**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ**

**И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

(СПбГУТ)

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ (**ИТПИ**)

КАФЕДРА ПРОГРАММНОЙ ИНЖЕНЕРИИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ (ПИ И ВТ)

**Проект**

по дисциплине «Операционные системы и сети»

Тема «Сервис для операционной системы Linux»

Выполнили:

Студенты группы ИКПИ-23

Даненко Д. А.

Харлова А. А.

Стебеньков А. А.

Подпись \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Принял:

Преподаватель

Дагаев А. В.

Подпись \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Санкт-Петербург

2024 г.

# Оглавление

[Оглавление 1](#_Toc13157)

[Введение 1](#_Toc17572)

[Постановка задачи 1](#_Toc31183)

[Теория администрирования операционной системы Linux 1](#_Toc13577)

[Разработка проекта](#_Toc29398)[Варианты использования сервиса 11](#_Toc29398)

[Описание внутренней логики 14](#_Toc11246)

[Пример работы программы 18](#_Toc9822)

[Графическое представление интерфейса программы 20](#_Toc3673)

[Схема алгоритма работы 22](#_Toc14137)

[Пути дальнейшего развития проекта 23](#_Toc9745)

[Заключение 28](#_Toc14670)

# **Введение**

В современных операционных системах управление пользователями и системными ресурсами является важной задачей для обеспечения безопасности и стабильности работы. Автоматизация таких процессов позволяет администраторам эффективно управлять системой, минимизировать человеческий фактор и упростить выполнение повседневных задач. В данной работе представлен проект сервиса, который позволяет автоматизировать управление пользователями и очистку системных ресурсов.

# **Постановка задачи**

В рамках данного проекта необходимо разработать сервис для управления пользователями и системными ресурсами в операционной системе *Linux*. Сервис должен предоставлять следующие функции:

* Добавление и удаление пользователей.
* Изменение прав доступа пользователей.
* Очистка кэша пользователей.
* Остановка приложений, использующих большой объём трафика.

Сервис будет реализован на языке *Bash* и интегрирован с системой инициализации *systemd* в *Linux*.

# **Теория администрирования операционной системы Linux**

*Основы изучения написания скриптов на языке программирования Bash. Администрирование Linux.*Bash (Bourne Again SHell) — это командная оболочка и язык сценариев, используемый в Unix-подобных системах. Скрипты на Bash позволяют автоматизировать задачи, управлять системными процессами и выполнять команды.  
  
Перед началом использования данного инструмента программирования, необходимо ознакомится с логикой и основными элементами данного языка.  
  
1.1 Структура скрипта.  
  
- Скрипт начинается с шебанга: #!/bin/bash.  
  
Шебанг в Linux — это строка, которая указывает операционной системе, какой интерпретатор нужно использовать для выполнения скрипта.  
  
Общий синтаксис шебанга: #!<interpreter> [arguments]. В этой конструкции: #! — сам шебанг, с которого начинается директива; <interpreter> — путь к интерпретатору, который должен быть использован для выполнения скрипта.  
  
-Комментарии начинаются с #.  
  
1.2Переменные  
  
-Объявление:

variable\_name=value.  
  
Переменные в Bash объявляются без использования ключевых слов, просто присваивая переменной значение. Например: name="John" age=30

Некоторые правила именования переменных в Bash:

* Имя переменной должно начинаться с буквы латинского алфавита (нижнего или верхнего регистра) или символа подчёркивания \_.
* Оно может включать буквы, цифры и символ подчёркивания, но не может содержать пробелы или специальные символы.
* Регистр букв в именах переменных имеет значение. Например, переменные var1 и Var1 будут рассматриваться как разные.
* Не рекомендуется использовать зарезервированные слова Bash в качестве имён переменных.

Символ $, поставленный перед именем переменной, указывает на то, что нужно использовать именно значение переменной, а не одноимённую строку.   
  
- Использование: ${variable\_name} или $variable\_name.

1.3 Условия.  
  
- Использование конструкций if, else, elif.  
  
Конструкция if в языке Bash — это условная конструкция, которая позволяет выполнять определённые команды или действия в зависимости от истинности заданного условия.

Синтаксис конструкции if в Bash выглядит следующим образом:  
single if : if [условие] then команда fi

\*Пробелы являются частью синтаксиса и не должны быть удалены.

Ключевое слово fi в конце конструкции if всегда является последним ключевым словом строки или его нужно завершить точкой с запятой (;) перед использованием других ключевых слов.  
  
Синтаксис конструкции else позволяет определить команды, которые выполняются, если условие ложно.  
  
Шаблон написания синтаксиса:  
  
if [ условие ]; then

# команды, выполняемые, если условие истинно

else

# команды, выполняемые, если условие ложно

fi  
  
Синтаксис конструкции elif позволяет проверять дополнительные условия и определять команды, которые выполняются при истинности этих условий.  
  
  
  
  
Шаблон написания синтаксиса:  
  
if [ условие1 ]; then

# Код, который должен быть выполнен, если условие1 равно true

elif [ условие2 ]; then

# Код, который должен быть выполнен, если условие2 равно true

else

# Код, который должен быть выполнен, если все условия равны false

fi  
  
1.4 Циклы  
  
- Использование циклов for,while,until.  
  
Цикл for в языке Bash позволяет перебрать все элементы из массива или использовать переменную-счётчик для определения количества повторений.  
  
Синтаксис цикла for для перебора элементов списка:  
  
for переменная in список do команда1 команда2 done  
  
Цикл while в языке Bash повторяется до тех пор, пока выполняется условие, указанное в объявлении цикла.  
  
Синтаксис цикла while:  
  
while [ условие ] do команда1 команда2 команда3 done.  
  
Цикл until в языке Bash используется для выполнения заданного набора команд до тех пор, пока данное условие оценивается как ложное.  
  
Синтаксис цикла until:  
  
until [CONDITION]

do

[COMMANDS]

done

1.5. Функции.  
  
Функции в Bash — это блоки кода, которые могут быть вызваны по имени, позволяя упрощать и структурировать скрипты. Они помогают избегать повторения кода и улучшают его читаемость.  
  
