**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

ПРОЕКТИРОВАНИЕ VPN-СЕРВЕРА. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

РЕАЛИЗАЦИЯ VPN-СЕРВЕРА. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

ПРОЕКТИРОВАНИЕ ГЛАВНОГО СЕРВЕРА. . . . . . . . . . . . . . . . . . . . . . . . . . . 14

РЕАЛИЗАЦИЯ ГЛАВНОГО СЕРВЕРА. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16

ПРОЕКТИРОВАНИЕ САЙТА. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21

РЕАЛИЗАЦИЯ САЙТА. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22

РЕАЛИЗАЦИЯ ДЕСКТОПНОГО ПРИЛОЖЕНИЯ. . . . . . . . . . . . . . . . . . . . . . . 24

ЗАКЛЮЧЕНИЕ. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26

ИСТОЧНИКИ. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27

ПРИЛОЖЕНИЯ. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28

**ВВЕДЕНИЕ**

В современном мире российские пользователи сети «интернет» сталкиваются со значительными ограничениями – блокировками различных ресурсов. Причем исходить они могут как от властей РФ, так и от самих зарубежных сервисов («между молотом и наковальней»). В таких условиях актуальной становится разработка VPN-сервиса, обеспечивающего туннелирование траффика до серверов в иных юрисдикциях, обеспечивающих отсутствие ограничений.

Разрабатываемое приложение лицензируется согласно MPLv2[1] – Mozilla Public License 2.0. В отличии от GPLv3[2] - General Public License 3.0, которая, в случае заимствования кода, требует опубликовать весь проект под ней же, данная лицензия требует сохранить лицензию только для заимствованного кода, а не для всего проекта. В целом, MPLv2 выбрана как одна из самых свободных лицензий, которая, тем не менее, запрещает использовать опубликованный под ней код в закрытом исходном коде.

В настоящем проекте выделяется несколько логических частей, которые можно выделить как по технологиям, так и по «бизнес-задачам». На текущем этапе они включают: VPN-сервера (т.н. ноды), главный сервер, сайт с информацией о проекте, клиентские приложения. Предполагаемой к использованию процедурой установления VPN-соединения предлагается следующее: клиент, после авторизации и получения JWT-токенов, регистрирует свой девайс на главном сервере, сообщая ему свой публичный ключ, а также запрашивает список доступных нод; клиент запрашивает у главного сервера подключение к выбранной ноде; главный сервер, сначала удостоверившись, что клиент отключен от предыдущей ноды, проведя валидацию: проверив лимиты, соответствие уровня доступа пользователя запрашиваемой ноде и удостоверившись в доступности последней, передаёт ей публичный ключ пользователя; нода, после валидации ключа (с целью защиты от инъекций), выполняет его добавление, выделяет ему адрес в приватной сети, после чего возвращает главному серверу свой публичный ключ и выделенный адрес; главный сервер добавляет к полученному от ноды ответу IP-адрес ноды и порт, прослушиваемый сервером Wireguard, после чего возвращает всю полученную модель данных клиенту; клиент устанавливает Wireguard-туннель.

Визуальное представление взаимодействий размещено на схеме 1, где полупрозрачные прямоугольники обозначают отдельные операционные системы, сплошные – docker/compose сборки в случае с серверами и лишь отдельные части приложения в случае с клиентами. Сплошные стрелки обозначают соединения по протоколу TCP, штриховые – по UDP.



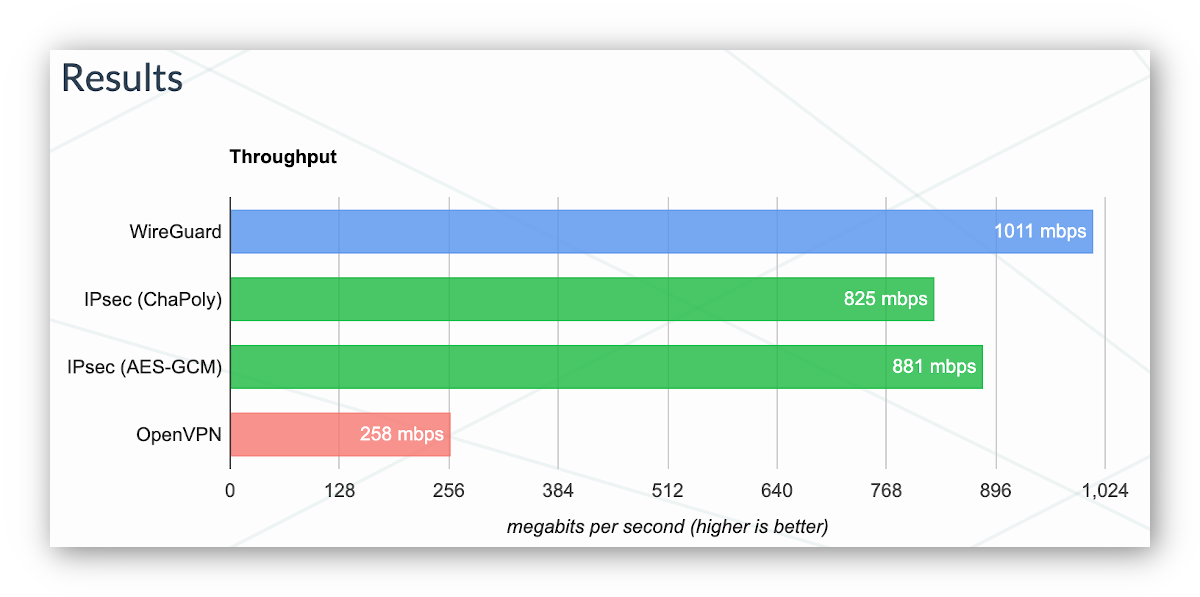
Схема 1

Формирование технологического стека выполнялось, в первую очередь, с учетом наличия у разработчика опыта в конкретных технология, а также их пригодности для решения поставленной задачи. В результате были выбраны следующие технологии:

1. WireGuard – непосредственно протокол туннелирования (установления приватной сети). Превосходит любой другой распространенный современный протокол по пропускной способности и объему служебного траффика. Также является очень простым в конфигурировании. В целом оправдывает собственный слоган: «extremely simple yet fast and modern». Дополнительным преимуществом является использование современного криптографического алгоритма – ChaCha20-Poly1305, в сравнении с AES256\_GCM – слабейшим из достаточных шифров на базе AES, обеспечивающего лучшую производительность, а также экономию заряда батареи и более высокую производительность (по некоторым оценкам[10] – в 3 раза) на мобильных устройствах, которые не имеют аппаратного ускорения AES. Критерий выбора – превосходство над аналогами и наличие опыта.
2. ASP.NET – реализация любого API-интерфейса по всему проекту. Не уступает прочим современным аналогичным технологиям. Критерий выбора – наличие опыта.
3. PostgreSQL – реализация любой базы данных по всему проекту. Неоспоримый лидер среди баз данных по функционалу, хотя по скорости может незначительно уступать MySQL. Позволяет применять как реляционную модель, так и модель Key/Value на подобии Redis/MongoDB с применением типа JSONB (Json Binary), при том превосходя вышеназванные по скорости работы. Критерий выбора – выдающаяся функциональность и наличие опыта.
4. ReactTS – реализация браузерного интерфейса. Представляет собой самый распространённый фронтенд-фреймворк. Язык TypeScript (TS) является диалектом JavaScript и позволяет применять в нём строгую типизацию, что не предполагалось в оригинале. Критерий выбора – общепризнанность и наличие опыта.
5. NGINX – реализация любого ревёрс-проксирования по всему проекту. Представляет собой объективно лучший прокси-сервер общего назначения на момент разработки. Критерий выбора – превосходство над аналогами и наличие опыта.
6. Alpine-Linux / Docker-Compose – развертывание любого серверного приложения. Контейнеризация является современным стандартом изоляции приложений. Реализация ОС Linux – Alpine является одной из самых легковесных на сегодняшний день, построена на Си-библиотеке MUSL и занимает в готовом виде 5 (пять) МБ, в отличии от, например, популярной Ubuntu, построенной на GLIBC, занимающей 200 (двести) МБ. Критерий выбора – общепризнанность и наличие опыта.
7. WPF.NET – реализация десктопного приложения. Является не лучшим выбором, в частности, в качестве аналога, позволяющего, в том числе, переиспользовать код с веб-сайта, можно было бы выбрать electron – фреймворк, использующий движок Chromium в качестве средства рендера, по сути, позволяя запускать приложения в окне браузера, у которого удалены адресная строка, закладки, прочие меню и т.п., а оставлена лишь область отрисовки приложения, при этом такое «веб-приложение» имеет полный доступ к локальной машине. Естественно, такая концепция позволяет использовать для разработки приложения любой веб-фреймворк, в том числе, вышеупомянутый ReactTS, переиспользуя уже написанный код, что сокращает работу, по приблизительным оценкам автора, втрое. Однако WPF выбран из-за ограниченности ресурсов проекта – отсутствия времени на изучение. Критерий выбора – исключительно наличие опыта.
8. InnoSetup – упаковщик десктопного приложения. Одна из самых распространенных и мощных утилит для создания установщиков путем написания соответствующего скрипта. Критерий выбора – общепризнанность.

