Учебный проект "Daibot.fun". Часть 1.
Автор: Гудаев Олег Александрович (с), 2025 г.

Приведенные linux-команды от лица администратора (#) иллюстрированы экранными формами для ознакомления и понимания

Задание проекта: Создать мини-приложение Телеграм бота генерации визуального улучшения плакатов спецэффектами SVG-фильтов, на базе виртуального выделенного сервера с доменным именем daibot.fun.

Базовый алгоритм ИИ: написан на языке программирования JavaScript.

Базовые параметры конфигурации.
Телеграмм-бот требует вычислительного сервера для своего размещения.
1. Мини приложения бота общается с Телеграмм по программному интерфейсу REST API по протоколу https. Буква s в протоколе – это защита данных при обмене. Веб-приложения WebApps для встраивания в Telegram поддерживает только защищенные протоколом HTTPS для обмена данными [1].
2. Для размещения и запуска бота используем аренду виртуального VPS. Virtual Private Server – это услуга аренды виртуального сервера. VPS эмулирует работу реального физического сервера: есть root-доступ администратора к операционной системе, возможна установка своих операционных систем Linux и дополнительного программного обеспечения.
3. Необходимо установить на VPS дополнительный пакет ПО: веб-сервер Nginx и программный сервер Node JS. Провайдер VPS предоставляет образ ПО. Сервер Nginx может работать как веб-сервер, так и прокси-сервер.
4. Для работы веб-приложения по протоколу https нужно обеспечить домен сертификатом подлинности. Его можно сгенерировать самостоятельно с помощью самописного сертификата [2] или зарегистрировать.
5. Надо зарегистрировать домен daibot.fun. Акроним "D" плюс искусственный интеллект "AI" плюс сервлет "Bot". Доменная зона .fun означает развлечение. Доменная зона .fun — это тематическая зона, которая указывает на развлекательный характер бизнеса и самого сайта. В переводе с английского fun — «веселье» [3].

Ход работ:

Регистрация домена:
1. Выберем провайдера reg.ru.
2. Домен daibot.fun свободен и стоит 199 руб в год, на 11 октября 2024 года.
3. Сделаем учетную запись на reg.ru для почтового ящика good_*@mail.ru. Из письма в ящике подтвердите учетную запись. Перейдите в настройки reg.ru и укажите свои контакты (ФИО), паспорт, адрес регистрации, телефон. Пароль придется сбросить для ввода нового. Заполняйте аккуратно – это будет указано в регистрации домена.

4. Пополним баланс лицевого счёта.

На счёт занести 233 руб. для покупки домена в зоне .fun, используя смартфон. Способ оплаты SberPay и авторизацию сделки через SberID.

5. Купить домен и обязательно подтвердите указанный e-mail, на кого зарегистрирован домен. Надо подождать 24 часа пока все остальные DNS получать новые сведения.

Результат регистрации – пока нет ресурсов сайта и домен припаркован:

Покупка VPS:

1. Выберем провайдера ruvds.ru. Тип сервера VDS аналог VPS.
2. Регистрация для почтового ящика good_*@mail.ru.
3. Выбор конфигурации 240 руб. за месяц, ПО NodeJS - Ubuntu 20.04 LTS:

4. Оплата на прямую через карту, пока не привязывайте:

Результат получили выделенный IP-адрес 45.8.230.131:

Настройка VPS.

1. Безопасность сервера.

Создание пользователей.

root – суперпользователь системы.

sudo – консольная команда выполняющая команду переданную ей как аргумент с правами суперпользователя (**root**).

– символ подсказки в консоли, который явно указывает, что команда будет выполнена под учетной записью root. В отличии от символа **$**, который говорит что команда будет выполнена от имени обычного пользователя.

Все примеры Linux-команд учебного проекта запускаются от лица администратор **root**. Для запуска команды студентом-пользователем от лица администратора перед командой написать **sudo**:

Запускает администратор root:

apt update

Эту же команду запускает пользователь good:

$ sudo apt update

Список пользователей, для подключения к серверу:

№	Login	Пароль	User ID	Group ID	User Info	Home Directory	Default Shell
1.	user01		1701	1701	free account 1	/home/user01	/bin/bash
2.	user02		1702	1702	free account 2	/home/user02	/bin/bash
3.	user03		1703	1703	free account 3	/home/user03	/bin/bash
4.	user04		1704	1704	free account 4	/home/user04	/bin/bash
5.	user05		1705	1705	free account 5	/home/user05	/bin/bash
6.	user06		1706	1706	free account 6	/home/user06	/bin/bash
7.	user07		1707	1707	free account 7	/home/user07	/bin/bash
8.	user08		1708	1708	free account 8	/home/user08	/bin/bash
9.	user09		1709	1709	free account 9	/home/user09	/bin/bash
10.	user10		1710	1710	free account 10	/home/user10	/bin/bash
11.	user11		1711	1711	free account 11	/home/user11	/bin/bash
12.	user12		1712	1712	free account 12	/home/user12	/bin/bash
13.	user21		1801	1801	free account 1*	/home/user21	/bin/bash

№	Login	Пароль	User ID	Group ID	User Info	Home Directory	Default Shell
14.	user22		1802	1802	free account 2*	/home/user22	/bin/bash
15.	user23		1803	1803	free account 3*	/home/user23	/bin/bash
16.	user24		1804	1804	free account 4*	/home/user24	/bin/bash
17.	user25		1805	1805	free account 5*	/home/user25	/bin/bash
18.	user26		1806	1806	free account 6*	/home/user26	/bin/bash
19.	user27		1807	1807	Test User	/home/user27	/bin/bash
20.	user28		1808	1808	free account 7*	/home/user28	/bin/bash
21.	user29		1809	1809	free account 8*	/home/user29	/bin/bash
22.	user30		1810	1810	free account 9*	/home/user30	/bin/bash
23.	user31		1811	1811	free account 10*	/home/user31	/bin/bash
24.	user32		1812	1812	free account 11*	/home/user32	/bin/bash
25.	user33		1813	1813	free account 12*	/home/user33	/bin/bash
26.	user34		1814	1814	free account 13*	/home/user34	/bin/bash
27.	user35		1815	1815	free account 14*	/home/user35	/bin/bash
28.	user36		1816	1816	free account 15*	/home/user36	/bin/bash
29.	user37		1817	1817	free account 16*	/home/user37	/bin/bash
30.	user38		1818	1818	free account 17*	/home/user38	/bin/bash
31.	good		1888	1888	Gudaev O.A. ASU-92	/home/good	/bin/bash

Простой способ автоматического массового создания пользователей в Linux. Поместите в файл users-list.txt по одной информации о пользователе в каждой строке, следуя приведенному ниже шаблону:

```
Username:Password:User ID:Group ID:User Info:Home Directory:Default Shell
Username:Password:User ID:Group ID:User Info:Home Directory:Default Shell
Username:Password:User ID:Group ID:User Info:Home Directory:Default Shell
```

После создания файла с предопределенной информацией о пользователях используйте команду newusers для создания пользователей.
touch users-list.txt
newusers users-list.txt
Дать права администратора good:
usermod -aG sudo good
Перейти в роль пользователя good, увидеть папки новых пользователей и удалить навсегда файл с паролями в root:
# su - good
$ sudo ls -la /root
$ sudo shred -zvu -n 10 /root/user-list.txt
В команде shred ниже параметры:
-z – добавляет окончательную перезапись нулями, чтобы скрыть уничтожение.
-v – включает отображение хода выполнения операции.
-u – обрезает и удаляет файл после перезаписи.
-n – указывает, сколько раз перезаписывать содержимое файла (по умолчанию — 3).

2. Подключение к серверу VPS удаленно через терминал под Windows по протоколу SSH. Подробно описано в статье [4]. Скачайте SSH-клиент PuTTY [5]. SSH (Secure Shell — «безопасная оболочка») — это сетевой протокол для удалённого управления операционной системой с помощью командной строки и передачи данных в зашифрованном виде.
PuTTY — это свободно распространяемая программа с открытым исходным кодом для управления удалённым узлом (например, сервером). Управление происходит через командную строку: пользователь вводит команду на одном устройстве, а выполняется она на другом. Изначально PuTTY разрабатывалась для Microsoft Windows.

В терминале PuTTY используется клик правой кнопки мыши для копирования / вставки выделенного текста в буфер обмена Window. Для ввода пароля кликнули правой кнопкой мышки в нужном месте терминала (лаймовый цвет курсора) и нажали Enter. Для пароля в терминале PuTTY не отображается скопированный текст!

3. Команды Linux [6]

Соединяюсь по ssh с сервером для ввода команд, после непродолжительной паузы в работе, происходит дисконнект из-за бездействия, как увеличить время соединения? PuTTy options --> Connection "Enable TCP_keepalives (SO_KEEPALIVE option) ну и время keepalive выставите, секунд 20-30.

Команда	**Пример**	**Описание**
Ctrl+C		Прервать выполнение команды
Ctrl+L	printf "\033c"	Очистка терминала в эмуляторе
exit		Если вошли в диалог с какой-нибудь командой, когда будет приглашение (промт) " > ", то можно выйти.
pwd		Как понять, где находишься. Расшифровывается как Print Working Directory.
cd	cd /var/www cd .. cd /home	Смена директории Перейти в родительскую папку Здесь находится nodejs

Команда	Пример	Описание
	cd /var cd l	Папка где хранятся проекты. Папка www для сайта. Если нажать клавишу Tab, то система выведет подсказку – все папки начинающиеся на l lib/ local/ lock/ log/
ls	 ls -l ls a*	Список файлов и папок. Папки синие. Список файлов с атрибутами в виде таблицы Вывести все что начинается на букву "a"
mkdir	mkdir app02	Создать папку
touch	touch server.js	Создать пустой файл
nano vi	nano server.js vi server.js	Простой редактор. Выйти из него Ctrl+X, а затем N Сложный редактор файлов. Команда :wq сохранить и выйти, а команда :w просто сохранить; :q! – выйти
cat less	cat file.js	Посмотреть содержимое файла Посмотреть большой файл, листая стрелками по строкам. Выйти q
cp	cp src dist cp /dev/null dist	Копировать файл src – откуда, а dist – куда Сделать пустым файл dist. То есть затереть.
mv		Переместить файл
rm	rm test.txt rm * rm -r app02 rm -rf app02 rm -rfv app02	Удалить файл Удалит все файлы в текущей директории Удалить папку, надо добавить флаг -r (recursive) Удалить всё в папке без вопросов (force), если файл для чтения Показывать имена удаляемых файлов и поддиректорий
find	find / -name "*.deb"	Поиск файла по расширению
apt	apt install mc	Установить программу из репозитария
service	service test start service test stop	Запустить сервис под названием «test» Остановить сервис
chkconfig	chkconfig test on	Чтобы сервис test запускался автоматически при рестарте системы. Она добавит службу в автозапуск.
whereis	whereis test	Понять, где установлено приложение
df	df -h	Посмотреть свободное место на диске
hostname -i	ip a ifconfig	Как узнать IP компьютера
uname	cat /etc/*-release uname -a	Как узнать версию OS Версия ядра
id who lslogins	id user01	Проверить существование пользователя и увидеть его идентификатор Список авторизовавшихся в ОС Список всех с описанием User Info

Можно установить файловый менеджер mc (Midnight Commander Ubutnu) командой для sudo [7]:

```
%%http://archive.ubuntu.com/ubuntu/pool/universe/m/mc/mc_4.8.24-2ubuntu1_amd64.deb
# apt install libgpm2
# apt update
# dpkg -i mc_4.8.24-2ubuntu1_amd64.deb
# apt install -f
```

Ключ -f (сокращение от fix broken) в данном случае устанавливает необходимые зависимости и самостоятельно завершает установку программы.
Скрыть панели навигации Ctrl+O, включить назад Ctrl+O.
Если спросит, какой редактор использовать для файлов выбирайте 2: mcedit. Назначить редактор через командную строку. Для этого нужно нажать F2, выбрать знак «@» и ввести «select-editor».

4. Настройка NodeJS.

Проверить установленную версию: node -v
Ответ: v14.16.0
Для разработки сервлета установим Node под компьютер с Windows 7 [8].
Отметить автоматическую установку дополнительных инструментов компиляции C++.

Если не получилось, то нужно вручную добавить инструменты [9]. Установить утилиту node-gyp:

```
# npm install -g node-gyp
```

```
(C:\Daibot) - Far 2.0.1807 x86 Administrator
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                
npm WARN notsup Unsupported engine for cacache@18.0.4: wanted: {"node":"^16.14.0 || >=18.0.0"} (current: {"node":"13.14.0","npm":"6.
14.4"})
npm WARN notsup Not compatible with your version of node/npm: cacache@18.0.4
npm WARN notsup Unsupported engine for socks-proxy-agent@8.0.4: wanted: {"node":">= 14"} (current: {"node":"13.14.0","npm":"6.14.4"}
)
npm WARN notsup Not compatible with your version of node/npm: socks-proxy-agent@8.0.4
npm WARN notsup Unsupported engine for http-proxy-agent@7.0.2: wanted: {"node":">= 14"} (current: {"node":"13.14.0","npm":"6.14.4"})

npm WARN notsup Not compatible with your version of node/npm: http-proxy-agent@7.0.2
npm WARN notsup Unsupported engine for agent-base@7.1.1: wanted: {"node":">= 14"} (current: {"node":"13.14.0","npm":"6.14.4"})
npm WARN notsup Not compatible with your version of node/npm: agent-base@7.1.1
npm WARN notsup Unsupported engine for https-proxy-agent@7.0.5: wanted: {"node":">= 14"} (current: {"node":"13.14.0","npm":"6.14.4"}
)
npm WARN notsup Not compatible with your version of node/npm: https-proxy-agent@7.0.5
npm WARN notsup Unsupported engine for unique-filename@3.0.0: wanted: {"node":"^14.17.0 || ^16.13.0 || >=18.0.0"} (current: {"node":
"13.14.0","npm":"6.14.4"})
npm WARN notsup Not compatible with your version of node/npm: unique-filename@3.0.0
npm WARN notsup Unsupported engine for fs-minipass@3.0.3: wanted: {"node":"^14.17.0 || ^16.13.0 || >=18.0.0"} (current: {"node":"13.
14.0","npm":"6.14.4"})
npm WARN notsup Not compatible with your version of node/npm: fs-minipass@3.0.3
npm WARN notsup Unsupported engine for minipass-collect@2.0.1: wanted: {"node":">=16 || 14 >=14.17"} (current: {"node":"13.14.0","np
m":"6.14.4"})
npm WARN notsup Not compatible with your version of node/npm: minipass-collect@2.0.1
npm WARN notsup Unsupported engine for @npmcli/fs@3.1.1: wanted: {"node":"^14.17.0 || ^16.13.0 || >=18.0.0"} (current: {"node":"13.1
4.0","npm":"6.14.4"})
npm WARN notsup Not compatible with your version of node/npm: @npmcli/fs@3.1.1
npm WARN notsup Unsupported engine for unique-slug@4.0.0: wanted: {"node":"^14.17.0 || ^16.13.0 || >=18.0.0"} (current: {"node":"13.
14.0","npm":"6.14.4"})
npm WARN notsup Not compatible with your version of node/npm: unique-slug@4.0.0
npm WARN notsup Unsupported engine for abbrev@2.0.0: wanted: {"node":"^14.17.0 || ^16.13.0 || >=18.0.0"} (current: {"node":"13.14.0"
,"npm":"6.14.4"})
npm WARN notsup Not compatible with your version of node/npm: abbrev@2.0.0
npm WARN notsup Unsupported engine for isexe@3.1.1: wanted: {"node":">=16"} (current: {"node":"13.14.0","npm":"6.14.4"})
npm WARN notsup Not compatible with your version of node/npm: isexe@3.1.1

+ node-gyp@10.2.0
added 101 packages from 52 contributors in 74.922s

C:\Daibot>
1Left  2Right  3View..  4Edit..  5Print  6MkLink  7Find  8Histry  9Video  10Tree  11ViewHs  12FoldHs
```

Видно что версия Node не подходит для некоторых модулей.
Установите PowerShell 7. Запустите и дайте команду для установки choco:
Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))

```
Administrator: PowerShell 7 (x64)
PS C:\Users\Администратор> Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.ServicePointManager]::SecurityP
rotocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object System.Net.WebClient).DownloadS
tring('https://community.chocolatey.org/install.ps1'))
Forcing web requests to allow TLS v1.2 (Required for requests to Chocolatey.org)
Getting latest version of the Chocolatey package for download.
Not using proxy.
Getting Chocolatey from https://community.chocolatey.org/api/v2/package/chocolatey/2.3.0.
Downloading https://community.chocolatey.org/api/v2/package/chocolatey/2.3.0 to C:\Users\836D~1\AppData\Local\Temp\choco
latey\chocoInstall\chocolatey.zip
Not using proxy.
Extracting C:\Users\836D~1\AppData\Local\Temp\chocolatey\chocoInstall\chocolatey.zip to C:\Users\836D~1\AppData\Local\Te
mp\chocolatey\chocoInstall
Installing Chocolatey on the local machine
Creating ChocolateyInstall as an environment variable (targeting 'Machine')
  Setting ChocolateyInstall to 'C:\ProgramData\chocolatey'
WARNING: It's very likely you will need to close and reopen your shell
  before you can use choco.
Restricting write permissions to Administrators
We are setting up the Chocolatey package repository.
The packages themselves go to 'C:\ProgramData\chocolatey\lib'
  (i.e. C:\ProgramData\chocolatey\lib\yourPackageName).
A shim file for the command line goes to 'C:\ProgramData\chocolatey\bin'
  and points to an executable in 'C:\ProgramData\chocolatey\lib\yourPackageName'.

Creating Chocolatey CLI folders if they do not already exist.

chocolatey.nupkg file not installed in lib.
 Attempting to locate it from bootstrapper.
PATH environment variable does not have C:\ProgramData\chocolatey\bin in it. Adding...
WARNING: Not setting tab completion: Profile file does not exist at 'C:\Users\Администратор\Documents\PowerShell\Microso
ft.PowerShell_profile.ps1'.
Chocolatey CLI (choco.exe) is now ready.
You can call choco from anywhere, command line or powershell by typing choco.
Run choco /? for a list of functions.
You may need to shut down and restart powershell and/or consoles
 first prior to using choco.
Ensuring Chocolatey commands are on the path
Ensuring chocolatey.nupkg is in the lib folder
PS C:\Users\Администратор> choco
Chocolatey v2.3.0
Please run 'choco -?' or 'choco <command> -?' for help menu.
PS C:\Users\Администратор>
```

Наконец установите дополнительные инструменты с помощью choco:
choco install python visualstudio2022-workload-vctools -y
Не получилось – нет поддержки в Win7 шифрования канала связи SSH/TLS 1.2. Будем исправлять, как описано в статье [10].

5. Как узнать статус nginx?

Для проверки статуса сервера NGINX можно использовать команду:

$ systemctl status nginx

В результате выполнения этой команды будет представлена подробная информация о состоянии службы NGINX.

Вот некоторые основные пункты в выводе команды:

Loaded: указывает, была ли служба успешно загружена, и определяет путь к файлу модуля службы.

Active: отображает текущий статус службы. Обычно отображается одно из следующих состояний:

active (running): NGINX функционирует правильно;

inactive (dead): NGINX в данный момент неактивен;

failed: указывает на ошибку, которая остановила запуск службы NGINX.

Учебный проект "Daibot.fun". Часть - 2.

Приведенные linux-команды от лица администратора (#) иллюстрированы экранными формами для ознакомления и понимания.
Команды пользователя ($) можно тестировать на сервере.

Задание проекта: Создать мини-приложение Телеграм бота генерации визуального улучшения плакатов спецэффектами SVG-фильтов, на базе виртуального выделенного сервера с доменным именем daibot.fun.

Базовый алгоритм ИИ: написан на языке программирования JavaScript.

Общие сведения архитектуры системного программного обеспечения VPS.
Виртуальные сервера VPS (Virtual Private Server) и VDS (Virtual Dedicated Server) это одно и то же – это синонимы. Если какой-либо хостинг предлагает вам VPS и VDS как разные услуги – это должно вас насторожить, что-то тут не так. «Dedicated server» на русский язык переводится как «выделенный сервер». Dedicated (с англ. «выделенный») – это предоставление аппаратных компьютерных мощностей (отдельного «физического» серверного оборудования) клиенту.
Применяется в том случае, когда потенциальный потребитель нуждается в отдельном сервере, но приобрести его по ряду причин не способен.
План работ на VPS-сервере.
Необходимые подзадачи по настройке виртуального сервера с нуля:

- анализ производительности VPS, доверяй провайдеру, но проверяй;
- настройка безопасности сети;
- настройка безопасности удаленного управления сервером;
- настройка стандартных сервисов вычислительного сервера (СМО + ядро): веб-сервер nginx и программный сервер NodeJS или все в одном пакете инсталляции и интеграции между собой LAMP.

Пояснение: управление многопользовательским режимом происходит по теории систем массового обслуживания (СМО). Серверные программы (сервлеты) исполняются под управлением программного ядра в многозадачном режиме.
Программное ядро - это синонимы эмулятор процессора (LLVM), машина вывода (Lisp, SQL, NodeJS), виртуальная машина (JAVA, .net).

Ход работ [11].

1. Проверьте IP сервера на блокировку и СПАМ.

Вы должны проверить не только ip, но и домен вашего сайта на нахождение в спам-базах. Не надо спешить что либо настраивать в VPS, первым делом убедитесь что провайдер выдал вам незаблокированный IP. Вы понимаете, что если ваш IP проблемный, то никаких почтовых серверов вы установить не сможете или продвинуть ваш сайт по SEO или получите "Обход системы" рекламных агрегаторах.

IP адрес может попасть в "черные: списки по разным причинам, но в основном через жалобы пользователей, например, на спам рассылку. Если ваш IP засветился в одном из таких списков, то скорее всего в дальнейшем все отправленные вами письма будут отправляться в спам папки и не будут доставлены.

Тонкая проверка:

1. Проверить какой стране принадлежит ваш IP RIPE Database.
   RIPE (Réseaux IP Européens Network Coordination Centre) — это региональный интернет-реестр, отвечающий за управление IP-адресами и связанными с ними ресурсами в Европе, Ближнем Востоке и частях Центральной Азии. Основные функции RIPE включают распределение блоков IP-адресов, как IPv4, так и IPv6, а также управление номерами автономных систем, используемыми для идентификации каждой автономной системы в интернете.
2. Роскомнадзор – универсальный сервис проверки ограничения доступа к сайтам и (или) страницам сайтов сети «Интернет».
3. Наличие IP в СПАМ базах mxtoolbox проверка в Spam базах
4. Многие провайдеры автоматически блокируют доступ с IP указанных в базе Reputation Checker - Spamhaus, чтобы проверить свой ip на наличие в этих базах используйте сервис. Сюда попадают всегда за что-то. Если ваш IP добавлен в Spamhaus, то письмо на Mail.ru, Yandex, Google, Microsoft, Yahoo не будут доставлены. Социальная сеть LinkedIn не разрешит переходы по ссылке на ваш домен. Исключить свой адрес из Spamhaus можно, для этого нужно написать в их техническую поддержку и подтвердить права на домен и ip.
5. Сервис Proofpoint Proofpoint.использует Apple. Ваш IP попал в данный список? Это обозначает, что письма не будут доставляться на @icloud.com и @me.com.
6. В список URIBL лучше не попадать, из него трудно выбраться, особенно, если это не первое попадание.

На сайте mxtoolbox.com проверил для домена записи в базе DNS-сервера:

Перевод фразы «has no issues» на русский язык – **«нет проблем»**.

2. Измерить скорость загрузки и передачи Интернет-соединения
Не помещает, сразу после покупки VPS замерить скорость интернета, которую вам предоставил провайдер. Скорость можно замерить разными способами, например утилитой iperf, но самым простым способом будет использование приложения Speedtest CLI.

> Iperf – кроссплатформенная консольная клиент-серверная программа – генератор TCP, UDP и SCTP трафика для тестирования пропускной способности сети. Iperf доступен для различных операционных систем, включая Windows, Linux и macOS. Поддержка IPv4 и IPv6. Нужны две части программы сервер и клиент.

Speedtest CLI позволяет использовать глобальную серверную сеть Speedtest с помощью командной строки. Speedtest CLI позволяет измерять такие показатели интернет-соединения, как скорость загрузки и передачи, потеря пакетов и задержка, без использования веб-браузера.
Установка утилита speedtest в Debian/Ubuntu:

```
# curl -s https://install.speedtest.net/app/cli/install.deb.sh | bash
# apt install speedtest-cli
```

Запускаем без всяких параметров, обычно этого достаточно. В более новых версиях лучше использовать дополнительный ключ, auto-binary-bytes который будет показывать скорость в kiB/s, MiB/s, GiB/s.

```
$ speedtest --bytes
```

Все процессы, за исключением at, завершаются, когда вы выходите из системы (консоли). Если вы хотите, чтобы после вашего выхода из системы процесс в фоновом режиме продолжал выполняться, то вам необходимо использовать команду nohup.
Nohup (от "no hang up") - это утилита в Unix-подобных операционных системах, которая позволяет запускать команды или скрипты так, чтобы они продолжали работать, даже если сеанс терминала был завершен.
Команда nohup имеет следующий синтаксис:

```
$ nohup command &
```

Предположим, вы хотите, чтобы команда grep осуществила поиск во всех файлах в вашем текущем каталоге строки "word" и перенаправила вывод в файл word.list, и затем, не ожидая завершения, вы хотите выйти из системы, то введите следующую строку:

```
$ nohup grep word * > word.list &
```

Вы можете завершить команду nohup с помощью команды **kill**, где PID - это идентификатор процесса, который можно найти с помощью команды ps:

```
$ kill PID
```

Как вывести на экран запущенный через nohup процесс:

```
$ tail -f nohup.out
```

К слову, чтобы увидеть ближайшие сервера Speedtest введите команду:

```
$ speedtest --list
```

3. Измерить производительность VPS (bench)
Проведите простые тесты производительности новокупленного VPS, например:

```
# wget -qO- bench.sh | bash
$ curl -Lso- bench.sh | bash
```

На выходе вы получите информацию о конфигурации сервера, I/O, скорости загрузки данных из разных точек мира.

```
root@ruvds-d0ohk: ~
---------------------------------------------------------------------
Node Name        Upload Speed      Download Speed      Latency
Speedtest.net    69.87 Mbps        795.21 Mbps         152.05 ms
Los Angeles, US  98.70 Mbps        135.79 Mbps         178.25 ms
Dallas, US       99.21 Mbps        332.29 Mbps         160.42 ms
Montreal, CA     99.37 Mbps        643.90 Mbps         127.75 ms
Amsterdam, NL    99.05 Mbps        52.94 Mbps          43.99 ms
Chongqing, CN    89.69 Mbps        1.48 Mbps           207.72 ms
Hongkong, CN     98.85 Mbps        277.67 Mbps         183.54 ms
Mumbai, IN       98.81 Mbps        250.09 Mbps         163.17 ms
Singapore, SG    91.87 Mbps        437.95 Mbps         282.67 ms
---------------------------------------------------------------------
Finished in        : 5 min 29 sec
Timestamp          : 2024-10-22 14:07:32 MSK
---------------------------------------------------------------------
root@ruvds-d0ohk:~#
```

Для проверки скорости диска как виртуального, так и физического сервера можно использовать утилиту dd. Утилита dd (dataset definition — определение набора данных) – программа Unix, предназначенная как для копирования, так и для конвертации файлов. Утилита dd позволяет измерить общую скорость чтения или записи, но не предоставляет детализированной информации о IOPS, времени отклика или других важных метриках производительности в отличии от специализированной утилиты, описанной в статье Утилита Linux fio. Использование dd для проверки скорости диска может быть оправдано в ситуациях, когда требуется быстрый и простой тест без необходимости в детальном анализе производительности. Однако для более глубокого и всестороннего тестирования, особенно в профессиональных и коммерческих средах, предпочтительнее использовать fio из-за её расширенных возможностей и точности измерений.