Функцию можно определить несколькими способами:

a. Стандартный способ:  
  
function\_name() {

# код функции

}  
  
b. С использованием ключевого слова function:  
  
function function\_name {

# код функции

}  
  
Вызов функции:  
  
function\_name  
  
Функции могут принимать аргументы, которые доступны внутри функции через специальные переменные:

- $1, $2, ... — позиционные параметры.

- $# — количество переданных аргументов.

- $@ — все аргументы.  
  
Пример:

greet() {

echo "Hello, $1"

}

greet "Project" # Вывод: Hello, Project  
  
  
Bash не поддерживает возвращение значений как в других языках. Вместо этого используется return для возврата статуса завершения (0 — успех, 1-255 — ошибка):  
  
my\_function() {

return 42

}

my\_function

echo $? # Вывод: 42  
  
По умолчанию все переменные в функции имеют глобальную область видимости. Чтобы создать локальные переменные, используйте local:  
  
my\_function() {

local local\_var="I'm local"

echo $local\_var

}

my\_function # Вывод: I'm local

echo $local\_var # Не выведет ничего  
  
Полный пример функции, принимающей аргументы и возвращающей статус:

calculate\_square() {

local num=$1

echo $((num \* num))

}

result=$(calculate\_square 5)

echo "Square: $result" # Вывод: Square: 25  
  
Функции в Bash - инструмент, позволяющий создавать структурированные и переиспользуемые скрипты. Они облегчают обработку данных и автоматизацию задач.  
  
1.6 Обработка аргументов  
  
В Bash обработка аргументов является ключевой частью скриптов. Аргументы могут быть переданы скрипту через командную строку, и их можно обрабатывать для выполнения различных операций.  
  
Аргументы передаются в скрипт или функцию и доступны через специальные переменные:

- $0 — имя скрипта.

- $1, $2, ... — переданные аргументы.

- $# — количество переданных аргументов.

- $@ — все аргументы как отдельные слова.

- $\* — все аргументы как одна строка.  
  
Пример:

#!/bin/bash

echo "Имя скрипта: $0"

echo "Первый аргумент: $1"

echo "Всего аргументов: $#"  
  
Можно проверять количество аргументов и их значения с помощью условий:

if [ $# -lt 2 ]; then

echo "Нужно передать как минимум два аргумента."

exit 1

fi  
  
Аргументы можно присваивать переменным для более удобного использования:

first\_arg=$1

second\_arg=$2  
  
Для более сложного сценария обработки аргументов можно использовать getopts:

while getopts "a:b:c" opt; do

case $opt in

a) echo "Опция A: $OPTARG" ;;

b) echo "Опция B: $OPTARG" ;;

c) echo "Опция C" ;;

\*) echo "Неверная опция" ;;

esac

done  
  
Getopts — это встроенная команда в языке Bash, которая анализирует опции и аргументы команды, которые были переданы в скрипт bash.  
  
Как работает getopts: каждый раз, когда она запускается, она ищет одну из опций, определённых в строке опций (optstring). Если опция найдена, то буква опции помещается в переменную с именем optname. Если опция не соответствует чему-либо, что определено в optstring, то getopts установит переменную optname в символ знака вопроса (?). Если у опции ожидается аргумент, то getopts берёт этот аргумент и помещает его в переменную $OPTARG.  
  
Пример простого скрипта, который принимает два числа и производит над ними окружение:

#!/bin/bash

if [ $# -ne 2 ]; then

echo "Использование: $0 <число1> <число2>"

exit 1

fi

sum=$(( $1 + $2 ))

echo "Сумма: $sum"  
  
1.7 Ввод/вывод  
  
- Чтение ввода: read variable.  
  
В языке bash данные делятся на три потока:

1. Стандартный ввод (stdin). В него поступают данные, которые вводятся с клавиатуры.
2. Стандартный вывод (stdout). По умолчанию выводится на экран монитора.
3. Стандартная ошибка (stderr). Записывает ошибки, возникающие в ходе исполнения программы.

По умолчанию данные выводятся на экран и считываются с клавиатуры, так как стандартные потоки ассоциированы с терминалом пользователя. Однако такое поведение можно изменить, перенаправив потоки ввода и вывода.

Некоторые операторы перенаправления в bash:

< file — использовать файл как источник данных для стандартного потока ввода;

> file — направить стандартный поток вывода в файл. Если файл не существует, он будет создан, если существует — перезаписан сверху;

2> file — направить стандартный поток ошибок в файл. Если файл не существует, он будет создан, если существует — перезаписан сверху;

>>file  — направить стандартный поток вывода в файл. Если файл не существует, он будет создан, если существует — данные будут дописаны к нему в конец;

&>file или >&file — направить стандартный поток вывода и стандартный поток ошибок в файл. Другая форма записи: >file 2>&1  
  
Вывод в языке Bash осуществляется через стандартный поток вывода оболочки (STDOUT). По умолчанию это — экран, и большинство bash-команд выводят данные в STDOUT, что приводит к их появлению в консоли.

Ещё одна возможность bash-скриптов — извлекать информацию из вывода команд и назначать её переменным. Для этого можно использовать значок обратного апострофа «» или конструкцию $().  
  
Bash-скрипты – мощный инструмент для автоматизации и управления системами. Они просты в написании и позволяют выполнять сложные задачи с минимальными усилиями.  
  
2.1 Администрирование Linux.  
  
Администрирование Linux с помощью языка Bash подразумевает использование командной строки и написание скриптов для управления системой. Bash предоставляет мощные инструменты для выполнения задач администрирования, таких как управление пользователями, файловыми системами и сетью.  
  
- Управление пользователями.  
  
Создание, удаление и модификация пользователей осуществляется с командами:

1. Создание пользователя:

sudo adduser имя\_пользователя

1. Удаление пользователя:

sudo deluser имя\_пользователя

1. Изменение пароля:

sudo passwd имя\_пользователя  
  
- Управление пакетами.  
  
Для установки, удаления и обновления программного обеспечения используются пакетные менеджеры, например:

1. APT для Debian/Ubuntu:

sudo apt update  
sudo apt install имя\_пакета

1. YUM для CentOS/RHEL:

sudo yum install имя\_пакета

- Работа с файлами и директориями.  
  
Управление файлами осуществляется через команды:

1. Копирование:

cp файл\_источник файл\_назначение

1. Перемещение:   
   mv файл\_источник файл\_назначение
2. Удаление:

rm файл  
  
- Мониторинг системы.  
  
