

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

**Система создания, хранения и развертывания
снапшотов веб-приложений**

Студент _____ Кеделидзе Георгий Гочаевич
(подпись, дата) (ФИО полностью)

Руководитель _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от 03.06.2026г. № 12)

Заведующий кафедрой _____ Козлов А.Ю.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК _____ Попова Н.А.
(подпись) (фамилия, инициалы)

Пенза, 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ А.Ю. Козлов
«20» декабря 2025 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Кеделидзе Георгий Гочаевич гр. 22ВП2 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Система создания, хранения и развёртывания снапшотов веб-приложений

Тема утверждена приказом ПГУ №1583/о от "30" декабря 2025 г.

3. Руководитель работы доцент кафедры «МОиПЭВМ», доцент, к.т.н., Такташкин Д.В.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

1) Требуется разработать систему создания, хранения и
развёртывания снапшотов веб-приложений.

2) Среда выполнения: операционная система Windows 11, Web-браузер

3) Языковые средства разработки: TypeScript, UML

4) Инструментальные средства: VS Code, ProjectLibre, Visual Paradigm
17.2, Draw.io, Vup

5) Основные функции:

а) создание снапшота;

б) разворачивание снапшота;

в) остановка снапшота;

г) удаление снапшота;

д) возвращение снапшота в
исходное состояние;

5. Перечень подлежащих разработке вопросов

1) Анализ предметной области и постановка задачи
2) Проектирование
3) Реализация
4) Контроль качества программного обеспечения
5) План разработки и оценка бюджета
6) Оформление пояснительной записки

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2026 – 14.04.2026	
2	Проектирование программных средств	20%	15.04.2026 – 25.04.2026	
3	Реализация программных средств	30%	26.04.2026 – 12.05.2026	
4	Тестирование программных средств	20%	13.05.2026 – 22.05.2026	
5	Оформление пояснительной записки	20%	23.05.2026 – 03.06.2026	

Дата выдачи задания «26» декабря 2025г.

Руководитель бакалаврской работы _____ 26.12.2025 г.
(подпись, дата)

Задание к исполнению принял студент _____ 26.12.2025 г.
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 74 листа, 24 рисунков, 11 таблиц, 21 использованных источников и 4 приложения.

СНАПШОТЫ, РАЗВЕРТЫВАНИЕ ПРИЛОЖЕНИЙ, POSTGRESQL, UML, TYPESCRIPT, VUE JS, BUN

Объектом исследования являются программные средства развертывания экземпляров приложений с копированием состояния базы данных.

Предметом исследования являются программные средства для создания, хранения и развертывания снапшотов веб приложений.

Разработка приложения проводилась на языке программирования TypeScript с использованием библиотеки tRPC и специализированной библиотеки Vue JS. В качестве среды разработки использовалась Visual Studio Code.

Среда исполнения – Windows 11, web-браузер.

Результаты работы – в ходе выполнения выпускной квалификационной работы были созданы клиентская и серверная части системы создания, хранения и развертывания веб-приложений.

					ПГУ 09.03.04-08BP222.14 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Система создания, хранения и развёртывания веб-приложений Пояснительная записка	Лит.	Лист	Листов
Разраб.		Кеделидзе Г. Г.						
Провер.		Такташкин Д.В.					4	74
Реценз.						гр. 22ВП2		
Н. Контр.		Такташкин Д.В.						
Утв.		Козлов А.Ю.						

Содержание

Введение.....	7
1 Постановка задачи и анализ требований программных средств для разворачивания веб-приложений	9
1.1 Анализ предметной области и постановка задачи.....	9
1.2 Анализ программного обеспечения аналогичного назначения	9
1.2.1 Платформа «Coolify»	9
1.2.2 Web-приложение «CapRover».....	10
1.2.3 Web-приложение «Easypanel»	11
1.2.4 Сравнение программных средств для автоматизации развертывания и управления контейнеризированными приложениями	12
1.3 Анализ функциональных требований.....	13
1.4 Анализ нефункциональных требований.....	16
2 Проектирование системы создания, хранения и развёртывания веб- приложений.....	18
2.1 Архитектура приложения	18
2.2 Проектирование структуры хранения	20
2.3.1 Концептуальная модель базы данных	20
2.3.2 Логическая модель базы данных	21
2.3 Проектирование серверной части	22
2.4 Проектирование пользовательского интерфейса	25
2.5 Требования к аппаратному обеспечению.....	25
2.6 Планирование разработки и оценка бюджета.....	27

3 Реализация программных средств системы создания, хранения и развёртывания снапшотов веб-приложений.....	30
3.1 Реализация архитектуры приложения	30
3.2 Реализация базы данных	31
3.3 Реализация серверной части приложения	32
3.4 Реализация интерфейса приложения	34
4 Контроль качества программного обеспечения.....	38
4.1 Обоснование выбора режима тестирования и метрик кода	38
4.2 Функциональное тестирование создания снапшота	38
4.3 Метрика Холстеда.....	42
Заключение.....	46
Список использованных источников	48
Приложение А (справочное) Глоссарий	51
Приложение Б (обязательное) Листинг серверной части приложения	53
Приложение В (обязательное) Листинг Sidecar-приложения	62
Приложение Г (обязательное) Листинг клиентского приложения	65

Введение

В современном мире веб-приложения стали одним из основных способов предоставления информационных сервисов пользователям. Развитие контейнеризации приложений значительно упростило выкатку приложений, что позволило значительно ускорить процесс доставки программного продукта конечному пользователю. Однако при активной разработке приложения возникает необходимость в сохранении различных состояний системы, включая версии исходного кода, конфигурации окружения и состояния базы данных. Развертывание нескольких копий одного и того же приложения, но с разным состоянием базы данных и кодовой базы, может помочь при аналитике последствий тех или иных изменений в работе приложения.

Существующие средства развертывания веб-приложений хорошо справляются с автоматизацией процесса публикации новых версий программного продукта, однако они не поддерживают возможность сохранения текущего состояния системы, для дальнейшего его развертывания.

Объект исследования – программные средства развертывания экземпляров приложений с копированием состояния базы данных

Предмет исследования – система создания, хранения и развертывания снапшотов веб приложений, с возможностью восстановления исходного состояния базы данных

Целью выпускной квалификационной работы является разработка системы создания, хранения и развертывания веб-приложений, обеспечивающей одновременную работу с разными версиями приложения, содержащих разное состояние базы данных.

Для достижения поставленной цели необходимо решить следующие задачи:

- проанализировать предметную область и выделить основные требования к разрабатываемой системе;
- проанализировать и сравнить продукты-аналоги;

- спроектировать программные средства и структуры данных;
- разработать систему, учитывая выделенные требования;
- осуществить контроль качества разработанной системы.

Система создания, хранения и развертывания снапшотов веб-приложений поможет облегчить анализ изменений состояния базы данных в процессе жизнедеятельности приложения и обеспечит удобную выкатку приложений.

1 Постановка задачи и анализ требований программных средств для разворачивания веб-приложений

1.1 Анализ предметной области и постановка задачи

Инструменты разворачивания представляют собой комплексный набор инструментов, технологий и передовых методов, которые в совокупности облегчают разворачивание программных приложений из среды разработки в целевую среду, например, в производственную, промежуточную или тестовую среду [1].

Современные веб-приложения чаще всего распространяются в виде Docker образов и конфигураций Docker Compose [2], это значительно упрощает процесс их разворачивания. Имея Docker образ приложения можно запустить его копию на любой машине, однако, если приложение использует СУБД для хранения своего состояния, информацию, хранимую в базе данных, нужно получать непосредственно из базы данных.

Разрабатываемая система должна подключаться к копируемому приложению посредством разворачивания вспомогательного приложения в его окружении. В свою очередь приложение, управляющее снапшотами, будет обращаться к вспомогательному сервису для получения всей необходимой информации для разворачивания копии программы.

Основная задача ВКР – разработка системы создания, хранения и разворачивания снапшотов веб-приложений, включающая в себя разработку сервиса, подключаемого к копируемому приложению, и сервису управления снапшотами с которым можно будет работать через клиентское приложение.

1.2 Анализ программного обеспечения аналогичного назначения

1.2.1 Платформа «Coolify»

Coolify представляет собой open-source платформу для публикации и разворачивания веб-приложений, позиционируемую как альтернатива таким облачным сервисам, как Heroku и Vercel. Coolify предоставляет возможность

использовать его как self-hosted приложение или же работать с ним в облаке [3].

Интерфейс приложения представлен на рисунке 1.

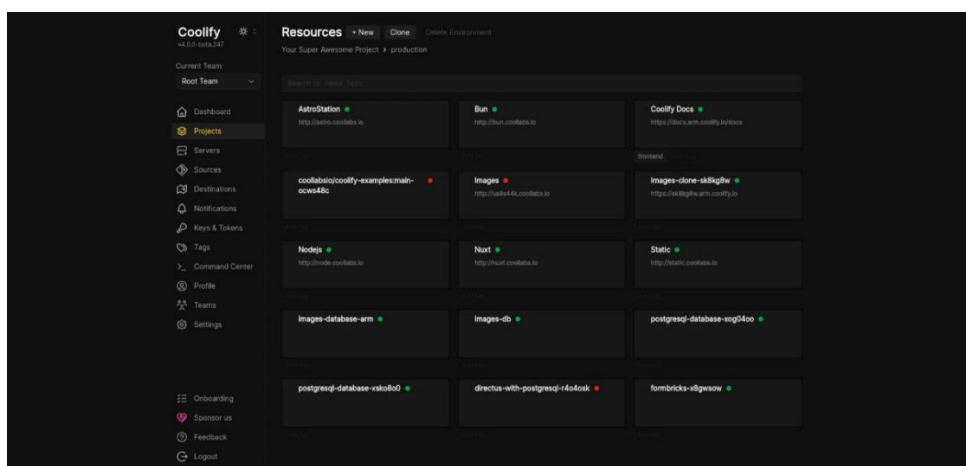


Рисунок 1 – Интерфейс приложения «Coolify»

К преимуществам платформы можно отнести:

- возможность выбора любого сервера для хостинга приложения;
- интеграция с git-репозиторием;
- автоматизация развертки приложения.

К недостаткам можно отнести сложность кастомизации конфигов окружения и отсутствия мониторинга приложений.

1.2.2 Web-приложение «CapRover»

CapRover является платформой автоматизированного развертывания веб-приложений, построенной на основе Docker Swarm. Платформа поддерживает развертывание приложений из Docker-образов и Git-репозиториев, а также предоставляет механизмы настройки SSL-сертификатов.

Интерфейс приложения представлен на рисунке 2.

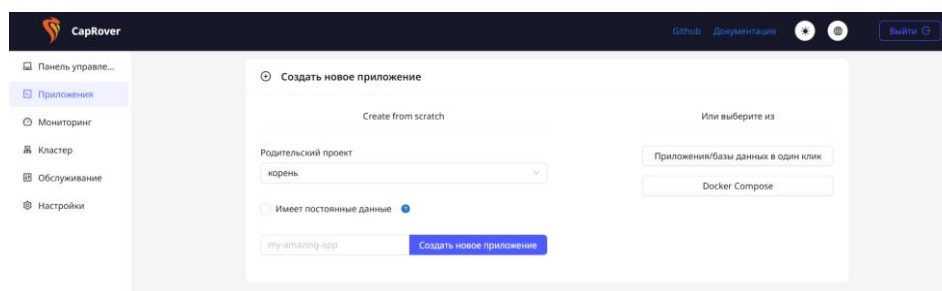


Рисунок 2 – Интерфейс приложения «CapRover»

К преимуществам платформы относятся:

- настройка SSL сертификатов;
- наличие one-click deploy.

К недостаткам системы можно отнести:

- ограниченные встроенные средства резервного копирования;
- отсутствие поддержки многих Docker Compose функций.

Таким образом, CapRover предоставляет удобные средства автоматизированного развертывания контейнеризированных приложений и хорошо подходит для небольших и средних проектов, однако часть возможностей требует дополнительной настройки.

1.2.3 Web-приложение «Easypanel»

Easypanel – это современная панель управления сервером, ориентированная на упрощение процесса развертывания и управления приложениями, базами данных и услугами [4]. Платформа поддерживает запуск Docker-контейнеров, управление базами данных, подключение пользовательских доменов и настройку SSL сертификатов.

Интерфейс приложения представлен на рисунке 3.

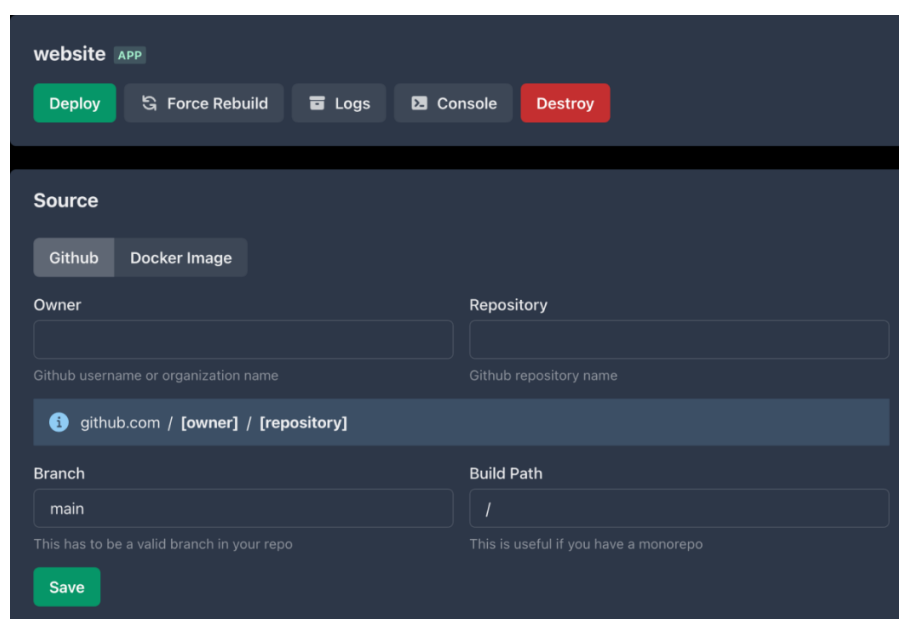


Рисунок 3 – Интерфейс приложения «Easypanel»

К преимуществам платформы относятся:

- поддержка Docker контейнеров;
- настройка SSL сертификатов;
- поддержка баз данных.

Среди недостатков можно выделить:

- ограниченные возможности резервного копирования;
- меньшую гибкость конфигурации по сравнению с более крупными платформами;
- ограниченное количество встроенных интеграций;
- зависимость части функционала от коммерческой версии продукта.

Таким образом, Easypanel является удобным решением для быстрого развертывания веб-приложений и управления контейнеризированной инфраструктурой, однако уступает более функциональным платформам по возможностям расширения и глубине настройки.

