

Содержание

Введение	5
1 Постановка задачи	9
2 Анализ поставленных задач и разработка системы на общесистемном уровне	13
2.1 Аналитический обзор платформ и средств построения сервис-ориентированной системы.....	13
2.2 Определение основных компонентов системы	14
2.3 Обзор средств управления прокси-сервером.....	15
2.4 Формальное описание реализации системы	17
3 Разработка модели данных	18
3.1 Диаграмма модели данных	22
4 Разработка алгоритма построения конфигурации прокси.....	23
5 Разработка серверной части	27
6 Разработка клиентской части	37
6.1 Файловая структура клиентской части.....	39
6.2 Компоненты клиентского сервиса	40
7 Подготовка программно-аппаратной платформы	48
8 Тестирование.....	51
9 Экспериментальная часть	56
Заключение.....	60
Перечень сокращений, условных обозначений, символов, терминов.....	63
Список источников и литературы.....	64
Приложение А. Веб-панель альтернативного решения.....	65
Приложение Б. Блок-схемы алгоритма построения конфигурации прокси	66
Приложение В. Команды и настройки для подготовки программного окружения.....	70
Приложение Г. Тестирование серверной части.....	73
Приложение Д. Иллюстрационное сопровождение экспериментальной части	89

Изм.	Лист	№ докум.	Подпись	Дат
Разраб.	Докукин Д.В.			
Провер.	Макаров Н.Н.			
Н. контр.	Кулясов П.С.			
Утврд	Жевнерчук Д.В.			

ВКР-НГТУ-09.03.01-(20-ПО)-05-2024 (П3)

Цифровой сервис мониторинга и
управления доступом к
интернет ресурсам
Пояснительная записка

Лит. Лист Листов
4 99

НГТУ кафедра ВСТ

Введение

Объект исследования – контроль доступа пользователей к сети интернет.

Предмет исследования – инструменты обеспечения контроля доступа к сети интернет.

В рамках темы данной выпускной квалификационной работы, следующей программно-информационному, а также системотехническому и сетевому направлениям, основной целью поставлена разработка системы контроля доступа в сеть интернет с пользовательским интерфейсом, – предлагающим возможности просмотра и управления комплексными настройками, представленными в простой и понятной форме, – ориентированным на мобильные устройства. По ходу её реализации, будет рассмотрен процесс настройки аппаратно-программной среды для развёртывания компонентов системы и их применения.

Для достижения поставленной цели, были выработаны следующие задачи:

- сформулировать перечень функциональных возможностей системы;
- определить архитектуру системы и её компоненты;
- выбрать инструментальные средства разработки программного обеспечения;
- выполнить разработку и настройку компонентов;
- определить аппаратную платформу и выполнить подготовительные настройки;
- обеспечить условия целевого применения и провести тестирование.

При исследовании данных, применялись такие теоретические методы, как:

- анализ – разложение исследуемого объекта на составляющие и их изучение;
- индукция – заключение о группах объектов на примере частных случаев;
- аналогия – сравнение объектов по наличию у них схожих свойств.

В процессе разработки, из практических методов исследования применялись эксперимент и наблюдение.

Актуальность

Актуальность темы проявляется себя в различных сценариях. Средства и методики обеспечения контроля и ограничения доступа к сетевым ресурсам решают ряд задач, представляющих важность как для корпоративного сектора, государственных предприятий, так и для индивидуальных проектов энтузиастов и нужд простых граждан. К таким задачам можно отнести:

- обеспечение безопасности данных и оборудования;

Изм.	Лист	№ докум.	Подп.	Дата	Лист
					BKR-НГТУ-09.03.01-(20-ПО)-05-2024

- ограничение доступа персонала к ненадлежащим ресурсам для повышения продуктивности;
- предотвращение противоправных действий пользователями сети интернет в публичных сетях;
- фильтрация сетевых соединений для защиты несовершеннолетних от неподобающих материалов.

Базовой идеей ВКР является продолжение темы т.н. «Родительского контроля» – компонента операционных систем, программ или отдельного программного комплекса, ориентированного на обеспечение наблюдения за активностью детей в виртуальном пространстве, управления доступом к приложениям, сети интернет и устройству в целом, а также получения таких метаданных, как местоположение и перемещение. Была сформулирована концепция системы, предлагающей простой и эффективный способ создания ограничений к нежелательным интернет ресурсам, настройку доступа к определённым сайтам в установленное время, с возможностью мгновенно применять изменения, распространяемые на широкий спектр устройств, индивидуально для каждого пользователя. Однако, применение сервиса имеет место также для малых предприятий, небольших публичных сетей, например, в местах общественного питания, и т.д.

Среди ключевых особенностей, заложенных в концепт, стоит выделить простоту и гибкость использования, ориентированность панели управления к мобильным устройствам, а также свободное распространение и приватность, в случае разворачивания проекта на персональном компьютере. Принципы свободного ПО лежат в основе данной работы, и предполагается, что программные компоненты будут установлены на личных мощностях или арендованном оборудовании. Тем не менее, допускается возможность оказывать услуги по настройке системы, либо предоставлять доступ к панели управления, как сервису, поддерживаемому на стороне компании или третьих лиц.

Альтернативные решения

В процессе поиска альтернативных решений, был выполнен обзор некоторых популярных реализаций родительского контроля и сопоставление с текущей работой. Они могут быть представлены как на системном уровне, так и на уровне прикладных программ.

Интеграция функционала на системном уровне имеет ряд преимуществ, среди которых простота использования и тесная связь с операционной системой, благодаря чему, ожидается, что разработчики способны предоставить наиболее полный спектр возможностей. Однако, подобные средства рассчитаны на широкую публику, что

Изм.	Лист	№ докум.	Подп.	Дата	Лист
					6

отражается на сложности и глубине настроек, поскольку, доступность для рядовых пользователей становится одним из ведущих принципов в проектировании программы.

Так, встроенный родительский контроль на операционных системах «Android» и «iOS» позволяет ограничить доступ к приложениям на устройстве, блокировать возможность покупки цифровых продуктов, установить настройки отслеживания устройства. Функции фильтрации веб-контента также представлены, но ограничены определением списка доменных имён и не позволяют сделать более детальную настройку. Таким образом, встроенные решения отличает простота и надёжность, но предложенного функционала может оказаться недостаточно.

Функции родительского контроля также могут встречаться на отдельных сайтах: в социальных сетях, видео-хостингах, поисковиках и не только. Они нацелены на фильтрацию неприемлемых материалов и контента для взрослых, а некоторые сервисы позволяют задавать т.н. «чёрные» и «белые» списки – белыми списками именуют набор разрешенных условий, правил, ресурсов и т.п., тогда, как прочее будет считаться запрещенным; чёрные списки представляют обратный подход – для материалов, размещённых на цифровых площадках, однако, только в рамках собственных платформ. Такие решения предоставляет компания «Google» в рамках собственных сервисов, например, для видео-хостинга «YouTube» и одноимённой поисковой системы. Так или иначе, часто можно найти альтернативные сервисы, вследствие чего, точечная настройка некоторых из них теряет смысла, в случае если не ограничивать доступ к другим сервисам сторонними инструментами.

Многие из существующих приложений для родительского контроля делают акцент на контроле перемещения, ограничения доступа к приложениям, т.е. в большинстве случаев, предлагают функционал, аналогичный встроенным системным решениям, нередко требуя за это плату. Тем не менее, существуют приложения, предлагающие расширенный функционал.

Среди коммерческих решений, представлено приложение, частично реализующее целевой функционал данной работы – «Parental Control Kroha». Оно является платным и предлагает функцию веб контроля, благодаря которой, можно ограничивать доступ к определённым сайтам и наблюдать за сетевой активностью. Согласно отзывам пользователей устройств под операционной системой «iOS» на странице приложения в цифровом магазине «AppStore» [6], при заданных настройках ограничения сетевого доступа, приложение создаёт VPN-конфигурацию, используемую для туннелирования сетевого трафика устройства. Это даёт возможность родителям просматривать сетевую

Изм.	Лист	№ докум.	Подп.	Дата	Лист
					7

активность ребёнка и мгновенно вносить изменения в настройки. Однако, данный функционал обеспечивается удалёнными серверами компании, где данные могут быть подвержены сбору, анализу и передаче третьим лицам. При использовании подобных программ подвергается риску приватность сетевой активности ребёнка, что может быть критичным для небезразличных родителей. Более того, сторонние программы, реализующие глубокий контроль над устройством, также имеют прямой доступ к локальным данным, и недоступность исходных кодов программного обеспечения не позволяет удостовериться в добросовестности владельцев сервиса.

Проект ВКР нельзя в полной мере считать альтернативой программам родительского контроля, поскольку он решает узкий спектр задач, а для подключения устройств предполагается использование сторонних программ. Вместе с этим, объединив его с прочими решениями, наиболее удовлетворяющими запросам пользователей, можно добиться большей гибкости в управлении, при сохранении приватности, а также повышении безопасности данных и устройства.

Наиболее близким решением к проекту ВКР, согласно результатам поиска, является проект с открытым исходным кодом «Gatesentry» [2]. Он также совмещает в себе функции фильтрации сетевого трафика посредством интегрированных сервисов proxy и dns, что делает его таким же приватным, при развертывании на собственном оборудовании. Однако, настройки ограничений доступа представлены простым списком доменных имён, что не отличает данный проект от встроенных системных решений, но для применения настроек требуется включить функцию MITM (англ. Man In The Middle), которая расшифровывает сетевые пакеты HTTPS трафика для анализа, что требует повышенных мер защиты для сервера и дополнительных настроек подключения. Также, пользовательский интерфейс слабо адаптирован к мобильным форматам экрана. Демонстрация веб-интерфейса приложения приводится в приложении А.

Таким образом, реализация темы ВКР остаётся актуальной за счёт уникальности предлагаемого функционала, совмещенного с сохранением приватности относительно коммерческих решений, и позиционирования в качестве свободно распространяемого продукта.

Изм.	Лист	№ докум.	Подп.	Дата	Лист	ВКР-НГТУ-09.03.01-(20-ПО)-05-2024	8

1 Постановка задачи

Разрабатываемая система предназначается для создания правил доступа к сетевым адресам.

К областям применения могут относиться:

- обеспечение родительского контроля для доступа детей к сети интернет;
- повышение безопасности сетевой активности при помощи блокировки мошеннических сайтов и доменов, не вызывающих доверие;
- ограничение выхода в интернет для сотрудников с целью обеспечения сетевой безопасности организации, предотвращения утечек конфиденциальной информации и отвлечения персонала от рабочих задач.

С целью создания ограничений доступа к компьютерным сетям, широко применяется такой тип программного обеспечения, как прокси-сервер. Он представляет собой программу, – непрерывно выполняемую на компьютере, интегрированном в цифровую сеть – служащую промежуточным узлом для пользователей между ними и сетью интернет. Прокси-сервер получает запрос от клиента на доступ к ресурсу, после чего, прокси может подключиться к нему, либо, согласно настройкам в конфигурации, например, пресечь доступ.

Существует ряд сетевых протоколов, обеспечивающих соединения с прокси-серверами. Одним из самых распространённых и поддерживаемых является протокол HTTP – главный протокол для обмена данными в интернете. Однако, сам по себе, в настоящее время, он не является оптимальным решением для передачи данных, поскольку HTTP-запросы передаются в открытом виде, что позволяет просматривать сетевые пакеты, следствием чего возникает риск утечки передаваемой, незащищённой информации. Это делает методы аутентификации пользователей ненадёжными, однако, для публичных прокси-серверов, данное обстоятельство не является проблемой и позволяет обрабатывать большее число соединений за счёт отсутствия необходимости реализации шифрования трафика. При таком сценарии, в ходе обработки сетевых пакетов, будет видно, куда подключается клиент, а также содержащуюся информацию, но в случае поддержки целевым ресурсом шифрования, передача данных с клиентом может быть безопасной.

Чтобы обеспечить защиту соединений по протоколу HTTP, используются дополнительные протоколы шифрования. Наиболее современным, из принятых на текущее время, является TLS. В ходе обеспечения шифрования, используются асимметричные алгоритмы, такие, как RSA, или ECDSA и EdDSA, основанные на

							Лист
Изм.	Лист	№ докум.	Подп.	Дата		VKP-НГТУ-09.03.01-(20-ПО)-05-2024	9

эллиптических кривых. При инициировании соединения, используется т.н. TLS-рукопожатие – набор сообщений, которыми обмениваются клиент и сервер, чтобы совершить передачу криптографических ключей для реализации защищённого канала связи. В целях удостоверения, что соединение безопасно и выполняется с правильным ресурсом, используются TLS-, или, по-другому, SSL-сертификаты. Сертификаты могут быть связаны в цепочку, чтобы один корневой сертификат мог удостоверять подлинность остальных. Они хранятся на устройстве клиента. Ключи шифрования, передаваемые сервером клиенту, должны быть связаны с сертификатом, либо имеющимся на стороне клиента, либо быть подписанным другим из имеющихся. В противном случае, устройство распознает подключение, как ненадёжное.

Для части системы, отвечающей за осуществление ограничений доступа к сети, были определены следующие требования:

- обеспечение соединения пользователя с интернетом посредством прокси-сервера;
- использование протокола HTTP для коммуникации с прокси-сервером;
- обеспечение защищённого соединения с применением протокола TLS;
- многопользовательская поддержка;
- реализация аутентификации пользователей;
- возможность создания, как общих, так и индивидуальных правил доступа;
- настройка правил по времени.

Опираясь на цель, поставленную в рамках ВКР, возникает необходимость осуществления интерфейса, следующего принципам простоты и доступности для рядовых пользователей. Согласно результатам публичной интернет площадки «statcounter» [1], доля пользователей мобильного интернета в апреле 2024 года составляет практически 60%. Смартфонами пользуются 4.88 миллиарда людей, по информации из источника «bankmycell» [3]. С 2018 года начался рост популярности такой технологии, как прогрессивные веб-приложения (или PWA), связанный с добавлением поддержки в операционную систему «iOS» с версии 11.3. Её суть заключается в возможности адаптировать сайт под экран смартфона и открывать его, как обычное мобильное приложение. С опорой на перечисленное, сформулированы требования для части системы, выступающей в роли прослойки между управляющим лицом и программными средствами реализации контроля:

- создание простого графического интерфейса, оформленного в едином стиле;
- поддержка интерфейсом формата экранов смартфонов в портретном режиме;

Изм.	Лист	№ докум.	Подп.	Дата	Лист	10
					BKR-НГТУ-09.03.01-(20-ПО)-05-2024	

- поддержка технологии PWA;
- обеспечение аутентификации пользователей с разделением по правам доступа;
- просмотр своего имени пользователя (для авторизованных пользователей);
- возможность обновления учетных данных (для авторизованных пользователей);
- предоставление сведений о прокси-сервере (для авторизованных пользователей);
- возможность установить SSL-сертификат, удостоверяющий подлинность прокси-сервера (для авторизованных пользователей);
- возможность экстренного отключения прокси-сервера (для администраторов);
- просмотр списка пользователей (для администраторов);
- возможность изменения учетных данных пользователей (для администраторов);
- возможность добавления и удаления пользователей (для администраторов);
- возможность просмотра, добавления, изменения и удаления доменных имен для управления доступом (для администраторов);
- возможность разрешить или запретить доступ к домену (для администраторов);
- возможность настраивать доступ по времени (для администраторов);
- возможность группировать домены по спискам (для администраторов);
- возможность устанавливать общее правило доступа для всего списка (для администраторов);
- возможность устанавливать общее правило доступа индивидуально для каждого пользователя (для администраторов);
- возможность устанавливать глобальные настройки доступа для всех пользователей (для администраторов);
- возможность устанавливать общее правило доступа в рамках глобальных настроек (для администраторов);
- возможность создавать списки доменов для глобальных правил доступа (для администраторов);
- разделение функций внесения изменений и их сохранения.

Описанные требования по возможностям системы создают необходимость организации комплексной логики приложения и хранения служебных данных. С этой целью должна быть разработана часть программы, обеспечивающая функциональную

Изм.	Лист	№ докум.	Подп.	Дата	Лист
					11

составляющую графического интерфейса и управление прокси-сервером. Это приводит к следующим задачам:

- использование сервис-ориентированной архитектуры при разработке системы;
- развертывание и настройка сервиса прокси-сервера;
- разработка серверной части, отвечающей за логику системы;
- использование СУБД для хранения данных;
- разработка клиентской части, предоставляющей графический, пользовательский интерфейс системы.

В целях обеспечения работы всех компонентов, необходимо подготовить программно-аппаратную платформу. Среди операционных систем, для серверов и суперкомпьютеров широко используются системы на базе ядра Linux. Их отличает гибкость настроек, высокая производительностью, а также возможность использовать программное обеспечение без необходимости платить за него. На основании перечисленных особенностей, был выбран дистрибутив Linux Debian редакции Bookworm (версия 12) – бесплатная и свободно распространяемая операционная система, поддерживаемая сообществом разработчиков. В качестве аппаратной платформы, исходя из рекомендаций разработчиков Debian [8] и занимаемого пространства данными разрабатываемой системы, необходима конфигурация оборудования со следующими характеристиками:

- 64-битный процессор на архитектуре x86-64 с частотой от 1.5 гигагерц;
- объём оперативной памяти, составляющий не менее 1 гигабайт;
- объём постоянного накопителя не менее 5 гигабайт.

Тестирование будет проведено в локальной сети, с настроенным dns-сервером, чтобы обеспечить условия, приближенные к целевым. SSL-сертификат требует, при создании, определить доменное имя сервиса, для подтверждения безопасности соединения.

Изм.	Лист	№ докум.	Подп.	Дата	Лист	12
					VKR-НГТУ-09.03.01-(20-ПО)-05-2024	

2 Анализ поставленных задач и разработка системы на общесистемном уровне

2.1 Аналитический обзор платформ и средств построения сервис-ориентированной системы

Существует множество вариаций создания сервис-ориентированных систем. Так, среди наиболее популярных инструментов разработки пользовательских интерфейсов можно выделить:

- React – является JavaScript-библиотекой для создания пользовательских интерфейсов. Имеет поддержку как одностраничных, так и мультистраничных приложений.
- Vue.js – фреймворк для создания пользовательских интерфейсов на JavaScript. Используется при написании простых и гибких приложений.
- Angular – фреймворк от компании Google. По сравнению с предыдущими, имеет более высокий порог вхождения в разработку.

Для создания серверной части используются следующие инструменты:

- Java – серверная часть разрабатывается на Java и выполняется на виртуальной машине Java. Используется во многих высоконагруженных корпоративных проектах.
- Node.js – платформа на основе движка V8 от Google, для разработки сервис-ориентированных приложений на JavaScript. Лёгок в освоении, позволяет очень быстро написать рабочий проект.
- Go – компилируемый язык программирования, разработанный компанией Google. Предназначается для разработки высокоэффективных программ. Его преимущество заключается в том, что программы, написанные на Go, транслируются в объектный код и исполняются без использования виртуальной машины.

Среди систем управления базами данных (СУБД) можно отметить:

- PostgreSQL – объектно-реляционная СУБД. Бесплатное ПО с открытым исходным кодом. Отличается надежностью и целостностью данных. Среди недостатков можно выделить определённую сложность при настройке, а также повышенное потребление ресурсов в сравнении с аналогами.
- MySQL – давно зарекомендовавшая себя реляционная СУБД. Однако, скоростные показатели уступают остальным при работе с большими объёмами данных.

Изм.	Лист	№ докум.	Подп.	Дата	Лист	13
					VKR-НГТУ-09.03.01-(20-ПО)-05-2024	

- MongoDB – документоориентированная СУБД. К преимуществам относятся легкость в масштабировании, скорость при поиске и чтении. Рекомендуется в случаях, когда нужно работать с большим количеством не связанных друг с другом данных.

Из программ прокси-серверов, можно отметить следующие варианты:

- Squid Proxy – программный пакет, предлагающий поддержку HTTP и HTTPS соединений. Обеспечивает TLS-рукопожатие. Имеет поддержку аутентификации пользователей, многопользовательский режим, широкий набор правил доступа. Легко настраивается через конфигурационный файл. Популярен среди энтузиастов.
- HAProxy – обладает всеми преимуществами Squid Proxy. Используется в качестве балансировщика нагрузки на серверах.
- TinyProxy – легковесный прокси с минимальным набором настроек. Уступает по функционалу Squid Proxy.

2.2 Определение основных компонентов системы

При выполнении выпускной квалификационной работы, ставится условная задача закрепления полученных знаний и опыта,обретенных в ходе обучения, вследствие чего, были выбраны наиболее эффективные или подходящие инструменты, с которыми автору приходилось взаимодействовать ранее.

В качестве прокси был выбран Squid Proxy, за счёт простой конфигурации, возможности управлять его состоянием автономно, а также большой распространённости среди целевой аудитории.

Для серверной части системы, была выбрана платформа node.js с использованием фреймворка Express, для обеспечения серверного функционала, поскольку преимущества простоты языка JavaScript, используемой функциональной парадигмы и высокая скорость написания проекта были расценены с большим приоритетом в условиях ограниченного времени. Погружение в возможности языка JavaScript происходило при помощи документации, размещённой на сайте «MDN Web Docs» [4].

Для организации хранения информации и описания модели данных, была выбрана ORM – технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных» [5] – под названием Sequelize. Она позволяет работать с данными из СУБД средствами языка программирования, что значительно упрощает и ускоряет процесс разработки. Изучение возможностей ORM выполнялось с помощью

Изм.	Лист	№ докум.	Подп.	Дата	Лист	14
					BKR-НГТУ-09.03.01-(20-ПО)-05-2024	

документации, размещённой на сайте ресурса [10]. В качестве СУБД использована PostgreSQL версии 16. Побочным преимуществом PostgreSQL над MySQL можно отметить простоту установки на операционной системе Linux Debian. В то время, как вторую СУБД необходимо устанавливать с сайта разработчика, а процесс установки неочевиден и сложен, PostgreSQL можно установить легко и быстро, при помощи стандартного пакетного менеджера.

Для клиентской части, была выбрана библиотека React – её преимуществом можно отметить лёгкость в создании компонентов интерфейса, при использовании языка HTML и каскадной таблицы стилей CSS, и дальнейшем их применении. Для коммуникации с серверной частью, была взята библиотека axios.

2.3 Обзор средств управления прокси-сервером

Согласно информации, приведённой в книге, посвященной изучению Squid [11], управление прокси-сервером Squid осуществляется при помощи конфигурационного файла. В случае установки с помощью пакетного менеджера линукс дистрибутива, конфигурационный файл будет расположен по пути «/etc/squid/squid.conf». В случае сборки программы из исходных кодов и ручной установки по пути «/usr/local/squid», конфигурационный файл будет расположен по пути «/usr/local/squid/etc/squid.conf». Настройки задаются через специальные теги.

Для настройки доступа к сети через прокси сервер используются так называемые списки контроля доступа или acl (англ. access control list). ACL-параметры конфигурации, в контексте составления настроек доступа в сеть через прокси, представляют собой описание условий, согласно которым осуществляется проверка сетевых пакетов для последующей переадресации на адрес назначения или блокировки. Данные параметры имеют следующий формат:

«acl имя тип аргументы ...»

Среди типов acl, интерес для проекта представляют следующие:

- dst – позволяет определить ip-адрес назначения или перечень адресов для применения к ним настроек доступа;
- dstdomain – отличается от предыдущего тем, что в качестве аргументов принимает URL и доменные имена ресурсов назначения;
- time [аббревиатуры_дней] [диапазон_времени] – тип позволяет задать режим работы правил, исходя из указанных дней и диапазона времени, для настроек доступа. Дни, либо время могут быть опущены. В таком случае, правило будет работать для всего диапазона значений, если прочие правила не исключают

Изм.	Лист	№ докум.	Подп.	Дата	Лист	15
					BKR-НГТУ-09.03.01-(20-ПО)-05-2024	

этого. Время необходимо указывать парами «час:минута» через дефис в следующем формате: «ч1:м1-ч2:м2». Значение первой пары должно быть меньше второй. Чтобы правило действовало до полуночи, допускается указывать значение «24:00». Дни задаются при помощи специальных аббревиатур:

- 1) S – Воскресенье
- 2) М – Понедельник
- 3) Т – Вторник
- 4) W – Среда
- 5) Н – Четверг
- 6) F – Пятница
- 7) А – Суббота
- 8) D – Все дни

- proxy_auth – при передаче в качестве аргументов имён зарегистрированных пользователей, позволяет применять к ним правила доступа. Требует внешнюю программу аутентификации для сверки имён;
- any-of – даёт возможность объединить несколько acl-параметров конфигурации под одним именем. Запрос будет удовлетворён при выполнении условий хотя бы одного из объединённых параметров.

Настройки, позволяющие передавать неограниченное число параметров, можно разбить на несколько строк, используя одно имя acl. В таком случае, настройки, применяемые к данному acl, будут распространяться на все параметры.

Также, следует рассмотреть следующие теги:

- http_access – определяет значение доступа для списков контроля доступа (acl), переданных в качестве аргументов. Имеет следующий формат записи: «http_access allow|deny [!]aclname ...», где allow – разрешить, deny – запретить, «[!]aclname ...» перечисление имён определённых acl, если указать восклицательный знак перед именем acl, настройка будет работать при несоответствии условиям acl.
- auth_param – используется для определения схемы аутентификации, поддерживаемой squid, то есть, настраивает способ проверки аутентификации пользователей. Данный тег имеет следующий формат записи: «auth_param схема параметр [настройки]».

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

- `http_port` номер – атрибут тега определяет порт, на котором прокси-сервер будет слушать входящие подключения.
- `https_port` номер – применяется для обеспечения защищённого соединения по протоколу HTTPS. Обязательными опциями данного тега являются «`tls-cert=»` и «`tls-key=»` – первая указывает на сертификат, используемый при TLS-рукопожатии, а вторая на приватный ключ сервера.

2.4 Формальное описание реализации системы

Таким образом, подводя итоги по рассмотренным средствам, общая схема проекта обретает следующий вид:

- Серверная часть в лице сервиса на платформе «node.js» для настройки прокси-сервера, обеспечения логики системы, взаимодействия с базой данных и предоставления интерфейса для обмена и обработки данных с клиентских приложений администраторов. Включает такие библиотеки, как express – для реализации функций веб-сервера, и sequelize – для общения с СУБД;
- Прокси-сервер Squid для фильтрации сетевого трафика и осуществления ограничений доступа пользователей;
- Система управления базами данных PostgreSQL для хранения данных пользователей и настроек доступа;
- Клиентская часть, реализованная на языке программирования JavaScript с применением библиотеки React для простого, гибкого и эффективного создания графического веб-интерфейса. Включает библиотеку axios для взаимодействия с веб-сервером.

Изм.	Лист	№ докум.	Подп.	Дата	Лист
					17

3 Разработка модели данных

Исходя из требований, определённых при постановке задачи, разрабатываемая модель данных должна включать в себя такие сущности, как:

- пользователь – для хранения учетных данных пользователей, их уровня доступа и прочих индивидуальных параметров;
- список – списки, используемые для группировки адресов, должны иметь собственные имена, а также настройки доступа, применяемые ко всей группе, что следует выделить в отдельную модель;
- адрес – исходя из названия, предназначается для хранения адресов интернет-ресурсов, для которых будут применяться настройки доступа;
- правило – настройки доступа могут применяться как к адресам, так и к группам адресов, отдельным пользователям и глобально, поэтому их можно отнести в одну общую модель;
- исключение – у каждого правила может быть одно или несколько исключений, имеющих разные параметры, которые необходимо хранить отдельно, с возможностью доступа к ним в соответствии с рассматриваемым правилом.