IOPS (аббр. от англ. input/output operations per second «количество операций ввода-вывода в секунду»; произносится как [ай-опс]) - количество операций ввода-вывода, выполняемых системой хранения данных, за одну секунду. Один из параметров, используемых для сравнения систем хранения данных (жёстких дисков (НЖМД), твердотельных накопителей (SSD), сетевых хранилищ SAN, NAS) и оценки их производительности. По сути, IOPS это количество блоков, которое успевает считаться или записаться на носитель. Чем больше размер блока, тем меньше кусков, из которых состоит файл, и тем меньше будет IOPS, так как на чтение куска большего размера будет затрачиваться больше времени. Значит, для определения IOPS надо знать скорость и размер блока при операции чтения / записи. Многие хостинг-провайдеры лгут нам, установив обычный диск SATA с кэшированием или SAS вместо SSD. Можно проверить наличие и скорость SSD, выполнив простой тест с dd. Команда создает файл sb-io-test из нулевых байтов, получаемых из устройства /dev/zero, в который записывает данные, а затем удаляет его. Это покажет вам скорость доступа к диску:

```
dd if=/dev/zero of=sb-io-test bs=1M count=1k conv=fdatasync; rm -rf sb-io-test
```

Ответ dd:

```
1073741824 bytes (1.1 GB, 1.0 GiB) copied, 2.01973 s, 532 MB/s
```

Если вы найдете число ниже 150 МБ/с, убедитесь в технической поддержке, что это не SSD накопитель. Если значение ниже 50 МБ/с — это явно проблема с диском SATA.

Важная информация. Средняя скорость дисковой подсистемы на виртуальных серверах колеблется от 100 Мб/с до 450 Мб/с.

Настройка безопасности сети

1. Обновление операционной системы. Для Ubuntu рекомендую использовать встроенные менеджер пакетов APT, альтернатива Aptitude, dpkg.:

```
# apt update
# apt list --upgradable
# apt upgrade
показать активных пользователей
# who -u
# reboot 0
```

Добавим системные утилиты, которые всегда нужны, такие как: dig, nslookup, host, netstat:

```
# apt install lsb-release htop net-tools nmap
```

htop – UI-монитор показывает запущенные процессы (службы, service);
nmap -sT daibot.fun ## информация о домене

2. Подменяем порты

Боты – «существа» не очень одаренные умом и выполняют грубо прописанные в них инструкции. Сказано, что ломимся на SSH порт, значит, ломимся на 22 порт, сказано, что ломимся на FTP порт, значит ломимся на 21 порт и так далее. Логично будет запутать бота, поменяв номера ключевых портов, а неиспользуемые порты вообще отключить. Главное действовать последовательно и не обмануть самого себя. План действий:

– открываем нужный нам порт через iptables или ufw;

– меняем старый порт на новый;
– старый порт закрываем.
Важно помнить простое правило: все, что не разрешено - запрещено. Иными словами, открыты только порты для используемых сервисов, все остальные должны быть закрыты.
Кстати говоря, некоторые порты у провайдера, будь то бесплатный VPS или нет, закрыты на уровне роутера, поэтому не лишним будет перед сменой портов уточнить у провайдера, может ли быть открыт нужный вам порт.
Сначала нужно проверить, а какие порты вообще используются. Для того, чтобы определить, какие порты слушаются локально, воспользуемся утилитой netstat. Для портов, доступных извне, применим утилиту nmap.

```
# netstat –ntulp
# nmap localhost
```

Теперь зная, какие порты используются, мы можем закрыть ненужные с помощью утилиты iptables или ufw.
Не рекомендуется использовать одновременно два инструмента для настройки firewall во избежание конфликта правил. Если одновременно работают iptables и ufw, то будет применено последнее созданное правило.

```
Не рекомендуется использовать одновременно два инструмента для настройки
firewall во избежание конфликта правил. Если одновременно работают iptabl
es и ufw, то будет применено последнее созданное правило.
```

IPTABLES
Пример синтаксиса iptables для закрытия порта:

```
# iptables -A INPUT -p tcp --dport <номер порта> -j DROP
```

После того, как вы завершите настройку портов, нужно сохранить все созданные правила, чтобы они остались активными даже после перезагрузки. Для этого выполните:

```
# iptables -save
```

Теперь можно посмотреть активные правила:

```
# iptables -L -n -v
```

Возможности iptables очень широки и есть огромная куча мануалов, где они описаны более подробно, здесь мы привели лишь базовый ее функционал.
Внимание! Если вы на сервере используете утилиту ufw вместо iptables, то все приведенные выше команды стоит выполнять через ufw. Одновременное использование обеих утилит может привести к коллизии (конфликту) правил. В случае, если у вас одновременно работает и iptables, и ufw, то будет применено последнее созданное правило.

По примеру портов также можно блокировать целые сервисы. Для этого используется утилита update-rc.d. Утилита update-rc.d – служит для запуска/остановки скриптов, при старте/выключении системы [12]. Она позволяет администраторам контролировать, какие сервисы должны автоматически запускаться или останавливаться на разных уровнях загрузки во время старта или выключения системы.
Команды запуска сервиса iptables:
systemctl start iptables
systemctl enable iptables
В конце перезагрузите систему.
Консольный интерфейс iptables в Linux используется для управления системой пакетной фильтрации ip на уровне ядра. Когда пакет попадает на сервер, он передаётся подсистеме Netfilter для одобрения, изменения или отказа на основе правил, которые она получает от интерфейса пользователя через iptables.

UFW
Консольный интерфейс ufw – инструмент для лёгкой настройки iptables в Ubuntu. Он предоставляет дружественный способ создания сетевой защиты для IPv4 и IPv6. По умолчанию изначально ufw выключен.
Политики безопасности по умолчанию находятся в файле /etc/default/ufw и могут быть изменены с помощью следующей команды:
ufw default deny incoming
ufw default allow outgoing
Некоторые примеры использования ufw:
Для включения нужно набрать в терминале команду:
ufw enable.
Чтобы открыть порт (в данном примере SSH), нужно ввести:
ufw allow 22.
Для закрытия открытого порта:
ufw deny 22.
Для удаления правила нужно использовать команду:
ufw delete deny 22.
Первое правило запрещает все входящие подключения, второе разрешает исходящие:
ufw default deny incoming
ufw default allow outgoing
Утилита Ufw умеет работать с профилями приложений, находящиеся в каталоге /etc/ufw/applications.d, который определяет приложение или службу и соответствующие им настройки безопасности, например открытые или закрытые порты.
Посмотреть созданные профили можно при помощи команды:
ufw app list
Просмотр детальной информации, например о приложении OpenSSH:
ufw app info OpenSSH
Например, настроим правило – разрешим приложению OpenSSH доступ:

```
# ufw allow OpenSSH
```

Проверяем что получилось:
ufw status verbose

```
Status: active
Logging: on (low)
Default: deny (incoming), allow (outgoing), disabled (routed)
New profiles: skip

To                         Action      From
--                         ------      ----
22/tcp (OpenSSH)           ALLOW IN    Anywhere
```

Default: deny (incoming), allow (outgoing), disabled (routed) – По умолчанию: запретить (входящие), разрешить (исходящие), отключено (перенаправленные).

Из листинга мы видим, что все входящие пакеты на серверы запрещены, кроме службы SSH, а все исходящие разращены.
Отключение или сброс UFW. Если вы уже перенастроили правила UFW, но решите начать заново, вы можете использовать эту команду.
ufw reset
Эта команда отключит UFW и удалит все ранее заданные правила, кроме правил по умолчанию. Правила по умолчанию (запрет входящих соединений и разрешение исходящих.) задаются командами:
ufw default deny incoming
ufw default allow outgoing

Настройка безопасности удаленного управления сервером [13**]**

```
Для удаленного управления сервером VPS рекомендуется использовать учетную запись нового пользователя, с правами администратора, и файлы SSH-ключей шифрования, вместо доступа по логину и паролю.
```

Ряд шагов описан в статье 3:
1. Необходимо администратору root создать SSH-ключи и добавить их в личном кабинете провайдера RUVDS.
2. Чтобы не использовать при подключении к Linux-машине учётную запись root создайте нового пользователя.
3. Затем наделите созданного пользователя полномочиями администратора, чтобы вы могли использовать его вместо root-а.
4. Перенести файл публичного SSH-ключа в профиль нового пользователя.
5. Настроить подключение нового пользователя через PuTTY с использованием файлов с SSH-ключями.

Настройка стандартных сервисов вычислительного сервера
Первоначальный план настройки VPS сервера выполнен частично: нет настройки веб-сервера nginx. Настройку веб-сервера органично увязать с получение SSH-сертификата для доступа из Интернета на порт 443.

Первоначальная настройка Node.is. Переключение версий и установка библиотек

0. Удаление nvm под **root**:

```
# nvm_dir="${NVM_DIR:-~/.nvm}"
# nvm unload
# rm -rf "$nvm_dir"
```

Редактируем файл ~/.bashrc Удаляем строчку:

```
# export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh" # This loads nvm
[[ -r $NVM_DIR/bash_completion ]] && \. $NVM_DIR/bash_completion
```

Установка для всех пользователей группы dev, под пользователем **root**:

1. Установите необходимые пакеты:

```
# apt-get install build-essential openssl libssl-dev curl
```

2. Создайте новую группу пользователей и добавьте тех пользователей, которым разрешено управлять установкой Node.js.

Следующая команда создает новую группу с именем dev:

```
groupadd dev
```

Команда adduser. Например, в script.sh файле напишем команды:

```
#!/bin/bash
adduser user01 dev
adduser user02 dev
```

Измените его атрибуты и запустите:

```
# chmod ugo+x scr.sh
# ./script.sh
```

3. Загрузите исходный код nvm с GitHub в /opt/nvm

```
$ git clone https://github.com/creationix/nvm.git /opt/nvm
```

4. Создайте каталог /usr/local/nvm, который будет служить NVM_DIR и хранить все загруженные Node.js:

```
$ mkdir /usr/local/nvm
```

5. Создайте каталог /usr/local/node , который будет использоваться в качестве NPM_CONFIG_PREFIX и в котором будут храниться все глобальные файлы Node Package Manager (npm):

```
$ mkdir /usr/local/node
```

6. Убедитесь, что группа разработчиков может получить доступ к двум вновь созданным папкам.

```
$ chown -R root:dev /usr/local/nvm
$ chmod -R 775 /usr/local/nvm
$ chown -R root:dev /usr/local/node
$ chmod -R 775 /usr/local/node
```

7. Создайте /etc/profile.d/nvm.sh следующим образом. Этот файл будет выполняться каждый раз, когда любой пользователь входит в систему:

```
export NVM_DIR=/usr/local/nvm
source /opt/nvm/nvm.sh
export PATH="/usr/local/node/bin:$PATH"
```

8. Откройте еще один сеанс SSH и войдите как пользователь в группе dev. Проверьте установку nvm следующим образом.

```
$ nvm --version
```

9. Установите версию Node.js по умолчанию, чтобы она активировалась автоматически при входе пользователя в систему.
```
nvm alias default 20
```
10. Теперь можно установить последнюю версию Node.js, например, 20 с помощью команды:

```
# nvm install 20
```

Чтобы начать использовать её, введите команду, например:

```
# nvm use 20
```

Таким образом, можно быстро переключаться между версиями, например, для тестирования.
11. Установка библиотеки:
```
# npm install --save jimp
```
Опция --save в команде npm указывает, что установленный модуль и его версию нужно внести в файл package.json.
```
# npm update jimp
```

Настройка автозапуска скриптов в Linux Ubuntu [14]
Способ 1. Создание скрипта в init.d для запуска во время инициализации системы.
Для начала нужно создать скрипт и скопировать его в директорию /etc/init.d/ удобным для вас способом, а затем сделать его исполняемым командой:
```
# chmod +x /etc/init.d/имя_скрипта
```
Теперь необходимо добавить его в автозагрузку:
```
# update-rc.d имя_скрипта defaults
```
Скрипт запуститься во время инициализации системы.
Удалить из автозагрузки можно так:
```
# update-rc.d -f имя_скрипта remove
```
Загрузка с помощью rc.local после загрузки системы
Способ 2. Необходимо создать скрипт в любой директории, где вам удобно и сделать его исполняемым как в первом способе.
```
# chmod +x /etc/init.d/имя_скрипта
```
Затем подправить файл rc.local любым редактором текста, например nano:
```
# nano /etc/rc.local
```
Изначально скрипт rc.local пустой и содержит только:

```
#!/bin/sh -e

# rc.local

# This script is executed at the end of each multiuser runlevel.

# Make sure that the script will "exit 0" on success or any other

# value on error.

# In order to enable or disable this script just change the execution

# bits.
```

```
# By default this script does nothing.
/home/user/local-script.sh
exit 0
```

Пропишите полный путь скрипта перед строчкой exit 0 и сохраните файл.
Как сказано в комментариях в эталонном rc.local делаем его исполняемым (хотя во многих дистрибутивах он изначально исполняемый):
`# chmod +x /etc/rc.local`
Скрипт выполниться после загрузки системы.
Однако в последних версиях Ubuntu (например в Ubuntu 18.04) в директории /etc нет файла rc.local и его необходимо создать и сделать исполняемым:
```
# touch /etc/rc.local
# chmod +x /etc/rc.local
```
Все, скрипт должен автоматически запускаться.

Защита файла от редактирования

Неизменяемые файлы (запрет на модификацию файла)
В неизменяемые файлы не может внести изменения никакой пользователь, даже root. Он не может изменить их или удалить, пока не удалит неизменяемый бит. Только root может сделать это. Можно сказать, что эта функция защищает вас как root от любых ошибок, которые могут повредить вашу систему.
Вы можете защитить любой файл, который захотите. Используем команду chattr.
`$ chattr +i /myscript`

Учебный проект "Daibot.fun". Часть - 3.

Команды пользователя ($) можно тестировать на сервере. Скрипт бота может не заработать ввиду "сырой" версии, приведен для ознакомления. "Идеальный" корпаративный бот приведен в последних лекциях

Задание проекта: Создать мини-приложение Телеграм бота генерации визуального улучшения плакатов спецэффектами SVG-фильтов, на базе виртуального выделенного сервера с доменным именем daibot.fun.

Базовый алгоритм ИИ: написан на языке программирования JavaScript.

Постановка задачи

Создать для каждого студента свой поддомен. Защитить папку сайта от просмотра других студентов. В папке разработать персональный статический сайт студента. Перенаправить поток данных на порт. Для формирования динамического контента сайта студента на порту разместить node-сервлет.

```
План работ:
- создать поддомены на DNS-сервере;
- изменить права доступа к папкам в директории /home;
- создать папки для хранения сайтов студентов в каталоге /var/www/;
- открыть порты с 3001 по 3038 на VPS-сервере;
- настроить веб-сервер nginx, на примере одного сайта студента user27.
```

Общие сведения о сервлете на сервере VPS

Любой веб-фреймворк сервлета работает не сам по себе. Для запуска написанного на нем приложения нужен веб-сервер. Такой веб-сервер загружает приложение внутрь себя и запускает или передает данные сервлету, запущенному, как сервис ОС Linux. По идее, такой сервлет должен быть написан на том же языке что и веб-сервер, веб-сервер и фреймворк должны знать друг о друге. Но сейчас не обязательно запускать внутри себя. Клей между фреймворками и веб-серверами написан еще в конце девяностых, когда появилась спецификация сервлетов.

Сервлет-бот – это приложение запускаемое под управлением веб-сервера nginx в среде NodeJS для каждого клиента Интернет, подключившегося по протоколу http или https.
Вы уже знакомы с SSH-клиентом терминала PuTTY. Ключевое различие между SSH и SSL заключается в том, что SSH (Secure Shell) используется для создания защищенного туннеля к другому компьютеру, с которого вы можете выдавать команды, передавать данные [15]. С другой стороны, SSL используется для подтверждения аутентичности передаваемых данных между двумя сторонами. SSL (Secure Sockets Layer), который в настоящее время в значительной степени заменен протоколом TLS (Transport Layer Security).

Нативное приложение Node JS не являющееся тг-ботом
Создайте файл сервлета server.js в папке /home/user00/app:

```
$ mkdir -p /home/user27/app
```

Поместите в файл код:

```
##mainapp code
const http = require('http');

const hostname = 'localhost';
const port = 30027;

const server = http.createServer((req, res) => {
        res.statusCode = 200;
        res.setHeader('Content-Type', 'text/plain');
        res.end('This is the Main App!\n');
});

server.listen(port, hostname, () => {
        console.log(`Server running at http://${hostname}:${port}/`);
});

##adminside code
const http = require('http');

const hostname = '45.8.230.131';
const port = 3001;
```

```
const server = http.createServer((req, res) => {
        res.statusCode = 200;
        res.setHeader('Content-Type', 'text/plain');
        res.end('This is the Admin Side!\n');
});

server.listen(port, hostname, () => {
        console.log(`Server running at http://${hostname}:${port}/`);
});
```

Для управления NodeJS-сервлетами, которые работают постоянно в Linux в виде сервисов, используется утилита экосистемы pm2. Опция -g означает установить в системе глобально. Установка последней версии:

```
# npm i -g pm2
```

После установки PM2 Вы всё равно можете запускать свои приложения node для тестирования, используя следующие команды:

```
$ node /home/user27/app/server.js
$ node /home/user27/app/adminside/server.js
```

Обратите внимание, что в производственной среде node.js, когда они должны постоянно работать, Вы должны запускать их с помощью PM2:

```
$ sudo pm2 start /home/user27/app/server.js
$ pm2 start /home/user27/app/adminside/server.js
```

Команда sudo может вам не понадобиться, если ваше приложение хранится в месте, где у обычного пользователя есть права на чтение и запись.
Как посмотреть список запущенных сервлетов:

```
$ pm2 list
```

Видим один запущенный пользовательский http-сервер app, который занимает 26 Мб памяти VPS:

У сервлетов есть идентификатор id. Как мониторить метрику сервлетов:

```
$ pm2 monit
```

Видно папка где расположен сервлет, занимаемая память и процессорное время:

В скобках [0] указан идентификатор сервлета app. Можно посмотреть детальную информацию о процессе работы сервлета с id=0:

```
$ pm2 show 0
```

Видим что сервлет работает 35 часов, так как я перегружал VPS у провайдера после обновления ядра Ubuntu:

Список команд по управлению сервлетами [16]:

```
$ pm2 stop all                  #stop all apps
$ pm2 stop 0                    #stop process with ID 0
$ pm2 restart all               #restart all apps
$ pm2 reset 0                   #reset all counters
$ pm2 delete all                #kill and remove all apps
$ pm2 delete 1                  #kill and delete app with ID 1
```

Видим что для перезагрузки конкретного сервлета надо дать команду: pm2 restart 0. В результате получим на страничке браузере документ, после обновления сайта по ip-адресу сервера VPS:

Для управления логами работы сервлета используются команды:

```
$ pm2 logs            #view logs for all processes
$ pm2 logs 0        #view logs for app 0
$ pm2 logs --json     #view logs for all processes in JSON format
$ pm2 flush           #flush all logs (очистить)
```

Для управления процессом работы самой экосистемы pm2 используются команды:

```
$ pm2 startup            #enable PM2 to start at system boot
$ pm2 startup systemd    #or explicitly specify systemd as startup system
$ pm2 save               #save current process list on reboot
$ pm2 unstartup          #disable PM2 from starting at system boot
$ pm2 update             #update PM2 package
```

Чтобы "жёстко" перезагрузить сам сервер Node.js, можно использовать команду pkill. Например, запустить pkill -HUP node, чтобы остановить все процессы Node.js, а затем запустить его снова.

Получите доступ к приложениям Node, не тг-боту, из веб-браузера

Чтобы получить доступ ко всем вашим приложениям node из удаленного веб-браузера, сначала вам нужно открыть следующие порты (3000 и 3001) на вашем

системном брандмауэре ufw, чтобы разрешить клиентские подключения к приложениям:

```
# ufw allow 3000/tcp
# ufw allow 3001/tcp
# ufw reload
```

Если у Вас нет root-прав администратора, то ничего не получится. Теперь введите в адресной строке браузера:

```
http://45.8.230.131:3001
```

Увидите ответ сервлета `"This is the Main App!"`.

Запрет пользователей просматривать чужие папки, но запускать можно
Изменение прав команда chmod:

Числовой формат:
0 никаких прав;
1 только выполнение;
2 только запись;
3 выполнение и запись;
4 только чтение;
5 чтение и выполнение;
6 чтение и запись;
7 чтение, запись и выполнение.

Создайте скрипт назначение новых прав на папки:
$ **touch** user_view_dir.sh
$ **chmod** ug+x user_view_dir.sh
$ **nano** user_view_dir.sh
#!/bin/bash
chmod 711 /home/user01
chmod 711 /home/user01
....
chmod 711 /home/user38
Ctrl+O
Ctrl+X

Результат показывает, никто из пользователей не может подсматривать чужие проекты:

```
root@ruvds-d0ohk: /home/good
total 128
drwxr-xr-x 7 good   good   4096 Oct 28 10:04 good
drwxr-xr-x 2 nodejs nodejs 4096 Mar 12  2021 nodejs
drwx--x--x 5 user01 user01 4096 Oct 28 09:38 user01
drwx--x--x 2 user02 user02 4096 Oct 21 18:58 user02
drwx--x--x 9 user03 user03 4096 Oct 24 22:04 user03
drwx--x--x 2 user04 user04 4096 Oct 17 18:27 user04
drwx--x--x 5 user05 user05 4096 Oct 24 17:45 user05
drwx--x--x 2 user06 user06 4096 Oct 17 18:27 user06
drwx--x--x 2 user07 user07 4096 Oct 17 18:27 user07
drwx--x--x 2 user08 user08 4096 Oct 17 18:27 user08
drwx--x--x 2 user09 user09 4096 Oct 17 18:27 user09
drwx--x--x 2 user10 user10 4096 Oct 17 18:27 user10
drwx--x--x 2 user11 user11 4096 Oct 17 18:27 user11
drwx--x--x 2 user12 user12 4096 Oct 17 18:27 user12
drwx--x--x 2 user21 user21 4096 Oct 17 18:27 user21
drwx--x--x 2 user22 user22 4096 Oct 17 18:27 user22
drwx--x--x 2 user23 user23 4096 Oct 17 18:27 user23
drwx--x--x 2 user24 user24 4096 Oct 17 18:27 user24
drwx--x--x 2 user25 user25 4096 Oct 17 18:27 user25
drwx--x--x 2 user26 user26 4096 Oct 17 18:27 user26
drwx--x--x 6 user27 user27 4096 Oct 28 09:32 user27
drwx--x--x 2 user28 user28 4096 Oct 17 18:27 user28
drwx--x--x 2 user29 user29 4096 Oct 17 18:27 user29
drwx--x--x 2 user30 user30 4096 Oct 17 18:27 user30
drwx--x--x 2 user31 user31 4096 Oct 17 18:27 user31
drwx--x--x 2 user32 user32 4096 Oct 17 18:27 user32
drwx--x--x 2 user33 user33 4096 Oct 17 18:27 user33
drwx--x--x 2 user34 user34 4096 Oct 17 18:27 user34
drwx--x--x 2 user35 user35 4096 Oct 17 18:27 user35
drwx--x--x 2 user36 user36 4096 Oct 17 18:27 user36
drwx--x--x 2 user37 user37 4096 Oct 17 18:27 user37
drwx--x--x 2 user38 user38 4096 Oct 21 18:57 user38
root@ruvds-d0ohk:/home/good#
```

Создание поддоменов на reg.ru

Создадим поддомены для DNS-службы регистратора:

Создадим запись TXT для домена @ со значением "goodarget" для проверки SSL-сертификата.

После того как DNS-записи будут прописаны, через несколько минут они появятся на NS-сервере и станут доступны для всех прочих DNS-серверов [17]. В период от 5 минут до часа информация о записях обновится на всех серверах или через день. Отслеживать этот процесс можно с помощью DNS-чекеров, например DNS Checker или Global DNS Checker. Укажите имя домена и тип

ресурсной записи, чтобы узнать, на каких DNS-серверах доступна обновленная информация. Если вы также меняли список DNS-серверов, на которые делегирован домен, обновление информации может занять до 24 часов.