1.2.4 Сравнение программных средств для автоматизации развертывания и управления контейнеризированными приложениями

Проведем сравнительный анализ рассмотренных выше программных средств. Для каждого из критериев используется трехбалльная шкала, где: 0 баллов – «полностью не соответствует критерию», 1 балл – «частично соответствует критерию», 2 балла – «полностью соответствует критерию».

Результаты сравнительного анализа представлены в таблице 1.

Таблица 1 – Сравнительный анализ аналогичных программных средств

Критерии	Coolify	CapRover	Easypanel
Поддержка Docker-контейнеров	2	1	1
Интеграция с Git-репозиториями	2	1	2
Автоматизация развертывания	2	2	1

Продолжение таблицы 1

Критерии	Coolify	CapRover	Easypanel
Управление базами данных	1	1	1
Настройка SSL-сертификатов	1	2	1
Резервное копирование	1	0	1
Расширяемость и настройка конфигурации	2	2	1
ИТОГО	11	9	8

Лидером среди обозреваемых решений оказалась платформа Coolify. Тем не менее даже обладая таким преимуществом против других систем, оно не позволяет сохранять и развертывать текущие состояния приложений.

Разрабатываемая система должна учитывать сильные стороны рассмотренных аналогов, включая автоматизацию развертывания, управление доменами и восстановление состояния приложения. При этом система должна быть ориентирована на упрощенный сценарий клонирования и развёртывания приложения с сохранением пользовательских данных и минимальным количеством действий со стороны пользователя.

1.3 Анализ функциональных требований

После проведения исследования и анализа предметной области, поиска и детального рассмотрения существующих программных средств аналогичного назначения, а также их сравнения, были выявлены основные функциональные требования к разрабатываемой системе:

- создание снапшота;
- разворачивание снапшота;
- отключение снапшота;
- возвращение снапшота в исходное состояние;
- удаление снапшота.

В соответствии с выявленными функциональными требованиями была

разработана диаграмма вариантов использования UML. Разработанная диаграмма вариантов использования для системы создания, хранения и развертывания снимков веб приложений представлена на рисунке 4.

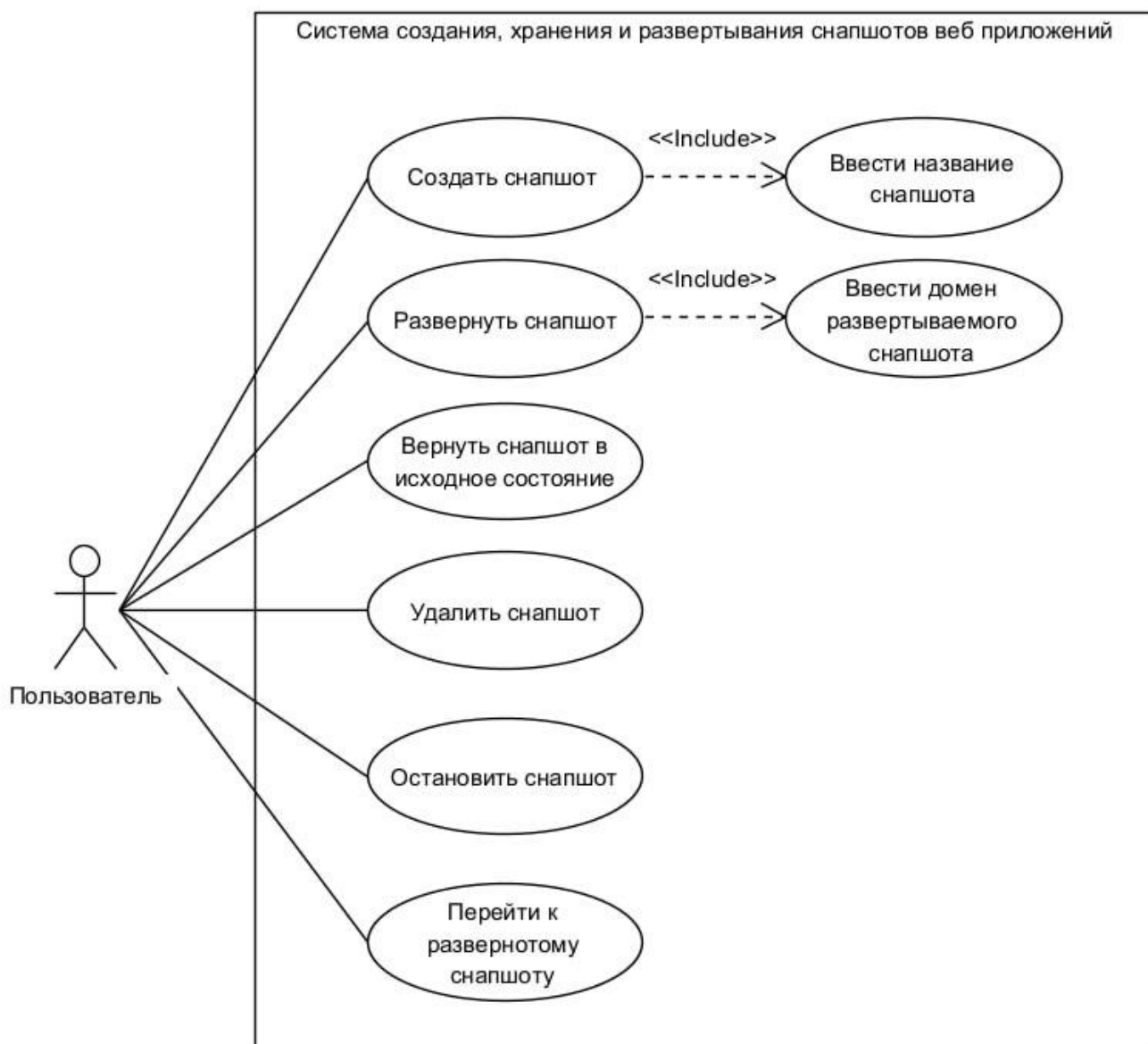


Рисунок 4 – Диаграмма вариантов использования для разрабатываемых программных средств

Диаграмма вариантов использования позволяет четко определить границы системы и понять потребности пользователей, фокусируясь на том, что система должна делать, а не на деталях ее внутренней реализации или архитектуры [5].

На диаграмме также используются несколько видов связей: отношение ассоциации, обозначающее связь между актером и отдельным вариантом

использования и отношение включения, позволяющее включить поведение одного прецедента в поток другого [6].

Спецификация прецедента «Создать снимок» представлена на рисунке 5.

Прецедент: Создать снимок
ID: 1
Краткое описание: Пользователь создает снимок веб-приложения
Главные актеры: Пользователь
Предусловие: Существует Docker образ веб-приложения
Основной поток: <ol style="list-style-type: none"> 1. Пользователь нажимает на кнопку «Создать снимок» 2. Пользователь вводит название снимка 3. Пользователь нажимает на кнопку «Создать»
Постусловие: Пользователь создал снимок

Рисунок 5 – Спецификация прецедента «Создать снимок»

Спецификация прецедента «Развернуть снимок» представлена на рисунке 6.

Прецедент: Развернуть снимок
ID: 2
Краткое описание: Пользователь разворачивает снимок веб-приложения на выбранном им домене
Главные актеры: Пользователь
Предусловие: Снимок создан
Основной поток: <ol style="list-style-type: none"> 1. Пользователь нажимает на кнопку «Развернуть» 2. Пользователь вводит домен, на котором будет развернут снимок 3. Пользователь нажимает на кнопку «Развернуть»
Постусловие: Снимок веб-приложения развернут

Рисунок 6 – Спецификация прецедента «Развернуть снимок»

Спецификация прецедента «Вернуть снимок в исходное состояние» представлена на рисунке 7.

Прецедент: Вернуть снимок в исходное состояние
ID: 3
Краткое описание: Пользователь возвращает снимок в исходное состояние
Главные актеры: Пользователь
Предусловие: Снимок создан
Основной поток: Пользователь нажимает на кнопку «Вернуть в исходное состояние»
Постусловие: Снимок возвращен в исходное состояние

Рисунок 7 – Спецификация прецедента «Вернуть снимок в исходное состояние»

В результате анализа функциональных требований для разрабатываемой системы были выделены варианты использования, которые будут реализованы в ходе разработки приложения. Для части вариантов использования были составлены спецификации, подробно описывающие выполнение конкретного варианта использования.

1.4 Анализ нефункциональных требований

Нефункциональные требования, они же свойства архитектуры, играют решающую роль в успехе приложения [7]. Следовательно, разрабатываемая система должна удовлетворять ряду нефункциональных требований, для успешной работы.

Должна быть обеспечена возможность легко разворачивать приложение и конфигурировать его под работу с другими системами. Конфигурация работы системы должна быть понятна для пользователя.

Пользовательский интерфейс должен быть интуитивно понятен. Создание снимков, управление ими и другие основные функции приложения, не должны

составлять труда для пользователя.

Приложение должно корректно функционировать в последних стабильных версиях современных веб-браузеров: Google Chrome, Mozilla Firefox, Safari, Яндекс Браузер.

Все перечисленные аспекты нефункциональных требований будут заложены в основу проектирования архитектуры и программной реализации системы для обеспечения ее стабильной, безопасной и эффективной эксплуатации

2 Проектирование системы создания, хранения и развёртывания веб-приложений

2.1 Архитектура приложения

При проектировании архитектуры разрабатываемой системы основной задачей являлось обеспечение возможности создания, хранения и развёртывания снапшотов веб-приложений с сохранением независимости отдельных компонентов системы.

Разрабатываемое программное средство состоит из нескольких функциональных частей, каждая из которых отвечает за отдельную область ответственности. Такое разделение позволяет упростить дальнейшее развитие системы, повысить удобство сопровождения и обеспечить возможность независимого изменения отдельных компонентов.

В качестве основной архитектурной модели была выбрана сервисная архитектура, основанная на разделении приложения на несколько взаимодействующих модулей. Данный подход позволяет изолировать функции управления снапшотами, хранения файлов и взаимодействия с пользовательским интерфейсом.

Учитывая специфику системы как инфраструктурного инструмента управления развёртываемыми единицами приложения, в дополнение к сервисной архитектуре было решено применить архитектурный паттерн Sidecar.

Паттерн Sidecar предполагает размещение вспомогательных компонентов вместе с основным приложением в отдельном процессе или контейнере [8]. Это позволяет изолировать логику управления снапшотами от среды выполнения целевого приложения, таким образом, что Sidecar-приложение отдаёт необходимую для развёртывания снапшота информацию приложению управляющим снапшотами. Таким образом основное приложение может получать необходимую для создания снапшотов информацию будучи развернутым как отдельный элемент системы

Схема архитектуры представлена на диаграмме развертывания на рисунке 8.

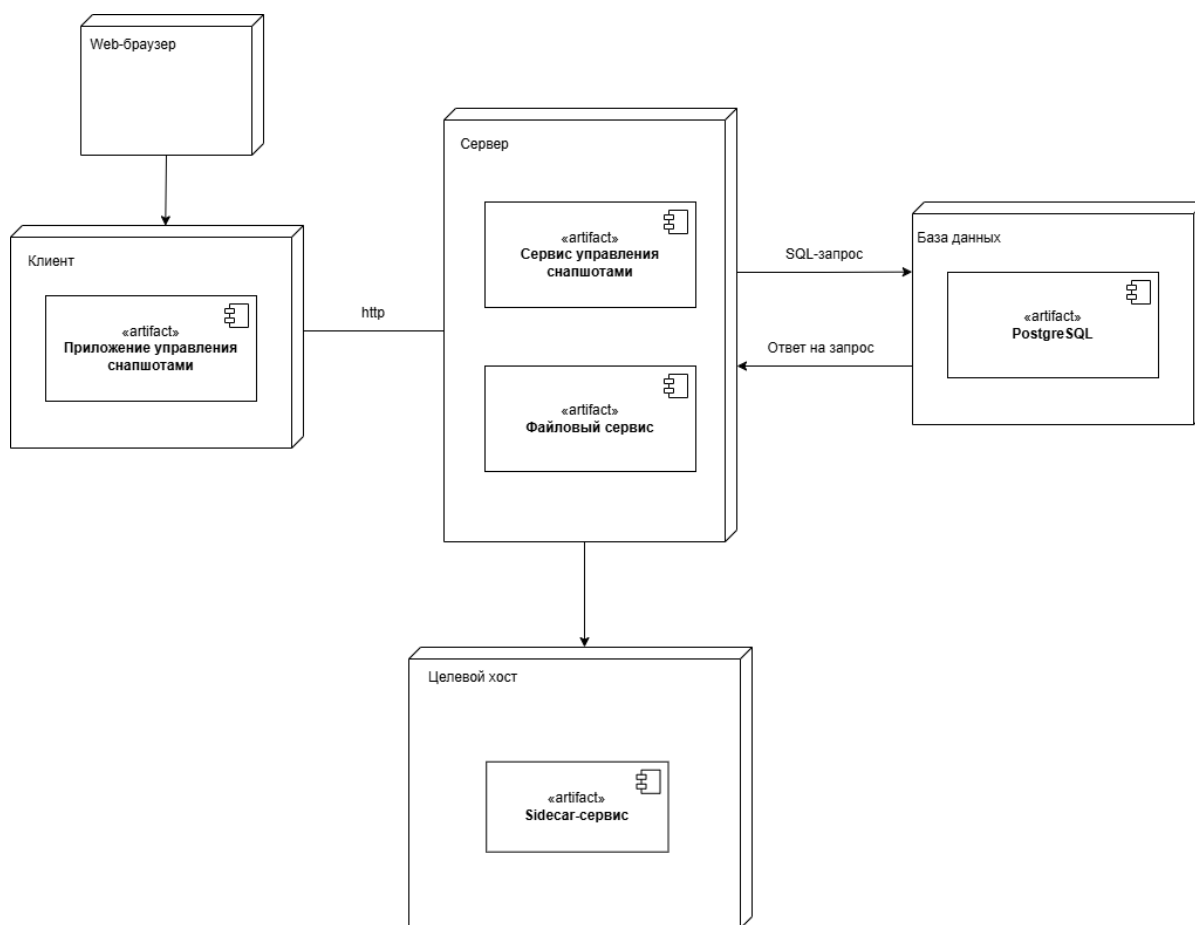


Рисунок 8 – Архитектура программных средств для системы создания, хранения и развёртывания снимотов веб-приложений

Серверная часть управляющей системы представлена в виде двух независимых сервисов, каждый из которых обладает собственным функционалом, отвечающим за определенную часть целостного приложения: центрального управляющего модуля управления снимками и выделенного сервера хранения статических файлов.

Пользователю предоставляется интерфейс взаимодействия с сервером с помощью клиентского веб-приложения, в котором отображаются все доступные для работы снимоты.

Программные модули должны быть обернуты в docker-контейнеры, для их дальнейшего развертывания на сервер. Управление порядком развёртывания занимается технология Docker-Compose.