Рассмотрим каждую сущность подробнее.

Сущность пользователя должна хранить данные, обеспечивающие авторизацию, разделение привилегий и индивидуальную настройку сетевого доступа. Исходя из этого, в модели необходимо отразить такие атрибуты, как:

- имя пользователя;
- пароль;
- роль (например, администратор или пользователь);
- правило доступа пользователя.

Сущность списка является промежуточным звеном между адресами и пользователями, поскольку каждый список относится к одному пользователю, но в них может храниться множество адресов. Атрибуты данной сущности должны выглядеть следующим образом:

- название списка;
- ссылка на пользователя;
- правило доступа списка.

Если представить иерархию пользователей, списков и адресов, адреса расположатся на нижнем уровне, но будут представлять собой самую многочисленную

Изм.	Лист	№ докум.	Подп.	Дата	Лист	18
					VKR-НГТУ-09.03.01-(20-ПО)-05-2024	

прослойку. Поскольку они относятся к спискам, которые, в свою очередь, относятся к пользователям, адрес должен иметь ссылку только на список, к которому он относится. Помимо непосредственно значения адреса, может различаться его тип – доменное имя или ip-адрес. Таким образом, был сформулирован следующий набор атрибутов:

- тип адреса;
- значение адреса;
- ссылка на список;
- правило доступа адреса.

Сущность правила хранит параметр доступа для пользователей, списков и адресов. Для сущности пользователей, значение параметра доступа определяет правило доступа ко всем интернет ресурсам. Так, если параметру присвоить условное значение «запретить», доступ к интернету для данного пользователя по умолчанию будет закрыт. Для сущности списка, параметр доступа определяет правило доступа, ко всем адресам в списке, по умолчанию, тогда, как для сущности адреса правило применяется непосредственно по отношению к указанному адресу, но в соответствии с владельцем списка, к которому относится адрес. Правила работают таким образом, что параметр доступа для списка будет переопределять правило пользователя, а параметр адреса переопределит правило списка. Дополнительно, исходя из возможности наличия для правил исключений в множественном количестве, следует предоставить возможность для их быстрой отмены. Согласно описанной концепции, определены атрибуты:

- тип доступа (например, «разрешить» или «запретить»);
- состояние исключений (например, «включить» или «отключить»).

Сущность исключений по своему содержанию представляет собой расписание, согласно которому значение доступа будет обратным по отношению к параметру правила, к которому применяется исключение. Таким образом, хранить непосредственно значение доступа не требуется. Исходя из возможностей настроек прокси-сервера Squid, доступ может быть ограничен по дням недели и по времени, причём, если опустить один из параметров, значение второго будет применяться в общем случае: так, если указать настройку доступа в диапазоне заданного времени без указания дней, она будет применяться в любой день, а настройка доступа с заданными днями без времени будет действовать в течение всех указанных суток. Таким образом, список атрибутов обретает следующий вид:

- дни недели;
- диапазон времени;

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

- ссылка на правило доступа.

Следующим шагом идёт разработка модели согласно возможностям используемой системы управления базами данных.

PostgreSQL предлагает расширенную поддержку типов данных по сравнению с альтернативными решениями. Следует отметить такие типы, как «ENUM» или перечисление, позволяющее задать конечное число допустимых значений поля таблицы, и «JSON», для хранения json-объектов, представляющих собой множество неупорядоченных пар «ключ:значение», заключенных в фигурные скобки. Поддержка второго типа позволяет вносить объекты данного типа в базу данных без необходимости предварительной обработки, например, конвертации в строку. Помимо упомянутых типов данных, из полезных для проекта, в PostgreSQL имеется широкая поддержка целочисленных переменных различной разрядности (однако, в рамках поставленных задач, будет достаточно 32 битного типа «INTEGER»), а также строковых значений с определяемым максимальным размером при помощи типа «STRING».

В таблице 1 ниже приводится разработанная модель с указанием сущностей, атрибутов и их типов.

Таблица 1. – Модель данных

Сущности	Атрибуты	Тип данных атрибутов	Описание атрибута
User	id	INTEGER	Идентификатор пользователя
	name	STRING	Имя и логин пользователя
	password	STRING	Пароль пользователя
	role	ENUM('ADMIN', 'USER', 'GLOBAL')	Роль пользователя: ADMIN – для администратора USER – для пользователя прокси-сервера GLOBAL – специальная роль, используемая для применения настроек на всех пользователей
	ruleId	INTEGER	Идентификатор правила доступа пользователя
List	id	INTEGER	Идентификатор списка
	name	STRING	Название списка

Продолжение таблицы 1

Сущности	Атрибуты	Тип данных атрибутов	Описание атрибута
	userId	INTEGER	Идентификатор пользователя-владельца списка
	ruleId	INTEGER	Идентификатор правила доступа списка
Address	id	INTEGER	Идентификатор адреса
	type	ENUM('DOMAIN', 'IP')	Тип адреса: DOMAIN – для доменных имён IP – для IP-адресов
	value	STRING	Значение адреса
	listId	INTEGER	Идентификатор списка, к которому относится адрес
	ruleId	INTEGER	Идентификатор правила доступа адреса
Rule	id	INTEGER	Идентификатор правила доступа
	access	ENUM('ALLOW', 'DENY')	Параметр доступа: ALLOW – разрешить доступ DENY – запретить доступ
	schedule	ENUM('ON', 'OFF')	Параметр исключений: ON – использовать исключения OFF – не использовать исключения
Exception	id	INTEGER	Идентификатор исключения
	days	STRING(8)	Атрибут для хранения дней недели в строковом формате, ограниченный размером в 8 байт (каждый день выражается специальной буквой)
	time	JSON	Атрибут для хранения диапазона времени действия исключения (ожидается наличие двух ключей, обозначающих начало и конец диапазона)
	ruleId	INTEGER	Идентификатор правила доступа

Между сущностями User (пользователь) и List (список) действует связь «один ко многим» соответственно. Это означает, что у каждого пользователя может быть несколько списков, но каждый список может принадлежать только одному пользователю. Из этого также следует, что у каждого списка должен быть идентификатор пользователя-владельца списка.

Такой вид связи также действует между сущностями List (список) и Address (адрес) соответственно, и между Rule (правило) и Exception (исключение) соответственно, из чего следует, что у адресов должен быть атрибут идентификатора списка, а у исключений идентификатор правила.

Между сущностями User, List, Address и сущностью Rule действует связь «один к одному». Согласно ней, каждому пользователю, списку или адресу может соответствовать одно правило. Поскольку сущность правила служит для трёх различных сущностей, в этом случае, необходимо помещать идентификатор правила в атрибуты сущностей пользователей, списков и адресов. В свою очередь, сущность правила остаётся единственной, не имеющей ссылок в атрибутах.

3.1 Диаграмма модели данных

Ниже, на рисунке 1, приводится диаграмма разработанной модели данных.

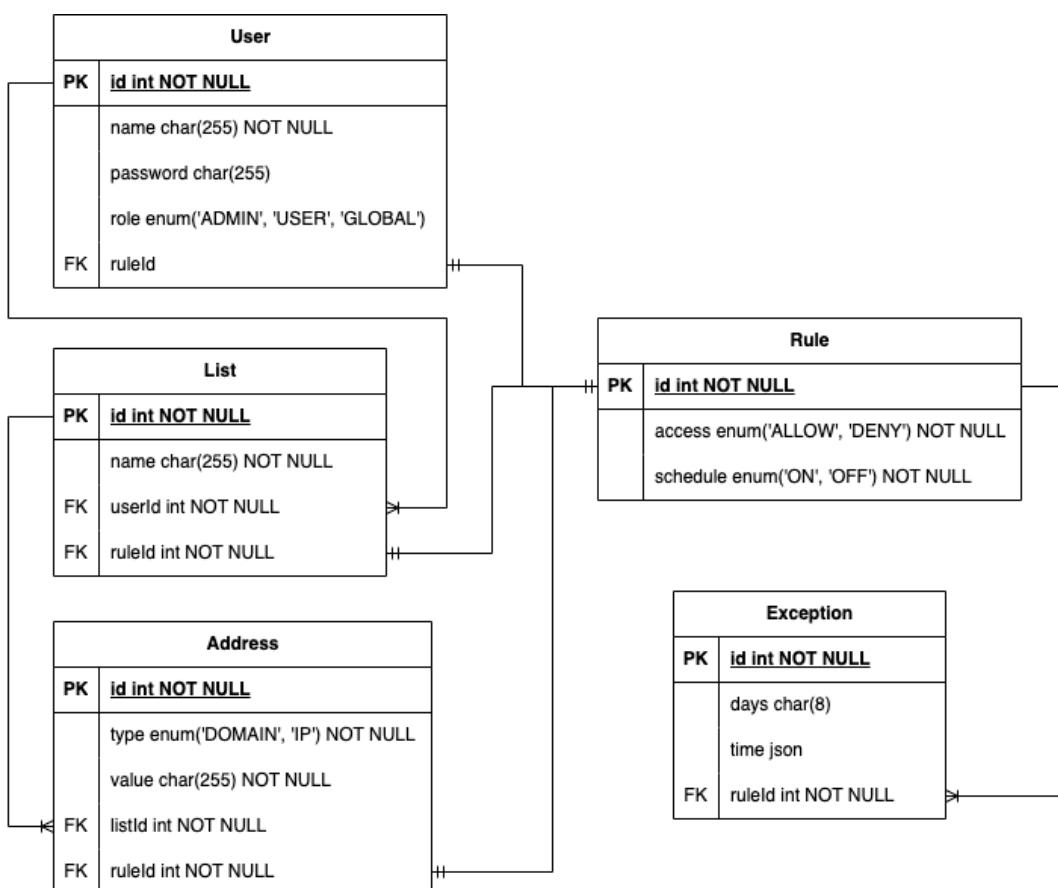


Рисунок 1. – Диаграмма модели данных

4 Разработка алгоритма построения конфигурации прокси

Чтобы информация и настройки, занесённые в базу данных, влияли на работу прокси-сервера, необходимо выполнить их обработку и преобразование в формат, соответствующий правилам оформления конфигурации Squid.

Было разработано три функции, вызывающиеся в каскаде, расположенные в модуле функций-помощников для класса-контроллера роутера прокси, предоставляющего конечные точки для управления параметрами и состоянием Squid:

- digestEntities – экспортруемая функция, реализующая начало и завершение обработки базы данных. Она выгружает записи пользователей и объекта глобального пользователя, соответствующего всем пользователям прокси, для последовательного формирования параметров конфигурации.
- digestUser – локальная функция модуля для обеспечения обработки соответствующих пользователю списков и адресов и формирования параметров конфигурации на основе выгруженных сведений из базы данных. Вызывается из функции digestEntities для каждого пользователя отдельно, а также для глобального пользователя.
- digestRule – локальная функция модуля для обработки сведений из таблиц правил (Rule) и исключений (Exception). В зависимости от параметра правила, определяющего применение исключений, формируются настройки конфигурации прокси с учётом существующих исключений или без.

Все функции возвращают объект с двумя строковыми массивами, содержащими списки контроля доступа (acl) и список опций, применяющих настройки доступа (http_access).

Далее приводится детальное описание выполняемых действий приведёнными функциями.

В начале вызова, функция digestEntities записывает первую строку в массив acl:

“acl auth proxy_auth REQUIRED”, – означающую, что авторизация обязательна для того, чтобы запрос прошёл через прокси к получателю. Слово «auth» будет означать всех авторизованных пользователей, что позволит применять глобальные настройки доступа. Затем, добавляет первую строку в массив http_access, что обеспечит применение данного правила. Далее, из базы данных выгружаются объект глобального пользователя, объект правила доступа глобального пользователя и массив объектов пользователей с ролью ‘USER’. После этого, в цикле выполняется обработка элементов массива пользователей.

Изм.	Лист	№ докум.	Подп.	Дата

Для каждого пользователя создаётся уникальное acl-имя на основе их идентификаторов в формате «иID», где ID – идентификационный номер, а «и» - символ, обозначающий имя пользователя. Формирование уникальных имён необходимо для применения индивидуальных параметров и устранения возможных коллизий имён пользователей с именами опций конфигурации прокси. После, в массив acl заносится запись, связывающая имя пользователя со сформированным acl-именем, которая имеет вид:

```
«acl acl_имя proy_auth имя_пользователя»
```

Из базы данных извлекается объект правила доступа пользователя. Затем, вызывается функция digestUser, в которую передаются идентификатор пользователя, имя пользователя, а также объекты правила доступа пользователя и глобального пользователя. По завершении выполнения функции digestUser, полученные в результате элементы добавляются к массивам acl и http_access. После цикла обработки пользователей, для объекта глобального пользователя вызывается функция digestUser, результаты выполнения которой в том числе добавляются в упомянутые массивы. Последней записью в массив http_access добавляется следующее:

«http_access deny all», – она означает, что доступ для запросов, не соответствующих условиям, сформированным выше, будет отклонён.

В функции digestUser объявляются локальные массивы acl и http_access. Из базы данных извлекаются все объекты списков пользователя. В цикле «for» выполняется обработка элементов полученного массива. Для каждого списка формируется уникальное имя в формате «lID», где первый символ латинская «l» (произносится “эль”), а ID – идентификационный номер списка. Из базы данных извлекается правило доступа списка и массив объектов адресов, относящихся к списку. Затем, в цикле происходит обработка элементов массива адресов. Для каждого адреса формируется уникальное имя в формате «aID», где символ латинского алфавита «a» означает имя адреса, а на месте ID указывается идентификатор. Затем, в локальный массив acl добавляется строка следующего вида:

«acl aID тип адрес», – где «тип» принимает значение «dst» для ip-адресов, либо «dstdomain» для доменных имён, а вместо «адрес» занимает непосредственно значение адреса. Следом, в массив заносится строка формата:

«acl lID any-of aID», – которая объединяет все адреса списка под одним именем списка, позволяя применять правила доступа для всего списка. Из базы данных извлекается объект правила доступа адреса. Для каждого объекта адреса вызывается функция digestRule, в которую передаются в качестве аргументов объект правила доступа

Изм.	Лист	№ докум.	Подп.	Дата	Лист	24
					BKR-НГТУ-09.03.01-(20-ПО)-05-2024	

адреса, а также строка с acl-именем адреса и acl-именем пользователя соответственно. Полученные в результате выполнения функции массивы добавляются к локальным массивам acl и http_access, с тем отличием, что в массив http_access результат будет добавлен только в том случае, если правило доступа адреса отличается от правила доступа списка. Такой подход помогает не добавлять избыточных параметров в конфигурацию. После завершения цикла обработки элементов массива адресов, в том случае, если в списке присутствовали адреса, для списка вызывается функция digestRule, в которую передаётся правило доступа списка и строка с acl-именем списка и acl-именем пользователя соответственно. Результат выполнения прибавляется к локальным массивам acl и http_access. Добавление в массив http_access выполняется только в том случае, если правило доступа списка отличается от правила доступа пользователя. По завершении цикла обработки списков, вызывается функция digestRule для пользователя, в которую передаётся правило доступа пользователя acl-имя пользователя. Полученный в результате выполнения массив acl прибавляется к локальному массиву acl, а полученный массив http_access прибавляется к локальному массиву функции digestUser только в случае, если правило доступа пользователя отличается от правила доступа глобального пользователя. Итоговые локальные массивы acl и http_access возвращаются в качестве результата выполнения функции.

Функция digestRule принимает в качестве аргументов объект правила (rule) и строку (acltail). Передаваемая строка необходима для добавления к итоговой строке параметра http_access необходимых acl-имён для корректного применения настройки. В теле функции также объявляются локальные массивы acl и http_access.

В том случае, если значение параметра применения исключений положительное, формируется общее acl-имя для исключений в формате «rID», где латинский символ «r» означает правило, а «ID» представляет идентификатор правила. Из базы данных извлекаются исключения, связанные с правилом, и обрабатываются в цикле. В локальный массив acl добавляются строки следующего формата:

«acl rID time дни время», – где «дни» занимает значение дней из объекта исключения, если оно представлено, а «время» оформляется, как диапазон значений времени вида «чч:мм-чч:мм», где первая пара меньше по значению, чем вторая, но так же, если значение времени представлено в объекте исключения. По завершении цикла обработки исключений, в локальный массив http_access добавляется строка следующего формата:

Изм.	Лист	№ докум.	Подп.	Дата	Лист	25
					VKP-НГТУ-09.03.01-(20-ПО)-05-2024	

«`http_access доступ !acl-имя acltail`», – где «доступ» принимает значение «allow» или «deny», в зависимости от соответствующего параметра правила, «`!acl-имя`» занимает сформированное acl-имя исключений с восклицательным знаком в начале, что означает, если HTTP-запрос, направленный через прокси, проходит по условиям исключения, то настройка применяться не будет, а «`acltail`» занимают переданные строкой acl-имена, по отношению к которым будет действовать настройка.

В противном случае, если значение параметра применения исключений отрицательное, исключения не выгружаются и сразу формируется строка для массива `http_access` без указания acl-имени исключений.

По выполнении функции, итоговые локальные массивы `acl` и `http_access` возвращаются в качестве выходных значений.

Функция `digestEntities` вызывается из метода класса-контроллера прокси `update`, где, при помощи встроенной асинхронной функции записи данных в файл `writeFile` платформы `node.js` осуществляется перенос строк из полученных массивов в конфигурационный файл, после чего, выполняется команда обновления конфигурации `squid` «`squid -k reconfigure`» при помощи встроенной функции «`exec`».

Блок-схемы рассмотренных функций приводятся в приложении Б.

Изм.	Лист	№ докум.	Подп.	Дата	Лист	26
					<i>BKR-НГТУ-09.03.01-(20-ПО)-05-2024</i>	

5 Разработка серверной части

Серверная часть представляет собой центральное звено проекта, связывающее компоненты в одну единую, функционирующую систему. Она предоставляет внешний интерфейс для обмена данными и выполнения доступных операций; отвечает за авторизацию пользователей, проверку прав доступа; управление моделью данных и взаимодействие с базой данных; создание конфигурации прокси-сервера и управление его состоянием.

Основу серверной части составляет платформа node.js версии 20 в редакции для долгосрочной поддержки. Для расширения функциональных возможностей применяется ряд дополнительных пакетов:

- express – предоставляет функции, обеспечивающие работу сервера;
- bcrypt – применяется с целью безопасного хранения паролей благодаря генерации хеша из значения пароля;
- cors – добавлен для расширения заголовков HTTP, что позволяет запрашивать данные с доменов, отличных от домена веб-страницы;
- dotenv – добавляет возможности создания файла «.env» в корневой директории проекта для хранения константных значений и обращения к ним из любого вложенного файла;
- jsonwebtoken – используется для создания и проверки токенов, необходимых для аутентификации пользователей;
- sequelize – ORM для взаимодействия с СУБД средствами языка программирования. Позволяет описать модель данных и обращаться к ней без необходимости писать SQL-запросы;
- pg – JavaScript клиент для доступа к PostgreSQL. Необходим для работы пакета sequelize с СУБД PostgreSQL;
- Pg-hstore – пакет для сериализации и десериализации данных формата JSON в формат hstore. Необходим для работы sequelize с СУБД PostgreSQL.

Файловая структура серверной части имеет следующую схему:

- Файл index.js – основной файл серверного приложения. Внутри него подключаются внешние и сторонние модули; определяется экземпляр класса express для обеспечения серверного функционала, задаётся корневой путь к конечным точкам внешнего интерфейса; выполняется запуск сервера;

Изм.	Лист	№ докум.	Подп.	Дата	Лист	27
					BKR-НГТУ-09.03.01-(20-ПО)-05-2024	

- Файл db.js – внутри объявляется экспортруемый метод создания класса sequelize для подключения к СУБД и создания модели данных;
- Файл .env – служит для хранения константных значений, таких, как сведения для доступа к СУБД, доменное имя сервера, номер порта;
- Файл utils/initFunc.js – файл содержит экспортруемую функцию для инициализации базы данных. При её вызове, старые значения будут стёрты. Вызывается при первом старте сервера и при сбросе настроек;
- Каталог ssl – содержит сертификат и приватный ключ сервера для обеспечения защищённого соединения с сервером;
- Файл models/models.js – внутри описывается модель данных при помощи средств sequelize и экспортруются сущности для доступа к таблицам базы данных;
- Каталог routes – содержит файлы, в которых описываются маршруты к конечным точкам внешнего интерфейса сервера. Для каждой конечной точки применяются специальные методы обработки данных входящих и исходящих запросов.
- Каталог controllers – содержит реализации классов контроллеров, методы которых отвечают за обработку запросов HTTP;
- Каталог helpers – предоставляет экспортруемые модули, отвечающие за взаимодействие с базой данных, подготовку и обработку данных, получаемых или извлекаемых из неё;
- Каталог middlewares – хранит функции промежуточной обработки входящих запросов HTTP;
- Файл error/ApiError.js – экспортрует класс, наследующий класс Error (ошибка). Содержит статические методы для удобного объявления типовых ошибок взаимодействия с сервером;
- Каталог node_modules – содержит все необходимые компоненты для работы платформы node.js, а также файлы сторонних модулей и библиотек;
- Файл package.json – документ, описывающий проект, его требования и дополнительные настройки;
- Файл package-lock.json – автоматически генерируемый файл, отражающий изменения от любых операций, влияющих на дерево модулей node_modules или package.json.

Изм.	Лист	№ докум.	Подп.	Дата

Для реализации маршрутов на сервере, в Express используются роутеры – экземпляры класса Router. Данные объекты имеют в своём составе методы обработки HTTP запросов с соответствующими именами. В проекте реализовано пять роутеров, обеспечивающих сервер конечными точками, формирующими внешний интерфейс, следующий принципам архитектурного подхода REST API:

- userRouter – роутер, отвечающий за предоставление конечных точек для управления данными сущности пользователя модели данных;
- listRouter – создаёт конечные точки для взаимодействия с сущностью списка и связанными данными;
- addressRouter – предоставляет конечные точки для управления данными сущности адреса модели данных;
- ruleRouter – роутер, принимающий запросы к сущности правила модели данных и связанным объектам сущности исключения;
- proxyRouter – предоставляет конечные точки для управления прокси-сервером.

Данные роутеры располагаются в каталоге routes.

Самое большое количество конечных точек определено в роутере userRouter:

- /users – реализует CRUD-операции при взаимодействии с сущностью User. Имеет четыре метода:
 - 1) post – создаёт нового пользователя. Выполняется проверка на роль администратора. Ожидает три параметра в теле запроса: имя пользователя, пароль и необязательную роль. Возвращает объект с идентификатором нового пользователя, именем и идентификатором правила доступа.
 - 2) get – возвращает список пользователей с ролью «пользователь» ('USER'). Выполняется проверка на роль администратора.
 - 3) put – обновляет учётные данные существующего пользователя. Выполняется проверка на авторизацию. Ожидает три параметра в теле запроса: идентификатор пользователя, имя и пароль. Возвращает объект с идентификатором обновлённого пользователя, именем, ролью и идентификатором правила доступа.
 - 4) delete – удаляет пользователя, правило доступа пользователя и связанные исключения, а также списки пользователя, их правила доступа и все связанные исключения. Выполняется проверка на роль администратора. Ожидает идентификатор пользователя в параметрах запроса. При успехе, возвращает код статуса 200.

Изм.	Лист	№ докум.	Подп.	Дата

- /users/login – осуществляет авторизацию пользователей. Отвечает на метод post. Возвращает токен пользователя.
- /users/auth – отвечает за проверку и обновление аутентификации пользователя. Выполняется проверка запроса на авторизацию пользователя по токену. При успехе, возвращает обновлённый токен. Ожидает метод get.
- /users/exist – возвращает ответ булевого значения, существует ли пользователь по запрошенному имени. Выполняется проверка на роль администратора. Ожидает имя пользователя в параметрах запроса. Отвечает на метод get.
- /users/global – возвращает объект с идентификатором глобального пользователя и идентификатором правила доступа. Выполняется проверка на роль администратора. Отвечает на метод get.

Двигаясь по иерархии сущностей вниз, ниже приводятся конечные точки роутера listRouter:

- /lists – реализует CRUD-операции при взаимодействии с сущностью List. Предоставляет пять методов:
 - 1) post – создаёт один или несколько списков. Выполняется проверка на роль администратора. Ожидает два параметра в теле запроса: строковый массив названий новых списков и идентификатор пользователя. Возвращает объект с массивом всех списков пользователя и опционально строковый массив не сохранённых названий.
 - 2) get – возвращает массив всех списков пользователя. Выполняется проверка на роль администратора. Ожидает идентификатор пользователя в параметрах запроса.
 - 3) put – обновляет название списка, позволяет добавлять и удалять объекты адресов, поскольку они привязываются к конкретному списку. Выполняется проверка на роль администратора. Ожидает в теле запроса идентификатор списка и три опциональных параметра: новое название списка, массив новых объектов адресов для добавления и массив существующих адресов для удаления. Возвращает объект с массивом всех адресов списка, а также, в зависимости от переданных параметров, может включать обновленный объект списка и элементы из предоставленных массивов, которые не были обработаны.

Изм.	Лист	№ докум.	Подп.	Дата	Лист
					ВКР-НГТУ-09.03.01-(20-ПО)-05-2024 30

- 4) patch – обновляет название списка. Выполняется проверка на роль администратора. Ожидает два параметра в теле запроса: идентификатор списка и новое название. Возвращает объект обновлённого списка.
- 5) delete – удаляет список, правило доступа списка и связанные исключения, а также адреса списка, правила доступа списков и все связанные исключения. Выполняется проверка на роль администратора. Ожидает идентификатор списка в параметрах запроса. При успехе, возвращает код статуса 200.
- /lists/exist – возвращает ответ булевого значения, существует ли список по запрошенному названию у конкретного пользователя. Выполняется проверка на роль администратора. Ожидает два параметра в параметрах запроса: название списка и идентификатор пользователя. Отвечает на метод get.

Далее, рассмотрены конечные точки для роутера addressRouter:

- /addresses – реализует CRUD-операции при взаимодействии с сущностью Address. Отвечает на 5 методов:
 - 1) post – создаёт один или несколько объектов адреса. Выполняется проверка на роль администратора. Ожидает два параметра в теле запроса: массив объектов адреса и идентификатор списка. Возвращает объект с созданными объектами адреса и опционально массив объектов, не добавленных в базу данных.
 - 2) get – возвращает все адреса указанного списка. Выполняется проверка на роль администратора. Ожидает идентификатор списка в параметрах запроса.
 - 3) patch – обновляет существующий адрес. Выполняется проверка на роль администратора. Ожидает три параметра в теле запроса: идентификатор адреса, тип нового адреса и значение нового адреса. Возвращает объект обновленного адреса.
 - 4) put – позволяет добавить или убрать несколько объектов адреса. Выполняется проверка на роль администратора. Ожидает три параметра в теле запроса: идентификатор списка, массив объектов новых адресов на добавление и массив объектов существующих адресов на удаление. Возвращает объект с массивом всех адресов списка и опционально массивы не принятых к обработке объектов.
 - 5) delete – удаляет объект адреса, а также объект правила доступа адреса и связанные исключения. Выполняется проверка на роль администратора.