Предлагается назвать приложение Vdb (Vpn-Developed-Badly), а в качестве логотипа использовать латинскую букву V на синем фоне.

**ПРОЕКТИРОВАНИЕ VPN-СЕРВЕРА**

 В первую очередь поясним упомянутое во введении превосходство протокола Wireguard над остальными. На изображении 1 представлено сравнение пропускной способности наиболее распространенных протоколов. Дополнительно замечу, что Wireguard использует в качестве протокола шифрования ChaCha20-Poly1305, обеспечивающий экономии заряда батареи на мобильных устройствах, ибо потребляет значительно меньше ресурсов на архитектуре ARM, хотя, как и AES, не имеет на ней аппаратного ускорения.

Изображение 1

Данная глава описывает прототипирование базовой единицы разрабатываемого проекта. Сам VPN сервер представляет из себя Wireguard сервис и ASP-приложение, которое, по запросу, выполняет с ним определенные манипуляции – добавляет и удаляет пиры, возвращает список клиентов и удаляет неактивных. Данные приложения запускаются в одном контейнере, основывающемся на образе Alpine Linux – полноценной операционной системе архитектуры MUSL, занимающей в готовом виде 5 МБ, в отличии от, например, Ubuntu – занимающей «из коробки» 200 МБ. Изучение документации[6][7]8] Docker показало, что разумным вариантом является запуск двух приложений в одном контейнере, нежели их разделение, т.к. в этом случае возникают проблемы с получением доступа из одного контейнера в другой, а также нарушается в целом идея docker-контейнеризации, которая подразумевает взаимодействие с контейнерами по веб-протоколам (в т.ч. HTTP, RPC). Проблема состоит в том, что Wireguard не отслеживает обновление файла конфигурации. Таким образом, если ASP и может изменить файл конфигурации Wireguard (путем, например, использования docker volumes), то это не привет к обнаружению и применению этого изменения. Наивным решением могла бы быть регулярная перезагрузка файла конфигурации, например с применением cronjob, однако это однозначно более медленно, нежели стандартный способ добавления пиров «на лету», т.е. через выполнение команды ‘wg set wg0 peer KEY allowed-ips IP’.

Функционал Wireguard является стандартным и состоит в прослушивании порта 51820 (стандартный порт wg), маскировании трафика (англ. masquerade), перенаправлении на внешний адрес. Пиры должны добавляться и удаляться по CL-команде от ASP-приложения.

Функционал ASP-приложения состоит в принятии запросов от удаленного сервера, проверке действительности API-ключа, выполнении действия. Само приложение, как предполагается, должно хранить ключи в виде их SHA512-хеша, не храня оригиналы, т.к. это создает концептуальную уязвимость и не соответствует принципу secure-by-design.

В контейнер также устанавливается NGINX, выполняющий функции шифрования TCP-траффика, ограничения запросов в секунду и применения белого списка IP-адресов, поддержания keep-alive соединений с ASP. Белый список может быть использован, если IP главного сервера заранее известен, а значит TCP-соединения со всех остальных адресов можно отклонять «без разбора», что обеспечивает высочайшую устойчивость в DoS-атакам, в то время как Wireguard устойчив к DoS архитектурно:

*In fact, the server does not even respond at all to an unauthorized client; it is silent and invisible. The handshake avoids a denial of service vulnerability created by allowing any state to be created in response to packets that have not yet been authenticated.*  - wireguard.com

**РЕАЛИЗАЦИЯ VPN-СЕРВЕРА**

**Разработка WebAPI**

С помощью Powershell создадим проекты типов classlib (для управления Wireguard) и webapi (для получения команд), после чего создадим файлы dockerignore, gitignore и инициализируем git-репозиторий – см. приложение 1А.

Для начала введем базу для дальнейшей разработки. Из предыдущих проектов импортируем класс провайдера настроек[3], повышающий абстракцию использования файла appsettings.json и прочих, позволяя использовать их как набор C#-объектов в классах остальных сервисов. Дополнительно создадим класс, извлекающий переменные среды – см. приложение 2А.

Теперь опишем сервис, отвечающий за выдачу адресов. Данный сервис должен реализовывать отношения рода один-к-одному, т.е. один уникальный публичный ключ к одному уникальному IP-адресу. При необходимости повторного использования можно было бы разработать собственный класс «двунаправленного словаря», т.е. в котором, помимо ключа, каждое значение также выступает ключом и не может повторяться. Для хранения адреса используется Int32, байты из которого извлекаются методом BitConverter.GetBytes. Само хранение реализовано словарем (ключ к адресу) и дополнительным HashSet’ом, хранящим только адреса. Сам по себе данный сервис способен адресовать до (250\*256\*256). Первая цифра адреса каждого клиента выбрана за 10 (согласно RFC 1918[4]), вторая – за 6 в связи с техническими ограничениями Alpine Linux, особого значения это не имеет.

*The Internet Assigned Numbers Authority (IANA) has reserved the*

*following three blocks of the IP address space for private internets:*

*10.0.0.0 - 10.255.255.255 (10/8 prefix)*

*172.16.0.0 - 172.31.255.255 (172.16/12 prefix)*

*192.168.0.0 - 192.168.255.255 (192.168/16 prefix).*

- RFC 1918, февраль 1996

Маска подсети выбрана за /32, т.е. весь адрес относится к узлу, подсети отсутствуют, что является стандартной практикой конфигурирования Wireguard, ибо мы хотим развернуть туннель, а не сеть. Когда сервису необходимо выдать клиенту адрес – он в первую очередь пытается сгенерировать адрес на основании текущего количества клиентов, проще говоря, пытается угадать, какой адрес свободен. К сожалению, одни клиенты могут оставаться подключенными длительное время, другие же – отключаться спустя секунды, что, в действительности, порождает случайное распределение занятых адресов по адресному пространству. Это приводит к необходимости каждый раз проверять свободность того или иного адреса. В целом, данные операции не должны существенно снижать производительность, так как и поиск, и добавление в HashSet’а имеет сложность О(1). При удалении пира его публичный ключ и выделенный адрес также должны удаляться из соответствующих коллекций (что, к сожалению, будет «стоить» уже О(n)), таким образом происходит освобождение адресов. Отдельным вопросом является синхронизация реального состояния Wireguard-сервиса с WebAPI, соответствие коих, как можно понять после небольшого размышления, должна гарантироваться, в противном случае отсутствует даже возможность корректно выделять адреса, не говоря уже о более сложных функциях программы. Например, клиент может отключиться от сервера по причине потери у клиента электропитания машины, что приведет к тому, что де-факто пир будет отключен, но об этом не будет сообщено ASP-приложению. Тогда ASP-приложение должно регулярно проверять список пиров и удалять те, что уже не подключены, для этого реализован метод SyncState – принимающий «реальной состояние» и сверяющийся с ним. Код некоторых методов сервиса представлен в кратком варианте – см. приложение 3А.

Следующим этапом являет разработка авторизации. Она представляет собой т.н. промежуточное ПО (англ. middleware), предшествующее на конвейере обработки запросом конечной точке (англ. endpoint). Для начала нам потребуется сервис, который будет валидировать ключи, вычисляя их хеш и сверяя с переданным путем секретного json-файла. Его код представлен в приложении 4А – он имеет достаточно простую реализацию. Задача middleware состоит в извлечении значения заголовка Authorization, передаче его в сервис, возврат семантически корректного кода ошибки (согласно RFC 9110[5]) либо же передача запроса далее по конвейеру (в случае успешной валидации), частичный код представлен в приложении 4А.

