

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»

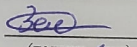
БАКАЛАВРСКАЯ РАБОТА

на тему

СИСТЕМА ВАСКУР ДЛЯ КЛАСТЕРОВ

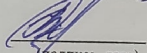
ХСР-NG. КЛИЕНТ

Студент


(подпись, дата) 31.05.2023

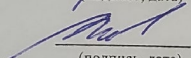
Земляков В.Д.

Руководитель


(подпись, дата) 9.06.2023

Попов К.В.

Нормоконтролёр


(подпись, дата) 9.06.2023

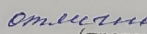
Бычков А.С.

Работа допущена к защите (протокол зас. кафедры от 2.06.23 г. № 14)

Заведующий кафедрой


(подпись)

Митрохин М.А.

Работа защищена с отметкой  (протокол заседания ГЭК от 14.06.23 № 13)

Секретарь ГЭК


(подпись)

Гурин Е.И.

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» 04 2023 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

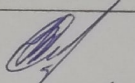
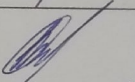
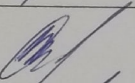
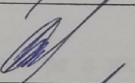
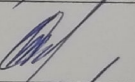
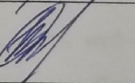
1. Студент Земляков Владислав Дмитриевич гр. 19ВВП2 факультета вычислительной техники
2. Тема работы: «Система Vaskip для кластеров xcp-ng. Клиент»
Тема утверждена приказом ПГУ №468 от "26" апреля 2023г.
3. Руководитель бакалаврской работы Попов К.В.
4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать приложение «Система *Vaskip* для кластеров *xcp-ng*. Клиент». Оно должно иметь клиент-серверную архитектуру. Клиентская часть программы должна быть интуитивно понятна пользователю и упрощать работу с кластером виртуальных машин *xcp-ng*. Приложение должно позволять пользователю: настраивать планы резервного копирования, выполнять бэкап и восстанавливать из него устройство, отслеживать результаты этих действий с помощью уведомлений и виджетов, отображающих сводную информацию о системе.

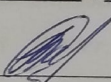
5. Перечень подлежащих разработке вопросов

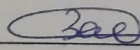
Требуется произвести анализ предметной областей: изучить аналоги существующих систем для резервного копирования виртуальных машин, изучить актуальные способы создания клиентской части web-приложений. Также необходимо изучить функционал фреймворка *React*.

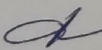
6. Календарный график выполнения работы

№ 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.	Выбор технологий, применяемых при проектировании и разработке	30	2.05.2023 – 4.05.2023	
4.	Описание алгоритмов	40	5.05.2023 – 9.05.2023	
5.	Разработки приложения: «Система Backup для кластеров xcp-ng»	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  Земляков В.Д.
(подпись, дата)

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

Реферат

Бакалаврская работа содержит 120 листов, 25 рисунков, 4 источника и 3 приложения

РЕЗЕРВНОЕ КОПИРОВАНИЕ, БЭКАПЫ, WEB-ПРИЛОЖЕНИЕ, *хср-ng*, *JAVASCRIPT*, *REACT*

Цель работы – разработать клиентскую часть *web*-приложения для резервного копирования кластера виртуальных машин *хср-ng*.

Приложение должно обладать таким функционалом, как:

- подключение к управляющей виртуальной машине по *ip*, логину и паролю;
- просмотр, создание, редактирование планов;
- создание резервных копий и восстановление из них;
- просмотр и удаление уведомлений о событиях;
- остановка, запуск, перезагрузка устройств;
- просмотр состояния виртуальных машин;
- отображение отчетов о устройствах, находящихся в кластере и их копиях;
- настройка точки монтирования.

В результате проведенной работы разработана клиентская часть *web*-приложения, полностью соответствующее поставленной задаче. Для достижения результата были использованы следующие технологии: *JavaScript*, *React*, *Webpack*, *React Router*, *MobX*, *Ant Design*, *nginx*. Приложение протестировано и готово к использованию.

					ПГУ1.090301.29.001 ПЗ		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Земляков В.Д.			Лит.	Лист	Листов
Провер.		Попов К.В.				2	120
Н. Контр		Бычков А.С.			ФВТ, каф. ВТ		
Утверд.		Митрохин М.А.			19ВВП2		

Содержание

Перечень сокращений и обозначений	5
Введение	6
1 Система Ваксир для кластеров хср-ng. Клиент.....	8
1.1 Анализ предметной области	8
1.1.1 Анализ приложений для резервного копирования.....	10
1.1.2 Постановка задачи на разработку	11
1.1.3 Выбор языка программирования.....	12
1.1.4 Выбор библиотеки для разработки интерфейсов	13
1.1.5 Выбор библиотеки для управления состоянием приложения.....	14
1.1.6 Выбор среды разработки.....	16
1.1.7 Выбранные технологии	17
1.2 Проектирование и разработка.....	17
1.2.1 Реализация сборки проекта	18
1.2.2 Реализация маршрутизации страниц	19
1.2.3 Реализация хранилищ.....	24
1.2.4 Реализация запросов к серверу.....	26
1.2.5 Реализация авторизации.....	31
1.2.6 Реализация планов	33
1.2.7 Реализация уведомлений.....	42
1.2.8 Реализация виджетов.....	45
1.2.9 Реализация резервных копий.....	52
2 Охрана труда.....	55
2.1 Основные вредоносные факторы	55
2.2 Общие требования к организации и оборудованию рабочих мест.....	56
2.3 Требования к помещениям с компьютерами	56
2.4 Правила размещения монитора	57
2.5 Режим труда и отдыха при использовании компьютера	58
Заключение	59

Список использованных источников	60
Приложение А – <i>UML</i> -Диаграммы.....	61
Приложение Б – Листинг программы	66
Приложение В – Презентация	103

					ПГУ1.090301.29.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		4

Перечень сокращений и обозначений

В настоящей работе применяют следующие сокращения и обозначения
ГОСТ – государственный стандарт

ПК – персональный компьютер

API (Application Programming Interface) – программный интерфейс приложения

CSS (Cascading Style Sheets) – формальный язык описания внешнего вида документа, написанного с использованием языка разметки

DOM (Document Object Model) – это представление HTML-документа в виде дерева тегов

HTML (Hyper Text Markup Language) – язык гипертекстовой разметки

HTTP (HyperText Transfer Protocol) – протокол прикладного уровня передачи данных

IP (Internet Protocol) – протокол, который используется для передачи данных в сети

JSX (JavaScript XML) – расширение языка *JavaScript*, которое используется в библиотеке *React* для описания пользовательского интерфейса

URL (Uniform Resource Locator) – система унифицированных адресов электронных ресурсов, или единообразный определитель местонахождения ресурса

Введение

Виртуальные серверы стали неотъемлемой частью современных информационных технологий. Они позволяют компаниям и организациям значительно сократить затраты на оборудование и управление серверами, а также повысить гибкость и масштабируемость своих инфраструктур. Однако, как и любые другие технологии, виртуальные серверы могут стать объектом кибератак и сбоев в работе, что может привести к потере данных и простоям в работе системы. Поэтому резервное копирование виртуальных серверов является важным этапом в обеспечении безопасности инфраструктуры компании или организации.

Одним из наиболее популярных решений для виртуализации серверов является *xscr-ng*. Это бесплатное и открытое программное обеспечение, которое позволяет создавать и управлять виртуальными серверами на базе операционной системы *xen*. *Xscr-ng* предлагает широкий набор инструментов для управления виртуальными машинами, в том числе возможность создания резервных копий и восстановления данных в случае сбоев в работе системы. Правильное использование этих инструментов поможет обеспечить надежную защиту данных и бесперебойную работу инфраструктуры.

Несмотря на то, что *xscr-ng* предлагает широкий набор инструментов для создания виртуальных копий, многие администраторы кластеров виртуальных машин считают, что эти инструменты не слишком удобны в использовании. В некоторых случаях, для создания резервных копий требуется выполнить несколько действий, что может занять много времени и потребовать от пользователя дополнительных знаний и опыта работы с системой.

Кроме того, некоторые инструменты *xscr-ng* могут быть сложны в настройке и управлении, что может привести к ошибкам и проблемам при создании резервных копий. Это может привести к потере данных или просто занять много времени и усилий для исправления ошибок.

					ПГУ1.090301.29.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

Несмотря на то, что *xcp-ng* предлагает широкий набор инструментов для создания виртуальных копий, администраторам кластеров виртуальных машин может быть удобнее использовать другие инструменты, которые более просты в использовании и управлении. Это поможет обеспечить надежную защиту данных и бесперебойную работу инфраструктуры компании или организации.

Для упрощения работы с *xcp-ng* можно разработать *web*-приложение, которое будет предоставлять удобный и простой интерфейс для создания резервных копий виртуальных машин. Это приложение может быть доступно через браузер с любого устройства, что позволит администраторам быстро и легко создавать копии данных.

Web-приложение может иметь интуитивно понятный интерфейс, который позволит пользователям легко выбрать виртуальную машину, для которой нужно создать резервную копию, выбрать тип копии и запустить процесс создания. Приложение может также предоставлять возможность настройки расписания создания копий и автоматической отправки их на удаленный сервер.

Такое *Web*-приложение может значительно упростить работу администраторов кластеров виртуальных машин и помочь обеспечить надежную защиту данных. Кроме того, такое приложение может быть разработано с учетом специфических потребностей конкретной компании или организации, что позволит оптимизировать процесс создания резервных копий и повысить эффективность работы инфраструктуры.

1 Система Backup для кластеров хср-ng. Клиент

1.1 Анализ предметной области

Трудно переоценить значения резервных копий в современном мире. Они отвечают за сохранение и обеспечение безопасности данных пользователей в случае сбоев в работе системы. Резервные копии представляют собой точную копию данных, которые хранятся на основной системе, и могут быть использованы для восстановления данных в случае их потери или повреждения.

Администраторы кластеров виртуальных машин должны быть особенно внимательны к созданию резервных копий виртуальных машин. Виртуальные машины могут содержать важные данные, такие как базы данных, почтовые серверы, файловые серверы и многое другое. Потеря этих данных может привести к серьезным последствиям для бизнеса.

В случае сбоя в работе системы, резервные копии виртуальных машин могут быть использованы для быстрого восстановления данных и сокращения времени простоя бизнеса. Кроме того, резервные копии могут быть использованы для тестирования новых версий программного обеспечения и обновлений без риска потери данных.

Создание резервных копий виртуальных машин является неотъемлемой частью работы администраторов кластеров виртуальных машин. Регулярное создание резервных копий и проверка их целостности помогут обеспечить надежную защиту данных и снизить риск потери информации в случае сбоя системы.

Создание резервных копий может быть выполнено как вручную, так и автоматически с помощью специальных программных решений. Такие решения предусмотрены в хср-ng.

Хср-ng является свободным и открытым гипервизором на основе Xen, который предназначен для виртуализации серверов и облачных вычислений.

					ПГУ1.090301.29.001 ПЗ	Лист
						8
Изм.	Лист	№ докум.	Подпись	Дата		

Он имеет множество инструментов для управления и мониторинга виртуальных машин, а также имеет возможность создания резервных копий и восстановления данных, что делает его надежным инструментом для обеспечения безопасности данных и сохранения информации в случае сбоев в работе системы [1].

Не смотря на все преимущества *xcp-ng*, он не удобен для пользователей, которые не имеют опыта работы с командной строкой и другими техническими инструментами. Это может привести к тому, что пользователи будут испытывать трудности при управлении виртуальными машинами и настройке планов резервных копий.

Чтобы решить эту проблему, необходимо разработать клиентскую часть для *xcp-ng*, которая будет обеспечивать более простой и интуитивно понятный интерфейс для пользователей.

Для решения проблемы рассматривалось два пути: десктопное-приложение и *web*-приложение. Десктопное приложение может быть установлено на компьютер пользователя и работать независимо от браузера. *Web* -приложение, с другой стороны, доступно через браузер и может быть запущено на любом устройстве с доступом в Интернет.

В качестве решения была выбрана реализация приложения с помощью *web*-технологий. *Web*-приложения не требуют установки на компьютер пользователя и могут быть запущены на любом устройстве с доступом в Интернет. Это делает их более удобными для пользователей, которые часто используют разные устройства для работы. Кроме того, *web*-приложения обычно обновляются автоматически, что позволяет пользователям получать новые функции и исправления без необходимости установки обновлений. Что соответствует запросам пользователя.

Клиентская часть *web*-приложения является важной составляющей его функциональности и удобства использования. Она отвечает за интерфейс

пользователя и взаимодействие с сервером, обеспечивая быстрый и удобный доступ к функциям приложения.

Кроме того, клиентская часть web-приложения должна быть удобной и интуитивно понятной для пользователей, что позволит им быстро освоиться с приложением и эффективно использовать его возможности.

Важно также обеспечить безопасность клиентской части приложения, чтобы предотвратить возможные угрозы безопасности, такие как взлом или кража данных.

Таким образом, клиентская часть web-приложения играет важную роль в обеспечении его функциональности, удобства использования и безопасности. Ее разработка и поддержка должны быть тщательно продуманы и организованы для достижения наилучших результатов.

1.1.1 Анализ приложений для резервного копирования

Web-приложения для резервного копирования являются важными инструментами для обеспечения безопасности данных и защиты от потери информации. Они позволяют пользователям создавать резервные копии данных, сохранять их на удаленных серверах и восстанавливать данные в случае их потери.

Популярными приложениями для резервного копирования виртуальных машин являются «Acronis Backup» и «Киберпротект».

«Acronis» является компанией, которая предоставляет решения для резервного копирования и восстановления данных. Одним из таких решений является интеграция «Acronis Backup» с xcp-ng.

«Acronis Backup» позволяет создавать резервные копии данных, включая файлы, базы данных и виртуальные машины. Оно также обеспечивает защиту данных с помощью шифрования и поддержки различных методов аутентификации.

Интеграция «Acronis Backup» с xcp-ng позволяет использовать возможности обоих продуктов для обеспечения максимальной защиты

					ПГУ1.090301.29.001 ПЗ	Лист
						10
Изм.	Лист	№ докум.	Подпись	Дата		

данных. Она позволяет автоматически выполнять резервное копирование данных на удаленные серверы, а также восстанавливать данные в случае их потери.

Кроме того, «Acronis Backup» имеет множество дополнительных функций, таких как мониторинг состояния системы, управление правами доступа и создание отчетов о резервном копировании.

В целом, интеграция «Acronis Backup» с «хср-ng» обеспечивает высокий уровень защиты данных и удобство использования.

Но есть одна значительная проблема в работе этого приложения. Его агенты размещаются на каждом узле кластера, что замедляет его работу.

«Киберпротект» работает аналогично.

В целом, web-приложения для резервного копирования являются важными инструментами для обеспечения безопасности данных и защиты от потери информации. Однако, при их использовании необходимо учитывать возможные угрозы безопасности и выбирать надежные приложения с высоким уровнем защиты данных.

1.1.2 Постановка задачи на разработку

Целью данной работы является разработка клиентской части web-приложения для резервного копирования виртуальных машин кластера «хср-ng». Пользовательская часть должно предоставлять собой понятный и удобный интерфейс, позволяющий просматривать состояния виртуальных машин всего кластера, настраивать планы резервного копирования, отслеживать результаты выполнения бэкапов, восстанавливать машины с помощью резервной копии.

Выделим основные требования к приложению:

- список машин с их состоянием и сводной информацией;
- настройка планов (ежедневно, еженедельно, ежемесячно);
- просмотр планов на месяц;
- подключение к управляющей виртуальной машине по ip, логину и

пароллю;

- консоль пользователя с выбранными им виджетами.

Также, архитектура приложения должна упрощать его будущую разработку.

Правильность работы программного средства должны проверяться и тестироваться. Ручное тестирование программного средства должно осуществляться путем выполнения тестовых примеров и сравнения выходных данных с ожидаемым результатом.

Для корректной работы приложения требуется браузер на движке chromium V8 версии 5.1. и старше.

1.1.3 Выбор языка программирования

Среди языков программирования для разработки клиента web-приложений можно выделить такие, как *TypeScript*, *CoffeeScript*, *Dart*, *Python* и многие другие. Однако, JavaScript остается наиболее популярным и широко используемым языком программирования.

JavaScript имеет множество преимуществ, которые делают его идеальным выбором для разработки клиента web-приложений. Во-первых, *JavaScript* является универсальным языком программирования, который может быть использован для создания как простых, так и сложных web-приложений. Во-вторых, JavaScript имеет большое сообщество разработчиков, которые создают множество библиотек и фреймворков, которые упрощают процесс разработки web-приложений. В-третьих, JavaScript имеет широкую поддержку web-браузерами, что позволяет создавать кросс-браузерные web-приложения [2].

Кроме того, *JavaScript* имеет множество инструментов и библиотек, которые позволяют создавать высокопроизводительные и масштабируемые web-приложения. Например, *React*, *Angular* и *Vue.js* – это популярные фреймворки *JavaScript*, которые позволяют создавать сложные web-приложения с использованием компонентной архитектуры.

					ПГУ1.090301.29.001 ПЗ	Лист
						12
Изм.	Лист	№ докум.	Подпись	Дата		

В целом, *JavaScript* является наиболее популярным и широко используемым языком программирования для создания клиента *web*-приложений. Он имеет множество преимуществ, которые делают его идеальным выбором для создания как простых, так и сложных *web*-приложений.

1.1.4 Выбор библиотеки для разработки пользовательских интерфейсов

Одним из популярных фреймворков на *JavaScript* является *Angular*. Он используется для создания одностраничных приложений и имеет большое сообщество разработчиков. *Angular* основан на компонентах и имеет свою собственную систему шаблонов и директив.

Еще одним фреймворком на *JavaScript* является *Vue.js*. Он также используется для создания одностраничных приложений и имеет простой и интуитивно понятный синтаксис. *Vue.js* также обладает высокой производительностью и может быть легко интегрирован с другими библиотеками и фреймворками.

Ember.js – это еще один фреймворк на *JavaScript*, который используется для создания *web*-приложений. Он имеет свою собственную архитектуру и принципы проектирования, которые облегчают разработку сложных приложений. *Ember.js* также обладает большим сообществом разработчиков и множеством инструментов и плагинов.

Но, возможно, самым популярным и широко используемым решением на *JavaScript* является *React*. Он также используется для создания одностраничных приложений и имеет модульную архитектуру, которая облегчает разработку и поддержку проектов. *React* также обладает большим сообществом разработчиков и множеством инструментов и библиотек для улучшения производительности и функциональности приложений. Выбор решения зависит от конкретных потребностей проекта и предпочтений разработчика, но *React* является одним из наиболее популярных и эффективных на сегодняшний день [3].

1.1.5 Выбор библиотеки для управления состоянием приложения

Управление состоянием приложения является одним из важнейших аспектов при разработке приложений на *React*. Состояние приложения может меняться в зависимости от действий пользователя, полученных данных и других факторов. Без эффективного управления состоянием приложение может стать нестабильным, трудно поддерживаемым и неудобным для пользователей.

Библиотеки управления состоянием, такие как *MobX* и *Redux*, помогают разработчикам создавать стабильные, эффективные и легко поддерживаемые приложения. Они позволяют создавать хранилища, которые содержат все данные приложения и методы для их изменения. Благодаря этому, разработчики могут легко контролировать состояние приложения и обновлять пользовательский интерфейс при необходимости.

Использование библиотек управления состоянием также позволяет упростить процесс разработки, уменьшить количество ошибок и ускорить время разработки. Они предоставляют множество инструментов и функций, которые помогают справиться с различными задачами, связанными с управлением состоянием.

В целом, использование библиотек управления состоянием является необходимым для создания современных и эффективных приложений на *React*. Они помогают улучшить производительность, стабильность и пользовательский опыт, что делает их незаменимыми для любого серьезного проекта.

MobX – это библиотека для управления состоянием, которая использует наблюдение за изменениями данных для автоматического обновления пользовательского интерфейса. Она позволяет создавать реактивные приложения, где изменения данных автоматически отражаются в пользовательском интерфейсе [4].

Redux – это библиотека для управления состоянием, которая

использует однонаправленный поток данных и неизменяемые объекты для управления состоянием приложения. Она позволяет создавать предсказуемые приложения, где изменения состояния происходят только через действия «actions», что делает их более легко отслеживаемыми и тестируемыми.

MobX и *Redux* позволяют создавать хранилища «stores» для управления состоянием приложения. В *MobX* хранилища представляют собой классы, которые содержат данные и методы для изменения этих данных. В *Redux* хранилище представляет собой неизменяемый объект, который содержит все состояние приложения и может быть изменен только через действия «actions», которые передаются в редьюсеры «reducers». Обе библиотеки позволяют подписываться на изменения хранилища и автоматически обновлять пользовательский интерфейс при изменении данных.

Каждая библиотека управления состоянием имеет свои преимущества и недостатки, и выбор между *MobX* и *Redux* может зависеть от конкретных потребностей проекта. Однако, *MobX* обычно считается более простой и интуитивной в использовании, что может быть привлекательным для разработчиков, которые не имеют опыта работы с управлением состоянием.

MobX также предоставляет более гибкий подход к управлению состоянием, позволяя использовать декларативный синтаксис для определения зависимостей и автоматического обновления компонентов при изменении состояния. Это может упростить процесс разработки и сделать код более понятным и легко поддерживаемым.

Кроме того, *MobX* имеет более высокую производительность, чем *Redux*, благодаря более эффективной системе отслеживания изменений состояния. Это может быть особенно важно для крупных и сложных проектов с большим количеством данных.

В целом, выбор между *MobX* и *Redux* зависит от конкретных потребностей проекта и опыта разработчиков. Однако, *MobX* может быть более простым, гибким и производительным в использовании, что делает его

привлекательным выбором для многих проектов на *React*.

1.1.6 Выбор среды разработки

React – это одна из самых популярных библиотек для создания пользовательских интерфейсов. Для разработки приложений на *React* существует множество сред разработки, которые предлагают различные возможности и преимущества.

Одной из самых популярных сред разработки является *Visual Studio Code*. Она предлагает широкий набор инструментов для разработки на *React*, включая подсветку синтаксиса, автодополнение и отладку кода. *Visual Studio Code* также поддерживает множество плагинов, которые позволяют расширять функциональность среды разработки и упрощать процесс разработки.

Другой популярной средой разработки является *WebStorm*. Эта среда разработки предлагает мощный набор инструментов для разработки на *React*, включая поддержку *TypeScript* и *JSX*, автодополнение и отладку кода. *WebStorm* также предлагает широкий набор инструментов для управления зависимостями и сборки проекта.

Еще одной популярной средой разработки является *Atom*. Эта среда разработки предлагает широкий набор инструментов для разработки на *React*, включая подсветку синтаксиса, автодополнение и отладку кода. *Atom* также поддерживает множество плагинов, которые позволяют расширять функциональность среды разработки и упрощать процесс разработки.

Преимущества использования сред разработки на *React* включают улучшенную производительность и эффективность разработки, а также возможность использования множества инструментов и плагинов для упрощения процесса разработки. Кроме того, использование сред разработки на *React* позволяет быстро создавать высококачественные приложения с помощью широкого набора инструментов и возможностей оптимизации.

Проанализировав вышеизложенное выбор остановился на среде

					ПГУ1.090301.29.001 ПЗ	Лист
						16
Изм.	Лист	№ докум.	Подпись	Дата		

разработки *WebStorm*. Она является одной из наиболее продвинутых сред разработки на *React*, которая предлагает широкий набор возможностей для оптимизации процесса разработки и создания качественных приложений.

1.1.7 Выбранные технологии

По результатам проведенных исследований были приняты следующие решения:

- использовать *JavaScript* в качестве языка программирования;
- для создания интерфейса использовать *React*;
- для управления состоянием приложения использовать библиотеку *MobX*;
- разрабатывать программу в среде разработки *WebStorm*.

1.2 Проектирование и разработка

React-проект разворачивается на сервере *nginx*. Для доступа к интерфейсу пользователь должен подключиться к управляющей виртуальной машине.

В интерфейсе доступны списки машин, планов, уведомлений и бэкапов, а также возможность изменять и удалять планы и восстанавливать машины из бэкапов. Также пользователь может управлять машинами, запускать, перезагружать и восстанавливать их.

Для работы с *http*-запросами и данными, полученными из них, в приложении используются *mobx*-хранилища. Это позволяет более эффективно обрабатывать данные и управлять состоянием приложения.

Важно отметить, что *React*-приложение имеет только один *html* файл. Это означает, что для маршрутизации в приложении используется библиотека *React Router*. Она позволяет определить, какой компонент должен быть отображен в зависимости от *URL*-адреса.

Для сборки проекта используется библиотека *webpack*. Она позволяет объединить все компоненты приложения в один файл, что ускоряет его

					ПГУ1.090301.29.001 ПЗ	Лист
						17
Изм.	Лист	№ докум.	Подпись	Дата		

загрузку и уменьшает количество запросов к серверу. Кроме того, *webpack* позволяет использовать различные модули и плагины для оптимизации процесса сборки и улучшения производительности приложения.

1.2.1 Реализация сборки проекта

Для сборки проекта была использована библиотека *webpack*. *Webpack* – это инструмент, который позволяет собирать и упаковывать различные файлы проекта в один или несколько бандлов. Он может использоваться для сборки проектов на *JavaScript*, *TypeScript*, и других языках программирования.

Конфигурация сервера задается в файле «*webpack.config.js*».

Содержимое файла указано ниже:

```
const path = require("path");
const HtmlWebpackPlugin = require("html-webpack-plugin");

module.exports = {
  output: {
    path: path.join(__dirname, "/dist"),
    filename: "bundle.js",
    publicPath: '/',
  },
  devtool: 'source-map',
  plugins: [
    new HtmlWebpackPlugin({
      template: "index.html",
    }),
  ],
  devServer: {
    port: 3030,
    historyApiFallback: true,
  },
  module: {
    rules: [
      {
        test: /\.js$/,
        exclude: /node_modules/,
        use: {
          loader: "babel-loader",
        },
      },
      {
        test: /\.css$/,
        use: [
          {loader: "style-loader"},
          {loader: "css-loader"},
        ],
      },
    ],
  },
}
```

```

    ],
    },
    {
      test: /\. (png|woff|woff2|eot|ttf|svg) $/,
      loader: "url-loader",
      options: {limit: false},
    },
  ],
},
};

```

В данном конфигурационном файле настраиваются параметры сборки проекта. Например, указывается, что собранные файлы должны быть помещены в папку «*dist*», а имя главного файла должно быть «*bundle.js*». Также указывается, что путь к собранным файлам должен начинаться с корневой директории проекта.

Для удобства отладки включается генерация *source-map*. Это позволяет видеть исходный код в инструментах разработчика браузера, даже если используется сжатый и оптимизированный код.

Также используется плагин *HtmlWebpackPlugin*, который генерирует *HTML*-файл на основе шаблона и вставляет в него ссылки на собранные файлы.

Для обработки *JavaScript*-файлов используется загрузчик *babel-loader*, который позволяет использовать современный синтаксис *JavaScript* и транспилировать его в код, понятный более старым браузерам.

Для обработки *CSS*-файлов используются два загрузчика: *style-loader* и *css-loader*. *Style-loader* добавляет стили в *DOM*, а *css-loader* позволяет импортировать *CSS*-файлы в *JavaScript*-код.

Наконец, используется загрузчик *url-loader* для обработки различных типов файлов, таких как изображения и шрифты. Он позволяет включать эти файлы в собранный бандл в виде *base64*-кода или сохранять их в отдельные файлы, в зависимости от их размера.

1.2.2 Реализация маршрутизации страниц

Для управления маршрутизацией в *React*-приложениях используется *React Router*. Он позволяет определить маршруты для различных *URL*-адресов и связать их с соответствующими компонентами *React*. Это позволяет пользователям перемещаться между различными страницами в приложении, не перезагружая страницу.

Помимо объявления маршрутов требуется определить подсказки для каждого маршрута и структуру меню. Для этого была создана константа *PagesData*, которая представляет собой массив из объектов.

Рассмотрим реализацию на примере объекта, отвечающего за страницы «безопасности»:

```
{
  title: "Безопасность", key: "security", icon: <SafetyOutlined/>,
  routes: [
    {
      title: "Устройства",
      path: "/protectedDevices",
      hint: "На этой странице отображаются машинами с
их планами и резервными копиями",
      element: <ProtectedDevicesPage/>
    },
    {
      title: "Планы",
      path: "/plans",
      hint: "Эта страница предназначена для просмотра
планов и их настройки",
      element: <PlansPage/>
    }
  ]
}
```

В данном примере можно увидеть, что объект состоит из таких ключей, как:

- *title* – отвечает за название секции меню;
- *key* – уникальный ключ секции;
- *icon* – изображение секции;
- *routes* – массив отвечающий за пути и связанный с ними контент.