Изм.	Лист	№ докум.	Подп.	Дата	Лист
					ВКР-НГТУ-09.03.01-(20-ПО)-05-2024

Ожидает идентификатор адреса в параметрах запроса. При успехе, возвращает ответ с кодом статуса 200.

- /addresses/exist – возвращает результат булевого типа, при запросе на проверку наличия указанного адреса. Выполняется проверка на роль администратора. Ожидает три параметра в параметрах запроса: идентификатор списка, тип и значение адреса.

Описание конечных точек роутера ruleRouter:

- /rules – реализует CRUD-операции при взаимодействии с сущностью Rule, а также позволяет управлять связанными объектами сущности Exception. Отвечает на 4 метода:
 - 1) post – создаёт правило доступа. Выполняется проверка на роль администратора. Ожидает значение доступа в теле запроса. Возвращает объект созданного правила.
 - 2) get – возвращает правило и набор связанных исключений. Выполняется проверка на роль администратора. Ожидает идентификатор правила в параметрах запроса.
 - 3) put – отвечает за обновление значения доступа правила, состояния действия исключений, а также за добавление, обновление и удаление исключений. Выполняется проверка на роль администратора. Ожидает на входе 6 параметров в теле запроса: идентификатор правила, значение доступа, состояние действия исключений, массив новых исключений на добавление, массив изменённых исключений на обновление и массив существующих исключений на удаление. Возвращает набор исключений правила, а также, дополнительно, объект обновлённого правила и массивы не обработанных объектов исключений.
 - 4) delete – удаляет правило и связанные исключения. Выполняется проверка на роль администратора. Ожидает на входе идентификатор правила в параметрах запроса. При успехе, возвращает ответ с кодом статуса 200.
- /rules/all – возвращает список всех правил доступа. Выполняется проверка на роль администратора. Отвечает на метод get.

И последний роутер в лице proxyRouter:

- /proxy – принимает запросы для управления состоянием прокси-сервера. Отвечает на 4 метода:

Изм.	Лист	№ докум.	Подп.	Дата

- 1) post – включает или выключает прокси-сервер. Выполняется проверка на роль администратора. Возвращает новое состояние прокси-сервера.
 - 2) get – возвращает статус прокси-сервера, доменное имя и порт. Выполняется проверка на авторизацию.
 - 3) put – обновляет конфигурацию прокси-сервера. Выполняется проверка на роль администратора.
 - 4) delete – сбрасывает систему до заводских настроек, очищает базу данных и удаляет настройки прокси-сервера. Выполняется проверка на роль администратора.

– /proxy/cert – возвращает файл сертификата прокси-сервера. Выполняется проверка на авторизацию. Отвечает на метод get.

Функции обработки HTTP запросов реализованы в методах классов, объявленных в файлах каталога controllers. К данным классам относятся:

- UserController – класс, методы которого отвечают за обработку запросов на конечные точки роутера userRouter;
 - ListController – класс, методы которого предназначаются для обработки запросов к конечным точкам роутера listRouter;
 - AddressController – класс, методы которого применяются с целью обработки запросов, направленных на конечные точки роутера addressRouter;
 - RuleController – класс, методы которого отвечают за обработку запросов, присылаемых на роутер ruleRouter;
 - ProxyController – класс, методы которого обеспечивают управление прокси-сервером и вызываются при получении запросов роутером proxyRouter.

В таблице 2 далее приводятся соответствия методов классов контроллеров и конечных точек серверной части.

Таблица 2. – Конечные точки

Конечная точка	HTTP-метод	Условие выполнения	Класс контроллера	Метод контроллера
/users	post	Роль администратора	UserController	create
	get	Роль администратора		read
	put	Пользователь авторизован		update
	delete	Роль администратора		delete

Продолжение таблицы 2

Конечная точка	HTTP метод	Условие выполнения	Класс контроллера	Метод контроллера
/users/login	post	Нет		login
/users/auth	get	Пользователь авторизован		auth
/users/exist	get	Роль администратора		isUser
/users/global	get	Роль администратора		readGlobal
/lists	post	Роль администратора	ListController	create
	get			read
	put			update
	patch			patch
	delete			delete
/lists/exist	get			isList
/addresses	post	Роль администратора	AddressController	create
	get			read
	patch			patch
	put			update
	delete			delete
/addresses/exist	get			isAddress
/rules	post	Роль администратора	RuleController	create
	get			read
	put			update
	delete			delete
/rules/all	get			readAll
/proxy	post	Роль администратора	ProxyController	toggle
	get	Пользователь авторизован		get
	put	Роль администратора		update
	delete	Роль администратора		reset
/proxy/cert	get	Пользователь авторизован		downloadCert

В каталоге helpers располагаются модули с функциями, обеспечивающими взаимодействие с базой данных. Экспортируемые функции независимы друг от друга и могут быть вызваны тогда, когда в них возникает потребность. В директории присутствует шесть модулей, пять из которых соответствуют сущностям модели данных:

- userHelpers – модуль содержит функции взаимодействия с данными пользователя. Экспортируются следующие функции:
 - 1) createOne – создать пользователя и правило доступа пользователя;
 - 2) getUsers – получить список пользователей с ролью пользователя;
 - 3) updateOne – обновить пользователя;
 - 4) deleteOne – удалить пользователя, правило доступа пользователя, списки пользователя, правила доступа списков, адреса списков, правила доступа адресов и все связанные исключения правил;
 - 5) getGlobal – получить глобального пользователя;
 - 6) login – выполнить авторизацию пользователя;
 - 7) renewToken – выполнить проверку авторизации и выдать новый токен;
 - 8) isExist – проверить, существует ли пользователь с указанным именем.
- listHelpers – модуль с функциями для обработки данных списка. Экспортируются функции:
 - 1) create – создать списки пользователя и правила доступа списков;
 - 2) getAll – получить все списки пользователя;
 - 3) updateName – обновить название списка;
 - 4) deleteOne – удалить список, правило доступа списка, а также адреса списка, правила доступа адресов списка и все связанные исключения;
 - 5) isExist – проверить, существует ли список с указанным названием у конкретного пользователя;
- addressHelpers – модуль с функциями, предназначенными для обработки данных адресов. Экспортируемые функции:
 - 1) create – создать адреса и правила доступа адресов для конкретного списка;
 - 2) getAll – получить набор адресов для одного списка;
 - 3) updateOne – обновить адрес;
 - 4) deleteOne – удалить адрес, правило доступа адреса и связанные исключения;
 - 5) isExist – проверить, присутствует ли указанный адрес в конкретном списке.
- ruleHelpers – модуль с функциями для взаимодействия с данными правила. Экспортируемые функции:

Изм.	Лист	№ докум.	Подп.	Дата	Лист	35
					BKR-НГТУ-09.03.01-(20-ПО)-05-2024	

- 1) createOne – создать правило;
- 2) getOne – получить правило;
- 3) updateOne – обновить правило;
- 4) deleteOne – удалить правило и все связанные исключения;
- 5) getAll – получить все правила.
- exceptionHelpers – модуль с функциями для обращения с данными исключений. Экспортируются функции:
 - 1) create – создать одно или несколько исключений для одного правила;
 - 2) getAll – получить все исключения для конкретного правила;
 - 3) updateOne – обновить исключение;
 - 4) update – обновить одно или несколько исключений для одного правила;
 - 5) deleteOne – удалить исключение;
 - 6) delete – удалить одно или несколько исключений для одного правила.
- proxyHelpers – модуль, содержащий функции для обработки таблиц базы данных и составления параметров конфигурации прокси. Экспортируется функция digestEntities, не имеющая аргументов и возвращающая два строковых массива, значения которых впоследствии записываются в конфигурационный файл.

Перечисленные функции-помощники напрямую взаимодействуют с базой данных через классы сущностей модели данных, импортируемых из файла models.js, и вызываются из методов классов-контроллеров. В свою очередь, методы контроллеров взаимодействуют с базой данных только посредством функций-помощников.

Изм.	Лист	№ докум.	Подп.	Дата

6 Разработка клиентской части

Клиентская часть, так же, как и серверная часть, основывается на платформе «node.js», позволяющей разрабатывать web-приложения на высокоуровневом языке программирования JavaScript. Существует множество средств реализации графического интерфейса, и, хотя платформа node.js предоставляет широкие возможности разработки при помощи JavaScript за счёт встроенных библиотек, сторонние библиотеки и фреймворки могут предоставить больше гибкости или более простые и удобные подходы в решении задач. В тройку самых востребованных и используемых средств разработки графических веб-интерфейсов, согласно результатам выдачи поисковых сервисов, входят React, Angular и Vue.js. Доминирующее место занимает библиотека React, которая также лидирует по количеству представленных вакансий на рынке труда в России, в сфере фронтенд-разработки.

React представляет собой инструментарий для реализации и использования компонентов пользовательского интерфейса, а также программирования логики клиентских сервисов. Подход вынесения логических операций за пределы серверной части подвергается некоторой критике, однако у этого подхода достаточно преимуществ, а потери в производительности компенсируются высокими мощностями современных, персональных вычислительных устройств. Более того, такой подход позволяет снять возрастающую нагрузку с серверов и создавать автономные, не требующие постоянного подключения к серверу, приложения. В React присутствует поддержка набирающих популярность PWA – при подготовке некоторых настроек и формированию особого манифеста – веб-приложений, адаптируемых под смартфоны, с внешними атрибутами и функциональными чертами, очень близкими к нативным реализациям.

Помимо библиотеки React, для упрощения и ускорения разработки был выбран фреймворк Bootstrap, а именно, специально собранную под данную библиотеку версию – React-Bootstrap. Этот фреймворк представляет собой коллекцию компонентов, обеспечивающих построение композиций элементов интерфейса, способов взаимодействия, при поддержке адаптивной перестройки под различные экраны устройств, с достаточной палитрой тем и цветовых решений. Разработчики заявляют, что он ориентирован в первую очередь на мобильные устройства. Именно это утверждение стало наиболее значимым при принятии решения включить программный пакет в проект.

По той причине, что преобладающее количество времени пользования персональными компьютерными устройствами относится к мобильным устройствам, а

Изм.	Лист	№ докум.	Подп.	Дата	Лист	37
					VKR-НГТУ-09.03.01-(20-ПО)-05-2024	

смартфоны стали основным порталом в цифровые пространства социальных коммуникаций, приоритет в разработке был отдан созданию веб-интерфейса, наилучшим образом, воспринимающимся с небольших экранов. Так, пользователи, обладающие доступом к административным возможностям сервиса, смогут легко подключиться к клиентской части для выполнения необходимых настроек.

Однако, в ходе разработки, возникло множество трудностей, при взаимодействии с фреймворком React-Bootstrap. Процессы модификации и персонализации компонентов связаны с рядом сложностей. В документации фреймворка React-Bootstrap ощущается недостаток информации, описаний и примеров. Попытки изменения форм и цвета на нестандартные сопряжены с трудностями поиска оптимальных решений для реализации поставленных задач. Так, вместо привычного и хорошо задокументированного формального языка стилей CSS, предлагается использовать метаязык Sass, требующий собственного компилятора, переводящего код Sass в CSS, тем самым, возникает дополнительная прослойка, не оказывающая существенным преимуществом, исходя из субъективного опыта. Также, в документации присутствовало недостаточно примеров использования инструмента и исчерпывающих пояснений.

Дополнительно, на мобильных устройствах, а именно, на смартфонах под управлением операционной системы iOS, происходят т.н. графические баги, при определённых типах анимации. Так, была обнаружена проблема зависания анимации, ориентированной на возникновение при наведении указателя мыши на элемент. Помимо мобильной платформы компании Apple, подобная проблема может встречаться, в том числе, на смартфонах под управлением операционной системы Android. Браузер Safari под системой iOS эмулирует эффект наведения указателя, при нажатии на соответствующие элементы, к примеру, кнопки. Найти объяснения данному поведению программной среды Apple не удалось, однако, при разработке компонентов интерфейса из базовых элементов HTML и каскадных таблиц CSS, данная ошибка исправляется помещением стилей, описывающих такое поведение анимации, в специальный блок кода, благодаря чему, описанные внутри стили будут применяться только на устройствах, полноценно поддерживающих наведение курсора. А поскольку смартфоны к таким устройствам не относятся, проблема полностью исчезает.

Резюмируя полученный опыт от взаимодействия с данным фреймворком, было решено, что большую часть компонентов следует разработать самостоятельно, используя базовые элементы HTML и определять стилистику с анимациями при помощи простых и понятных каскадных таблиц стилей CSS.

Изм.	Лист	№ докум.	Подп.	Дата	Лист	38
					VKR-НГТУ-09.03.01-(20-ПО)-05-2024	

6.1 Файловая структура клиентской части

Начальной точкой служит файл index.js, внутри которого импортируется пакет ReactDOM, позволяющий управлять DOM-элементами, имеющимися на странице. Его метод createRoot() возвращает объект типа ReactDOM.Root, который создаёт корневой элемент для отображения компонентов React внутри DOM-дерева браузера. Метод render() данного объекта инициирует отрисовку приложения. Внутрь него помещается основной компонент приложения, который также импортируется в текущий файл.

В функциональной парадигме написания JavaScript кода, которая, на текущий момент, является основной, наиболее предпочтительной, удобной и рекомендуемой разработчиками, первичный компонент приложения, обычно, именуемый App, является функцией, возвращающей пользовательские элементы интерфейса. В данном сервисе используется одностраничная модель интерфейса, поэтому этот компонент возвращает компонент BrowserRouter, импортированный из пакета react-router-dom, который отвечает за настройку маршрутов в приложении. Внутрь него вложен компонент AuthProvider, предоставляющий доступ к вспомогательным функциям, необходимым при авторизации и выходе пользователя, а также сведениям авторизованного пользователя, из вложенных в него компонентов. В свою очередь, в AuthProvider помещается компонент AppRouter, отвечающий за маршруты в клиентском сервисе. Компонент располагается в файле по пути «src/App.jsx».

Помимо упомянутых файлов, клиентская часть включает следующие элементы:

- каталог public – содержит документ index.html, внутрь которого React встроит разработанные, подключаемые компоненты, также манифест manifest.json, описывающий параметры для PWA-приложения, и иконки приложения.
- каталог src – здесь содержатся все основные файлы разработки, классы, компоненты, utilitites, стили и прочее:
 - 1) каталог components – содержит компоненты, из которых строится графический интерфейс веб-приложения;
 - 2) каталог modals – содержит компоненты, представляющие собой модальные окна, которые всплывают поверх основных страниц интерфейса;
 - 3) каталог pages – в нём располагаются компоненты страниц интерфейса;
 - 4) каталог css – хранит файлы CSS стилей компонентов страниц;
 - 5) каталог hooks – содержит так называемые хуки, представляющие собой функции, обеспечивающие компоненты состояниями и методами жизненного цикла;

Изм.	Лист	№ докум.	Подп.	Дата

- 6) каталог png – хранит изображения формата png;
 - 7) каталог svg – хранит векторные изображения формата svg;
 - 8) файл utils/constants.js – содержит константные значения путей в клиентском интерфейсе;
 - 9) каталог http – хранит файлы, содержащие функции взаимодействия с сервером;
 - 10) файл App.css – файл CSS стилей, общих для всего веб-приложения;
 - 11) файл App.jsx
 - 12) файл routes.js – содержит экспортруемые массивы объектов, представляющих собой описание маршрутов для различных уровней доступа;
 - 13) файл index.js
- файл .env – позволяет хранить константные значения и обращаться к ним из любого места проекта.
 - package.json – как в случае с серверной частью, этот документ описывает проект, его требования и прочие настройки.

6.2 Компоненты клиентского сервиса

Всего в клиентской части реализовано 7 статических маршрутов:

- /auth – страница авторизации (рисунок 2). Открывается у неавторизованных пользователей;

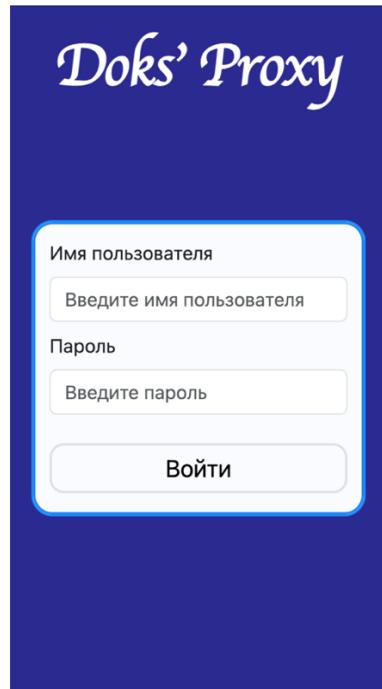


Рисунок 2. – Страница авторизации

Изм.	Лист	№ докум.	Подп.	Дата

- /home – домашняя страница веб-приложения (рисунок 3). Первая из трёх первичных страниц, с которых начинается навигация по приложению. Доступна авторизованным пользователям для наблюдения статуса работы прокси-сервера, получения сведений, необходимых для подключения, установки SSL/TLS сертификата. Для администраторов доступно обновление конфигурации прокси-сервера, включение и выключение сервиса. При ручном указании пути в URL, после доменного имени, будет выполнена навигация на текущую страницу, для авторизованных пользователей. Ручная навигация по сайту не предусмотрена и предотвращается.
 - /settings – страница настроек веб-сервиса (рисунок 3). Доступна авторизованным пользователям. Позволяет просмотреть имя пользователя, обновить логин и пароль, а также выполнить выход. Вторая стартовая страница навигации.
 - /manage – страница управления правилами доступа и пользователями (рисунок 3). Это ядро клиентской части, откуда происходят все настройки прокси-сервера. Является третьей стартовой страницей и отправной точкой при навигации по интерфейсу. Доступна только администраторам, как и все последующие.

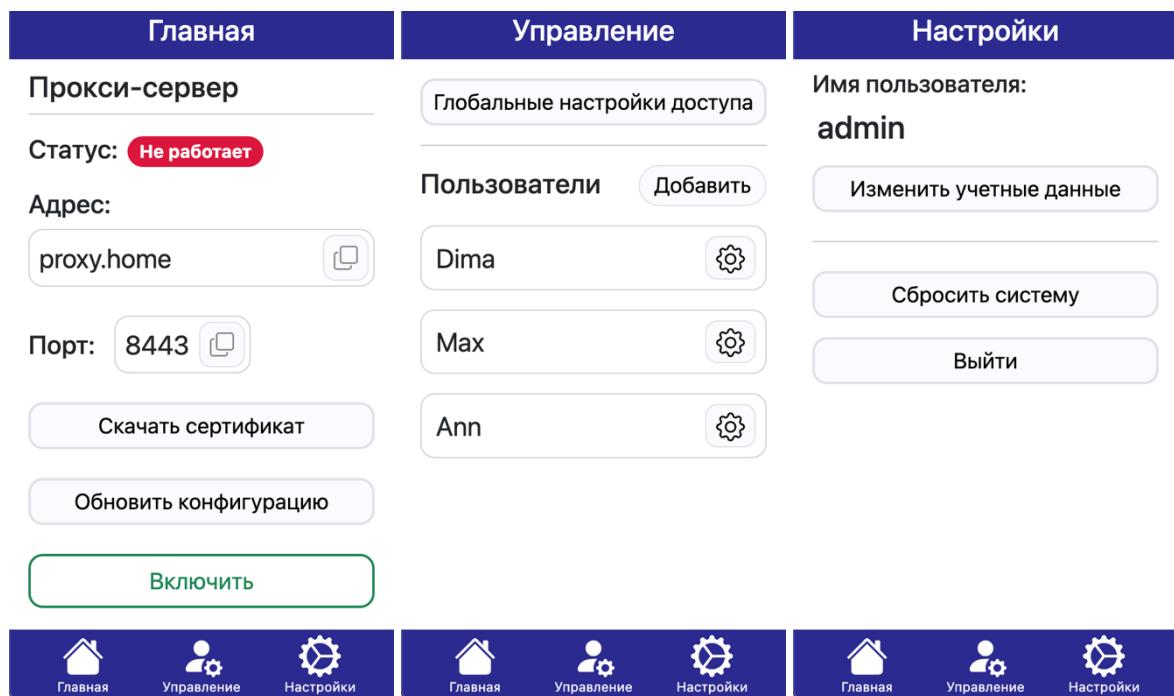


Рисунок 3. – Главная, Управление, Настройки

- /user – страница настроек пользователя (рисунок 4). Открывается при переходе как в настройки пользователей, так и в глобальные настройки доступа, применяемые для всех пользователей.

The screenshot shows two side-by-side panels. The left panel is titled 'Настройки пользователя' (User Settings) and shows a user profile for 'Dima'. It includes buttons for 'Изменить учетные данные' (Change account data) and 'Удалить пользователя' (Delete user). Below this is a section titled 'Правила доступа' (Access rules) with a button for 'Общее правило доступа' (General access rule). Further down is a section titled 'Списки правил:' (List of rules:) with a 'Добавить' (Add) button, followed by a button for 'Адреса TikTok' (TikTok addresses). The right panel is titled 'Глобальные настройки доступа' (Global access settings) and includes sections for 'Правила доступа' (Access rules) with a 'Общее правило доступа' (General access rule) button, 'Списки правил:' (List of rules:) with a 'Добавить' (Add) button, and categories like 'Социальные сети' (Social networks), 'Видео сайты' (Video sites), and 'Недопустимое' (Prohibited).

Рисунок 4. – Настройки пользователя, Глобальные настройки

- /list – страница списка адресов (рисунок 5). Отличительная особенность текущего проекта. Современные сайты и интернет-сервисы могут использовать различные доменные имена для обеспечения своего функционала, вследствие чего, подключаясь к одному адресу, запрос может быть перенаправлен на другой адрес, например, для TLS-рукопожатия, либо, с целью распределения нагрузки на сервера или по другой причине. Разные адреса могут применяться для различных версий одного сервиса, например, для мобильного приложения и веб-сайта. Найти домены, принадлежащие определённому ресурсу, не является проблемой, отправив соответствующий запрос в поисковую систему. Если добавлять адреса в один длинный список, будет трудно ориентироваться в настройках, и, при необходимости, точечно задавать правила доступа конкретным ресурсам. Данную проблему было решено нивелировать подходом группировки адресов по категориям, определяемых пользователем. Так, можно задать категорией популярное название интернет ресурса, или объединить несколько под один список, к примеру, социальные сети, видео-хостинги и т.д. На странице списка адресов можно задавать как общее правило для всего

Изм.	Лист	№ докум.	Подп.	Дата

списка, так и правила для конкретных адресов, которые переопределят правило списка, при их назначении.

- /rule – страница настройки правила доступа (рисунок 5). Переиспользуется, как для общих правил доступа – глобального правила, правил списков и пользователей, так и для правил по отношению к отдельным сетевым адресам. Для последних, включает возможность изменения значения адреса. Доступны две опции определения правила доступа: разрешить доступ к ресурсу и запретить. Помимо определения доступа, присутствует возможность применения исключений по заданному расписанию. В силу особенностей конфигурации Squid Proxy, диапазон времени устанавливается таким образом, чтобы начальное значение диапазона было меньше конечного.

Список правил

◀ назад

Адреса TikTok

Правило доступа списка

Список адресов: Добавить

www.tiktok.com

www.tiktok.de

100.50.0.230

Правило доступа

◀ назад

Адрес:
www.tiktok.com

Изменить адрес

Доступ: Разрешить Запретить

Расписание:

Доступ: Запретить

Дни: Пн Вт Ср Чт Пт Сб Вс

Время: 00:00 – 23:59

Добавить

Правило доступа списка

◀ назад

Доступ: Разрешить Запретить

Расписание:

Доступ: Запретить

Дни: Пн Вт Ср Чт Пт Сб Вс

Время: 00:00 – 23:59

Добавить

Рисунок 5. – Список. Правило доступа адреса, Общее правило доступа

Каждому маршруту соответствует одноимённый, разработанный компонент React, описывающий интерфейс страницы.

Помимо компонентов страниц, были реализованы компоненты всплывающих окон:

- **UserModal** – компонент, отвечающий за создание пользователей и обновление учетных данных (рисунок 6). Применяется на страницах управления и настроек.

<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подп.</i>

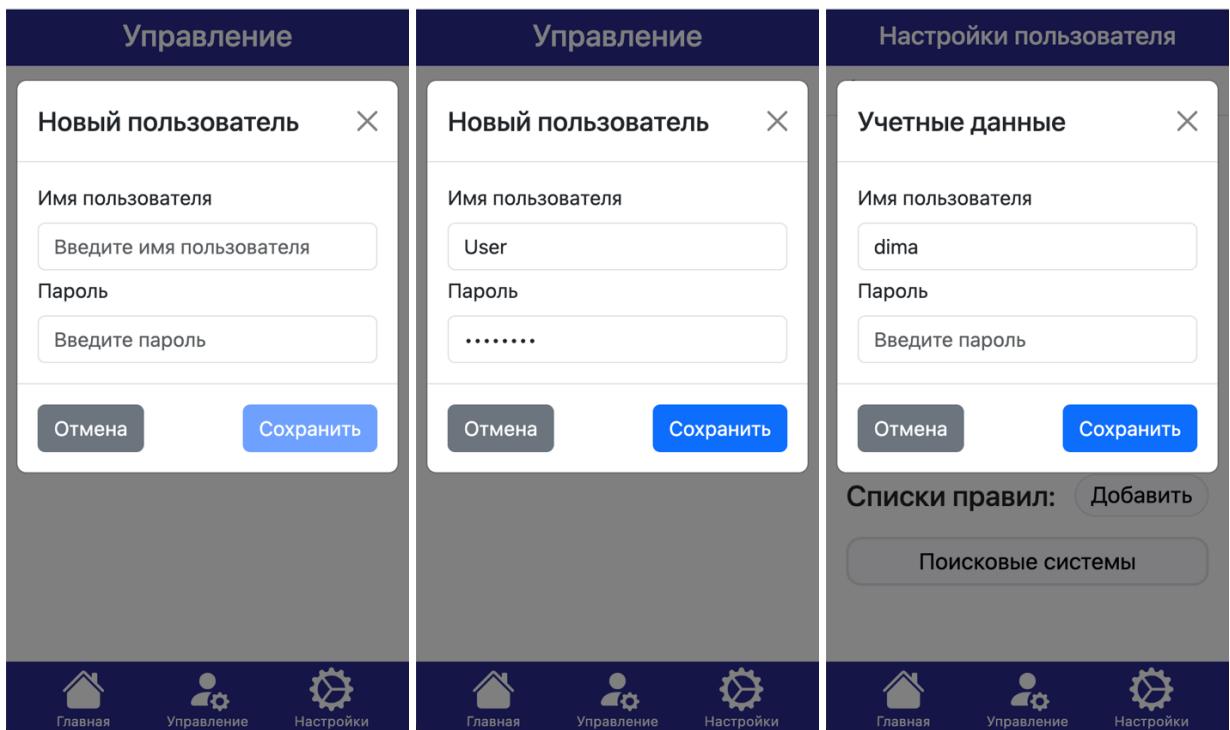


Рисунок 6. – Всплывающее окно UserModal

- ListModal – отвечает за добавление новых списков и переименование существующих (рисунок 7). Используется на страницах настроек пользователя и списков адресов.