Для мониторинга ресурсов системы можно использовать:  
  
a. Память:  
  
 free -h  
  
b. Загрузка процессора:  
  
 top  
  
c. Журнал системных сообщений:  
  
 tail -f /var/log/syslog  
  
- Написание скриптов для автоматизации.  
  
Скрипты помогают автоматизировать рутинные задачи. Пример простого скрипта для резервного копирования:  
  
#!/bin/bash

tar -czvf backup\_$(date +%F).tar.gz /путь/к/директории  
  
**-** Некоторые часто используемые команды Bash:

cd — изменение директории на другую;

ls — отображение содержимого в текущей директории;

mkdir — создание новой директории;

touch — создание нового файла;

rm — удаление файла или директории;

cp — копирование файла или директории;

mv — перемещение или переименование файла или директории;

echo — печать текста в терминал;

cat — чтение данных из файла и вывод их на экран;

grep — поиск в файлах;

chmod — изменение прав доступа к файлу или директории;

sudo — запуск команд с правами администратора.

Администрирование Linux через Bash — это эффективный способ управления системой. Знание базовых команд и написание скриптов позволяет автоматизировать задачи и улучшать управление сервером.

# Разработка проектаВарианты использования сервиса

По итогам разработки данного проекта, мы получаем сервис который может быть использован в нескольких направлениях:  
  
1.1 Управление пользователями на сервере: Сервис отлично подходит для администрирования пользователей в многопользовательских средах, таких как серверы Linux.  
  
- Созданный проект предоставляет интерфейс для управления учетными записями. С его помощью можно легко добавлять новых пользователей, удалять неактивных или изменять их характеристики, такие как пароли или права доступа, используя заранее определенные команды.  
  
1.2 Автоматизация рутинных задач: Позволяет автоматизировать создание, удаление и изменение прав пользователей без необходимости вручную вводить команды.  
  
- С помощью скриптов можно автоматизировать процессы создания и удаления учетных записей, а также изменения настроек. Это минимизирует время и усилия администратора, позволяя выполнять массовые операции за один раз.  
  
1.3 Мониторинг приложений: Функция остановки приложений по трафику полезна для управления ресурсами сервера и предотвращения перегрузок.  
  
- Сервис может отслеживать использование ресурсов приложениями и останавливать их при превышении определенных порогов трафика. Это помогает предотвратить перегрузку системы и поддерживать её стабильную работу.  
  
1.4 Очистка кэша: Полезно для поддержания чистоты профилей пользователей и освобождения места на диске.  
  
- Регулярная очистка профилей пользователей от устаревших кэшей позволяет экономить дисковое пространство. Сервис может быть настроен на автоматическое выполнение этих задач по расписанию.  
  
1.5 Управление правами доступа: Упрощает управление пользовательскими группами и правами, что особенно важно в больших организациях.  
  
- Инструменты созданного проекта позволяют создавать группы пользователей с определенными правами доступа. Это упрощает процесс управления и делает его более эффективным, обеспечивая безопасность данных и ресурсов.  
  
По итогам разработки данный проект может быть полезен как в малом, так и в крупном бизнесе, а также в учебных заведениях для управления компьютерными лабораториями.  
  
2.1 Варианты использования проекта для пользователей и администратора.  
  
Примеры использования готового проекта для пользователей:  
  
- Регистрация и вход: Пользователи могут создавать учетные записи и входить в систему через защищенные формы, что обеспечивает защиту личной информации.

- Персональный кабинет: Это интерфейс, где пользователи могут управлять своими настройками, изменять пароли и просматривать историю действий.

- Поддержка: Доступ к службе поддержки через различные каналы, включая FAQ, чаты или электронную почту. Это помогает быстро решать возникающие вопросы.  
  
  
- Настройки уведомлений: Пользователи могут настроить уведомления о важных событиях, таких как изменения пароля или обновления системы.  
  
Примеры использования готового проекта для администратора:  
  
Варианты для администратора

- Управление пользователями: Администратор имеет возможность добавлять, удалять и редактировать учетные записи пользователей, а также управлять правами доступа.

- Мониторинг активности: Сервис предоставляет инструменты для отслеживания действий пользователей и использования ресурсов, что позволяет выявлять возможные мошеннические действия.

- Автоматизация процессов: Администраторы могут создавать скрипты для автоматизации рутинных задач, таких как создание отчетов или обновление настроек.

- Настройка безопасности: Возможность применения политик безопасности, включая двухфакторную аутентификацию и шифрование данных, для повышения уровня защиты.

- Аналитика и отчеты: Доступ к детализированным отчетам о работе системы, что помогает в принятии обоснованных решений и оптимизации процессов.  
  
Предоставленные сервисом возможности как для пользователей, так и для администраторов направлены на оптимизацию взаимодействия и повышение уровня безопасности. Пользователи имеют доступ к удобным функциям, упрощая процесс управления своими учетными записями и получая необходимую поддержку. В то же время администраторы получают мощные инструменты для управления системой, контроля активности и обеспечения надежности. Это создает сбалансированную и эффективную экосистему, способствующую удовлетворению потребностей обоих сторон.

# **Описание внутренней логики**

Сервис реализован в виде *Bash*-скрипта, который выполняет следующие функции:

1. Добавление пользователя (*add\_user*)

Описание:

Создаёт нового пользователя в системе.

Логика:

* Принимает имя пользователя (*username*) в качестве аргумента.
* Использует команду *sudo* *useradd* *-m* для создания пользователя с домашней директорией.
* Записывает результат операции в лог-файл */var/log/user\_management.log*.

*Пример*:

add\_user "new\_user"

1. Удаление пользователя (*remove\_user*)

Описание:

Удаляет существующего пользователя.

Логика:

* Принимает имя пользователя (*username*) в качестве аргумента.
* Использует команду *sudo userdel -r* для удаления пользователя вместе с его домашней директорией.
* Записывает результат операции в лог-файл */var/log/user\_management.log*.

Пример:

*remove\_user "old\_user"*

1. Изменение прав доступа (*change\_permissions*)

Описание:

Изменяет права доступа пользователя, добавляя его в указанную группу.