Теперь разработаем «манипулятор» WireGuard сервером. Он представляет собой надстройку над исполнением обычных CL-команд. Выполнение осуществляется асинхронно с точками возврата управления (команда await) после запуска процесса и после начала чтения потока вывода вызванной команды. Его частичный код представлен в приложении 5А.

Последним этапом разработки являются непосредственно конечные точки (англ. endpoints) разрабатываемого WebAPI. Привести их все на бумажном носителе невозможно. Отмечу, что к каждой точке была добавлена валидация wg-ключа с целью защиты от инъекции с исполнением произвольного кода.

**Конфигурация NGINX**

Поскольку ASP имеет возможность авторизации по API-ключу, который передается в заголовке Authorize в формате base64, что является кодированием, а не шифрованием, необходимо защитить подключение. В конфигурации установлены только современные протоколы TLS – 1.2 и 1.3 а также запрещены все слабые шифры[6], из разрешенных – «ChaCha20\_Poly1305» и «AES256\_GCM\_SHA384». Помимо вышеназванного обеспечивается поддержание keep\_alive соединений с WebAPI, ограничение частоты запросов или разрешение запросов только с определенного адреса на основании переданных в контейнер переменных среды.

**Конфигурация Wireguard**

В конфигурации WG применим наиболее простую из обеспечивающих полную функциональность конфигурацию – программный код 1. Замечу, что PrivateKey оставлен пустым и генерируется при запуске уже непосредственно в контейнере.Применение iptables потребует выдачи контейнеру некоторых привилегий (NET\_ADMIN), ибо изначально контейнер не имеет доступа к настройкам сети. Директива MASQUERADE означает маскировку траффика, как будто он исходит от сервера, а не от удаленного клиента посредствам туннеля.

Программный код 1

[Interface]

Address = 10.0.0.0/8

PostUp = iptables -I FORWARD -i wg0 -j ACCEPT;

PostUp = iptables -t nat -I POSTROUTING -o eth0 -j MASQUERADE;

ListenPort = 51820

PrivateKey =

**Конфигурация Docker и сборка образа**

Как и планировалось, мы базируем свой образ на Alpine Linux – одной из самый легковесных разновидностей Linux, полноценный образ которой веси 5 МБ. Мы применяем так называемую multi-stage build: создаем контейнер, в который устанавливаем SDK, там производим компиляцию, после чего создаём финальный контейнер, куда копируем уже собранное приложение, а в нём добавляем только Runtime, что уменьшает финальную сборку – программный код 2.

Программный код 2

FROM alpine:3 AS base

FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build

COPY ./vdb\_node\_api /vdb\_node\_api

COPY ./vdb\_node\_wireguard\_manipulator /vdb\_node\_wireguard\_manipulator

RUN dotnet publish /vdb\_node\_api/vdb\_node\_api.csproj -c "Release" -r linux-musl-x64 --no-self-contained -o /app/publish

FROM base AS final

COPY --from=build /app/publish /app

RUN apk add -q --no-progress nginx

RUN apk add -q --no-progress openssl

RUN apk add -q --no-progress wireguard-tools

RUN apk add -q --no-progress aspnetcore7-runtime

COPY ./build\_alpine/pre-setup.sh ./etc/rest2wg/pre-setup.sh

COPY ./build\_alpine/pre-wg0.conf ./etc/rest2wg/pre-wg0.conf

COPY ./build\_alpine/pre-nginx-limit\_req.conf.template ./etc/rest2wg/pre-nginx-limit\_req.conf.template

COPY ./build\_alpine/pre-nginx.conf/ ./etc/nginx/nginx.conf

COPY ./build\_alpine/pre-ssl-params.conf ./etc/nginx/snippets/ssl-params.conf

COPY ./build\_alpine/pre-self-signed.conf ./etc/nginx/snippets/self-signed.conf

ENV ASPNETCORE\_ENVIRONMENT=Production

ENV REST2WG\_LIMIT\_REQ=100000;

CMD ["bash", "-c",

"chmod +x /etc/rest2wg/pre-setup.sh && /etc/rest2wg/pre-setup.sh"]

**Точка входа контейнера**

Рассмотрим точку входа контейнера, коей является bash-скрипт ‘pre\_setup.sh’. Предшествующая ему команда ‘chmod +x /etc/rest2wg/pre-setup.sh’ исправляет потенциальную неспособность оболочки распознать файл как исполняемый. Данный скрипт выполняет несколько задач. Он генерирует приватный ключ Wireguard, если таковой отсутствует; генерирует self-signed x509 сертификат, если таковой отсутствует, для шифрования траффика посредствам NGINX; исправляет значения переменных среды, если пользователь задал их некорректно; запускает приложения. Файл частично представлен в программном коде 3.

Программный код 3

**if** ! test -e "/etc/wireguard/wg0.conf"; then

cp /etc/rest2wg/pre-wg0.conf /etc/wireguard/wg0.conf

wg genkey >> /etc/wireguard/wg0.conf

fi

**if** ! test -e /etc/ssl/private/nginx-selfsigned.key then

openssl req -x509 -nodes -days 36500 -newkey rsa:2048   
 -subj "/CN=US/C=US/L=Miami" -keyout /etc/ssl/private/nginx-selfsigned.key   
 -out /etc/ssl/certs/nginx-selfsigned.crt

fi

**if** !((REST2WG\_LIMIT\_REQ > 0)) && ((REST2WG\_LIMIT\_REQ <= 9999999)); then

REST2WG\_LIMIT\_REQ=100000

fi

**if** ! test -e "/etc/nginx/snippets/nginx-limit\_req.conf"; then

cp /etc/rest2wg/pre-nginx-limit\_req.conf.template   
 /etc/nginx/snippets/nginx-limit\_req.conf

echo "${REST2WG\_LIMIT\_REQ}r/s;" >> /etc/nginx/snippets/nginx-limit\_req.conf

fi

**if** (testvar='all' && [[ $REST2WG\_ALLOWED\_IP = $testvar ]]); then

unset testvar;

**elif** !(ipcalc -n "${REST2WG\_ALLOWED\_IP}"); then

REST2WG\_ALLOWED\_IP="all";

fi

**if** ! test -e "/etc/nginx/snippets/while\_list.conf"; then

echo "allow ${REST2WG\_ALLOWED\_IP}; deny all;"

> /etc/nginx/snippets/while\_list.conf

fi

echo "Spinning up the Nginx reverse-proxy..."

nginx

echo "Spinning up the Wireguard service..."

wg-quick up wg0 && wg show wg0

echo "Spinning up the ASP WebAPI..."

dotnet /app/vdb\_node\_api.dll --no-launch-profile

tail -f /dev/null

**ПРОЕКТИРОВАНИЕ ГЛАВНОГО СЕРВЕРА**

Главный сервер в результате должен представлять собой Docker/Compose сборку из двух контейнеров – бекэнд приложения ASP.NET и базы данных PostgreSQL. Секретные файлы планируется передавать с использованием Docker/Secrets, а данные базы данных вынести в т.н. «Docker/Volume».

**Проектирование базы данных**

|  |  |  |
| --- | --- | --- |
| Название поля | Тип | Комментарий |
| Пользователь (User) | | |
| Id | int32 | Primary Key |
| IsAdmin | Boolean | Является ли пользователь администратором. |
| Email | varchar(50) | Alternate Key |
| IsEmailConfirmed | Boolean | Подтверждена ли почта. |
| PasswordSalt | bytea (byte[512/8]) | Случайные 512 бит. |
| PasswordHash | batea (byte[512/8]) | SHA512 от объединения пароля и «соли». |
| PayedUntil | timestamp with  time zone | Определяет время, до которого пользователь имеет платный доступ. |
| RefreshTokens Entopies | long[] | Энтропии выданных refresh JWT, служащие для их инвалидации по запросу пользователя. |
| Пользовательское устройство (UserDevice) | | |
| Id | int32 | Primary key |
| UserId | int32 | Foreign Key |
| Wireguard PublicKey | varchar(256/8\*4/3+3),  т.е. длинны base64-кодированного wg-ключа. | Alternate Key. В теории может повториться  (с шансом N к 2256), гарантирование уникальности потребует дополнительной валидации и на сервере, и на клиенте. |
| LastConnected NodeId | int32 (nullable) | Последняя нода, на которой было замечено устройство. |
| LastSeenUtc | timestamp with time zone (nullable) | Время, когда устройство коммуницировало с главным сервером последний раз. |

В разработке базы данных я не буду ориентироваться на некоторые «правильные» подходы, которым меня обучал в РГУ им. А.Н,Косыгина некий Смирнов Е.Е., ведь, как известно, «база данных в третьей нормальной форме может и лучше обычной, но в реальных приложениях она не выступает» - Рыжов Р.В., а в действительности я буду руководствоваться желанием реализовать удобное и быстрое извлечение, в первую очередь – ориентируясь на уникальные ключи. Наша база данных, в целом, будет иметь две модели – «пользователь» и «пользовательское устройство» со следующими полями – таблица 1.