Список поддоменов для пользователей:

Юзер	Поддомен	Поддомен/app Node.js порт	Папка сайта NGINX
user01	bot1.daibot.fun www.bot1.daibot.fun	3001	/var/www/bot1.daibot.fun
user02	bot2.daibot.fun www.bot2.daibot.fun	3002	/var/www/bot2.daibot.fun
user03	bot3.daibot.fun www.bot3.daibot.fun	3003	/var/www/bot3.daibot.fun
user04	bot4.daibot.fun www.bot4.daibot.fun	3004	/var/www/bot4.daibot.fun
user05	bot5.daibot.fun www.bot5.daibot.fun	3005	/var/www/bot5.daibot.fun
user06	bot6.daibot.fun www.bot6.daibot.fun	3006	/var/www/bot6.daibot.fun
user07	bot7.daibot.fun www.bot7.daibot.fun	3007	/var/www/bot7.daibot.fun
user08	bot8.daibot.fun www.bot8.daibot.fun	3008	/var/www/bot8.daibot.fun
user09	bot9.daibot.fun www.bot9.daibot.fun	3009	/var/www/bot9.daibot.fun
user10	bot10.daibot.fun www.bot10.daibot.fun	3010	/var/www/bot10.daibot.fun

Юзер	Поддомен	Поддомен/app Node.js порт	Папка сайта NGINX
user11	bot11.daibot.fun www.bot11.daibot.fun	3011	/var/www/bot11.daibot.fun
user12	bot12.daibot.fun www.bot12.daibot.fun	3012	/var/www/bot12.daibot.fun
user21	bot21.daibot.fun www.bot21.daibot.fun	3021	/var/www/bot21.daibot.fun
user22	bot22.daibot.fun www.bot22.daibot.fun	3022	/var/www/bot22.daibot.fun
user23	bot23.daibot.fun www.bot23.daibot.fun	3023	/var/www/bot23.daibot.fun
user24	bot24.daibot.fun www.bot24.daibot.fun	3024	/var/www/bot24.daibot.fun
user25	bot25.daibot.fun www.bot25.daibot.fun	3025	/var/www/bot25.daibot.fun
user26	bot26.daibot.fun www.bot26.daibot.fun	3026	/var/www/bot26.daibot.fun
user27	bot27.daibot.fun www.bot27.daibot.fun	3027	/var/www/bot27.daibot.fun
user28	bot2.daibot.fun www.bot28.daibot.fun	3028	/var/www/bot28.daibot.fun
user29	bot29.daibot.fun www.bot29.daibot.fun	3029	/var/www/bot29.daibot.fun
user30	bot30.daibot.fun www.bot30.daibot.fun	3030	/var/www/bot30.daibot.fun
user31	bot31.daibot.fun www.bot31.daibot.fun	3031	/var/www/bot31.daibot.fun
user32	bot32.daibot.fun www.bot32.daibot.fun	3032	/var/www/bot32.daibot.fun
user33	bot33.daibot.fun www.bot33.daibot.fun	3033	/var/www/bot33.daibot.fun
user34	bot34.daibot.fun www.bot34.daibot.fun	3034	/var/www/bot34.daibot.fun
user35	bot35.daibot.fun www.bot35.daibot.fun	3035	/var/www/bot35.daibot.fun
user36	bot36.daibot.fun www.bot36.daibot.fun	3036	/var/www/bot36.daibot.fun
user37	bot37.daibot.fun www.bot37.daibot.fun	3037	/var/www/bot37.daibot.fun
user38	bot39.daibot.fun www.bot38.daibot.fun	3038	/var/www/bot38.daibot.fun

Веб-сервер Nginx может выступать прокси-сервером, то есть входящий поток данных, поступающих по протоколу http, можно перебрасывать на порт.
Создадим файл create_www_dir.sh:

$ **touch** create_www_dir.sh
$ **chmode** ug+x create_www_dir.sh
$ **nano** create_www_dir.sh

Внесём в файл create_www_dir.sh текст для всех папок пользователей:

#!/bin/bash
mkdir /var/www/bot1.daibot.fun
chown -R user01:www-data /var/www/bot1.daibot.fun
chmod 771 /var/www/bot1.daibot.fun
....
Ctrl+O
Ctrl+X

Сделайте его исполняемым и запустим на выполнение:
$ **chmod** +x create_www_dir.sh
Теперь добавим Nginx в список исключений ufw. Для этого проверим список доступных приложений ufw:
ufw app list
В этой инструкции мы не будем защищать наши сайты SSL-сертификатом, поэтому добавим в список исключений ufw только службу Nginx, отвечающую за HTTP-соединение:

```
# ufw allow 'Nginx HTTP'
# ufw reload
```

Можно проверить, что Nginx добавлен в список исключений фаервола:
ufw status
По умолчанию в Nginx создан один серверный блок default, который можно использовать в качестве шаблона для создания собственных блоков сайтов.
Приведём текст блока default:

```
server {
    # Настройки по умолчанию
    listen 80 default_server;
    listen [::]:80 default_server;

    # nginx будет смотреть только публичные статические файлы проекта
    root /var/www/daibot.fun/public;
    # Заменить _ на доменное имя сайта, если есть
    server_name daibot.fun www.daibot.fun;
    index index.html index.htm index.nginx-debian.html;

    # Перехватим статические файлы, отключим логи, увеличим время expire, отключим лог
доступа и ошибок для favicon.ico и robots.txt
    location ~* ^.+.(js|css|png|jpg|jpeg|gif|ico|woff)$ {
    access_log off;
    expires max;
    }
```

```
    location = /favicon.ico {
    log_not_found off;
    access_log off;
    }

    location = /robots.txt {
    allow all;
    log_not_found off;
    access_log off;
    }
}
```

Скопируем файл default для конфигурирования сайта поддомена студента:
cp /etc/nginx/sites-available/default /etc/nginx/sites-available/bot27.daibot.fun
Откроем наш новый файл:
$ **nano** /etc/nginx/sites-available/bot27.daibot.fun
Это большой файл с закомментированным описанием каждого из блоков. Если исключить строки комментариев, то содержимое файла будет таким:

```
server {
        listen 80;
        listen [::]:80;

        root /var/www/bot27.daibot.fun;
        index index.html index.htm index.nginx-debian.html;

        server_name bot27.daibot.fun www.bot27.daibot.fun;

        location / {
                try_files $uri $uri/ =404;
        }
}
```

Например, подключение виртуальных хостов для поддомемов bot[id]. Наш виртуальный хост создан, и его нужно подключить в nginx. Создадим для этого символическую ссылку для сайта:
ln -s /etc/nginx/sites-available/bot27.daibot.fun /etc/nginx/sites-enabled/
Теперь у нас два виртуальных хоста:
default – создан по умолчанию и отвечает за обработку запросов к сайтам, чьи доменные имена не совпадают с нашими сайтами поддоменов;
bot27.daibot.fun – новый виртуальный хост для размещения сайта студента.
Чтобы избежать проблем с количеством доступной хэш-памяти, исправим конфигурационный файл Nginx:

$ **nano** /etc/nginx/nginx.conf

Найдём в этом файле строку server_names_hash_bucket_size и раскомментируем её (уберём # в начале строки).
Сохраним изменённый файл и проверим правильность конфигурации веб-сервера:
nginx -t

В результате, если всё было сделано правильно, на выходе получим сообщение:

```
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

Перезагрузим наш веб-сервер, чтобы все изменения вступили в силу:

systemctl restart nginx

В файле bot27.daibot.fun добавим ещё один раздел location и внесем изменения, приведя его к виду:

```
        location /app {
                proxy_pass http://localhost:3027;
                proxy_set_header Host $host;
        }
```

Создать файл /home/user27/index.js

```
const http = require('http');

const hostname = 'localhost';
const port = 3027;

const server = http.createServer((req, res) => {
        res.statusCode = 200;
        res.setHeader('Content-Type', 'text/plain');
        res.end('This is the Main App user27 subdomain
bot27.daibot.fun!\n');
});

server.listen(port, hostname, () => {
        console.log(`Server running at http://${hostname}:${port}/`);
});
```

Команды подключения Node-приложения на порт:

```
# nginx -t    # Проверить нет ли ошибок
```

```
# systemctl restart nginx
# ufw allow 3027/tcp
# ufw reload
# chmod +x index.js
# pm2 start /home/user27/app/index.js
# curl http://localhost:3027          # Проверить как работает сервлет
```

Или ввести в адресной строке:

Открыл порты:

```
# ufw allow 3001/tcp
# ufw allow 3002/tcp
# ufw allow 3003/tcp
# ufw allow 3004/tcp
# ufw allow 3005/tcp
# ufw allow 3006/tcp
# ufw allow 3007/tcp
# ufw allow 3008/tcp
# ufw allow 3009/tcp
# ufw allow 3010/tcp
# ufw allow 3011/tcp
# ufw allow 3012/tcp
# ufw allow 3021/tcp
# ufw allow 3022/tcp
# ufw allow 3023/tcp
# ufw allow 3024/tcp
# ufw allow 3025/tcp
# ufw allow 3026/tcp
# ufw allow 3027/tcp
# ufw allow 3028/tcp
# ufw allow 3029/tcp
# ufw allow 3030/tcp
# ufw allow 3031/tcp
# ufw allow 3032/tcp
# ufw allow 3033/tcp
# ufw allow 3034/tcp
# ufw allow 3035/tcp
# ufw allow 3036/tcp
# ufw allow 3037/tcp
# ufw allow 3038/tcp
# ufw reload
# systemctl restart nginx
```

Учебный проект "Daibot.fun". Часть - 4.

Команды пользователя ($) можно тестировать на сервере. Скрипт бота может не заработать ввиду "сырой" версии, приведен для ознакомления. "Идеальный" корпаративный бот приведен в последних лекциях.

Задание проекта: Создать мини-приложение Телеграм бот генерации визуального улучшения плакатов спецэффектами SVG-фильтов, на базе виртуального выделенного сервера с доменным именем daibot.fun.

Базовый алгоритм ИИ: написан на языке программирования JavaScript.

Постановка задачи

```
План работ:
- получить сертификат SSL-шифрования для https-доступа к веб-серверу, на основе имени домена и для каждого поддомена;
- открыть порт 433 на VPS-сервере;
- настроить веб-сервер nginx для работы поддоменов с SSL-сертификатом.
```

Продолжаем улучшать экосистему учебного проекта

Провайдер VPS-сервера предлагает свои услуги DNS бесплатно для VPS-сервера, не стоит отказываться от них.

Добавить у reg.ru регистратора NS-записи ns1.ruvds.com и ns2.ruvds.com не возможно для домена, только для поддоменов.

Добавим еще поддержку ipv6 у регистратора reg.ru. Это AAAA запись в рабочей зоне fe80:0:0:0:0:14d5:fe01:da15. В Window и Linux не отвечает на команду ping -6 fe80:0:0:0:0:14d5:fe01:da15. Поэтому добавил правило в банмаэуер:

```
# ufw allow from fe80:0:0:0:0:14d5:fe01:da15
# ufw reload
```

Ничего не изменилось, не пингует. Поэтому удаляю правило по номеру:

```
# ufw status numbered
# ufw delete 69
# ufw reload
```

И у регистратора пока убрал запись AAAA. У регистратора убрал запись A www->45.8.230.131 и вставил правильную: CNAME www-> diabot.fun.

Получение сертификата на домен для доступа по протоколу https

Установка на VPS сервер SSL-сертификата относится к решениям типа “good-practice” в сфере обеспечения безопасности сайтов и онлайн-сервисов. SSL-сертификаты позволяют шифровать поток информации между пользователями вашего сервиса (сайта) и сервером, а также удостоверять подлинность соединения [18].

Он может быть бесплатным (Let’s Encrypt) и платным. Разница состоит в типе сертификата и длительности его действия. Бесплатный действует 90 дней, т.е. каждые три месяца его придется выписывать заново. Коммерческие сертификаты, как правило, выпускаются на один-два, а то и целых три года, что, очевидно, снижает частоту перевыпуска в разы. Также при взломе бесплатного сертификата ответственность лежит только на владельце сервиса.

SSL-сертификаты это про веб-сайты, а не про VPS, однако, а что, собственно, крутится на сервере, если не веб-сервис или сайт?

Let’s Encrypt (https://letsencrypt.org/) – бесплатный сервис выдачи SSL (Secure Sockets Layer) и TLS (Transport Layer Security) сертификатов, которые обеспечивают шифрование данных между сервером и браузером пользователя.

Что делает Let’s Encrypt особенно удобным и популярным, так это то, что сертификаты выдаются бесплатно.

Прежде чем начать, необходимо знать о некоторых нюансах получения сертификата Let's Encrypt [19]. Для получения сертификата используется утилита Certbot [20]. В Let’s Encrypt утилита Certbot отвечает за управление SSL/TLS сертификатами. При запросе Certbot выполняет проверку домена. Для прохождения проверки необходимо выбрать один из двух вариантов:

1) создать TXT-запись в DNS. Я сделал TXT для домена @ со значением "goodarget". Данный метод проще второго, но он позволит настроить автоматическое продление сертификата только для некоторых DNS, для которых есть отдельные certbot-плагины. Поэтому данный способ, в большинстве случаев, будет удобен для проведения тестов;

2) поднять веб-сервер nginx, далее в его корне создается каталог .well-known, а в нем файл с произвольным названием. После корневой центр отправляет запрос серверу на загрузку данного файла и, в случае успеха, выдает сертификаты для указанного доменного имени.

SSL-сертификат выдается на 90 дней, поэтому необходимо по расписанию запускать команду на автоматическое продление ключа. Когда проходит 60 дней после начала использования нового сертификата, центр Let's Encrypt может выдать новый.

Если выполнять запрос для домена 3 уровня и выше, он должен пройти DNS проверку на всех уровнях. Например, домен layer3.layer2.com должен отвечать на запросы как для layer3.layer2.com, так и для layer2.com.

Выберем второй вариант.

Данный способ позволяет без проблем настроить автоматическое продление срока годности сертификата. Запрашивать сертификат Let's Encrypt проще всего с веб-сервера daibot.fun, на котором запущен сайт для домена.

Настройка веб-сервера Linux NGINX для проверки домена.

Сервер nginx должен используется для сайта только http, в секцию server добавляем возможность отдачи скриптов по запросу:

```
location ~ /.well-known {
    root /var/www/daibot.fun/public;
    allow all;
}
if ($uri !~ /.well-known){
    return 301 https://$host$request_uri;
}
```

Данными строчками мы говорим, что для всех запросов после /.well-known необходимо отдавать скрипты из каталога /var/www/daibot.fun/public; allow all предоставляет доступ всем.

После проверяем конфигурацию и перезапускаем nginx:

```
# nginx -t
```

```
# systemctl reload nginx
```

Установка утилиты для получения сертификата Certbot для Linux.

```
# apt update
# apt install certbot
```

Первое получение сертификата

Если мы подтверждаем право на домен при помощи веб-сервера nginx, выполняем команду с таким синтаксисом:

```
# certbot certonly --webroot --agree-tos --email <почта администратора домена> --webroot-path <путь до каталога с файлами проверки> -d <домен 1> -d <домен 2> -d ...
```

Где:
certonly – опция, что это запрос нового сертификата;
webroot – опция, что проверка будет выполняться на основе запроса к корню сайта;
agree-tos — даём согласие на лицензионное соглашение;
email — почтовый адрес администратора домена;
webroot-path — каталог в системе Linux, который является корневым для сайта;
d — перечисление доменов, для которых запрашиваем сертификат.
Пример команды для учебного проекта при использовании веб-сервера NGINX:

```
# certbot certonly --webroot --agree-tos --email good_arget@mail.ru --webroot-path /var/www/daibot.fun/public -d daibot.fun -d www.daibot.fun
```

Внимание, что для поддоменов нельзя задать значение -d *.daibot.fun.

После успешного выполнения команды, сертификаты будут созданы в каталоге /etc/letsencrypt/archive/daibot.fun, а также симлинки, команда ln -s ..., на них в

каталоге /etc/letsencrypt/live/daibot.fun. Публичный ключ будет с именем cert.pem, а приватный — privkey.pem.
При настройке приложений, стоит указывать пути до симлинков ../live/daibot.fun, так как при обновлении файлы в первом каталоге будут меняться, во втором — нет.

Примечание. Команда ln в Linux используется для создания жёстких или символических ссылок на файлы или директории. Создание ссылки на директорию: ln -s /path/to/source_dir /path/to/target_dir. Это создаст символическую ссылку на директорию source_dir в директории target_dir.

Настройка HTTPS

Сначала проверьте текущий статус брандмауэра вашей системы:

```
# ufw status
```

Скорее всего, трафика HTTPS не будет в списке разрешённых, только HTTP. Чтобы это исправить, необходимо разрешить Nginx HTTPS.

```
# ufw allow 'Nginx HTTPS'
# ufw reload
```

Пока не будем закрывать разрешение на Nginx HTTP. Но для безопасности необходимо сделать это # ufw delete allow 'Nginx HTTP'.

Настройка Nginx

Файл privkey.pem – приватный ключ сертификата, надо прописать в параметре ssl_certificate_key. В случае получение бесплатного сертификата от Let's Encrypt, мы получаем 4 файла, один из которых называется fullchain.pem – именно он и содержит все необходимые последовательности для всех центров сертификации. Один сертификат на домен. В файле default добавить порт:

```
server {
    listen 443 ssl default_server;
    listen [::]:443 ssl default_server;

    ssl_certificate /etc/letsencrypt/live/daibot.fun/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/daibot.fun/privkey.pem;
```

Расширение уже выданного сертификата

У LetsEncrypt есть ограничения на использования одного сертификата для нескольких доменных имён. Можно создать один сертификат для максимум 100 доменов. Проверим существующие `$ cerbot certificates`:

В 2017 году сообщали, что добавить поддомены к существующему сертификату от Let's Encrypt не получится. Нужно сначала удалить сертификат, командой revoke и delete, а затем создать новый сертификат сразу для нескольких доменов. Но.

Иногда нам может потребоваться добавить дополнительные домены к уже выданному сертификату. Для этой цели пригодится опция –expand от Certbot. После выдачи сертификата доменов мы можем расширить его, включив дополнительные поддомены:

```
# certbot certonly --expand --manual --preferred-challenges=http -d bot1.daibot.fun -d www.bot1.daibot.fun
```

Не получилась сделать расширение просто. То отзовем сертификат:

```
# certbot revoke --cert-path /etc/letsencrypt/live/daibot.fun/cert.pem
```

И создадим снова из файла sert_all.sh:

```
#!/bin/bash
certbot certonly --webroot --agree-tos --email good_arget@mail.ru --webroot-path /var/www/daibot.fun/public -d daibot.fun -d www.daibot.fun -d bot1.daibot.fun -d www.bot1.daibot.fun
certbot certonly --webroot --agree-tos --email good_arget@mail.ru --webroot-path /var/www/bot2.daibot.fun -d bot2.daibot.fun -d www.bot2.daibot.fun
certbot certonly --webroot --agree-tos --email good_arget@mail.ru --webroot-path /var/www/bot3.daibot.fun -d bot3.daibot.fun -d www.bot3.daibot.fun
certbot certonly --webroot --agree-tos --email good_arget@mail.ru --webroot-path /var/www/bot4.daibot.fun -d bot4.daibot.fun -d www.bot4.daibot.fun
certbot certonly --webroot --agree-tos --email good_arget@mail.ru --webroot-path /var/www/bot5.daibot.fun -d bot5.daibot.fun -d www.bot5.daibot.fun
certbot certonly --webroot --agree-tos --email good_arget@mail.ru --webroot-path /var/www/bot6.daibot.fun -d bot6.daibot.fun -d www.bot6.daibot.fun
certbot certonly --webroot --agree-tos --email good_arget@mail.ru --webroot-path /var/www/bot7.daibot.fun -d bot7.daibot.fun -d www.bot7.daibot.fun
certbot certonly --webroot --agree-tos --email good_arget@mail.ru --webroot-path /var/www/bot8.daibot.fun -d bot8.daibot.fun -d www.bot8.daibot.fun
certbot certonly --webroot --agree-tos --email good_arget@mail.ru --webroot-path /var/www/bot9.daibot.fun -d bot9.daibot.fun -d www.bot9.daibot.fun
certbot certonly --webroot --agree-tos --email good_arget@mail.ru --webroot-path /var/www/bot10.daibot.fun -d bot10.daibot.fun -d www.bot10.daibot.fun
certbot certonly --webroot --agree-tos --email good_arget@mail.ru --webroot-path /var/www/bot11.daibot.fun -d bot11.daibot.fun -d www.bot11.daibot.fun
certbot certonly --webroot --agree-tos --email good_arget@mail.ru --webroot-path /var/www/bot12.daibot.fun -d bot12.daibot.fun -d www.bot12.daibot.fun
certbot certonly --webroot --agree-tos --email good_arget@mail.ru --webroot-path /var/www/bot21.daibot.fun -d bot21.daibot.fun -d www.bot21.daibot.fun
certbot certonly --webroot --agree-tos --email good_arget@mail.ru --webroot-path /var/www/bot22.daibot.fun -d bot22.daibot.fun -d www.bot22.daibot.fun
certbot certonly --webroot --agree-tos --email good_arget@mail.ru --webroot-path /var/www/bot23.daibot.fun -d bot23.daibot.fun -d www.bot23.daibot.fun
certbot certonly --webroot --agree-tos --email good_arget@mail.ru --webroot-path /var/www/bot24.daibot.fun -d bot24.daibot.fun -d www.bot24.daibot.fun
certbot certonly --webroot --agree-tos --email good_arget@mail.ru --webroot-path /var/www/bot25.daibot.fun -d bot25.daibot.fun -d www.bot25.daibot.fun
certbot certonly --webroot --agree-tos --email good_arget@mail.ru --webroot-path /var/www/bot26.daibot.fun -d bot26.daibot.fun -d www.bot26.daibot.fun
certbot certonly --webroot --agree-tos --email good_arget@mail.ru --webroot-path /var/www/bot27.daibot.fun -d bot27.daibot.fun -d www.bot27.daibot.fun
certbot certonly --webroot --agree-tos --email good_arget@mail.ru --webroot-path /var/www/bot28.daibot.fun -d bot28.daibot.fun -d www.bot28.daibot.fun
certbot certonly --webroot --agree-tos --email good_arget@mail.ru --webroot-path /var/www/bot29.daibot.fun -d bot29.daibot.fun -d www.bot29.daibot.fun
certbot certonly --webroot --agree-tos --email good_arget@mail.ru --webroot-path /var/www/bot30.daibot.fun -d bot30.daibot.fun -d www.bot30.daibot.fun
certbot certonly --webroot --agree-tos --email good_arget@mail.ru --webroot-path /var/www/bot31.daibot.fun -d bot31.daibot.fun -d www.bot31.daibot.fun
certbot certonly --webroot --agree-tos --email good_arget@mail.ru --webroot-path /var/www/bot32.daibot.fun -d bot32.daibot.fun -d www.bot32.daibot.fun
certbot certonly --webroot --agree-tos --email good_arget@mail.ru --webroot-path /var/www/bot33.daibot.fun -d bot33.daibot.fun -d www.bot33.daibot.fun
certbot certonly --webroot --agree-tos --email good_arget@mail.ru --webroot-path /var/www/bot34.daibot.fun -d bot34.daibot.fun -d www.bot34.daibot.fun
certbot certonly --webroot --agree-tos --email good_arget@mail.ru --webroot-path /var/www/bot35.daibot.fun -d bot35.daibot.fun -d www.bot35.daibot.fun
certbot certonly --webroot --agree-tos --email good_arget@mail.ru --webroot-path /var/www/bot36.daibot.fun -d bot36.daibot.fun -d www.bot36.daibot.fun
certbot certonly --webroot --agree-tos --email good_arget@mail.ru --webroot-path /var/www/bot37.daibot.fun -d bot37.daibot.fun -d www.bot37.daibot.fun
certbot certonly --webroot --agree-tos --email good_arget@mail.ru --webroot-path /var/www/bot38.daibot.fun -d bot38.daibot.fun -d www.bot38.daibot.fun
```

После создания сертификатов необходимо удалить из файла default настройки nginx разрешающие отдавать скрипт, а то будет зацикливание потоков:

```
location ~ /.well-known {
    root /var/www/daibot.fun/public;
    allow all;
}
if ($uri !~ /.well-known){
    return 301 https://$host$request_uri;
}
```

Добавить во все конфигурационные файлы nginx для сайтов поддоменов:

```
listen 443 ssl;
listen [::]:443 ssl;

ssl_certificate /etc/letsencrypt/live/bot1.daibot.fun/fullchain.pem;
ssl_certificate_key /etc/letsencrypt/live/bot1.daibot.fun/privkey.pem;
```

Обновление сертификата

Если планируется использовать сертификат SSL от Let’S Encrypt более 90 дней, его придется систематически обновлять. И делать это рекомендуется заранее, минимум за 30 дней до истечения срока действия. Иначе возникают риски временной неработоспособности протокола SSL, когда сайт перестает открываться по прежним ссылкам (браузеры предупреждают об ошибке).

Выполняется процедура обновления командой:

```
$ certbot renew
```

После ее ввода будут проверены все ранее выпущенные и установленные в системе сертификаты и заново созданы те, по которым сроки подходят к концу.

Если хочется настроить автоматический перевыпуск SSL, нужно ввести команду cron:

```
$ crontab -e
30 2 * * 1 /usr/bin/certbot renew >> /var/log/renew-ssl.log
```

В таком виде она каждый понедельник в 2:30 будет выполнять проверку актуальности всех SSL и записывать результат в файл с указанным названием.

Учебный проект "Daibot.fun". Часть - 5.

Скрипт бота может не заработать ввиду "сырой http-версии", приведен для ознакомления. "Идеальный" HTTPS-корпаративный бот далее.

Задание проекта: Создать мини-приложение Телеграм бот генерации визуального улучшения плакатов спецэффектами SVG-фильтов, на базе виртуального выделенного сервера с доменным именем daibot.fun.

Базовый алгоритм ИИ: написан на языке программирования JavaScript.

Постановка задачи

Настроить WEBHOOK для общения ТГ-сервера с Телеграм-ботам [21]. В VPS-сервере по умолчанию инсталлирован веб-сервер Nginx.

Выбран вариант общения Телеграм-бота с ТГ-сервером: через WEBHOOK по протоколу https. На рисунке приведен пример конкретной схемы работы ТГ-бота в сети из статьи 1.

Почему WEBHOOK? Потому что, привязка бота к доменному имени гарантирует доступ ТГ-сервера к сервлету бота на VPS по протоколу защиты данных SSL и что не маловажно можно сменить физическое размещение сервлета на другой

VPS-сервер, так как бот не будет привязан к ip-адресу VPS-сервера, да и SSL-сертификат трудно получить на ip-адрес VPS-сервера. Трудно; точнее спорно, что этот самоподписанный SSL-сертификат – это хорошо с точки зрения безопасности.
Если в обмене данными участвует протокол https, то посредником между ТГ-сервером и ботом будет веб-сервер. Тогда бот является CGI-сценарием. В дальнейшем назовём его сервлетом веб-сервера, по старой традиции многозвенных архитектур Java.

```
План работ:
- сделать телеграм-бота.
```

Проверим полученный сертификат SSL
Убедимся что полученный сертификат – именно тот, что нам нужен:
openssl x509 -text -in /etc/letsencrypt/live/daibot.fun/cert.pem
openssl x509 -text -in /etc/letsencrypt/live/bot1.daibot.fun/cert.pem

Теория о сети. Типы компьютерных сетей:
WAN (Глобальная сеть);
MAN (Городская сеть);
LAN (Локальная вычислительная сеть);
PAN (Персональная локальная сеть);
WLAN (Беспроводная локальная сеть).

Разница между 127.0.0.1 и 0.0.0.0.
Имеем два специальных и зарезервированных IP-адреса:
– 127.0.0.1 (называется "localhost" или "loopback");
– 0.0.0.0.
В сети IP-адрес, такой как "127. 0. 0. 1" или "0. 0. 0. 0", имеет фундаментальное значение для управления тем, как осуществляется обмен данными между сетевыми приборами и инфраструктурой сети. Компьютеры могут взаимодействовать друг с другом по IP-адресу, но этот адрес 127.0.0.1 не может использоваться компьютерами для связи друг с другом.

В качестве примера: веб-сервер в определенной операционной системе может объявить 127.0.0.1 в качестве локального имени хоста, поэтому html-страницы можно тестировать, запуская локально доступ к сайту, до их развертывания в публичный доступ.

Как работает 127.0.0.1?

IP-адреса всегда прикрепляются к сообщениям прикладного программного обеспечения протокола TCP / IP.

Если Ваше разработанное приложение отправляете сообщения по сети LAN на адреса обратной связи, такие как 127.0.0.1, ни одно из сообщений не попадет за пределы локальной сети.

Ваши сетевые приложения не имеют доступа к внешней сети и не требуют доступа к Интернету.

Что такое 0.0.0.0?

В IPV4 адрес 0.0.0.0 используется для обозначения недопустимого, неизвестного или недоступного пункта назначения.

В прошлом этот адрес использовалось для указания того, что нет конкретного адреса, на который нужно перенаправлять сетевые пакеты. Фактически, это обозначение для сети: "нет конкретного адреса", потому что IP-адрес 0.0.0.0 не является маршрутизирующим. Маршрута к указанному адресу места назначения нет, и адрес не поддается маршрутизации.

При отсутствии подключения к Интернету Ваш персональный компьютер и другие клиентские устройства обычно отображают 0.0.0.0 в качестве своего IP-адреса.

Сетевые приложения, которые используют 0.0.0.0 в исходном коде программы, анализируют весь входящий сетевой трафик компьютера. Таким образом, использование адрес 0.0.0.0 в программе, имеет риски безопасности, так как может быть уязвим для внешних вызовов.

Сервлет server27.js для Node.js использующий специальный адрес.

Так как я настроил запросы через nginx, нужно поменять некоторые настройки. Открываем файл с вашим сервлетом:

```
$ nano /home/user27/app/server27.js
```

Находим строки:

```
server.listen(3027, '0.0.0.0', () => {
  console.log('Start NodeJS Server on http://0.0.0.0:3027/');
});
```

И меняем их не на такой исходный код:

```
server.listen(3027, '127.0.0.1', () => {
  console.log('Start NodeJS Server on http://127.0.0.1:3027/');
});
```

А на такой полный текст:

```
const http = require('http');
const hostname = 'localhost';
```

```
const port = 3027;
const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World!\n');
});
server.listen(port, hostname, () => {
  console.log(`Start NodeJS Server on http://${hostname}:${port}/`);
});
```

Что одно и то же, но более элегантно.
Мы поменяли 0.0.0.0 на 127.0.0.1, чтобы наш сервер не слушал на всех сетевых интерфейсах – нам достаточно только локальной петли. Это более безопасный вариант работы сервлета.
Перезапустим наше приложение node:

```
$ pm2 stop 0
$ pm2 start /home/user27/app/server27.js
```

Проверим, что наше приложение слушает на порту 3037 и уже на адресе 127.0.0.1:

```
$ ss -tunlp | grep :3027
```

Мы должны увидеть:

```
tcp   LISTEN  0      128     127.0.0.1:3027     0.0.0.0:*    users:(("node",pid=2503,fd=10))
```

Развёртывание Node.js приложение.
Сетевые js-сервлеты, написанные для Node.js, поставляются с файлом package.json. Это файл манифеста, в котором описаны необходимые программные библиотеки для запуска сервлета.
Как правило, разработчик поставляет приложение с файлом package.json. В этом файле описаны js-пакеты, необходимые для работы приложения.
Перейдем в папку приложения, куда скопирован исходный код сервлета server27.js и манифест package.json:

```
$ cd /home/user27/app/
```

Если файл package.json отсутствует в проекте, то создайте его командой:

```
$ npm init
```

Можно нажимать Enter и ничего не вводить менеджер пакетов сам обнаруживает файл server27.js, как точку входа в приложение.
Я ввёл два поля description "App site bot27.daibot.fun port-3027" и поле author "goodarget (c)".