Логика:

* Принимает имя пользователя (*username*) и группу (*permissions*) в качестве аргументов.
* Использует команду *sudo usermod -aG* для добавления пользователя в указанную группу.
* Записывает результат операции в лог-файл */var/log/user\_management.log*.

Пример:

*change\_permissions "new\_user" "sudo"*

1. Очистка кэша (*clear\_cache*)

Описание:

Очищает кэш пользователя, удаляя содержимое директории *~/.cache/*.

Логика:

* Принимает имя пользователя (*username*) в качестве аргумента.
* Определяет домашнюю директорию пользователя с помощью команды *eval echo "~$username"*.
* Проверяет существование директории *~/.cache/*.
* Использует команду *sudo rm -rf* для удаления содержимого директории.
* Записывает результат операции в лог-файл */var/log/user\_management.log*.

Пример:

*clear\_cache "new\_user"*

1. Остановка приложения по трафику (*stop\_app\_by\_traffic*)

Описание:

Останавливает приложение, использующее большой объём трафика.

Логика:

* Принимает пороговое значение трафика (*threshold*) в качестве аргумента.
* Использует команду *sudo netstat -tunp* для анализа сетевого трафика.
* Фильтрует приложения, использующие трафик выше указанного порога.
* Останавливает приложение с помощью команды *sudo kill -9*.
* Записывает результат операции в лог-файл */var/log/user\_management.log*.

Пример:

*stop\_app\_by\_traffic "1000"*

1. Главный цикл демона (*main*)

Главный цикл обеспечивает работу демона в режиме ожидания и обработки команд.

6.1. Ожидание команд

Использует утилиту *inotifywait* для отслеживания изменений в файле */tmp/user\_management\_commands.txt*. Когда файл изменяется, демон начинает обработку команд.

6.2. Обработка команд

* Читает команды из файла построчно.
* Каждая команда состоит из имени функции и аргументов, разделённых пробелами.
* Вызывает соответствующую функцию в зависимости от команды.

6.3. Очистка файла команд

После обработки всех команд файл */tmp/user\_management\_commands.txt* очищается с помощью команды *> /tmp/user\_management\_commands.txt*.

6.4. Бесконечный цикл

Демон работает в бесконечном цикле, постоянно ожидая новых команд.

1. Логирование

Все операции записываются в лог-файл */var/log/user\_management.log*. Логирование позволяет отслеживать выполнение команд и диагностировать возможные ошибки.

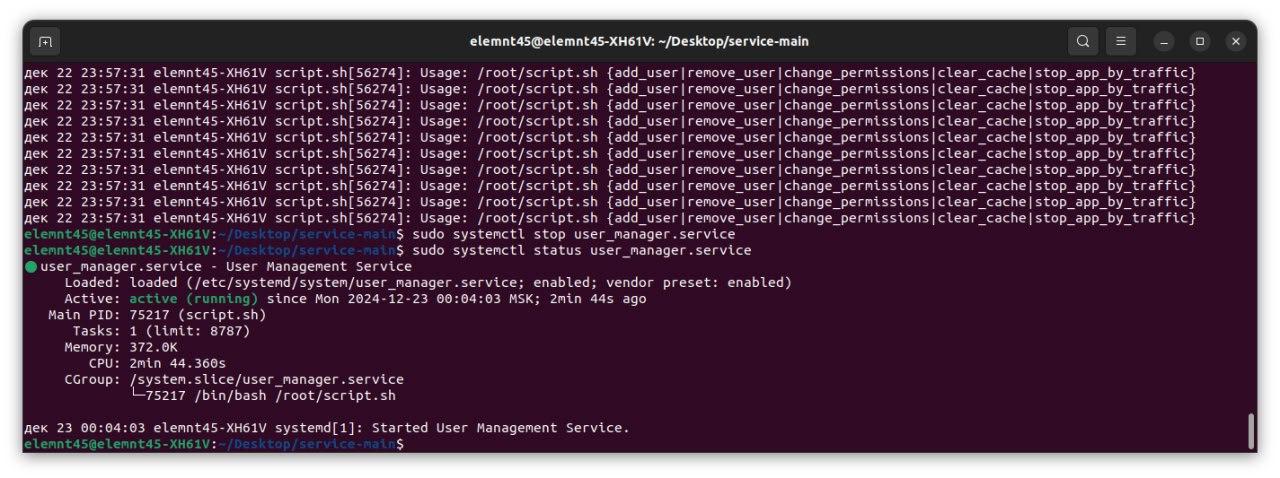
# Пример работы программы

Тестирование разработанного сервиса для управления пользователями и системными ресурсами в *Linux* является важным этапом для обеспечения его корректной работы, стабильности и безопасности. Тестирование сервиса позволит убедиться в его готовности к использованию в реальных условиях и обеспечит надежную работу системы управления пользователями и ресурсами.

В ходе тестирования была проведена проверка работы отдельных функций и команд, реализованных в скрипте *Bash*, взаимодействия сервиса с системными инструментами и другими компонентами *Linux*, а также интеграции с *systemd* (запуск, остановка, перезапуск сервиса), с системными командами (например, *useradd*, *deluser*, *chmod*, *kill*).

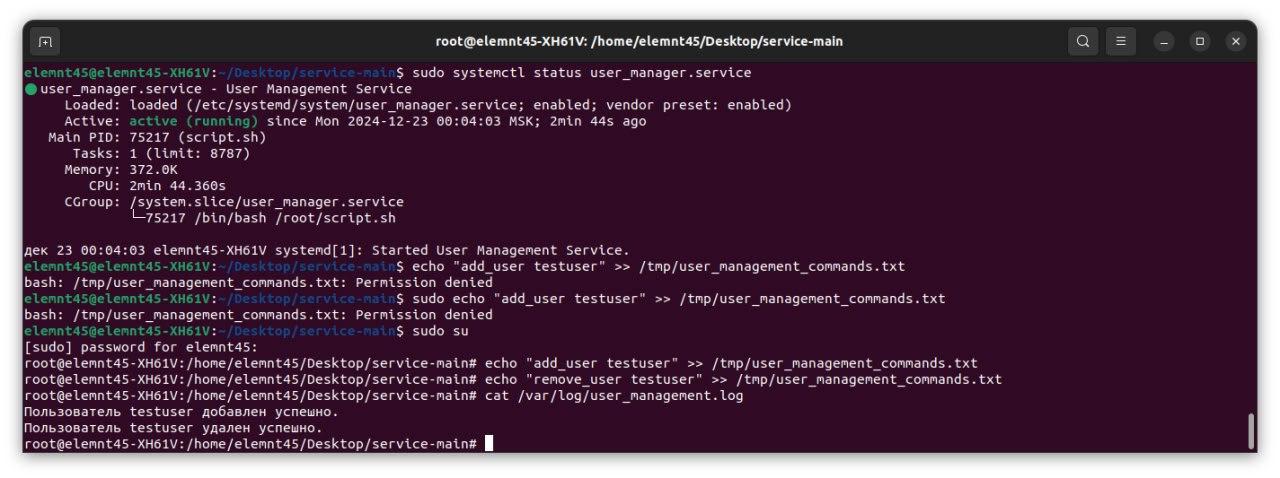
Проверка работы сервиса в реальной системной среде представлена на рисунках 1-3.