Таблица 1

**Проектирование API**

Теперь перейдем к бекэнд приложению. Исходя из стоящих перед приложением задач, оно должно иметь следующие контроллеры: аутентификации, сущности устройства, подключения. В таблице 2 представлены предусмотренные на текущем этапе конечные точки. Отмечу, что всем URN должен предшествовать путь /api/, что позволяет гарантированного отличить обращения к API от обращения клиенту. Главный сервер оперирует базой данных, за которую предлагается выбрать PgSQL в связи с наличием опыта.

|  |  |  |
| --- | --- | --- |
| URN | Метод | Действие сервера и комментарий |
| Контроллер аутентификации (AuthController) | | |
| /auth | GET | Всегда отвечает кодом 200 (ОК), если пользователь авторизован. Служит для проверки клиентом своего статуса. |
| /auth /sessions | GET | Возвращает количество записей энтропий рефреш-токенов из БД для данного пользователя. |
| /auth | POST | Выполняет вход по логину и паролю с получением JWT. |
| /auth | PUT | Выполняет регистрацию по логину и паролю с получением JWT. |
| /auth  /password | PATCH | Выполняет смену пароля на переданный. |
| /auth /refresh | PATCH | Выполняет обновление refresh и access токенов на основании действительного refresh-токена. |
| /auth | DELETE | Выполняет удаление всех энтропий рефреш-токенов из базы данных, тем самым приводя их в невалидное состояние. |
| /auth /self | DELETE | Выполняет удаление собственной энтропии сессии, требуется для функции выхода из приложения. |
| Контроллер подключений (ConnectionController) | | |
| /connection /nodes-list | GET | Возвращает список всех доступных нод. Эндпоинт не требует авторизации и поэтому его ответ может быть кеширован. |
| /connection | PUT | После осуществления валидации, добавляет переданный публичный ключ на ноду согласно переданному идентификатору. |
| Контроллер устройств (DeviceController) | | |
| /device /user-devic es-limits | GET | Возвращает лимиты устройств для каждого уровня доступа пользователя. Эндпоинт не требует авторизации и поэтому его ответ может быть кеширован. |
| /device | GET | Получает список всех устройств данного пользователя. |
| /device | PUT | Добавляет пользователю новое устройство. |
| /device | PATCH | Делает то же, что и описанное ниже, но в нарушение RFC 9110, однако позволяя при этом передавать ключ в теле запроса. |
| /device  /{pkB64} | DELETE | Удаляет устройство согласно переданному идентификатору в виде публичного ключа. |

Таблица 2

**РЕАЛИЗАЦИЯ ГЛАВНОГО СЕРВЕРА**

Ранее упомянутым способом с помощью Powershell создадим структуру решения: один проект webapi, один проект xunit, два проекта classlib – для доступа к данным и работы с сервисами. В данной работе не будет рассматриваться тестирование с применением xunit.

**Реализация слоя доступа к данным**

Поскольку в настоящей работе не предполагаются сложные запросы к реляционной базе данных, применим EntityFramework (далее – EF) – самую популярную ORM (Object Relational Maping) для ASP, обеспечивающую абстрактную работу с базой данных, как с CLR-типами. Перенесем упомянутую выше структуру базы данных в CLR-типы на языке C# с применением описанных в документации[11] EF. Помимо моделей данных создадим контекст данных, в котором укажем лишь правила, связанные с уникальностью некоторых ключей – программный код 4, в то время как правила, связанные, например, с максимальной длинной того или иного поля, указываются в аннотациях к свойствам в самих моделях.

Программный код 4

modelBuilder.Entity<User>(entity => {

entity.HasAlternateKey(u => u.Email);

});

modelBuilder.Entity<UserDevice>(entity => {

entity.HasAlternateKey(u => u.WireguardPublicKey);

});

**Реализация слоя сервисов**

В данной работе выделяется необходимость создания некоторых сервисов. Во-первых, описанные в главе о VPN-сервере сервисы провайдера настроек и провайдера значений переменных среды мы переиспользуем. Сервис хеширования паролей будет скопирован из предыдущих проектов, ранее также упомянутых[3]. Вновь разработанными же сервиса станут: сервис управления VPN-нодами и сервис кеширования состояния VPN-нод.

|  |  |
| --- | --- |
| Имя константы | Значение |
| protocol | @"https" |
| peersControllerPath | @"api/peers" |
| statusControllerPath | @"api/status" |

Рассмотрим только сервис VPN-нод, тогда как сервис кеширования состояний является небольшой «надстройкой», внедряющей зависимость от вышеназванного сервиса. Рассматриваемый класс несколько нарушает принцип единственной обязанности, но, поскольку данная курсовая работа предполагается не для Смирнова Е.Е. – преподавателя РГУ им. А.Н. Косыгина, которому «нечего вам сказать, если вы делаете лишь бы сдать», мы понадеемся, что читающему таки будет что сказать и опустим данную недоработку для, возможно, дальнейшего исправления. Настоящая реализация представляет собой наследника класса Microsoft.Extensions.Hosting.BackgroundService, представляющего собой, как следует из названия, «фоновый сервис». Это позволяет должным образом внедрять от него зависимости. Данный сервис, помимо того, что регулярно вызывает собственный метод «PingAllNodesAsync», который отправляет запрос к каждой ноде с целью проверки его статуса (в отличии от сервиса кеширования состояний, не запрашивая, например, число подключенных клиентов, а лишь проверяя доступность), также содержит в себе и методы, не требующие фонового существования сервиса – что и является главной проблемой, признаком плохого разделения обязанностей. И все же, приведем в таблице 3 константы, а в таблице 4 – некоторые методы, содержащиеся в данном сервисе.

Таблица 3

|  |  |
| --- | --- |
| Имя и сигнатура метода (-ов) | Комментарий |
| GetPeersPathForNode(  VpnNodeInfo nodeInfo, bool withCleanup) GetStatusPathForNode (  VpnNodeInfo nodeInfo) GetAlternatePathForNode(  VpnNodeInfo nodeInfo) | Создает готовый URL для обращения к определенному эндпоинту определенной ноды. |

Таблица 4 (начало)

|  |  |
| --- | --- |
| VpnNodesService(  SettingsProviderService settingsProvider,   ILogger<VpnNodesService> logger) | Конструктор. Инициализирует экземпляры HTTP-клиентов с заданными параметрами аутентификация, протоколом TLSv1.2. |
| async Task<bool> CheckIsNodeAccessible(   string nodeName) | Выполняет асинхронный запрос с выбранной ноде с целью установления её статуса. |
| async Task<WgShortPeerInfo[]?>  GetPeersFromNode(string nodeName) | Асинхронно запрашивает у ноды текущий список пиров и возвращает его. |
| async Task<AddPeerResponse?>  AddPeerToNode(  string peerPubkey, string nodeName) | Запрашивает у ноды добавление публичного ключа. В случае успеха возвращает её ответ, иначе – null. |
| async Task<bool> RemovePeerFromNode(  string peerPubkey, string nodeName) | Запрашивает у ноды удаление пира. Возвращает bool, отражающий успешность. |
| override async Task ExecuteAsync(  CancellationToken stoppingToken) | С заданным периодом отправляет нодам команду на удаление давно отключившихся пиров. |
| async Task PingAllNodesAsync(  CancellationToken cancellationToken,   int millisecondsInterval) | С заданным интервалом осуществляет проверку статуса каждой ноды. |

Таблица 4 (продолжение)

Этот и прочие классы внедряются в WebAPI используя стандартный инструментарий внедрения зависимостей ASP – программный код 5.