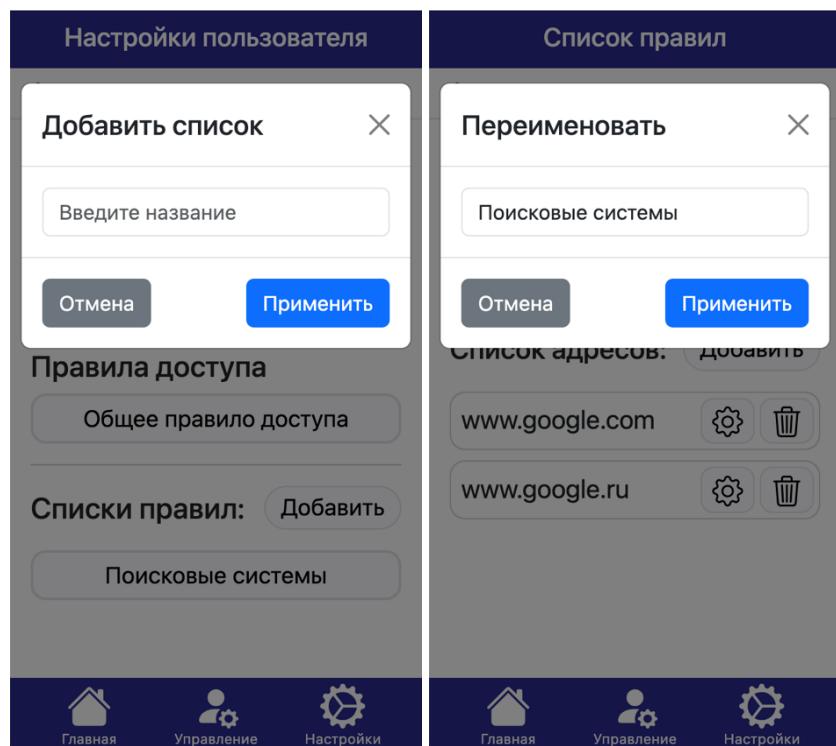


Рисунок 7. – Всплывающее окно ListModal

Изм.	Лист	№ докум.	Подп.	Дата

- AddressModal – предназначается для добавления новых адресов и изменения существующих (рисунок 8). Применяется на страницах списков адресов и настроек правил доступа адресов.

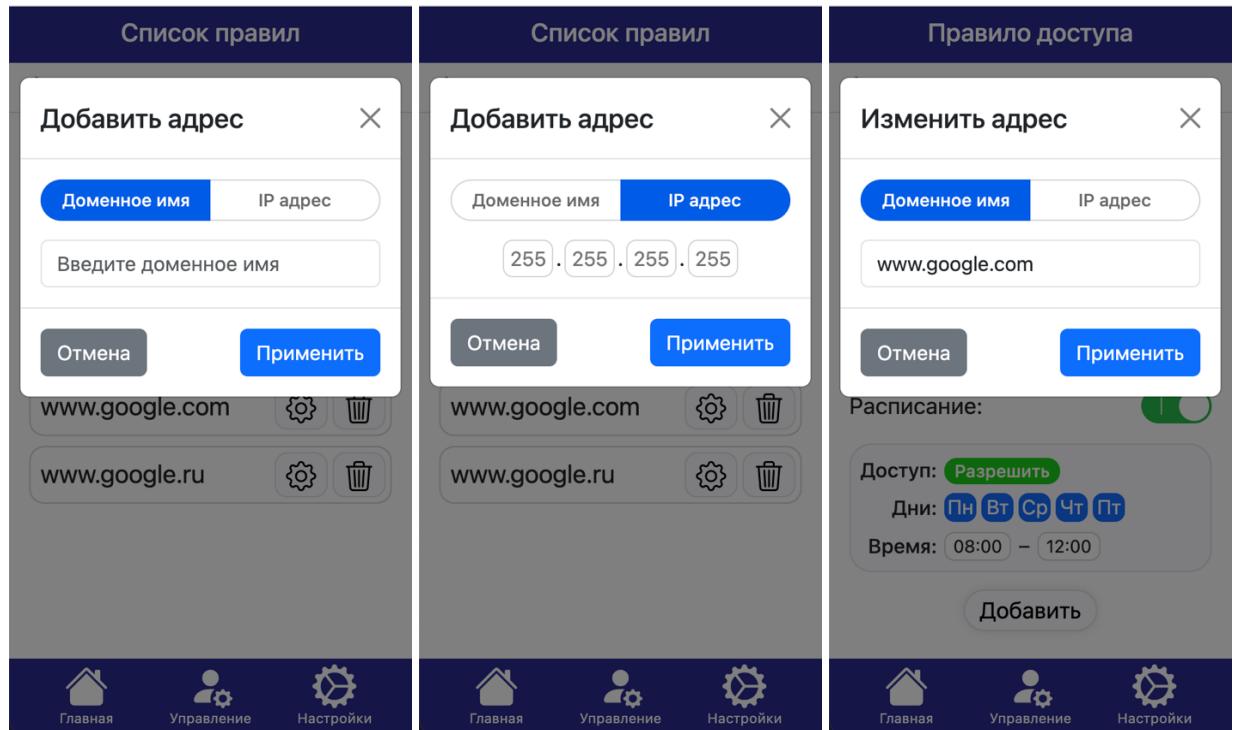


Рисунок 8. – Всплывающее окно AddressModal

- SheetModal – всплывающее окно, появляющееся при добавлении и изменении исключений правил доступа (рисунок 9). Используется на страницах настроек правил доступа.

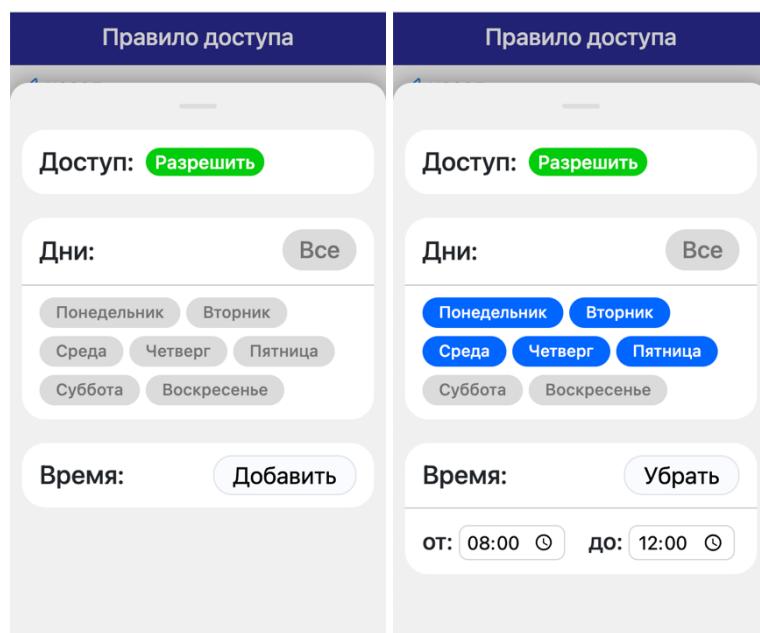


Рисунок 9. – Всплывающее окно SheetModal

Для построения веб-интерфейса, были созданы компоненты, служащие составными частями страниц, с возможностью их многоократного использования в различных местах приложения, по необходимости. Ниже приводится список используемых компонентов:

- BottomBar – нижний, фиксированный, навигационный бар. Отвечает за перемещение между такими маршрутами сайта, как «/home», «/manage» и «/settings».
- TopBar – верхний, фиксированный бар, содержащий заголовок страницы, имеющий дополнительную панель с кнопкой возврата на предыдущую страницу, а также кнопку сохранения внесённых изменений.
- AppRouter и ProtectedRoute – элементы, определяющие маршруты, доступные для перемещения по сайту. Не являются видимыми, но представляют важное значение для работы веб-приложения.
- MainWindow – контейнер для компонентов первичных страниц. Требуется для корректного выравнивания элементов интерфейса.
- SecondWindow – контейнер для компонентов страниц, не являющихся первичными. По причине расширения компонента TopBar для возможности навигации к предыдущей странице, требуются альтернативные параметры размещения, которые применяются в данном компоненте.
- VStack и HStack – контейнеры для вертикального и горизонтального расположения компонентов соответственно, а также применения общих стилей.
- UserCard – компонент для отображения карточек пользователей. Имеет кнопку перехода к настройкам пользователя, для определения правил доступа, а также изменения учетных данных.
- AddressCard – компонент карточки адреса со значением, кнопками настроек и удаления.
- ScheduleCard – компонент исключения правила доступа. Представляет собой карточку с информацией о днях, диапазоне времени и значении доступа. Поддерживает жест свайпа в направлении справа налево, для получения доступа к кнопке удаления.
- BigButton и SmallButton – общие компоненты кнопок, с настроенными анимацией, внешним обликом и цветовыми акцентами.
- AllowDenyButton, DuoButton и ToggleSwitch – кнопки для переключения между двумя состояниями. Различаются оформлением и реализуемой логикой.

Изм.	Лист	№ докум.	Подп.	Дата

- MenuButton – компонент кнопок нижнего бара для навигации по первичным страницам. Поскольку данные кнопки включают в себя векторные иконки и подписи, потребовалась тщательная настройка их облика.
 - AccessBadge – компонент бейджа для обозначения типа доступа.
 - DayBadge – компонент бейджа для обозначения дней в карточке исключения.
 - IPAddressForm – представляет собой форму для ввода ip-адреса в соответствии с допустимыми значениями.

					<i>BKP-НГТУ-09.03.01-(20-ПО)-05-2024</i>	<i>Лист 47</i>
<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подп.</i>	<i>Дата</i>		

7 Подготовка программно-аппаратной платформы

Компоненты и средства разработки проекта требуют использования 64 битной x86_64 архитектуры процессора. В качестве тестовой платформы используется персональный компьютер с процессором компании AMD на 8 физических ядер и 16 логических потоков под брендовым названием Ryzen 7 5800H, имеющим базовую частоту в 3.2ГГц, также располагающим 32 гигабайтами оперативной памяти стандарта DDR4 с частотой работы 3200МГц и твердотельным накопителем данных на 512 гигабайт. Данная конфигурация является достаточной и избыточной для запуска компонентов проекта.

После установки операционной системы, а именно, linux дистрибутива Debian версии выпуска 12, можно переходить к установке необходимого программного обеспечения. Для достижения работоспособности всех компонентов, требуется права администратора. По завершении установки системы Debian при помощи установочного образа, распространяемого с сайта дистрибутива, может отсутствовать программный пакет, обеспечивающий выполнение команд от имени администратора из аккаунта пользователя. Чтобы это исправить, нужно перейти под управление корневого пользователя, провести установку пакета и добавить аккаунт пользователя в группу, предоставляющую полномочия администратора. В приложении В, на рисунке В.1, демонстрируются команды, которые необходимо выполнить в терминале для достижения результата. С привилегиями администратора, в системе linux можно выполнять установку программ, а также чтение, изменение и удаление любых файлов.

Чтобы иметь возможность подключаться к сервисам сервера, ему следует назначить статический ip-адрес. Поскольку интерфейс интернет-роутера может сильно отличаться у разных производителей, описание данного шага опущено.

После, нужно выполнить перечень команд для настройки программной среды. В целях тестирования, были разработаны скрипты, выполняющие все действия автоматически. В случае подготовки материала к полноценному использованию, специалист или энтузиаст должен иметь соответствующую квалификацию, позволяющую выполнить настройку сетевого оборудования, сборку и запуск веб-сервиса. Поскольку текущая работа рассматривает разработку и применение сервис-ориентированной системы, но не её конечную эксплуатацию, вопрос подготовки окружения был упрощён до настройки тестовых условий, приближённых к целевым. Далее, будут рассмотрены команды подготовленных скриптов. Скрипты следует выполнять из домашней директории пользователя.

Изм.	Лист	№ докум.	Подп.	Дата	Лист	48
					VKR-НГТУ-09.03.01-(20-ПО)-05-2024	

Сперва, нужно выполнить установку необходимых пакетов. На рисунке В.2 демонстрируются команды, которые требуется выполнить.

По завершении установок, можно приступать к настройке программ. Для того, чтобы сервис серверной части мог взаимодействовать с базой данных, нужно её создать. Помимо этого, в PostgreSQL действует система ролей, определяющих полномочия пользователей. После установки, будет автоматически создана роль «*postgres*», однако подключиться с помощью неё к базе данных не получится, пока не будет установлен пароль для роли. На рисунке В.3 показаны команды для решения этих задач, которые были разработаны при изучении документации с сайта PostgreSQL [7].

Поскольку система будет управлять состоянием прокси, нужно отключить его автозапуск. Команды приведены на рисунке В.4.

Чтобы система имела доступ к сервису прокси и могла управлять его состоянием, требуется внести соответствующие настройки в конфигурацию sudo. Команды приводятся на рисунке В.5.

С целью подключения к прокси-серверу, нужно установить доменное имя, которое смогут использовать клиенты сервиса. В качестве тестового доменного имени будет использоваться значение «*proxy.home*». Создать DNS сервер поможет программа *dnsmasq*, установку которой выполняет скрипт из приложения В2. Команда для добавления домена приводится на рисунке В.6.

Для того, чтобы соединение с прокси работало по зашифрованному протоколу HTTPS, необходимо создать криптографический ключ и сертификат. В целях теста, был сгенерирован самоподписанный сертификат, который можно загрузить из веб-панели системы. Команды приведены на рисунке В.7.

Файл конфигурации прокси-сервера, располагающийся по пути «*/etc/squid/squid.conf*», необходимо переписать в соответствии с приведённым на рисунке В.8 материалом. Данный шаг можно выполнить вручную, воспользовавшись текстовым редактором, запущенным от имени администратора, например, с помощью команды: «*sudo nano /etc/squid/squid.conf*». Либо, записав конфигурацию в отдельный файл, открыв терминал из каталога, содержащего данный файл, и выполнив команду:

«*cat ИМЯ_ФАЙЛА | sudo tee /etc/squid/squid.conf*», где «*ИМЯ_ФАЙЛА*» нужно заменить на название файла, содержащего конфигурацию.

После выполнения подготовительных операций, можно переходить к сборке программных кодов. Загрузив исходные коды клиентской части, нужно перейти в корневой каталог и выполнить простую команду: «*npm build*». Это запустит сборку

Изм.	Лист	№ докум.	Подп.	Дата	Лист	49
					ВКР-НГТУ-09.03.01-(20-ПО)-05-2024	

проекта и подготовку файлов, которые серверная часть будет отдавать при подключении. По завершении сборки, появится каталог `build`, в котором расположатся файлы веб-клиента.

Затем, загрузив исходные коды серверной части в домашнюю папку пользователя, нужно перейти в каталог и поместить в папку «`static`» файлы собранной клиентской части. Каталог серверной части следует переименовать в «`web-panel`», для соответствия настройкам, выполненным с помощью скриптов.

Последним шагом в настройке служит создание `systemd`-сервиса, выполняющего запуск серверной части. Команда, обеспечивающая выполнение данной задачи, приводится на рисунке В.9.

Изм.	Лист	№ докум.	Подп.	Дата	Лист	50
					<i>BKR-НГТУ-09.03.01-(20-ПО)-05-2024</i>	

8 Тестирование

В текущем разделе ставится задача тестирования конечных точек серверной части, в которую входят проверка обработки правильных и некорректных данных, анализ результатов выполнения команд. В качестве вспомогательного программного обеспечения для отправки HTTP-запросов используется Postman.

Сперва, чтобы иметь полный доступ к возможностям сервера, требуется выполнить авторизацию под аккаунтом администратора и получить токен, благодаря которому обеспечивается аутентификация пользователя и проверка прав доступа. Для упрощения, начальным паролем администратора был определён – «1234», а именем – «admin». В тестовом режиме, сервер прослушивает порт 5000 и работает по протоколу HTTP. Сведения для авторизации посылаются в теле запроса на конечную точку по адресу «<http://192.168.50.139:5000/api/users/login>» методом «post», где «192.168.50.139» – ip-адрес сервера в локальной сети, «:5000» – обозначение порта, который прослушивает сервер, «/api/» – часть url-пути, обозначающая внешний интерфейс сервера, где располагаются конечные точки. В дальнейшем, url-адреса будут приводиться в приложениях. Ответом от сервера приходит объект JSON с полем «token», содержащим значение токена. Данное значение нужно сохранить для применения в последующих запросах. Форма создания запроса с полученным ответом приводится в приложении Г, на рисунке Г.1. В случае, если направить некорректный пароль, сервер пришлёт ответ, информирующий об этом (рисунок Г.2). Если направить некорректное имя пользователя, сервер уведомит, что такой пользователь не найден (рисунок Г.3).

После авторизации, чтобы пользователю не приходилось вновь вводить учетные данные в течение некоторого времени, а именно – 24 часов, реализована конечная точка, обеспечивающая проверку авторизации и обновления токена. Не её посыпается запрос методом «get» с заголовком «Authorization» и значением «Bearer TOKEN», где вместо «TOKEN» указывается значение токена. В случае успеха, сервер высылает новый токен с обновлённым временем жизни (рисунок Г.4). В случае некорректного значения заголовка или устаревшего токена, сервер отправит ответ с кодом 401 и сообщением, что пользователь не авторизован (рисунок Г.5). Все последующие конечные точки требуют наличие упомянутого заголовка с токеном для проверки аутентификации и полномочий.

Первым шагом в составлении конфигурации прокси является добавление пользователей. В теле запроса указываются поля имени («name») и пароля («password») и методом «post» направляются на конечную точку. В ответе сервер присыпает объект JSON

Изм.	Лист	№ докум.	Подп.	Дата	Лист	51
					VKP-НГТУ-09.03.01-(20-ПО)-05-2024	

с полями «`id`», содержащим идентификатор нового пользователя, «`name`» и полем «`ruleId`», содержащим идентификатор правила доступа пользователя (рисунок Г.6). В случае, если направить запрос на добавление пользователя с именем, которое уже присутствует в базе данных, сервер пришлёт отрицательный ответ с кодом 400 (рисунок Г.7). Если обязательные поля будут отсутствовать, например поле пароля, сервер также пришлёт отрицательный ответ с соответствующим уведомлением (рисунок Г.8). Для дальнейших тестов были добавлены три пользователя с именами «User 1», «User 2» и «User 3».

Чтобы получить список пользователей, направляется запрос методом «`get`». В качестве ответа, сервер посыпает массив объектов пользователей с полями, аналогичными при создании пользователя (рисунок Г.9). В данном списке присутствуют только пользователи с ролью «`USER`». Администратор и объект глобального пользователя не учитываются, поскольку не применяются для подключения к прокси.

Для обновления данных пользователя, используется метод «`put`». В теле запроса обязательным полем выступает «`id`» – идентификатор пользователя. Для обновления имени, используется поле «`name`», а для обновления пароля – поле «`password`». При успешном выполнении запроса, сервер возвращает объект JSON, аналогичный получаемому объекту при добавлении нового пользователя, но с дополнительным полем «`role`», сообщающим клиентской части о полномочиях пользователя (рисунок Г.10). Если новое имя уже присутствует в базе данных, сервер вернёт сообщение об ошибке (рисунок Г.11). При отсутствии идентификатора, сервер вернёт сообщение об ошибке с кодом 400 (рисунок Г.12), а при некорректном идентификаторе, будет указан код 404 и соответствующее уведомление (рисунок Г.13).

Удаление пользователя осуществляется при помощи передачи идентификатора пользователя методом «`delete`» в параметрах запроса. В случае успеха, сервер возвращает ответ с кодом 200 (рисунок Г.14). В зависимости от того, указан некорректный идентификатор (рисунок Г.15) или он отсутствует (рисунок Г.16), будет отличаться ответ сервера об ошибке.

При обновлении имени, возникает необходимость проверки на уникальность. С этой целью, на выделенную конечную точку направляется «`get`» запрос с полем «`name`», принимающим в качестве значения потенциальное новое имя пользователя. Для корректного запроса, сервер возвращает результат булевого типа, принимающего значение «`true`» в случае, если предоставленное имя присутствует в базе данных (рисунок Г.17), либо значение «`false`», если имя свободно для использования (рисунок Г.18). При отсутствии поля имени, сервер вернёт сообщение об ошибке (рисунок Г.19).

Изм.	Лист	№ докум.	Подп.	Дата	Лист	52
					ВКР-НГТУ-09.03.01-(20-ПО)-05-2024	

Согласно архитектуре модели данных, адреса группируются при помощи списков и не предоставляются обособленно от них. Следующим шагом по управлению конфигурацией является создание списков адресов. На соответствующую конечную точку нужно направить запрос методом «post» с полем «names», в котором требуется передать строковый массив, содержащий имена новых списков, и полем «userId» с идентификатором пользователя, которому предназначаются списки. Результатом сервер вернёт объект JSON с полем «lists», содержащим массив объектов созданных списков с полями идентификатора списка, имени, идентификаторов пользователя и правила доступа списка (рисунок Г.20). В случае, если в строковом массиве имён новых списков будет присутствовать название, добавленное ранее, сервер дополнит свой ответ полем «rejected» со строковым массивом отклонённых имён (рисунок Г.21). При отсутствии полей «names» или «userId», либо, при указании некорректного идентификатора пользователя, сервер вернёт сообщение об ошибке с кодом 400 (рисунки Г.22 и Г.23).

Для получения всех списков пользователя, нужно отправить запрос методом «get» на соответствующую конечную точку с параметром запроса «userId», содержащим значение идентификатора пользователя. В ответе сервер вернёт список объектов списков, относящихся к пользователю (рисунок Г.24). При отправке запроса с некорректным идентификатором пользователя, сервер вернёт пустой массив объектов (рисунок Г.25). В случае пропуска параметра, сервер вернёт сообщение об ошибке (рисунок Г.26).

Обновить наименование списка можно с помощью метода «patch». В теле запроса сервер ожидает идентификатор списка в поле «id» и новое название списка в поле «name». В ответ на успешный запрос, сервер отправляет обновлённый объект списка (рисунок Г.27). При отправке запроса с неизменённым именем, сервер также пришлёт объект списка. В случае отправки имени, присутствующем среди списков пользователя, сервер направит сообщение об ошибке с кодом 409 (рисунок Г.28).

Удаление списка выполняется при помощи метода «delete». В параметрах запроса передаётся идентификатор списка. В случае успеха, сервер возвращает ответ с кодом 200 (рисунок Г.29). Если идентификатор списка некорректен или отсутствует, сервер отправит сообщение об ошибке (рисунки Г.30 и Г.31).

Чтобы проверить, используется ли название списка, при добавлении или переименовании, можно направить запрос методом «get» на соответствующую конечную точку с передачей имени параметром запроса «name» и указанием идентификатора пользователя в параметре «userId». При совпадении, сервер отправит результат булевого типа со значением «true» (рисунок Г.32), в противном случае, значение будет «false»

Изм.	Лист	№ докум.	Подп.	Дата	Лист	53
					VKP-НГТУ-09.03.01-(20-ПО)-05-2024	

(рисунок Г.33). В случае пропуска какого-либо из требуемых параметров, сервер вернёт сообщение об ошибке (рисунок Г.34).

После создания списка, в него можно добавлять адреса. Добавление адресов выполняется при передаче запроса методом «post» с объектом JSON в теле запроса, включающим массив объектов новых адресов «addresses» и идентификатор списка «listId». В случае успеха, сервер возвращает JSON объект с полем «created», содержащим массив созданных объектов адресов (рисунок Г.35). Если отправить адреса, присутствующие в списке, либо направить объект адреса без указания типа или значения, сервер вернёт их в поле «rejected» (рисунок Г.36). При отсутствии обязательных полей в запросе, сервер вернёт сообщение об ошибке (рисунок Г.37).

Для получения адресов списка, нужно отправить запрос «get» на соответствующую конечную точку с указанием идентификатора списка в параметрах запроса. При успехе, сервер возвращает массив объектов адресов, связанных со списком (рисунок Г.38). Если передать некорректный идентификатор списка, сервер вернёт пустой массив (рисунок Г.39). В случае отсутствия параметра идентификатора, сервер вернёт сообщение об ошибке (рисунок Г.40).

В целях изменения адреса, используется метод «patch». В теле запроса нужно передать идентификатор адреса «id», тип адреса «type» и значение «value». В случае успешного выполнения запроса, сервер возвращает обновлённый объект адреса (рисунок Г.41). При отсутствии любого из полей, сервер вернёт сообщение об ошибке (рисунок Г.42).

Удаление адреса выполняется при помощи метода «delete». В параметрах запроса нужно указать идентификатор адреса. При успехе, сервер возвращает ответ с кодом 200 (рисунок Г.43). Если указать некорректный идентификатор или не передать его в параметрах, сервер вернёт сообщение о соответствующей ошибке (рисунки Г.44 и Г.45).

Поскольку правила создаются и удаляются вместе с аналогичными операциями для пользователей, списков и адресов, актуальными остаются действия получения правил и их обновления.

Чтобы изменить правило доступа, следует воспользоваться методом «put». При передаче в теле запроса параметра доступа «access» или параметра применения исключений «schedule» вместе с идентификатором правила «id», можно обновить данные значения объекта правила в базе данных (пример на рисунке Г.46). Также, при помощи данного метода, можно добавить, обновить и удалить исключения. Пример добавления

Изм.	Лист	№ докум.	Подп.	Дата	Лист	54
					VKR-HГТУ-09.03.01-(20-ПО)-05-2024	

исключения приводится на рисунке Г.47. На рисунках Г.48 и Г.49 также приводятся примеры обновления и удаления исключения.

Для получения объекта правила доступа, нужно отправить запрос с методом «get» на соответствующую конечную точку и передать в параметрах запроса идентификатор правила «id». В случае успеха, сервер вернёт объект правила и массив объектов исключений, при их наличии (пример приведён на рисунке Г.50). Если передать некорректный идентификатор или не указывать его, сервер вернёт ошибку с соответствующим сообщением (рисунки Г.51 и Г.52).

Создав перечень правил доступа к сетевым ресурсам для пользователей прокси, можно переходить к управлению состоянием прокси-сервера. Чтобы получить информацию о состоянии сервиса прокси, используемом доменном имени и порте, нужно отправить «get» запрос на соответствующую конечную точку. Пример выполнения запроса приведён на рисунке Г.53.

Для включения и выключения сервиса прокси применяется запрос с методом «post». В качестве ответа, сервер возвращает обновлённый статус работы прокси (рисунок Г.54).

Чтобы применить описанные настройки доступа, необходимо обновить конфигурацию прокси. Для этого, нужно направить запрос методом «put». Сервер, согласно алгоритму, сгенерирует параметры конфигурации, запишет их в файл внутри каталога конфигураций прокси-сервера и выполнит команду обновления настроек. По завершении, сервер вернёт ответ с кодом статуса 200 (рисунок Г.55). Пример сгенерированных параметров конфигурации приводится на рисунке Г.56.

При необходимости сброса настроек до заводских, нужно отправить запрос с методом «delete». Сервер пришлёт ответ с кодом статуса 200 и сообщением об успешном сбросе системы (рисунок Г.57). Это приведёт к очистке базы данных, имя и пароль администратора примут исходные значения, сервис прокси будет отключен, а файл конфигурации стёрт.