```
mc [user27@ruvds-d0ohk]:~/app
package name: (app)
version: (1.0.0)
description: App site bot27.daibot.fun port-3027
entry point: (server27.js)
test command:
git repository:
keywords:
author: goodarget (c) 2024
license: (ISC)
About to write to /home/user27/app/package.json:

{
  "name": "app",
  "version": "1.0.0",
  "description": "App site bot27.daibot.fun port-3027",
  "main": "server27.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "goodarget (c)",
  "license": "ISC"
}


Is this OK? (yes)
```

Вопросы может вызвать поле scripts. Давайте разберёмся с ним. Раздел scripts содержит набор команд, которые можно будет использовать при работе с приложением npm start. Указание стартового скрипта считается правилом хорошего тона. Поэтому узел "test" замените на "start". А в качестве скрипта запуска необходимо ввести команду "node server27.js", запускающую по умолчанию находящийся в папке проект. Знайте, теперь, находясь в папке приложения app, можно ввести команду "npm start" и это будет синонимом команды "node server27.js".

```
mc [user27@ruvds-d0ohk]:~/app
/home/us~ge.json   [----] 30 L:[  1+ 6   7/ 12] *(16[*][X]
{
  "name": "app",
  "version": "1.0.0",
  "description": "App site bot27.daibot.fun port-3027",
  "main": "server27.js",
  "scripts": {
    "start": "node server27.js"
  },
  "author": "goodarget (c)",
  "license": "ISC"
}
 1Help  2Save  3Mark  4Re~ac 5Copy  6Move  7Se~ch 8De~te
```

Оказалась я ранее запустил сервлет "node index.js", который прослушивает порт 3027 и под правами root. Это можно посмотреть командой:

```
$ ps axu | grep node
```

```
mc [user27@ruvds-d0ohk]:~/app
user27@ruvds-d0ohk:~/app$ ps axu | grep node
root       67550  0.2  2.6 1095148 26056 ?       Ssl  Oct28  31:46 node /home/user27/app/index.js
user32     90642  0.2  4.6 1238516 44860 ?       Ssl  Oct30  32:26 node /home/user32/app/app.js
user33    111969  0.1  3.7 11717320 36820 ?      Ssl  Oct31  22:38 node /home/user33/app/server.mjs
user22    112401  0.2  3.6 11718596 35040 ?      Ssl  Oct31  23:47 node /home/user22/app/server.js
user21    120309  0.0  0.7 725836  7196 ?        Ssl  Nov01   0:00 /usr/local/nvm/versions/node/v20.18.0/bin/
user21    123377  0.1  2.4 1026652 23488 ?       Ssl  Nov01  19:37 node /home/user21/app/app.js
user03    179157  0.2  3.5 1035160 34856 ?       Ssl  Nov03  14:40 node /home/user03/app/server.js
user11    276133  0.2  3.6 1028616 35636 ?       Ssl  13:35   1:12 node /home/user11/app/server.js
user27    286189  0.0  0.0  10760   656 pts/0    S+   22:58   0:00 grep node
user27@ruvds-d0ohk:~/app$
```

Теперь можно остановить приложение index.js, которое занимает порт 3027, только под пользователем root:

```
# kill 67550
```

Но лучше найти его у себя менеджером pm2 и остановить:

```
$ pm2 monit
$ pm2 delete 0
```

Теперь можно запустить приложение командой `$ npm start`.
Посмотреть работу приложения можно в любом браузере:

```
https://bot27.daibot.fun/app
```

А вот доступ к порту по ip-адресу VPS-сервера не получится, так как порт 3027 не прослушивается сервлетом server27.js:

```
http://45.8.230.131:3027/
```

Прервать работу сервлета можно Ctrl+X, так как он не запускался через утилиту pm2.

Продолжим конфигурировать проект.
Ранее описанный исходный код примера server27.js содержал подключение пакета программной библиотеки http, но в файле управления версиями package.json это не отражено. Исправляем это командой [22]:

```
$ npm install http --save
```

Мы видим, появилась зависимость от библиотеки в поле "dependencies".

```
mc [user27@ruvds-d0ohk]:~/app
/home/us~ge.json   [----] 30 L:[  1+ 6   7/ 15] *(167 / 279[*][X]
{
  "name": "app",
  "version": "1.0.0",
  "description": "App site bot27.daibot.fun port-3027",
  "main": "server27.js",
  "scripts": {
    "start": "node server27.js"
  },
  "author": "goodarget (c)",
  "license": "ISC",
  "dependencies": {
    "http": "^0.0.1-security"
  }
}
 1Help  2Save  3Mark  4Re~ac 5Copy  6Move  7Se~ch 8De~te 9Pu~Dn
```

В каталоге нашего проекта появилась папка с описанием библиотеки http:

В нашем каталоге ./app, с приложением server27.js, появилась папка node_modules и файл package-lock.json.

Файл package-lock.json для Node.js – это файл блокировки различий в версиях библиотек, содержащий информацию о зависимостях сервлета от версий библиотечного пакета, с их точными номерами версий, которые были установлены для проекта сервлена на Node.js. Он помогает членам команды, работающим над одним и тем же репозиторием исходного кода сервлета, инсталлировать именно те версии пакетов, которые были установлены ранее, даже если для пакетов были выпущены их новые версии и установлены вместе с ядром Node.js. Это обеспечивает одинаковое дерево среды запуска node_modules на разных компьютерах/ОС-средах исполнения.

Файл package-lock.json создаётся автоматически при установке Node.js-пакетов.

Сервлет server27s.js на https.

Создадим и протестируем сервлет с доступом из Интернета по SSL-сертификату. Необходимо дать доступ пользователю user27 возможность читать сертификаты privkey.pem и fullchain.pem в папке /etc/letsencrypt/live/bot27.daibot.fun/ используя переброску https-потока на порт 443 в веб-сервер nginx.

Если у пользователя в Linux нет права на чтение папки, а есть право на чтение файла, напрямую просмотреть файл он не сможет. Не обрабатывая рекурсивно файлы в папках live и archive, но к вложенным поддиректориям дать доступ только соответствующим пользователям. Сделаем владельцем вложенных папок, но разрешить только чтение файлов:

```
#!/bin/bash
chmod go+r+x /etc/letsencrypt/live
chmod go+r+x /etc/letsencrypt/archive
for (( i=1; i <= 9; i++ ))
do
chown user0$i /etc/letsencrypt/archive/bot$i.daibot.fun
chmod u+rx,o-rx /etc/letsencrypt/archive/bot$i.daibot.fun
chown user0$i /etc/letsencrypt/live/bot$i.daibot.fun
chmod u+rx,o-rx /etc/letsencrypt/live/bot$i.daibot.fun
chown user0$i /etc/letsencrypt/archive/bot$i.daibot.fun/privkey1.pem
```

```
chown user0$i /etc/letsencrypt/archive/bot$i.daibot.fun/fullchain1.pem
chown -h user0$i /etc/letsencrypt/live/bot$i.daibot.fun/privkey.pem
chown -h user0$i /etc/letsencrypt/live/bot$i.daibot.fun/fullchain.pem
done
for (( i=10; i <= 12; i++ ))
do
chown user$i /etc/letsencrypt/archive/bot$i.daibot.fun
chmod u+rx,o-rx /etc/letsencrypt/archive/bot$i.daibot.fun
chown user$i /etc/letsencrypt/live/bot$i.daibot.fun
chmod u+rx,o-rx /etc/letsencrypt/live/bot$i.daibot.fun
chown user$i /etc/letsencrypt/archive/bot$i.daibot.fun/privkey1.pem
chown user$i /etc/letsencrypt/archive/bot$i.daibot.fun/fullchain1.pem
chown -h user$i /etc/letsencrypt/live/bot$i.daibot.fun/privkey.pem
chown -h user$i /etc/letsencrypt/live/bot$i.daibot.fun/fullchain.pem
done
for (( i=21; i <= 38; i++ ))
do
chown user$i /etc/letsencrypt/archive/bot$i.daibot.fun
chmod u+rx,o-rx /etc/letsencrypt/archive/bot$i.daibot.fun
chown user$i /etc/letsencrypt/live/bot$i.daibot.fun
chmod u+rx,o-rx /etc/letsencrypt/live/bot$i.daibot.fun
chown user$i /etc/letsencrypt/archive/bot$i.daibot.fun/privkey1.pem
chown user$i /etc/letsencrypt/archive/bot$i.daibot.fun/fullchain1.pem
chown -h user$i /etc/letsencrypt/live/bot$i.daibot.fun/privkey.pem
chown -h user$i /etc/letsencrypt/live/bot$i.daibot.fun/fullchain.pem
done
```

Теперь сервлет сможет считать сертификаты. Создадим файл server27s.js:

```
const https = require('https');
const fs = require('fs');

const hostname = 'localhost';
const port = 3027;

var options = {
  key: fs.readFileSync('/etc/letsencrypt/live/bot27.daibot.fun/privkey.pe
m'),
  cert: fs.readFileSync('/etc/letsencrypt/live/bot27.daibot.fun/fullchain
.pem')
};

const server = https.createServer(options,(req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World from user27!\n');
});

server.listen(port, hostname, () => {
  console.log(`Server running at https://${hostname}:${port}/`);
});
```

Создадим пакет и добавим зависимости:

```
$ npm init
$ npm install fs https --save
```

Заменим поле "test" на строчку "start" : "node server27s.js" в файле package.json. Запустим сервлет:

```
$ npm start
```

Изменим одну букву "**s**", http на https, в настройке конфигурации Nginx для поддомена пользователя user27 в файле "/etc/nginx/sites-available/bot27.daibot.fun":

```
server {
        listen 80;
        listen [::]:80;
        listen 443 ssl;
        listen [::]:443 ssl;

        ssl_certificate /etc/letsencrypt/live/bot27.daibot.fun/fullchain.pem;
        ssl_certificate_key /etc/letsencrypt/live/bot27.daibot.fun/privkey.pem;

        root /var/www/bot27.daibot.fun;
        index index.html index.htm index.nginx-debian.html;

        server_name bot27.daibot.fun www.bot27.daibot.fun;

        location / {
                try_files $uri $uri/ =404;
        }
        location /app {
                proxy_pass https://localhost:3027;
                proxy_set_header Host $host;
        }
}
```

И протестируем доступ к сайту в браузере:

`https://bot27.daibot.fun/app`

Общие сведения о Телеграм-боте

Телеграм использует собственный протокол шифрования MTProto. MTProto API (он же Telegram API) – это API, через который ваше приложение Телеграм связывается с сервером. Telegram API полностью открыт, так что любой разработчик может написать свой клиент мессенджера. Но, для написания ботов был создан Telegram Bot API – надстройка над Telegram API.

Чтобы использовать Bot API, вам не нужно ничего знать о том, как работает протокол шифрования MTProto — наш вспомогательный сервер будет сам

обрабатывать все шифрование и связь с Telegram API. Вы соединяетесь с сервером через простой HTTPS-интерфейс, который предоставляет простую версию Telegram API.

Бот должен уметь отправлять запросы Телеграм-серверу и получать от него апдейты (updates, обновления).

Получать апдейты можно одним из двух способов:

1) поллинг – просто регулярно отправлять запрос к серверу Телеграма для получения обновлений;

2) вебхук – сделать так, чтобы Телеграмм сам отправлял запросы по нужному URL.

Апдейты (они же updates, то есть обновления) – это события, о которых ТГ-сервер уведомляет бота (сервлет). Это может быть апдейт о входящем сообщении, апдейт о вступлении участника в группу и так далее. Главная сложность разработки ботов: апдейты – это почти единственный способ для вашей программы (бот-сервлета) узнать что-то о чатах и сообщениях. Например, ваша программа (бот) не может спросить у Телеграма, каким было последнее сообщение от пользователя или в каких чатах состоит бот. Телеграм даёт такую информацию только вместе с апдейтами: например, когда пользователь присылает сообщение или бота добавляют в группу.

Если пользователь отправил боту сообщение, и вам нужно будет использовать текст сообщения позже, сохраните его. Если вам понадобится иметь список пользователей бота, полученных сообщений и так далее — вам следует хранить эти данные. Скорее всего, для этого будет нужна база данных.

Если вы потеряете эту информацию, больше вы никак её не получите.

Некоторую информацию всё-таки можно запросить у ТГ-сервера, используя Telegram API: например, так можно получить сообщение по ID или полную информацию о пользователе.

Разработчики понимают разницу между Telegram API и Bot API. В бот-сервлете вы можете узнать что-то у сервера Телеграм, вызвав его Telegram API. А вот автоматическое взаимодействие сервера Телеграм и бот-сервлета на VPS происходит по протоколу Bot API.

При разработке Telegram-ботов с помощью Node.js есть два основных способа получения сообщений от пользователей: Polling и Webhook [23]. Оба метода служат для обработки входящих запросов, но имеют свои особенности, которые делают их подходящими для разных сценариев.

Что такое Polling?

Polling – это метод получения обновлений от сервера Telegram путем периодического опроса ботом сервера Телеграм. Бот отправляет запрос на сервер через определенные интервалы времени для проверки наличия новых сообщений или событий.

Polling бывает двух видов: Long Polling и Short Polling.

Long Polling – это подход, при котором бот отправляет запрос к серверу и ждет ответа. Если новых сообщений нет, сервер будет удерживать запрос открытым до тех пор, пока не появится новое сообщение или не истечет таймаут. После получения ответа, бот сразу отправляет новый запрос.

В данном примере бот настроен на опрос сервера Telegram с интервалом в 3 секунды. Таймаут составляет 10 секунд, что позволяет поддерживать открытое соединение в течение этого времени, прежде чем отправить новый запрос.

```
// Создаем экземпляр бота с включенным Long Polling
const bot = new TelegramBot(token, {
    polling: {
        interval: 3000,   // Интервал между запросами (3 секунды)
        autoStart: true,  // Автоматический старт polling
        params: {
            timeout: 10   // Тайм-аут запроса (10 секунд)
        }
    }       });
```

Short Polling – это подход, при котором бот отправляет запросы на сервер с короткими интервалами, независимо от наличия новых сообщений. Это менее эффективно, так как создается больше сетевых запросов и требует больше ресурсов.

Здесь бот постоянно запрашивает обновления от сервера, не удерживая соединение открытым на длительное время. Это может привести к большим затратам на сетевые ресурсы, особенно при высоком трафике.

```
// Создаем экземпляр бота с включенным Short Polling
const bot = new TelegramBot(token, { polling: true });
```

В учебных проектах категорически запрещается использования Polling.

Что такое Webhook?

Webhook – это метод, позволяющий боту получать обновления автоматически. Вместо периодического опроса сервера, бот указывает серверу Telegram URL, на который будут отправляться POST-запросы при появлении новых обновлений. Это позволяет более эффективно использовать ресурсы и минимизировать задержки.

В этом примере бот bot27.js автоматически создаёт webhook и принимает запросы от Telegram через webhook, что позволяет избежать частого опроса сервера. Это снижает нагрузку на сервер и обеспечивает мгновенную обработку новых сообщений.

```
const TelegramBot = require('node-telegram-bot-api');
const express = require('express');
const https = require('https');
const bodyParser=require('body-parser');
const fs = require('fs');
const token = '7645673456:yyyyyDi7BXLOVViALgCyyyyyyooDPcKyyyy';
const url = 'https://bot27.daibot.fun';
const hostname = 'localhost';
const port = 3027;
var options = {
  key: fs.readFileSync('/etc/letsencrypt/live/bot27.daibot.fun/privkey.pe
m'),
  cert: fs.readFileSync('/etc/letsencrypt/live/bot27.daibot.fun/fullchain
.pem')
};
```

```
// Создаем экземпляр бота без автоматического polling
const bot = new TelegramBot(token, { webHook: true });

// Устанавливаем Webhook на URL вашего сервера
bot.setWebHook(`${url}/tgbot`);

const app = express();

app.use(bodyParser.json());

// Обработчик запросов от Telegram
app.post('/tgbot', (req, res) => {
    console.log(req.body);
    bot.processUpdate(req.body);
    res.sendStatus(200);
});

bot.on('message', (msg) => {
    const chatId = msg.chat.id;
    bot.sendMessage(chatId, `Вы написали: ${msg.text}`);
});
var httpsServer = https.createServer(options, app);
httpsServer.listen(port, hostname, () => {
    console.log(`Сервер запущен на сервере ${hostname} на порту ${port}`)
;
});
```

Для работы кода и запуска бота необходимо установить библиотеки с помощью команд:

```
$ npm install node-telegram-bot-api  # Для работы с API Telegram
$ npm install express  # Для обработки HTTP-запросов
$ npm i https          # Сервер https
$ npm i body-parser    # Преобразуйте тело запроса в JSON
$ npm i fs    # Работа с файлами SSL-ключей
```

Можно установить одной командой и сохранить в репозитарии:

```
$ npm install node-telegram-bot-api express https body-parser fs --save
```

Проверим синтаксис программы:

```
$ node --check bot27.js
```

Настроим Nginx

В настройке конфигурации Nginx для поддомена пользователя user27 в файле "/etc/nginx/sites-available/bot27.daibot.fun" добавим еще одну секцию location:

```
        location /tgbot {
                proxy_set_header Host $host;
                proxy_pass https://localhost:3027/tgbot;
        }
```

И перезапустим nginx.

Создадим ТГ-бота.

Для получения токена открыли в Телеграме специального бота-регистратора BotFather и запустили его с помощью команды /start, потом выполнили все необходимые действия и плучили токен token, в примере просто набор символов: 7638989013:BBEl7PN4Ts7ZF3VLbgVdEm9I-bj50yc87Aq

Список вводимых данных в папу-боте BotFather минимальный:

```
/start
/newbot
Name: user27bot
UserName: user27_daibot
```

Вставим токен token в файл bot27.js и запустим его npm start.

Для запуска бота наберите первоначально:
t.me/user27_daibot

После запуска через утилиту pm2 двух серверов слушающих один порт 3027, второе приложение перестало работать, что видно на экранной форме:

Выводы.

Я сделал надежного Телеграм-бота, работающего по запросу от серверов ТГ через webhook. Но не придерживался документации в названии персонального REST API назвав его просто "tgbot". Это позволит мне исправить все конфигурационные файлы nginx для каждого из поддоменов однообразно. Поменять лишь порт.

```
location /tgbot {
                proxy_set_header Host $host;
                proxy_pass https://localhost:3027/tgbot;
}
```

Для location "/app" я везде исправлю на "https". Поэтому Ваши ранние приложения перестанут работать.

В проектах ТГ-бота категорически запрещается использования Polling.

Дополнительный материал и ссылки

TCP/IP

Difference Between 127.0.0.1 and 0.0.0.0 https://www.geeksforgeeks.org/difference-between-127-0-0-1-and-0-0-0-0/

DNS

Что такое DNS? Объясняем простыми словами https://www.nic.ru/info/blog/dns/
CHECK DNS PROPAGATION https://dnschecker.org/#A/daibot.fun

SSL

Let's Encrypt и nginx: настройка в Debian и Ubuntu https://habr.com/ru/articles/318952/
Получение бесплатного SSL сертификата Let's Encrypt https://www.dmosk.ru/miniinstruktions.php?mini=get-letsencrypt
Правильная настройка SSL в NGINX https://www.dmosk.ru/miniinstruktions.php?mini=nginx-ssl&ysclid=m2s0nnrs66657192176
Как настроить SSL-сертификат на Nginx https://timeweb.com/ru/community/articles/kak-nastroit-ssl-sertifikat-na-nginx
Самоподписной SSL:
Размещение телеграмм бота (Aiogram3) на VPS/VDS https://habr.com/ru/articles/783398/

NGINX

Введение в NGINX: как его установить и настроить https://proglib.io/p/nginx
Настройка NGINX + NodeJS на Ubuntu Server https://www.dmosk.ru/miniinstruktions.php?mini=nginx-nodejs-ubuntu
Создание виртуальных хостов на Nginx https://jino.ru/spravka/articles/nginx_server_blocks.html
How To Serve NGINX Subdomains or Multiple Domains https://adamtheautomator.com/nginx-subdomain/

NODE.JS

Как запустить собственное приложение Node.js на VPS сервере(Ubuntu)?
https://vc.ru/dev/1403898-kak-zapustit-sobstvennoe-prilozhenie-nodejs-na-vps-servereubuntu
nvm: UBUNTU – INSTALL NODE VER-SION MAN-AGER GLOBALLY FOR ALL USERS
https://eureka.ykyuen.info/2014/05/07/ubuntu-install-node-ver%C2%ADsion-man%C2%ADager-globally-for-all-users/
Свой веб-сервер на NodeJS, и ни единого фреймворка. Часть 1
https://habr.com/ru/articles/327440/

TG-Bot

Схема вебхука: Размещение телеграмм бота (Aiogram3) на VPS/VDS https://habr.com/ru/articles/783398/
Телега для датасайентиста https://habr.com/ru/companies/ods/articles/462141/
Всё, о чём должен знать разработчик Телеграм-ботов https://habr.com/ru/articles/543676/
Запуск бота как службу на RUVDS. Бот на Питоне:
Будни техпода. Как разместить Telegram-бота на виртуальном сервере
https://habr.com/ru/companies/ruvds/articles/786014/

API

REST vs. GraphQL vs. gRPC – Which API to Choose? https://www.baeldung.com/rest-vs-graphql-vs-grpc

Учебный проект "Daibot.fun". Часть - 6.

"Идеальный" корпаративный бот приведен в лекции.

Задание проекта: Создать мини-приложение Телеграм бот генерации визуального улучшения плакатов спецэффектами SVG-фильртов, на базе виртуального выделенного сервера с доменным именем daibot.fun.

Базовый алгоритм ИИ: написан на языке программирования JavaScript.

Постановка задачи

```
План работ:
- сделать телеграм-бот, который получает и отправляет растровое изображен
ие.
```

Основы проектов на Linux

Часто встречающийся термин в репозитариях исходного кода:
CLI – command-line interface, интерфейс командной строки;
.yaml или **.yml** – YAML (YAML Ain't Markup Language или Yet Another Markup Language) – человекопонятный язык для хранения структурированной информации. Он для форматирования иерархии сериализуемых данных учитывает отступы в виде пробелов, наподобие языка Питон. В отличие от JSON в нём можно создавать переменные и ссылаться на них. Используется для написания конфигов, чтобы стартовать проект, проверять тесты и для линтеров. Линтер – это инструмент для анализа исходного кода с целью выявления проблем, структурных ошибок, стилевых нарушений и других недочетов.
Преобразуем наш package.json в файл конфигурации проекта package.yaml [24]:

```
# npm install -g js-yaml
```

`// --------- json-to-yaml ---------- -`
`//Load the package`
`const yaml = require('js-yaml');`
`const fs = require('fs');`
`//Read the JSON file`
`const data = fs.readFileSync('./input_file.json', 'utf8');`
`const jsonData = JSON.parse(data);`
`//Convert JSON to Yaml`
`const yamlData = yaml.dump(jsonData);`
`//Save to file`
`fs.writeFileSync('./file.yml', yamlData, 'utf8');`
	`// --------- yaml-to-json ----------`
	`//Load the package`
	`const yaml = require('js-yaml');`
	`const fs = require('fs');`
	`//Read the Yaml file`
	`const data = fs.readFileSync('./input_file.yml', 'utf8');`
	`const yamlData = yaml.load(data);`
	`//Convert Yml object to JSON`
	`const jsonData = JSON.stringify(yamlData);`
	`//Save to file`
	`fs.writeFileSync('./file.json', jsonData, 'utf8');`

Результат преобразования:

JSON	YAML

`{`
`  "name": "app",`
`  "version": "1.0.0",`
`  "description": "HTTPS servlet",`
`  "main": "server27s.js",`
`  "scripts": {`
`  "start": "node server27s.js"`
`  },`
`  "author": "goodarget (c)",`
`  "license": "ISC",`
`  "dependencies": {`
`  "fs": "^0.0.1-security",`
`  "https": "^1.0.0"`
`  }`
`}`
	`name: app`
	`version: 1.0.0`
	`description: HTTPS servlet`
	`main: server27s.js`
	`scripts:`
	`  start: node server27s.js`
	`author: goodarget (c)`
	`license: ISC`
	`dependencies:`
	`  fs: ^0.0.1-security`
	`  https: ^1.0.0`

Добавим списки и переменные. Якорь – это переменная или ссылка. Якорь – это механизм для создания переменных, на которые затем можно ссылаться. Якорь решает эту проблему. Для его создания используется символ якоря (&), а для вставки – алиас (*).

YAML	JSON

`base-config: &var1`
`  main: server27s.js`
`  start: "node server27s.js"`
`commands:`
`  - test`
`  - start`
`  - run`
`env-testing:`
`  <<: *var1`
`  host: 45.8.230.131`
`  port: 3027`
`env-deploy:`
`  <<: *var1`
`  host: bot27.daibot.fun`
`  port: 80`
	`{`
	`  "base-config":{`
	`  "main":"server27s.js",`
	`  "start":"node server27s.js"`
	`  },`
	`  "commands":["test","start","run"],`
	`  "env-testing":{`
	`  "main":"server27s.js",`
	`  "start":"node server27s.js",`
	`  "host":"45.8.230.131",`
	`  "port":3027`
	`  },`
	`  "env-deploy":{`
	`  "main":"server27s.js",`
	`  "start":"node server27s.js",`
	`  "host":"bot27.daibot.fun",`
	`  "port":80`
	`  }`
	`}`

Символ оператора "<<:" называется ключ слияния, который делает вставку. Если написать строку node *var2, то она останется текстом "node *var2". Только если переменная будет стоять вначале строки без последующего текста.

Автоматизация запуска сервлета с помощью менеджера pm2 позволяет передать данные из конфигурационного файла ecosystem, в зависимости от сценария использования.

Допустим у нас два сценария: запуск на локальное машине для тестирования (developer) и непосредственное развертывание при внедрении (deploy). Можно базовые параметры сервлета <доменное имя бота; ip-адрес сервлета, на котором он слушает сеть; и его сетевой порт> поместить в конфигурационный файл ecosystem. Файл ecosystem содержит два сценария и они подключаются менеджером pm2 по опциональному флагу.