На рисунке 1 представлен статус работы сервиса, интегрированного с системой инициализации systemd. Вывод команды systemctl status отображает текущее состояние сервиса, включая информацию о том, активен ли он, запущен ли процесс, а также последние логи, связанные с его работой. Это позволяет администратору оперативно отслеживать состояние сервиса и устранять возможные проблемы.

****

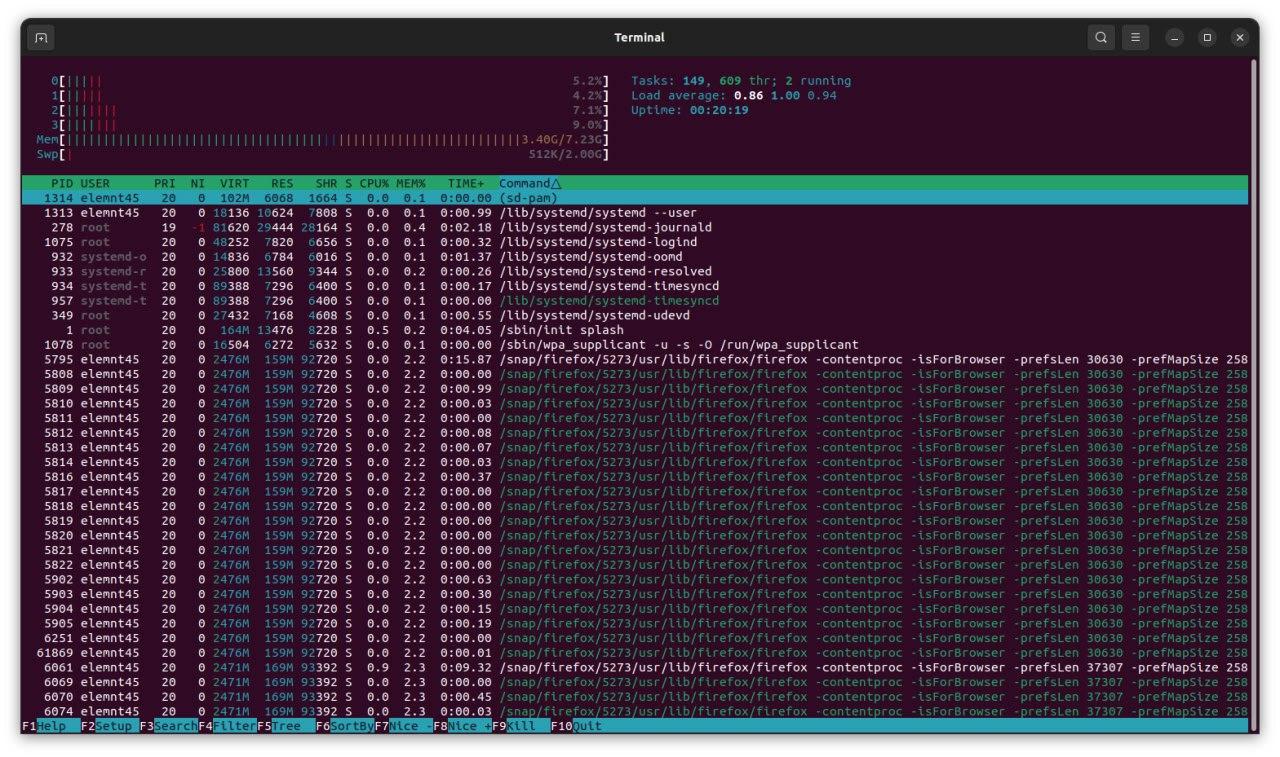
*Рисунок 1 - Статус сервиса в systemd*

На рисунке 2 показан пример выполнения команд, предоставляемых сервисом для управления пользователями и системными ресурсами. Вывод команды, выполненной через терминал, демонстрирует, как сервис обрабатывает запросы на добавление пользователя, изменение прав доступа или очистку кэша. В данном случае показан процесс добавления нового пользователя с указанием имени и пароля, а также автоматическое создание домашнего каталога.