Изм.	Лист	№ докум.	Подп.	Дата	Лист	55
					BKR-НГТУ-09.03.01-(20-ПО)-05-2024	

9 Экспериментальная часть

В рамках эксперимента, будет рассмотрен сценарий использования системы посредством управления через веб-интерфейс. Сценарий включает такие действия, как:

- Авторизация с доступом администратора;
- Создание нового пользователя;
- Создание списка адресов;
- Внесение новых адресов в список;
- Изменение доступа к адресам;
- Добавление исключений к настройкам доступа;
- Обновление конфигурации;
- Включение прокси-сервера;
- Авторизация с доступом пользователя;
- Установка сертификата прокси-сервера;
- Изменение учетных данных пользователя;
- Подключение к прокси-серверу;
- Проверка применённых настроек.

Запуск веб-интерфейса осуществлялся с мобильного устройства компании Apple под брендовым названием iPhone Xr под управлением операционной системы iOS версии 15.7.1. Клиент применялся в формате PWA-приложения. Для подключения к прокси-серверу использовалось приложение Shadowrocket версии 2.2.51 (2285).

Выполнив переход по сетевому адресу в браузере смартфона, на котором запущен веб-сервер, открывается страница авторизации (приложение Д, рисунок Д.1). Нажав на среднюю кнопку в нижней панели управления, откроется меню страницы браузера (рисунок Д.2). Выбрав пункт «Add to Home Screen» (добавить на домашний экран), открывается форма добавления PWA-приложения, где можно задать имя по желанию (рисунок Д.3).

Открыв приложение, пользователя встречает экран авторизации (рисунок Д.4). В момент первого взаимодействия, единственным доступным для входа является аккаунт администратора. По завершении авторизации, открывается главная страница с информацией о прокси-сервере, кнопками загрузки сертификата, обновления конфигурации и включения/выключения сервиса (рисунок Д.5).

В соседнем разделе «Управление» располагается список пользователей с кнопкой добавления, а также раздел глобальных настроек (рисунок Д.6). По нажатии на кнопку,

Изм.	Лист	№ докум.	Подп.	Дата	Лист	56
					BKR-НГТУ-09.03.01-(20-ПО)-05-2024	

появляется всплывающее окно с полями имени нового пользователя и пароля, а также кнопками отмены и сохранения. В рамках проведения эксперимента, создан пользователь с именем «dima» и паролем «12345678» (рисунок Д.7). Эти учетные данные используются для подключения к прокси-серверу, а также для входа в веб-клиент. После добавления пользователя, появляется карточка с именем и кнопкой перехода к настройкам.

На странице пользователя доступно изменение учетных данных, удаление пользователя, управление правилом доступа пользователя, а также списки адресов с кнопкой добавления новых. По нажатии на кнопку добавления списка, появляется всплывающее окно с полем ввода названия (рисунок Д.8). В рамках эксперимента, будет создан список с названием «Поисковые системы». После добавления, список отображается в отдельном перечислении и недоступен для дальнейших настроек. Чтобы перейти к управлению списком, необходимо выполнить сохранение. Кнопка сохранения появляется в верхнем правом углу при изменении списков, адресов, правил доступа и исключений.

Страница списка содержит кнопки переименования и удаления, перехода к управлению правилом доступа списка, а также список адресов с кнопкой добавления. При нажатии, открывается всплывающее окно с двумя вкладками: «доменное имя» и «ip адрес». По ходу эксперимента, добавляются домены поисковой системы «google» (рисунок Д.9). После сохранения добавленных адресов, на появившихся карточках становится доступна кнопка перехода к настройкам адреса.

Прежде, чем перейти к адресам, в рамках эксперимента, в разделе управления правилом доступа списка, параметр доступа был изменён со значения «разрешить» на «запретить» (рисунок Д.10).

Страница адресов представлена последней в иерархии разделов. Здесь приводятся значение адреса, кнопка его изменения, тумблер изменения параметра доступа, тумблер применения исключений и скрывающаяся кнопка добавления исключений (рисунок Д.11). По нажатии на кнопку добавления, появляется всплывающее окно с настройками исключения. В нём можно выбрать дни, в которые будет действовать исключение, и указать диапазон времени действия. Допускается опустить один из параметров. В ходе эксперимента, были указаны дни с понедельника по пятницу с временным диапазоном от 8:00 до 12:00. Сверка даты происходит на сервере, поэтому пользователи прокси-сервера не смогут манипулировать этими показателями. После сворачивания заполненного окна, появляется карточка исключения. По нажатии на неё, будет открываться такое же окно с

Изм.	Лист	№ докум.	Подп.	Дата

возможностью внесения изменений. Добавление, изменение и удаление исключений также необходимо сопровождать сохранением (рисунок Д.12).

После завершения настроек доступа, нужно выполнить обновление конфигурации прокси-сервера. Для этого, необходимо вернуться на главную страницу и нажать кнопку «обновить конфигурацию», предварительно, убедившись, что прокси-сервер работает, в противном случае, включив его (рисунок Д.13).

Чтобы выйти из аккаунта, нужно перейти в раздел «Настройки», где располагаются кнопки изменения собственных учетных данных, сброса системы к заводским настройкам и выхода, а затем нажать на последнюю (рисунок Д.14).

Рядовые пользователи прокси-сервера также могут авторизоваться в веб-клиенте, где им будут доступны сведения для подключения к прокси. В ходе эксперимента, выполняется авторизация под новым созданным пользователем (рисунок Д.15).

Для корректного подключения к прокси серверу и обеспечения защищённого соединения, необходимо установить сертификат. Веб-сервер предоставляет авторизованным пользователям возможность загрузки требуемого файла (рисунок Д.16). Операционная система iOS автоматически распознаёт сертификаты и вносит их в очередь на проверку и установку в меню настроек смартфона. Кнопка перехода к управлению сертификатом появится в начале меню настроек (рисунок Д.17). По ходу эксперимента, сертификат был установлен.

Дополнительно, в настройках Wi-Fi был добавлен адрес локального DNS-сервера, содержащего запись домена прокси-сервера, по которому будет осуществляться подключение (рисунок Д.18).

Штатные настройки подключения к Wi-Fi сетям позволяют указать прокси и данных для аутентификации, однако для мобильных сетей данный функционал отсутствует. Вследствие этого, возникает необходимость применения стороннего программного обеспечения. Одним из лучших решений, предоставляющих широкий спектр поддерживаемых протоколов, выступает приложение Shadowrocket. При добавлении конфигурации, требуется указать тип прокси, адрес, порт, имя пользователя и пароль. В данном случае, указывается тип HTTPS, адрес «proxy.home», порт 8443 и учётные данные пользователя, созданного ранее (рисунок Д.19). После подключения, в панели управления операционной системы появляется значок «VPN», информирующий об использовании прокси (рисунок Д.20).

Для проверки работы ограничений доступа, был использован встроенный браузер Safari. Согласно принятым настройкам, доступ к домену «google.ru» должен быть

Изм.	Лист	№ докум.	Подп.	Дата

заблокирован, тогда, как домен «google.com» обязан быть доступен в установленный диапазон времени. Демонстрация приводится на рисунке Д.21. По итогам результатов, можно сделать вывод, что настройки доступа работают корректно.

							Лист
Изм.	Лист	№ докум.	Подп.	Дата		BKR-НГТУ-09.03.01-(20-ПО)-05-2024	59

Заключение

По завершении работы, автором были достигнуты все поставленные задачи. При проработке задач, были определены средства разработки, оптимальные по таким параметрам, как время и сложность разработки. Веб-приложение, в текущей реализации, рассчитано на небольшое число пользователей и располагает одним административным аккаунтом. Вследствие этого, а также отсутствия технической возможности выполнить проверку на критическую нагрузку, задача обеспечения высокой эффективности системы не рассматривалась. Согласно ответу разработчиков прокси Squid [9], сервер на однопоточном процессоре с тактовой частотой 1.2ГГц способен обрабатывать около 950 запросов в секунду, а 2 гигабайта оперативной памяти теоретически могут обеспечить до 20 тысяч соединений за счёт использования 64 килобайт для каждого сетевого сокета в целях сохранения состояния трафика. Также, исходя из результатов тестирования, опубликованных на интернет платформе DEV [12], node.js сервер с одним вычислительным ядром и 1 гигабайтом оперативной памяти способен выдержать около 2600 запросов в секунду. Таким образом, можно сделать вывод, что, на современном оборудовании, допустимое количество пользователей прокси, в большей степени, будет ограничиваться пропускной способностью сетевого канала и техническими возможностями сетевого оборудования. При достаточном объёме вычислительных ресурсов, не уступающих рекомендуемым техническим требованиям, описанным в разделе постановки задачи, система может применяться в компаниях со штатом до тысячи человек.

В ходе изучения документации и учебной литературы по настройке и управлению прокси-сервером Squid, была оформлена начальная конфигурация и разработан алгоритм построения её параметров.

Перед началом разработки программных компонентов, была определена модель данных, соответствующая возможностям конфигурации прокси и поставленным задачам по ожидаемому функционалу веб-приложения.

Для хранения данных, использована СУБД PostgreSQL. При проектировании модели данных, также были учтены её преимущества в виде поддержки хранения объектов JSON и перечислений.

В процессе разработки, на платформе node.js, с помощью фреймворка express, был реализован веб-сервис, обеспечивающий управление сервисом прокси, работу с базой данных, контроль пользователей и предоставление информации. Сущности модели

Изм.	Лист	№ докум.	Подп.	Дата	Лист	60
					VKR-НГТУ-09.03.01-(20-ПО)-05-2024	

данных были определены при помощи ORM Sequelize. За счёт этого, взаимодействие с базой данных не требовало применения SQL-запросов, что положительно отразилось на скорости разработки. Однако, PostgreSQL требует начальной настройки для последующего подключения, что делает выбор в пользу данной СУБД не самым оптимальным и усложняет процесс подготовки компонентов проекта к работе.

Графический интерфейс веб-приложения реализован при помощи библиотеки React, также на платформе node.js. По результату, страницы отлично адаптируются к экранам смартфонов, а на устройствах под управлением операционных систем Android версии 4.4 и выше, а также iOS версии 11.3, поддерживается возможность установки сайта в качестве PWA-приложения, обладающего внешним обликом нативных приложений. Разработка клиентской части была сопряжена с трудностями в настройке внешнего облика и поведения анимаций на мобильных устройствах. Выбранный в помощь фреймворк bootstrap, с библиотекой готовых экранных форм, предоставлял неудовлетворительный результат, при попытках подстроить компоненты интерфейса под общий стиль приложения. Большая часть компонентов фреймворка была переписана или заменена свободно распространяемыми модулями. Исключением стали компоненты всплывающих окон, поведение которых было предсказуемым и удовлетворительным. Для составления интерфейса в единообразном стиле, была разработана собственная библиотека компонентов React. Разработаны элементы управления с корректными анимациями, а также поддержкой жестов, где это допустимо, что представляется удобным при пользовании со смартфона, обладающего сенсорным дисплеем. С целью разделения функционала веб-приложения по признакам авторизации и прав доступа, созданы различные маршруты навигации по интерфейсу. Реализованы механизмы авторизации и выхода. По итогу, веб-приложение полностью соответствует заданным требованиям.

По завершении разработки, было выполнено успешное тестирование серверной части на предмет обработки запросов с корректными и некорректными данными.

Завершающим этапом проекта стала экспериментальная часть, в ходе которой была проверена работа веб-приложения и прокси на основе предопределённого сценария. В результате, при взаимодействии с интерфейсом, ошибки отсутствовали, а заданные настройки были успешно применены.

При сочетании данного проекта со встроенными функциями родительского контроля, корпоративными инструментами управления устройствами и т.п., можно достичь большей гибкости в контроле доступа пользователей к интернет ресурсам. Однако, независимо от других программных средств система не может применяться,

Изм.	Лист	№ докум.	Подп.	Дата	Лист	61
					VKR-НГТУ-09.03.01-(20-ПО)-05-2024	

поскольку она нацелена на решение заданного спектра задач в комплексе со сторонним программным обеспечением.

В качестве последующего развития, возможны следующие шаги:

- Замена PostgreSQL на SQLite. Преимуществом второго решения является возможность интегрировать его в приложение, результатом чего станет отсутствие необходимости предварительной настройки СУБД. Обратной стороной такого решения станет необходимость изменений в серверной части.
- Объединение исходных кодов клиентской и серверной частей, для упрощения сборки проекта.
- Реализация добавления пользователей с административными правами в клиентской части (поддержка в серверной части присутствует).
- Добавление поддержки группировки пользователей и списков адресов.

Изм.	Лист	№ докум.	Подп.	Дата	Лист	62
					VKR-НГТУ-09.03.01-(20-ПО)-05-2024	

Перечень сокращений, условных обозначений, символов, терминов

ПО – программное обеспечение

СУБД – система управления базами данных

URL – англ. Uniform Resource Locator – унифицированный указатель ресурса

Изм.	Лист	№ докум.	Подп.	Дата

Список источников и литературы

1. Desktop vs Mobile Market Share Worldwide : раздел // StatCounter : сайт. 2024. URL: <https://gs.statcounter.com/platform-market-share/desktop-mobile/worldwide> (дата обращения: 27.04.2024)
2. Gatesentry : раздел // github : сайт. 2024. URL: <https://github.com/fifthsegment/Gatesentry> (дата обращения: 27.04.2024)
3. How Many Smartphones Are In The World? : раздел // BankMyCell.com : сайт. 2024. URL: <https://www.bankmycell.com/blog/how-many-phones-are-in-the-world> (дата обращения: 27.04.2024)
4. JavaScript : раздел // MDN Web Docs : сайт. 2024. 5 мар. URL: <https://developer.mozilla.org/en-US/docs/Web/JavaScript> (дата обращения: 27.04.2024)
5. ORM : раздел // Wikipedia : сайт. 2024. URL: <https://ru.wikipedia.org/wiki/ORM> (дата обращения: 27.04.2024)
6. Parental control Kroha : раздел // apps.apple.com : сайт. 2024. URL: <https://apps.apple.com/us/app/parental-control-kroha/id1365784521> (дата обращения: 27.04.2024)
7. PostgreSQL 16 Documentation : раздел // PostgreSQL : сайт. 2024. URL: <https://www.postgresql.org/docs/16/index.html> (дата обращения: 27.04.2024)
8. Requirements : раздел // Debian Wiki : сайт. 2024. URL: <https://wiki.debian.org/DebianEdu/Documentation/Bookworm/Requirements> (дата обращения: 27.04.2024)
9. Re: [squid-users] About bottlenecks (Max number of connections, etc.) : веб-страница, 25 фев. 2013. // squid-cache.org : сайт. 2024. URL: <http://www.squid-cache.org/mail-archive/squid-users/201302/0304.html> (дата обращения: 27.04.2024)
10. Sequelize v6 : раздел // Sequelize : сайт. 2024. URL: <https://sequelize.org/docs/v6/> (дата обращения: 27.04.2024)
11. Squid Proxy Server 3.1 Beginner's Guide / Kulbir Saini. – Packt Publishing Ltd., 2011. – 332 с.
12. Under Pressure: Benchmarking Node.js on a Single-Core EC2 : раздел, 23 дек. 2023. // DEV : сайт. 2024. URL: <https://dev.to/ocodista/under-pressure-benchmarking-nodejs-on-a-single-core-ec2-5ghe> (дата обращения: 27.04.2024)