В самом массиве *routes* содержатся следующие данные:

- *title* – название пункта меню;

- *path* – путь к компоненту;
- *hint* – подсказка, отображающаяся на каждой странице;
- *element* – компонент, который будет отображаться по этому пути.

Такая реализация маршрутов позволяет прописывать и структуру меню и связывать пути, подсказки и компоненты в одном месте.

Помимо определения в константе маршрутов нужно их проинициализировать в приложении. Для этого можно использовать хук *useRoutes*, который указывает маршруты приложения.

Он был использован в файле «*routes.js*»:

```
const Router = () => {
  const importantRoutes = [
    {path: "/login", element: <LoginPage/>},
    {path: "/*", element: <NotFoundPage/>}
  ]
  const dynamicRoutes = PagesData.flatMap(pd =>
    pd.routes.map(r => {
      return {path: r.path, element: sessionExist() ?
r.element : <Navigate to='/login'/>}
    })
  )

  return useRoutes([...dynamicRoutes, ...importantRoutes])
};
export default Router
```

В компоненте *Routes* определяются два массива маршрутов: *importantRoutes* и *dynamicRoutes*.

importantRoutes содержит два объекта, каждый из которых представляет собой маршрут и элемент, который должен быть отображен при переходе по этому маршруту. Первый объект определяет маршрут «/login» и элемент *LoginPage*, который должен быть отображен при переходе по этому маршруту. Второй объект определяет маршрут «/*» (любой другой маршрут) и элемент *NotFoundPage*, который должен быть отображен при переходе по любому другому маршруту. Эти маршруты объявлены отдельно от *PagesData* по причине того, что они не отображаются в основном меню.

dynamicRoutes создается путем преобразования массива *PagesData*, который содержит данные о страницах приложения и их маршрутах. Для каждого маршрута из каждой страницы создается объект, который содержит путь и элемент, который должен быть отображен при переходе по этому маршруту. Элемент определяется на основе функции *sessionExist*, которая проверяет, существует ли текущая сессия пользователя. Если сессия существует, то элементом является компонент, связанный с этим маршрутом, если сессия не существует, то происходит перенаправления на страницу авторизации.

Наконец, хук *useRoutes* используется для определения маршрутов и отображения соответствующих элементов при переходе по этим маршрутам. Массив *dynamicRoutes* и *importantRoutes* объединяются и передаются в него. Результатом выполнения компонента *Router* является элемент, содержащий маршруты и связанные с ними элементы.

Чтобы увидеть результат нужно вызвать компонент *Router* в месте, где будет отображаться контент для каждой страницы.

Результатом работы компонента является элемент, отображаемый в соответствии с текущим путем.

Для перехода между страницами и отображения меню были реализованы компоненты:

- *MenuList* – отвечает за отображение меню;
- *MenuItem* – отвечает за отображение секции меню;
- *SubMenuItem* – отвечает за отображения пунктов в секциях меню.

Для перехода между маршрутами используется хук *useNavigate* в файле «*SubMenuItem.js*»:

```
const navigate = useNavigate();  
const onClick = () => navigate(path)
```

Для перехода между страницами в функцию, созданную хуком при вызове *onClick* передается путь и изменяет отображаемый компонент.

Получившееся меню изображено на рисунке 1.1.

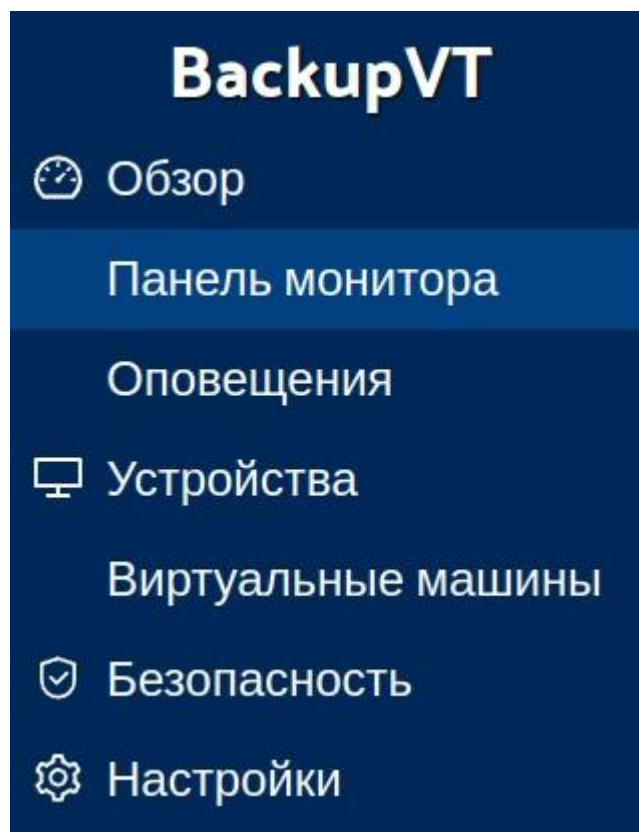


Рисунок 1.1 – Меню приложения

Также, для каждой страницы были объявлены свои подсказки. Для определения страницы был использован хук *useLocation*;

Он используется в компоненте *HeaderBlock*:

```
const location = useLocation()
const [currentRoute, setCurrentRoute] = useState();

useEffect(() => {
  PagesData.forEach(pd => {
    pd.routes.forEach(route => {
      if (route.path === location.pathname)
        setCurrentRoute(route)
    })
  })
}, [location.pathname]);
```

С помощью *useLocation* определяется текущий маршрут для получения текущего пути страницы. Затем, используя хук *useState*, создается состояние *currentRoute*, которое будет содержать информацию о текущем маршруте страницы.

Далее, используя хук *useEffect*, происходит перебор всех маршрутов в объекте *PagesData* и сравнение пути каждого маршрута с текущим путем страницы. Если путь маршрута совпадает с текущим путем, то информация о данном маршруте сохраняется в состоянии *currentRoute* с помощью функции *setCurrentRoute*.

Таким образом, после выполнения данного кода, в состоянии *currentRoute* будет содержаться информация о текущем маршруте страницы.

После чего эта информация передается в компонент *UserActions*, затем в *HintPopover*, отвечающий за отображения подсказки.

Пример подсказки представлен на рисунке 1.2.

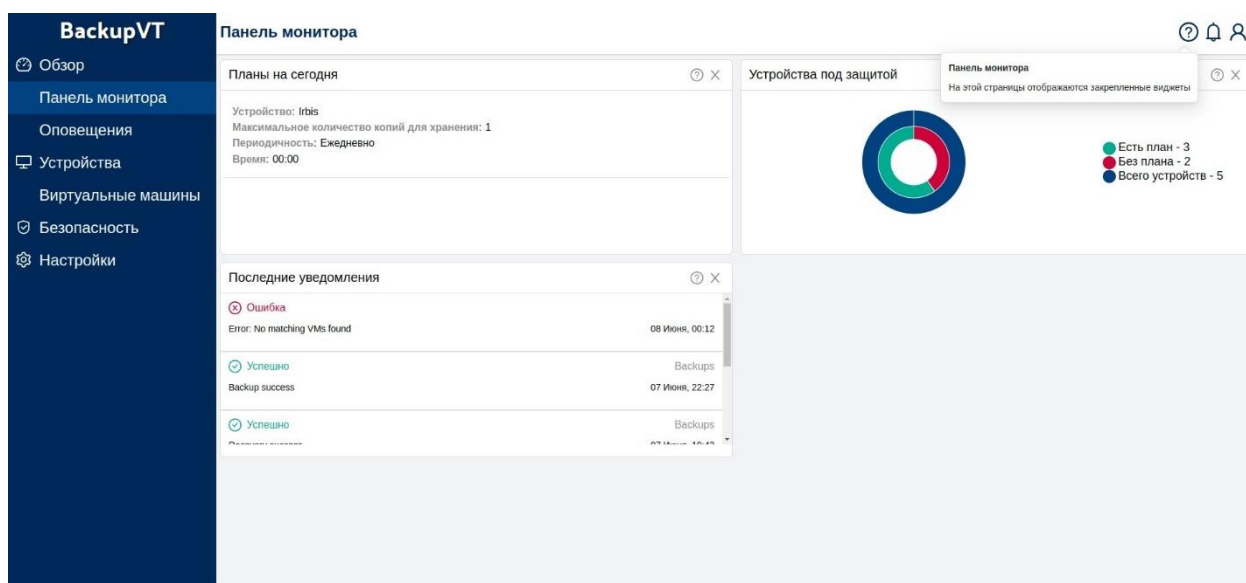


Рисунок 1.2 – Подсказка по странице

Такая реализация маршрутов позволяет удобно их добавлять и удалять в одном месте со всеми соответствующими данными.

1.2.3 Реализация хранилищ

Хранилища *Mobx* представляют собой объекты, которые содержат данные и методы для их изменения. Они могут быть использованы для хранения состояния приложения, такого как данные пользователя, настройки приложения и т.д.

Одним из главных преимуществ использования хранилищ *Mobx* является то, что они позволяют легко управлять состоянием приложения. Вместо того, чтобы передавать данные через множество компонентов, можно использовать хранилища, которые будут доступны в любом месте приложения.

Кроме того, хранилища *Mobx* позволяют легко реагировать на изменения данных. Когда данные в хранилище изменяются, все компоненты, которые используют эти данные, автоматически обновляются. Это позволяет избежать необходимости вручную обновлять компоненты при изменении данных.

На рисунке 1.3 представлена UML-диаграмма классов-хранилищ.

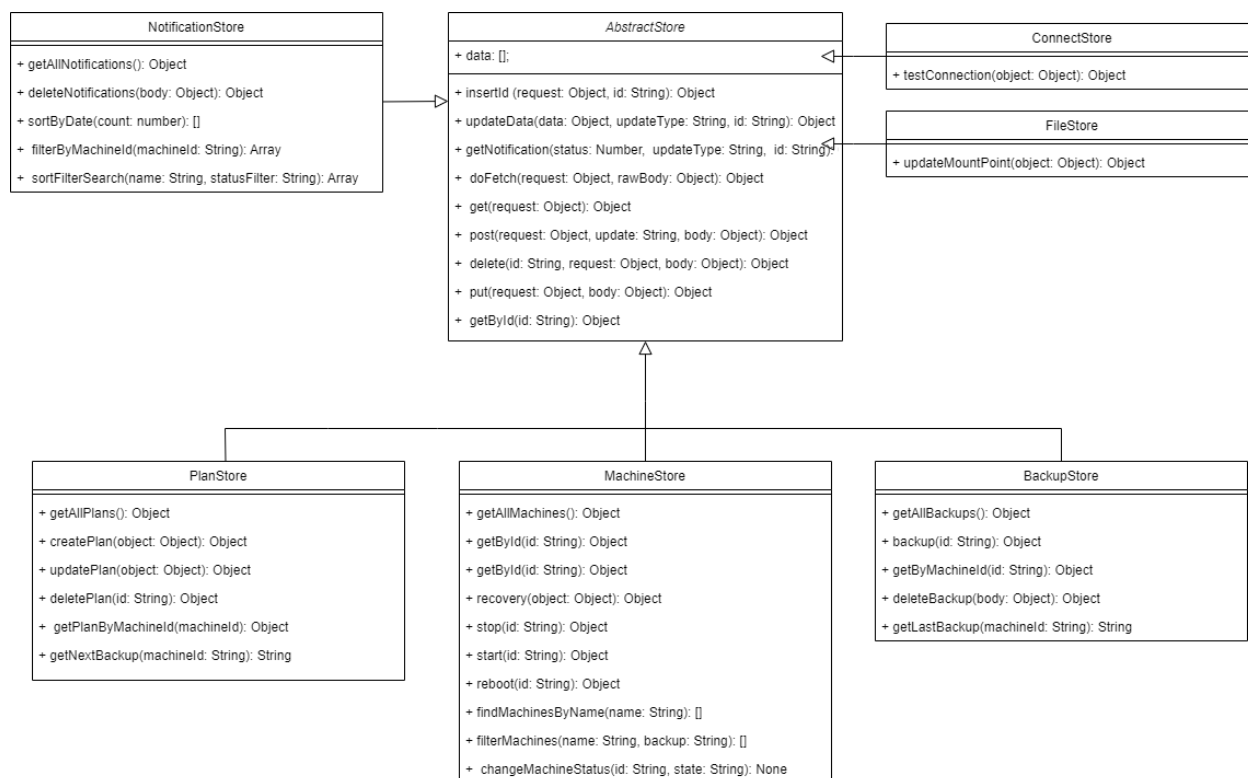


Рисунок 1.3 – UML-диаграмма классов хранилищ

Всего было создано 6 хранилищ, которые представляют собой классы с данными и методами, для отправки соответствующих запросов, сортировки, фильтрации и поиска данных:

- *AbstractStore* – содержит в себе методы описывающие разные типы запросов к серверу и переменную *data*, в которую они записываются;
- *ConnectStore* – отвечает за подключение к управляющей виртуальной машине;
- *NotificationStore* – содержит методы получения и удаления уведомлений;
- *PlanStore* – реализует методы получения, изменения, удаления и поиска планов резервного копирования;
- *MachineStore* – отвечает за получение, остановку, запуск, перезагрузку и поиска виртуальных машин;
- *BackupStore* – используется для получения списка бэкапов и создания резервной копии.

1.2.4 Реализация запросов к серверу

В качестве связи с сервером используются *http*-запросы. Они позволяют отправлять запросы на сервер и получать от него ответы. В *JavaScript* для этого можно использовать метод *fetch*.

Метод *fetch* – это функция, встроенная в браузер *JavaScript*, предназначенная для выполнения сетевых запросов на сервер и получения данных в формате ответа. Этот метод является одним из современных API *web*-разработки, загружаемых в браузер при запуске сценария.

Fetch API предоставляет простой и легкий в использовании интерфейс для получения ресурсов (изображения, скрипты, видео, данные *JSON* и другие форматы) от сервера без обновления (перезагрузки) страницы. Ответы могут быть получены и обработаны в текстовом, *JSON*, *Blob*, *ArrayBuffer* и формате *FormData*.

Fetch на основе обещаний асинхронный метод, что означает, что запрос выполняется асинхронно в фоновом режиме, и *JavaScript* может продолжать работать во время ожидания ответа сервера.

В качестве аргумента *fetch* принимает *URL*-адрес, который нужно загрузить, а также опции запроса, такие как метод *HTTP*, заголовки и параметры. *Fetch* возвращает *Promise*, который разрешается в объект *Response*. Из объекта *Response* вы можете получить различные свойства, такие как текст, *JSON*, обработанный экземпляр зашифрованных данных.

Fetch может быть использован для получения данных с сервера, отправки данных на сервер или даже простого выполнения запросов на сервер. Без использования сторонних библиотек, это наиболее простой и надежный способ выполнения *HTTP*-запросов, доступный в браузере.

Синтаксис метода *fetch* выглядит следующим образом:

```
fetch(url, options);
```

Где *url* – это адрес сервера, к которому нужно отправить запрос, а *options* - это объект с настройками запроса, такими как методы *GET*, *POST*, *PUT*, *DELETE*, заголовки и тело запроса.

Запросы были реализованы в классе *AbstractStore*. Этот класс представляет собой абстрактный класс, который используется в качестве базового класса для других классов хранилища данных. Он содержит общие методы для работы с данными, такие как получение, добавление, обновление и удаление данных.

На рисунке 1.4 представлена диаграмма класса *AbstractStore*.

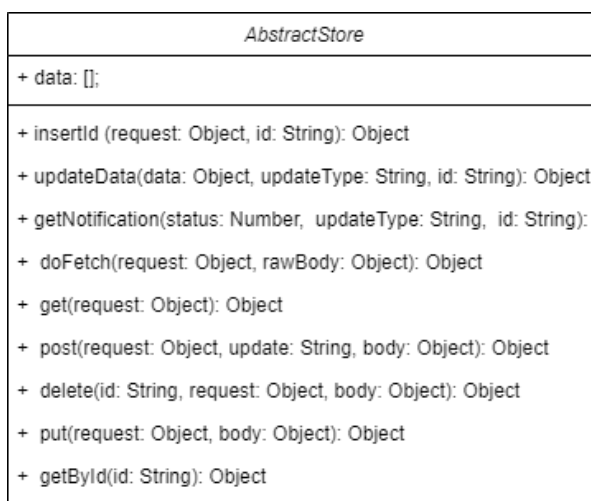


Рисунок 1.4 - Диаграмма класса *AbstractStore*

Сам запрос выполняется в методе *doFetch*:

```
@action
async doFetch(request, rawBody) {
  const session = getSession();
  const body = !request.session ? JSON.stringify(rawBody)
: JSON.stringify({...rawBody, ...session})
  const response = await fetch(request.link, {
    method: request.method,
    headers: request.headers ?? {
      'Accept': 'application/json',
      'Content-Type': 'application/json'
    },
    body
  })

  if (!response.ok) response.clone().json().then(v =>
notifications(Enums.Error, response.status, v ??
response.statusText))
  if (request?.notification &&
request?.notifications?.length > 0)
this.getNotification(response.status, request.notifications)
  else if (request?.notification)
response.clone().json().then(res =>
notifications(Enums.Success, response.status, res))

  return response
}
```

Метод «@action» - это декоратор, предоставляемый библиотекой *MobX*. Он используется для обозначения методов класса, которые меняют состояние объекта или изменяют данные в хранилище состояний.

Метод *doFetch* принимает в качестве параметров объекты *request*, содержащий данные для запроса и *rawBody*, являющийся телом запроса. Для получения текущего подключения, хранящегося в *sessionStorage* используется функция *getSession*. Если в объекте *request* установлено, что для запроса требуется подключение, то объект подключения объединяется с объектом тела запроса и преобразовывается в строку для последующей передачи в *fetch*.

Затем выполняется запрос, если он совершился неудачно, то вызывается функция *notifications* со статусом ошибки и деталями ошибки.

Если запрос выполнен успешно, и в объекте `request` переопределены уведомления, то вызывается функция `getNotification` со статусом успеха и массивом определенных сообщений.

Если уведомления не переопределены и разрешены, то выводится сообщение от сервера с помощью функции `notifications`.

В конце возвращается ответ.

Метод `doFetch` вызывается в методах `get`, `delete`, `post`, `put`, которые определяют обработку данных в зависимости от типа запроса.

Например, метод `delete`:

```
@action
async delete(id, request, body) {
  const response = await this.doFetch(request, body)
  return response.json().then(data =>
this.updateData(data, UpdateEnum.Delete, id));
}
```

Он вызывает функцию `doFetch`, передавая в нее параметры `request` и `body` и записывает ответ запроса в переменную `response`. После получения ответа вызывается метод `updateData`, куда передаются параметры нужные для обновления хранилища.

Код метода `updateData`:

```
@action updateData(data, updateType = UpdateEnum.Full, id) {
  switch (updateType) {
    case UpdateEnum.Full:
      this.data = data
      break;
    case UpdateEnum.Single:
      this.data.push(data)
      break;
    case UpdateEnum.Delete:
      this.data.splice(this.data.findIndex(v => v.id
=== id), 1)
      break;
    case UpdateEnum.Update:
      this.data.splice(this.data.findIndex(v => v.id
=== data.id), 1, data)
      break;
    case UpdateEnum.Response:
      return data
  }
}
```

Этот метод определяет, как обновить переменную *data*, используемую для хранения массивов данных пришедших с сервера, в зависимости от *updateType*, который по умолчанию принимает значение *UpdateEnum.Full*.

Метод *getNotification* – это функция, которая определяет тип уведомления и вызывает функцию *notifications* соответствующего типа. Она используется для обновления состояния уведомлений на основе статуса, который возвращает сервер в ответ на запрос.

Функция принимает два аргумента - статус ответа с сервера и массив объектов уведомлений. Она ищет объект уведомления в массиве *notificationsArray*, у которого значение поля *status* соответствует статусу ответа. Если такой объект найден, вызывается функция *notifications* с параметрами типа уведомления, статуса и описания. Функция *notifications* используется для отображения уведомлений пользователю.

Данный метод используется, когда нужно управлять отображением уведомлений на клиенте на основе статусов ответов, полученных от сервера. Например, в случае ошибки можно отображать сообщение об ошибке, а в случае успешного выполнения запроса - сообщение об успешной операции.

В случае если к ссылке запроса нужно добавить *id*, то используется метод *insertId*, который дописывает его к ссылке запроса.

Методы, которые вызываются в компонентах определены в хранилищах. Например, метод *stop* из класса *MachinesStore*:

```
@action stop(id) {  
  return this.post(this.insertId(requests.stopMachine, id),  
    UpdateEnum.Response) }
```

Данный метод принимает в качестве параметра *id* машины, которую нужно остановить и возвращает значение из объект из метода *post*. В метод пост передается элемент *stopMachine* объекта *requests*, в котором определены параметры запроса. Так как запрос требует указания *id* в ссылке, то он добавляется с помощью метода *insertId*. Также, в параметры передается

UpdateEnum.Response, чтобы данные в хранилище не обновлялись, так как не приходят новые из ответа.

Одним из самых важных моментов в организации запросов в данном приложении является объект *request*, где к каждому ключу относится конкретный объект запроса с параметрами.

Например, ключ *stopMachine* содержит объект:

```
{
    link: hostName + "machine/stop/",
    method: "POST",
    session: true,
    notification: true,
    headers: undefined,
    notifications: [
        {
            status: 200,
            description: "Машина успешно перезагружена",
            type: StatusEnum.Success
        }
    ]
}
```

В объекте имеются следующие поля:

- *link* – содержит ссылку для запроса;
- *method* – определяет метод запроса;
- *session* – добавлять ли объект подключения в запрос;
- *notification* – разрешить ли уведомления от сервера;
- *headers* – заголовки запроса, если *undefined*, то используются заголовки по умолчанию в методе *doFetch*;
- *notifications* – переопределяют сообщения от сервера по статусу.

Данная реализация запросов позволяет удобно их отправлять из компонентов и добавлять новые методы путем создания новых хранилищ и элементов в объекте *routes*.

1.2.5 Реализация авторизации

Чтобы получить доступ к функционалу сайта, нужно подключиться к управляющей виртуальной машине. Для этого нужно заполнить поля на странице авторизации.

Страница авторизации показана на рисунке 1.5.

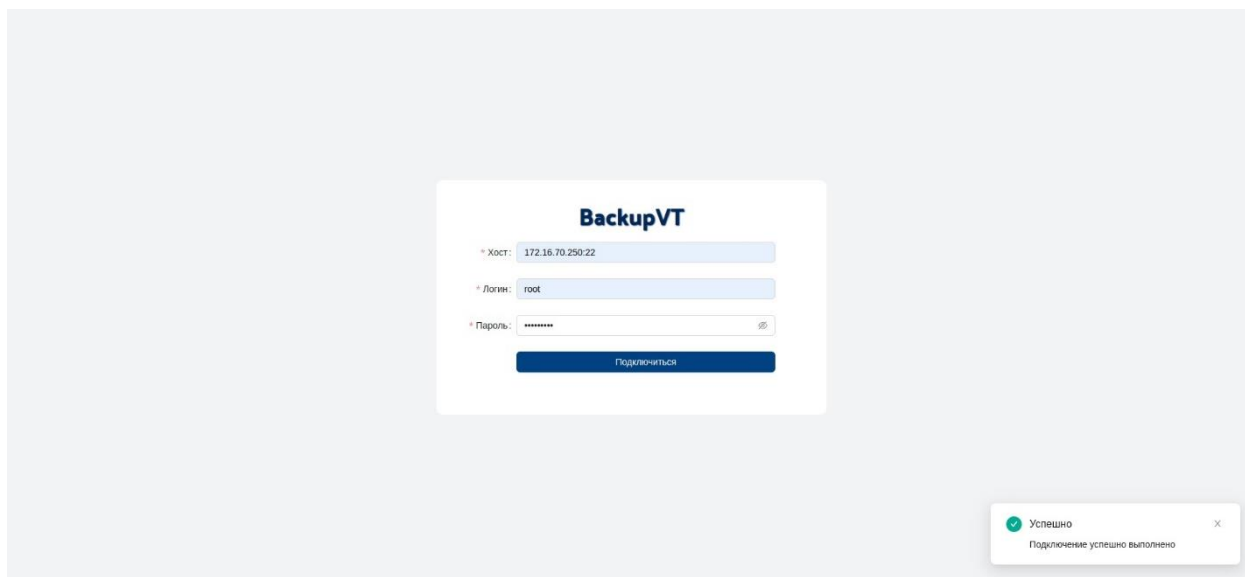


Рисунок 1.5 - Страница авторизации

После авторизации произойдет переход на другую страницу.

Введенные данные сохраняются в *sessionStorage*, чтобы сессия пользователя сохранялась до момента закрытия вкладки браузера. Функции для взаимодействия с ним находятся в файле «utils.js»:

```
export const getSession = () =>
JSON.parse(sessionStorage.getItem('currentSession'));
export const setSession = (session) =>
sessionStorage.setItem('currentSession',
JSON.stringify(session));
export const sessionExist = () => !!getSession();
```

getSession – возвращает данные текущей сессии.

setSession – устанавливает данные текущей сессии.

sessionExist – проверяет, существует ли текущая сессия.

Чтобы отключиться от сессии можно воспользоваться всплывающим меню в правом верху экрана и нажать кнопку отключиться или закрыть вкладку браузера, после чего *sessionStorage* очиститься.

Меню сессии показано на рисунке 1.6.

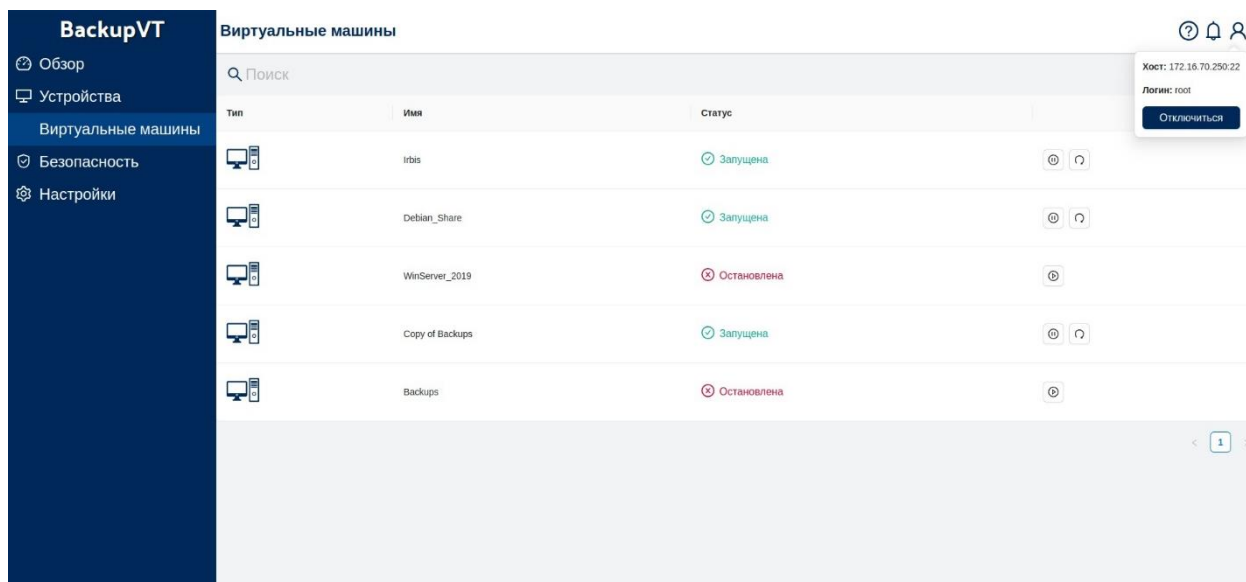


Рисунок 1.6 - Меню сессии

1.2.6 Реализация планов

Одним из самых важных аспектов приложения является взаимодействие с планами резервного копирования. Они автоматизируют выполнения бэкапов тем самым обеспечивают безопасность виртуальных машин.

Для настройки планов был разработан ряд компонентов.