****

*Рисунок 2 - Пример выполнения команд, предоставляемых сервисом*

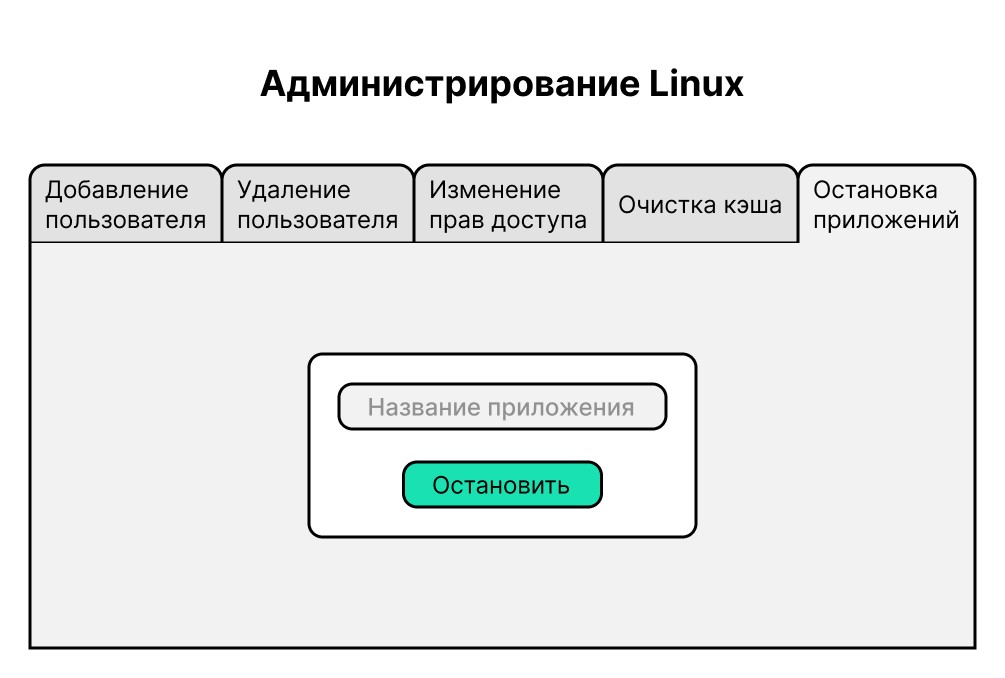
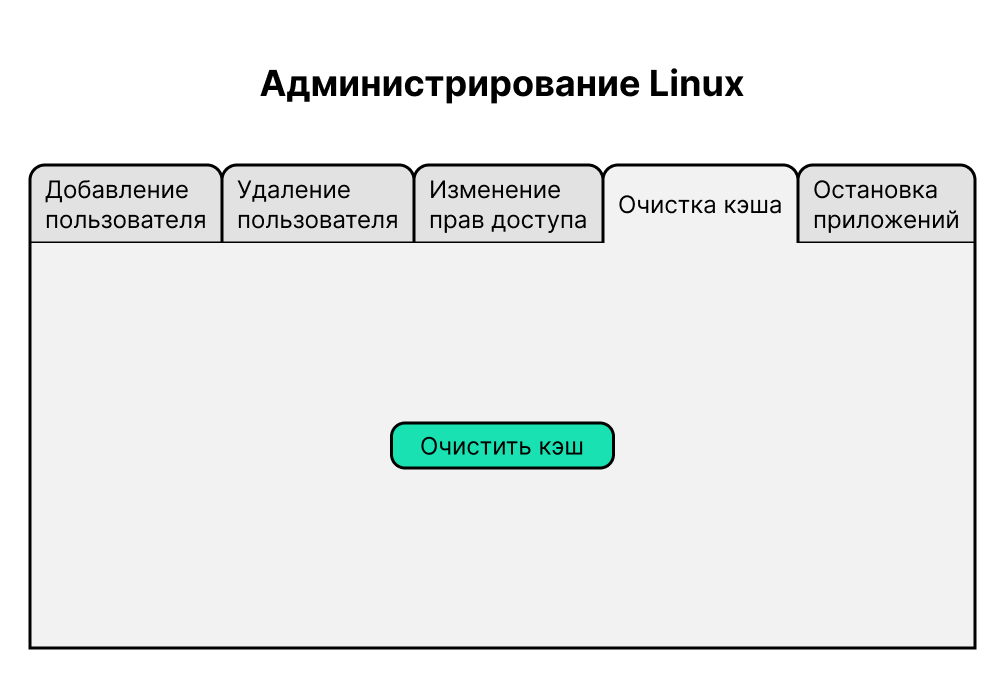
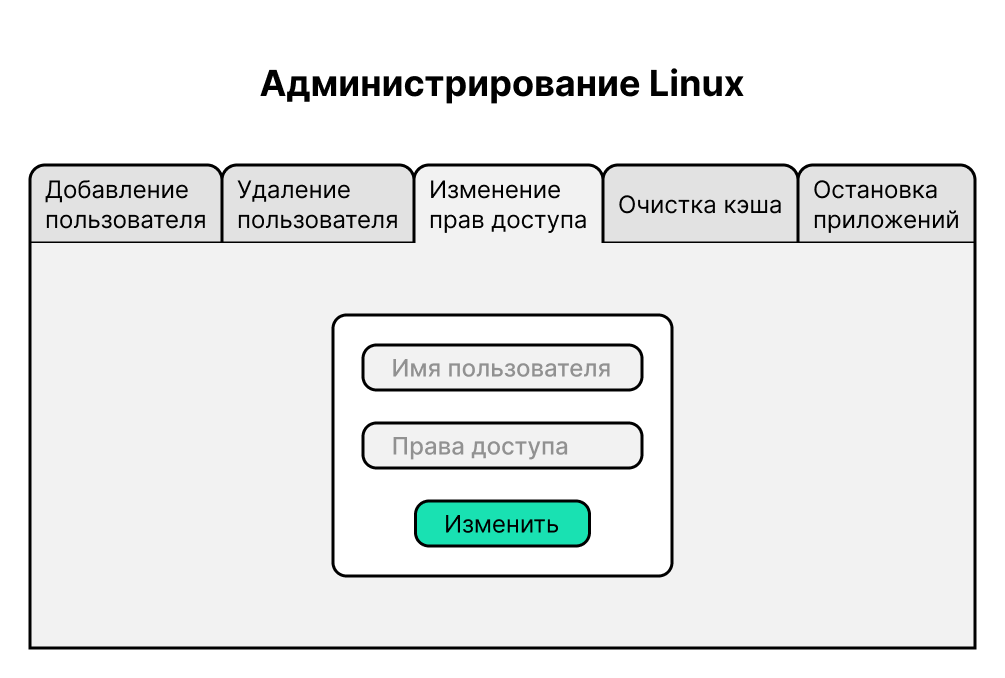
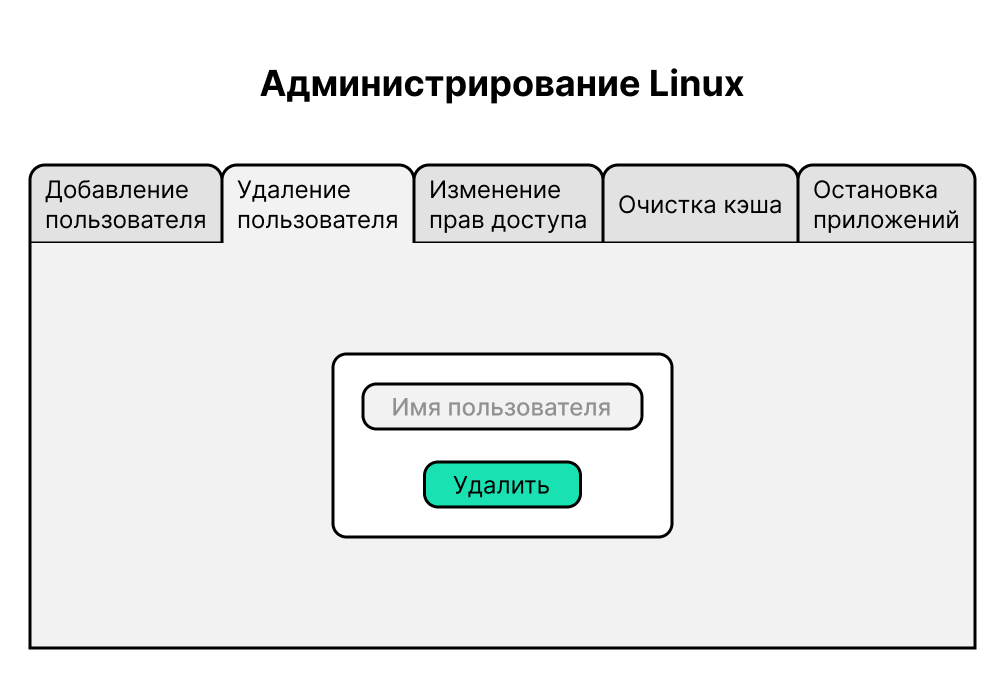
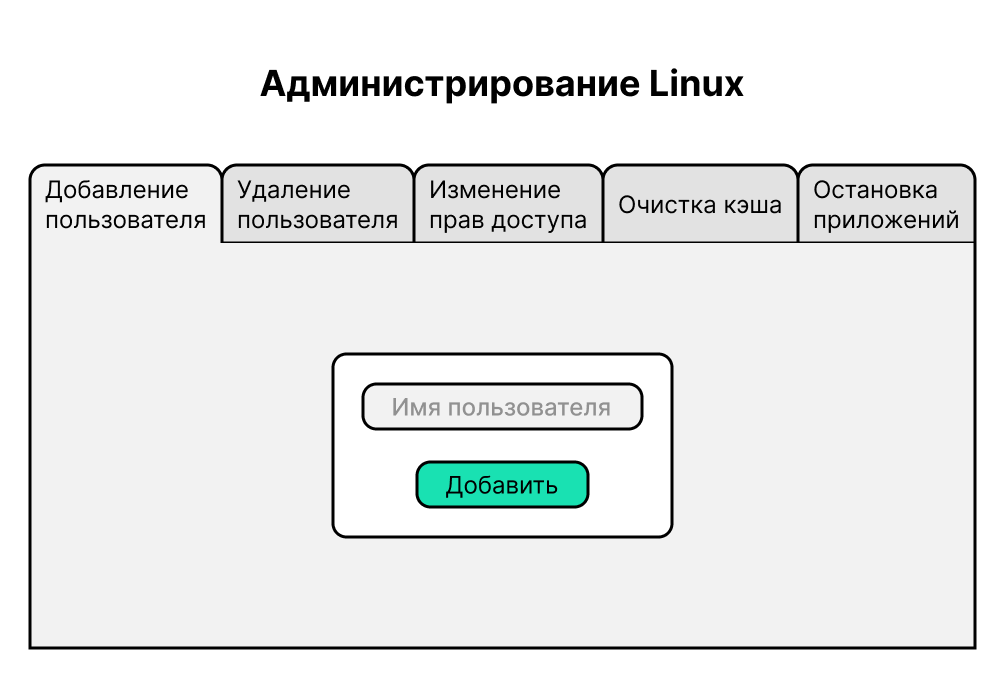
На рисунке 3 представлено отображение сервиса в трее процессов операционной системы. Сервис, работающий в фоновом режиме, отображается в списке процессов, что позволяет администратору контролировать его работу, проверять потребление ресурсов (например, CPU и памяти) и при необходимости останавливать или перезапускать его. Это важный инструмент для мониторинга стабильности и производительности сервиса.

