное)

Система Ваксирдля кластеров хср-ng. Сервер.

Презентация

					ПГУ1.090301.35.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		81





аботы

ой работы является разработка
асти приложения для резервного
и машин кластера ХСР-ng. Сервер
вать планы резервного
и на управляющей машине и
с в базу данных, возвращать
о о машинах, их состоянии и
резервного копирования, а также
ь стабильный обмен данными с

Изм.	Лист	№ докум.	Подпись	Дата



СИ

мощное и гибкое приложение для
копирования, которое может быть
использовано на различных платформах,
включая Windows, macOS и Linux.

Это бесплатное и открытое
приложение для резервного копирования,
которое может использоваться на различных
платформах, включая Windows, macOS и Linux.

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.35.001 ПЗ

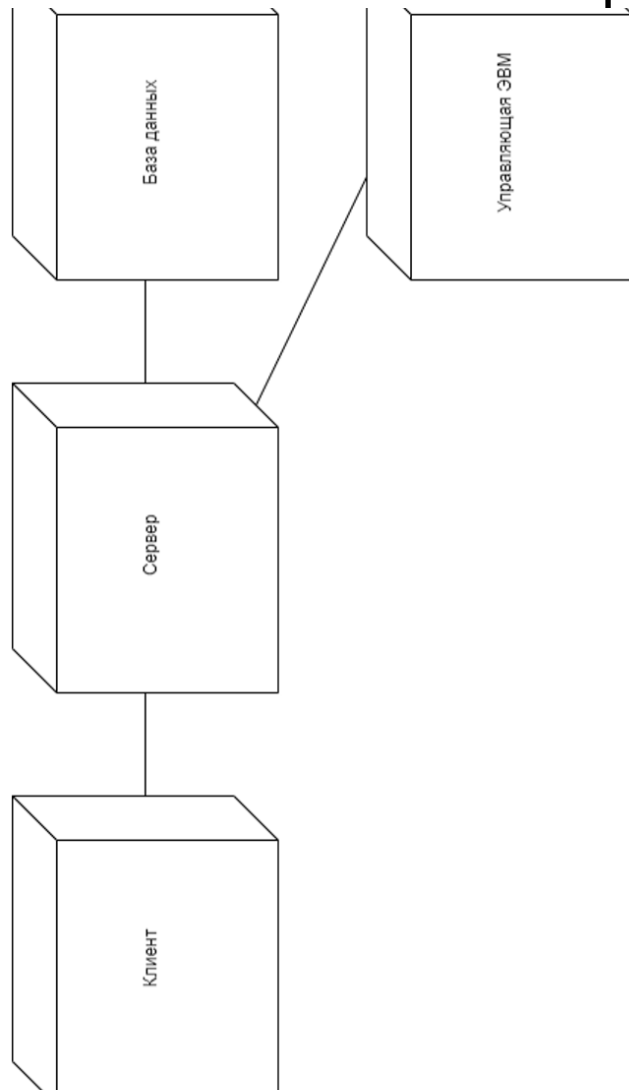
Лист
85



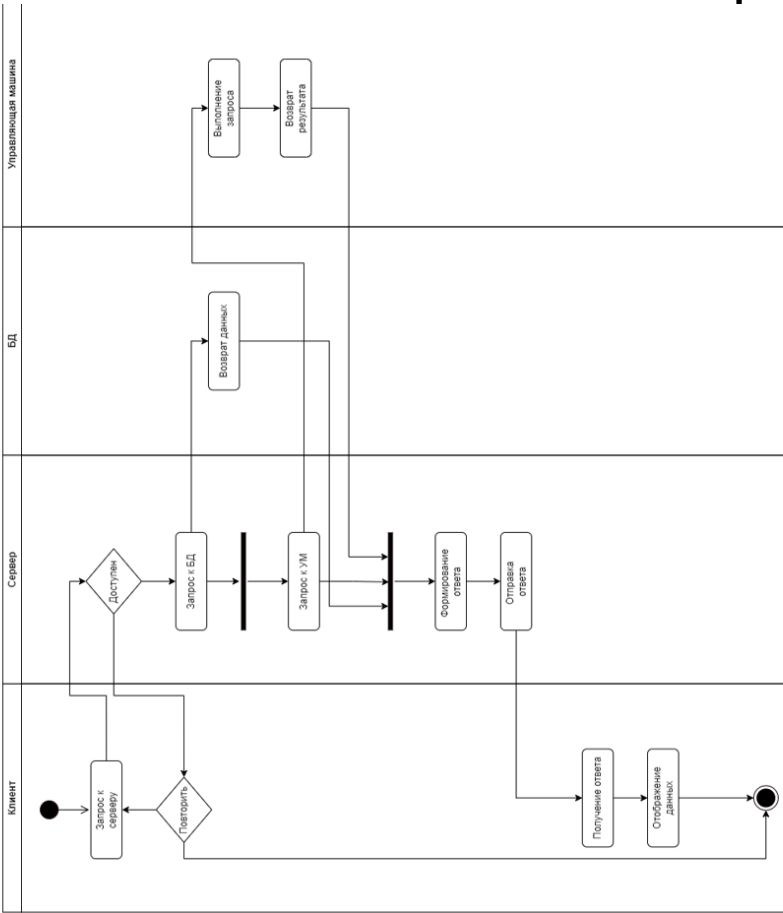
твa разработки

работки CLion
раммирования Rust
ка Actix
ка diesel
greSQL
зационная платформа Xcp-ng

ктура проекта



UML диаграмма деятельности





Аннотация задачи

Аннотация к управляющей машине
задание расписания резервного
копирования на управляющей машине
задание резервного копирования
задание машины по резервной

скрипта на управляющую машину

ие ip

Изм.	Лист	№ докум.	Подпись	Дата

Включение к УМ

POST

Http://127.0.0.1:9090/connect

ParamsAuthorizationHeaders (12)BodyPre-request ScriptTestsSettings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

1

2

3

4

5

"login" : "root",

"password" : "xenbackup",

"host" : "172.16.70.250:22"

bodyCookiesHeaders (3)Test Results

PrettyRawPreviewVisualizeJSON

1true

Status: 2

ание плана

PUThttp://127.0.0.1:8080/jpan