2.2 Проектирование структуры хранения

2.3.1 Концептуальная модель базы данных

Концептуальная модель – это абстрактная модель, определяющая структуру моделируемой системы, свойства её элементов и причинно-следственные связи, присущие системе и существенные для лучшего понимания какие данные необходимо хранить в системе [9]. Для построения концептуальной модели не нужна привязка к конкретной СУБД, что делает ее подходящий для этапа проектирования структуры данных.

Концептуальная модель представляет собой выделение в список всех основных сущностей предметной области разрабатываемого приложения, а также ее графическое представление. Описание сущностей предметной области представлено в таблице 2.

Таблица 2 – Сущности БД

Название сущности	Идентификатор	Назначение
Снапшот	snapshot	Хранит сведения о доступных в системе конфигурациях снапшотов
Файл	file	Хранит сведения о сохраненных дампах БД

В ходе описания сущностей предметной области были выделены две сущности: снапшот и файл. Снапшот хранит метаданные для развертки копии приложения, а файл, будучи связанным со снапшотом с помощью внешнего ключа, хранит сведения о дампе базы данных снапшота.

Графическое представление концептуальной модели представлено на рисунке 11.

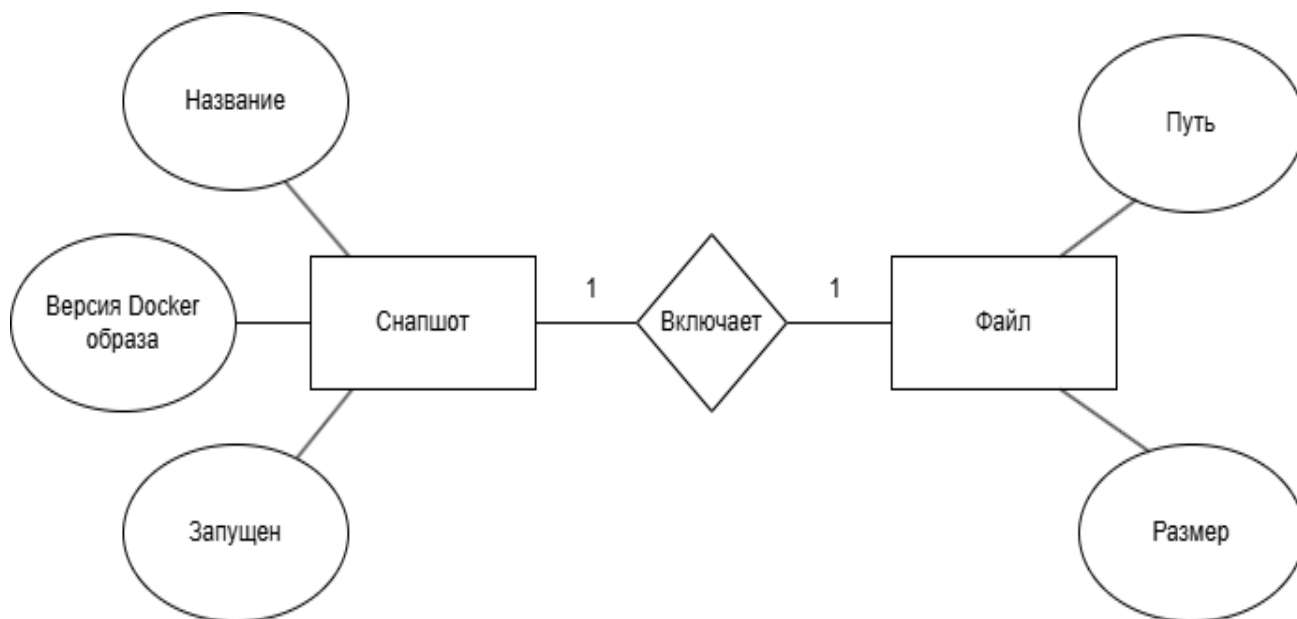


Рисунок 11 – Концептуальная модель БД

Концептуальная модель базы данных построена в среде draw.io.

2.3.2 Логическая модель базы данных

Составив концептуальную модель базы данных можно перейти к построению логической модели, которая строится на основе концептуальной с добавлением описания атрибутов к сущностям, соответствующие планируемой схеме базы данных.

Была составлена логическая модель базы данных «Снапшоты». Результат представлен на рисунке 12.

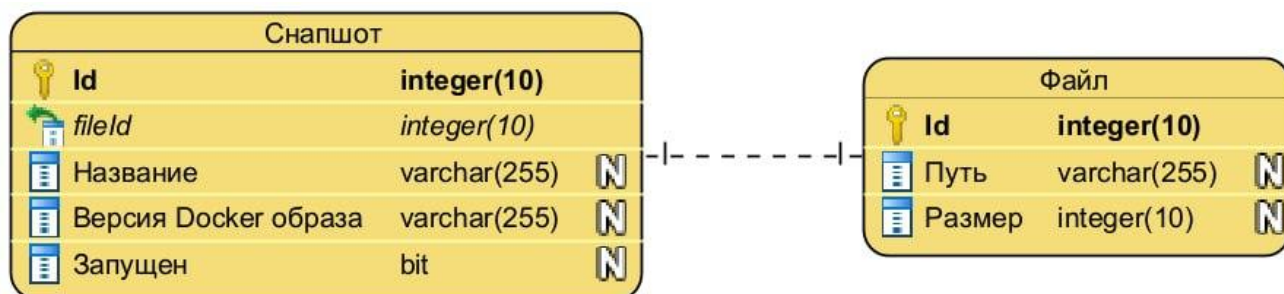


Рисунок 12 – Логическая модель БД «Снапшоты»

При описании логической модели для сущности «Снапшот» были выделены такие атрибуты как: название, версия Docker образа, запущен ли снапшот. Для сущности «Файл» определены атрибуты: путь и размер.

2.3 Проектирование серверной части

Была спроектирована архитектура программного средства для создания, хранения и развертывания снапшотов веб-приложений, основывающаяся на проделанном анализе функциональных и нефункциональных требований.

На этапе проектирования были определены основные компоненты системы, их назначение и принципы взаимодействия между собой. Архитектура приложения построена с учетом разделения ответственности между отдельными сервисами, что позволяет упростить сопровождение системы и обеспечить возможность дальнейшего расширения функциональности.

Были выделены 3 основных компонента системы хранения и управления снапшотами:

- компонент управления снапшотами;
- компонент файлового сервера;
- компонент Docker команд.

Разделение программы на компоненты было проведено таким образом, чтобы обеспечить высокую связность внутренних элементов и низкую связанность компонентов между друг другом.

Компонент управления снапшотами отвечает за основную логику приложения. В его функционал входит создание, развертка, остановка, удаление и сброс состояния снапшота

Компонент файлового сервера отвечает за хранение и отдачу файлов дампа базы данных для тех или иных снапшотов. Принадлежность дампа базы данных к определенному снапшоту обеспечивается благодаря связи в базе данных

Компонент Docker команд отвечает за развертку снапшотов и внесение их в сеть traefik, для того, чтобы до них можно было достать по пути введенным пользователем при создании снапшота.

Основным сценарием работы системы является процесс создания нового снимка приложения. Для описания данного процесса была разработана диаграмма деятельности, представленная на рисунке 9.

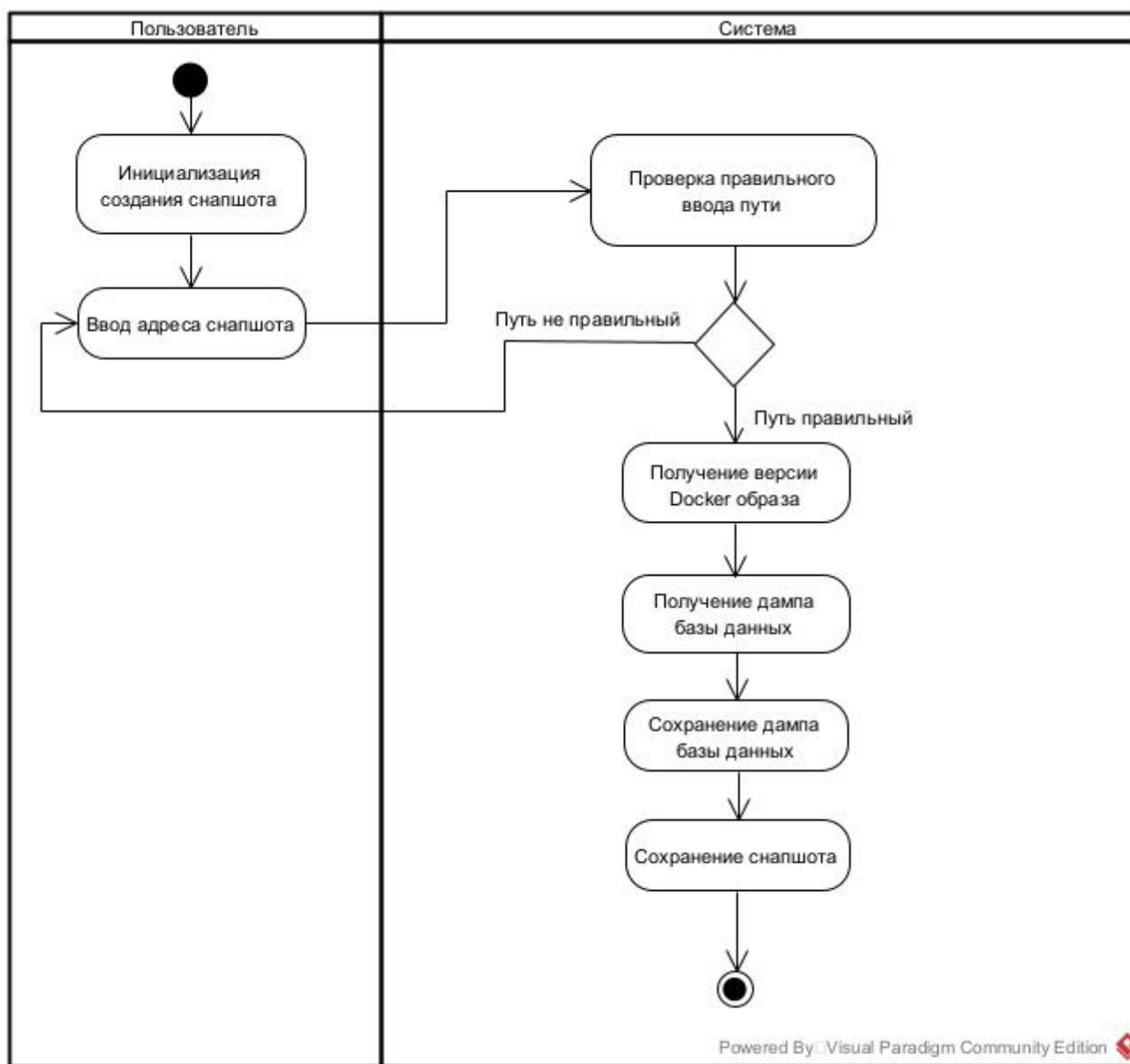


Рисунок 9 – Диаграмма деятельности создания снимка

На диаграмме выделены две основные области взаимодействия: «Пользователь» и «Система». Пользователь инициирует создание снимка через интерфейс приложения, после чего сервер выполняет последовательность необходимых операций. Система проверяет корректность введенных данных, получает информацию о текущем состоянии приложения, в частности текущую версию Docker образа приложения и дампы базы данных, формирует резервную

копию базы данных и сохраняет необходимые файлы в хранилище.

После успешного выполнения операции информация о созданном снимке сохраняется в базе данных. Пользователю возвращается результат выполнения операции с возможностью дальнейшего управления созданным объектом.

Для описания взаимодействия компонентов при выполнении операции создания снимка была разработана диаграмма последовательности, представленная на рисунке 10.

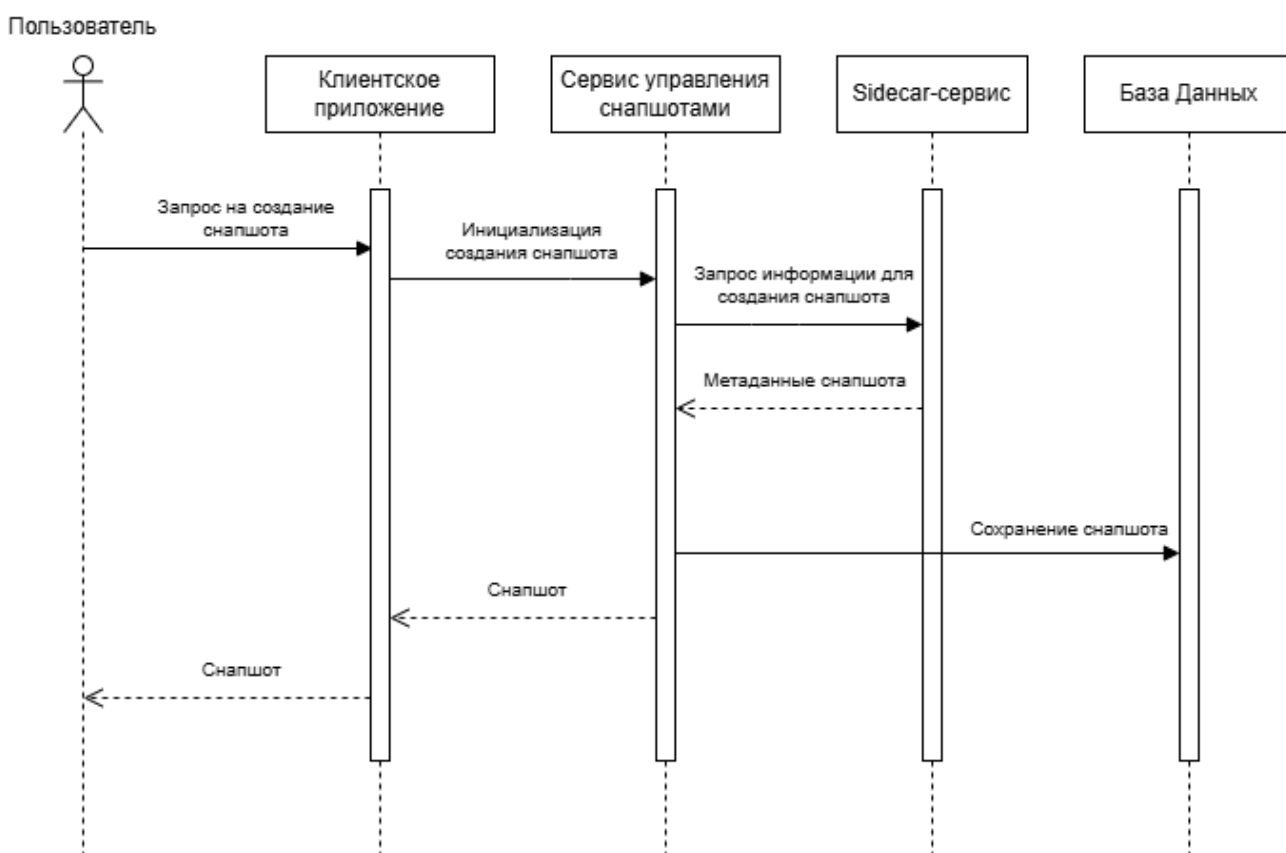


Рисунок 10 – Диаграмма последовательности создания снимка

В данном сценарии участвуют основные элементы системы: пользователь, клиентское приложение, сервис управления снимками, Sidecar-сервис и база данных. Пользователь отправляет запрос на создание снимка, после чего сервис снимков выполняет необходимые проверки и инициирует получение данных от вспомогательных сервисов.