****

*Рисунок 3 - Отображение сервиса в трее процессов*

# Графическое представление интерфейса программы

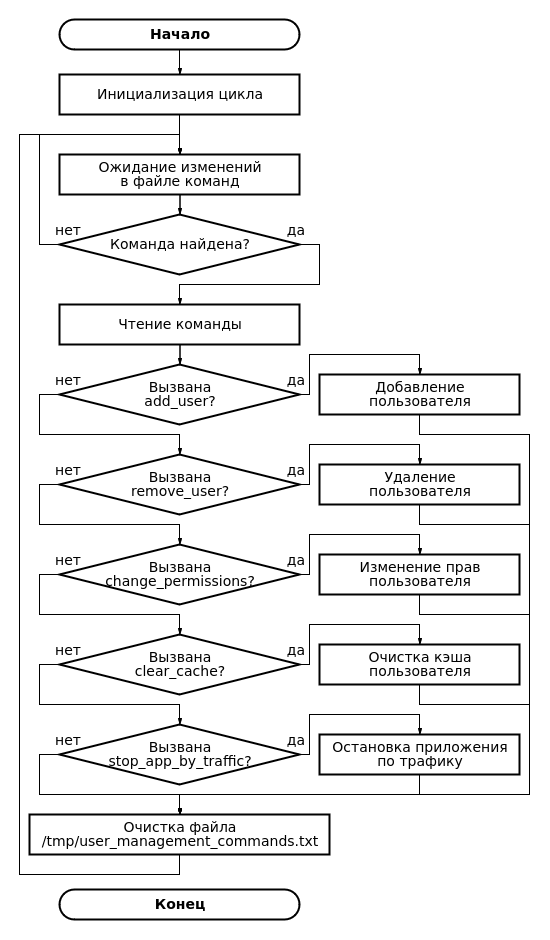
Графическое представление интерфейса программы (Рисунок 4) иллюстрирует основные элементы пользовательского интерфейса, которые обеспечивают удобное взаимодействие с сервисом для управления пользователями и системными ресурсами в операционной системе Linux. Интерфейс разработан с учетом простоты использования и интуитивной понятности, что позволяет администраторам эффективно выполнять задачи без необходимости глубокого знания командной строки.