Программный код 5

builder.Services.AddSingleton<EnvironmentProvider>();

builder.Services.AddSingleton<SettingsProviderService>();

builder.Services.AddSingleton<JwtService>();

builder.Services.AddSingleton<VpnNodesService>();

builder.Services.AddHostedService(pr   
 => pr.GetRequiredService<VpnNodesService>());

builder.Services.AddSingleton<VpnNodesStatusService>();

builder.Services.AddHostedService(pr   
 => pr.GetRequiredService<VpnNodesStatusService>());

builder.Services.AddDbContext<VpnContext>(opts => {

opts.UseNpgsql(builder.Environment.IsDevelopment()

? builder.Configuration["ConnectionStrings:LocalhostConnection"]

: builder.Configuration["ConnectionStrings:DatabaseConnection"]

, opts => { opts.MigrationsAssembly(nameof(main\_server\_api)); }); });

**Реализация WebAPI**

Сами конечные точки были описаны еще на этапе проектирования. Их реализация не представляет сложности и я не буду тратить на это бумагу. Лишь немного раскрою детали выдачи JWT. Код метода представлен в приложении 1Б. В первую очередь мы генерирует access-токен. Если refresh не требуется, то на этом исполнение метода заканчивается. В противном случае мы проверяем количество уже выданных токен, в случае превышение лимит – выполняет сжатие массива в 2 раза, удаляя наиболее старые токены. Далее создает модель, содержащая два утверждения – Id пользователя и энтропия. Они передаются JWT-сервису, который кодирует токен и подписывает его секретным симметричным ключом по алгоритму HMAC SHA512. Далее, в зависимости от запроса, токен записывается либо тело ответа, либо в HttpOnly-куки, т.е. куки, недоступные через JavaScript, с установкой флага Secure, т.е. «передаваемый только по защищенному соединению» и флага SameSite/Strict, т.е. запрещающего передачу данного значения куда-либо за пределы конкретного домена нижнего уровня.

**Конфигурация NGINX**

Конфигурация NGINX похожа на таковую у VPN-сервиса. И если тогда мы не показали сам код конфигурации, то на это есть причина – она является более простой, чем представляемая здесь – в программном коде 6, и это сделано во избежание дублирования.

Программный код 6

worker\_processes 1; events { worker\_connections 8192; }

http {

include snippets/self-signed.conf; include snippets/ssl-params.conf;

include snippets/nginx-limit\_req.conf; include snippets/while\_list.conf;

upstream http\_backend {

server 0.0.0.0:5000;

keepalive 32; keepalive\_time 1h; keepalive\_timeout 5m; keepalive\_requests 2048;

}

proxy\_http\_version 1.1;

limit\_req zone=first\_zone burst=30 nodelay;

proxy\_cache\_path /data/nginx/cache keys\_zone=cache\_zone:15m;

proxy\_cache cache\_zone;

server {

listen 5001 ssl http2; listen 5002;

location / {

proxy\_pass <http://http_backend>; }

location /api/connection/nodes-list {

proxy\_pass http://http\_backend;

proxy\_cache\_valid any 1m; }

location /api/device/user-devices-limits {

proxy\_pass http://http\_backend;

proxy\_cache\_valid any 10m; }

}}

**Конфигурация Docker, сборка образа и точка входа**

Поскольку сборка контейнера и точка входа главного сервера незначительно отличаются от таковых для VPN-сервера они не будут разбираться отдельно. Лишь добавлю, что в WebAPI была встроена переменная среды «VDB\_GENERATE\_JWT\_SIG» включающая или отключающая генерацию случайного 512-битного секретного ключа подписи для подписи JWT при первом запуске контейнера. Рассмотрим часть файла точки входа, отвечающего за это – программный код 7. Отмечу, что автору известно о существовании envsubst, отказ от него связан с техническими сложностями.

Программный код 7

**if** test "$VDB\_GENERATE\_JWT\_SIG" = "true"; then

**if** ! test -e "/run/secrets/generated\_sig.json"; then  
 echo "{\"GeneratedSigningKey\":{\"SigningKeyBase64\":  
 \"$(head -c 64 /dev/random | base64 -w 0)\"}}"   
 > /run/secrets/generated\_sig.json

**fi**

**fi**

**Создание Docker/Compose сборки**

В программном коде 8 представлен частичный код docker-compose.yml.

Программный код 8

secrets:

  backSecs:

    file: ./backendsecrets.json

  backNodes:

    file: ./backendnodes.json

volumes:

  vdb\_main\_server\_database:

    external: false  
services:

  vdb\_main\_server\_api:

    image: luminodiode/vdb\_main\_server\_api:latest

    environment:

      - VDB\_GENERATE\_JWT\_SIG=true

    secrets:

      - source: backSecs

        target: aspsecrets.json

      - source: backNodes

        target: nodes.json

    ports:

      - 127.0.0.1:5080:5002/tcp

  database:

    image: postgres:15-alpine

    environment:

      POSTGRES\_PASSWORD: some\_password\_here

      POSTGRES\_DB: vdb\_main\_server\_db

    volumes:

      - vdb\_main\_server\_database:/var/lib/postgresql/data/

**ПРОЕКТИРОВАНИЕ САЙТА**

Поскольку разработка сайта является наиболее тривиальной из задач, его проектирование начнем первым.

Разработка сайта является наиболее тривиальной из задач. Начнем с дизайна. Сайт представляет собой несколько страниц: главная страница – информация о проекте и статус серверов, «загрузки», «регистрация и вход», «личный кабинет». Навигация осуществляется по меню сверху.

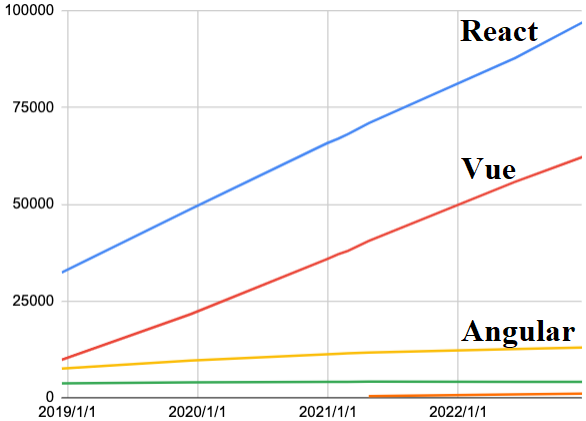
 Сайт разрабатывается c применением React и языка TypeScript в связи с их распространённостью и актуальностью. На схеме 2 изображена динамика количества npm-пакетов, зависимых от соответствующих фреймворков за последние 3 года. При разработке предлагается применять компонентный подход с написанием обобщенных компонентов. Предлагается отказаться от использования т.н. контекстов, предлагаемых фреймворком, в пользу использования глобального класса в связи со сложностями с их применением в языке TypeScript. Хранение путей API предлагается реализовать в качестве JSON файла. Дополнительные настройки приложения предлагается реализовать с помощью переменных среды, устанавливаемых в файле docker-compose.yml. Запросы к API предлагается реализовать с помощью пакета Axios.