Изм.	Лист	№ докум.	Подп.	Дата	Лист
					BKR-НГТУ-09.03.01-(20-ПО)-05-2024

Приложение А. Веб-панель альтернативного решения

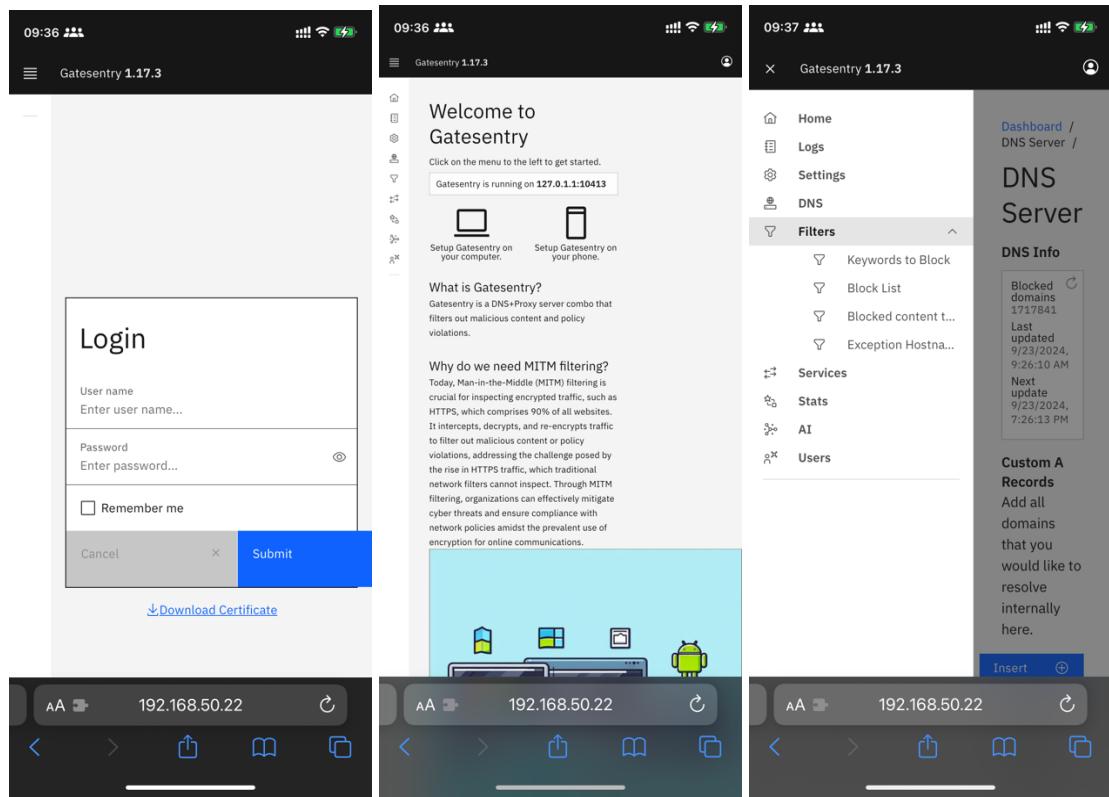


Рисунок А.1. – Страница авторизации, Главная, Меню

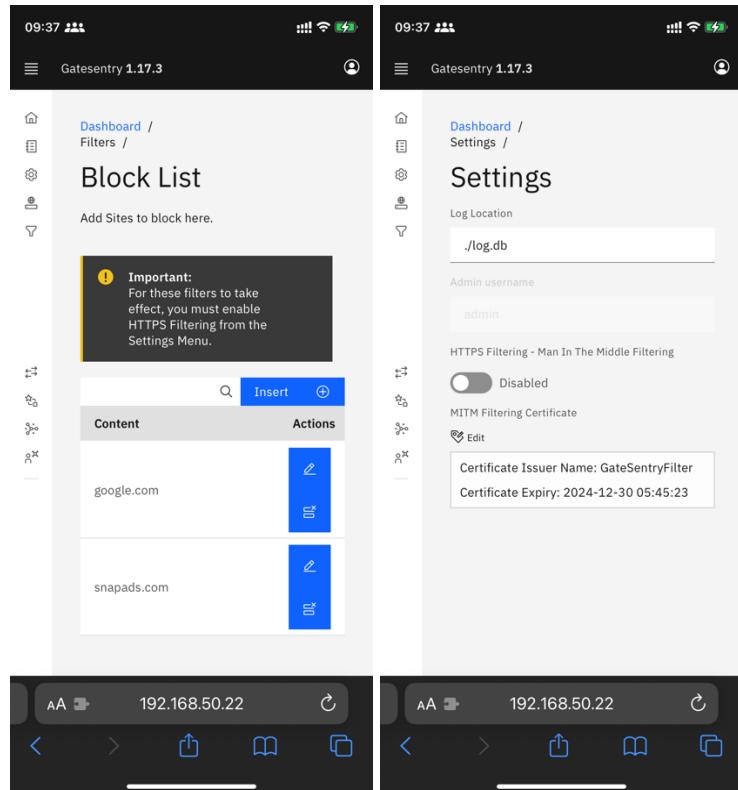


Рисунок А.2. – Страницы фильтрации и настроек

Изм.	Лист	№ докум.	Подп.	Дата

Приложение Б. Блок-схемы алгоритма построения конфигурации прокси

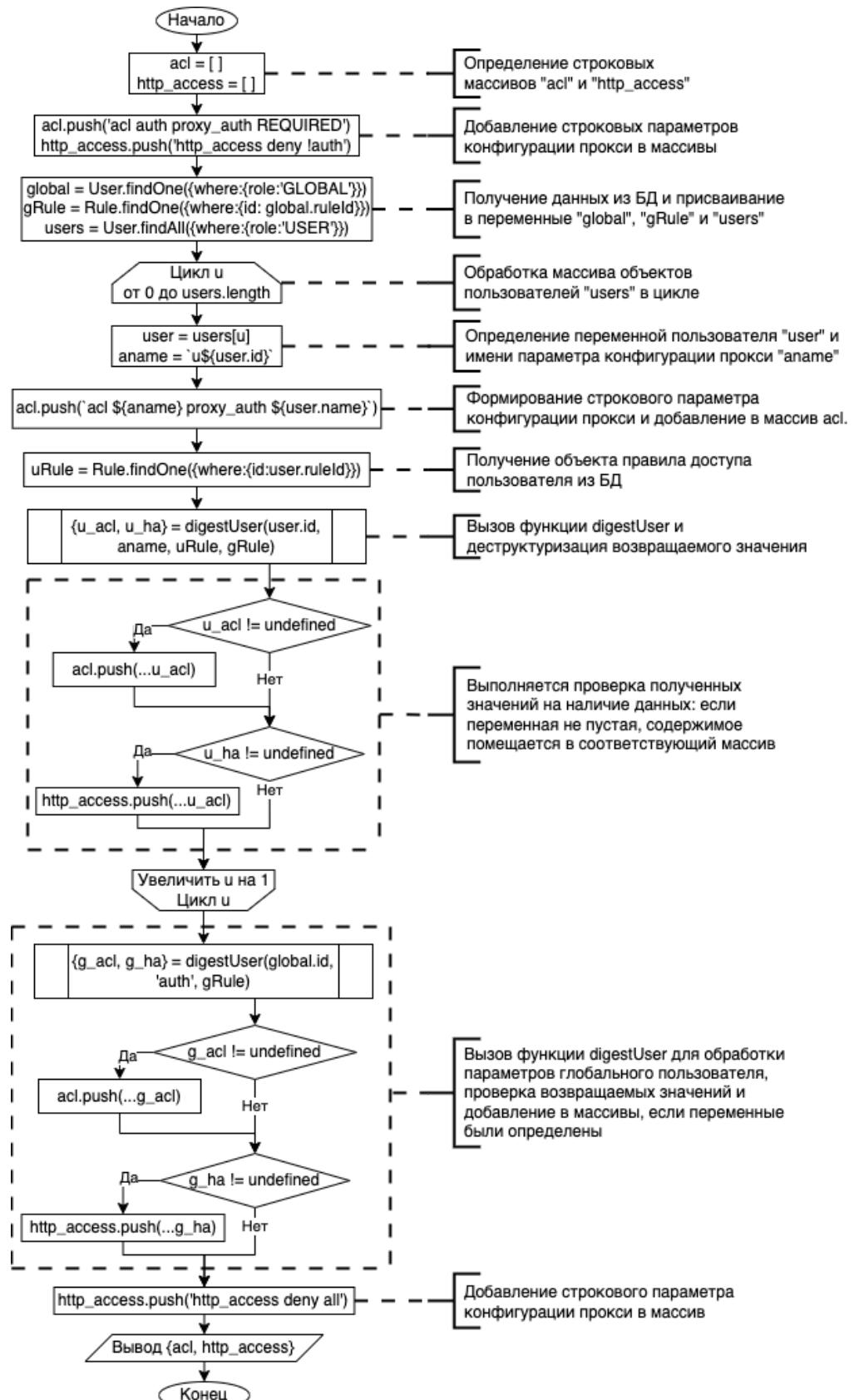


Рисунок Б.1. – Блок-схема функции digestEntities

Изм.	Лист	№ докум.	Подп.	Дата

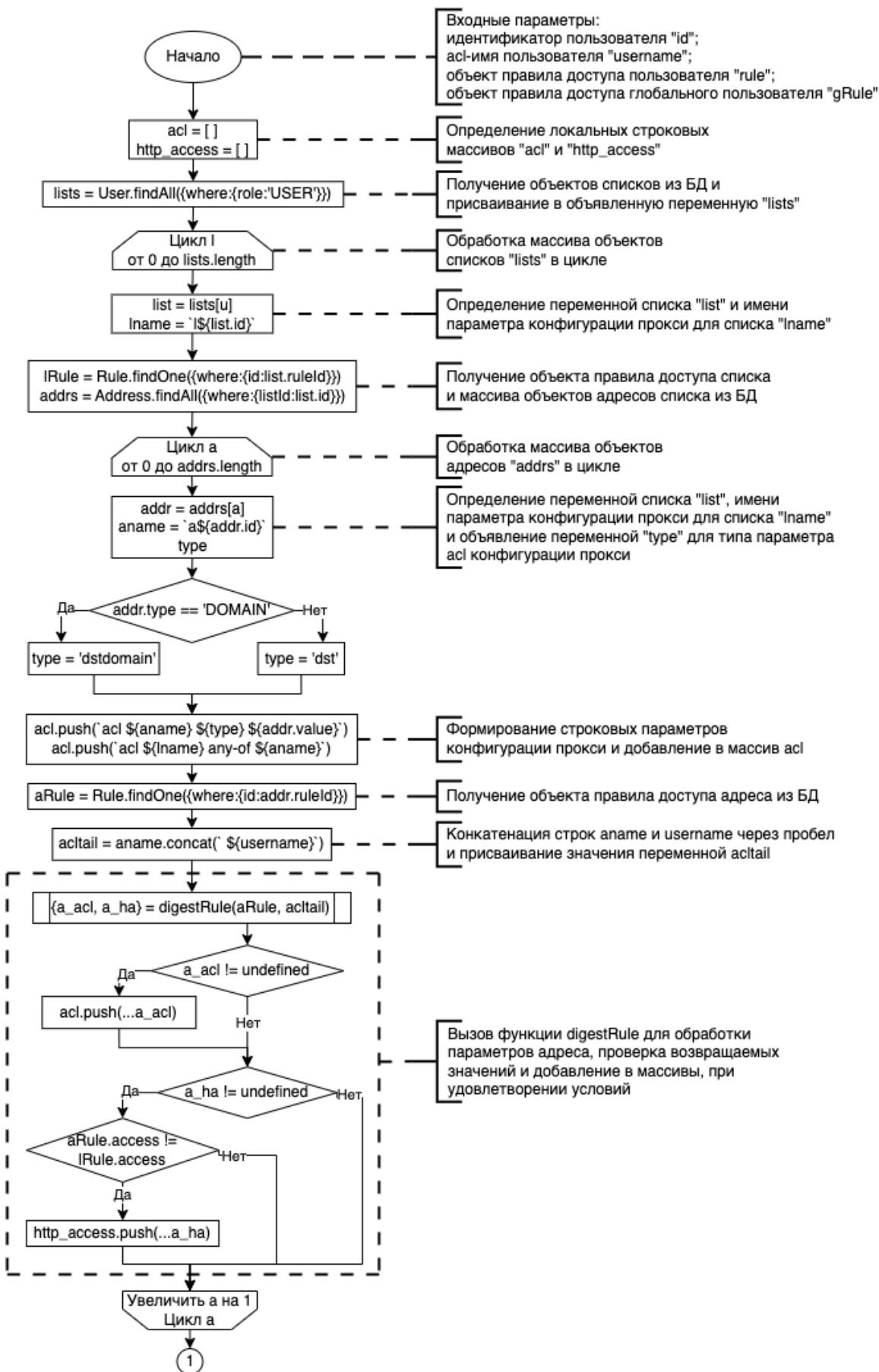


Рисунок Б.2. – Блок-схема функции digestUser, Часть 1 из 2

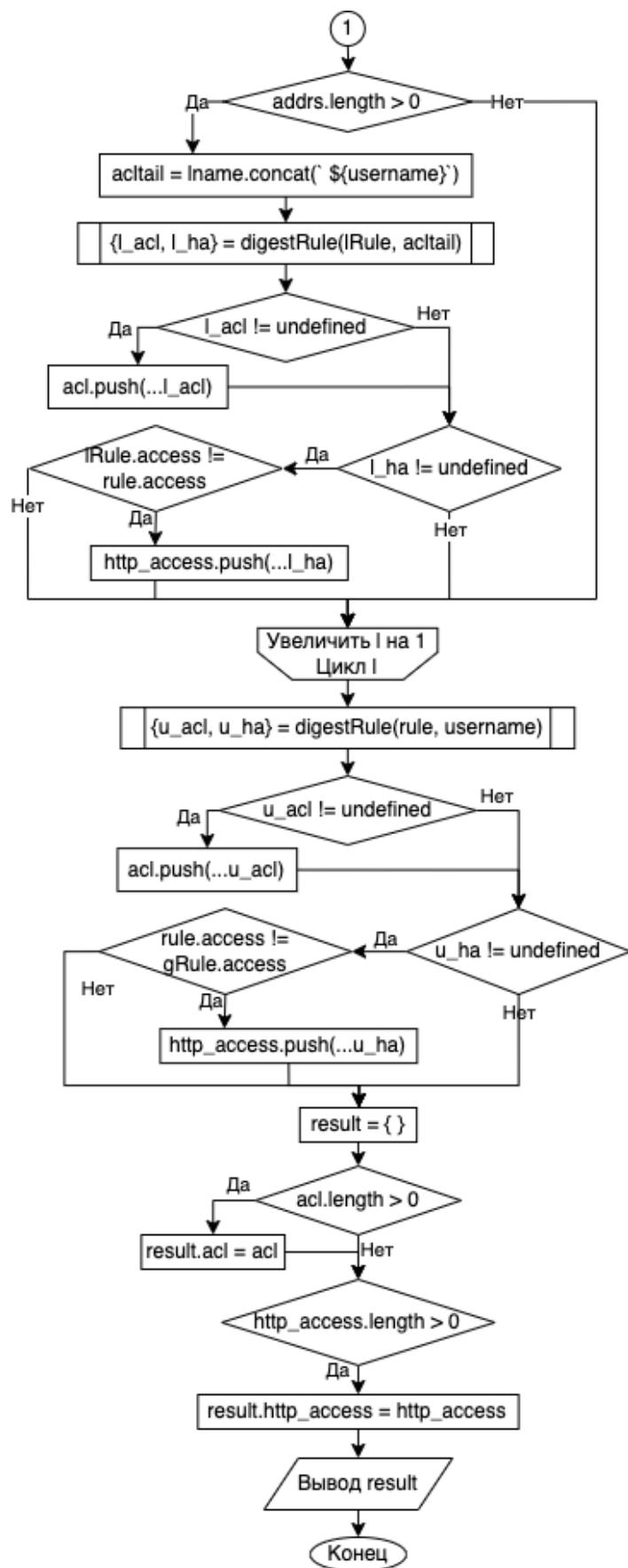


Рисунок Б.3. – Блок-схема функции digestUser, Часть 2 из 2

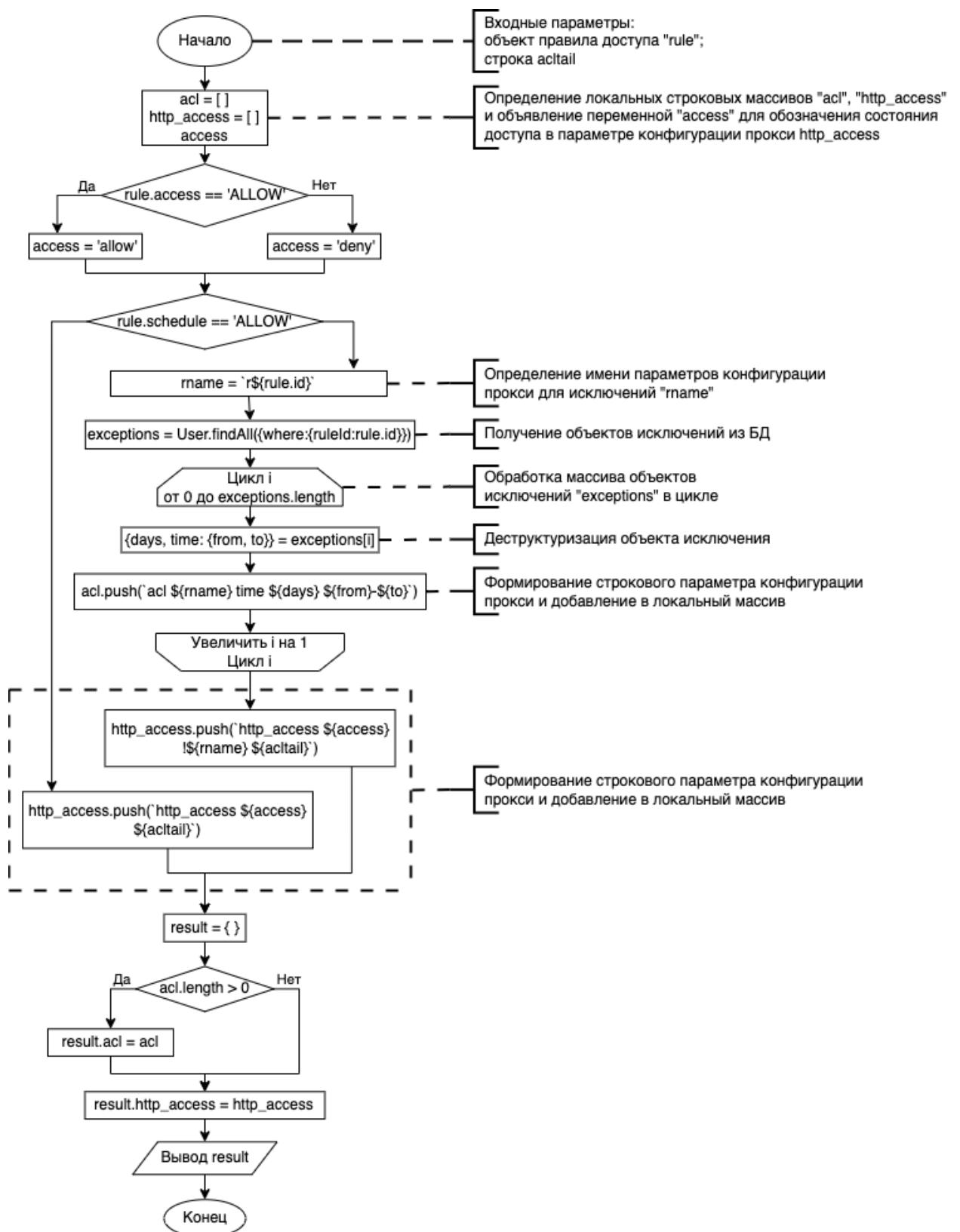


Рисунок Б.4. – Блок-схема функции digestRule

Изм.	Лист	№ докум.	Подп.	Дата

Приложение В. Команды и настройки для подготовки программного окружения

```
su - root  
apt install sudo  
usermod -aG sudo <имя_пользователя>  
su - <имя_пользователя>
```

Рисунок В.1. – Получение привилегий администратора

```
sudo apt install nano openssl dnsmasq curl hostname squid postgres -y  
curl -o https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.1/install.sh | bash  
nvm install --lts v20.15.0  
nvm use --lts v20.15.0
```

Рисунок В.2. – Установка пакетов

```
sudo -u postgres psql -c "ALTER USER postgres PASSWORD 'postgres';"  
sudo -u postgres createdb proxydb
```

Рисунок В.3. – Настройка PostgreSQL

```
sudo systemctl stop squid  
sudo systemctl disable squid
```

Рисунок В.4. – Отключение автозапуска Squid

```
echo '%sudo ALL=NOPASSWD: /usr/bin/systemctl is-active squid' | sudo EDITOR='tee -a' visudo  
echo '%sudo ALL=NOPASSWD: /usr/bin/systemctl start squid' | sudo EDITOR='tee -a' visudo  
echo '%sudo ALL=NOPASSWD: /usr/sbin/squid' | sudo EDITOR='tee -a' visudo
```

Рисунок В.5. – Настройка параметров sudo

```
IP_ADDR="$(hostname -I)"  
sudo tee -a /etc/hosts << EOF  
$IP_ADDR proxy.home  
EOF  
sudo systemctl restart dnsmasq
```

Рисунок В.6. – Настройка DNS-сервера

Изм.	Лист	№ докум.	Подп.	Дата

```

sudo mkdir /etc/squid/ssl
sudo openssl req -x509 -newkey ed25519 -keyout /etc/squid/ssl/key.pem -noenc \
    -out /etc/squid/ssl/cert.pem -days 3650 -subj "/CN=proxy.home" \
    -addext "subjectAltName=DNS:proxy.home, IP:$IP_ADDR"
sudo chmod 644 /etc/squid/ssl/cert.pem

```

Рисунок В.7. – Создание криптографических ключа и сертификата

```

auth_param basic program /usr/lib/squid/basic_db_auth --dsn "dbi:Pg:database=proxydb;port=5432" --
    -user "postgres" --password "postgres" --table "users" --usercol "name" --passwdcol "password" --
    cond ""
auth_param basic children 5 startup=5 idle=1
auth_param basic credentialsttl 2 hours

acl SSL_ports port 443
acl SSL_ports port 8443
acl Safe_ports port 80          # http
acl Safe_ports port 21          # ftp
acl Safe_ports port 443         # https
acl Safe_ports port 70          # gopher
acl Safe_ports port 210         # wais
acl Safe_ports port 1025-65535 # unregistered ports
acl Safe_ports port 280         # http-mgmt
acl Safe_ports port 488         # gss-http
acl Safe_ports port 591         # filemaker
acl Safe_ports port 777         # multiling http

http_access deny !Safe_ports
http_access deny CONNECT !SSL_ports

include /etc/squid/conf.d/*.conf
https_port 8443 tls-cert=/etc/squid/ssl/cert.pem tls-key=/etc/squid/ssl/key.pem

acl hasRequest has request

logformat my_format %{%d.%m.%Y %H:%M:%S}tl %>a %un %Ss/%03>Hs %rm %ru
access_log daemon:/var/log/squid/access.log my_format hasRequest
coredump_dir /var/spool/squid

refresh_pattern ^ftp:           1440   20%   10080
refresh_pattern ^gopher:        1440   0%    1440
refresh_pattern -i (/cgi-bin/|\.?) 0     0%    0
refresh_pattern .               0     20%   4320

shutdown_lifetime 0 seconds

```

Рисунок В.8. – Конфигурация Squid прокси

Изм.	Лист	№ докум.	Подп.	Дата	Лист
					71

```
NPM="$ (which npm)"  
sudo touch /etc/systemd/system/proxy-web-panel.service  
sudo tee /etc/systemd/system/proxy-web-panel.service << EOF  
[Unit]  
Description=Run proxy web services at start  
After=multi-user.target  
  
[Service]  
ExecStart=$NPM run /home/$USER/web-panel start  
Type=simple  
  
[Install]  
WantedBy=multi-user.target  
EOF  
sudo systemctl daemon-reload  
sudo systemctl enable proxy-web-panel.service
```

Рисунок В.9. – Создание systemd-сервиса

Изм.	Лист	№ докум.	Подп.	Дата

Приложение Г. Тестирование серверной части

The screenshot shows a POST request to `http://192.168.50.139:5000/api/users/login`. The request body is JSON with fields `"name": "admin"` and `"password": "1234"`. The response status is `200 OK`, time is `137 ms`, size is `457 B`. The response body contains a token: `"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwibmFtZSI6ImFkbWluIiwicm9sZSI6IkFETUlOIiwiaWF0IjoxNzI2NjA5ODMzLCJleHAiOjE3MjY2OTYyMzN9.XstZZuptT0fba2qH7N_n1b9f1lDleCIDF8qK_ZEXA4"`.

Рисунок Г.1. – Авторизация

The screenshot shows a POST request to `http://192.168.50.139:5000/api/users/login`. The request body is JSON with fields `"name": "admin"` and `"password": "12345"`. The response status is `400 Bad Request`, time is `37 ms`, size is `332 B`. The response body contains a message: `"message": "Указан неверный пароль"`.

Рисунок Г.2. – Ошибка авторизации: некорректный пароль

The screenshot shows a POST request to `http://192.168.50.139:5000/api/users/login`. The request body is JSON with fields `"name": "adm"` and `"password": "1234"`. The response status is `404 Not Found`, time is `37 ms`, size is `330 B`. The response body contains a message: `"message": "Пользователь не найден"`.

Рисунок Г.3. – Ошибка авторизации: некорректное имя

Изм.	Лист	№ докум.	Подп.	Дата

The screenshot shows a POST request to `http://192.168.50.139:5000/api/users/auth`. The Headers tab is selected, showing an `Authorization` header with the value `Bearer eyJhbGciOiJIUzI1N...
eyJpZCI6MiwibmFtZSI6ImFkbWluIiwi...
I2NjExOTYyLC3leHA10je3mjY20tgzNj9...
Z4sUwCNBUUkqkPaV46KcuJiBt19ZnEhTdaBcEQ3rUho"`. The response status is `200 OK` with a response body containing a JSON object with a `token` field.

```

1 {
2   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
eyJpZCI6MiwibmFtZSI6ImFkbWluIiwi...  
I2NjExOTYyLC3leHA10je3mjY20tgzNj9...  
Z4sUwCNBUUkqkPaV46KcuJiBt19ZnEhTdaBcEQ3rUho"
3 }

```

Рисунок Г.4. – Проверка аутентификации и обновление токена

The screenshot shows a POST request to `http://192.168.50.139:5000/api/users/auth`. The Headers tab is selected, showing an `Authorization` header with the value `Bearer`. The response status is `401 Unauthorized` with a response body containing a JSON object with a `message` field.

```

1 {
2   "message": "Пользователь не авторизован"
3 }

```

Рисунок Г.5. – Неудачная проверка аутентификации

The screenshot shows a POST request to `http://192.168.50.139:5000/api/users`. The Body tab is selected, showing a JSON payload with `name` and `password` fields. The response status is `200 OK` with a response body containing a JSON object with `id`, `name`, and `ruleId` fields.

```

1 {
2   "name": "User 1",
3   "password": "1234"
4 }

```

```

1 {
2   "id": 3,
3   "name": "User 1",
4   "ruleId": 2
5 }

```

Рисунок Г.6. – Добавление пользователя

Body

```

1  {
2   |   "name": "User 1",
3   |   "password": "1234"
4 }

```

400 Bad Request • 22 ms • 369 B • [e.g.](#) [ooo](#)

Pretty Raw Preview Visualize JSON [≡](#) [✖](#) [✖](#) [✖](#)

```

1  {
2   |   "message": "Пользователь с таким именем уже существует"
3 }

```

Рисунок Г.7. – Ошибка добавления пользователя: повтор имени

Body

```

1  {
2   |   "name": "User 2"
3 }

```

400 Bad Request • 5 ms • 347 B • [e.g.](#) [ooo](#)

Pretty Raw Preview Visualize JSON [≡](#) [✖](#) [✖](#) [✖](#)

```

1  {
2   |   "message": "Не заполнены обязательные поля"
3 }

```

Рисунок Г.8. – Ошибка добавления пользователя: не заполнены обязательные поля

GET <http://192.168.50.139:5000/api/users> Send

Params Auth Headers (8) Body Scripts Tests Settings [...](#)

Headers [7 hidden](#)

	Key	Value	...	Bulk Edit	Presets ...
<input checked="" type="checkbox"/>	Authorization	Bearer eyJhbGciOiJIUzI1N... ...			
	Key	Value		Description	

Body

200 OK • 29 ms • 377 B • [e.g.](#) [ooo](#)

Pretty Raw Preview Visualize JSON [≡](#) [✖](#) [✖](#) [✖](#)

```

1  [
2   {
3     "id": 3,
4     "name": "User 1",
5     "ruleId": 2
6   },
7   {
8     "id": 4,
9     "name": "User 2",
10    "ruleId": 3
11  },
12  {
13    "id": 5,
14    "name": "User 3",
15    "ruleId": 4
16  }
17 ]

```

Рисунок Г.9. – Получение пользователей

The screenshot shows a POST request to `http://192.168.50.139:5000/api/users`. The request body is JSON:

```

1 {
2   "id": 5,
3   "name": "User third"
4 }

```

The response is `200 OK` with a duration of 38 ms and a size of 320 B. The response body is:

```

1 {
2   "id": 5,
3   "name": "User third",
4   "role": "USER",
5   "ruleId": 4
6 }

```

Рисунок Г.10. – Обновление пользователя

The screenshot shows a POST request to `http://192.168.50.139:5000/api/users`. The request body is JSON:

```

1 {
2   "id": 5,
3   "name": "User 2"
4 }

```

The response is `409 Conflict` with a duration of 32 ms and a size of 354 B. The response body is:

```

1 {
2   "message": "Пользователь с таким именем уже есть"
3 }

```

Рисунок Г.11. – Ошибка обновления пользователя: повтор имени

The screenshot shows a POST request to `http://192.168.50.139:5000/api/users`. The request body is JSON:

```

1 {
2   "name": "User 2"
3 }

```

The response is `400 Bad Request` with a duration of 5 ms and a size of 359 B. The response body is:

```

1 {
2   "message": "Не указан идентификатор пользователя"
3 }

```

Рисунок Г.12. – Ошибка обновления пользователя: отсутствие идентификатора

The screenshot shows a POST request to `http://192.168.50.139:5000/api/users`. The request body is JSON:

```

1 {
2   "id": 100,
3   "name": "User 2"
4 }

```

The response is `404 Not Found` with a duration of 17 ms and a size of 330 B. The response body is:

```

1 {
2   "message": "Пользователь не найден"
3 }

```

Рисунок Г.13. – Ошибка обновления пользователя: неверный идентификатор

Изм.	Лист	№ докум.	Подп.	Дата

The screenshot shows a Postman interface with the following details:

- Method:** DELETE
- URL:** http://192.168.50.139:5000/api/users?id=5
- Headers:** (8)
- Body:** (empty)
- Query Params:**

Key	Value	Description
id	5	
Key	Value	Description
- Response:** 200 OK (Request successful. The server has responded as required.)

Рисунок Г.14. – Удаление пользователя

The screenshot shows a Postman interface with the following details:

- Method:** DELETE
- URL:** http://192.168.50.139:5000/api/users?id=100
- Headers:** (empty)
- Query Params:**

Key	Value	Description
id	100	
Key	Value	Description
- Response:** 404 Not Found ({"message": "Пользователь по указанному ID не найден"})

Рисунок Г.15. – Ошибка удаления пользователя: некорректный идентификатор

The screenshot shows a Postman interface with the following details:

- Method:** DELETE
- URL:** http://192.168.50.139:5000/api/users?id=100
- Headers:** (empty)
- Query Params:**

Key	Value	Description
id	100	
Key	Value	Description
- Response:** 400 Bad Request ({"message": "Не указан ID"})

Рисунок Г.16. – Ошибка удаления пользователя: отсутствует идентификатор

The screenshot shows a Postman interface with the following details:

- Method:** GET
- URL:** http://192.168.50.139:5000/api/users/exist?name=User 1
- Headers:** (empty)
- Query Params:**

Key	Value	Description
name	User 1	
Key	Value	Description
- Response:** 200 OK (true)

Рисунок Г.17. – Проверка нового имени пользователя: имя присутствует в базе данных

Изм.	Лист	№ докум.	Подп.	Дата	Лист
					BKR-НГТУ-09.03.01-(20-ПО)-05-2024
					77

A screenshot of a REST API response in a browser-based interface. The response status is 200 OK. The body of the response contains a single key-value pair: "name" with the value "User 100". Below the response, there are tabs for Pretty, Raw, Preview, Visualize, and JSON, along with other interface elements.

Key	Value	Description
name	User 100	
Key	Value	Description

Body 200 OK 21 ms 270 B ⏺ ⏹ ⏷ ⏸ ⏹

Pretty Raw Preview Visualize JSON ⏹ ⏻ ⏻

```
1 false
```

Рисунок Г.18. – Проверка нового имени пользователя: имя отсутствует в базе данных

A screenshot of a REST API response in a browser-based interface. The response status is 400 Bad Request. The body of the response is a JSON object with a single key "message" and the value "Не указано имя" (Name is not specified). Below the response, there are tabs for Pretty, Raw, Preview, Visualize, and JSON, along with other interface elements.

Key	Value	Description
-----	-------	-------------

Body 400 Bad Request 5 ms 316 B ⏺ ⏹ ⏷ ⏸ ⏹ ⏹

Pretty Raw Preview Visualize JSON ⏹ ⏻ ⏻

```
1 {
2   "message": "Не указано имя"
3 }
```

Рисунок Г.19. – Ошибка проверки нового имени пользователя: отсутствует поле имени

A screenshot of a POST request to the endpoint http://192.168.50.139:5000/api/lists. The request body is a JSON object with "names" (["Social Networks", "Game Sites"]) and "userId" (3). The response status is 200 OK. The response body is a JSON object with "lists" containing two items, each with "id", "name", "createdAt", "updatedAt", "userId", and "ruleId". Below the response, there are tabs for Pretty, Raw, Preview, Visualize, and JSON, along with other interface elements.

POST http://192.168.50.139:5000/api/lists Send

Params Auth Headers (10) Body Scripts Tests Settings ⏹

raw JSON Beautify

```
1 {
2   "names": [
3     "Social Networks",
4     "Game Sites"
5   ],
6   "userId": 3
7 }
```

Body 200 OK 51 ms 543 B ⏺ ⏹ ⏷ ⏸ ⏹ ⏹

Pretty Raw Preview Visualize JSON ⏹ ⏻ ⏻

```
1 {
2   "lists": [
3     {
4       "id": 1,
5       "name": "Social Networks",
6       "createdAt": "2024-09-18T00:55:21.423Z",
7       "updatedAt": "2024-09-18T00:55:21.423Z",
8       "userId": 3,
9       "ruleId": 5
10      },
11      {
12        "id": 2,
13        "name": "Game Sites",
14        "createdAt": "2024-09-18T00:55:21.431Z",
15        "updatedAt": "2024-09-18T00:55:21.431Z",
16        "userId": 3,
17        "ruleId": 6
18      }
19    ]
20 }
```

Рисунок Г.20. – Добавление списков

A screenshot of a REST API response in a browser-based interface. The response status is 200 OK. The response body is a JSON object with "names" (["Social Networks"]), "userId" (3), and "rejected" (["Social Networks"]). Below the response, there are tabs for Pretty, Raw, Preview, Visualize, and JSON, along with other interface elements.

```
1 {
2   "names": [
3     "Social Networks"
4   ],
5   "userId": 3
6 }
```

Body 200 OK 33 ms 574 B ⏺ ⏹ ⏷ ⏸ ⏹ ⏹

Pretty Raw Preview Visualize JSON ⏹ ⏻ ⏻

```
20 "rejected": [
21   "Social Networks"
22 ]
```

Рисунок Г.21. – Добавление списков: возврат отклонённого значения

Изм.	Лист	№ докум.	Подп.	Дата

Body

```

1 {
2   "userId": 3
3 }

```

400 Bad Request

Pretty Raw Preview Visualize JSON

```

1 {
2   "message": "Имена списков и идентификатор пользователя не указаны или некорректны"
3 }

```

Рисунок Г.22. – Ошибка добавления списков: отсутствие обязательного поля

Body

```

5   "userId": 100
6 }

```

400 Bad Request

Pretty Raw Preview Visualize JSON

```

1 {
2   "message": "Имена списков и идентификатор пользователя не указаны или некорректны"
3 }

```

Рисунок Г.23. – Ошибка добавления списков: некорректный идентификатор

GET http://192.168.50.139:5000/api/lists?userId=3 Send

Params Auth Headers (8) Body Scripts Tests Settings

Query Params

<input checked="" type="checkbox"/> Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/> userId	3			

Body

200 OK

Pretty Raw Preview Visualize JSON

```

1 [
2   {
3     "id": 1,
4     "name": "Social Networks",
5     "createdAt": "2024-09-18T00:55:21.423Z",
6     "updatedAt": "2024-09-18T00:55:21.423Z",
7     "userId": 3,
8     "ruleId": 5
9   },
10  {
11    "id": 2,
12    "name": "Game Sites",
13    "createdAt": "2024-09-18T00:55:21.431Z",
14    "updatedAt": "2024-09-18T00:55:21.431Z",
15    "userId": 3,
16    "ruleId": 6
17  }
18 ]

```

Рисунок Г.24. – Получение списков пользователя

Query Params

<input checked="" type="checkbox"/>	Key	Value	Description
<input checked="" type="checkbox"/>	userId	100	
	Key	Value	Description

Body **200 OK** 30 ms 267 B

Pretty Raw Preview Visualize JSON **[]**

Рисунок Г.25. – Получение списков пользователя: некорректный идентификатор

Query Params

<input type="checkbox"/>	Key	Value	Description
<input type="checkbox"/>	userId	100	
	Key	Value	Description

Body **400 Bad Request** 5 ms 398 B

Pretty Raw Preview Visualize JSON **{ "message": "Не указан идентификатор пользователя, при запросе списков" }**

Рисунок Г.26. – Ошибка получения списков: отсутствие идентификатора

PATCH http://192.168.50.139:5000/api/lists

Params Auth Headers (10) Body Scripts Tests Settings

raw JSON **Beautify**

```

1 {
2   "id": 2,
3   "name": "Video games"
4 }

```

Body **200 OK** 61 ms 397 B

Pretty Raw Preview Visualize JSON **{ "id": 2, "name": "Video games", "createdat": "2024-09-18T00:55:21.431Z", "updatedat": "2024-09-18T02:41:25.238Z", "userId": 3, "ruleId": 6 }**

Рисунок Г.27. – Обновление списка

Body **409 Conflict** 22 ms 354 B

Pretty Raw Preview Visualize JSON **{ "message": "Список с таким именем уже существует" }**

Рисунок Г.28. – Ошибка обновления списка: повтор имени

Изм.	Лист	№ докум.	Подп.	Дата

DELETE http://192.168.50.139:5000/api/lists?id=2

Params: Auth Headers (8) Body Scripts Tests Settings

Query Params:

Key	Value	Description	Bulk Edit
id	2		

Body: 200 OK 30 ms 177 B

Pretty Raw Preview 200 OK
1 Request successful. The server has responded as required.