Sidecar-сервис отвечает за взаимодействие с окружением копируемого приложения и получение данных, необходимых для восстановления состояния

системы. После получения информации сервер сохраняет метаданные снапшота в базе данных, а файлы резервной копии передаются в файловое хранилище.

Таким образом, в рамках данного этапа была выполнена детализация архитектуры серверной части и описаны основные сценарии взаимодействия компонентов системы. Разработанные модели позволяют определить структуру приложения, распределение ответственности между сервисами и последовательность выполнения ключевых операций, что является основой для дальнейшей реализации программного продукта.

2.4 Проектирование пользовательского интерфейса

Интерфейс веб-приложения должен отвечать всем функциональным требованиям, а также должен быть удобным, минималистичным и интуитивно понятным в использовании для администраторов и разработчиков.

Проектируемый интерфейс включает в себя главную страницу панели управления снапшотами. На главной странице располагается интерактивный рабочий стол, отображающий реестр имеющихся в системе снапшотов веб-приложений, их текущие статусы (активен или остановлен), а также управляющие элементы для вызова функций запуска, остановки и удаления сущностей. При отсутствии зарегистрированных в системе снапшотов на панели автоматически отображается соответствующее информационное уведомление об отсутствии данных.

Также в интерфейсе приложения должны быть реализованы модальные окна при создании снапшота, для ввода настроек, и при удалении снапшота, для подтверждения действия.

Каждая составляющая интерфейса будет реализована с использованием компонентов библиотеки Naive-UI, контролирующих свое состояние и функциональность.

2.5 Требования к аппаратному обеспечению

Также были проанализированы требования системы к аппаратному

обеспечению и составлен перечень характеристик, соблюдение которых будет гарантировать совместимость программы с устройством на котором она будет исполняться. Перечень требований представлен в таблице 3.

Таблица 3 – Требования к программному обеспечению

Характеристика	Минимальные требования	Рекомендуемые требования
Процессор	x64 с 1.5 ГГц	x64 с 3 ГГц и более
Оперативная память	4 Гб	8 Гб и более
Тип и объем жесткого диска	HDD со 100 Гб	SSD с 1 Тб и более
Пропуская способность сети	1 Гбит/сек	1 Гбит/сек
Операционная система	Ubuntu 20.04 LTS	Ubuntu 22.04 LTS и выше
Обязательное ПО	Docker 20.10	Docker 26.0 и выше

Специфика системы предполагает локальное размещение дампов базы данных снапшотов на едином сервере. По этой причине важным требованием к аппаратному обеспечению является использование накопителей постоянной памяти большого объема.

Таким образом были сформированы требования к аппаратному обеспечению устройства, на котором будет развернута серверная часть приложения. Минимальные требования позволят пользоваться функционалом приложения в полном объеме. Рекомендуемые требования, в добавок к этому, позволят достичь максимальной производительности системы, ее стабильной работы и эффективности.

2.6 Планирование разработки и оценка бюджета

Неотъемлемой частью каждого проекта является планирование и оценка бюджета. Для управления проектом использовалась программа ProjectLibre [11]. Данная программа удовлетворяет всем требованиям для составления плана разработки проекта ввиду своей простоты использования и богатого функционала.

Планирование разработки включает в себя определение последовательности выполнения работ, оценки требуемых трудовых ресурсов и прогноз сроков реализации проекта. На основе этих данных можно будет получить обзорный отчет проекта, содержащий всю необходимую информацию.

В процессе планирования разработки была выделена дата старта проекта, а также были спланированы задачи, описание которых представлено на рисунке 13

	Название	Продолжите...	Начало	Окончание	Предшествующие
1	☐ Разработка приложения	49,5 дней	08.04.2026, 08:00	16.06.2026, 13:00	
2	☐ Постановка задачи и анализ требований	7 дней	08.04.2026, 08:00	16.04.2026, 17:00	
3	Анализ предметной области	3 дней	08.04.2026, 08:00	10.04.2026, 17:00	
4	Обзор аналогов	2 дней	13.04.2026, 08:00	14.04.2026, 17:00	3
5	Анализ функциональных требований	2 дней	15.04.2026, 08:00	16.04.2026, 17:00	4
6	Анализ нефункциональных требований	2 дней	15.04.2026, 08:00	16.04.2026, 17:00	4
7	Завершение постановки задачи и анализа требований	0 дней	16.04.2026, 17:00	16.04.2026, 17:00	6
8	☐ Проектирование программных средств	14 дней	17.04.2026, 08:00	06.05.2026, 17:00	
9	Проектирование архитектуры приложения	3 дней	17.04.2026, 08:00	21.04.2026, 17:00	7
10	Проектирование структуры хранения	2 дней	22.04.2026, 08:00	23.04.2026, 17:00	9
11	Проектирование пользовательского интерфейса	5 дней	24.04.2026, 08:00	30.04.2026, 17:00	10
12	Требования к аппаратному обеспечению	3 дней	01.05.2026, 08:00	05.05.2026, 17:00	11
13	Планирование разработки и оценка бюджета	1 день	06.05.2026, 08:00	06.05.2026, 17:00	12
14	Завершение проектирования программных средств	0 дней	06.05.2026, 17:00	06.05.2026, 17:00	13
15	☐ Реализация программных средств	16,5 дней	07.05.2026, 08:00	29.05.2026, 13:00	
16	Реализация архитектуры приложения	3 дней	07.05.2026, 08:00	11.05.2026, 17:00	14
17	Реализация базы данных	6 дней	12.05.2026, 08:00	19.05.2026, 17:00	16
18	Реализация серверной части	6 дней	20.05.2026, 08:00	27.05.2026, 17:00	17
19	Реализация интерфейса приложения	1,5 дней	28.05.2026, 08:00	29.05.2026, 13:00	18
20	Завершение реализации программных средств	0 дней	29.05.2026, 13:00	29.05.2026, 13:00	19
21	☐ Контроль качества	7 дней	29.05.2026, 13:00	09.06.2026, 13:00	
22	Выбор метрик программного кода	3 дней	29.05.2026, 13:00	03.06.2026, 13:00	20
23	Функциональное тестирование	3 дней	03.06.2026, 13:00	08.06.2026, 13:00	22
24	Вычисление метрик программного кода	1 день	08.06.2026, 13:00	09.06.2026, 13:00	23
25	Завершение тестирования	0 дней	09.06.2026, 13:00	09.06.2026, 13:00	24
26	Оформление и сбор документации	4 дней	09.06.2026, 13:00	15.06.2026, 13:00	25
27	Ввод в эксплуатацию	1 день	15.06.2026, 13:00	16.06.2026, 13:00	26
28	Завершение проекта	0 дней	16.06.2026, 13:00	16.06.2026, 13:00	27

Рисунок 13 – Список задач проекта

Визуализация запланированных задач была проведена с помощью диаграммы Ганта, состоящая из горизонтальных полос, показывающих

длительность той или иной задачи [12].

Список ресурсов, используемых при выполнении задач представлен на рисунке 14.

	⦿	Название	Тип	Максимальные единицы	Стандартная ставка	Ставка сверхурочных	Затраты на одно испол...	Начислять	Основной календарь
1	👤	Аналитик	Работа	100 %	238,00 Р/час	357,00 Р/час	0,00 Р Пропорционально	Пятидневка	
2	👤	Программист	Работа	100 %	300,00 Р/час	450,00 Р/час	0,00 Р Пропорционально	Пятидневка	
3	👤	Тестировщик	Работа	100 %	286,00 Р/час	429,00 Р/час	0,00 Р Пропорционально	Пятидневка	
4	👤	Дизайнер	Работа	100 %	286,00 Р/час	429,00 Р/час	0,00 Р Пропорционально	Пятидневка	
5	👤	Проектировщик	Работа	100 %	290,00 Р/час	435,00 Р/час	0,00 Р Пропорционально	Пятидневка	
6	💻	Компьютер	Материал		40000,00 Р		0,00 Р Начало		
7	📄	Бумага	Материал		500,00 Р		0,00 Р Начало		
8	🖨	Принтер	Материал		5000,00 Р		0,00 Р Начало		

Рисунок 14 – Список ресурсов проекта

Полученная диаграмма Ганта представлена на рисунке 15.

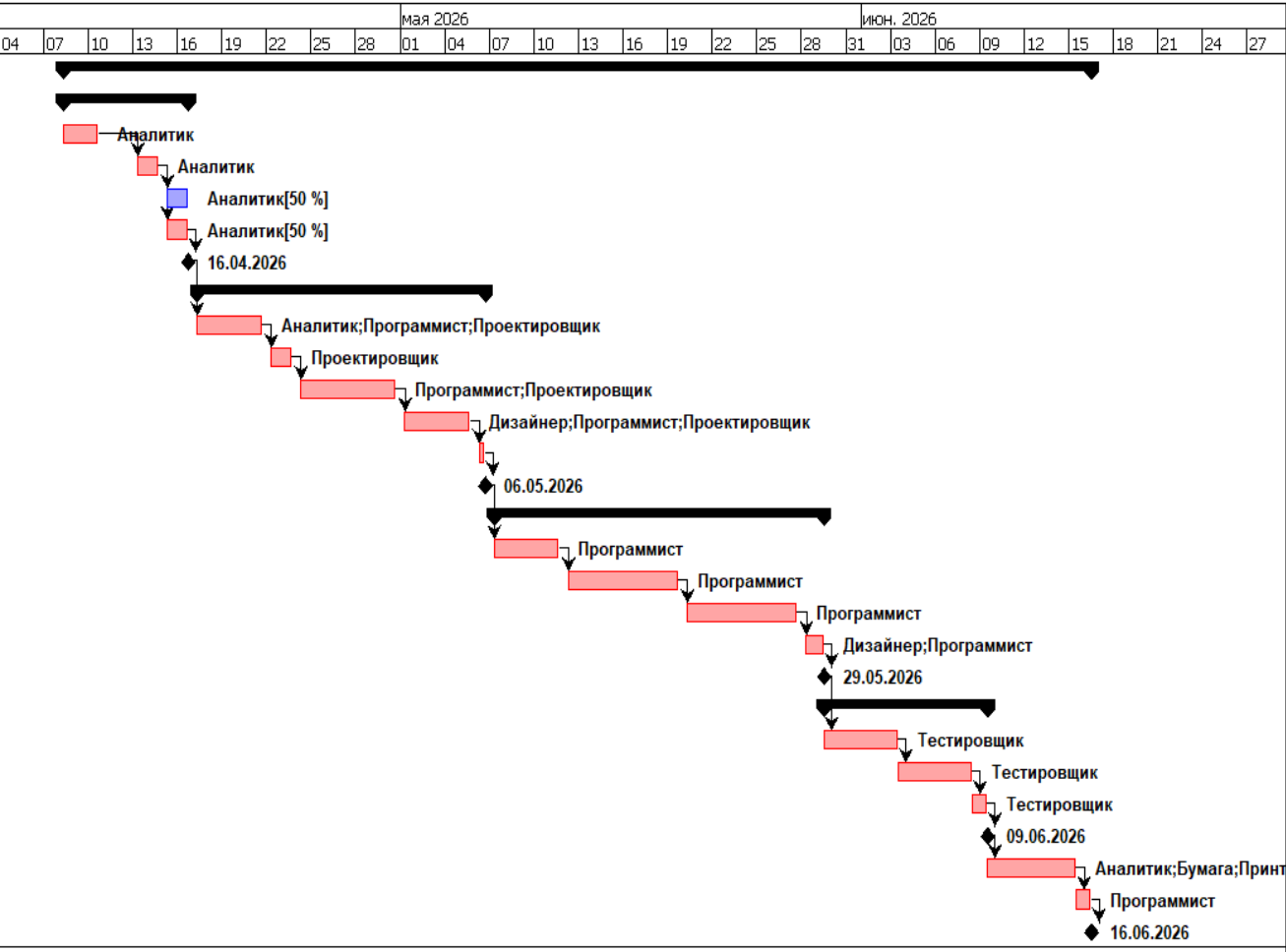


Рисунок 15 – Диаграмма Ганта проекта

Для оценки бюджета программой был сгенерирован обзорный отчет

проекта, в котором содержатся сведения о стоимости проекта и времени, необходимом для реализации проекта. На рисунке 16 представлен обзорный отчет проекта.

Начало:	07.04.2026, ...	Окончание:	09.06.2026, 17:00
Начало базового плана :		Окончание базового плана :	
Фактическое начало:		Фактическое окончание:	
Продолжительность:	46 дней	Длительность базового плана :	0 дней
Фактическая продолжительность:	0 дней	Оставшаяся продолжительность:	46 дней
Работа:	520 часов	Работа базового плана :	0 часов
Фактическая работа:	0 часов	Оставшаяся работа:	520 часов
Стоимость:	151948,00 Р	Затраты базового плана :	0,00 Р
Фактические затраты:	0,00 Р	Оставшаяся стоимость:	151948,00 Р

Рисунок 16 – Обзорный отчет проекта

Из отчета, представленного на рисунке 16, видно, что предполагаемые затраты на разработку составляют 151948 рублей, а общая длительность проекта – 46 дней.

3 Реализация программных средств системы создания, хранения и развёртывания снапшотов веб-приложений

3.1 Реализация архитектуры приложения

Для реализации программного обеспечения системы создания, хранения и развёртывания снапшотов веб-приложений была выбрана распределенная сервисная архитектура в сочетании с системным шаблоном проектирования Sidecar. Этот паттерн нужен для расширения и улучшения функциональности основного приложения без внесения в него изменений [13].

Sidecar – это независимо разворачиваемые инфраструктурные службы, функционирующие в едином пространстве имен совместно с целевым бизнес-приложением. Они предлагают гибкий подход к расширению возможностей распределенных систем без прямого вмешательства в код основного приложения. С технологической точки зрения, сопутствующий контейнер инкапсулирует внутри себя низкоуровневые операции взаимодействия с окружением веб-приложения и его базой данных, предоставляя центральному управляющему серверу четко определенный интерфейс. Благодаря этому, утилиты снятия дампов и внутренней конфигурации сред выполнения остаются полностью скрыты в пределах изолированного сервиса, обеспечивая должный уровень безопасности и слабую связанность компонентов.