Схема 2

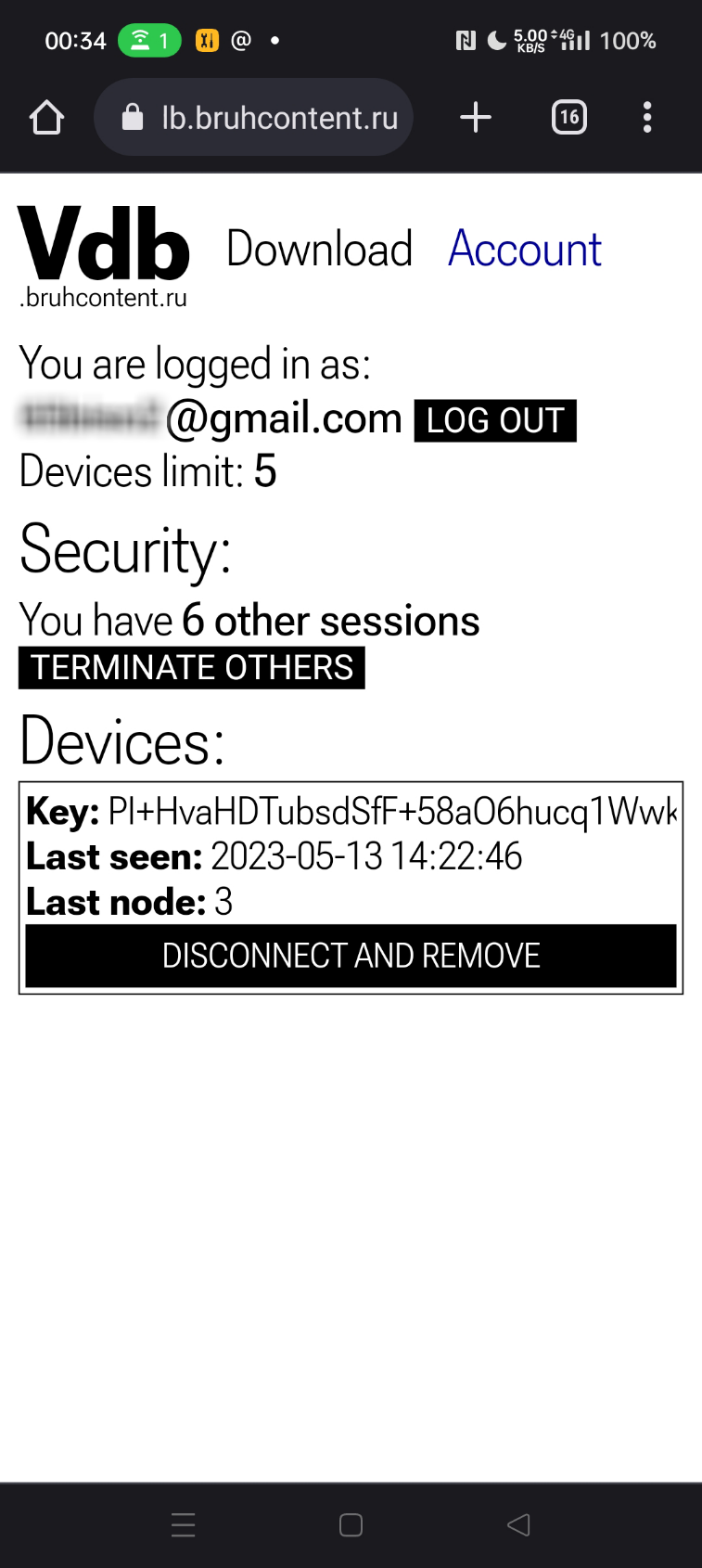
В личном кабинете предполагается реализовать функционал просмотр текущего пользователя, лимита устройств, текущего списка устройств и их удаления. Также реализуется возможность ликвидировать все сессии кроме текущей.

Статус серверов для соответствующей страницы предлагается получать от главного сервера, т.к. сами ноды не принимают запросы с неизвестных IP.

**РЕАЛИЗАЦИЯ САЙТА**

В данной работе у меня отсутствует желание приводить «копи-пасту» из интернета. Разработка приложений на ReactTS описана неоднократно и мой сайт не представляет никакого уникального решения. Отмечу только важное.

**Разработка интерфейса**

Реализован выбор режима работы – debug/release через переменные среды. Строковые константы вынесены в JSON-файлы. Применен современный подход в шрифтах различной жирности – т.н. flex, обеспечивающий передачу одного файла шрифта с переменной жирностью. Применены CSS-модули. Максимальное количество валидаций выполнено на клиенте, до передачи на сервер, однако сервер их также дублирует. На изображении 2 представлен интерфейс личного кабинета на смартфонах. В целях создания общего представления о процессе разработки, в приложении 1В представлен программный код компонента, отвечающего за карточку устройства, такую, какая отрисована под заголовком «Devices» на изображении 2.

Изображение 2

**Конфигурация NGINX и Docker. Точка входа**

Поскольку React-приложение в собранном виде представляет всё тот же набор обычных HTML/CSS/JS файлов, то нам достаточно раздавать их с помощью какой-нибудь сервера. Самым легковесным здесь, как и почти всегда, является NGINX с применением директивы sendfile – программный код 9. В связи с вышесказанным реализация многоэтапной сборки, принцип которой уже был показан ранее, становится особенно важной, т.к. позволяет избавиться от любого вида SDK/Runtime’ов в конечном образе, что в итоге даёт нам образ размером 6 МБ в сжатом виде. Точкой входа же в нем является не более, чем запуск NGINX из dockerfile – командой ‘CMD ["nginx", "-g", "daemon off;"]’.

worker\_processes 1;

events {

worker\_connections 8192;

}

error\_log /var/log/nginx/error.log error;

http {

access\_log off;

sendfile on;

tcp\_nopush on;

tcp\_nodelay on;

include /etc/nginx/mime.types;

gzip on;

gzip\_vary on;

gzip\_min\_length 1024; # bytes

gzip\_comp\_level 1;

gzip\_http\_version 1.0;

gzip\_types text/css application/javascript image/jpeg;

server {

listen 3000;

root /usr/share/nginx/html;

index index.html;

location / {

try\_files $uri $uri/ /index.html;

}

}

}

Программный код 9

**РЕАЛИЗАЦИЯ ДЕСКТОПНОГО ПРИЛОЖЕНИЯ**

Десктопное приложение не представляет приоритетный объект разработки и для него не выполнялось специального проектирования. У приложения 2 основных состояния с точки зрения интерфейса – отображение окна входа и отображение окна выбора сервера. Приложение является одностраничным, но панели входа и выбора сервера попеременно отображаются или скрываются.

На верхнем уровне управление находится у класса ‘UserInterfaceManager’, реализующего паттерн «компоновщик» относительно прочих. Спускаясь ниже, в начальный момент времени управление находится у класса ‘VdbClient’. После аутентификации одним из двух способов – загрузкой сохраненного на диске токена (метод ‘TryLoadUser’) либо вводом логина и пароля с дальнейшей записью refresh-токена на диск (метод ‘Authenticate’), в свойство класса устанавливается access-токен. Управление передается классу ‘TunnelManager’, создающего пару ключей, после чего возвращается обратно. Метод ‘RegisterDevice’ с использованием сгенерированного публичного ключа регистрирует девайс на сервере. Сервер может возвращать различные коды ответа. В случае если сервер ответил кодом HTTP\_303\_SEE\_OTHER – «сыграла» вероятность (N-K) к 2256 генерации дубликата ключа для разных пользователей, где N – число ключей в системе, K – число ключей данного пользователя. В этом случае генерация выполняется заново. Ответ кодом HTTP\_302\_FOUND отвечает дубликат ключа для текущего пользователя, что, на данный момент, просто игнорируется, вероятность такого дублирования либо крайне незначительна и ведет лишь к необходимость выполнить вход заново на одном из устройств, либо связана с локальной ошибкой исполнения и не ведет ни к каким проблемам вовсе. Ответ кодом HTTP\_409\_CONFLICT означает достижение лимита устройств пользователем – в это случае выводится ошибка.

Далее, после запроса доступного списка нод методом ‘GetNodesList’, осуществляется запрос на добавление ключа на выбранную ноду методом ‘ConnectToNode’. В случае успеха можно утверждать, что имеющихся данных достаточно для установления туннеля. Управление передается классу ‘TunnelManager’, он записывает конфигурацию в файл, после чего передаёт его Wireguard. На изображении 3 представлен интерфейс программы и результат «проверки» работоспособности – IP-адрес в Амстердаме при использовании программы в Москве, а выделенный в приватной WireGuard-сети адрес представляет собой ‘10.6.0.0’, что соответствует ожидаемому поведению, описанному в главах о проектировании и реализации VPN-сервера.

 В приложении 1Г находится код сборки установщика на InnoSetup. Из особенностей можно отметить только гарантию не удаления пользовательских файлов, если он по какой-то причине решил поместить их в директорию с программой.

Изображение 3

**ЗАКЛЮЧЕНИЕ**

В данной работе был кратко описан процесс создания сервиса туннелирования трафика с применением большого стека технологий. Была получена гибкая архитектура, позволяющая добавить VPN-сервис из любой локации к системе путем редактирования единственного JSON-файла главного сервера.

Ресурсы любого проекта ограничены и в данном случае это привело к двум негативным последствиям. Во-первых, не удалось в заданные сроки изучить electronJS и реализовать на нем десктопный клиент. Во-вторых, сам текст данной работы написана не более чем за 3 дня, что может вести не к самому точному описанию технических решений.