ParamsAuthorizationHeaders (12)BodyPre-request ScriptTestsSettings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

"login": "root",

"password": "xenobackup",

"host": "172.16.78.250:22",

"id": "43a4d0ed-e737-4ed0-957e-544f94d721b4",

"copies": 4,

"dayOfWeek": 4,

"dayOfMonth": 21,

"hour": [

10,

11,

12

],

"minute": [

30,

40

],

"machines": "4a48f3f-5e46-7b5d-7eda-7a469a9a50e8"

Body

Cookies

Headers (3)

Test Results

Pretty

Raw

Preview

Visualize

JSON

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

"id": "43a4d0ed-e737-4ed0-957e-544f94d721b4",

"copies": 4,

"dayOfWeek": 4,

"dayOfMonth": 21,

"hour": [

10,

11,

12

],

"minute": [

30,

40

],

"machines": "4a48f3f-5e46-7b5d-7eda-7a469a9a50e8"

Status 200 OK

ление резервного копирова

POST

▼

http://127.0.0.1:8080/backup/3e8fa094-eb65-be22-973f-9a42b02cd592

Params

Authorization

Headers (12)

Body ●

Pre-request Script

Tests

Settings

● none

● form-data

● x-www-form-urlencoded ● raw ● binary ● GraphQL

JSON ▼

1

2

3

4

5

....."login": "root",

....."password": "xenbackup",

....."host": "172.16.70.250:22"

Body

Cookies

Headers (3)

Test Results

Pretty

Raw

Preview

Visualize

JSON ▼

🔗

1 "Backup run"

ановление машины

POST [http://127.0.0.1:8080/machine/recovery](#)

Params	Authorization	Headers (12)	Body	Pre-request Script	Tests	Settings
☐ none	☐ form-data	☐ x-www-form-urlencoded	☒ raw	☐ binary	☐ GraphQL	☒ JSON

```

1 {
2   ... "login": "root",
3   ... "password": "xenbackup",
4   ... "host": "172.16.70.250:22",
5   ... "id": "ac0a7fa8-7e36-e330-cd02-b3d632788225",
6   ... "path": "Backups-29052023-05:00-ac0a7fa8-7e36-e330-cd02-b3d632788225.xva",
7   ... "name": "Backups"
8 }
```