На рисунке 17 представлено использование паттерна Sidecar в системе управления снапшотами.

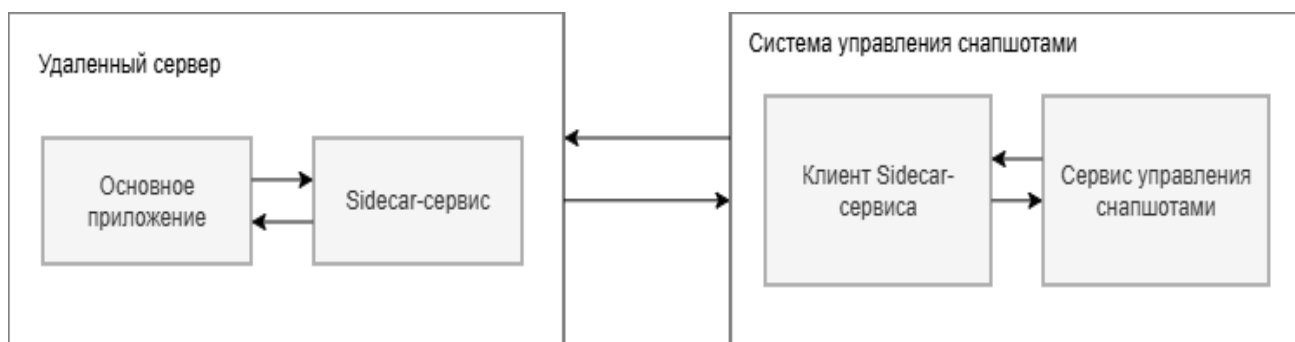


Рисунок 17 – Паттерн Sidecar в системе управления снапшотами

Серверная часть разрабатываемой системы состоит из сервиса управления снапшотами и изолированного модуля долгосрочного хранения статических файлов. Сервисы общаются между собой по средствам

TRPC позволяет разрабатывать полностью безопасные с точки зрения типов API для клиент-серверных приложений [14], что в свою очередь облегчает разработку интерфейсов взаимодействия сервисов и их дальнейшее использование.

Для обеспечения переключения пользователей между развернутыми снапшотами в архитектуру серверной части интегрирован высокопроизводительный инверсный прокси-сервер Traefik [15]. Он выступает в качестве единой точки входа для снапшотов приложений.

3.2 Реализация базы данных

Физическая модель базы данных – конкретное техническое описание того, как будет реализована база данных в реальной СУБД [16], включающая в себя описание типов данных полей, связей таблиц и индексов.

На рисунке 18 представлена физическая модель базы данных «Снапшоты»

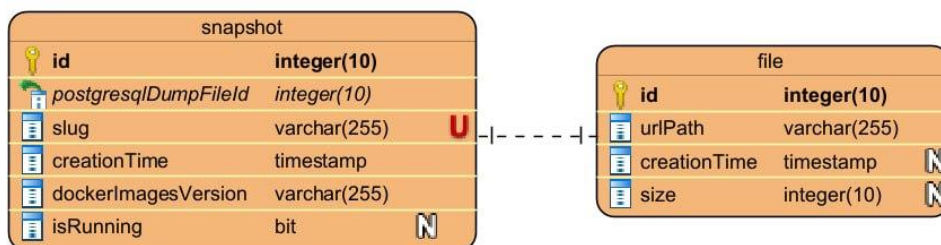


Рисунок 18 – Физическая модель базы данных «Снапшоты»

Описание схемы базы данных, соответствующая разработанной физической модели была реализованна помощью инструментов библиотеки Drizzle.

Описание схемы базы данных «Снапшоты» представлено на листинге 1.

Листинг 1 – Описание схемы базы данных «Снапшоты»

```

import { relations } from 'drizzle-orm';
import * as drizzle from 'drizzle-orm';
import { boolean, doublePrecision, integer, pgTable, text } from
  
```

```
'drizzle-orm/pg-core';
const defaultFields = {
  id: integer().primaryKey().generatedAlwaysAsIdentity(),
  creationTime:
doublePrecision().notNull().default(drizzle.sql.raw('extract(epoch
from now()) * 1000')),
};
export const files = pgTable('files', {
  ...defaultFields,
  urlPath: text().notNull(),
  size: integer().notNull(),
});
export const snapshots = pgTable('snapshots', {
  ...defaultFields,
  postgresqlDumpFileId: integer().notNull().references(() =>
files.id),
  slug: text().notNull().unique(),
  dockerImagesVersion: text().notNull(),
  isRunning: boolean().notNull(),
});
```

База данных была развернута с помощью механизма миграций, предоставленного инструментарием библиотеки Drizzle.

3.3 Реализация серверной части приложения

Организация межкомпонентного взаимодействия клиентской и серверной частей системы реализована на основе монорепозитария [17]. Архитектура монорепозитария обеспечивает хранение исходного кода панели управления снапшотами и Sidcar-приложения в единой кодовой базе, предоставляя возможность удобной отладки приложения и быстрого добавления новых компонентов. Фрагменты кода серверной части системы приведены в приложении Б. С полным кодом можно ознакомиться на GitHub [18]

Вся серверная часть программного интерфейса функционирует в среде выполнения Bun. Вместо классической архитектуры REST API, в качестве технологии построения программного интерфейса задействован фреймворк удаленного вызова процедур tRPC, обеспечивающий синхронизацию предоставляемых программных интерфейсов на сервере и клиента.

Использование tRPC позволяет напрямую экспортировать типы функций серверной части на сторону клиентской части приложения, обеспечивая строгую статическую типизацию на этапе компиляции: любое изменение контракта

удаленной процедуры автоматически вызывает ошибку сборки клиентского приложения, полностью предотвращая появление несогласованных интерфейсов.

В соответствии с функциональной спецификацией проекта, базовый узел API управляющей системы описывается набором удаленных процедур, разделенных на запросы на чтение данных (query) и мутации, изменяющие состояние системы (mutation). Описание структуры и возможностей разработанного программного интерфейса tRPC API приведено в таблице 4.

Таблица 4 – Методы tRPC серверной части приложения

Название процедуры	Тип процедуры	Входные параметры	Описание процедуры
list	Query	Отсутствуют	Получение списка всех зарегистрированных снапшотов.
create	Mutation	slug: string	Создание нового снапшота приложения.
up	Mutation	id: number	Развертывание сохраненного снапшота.
down	Mutation	id: number	Остановка развернутого снапшота
resetPostgresStateFromDump	Mutation	id: number	Принудительный сброс состояния базы данных снапшота.
remove	Mutation	id: number	Полное удаление остановленного снапшота.

Также было реализовано API Sidcar-приложения рассчитанного на получение информации необходимой для развертывания приложения. Методы tRPC api Sidcar-приложения приведены в таблице 5.

Таблица 5 – Методы tRPC серверной части приложения

Название процедуры	Тип процедуры	Входные параметры	Описание процедуры
getDockerImageVersion	Query	Отсутствуют	Получение версии Docker образа.
getPostgresqlDump	Query	Отсутствуют	Получение текущего дампа базы данных

При создании снапшота, серверная часть приложения обращается к Sidcar-приложению, развернутому рядом с основным приложением. Фрагмент

кода создания снимка представлен на листинге 2.

Листинг 2 – Фрагмент кода создания снимка

```
const dockerImagesVersion = await
apiClients.appSidecar.getDockeImagesVersion.query();
const postgresqlDumpFileContent = await
apiClients.appSidecar.getPostgresqlDump.query();
```

Таким образом серверная часть приложения получает информацию от Sidecar-приложения для сохранения данных снимка приложения, для его дальнейшего разворачивания на выбранном пользователем домене.

3.4 Реализация интерфейса приложения

В соответствии с требованиями к пользовательскому интерфейсу, предоставленными в пункте 2.3, были реализованы следующие компоненты, отображаемые на главной странице:

- компонент списка созданных снимков;
- компонент элемента списка снимков;
- модальное окно создания снимка;
- модальное окно подтверждения удаления снимка;
- поддержку резервного копирования и восстановления приложений.

Интерфейс программы реализован в формате веб страницы, на которой находятся все компоненты описанные выше. Запуская приложение, пользователь видит страницу со списком снимков (рисунок 19).

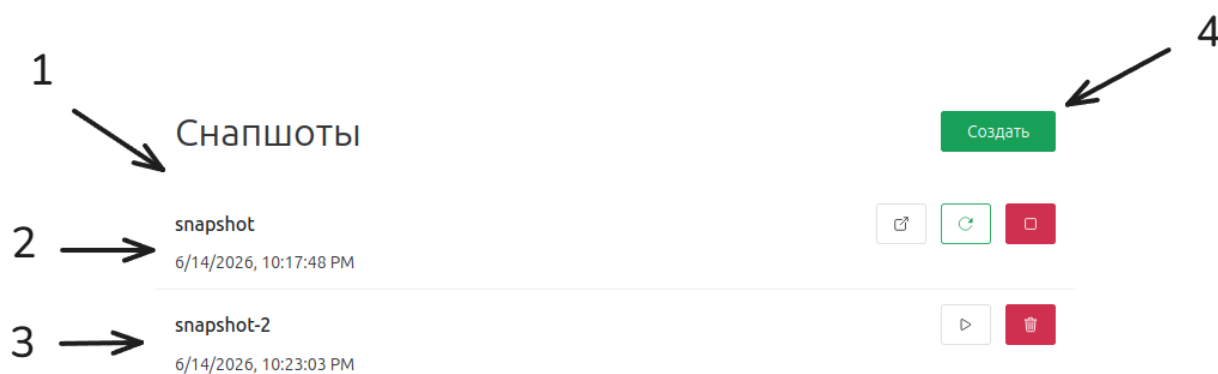


Рисунок 19 – Главная страница приложения

Основные элементы главной страницы:

- 1 – Список созданных снимков;
- 2 – Развернутый снимок;
- 3 – Не активный снимок;
- 4 – Кнопка создания нового снимка.

Элемент списка снимков, может иметь разные наборы кнопок управления в зависимости от текущего его состояния. Для активного снимка доступны такие варианты действий как:

- остановить снимок;
- сброс текущего состояния базы данных снимка;
- переход к снимку в новой вкладке.

При нажатии на кнопку сброса текущего состояния базы данных или остановки снимка пользователь увидит анимацию загрузки на время выполнения операции. Активный элемент списка снимков представлен на рисунке 20.



Рисунок 20 – Активный элемент списка снимков

Для не активного снимка доступны такие варианты действий как:

- запустить снимок;
- удалить снимок.

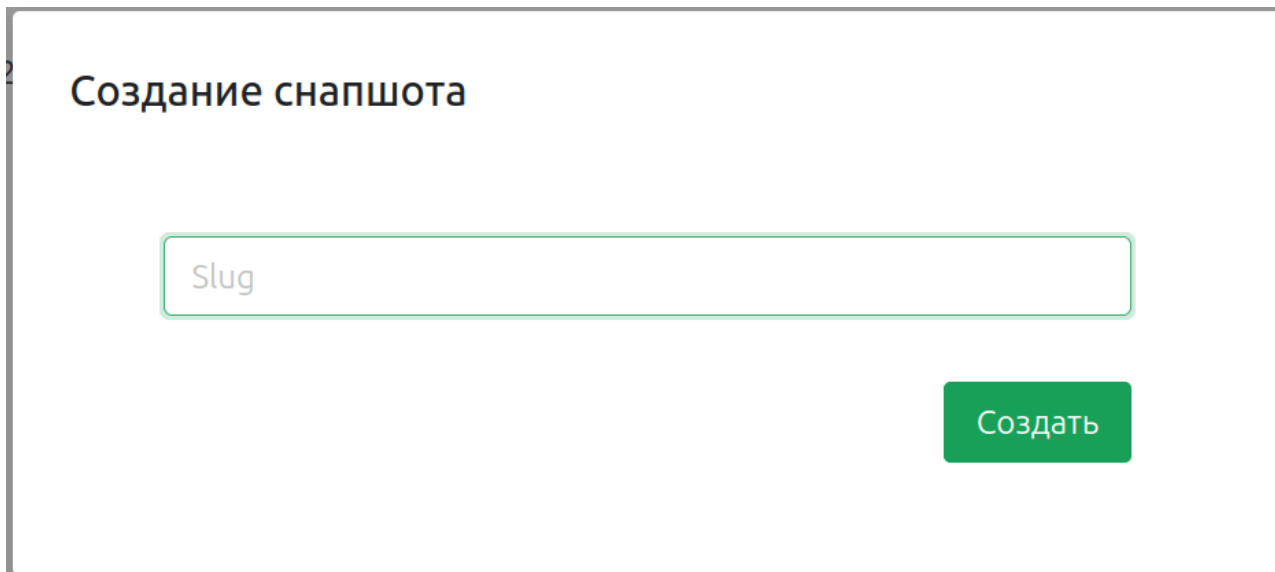
Не активный элемент списка снимков представлен на рисунке 21.



Рисунок 21 – Не активный элемент списка снимков

Также при нажатии на кнопку создания снимка открывается модальное окно ввода адреса, на котором будет развернут снимок. Введенный адрес в дальнейшем повлияет на путь, по которому можно будет открыть развернутый

снимок. Модальное окно создания снимка представлено на рисунке 22.



Создание снимка

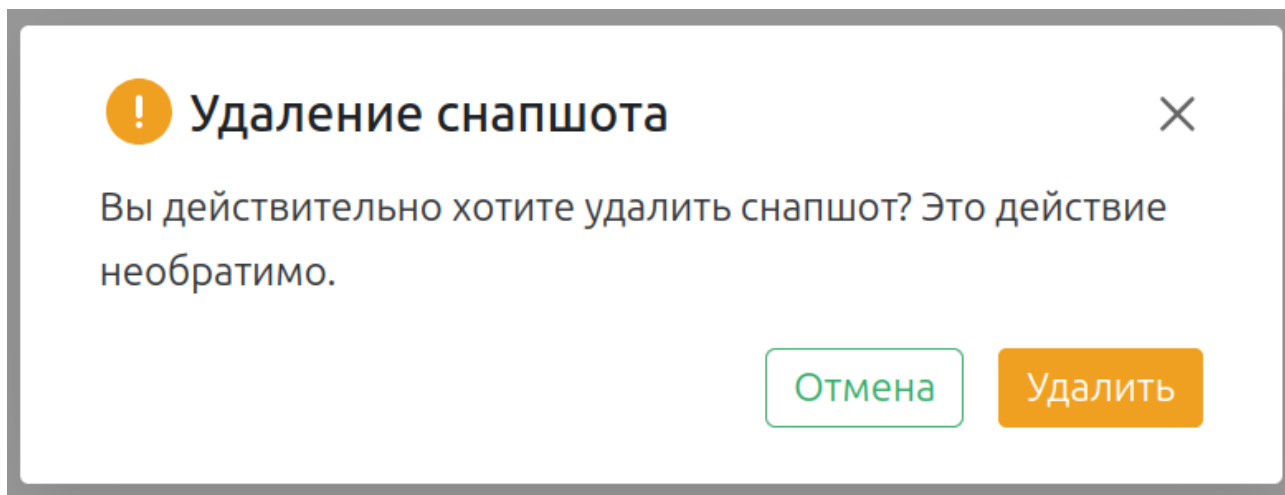
Slug

Создать

Рисунок 22 – Модальное окно создания снимка

Начало создания снимка сопровождается показом анимации загрузки, дающий пользователю понять, что идет процесс создания снимка

При нажатии на кнопку удаления снимка открывается модальное окно подтверждения удаления снимка. Модальное окно подтверждения удаления снимка представлено на рисунке 23.