Одним из таких компонентов является *Calendar*. Он представляет собой *html* таблицу, которая отрисовывается с помощью методов документа.

При монтировании компонента происходит инициализация календаря в хуке *useEffect* и функции *identDate*:

```
useEffect(() => {
  const year = date.getFullYear();
  identDate()
  setListOfYears(numberRange(year - 2, year + 5))
}, [])
const identDate = () => {
  const year = date.getFullYear();
  const month = date.getMonth();
  const dayOfWeek = date.getDay();
  const day = date.getDate();
  setCurrDate(customObjectDateCreate(dayOfWeek, day,
month, year))
}
```

После изменения значений *currDate*, *plansOnDays*, *machines*, *visibleMachines* календарь обновляется, и вызывает функцию *renderCalendar*:

```
const renderCalendar = (month, year, lastActive) => {
  let array = collectDays(month, year);
  let tB = document.getElementById('table-body');
  let tr = document.createElement('tr');
  let i = 0;
  for (i; i < array.length; i++) {
    if (array[i].dayOfWeek === 1)
      tr = document.createElement('tr');
    tr.appendChild(createCell(array[i], lastActive))
    if (array[i].dayOfWeek === 0)
      tB.appendChild(tr);
  }
}
```

Функция *renderCalendar* создает и отображает календарь для переданного месяца *month* и года *year*, при этом последний активный элемент календаря будет равен *lastActive*.

Сначала она вызывает функцию *collectDays*, которая создает массив объектов дней месяца. Затем она ищет элемент с *id* «*table-body*», создает новый элемент *tr*, затем в цикле перебирает массив всех дней и добавляет элементы *td* с помощью функции *createCell*, которая возвращает дочерний элемент для ячейки таблицы.

После этого происходит проверка, является ли текущий день первым днем недели. Если это так, создается новый элемент *tr*, иначе в текущий элемент *tr* добавляется новый элемент *td*.

Наконец, если текущий день является последним днем недели, элемент *tr* добавляется в родительский элемент таблицы.

Функция *collectDays*:

```
const collectDays = (month, year) => {
const lastDateOfMonth = new Date(year, month + 1, 0).getDate(),
lastDayOfMonth = new Date(year, month, lastDateOfMonth).getDay(),
firstDayOfMonth = new Date(year, month, 1).getDay(),
lastDateOfLastMonth = new Date(year, month, 0).getDate();

const resMonth = []

// прошлый месяц
if (firstDayOfMonth !== 1) {
```

```

        let tempFirstDay = firstDayOfMonth - 1;
        if (firstDayOfMonth === 0)
            tempFirstDay = 6
        let i = tempFirstDay;
        let lastMonth = lastDateOfLastMonth;
        for (i; i > 0; i--) {
            resMonth.push(customObjectDateCreate(i === 7 ? 0 : i,
lastMonth, currDate.month === 0 ? 11 : currDate.month - 1,
currDate.month === 0 ? currDate.year - 1 : currDate.year))
            lastMonth--;
        }
        resMonth.reverse();
    }

    // текущий месяц
    let i = 1;
    let tempFirstDay = firstDayOfMonth;
    for (i; i <= lastDateOfMonth; i++) {
        resMonth.push(customObjectDateCreate(tempFirstDay, i,
currDate.month, currDate.year))
        tempFirstDay === 6 ? tempFirstDay = 0 : tempFirstDay++;
    }

    // следующий месяц
    if (lastDayOfMonth !== 0) {
        let i = lastDayOfMonth === 6 ? 0 : lastDayOfMonth + 1;
        let dateOfFutureMonth = 1;
        for (i; i <= 7; i++) {
            resMonth.push(customObjectDateCreate(i === 7 ? 0 : i,
dateOfFutureMonth, currDate.month === 11 ? 0 : currDate.month +
1, currDate.month === 11 ? currDate.year + 1 : currDate.year))
            dateOfFutureMonth++;
        }
    }
    return resMonth
}

```

Функция *collectDays* используется для создания массива объектов дней месяца для отображения на календаре в компоненте *Calendar*.

Она принимает аргументы *month* и *year*, определяя последнее число месяца *lastDateOfMonth*, последний день месяца *lastDayOfMonth*, первый день месяца *firstDayOfMonth* и последнее число предыдущего месяца.

Функция затем создает пустой массив и добавляет даты предыдущего месяца, текущего месяца и следующего месяца в массив, чтобы заполнить календарь.

Сначала в массив добавляются даты предыдущего месяца, при этом цикл начинается с последней даты предыдущего месяца и заканчивается до первого дня текущего месяца.

Затем в массив добавляются даты текущего месяца, при этом цикл начинается с первого дня текущего месяца и заканчивается последним днем.

Наконец, добавляются даты следующего месяца, если последний день текущего месяца не является воскресеньем. Это происходит при помощи цикла, который начинается со следующего дня после последнего дня текущего месяца и заканчивается с семидневным циклом.

Функция *createCell*:

```
const createCell = (item, lastActive) => {
  // создание ячейки
  let cell = document.createElement('td');
  cell.dataset.value = JSON.stringify(item);
  if (lastActive && lastActive.date === item.date &&
lastActive.month === item.month && lastActive.year === item.year
&& lastActive.dayOfWeek === item.dayOfWeek) {
    cell.classList.add("calendar-active-day");
    setLastActiveDay(cell)
  } else
    cell.className = currDate.month !== item.month ?
"calendar-day calendar-day-of-other-months" : "calendar-day";

  // заголовок ячейки
  let cellHeader = document.createElement('div');
  itIsCurDay(item.year, item.month, item.date) ?
cellHeader.className = "cur-cell-header" : cellHeader.className
= "cell-header";
  let cellDate = document.createTextNode(item.date);
  cellHeader.appendChild(cellDate);

  // тело ячейки
  let cellBody = document.createElement('div')
  cellBody.className = "cell-body";
  const thisDayPlan = planAtThisDay(item, plansOnDays,
true);
  if (thisDayPlan.length > 0)
    for (let plan of thisDayPlan) {
      let tempText =
document.createTextNode(machines.find(machine => machine.uuid
=== plan.machines)?.name);
      let tempP = document.createElement('p');
      tempP.appendChild(tempText);
      cellBody.appendChild(tempP);
    }
}
```

```

    }

    // сборка ячейки
    cell.appendChild(cellHeader);
    cell.appendChild(cellBody);
    return cell
}

```

Функция *createCell* создает и возвращает *HTML*-элемент *td*, представляющий ячейку календаря. Входные параметры функции - объект *item* и объект *lastActive*.

В теле функции создаются два вложенных *HTML*-элемента - заголовок *cellHeader* и тело *cellBody*. Заголовок содержит дату, а тело - информацию о задачах на этот день. Если на этот день есть записи о задачах, в *cellBody* создаются несколько *HTML*-элементов *p* с именами машин, связанных с этими задачами.

Эта функция также устанавливает класс ячейке в зависимости от того, принадлежит ли день текущему месяцу или не текущему месяцу. Если день является последней активной ячейкой, ему также устанавливается класс «*calendar-active-day*», и вызывается функция *setLastActiveDay*. Функция возвращает готовую ячейку.

Для переключения месяцев по нажатию на дни не относящиеся к текущему была реализована функция *setDateOnClick*:

```

const setDateOnClick = (event) => {
  const targetParent = event.target.parentNode;
  if (targetParent.nodeName === 'TD') {
    let cellData =
JSON.parse(targetParent.getAttribute("data-value"))
    if (targetParent !== lastActiveDay) {
      targetParent.classList.add("calendar-active-
day");
      targetParent.classList.remove("calendar-day");
      if (lastActiveDay) {
        lastActiveDay?.classList.remove("calendar-
active-day");
        lastActiveDay?.classList.add("calendar-
day");
      }
      setLastActiveDay(targetParent);
      setDayInfoDrawerVisible(true);
    }
  }
}

```

```

setCurrDate(customObjectDateCreate(cellData.dayOfWeek,
cellData.date, cellData.month, cellData.year));
    }
}
}

```

Эта функция устанавливает обработчик событий *click* на ячейки календаря, чтобы отслеживать клики пользователя по ячейкам.

Внутри функции происходит проверка того, что кликнутый элемент является родительским элементом для *td*, инициирующим клик. Если это так, извлекаются данные ячейки с помощью метода *JSON.parse* из атрибута «*data-value*» этой ячейки.

Если текущая ячейка не соответствует последней активной ячейке, устанавливается ей класс «*calendar-active-day*», удаляется класс «*calendar-day*» и устанавливается в качестве последней активной ячейки с помощью функции *setLastActiveDay*.

Если ранее уже была выбрана активная ячейка, ей удаляется класс «*calendar-active-day*», и ему добавляется класс «*calendar-day*».

Также функция вызывает *setCurrDate*, которая устанавливает текущую дату, на основании данных ячейки, и *setDayInfoDrawerVisible*, которая устанавливает видимость элемента информации о дне, чтобы он был видимым после выбора ячейки.

Компонент *Calendar* представлен на рисунке 1.7.

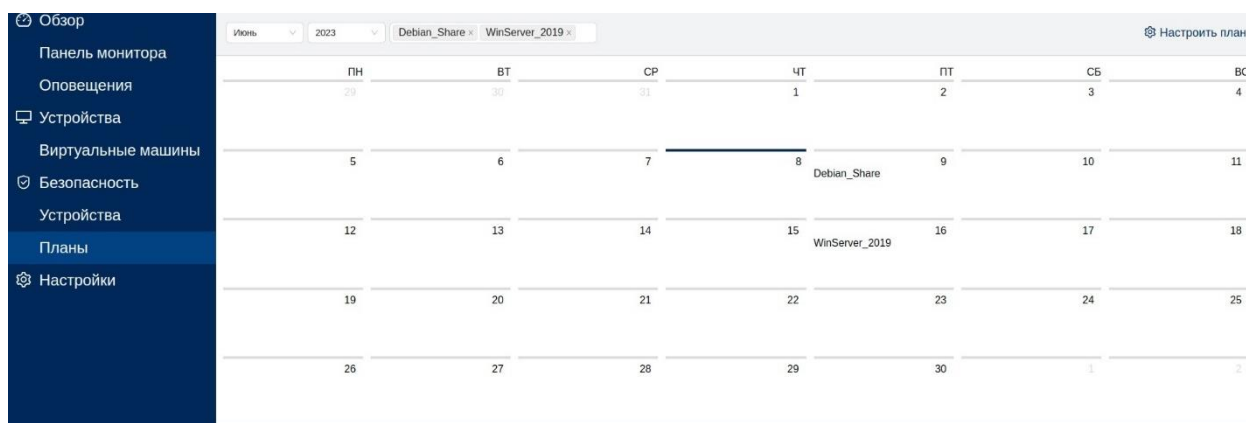


Рисунок 1.7 – Компонент *Calendar*

Для настройки планов были реализованы компоненты *EveryDayPlan*, *EveryWeekPlan*, *EveryMonthPlan*, *PlanSettings* и *PlanSettingsModal*.

Модально окно для настройки планов показано на рисунке 1.8.

Рисунок 1.8 – Модально окно настройки планов

Также, для просмотра планов были реализованы компоненты *PlanCard* и *PlanCardList*.

Для просмотра планов относительно машин была разработана страница с таблицей, содержащей информацию о наличии плана и следующей дате бэкапа.

Следующая дата бэкапа вычисляется с помощью метода *getNextBackup* класса *PlanStore*:

```
getNextBackup(machineId) {
    const plan = this.getPlanByMachineId(machineId)
    if (!plan)
        return "План не задан"
    let nextEventDate;
    const currentDate = new Date();
    const currentYear = currentDate.getFullYear();
    const currentDayOfWeek = currentDate.getDay();

    if (plan.dayOfMonth !== undefined) {
        nextEventDate = nextEventDateSwitch(currentDate,
        plan.dayOfMonth);
        nextEventDate.setDate(nextEventDate.getDate() +
        (plan.dayOfWeek - nextEventDate.getDay() + 7) % 7);

        if (nextEventDate.getFullYear() !== currentYear)
            nextEventDate.setFullYear(compoundYears(nextEventDate.getFullYear(), currentYear))
    }
}
```

```

        if (currentDate.getTime() >=
nextEventDate.getTime()) {
            nextEventDate = nextEventDateSwitch(currentDate,
plan.dayOfMonth, 1);
            nextEventDate.setDate(nextEventDate.getDate() +
(plan.dayOfWeek - nextEventDate.getDay() + 7) % 7);

            if (nextEventDate.getFullYear() !== currentYear)

nextEventDate.setFullYear(compoundYears(nextEventDate.getFullYea
r(), currentYear))
        }
        } else if (plan.dayOfWeek !== undefined) {
            let daysUntilNextEvent = currentDayOfWeek >=
plan.dayOfWeek ? (7 - currentDayOfWeek + plan.dayOfWeek) :
(plan.dayOfWeek - currentDayOfWeek);
            nextEventDate = new Date(currentDate.getTime() +
daysUntilNextEvent * 24 * 60 * 60 * 1000);

            if (nextEventDate.getMonth() !==
currentDate.getMonth()) {
                nextEventDate.setDate(1);
                nextEventDate.setMonth(nextEventDate.getMonth()
+ 1);
            }
        } else {
            nextEventDate = new Date();
        }

        return nextEventDate.getDate() + " " +
MonthsEnum.AlternateName[nextEventDate.getMonth()]
    }

```

Эта метод используется для определения даты следующего бэкапа для машины с указанным идентификатором.

Она начинается с получения плана на основании идентификатора машины с помощью метода *getPlanByMachineId*. Если план не был найден, функция возвращает строку "План не задан".

Затем функция определяет день следующего события в зависимости от того, задан ли день месяца, день недели или не задано ничего. Для каждого случая она определяет дату следующего события, добавляет нужное количество дней и проверяет, является ли следующее событие в том же году, что и текущая дата. Если следующее событие уже произошло, функция

сначала определяет дату следующего события для следующего месяца и дня месяца согласно плану.

Результатом функции является строка, состоящая из даты месяца и альтернативного названия месяца для указанной даты.

Страница «устройства» показана на рисунке 1.9.

<div>ВаскупVT</div> <div>Обзор</div> <div>Панель монитора</div> <div>Оповещения</div> <div>Устройства</div> <div>Виртуальные машины</div> <div>Безопасность</div> <div>Устройства</div> <div>Планы</div> <div>Настройки</div>	Устройства					<div>?</div> <div>🔔</div> <div>👤</div>	
	<div>🔍 Поиск</div>						
	Тип	Имя	План	Количество копий	Последний бэкап	Следующий бэкап	
	<div>🖥️</div>	Irbis	✔️ Создан	1	6 Июня	8 Июня	
	<div>🖥️</div>	Debian_Share	✔️ Создан	1	2 Июня	9 Июня	
	<div>🖥️</div>	WinServer_2019	✔️ Создан	1	17 Мая	16 Июня	
	<div>🖥️</div>	Copy of Backups	❌ Отсутствует	1	29 Мая	План не задан	
	<div>🖥️</div>	Backups	❌ Отсутствует	1	7 Июня	План не задан	
							<div>< 1 ></div>

Рисунок 1.9 – Страница «устройства»

При клике на строку таблицы можно увидеть выдвигающийся контейнер с информацией, связанной с этой машиной на рисунке 1.10.

BackupVT

Обзор

Панель монитора

Оповещения

Устройства

Виртуальные машины

Безопасность

Устройства

Планы

Настройки

Устройства

Поиск

Тип	Имя	План	Количество копий
	Irbis	✔ Создан	1
	Debian_Share	✔ Создан	1
	WinServer_2019	✔ Создан	1
	Copy of Backups	❌ Отсутствует	1
	Backups	❌ Отсутствует	1

WinServer_2019 - Планы

Следующий бэкап: 16 Июня

Настроить план

Устройство: WinServer_2019

Максимальное количество копий для хранения: 1

Периодичность: Ежемесячно

Неделя месяца: 3

День недели: Пятница

Время 0: 06:07

Время 1: 05:06

Рисунок 1.10 – Выдвигающийся блок о машине

1.2.7 Реализация уведомлений

Уведомления представляют собой сообщения, приходящие от сервера. Все уведомления можно посмотреть на странице «оповещения», показанной на рисунке 1.11.

Машина	Дата	Тип	Сообщение	
	08 Июня, 00:12	❌ Ошибка	Error: No matching VMs found	❌
Backups	07 Июня, 22:27	✅ Успешно	Backup success	❌
Backups	07 Июня, 19:43	✅ Успешно	Recovery success	❌
Backups	07 Июня, 18:51	✅ Успешно	Backup success	❌
Backups	07 Июня, 08:50	✅ Успешно	Recovery success	❌
	06 Июня, 14:13	✅ Успешно	Backup success	❌
ntfs	06 Июня, 03:30	✅ Успешно	Backup success	❌
	02 Июня, 11:20	❌ Ошибка	Another task executable!	❌
Debian_Share	02 Июня, 02:37	✅ Успешно	Backup success	❌
	01 Июня, 21:00	❌ Ошибка	Error: No matching VMs found	❌

Рисунок 1.11 – Страница с оповещениями

Сообщения можно фильтровать по статусу и искать по имени с помощью *sortFilterSearch* класса *NotificationStore*:

```
sortFilterSearch(name, statusFilter) {
    let notifications = this.sortByDate()?.map(not => {
        return {
            ...not,
            machineName:
machineStore.getById(not.machine)?.name
        }
    })

    notifications = name === "" ? notifications :
notifications.filter(not => not?.machineName &&
not?.machineName?.toLowerCase().includes(name.toLowerCase()))
    notifications = !statusFilter ? notifications :
notifications.filter(not => not?.typeMessage === statusFilter)
    return notifications
}
```

Сначала функция вызывает метод *sortByDate* у объекта уведомлений и преобразует каждый объект уведомления с использованием метода *map*.

Внутри метода *map* происходит извлечение имени машины из объекта *machineStore*, соответствующего машине из объекта уведомления, и добавление имени машины вместе со всеми остальными свойствами объекта уведомления.

Затем функция фильтрует уведомления на основе строкового значения *name*. Если *name* пусто, она возвращает все уведомления. В противном случае, функция сравнивает полученные имена машин с *name* с помощью метода *toLowerCase* для обеспечения безусловного сравнения и фильтрует объекты уведомлений, в которых имя машины содержит заданную строку *name*.

Затем функция фильтрует уведомления на основе заданного флага *statusFilter*. Если флаг не задан, возвращаются все уведомления. Если флаг задан, возвращаются объекты уведомлений, у которых свойство *typeMessage* равно значению фильтра *statusFilter*.

Результатом функции является массив объектов уведомлений, отфильтрованных по имени машины и флагу фильтрации по статусу.

Пример фильтрации и поиска показан на рисунке 1.12.

Машина	Дата	Тип	Сообщение
Backups	07 Июня, 22:27	Успешно	Backup success
Backups	07 Июня, 19:43	Успешно	Recovery success
Backups	07 Июня, 18:51	Успешно	Backup success
Backups	07 Июня, 08:50	Успешно	Recovery success

Рисунок 1.12 – Пример фильтрации

Также, на этой странице реализован функционал удаления уведомлений, после чего выводится уведомления о действии.

Удаление уведомлений показано на рисунке 1.13.

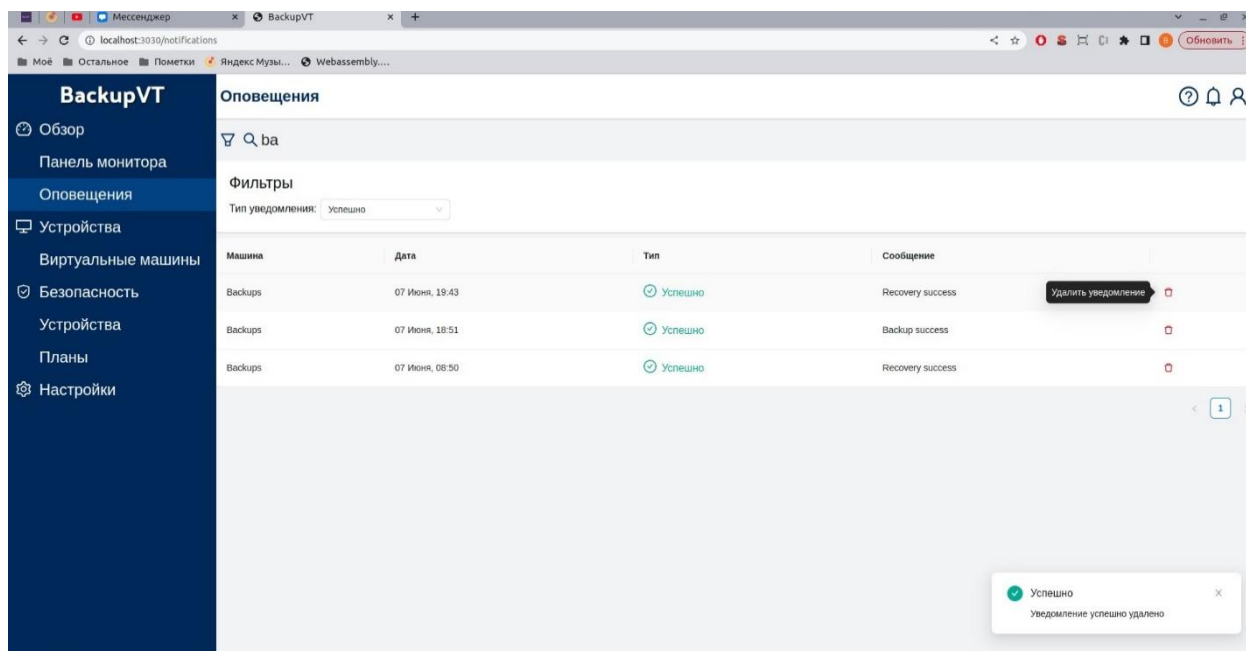


Рисунок 1.13 – Удаление уведомления

Помимо таблицы реализованы списки уведомлений.

Для фильтрации последних уведомлений используется метод *sortByDate* класса *NotificationStore*:

```
sortByDate(count) {
    const sortedData = this.data.slice().sort((a, b) => new
    Date(b.data) - new Date(a.data))
    return !count ? sortedData : sortedData.slice(0, count)
}
```

Этот метод сначала сортирует массив уведомлений по дате, и если передан параметр *count*, то отсекает первые уведомления в количестве равном параметру.

Список уведомлений используется во всплывающих уведомлениях и всплывающем блоке машины. Они показаны на рисунках 1.14 и 1.15 соответственно.

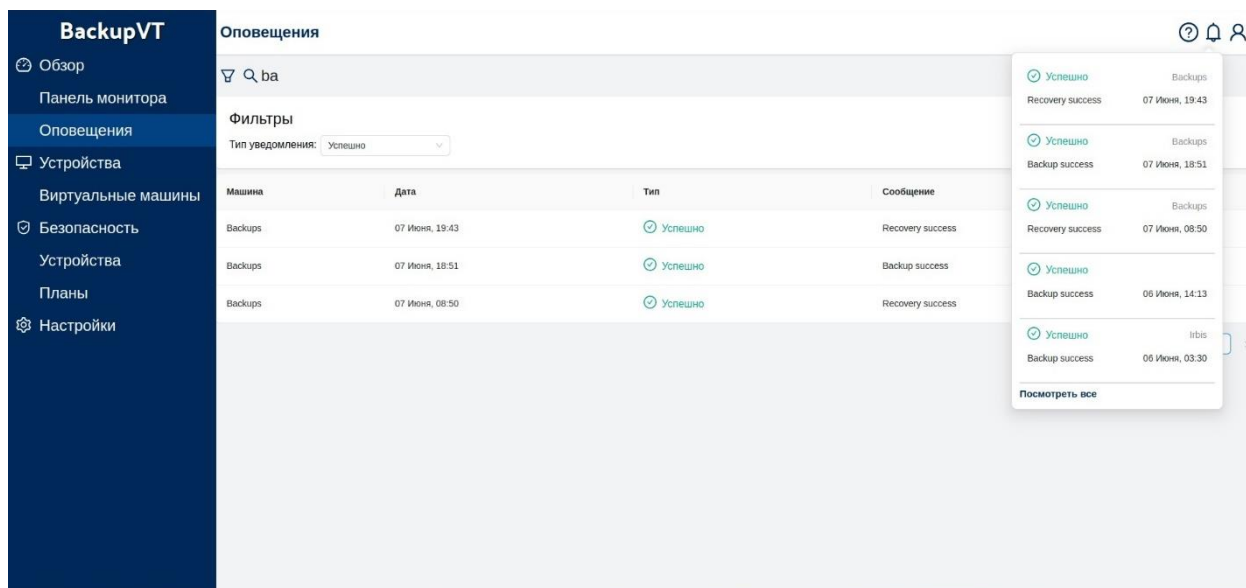


Рисунок 1.14 – Всплывающие уведомления

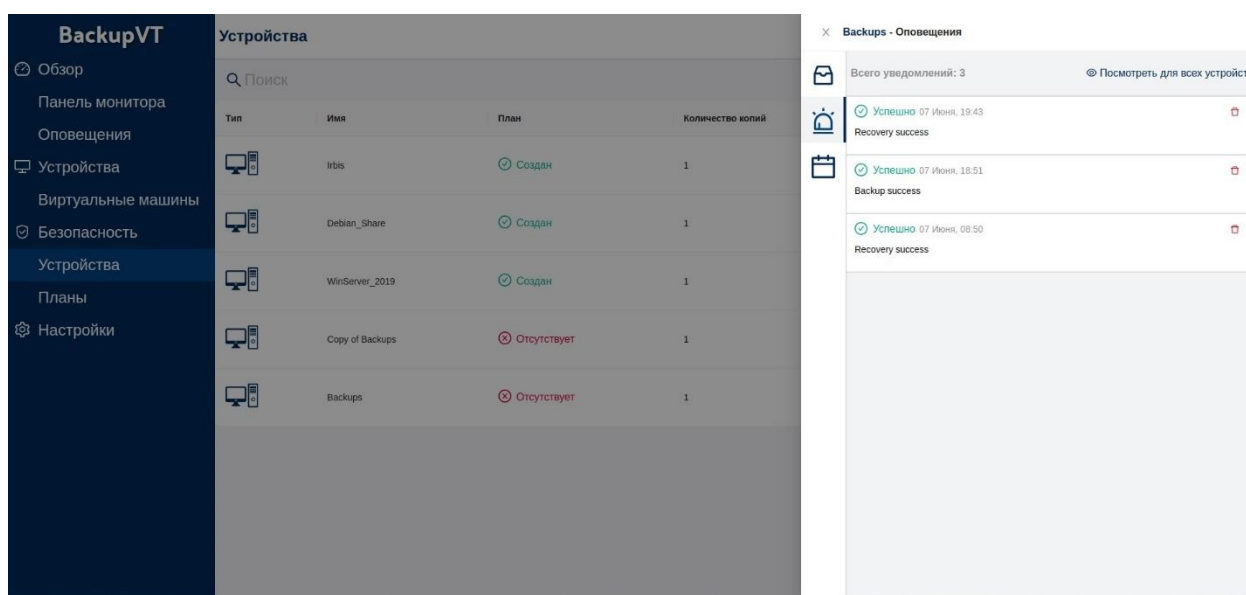


Рисунок 1.15 – Уведомления о машине

1.2.8 Реализация виджетов

Виджеты – это компоненты пользовательского интерфейса, которые отображают краткую информацию о машинах кластера. В них могут отображаться, оповещения, статусы машин, наличие резервных копий, планы на сегодня.

Страница «виджеты» представлена на рисунке 1.16.