```
$ pm2 start ecosystem --developer   # Запуск сервлета по первому сценарию
$ pm2 start ecosystem --deploy   # Запуск сервлета по второму сценарию
```

Получается такая тройная вложенность запуска сервлета на выполнение. На верхнем уровне опциональным ключом --developer указываем сценарий запуска сервлета. Промежуточный уровень описан в файле конфигурации ecosystem. Файл ecosystem, в нашем случае, разделен на две секции с параметрами сценария для тестирования и внедрения. Параметры сценария явно описываю домен, ip-адрес, токен бота, порт. Менеджеж pm2 выберет, по опциональному ключу, по какому сценарию грузить сервлет в ядро node.js. Менеджер pm2 создасть в памяти ядра node.js глобальные переменные (порт, домен, токен, адрес) соответствующие только сценарию в файле конфигурации ecosystem. Файл ecosystem указан без расширения, потому что его можно задавать трех видов: js, json, yaml. Третий, внутренний уровень находится в исходном коде сервлета. Там необходимо запрограммировать обращение к классу process, среды исполнения node.js, для считывания параметров конфигурации env. В класс process менеджер pm2 загрузил параметры (порт, домен, токен, адрес) только выбранного сценария.

То есть простая строчка в файле bot.js, по объявлению константы "const port=3027;" превращается в набор служебных команд. Но это позволяет создавать масштабируемое программное решение и относится к разделу профессионального ведения IT-проекта.

Начнём с конца. Подключаем библиотеку с загрузкой переменных окружения:

```
$ npm install dotenv --save
```

В начале файла приложения app.js экспортируем библиотеку в исходный код:

```
require('dotenv').config();
```

Теперь можно считать значение перемененной из процесса конфигурации:

```
const port = process.env.PORT;
```

Создадим тестовое приложение app.js:

```
require('dotenv').config();
const port = process.env.PORT;
const host = process.env.HOST;
console.log('App read from  config PORT=',port);
console.log('App read from config HOST=',host);
```

Теперь сохраним в файл ecosystem.config.json настройки приложения:

```
{
  "apps": [
    {
      "name"           : "App load config test",
      "script"         : "./app.js",
      "exec_mode"      : "fork",
      "instances"      : "max",
      "env": {
        "PORT"       : "3027"
      },
      "env_testing": {
        "NODE_ENV"     : "developer",
        "HOST"         : "45.8.230.131"
      },
      "env_production" : {
        "NODE_ENV"     : "deploy",
        "HOST"         : "localhost"
      }
    }
  ]
}
```

Для проекта app.js создадим также и файл package.json:

```
{
  "name": "AppENV",
  "version": "1.0.0",
  "main": "app.js",
  "scripts": {
    "start": "node app.js"
  },
  "author": "goodarget (c)",
  "license": "ISC",
  "description": "Test app load config",
  "dependencies": {
    "dotenv": "^16.4.5"
  }
}
```

Проверим все варианты запуска проекта. Простой `node app.js`:

```
mc [user27@ruvds-d0ohk]:~/prg2
user27@ruvds-d0ohk:~/prg2$ node app.js
App read from  config PORT= undefined
App read from config HOST= undefined
user27@ruvds-d0ohk:~/prg2$
```

Переменные среды не объявлены. Поэтому при загрузке необходимо подключать их по умолчанию. Проверим запуск с помощью `npm start`:

```
mc [user27@ruvds-d0ohk]:~/prg2
user27@ruvds-d0ohk:~/prg2$ npm start

> AppENV@1.0.0 start
> node app.js

App read from  config PORT= undefined
App read from config HOST= undefined
user27@ruvds-d0ohk:~/prg2$
```

Проверим запуск с помощью менеджера `pm2 start app.js`:

```
mc [user27@ruvds-d0ohk]:~/prg2
user27@ruvds-d0ohk:~/prg2$ pm2 start app.js
[PM2] Starting /home/user27/prg2/app.js in fork_mode (1 instance)
[PM2] Done.
```

id	name	namespace	version	mode	pid	uptime	↺	status	cpu	mem	user	watching
0	app	default	1.0.0	fork	2091663	0s	0	online	0%	29.9mb	user27	disabled

```
user27@ruvds-d0ohk:~/prg2$
```

Как увидит результат работы приложения? Для этого необходимо запустить монитор менеджера `pm2 show 0` и в правой панели мы увидим результаты:

```
PM2 Dashboard
Process List                    app Logs
[ 0] app        Mem:  51 MB     app > App read from  config PORT= undefined
                                app > App read from config HOST= undefined
                                app > App read from  config PORT= undefined
                                app > App read from config HOST= undefined
                                app > App read from  config PORT= undefined
                                app > App read from config HOST= undefined
                                app > App read from  config PORT= undefined
                                app > App read from config HOST= undefined
                                app > App read from  config PORT= undefined
                                app > App read from config HOST= undefined
                                app > App read from  config PORT= undefined
                                app > App read from config HOST= undefined
                                app > App read from  config PORT= undefined
                                app > App read from config HOST= undefined
                                app > App read from  config PORT= undefined
                                app > App read from config HOST= undefined
Custom Metrics                  Metadata
                                App Name        app
                                Namespace       default
                                Version         1.0.0
                                Restarts        99
                                Uptime          1s
 left/right: switch boards | up/down/mouse: scroll | Ctrl-C: exit     To go further check out https://pm2.io/
```

Видим, что приложение многократно запускается. Поэтому мы видим повторение двух строчек. Но переменные не загружены в программу.
Наконец запустим менеджер с указанием только конфигурационного файла `pm2 start ecosystem.config.json`:

```
PM2 Dashboard
user27@ruvds-d0ohk:~/prg2$ pm2 start ecosystem.config.json
[PM2][WARN] Applications App load config test not running, starting...
[PM2] App [App load config test] launched (1 instances)

| id | name | mode | □ | status | cpu | memory |
|---|---|---|---|---|---|---|
| 0 | App load config t… | fork | 0 | online | 0% | 26.4mb |

user27@ruvds-d0ohk:~/prg2$
```

В мониторе процесса мы видим ожидаемый результат для ветки ключей "env". Наконец переменная port получила своё значение 3027, а вот переменная host осталась не инициализированной.

```
PM2 Dashboard
Process List                App load config test Logs
[ 0] App load config        App load config test > App read from  config PORT= 3027
                            App load config test > App read from config HOST= undefined

Custom Metrics              Metadata
Heap Size        10.11      App Name          App load config test
Heap Usage       71.41 %    Namespace         default

left/right: switch boards | up/down/mouse: scroll | Ctrl-C: exit      To go further check
```

Запустим с флагом тестирования приложения при его разработке:
`pm2 start ecosystem.config.json --env testing`

```
PM2 Dashboard
Process List                App load config test Logs
[ 0] App load config test   App load config test > App read from  config PORT= 3027
                            App load config test > App read from config HOST= 45.8.230.131

Custom Metrics              Metadata
                            App Name          App load config test
                            Namespace         default

left/right: switch boards | up/down/mouse: scroll | Ctrl-C: exit      To go further check out
```

Мы чётко видим, что общая переменная port была инициализирована, а переменной host приняло значение ip-адреса сервера. Также понятно, что имя ключа в файле конфигурации должно быть "`env_testing`" и начинается с префикса "`env_`", а ключ NODE_ENV является специфическим.
Поле NODE_ENV в файле ecosystem.config.js для PM2 указывает ядру node.js типовой режим работы <developer, production>. Например, при использовании библиотеки сервлетов Express установка NODE_ENV в значение «production» может улучшить производительность в 3 раза.
Запустим приложение с флагом внедрения:

```
pm2 start ecosystem.config.json --env production
```

Видим, что конфигурация загружена для безопасного использования локальной петли localhost. Переменная "NODE_ENV" : "deploy" была объявлена не правильно, так как "deploy" не задан в спецификации node.js. Параметр NODE_ENV может принимать только значение usually, staging, test, development или production.
Существует еще один, не учтенный, способ задать переменные окружения. Занести их в виде списка в безымянный файл с расширениев ".env" и разместить в папке проекта:

```
PORT=3027
HOST=localhost
```

И запускать программу, как проект npm или node.

Конвертируем json-файл конфигурации в yaml-файл:

```
apps:
  - name: App load config test
    script: ./app.js
    exec_mode: fork
    instances: max
    env:
      PORT: '3027'
    env_testing:
      NODE_ENV: developer
      HOST: 45.8.230.131
    env_production:
      NODE_ENV: deploy
      HOST: localhost
```

Запустим `pm2 start ecosystem.config.yml --env production` и увидим результат:

Если в файл `ecosystem.config.json` добавить начальную строчку "module.exports =" и изменить расширение файла на ".js":

То запуск сервлета будет:

```
pm2 start ecosystem.config.js --env testing
```

Получим ожидаемый результат:

Таким образом, мы имеет три типа файлов конфигурации для менеджера pm2 которые можно встретить в репозитариях исходного кода серверного ПО. А для простых приложений подойдет и файл ".env".

Если в файле конфигурации для pm2 указать путь к log-файлам протоколирования работы приложения и err-файлам возникающих ошибок, то это позволить отлаживать программы в режиме разработки. Достаточно указать:

```
"error_file"     : "./err.txt",
"log_file"       : "./log.txt",
```

В ходе запуска pm2 или работы приложения файлы будут созданы автоматически менеджером pm2. И в папке проекта появятся файлы err.txt и log.txt, с сохраненными результатами. Монитор pm2 тоже покажет log-сообщения.

Бот и анализ подключения через webhook

Проверить статус webhook’а можно таким запросом:

```
https://api.telegram.org/botYOUR-TOKEN/getWebhookInfo
```

Если всё нормально, должно вернуть такое:

```
{
    "ok": true,
    "result": {
        "url": "https://YOUR.DOMAIN:8443/YOUR-TOKEN/",
        "has_custom_certificate": false,
        "pending_update_count": 0,
        "max_connections": 40
    }
}
```

Как видим, в поле url стоит наша конечная точка назначения REST API. Описание полей:
"can_join_groups": true – говорит о том, что вашего бота можно добавлять в группы.
"can_read_all_group_messages": false – говорит о том, что ваш бот не может читать сообщения пользователей в группе. Вы можете изменить это значение на True и тогда бот сможет читать, что пишут другие пользователи.
"supports_inline_queries": false – говорит о том, что вашему боту нельзя отправлять команды, используя символ @. Вы можете изменить это значение на True и тогда пользователи смогу это делать.
Самый простой сценарий использования бота, который не требует никакого программирования и специальных программ – это отправка сообщений от имени бота. Например, вы можете добавить бота в группу и используя Telegram Bot API отправлять от его имени сообщения в эту группу.

```
https://api.telegram.org/bot0123456789:AAFYNwvkDdfgHJ3jhgrs3cUH4oSOEAVSZp8/sendMessage?chat_id=0123456789&text=Hello World
```

Обновление SSL-сертификатов

Выполните пробное обновление индивидуального сертификата с измененными параметрами. Например, если у вас есть один сертификат, полученный с помощью автономного плагина, вам может потребоваться остановить веб-сервер перед обновлением, чтобы автономный плагин мог привязаться к необходимым портам:

```
certbot renew --dry-run --pre-hook "service nginx stop" --post-hook "service nginx start"
```

Изучаем терминологию Linux
Часто встречающийся термин в репозитариях исходного кода:

endpoint – конечная часть имени в URL, без учета доменного имени, для идентификации вызываемого REST API;

MJS – файлы с расширением .mjs связаны с системой модулей ECMAScript Modules ES6, в основе которой лежат директивы import и export. А файлы с расширением .js относятся к системе модулей языка CommonJS, которая применяется в Node.js и предполагает использование require и module.exports. Это различие между модульной системой ES6 и CommonJS;

Yarn – это менеджер пакетов JavaScript, совместимый с npm, который помогает автоматизировать процесс установки, обновления, настройки и удаления пакетов npm. Он кэширует каждый загружаемый пакет и ускоряет процесс установки за счет распараллеливания операций;
Sharp – библиотека обработки изображений [25];

CURL – кроссплатформенная служебная программа командной строки, позволяющая взаимодействовать с множеством различных серверов по множеству различных протоколов с синтаксисом URL. Программа поддерживает

протоколы: FTP, FTPS, HTTP, HTTPS, TFTP, SCP, SFTP, Telnet, DICT, LDAP, а также POP3, IMAP и SMTP.

Возьмем за аналог типовой проект бота на github, использующий графическую библиотеку для обработки изображения [26]. Анализ показывает небрежное формирование файла проекта package.json: в нем упоминается index.js, хотя файл с ботом называется bot.js. А графическая библиотека jimp написана на TypeScript и будет заменена на sharp для совместимости с TG Bot API.
В этом боте реализован классический интерфейс общения ТГ-бота с клиентом. Вы должны давать боту команды через слеш, то есть это аналог CLI-интерфейса для любой операционной системы. Оставим в боте одну команду "создать ч/б изображение" для манипулирования изображением /bw и сделаем его "истинно" сетевым сервлетом с webhook. Схема бизнес-процесса бота: клиент отправляет боту фотографию; бот сохраняет изображение с id-номером телеграма клиента на диске и запоминает путь к файлу в массиве ключей; таким образом бот может обрабатывать изображение полученное от клиента хоть через час, если конечно он не перегрузился и не потерял таблицу путей к изображению; клиент даёт команду на обработку изображения и бот обрабатываете изображение и перезаписывает файл на диск; окончательно клиент должен дать команду /print для получения изображения от бота. Отвратительная UX-модель диалогового интерфейса с пользователем.
Установим библиотеки:

```
$ npm install node-telegram-bot-api express https body-parser fs sharp path dotenv --save
```

Получим файл package.js:

```
{
  "name": "TGbotSharp",
  "version": "1.0.0",
  "description": "TG-bot Sharp user27",
  "main": "botsharp.js",
  "scripts": {
    "start": "node botsharp.js"
  },
  "author": "goodarget (c)",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.20.3",
    "dotenv": "^16.4.5",
    "express": "^4.21.1",
    "fs": "^0.0.1-security",
    "https": "^1.0.0",
    "sharp": "^0.33.5",
    "node-telegram-bot-api": "^0.66.0",
    "path": "^0.12.7"
  }
}
```

Создадим файл .env:

```
TG_TOKEN=7645673456:yyyyyDi7BXLOVViALgCyyyyyyooDPcKyyyy
URL=https://bot27.daibot.fun
HOSTNAME=localhost
PORT=3027
```

Сделаем файл сервлета botsharp.js и проверим его $ **node** `--check botsharp.js`:

```
const TelegramBot = require('node-telegram-bot-api');
const express = require('express');
const https = require('https');
const bodyParser=require('body-parser');
const fs = require('fs');
const fsp = require('fs').promises;
require('dotenv').config();
const sharp  = require('sharp');
const path=require('path');

const token = process.env.TG_TOKEN;
const url = process.env.URL;
const hostname = process.env.HOSTNAME;
const port = process.env.PORT;

var options = {
  key: fs.readFileSync('/etc/letsencrypt/live/bot27.daibot.fun/privkey.pem'),
  cert: fs.readFileSync('/etc/letsencrypt/live/bot27.daibot.fun/fullchain.pem')
};

const bot = new TelegramBot(token, { webHook: true });
bot.setWebHook(`${url}/tgbot`);

const app = express();

app.use(bodyParser.json());

// Обработчик запросов от Telegram
app.post("/tgbot", (req, res) => {
//    console.log(req.body);
    bot.processUpdate(req.body);
    res.sendStatus(200);
});

const imageProcessingState = {};

bot.onText(/\/start/, (msg) => {
  bot.sendMessage(msg.chat.id, 'Bot is online! Send an image then use commands /bw to
manipulate the image! Use /print to recieve your edited image');
});

bot.on('photo', async (msg) => {
  const chatId = msg.chat.id;
  const userId = msg.from.id;
  const photo = msg.photo[msg.photo.length - 1].file_id;

  try {
    const filePath = await bot.downloadFile(photo, ".");

    imageProcessingState[userId] = {
      filePath,
      modifications: [],
    };
    bot.sendMessage(chatId, 'image downloaded successfully.');
  } catch (error) {
    console.error("Error downloading image", error);
    bot.sendMessage(chatId, 'Error downloading image. Please try again.');
  }
});
```

```
bot.onText(/\/bw/, async (msg) => {
  const chatId = msg.chat.id;
  const userId = msg.from.id;

  if (imageProcessingState[userId]) {
    try {
      const currentImagePath = imageProcessingState[userId].filePath;
      const    outputImagePath    =    path.dirname(currentImagePath)    +    '/out'    +
path.basename(currentImagePath);
      sharp(currentImagePath).greyscale().threshold(100).toFile(outputImagePath,
function(err) {
        if (err) {
          console.error("Error save image", err);
          bot.sendMessage(chatId, 'Error save b/w image file' + currentImagePath + '.
Please try again.')
        } else {
          imageProcessingState[userId].filePath = outputImagePath;
          bot.sendMessage(chatId, 'Image has been black/white!');
        }
      });
    } catch (error) {
      console.error("Error applying /bw operation", error);
      bot.sendMessage(chatId, 'Error in process image b/w. Please try again.');
    }
  } else {
    bot.sendMessage(chatId, 'No active session. Send an image to get started.');
  }
});

bot.onText(/\/print/, async (msg) => {
  const chatId = msg.chat.id;
  const userId = msg.from.id;

  if (imageProcessingState[userId]) {
    try {
      const currentImagePath = imageProcessingState[userId].filePath;

      await bot.sendPhoto(chatId, currentImagePath);

      delete imageProcessingState[userId];
      await fsp.unlink(currentImagePath);
    } catch (error) {
      console.error("Error sending image", error);
      bot.sendMessage(chatId, 'Error sending image');
    }
  } else {
    bot.sendMessage(chatId, 'No active session');
  }
});

var httpsServer = https.createServer(options, app);

httpsServer.listen(port, hostname, () => {
    console.log(`Сервер TGbot Sharp  запущен на ${hostname} на порту ${port}`);
});
```

Запустим его $ **npm** `start`.

Предупреждение при отправке файла изменить заголовок POST на octet-stream:

`https://github.com/yagop/node-telegram-bot-api/blob/master/doc/usage.md#sending-files`

Описание обработки запросов сетевой библиотекой.

Модуль синтаксического анализа текста `body-parser` анализирует данные в формате JSON, buffer, string и URL, закодированные с помощью HTTP POST-запроса. Метод класса парсера bodyParser.json() предназначен для упрощения обработки данных в формате JSON, автоматически преобразуя входящие полезные данные JSON в объекты JavaScript, выставляемые в виде req.body-свойств функций библиотеки Express.js в приложениях Node.js. Метод используется для извлечения тела запроса и отображения его в объектной форме в память приложения. Метод `body-parser` необходимо включить в приложение с помощью метода `express()`.use() .

```
const express = require('express');
const bodyParser = require('body-parser');
const app = express();
// JSON parsing
app.use(bodyParser.json());
app.post('/your-endpoint', (req, res) => {
    console.log(req.body);
    res.send('Data received');
});
```

Метод express.use() в Node.js используется для создания промежуточных обработчиков, когда можно дополнительно обрабатывать POST-запрос протокола https в любой момент в конвейере обработки запросов сервлетом.

Следующий метод "app.post('/tgbot', (req, res)" перехватывает поступающий запрос к сервлету на обработку по конечной точке в URL-адресе вызова REST API браузером или CURL-клиентом. Затем в обработчике перехвата мы вызывает обработку данных непосредственно библиотекой Telegram Bot API и отвечаем ТГ-серверу кодом "200", то есть "Успешно", что все данные получили.

```
// Обработчик запросов от Telegram
app.post("/tgbot", async (req, res) => {
    await bot.processUpdate(req.body);
    res.sendStatus(200);                });
```

Учебный проект "Daibot.fun". Часть - 7.

Настройка и дизайн корпаративного бота.

Задание проекта: Создать мини-приложение Телеграм бота генерации визуального улучшения плакатов спецэффектами SVG-фильртов, на базе виртуального выделенного сервера с доменным именем daibot.fun.

Базовый алгоритм ИИ: написан на языке программирования JavaScript.

Постановка задачи

```
План работ:
– сделать телеграм-бота с интерфейсом GUI;
- сделать телеграм-бота с интерфейсом Mini Web App.
```

Дружественный интерфейс ТГ-бота от CLI до Mini Web App

Начнем с самого простого элемента GUI, поражающего воображение любого пользователя компьютера – меню. Она может содержать список команд ТГ-бота CLI [27]. Таким нехитрым способом, служа некоторым руководством пользователя по использованию программы. Вставим код после webhook:

```
const commands = [
    {
        command: "start",
        description: "Запуск бота"

    },
    {
        command: "bw",
        description: "Преобразовать изображение в ч/б цвет"
    },
    {
        command: "print",
        description: "Прислать мне изображение"
    }
];
bot.setMyCommands(commands);
```

Меню, содержащее все доступные команды к боту, своего рода подсказка, что может делать бот. Этот элемент, неявного описания бизнес-процесса работы в прикладном приложении, делает его в стиле UX-дизайна. Мы незаметно, не перегружая пользователя справочными материалами, обучаем работать с прикладной программой.

Приветствие бота

Давайте изменим бота из предыдущего раздела лекций и добавим ему меню. Так же будем оформлять интерфейс бот некоторыми декоративными элементами GUI-интерфейса, делающими его понятным – это стартовое приветствие с правильными словами и с кнопкой перехода в меню, которое я увидел в обучающих роликах сайтов конструкторов ТГ-ботов [28, 29].

Меня поразила очень важная фраза "Магазин выглядит как сайт. Мы вас ждём" говорящая пользователю, что ввод данных от него будет осуществляется в html-интерфейсе, как на сайте. Это обращение к уже известному опыту пользователь. Один зачётный плюс в сторону UX-методологии, смотри на рисунке а). Наличие в тексте сообщений эмодзи делает приветствие вообще суперузнаваемым в стиле мессенджеров, как для меня. А наличие сразу под блоком текста "/start" кнопки с пояснение в виде предварительного текста с эмодзи "указатель руки", что это интерактивная ссылка для перехода. Второй зачётный плюс, смотри на рисунке б).

а) Приветствие

б) Текстовое пояснение к GUI-элементу

Мессенджеры – это "винегрет" из текстовых сообщений, заключенных в красивые, цветные блоки и поди догадайся, что это интерактивный элемент. Оформление бота настраивается в BotFather: меню /mybots → Edit Bot. Там можно изменить: Описание (Description) — это текст, который пользователи будут видеть в начале диалога с ботом под заголовком "Что может делать этот бот?". Введем текст:

```
Вас приветствует Лаборатория "PixVix"!
В нашем боте Вы сможете преобразовать растровое изображение
Лаборатория выглядит как сайт
```

Форматирование текста

Для форматирования, стилизации, текста можно использовать либо HTML-верстку, либо Markdown-верстку [30]. Также если мы хотим вставить эмодзи в наше сообщение, то можно просто скопировать эмодзи из телеграма и вставить в нашу строку в коде "□".

Для HTML необходимо передать тип `parse_mode = HTML`:

```
await bot.sendMessage(msg.chat.id, `<b>Жирный Текст</b>\n<a
href='t.me'>Гиперссылка</a>`, { parse_mode: "HTML" });
<b> Текст </b> - Жирный текст
<i> Текст </i> - Текст курсивом
<code> Текст </code> - Текст, который можно скопировать нажатием на него
<s> Текст </s> - Перечеркнутый текст
<u> Текст </u> - Подчеркнутый текст
<pre language='язык'> Текст </pre> - Текст с оформлением кода
<a href='ссылка'> Текст </a> - Текст-гиперссылка
```

Также обратите внимание на то, что переход на следующую строку выполняется при помощи "\n" и в стилизации HTML, и в стилизации Markdown.

Markdown – облегчённый язык разметки, созданный с целью обозначения форматирования в простом тексте, с максимальным сохранением его читаемости человеком, и пригодный для машинного преобразования.

Для Markdown необходимо передать тип `parse_mode = MarkdownV2` :

```
await bot.sendMessage(msg.chat.id, '*Жирный Текст*\n[Гиперссылка](t.me)', {
parse_mode: "MarkdownV2" });
*Текст* – Жирный текст
_Текст_ – Текст курсивом
`Текст` – Текст, который можно скопировать нажатием на него
~Текст~ – Перечеркнутый текст
``` Текст ``` – Текст с оформлением кода
|| Текст || – Скрытый текст
[Текст](Ссылка) – Текст-гиперссылка
```

Если мы хотим убрать превью в сообщении, то нам необходимо передать в метод sendMessage объект с параметром disable_web_page_preview со значением true:

```
await bot.sendMessage(msg.chat.id, `https://habr.com/`, {disable_web_page_preview: true});
```

Кстати, в этот же объект, помимо parse_mode и disable_web_page_preview, мы можем передать параметр disable_notification – это позволит отправить сообщение пользователю без уведомления.

Создадим приветствие. Точка и восклицательный знак спецсимволы в MarkdownV2. Поэтому их необходимо экранировать двумя обратными слешами:

```
bot.onText(/\/start/, (msg) => {
  bot.sendMessage(msg.chat.id, '*Bot is online*\\!\n' +
        'Send an image then use commands /bw to manipulate the image\\.\n' +
        'Use /print to recieve your edited image\\.\n' +
        'Click button □ ', { parse_mode: "MarkdownV2" } );
});
```
```

Добавим один пункт меню под текстовый блок "/start".