Body Cookies Headers (3) Test Results

Pretty Raw Preview Visualize JSON

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.35.001 ПЗ

ка скрипта на управляющую ма

16.70.244/ap/file

headers (12)

Body

Pre-request Script

Tests

Settings

k-ww-form-urlencoded

raw

binary

GraphQL

JSON

ot",

"xenbackup",

".16.70.250:22",

172.16.254.190/xc",

:"adminarc",

d" : "Hw@Archive2020",

"172.16.70.250"

est Results

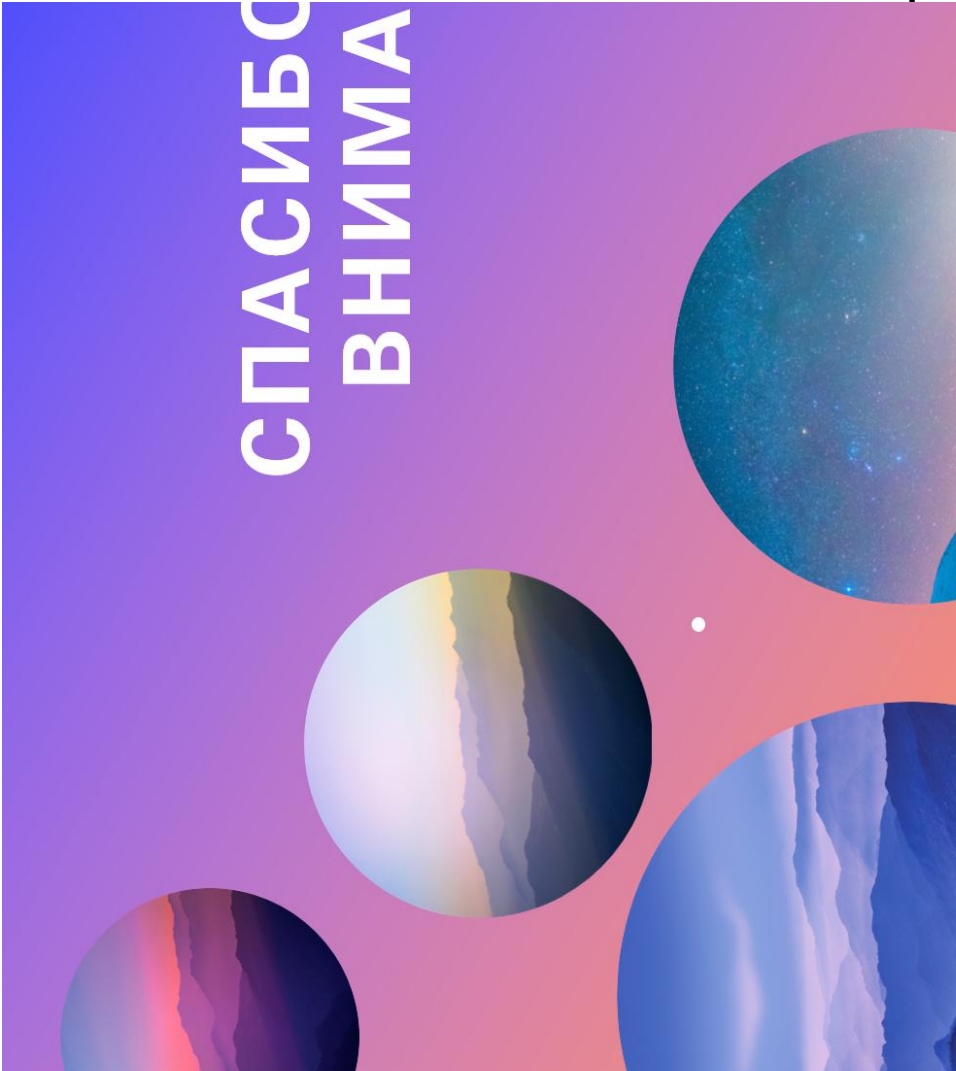
Visualize

Text

Ючение

гат выполнения данной выпускн
икационной работы можно счит
ым, серверная часть системы В
стеров хср-пг реализована в
гствии с заданием.

					ПГУ1.090301.35.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		95



					ПГУ1.090301.35.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		96