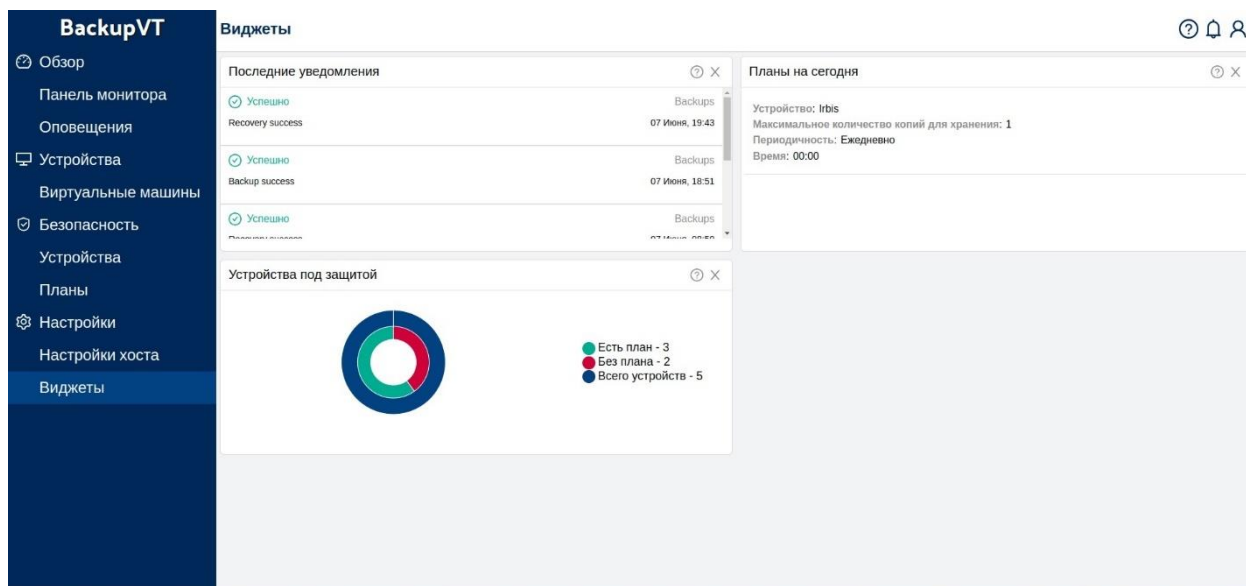


Рисунок 1.16 - Страница «виджеты»

Каждый виджет можно закрепить или открепить от «панели монитора», для этого нужно нажать на соответствующую кнопку в углу виджета.

Эта логика вынесена в отдельный хук, потому что используется в нескольких компонентах.

Код хука *useWidgets*:

```
const useWidgets = () => {
  const [widgets, setWidgets] = useState([]);

  useEffect(() => {
    const w = getWidgetsStore()
    if (w && w?.length > 0) {
      const widgetsFromStore = Array.from(JSON.parse(w))
      const checkedWidgets =
        widgetsFromStore.filter(widget => WidgetsData.some(v => v.key
        === widget))
      if (checkedWidgets.length !==
        widgetsFromStore.length)
        setWidgetsStore(checkedWidgets)
        setWidgets(checkedWidgets)
    }
  }, [])

  const getWidgetsStore = () =>
    localStorage.getItem("widgets")
  const setWidgetsStore = (array) =>
    localStorage.setItem("widgets", JSON.stringify(array))
  const getWidgetContent = (key) => WidgetsData.find(widget =>
    widget.key === key)
```

```

const pinned = (key) => widgets.includes(key)

const updateWidgets = (array) => {
  setWidgets(array)
  setWidgetsStore(array)
}

const handlePin = (key) => {
  let w = [...widgets, key]
  updateWidgets(w)
}

const handleUnpin = (key) => {
  let w = widgets.filter(p => p !== key)
  updateWidgets(w)
}

return [widgets, setWidgets, pinned, getWidgetContent,
handlePin, handleUnpin, setWidgetsStore]
}
export default useWidgets;

```

Хук использует функцию *useState* для создания состояния *widgets*, представляющего список выбранных пользователем виджетов. Хук также использует *useEffect* для загрузки списка виджетов из хранилища при монтировании страницы. При этом проверяется существуют ли ключи, записанные в хранилище в *WidgetsData*, чтобы избежать проблем с приложением при обновлении списка виджетов.

Хук предоставляет несколько функций для управления виджетами:

- *getWidgetsStore* - функция для получения списка виджетов из хранилища;
- *setWidgetsStore* - функция для сохранения массива виджетов в хранилище;
- *getWidgetContent* - функция для получения содержимого виджета по ключу;
- *pined* - функция, которая возвращает true, если виджет с заданным ключом был закреплен пользователем;
- *updateWidgets* - функция для обновления списка выбранных пользователем виджетов в состоянии и в хранилище;

- *handlePin* - функция для добавления нового виджета к списку выбранных пользователем виджетов;

- *handleUnpin* - функция для удаления виджета из списка выбранных пользователем виджетов.

Хук возвращает массив с функциональностью, определенной выше: список выбранных пользователем виджетов, функцию для установки нового состояния виджетов, функцию для проверки, является ли виджет закрепленным, функцию для получения содержимого виджета, функцию для добавления виджета в список выбранных, функцию для удаления виджета из списка выбранных и функцию для сохранения списка виджетов в хранилище.

Также, у каждого виджета есть символ вопроса, при наведении на который отображается подсказка для чего нужен виджет.

Отображение подсказки у виджета показано на рисунке 1.17.

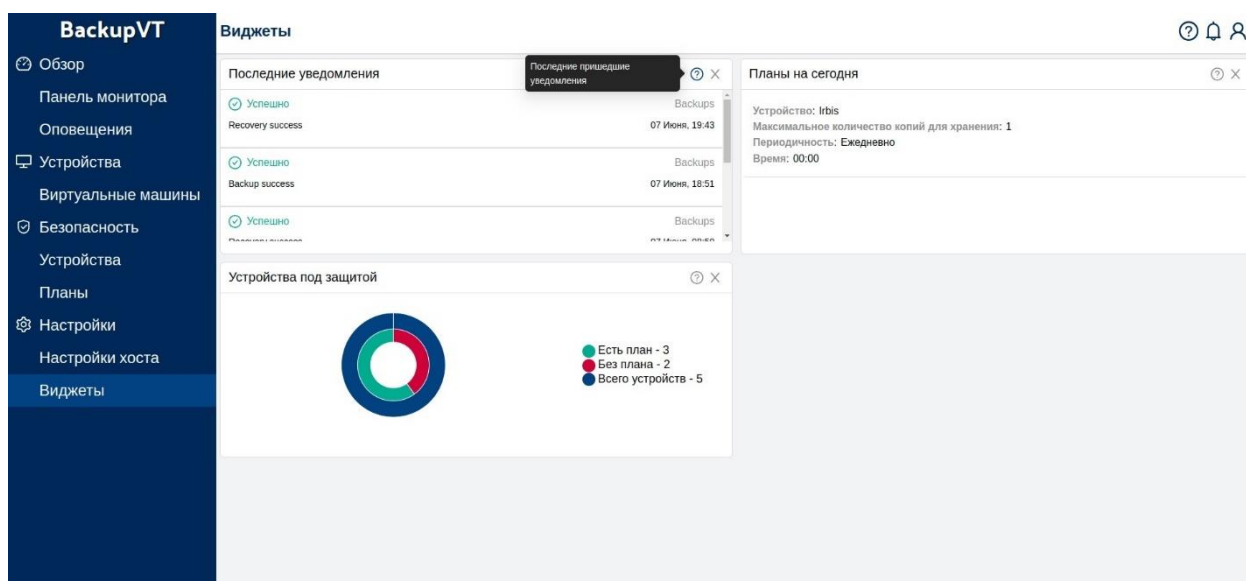


Рисунок 1.17 – Отображение подсказки виджета

Все закрепленные виджеты отображаются на странице «панель монитора». За контент на ней отвечает компонент *DashboardPage*. В нем, также, реализована логика перетаскивания элементов.

Компонент *DashboardPage*:

```
const [widgets, setWidgets, pinned, getWidgetContent, handlePin, handleUnpin, setWidgetsStore] = useWidgets();
```

```

const dragzoneRef = useRef(null);
const blocksRef = useRef([]);

const [draggedBlock, setDraggedBlock] = useState(null);
const [dragOffsetX, setDragOffsetX] = useState(0);
const [dragOffsetY, setDragOffsetY] = useState(0);

function handleDragStart(event) {
  const {target, clientX, clientY} = event;
  setDraggedBlock(target);
  setDragOffsetX(clientX - target.offsetLeft);
  setDragOffsetY(clientY - target.offsetTop);
}

function handleDragEnd() {
  setDraggedBlock(null);
  setWidgetsStore(widgets);
  setDragOffsetX(0);
  setDragOffsetY(0);
}

function handleDragOver(event, wrapperIndex) {
  event.preventDefault();
  const dragIndex =
blocksRef.current.indexOf(draggedBlock);
  const tmpBlocks = widgets.slice();
  const block = tmpBlocks.splice(dragIndex, 1)[0];
  if (block) {
    tmpBlocks.splice(wrapperIndex, 0, block);
    setWidgets(tmpBlocks);
  }
}

return <div className={widgets-page ${draggedBlock ? "drag-
zone" : ""}} ref={dragzoneRef}>
  {widgets.map((widget, index) => {
    const content = getWidgetContent(widget);
    const wrapperRef = element =>
(blocksRef.current[index] = element);
    return <WidgetWrapper
      widget={content}
      key={widget}
      refLink={wrapperRef}
      draggable={true}
      handlePin={handlePin}
      handleUnpin={handleUnpin}
      pinned={pined(widget)}
      onDragStart={handleDragStart}
      onDragEnd={handleDragEnd}
      index={index}

```

```

        active={draggedBlock                                     ===
        blocksRef.current[index] && "active"}
        onDragOver={ (event) => handleDragOver(event,
        index) }

    />
  )))
</div>

```

Компонент содержит функцию *handleDragStart*, которая вызывается, когда пользователь начинает перетаскивать виджет. Данная функция записывает перетаскиваемый блок и задает *dragOffsetX* и *dragOffsetY*, чтобы вычислить расстояние между курсором мыши и верхним левым углом блока.

Функция *handleDragEnd* вызывается, когда пользователь завершает перетаскивание блока. Она обновляет список виджетов, сохраненных в *localStorage*, и сбрасывает значения *draggedBlock*, *dragOffsetX* и *dragOffsetY*.

Функция *handleDragOver(event, wrapperIndex)* вызывается, когда пользователь перемещает блок над другим блоком. Она изменяет список виджетов, перемещает перетаскиваемый блок на указанную позицию и обновляет *widjets*.

Компонент также содержит функции *handlePin* и *handleUnpin*, которые устанавливают и снимают флажок закрепления виджета. Есть также *pined*, который проверяет, закреплен ли виджет.

Возвращаемое значение компонента является корневым элементом, содержащим виджеты в элементах-обертках, обработчики перетаскивания, а также классы стилей, отражающие состояние перетаскивания.

Перетаскивание элемента показано на рисунках 1.18 - 1.20.

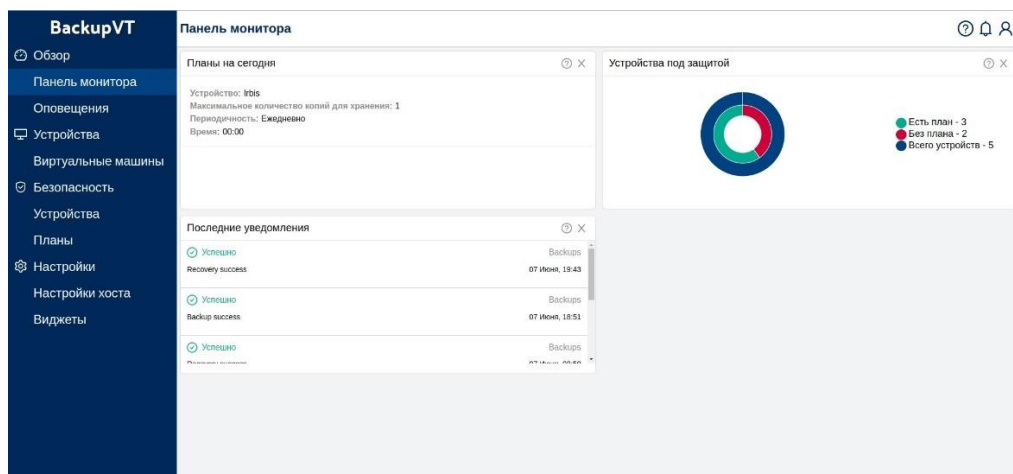


Рисунок 1.18 – Состояние виджетов до начала перемещения

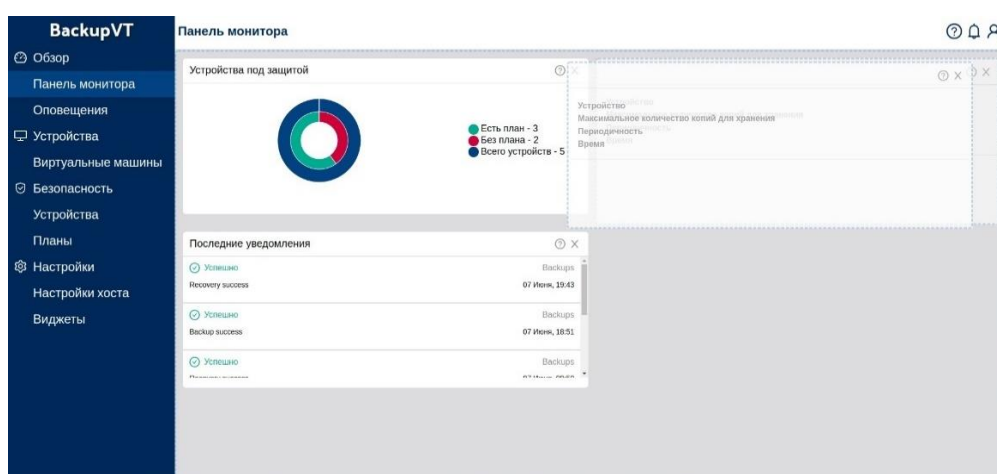


Рисунок 1.19 - Состояние виджетов во время перемещения

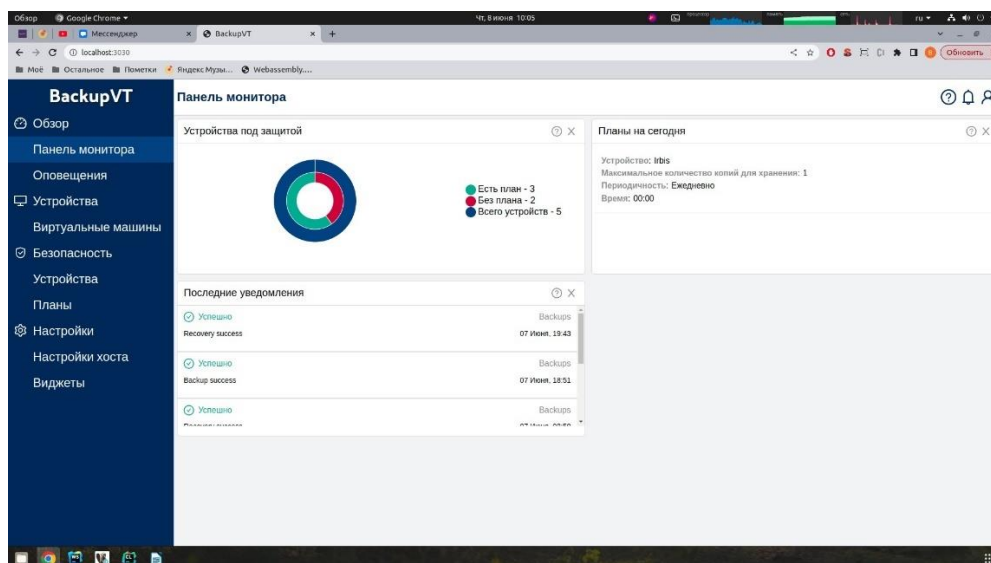


Рисунок 1.20 – Состояние виджетов после перемещения

1.2.9 Реализация резервных копий

Резервные копии отображаются отфильтрованными по машине в выдвигающемся контейнере на странице «устройства».

Список резервных копий показан на рисунке 1.21.

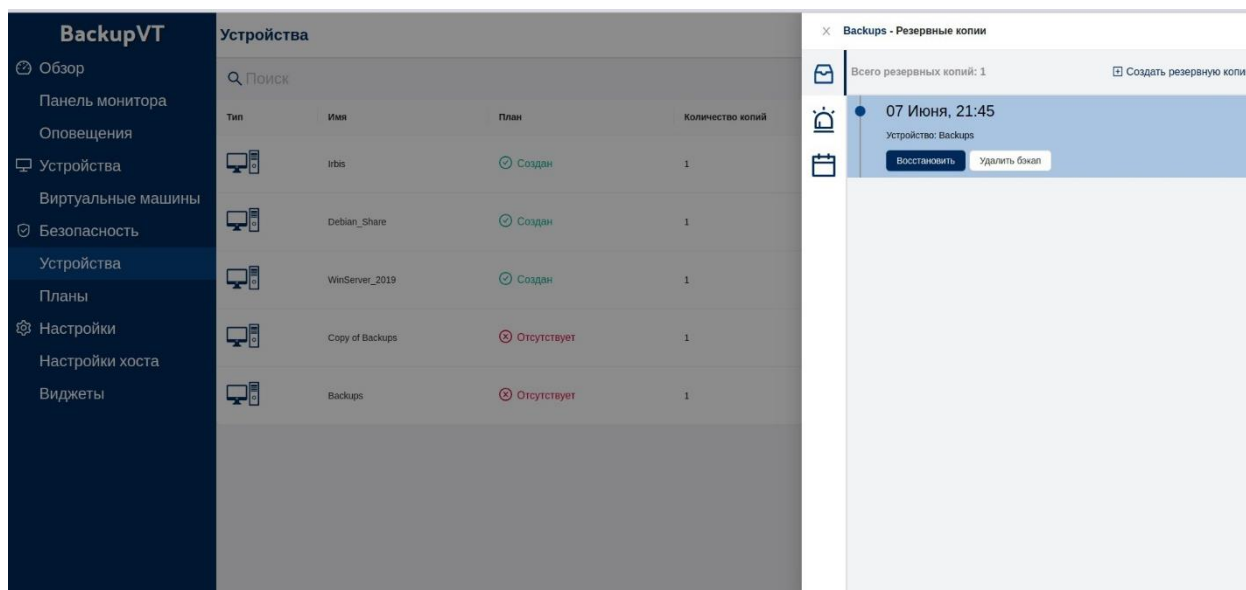


Рисунок 1.21 – Список резервных копий

В этом же списке можно выбрать копию для удаления или восстановления.

Удаление копии показано на рисунке 1.22.

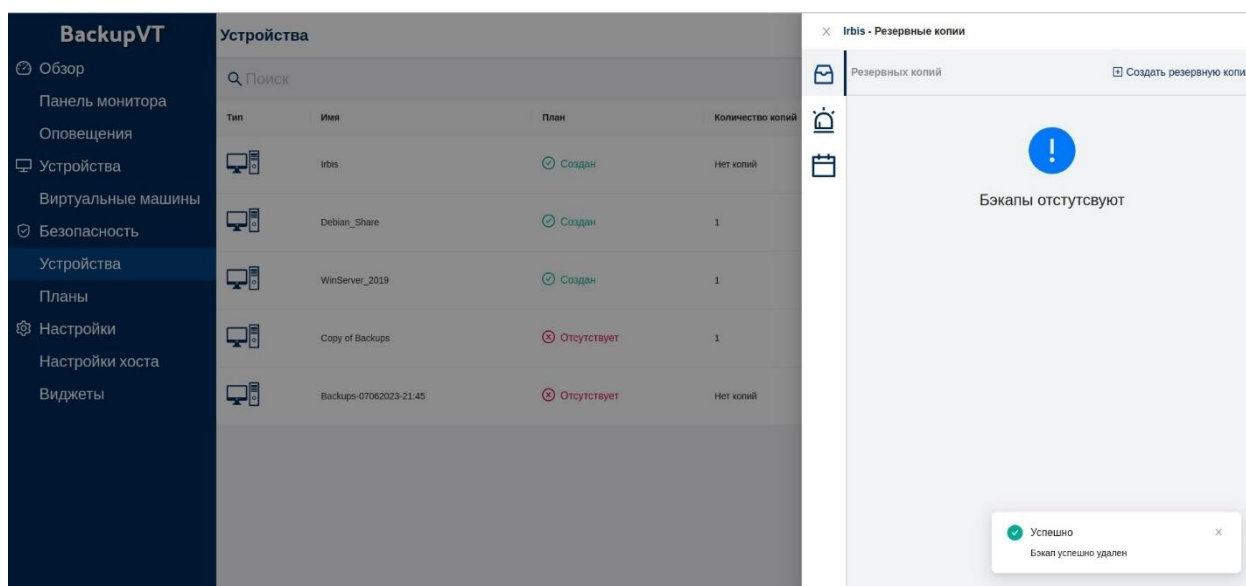


Рисунок 1.22 – Удаление копии

Для восстановления и удаления копий требуется собрать название файла, для этого используются функции:

```
const string = (name, date, id) => `${name}-${
fromBackupDate(date, "recovery")}-${id}.xva`
export const fromBackupDate = (backupDate, type) => {
  const data = new Date(backupDate);
  const date = data.getDate();
  const day = data.getDay();
  const month = data.getMonth();
  const year = data.getFullYear();
  const hours = data.getHours();
  const minutes = data.getMinutes();

  switch (type) {
    case "timelineTitle":
      return `${zeroBefore(date)}
${MonthsEnum.AlternateName[month]},
${zeroBefore(hours)}:${zeroBefore(minutes)}`
    case "recovery":
      return `${zeroBefore(date)}${zeroBefore(month +
1)}${zeroBefore(year)}-
${zeroBefore(hours)}:${zeroBefore(minutes)}`
    default:
      return {date, day, month, year, hours, minutes}
  }
}
```

Функция *string* принимает три параметра: *name*, *date*, *id*. Она используется для создания уникального имени файла.

Функция *fromBackupDate* принимает два параметра: *backupDate* и *type*. Она используется для преобразования даты резервной копии в нужный формат. В зависимости от значения параметра *type* функция может возвращать дату в трех форматах: *timelineTitle*, *recovery* или объект *date*, *day*, *month*, *year*, *hours*, *minutes*.

Функция использует объект *Date* для преобразования строки *backupDate* в дату. Затем она извлекает значения даты, времени и дня недели из объекта *Date*. Функция возвращает значения, отформатированные с помощью функции *zeroBefore*, которая добавляет ведущий ноль, если число меньше 10.

В зависимости от параметра *type* возвращает разный формат даты.

В противном случае функция возвращает объект, содержащий значения даты, времени, дня недели и т.д.

После создания копии или восстановления приходят соответствующие уведомления, показанные на рисунке 1.23.

Машина	Дата	Тип	Сообщение
Backups	08 Июня, 07:08	Успешно	Recovery success
Backups	07 Июня, 19:43	Успешно	Recovery success
Backups	07 Июня, 18:51	Успешно	Backup success

Рисунок 1.23 – Уведомления о бэкапе и восстановлении

Также в списке бэкапов можно запустить моментальный бэкап, если он будет успешно запущен, то появится уведомление как на рисунке 1.24.

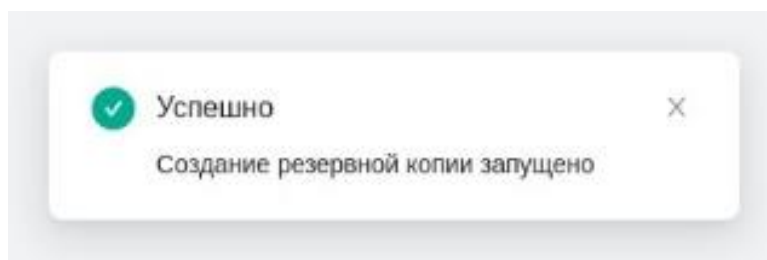


Рисунок 1.24 – Уведомления о начале бэкапа

1.2.10 Реализация устройств

Для отображения устройств, была создана таблица. В ней отображается имя устройства, ее статус и действия: запуск, остановка, перезагрузка.

Страница с устройствами показана на рисунке 1.25.

ВаскурVT		Виртуальные машины																									
<ul style="list-style-type: none"> Обзор Панель монитора Оповещения Устройства Виртуальные машины Безопасность Устройства Планы Настройки 		<div>Поиск</div> <table> <tr> <th>Тип</th><th>Имя</th><th>Статус</th><th></th></tr> <tr> <td>Имя</td><td>Имя</td><td>Запущена</td><td>Остановить</td></tr> <tr> <td>Online_2021</td><td>Online_2021</td><td>Запущена</td><td></td></tr> <tr> <td>WebServer_2021</td><td>WebServer_2021</td><td>Остановлена</td><td></td></tr> <tr> <td>Copy of Backups</td><td>Copy of Backups</td><td>Запущена</td><td></td></tr> <tr> <td>Backups</td><td>Backups</td><td>Остановлена</td><td></td></tr> </table>		Тип	Имя	Статус		Имя	Имя	Запущена	Остановить	Online_2021	Online_2021	Запущена		WebServer_2021	WebServer_2021	Остановлена		Copy of Backups	Copy of Backups	Запущена		Backups	Backups	Остановлена	
Тип	Имя	Статус																									
Имя	Имя	Запущена	Остановить																								
Online_2021	Online_2021	Запущена																									
WebServer_2021	WebServer_2021	Остановлена																									
Copy of Backups	Copy of Backups	Запущена																									
Backups	Backups	Остановлена																									

Рисунок 1.25 – Страница «виртуальные машины»

2 Охрана труда

Каждый день персональные компьютеры помогают людям решать различные задачи, будь то работа, учеба или развлечения. Однако, несмотря на все преимущества, использование компьютеров может нанести вред здоровью пользователя и окружающим людям. Чтобы минимизировать риск травм и заболеваний, необходимо строго соблюдать рекомендации по безопасности, такие как ограничение времени работы на компьютере, правильное расположение рабочего места, соблюдение норм освещения и уровня шума в помещении. Кроме того, важно следить за своим здоровьем и проводить регулярные перерывы для разминки и глазной гимнастики.

2.1 Основные вредоносные факторы

Правила выполнения работ с ПК обязаны обеспечивать минимизацию основных рисков, перечисленных в ГОСТ 12.0.003-2015. В первую очередь к ним относятся:

- повышенная температура отдельных элементов персонального компьютера;
- монотонность трудового процесса;
- высокий уровень зрительных нагрузок;
- вероятность поражения статическим электричеством, генерируемым техникой;
- недостаточная освещенность в зоне выполнения операций у компьютера;
- высокий уровень напряженности электрического и магнитного полей в зоне установки компьютера;
- другие факторы.

Полностью избавиться от вредного воздействия компьютера на здоровье невозможно, однако необходимо принимать меры для сокращения

					ПГУ1.090301.29.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		55

его воздействия на человека. Важно обеспечить возможность восстановления нормального самочувствия и работоспособности после работы за компьютером, не прибегая к дополнительным средствам, таким как массаж или медикаментозная терапия.

2.2 Общие требования к организации и оборудованию рабочих мест

Действующее законодательство выделяет основные нормы, которых требует работа у компьютера в постоянном режиме:

- рабочие места изолированы друг от друга и обеспечиваются организованным воздухообменом. Если занимающие их сотрудники заняты творческой деятельностью, следует оборудовать их места перегородками высотой 1,5-2 метра;

- конструкция кресла регулируется, чтобы обеспечить соответствие его положения индивидуальным физическим параметрам сотрудника. Его поверхность должна быть нескользящей и гарантировать возможность легкой чистки;

- конструкция стола должна обеспечивать рациональное и удобное размещение имеющегося оборудования и иметь коэффициент отражения, не превышающий 0,7.

2.3 Требования к помещениям с компьютерами

Для обеспечения безопасности при работе за компьютером устанавливаются определенные требования к помещениям, в которых происходит работа. Эти требования включают следующие меры:

- помещения должны иметь окна, через которые проходит естественный свет, которые должны быть ориентированы преимущественно на север или северо-запад. Если работа происходит в помещениях без естественного освещения, то необходимо провести высокоточные расчеты,

чтобы обосновать эквивалентность организованного искусственного освещения нормативам естественного света;

- окна в помещении должны быть оборудованы устройствами для регулирования яркости естественного освещения, такими как шторы, жалюзи и прочее;

- площадь одного рабочего места сотрудника должна составлять не менее 6 квадратных метров, а если в работе используются плоские мониторы компьютеров жидкокристаллического или плазменного типа, то не менее 4,5 квадратных метров;

- внутренняя отделка помещений должна производиться с использованием материалов, имеющих санитарно-эпидемиологические заключения. Максимальный коэффициент отражения используемых при отделке материалов составляет для потолка – 0,8, для стен – 0,6, для пола – 0,5;

- помещения должны находиться вдали от силовых кабелей, высоковольтных трансформаторов и другого оборудования, способного создавать помехи для офисной техники. В комнатах необходимо организовать защитное заземление, обеспечивающее безопасную работу компьютеров.