```
bot.sendMessage(msg.chat.id, `.....`, {
        reply_markup: {
            inline_keyboard: [
                [{text: 'Настроить параметры фильтра', callback_data: 'runWebApp'}]
            ]
        }
})
```

Как видим, в объекте возвращается информация о сообщении, к которому привязана кнопка, на которую нажали, информация о пользователе, который нажал кнопку, и самое важное - callback_data, которую мы указывали в кнопке. С помощью вот этого параметра callback_data будем обрабатывать нажатие на кнопку:

```
bot.on('callback_query', async (ctx) => {
  try {
    switch(ctx.data) {
      case "runWebApp":
         await bot.deleteMessage(ctx.message.chat.id, ctx.message.message_id);
         bot.sendMessage(ctx.message.chat.id, 'Select test menu "Run Mini Web App".');
         break;
      }
  }
  catch(error) {
        console.log(error);
  }
})
```

В результате нажатия на пункт меню удаляется сообщение "/start" и меню. Затем выводится тестовое сообщение о успешности выбора пункта меню.

Сделаем gif-анимацию из двух изображений до и после преобразования. Размер рекомендуемый для телеграм 320 на 180, будет анимироваться на смартфоне. Большего размера холст gif-анимации замораживается Телеграм, как обычная картинка. На сайте [31] можно указать морфинг-переход между двумя кадрами, задав параметр Fader с частотой в 10 переходов прозрачности. Базовая задержка смены кадра задаётся в 1/10 секунды, то есть 350 будет соответствовать 3500 мс равное 3,5 секунды.

Создание Web App без Angular

Mini App или Web App – это относительно новый и удобный способ добавления веб-приложения прямо в интерфейсе Telegram описан в статье [32]. Открытие Mini App тобы открыть приложение из телеграм-бота, нужно в сообщение (sendMessage) добавить поле reply_markup следующего содержания:

```
{
 inline_keyboard: [
   [
     {
       text: "Открыть приложение",
       web_app: { url: "https://bot27.daibot.fun/app"}
     }
   ]
 ]
}
```

Создаём ссылку на веб-приложение в @BotFather командой /newapp

Указываем имя бота @user27_daibot, вводим заголовок приложения "WebApp - Параметры фильтра" и короткое описание "Диалог ввода параметров преобразования растрового изображения.". Затем предлагается загрузить фото 640x360. Ещё предлагает загрузить демонстрационную gif-анимацию.
Самое главное просит прислать URL-ссылку на наше приложение:

`https://bot27.daibot.fun/app`

Указал еще короткое имя "bot27daibotfun" и Телеграм ответит:

`t.me/user27_daibot/bot27daibotfun`

Разместим html-приложение в папке с ботом в файле webapp.html:

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        * {
            margin: 0;
            padding: 0;
            box-sizing: border-box;
        }

        body {
            color: var(--tg-theme-text-color);
            background: var(--tg-theme-bg-color);
        }

        h1 {
            margin-top: 40px;
            margin-bottom: 10px;
            text-align: center;
        }

        .btn {
            border: 0;
            border-radius: 5px;
```

```
            margin-top: 50px;
            height: 60px;
            width: 200px;
            font-style: 20px;
            font-weight: 500;
            cursor: pointer;
            color: var(--tg-theme-button-text-color);
            background: var(--tg-theme-button-color);
        }

        form {
            display: block;
            text-align: center;
        }

        input {
            outline: none;
            border-radius: 5px;
            border: 2px solid #535353;
            padding: 15px 10px;
            margin: 10px 0 0;
            background: var(--tg-theme-section-separator-color);
            color: var(--tg-theme-text-color);
            transition: all .2s;
        }

        input:focus {
            border-color: var(--tg-theme-secondary-bg-color)
        }

    </style>
</head>
<body>
    <h1> Введите число порогового разделения, черно-белого преобразования изображения
[1 до 254]</h1>
    <form class="test-form">
        <input type="range" placeholder="Порого: " class="title-inp" min="1" max="254"
value="100" step="1" />
        <button class="btn s-btn">Отправить</button>
    </form>

    <script src="https://telegram.org/js/telegram-web-app.js"></script>

    <script>
        let tg = window.Telegram.WebApp;

      tg.expand();
        let sBtn = document.getElementsByClassName("s-btn")[0]

        sBtn.addEventListener("click", () => {
            let numwb = document.getElementsByClassName("title-inp")[0];
            let data = {
                numwb: numwb.value
            }

            tg.sendData(JSON.stringify(data));
            tg.close();
        });
    </script>
</body>
</html>
```

Улучшение. На данный момент в документации @telegram-apps/sdk нет expand. Он предназначен для того, чтобы при открытии Telegram Mini App на телефоне, оно открывалось во весь свой размер, а не на половину. Чтобы добавить expand, откройте файл webapp.html, который располагается в папке бота, и добавьте в его:

```
Telegram.WebApp.expand();
```

Добавим немного цвета header-заголовку в Telegram Mini App, для этого добавим следующий код после инициализации Mini App в index.js:

```
miniApp.setHeaderColor('#fcb69f');
```

Будем по запросу "https:bot27.daibot.fun/app" отдавать файл из папки скрипта html-документ с ТМА-приложением. Добавим перехват сообщения в боте:

```
const TelegramBot = require('node-telegram-bot-api');
const express = require('express');
const https = require('https');
const bodyParser=require('body-parser');
const fs = require('fs');
const fsp = require('fs').promises;
require('dotenv').config();
const sharp  = require('sharp');
const path=require('path');

const token = process.env.TG_TOKEN;
const url = process.env.URL;
const hostname = process.env.HOSTNAME;
const port = process.env.PORT;

// Переменная для хранения выбранного фильтра
let selectedFilter = null;

var options = {
  key: fs.readFileSync('/etc/letsencrypt/live/bot27.daibot.fun/privkey.pem'),
  cert: fs.readFileSync('/etc/letsencrypt/live/bot27.daibot.fun/fullchain.pem')
};

const bot = new TelegramBot(token, { webHook: true });
bot.setWebHook(`${url}/tgbot`);

const commands = [
    {
        command: "start",
        description: "Запуск бота"

    },
    {
        command: "bw",
        description: "Преобразовать изображение в ч/б цвет"
    },
    {
        command: "print",
        description: "Прислать мне изображение"
    }
  ];

// Создать системное меню из пунктов команд боту
bot.setMyCommands(commands);
```

```
const app = express();

app.use(bodyParser.json());

// Обработчик запросов от Nginx для бота. От Nginx передаём данные в обработку Telegram-
боту
app.post("/tgbot", (req, res) => {
    console.log("ТГ-сервер прислат апдейт");
    bot.processUpdate(req.body);
    res.sendStatus(200);
});

// Обработчик запросов от веб-сервера nginx на получение html-страницы Web App
app.get("/app", (req, res) => {
    console.log("Браузер запросил webapp.html");
    const filePath = path.join(__dirname, 'webapp.html');
    fs.readFile(filePath, (err, data) => {
        if (err) {
            console.log("ошибка чтения файла webapp.html");
            res.statusCode = 500;
            res.setHeader('Content-Type', 'text/plain');
            res.send('Ошибка сервера считывания webapp.html');
            return;
        } else {
            res.statusCode = 200;
            res.setHeader('Content-Type', 'text/html; charset=utf-8');
            res.send(data);
        }
    });
});

const imageProcessingState = {};

// Команда начала запуска Web Mini App для ввода параметров фильтра
bot.onText(/\/start/, (msg) => {
  bot.sendMessage(msg.chat.id, '*Bot is online*\\!\n Send an image then use commands /bw
to manipulate the image\\.\n' +
                                'Use /print to recieve your edited image\\.\n' +
                                'Click button □ ',
        { parse_mode: "MarkdownV2",
           reply_markup: {
              one_time_keyboard: true,
              inline_keyboard: [
                 [{text: 'Веб-окно  ввода параметров  □',
                   web_app: { url: "https://bot27.daibot.fun/app" }}]
              ]
           }
        });
});

// Обработка данных полученных из веб-формы HTML-приложения Web Mimi Apps
bot.on('message', async (msg) => {
  if (msg.web_app_data) {
    console.log("WebApp-окно прислало данные")
    const chatId = msg.chat.id;
    let webAppData = JSON.parse(msg.web_app_data.data);
    selectedFilter = { numwb : parseInt(webAppData.numwb) };
    bot.sendMessage(chatId, `Вы установили порог ч/б: ${selectedFilter.numwb}\n\n` +
                            `В окне webApp-ссылки "${msg.web_app_data.button_text}"` );
  }
});
```

```
// Обработка полученного графического файла - сохранение на диск
bot.on('photo', async (msg) => {
  const chatId = msg.chat.id;
  const userId = msg.from.id;
  const photo = msg.photo[msg.photo.length - 1].file_id;
  try {
    const filePath = await bot.downloadFile(photo, ".");

    imageProcessingState[userId] = {
      filePath,
      modifications: [],
    };
    bot.sendMessage(chatId, 'Image downloaded successfully.');
  } catch (error) {
    console.error("Error downloading image", error);
    bot.sendMessage(chatId, 'Error downloading image. Please try again.');
  }
});

// Команда обработки изображения
bot.onText(/\/bw/, async (msg) => {
  const chatId = msg.chat.id;
  const userId = msg.from.id;

  if (!selectedFilter) {
    bot.sendMessage(chatId, 'Сначала установите порог для фильтра в веб-приложении.');
    return;
  }

  if (imageProcessingState[userId]) {
    try {
      const currentImagePath = imageProcessingState[userId].filePath;
      const outputImagePath = path.dirname(currentImagePath) + '/out' +
path.basename(currentImagePath);

sharp(currentImagePath).greyscale().threshold(selectedFilter.numwb).toFile(outputImagePath,
function(err) {
        if (err) {
          console.error("Error save image", err);
          bot.sendMessage(chatId, 'Error save b/w image file' + currentImagePath +
                                  '. Please try again.')
        } else {
          imageProcessingState[userId].filePath = outputImagePath;
          bot.sendMessage(chatId, 'Image has been transform to ' +
                                  'black/white color successfully!');
        }
      });
    } catch (error) {
      console.error("Error applying /bw operation", error);
      bot.sendMessage(chatId, 'Error in process image b/w. Please try again.');
    }
  } else {
    bot.sendMessage(chatId, 'No active session. Send an image to get started.');
  }
});

// Команда отсылки изображения пользователю Telegram
bot.onText(/\/print/, async (msg) => {
  const chatId = msg.chat.id;
  const userId = msg.from.id;
  if (imageProcessingState[userId]) {
    try {
```

```
      const currentImagePath = imageProcessingState[userId].filePath;

      await bot.sendPhoto(chatId, currentImagePath);

      delete imageProcessingState[userId];
      await fsp.unlink(currentImagePath);
    } catch (error) {
      console.error("Error sending image", error);
      bot.sendMessage(chatId, 'Error sending image');
    }
  } else {
    bot.sendMessage(chatId, 'No active session');
  }
});

var httpsServer = https.createServer(options, app);

httpsServer.listen(port, hostname, () => {
    console.log(`Сервер TGbot Sharp  запущен на ${hostname} на порту ${port}`);
});
```

Перехват сообщения от веб-приложения осуществляется на уровне `bot.on('message' ...)`, что обусловлено отсутствием поля text в апдейте. Что очень странно! Отдачу html-документа с интерфейсом Telegram Web Mini Apps осуществляет сервлет бота загружая файл webapp.html из рабочей папки бота.

```
PM2 Dashboard
lq Process List qqqqqqqklqq  botsharp Logs  qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqk
x[ 0] botsharp          xx botsharp > ТГ-сервер прислат апдейт                    x
x                       xx botsharp > Браузер запросил webapp.html                x
x                       xx botsharp > Браузер запросил webapp.html                x
x                       xx botsharp > ТГ-сервер прислат апдейт                    x
x                       xx botsharp > WebApp-окно прислало данные                 x
x                       xx botsharp > ТГ-сервер прислат апдейт                    x
x                       xx botsharp > ТГ-сервер прислат апдейт                    x
x                       xx botsharp > ТГ-сервер прислат апдейт                    x
x                       xx                                                        x
x                       xx                                                        x
x                       xx                                                        x
x                       xx                                                        x
x                       xx                                                        x
x                       xx                                                        x
x                       xx                                                        x
mqqqqqqqqqqqqqqqqqqqqqqjmqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqj
lq Custom Metrics qqqqqklq Metadata qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqk
x Heap Size            xx App Name              botsharp                         x
x Heap Usage            xx Namespace             default                          x
x Used Heap Size        xx Version               1.0.0                            x
x Active requests       xx Restarts              0                                x
mqqqqqqqqqqqqqqqqqqqqqqjmqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqj
 left/right: switch boards | up/down/mouse: scroll | Ctrl-C: exit          To go
```

Результат работы можно посмотреть в папке /home/user27/pm2/logs в файле botsharp.log или в панели монитора $pm2 monit. Структура данных от команды и от html-формы WebApp-приложения отличаются. В апдете сообщения нет поля "text" и нельзя применить обработчик событий bot.onText. Ниже приведены сохраненные JSON-структуры запросов от ТГ-сервера к боту.

JSON-структура команды /start	JSON-данные от веб-окна Mini Web App

```
{
  update_id: 645603716,
  message: {
    message_id: 313,
    from: {
      id: 5712345383,
      is_bot: false,
      first_name: 'Олег',
      last_name: 'Гудаев',
      username: 'goodarget',
      language_code: 'ru'
    },
    chat: {
      id: 5712345383,
      first_name: 'Олег',
      last_name: 'Гудаев',
      username: 'goodarget',
      type: 'private'
    },
    date: 1732225607,
    text: '/start',
    entities: [ [Object] ]
  }
}
```

```
{
  update_id: 645603717,
  message: {
    message_id: 315,
    from: {
      id: 5712345383,
      is_bot: false,
      first_name: 'Олег',
      last_name: 'Гудаев',
      username: 'goodarget',
      language_code: 'ru'
    },
    chat: {
      id: 5712345383,
      first_name: 'Олег',
      last_name: 'Гудаев',
      username: 'goodarget',
      type: 'private'
    },
    date: 1732225615,
    web_app_data: {
      button_text: 'Веб-приложение ввода параметров □',
      data: '{"numwb":"158"}'
    }
  }
}
```

Вид веб-интерфейса Web Mini App. Развернут по высоте окна.

Полученные результаты из веб-окна html-ввода данных подтверждаются ботом в виде ответного сообщения, смотри на экранной форме.

Вывод проекта. Телеграм-бот был создан с дружественным интерфейсом!

Завершить все процессы пользователя

Как мы знаем, Linux – многопользовательская ОС, поэтому с каждым пользователем могут быть связаны его собственные процессы [33].

Используем конвейер и программу grep. Символ | обозначает передачу результатов выполнения первой команды второй команде по конвейеру:

```
# ps aux | grep -e '^user27'
```

С помощью grep были выбраны все процессы пользователя user27. Далее необходимо из второй колонки слева выбрать все PID. Сделаем это с помощью программы awk, добавив, соответственно, ещё один конвейер:

```
# ps aux | grep -e '^user27' | awk '{print $2}'
```

Теперь нужно завершить процессы с полученным значением PID.

```
# ps au | grep -e '^user27' | awk '{print $2}' | xargs kill -9
```

или

```
# killall -u user27
```

Учебный проект "Daibot.fun". Часть - 8.

Задание проекта: Создать мини-приложение Телеграм генерации визуального улучшения плакатов спецэффектами SVG-фильртов, на базе виртуального выделенного сервера с доменным именем daibot.fun.

Базовый алгоритм ИИ: написан на языке программирования JavaScript.

Постановка задачи

```
План работ:
- Многопользовательское подключение ботов на VPS-сервере.
```

Ошибки проекта

Возникает ошибка при запуске сервлет-бота:

```
# Ошибка Node ------------------------------------------------
Error: listen EADDRINUSE: address already in use 0.0.0.0:8443
  code: 'EADDRINUSE',
  errno: -98,
  syscall: 'listen',
  address: '0.0.0.0',
  port: 8443
}
```

1. Плохое решение сразу "на лету". Открыть порты 8443 в предположении, что неправильно интерпретирована ситуация Node.js и порт просто не открыть.
Открыть порт в брандмауэре ufw:

```
# ufw allow 8443/tcp
# ufw reload
# systemctl restart nginx
```

2. Правильный разбор ошибки. Ошибка "Error: listen EADDRINUSE" означает что порт, на котором вы пытаетесь запустить приложение, уже занят – другим приложением или другим инстансом вашего же сервера.
Две разные программы не могут слушать один и тот же порт. Одну вы запустили через pm2, а потому пытаетесь запустить вторую. Если вы столкнулись с ошибкой EADDRINUSE, которая указывает на то, что порт 8443 как будто бы занят. Примените команду:

```
$ lsof -i :8443
```

Для того, чтобы обнаружить скрытые процессы. Команда lsof ищет в открытых файлах процессов текст ":8443". Команда lsof – это утилита для вывода списка открытых файлов в Linux и отображения процессов, которые их используют.
Чтобы их прервать, введите в терминале:

```
$ kill -9 $(lsof -t -i :8443).
```

Как проверить наличие используемых, прослушиваемых портов в Linux [34]. Утилита netstat – это утилита командной строки для проверки портов прослушивания.

```
$ netstat -tunlp | head -5
```

Ситуация пока не прояснилась, кто-то слушает этот порт под user27:

```
mc [user27@ruvds-d0ohk]:/etc/nginx/sites-available
user27@ruvds-d0ohk:/etc/nginx/sites-available$ netstat -tunlp | head -5
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:8443            0.0.0.0:*               LISTEN      -
user27@ruvds-d0ohk:/etc/nginx/sites-available$ pm2 list

| id | name | mode | □ | status | cpu | memory |

user27@ruvds-d0ohk:/etc/nginx/sites-available$
```

Порт 8443 занят. Порт 22 – это SSH-терминал, в котором мы сейчас связались с с VPS-сервером. Порт 80 – это веб-сервер NGINX. А кто слушает 8443? Тоже node.js.
Запусти приложение botsharp.js и проверим порты утилитой lsoft:

```
mc [user27@ruvds-d0ohk]:~/botsharp
user27@ruvds-d0ohk:~/botsharp$ pm2 start  botsharp.js
[PM2] Starting /home/user27/botsharp/botsharp.js in fork_mode (1 instance)
[PM2] Done.

| id | name | mode | □ | status | cpu | memory |
| 0 | botsharp | fork | 0 | online | 0% | 26.2mb |

user27@ruvds-d0ohk:~/botsharp$ lsof -i :3027
COMMAND        PID   USER   FD   TYPE   DEVICE SIZE/OFF NODE NAME
node\x20/ 3110373 user27   20u  IPv6 55694121      0t0  TCP localhost:3027 (LISTEN)
user27@ruvds-d0ohk:~/botsharp$ lsof -i :8443
user27@ruvds-d0ohk:~/botsharp$
```

Видим теперь порт 3027 занять node.js. Так и должно быть, а порт 8443 неизвестно кто занял.
Команда `netstat -ano` покажет все используемые порты.

```
mc [user27@ruvds-d0ohk]:~/.pm2/logs
user27@ruvds-d0ohk:~/.pm2/logs$ netstat -ano
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       Timer
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      off (0.00/0/0)
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      off (0.00/0/0)
tcp        0      0 0.0.0.0:8443            0.0.0.0:*               LISTEN      off (0.00/0/0)
tcp        0      0 0.0.0.0:443             0.0.0.0:*               LISTEN      off (0.00/0/0)
tcp        0      0 45.8.230.131:22         193.23.58.118:23016     ESTABLISHED keepalive (2780.27/0/0)
tcp        0      0 45.8.230.131:43698      149.154.167.220:443     ESTABLISHED keepalive (0.68/0/0)
tcp        0      0 45.8.230.131:43710      149.154.167.220:443     ESTABLISHED keepalive (0.12/0/0)
tcp6       0      0 ::1:3022                :::*                    LISTEN      off (0.00/0/0)
tcp6       0      0 :::80                   :::*                    LISTEN      off (0.00/0/0)
tcp6       0      0 ::1:3027                :::*                    LISTEN      off (0.00/0/0)
tcp6       0      0 ::1:3028                :::*                    LISTEN      off (0.00/0/0)
tcp6       0      0 ::1:3030                :::*                    LISTEN      off (0.00/0/0)
tcp6       0      0 :::22                   :::*                    LISTEN      off (0.00/0/0)
tcp6       0      0 :::3032                 :::*                    LISTEN      off (0.00/0/0)
tcp6       0      0 ::1:3033                :::*                    LISTEN      off (0.00/0/0)
tcp6       0      0 :::443                  :::*                    LISTEN      off (0.00/0/0)
tcp6       0      0 :::3004                 :::*                    LISTEN      off (0.00/0/0)
tcp6       0      0 :::3011                 :::*                    LISTEN      off (0.00/0/0)
Active UNIX domain sockets (servers and established)
```

Дополнительные сведения о программах в Linux. Чтобы получить подробные сведения о Nginx, выполните следующие команды:

```
$ whatis nginx
$ whereis nginx
$ man nginx
```

Поиск номера процесса командой `fuser` и просмотр командой `ls` имени, от которого запущен процесс, выдаёт сведения только для порта 3037, а для 8443 выдают пустые строки, потому что нет прав root.

```
mc [user27@ruvds-d0ohk]:~/.pm2/logs
user27@ruvds-d0ohk:~/.pm2/logs$ fuser 3027/tcp
3027/tcp:             3110373
user27@ruvds-d0ohk:~/.pm2/logs$ ls -l /proc/3110373/exe
lrwxrwxrwx 1 user27 user27 0 Nov 24 17:20 /proc/3110373/exe -> /usr/local/nvm/versions/node/v20.18.0/bin/node
user27@ruvds-d0ohk:~/.pm2/logs$
```

Проверим запущенные процессы командой htop с настройками фильтра.

```
mc [user27@ruvds-d0ohk]:~/.pm2/logs
  CPU[||||||||||||||||||||||||||||||||||||||||100.0%]   Tasks: 64, 204 thr; 1 running
  Mem[||||||||||||||||||||||||||||||||||710M/949M]   Load average: 1.68 1.68 1.64
  Swp[                                      0K/0K]   Uptime: 9 days, 21:01:58

    PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
2937814 root       20   0 13940  3324  1860 S  0.0  0.3  0:00.03 sshd: user27 [priv]
3210878 root       20   0 1001M 50272 39532 S  0.0  5.2  0:00.00 node /home/user27/app/index.js
3210879 root       20   0 1001M 50272 39532 S  0.7  5.2  0:00.01 node /home/user27/app/index.js
3210880 root       20   0 1001M 50272 39532 S  0.0  5.2  0:00.00 node /home/user27/app/index.js
3210881 root       20   0 1001M 50272 39532 S  0.7  5.2  0:00.01 node /home/user27/app/index.js
3210882 root       20   0 1001M 50272 39532 S  1.3  5.2  0:00.02 node /home/user27/app/index.js
3210883 root       20   0 1001M 50272 39532 S  0.0  5.2  0:00.00 node /home/user27/app/index.js
3210884 root       20   0 1001M 50272 39532 S  0.0  5.2  0:00.00 node /home/user27/app/index.js
3210885 root       20   0 1001M 50272 39532 S  0.0  5.2  0:00.00 node /home/user27/app/index.js
3210886 root       20   0 1001M 50272 39532 S  0.0  5.2  0:00.00 node /home/user27/app/index.js
3210887 root       20   0 1001M 50272 39532 S  0.0  5.2  0:00.00 node /home/user27/app/index.js
3210877 root       20   0 1001M 50272 39532 R 11.2  5.2  0:00.17 node /home/user27/app/index.js
2798635 user27     20   0 1076M 38012 14172 S  0.0  3.9  0:00.00 PM2 v4.5.5: God Daemon (/home/user27/.pm2)
2798636 user27     20   0 1076M 38012 14172 S  0.0  3.9  0:01.31 PM2 v4.5.5: God Daemon (/home/user27/.pm2)
2798637 user27     20   0 1076M 38012 14172 S  0.0  3.9  0:00.85 PM2 v4.5.5: God Daemon (/home/user27/.pm2)
2798638 user27     20   0 1076M 38012 14172 S  0.0  3.9  0:00.80 PM2 v4.5.5: God Daemon (/home/user27/.pm2)
2798639 user27     20   0 1076M 38012 14172 S  0.0  3.9  0:00.72 PM2 v4.5.5: God Daemon (/home/user27/.pm2)
2798640 user27     20   0 1076M 38012 14172 S  0.0  3.9  0:00.00 PM2 v4.5.5: God Daemon (/home/user27/.pm2)
2798715 user27     20   0 1076M 38012 14172 S  0.0  3.9  0:00.73 PM2 v4.5.5: God Daemon (/home/user27/.pm2)
2798716 user27     20   0 1076M 38012 14172 S  0.0  3.9  0:00.69 PM2 v4.5.5: God Daemon (/home/user27/.pm2)
2798717 user27     20   0 1076M 38012 14172 S  0.0  3.9  0:00.74 PM2 v4.5.5: God Daemon (/home/user27/.pm2)
2798718 user27     20   0 1076M 38012 14172 S  0.0  3.9  0:00.72 PM2 v4.5.5: God Daemon (/home/user27/.pm2)
2798629 user27     20   0 1076M 38012 14172 S  0.0  3.9  2:28.65 PM2 v4.5.5: God Daemon (/home/user27/.pm2)
Enter Done  Esc Clear   Filter: user27
```

Кликнем на поле filter и введем текст "user27", для сравнения при фильтрации процессов. Видим, на мгновение, появляется и исчезает запуск приложения index.js от имени root, но в папке "/home/user27/". Недоброжелательный процесс

найден. Я когда-то запустил приложение из-под администратора root. С правами администратора сетевые утилиты netstat выдают больше информации:

```
root@ruvds-d0ohk: ~
root@ruvds-d0ohk:~# lsof -i :8443
COMMAND       PID   USER   FD   TYPE   DEVICE SIZE/OFF NODE NAME
node\x20/ 1064049 user33   20u  IPv4 52178785      0t0  TCP *:8443 (LISTEN)
root@ruvds-d0ohk:~# netstat -tunlp | head -5
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      509676/nginx: maste
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      548/sshd: /usr/sbin
tcp        0      0 0.0.0.0:8443            0.0.0.0:*               LISTEN      1064049/node /home/
root@ruvds-d0ohk:~# fuser 8443/tcp
8443/tcp:            1064049
root@ruvds-d0ohk:~# ls -l /proc/1064049/exe
lrwxrwxrwx 1 user33 user33 0 Nov 24 17:51 /proc/1064049/exe -> /usr/local/nvm/versions/node/v20.18.0/bin/node
```

Зайдём root, затем pm2 monit и удалить запуск pm2 delete 0. Ошибки "на лицо":

```
PM2 Dashboard
Process List                                  index Logs
[ 0] index     Mem:  47 MB    CPU:  0 %  onl
                                index >     at Module.load (node:internal/modules/cjs/loader:1288:32)
                                index >     at Module._load (node:internal/modules/cjs/loader:1104:12)
                                index >     at Function.executeUserEntryPoint [as runMain] (node:internal/modules/run_main:174:12)
                                index >     at node:internal/main/run_main_module:28:49 {
                                index >   code: 'MODULE_NOT_FOUND',
                                index >   requireStack: []
                                index > }
                                index > Error: Cannot find module '/home/user27/app/index.js'
                                index >     at Module._resolveFilename (node:internal/modules/cjs/loader:1225:15)
                                index >     at Module._load (node:internal/modules/cjs/loader:1051:27)
                                index >     at Object.<anonymous> (/usr/local/share/.config/yarn/global/node_modules/pm2/lib/ProcessCont
                                index >     at Module._compile (node:internal/modules/cjs/loader:1469:14)
                                index >     at Module._extensions..js (node:internal/modules/cjs/loader:1548:10)
                                index >     at Module.load (node:internal/modules/cjs/loader:1288:32)
                                index >     at Module._load (node:internal/modules/cjs/loader:1104:12)
                                index >     at Function.executeUserEntryPoint [as runMain] (node:internal/modules/run_main:174:12)
                                index >     at node:internal/main/run_main_module:28:49 {
                                index >   code: 'MODULE_NOT_FOUND',
                                index >   requireStack: []
                                index > }
                                index > Error: Cannot find modode:internal/modules/cjs/loader:1104:12)
                                index >     at Function.executeUserEntryPoint [as runMain] (node:internal/modules/run_main:174:12)
                                index >     at node:internal/main/run_main_module:28:49 {
                                index >   code: 'MODULE_NOT_FOUND',
                                index >   requireStack: []
                                index > }
Custom Metrics                  Metadata
                                App Name          index
                                Namespace         default
                                Version           1.0.0
                                Restarts          2734130
                                Uptime            0s
                                Script path       /home/user27/app/index.js
                                Script args       N/A
                                Interpreter       node
 left/right: switch boards | up/down/mouse: scroll | Ctrl-C: exit                To go further check out https://pm2.io/
```

Вывод 1. Не оставляйте nodejs-программы с ошибками в запуске менеджера памяти pm2.

Вывод 2. Осторожно с использованием сетевых библиотек.

Библиотека `node-telegram-bot-api`, `node-fetch` использует адрес 0.0.0.0. по умолчанию. Может это оно?

Переписка на сайте github.com:

https://github.com/strapi/foodadvisor/issues/124?ysclid=m3vv4rb81s222050831

"Надеюсь, я уже стал жертвой этой проблемы и хочу поделиться простым решением (найдено здесь):

node-fetch/node-fetch#1624

Вместо «localhost» вы бы предпочли установить «127.0.0.1» в ключе конфигурации API_URL."