! Удаление снимка

Вы действительно хотите удалить снимок? Это действие необратимо.

Отмена Удалить

Рисунок 23 – Модальное окно подтверждения удаления снимка

Программная реализация пользовательского интерфейса системы автоматизации управления снимками выполнена в виде независимого клиентского приложения, интегрированного в монорепозитарий. Клиентская

часть спроектирована по архитектурному шаблону *SPA (Single Page Application)* на базе компонентно-ориентированного фреймворка Vue 3 и инструмента сборки Vite.

Также при создании пользовательского интерфейса использовалась библиотека компонентов Naïve-UI, на основе которых были разработаны свои компоненты пользовательского интерфейса.

4 Контроль качества программного обеспечения

4.1 Обоснование выбора режима тестирования и метрик кода

Тестирование служит ключевым фактором обеспечения качества программного обеспечения [19], поэтому при разработке программного обеспечения является одним из основных этапов. Контроль качества кодовой базы позволяет существенно повысить надежность системы, усилить уровень информационной безопасности против потенциальных угроз, а также гарантировать удобство и безошибочность взаимодействия пользователя с приложением.

Наиболее популярными способами тестирования являются:

- интеграционное тестирование;
- функциональное тестирование;
- тестирование производительности;
- модульное тестирование.

Функциональное тестирование – это методика верификации программного обеспечения, целью которой является проверка полного соответствия фактического поведения разработанного программного продукта заданным функциональным и техническим требованиям.

Существуют метрики, помогающие понять общее качество кода, его читаемость и модифицируемость. В рамках данной выпускной квалификационной работы была выбрана метрика Холстеда.

Метрика Холстеда – это система количественной оценки сложности программного обеспечения, основанная на подсчете числа уникальных операторов (ключевые слова, знаки операций) и операндов (переменные, константы) в исходном коде [20].

4.2 Функциональное тестирование создания снимка

Функциональное тестирование помогает обнаружить проблемы, которые могут возникнуть при взаимодействии пользователя с тестируемой системой.

Для системы создания, хранения и развертывания снапшотов веб-приложений, одним из наиболее важных вариантов использования является создания снапшота. Данный способ взаимодействия пользователя с системой включает в себя этап ввода пользователем входных данных, которые должны правильно обрабатываться на сервере.

Программа должна правильно реагировать при возникновении проблем с неправильно введенными входными данными, невозможности достать данные снапшота или техническими проблемами при создании снапшота, например нехватки памяти на устройстве.

В процессе анализа поведения системы были определены причины и следствия, которые представлены в таблице 6.

Таблица 6 – Причины и следствия

Причины	
П1	Корректная длина имени
П2	Имя содержит допустимые символы
П3	Имя свободно
П4	Источник существует
П5	Источник доступен
Следствия	
С1	Сообщение: «Некорректная длина имени»
С2	Сообщение: «Недопустимые символы в названии»
С3	Сообщение: «Имя занято»
С4	Сообщение: «Источник не найден»
С5	Сообщение: «Источник недоступен»
С6	Снапшот успешно создан

На основе таблицы 6 была разработана функциональная диаграмма, описывающая поведение функции с учетом ошибок, возникающих во время ее работы.

Продолжение таблицы 7

Параметр	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
П5	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
С1	1	1	0	0	0	1	1	0	0	0	1	1	0	0	0
С2	1	0	1	0	0	1	0	1	0	0	1	0	1	0	0
С3	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0
С4	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
С5	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0
С6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

На основе таблицы решений были описаны следующие тест-кейсы для проверки системы на обработку возникающих во время выполнения функции ошибок. Тест-кейсы представлены в таблице 8.

Таблица 8 – Тест-кейсы

№	Входные данные	Выходные данные
1	Имя: ab#; Источник: unknown_app	Некорректная длина имени, Имя содержит недопустимые символы, Источник не существует
2	Имя: ab; Источник: unknown_app	Некорректная длина имени, Источник не существует
3	Имя: snap#main; Источник: unknown_app	Имя содержит недопустимые символы, Источник не существует
4	Имя: snapshot_busy; Источник: unknown_app	Имя уже занято, Источник не существует
5	Имя: snapshot_main; Источник: unknown_app	Источник не существует
6	Имя: ab#; Источник: app_offline	Некорректная длина имени, Имя содержит недопустимые символы, Источник недоступен
7	Имя: ab; Источник: app_offline	Некорректная длина имени, Источник недоступен
8	Имя: snap#main; Источник: app_offline	Имя содержит недопустимые символы, Источник недоступен
9	Имя: snapshot_busy; Источник: app_offline	Имя уже занято, Источник недоступен

Продолжение таблицы 8

№	Входные данные	Выходные данные
10	Имя: snapshot_main; Источник: app_offline	Источник недоступен
11	Имя: ab#; Источник: app_prod	Некорректная длина имени, Имя содержит недопустимые символы
12	Имя: ab; Источник: app_prod	Некорректная длина имени
13	Имя: snap#main; Источник: app_prod	Имя содержит недопустимые символы
14	Имя: snapshot_busy; Источник: app_prod	Имя уже занято
15	Имя: snapshot_main; Источник: app_prod	Снапшот успешно создан

Таблица содержит входные и соответствующие выходные данные

4.3 Метрика Холстеда

В качестве рассматриваемой функции для вычисления метрик Холстеда была выбрана функция развертывания снапшота – `up`. Код данной функции представлен в листинге Б.10 приложения Б.

Прежде всего для метрик Холстеда нужно рассчитать количество уникальных операторов программы ($n1$), количество операндов ($n2$) и число их вхождений в коде ($N1$ и $N2$ соответственно) [21].

Операторы функции `up` и число их вхождений приведены в таблице 9.

Таблица 9 – Операторы функции `up`

№	Оператор	Количество повторений
1	<code>async</code>	1
2	<code>const</code>	3
3	<code>await</code>	7
4	<code>if</code>	1
5	<code>try</code>	1

Продолжение таблицы 9

№	Оператор	Количество повторений
6	catch	1
7	=>	2
8	=	3
9	. (вызов метода, обращение к полю)	14
10	!	1
11	:	4
12	() (круглые скобки)	14
13	{ } (фигурные скобки)	5
14	; (точка с запятой)	8
	n1 = 14	N1 = 65

Операнды функции `up` и число их вхождений приведены в таблице 10.

Таблица 10 – Операнды функции `up`

№	Оператор	Количество повторений
1	ctx	2
2	input	3
3	snapshot	5
4	db	2
5	query	1
6	snapshots	2
7	postgresqlDumpFile	1
8	q	3
9	id	3
10	isRunning	2
11	snapshotCmdParams	4
12	slug	2
13	imageVersion	1
14	‘Снапшот не найден’	1
15	‘Не удалось запустить снапшот’	1
	n2 = 15	N2 = 33

Учитывая уже рассчитанные данные можно составить сводку исходных данных, которые понадобятся для вычисления метрики Холстеда. Это данные

которые можно рассчитать визуально, проанализировав код функции. Сводку исходных данных приведена в таблице 11.

Таблица 11 – Сводка исходных данных для расчета метрики Холстеда

Обозначение	Величина	Описание
n1	14	Количество уникальных операторов программы
N1	65	Число вхождений операторов в коде
n2	15	Количество уникальных операндов программы
N2	33	Число вхождений операндов в коде
CC	2	Цикломатическая сложность кода
LOC	32	Количество строк кода

Используя полученные данные можно рассчитать искомую сложность функции с помощью формул метода Холстеда.

Длина программы рассчитывается по формуле:

$$N=N1+N2,$$

где N1 – Число вхождений операторов в коде;

N2 – Число вхождений операндов в коде.

Таким образом, на основе данных из таблицы 11, длина программы:

$$N=65+33=98.$$

Словарь программы рассчитывается по формуле:

$$n=n1+n2,$$

где n1 – количество уникальных операторов программы;

n2 – количество уникальных операндов программы.

Словарь программы на основе данных их таблицы 11:

$$n=14+15=29.$$

Вычислительная сложность программы находится по формуле:

$$V=N\log_2 n,$$

где N – длина программы;

n – словарь программы.

По результатам раннего расчета длины и словаря:

$$V=98*\log_2 29=476,084.$$

Сложность кода программы находится по формуле:

$$D = \frac{n1}{2} * \frac{N2}{n2},$$

где $n1$ – количество уникальных операторов программы;

$N2$ – число вхождений операндов в коде;

$n2$ – количество уникальных операндов программы.

Таким образом мы можем рассчитать:

$$D = \frac{14}{2} * \frac{33}{15} = 15.4.$$

Для оценки качества кода был проведен расчет показателя Maintainability index, показывающая относительную сложность поддержки кода. Показатель находится по формуле:

$$MI = 171 - 5.2 * \ln(V) - 0.23 * CC - 16.2 * \ln(LOC),$$

где V – сложность кода программы;

CC – Цикломатическая сложность кода;

LOC – Количество строк кода.

Используя ранее рассчитанную вычислительную сложность кода и исходные данные из таблицы 11:

$$MI = 171 - 5.2 * \ln(476.084) - 0.23 * 2 - 16.2 * \ln(32) = 82.334.$$

Сложность функции ur ($D = 15,4$) является умеренной. В коде сбалансировано количество операторов и переменных ($n1 = 14$, $n2 = 15$). Объем программы ($V = 476,084$) указывает на то, что метод легко прочесть и понять с первого раза, он не перегружен лишней логикой, а все переменные используются по назначению.

Проанализировав получившиеся результаты метрики, можно сделать вывод о соответствии кода критериям сложности, сопровождаемости и общего качества. Также код соответствует критериям модифицируемости, что делает его простым для изменения разработчиками.

Заключение

В рамках квалификационной работы была разработана система создания, хранения и развертывания снапшотов веб-приложений.

На этапе анализа требований было выполнено исследование предметной области и проведён сравнительный анализ существующих программных аналогов. На основе полученных данных сформирован полный перечень функциональных и нефункциональных требований, предъявляемых к разрабатываемому программному обеспечению.

В процессе проектирования системы была определена архитектурная концепция системы, послужившая основой для реализации серверной части. Кроме того, были построены концептуальная и логическая модели базы данных, а также сформированы минимальные и рекомендуемые требования к аппаратному обеспечению для стабильного функционирования системы.

Разработан серверный компонент системы, отвечающий за развертывание и управление снапшотами. В качестве языка программирования используемого для реализации компонента был выбран TypeScript с использованием фреймворка tRPC исполняемый в среде выполнения Bun. Архитектура решения представлена тремя независимыми сервисами, отвечающими за управление снапшотами, развертку приложения в изолированной Docker Compose среде и хранение дампа базы данных.

Проведено ручное функциональное тестирование методом «черного ящика» по разработанным тест-кейсам, а также осуществлен расчет метрики сложности программного кода Холстеда, подтвердивший высокое качество и сопровождаемость архитектуры.

Осуществлено календарное планирование и проведён функционально-стоимостной анализ процесса разработки. Общие затраты на реализацию проекта составили 151948 рублей при совокупных трудозатратах в размере 520 часов; весь комплекс работ был запланирован на период с 8 апреля по 13 июня текущего года.

Таким образом, в результате выполнения выпускной квалификационной работы разработана система создания, хранения и развертывания снапшотов веб-приложений. Результаты данной работы могут быть полезны аналитикам, для изучения изменений базы данных приложений в процессе их работы.

Список использованных источников

1. Инструменты развертывания // <https://appmaster.io/> [Электронный ресурс]. URL: <https://appmaster.io/ru/glossary/instrumenty-razvertyvaniia> (дата обращения: 10.04.2026).
2. Орлов С. А. Технологии разработки программного обеспечения / С. А. Орлов. – СПб. : Питер, 2020. – 624 с.
3. Используем Coolify для хостинга проектов на своём VPS // <https://habr.com/> [Электронный ресурс] URL: <https://habr.com/ru/companies/timeweb/articles/975384/> (Дата обращения: 10.05.2026).
4. Обзор на Easypanel // <https://ddpa.ru/> [Электронный ресурс]. URL: <https://ddpa.ru/p/easypanel> (дата обращения: 11.05.2026).
5. Фаулер М. Корпоративные шаблоны разработки приложений / М. Фаулер. – М. : Вильямс, 2018. – 544 с
6. Буч Г. Язык UML 2.0. Руководство пользователя / Г. Буч, Дж. Рамбо, А. Якобсон. – М. : ДМК-Пресс, 2021. – 432 с
7. Арлоу, Д. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование / Д. Арлоу – СПб.: Символ-Плюс, 2007. – 624 с.
8. Функциональные и нефункциональные требования: Полное руководство // <https://scand.com> [Электронный ресурс]. URL: <https://scand.com/ru/company/blog/functional-vs-non-functional-requirements/> (дата обращения: 19.05.2026).
9. The Sidecar Pattern: лучший спутник ваших микросервисов // <https://ru.linkedin.com> [Электронный ресурс]. URL: <https://ru.linkedin.com/pulse/sidecar-pattern-your-microservices-best-companion-vimal-prakash-dhkjc?tl=ru> (дата обращения: 20.05.2026).
10. Моделирование данных: основные понятия и методы. Что такое модель данных // <https://denvic.tech> [Электронный ресурс]. URL:

<https://denvic.tech/blog/ekspertnye-stati/modelirovanie-dannykh-osnovnye-ponyatiya-i-metody-chto-takoe-model-dannykh/> (дата обращения: 21.05.2026).

11. Моделирование и проектирование данных // <https://dataliteracy.ru> [Электронный ресурс]. URL: <https://dataliteracy.ru/dmbok-notes-5> (дата обращения: 23.05.2026).

12. Иванов, П.В. Управление проектами: Учебное пособие / П.В. Иванов. – Рн/Д: Феникс, 2018. – 102 с.

13. Как проджект-менеджеру использовать диаграмму Ганта, чтобы следить за прогрессом проекта // <https://mindbox.ru/> [Электронный ресурс] URL: <https://mindbox.ru/journal/education/diagramma-ganta/> (Дата обращения: 23.05.2026).