Рисунок Г.29. – Удаление списка

Query Params:

Key	Value	Description	Bulk Edit
id	2		

Body: 404 Not Found 19 ms 318 B

Pretty Raw Preview Visualize JSON
1 {
2 | "message": "Список не найден"
3 }

Рисунок Г.30. – Ошибка удаления списка: некорректный идентификатор

Body: 400 Bad Request 16 ms 347 B

Pretty Raw Preview Visualize JSON
1 {
2 | "message": "Не указан идентификатор списка"
3 }

Рисунок Г.31. – Ошибка удаления списка: отсутствие идентификатора

GET http://192.168.50.139:5000/api/lists/exist?name=Social ...

Params: Auth Headers (8) Body Scripts Tests Settings

Query Params:

Key	Value	Description	Bulk Edit
name	Social Networks		
userid	3		

Body: 200 OK 30 ms 269 B

Pretty Raw Preview Visualize JSON
1 true

Рисунок Г.32. – Проверка имени списка на повтор: имя уже используется

Query Params:

Key	Value	Description	Bulk Edit
name	Lifestyle		
userid	3		

Body: 200 OK 28 ms 270 B

Pretty Raw Preview Visualize JSON
1 false

Рисунок Г.33. – Проверка имени списка на повтор: имя не используется

Изм.	Лист	№ докум.	Подп.	Дата

Query Params

	Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/>	name	Lifestyle			
<input type="checkbox"/>	userid	3			

Body 400 Bad Request 10 ms 392 B ⏺ ⏻ ⏹

Pretty Raw Preview Visualize JSON JSON ⏺ ⏻ ⏹

```

1 {
2   "message": "Не указаны имя списка и/или идентификатор
3   пользователя"
}

```

Рисунок Г.34. – Ошибка проверки имени списка: отсутствует обязательный параметр

POST http://192.168.50.139:5000/api/addresses Send

Params Auth Headers (10) Body JSON Scripts Tests Settings Cookies

raw JSON Beautify

```

1 {
2   "addresses": [
3     {
4       "type": "DOMAIN",
5       "value": "www.example.com"
6     },
7     {
8       "type": "IP",
9       "value": "90.90.90.90"
10    }
11  ],
12  "listId": 1
13 }

```

Body 200 OK 51 ms 576 B ⏺ ⏻ ⏹

Pretty Raw Preview Visualize JSON ⏺ ⏻ ⏹

```

1 {
2   "created": [
3     {
4       "id": 1,
5       "type": "DOMAIN",
6       "value": "www.example.com",
7       "listId": 1,
8       "ruleId": 8,
9       "updatedAt": "2024-09-18T04:13:49.292Z",
10      "createdAt": "2024-09-18T04:13:49.292Z"
11    },
12    {
13      "id": 2,
14      "type": "IP",
15      "value": "90.90.90.90",
16      "listId": 1,
17      "ruleId": 9,
18      "updatedAt": "2024-09-18T04:13:49.300Z",
19      "createdAt": "2024-09-18T04:13:49.300Z"
20    }
21  ]
22 }

```

Рисунок Г.35. – Добавление адресов

```

1 {
2   "addresses": [
3     {
4       "type": "DOMAIN",
5       "value": "www.example.com"
6     },
7     {
8       "value": "90.90.90.90"
9     }
10  ],
11  "listId": 1
12 }

```

Body 200 OK 29 ms 349 B ⏺ ⏻ ⏹

Pretty Raw Preview Visualize JSON ⏺ ⏻ ⏹

```

1 {
2   "rejected": [
3     {
4       "type": "DOMAIN",
5       "value": "www.example.com"
6     },
7     {
8       "value": "90.90.90.90"
9     }
10  ]
11 }

```

Рисунок Г.36. – Добавление адресов: возврат отклонённых адресов

Изм.	Лист	№ докум.	Подп.	Дата

```

1  {
2  |   "listId": 1
3  }

Body <v> 400 Bad Request 5 ms 356 B ⏺ ⓘ ○○○

Pretty Raw Preview Visualize JSON ⏺ 🔍 ⏺ 🔍 🔍
```

1 {
2 | "message": "Отсутствуют обязательные параметры"
3 }

Рисунок Г.37. – Ошибка добавления адресов: отсутствие обязательного параметра

```

GET http://192.168.50.139:5000/api/addresses?listId=1 Send ▾

Params • Auth Headers (8) Body Scripts Tests Settings Cookies

Query Params


| <input checked="" type="checkbox"/> Key    | Value | Descrip... | ... | Bulk Edit |
|--------------------------------------------|-------|------------|-----|-----------|
| <input checked="" type="checkbox"/> listId | 1     |            |     |           |


Body <v> 200 OK 28 ms 564 B ⏺ ⓘ ○○○

Pretty Raw Preview Visualize JSON ⏺ 🔍 ⏺ 🔍 🔍
```

1 [
2 {
3 "id": 1,
4 "type": "DOMAIN",
5 "value": "www.example.com",
6 "createdAt": "2024-09-18T04:13:49.292Z",
7 "updatedAt": "2024-09-18T04:13:49.292Z",
8 "listId": 1,
9 "ruleId": 8
10 },
11 {
12 "id": 2,
13 "type": "IP",
14 "value": "90.90.90.90",
15 "createdAt": "2024-09-18T04:13:49.300Z",
16 "updatedAt": "2024-09-18T04:13:49.300Z",
17 "listId": 1,
18 "ruleId": 9
19 }
20]

Рисунок Г.38. – Получение адресов

```

Params • Auth Headers (8) Body Scripts Tests Settings Cookies

Query Params


| <input checked="" type="checkbox"/> Key    | Value | Descrip... | ... | Bulk Edit |
|--------------------------------------------|-------|------------|-----|-----------|
| <input checked="" type="checkbox"/> listId | 100   |            |     |           |


Body <v> 200 OK 27 ms 267 B ⏺ ⓘ ○○○

Pretty Raw Preview Visualize JSON ⏺ 🔍 ⏺ 🔍 🔍
```

1 []

Рисунок Г.39. – Получение адресов: некорректный идентификатор списка

```

Body <v> 400 Bad Request 5 ms 347 B ⏺ ⓘ ○○○

Pretty Raw Preview Visualize JSON ⏺ 🔍 ⏺ 🔍 🔍
```

1 {
2 | "message": "Не указан идентификатор списка"
3 }

Рисунок Г.40. – Ошибка получения адресов: не указан идентификатор списка

The screenshot shows a POSTMAN interface. The method is set to PATCH, and the URL is http://192.168.50.139:5000/api/addresses. The 'Body' tab is selected, showing a JSON payload:

```

1 {
2   "id": 1,
3   "type": "DOMAIN",
4   "value": "www.example.ru"
5 }

```

The response status is 200 OK, with a response time of 29 ms and a body size of 417 B. The response JSON is:

```

1 {
2   "id": 1,
3   "type": "DOMAIN",
4   "value": "www.example.ru",
5   "createdAt": "2024-09-18T04:13:49.292Z",
6   "updatedAt": "2024-09-18T04:42:09.771Z",
7   "listId": 1,
8   "ruleId": 8
9 }

```

Рисунок Г.41. – Обновление адреса

The screenshot shows a POSTMAN interface. The method is set to PATCH, and the URL is http://192.168.50.139:5000/api/addresses. The 'Body' tab is selected, showing a JSON payload:

```

1 {
2   "id": 1,
3   "value": "www.error.ru"
4 }

```

The response status is 400 Bad Request, with a response time of 6 ms and a body size of 347 B. The response JSON is:

```

1 {
2   "message": "Не заполнены обязательные поля"
3 }

```

Рисунок Г.42. – Ошибка обновления адреса: отсутствие обязательного поля

The screenshot shows a POSTMAN interface. The method is set to DELETE, and the URL is http://192.168.50.139:5000/api/addresses?id=2. The 'Params' tab is selected, showing a query parameter:

Key	Value	Description	Bulk Edit
id	2		

The response status is 200 OK, with a response time of 39 ms and a body size of 177 B. The response JSON is:

```

1

```

The response message is: Request successful. The server has responded as required.

Рисунок Г.43. – Удаление адреса

The screenshot shows a POSTMAN interface. The method is set to DELETE, and the URL is http://192.168.50.139:5000/api/addresses?id=200. The 'Params' tab is selected, showing a query parameter:

Key	Value	Description	Bulk Edit
id	200		

The response status is 404 Not Found, with a response time of 20 ms and a body size of 316 B. The response JSON is:

```

1 {
2   "message": "Адрес не найден"
3 }

```

Рисунок Г.44. – Ошибка удаления адреса: некорректный идентификатор

Изм.	Лист	№ докум.	Подп.	Дата

Body 400 Bad Request 4 ms 347 B ⌂ ⌂ ⌂

Pretty	Raw	Preview	Visualize	JSON	≡	□	Q
--------	-----	---------	-----------	------	--	--	--

```

1 {
2   "message": "Не указан идентификатор адреса"
3 }

```

Рисунок Г.45. – Ошибка удаления адреса: отсутствие идентификатора

PUT http://192.168.50.139:5000/api/rules Send

Params	Auth	Headers (10)	Body	Scripts	Tests	Settings	Cookies
--------	------	--------------	------	---------	-------	----------	---------

raw	JSON	Beautify
-----	------	----------

```

1 {
2   "id": 8,
3   "access": "DENY",
4   "schedule": "ON"
5 }

```

Body 200 OK 8 ms 316 B ⌂ ⌂ ⌂

Pretty	Raw	Preview	Visualize	JSON	≡	□	Q
--------	-----	---------	-----------	------	--	--	--

```

1 {
2   "rule": {
3     "id": 8,
4     "access": "DENY",
5     "schedule": "ON"
6   }
7 }

```

Рисунок Г.46. – Обновление правила

PUT http://192.168.50.139:5000/api/rules Send

Params	Auth	Headers (10)	Body	Scripts	Tests	Settings	Cookies
--------	------	--------------	------	---------	-------	----------	---------

raw	JSON	Beautify
-----	------	----------

```

1 {
2   "id": 8,
3   "add": [
4     {
5       "days": "MTWTF",
6       "time": {
7         "from": "08:00",
8         "to": "17:00"
9       }
10    }
11  ]
12 }

```

Body 200 OK 52 ms 355 B ⌂ ⌂ ⌂

Pretty	Raw	Preview	Visualize	JSON	≡	□	Q
--------	-----	---------	-----------	------	--	--	--

```

1 {
2   "exceptions": [
3     {
4       "id": 5,
5       "days": "MTWTF",
6       "time": {
7         "from": "08:00",
8         "to": "17:00"
9       },
10      "ruleId": 8
11    }
12  ]
13 }

```

Рисунок Г.47. – Обновление правила: добавление исключений

Изм.	Лист	№ докум.	Подп.	Дата

PUT http://192.168.50.139:5000/api/rules

Body (10) Params Auth Headers Cookies Scripts Tests Settings

raw JSON

```

1 {
2   "id": 8,
3   "update": [
4     {
5       "id": 5,
6       "days": "FAS",
7       "time": {
8         "from": "12:00",
9         "to": "21:00"
10      }
11    }
12  ]
13 }

```

Body (10) 200 OK 35 ms 353 B

Pretty Raw Preview Visualize JSON

```

1 {
2   "exceptions": [
3     {
4       "id": 5,
5       "days": "FAS",
6       "time": {
7         "from": "12:00",
8         "to": "21:00"
9       },
10      "ruleId": 8
11    }
12  ]
13 }

```

Рисунок Г.48. – Обновление правила: обновление исключений

PUT http://192.168.50.139:5000/api/rules

Body (10) Params Auth Headers Cookies

raw JSON

```

1 {
2   "id": 8,
3   "remove": [
4     {
5       "id": 5
6     }
7   ]
8 }

```

Body (10) 200 OK 28 ms 267 B

Pretty Raw Preview 200 OK Request successful. The server has responded as required.

Рисунок Г.49. – Обновление правила: удаление исключений

```

1  {
2   "rule": {
3     "id": 8,
4     "access": "DENY",
5     "schedule": "ON"
6   },
7   "exceptions": [
8     {
9       "id": 6,
10      "days": "MTWHF",
11      "time": {
12        "from": "08:00",
13        "to": "17:00"
14      },
15      "ruleId": 8
16    }
17  ]
18 }

```

Рисунок Г.50. – Получение правила

```

1  {
2   "message": "Правило по указанному идентификатору не найдено"
3 }

```

Рисунок Г.51. – Ошибка получения правила: некорректный идентификатор

```

1  {
2   "message": "Не указан идентификатор правила"
3 }

```

Рисунок Г.52. – Ошибка получения правила: отсутствие идентификатора

```

1  {
2   "isActive": false,
3   "domain": "proxy.home",
4   "port": "8443"
5 }

```

Рисунок Г.53. – Получение сведений прокси-сервера

POST http://192.168.50.139:5000/api/proxy

Params Auth Headers (9) Body Scripts Tests Settings Cookies

Body

Pretty Raw Preview Visualize JSON

```

1 {
2   "isActive": true
3 }

```

200 OK 5 ms 284 B

Рисунок Г.54. – Обновление состояния прокси-сервера

PUT http://192.168.50.139:5000/api/proxy

Params Auth Headers (9) Body Scripts Tests Settings Cookies

Body

Pretty Raw Preview

200 OK

Request successful. The server has responded as required.

Рисунок Г.55. – Обновление конфигурации прокси-сервера

```

acl auth proxy_auth REQUIRED
acl u3 proxy_auth 'User 1'
acl a1 dstdomain www.example.ru
acl l1 any-of a1
acl r8 time MTWHF 08:00-17:00
acl u4 proxy_auth 'User 2'
http_access deny !auth
http_access deny !r8 a1 u3
http_access allow auth
http_access deny all

```

Рисунок Г.56. – Сгенерированные параметры конфигурации прокси-сервера

DELETE http://192.168.50.139:5000/api/proxy

Params Auth Headers (8) Body Scripts Tests Settings Cookies

Body

Pretty Raw Preview Visualize JSON

```

1 {
2   "message": "Система успешно сброшена"
3 }

```

200 OK 572 ms 327 B

Рисунок Г.57. – Сброс системы

Изм.	Лист	№ докум.	Подп.	Дата

Приложение Д. Иллюстрационное сопровождение экспериментальной части

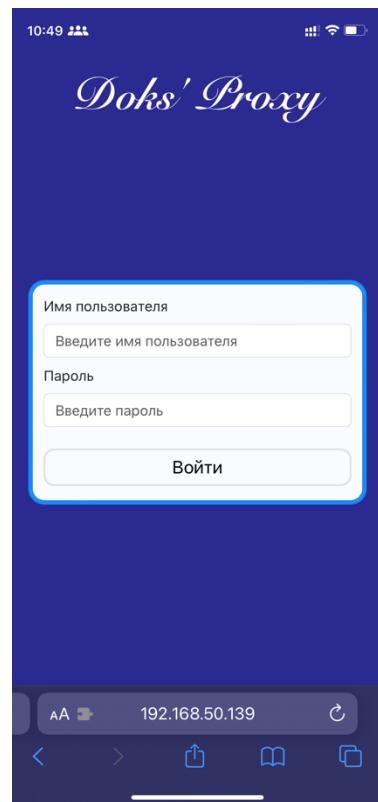


Рисунок Д.1. – Страница авторизации в браузере

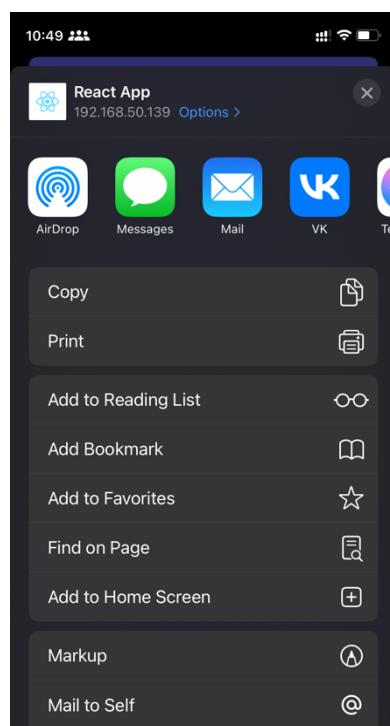


Рисунок Д.2. – Меню страницы браузера

Изм.	Лист	№ докум.	Подп.	Дата

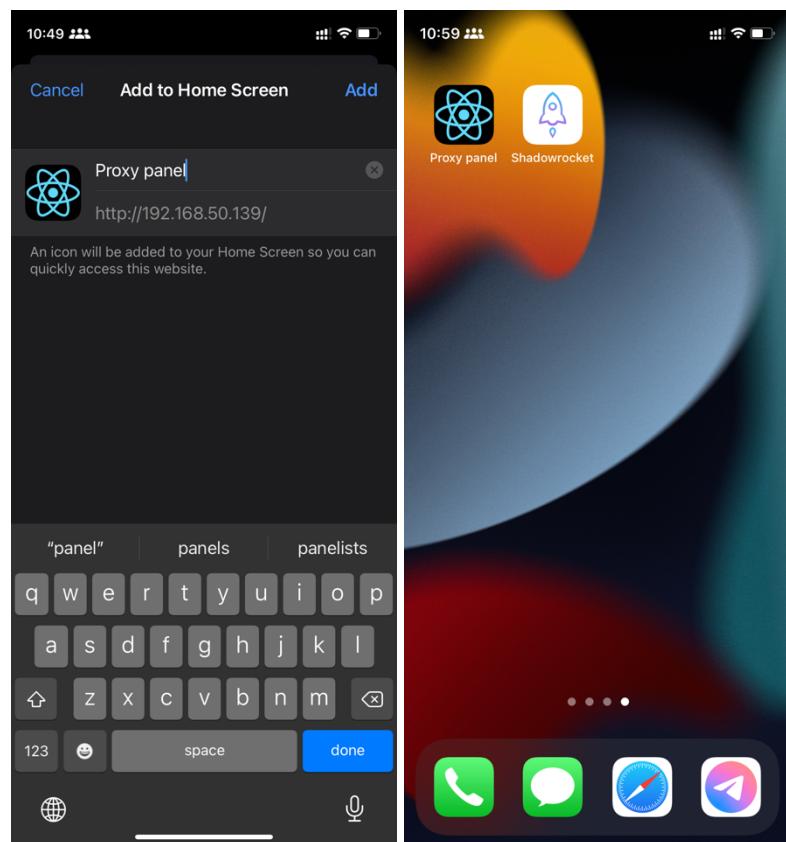


Рисунок Д.3. – Добавление PWA-приложения на домашний экран

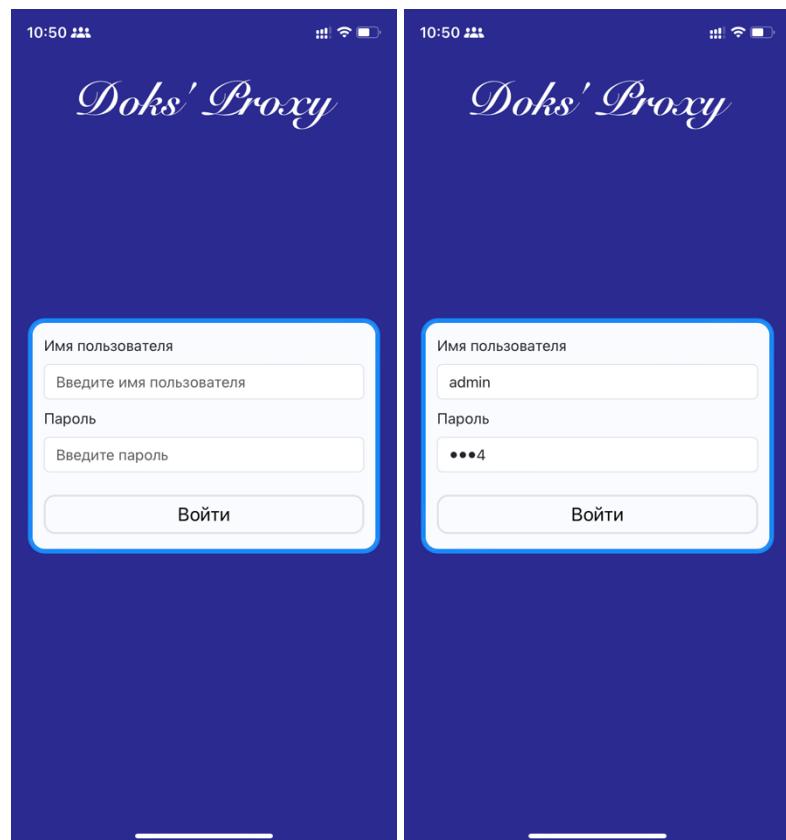


Рисунок Д.4. – Авторизация в PWA-приложении

					Лист
					90
Изм.	Лист	№ докум.	Подп.	Дата	<i>BKP-HГТУ-09.03.01-(20-ПО)-05-2024</i>