2.4 Правила размещения монитора

Согласно обязательным нормам техники безопасности, при размещении монитора необходимо соблюдать следующие требования:

- расстояние между столами составляет не менее 2 метров, а между боковыми поверхностями мониторов – не менее 1,2 метра;

- экран используемого монитора находится на расстоянии 0,6-0,7 метра от глаз.

При этом электростатический потенциал дисплея не должен превышать 500 В.

					ПГУ1.090301.29.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		57

2.5 Режим труда и отдыха при использовании компьютера

В инструкции по охране труда содержатся правила безопасного поведения и техники выполнения трудовых операций. В зависимости от категории выполняемой работы и уровня зрительной нагрузки, которая возникает у пользователя в течение смены, существует определенное время перерывов, которое зависит от продолжительности работы:

- при восьмичасовой смене рекомендуемый перерыв от 30 до 70 минут;
- при двенадцатичасовой смене рекомендуемый перерыв от 70 до 120 минут.

					ПГУ1.090301.29.001 ПЗ	Лист
						58
Изм.	Лист	№ докум.	Подпись	Дата		

Заключение

В ходе выполнения данной работы изучены способы резервного копирования кластеров виртуальных машин, получены знания по взаимодействию с *xcp-ng* и проанализированы существующие *web*-приложения для бэкапов.

Также получены навыки написания клиента *web*-приложений на языке программирования *JavaScript*, изучены и применены методы написания проектов с помощью фреймворка *React*. Освоен функционал библиотеки для управления состояниями приложения *MobX*.

Во время выполнения работы был разработан собственный клиент *web*-приложения, который позволяет пользователю взаимодействовать с виртуальными машинами, планами, резервными копиями, точками монтирования и уведомлениями.

В дальнейшем можно усовершенствовать данную программу, добавив отслеживание прогресса восстановления или бэкапа машины. Добавить новые виджеты, связанные с информацией о загрузенности ресурсов управляющей машины и свободного места в хранилищах.

Список использованных источников

1. Документация к *xcr-ng*. [Электронный ресурс] / Режим доступа <https://xcr-ng.org/about.html>, свободный (дата обращения 27.04.2023).
2. Дуглас Крокфорд. *JavaScript: сильные стороны*. Питер, 2013. – 176 с.
3. Документация к React. [Электронный ресурс] / Режим доступа <https://ru.legacy.reactjs.org/>, свободный (дата обращения 3.05.23).
4. Документация к MobX. [Электронный ресурс] / Режим доступа <https://mobx.js.org/>, свободный (дата обращения 3.05.23).

					ПГУ1.090301.29.001 ПЗ	Лист
						60
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А

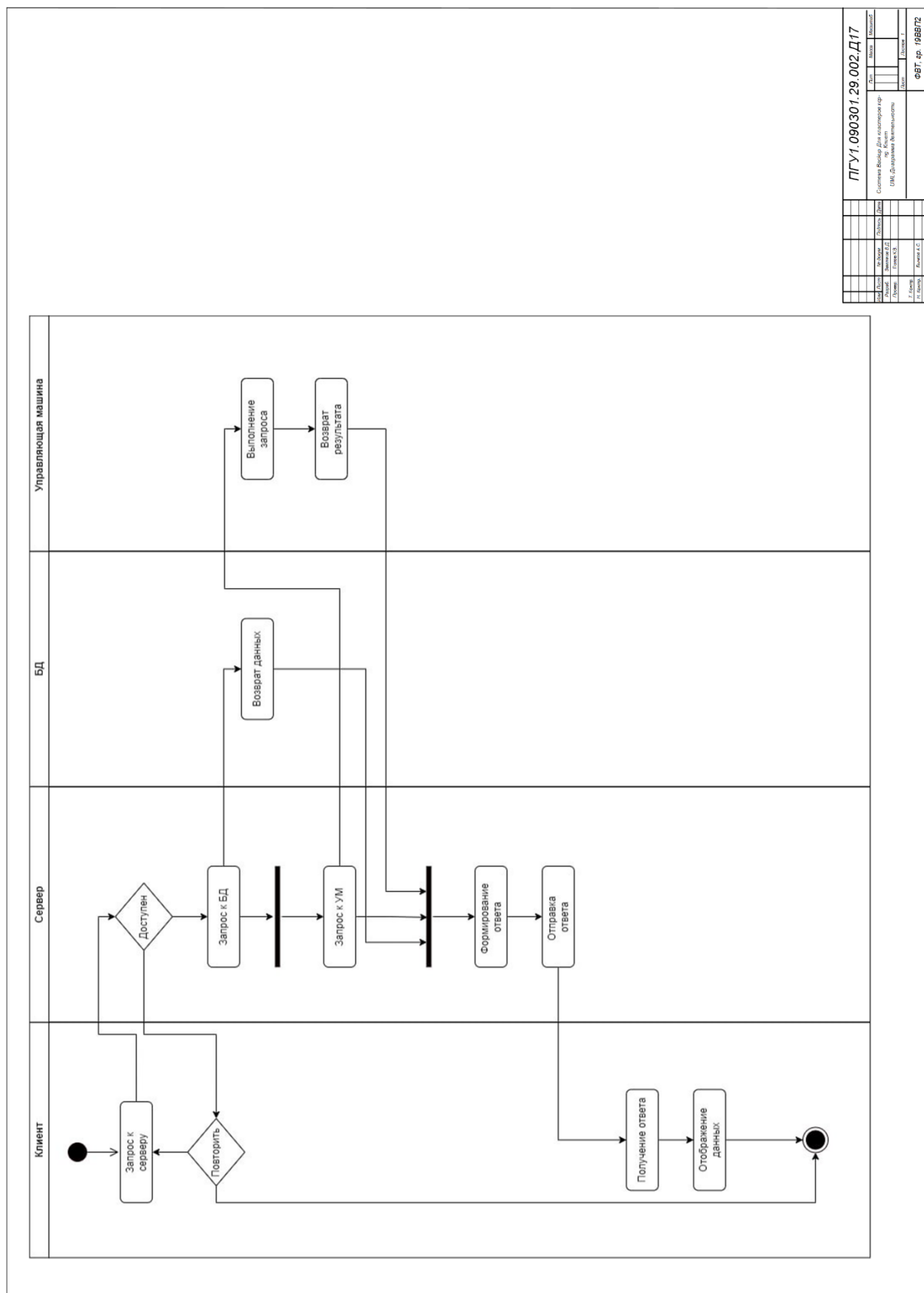
(Обязательное)

Система *Васкир* для кластеров *хср-ng*. Клиент

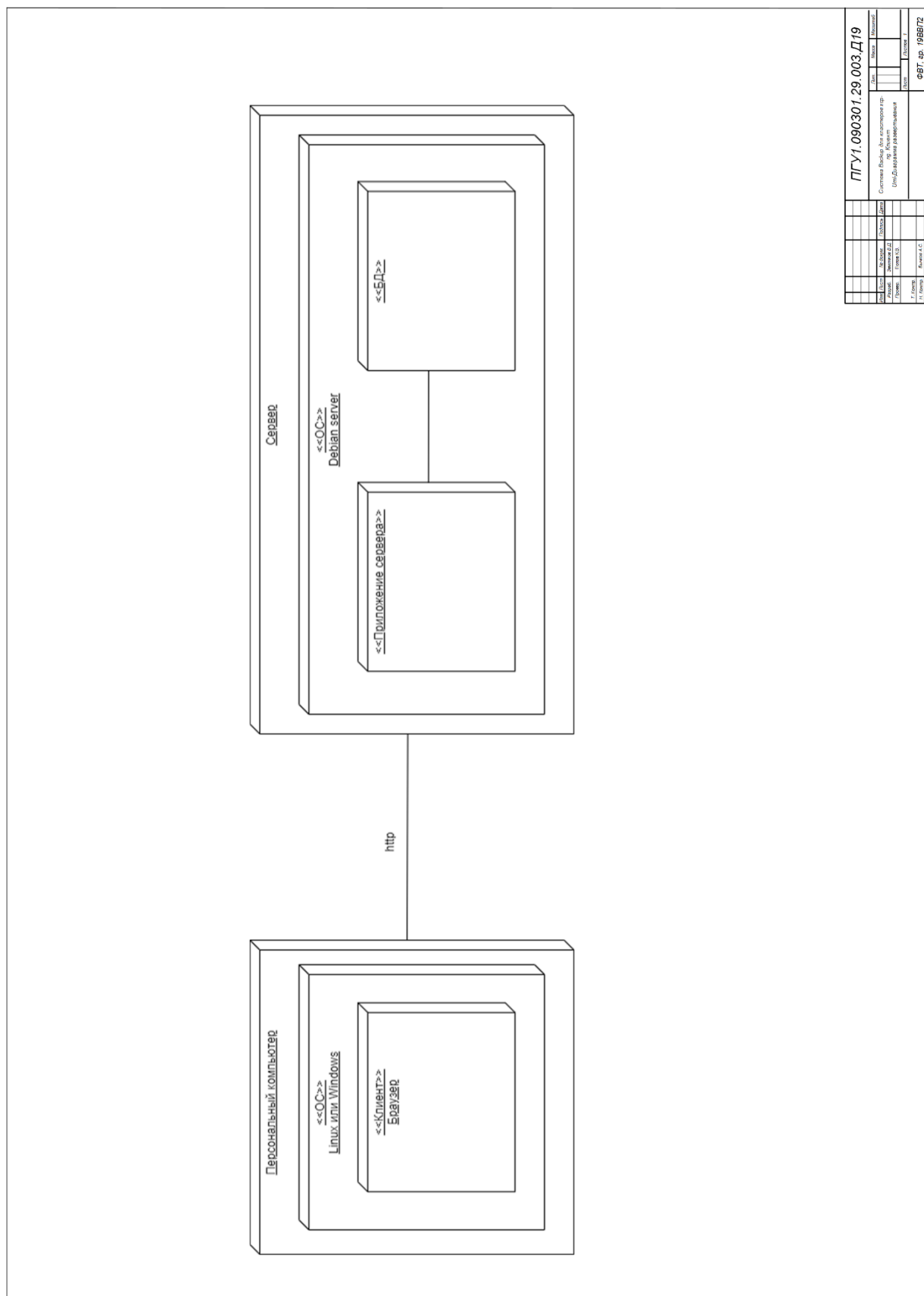
UML-Диаграммы

					ПГУ1.090301.29.001 ПЗ	Лист
						61
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А.1 – Диаграмма последовательностей



Приложение А.2 – Диаграмма развертывания



ПГУ1.090301.29.003.Д19									
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Лист	Лист	Лист	Лист	Лист	Лист	Лист	Лист	Лист	Лист
Лист	Лист	Лист	Лист	Лист	Лист	Лист	Лист	Лист	Лист
Лист	Лист	Лист	Лист	Лист	Лист	Лист	Лист	Лист	Лист
Лист	Лист	Лист	Лист	Лист	Лист	Лист	Лист	Лист	Лист
Лист	Лист	Лист	Лист	Лист	Лист	Лист	Лист	Лист	Лист
Лист	Лист	Лист	Лист	Лист	Лист	Лист	Лист	Лист	Лист
Лист	Лист	Лист	Лист	Лист	Лист	Лист	Лист	Лист	Лист
Лист	Лист	Лист	Лист	Лист	Лист	Лист	Лист	Лист	Лист

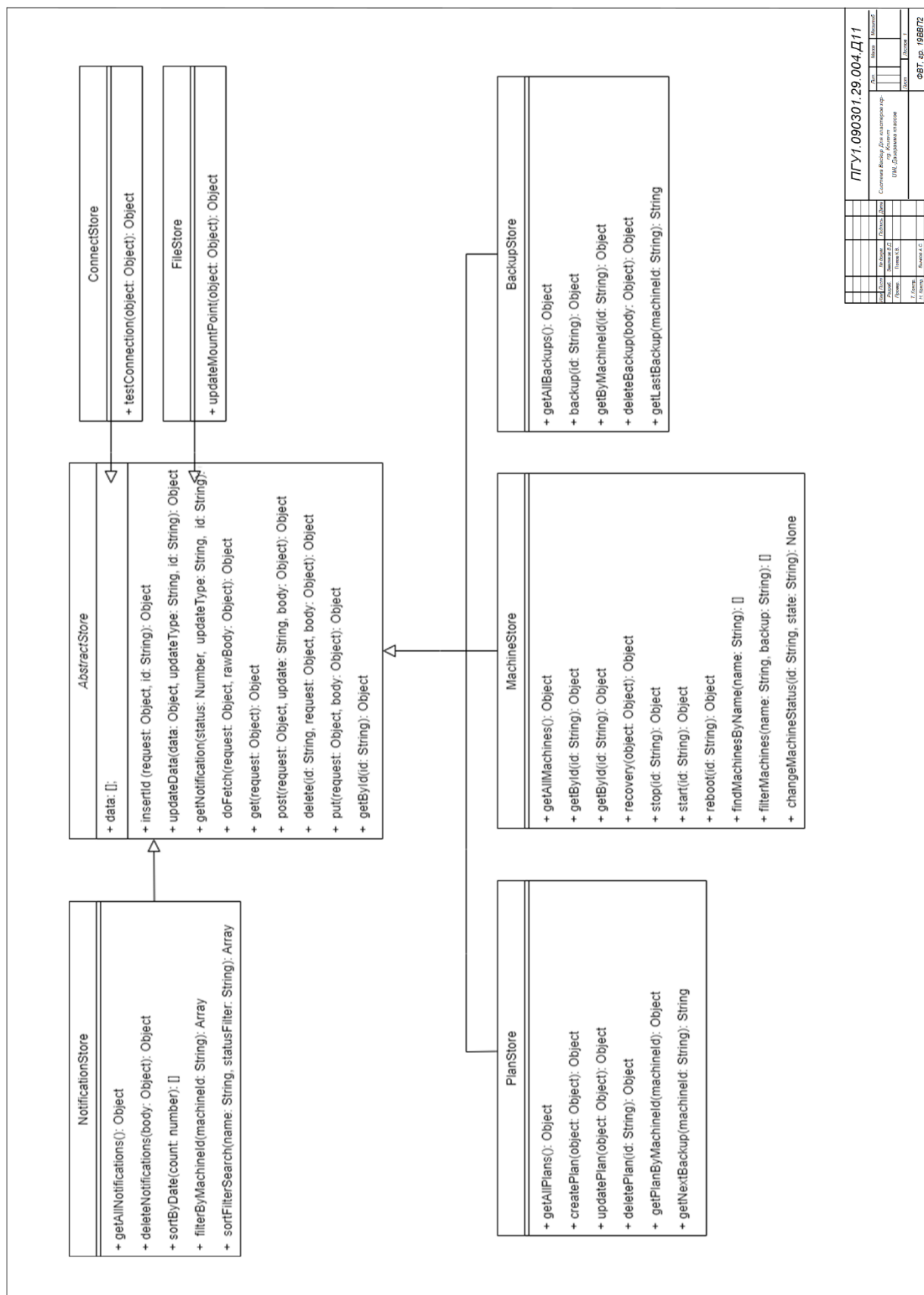
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.29.001 ПЗ

Лист

63

Приложение А.3 – Диаграмма классов



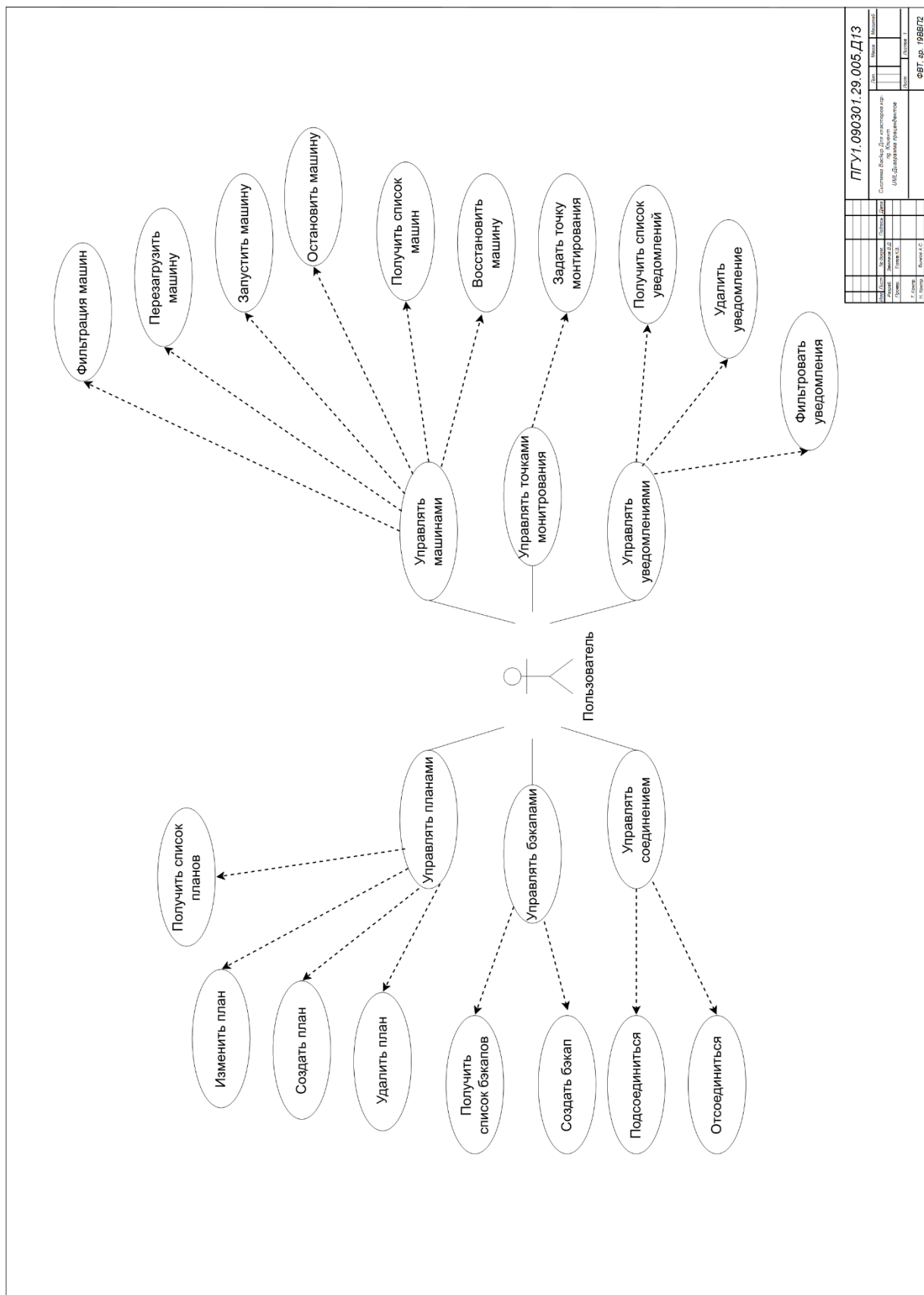
ПГУ1.090301.29.004.Д11									
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.29.001 ПЗ

Лист
64

Приложение А.4 – Диаграмма прецедентов

[illegible]

Приложение Б

(Обязательное)

Система *Васкир* для кластеров *хср-пг*. Клиент

Листинг программы

					ПГУ1.090301.29.001 ПЗ	Лист
						66
Изм.	Лист	№ докум.	Подпись	Дата		

Файл package.json

```
{
  "name": "backups",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "build": "webpack --mode production",
    "start": "webpack serve --mode development"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/Kerkey1/backupS.git"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/Kerkey1/backupS/issues"
  },
  "homepage": "https://github.com/Kerkey1/backupS#readme",
  "devDependencies": {
    "@babel/core": "^7.19.6",
    "@babel/preset-env": "^7.19.4",
    "@babel/preset-react": "^7.18.6",
    "babel-loader": "^9.0.1",
    "css-loader": "^6.7.1",
    "html-webpack-plugin": "^5.5.0",
    "react-router-dom": "^6.4.2",
    "style-loader": "^3.3.1",
    "url-loader": "^4.1.1",
    "webpack": "^5.74.0",
    "webpack-cli": "^4.10.0",
    "webpack-dev-server": "^4.11.1"
  },
  "dependencies": {
    "@ant-design/icons": "^4.7.0",
    "@babel/plugin-proposal-decorators": "^7.20.5",
    "antd": "^5.0.2",
    "dayjs": "^1.11.7",
    "lodash": "^4.17.21",
    "mobx": "^6.6.2",
    "mobx-react": "^7.5.3",
    "react": "^18.2.0",
    "react-device-detect": "^2.2.2",
    "react-dom": "^18.2.0",
    "react-router": "^6.4.2",
    "recharts": "^2.6.2"
  }
}
```

Файл webpack.config.js

```
const path = require("path");
const HtmlWebpackPlugin = require("html-webpack-plugin");

module.exports = {
  output: {
    path: path.join(__dirname, "/dist"),
    filename: "bundle.js",
  },
}
```

					ПГУ1.090301.29.001 ПЗ	Лист
						67
Изм.	Лист	№ докум.	Подпись	Дата		

```

        publicPath: '/',
      },
      devtool: 'source-map',
      plugins: [
        new HtmlWebpackPlugin({
          template: "index.html",
        }),
      ],
      devServer: {
        port: 3030,
        historyApiFallback: true,
      },
      module: {
        rules: [
          {
            test: /\.js$/,
            exclude: /node_modules/,
            use: {
              loader: "babel-loader",
            },
          },
          {
            test: /\.css$/,
            use: [
              {loader: "style-loader"},
              {loader: "css-loader"},
            ],
          },
          {
            test: /\.png|woff|woff2|eot|ttf|svg$/,
            loader: "url-loader",
            options: {limit: false},
          },
        ],
      },
    },
  ];

```

Файл src/App.js

```

import React, {useEffect} from "react";
import {observer} from "mobx-react";
import Router from "./routes";
import 'antd/dist/reset.css';
import './App.css';
import {useLocation} from "react-router";
import Sidebar from "./components/layout/Sidebar";
import MainContentWrapper from "./components/layout/MainContentWrapper";
import {getSession} from "./utils";
import machineStore from "./stores/MachineStore";
import planStore from "./stores/PlansStore";
import backupStore from "./stores/BackupStore";
import notificationStore from "./stores/NotificationStore";

const App = observer(() => {
  const location = useLocation();

  useEffect(() => {
    const session = getSession();
    if (session) {
      machineStore.getAllMachines()
      planStore.getAllPlans()
      backupStore.getAllBackups()
    }
  }, []);

```

					ПГУ1.090301.29.001 ПЗ	Лист
						68
Изм.	Лист	№ докум.	Подпись	Дата		

```

        notificationStore.getAllNotifications()
    }
}, [location?.pathname])

return <>
    <Sidebar/>
    <MainContentWrapper>
        <Router/>
    </MainContentWrapper>
</>

});
export default App

```

Файл src/index.js

```

import React from "react";
import {createRoot} from "react-dom/client";
import App from "../App";
import {BrowserRouter} from "react-router-dom";

const rootElement = document.getElementById('root');
const root = createRoot(rootElement);

root.render(
    <BrowserRouter>
        <App/>
    </BrowserRouter>
);

```

Файл src/routes.js

```

import React from "react";
import {Navigate, useRoutes} from "react-router";
import {PagesData} from "../constants/PagesContent";
import {sessionExist} from "../utils";
import LoginPage from "../pages/LoginPage";

const Router = () => {
    const importantRoutes = [
        {path: "/login", element: <LoginPage/>},
        {path: "/*", element: <>NotFound</>}
    ]

    const dynamicRoutes = PagesData.flatMap(pd =>
        pd.routes.map(r => {
            return {path: r.path, element: sessionExist() ? r.element :
<Navigate to='/login'/>}
        })
    )

    return useRoutes([...dynamicRoutes, ...importantRoutes])
};
export default Router

```

Файл src/utils.js

```

//SESSION
import {PartOfMonthEnum} from "../constants/enums/PartOfMonthEnum";
import {MonthsEnum} from "../constants/enums/MonthsEnum";

export const getSession = () =>

```

					ПГУ1.090301.29.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		69

```

JSON.parse(sessionStorage.getItem('currentSession'));
export const setSession = (session) =>
sessionStorage.setItem('currentSession', JSON.stringify(session));
export const sessionExist = () => !!getSession();

//translation

//TIMES
export const newEventDate = (currentDate, date, extraMonth) => new
Date(currentDate.getFullYear(), currentDate.getMonth() + extraMonth, date);
export const compoundYears = (year, currentYear) => year < currentYear ?
currentYear + 1 : currentYear - 1;
export const zeroBefore = (value) => value.toString().length === 1 ? "0" +
value : value

export const getArrayForPartOfMonthEnum = (constant) => {
  switch (constant) {
    case PartOfMonthEnum.FirstWeek:
      return [1, 2, 3, 4, 5, 6, 7]
    case PartOfMonthEnum.SecondWeek:
      return [8, 9, 10, 11, 12, 13, 14]
    case PartOfMonthEnum.ThirdWeek:
      return [15, 16, 17, 18, 19, 20, 21]
    case PartOfMonthEnum.ForthWeek:
      return [22, 23, 24, 25, 26, 27, 28]
  }
}

export const getPartOfMonth = (constant) => {
  switch (constant) {
    case PartOfMonthEnum.FirstWeek:
      return 1
    case PartOfMonthEnum.SecondWeek:
      return 2
    case PartOfMonthEnum.ThirdWeek:
      return 3
    case PartOfMonthEnum.ForthWeek:
      return 4
  }
}

export const nextEventDateSwitch = (currentDate, dayOfMonth, extraMonth = 0)
=> {
  switch (dayOfMonth) {
    case "1-7":
      return newEventDate(currentDate, 1, extraMonth);
    case "8-14":
      return newEventDate(currentDate, 8, extraMonth);
    case "15-21":
      return newEventDate(currentDate, 15, extraMonth);
    case "22-28":
      return newEventDate(currentDate, 22, extraMonth);
  }
}

export const fromBackupDate = (backupDate, type) => {
  const data = new Date(backupDate);
  const date = data.getDate();
  const day = data.getDay();
  const month = data.getMonth();
  const year = data.getFullYear();
  const hours = data.getHours();

```

					ПГУ1.090301.29.001 ПЗ	Лист
						70
Изм.	Лист	№ докум.	Подпись	Дата		

```

const minutes = data.getMinutes();

switch (type) {
  case "timelineTitle":
    return `${zeroBefore(date)} ${MonthsEnum.AlternateName[month]},
    ${zeroBefore(hours)}:${zeroBefore(minutes)}`
  case "recovery":
    return `${zeroBefore(date)}${zeroBefore(month +
1)}${zeroBefore(year)}-${zeroBefore(hours)}:${zeroBefore(minutes)}`
  default:
    return {date, day, month, year, hours, minutes}
}
}

```

Файл src/stores/AbstractStore.js

```

import {action, makeObservable, observable} from "mobx";
import {UpdateEnum} from "../constants/enums/UpdateStoreEnum";
import {getSession} from "../utils";
import {notifications} from "../components/common/notification";
import {StatusEnum} from "../constants/enums/StatusEnum";

const array = require("lodash/array");

class AbstractStore {
  @observable data = observable.array([]);

  constructor() {
    makeObservable(this);
  }

  @action insertId(request, id) {
    return {
      ...request,
      link: request.link + id
    }
  }

  @action updateData(data, updateType = UpdateEnum.Full, id) {
    switch (updateType) {
      case UpdateEnum.Full:
        this.data = data
        break;
      case UpdateEnum.Single:
        this.data.push(data)
        break;
      case UpdateEnum.Delete:
        this.data.splice(this.data.findIndex(v => v.id === id), 1)
        break;
      case UpdateEnum.Update:
        this.data.splice(this.data.findIndex(v => v.id === data.id),
1, data)
        break;
      case UpdateEnum.Response:
        return data
    }
  }

  @action getNotification(status, notificationsArray) {
    const notification = notificationsArray.find(n => n.status ===
status)
    if (notification)