В настоящий момент решение уже развернуто, а сайт доступен в сети «интернет» по адресу ‘vpn.bruhcontent.ru’. В свою очередь репозиторий с документацией и ссылками на исходный код, задействованный в настощем проекте расположен в сети «интернет» по адресу ‘github.com/LuminoDiode/vdb\_documentation’ – изображение 4.



Изображение 4

**ИСТОЧНИКИ**

1. Текст лицензии Mozilla Public License [Электронный ресурс] -   
   mozilla.org/en-US/MPL/2.0/ (дата обращения: 01.06.2023)
2. Текст лицензии GNU General Public License [Электронный ресурс] -   
   www.gnu.org/licenses/gpl-3.0.en.html (дата обращения: 01.06.2023)
3. Репозиторий siteblog [Электронный ресурс] - [github.com/LuminoDiode/siteblog/blob/main/backend/Services/SettingsProviderService.cs](https://github.com/LuminoDiode/siteblog/blob/main/backend/Services/SettingsProviderService.cs) (дата обращения: 01.06.2023)
4. Международный стандарт RFC 1918 [Электронный ресурс] -   
   rfc-editor.org/rfc/rfc1918 (дата обращения: 01.06.2023)
5. Международный стандарт RFC 9110 [Электронный ресурс] -   
   rfc-editor.org/rfc/rfc9110 (дата обращения: 01.06.2023)
6. Документация Docker [Электронный ресурс] -   
   [docs.docker.com/config/containers/multi-service\_container/](https://docs.docker.com/config/containers/multi-service_container/)   
   (дата обращения: 01.06.2023)
7. Сайт от программировании [Электронный ресурс] -   
   [stackoverflow.com/questions/50845198/periodically-running-docker-command](https://stackoverflow.com/questions/50845198/periodically-running-docker-command)   
   (дата обращения: 01.06.2023)
8. Сайт от программировании [Электронный ресурс] -   
   [serverfault.com/questions/1101002/wireguard-client-addition-without-restart](https://serverfault.com/questions/1101002/wireguard-client-addition-without-restart)   
   (дата обращения: 01.06.2023)
9. Рекомендации по разработке от SSL Labs [Электронный ресурс] -   
   github.com/ssllabs/research/wiki/SSL-and-TLS-Deployment-Best-Practices   
   (дата обращения: 01.06.2023)
10. Исследование производительности алгоритмов шифрования   
    [Электронный ресурс] - calomel.org/aesni\_ssl\_performance.html   
    (дата обращения: 01.06.2023)
11. Правила трансляции типов в EntityFramework [Электронный ресурс] - https://www.npgsql.org/doc/types/basic.html  
    (дата обращения: 01.06.2023)

**ПРИЛОЖЕНИЕ 1А  
Неполный код скрипта создания начальной структуры проекта (PowerShell)**

dotnet new globaljson;

dotnet new classlib -o vdb\_node\_wireguard\_manipulator;

dotnet new webapi -o vdb\_node\_api;

dotnet new sln --name vdb\_node;

dotnet sln add vdb\_node\_wireguard\_manipulator/vdb\_node\_wireguard\_manipulator.csproj;

dotnet sln add vdb\_node\_api/vdb\_node\_api.csproj;

New-Item dockerfile;

dotnet new gitignore;  
git init;

**ПРИЛОЖЕНИЕ 2А  
Неполный код класса EnvironmentProvider (C#)**

private const string ENV\_ALLOW\_NOAUTH = "REST2WG\_ALLOW\_NOAUTH"; // названия ПС

public bool? ALLOW\_NOAUTH { get; init; } = null; // извлеченные значения ПС

private readonly ILogger<EnvironmentProvider> \_logger; // логгер

public EnvironmentProvider(ILogger<EnvironmentProvider> logger) { // конструктор

\_logger = logger;

ALLOW\_NOAUTH = ParseBoolValue(ENV\_ALLOW\_NOAUTH);

}

private bool? ParseBoolValue(string EnvName) // функция извлечения и парсинга ПС

{

string? str = Environment.GetEnvironmentVariable(EnvName);

if (str is not null) {

if (str.Equals("true", StringComparison.InvariantCultureIgnoreCase))

\_logger.LogInformation($"{EnvName}={true}."); return true;

if (str.Equals("false", StringComparison.InvariantCultureIgnoreCase))

\_logger.LogInformation($"{EnvName}={false}."); return false;

\_logger.LogWarning(GetIncorrectIgnoredMessage(EnvName)); }

\_logger.LogInformation($"{EnvName} was not present.");

return null;

}

**ПРИЛОЖЕНИЕ 3А  
Неполный код класса IpDedicationService (C#)**

private const byte FirstIpByteStart = 10;

private const byte SecondIpByteStart = 6;

private const byte NetworkMask = 32;

private Dictionary<string, int> \_dedicatedAddresses; // публичный ключ к адресу

private readonly HashSet<int> \_usedAddresses; // коллекция всех занятых адресов

public string EnsureDedicatedAddressForPeer(string pubKey) {

if (\_dedicatedAddresses.TryGetValue(pubKey, out int address)) return IndexToString(address);

else {

int addr = \_dedicatedAddresses.Count; // сначала попытаемся угадать свободный адрес

if (\_usedAddresses.Contains(addr)) {

for (addr = 0; addr < MaxClients; addr++) {

if (!\_usedAddresses.Contains(addr)) {

AddPeer(pubKey, addr); return IndexToString(addr); } } }

AddPeer(pubKey, addr); return IndexToString(addr); }

}

private bool AddPeer(string pubKey, int addressIndex) {

\_dedicatedAddresses.Add(pubKey, addressIndex);return \_usedAddresses.Add(addressIndex);

}

public bool DeletePeer(string pubKey){

if (!\_dedicatedAddresses.TryGetValue(pubKey, out int address)) { return false; }

else { usedAddresses.Remove(address); return \_dedicatedAddresses.Remove(pubKey); }

}

public void SyncState(Dictionary<string, int> keyToIpActulState) {

\_dedicatedAddresses = keyToIpActulState; \_usedAddresses.Clear();

foreach (int ip in \_dedicatedAddresses.Values) \_usedAddresses.Add(ip); }

**ПРИЛОЖЕНИЕ 4А  
Неполный код классов MasterAccountsService и AuthorizationMiddleware (C#)**

public sealed class MasterAccountsService {

private readonly List<byte[]> \_mastersKeyHashes;

public MasterAccountsService(SettingsProviderService settingsProvider) {

\_mastersKeyHashes = settingsProvider.MasterAccounts

.Select(x => Convert.FromBase64String(x.KeyHashBase64)).ToList();

}

public bool IsValid(string keyBase64) {

var search = SHA512.HashData(Convert.FromBase64String(keyBase64));

return \_mastersKeyHashes.Any(k => k.SequenceEqual(search));

}}  
public sealed class AuthorizationMiddleware : IMiddleware  
{  
private readonly MasterAccountsService \_accountsService;  
private readonly ILogger \_logger;

public AuthorizationMiddleware(MasterAccountsService accountsService, ILogger lgr) {  
 \_accountsService = accountsService; \_logger = lgr;  
}

public Task InvokeAsync(HttpContext context, RequestDelegate next)

{

var header = context.Request.Headers.Authorization; string? key;

try { key = header.Single(); } // Acc. to RFC, Authorize may appear only once

catch (InvalidOperationException) {

\_logger.LogWarning(GetRejectionMessage(context,

"Authorization header seen more than once."));

return Task.FromResult(new StatusCodeResult(Status400BadRequest)); }

if (string.IsNullOrEmpty(key)) {

\_logger.LogWarning(GetRejectionMessage(context,

"Authorization key was null or empty."));

return Task.FromResult(new StatusCodeResult(Status400BadRequest)); }

try {

if (!\_accountsService.IsValid(key)) {

\_logger.LogWarning(GetRejectionMessage(context,

"Authorization key was not found on the server."));

return Task.FromResult(new StatusCodeResult(401));

}} catch{

\_logger.LogWarning(GetRejectionMessage(context,

"Authorization key format was not valid."));

return Task.FromResult(new StatusCodeResult(Status400BadRequest)); }

return next(context); /\* key was successfully validated \*/

}}

**ПРИЛОЖЕНИЕ 5А  
Неполный код класса WgCommandsExecutor (C#)**

public static class WgCommandsExecutor

{

public static string? LastSeenInterfacePubkey

=> WgStatusStreamParser.LastSeenInterfacePubkey;

private static async Task<string> RunCommand  
 (string command, string fileName = @"wg") {

return await (await RunCommandStream(command, fileName)).ReadToEndAsync();