Решение проблемы: задать для бота свой порт. Для всех разработчиков были открыты дополнительные порты с номером 40000 + user_port. Не забудьте подключить свои SSL-сертификаты. При создании бота в исходном коде программы указать опции для webHook:

```
const ssl_key = '/etc/letsencrypt/live/bot27.daibot.fun/privkey.pem';
const ssl_cert = '/etc/letsencrypt/live/bot27.daibot.fun/fullchain.pem';
const hostname = process.env.HOSTNAME;
const ssl_port = process.env.PORTSSL;
const options = { host: hostname, port: ssl_port, key: ssl_key, cert: ssl_cert };
const bot = new TelegramBot(token, { webHook: options });
```

Программирование модулей: сервер + бот

Улучшим логи выдаваемые ботом-сервлетом и добавим форматирование даты и времени.

```
$ npm install moment --save
```

Подключим бибилиотеку и сохраним в проект. Я в папке проекта сделал файл install.sh в который записал команду npm install со всеми библиотеками. Программирование даты и времени в боте:

```
const moment = require('moment');
function dtn () {
 return moment().format('YYYY-MM-DD HH:mm:ss') + ' ';
}
```

Добавим время события во все команды consol.log(dtn() + "Сообщение ... ").

Разделим исходный код на сервер sharpbot.js и модуль ТГ-бота bot.js

Листинг bot.js

```
const TelegramBot = require('node-telegram-bot-api');
const fs = require('fs');
const fsp = require('fs').promises;
require('dotenv').config();
const sharp  = require('sharp');
const path=require('path');
const moment = require('moment');

// Функция даты и времени
function dtn () {
 return moment().format('YYYY-MM-DD HH:mm:ss') + ' БОТ: ';
}

// Переменная для хранения выбранного фильтра
let selectedFilter = null;
const token = process.env.TG_TOKEN;
const url = process.env.URL;
const ssl_key = process.env.KEY;
const ssl_cert = process.env.CERT;
const hostname = process.env.HOSTNAME;
const ssl_port = process.env.PORTSSL;
const options = { host: hostname, port: ssl_port, key: ssl_key, cert: ssl_cert };
const bot = new TelegramBot(token, { webHook: options });
bot.setWebHook(`${url}/tgbot`);

const commands = [
    {
        command: "start",
        description: "Запуск бота"
    },
    {
```

```
        command: "bw",
        description: "Преобразовать изображение в ч/б цвет"
    },
    {
        command: "print",
        description: "Прислать мне изображение"
    }
  ];
// Создать системное меню из пунктов команд боту
bot.setMyCommands(commands);

const imageProcessingState = {};

// Команда начала запуска Web Mini App для ввода параметров фильтра
bot.onText(/\/start/, (msg) => {
  bot.sendMessage(msg.chat.id, '*Bot is online*\\!\n Send an image then use commands
/bw to manipulate the image\\.\n' +
                                'Use /print to recieve your edited image\\.\n' +
                                'Click button □ ',
        { parse_mode: "MarkdownV2",
           reply_markup: {
//              one_time_keyboard: true,
              resize_keyboard: true,
              keyboard: [
                 [{text: 'Веб-окно  ввода параметров  □',
                   web_app: { url: "https://bot27.daibot.fun/app" }}]
              ]
           }
        });
});

// Обработка данных полученных из веб-формы HTML-приложения Web Mimi Apps
bot.on('message', async (msg) => {
  if (msg.web_app_data) {
    console.log(dtn() + "WebApp-окно прислало данные")
    const chatId = msg.chat.id;
    let webAppData = JSON.parse(msg.web_app_data.data);
    selectedFilter = { numwb : parseInt(webAppData.numwb) };
    bot.sendMessage(chatId, `Вы установили порог ч/б: ${selectedFilter.numwb}\n\n` +
                            `В окне webApp-ссылки "${msg.web_app_data.button_text}"` );
  }
});

// Обработка полученного графического файла - сохранение на диск
bot.on('photo', async (msg) => {
  const chatId = msg.chat.id;
  const userId = msg.from.id;
  const photo = msg.photo[msg.photo.length - 1].file_id;
  try {
    const filePath = await bot.downloadFile(photo, ".");

    imageProcessingState[userId] = {
      filePath,
      modifications: [],
    };
    bot.sendMessage(chatId, 'Image downloaded successfully.');
  } catch (error) {
    console.error(dtn() + "Error downloading image", error);
    bot.sendMessage(chatId, 'Error downloading image. Please try again.');
  }
});
```

```
// Команда обработки изображения
bot.onText(/\/bw/, async (msg) => {
  const chatId = msg.chat.id;
  const userId = msg.from.id;
  if (!selectedFilter) {
    bot.sendMessage(chatId, 'Сначала установите порог для фильтра в веб-приложении.');
    return;
  }
  if (imageProcessingState[userId]) {
    try {
      const currentImagePath = imageProcessingState[userId].filePath;
      const outputImagePath = path.dirname(currentImagePath) + '/out' +
path.basename(currentImagePath);

sharp(currentImagePath).greyscale().threshold(selectedFilter.numwb).toFile(outputImageP
ath, function(err) {
        if (err) {
          console.error(dtn() + "Error save image", err);
          bot.sendMessage(chatId, 'Error save b/w image file' + currentImagePath + '.
Please try again.')
        } else {
          imageProcessingState[userId].filePath = outputImagePath;
          bot.sendMessage(chatId, 'Image has been transform to black/white color
successfully!');
        }
      });
    } catch (error) {
      console.error(dtn() + "Error applying /bw operation", error);
      bot.sendMessage(chatId, 'Error in process image b/w. Please try again.');
    }
  } else {
    bot.sendMessage(chatId, 'No active session. Send an image to get started.');
  }
});

// Команда отсылки изображения пользователю Telegram
bot.onText(/\/print/, async (msg) => {
  const chatId = msg.chat.id;
  const userId = msg.from.id;
  if (imageProcessingState[userId]) {
    try {
      const currentImagePath = imageProcessingState[userId].filePath;

      await bot.sendPhoto(chatId, currentImagePath);

      delete imageProcessingState[userId];
      await fsp.unlink(currentImagePath);
    } catch (error) {
      console.error(dtn() + "Error sending image", error);
      bot.sendMessage(chatId, 'Error sending image');
    }
  } else {
    bot.sendMessage(chatId, 'No active session');
  }
});
module.exports = bot;
```

Листинг botsharp.js

```
const express = require('express');
const https = require('https');
const bodyParser=require('body-parser');
const fs = require('fs');
require('dotenv').config();
const path=require('path');
const moment = require('moment');
const bot = require('./bot.js')

const hostname = process.env.HOSTNAME;
const port = process.env.PORT;
const key = process.env.KEY;
const cert = process.env.CERT;

// Функция даты и времени
function dtn () {
 return moment().format('YYYY-MM-DD HH:mm:ss') + ' SRV: ';
}

var options = {
  key: fs.readFileSync(key),
  cert: fs.readFileSync(cert)
};

const app = express();

app.use(bodyParser.json());

// Обработчик запросов от Nginx для бота. От Nginx передаём данные в обработку
Telegram-боту
app.post("/tgbot", (req, res) => {
    console.log(dtn() + "ТГ-сервер прислат апдейт");
    bot.processUpdate(req.body);
    res.sendStatus(200);
});

// Обработчик запросов от веб-сервера nginx на получение html-страницы Web App
app.get("/app", (req, res) => {
    console.log(dtn() + "Браузер запросил webapp.html")
    const filePath = path.join(__dirname, 'webapp.html');
    fs.readFile(filePath, (err, data) => {
        if (err) {
            console.log(dtn() + "Ошибка чтения файла webapp.html");
            res.statusCode = 500;
            res.setHeader('Content-Type', 'text/plain');
            res.send(dtn() + 'Ошибка сервера считывания webapp.html');
            return;
        } else {
            res.statusCode = 200;
            res.setHeader('Content-Type', 'text/html; charset=utf-8');
            res.send(data);
        }
    });
});

var httpsServer = https.createServer(options, app);

httpsServer.listen(port, hostname, () => {
    console.log(dtn() + `Сервер TGbot Sharp  запущен на ${hostname} на порту ${port}`);
});
```

Листинг .env

```
TG_TOKEN=7645673456:yyyyyDi7BXLOVViALgCyyyyyyooDPcKyyyy
URL=https://bot27.daibot.fun
HOSTNAME=127.0.0.1
PORT=3027
PORTSSL=43027
KEY=/etc/letsencrypt/live/bot27.daibot.fun/privkey.pem
CERT=/etc/letsencrypt/live/bot27.daibot.fun/fullchain.pem
```

Листинг install.sh

```
#/bin/bash
npm install node-telegram-bot-api express https body-parser fs sharp path dotenv moment --save
```

Список дополнительных портов для ТГ-бота

Юзер	Дополнительный порт
user01	43001
user02	43002
user03	43003
user04	43004
user05	43005
user06	43006
user07	43007
user08	43008
user09	43009
user10	43010
user11	43011
user12	43012
user21	43021
user22	43022
user23	43023
user24	43024
user25	43025
user26	43026
user27	43027
user28	43028
user29	43029
user30	43030
user31	43031
user32	43032
user33	43033
user34	43034
user35	43035
user36	43036
user37	43037
user38	43038

Учебный проект "Daibot.fun". Часть - 9.

Задание проекта: Создать мини-приложение Телеграм ведение SQLite базы данных, на базе виртуального выделенного сервера с доменным именем daibot.fun.

Базовая потребность: продуктовое программирование.

Постановка задачи

```
План работ:
- Ведение базы данных на VPS-сервере.
  проект любезно предоставлен магистром ПОИСм-24 (с) Кириленко Николаем.
```

База данных

База данных SQLite размещается в одном файле [35]. Для создания и просмотра БД используют инструменты трех видов [36]. Установим под Window утилиту "DBBrowser for SQLite ". Структура таблиц базы данных приведена ниже:

Для создания базы на стороне VPS-сервера создадим модуль db.js:

```
// db.js
const sqlite3 = require('sqlite3').verbose();
const path = require('path');
```

```
// Файл базы: shop.db
const dbPath = path.join(__dirname, 'shop.db');
const db = new sqlite3.Database(dbPath);

db.serialize(() => {
  // Таблица категорий
  db.run(`
    CREATE TABLE IF NOT EXISTS categories (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      name TEXT NOT NULL
    )
  `);

  // Таблица книг (связь на категорию)
  db.run(`
    CREATE TABLE IF NOT EXISTS books (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      title TEXT NOT NULL,
      author TEXT,
      price REAL NOT NULL,
      cover TEXT,
      description TEXT,
      category_id INTEGER,
      FOREIGN KEY(category_id) REFERENCES categories(id)
    )
  `);

  // Таблица корзин (carts)
  db.run(`
    CREATE TABLE IF NOT EXISTS carts (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      user_id INTEGER NOT NULL,
      created_at TEXT NOT NULL
    )
  `);

  // Таблица позиций в корзинах (cart_items)
  db.run(`
    CREATE TABLE IF NOT EXISTS cart_items (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      cart_id INTEGER NOT NULL,
      book_id INTEGER NOT NULL,
      quantity INTEGER NOT NULL,
      FOREIGN KEY(cart_id) REFERENCES carts(id),
      FOREIGN KEY(book_id) REFERENCES books(id)
    )
  `);

  console.log('Таблицы инициализированы (если их не существовало).');
});

function seedDatabase() {
  // Добавим 4 категории
  db.run(`
    INSERT INTO categories (name) VALUES
      ('Детектив'),
      ('Русская классика'),
      ('Фэнтези'),
      ('Антиутопия')
  `);



  const sql = `
    INSERT INTO books (title, author, price, cover, description, category_id)
    VALUES
      ('Приключения Шерлока Холмса', 'Артур Конан Дойл', 350, 'https://cdn1.ozone.ru/s3/multimedia-
o/6020017536.jpg', 'Знаменитые детективные истории...', 1),
```

```
      ('Мастер и Маргарита', 'М.А. Булгаков', 450, 'https://i.pinimg.com/736x/22/06/67/2206676a2896cc4ebcbeb4fc2751cec7.jpg', 'Один из величайших романов XX века...', 2),
      ('Война и мир', 'Л.Н. Толстой', 500, 'https://cdn1.ozone.ru/multimedia/1010257708.jpg', 'Эпопея о судьбах аристократических семей...', 2),
      ('Гарри Поттер и философский камень', 'Дж.К. Роулинг', 320, 'https://ir.ozone.ru/s3/multimedia-k/c1000/6155640512.jpg', 'Первая часть о юном волшебнике...', 3),
      ('1984', 'Джордж Оруэлл', 280, 'https://cdn1.ozone.ru/s3/multimedia-y/6353881258.jpg', 'Антиутопический роман...', 4)
  `;
  db.run(sql, (err) => {
    if (err) {
      console.error('Ошибка при начальном наполнении:', err.message);
    } else {
      console.log('Начальные тестовые данные добавлены.');
    }
  });
}

seedDatabase();

module.exports = db;
```

Подключим модуль в боте-сервлете server.js:

```
// server.js
const https = require('https');
const fs = require('fs');
const path = require('path');
require('dotenv').config();
const express = require('express');
const db = require('./db');        // Модуль работы с SQLite
const { bot } = require('./bot'); // Телеграм-бот

const hostname = process.env.HOSTNAME;
const port = process.env.PORT;
const key = process.env.KEY;
const cert = process.env.CERT;

const app = express();
app.use(express.json());

// Настройки для HTTPS
const options = {
  key: fs.readFileSync(key),
  cert: fs.readFileSync(cert)
};

// 1) Webhook от телеги
app.post('/tgbot', (req, res) => {
  bot.processUpdate(req.body);
  res.sendStatus(200);
});

// 2) Список категорий => /app/categories
app.get('/app/categories', (req, res) => {
  db.all(`SELECT id, name FROM categories`, (err, rows) => {
    if (err) {
      console.error('Ошибка выборки categories:', err);
      return res.status(500).json({ error: 'DB error' });
    }
    res.json(rows);
  });
});
```

```
// 3) Список книг => /app/books?categoryId=...
//    Сделаем LEFT JOIN на categories, чтобы вернуть category_name
app.get('/app/books', (req, res) => {
  const { categoryId } = req.query;
  let sql = `
    SELECT b.id, b.title, b.author, b.price, b.cover, b.description,
           c.name AS category_name
    FROM books b
    LEFT JOIN categories c ON b.category_id = c.id
  `;
  const params = [];

  if (categoryId) {
    sql += ` WHERE b.category_id = ?`;
    params.push(categoryId);
  }

  db.all(sql, params, (err, rows) => {
    if (err) {
      console.error('Ошибка выборки books:', err);
      return res.status(500).json({ error: 'DB error' });
    }
    res.json(rows);
  });
});

// 4) Сохранение корзины => /app/save-cart
app.post('/app/save-cart', (req, res) => {
  const { userId, cart } = req.body;
  // cart = [{ bookId, quantity }, ...]

  if (!userId || !Array.isArray(cart)) {
    return res.status(400).json({ error: 'Некорректные данные корзины' });
  }

  const createdAt = new Date().toISOString();
  db.run(
    `INSERT INTO carts (user_id, created_at) VALUES (?, ?)`,
    [userId, createdAt],
    function(err) {
      if (err) {
        console.error('Ошибка при вставке в carts:', err);
        return res.status(500).json({ error: 'DB insert error' });
      }
      const cartId = this.lastID;

      // Вставляем позиции
      const insertItem = db.prepare(`
        INSERT INTO cart_items (cart_id, book_id, quantity)
        VALUES (?, ?, ?)
      `);

      for (const item of cart) {
        insertItem.run(cartId, item.bookId, item.quantity);
      }

      insertItem.finalize((err2) => {
        if (err2) {
          console.error('Ошибка при вставке в cart_items:', err2);
          return res.status(500).json({ error: 'DB insert error (items)' });
        }
```

```
        console.log(`Корзина userId=${userId} сохранена (cartId=${cartId})`);
        res.json({ success: true, cartId });
      });
    }
  );
});

// 5) Отдаём webapp.html => /app/webapp
app.get('/app/webapp', (req, res) => {
  const filePath = path.join(__dirname, 'webapp.html');
  fs.readFile(filePath, (err, data) => {
    if (err) {
      console.error('Ошибка чтения webapp.html:', err);
      return res.status(500).send('Ошибка сервера');
    }
    res
      .status(200)
      .setHeader('Content-Type', 'text/html; charset=utf-8')
      .send(data);
  });
});

// 6) Отдаём statistics.html => /app/statistics
app.get('/app/statistics', (req, res) => {
  const filePath = path.join(__dirname, 'statistics.html');
  fs.readFile(filePath, (err, data) => {
    if (err) {
      console.error('Ошибка чтения statistics.html:', err);
      return res.status(500).send('Ошибка сервера');
    }
    res
      .status(200)
      .setHeader('Content-Type', 'text/html; charset=utf-8')
      .send(data);
  });
});

// Запуск HTTPS-сервера
const httpsServer = https.createServer(options, app);
httpsServer.listen(port, hostname, () => {
  console.log(`Сервер TGbot запущен на https://${hostname}:${port}`);
});
```

Создадим модуль Телеграм-бота bot.js:

```
// bot.js
const TelegramBot = require('node-telegram-bot-api');
require('dotenv').config();

const token = process.env.TG_TOKEN;
const url = process.env.URL;
const ssl_key = process.env.KEY;
const ssl_cert = process.env.CERT;
const hostname = process.env.HOSTNAME;
const ssl_port = process.env.PORTSSL;

const bot = new TelegramBot(token, {
  webHook: {
    port: ssl_port,
    host: hostname,
    key: ssl_key,
    cert: ssl_cert,
```

```
  },
});

bot.setWebHook(`${url}/tgbot`);

// При команде /start
bot.onText(/\/start/, (msg) => {
  const chatId = msg.chat.id;
  bot.sendMessage(chatId, 'Добро пожаловать в продвинутый книжный магазин!', {
    reply_markup: {
      inline_keyboard: [
        [
          {
            text: 'Меню (WebApp)',
            web_app: { url: `${url}/app/webapp?userId=${chatId}` }
          }
        ],
        [
          {
            text: 'Показать таблицы',
            web_app: { url: `${url}/app/statistics` }
          }
        ]
      ]
    }
  });
});

module.exports = { bot };
```

Подключим все зависимые библиотеки в проект в файле package.json:

```
{
  "dependencies": {
    "dotenv": "^16.4.7",
    "path": "^0.12.7",
    "fs": "^0.0.1-security",
    "https": "^1.0.0",
    "express": "^4.21.2",
    "node-telegram-bot-api": "^0.66.0",
    "sqlite3": "^5.1.7"
  }
}
```

Веб-приложение миниапп webapp.html должно получать ID-пользователя и отправлять данные ТГ-серверу, а он передаст Вашему боту. Это хороший вариант:

```
<body>
    <h1> Каталог электронного магазина</h1>
    <form id="sale-form">
        <input  type="range"  placeholder="Цена:  "  id="cena"  min="450"  max="500"
value="470" step="1" />
        <button id="sale">Отправить</button>
    </form>

    <script src="https://telegram.org/js/telegram-web-app.js"></script>
    <script language='javascript'>
        let tg = window.Telegram.WebApp;
        tg.setHeaderColor('#494992');
        tg.expand();
        let elm1 = document.getElementsByID("sale");
```

```
        elm1.addEventListener("click", () => {
            let price = document.getElementsByID("cena");
            let data = {
                numwb: price.value
            }
         tg.sendData(JSON.stringify(data));
         tg.close();
        });
    </script>
</body>
```

Но, ест второй вариант, получает ID-пользователя через адресную строку. Это "сырой" вариант приложения, без использования TG-API. Это приём гарантирует уникальность html-документа электронного магазина. Код разместите в файле webapp.html:

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <title>Продвинутый магазин книг</title>
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <style>
    /* Упрощённые стили */
    * { box-sizing: border-box; }
    body {
      margin: 0;
      font-family: Arial, sans-serif;
      background-color: #f4f4f9;
      color: #333;
      display: flex;
      flex-direction: column;
      height: 100vh;
    }
    .header {
      background-color: #6200ee;
      color: #fff;
      padding: 10px;
      text-align: center;
      font-weight: bold;
      font-size: 18px;
    }
    .top-controls {
      display: flex;
      gap: 10px;
      align-items: center;
      padding: 10px;
    }
    .content {
      flex: 1;
      padding: 10px;
      overflow-y: auto;
    }
    .catalog-grid {
      display: grid;
      grid-template-columns: 1fr 1fr;
      gap: 10px;
    }
    .catalog-item {
      background: #fff;
      border: 1px solid #ccc;
      border-radius: 8px;
      overflow: hidden;
      cursor: pointer;
      display: flex;
      flex-direction: column;
```

```
  justify-content: space-between;
}
.catalog-item img {
  width: 100%;
  height: auto;
}
.info {
  padding: 8px;
}
.price {
  font-weight: bold;
  color: #e91e63;
}
.author, .category {
  font-size: 14px;
  color: #555;
  margin: 3px 0;
}
.add-to-cart-btn {
  background-color: #4caf50;
  color: white;
  border: none;
  border-radius: 4px;
  padding: 6px;
  margin-top: 5px;
  cursor: pointer;
  font-size: 14px;
}
.add-to-cart-btn:disabled {
  background-color: #ccc;
  cursor: default;
}

/* Детальный просмотр */
.detail-view {
  display: none;
  margin-top: 10px;
  background-color: #fff;
  border-radius: 8px;
  border: 1px solid #ccc;
  padding: 10px;
}
.detail-cover {
  width: 100%;
  max-width: 200px;
  margin-bottom: 10px;
}
.back-btn {
  margin-top: 10px;
  background-color: #6200ee;
  color: #fff;
  border: none;
  border-radius: 4px;
  padding: 6px 12px;
  cursor: pointer;
}
/* Корзина */
.cart-list {
  list-style: none;
  padding: 0;
  margin: 0;
}
.cart-item {
  display: flex;
  justify-content: space-between;
  margin-bottom: 8px;
  background-color: #fff;
  border: 1px solid #ccc;
  padding: 6px;
```

```
      border-radius: 4px;
    }
    .cart-summary {
      margin-top: 10px;
      font-weight: bold;
    }
    .remove-btn {
      background-color: #f44336;
      color: #fff;
      border: none;
      border-radius: 4px;
      padding: 4px 8px;
      cursor: pointer;
    }

    /* Кнопка "Оплатить" */
    .pay-btn {
      margin-top: 10px;
      background-color: #ff9800;
      color: #fff;
      border: none;
      border-radius: 4px;
      padding: 8px 16px;
      cursor: pointer;
      font-size: 14px;
    }

    /* Модальное окно */
    .modal-overlay {
      position: fixed;
      top: 0; left: 0; right: 0; bottom: 0;
      background-color: rgba(0,0,0,0.5);
      display: none; /* flex, когда показываем */
      justify-content: center;
      align-items: center;
      z-index: 999;
    }
    .modal {
      background-color: #fff;
      padding: 20px;
      border-radius: 8px;
      min-width: 250px;
      text-align: center;
    }
    .modal h3 {
      margin-top: 0;
      margin-bottom: 15px;
    }
    .modal button {
      margin: 5px;
      padding: 6px 12px;
      cursor: pointer;
    }

  </style>
</head>
<body>
  <div class="header">Продвинутый магазин книг</div>

  <!-- Панель сверху (категории + сортировка) -->
  <div class="top-controls">
    <select id="category-select">
      <option value="">Все категории</option>
    </select>

    <select id="sort-select">
      <option value="none">Без сортировки</option>
      <option value="asc">Цена: по возрастанию</option>
      <option value="desc">Цена: по убыванию</option>
```

```
    </select>
  </div>

  <div class="content">
    <!-- Каталог (список книг) -->
    <div id="catalog">
      <div class="catalog-grid" id="catalog-grid"></div>
    </div>

    <!-- Детальный просмотр (заменяет каталог) -->
    <div class="detail-view" id="detail-view">
      <img id="detail-cover" class="detail-cover" src="" alt="Обложка книги">
      <div id="detail-title" style="font-weight:bold; margin-bottom:5px;"></div>
      <div id="detail-author" class="author"></div>
      <div id="detail-category" class="category"></div>
      <div id="detail-price" class="price"></div>
      <div id="detail-description" style="margin-top:10px;"></div>
      <button class="back-btn" onclick="closeDetailView()">Вернуться к каталогу</button>
    </div>

    <!-- Корзина -->
    <div id="cart">
      <ul class="cart-list" id="cart-list"></ul>
      <div class="cart-summary">
        Итого: <span id="cart-total">0</span> ₽
      </div>
      <button class="pay-btn" id="pay-btn" style="display:none;"
onclick="showPaymentModal()">Оплатить</button>
    </div>
  </div>

  <!-- Кнопки внизу (переключение каталог/корзина) -->
  <div style="display:flex; border-top:1px solid #ccc;">
    <button style="flex:1; padding:10px;" onclick="showCatalog()">Каталог</button>
    <button style="flex:1; padding:10px;" onclick="showCart()">Корзина</button>
  </div>

  <!-- Модальное окно для оплаты -->
  <div class="modal-overlay" id="modal-overlay">
    <div class="modal">
      <h3>Этот магазин создан в рамках курсового проекта, посему оплатить "ничего" у вас не
выйдет. Приносим извинения!</h3>
      <button onclick="closePaymentModal()">Понятно:)</button>
    </div>
  </div>

  <script>
    // Из URL-параметров вытащим userId (если передал бот-сервлет)
    const urlParams = new URLSearchParams(window.location.search);
    const userId = urlParams.get('userId') || 9999;

    let allBooks = []; // Для хранения загруженных книг
    let cart = [];     // [{ bookId, quantity }, ...]