**

*Рисунок 4 - Элементы интерфейса*

# **Схема алгоритма работы**

Блок-схема, представленная на рисунке 5, иллюстрирует процесс работы сервиса для управления пользователями и системными ресурсами в операционной системе Linux. Схема отражает основные этапы выполнения команд, взаимодействие между компонентами сервиса и системными инструментами Linux.

****

*Рисунок 5 - Схема, демонстрирующая логику работы сервиса*

**Пути дальнейшего развития проекта**

Развитие проекта сервиса администрирования *Linux* может идти в различных направлениях, в зависимости от потребностей пользователей, масштаба системы и технических возможностей. Вот несколько идей для дальнейшего развития:

1. Улучшение безопасности

Аутентификация и авторизация: Добавить вход в систему с использованием логина и пароля, двухфакторной аутентификации (2FA) или интеграции с LDAP/Active Directory.

Шифрование данных: Использовать HTTPS для защиты передаваемых данных и шифрование чувствительной информации на сервере.

Аудит действий: Вести журнал всех действий администратора (логирование) для отслеживания изменений и выявления подозрительных операций.

Ограничение доступа: Реализовать ролевую модель доступа (например, администратор, модератор, пользователь) с разными уровнями прав.

2. Расширение функционала

Управление группами пользователей: Добавить возможность создавать, удалять и управлять группами пользователей.

Мониторинг системы: Встроить инструменты для мониторинга использования ресурсов (CPU, RAM, Disk, Network).

Управление процессами: Добавить возможность просмотра и управления запущенными процессами (завершение, приоритизация).

Управление сервисами: Добавить возможность запуска, остановки и перезапуска системных сервисов.

Резервное копирование: Реализовать инструмент для создания и восстановления резервных копий важных данных.

Управление пакетами: Добавить возможность установки, обновления и удаления программных пакетов.

3. Улучшение пользовательского интерфейса

Интерактивные элементы: Добавить графики, диаграммы и таблицы для визуализации данных (например, использование ресурсов).

Адаптивный дизайн: Сделать интерфейс удобным для использования на мобильных устройствах.

Уведомления: Реализовать систему уведомлений о выполнении задач или возникновении ошибок.

Темы оформления: Добавить возможность смены темы (светлая/темная).

4. Интеграция с другими системами

API для внешних систем: Создать REST API для интеграции с другими приложениями или автоматизации задач.

Поддержка облачных сервисов: Добавить возможность управления облачными ресурсами (AWS, Google Cloud, Azure).

Интеграция с CI/CD: Добавить инструменты для управления процессами непрерывной интеграции и доставки.

5. Автоматизация

Скрипты и шаблоны: Добавить возможность создания и запуска пользовательских скриптов или шаблонов для выполнения часто повторяющихся задач.

Планировщик задач: Реализовать встроенный планировщик для автоматического выполнения задач (например, очистка кэша, резервное копирование).

Автоматическое масштабирование: Добавить возможность автоматического масштабирования ресурсов в зависимости от нагрузки.

6. Поддержка множества серверов

Управление кластерами: Добавить возможность управления несколькими серверами или кластерами из одного интерфейса.

Синхронизация настроек: Реализовать синхронизацию конфигураций между серверами.

Распределенные задачи: Добавить возможность выполнения задач на нескольких серверах одновременно.

7. Документация и обучение

Встроенная справка: Добавить раздел с документацией и инструкциями по использованию сервиса.

Интерактивные подсказки: Реализовать подсказки и туториалы для новых пользователей.

Поддержка сообщества: Создать форум или чат для обсуждения вопросов и обмена опытом.

8. Масштабирование и оптимизация

Поддержка больших систем: Оптимизировать сервис для работы с большим количеством пользователей и серверов.

Кэширование данных: Добавить кэширование для ускорения работы интерфейса.

Микросервисная архитектура: Перейти на микросервисную архитектуру для улучшения масштабируемости и отказоустойчивости.

9. Поддержка различных дистрибутивов Linux

Универсальность: Добавить поддержку различных дистрибутивов Linux (Ubuntu, CentOS, Debian и т.д.).

Автоматическое определение ОС: Реализовать автоматическое определение дистрибутива и применение соответствующих команд.

10. Открытый исходный код

Публикация на GitHub: Сделать проект открытым для привлечения сообщества разработчиков.

Поддержка плагинов: Добавить возможность расширения функционала через плагины или модули.

11. Коммерциализация

Платная версия: Добавить расширенные функции в платную версию (например, поддержка, дополнительные инструменты).

Корпоративная лицензия: Предложить корпоративную версию с расширенной поддержкой и функциями.

12. Интеграция с DevOps-инструментами

Docker/Kubernetes: Добавить управление контейнерами и оркестрацию.

Ansible/Puppet/Chef: Интегрировать с инструментами автоматизации конфигурации.

Мониторинг и логирование: Интегрировать с Prometheus, Grafana, ELK Stack и другими инструментами.

13. Мобильное приложение

Управление на ходу: Разработать мобильное приложение для удаленного управления серверами.

Push-уведомления: Добавить уведомления о критических событиях.

14. Искусственный интеллект и аналитика

Анализ логов: Использовать AI для анализа логов и выявления аномалий.

Прогнозирование нагрузки: Добавить инструменты для прогнозирования нагрузки на серверы.

Автоматическое устранение проблем: Реализовать автоматическое исправление типичных проблем.

Эти идеи помогут сделать проект более мощным, удобным и востребованным среди пользователей. Выбор направлений зависит от целевой аудитории и ресурсов, доступных для разработки.

# **Заключение**

В рамках данного проекта был разработан сервис для управления пользователями и системными ресурсами в операционной системе *Linux*. Сервис предоставляет функции добавления и удаления пользователей, изменения прав доступа, очистки кэша и остановки приложений по трафику. Реализация на языке *Bash* и интеграция с *systemd* позволили создать удобный и эффективный инструмент для автоматизации администрирования.

В будущем можно расширить функциональность сервиса, добавив поддержку дополнительных операций, таких как управление группами пользователей или мониторинг системных событий.