```

					ПГУ1.090301.29.001 ПЗ	Лист
						71
Изм.	Лист	№ докум.	Подпись	Дата		

```

        notifications(notification.type, status,
notification.description)
    }

    @action
    async doFetch(request, rawBody) {
        const session = getSession();
        const body = !request.session ? JSON.stringify(rawBody) :
JSON.stringify({...rawBody, ...session})
        const response = await fetch(request.link, {
            method: request.method,
            headers: request.headers ?? {
                'Accept': 'application/json',
                'Content-Type': 'application/json'
            },
            body
        })

        if (!response.ok) response.clone().json().then(v =>
notifications(StatusEnum.Error, response.status, v ?? response.statusText))
        if (request?.notification && request?.notifications?.length > 0)
this.getNotification(response.status, request.notifications)
        else if (request?.notification) response.clone().json().then(res =>
notifications(StatusEnum.Success, response.status, res))

        return response
    }

    @action
    async get(request) {
        const response = await this.doFetch(request)
        return response.json().then(data => this.updateData(data))
    }

    @action
    async post(request, update = UpdateEnum.Full, body) {
        const response = await this.doFetch(request, body)
        return response.json().then(data => this.updateData(data, update))
    }

    @action
    async delete(id, request, body) {
        const response = await this.doFetch(request, body)
        return response.json().then(data => this.updateData(data,
UpdateEnum.Delete, id));
    }

    @action
    async put(request, body) {
        const response = await this.doFetch(request, body)
        return response.json().then(data => this.updateData(data,
UpdateEnum.Update));
    }

    getById(id) {
        return this.data.find(v => v.id === id)
    }
}

export default AbstractStore;

```

Файл src/stores/BackupStore.js

					ПГУ1.090301.29.001 ПЗ	Лист
						72
Изм.	Лист	№ докум.	Подпись	Дата		

```

import AbstractStore from "../AbstractStore";
import {action} from "mobx";
import {requests} from "../Links";
import {UpdateEnum} from "../../constants/enums/UpdateStoreEnum";
import {MonthsEnum} from "../../constants/enums/MonthsEnum";

class BackupStore extends AbstractStore {
  constructor() {
    super();
  }

  @action getAllBackups() {
    return this.post(requests.getAllBackups, UpdateEnum.Full)
  }

  @action backup(id) {
    return this.post(this.insertId(requests.soloBackup, id),
UpdateEnum.Response)
  }

  @action
  async deleteBackup(body) {
    return this.post(requests.deleteBackup, UpdateEnum.Response, body);
  }

  getByMachineId(id) {
    const filteredData = this.data.filter(v => v.id === id)
    return filteredData.length === 0 ? undefined : filteredData.sort((a,
b) => new Date(a.date) - new Date(b.date));
  }

  getLastBackup(machineId) {
    const filteredData = this.data.filter(item => item.id === machineId);
    if (filteredData.length === 0)
      return "Бэкапы отсутствуют"
    const sortedData = filteredData.sort((a, b) => new Date(b.date) - new
Date(a.date));
    const newestRecord = new Date(sortedData.shift().date);
    return newestRecord.getDate() + " " +
MonthsEnum.AlternateName[newestRecord.getMonth()]
  }
}

const backupStore = new BackupStore();
export default backupStore;

```

Файл src/stores/ConnectStore.js

```

import AbstractStore from "../AbstractStore";
import {action} from "mobx";
import {requests} from "../Links";
import {UpdateEnum} from "../../constants/enums/UpdateStoreEnum";

class ConnectStore extends AbstractStore {
  constructor() {
    super();
  }

  @action testConnection(object) {
    return connectStore.post(requests.testConnect, UpdateEnum.Response,
object)
  }
}

```

```

}

const connectStore = new ConnectStore();
export default connectStore;

```

Файл src/stores/FileStore.js

```

import AbstractStore from "../AbstractStore";
import {requests} from "../Links";
import {action} from "mobx";
import {UpdateEnum} from "../../constants/enums/UpdateStoreEnum";

class FileStore extends AbstractStore {
  constructor() {
    super();
  }

  @action updateMountPoint(object) {
    return this.post(requests.updateMountPoint, UpdateEnum.Response,
object)
  }
}

const fileStore = new FileStore();
export default fileStore

```

Файл src/stores/Links.js

```

import {StatusEnum} from "../../constants/enums/StatusEnum";

//const hostName = "http://172.16.70.244/api/"; //test
const hostName = "http://" + window.location.host + "/api/" //prod

export const requests = {
  // machines
  getAllMachines: {
    link: hostName + "machine",
    method: "POST",
    session: true,
    notification: false,
    headers: undefined,
  },
  machineRecovery: {
    link: hostName + "machine/recovery",
    method: "POST",
    session: true,
    notification: false,
    headers: undefined
  },
  startMachine: {
    link: hostName + "machine/start/",
    method: "POST",
    session: true,
    notification: true,
    headers: undefined,
    notifications: [
      {
        status: 200,
        description: "Машина успешно запущена",
        type: StatusEnum.Success
      }
    ]
  }
}

```

```

    },
    stopMachine: {
        link: hostName + "machine/stop/",
        method: "POST",
        session: true,
        notification: true,
        headers: undefined,
        notifications: [
            {
                status: 200,
                description: "Машина успешно перезагружена",
                type: StatusEnum.Success
            }
        ]
    },
    rebootMachine: {
        link: hostName + "machine/reboot/",
        method: "POST",
        session: true,
        notification: true,
        headers: undefined,
        notifications: [
            {
                status: 200,
                description: "Машина успешно перезагружена",
                type: StatusEnum.Success
            }
        ]
    },
    //plans
    getAllPlans: {link: hostName + "plan/all", method: "GET", session: false,
notification: false, headers: undefined},
    createPlan: {link: hostName + "plan", method: "POST", session: true,
notification: false, headers: undefined},
    deletePlan: {link: hostName + "plan/", method: "DELETE", session: true,
notification: false, headers: undefined},
    updatePlan: {link: hostName + "plan", method: "PUT", session: true,
notification: false, headers: undefined},

    //connect
    testConnect: {
        link: hostName + "connect",
        method: "POST",
        session: false,
        notification: true,
        headers: undefined,
        notifications: [
            {
                status: 200,
                description: "Подключение успешно выполнено",
                type: StatusEnum.Success
            }
        ]
    },

    //notification
    getAllNotifications: {
        link: hostName + "notification/all",
        method: "GET",
        session: false,
        notification: false,
        headers: undefined
    }

```

					ПГУ1.090301.29.001 ПЗ	Лист
						75
Изм.	Лист	№ докум.	Подпись	Дата		

```

    },
    createNotification: {
        link: hostName + "notification",
        method: "POST",
        session: false,
        notification: false,
        headers: undefined
    },
    deleteNotification: {
        link: hostName + "notification/",
        method: "DELETE",
        session: false,
        notification: true,
        headers: undefined,
        notifications: [
            {
                status: 200,
                description: "Уведомление успешно удалено",
                type: StatusEnum.Success
            }
        ]
    },
    updateNotification: {
        link: hostName + "notification",
        method: "PUT",
        session: false,
        notification: false,
        headers: undefined
    },
    //backups
    getAllBackups: {
        link: hostName + "backup/list",
        method: "POST",
        session: true,
        notification: false,
        headers: undefined
    },
    soloBackup: {
        link: hostName + "backup/",
        method: "POST",
        session: true,
        notification: true,
        headers: undefined,
        notifications: [
            {
                status: 200,
                description: "Создание резервной копии запущено",
                type: StatusEnum.Success
            }
        ]
    },
    deleteBackup: {
        link: hostName + "backup/delete",
        method: "POST",
        session: true,
        notification: true,
        headers: undefined,
        notifications: [
            {
                status: 200,
                description: "Бэкап успешно удален",
                type: StatusEnum.Success
            }
        ]
    }
}

```

```

    }
  ],
  //file
  updateMountPoint: {link: hostName + "file", method: "POST", session:
true, notification: false, headers: undefined}
}

```

Файл src/stores/MachineStore.js

```

import AbstractStore from "../AbstractStore";
import {action} from "mobx";
import {requests} from "../Links";
import planStore from "../PlansStore";
import {UpdateEnum} from "../../constants/enums/UpdateStoreEnum";

class MachineStore extends AbstractStore {
  constructor() {
    super();
  }

  @action getAllMachines() {
    return this.post(requests.getAllMachines, UpdateEnum.Full)
  }

  getById(id) {
    return this.data.find(v => v.uuid === id)
  }

  @action recovery(object) {
    return this.post(requests.machineRecovery, UpdateEnum.Response,
object)
  }

  @action stop(id) {
    return this.post(this.insertId(requests.stopMachine, id),
UpdateEnum.Response)
  }

  @action start(id) {
    return this.post(this.insertId(requests.startMachine, id),
UpdateEnum.Response)
  }

  @action reboot(id) {
    return this.post(this.insertId(requests.rebootMachine, id),
UpdateEnum.Response)
  }

  findMachinesByName(name) {
    return name === "" ? this.data : this.data.filter(machine =>
machine.name.toLowerCase().includes(name.toLowerCase()))
  }

  filterMachines(name, backup) {
    const data = this.findMachinesByName(name)
    return backup ? data.filter(machine =>
planStore.getPlanByMachineId(machine.uuid)) : data
  }

  @action changeMachineStatus(id, state) {

```

					ПГУ1.090301.29.001 ПЗ	Лист
						77
Изм.	Лист	№ докум.	Подпись	Дата		

```

        const index = this.data.findIndex(source => source.uuid === id);
        let temp = this.data
        temp[index].state = state;
        this.data = [...temp];
    }
}

const machineStore = new MachineStore();
export default machineStore;

```

Файл src/stores/NotificationStore.js

```

import AbstractStore from "../AbstractStore";
import {action} from "mobx";
import {requests} from "../Links";
import machineStore from "../MachineStore";
import {UpdateEnum} from "../../constants/enums/UpdateStoreEnum";

class NotificationStore extends AbstractStore {
    constructor() {
        super();
    }

    @action getAllNotifications() {
        return this.get(requests.getAllNotifications)
    }

    @action createNotification(body) {
        return this.post(requests.createNotification, body,
UpdateEnum.Single)
    }

    @action updateNotifications(body) {
        return this.put(requests.updateNotification, body)
    }

    @action deleteNotifications(id, body) {
        return this.delete(id, this.insertId(requests.deleteNotification,
id), body)
    }

    sortByDate(count) {
        const sortedData = this.data.slice().sort((a, b) => new Date(b.data)
- new Date(a.data))
        return !count ? sortedData : sortedData.slice(0, count)
    }

    filterByMachineId(machineId) {
        const sorted = this.sortByDate()
        return sorted?.length > 0 ? sorted.filter(s => s.machine ===
machineId) : []
    }

    sortFilterSearch(name, statusFilter) {
        let notifications = this.sortByDate()?.map(not => {
            return {
                ...not,
                machineName: machineStore.getById(not.machine)?.name
            }
        })

        notifications = name === "" ? notifications :

```

					ПГУ1.090301.29.001 ПЗ	Лист
						78
Изм.	Лист	№ докум.	Подпись	Дата		

```

notifications.filter(not => not?.machineName &&
not?.machineName?.toLowerCase().includes(name.toLowerCase()))
    notifications = !statusFilter ? notifications :
notifications.filter(not => not?.typeMessage === statusFilter)
    return notifications
}
}

const notificationStore = new NotificationStore();
export default notificationStore;

```

Файл src/stores/PlansStore.js

```

import AbstractStore from "../AbstractStore";
import {action} from "mobx";
import {requests} from "../Links";
import {UpdateEnum} from "../constants/enums/UpdateStoreEnum";
import {compoundYears, nextEventDateSwitch} from "../utils";
import {MonthsEnum} from "../constants/enums/MonthsEnum";

class PlanStore extends AbstractStore {
    constructor() {
        super();
    }

    @action getAllPlans(object) {
        return planStore.get(requests.getAllPlans, object)
    }

    @action createPlan(object) {
        return planStore.post(requests.createPlan, UpdateEnum.Single, object)
    }

    @action updatePlan(object) {
        return planStore.put(requests.updatePlan, object)
    }

    @action deletePlan(id) {
        return planStore.delete(id, this.insertId(requests.deletePlan, id))
    }

    getPlanByMachineId(machineId) {
        return this.data.find(plan => plan.machines === machineId)
    }

    getNextBackup(machineId) {
        const plan = this.getPlanByMachineId(machineId)
        if (!plan)
            return "План не задан"

        let nextEventDate;
        const currentDate = new Date();
        const currentYear = currentDate.getFullYear();
        const currentDayOfWeek = currentDate.getDay();

        if (plan.dayOfMonth !== undefined) {
            nextEventDate = nextEventDateSwitch(currentDate,
            plan.dayOfMonth);
            nextEventDate.setDate(nextEventDate.getDate() + (plan.dayOfWeek -
            nextEventDate.getDay() + 7) % 7);

            if (nextEventDate.getFullYear() !== currentYear)

```

					ПГУ1.090301.29.001 ПЗ	Лист
						79
Изм.	Лист	№ докум.	Подпись	Дата		

```

nextEventDate.setFullYear(compoundYears(nextEventDate.getFullYear(),
currentYear))

        if (currentDate.getTime() >= nextEventDate.getTime()) {
            nextEventDate = nextEventDateSwitch(currentDate,
plan.dayOfMonth, 1);
            nextEventDate.setDate(nextEventDate.getDate() +
(plan.dayOfWeek - nextEventDate.getDay() + 7) % 7);

            if (nextEventDate.getFullYear() !== currentYear)

nextEventDate.setFullYear(compoundYears(nextEventDate.getFullYear(),
currentYear))
        }
        } else if (plan.dayOfWeek !== undefined) {
            let daysUntilNextEvent = currentDayOfWeek >= plan.dayOfWeek ? (7
- currentDayOfWeek + plan.dayOfWeek) : (plan.dayOfWeek - currentDayOfWeek);
            nextEventDate = new Date(currentDate.getTime() +
daysUntilNextEvent * 24 * 60 * 60 * 1000);

            if (nextEventDate.getMonth() !== currentDate.getMonth()) {
                nextEventDate.setDate(1);
                nextEventDate.setMonth(nextEventDate.getMonth() + 1);
            }
        } else {
            nextEventDate = new Date();
        }

        return nextEventDate.getDate() + " " +
MonthsEnum.AlternateName[nextEventDate.getMonth()]
    }
}

const planStore = new PlanStore();
export default planStore;

```

Файл src/pages/DashboardPage.js

```

import React, {useRef, useState} from "react";
import useWidgets from "../hooks/useWidgets";
import WidgetWrapper from "../components/widgets/WidgetWrapper";

const DashboardPage = () => {
    const [widgets, setWidgets, pinned, getWidgetContent, handlePin,
handleUnpin, setWidgetsStore] = useWidgets();

    const dragzoneRef = useRef(null);
    const blocksRef = useRef([]);

    const [draggedBlock, setDraggedBlock] = useState(null);
    const [dragOffsetX, setDragOffsetX] = useState(0);
    const [dragOffsetY, setDragOffsetY] = useState(0);

    function handleDragStart(event) {
        const {target, clientX, clientY} = event;
        setDraggedBlock(target);
        setDragOffsetX(clientX - target.offsetLeft);
        setDragOffsetY(clientY - target.offsetTop);
    }
}

```

					ПГУ1.090301.29.001 ПЗ	Лист
						80
Изм.	Лист	№ докум.	Подпись	Дата		

```

function handleDragEnd() {
  setDraggedBlock(null);
  setWidgetsStore(widgets);
  setDragOffsetX(0);
  setDragOffsetY(0);
}

function handleDragOver(event, wrapperIndex) {
  event.preventDefault();
  const dragIndex = blocksRef.current.indexOf(draggedBlock);
  const tmpBlocks = widgets.slice();
  const block = tmpBlocks.splice(dragIndex, 1)[0];
  if (block) {
    tmpBlocks.splice(wrapperIndex, 0, block);
    setWidgets(tmpBlocks);
  }
}

return <div className={`widgets-page ${draggedBlock ? "drag-zone" : ""}`}
ref={dragzoneRef}>
  {widgets.map((widget, index) => {
    const content = getWidgetContent(widget);
    const wrapperRef = element => (blocksRef.current[index] =
element);
    return <WidgetWrapper
      widget={content}
      key={widget}
      refLink={wrapperRef}
      draggable={true}
      handlePin={handlePin}
      handleUnpin={handleUnpin}
      pinned={pined(widget)}
      onDragStart={handleDragStart}
      onDragEnd={handleDragEnd}
      index={index}
      active={draggedBlock === blocksRef.current[index] &&
"active"}
      onDragOver={(event) => handleDragOver(event, index)}

    />
  })}
</div>
}
export default DashboardPage;

```

Файл src/pages/DevicesPage.js

```

import React, {useState} from "react";
import {observer} from "mobx-react";
import MachinesTable from "../components/tables/MachinesTable";
import machineStore from "../stores/MachineStore";
import TopPageSearch from "../components/common/TopPageSearch";

const DevicesPage = observer(() => {
  const [machineName, setMachineName] = useState('')

  return <>
    <TopPageSearch value={machineName} onChange={setMachineName}/>
    <div className="divider-block"/>
    <MachinesTable
      dataSource={machineStore.findMachinesByName(machineName)}/>
  </>
});

```

					ПГУ1.090301.29.001 ПЗ	Лист
						81
Изм.	Лист	№ докум.	Подпись	Дата		

```

    </>
  });
  export default DevicesPage;

```

Файл src/pages/LoginPage.js

```

import React from "react";
import {observer} from "mobx-react";
import {Button, Form, Input} from "antd";
import connectStore from "../stores/ConnectStore";
import {useNavigate} from "react-router";
import {setSession} from "../utils";

const formItemLayout = {
  labelCol: {
    span: 4,
  },
  wrapperCol: {
    span: 18,
  },
};

const formTailLayout = {
  labelCol: {
    span: 14,
  },
  wrapperCol: {
    span: 18,
    offset: 4,
  },
};

const LoginPage = observer(({}) => {
  const [form] = Form.useForm();
  const navigate = useNavigate();

  const rules = [
    {
      required: true,
      message: 'Пожалуйста, заполните поле',
    }
  ]

  const sleep = (ms) => {
    return new Promise(resolve => setTimeout(resolve, ms));
  }

  const logIn = () => {
    let data = form.getFieldsValue();
    connectStore.testConnection(data).then(res => {
      if (res === "Connect success") {
        setSession(data)
        sleep(1500).then(() => navigate('/'))
      }
    })
  }

  return <div className="login-page-wrapper">
    <div className="login-form">
      <div className="login-logo">
        BackupVT
      </div>

```

```

        <Form form={form} onFinish={logIn}>
          <Form.Item
            {...formItemLayout}
            name="host"
            label="Хост"
            rules={rules}
          >
            <Input/>
          </Form.Item>
          <Form.Item
            {...formItemLayout}
            name="login"
            label="Логин"
            rules={rules}
          >
            <Input/>
          </Form.Item>
          <Form.Item
            {...formItemLayout}
            name="password"
            label="Пароль"
            rules={rules}
          >
            <Input.Password/>
          </Form.Item>
          <Form.Item {...formTailLayout}>
            <Button type="primary" htmlType="submit" style={{width:
"100%"}}>
              Подключиться
            </Button>
          </Form.Item>
        </Form>
      </div>
    </div>
  );
  export default LoginPage;

```

Файл src/pages/NotificationsPage.js

```

import React, {useState} from "react";
import {observer} from "mobx-react";
import NotificationsTable from "../components/tables/NotificationsTable";
import notificationStore from "../stores/NotificationStore";
import TopPageSearch from "../components/common/TopPageSearch";
import StatusSelect from "../components/selects/StatusSelect";

const NotificationsPage = observer(({}) => {
  const [machineName, setMachineName] = useState('')
  const [notificationStatus, setNotificationStatus] = useState(undefined)

  const filters = [
    {
      key: "status",
      title: "Тип уведомления",
      content: <StatusSelect key="status" value={notificationStatus}
onChange={setNotificationStatus}/>
    }
  ]

  return <>
    <TopPageSearch value={machineName} onChange={setMachineName}
filters={filters}/>

```

					ПГУ1.090301.29.001 ПЗ	Лист
						83
Изм.	Лист	№ докум.	Подпись	Дата		

```

        <div className="divider-block"/>
        <NotificationsTable
        dataSource={notificationStore.sortFilterSearch(machineName,
        notificationStatus)}/>
    </>
  });
  export default NotificationsPage;

```

Файл src/pages/PlansPage.js

```

import React from "react";
import {observer} from "mobx-react";
import Calendar from "../components/plans/Calendar";
import planStore from "../stores/PlansStore";
import machineStore from "../stores/MachineStore";

const PlansPage = observer(() => {

  return <>
    <Calendar
      plansOnDays={planStore.data}
      machines={machineStore.data}/>
    </>

  </>
  export default PlansPage;

```

Файл src/pages/ProtectedDevicesPage.js

```

import React, {useEffect, useState} from "react";
import MachineSecurityTable from "../components/tables/MachineSecurityTable";
import {observer} from "mobx-react";
import machineStore from "../stores/MachineStore";
import TopPageSearch from "../components/common/TopPageSearch";
import DrawerTemplate from "../components/common/drawer/DrawerTemplate";
import {
  AlertOutlined,
  CalendarOutlined,
  EyeOutlined,
  InboxOutlined,
  PlusSquareOutlined,
  SettingOutlined
} from "@ant-design/icons";
import BackupsList from "../components/lists/BackupsList";
import PlanCard from "../components/cards/PlanCard";
import planStore from "../stores/PlansStore";
import DrawerSubHeader from "../components/common/drawer/DrawerSubHeader";
import backupStore from "../stores/BackupStore";
import NotificationCardList from "../components/lists/NotificationCardList";
import notificationStore from "../stores/NotificationStore";
import {useNavigate} from "react-router";
import {autorun} from "mobx";

const ProtectedDevicesPage = observer(() => {
  const navigate = useNavigate();
  const [machineId, setMachineId] = useState(undefined)

  const [machine, setMachine] = useState()
  const [plan, setPlan] = useState()
  const [backups, setBackups] = useState()
  const [notifications, setNotification] = useState()

```

```

const [machineName, setMachineName] = useState('')
const [disabled, setDisabled] = useState(false)
const [nextBackup, setNextBackup] = useState()

useEffect(() => {
  const disposer = autorun(() => {
    setMachine(machineStore.getById(machineId))
    setPlan(planStore.getPlanByMachineId(machineId))
    setBackups(backupStore.getByMachineId(machineId))
    setNotification(notificationStore.filterByMachineId(machineId))
  });
  return () => disposer()
}, [machineId])

useEffect(() => {
  if (plan)
    setNextBackup(planStore.getNextBackup(machineId))
  return () => setNextBackup(undefined)
}, [plan])

const onClose = () => setMachineId(undefined)

const makeCopy = () => {
  setDisabled(true)
  backupStore.backup(machineId).then(() =>
    setDisabled(false)
  );
}

const viewAll = () => navigate("/notifications")
const menu = [
  {
    key: "backupsList",
    title: "Резервные копии",
    icon: <InboxOutlined/>,
    content: <>
      <DrawerSubHeader
        text={backups?.length > 0 ? `Всего резервных копий:
${backups.length}` : `Резервных копий`}
        action={<><PlusSquareOutlined/> Создать резервную
копию</>}
        disabled={disabled}
        onActionClick={() => makeCopy()}
      />
      <BackupsList backups={backups} machineId={machineId}/>
    </>
  },
  {
    key: "notifications",
    title: "Оповещения",
    icon: <AlertOutlined/>,
    content: <>
      <DrawerSubHeader
        text={`Всего уведомлений: ${notifications?.length ?? 0}`}
        action={<><EyeOutlined/> Посмотреть для всех
устройств</>}
        onActionClick={viewAll}
      />
      <NotificationCardList notifications={notifications}/>
    </>
  },
  {
    key: "plan",

```

```

        title: "Планы",
        icon: <CalendarOutlined/>,
        content: <>
            <DrawerSubHeader
                text={nextBackup ? `Следующий бэкап: ${nextBackup}` :
"План не задан"}
                action={<><SettingOutlined/> Настроить план</>}
            />
            <PlanCard plan={plan}/>
        </>
    }
]

const title = (currentItem) => <>
    <span className="drawer-plans-title">{machine?.name}</span> -
{menu.find(m => m.key === currentItem).title}
</>

return <>
    <TopPageSearch value={machineName} onChange={setMachineName}/>
    <div className="divider-block"/>
    <MachineSecurityTable
        dataSource={machineStore.findMachinesByName(machineName)}
        onClick={setMachineId}/>
    <DrawerTemplate
        visible={machineId}
        title={(currentItem) => title(currentItem)}
        menu={menu} onClose={onClose}/>
</>

))
export default ProtectedDevicesPage;

```

Файл src/pages/SettingsPage.js

```

import React from "react";
import {Button, Form, Input} from "antd";
import fileStore from "../stores/FileStore";
import {observer} from "mobx-react";

const formItemLayout = {
    labelCol: {
        span: 4,
    },
    wrapperCol: {
        span: 18,
    },
};

const formTailLayout = {
    labelCol: {
        span: 14,
    },
    wrapperCol: {
        span: 18,
        offset: 4,
    },
};

const SettingsPage = observer(() => {
    const [form] = Form.useForm();
    const rules = [
        {

```

```

        required: true,
        message: 'Пожалуйста, заполните поле',
    }
]

const onFinish = () => {
    const data = form.getFieldsValue();
    fileStore.updateMountPoint(data)
}

return <div className="simple-wrapper">
    <div className="flex flex-column" style={{padding: "10px"}}>
        <h2>Настройка точки монтирования</h2>
        <Form form={form} name="dynamic_rule" onFinish={onFinish}>
            <Form.Item
                {...formItemLayout}
                name="mountIp"
                label="Маунт ip"
                rules={rules}
            >
                <Input/>
            </Form.Item>
            <Form.Item
                {...formItemLayout}
                name="mountLogin"
                label="Маунт логин"
                rules={rules}
            >
                <Input/>
            </Form.Item>
            <Form.Item
                {...formItemLayout}
                name="mountPassword"
                label="Маунт пароль"
                rules={rules}
            >
                <Input.Password/>
            </Form.Item>
            <Form.Item {...formTailLayout}>
                <Button type="primary" htmlType="submit" style={{width:
"100%}}>
                    Сохранить
                </Button>
            </Form.Item>
        </Form>
    </div>
</div>

));
export default SettingsPage;

```

Файл src/pages/WidgetsPage.js

```

import React from "react";
import {WidgetsData} from "../constants/WidgetsData";
import WidgetWrapper from "../components/widgets/WidgetWrapper";
import useWidgets from "../hooks/useWidgets";

const WidgetsPage = () => {
    const [widgets, setWidgets, pinned, getWidgetContent, handlePin,
    handleUnpin, setWidgetsStore] = useWidgets();

```

					ПГУ1.090301.29.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		87

```

    return <div className="widgets-page">
      {WidgetsData.map(widget => <WidgetWrapper
        widget={widget}
        key={widget.key}
        handlePin={handlePin}
        handleUnpin={handleUnpin}
        pinned={pined(widget.key)}
      />)}
    </div>
  }
  export default WidgetsPage;

```

Файл src/hooks/useColumns.js

```

import React, {useEffect, useState} from "react";

const useColumns = (columnsArray, stores) => {
  const [columns, setColumns] = useState([])
  useEffect(() => setColumns([...columnsArray]), [stores.map(s => s.data)])
  return columns
}
export default useColumns;

```

Файл src/hooks/useMenu.js

```

import {useEffect, useState} from "react";
import {PagesData} from "../constants/PagesContent";

const useMenu = () => {
  const [opened, setOpened] = useState([])

  useEffect(() => {
    PagesData.forEach(pd => {
      if (pd.routes.some(route => route.path === location.pathname))
        setOpened(prev => ([...prev, pd.key]))
    })
  }, []);

  const isOpened = (key) => opened.includes(key)
  const handleMenuClick = (key) =>
    opened.includes(key)?setOpened(opened.filter(sm => sm !==
    key)):setOpened([...opened, key]);

  return [isOpened, handleMenuClick]
};
export default useMenu;

```

Файл src/hooks/useWidgets.js

```

import React, {useEffect, useState} from "react";
import {WidgetsData} from "../constants/WidgetsData";

const useWidgets = () => {
  const [widgets, setWidgets] = useState([]);

  useEffect(() => {
    const w = getWidgetsStore()
    if (w && w?.length > 0) {
      const widgetsFromStore = Array.from(JSON.parse(w))
      const checkedWidgets = widgetsFromStore.filter(widget =>
        WidgetsData.some(v => v.key === widget.key))
      if (checkedWidgets.length !== widgetsFromStore.length)