    // DOM-элементы
    const categorySelect = document.getElementById('category-select');
    const sortSelect = document.getElementById('sort-select');

    const catalogSection = document.getElementById('catalog');
    const detailView = document.getElementById('detail-view');
    const catalogGrid = document.getElementById('catalog-grid');

    const cartSection = document.getElementById('cart');
    const cartList = document.getElementById('cart-list');
    const cartTotalEl = document.getElementById('cart-total');
    const payBtn = document.getElementById('pay-btn');

    // Предупреждение при закрытии
    window.addEventListener('beforeunload', (event) => {
```

```
  event.preventDefault();
  event.returnValue = 'Внесённые изменения могут быть потеряны.';
});

document.addEventListener('DOMContentLoaded', () => {
  // По умолчанию скрываем корзину и детальный просмотр
  cartSection.style.display = 'none';
  detailView.style.display = 'none';

  categorySelect.addEventListener('change', loadBooks);
  sortSelect.addEventListener('change', applySorting);

  fetchCategories();
  loadBooks();
});

// ====== ЗАГРУЗКА КАТЕГОРИЙ ======
function fetchCategories() {
  fetch('/app/categories')
    .then(res => res.json())
    .then(data => {
      data.forEach(cat => {
        const opt = document.createElement('option');
        opt.value = cat.id;
        opt.textContent = cat.name;
        categorySelect.appendChild(opt);
      });
    })
    .catch(err => console.error('Ошибка загрузки категорий:', err));
}

// ====== ЗАГРУЗКА КНИГ ======
function loadBooks() {
  let url = '/app/books';
  const catId = categorySelect.value;
  if (catId) {
    url += '?categoryId=' + catId;
  }
  fetch(url)
    .then(res => res.json())
    .then(books => {
      allBooks = books;
      applySorting();
    })
    .catch(err => console.error('Ошибка загрузки книг:', err));
}

// ====== ПРИМЕНИТЬ СОРТИРОВКУ ======
function applySorting() {
  const sortValue = sortSelect.value;
  let sorted = [...allBooks];

  if (sortValue === 'asc') {
    sorted.sort((a, b) => a.price - b.price);
  } else if (sortValue === 'desc') {
    sorted.sort((a, b) => b.price - a.price);
  }
  renderCatalog(sorted);
}

// ====== РЕНДЕР КАТАЛОГА ======
function renderCatalog(books) {
  // Скрываем детальный просмотр
  detailView.style.display = 'none';

  // Показываем сам каталог (grid)
  catalogSection.style.display = 'block';

  catalogGrid.innerHTML = '';
```

```
    books.forEach(book => {
      const itemDiv = document.createElement('div');
      itemDiv.className = 'catalog-item';
      itemDiv.addEventListener('click', () => openDetailView(book));

      const img = document.createElement('img');
      img.src = book.cover || '';
      itemDiv.appendChild(img);

      const infoDiv = document.createElement('div');
      infoDiv.className = 'info';

      // Цена
      const priceDiv = document.createElement('div');
      priceDiv.className = 'price';
      priceDiv.textContent = book.price + ' ';
      infoDiv.appendChild(priceDiv);

      // Автор
      const authorDiv = document.createElement('div');
      authorDiv.className = 'author';
      authorDiv.textContent = book.author || '';
      infoDiv.appendChild(authorDiv);

      // Категория
      const categoryDiv = document.createElement('div');
      categoryDiv.className = 'category';
      if (book.category_name) {
        categoryDiv.textContent = 'Категория: ' + book.category_name;
      }
      infoDiv.appendChild(categoryDiv);

      // Кнопка "В корзину"
      const btn = document.createElement('button');
      btn.className = 'add-to-cart-btn';
      btn.textContent = 'В корзину';
      btn.addEventListener('click', (e) => {
        e.stopPropagation();
        addToCart(book.id);
      });

      // Если уже в корзине — отключаем
      const inCart = cart.find(ci => ci.bookId === book.id);
      if (inCart) {
        btn.textContent = 'Добавлено';
        btn.disabled = true;
        btn.style.backgroundColor = '#999';
      }

      infoDiv.appendChild(btn);
      itemDiv.appendChild(infoDiv);
      catalogGrid.appendChild(itemDiv);
    });
  }

  // ====== ОТКРЫТЬ ДЕТАЛЬНЫЙ ПРОСМОТР ======
  function openDetailView(book) {
    // Скрываем каталог
    catalogSection.style.display = 'none';
    // Заполняем детальный просмотр
    detailView.style.display = 'block';

    document.getElementById('detail-cover').src = book.cover || '';
    document.getElementById('detail-title').textContent = book.title;
    document.getElementById('detail-author').textContent = book.author || '';
    document.getElementById('detail-category').textContent = book.category_name ? 'Категория: '
+ book.category_name : '';
    document.getElementById('detail-price').textContent = book.price + ' ';
    document.getElementById('detail-description').textContent = book.description || '';
```

```
}

function closeDetailView() {
  detailView.style.display = 'none';
  catalogSection.style.display = 'block';
}

// ====== ПЕРЕКЛЮЧЕНИЕ (каталог / корзина) ======
function showCatalog() {
  catalogSection.style.display = 'block';
  detailView.style.display = 'none';
  cartSection.style.display = 'none';
}
function showCart() {
  catalogSection.style.display = 'none';
  detailView.style.display = 'none';
  cartSection.style.display = 'block';
  renderCart();
}

// ====== КОРЗИНА - ADD ======
function addToCart(bookId) {
  if (cart.some(ci => ci.bookId === bookId)) return;
  cart.push({ bookId, quantity: 1 });
  // Перерисовать каталог (чтобы кнопка стала "Добавлено")
  applySorting();
  // Сохранить на сервер
  saveCartOnServer();
}

// ====== КОРЗИНА - REMOVE ======
function removeFromCart(bookId) {
  cart = cart.filter(ci => ci.bookId !== bookId);
  saveCartOnServer();
  renderCart();
  applySorting();
}

// ====== РЕНДЕР КОРЗИНЫ ======
function renderCart() {
  cartList.innerHTML = '';
  let total = 0;

  cart.forEach(ci => {
    const book = allBooks.find(b => b.id === ci.bookId);
    if (!book) return;
    total += book.price * ci.quantity;

    const li = document.createElement('li');
    li.className = 'cart-item';

    const span = document.createElement('span');
    span.textContent = `${book.title} (${book.price} ₽) x ${ci.quantity}`;
    li.appendChild(span);

    const btn = document.createElement('button');
    btn.className = 'remove-btn';
    btn.textContent = 'Убрать';
    btn.onclick = () => removeFromCart(book.id);
    li.appendChild(btn);

    cartList.appendChild(li);
  });
  cartTotalEl.textContent = total.toFixed(2);

  // Если есть товары, показываем кнопку "Оплатить"
  if (total > 0) {
    payBtn.style.display = 'inline-block';
  } else {
```

```
        payBtn.style.display = 'none';
      }
    }

    // ====== СОХРАНЕНИЕ КОРЗИНЫ НА СЕРВЕРЕ ======
    function saveCartOnServer() {
      fetch('/app/save-cart', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ userId, cart })
      })
      .then(res => res.json())
      .then(data => {
        console.log('Ответ /app/save-cart:', data);
      })
      .catch(err => console.error('Ошибка при сохранении корзины:', err));
    }

    // ====== МОДАЛКА ДЛЯ "ОПЛАТИТЬ" ======
    function showPaymentModal() {
      const overlay = document.getElementById('modal-overlay');
      overlay.style.display = 'flex';
    }

    function closePaymentModal() {
      const overlay = document.getElementById('modal-overlay');
      overlay.style.display = 'none';
    }
  </script>
</body>
</html>
```

Файл с переменными окружения .env:

```
TG_TOKEN=7645673456:yyyyyDi7BXLOVViALgCyyyyyyooDPcKyyyy
URL=https://bot27.daibot.fun
HOSTNAME=127.0.0.1
PORT=3027
PORTSSL=43027
KEY=/etc/letsencrypt/live/bot27.daibot.fun/privkey.pem
CERT=/etc/letsencrypt/live/bot27.daibot.fun/fullchain.pem
```

Шаблон нативного бота-сервлета электронного магазина готов.

Администрирование сервера. Регулирование нагрузки.

Чтобы послать скрипт-команду от лица пользователя в Linux, можно использовать следующие команды su. Если указать для команды su флаг -c, то команда su просто выполнит скрипт-команду, не переходя в новую оболочку в качестве целевого пользователя.

```
# su - user04 -c "pm2 delete 0"
```

Утилита "free". Выводит информацию о полном объёме памяти, свободной и занятой части памяти, включая swap-разделы.

```
# free -h  # top -o %MEM
```

```
              total        used        free      shared  buff/cache   available
Mem:         972076      422156      223968         748      325952      392748
Swap:             0           0           0
```

Утилита «mpstat». Для её использования нужно установить пакет sysstat для Ubuntu. Затем ввести команду «mpstat». Она покажет статистику по

использованию процессора, включая процент времени, проведённого в режиме пользователя, системного режима, простоя и ожидания ввода/вывода.

```
# apt install sysstat
# mpstat
```

```
Linux 5.4.0-198-generic (ruvds-d0ohk)   12/17/24        _x86_64_        (1 CPU)
22:17:01     CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
22:17:01     all   47.69    0.00    4.89    0.73    0.00    0.05    0.00    0.00    0.00   46.63
```

```
root@ruvds-d0ohk: ~
root@ruvds-d0ohk:/#
root@ruvds-d0ohk:~# free
              total        used        free      shared  buff/cache   available
Mem:         972076      447132      178364         748      346580      370836
Swap:             0           0           0
root@ruvds-d0ohk:~# mpstat
Linux 5.4.0-198-generic (ruvds-d0ohk)   12/17/24        _x86_64_        (1 CPU)

22:22:34     CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
22:22:34     all   47.76    0.00    4.89    0.73    0.00    0.05    0.00    0.00    0.00   46.55
root@ruvds-d0ohk:~#
```

Посмотреть конец лога с ошибками:

```
$ tail -n 20 server2-error.log
```

Запустить программу удалённо:

```
# su - user05 -c "pm2 start /home/user05/tgbot/ecosystem.config.json"
# su - user05 -c "pm2 delete 0"
```

Файл с конфигурацией проекта ecosystem.config.json:

```
{
  "apps": [
    {
      "name": "telegram-bot",
      "script": "/home/user05/tgbot/botsharp.js",
      "exec_mode": "fork",
      "instances": "1",
      "env": {
           "TG_TOKEN": "7645673456:yyyyyDi7BXLOVViALgCyyyyyyooDPcKyyyy",
           "URL": "https://bot5.daibot.fun",
           "HOSTNAME": "127.0.0.1",
           "PORT": "3005",
           "PORTSSL" : "43005",
           "KEY": "/etc/letsencrypt/live/bot5.daibot.fun/privkey.pem",
           "CERT": "/etc/letsencrypt/live/bot5.daibot.fun/fullchain.pem",
      },
      "env_testing": {
        "NODE_ENV": "developer",
        "HOST": "45.8.230.131"
      },
      "env_production": {
        "NODE_ENV": "production",
        "HOST": "localhost"
      }
    }
  ]
}
```

Программа pm2 запускает удалённо программу, но текущей директорией для проекта botsharp.js делает не папку "home/user05/tgbot", а папку "home/user05/". Поэтому, модуль bot.js содержит код с указанием полного пути к временной папке:

```
 if (msg.photo) {
    const fileId = msg.photo[msg.photo.length - 1].file_id;
    try {
      const filePath = await bot.downloadFile(fileId,
"/home/user05/tgbot/images");
      imageProcessingState[userId] = { filePath };
```

Необходимо, перед запуском pm2 изменить рабочую директорию [37]. Оператор AND (&&) в командной строке Ubuntu, выполняет вторую команду только в том случае, если первая выполнена успешно.

```
# su - user05 -c "cd ./tgbot && pm2 start /home/user05/tgbot/ecosystem.config.json"
```

Теперь всё прошло правильно.
Ошибка первенства старта веб-сервера при загрузке VPS-сервера, после перезагрузки у провайдера. Как отключить автозапуск Apache при загрузке сервера [38]:

```
# update-rc.d apache2 disable
```

Принудительное удаление службы дает указание системе удалить все ссылки инициализации службы, фактически удаляя службу со всех уровней выполнения:

```
# update-rc.d -f apache2 remove
```

Ключ -f принудительный вариант, который гарантирует, что процесс удаления будет выполнен даже при наличии несоответствий или проблем.
Чтобы отобразить список всех запущенных служб необходимо выполнить:

systemctl list-units --type=service --state=running

Утилита просмотра базы двнных SQLite Browser:

```
# apt install sqlitebrowser
```

Утилита с графической оболочкой и её пришлось удалить:

```
# apt purge sqlitebrowser
# apt clean && apt autoremove
```

Поиск больших файлов Linux. Это псевдографическая утилита, которая работает в терминале Linux.

apt install ncdu

Затем запустите утилиту, передав ей в качестве параметра папку, которую надо просканировать. Можно проверить ту же домашнюю папку:

ncdu /home

Чтобы удалить все данные из папки кэша в Linux вы можете использовать следующую команду npm cache:

```
$ npm cache clean --force
```

Чтобы убедиться, что кэш был успешно очищен, вы можете выполнить следующую команду:

```
 $ npm cache verify
```

Команда pm2 flush в контексте менеджера процессов PM2 в Linux удаляет все логи.

```
$ pm2 flush
```

Всё почистить:

```
# su - user06 -c "pm2 flush && npm cache verify && npm cache clean --force && npm cache verify"
```

Можно сократить до двух команд и вызывать для каждого пользователя только вторую команду:

```
# cmd1='pm2 flush && npm cache verify && npm cache clean --force && npm cache verify'
# su - user23 -c "$cmd1"
```

И напоследок сделать цикл в файле clr.sh:

```
#!/bin/bash
cmd1='pm2 flush && npm cache verify && npm cache clean --force && npm cache verify'
for i in {24..38}; do
    echo "clear log and cache for user$i"
    su - user$i -c "$cmd1"
done
```

Сохраните текущий список pm2 работающих сервлетов, после перезагрузки VPS-сервера они снова будут запущены:

```
# pm2 save
```

Можно остановить автозагрузку менеджера pm2:

```
# pm2 unstartup
```

Если проверите статус:

```
# pm2 status
```

То после выхода из сессии менеджер останется в памяти и его надо удалить:

```
$ pkill -u user27 -9 PM2
```

Вывод. Не оставляйте запущенным менеджер памяти pm2, если нет работающих, своих бот-сервлетов. Утилита pkill чувствительна к регистру. Наберите перед уходом из системы:

```
$ pkill -u user27 -9 PM2 && echo "Kill OK!" && pm2 unstartup
```

Учебный проект "Daibot.fun". Часть - 9а.

Задание проекта: Посфактум.

Базовая потребность: продуктовое программирование.

Постановка задачи

```
План работ:
- Управление группой пользователей на VPS-сервере.
```

Администрирование сервера.

Чтобы мониторить состояние групповой работы. Посмотрим, кто запустил менеджер pm2, с помощью команды:

```
# top -p $(pgrep -d',' -f "PM2")
```

Получим список уникальных процессов

Разберём команду pgrep.

```
$ pgrep -f "PM2"
```

Выводит список процессов соответствующих фильтру "PM2".

```
root@ruvds-d0ohk: ~
root@ruvds-d0ohk:~# pgrep -f "PM2"
16218
22916
55293
63091
67201
69540
70612
102165
root@ruvds-d0ohk:~#
```

Опция `-d','` делает список разделённый запятой:

```
root@ruvds-d0ohk: ~
root@ruvds-d0ohk:~# pgrep -d',' -f "PM2"
16218,22916,55293,63091,67201,69540,70612,102165
root@ruvds-d0ohk:~#
```

Получить список от имени пользователей для запущенных процессов PM2:

`$ ps -u -p $(pgrep -d',' -f "PM2")`

Отсортируем список пользователей по имени

`$ ps --sort=user -u -p $(pgrep -d',' -f "PM2")`

В результате получим:

```
root@ruvds-d0ohk: ~
root@ruvds-d0ohk:~# ps --sort=user -u -p $(pgrep -d',' -f "PM2")
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
user06     63091  0.0  3.0 1027580 29828 ?        Ssl  Dec27   0:02 PM2 v4.5.5: God Daemon (/home/user06/.pm2)
user10     69540  0.0  4.0 1026728 39300 ?        Ssl  Dec27   0:01 PM2 v4.5.5: God Daemon (/home/user10/.pm2)
user12     55293  0.0  3.4 1030428 33584 ?        Ssl  Dec27   0:07 PM2 v4.5.5: God Daemon (/home/user12/.pm2)
user21    102165  0.0  5.7 1030508 55668 ?        Ssl  06:58   0:08 PM2 v4.5.5: God Daemon (/home/user21/.pm2)
user25     22916  0.0  4.0 1100940 39100 ?        Ssl  Dec26   0:58 PM2 v4.5.5: God Daemon (/home/user25/.pm2)
user30     16218  0.0  4.0 1101516 39376 ?        Ssl  Dec26   1:05 PM2 v4.5.5: God Daemon (/home/user30/.pm2)
user32     70612  0.0  5.0 1034640 49380 ?        Ssl  Dec27   0:10 PM2 v4.5.5: God Daemon (/home/user32/.pm2)
user33     67201  0.0  4.1 1033824 40148 ?        Ssl  Dec27   0:19 PM2 v4.5.5: God Daemon (/home/user33/.pm2)
root@ruvds-d0ohk:~#
```

Получим список пользователей:

$ **ps** -o user= -p $(**pgrep** -d',' -f "PM2")

Можно мониторить watch, в реальном времени, указав частоту обновления одну секунду:

watch -n 1 "`ps --sort=user -u -p $(pgrep -d',' -f "PM2")`"

Получим окно процесса, где время обновления указывается в правом верхнем углу:

```
root@ruvds-d0ohk: ~
Every 3.0s: ps --sort=user -u -p 16218,22916,55293,63091,67201,69540,70612,102165    ruvds-d0ohk: Sat Dec 28 13:03:20 2024

USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
user06     63091  0.0  3.0 1027580 29828 ?       Ssl  Dec27   0:02 PM2 v4.5.5: God Daemon (/home/user06/.pm2)
user10     69540  0.0  4.0 1026728 39564 ?       Ssl  Dec27   0:01 PM2 v4.5.5: God Daemon (/home/user10/.pm2)
user12     55293  0.0  4.6 1097000 45144 ?       Ssl  Dec27   0:07 PM2 v4.5.5: God Daemon (/home/user12/.pm2)
user21    102165  0.0  5.7 1030764 56120 ?       Ssl  06:58   0:08 PM2 v4.5.5: God Daemon (/home/user21/.pm2)
user25     22916  0.0  4.0 1100940 39028 ?       Ssl  Dec26   0:58 PM2 v4.5.5: God Daemon (/home/user25/.pm2)
user30     16218  0.0  4.0 1101516 39188 ?       Ssl  Dec26   1:06 PM2 v4.5.5: God Daemon (/home/user30/.pm2)
user32     70612  0.0  4.9 1034640 48356 ?       Ssl  Dec27   0:10 PM2 v4.5.5: God Daemon (/home/user32/.pm2)
user33     67201  0.0  4.0 1033824 39584 ?       Ssl  Dec27   0:20 PM2 v4.5.5: God Daemon (/home/user33/.pm2)
```

Когда заходишь на сервер администратором, то по инерции можно набрать снова root, но уже как команду. В linux куда поместить скрипт .sh, чтобы он исполнялся из любой папки. Создадим исполняемый скрипт root, без расширения, в корне /bin и не забудем сделать исполняемым chmod a+x root:

```
#!/bin/bash
if [ -z "$1" ]; then
    files=$(pgrep -d',' -f "PM2")
    if [ -z "$files" ]; then
      watch -n 3 "echo 'No run process PM2.\n\nActive users list:' && who"
    else
      watch -n 3 "ps --sort=user -u -p $(pgrep -d',' -f "PM2") &&  echo '\nActive users' && who"
    fi
else
    watch -n 3 "ps --sort=user -u -p $(pgrep -d',' -f "$1") &&  echo '\nActive users' && who"
fi
```

Чтобы передать параметр в скрипт Bash в Linux, можно передать его сразу после имени скрипта при выполнении команды интерпретатора bash. Внутри скрипта к переданным параметрам можно получить доступ с помощью специальных переменных $1 – первый параметр. Будем вводить имена программ, например PM2, node, pts, которые часто запущены на VPS-сервере.
Получим прекрасный монитор активности пользователей:

```
root@ruvds-d0ohk: /bin
Every 3.0s: ps --sort=user -u -p 16218,22916,55293,63091,6...  ruvds-d0ohk: Sat Dec 28 13:24:06 2024

USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
user06     63091  0.0  2.9 1027580 28924 ?       Ssl  Dec27   0:02 PM2 v4.5.5: God Daemon (/home/use
user10     69540  0.0  4.0 1026728 39500 ?       Ssl  Dec27   0:01 PM2 v4.5.5: God Daemon (/home/use
user12     55293  0.0  4.5 1097000 44620 ?       Ssl  Dec27   0:07 PM2 v4.5.5: God Daemon (/home/use
user21    102165  0.0  5.7 1030764 55488 ?       Ssl  06:58   0:09 PM2 v4.5.5: God Daemon (/home/use
user25     22916  0.0  3.9 1100940 38828 ?       Ssl  Dec26   0:59 PM2 v4.5.5: God Daemon (/home/use
user30     16218  0.0  4.0 1101516 39412 ?       Ssl  Dec26   1:06 PM2 v4.5.5: God Daemon (/home/use
user32     70612  0.0  4.9 1034640 48292 ?       Ssl  Dec27   0:10 PM2 v4.5.5: God Daemon (/home/use
user33     67201  0.0  4.0 1033824 38988 ?       Ssl  Dec27   0:20 PM2 v4.5.5: God Daemon (/home/use
Active users
root     pts/0        2024-12-28 11:43 (193.23.58.119)
```

Если запустить скрипт root с параметром, то выведет процессы для программы:

```
root@ruvds-d0ohk: /bin
Every 3.0s: ps --sort=user -u -p 21821,24435,102181,108724,118796 &&...  ruvds-d0ohk: Sat Dec 28 13:57:56 2024

USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root      118796  0.0  0.2  11216  2908 pts/0    S+   13:57   0:00 /bin/bash /usr/bin/root node
user21    102181  0.2  7.7 11727932 75216 ?      Ssl  06:58   0:52 node /home/user21/tgbot2.0/public/bot.js
user25     24435  0.2  4.2 11609016 40908 ?      Ssl  Dec26   5:28 node /var/www/bot25.daibot.fun/main.js
user30     21821  0.2  4.5 1240808 44616 ?       Ssl  Dec26   5:40 node /home/user30/app/server.js
user33    108724  0.2  8.7 22211524 85484 ?      Ssl  10:05   0:30 node /home/user33/app/bot_server.mjs
Active users
root     pts/0        2024-12-28 11:43 (193.23.58.119)
```

Необходимо для пользователя, перед запуском pm2, изменить рабочую директорию [39]. Оператор:

```
# su - user26 -c "cd /var/www/bot26.daibot.fun/tov_kursovoi && pm2 start server.js"
# su - user34 -c "cd ./app && pm2 start serv.mjs"
# su - user25 -c "pm2 list"
# su - user25 -c "pm2 delete 0"
# pkill -u user25 -9 PM2
```

Сноски

[1] Telegram WebApps. Как встроить веб-приложения в чат-бота? https://habr.com/ru/articles/706446/

[2] Самописный сертификат https://core.telegram.org/bots/self-signed

[3] В зоне .fun могут размещаться сайты со смешными видео, историями, демотиваторами и шутками, страницы юмористических шоу и комиков, порталы с онлайн-играми, тестами, головоломками, проекты с контентом для детей, сервисы для прослушивания музыки и подкастов, страницы развлекательных заведений. Домен .fun был введён ICANN в 2016 году и стал доступен для широкой регистрации с 5 апреля 2017 года.

[4] Подключение к VPS Linux через SSH. https://ruvds.com/ru/helpcenter/ssh/

[5] SSH-клиент PuTTY http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

[6] Основные linux-команды для новичка https://habr.com/ru/articles/501442/

[7] Как установить mc в Ubuntu 20.04 https://losst.pro/kak-ustanovit-mc-v-ubuntu-18-04

[8] NodeJS Windows 7 64-bit Installer: https://nodejs.org/dist/v13.14.0/node-v13.14.0-x64.msi

NodeJS Windows 8 32-bit Installer: https://nodejs.org/dist/v14.16.0/node-v14.16.0-x86.msi

NodeJS Windows 8 64-bit Installer: https://nodejs.org/dist/v14.16.0/node-v14.16.0-x64.msi

[9] Утилита node-gyp - Node.js native addon build tool https://github.com/nodejs/node-gyp#on-windows

[10] Как включить протокол TLS 1.2 в Windows 7? https://winitpro.ru/index.php/2022/04/19/vklyuchit-protokol-tls-1-2-windows/

[11] Защищайся! Простые и не очень правила настройки безопасности для VPS/VDS на Linux https://habr.com/ru/companies/adminvps/articles/702912/

[12] Учимся управлять автозагрузкой в linux. https://habr.com/ru/sandbox/112588/

[13] Как подключиться через PuTTY с использованием SSH-ключей https://ruvds.com/ru/helpcenter/kak-podklyuchitsya-cherez-putty/

[14] Настройка автозапуска скриптов в Linux Ubuntu/Mint https://www.oslogic.ru/knowledge/970/nastrojka-avtozapuska-skriptov-v-linux-ubuntu-mint/?ysclid=m2kfod4pbw234329316

[15] SSH против SSL: основные различия и способы использования. https://www.ssldragon.com/ru/blog/ssh-vs-ssl/?ysclid=m2kl7sj3z0775888705

[16] How to Install PM2 to Run Node.js Apps on Production Server. https://www.tecmint.com/install-pm2-to-run-nodejs-apps-on-linux-server/

[17] Что такое DNS? Объясняем простыми словами https://www.nic.ru/info/blog/dns/

[18] Чек лист: Как настроить виртуальный сервер VPS/VDS. https://wiki.dieg.info/cheklist_nastrojka_vps_vds_vydelennogo_servera_linux

[19] Получение бесплатного SSL сертификата Let's Encrypt https://www.dmosk.ru/miniinstruktions.php?mini=get-letsencrypt

[20] Как обезопасить Nginx с Let’s Encrypt на Ubuntu 20.04 https://ruvds.com/ru/helpcenter/kak-obezopasit-nginx-s-lets-encrypt-ubuntu-20-04/

[21] Размещение телеграмм бота (Aiogram3) на VPS/VDS https://habr.com/ru/articles/783398/

[22] Как запустить JavaScript с помощью Node.js https://htmlacademy.ru/blog/js/how-start-javascript-on-nodejs?ysclid=m3936rlt32336702475

[23] Отличие Polling и Webhook в Telegram-ботах https://timeweb.cloud/tutorials/nodejs/otlichie-polling-i-webhook-v-telegram-botah?ysclid=m398palqtb926417382

[24] Yaml to JSON in Node.js – JSON to Yaml example https://hayageek.com/yaml-to-json-in-node-js-json-to-yaml-example/

[25] sharp https://sharp.pixelplumbing.com/

[26] Telegram bot handler for basic image and caption manipulation with Jimp library.https://github.com/erkendrick/jimpBot?ysclid=m3kni1hr35148989993

[27] Телеграм-боты на NodeJS https://habr.com/ru/articles/740796/

[28] Бесплатный конструктор магазина в Телеграм https://webbot.shop/

[29] Чит-код для предпринимателей: как чат-боты в Telegram помогают увеличивать прибыль на несколько миллионов https://vc.ru/marketing/1589223-chit-kod-dlya-predprinimatelei-kak-chat-boty-v-telegram-pomogayut-uvelichivat-pribyl-na-neskolko-millionov?ysclid=m3olw7fec2665743053

[30] Телеграм-боты на NodeJS https://habr.com/ru/articles/740796/

[31] Online GIF maker and image editor https://ezgif.com/maker

[32] Telegram Mini https://docs.cremax.ru/recepty/telegram-mini-apps

[33] Как убить все процессы пользователя в Linux https://losst.pro/kak-ubit-vse-protsessy-polzovatelya-v-linux

[34] How To Check For Listening Ports In Linux (Ports In Use) https://www.usessionbuddy.com/post/check-listening-ports-on-linux/

[35] DB Browser for SQLite https://sqlitebrowser.org/dl/

[36] Обзор средств для администрирования БД SQLite https://mattweb.ru/moj-blog/bazy-dannykh/item/44-obzor-sredstv-dlya-administrirovaniya-baz-dannykh-sqlite

[37] КОМАНДНАЯ СТРОКА LINUX, ПОВЫШЕНИЕ ПРИВИЛЕГИЙ: КОМАНДЫ SU, SUDO https://r4ven.me/it-razdel/zametki/komandnaya-stroka-linux-povyshenie-privilegij-komandy-su-sudo/

[38] Breaking Free from Terminal Hurdles: Your Ultimate Guide https://commandmasters.com/

[39] КОМАНДНАЯ СТРОКА LINUX, ПОВЫШЕНИЕ ПРИВИЛЕГИЙ: КОМАНДЫ SU, SUDO https://r4ven.me/it-razdel/zametki/komandnaya-stroka-linux-povyshenie-privilegij-komandy-su-sudo/