14. Kubernetes – изучаем паттерн Sidecar // <https://habr.com/> [Электронный ресурс] URL: <https://habr.com/ru/companies/nixys/articles/559368/> (Дата обращения: 23.05.2026).

15. Знакомьтесь, tRPC // <https://habr.com/> [Электронный ресурс] URL: <https://habr.com/ru/companies/timeweb/articles/723698/> (Дата обращения: 24.05.2026).

16. Обзор балансировщика traefik // <https://habr.com/> [Электронный ресурс] URL: <https://habr.com/ru/companies/otus/articles/797243/> (Дата обращения: 26.05.2026).

17. Логическая модель БД на практике: пример, ошибки, выводы // <https://quality-lab.ru/> [Электронный ресурс] URL: <https://quality-lab.ru/blog/kak-svyazat-dannye-logicheskaya-model-bd-na-primere/> (Дата обращения: 27.05.2026).

18. Репозиторий кода snapshots // <https://github.com/> [Электронный ресурс] URL: <https://github.com/GeoKD/snapshots> (Дата обращения: 28.05.2026).

19. Монорепо: typescript & workspaces npm. Настройка и публикация в npm // <https://habr.com/> [Электронный ресурс] URL: <https://habr.com/ru/articles/540114/> (Дата обращения: 28.05.2026).

20. Фундаментальная теория тестирования // <https://habr.com/>
[Электронный ресурс] URL: <https://habr.com/ru/articles/549054/> (Дата обращения:
28.05.2026).

21. Аверьянов А. В., Кошель И. Н., Кузнецов В. В. Применение метрик Холстеда для количественного оценивания характеристик программ ЭВМ // Изв. вузов. Приборостроение. 2019. Т. 62, № 11. С. 970-975.

Приложение А
(справочное)
Глоссарий

Снапшот – в контексте разрабатываемой системы, единица развертывания, копия версии приложения, разворачиваемая вместе с актуальным состоянием базы данных.

TypeScript – это язык программирования, разработанный Microsoft, который является расширенной версией JavaScript. Его главное отличие – статическая типизация, позволяющая находить большинство ошибок еще до запуска программы, делая код более надежным и удобным для крупных проектов

TRPC – фреймворк, обеспечивающий строгую типизацию API без необходимости генерации промежуточного кода или схем данных. Позволяет напрямую вызывать функции бэкенда в клиентском приложении. Основан на протоколе HTTP.

ORM (Object-Relational Mapping) – это технология программирования, которая связывает таблицы реляционной базы данных с объектами в коде. Она позволяет разработчику выполнять операции с данными (создание, чтение, обновление, удаление) через привычные методы и свойства классов, без использования SQL-запросов.

Drizzle ORM – TypeScript и JavaScript библиотека, которая позволяет общаться с базой данных (PostgreSQL, MySQL, SQLite) с помощью привычного кода на TypeScript вместо написания сырых SQL-запросов.

Docker-образ – Файл, содержащий исходный код приложения, зависимости и инструкции, необходимые для автономного запуска системы. Служит основой для развертывания контейнеров.

Приложение Б
(обязательное)
Листинг серверной части приложения

Листинг Б.1 – Файл config.ts

```
import z from 'zod';

export const env = z.object({
  DATABASE_URL: z.string(),
  SIDECAR_TRPC_URL: z.string(),
  FILES_URL_BASE: z.string(),
}).parse(Bun.env);
```

Листинг Б.2 – Файл api-clients.ts

```
import type { AppRouter as AppSidecarAppRouter } from '@app-
sidecar/trpc';
import { createTRPCClient, httpBatchLink } from '@trpc/client';
import * as config from './config';

export const appSidecar = createTRPCClient<AppSidecarAppRouter>({
  links: [
    httpBatchLink({
      url: config.env.SIDECAR_TRPC_URL,
    }),
  ],
});
```

Листинг Б.3 – Файл init.ts

```
import { initTRPC, TRPCError } from '@trpc/server';
import { drizzle } from 'drizzle-orm/node-postgres';

import * as config from '@control-backend/config';
import * as dbSchema from '@control-backend/db/schema';

const t = initTRPC.create({});

export const router = t.router;
export const createCallerFactory = t.createCallerFactory;
export const publicProcedure = t.procedure
  .use(t.middleware(async ({ ctx, next }) => {
    const db = drizzle(config.env.DATABASE_URL, { schema: dbSchema });

    return await db.transaction(async (tx) => {
      return await next({ ctx: { ...ctx, db: tx } });
    });
  }));

export function trpcErr(message: string): never {
  throw new TRPCError({ code: 'BAD_REQUEST', message });
}
```

Листинг Б.4 – Файл index.ts

```
import { router } from './init';
import snapshot from './snapshot';
```

```
export const appRouter = router({
  snapshot,
});

export type AppRouter = typeof appRouter;
```

Листинг Б.5 – Файл create.ts

```
import z from 'zod';
import * as apiClients from '@control-backend/api-clients';
import * as dbScheme from '@control-backend/db/schema';
import * as filestorage from '@control-backend/services/filestorage';
import * as utils from '@control-backend/utils';
import { publicProcedure, trpcErr } from '../init';

export default publicProcedure.input(z.object({
  slug: z.string('Обязательное поле')
    .min(3, 'Минимум 3 символа')
    .max(63, 'Максимум 63 символа')
    .regex(
      /^[a-z]([a-z0-9-]*[a-z0-9])?$/,
      'Только латинские буквы в нижнем регистре, цифры и дефис, начинается с латинской буквы, не заканчивается дефисом',
    ),
})).mutation(async ({ ctx, input }) => {
  const existing = await ctx.db.query.snapshots.findFirst({
    where: (snapshots, q) => q.eq(snapshots.slug, input.slug),
  });
  if (existing) {
    trpcErr(`Снапшот ${input.slug} уже существует`);
  }

  const dockerImagesVersion = await
    apiClients.appSidecar.getDockeImagesVersion.query();

  const postgresqlDumpFileContent = await
    apiClients.appSidecar.getPostgresqlDump.query();

  const postgresqlDumpFileParams = await
    filestorage.writeRandomFile(postgresqlDumpFileContent);

  const [postgresqlDumpFile] = await ctx.db.insert(dbScheme.files)
    .values(postgresqlDumpFileParams)
    .returning({ id: dbScheme.files.id });

  await ctx.db.insert(dbScheme.snapshots).values({
    slug: input.slug,
    dockerImagesVersion,
    postgresqlDumpFileId:
      utils.assert.defined(postgresqlDumpFile).id,
    isRunning: false,
```

```
});
});
```

Листинг Б.6 – Файл down.ts

```
import * as drizzle from 'drizzle-orm';
import z from 'zod';
import * as dbScheme from '@control-backend/db/schema';
import * as docker from '@control-backend/services/docker';
import { publicProcedure, trpcErr } from '../init';

export default publicProcedure.input(z.object({
  id: z.number(),
})).mutation(async ({ ctx, input }) => {
  const snapshot = await ctx.db.query.snapshots.findFirst({
    where: (snapshots, q) => q.and(
      q.eq(snapshots.id, input.id),
      q.eq(snapshots.isRunning, true),
    ),
  });
  if (!snapshot) {
    trpcErr(`Снапшот не найден`);
  }

  try {
    await docker.stopSnapshot({
      slug: snapshot.slug,
      imagesVersion: snapshot.dockerImagesVersion,
    });
  }
  catch (e) {
    trpcErr(`Не удалось остановить снапшот: ${e}`);
  }

  await ctx.db.update(dbScheme.snapshots)
    .set({ isRunning: false })
    .where(drizzle.eq(dbScheme.snapshots.id, input.id));
});
```

Листинг Б.7 – Файл list.ts

```
import { publicProcedure } from '../init';

export default publicProcedure.query(async ({ ctx }) => {
  return await ctx.db.query.snapshots.findMany({
    orderBy: (snapshots, q) => [q.desc(snapshots.isRunning),
      q.desc(snapshots.creationTime)],
  });
});
```

Листинг Б.8 – Файл remove.ts

```
import * as drizzle from 'drizzle-orm';
import z from 'zod';
import * as dbScheme from '@control-backend/db/schema';
```

```

import * as filestorage from '@control-backend/services/filestorage';
import { publicProcedure, trpcErr } from '../init';

export default publicProcedure.input(z.object({
  id: z.number(),
})).mutation(async ({ ctx, input }) => {
  const snapshot = await ctx.db.query.snapshots.findFirst({
    with: { postgresqlDumpFile: true },
    where: (snapshots, q) => q.and(
      q.eq(snapshots.id, input.id),
      q.eq(snapshots.isRunning, false),
    ),
  });
  if (!snapshot) {
    trpcErr(`Снапшот не найден`);
  }

  await filestorage.remove(snapshot.postgresqlDumpFile.urlPath);

  await
  ctx.db.delete(dbScheme.snapshots).where(drizzle.eq(dbScheme.snapshots.id, input.id));

  await
  ctx.db.delete(dbScheme.files).where(drizzle.eq(dbScheme.files.id, snapshot.postgresqlDumpFileId));
});

```

Листинг Б.9 – Файл reset-postgresql-state-from-dump.ts

```

import z from 'zod';
import * as snapshots from '@control-backend/services/snapshots';
import { publicProcedure, trpcErr } from '../init';

export default publicProcedure.input(z.object({
  id: z.number(),
})).mutation(async ({ ctx, input }) => {
  const snapshot = await ctx.db.query.snapshots.findFirst({
    with: { postgresqlDumpFile: true },
    where: (snapshots, q) => q.eq(snapshots.id, input.id),
  });
  if (!snapshot) {
    trpcErr(`Снапшот не найден`);
  }

  try {
    await snapshots.reloadPostgresqlState(snapshot);
  }
  catch (e) {
    trpcErr(`Не удалось остановить снапшот: ${e}`);
  }
});

```

Листинг Б.10 – Файл `up.ts`

```
import * as drizzle from 'drizzle-orm';
import z from 'zod';
import * as dbScheme from '@control-backend/db/schema';
import * as docker from '@control-backend/services/docker';
import * as snapshots from '@control-backend/services/snapshots';
import { publicProcedure, trpcErr } from '../init';

export default publicProcedure.input(z.object({
  id: z.number(),
})).mutation(async ({ ctx, input }) => {
  const snapshot = await ctx.db.query.snapshots.findFirst({
    with: { postgresqlDumpFile: true },
    where: (snapshots, q) => q.and(
      q.eq(snapshots.id, input.id),
      q.eq(snapshots.isRunning, false),
    ),
  });
  if (!snapshot) {
    trpcErr(`Снапшот не найден`);
  }

  const snapshotCmdParams = {
    slug: snapshot.slug,
    imagesVersion: snapshot.dockerImagesVersion,
  };

  try {
    await docker.startSnapshot(snapshotCmdParams);

    await docker.waitSnapshotPostgresqlReady(snapshotCmdParams);

    await snapshots.reloadPostgresqlState(snapshot);
  } catch (e) {
    await docker.stopSnapshot(snapshotCmdParams);
    trpcErr(`Не удалось запустить снапшот: ${e}`);
  }

  await ctx.db.update(dbScheme.snapshots)
    .set({ isRunning: true })
    .where(drizzle.eq(dbScheme.snapshots.id, input.id));
});
```

Листинг Б.11 – Файл `docker.ts`

```
import { $ } from 'bun';

// https://www.totaltypescript.com/concepts/the-prettify-helper
type Prettify<T> = {
  [K in keyof T]: T[K];
} & {};
```

```

function makeSnapshotCmd<TParams extends object>(cmdCb: (dcCmd:
string, params: TParams) => string) {
  return (params: Prettify<{ slug: string, imagesVersion: string }
& TParams>) =>
    `${sh -c "${cmdCb(`docker compose -p 'app-snapshot-
${params.slug}'`, params)}"`.env({
      ...Bun.env,
      SNAPSHOT_SLUG: params.slug,
      IMAGES_VERSION: params.imagesVersion,
    })};
}

export const startSnapshot = makeSnapshotCmd(dc => `${dc} up -d`);

export const stopSnapshot = makeSnapshotCmd(dc => `${dc} rm -sf`);

export const waitSnapshotPostgresqlReady = makeSnapshotCmd(
  dc => `${dc} exec -T db sh -c \'until pg_isready -U postgres; do
sleep 1; done\'`,
);

export const dropSnapshotPostgresqlDb = makeSnapshotCmd(
  dc => `${dc} exec -T db dropdb items -f -U postgres`,
);

export const createSnapshotPostgresqlDb = makeSnapshotCmd(
  dc => `${dc} exec -T db createdb items -U postgres`,
);

export const loadSnapshotPostgresqlStateFromDump =
makeSnapshotCmd((dc, { dumpUrl }: { dumpUrl: string }) =>
  `curl ${dumpUrl} | ${dc} exec -T db psql -d items -U postgres`,
);

export const executeSnapshotPostgresqlPostStateLoadScript =
makeSnapshotCmd(dc =>
  `echo 'UPDATE "transaction" SET is_confirmed = false' | ${dc}
exec -T db psql -d items -U postgres`,
);

```

Листинг Б.12 – Файл filestorage.ts

```

import path from 'node:path';
import * as config from '@control-backend/config';

const URL_ROOT = '/files';
const PATH_ROOT = resolveFromUrlUnsafe(URL_ROOT);

function resolveFromUrlUnsafe(url: string) {
  return path.resolve(__dirname, `../data${url}`);
}

export function resolveFromUrl(url: string) {
  const resolved = resolveFromUrlUnsafe(url);

```

```

    if (!resolved.startsWith(PATH_ROOT + path.sep)) {
      return null;
    }

    return resolved;
  }

export async function writeRandomFile(data: Blob |
NodeJS.TypedArray | ArrayBufferLike | string) {
  const urlPath = `${URL_ROOT}/${Bun.randomUUIDv7()}`;

  const path = resolveFromUrlUnsafe(urlPath);

  const size = await Bun.write(path, data);

  return { urlPath, size };
}

export function getFullUrl(urlPath: string) {
  return new URL(urlPath, config.env.FILES_URL_BASE).href;
}

export async function remove(urlPath: string) {
  const resolved = resolveFromUrl(urlPath);
  if (!resolved) {
    return;
  }
}

```

Листинг Б.13 – Файл schema.ts

```

const defaultFields = {
  id: integer().primaryKey().generatedAlwaysAsIdentity(),
  createTime:
doublePrecision().notNull().default(drizzle.sql.raw('extract(epoch
from now()) * 1000')),
};

export const files = pgTable('files', {
  ...defaultFields,
  urlPath: text().notNull(),
  size: integer().notNull(),
});

export const snapshots = pgTable('snapshots', {
  ...defaultFields,
  postgresqlDumpFileId: integer().notNull().references(() =>
files.id),
  slug: text().notNull().unique(),
  dockerImagesVersion: text().notNull(),
  isRunning: boolean().notNull(),
});

export const snapshotsRelations = relations(snapshots, ({ one })