```

					ПГУ1.090301.29.001 ПЗ	Лист
						88
Изм.	Лист	№ докум.	Подпись	Дата		

```

        setWidgetsStore (checkedWidgets)
        setWidgets (checkedWidgets)
    }
}, [])

const getWidgetsStore = () => localStorage.getItem("widgets")
const setWidgetsStore = (array) => localStorage.setItem("widgets",
JSON.stringify(array))
const getWidgetContent = (key) => WidgetsData.find(widget => widget.key
=== key)
const pinned = (key) => widgets.includes(key)

const updateWidgets = (array) => {
    setWidgets(array)
    setWidgetsStore(array)
}

const handlePin = (key) => {
    let w = [...widgets, key]
    updateWidgets(w)
}

const handleUnpin = (key) => {
    let w = widgets.filter(p => p !== key)
    updateWidgets(w)
}

return [widgets, setWidgets, pinned, getWidgetContent, handlePin,
handleUnpin, setWidgetsStore]
}
export default useWidgets;

```

Файл src/components/cards/NotificationCard.js

```

import React from "react";
import StatusIcon from "../../icons/StatusIcon";
import notificationStore from "../../stores/NotificationStore";
import {DeleteOutlined} from "@ant-design/icons";
import {Tooltip} from "antd";
import {fromBackupDate} from "../../utils";
import machineStore from "../../stores/MachineStore";
import {observer} from "mobx-react";

const NotificationCard = observer(({notification, small}) => {
    return <div className="notification-card">
        {!small ? <>
            <div className="notification-card-header">
                <span className="align-items-center">
                    <StatusIcon status={notification?.typeMessage}/>
                    <span className="notification-card-
date">{fromBackupDate(notification.data, "timelineTitle")}</span>
                </span>

                <Tooltip placement="left" title="Удалить уведомление">
                    <DeleteOutlined
                        style={{color: "red", marginRight: "15px"}}
                        onClick={() =>
notificationStore.deleteNotifications(notification.id)}
                    />
                </Tooltip>
            </div>
        </>
    </div>

```

					ПГУ1.090301.29.001 ПЗ	Лист
						89
Изм.	Лист	№ докум.	Подпись	Дата		

```

        </div>
        <div className="notification-card-content">
            <p>
                {notification.message}
            </p>
        </div>
    </> :
    <>
        <div className="notification-card-header">
            <StatusIcon status={notification?.typeMessage}/>
            <span className="notification-card-
date">{machineStore.getById(notification.machine)?.name}</span>
        </div>
        <div className="notification-card-content-small">
            <p>
                {notification.message}
            </p>
            <span>{fromBackupDate(notification.data,
"timelineTitle")}</span>
        </div>
    </>
</div>
});
export default NotificationCard

```

Файл src/components/cards/PlanCard.js

```

import React from "react";
import Description from "../../common/Description";
import machineStore from "../../stores/MachineStore";
import {DaysOfWeekEnum} from "../../constants/enums/DaysOfWeekEnum";
import {getPartOfMonth, zeroBefore} from "../../utils";
import {observer} from "mobx-react";

const PlanCard = observer(({plan, styles}) => {
    const generateTimes = (hours, minutes) => {
        if (hours.length === 1)
            return <Description label="Время"
description={` ${zeroBefore(hours[0])} : ${zeroBefore(minutes[0])} `}/>
        const results = []
        for (let i = 0; i < hours.length; i++) {
            results.push(
                <Description
                    key={plan.id + hours[i] + minutes[i]}
                    label={`Время ${i} `}
description={` ${zeroBefore(hours[i])} : ${zeroBefore(minutes[i])} `}/>
            )
        }
        return results
    }

    const generatePeriodicity = (dayOfWeek, dayOfMonth) => {
        if (dayOfMonth !== undefined) {
            return <
                <Description label="Периодичность" description="Ежемесячно"/>
                <Description label="Неделя месяца"
description={getPartOfMonth(dayOfMonth)}/>
                <Description label="День недели"
description={DaysOfWeekEnum.FullNames[dayOfWeek]}/>
            </>
        }
    }

```

```

    } else if (dayOfWeek !== undefined) {
      return <
        <Description label="Периодичность"
description="Еженедельно"/>
        <Description label="День недели"
description={DaysOfWeekEnum.FullNames[dayOfWeek]}/>
      </>
    } else {
      return <Description label="Периодичность"
description="Ежедневно"/>
    }
  }

  return plan && <div className={`plan-card ${styles}`}>
    <Description label="Устройство"
description={machineStore.getById(plan?.machines)?.name}/>
    <Description label="Максимальное количество копий для хранения"
description={plan?.copies}/>
    {generatePeriodicity(plan.dayOfWeek, plan.dayOfMonth)}
    {generateTimes(plan.hour, plan.minute)}
  </div>
  );
}
export default PlanCard

```

Файл src/components/common/drawer/DrawerContent.js

```
import React from "react";

const DrawerContent = ({content}) => {

  return <div className="drawer-content">
    {content}
  </div>
}

export default DrawerContent;
```

Файл src/components/common/drawer/DrawerMenu.js

```
import React from "react";
import {Tooltip} from "antd";

const DrawerMenu = ({menu, currentItem, onItemClick}) => {
  return <div className="drawer-menu-wrapper">
    {menu.map(i =>
      <Tooltip placement="left" key={i.key} title={i.title}>
        <div onClick={() => onItemClick(i.key)}
          className={currentItem === i.key ? "drawer-menu-item
active-drawer-item" : "drawer-menu-item"}>{i.icon}</div>
        </Tooltip>)}
    </div>
  };
export default DrawerMenu
```

Файл src/components/common/drawer/DrawerSubHeader.js

```
import React from "react";

const DrawerSubHeader = ({text, action, disabled, onActionClick}) => {
  const onClick = () => {
    if (onActionClick instanceof Function)
      onActionClick()
  }
}
```

					<div style="text-align: center;"> ПГУ1.090301.29.001 ПЗ </div>	Лист
Изм.	Лист	№ докум.	Подпись	Дата		91

```

    }
    return <div className="drawer-subtitle">
      <span className={"text"}>{text}</span>
      {action && <div onClick={onClick} className={!disabled ? "action" :
"action disabled"}>{action}</div>}
    </div>
  }
}
export default DrawerSubHeader

```

Файл src/components/common/drawer/DrawerTemplate.js

```

import React, {useState} from "react";
import {observer} from "mobx-react";
import {Drawer} from "antd";
import DrawerMenu from "../DrawerMenu";
import DrawerContent from "../DrawerContent";

const DrawerTemplate = observer(({visible, onClose, title, menu, content}) => {
  {
    const [currentItem, setCurrentItem] = useState(menu?.length > 0 ?
menu[0]?.key : undefined)
    const curContent = menu ? menu.find(m => m.key === currentItem)?.content
: content;
    const customTitle = () => title instanceof Function ? title(currentItem)
: title

    return <Drawer
      title={customTitle()}
      placement="right"
      closable={true}
      onClose={onClose}
      className="drawer-template"
      open={visible}
      width={700}
    >
      {menu && <DrawerMenu menu={menu} currentItem={currentItem}
onItemClick={setCurrentItem}/>}
      <DrawerContent content={curContent}/>
    </Drawer>
  );
});
export default DrawerTemplate

```

Файл src/components/common/Description.js

```

import React from "react";

const Description = ({label, description}) => {
  {
    return <span className="description-block">
      <span className="description-block-label">{label}</span>
      :
      <span className="description-block-description">{description}</span>
    </span>
  }
}
export default Description;

```

Файл src/components/common/FiltersBlock.js

```

import React from "react";

const FiltersBlock = ({filters}) => <div className={`dropdown-content`}>

```

					ПГУ1.090301.29.001 ПЗ	Лист
						92
Изм.	Лист	№ докум.	Подпись	Дата		

```

<h2>Фильтры</h2>
{filters.map(filter =>
  <span key={filter.key} className="filter-item">
    <span>`${filter.title}:`</span>
    <span className="ml-6">{filter.content}</span>
  </span>
)}
</div>

```

```
export default FiltersBlock;
```

Файл src/components/common/TopPageSearch.js

```

import React, {useState} from "react";
import Input from "antd/es/input/Input";
import {FilterOutlined, SearchOutlined} from "@ant-design/icons";
import FiltersBlock from "../FiltersBlock";

const TopPageSearch = ({value, onChange, filters}) => {
  const [isOpen, setIsOpen] = useState(false);
  const toggleDropdown = () => setIsOpen(!isOpen);

  return <>
    <div className="top-page-search">
      {filters && <FilterOutlined className="top-page-search-icon
cursor-pointer" onClick={toggleDropdown}/>}
      <Input
        bordered={false}
        placeholder="Поиск"
        prefix={<SearchOutlined className="top-page-search-icon"/>}
        value={value}
        onChange={e => onChange(e.target.value)} />
    </div>
    {isOpen && filters && <>
      <div className="divider-block"/>
      <FiltersBlock filters={filters}/></>
    }
  </>

};
export default TopPageSearch;

```

Файл src/components/common/notification.js

```

import {StatusEnum} from "../../constants/enums/StatusEnum";
import {notification} from "antd";

export const notifications = (status, message, description, placement =
"bottomRight") => {
  notification.config({
    placement,
    duration: 5,
    maxCount: 3,
  });

  switch (status) {
    case StatusEnum.Success:
      return notification.success({
        message: 'Успешно',
        description,
      });
    case StatusEnum.Warning:
      return notification.warning({

```

```

        message: 'Ошибка',
        description,
    });
    case StatusEnum.Error:
        return notification.error({
            message: `Ошибка - ${message}`,
            description,
        });
    }
}

```

Файл src/components/icons/StatusIcon.js

```

import React from "react";
import {CheckCircleOutlined, CloseCircleOutlined, WarningOutlined} from
"@ant-design/icons";
import {StatusEnum} from "../../constants/enums/StatusEnum";
import {Space} from "antd";

const StatusIcon = ({status, text}) => {
    switch (status) {
        case StatusEnum.Success:
            return <Space className="success-color status">
                <CheckCircleOutlined/>
                <span>{text ?? "Успешно"}</span>
            </Space>
        case StatusEnum.Warning:
            return <Space className="warning-color status">
                <WarningOutlined/>
                <span>{text ?? "Внимание"}</span>
            </Space>
        case StatusEnum.Error:
            return <Space className="error-color status">
                <CloseCircleOutlined/>
                <span>{text ?? "Ошибка"}</span>
            </Space>
    }
};
export default StatusIcon;

```

Файл src/components/icons/VirtualMachineIcon.js

```

import React from "react";

const VirtualMachineIcon = ({width = "50px", height = "50px"}) => <svg
    height={height}
    width={width}
    version="1.1"
    xmlns="http://www.w3.org/2000/svg"
    viewBox="0 0 34.418 34.418">
    <g>
        <path fill="#03295c" d="M32.335,11.606h-6.533v-1.009h6.533V11.606z
M32.335,12.264h-6.533v1.011h6.533V12.264z
M32.331,7.687h-6.523V9.91h6.523V7.687z M34.418,5.297v23.827c-
0.001,0.357-0.292,0.648-0.65,0.648h-9.397
c-0.359,0-0.649-0.291-0.649-0.649V5.297c0-0.358,0.29-0.65,0.649-
0.65h9.397C34.127,4.647,34.418,4.939,34.418,5.297z
M33.119,5.944h-8.1v22.53h8.1V5.944z M32.335,13.932h-
6.533v1.008h6.533V13.932z M31.272,21.45c0,1.162-0.939,2.099-2.099
c-1.16,0-2.099-0.938-2.099-2.099c0-1.158,0.94-2.098,2.099-
2.098C30.333,19.352,31.272,20.293,31.272,21.45z M30.515,21.45

```

```

c0-0.739-0.6-1.341-1.341-1.341c-0.742,0-1.342,0.601-
1.342,1.341s0.6,1.34,1.342,1.34C29.915,22.79,30.515,22.19,30.515,21.45z
M21.937,9.218v13.505c0,0.814-0.655,1.473-1.461,1.473H13.18c0,0-
0.414,2.948,2.212,2.948v1.475H13.18H8.758H6.546v-1.473
c2.529,0,2.212-2.948,2.212-
2.948H1.465C0.656,24.198,0,23.539,0,22.725V9.218c0-0.814,0.656-1.47,1.465-
1.47h19.01
C21.282,7.748,21.937,8.404,21.937,9.218z M12.332,22.394c0-0.698-
0.566-1.263-1.264-1.263c-0.699,0-1.266,0.565-1.266,1.263

s0.566,1.265,1.266,1.265C11.767,23.659,12.332,23.092,12.332,22.394z
M20.371,9.311H1.568v11.387H20.37h0.001
C20.371,20.698,20.371,9.311,20.371,9.311z M11.081,21.603c-0.434,0-
0.785,0.352-0.785,0.785s0.352,0.785,0.785,0.785
s0.785-0.352,0.785-0.785S11.515,21.603,11.081,21.603z"/>
</g>
</svg>
export default VirtualMachineIcon;

```

Файл src/components/lists/BackupsList.js

```

import React from "react";
import {observer} from "mobx-react";
import {useState} from "react";
import {Button, Result} from "antd";
import {fromBackupDate} from "../../utils";
import machineStore from "../../stores/MachineStore";
import backupStore from "../../stores/BackupStore";

const BackupsList = observer(({backups}) => {
  const [expanded, setExpanded] = useState(false)

  const handleExpand = (id) => {
    if (expanded === id) {
      setExpanded(null);
    } else {
      setExpanded(id);
    }
  };

  const string = (name, date, id) => `${name}-${fromBackupDate(date,
"recovery")}-${id}.xva`
  const handleRestore = (backup) => machineStore.recovery({
    id: backup.id,
    path: string(backup.name, backup.date, backup.id),
    name: backup.name
  })

  const handleDelete = (backup) => backupStore.deleteBackup({
    id: backup.id,
    path: string(backup.name, backup.date, backup.id),
    name: backup.name
  }).then(() => backupStore.data.splice(backupStore.data.indexOf(backup),
1))

  return backups?.length > 0 ?
    <div className="timeline">
      {backups?.map((item, index) => (
        <div key={index} className={`timeline-background ${expanded
=== index ? "active" : ""}`}>
          <div className={`timeline-item-wrapper`}>

```

					ПГУ1.090301.29.001 ПЗ	Лист
						95
Изм.	Лист	№ докум.	Подпись	Дата		

```

        <div
            className={`timeline-item ${expanded === index ?
"active" : ""}`}
            onClick={() => handleExpand(index)}>
            <span className="timeline-item-
title">{fromBackupDate(item.date, "timelineTitle")}</span>
            {expanded === index &&
            <>
                <br/>
                <br/>
                <p>Устройство: {item.name}</p>
                <Button type="primary" onClick={() =>
handleRestore(item)}>Восстановить</Button>
                <Button
                    style={{marginLeft: "6px"}}
                    onClick={() => handleDelete(item)}>
                    Удалить бэкап
                </Button>
            </>
            </div>
        </div>
    </div>))}
</div> :
<Result title="Бэкапы отсутствуют"/>

))
export default BackupsList;

```

Файл src/components/lists/MachineList.js

```

import React from "react";
import {observer} from "mobx-react";
import {Tooltip} from "antd";
import planStore from "../../stores/PlansStore";
import {DeleteOutlined} from "@ant-design/icons";
import machineStore from "../../stores/MachineStore";

const MachineList = observer(({curMachine, setMachine}) => {

    const deletePlan = (id) => {
        const session = JSON.parse(sessionStorage.getItem('currentSession'));
        if (session)
            planStore.deletePlan(id)
    }

    return <ul className="plan-machine-list">
        {machineStore.data.map(machine =>
            <li
                key={machine.uuid}
                onClick={() => {
                    setMachine(machine.uuid)
                }}
                className={machine?.uuid === curMachine ? "active" : ""}
            >
                {machine.name}
                {planStore.data.find(plan => plan.machines === machine.uuid)
                &&
                <Tooltip title={"Удалить план для " + machine.name}>
                    <DeleteOutlined
                        style={{color: "red", padding: "5px"}}
                        onClick={() => deletePlan(planStore.data.find(plan =>
plan.machines === machine.uuid).id)}

```

					ПГУ1.090301.29.001 ПЗ	Лист
						96
Изм.	Лист	№ докум.	Подпись	Дата		

```

        />
      </Tooltip>
    }
  </li>
)}
</ul>
});
export default MachineList

```

Файл src/components/lists/NotificationCardList.js

```

import React from "react";
import NotificationCard from "../../cards/NotificationCard";
import {observer} from "mobx-react";

const NotificationCardList = observer(({notifications, small}) =>
notifications.map(notification => <NotificationCard
  small={small}
  key={notification.id}
  notification={notification}/>))
export default NotificationCardList

```

Файл src/components/lists/PlanCardList.js

```

import React from "react";
import PlanCard from "../../cards/PlanCard";
import {observer} from "mobx-react";

const PlanCardList = observer(({plans, styles}) => {
  return <>
    {plans.map(plan => <PlanCard key={plan.id} styles={styles}
plan={plan}/>)}
  </>

});
export default PlanCardList;

```

файл src/components/plans/Calendar.js

```

import React, {useEffect, useState} from "react";
import {DaysOfWeekEnum} from "../../constants/enums/DaysOfWeekEnum";
import MachineSelect from "../../selects/MachineSelect";
import YearSelect from "../../selects/YearSelect";
import MonthSelect from "../../selects/MonthSelect";
import {getArrayForPartOfMonthEnum} from "../../utils";
import {observer} from "mobx-react";
import DrawerTemplate from "../../common/drawer/DrawerTemplate";
import PlanCardList from "../../lists/PlanCardList";
import {MonthsEnum} from "../../constants/enums/MonthsEnum";
import DrawerSubHeader from "../../common/drawer/DrawerSubHeader";
import PlansSettingsModal from "../../PlansSettingsModal";
import {PlusSquareOutlined, SettingOutlined} from "@ant-design/icons";

const Calendar = observer(({plansOnDays, machines}) => {
  const [currDate, setCurrDate] = useState({
    dayOfWeek: undefined,
    date: undefined,
    month: undefined,
    year: undefined,
  });

```

					ПГУ1.090301.29.001 ПЗ	Лист
						97
Изм.	Лист	№ докум.	Подпись	Дата		

```

});

const [date] = useState(new Date());
const [listOfYears, setListOfYears] = useState([]);
const [lastActiveDay, setLastActiveDay] = useState();
const [visibleMachines, setVisibleMachines] = useState([]);
const [dayInfoDrawerVisible, setDayInfoDrawerVisible] = useState(false);
const [planModalOpen, setPlanModalOpen] = useState(false)

// инициализация календаря
useEffect(() => {
  const year = date.getFullYear();
  identDate()
  setListOfYears(numberRange(year - 2, year + 5))
}, [])
const identDate = () => {
  const year = date.getFullYear();
  const month = date.getMonth();
  const dayOfWeek = date.getDay();
  const day = date.getDate();
  setCurrDate(customObjectDateCreate(dayOfWeek, day, month, year))
}
// обновление календаря
useEffect(() => {
  if (currDate.month !== undefined && currDate.year !== undefined &&
machines && plansOnDays) {
    const node = document.getElementById('table-body');
    node.innerHTML = '';
    lastActiveDay ? renderCalendar(currDate.month, currDate.year,
JSON.parse(lastActiveDay?.getAttribute("data-value"))) :
renderCalendar(currDate.month, currDate.year)
  }
}, [currDate, plansOnDays, machines, visibleMachines])

// useEffect(() => {
//   if (passedMachine)
//     setVisibleMachines(passedMachine)
// }, [passedMachine])

const numberRange = (start, end) => new Array(end - start).fill().map((d,
i) => i + start);
const itIsCurDay = (year, month, dateI) => date.getFullYear() === year &&
date.getMonth() === month && date.getDate() === dateI;
const updateMonth = (month) => setCurrDate(prev => ({...prev, month:
month}))
const updateYear = (year) => setCurrDate(prev => ({...prev, year: year}))
// const updateDay = (day) => setCurrDate(prev => ({...prev, dayOfWeek:
day}))
// const updateDate = (date) => setCurrDate(prev => ({...prev, date:
date}))
const customObjectDateCreate = (day, date, month, year) => {
  return {
    dayOfWeek: day,
    date: date,
    month: month,
    year: year,
  }
}

// выборка планов на день
const planAtThisDay = (dayOfCalendar, plans, useFilter) =>
plans.filter(plan => machinesAtThisDay(dayOfCalendar, plan, useFilter))
const machinesAtThisDay = (calendarDay, plan, useFilter) => {

```

```

        if (plan && visibleMachines.includes(plan.machines) || plan &&
!useFilter)
            if (plan.dayOfMonth !== undefined) {
                const arrayOfDates =
getArrayForPartOfMonthEnum(plan.dayOfMonth);
                return arrayOfDates.includes(calendarDay.date) &&
plan.dayOfWeek === calendarDay.dayOfWeek;
            } else if (plan.dayOfWeek !== undefined) {
                return calendarDay.dayOfWeek === plan.dayOfWeek;
            } else {
                return true
            }
        else
            return false
    }

    // генерация календаря
    const renderCalendar = (month, year, lastActive) => {
        let array = collectDays(month, year);
        let tB = document.getElementById('table-body');
        let tr = document.createElement('tr');
        let i = 0;
        for (i; i < array.length; i++) {
            // новая строка
            if (array[i].dayOfWeek === 1)
                tr = document.createElement('tr');
            // добавление ячейки в строку
            tr.appendChild(createCell(array[i], lastActive))
            // конец строки
            if (array[i].dayOfWeek === 0)
                tB.appendChild(tr);
        }
    }

    const createCell = (item, lastActive) => {
        // создание ячейки
        let cell = document.createElement('td');
        cell.dataset.value = JSON.stringify(item);
        if (lastActive && lastActive.date === item.date && lastActive.month
=== item.month && lastActive.year === item.year && lastActive.dayOfWeek ===
item.dayOfWeek) {
            cell.classList.add("calendar-active-day");
            setLastActiveDay(cell)
        } else
            cell.className = currDate.month !== item.month ? "calendar-day
calendar-day-of-other-months" : "calendar-day";

        // заголовок ячейки
        let cellHeader = document.createElement('div');
        itIsCurDay(item.year, item.month, item.date) ? cellHeader.className =
"cur-cell-header" : cellHeader.className = "cell-header";
        let cellDate = document.createTextNode(item.date);
        cellHeader.appendChild(cellDate);

        // тело ячейки
        let cellBody = document.createElement('div')
        cellBody.className = "cell-body";
        const thisDayPlan = planAtThisDay(item, plansOnDays, true);
        if (thisDayPlan.length > 0)
            for (let plan of thisDayPlan) {
                let tempText = document.createTextNode(machines.find(machine
=> machine.uuid === plan.machines)?.name);
                let tempP = document.createElement('p');

```

```

        tempP.appendChild(tempText);
        cellBody.appendChild(tempP);
    }

    // сборка ячейки
    cell.appendChild(cellHeader);
    cell.appendChild(cellBody);
    return cell
}

const collectDays = (month, year) => {
    const lastDateOfMonth = new Date(year, month + 1, 0).getDate(), //
    последнее число месяца
    lastDayOfMonth = new Date(year, month, lastDateOfMonth).getDay(),
    // последний день месяца
    firstDayOfMonth = new Date(year, month, 1).getDay(), // первый
    день месяца
    lastDateOfLastMonth = new Date(year, month, 0).getDate(); //
    последнее число предыдущего месяца

    const resMonth = []

    // прошлый месяц
    if (firstDayOfMonth !== 1) {
        let tempFirstDay = firstDayOfMonth - 1;
        if (firstDayOfMonth === 0)
            tempFirstDay = 6
        let i = tempFirstDay;
        let lastMonth = lastDateOfLastMonth;
        for (i; i > 0; i--) {
            resMonth.push(customObjectDateCreate(i === 7 ? 0 : i,
lastMonth, currDate.month === 0 ? 11 : currDate.month - 1, currDate.month ===
0 ? currDate.year - 1 : currDate.year))
            lastMonth--;
        }
        resMonth.reverse();
    }

    // текущий месяц
    let i = 1;
    let tempFirstDay = firstDayOfMonth;
    for (i; i <= lastDateOfMonth; i++) {
        resMonth.push(customObjectDateCreate(tempFirstDay, i,
currDate.month, currDate.year))
        tempFirstDay === 6 ? tempFirstDay = 0 : tempFirstDay++;
    }

    // следующий месяц
    if (lastDayOfMonth !== 0) {
        let i = lastDayOfMonth === 6 ? 0 : lastDayOfMonth + 1;
        let dateOfFutureMonth = 1;
        for (i; i <= 7; i++) {
            resMonth.push(customObjectDateCreate(i === 7 ? 0 : i,
dateOfFutureMonth, currDate.month === 11 ? 0 : currDate.month + 1,
currDate.month === 11 ? currDate.year + 1 : currDate.year))
            dateOfFutureMonth++;
        }
    }
    return resMonth
}

// переключение по нажатию на другой месяц в календаре
const setDateOnClick = (event) => {

```

```

const targetParent = event.target.parentNode;
if (targetParent.nodeName === 'TD') {
  let cellData = JSON.parse(targetParent.getAttribute("data-
value"))
  if (targetParent !== lastActiveDay) {
    targetParent.classList.add("calendar-active-day");
    targetParent.classList.remove("calendar-day");
    if (lastActiveDay) {
      lastActiveDay?.classList.remove("calendar-active-day");
      lastActiveDay?.classList.add("calendar-day");
    }
    setLastActiveDay(targetParent);
    setDayInfoDrawerVisible(true);
    setCurrDate(customObjectDateCreate(cellData.dayOfWeek,
cellData.date, cellData.month, cellData.year));
  }
}

const onCancel = () => {
  setDayInfoDrawerVisible(false)
  setLastActiveDay(undefined);
  identDate()
}

return <div>
  <div className="calendar-actions">
    <div>
      <MonthSelect value={currDate.month} onChange={updateMonth}/>
      <YearSelect value={currDate.year} onChange={updateYear}
listOfYears={listOfYears}/>
      <MachineSelect value={visibleMachines}
onChange={setVisibleMachines}/>
    </div>
    <span className="calendar-action-new" onClick={() =>
setPlanModalOpen(true)}>
      <SettingOutlined />
      <span className="calendar-action-new-text">Настроить
план</span>
    </span>
  </div>
  <div className="calendar-wrapper">
    <div className="divider-block-15"/>
    <table className="calendar-days-of-week-table">
      <thead>
        <tr className="th-days-of-week">
          {DaysOfWeekEnum.ShortNames.map(day => <th
key={day}>{day}</th>)}
        </tr>
      </thead>
      <tbody onClick={setDateOnClick} id="table-body"/>
    </table>
  </div>
  <DrawerTemplate
    title={currDate.date + " " +
MonthsEnum.AlternateName[currDate.month]?.toLowerCase() + " " + currDate.year
+ " года - " + DaysOfWeekEnum.FullNames[currDate.dayOfWeek]?.toLowerCase()}
    content={
      <>
        <DrawerSubHeader
          text={`Количество машин для которых запланировано
резервное копирование: ${planAtThisDay(currDate, plansOnDays,
false)?.length}`} />
        <PlanCardList plans={planAtThisDay(currDate, plansOnDays,

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.29.001 ПЗ

Лист

101