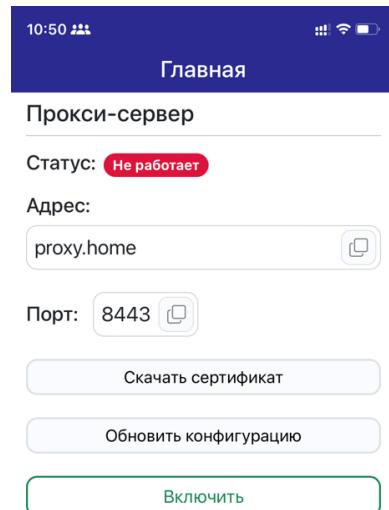


Рисунок Д.5. – Главная страница веб-приложения

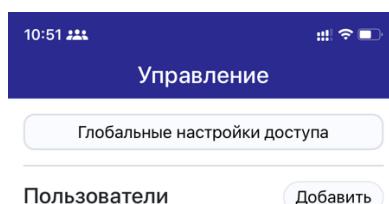


Рисунок Д.6. – Страница управления настройками прокси

Изм.	Лист	№ докум.	Подп.	Дата	ВКР-НГТУ-09.03.01-(20-ПО)-05-2024	

Лист 91

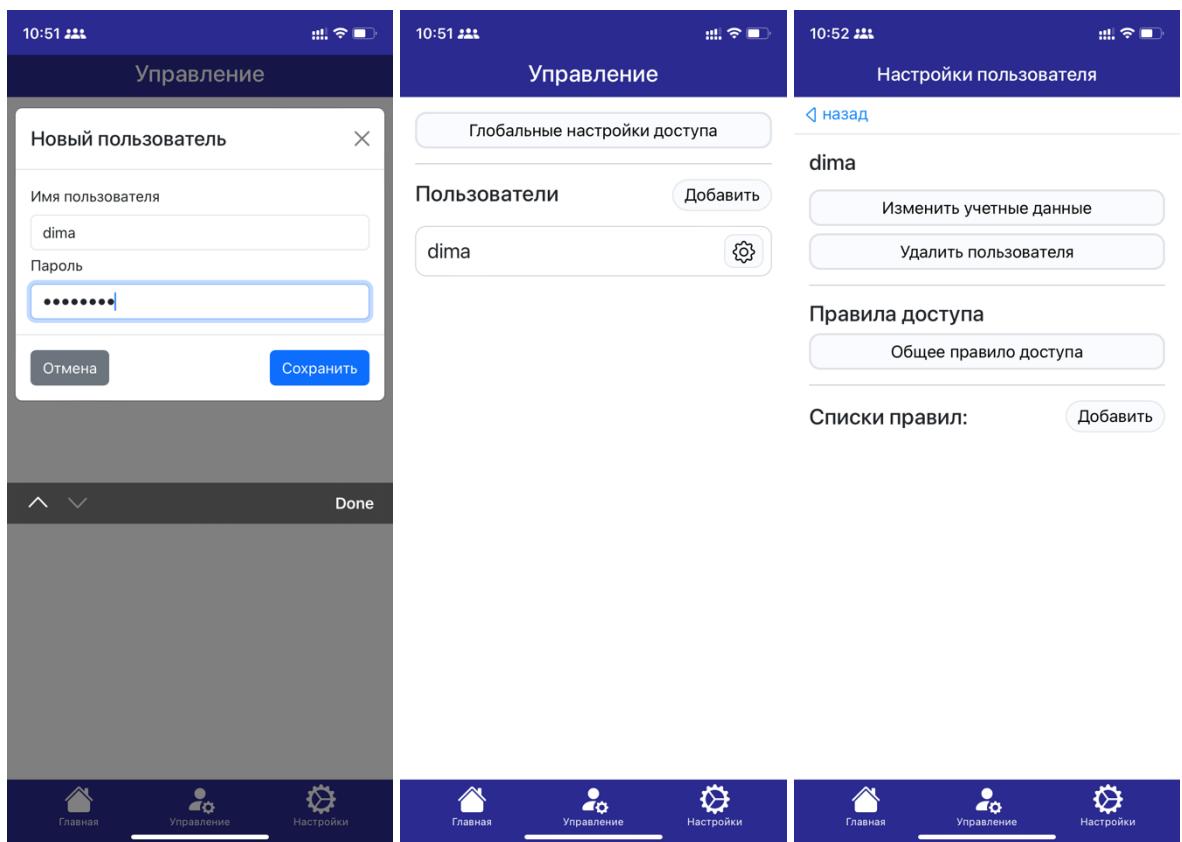


Рисунок Д.7. – Добавление нового пользователя

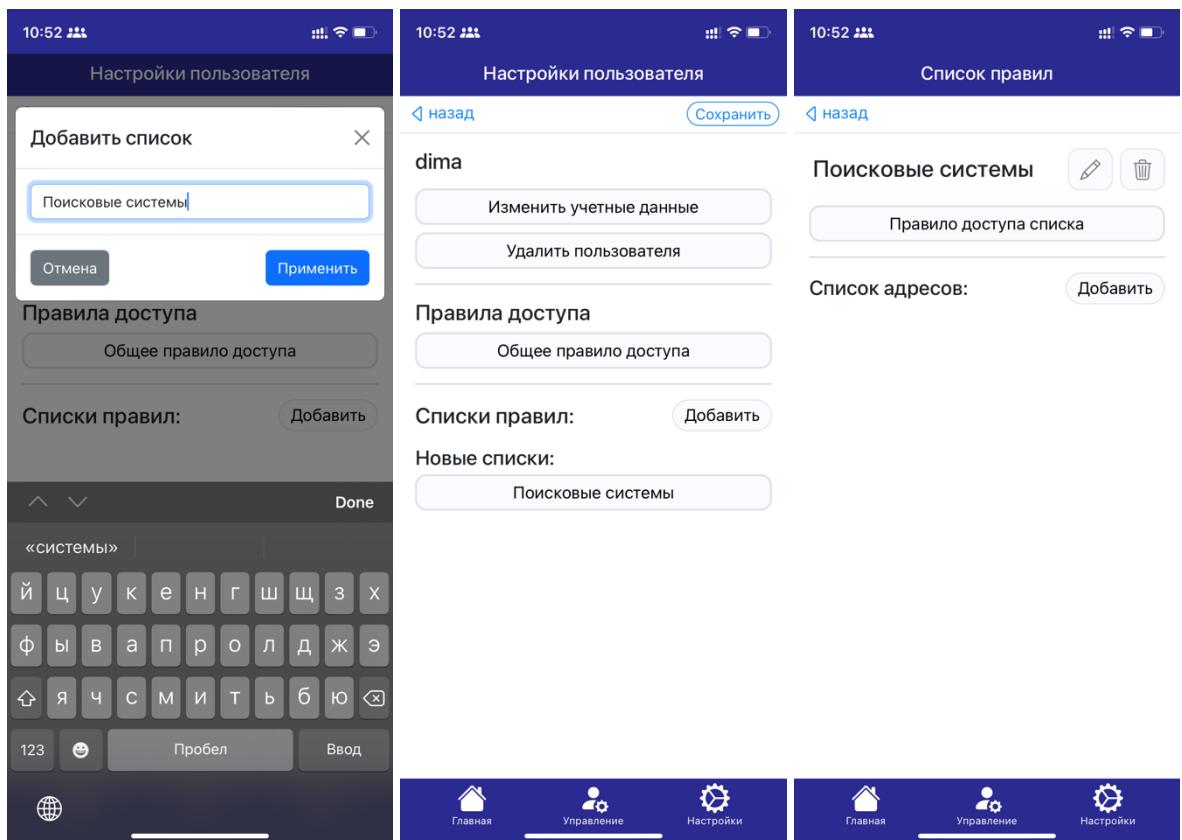


Рисунок Д.8. – Добавление нового списка

Изм.	Лист	№ докум.	Подп.	Дата

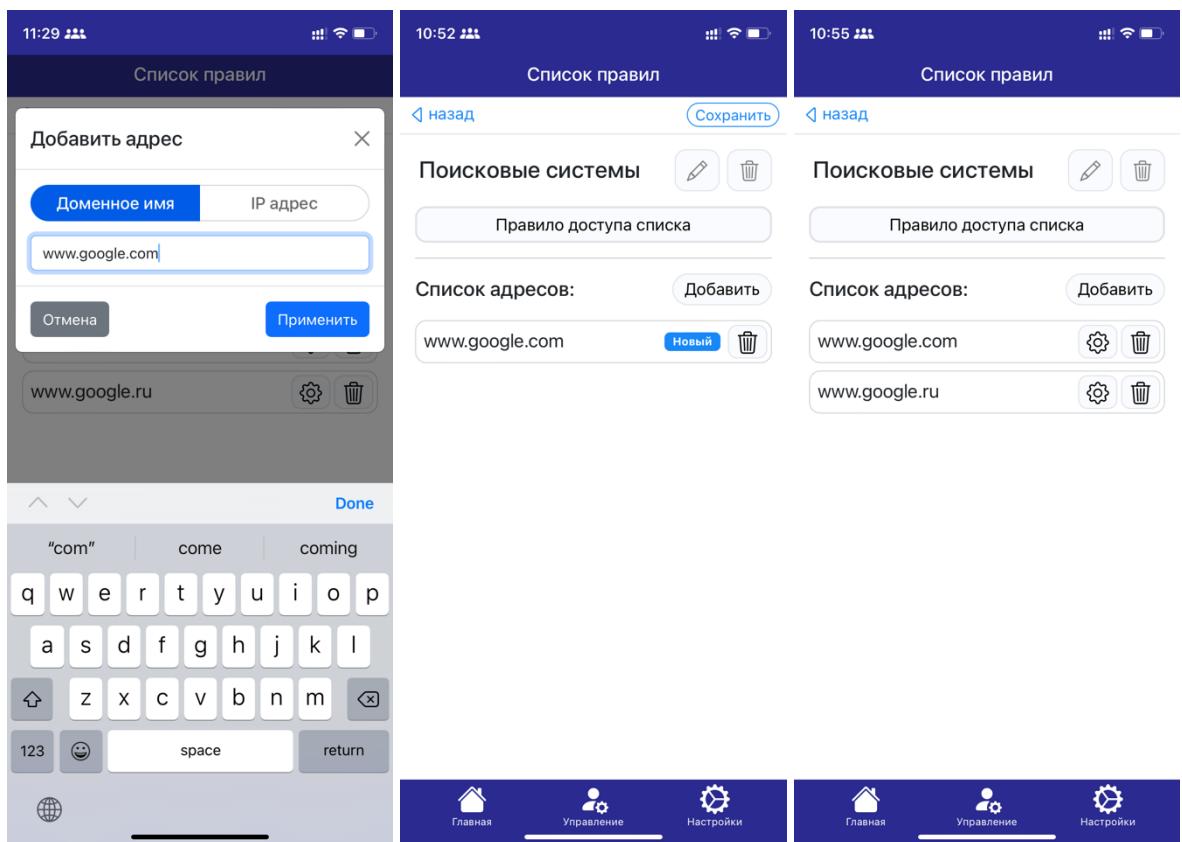


Рисунок Д.9. – Добавление новых адресов

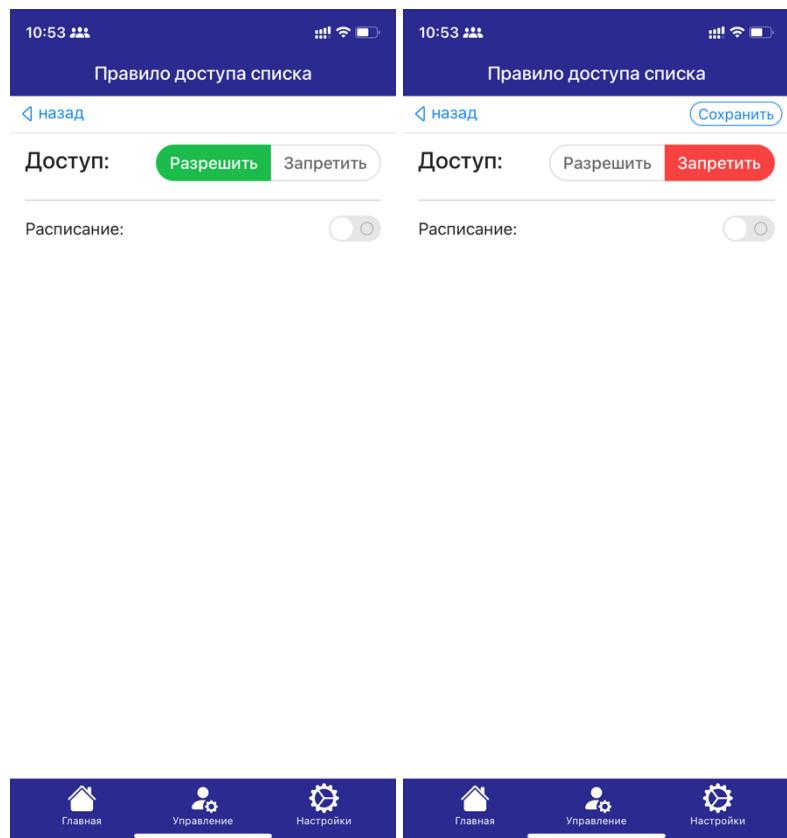


Рисунок Д.10. – Изменение правила доступа списка

Изм.	Лист	№ докум.	Подп.

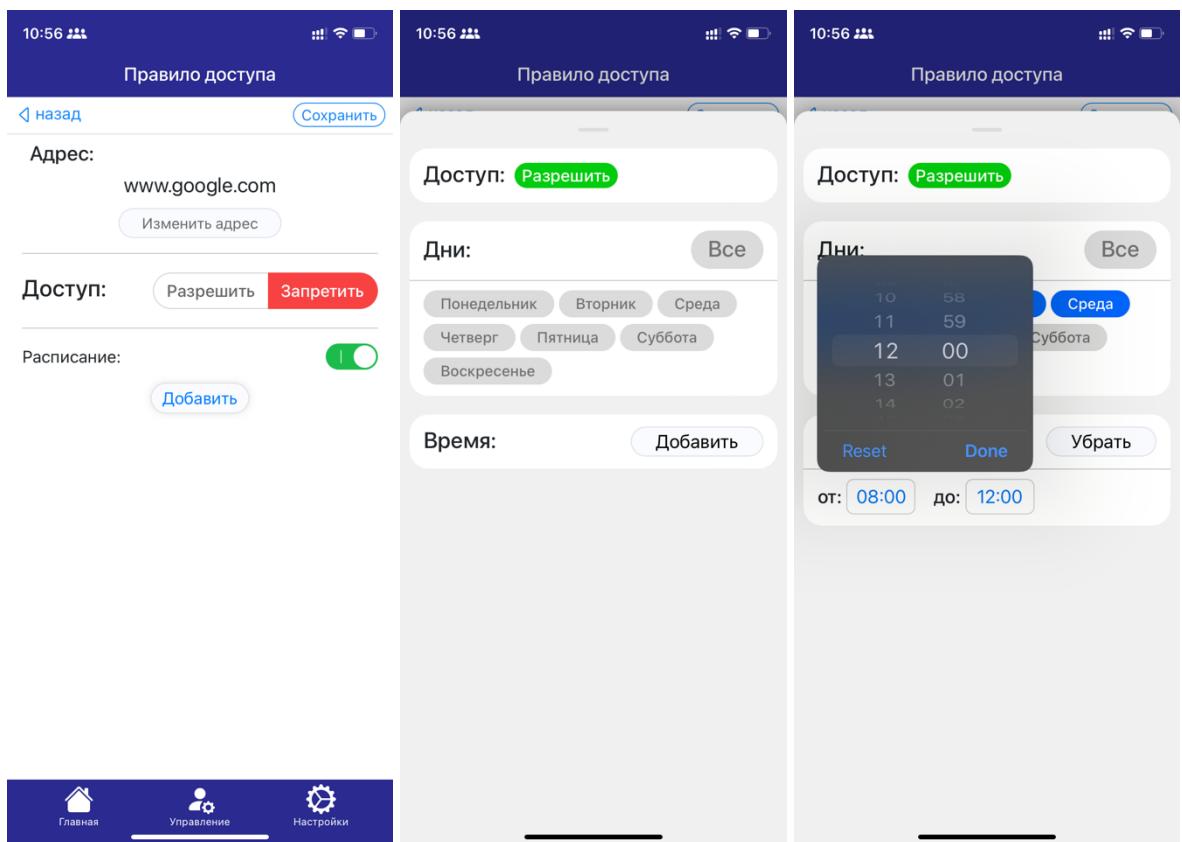


Рисунок Д.11. – Добавление исключения к правилу доступа адреса

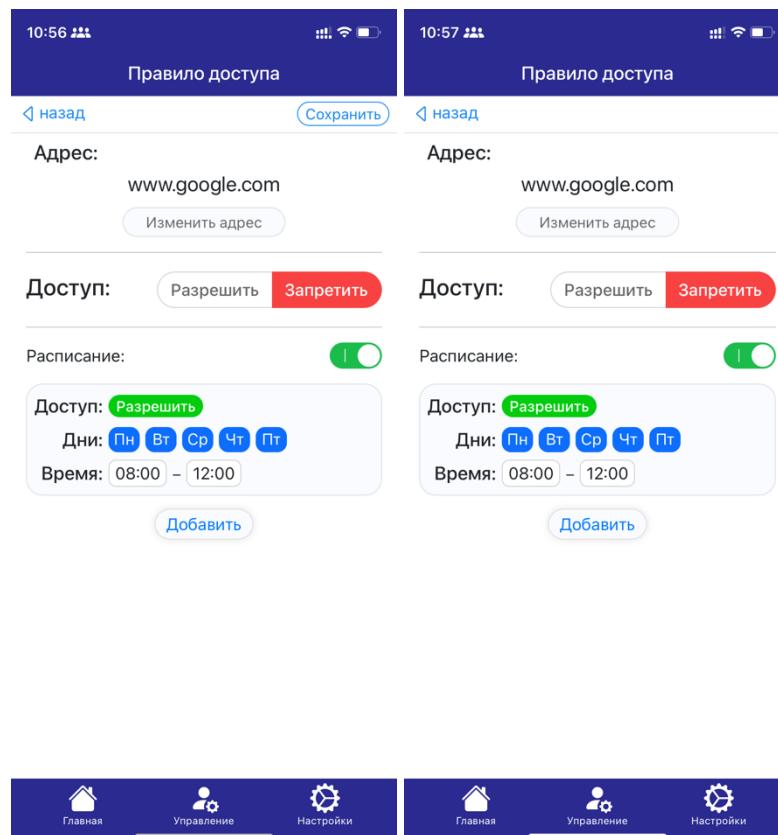


Рисунок Д.12. – Сохранение исключения

Изм.	Лист	№ докум.	Подп.	Дата

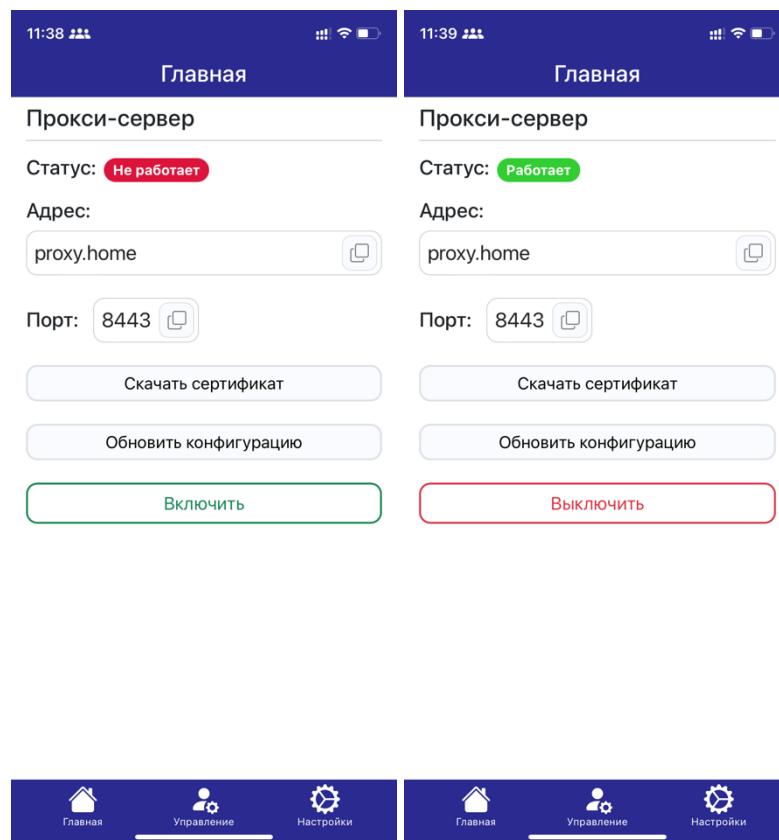


Рисунок Д.13. – Обновление состояния прокси-сервера

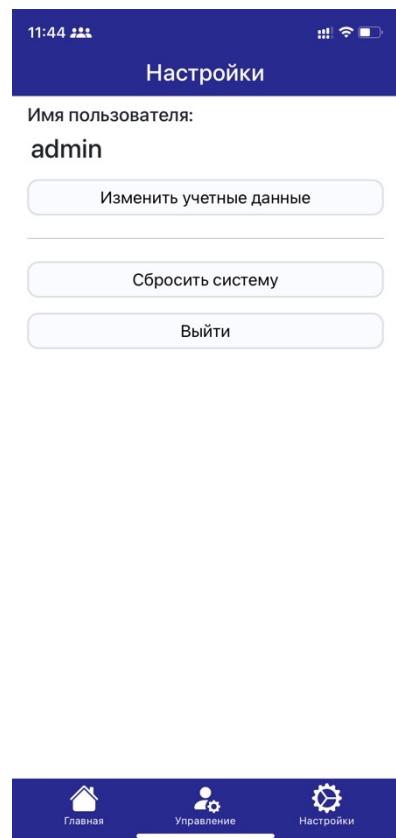


Рисунок Д.14. – Страница настроек и выхода из профиля

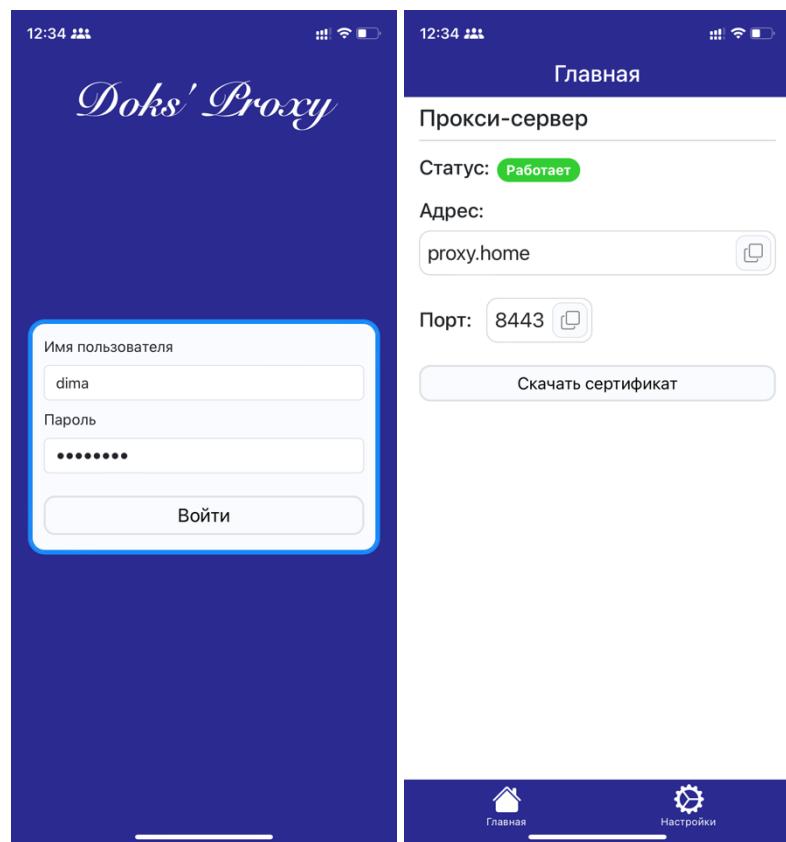


Рисунок Д.15. – Авторизация под аккаунтом пользователя прокси-сервера

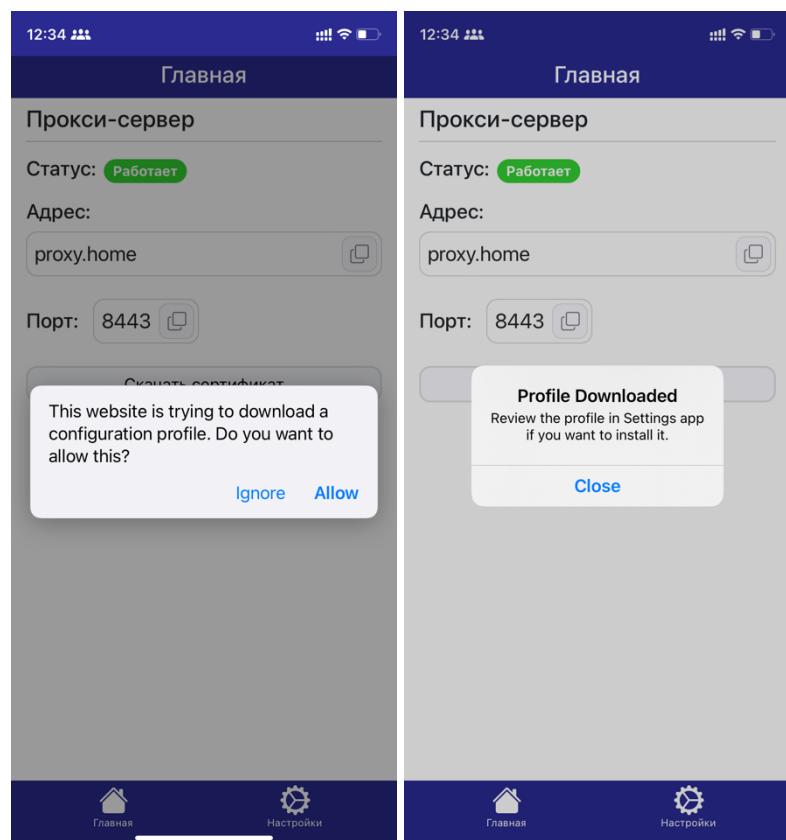


Рисунок Д.16. – Загрузка сертификата прокси-сервера

Изм.	Лист	№ докум.	Подп.	Дата

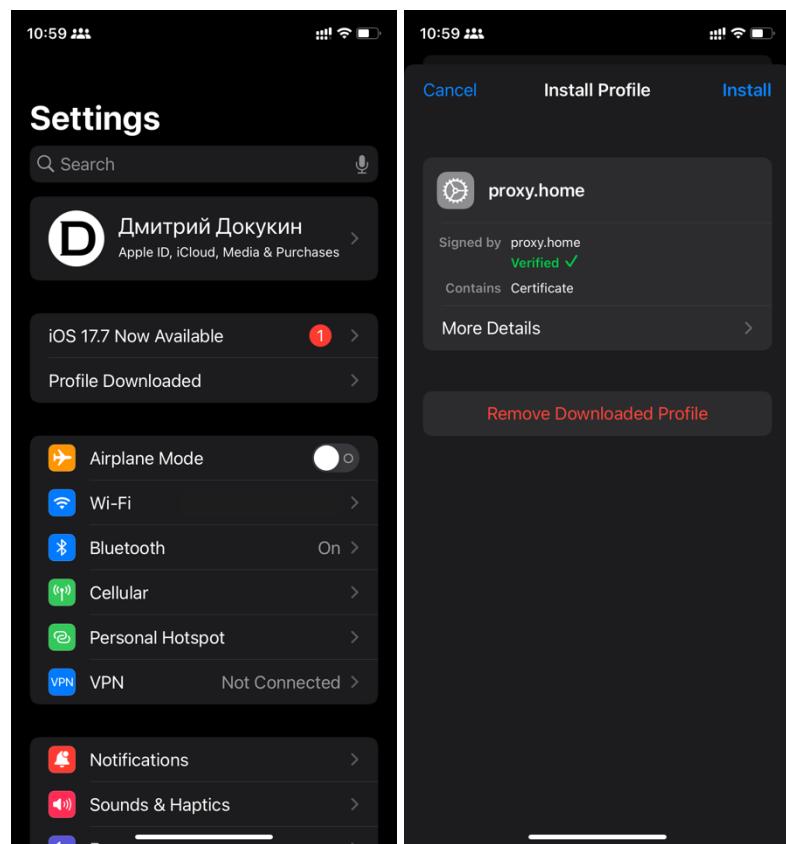


Рисунок Д.17. – Добавление сертификата в настройки операционной системы

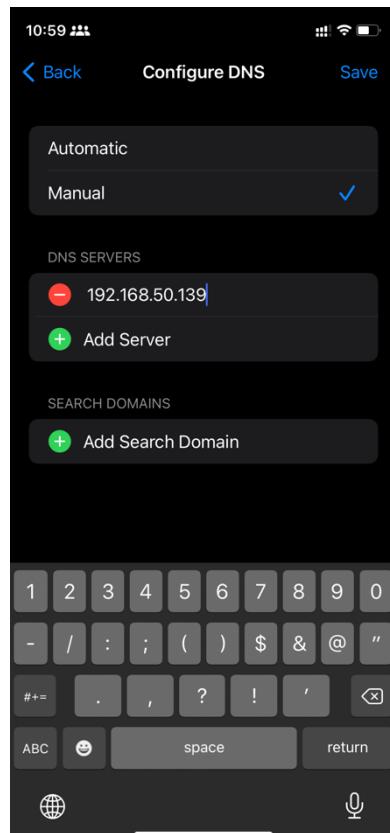


Рисунок Д.18. – Добавление локального DNS-сервера в настройки Wi-Fi

Изм.	Лист	№ докум.	Подп.	Дата

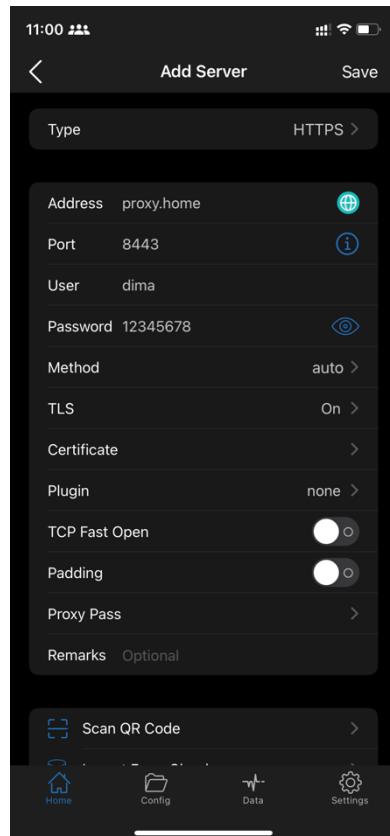


Рисунок Д.19. – Создание подключения к прокси-серверу в приложении Shadowrocket



Рисунок Д.20. – Демонстрация VPN-иконки соединения с прокси

Изм.	Лист	№ докум.	Подп.	Дата
------	------	----------	-------	------

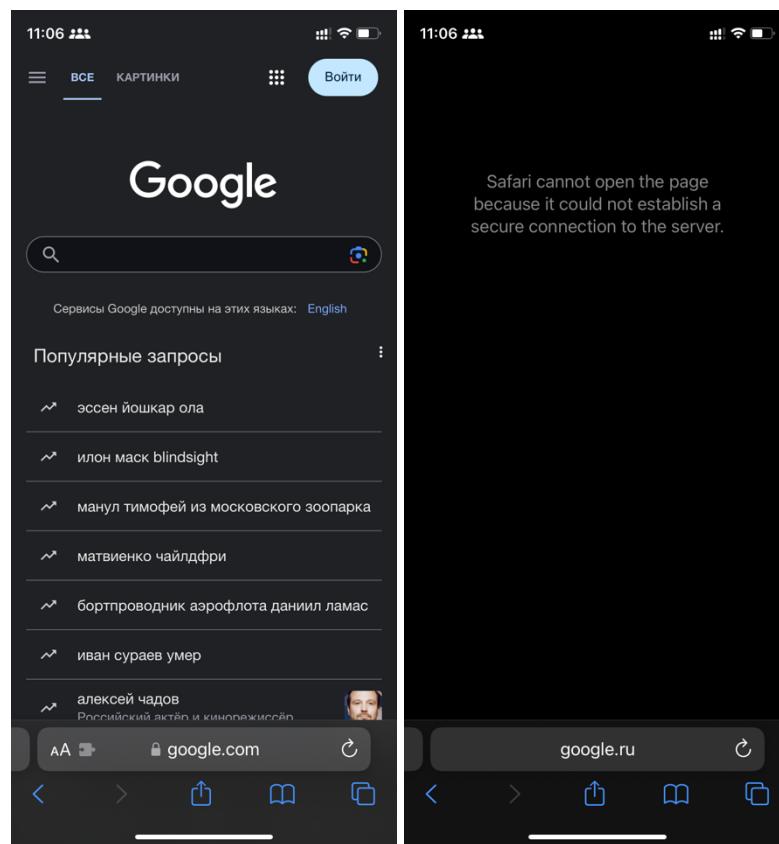


Рисунок Д.21. – Демонстрация работы правил доступа прокси-сервера