```

```
=> ({
  postgresqlDumpFile: one(files, {
    fields: [snapshots.postgresqlDumpFileId],
    references: [files.id],
  }),
}));
```

Листинг Б.14 – Файл files.ts

```
import * as filestorage from '@control-
backend/services/filestorage';

const server = Bun.serve({
  hostname: '0.0.0.0',
  port: 3001,
  async fetch(req) {
    if (req.method !== 'GET') {
      return new Response('Method Not Allowed', { status: 405 });
    }

    const resolved = filestorage.resolveFromUrl(new
URL(req.url).pathname);
    if (!resolved) {
      return new Response('Forbidden', { status: 403 });
    }

    const file = Bun.file(resolved);
    if (!(await file.exists())) {
      return new Response('Not Found', { status: 404 });
    }

    return new Response(file);
  },
});

// eslint-disable-next-line no-console
console.log(`Files server running at ${server.url}`);
```

Листинг Б.15 – Файл trpc.ts

```
import { createBunServeHandler } from 'trpc-bun-adapter';
import { appRouter } from '@control-backend/trpc';

const server = Bun.serve(createBunServeHandler({
  router: appRouter,
  onError: console.error,
}), {
  hostname: '0.0.0.0',
}));
```

Приложение В
(обязательное)
Листинг Sidcar-приложения

Листинг В.1 – Файл config.ts

```
import z from 'zod';

export const env = z.object({
  APP_POSTGRES_HOST: z.string(),
  APP_POSTGRES_PORT: z.string(),
  APP_DOCKER_IMAGES_VERSION: z.string(),
}).parse(Bun.env);
```

Листинг В.2 – Файл index.ts

```
import * as config from '../config';
import getPostgresqlDump from './get-postgresql-dump';
import { publicProcedure, router } from './init';

export const appRouter = router({
  getDockerImagesVersion: publicProcedure.query(() =>
    config.env.APP_DOCKER_IMAGES_VERSION),
  getPostgresqlDump,
});

export type AppRouter = typeof appRouter;
```

Листинг В.3 – Файл init.ts

```
import { initTRPC, TRPCError } from '@trpc/server';

const t = initTRPC.create({});

export const router = t.router;
export const createCallerFactory = t.createCallerFactory;
export const publicProcedure = t.procedure;

export function trpcErr(message: string): never {
  throw new TRPCError({ code: 'BAD_REQUEST', message });
}
```

Листинг В.4 – Файл get-postgresql-dump.ts

```
import { $ } from 'bun';
import * as config from '@app-sidecar/config';
import { publicProcedure, trpcErr } from './init';

export default publicProcedure.query(async () => {
  try {
    return await `$pg_dump -h ${config.env.APP_POSTGRES_HOST} -p
${config.env.APP_POSTGRES_PORT} -U postgres -d items`.text();
  }
  catch (e) {
    trpcErr(`Не удалось загрузить дамп: ${e}`);
  }
});
```

Листинг В.5 – Файл trpc.ts

```
import { createBunServeHandler } from 'trpc-bun-adapter';
import { appRouter } from '@app-sidecar/trpc';

const server = Bun.serve(createBunServeHandler({
  router: appRouter,
  onError: console.error,
}), {
  port: 3002,
});
```

Приложение Г
(обязательное)
Листинг клиентского приложения

Листинг Г.1 – Файл main.ts

```
app.use(VueQueryPlugin);

app.mount('#app');
```

Листинг Г.2 – Файл App.vue

```
<script setup lang="ts">
import { usePreferredDark } from '@vueuse/core';
import { darkTheme, lightTheme, NConfigProvider, NDialogProvider,
NModalProvider, NNotificationProvider } from 'naive-ui';
import SnapshotsPage from './pages/snapshots-page.vue';

const isSystemDark = usePreferredDark();
</script>

<template>
  <NConfigProvider :theme="isSystemDark ? darkTheme : lightTheme">
    <NDialogProvider>
      <NModalProvider>
        <NNotificationProvider>
          <SnapshotsPage />
        </NNotificationProvider>
      </NModalProvider>
    </NDialogProvider>
  </NConfigProvider>
</template>
```

Листинг Г.3 – Файл page.vue

```
<script setup lang="ts">
import { NFlex, NLayout, NLayoutContent, NLayoutHeader } from
'naive-ui';

defineProps<{
  title: string
}>();
</script>

<template>
  <div>
    <NLayoutContent position="absolute" style="top: 64px;"
content-style="padding: 16px 0 320px;">
      <NFlex vertical align="center">
        <div style="width: 768px">
          <slot />
        </div>
      </NFlex>
    </NLayoutContent>
  </NLayout>
</div>
</template>
```

Листинг Г.4 – Файл loading-overlay.vue

```
<script setup lang="ts">
import { NSpin } from 'naive-ui';
import { onBeforeUnmount } from 'vue';

function handleKeyDown(event: KeyboardEvent) {
  event.stopPropagation();
  event.preventDefault();
}

document.addEventListener('keydown', handleKeyDown);

onBeforeUnmount(() => {
  document.removeEventListener('keydown', handleKeyDown);
});
</script>

<template>
  <Teleport to="body">
    <div
      style="position: absolute; width: 100vw; height: 100vh; top:
0; left: 0; display: flex; justify-content: center; align-items:
center; background-color: rgba(0, 0, 0, 0.5); z-index: 1000000;"
    >
      <NSpin size="large" />
    </div>
  </Teleport>
</template>
```

Листинг Г.5 – Файл trpc.ts

```
import type { AppRouter } from '@control-backend/trpc';
import { createTRPCClient, httpBatchLink } from '@trpc/client';

export default createTRPCClient<AppRouter>({
  links: [
    httpBatchLink({
      url: '/trpc',
    }),
  ],
});
```

Листинг Г.6 – Файл snapshots-page.vue

```
<script setup lang="ts">
import * as snapshot from '@frontend/entities/snapshot';
import * as shared from '@frontend/shared';
</script>

<template>
  <shared.ui.Page title="Снапшоты">
    <template #extra>
      <snapshot.ui.CreateButton />
    </template>
  </shared.ui.Page>
</template>
```

```

    </template>

    <snapshot.ui.List />
  </shared.ui.Page>
</template>

```

Листинг Г.7 – Файл list.vue

```

<script setup lang="ts">
import { useMutation, useQuery, useQueryClient } from
  '@tanstack/vue-query';
import { OpenOutline as OpenOutlineIcon, PlayOutline,
  ReloadOutline, StopOutline, TrashOutline } from
  '@vicons/ionicons5';
import { useBrowserLocation } from '@vueuse/core';
import { NButton, NFlex, NIcon, NList, NListItem, NPopover, NSpin,
  NThing, useDialog, useNotification } from 'naive-ui';
import * as shared from '@/frontend/shared';
import * as model from '../model';

const dialog = useDialog();
const notification = useNotification();
const location = useBrowserLocation();
const queryClient = useQueryClient();

const cache = queryClient.getQueryCache();
cache.clear();

const { data: snapshots, isPending: isSnapshotsListPending } =
useQuery({
  queryKey: model.SNAPSHOTS_LIST_QUERY_KEY,
  queryFn: () => shared.api.trpc.snapshot.list.query(),
});

const { mutate: remove, isPending: isRemovePending } =
useMutation({
  mutationFn: (input: Parameters<typeof
shared.api.trpc.snapshot.remove.mutate>[0]) =>
shared.api.trpc.snapshot.remove.mutate(input),
  onSuccess: () =>
model.invalidateSnapshotsListQueryKey(queryClient),
  onError: () => {
    notification.error({
      content: 'Произошла ошибка при удалении снапшота',
    });
  },
});

const { mutate: up, isPending: isUpPending } = useMutation({
  mutationFn: (input: Parameters<typeof
shared.api.trpc.snapshot.up.mutate>[0]) =>
shared.api.trpc.snapshot.up.mutate(input),
  onSuccess: () =>
model.invalidateSnapshotsListQueryKey(queryClient),
});

```

```

    onError: () => {
      notification.error({
        content: 'Произошла ошибка при запуске снапшота',
      });
    },
  });

const { mutate: down, isPending: isDownPending } = useMutation({
  mutationFn: (input: Parameters<typeof
shared.api.trpc.snapshot.down.mutate>[0]) =>
shared.api.trpc.snapshot.down.mutate(input),
  onSuccess: () =>
model.invalidateSnapshotsListQueryKey(queryClient),
  onError: () => {
    notification.error({
      content: 'Произошла ошибка при остановке снапшота',
    });
  },
});

const { mutate: resetPostgresqlStateFromDump, isPending:
isResetPostgresqlStateFromDumpPending } = useMutation({
  mutationFn: (input: Parameters<typeof
shared.api.trpc.snapshot.resetPostgresqlStateFromDump.mutate>[0])
=>

shared.api.trpc.snapshot.resetPostgresqlStateFromDump.mutate(input
),
  onError: () => {
    notification.error({
      content: 'Произошла ошибка при сбросе состояния PostgreSQL',
    });
  },
});

const ACTIONS = [
  {
    label: 'Запустить',
    type: 'default',
    icon: PlayOutline,
    isForRunningSnapshot: false,
    handler: up,
  },
  {
    label: 'Сбросить состояние PostgreSQL',
    type: 'default',
    icon: ReloadOutline,
    isForRunningSnapshot: true,
    handler: (snapshot: NonNullable<typeof
snapshots.value>[number]) => dialog.warning({
      title: 'Сброс состояния PostgreSQL',
      content: 'Вы действительно хотите сбросить состояние
PostgreSQL? Это действие необратимо.',

```

```

        positiveText: 'Сбросить',
        negativeText: 'Отмена',
        onPositiveClick: () => {
            resetPostgresqlStateFromDump(snapshot);
        },
    )),
},
{
    label: 'Остановить',
    type: 'error',
    icon: StopOutline,
    isForRunningSnapshot: true,
    handler: down,
},
{
    label: 'Удалить',
    type: 'error',
    icon: TrashOutline,
    isForRunningSnapshot: false,
    handler: (snapshot: NonNullable<typeof
snapshots.value>[number]) => dialog.warning({
        title: 'Удаление снапшота',
        content: 'Вы действительно хотите удалить снапшот? Это
действие необратимо.',
        positiveText: 'Удалить',
        negativeText: 'Отмена',
        onPositiveClick: () => {
            remove(snapshot);
        },
    )),
},
] as const;
</script>

<template>
    <NFlex v-if="!snapshots?.length" vertical align="center">
        <NSpin v-if="isSnapshotsListPending" size="large" />

        <h2 v-else>
            Данные отсутствуют
        </h2>
    </NFlex>

    <NList v-else>
        <NListItem
            v-for="snapshot in snapshots"
            :key="snapshot.id"
            style="padding: 12px 16px;"
        >
            <NThing
                :title="snapshot.slug"
                :description="new
Date(snapshot.creationTime).toLocaleString()"

```

```

>
  <template #header-extra>
    <NFlex>
      <a
        v-if="snapshot.isRunning"

:href="\${location.protocol}://${snapshot.slug}.${location.host}"
        target="_blank"
      >
        <NButton>
          <NIcon>
            <OpenOutlineIcon />
          </NIcon>
        </NButton>
      </a>

    <template
      v-for="action in ACTIONS.filter((a) =>
a.isForRunningSnapshot === snapshot.isRunning)"
      :key="action.label"
    >
      <NPopover>
        <template #trigger>
          <NButton
            :type="action.type"
            @click="action.handler(snapshot)"
          >
            <NIcon>
              <component :is="action.icon" />
            </NIcon>
          </NButton>
        </template>

        {{ action.label }}
      </NPopover>
    </template>
  </NFlex>
</template>
</NThing>
</NListItem>
</NList>

<shared.ui.LoadingOverlay v-if="isRemovePending || isUpPending
|| isDownPending || isResetPostgresqlStateFromDumpPending" />
</template>

```

Листинг Г.8 – Файл create-form.vue

```

<script setup lang="ts">
import { useMutation, useQueryClient } from '@tanstack/vue-query';
import { NAlert, NButton, NCard, NFlex, NInput } from 'naive-ui';
import { ref } from 'vue';
import z from 'zod';
import * as shared from '@/frontend/shared';

```

```

import * as model from '../model';

const emit = defineEmits<{
  created: []
}>();

const SLUG_SCHEMA = z
  .string('Обязательное поле')
  .min(3, 'Минимум 3 символа')
  .max(63, 'Максимум 63 символа')
  .regex(
    /^[a-z]([a-z0-9-]*[a-z0-9])?$/,
    'Только латинские буквы в нижнем регистре, цифры и дефис, начинается с латинской буквы, не заканчивается дефисом',
  );

const queryClient = useQueryClient();

const { mutate: create, isPending: isCreating, error: createError, reset: resetCreate } = useMutation({
  mutationFn: (input: Parameters<typeof shared.api.trpc.snapshot.create.mutate>[0]) => shared.api.trpc.snapshot.create.mutate(input),
  onSuccess: () => {
    model.invalidateSnapshotsListQueryKey(queryClient);

    emit('created');
  },
});

const slug = ref('');

const slugValidationErrorMessage = ref<string>();

function validateSlug(value: string) {
  const { success, error } = SLUG_SCHEMA.safeParse(value);
  slugValidationErrorMessage.value = error?.issues[0]?.message;
  return success;
}
</script>

<template>
  <NCard size="huge" :bordered="false">
    <NFlex vertical>
      <NInput
        v-model:value="slug"
        placeholder="Slug"
        @update:value="(value) => {
          resetCreate();

          if (slugValidationErrorMessage !== undefined) {
            validateSlug(value);
          }
        }
      />
    </NFlex>
  </NCard>
</template>

```

```

        }"
      />
      <NAlert
        v-if="createError || slugValidationErrorMessage"
        :title="createError?.message ??
slugValidationErrorMessage"
        type="error"
      />
    </NFlex>

    <template #footer>
      <NFlex justify="end">
        <NButton
          type="primary"
          @click="() => {
            if (validateSlug(slug)) {
              create({ slug });
            }
          }"
        >
          Создать
        </NButton>
      </NFlex>
    </template>
  </NCard>

  <shared.ui.LoadingOverlay v-if="isCreating" />
</template>

```

Листинг Г.9 – Файл create-button.vue

```

<script setup lang="ts">
import { NButton, NCard, NModal } from 'naive-ui';
import { ref } from 'vue';
import CreateForm from './create-form.vue';

const isShowCreationModal = ref(false);
</script>

<template>
  <NButton
    type="primary"
    style="width: 96px;"
    @click="isShowCreationModal = true"
  >
    Создать
  </NButton>

  <NModal
    v-model:show="isShowCreationModal"
    style="width: 540px;"
  >
    <NCard
      title="Создание снапшота"
    >

```

```
        :bordered="false"  
    >  
        <CreateForm @created="() => isShowCreationModal = false" />  
    </NCard>  
</NModal>  
</template>
```