}

private static async Task<StreamReader> RunCommandStream  
 (string command, string fileName = @"wg") {

var psi = new ProcessStartInfo();

psi.FileName = fileName;

psi.Arguments = command;

psi.RedirectStandardOutput = true;

psi.UseShellExecute = false;

psi.CreateNoWindow = true;

var process = Process.Start(psi);

if (process is null) {

throw new AggregateException("Unable to perform the command");

}

await process.WaitForExitAsync();

return process.StandardOutput;

}

private static string GetAddPeerCommand(string pubKey, string allowedIps) {

return $"set wg0 peer \"{pubKey}\" allowed-ips {allowedIps}";

}

private static string GetRemovePeerCommand(string pubKey) {

return $"set wg0 peer \"{pubKey}\" remove";

}

private static string GetWgShowCommand(string wgInterfaceName = null!) {

return wgInterfaceName is null ? "show" : $"show {wgInterfaceName}";

}

public static async Task<string> AddPeer(string pubKey, string allowedIps) {

return await RunCommand(GetAddPeerCommand(pubKey, allowedIps));

}

public static async Task<string> RemovePeer(string pubKey) {

return await RunCommand(GetRemovePeerCommand(pubKey));

}

public static async Task<IEnumerator<WgShortPeerInfo>> GetPeersListEnumerator()

{

return WgStatusStreamParser.ParsePeersFromStreamShortly(

await RunCommandStream(GetWgShowCommand()));

} }

**ПРИЛОЖЕНИЕ 1Б  
Неполный код метода IssueJwtAndAddToUser (C#)**

[NonAction]

public JwtResponse IssueJwtAndAddToUser  
 (User user, bool provideRefresh = true, bool refreshJwtInBody = false)

{

// create access token using passed user model

var responseObj = new JwtResponse(  
 this.\_jwtService.GenerateAccessJwtToken(new UserInfo(user)));

if(!provideRefresh) return responseObj;

// is there too much tokens already? then just trim a half

if(user.RefreshTokensEntropies.Count > byte.MaxValue / 8) {

user.RefreshTokensEntropies = user.RefreshTokensEntropies

.Skip(user.RefreshTokensEntropies.Count / 2)

.ToList();

}

// create new refresh token for inserting into DB

var issuedTokenRecord = new RefreshToken {

IssuedToUser = user.Id,

Entropy = Random.Shared.NextInt64(long.MinValue, long.MaxValue)

};

// write refresh token

var refreshToken = this.\_jwtService.GenerateRefreshJwtToken(issuedTokenRecord);

// decide where to place refresh token

if(refreshJwtInBody) {

responseObj.RefreshToken = refreshToken;

} else {

// if static is not inited yet

if(\_jwtCookieOptions is null) {

\_jwtCookieOptions = new CookieOptions() {

Secure = true,

SameSite = SameSiteMode.Strict,

HttpOnly = true,

IsEssential = true,

Path = "/api/auth",

MaxAge = this.\_jwtService.RefreshTokenLifespan

};

}

this.Response.Cookies.Append(  
 JwtRefreshTokenCookieName, refreshToken, \_jwtCookieOptions);  
 }

user.RefreshTokensEntropies.Add(issuedTokenRecord.Entropy);

return responseObj;

}

**ПРИЛОЖЕНИЕ 1В  
 Код компонента DeviceCard (TypeSript, React)**

import { useState } from "react";

import cl from "./DeviceCard.module.css";

import UserApiHelper from "src/helpers/UserApiHelper";

import IUserDevice from '../../models/Device/IUserDevice';

const DeviceCard: React.FC<IUserDevice> = (props) => {

const [isButtonDisabled, setButtonDisabled] = useState(false);

const [errorMsg, setErrorMsg] = useState("");

  const [commonMsg, setCommonMsg] = useState("");

  const id = props.id;

  const pk = props.wireguardPublicKey;

  const lastNode = props.lastConnectedNodeId;

  const lastSeen = props.lastSeenUtc;

  const onDelete = async () => {

    setButtonDisabled(true);

    if (**await new UserApiHelper().removeUserDevice(pk)**) {

setCommonMsg("Removed successfully."); return;

} else {

setErrorMsg("Unable to remove device.");

  }

setTimeout(() => setButtonDisabled(false), 5000);  
 }

return (

<span className={[cl.cardWrapper].join(' ')} key={id}>

    <span className={cl.col}>

      <span className={cl.pkSpan}>

        <strong>Key:</strong>&nbsp;{pk ?? "unknown"}

        </span>

      <span>

      <strong>Last seen:</strong>&nbsp;  
 { lastSeen?.split('.')[0].split('T').join(' ') ?? "never" }

      </span>

    <span>

      <strong>Last node:</strong>&nbsp;{lastNode ?? "none"}

     </span>

     </span>

<span className={cl.removeWrapper}>

      <button className={cl.removeButton}   
 onClick={onDelete} disabled={isButtonDisabled}>

         DISCONNECT AND REMOVE

       </button>

        {errorMsg ? <span className={cl.errorMsg}>{errorMsg}</span> : <span />}

         {commonMsg ? <span className={cl.commonMsg}>{commonMsg}</span> : <span />}

    </span>

</span>);

}

export default DeviceCard;

**ПРИЛОЖЕНИЕ 1Г  
Неполный код InnoSetup-скрипта создания установщика десктопного приложения (Pascal-подобный язык)**

#define MyAppName "Vdb VPN"

#define MyAppExeName "UserInterface.exe"

#define DotNetFileName "windowsdesktop-runtime-8.0.0-preview.3.23178.1-win-x64.exe";

#define WgFileName "wireguard-amd64-0.5.3.msi";

[Setup]

AppId={{DA1F8E9D-521C-43BC-B6BD-9896B1D68951}

AppName={#MyAppName}

AppVersion={#MyAppVersion}

AppPublisher={#MyAppPublisher}

AppPublisherURL={#MyAppURL}

DefaultDirName={autopf}\{#MyAppName}

DisableProgramGroupPage=yes

LicenseFile=LICENSE.txt

OutputBaseFilename=VdbInstaller

Compression=none

WizardStyle=modern

[Files]

Source: "..\UserInterface\bin\Release\net8.0-windows\publish\win-   
 x64\{#MyAppExeName}"; DestDir: "{app}"

Source: ".\additional\_software\{#DotNetFileName}";   
 DestDir: "{tmp}"; Flags: deleteafterinstall; AfterInstall: InstallDotnet();

Source: ".\additional\_software\{#WgFileName}";   
 DestDir: "{tmp}"; Flags: deleteafterinstall; AfterInstall: InstallWg();

[Code]

procedure InstallDotnet;

var

resultCode: integer;

begin

WizardForm.StatusLabel.Caption := 'Installing dotNET...';

WizardForm.ProgressGauge.Style := npbstMarquee;

Exec(ExpandConstant('{tmp}\{#DotNetFileName}'), '/quiet /norestart', '', SW\_HIDE, ewWaitUntilTerminated, resultCode);

end;

procedure InstallWg;

var

resultCode: integer;

begin

WizardForm.StatusLabel.Caption := 'Installing WireGuard...';

WizardForm.ProgressGauge.Style := npbstMarquee;

Exec('cmd.exe', ExpandConstant('/q /c MsiExec.exe /i {tmp}\{#WgFileName} DO\_NOT\_LAUNCH=1 /qn'), '', SW\_HIDE, ewWaitUntilTerminated, resultCode);

end;

[Run]

Filename: "{app}\{#MyAppExeName}"; Description: "{cm:LaunchProgram,{#StringChange(  
MyAppName, '&', '&&')}}"; Flags: runascurrentuser nowait postinstall skipifsilent

[UninstallDelete]

Type: filesandordirs; Name: "{userdocs}\Vdb\refresh.token"

Type: filesandordirs; Name: "{userdocs}\Vdb\vdb0.conf"

Type: filesandordirs; Name: "{userdocs}\Vdb\vdb0.key"

Type: dirifempty; Name: "{userdocs}\Vdb";