```

false)}>
        </>
    }
    visible={dayInfoDrawerVisible}
    onClose={onCancel}
  />
  <PlansSettingsModal open={planModalOpen} setOpen={setPlanModalOpen}
mode="">
  </div>
});
export default Calendar

```

					ПГУ1.090301.29.001 ПЗ	Лист
						102
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение В

(Обязательное)

Система *Васкир* для кластеров *хср-пг*. Клиент

Презентация

					ПГУ1.090301.29.001 ПЗ	Лист
						103
Изм.	Лист	№ докум.	Подпись	Дата		

Система Ваксир для кластеров хср-ng. Клиент

Выполнил

студент группы 198ВП2 Земляков В.Д.

Руководитель:

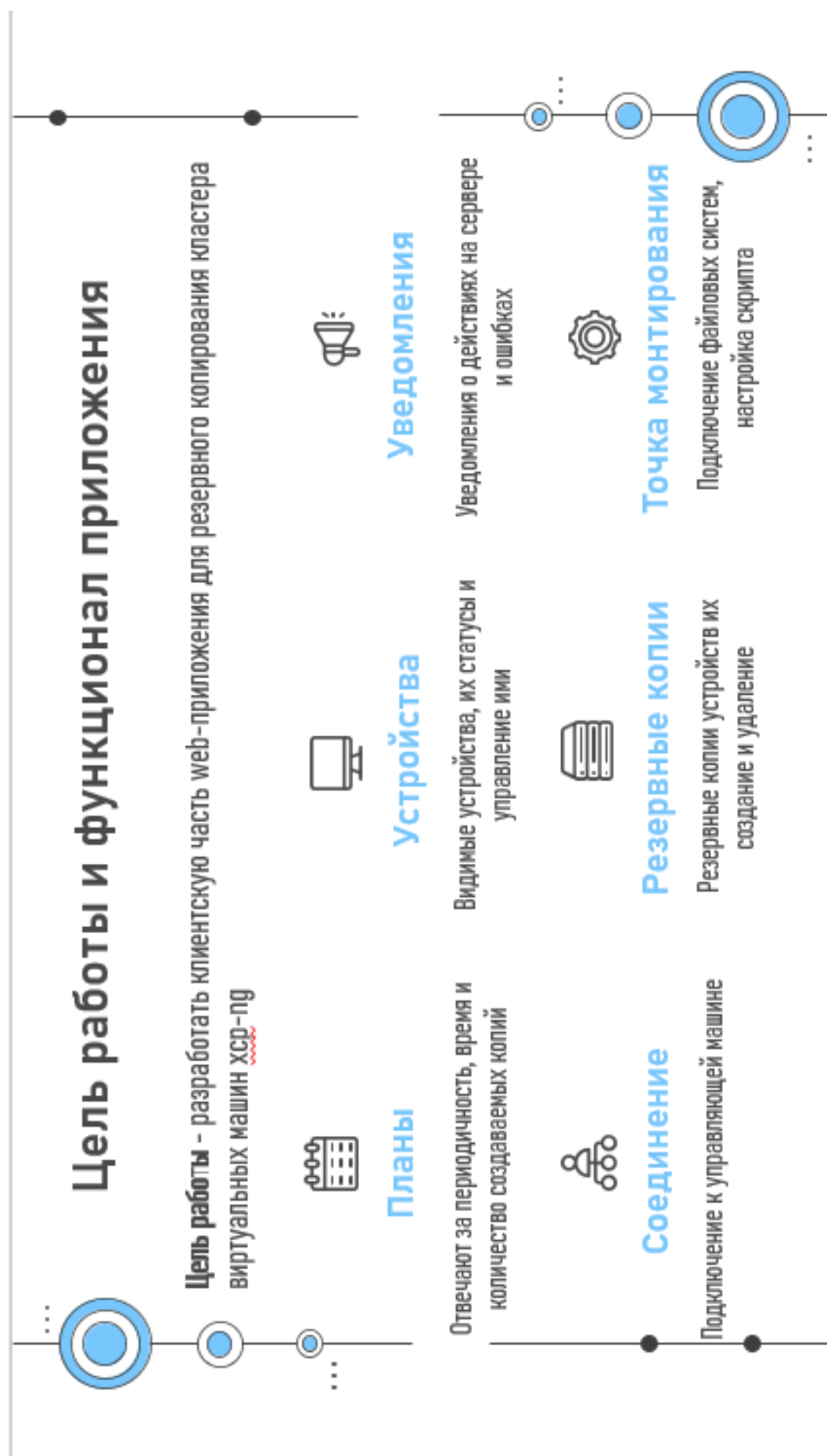
к.т.н., доцент кафедры ВТ ПГУ Попов К.В.

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.29.001 ПЗ

Лист

104



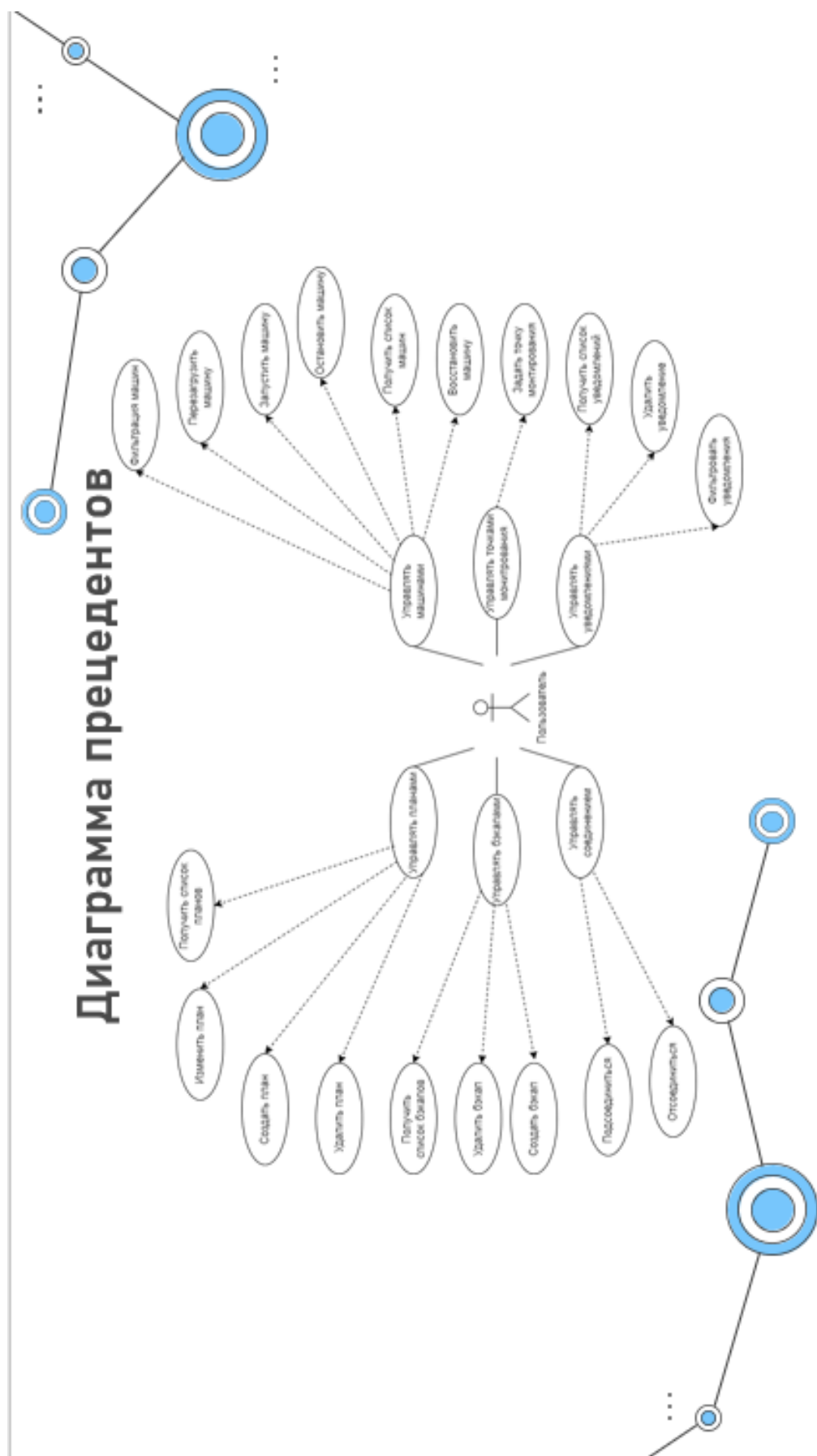
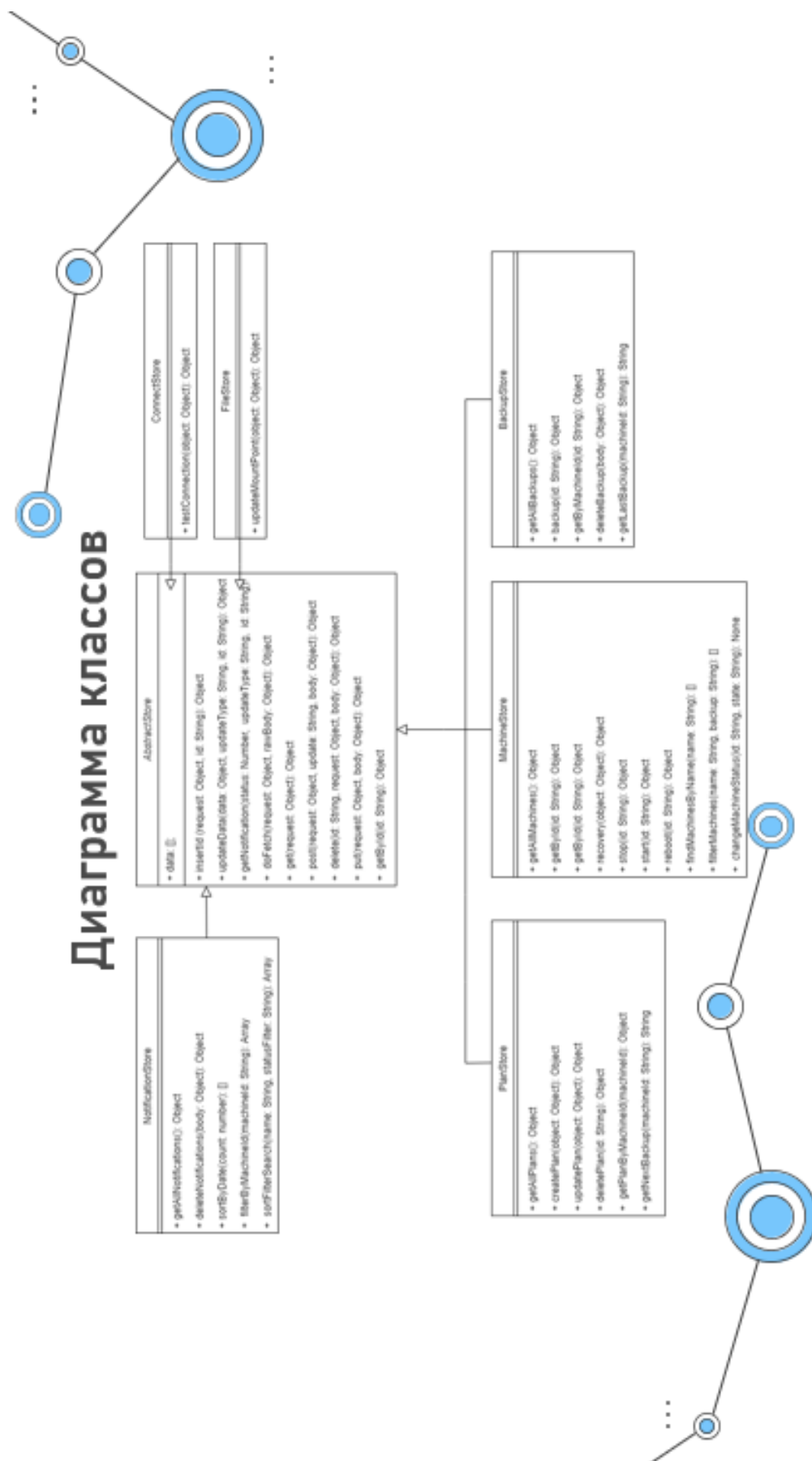
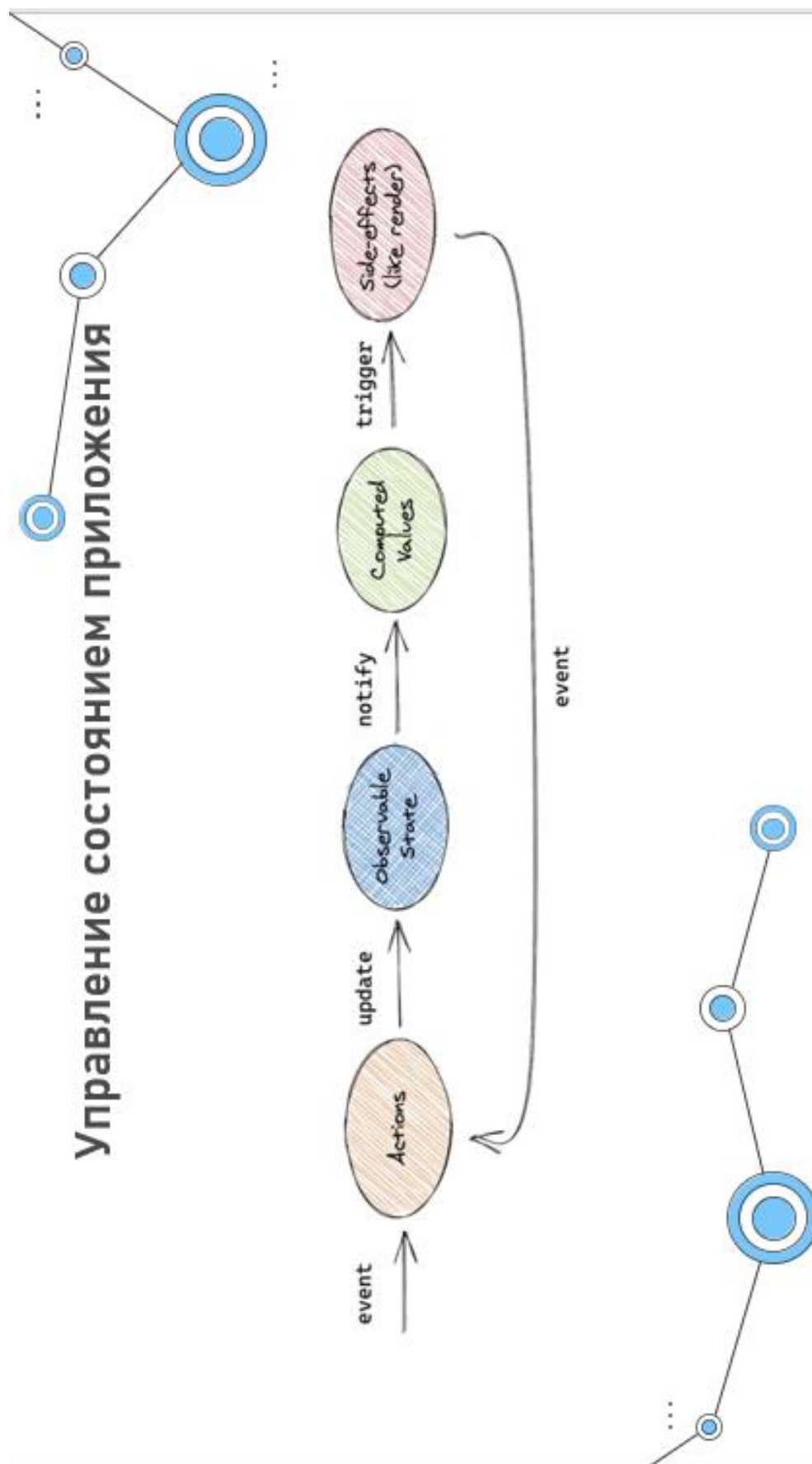




Диаграмма классов





Управление состоянием приложения

Авторизация

BackupVT

Имя:

Пароль:

Пользователь:

Порт:

☒ Подключиться

☐ Отменить

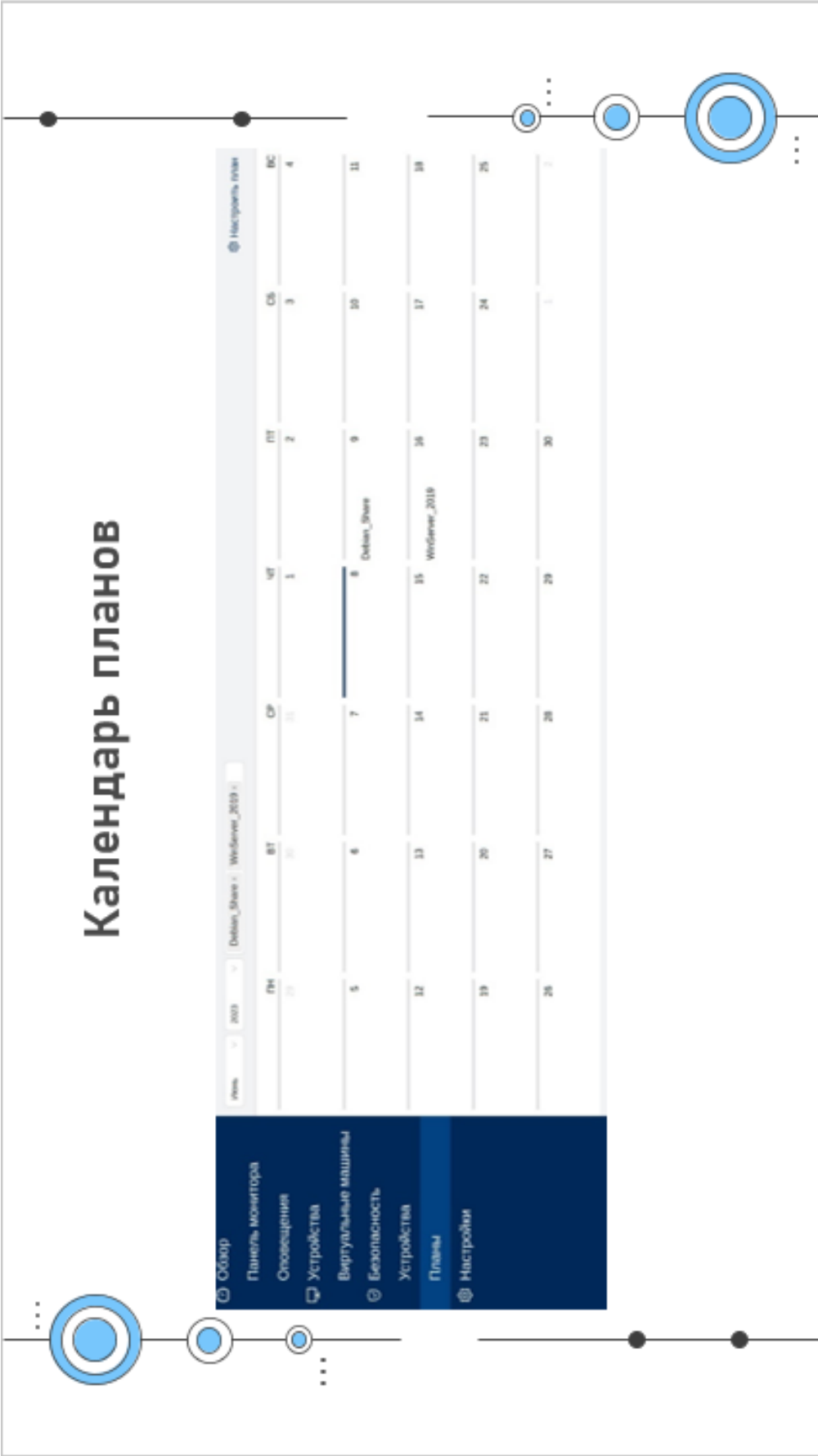
Пользователь:

Порт:



Список устройств

Виртуальные машины			
Имя	Статус	Описание	Действия
VM1	Запущена		Остановить
Online_VM1	Запущена		
VM100000_2019	Остановлена		
Сторг Виртуал	Запущена		
Виртуал	Остановлена		



Настройка планов

Настройка планов

☐ Ежедневно
 ☐ Еженедельно
 ☒ Ежемесячно

☐ Irbis
 ☐ Debian_Share
 ☐ WinServer_2019
 ☐ Copy of Backups
 ☐ Backups

Количество копий: 5

Время выполнения: 09.12

Добавить время

День недели: Пятница

Неделя месяца: Вторая

Отмена

OK

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

ПГУ1.090301.29.001 ПЗ

Лист
114



Информация о машине

Всего резервных копий: 1

07 Июня, 21:45

Устройство: Backup

Удалить Backup

Восстановить

Всего резервных копий: 1

07 Июня, 21:45

Устройство: Backup

Удалить Backup

Восстановить

Всего резервных копий: 1

07 Июня, 21:45

Устройство: Backup

Удалить Backup

Восстановить

Всего резервных копий: 1

07 Июня, 21:45

Устройство: Backup

Удалить Backup

Восстановить

Всего резервных копий: 1

07 Июня, 21:45

Устройство: Backup

Удалить Backup

Восстановить

Всего резервных копий: 1

07 Июня, 21:45

Устройство: Backup

Удалить Backup

Восстановить

Всего резервных копий: 1

07 Июня, 21:45

Устройство: Backup

Удалить Backup

Восстановить

Всего резервных копий: 1

07 Июня, 21:45

Устройство: Backup

Удалить Backup

Восстановить

Всего резервных копий: 1

07 Июня, 21:45

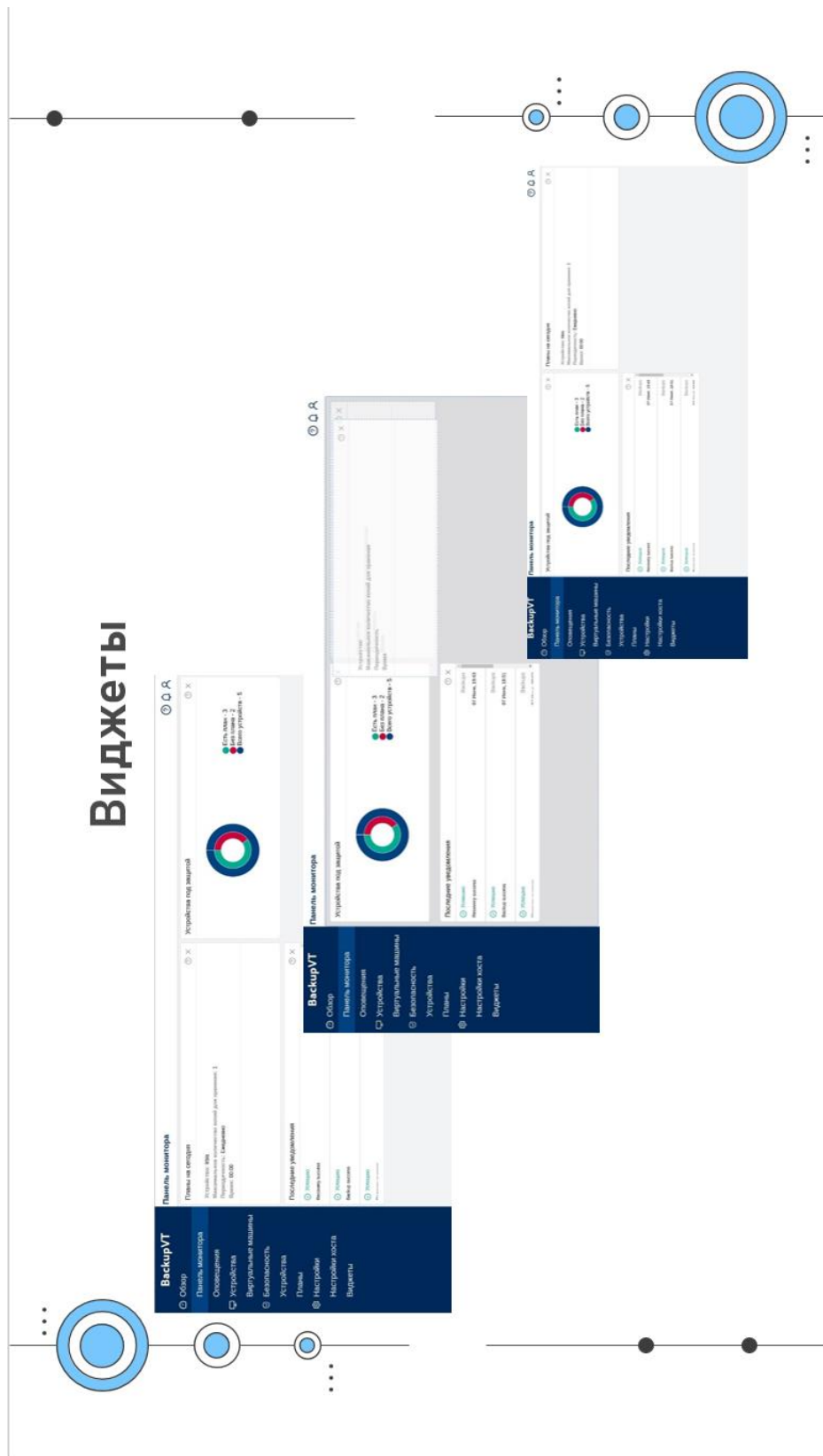
Устройство: Backup

Удалить Backup

Восстановить

[illegible]

Виджеты

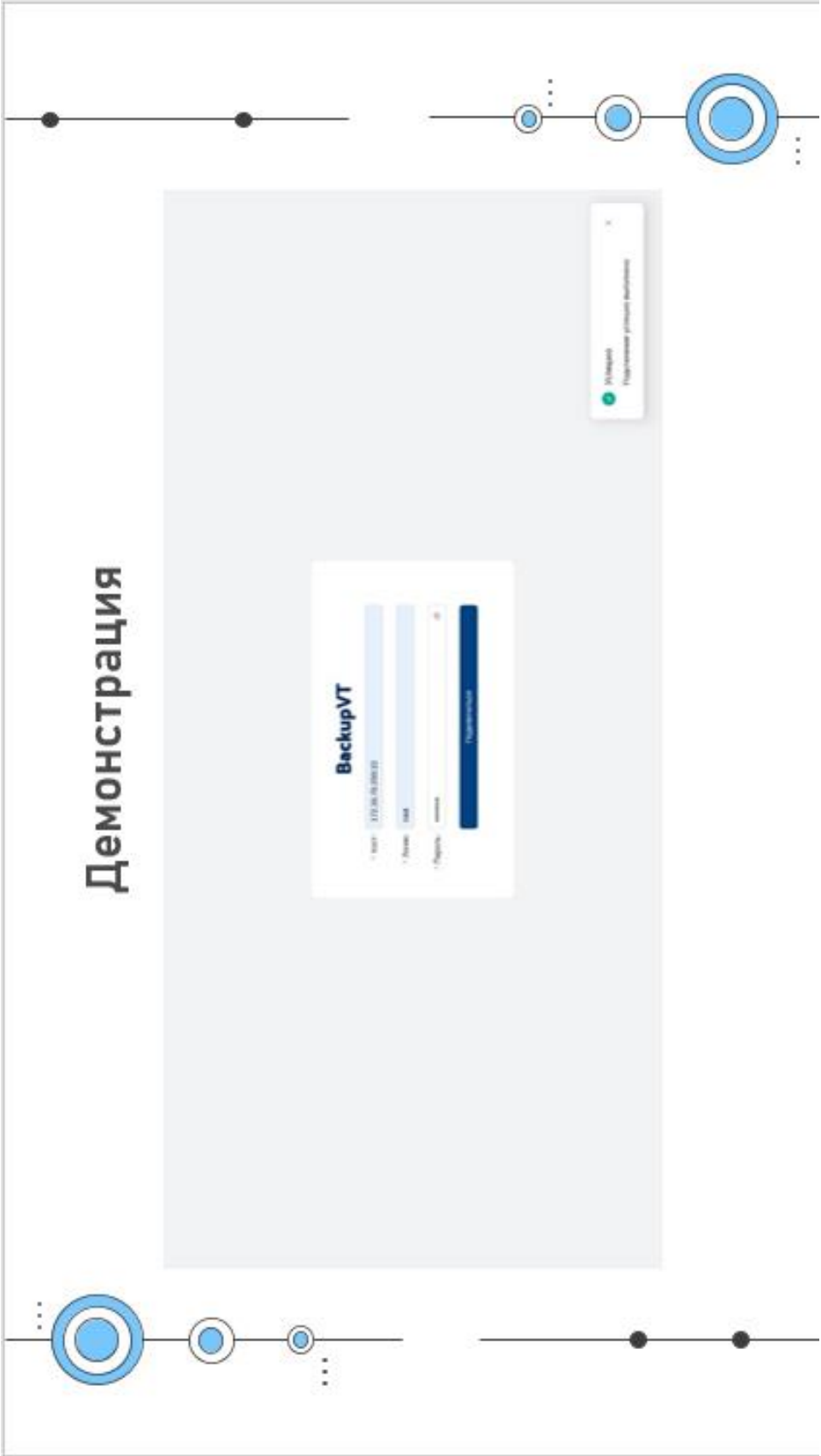


Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.29.001 ПЗ

Лист

118



Изм.	Лист	№ докум.	Подпись	Дата

Заключение

В результате выполнения данной выпускной квалификационной работы был создан удобный и понятный пользователю клиент web-приложения.

Приложение позволяет пользователю, создавать резервные копии виртуальных машин кластера xsr-ng и делать восстановление. Настраивать планы. Управлять питанием устройств. Отслеживать совершенные действия с помощью оповещений.