МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ
«ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Освітньо-професійна програма

«Комп'ютерна інженерія

Спеціальність 123

«Комп'ютерна інженерія»

Група: 2БКС-27

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

студента денного відділення
БКС 27.12.000 КРБ

КИР'ЯЗОВА ОЛЕГА ВІКТОРОВИЧА

м. Одеса
2023 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «Одеський технічний фаховий коледж ОНТУ»

Освітньо-професійна програма
«Комп'ютерна інженерія»
Спеціальність 123
«Комп'ютерна інженерія»
Група БКС-27

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи бакалавра на тему: **«Створення чат боту для моніторингу стану компонентів серверу на базі месенджера Telegram»**

Проектний матеріал складається з пояснювальної записки на 75 сторінках та мультимедійної презентації на 18 аркушах (слайдах)

Здобувач освіти _____ (Кир'язов О.В.)

Керівник роботи _____ (Скорнякова О.В.)

Консультанти:

з охорони праці _____ (Чорновол Н.І.)

за дотриманням вимог ЄСКД _____ (Петрашова В.І.)

старший консультант _____ (Кривченко Ю.В.)

До захисту допущений

Завідувач кафедри _____ (Іванова Л.В.)

Завідуючий відділенням _____ (Скорнякова О.В.)

Захист «20» 06 2023 р. Протокол ДКК № 1

Оцінка ДКК 5 (відмінно)

Секретар ДКК _____

АНОТАЦІЯ

Месенджери з моменту їх створення проникають в наше життя дуже стрімко. Це відбувається тому, що вони дозволяють швидко отримувати та надавати інформацію. Почали з'являтися чат-боти, які дозволяють зекономити час на пошук потрібного запиту. Одним із таких видів чат-ботів являє собою TelegramBot.

Метою даної роботи є дослідження на предмет використання та створення чат-ботів для повсякденного користування. Кваліфікаційна робота присвячена розробці чат-бота для моніторингу стану компонентів сервера на базі месенджера Telegram. У роботі буде представлено аналіз наявних рішень для моніторингу сервера, а також проаналізовано переваги та недоліки використання месенджера Telegram для створення чат-бота. У роботі буде запропоновано архітектуру чат-бота, а також будуть представлені приклади його реалізації.

Перевагами розробки бота в телеграмі є те, що телеграм має велику спільноту розробників, які готові допомогти та поділитися досвідом. Для розробки ботів у Телеграмі доступний великий вибір інструментів і бібліотек, які спрощують процес розробки. Розгортання бота в Телеграмі відбувається дуже швидко і не вимагає великих витрат часу і ресурсів. Боти в Телеграмі можуть бути використані для монетизації бізнесу, що робить їх привабливими для програмістів.

Розроблене програмне забезпечення для запуску телеграм-бота для моніторингу та отримання даних про стан апаратного забезпечення в даний момент часу. Розглянуто питання з охорони праці та техніки безпеки.

Ключові слова: Telegram, чат-бот, месенджер, система моніторингу, тестування, Pyrogram.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «Одеський технічний фаховий коледж ОНТУ»

Відділення *Комп'ютерних систем* Кафедра *Комп'ютерної інженерії*

Освітньо-професійна програма *«Комп'ютерна інженерія»*

Спеціальність *123 «Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР *Беркань І.В.*

"_____" _____ 20____ р.

ЗАВДАННЯ

на кваліфікаційну роботу бакалавра

здобувачу освіти *Кир'язову Олегу Вікторовичу*

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Створення чат боту для моніторингу стану компонентів серверу на базі месенджера Telegram*

затверджена наказом по коледжу від *17 жовтня 2022 р. № 235-А2-ОД*

2. Термін здачі студентом кваліфікаційної роботи _____

3. Вихідні дані до роботи *Бібліотека для телеграм-ботів Pyrogram. Операційна система Linux. BotAPI та MTProto.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

1. Технологічний розділ. 2. Охорона праці. Висновки. Список використаних джерел. Додатки

5. Перелік графічного матеріалу (слайдів мультимедійної презентації)

Слайд 1 – Титульний. Слайд 2 – Вступ. Слайд 3 – Платформа для проєкту. Слайд 4 – Мова програмування. Слайд 5 – Бібліотека Pyrogram. Слайд 6 – Порівняння BotAPI та MTProto. Слайд 7 – Вимоги до бота. Слайд 8 – Переваги модульної структури. Слайд 9 – Будова бота. Слайд 10 – Будова базового модуля. Слайд 11 – Вимоги до модуля моніторингу стану. Слайд 12 – Використані бібліотеки. Слайд 13 – Фрагмент коду моніторингу та надсилання пуш-повідомлення. Слайд 14 – Приклад повідомлення на телефон. Слайд 15 – Процес перекладу рядків іншими мовами. Слайд 16 – Висновки. Слайд 17 – Дякую за увагу.

6. Консультанти по кваліфікаційній роботі, із зазначенням розділів, що стосуються їх

Розділ	Консультант	ПІДПИС	
		Завдання видав	Завдання прийняв
Основний	*Скорнякова О.В.*		
Охорона праці	*Чорновол Н.І.*		
Нормоконтроль	*Петрашова В.І.*		
Старший консультант	*Кривченко Ю.В.*		

7. Дата видачі завдання 1 мая 2023

Керівник роботи *Скорнякова О.В.* _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

Пор. №	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	*Аналіз предметної галузі*	*20.02.2023*	Виконано
2.	*Аналіз технічного завдання та пошук літератури*	*01.03.2023*	Виконано
1.	*Аналітичний огляд. Огляд існуючих рішень*	*20.03.2023*	Виконано
4.	*Вивчення методів створення модульної структури*	*10.04.2023*	Виконано
5.	*Обґрунтування створення модульної структури*	*17.04.2023*	Виконано
6.	*Моделювання модульної структури бота*	*01.05.2023*	Виконано
7.	*Створення модулів для моніторингу стану сервера*	*01.05.2023*	Виконано
8.	*Розробка питань з охорони праці*	*15.05.2023*	Виконано
9.	*Виконання графічної частини дипломного проекту*	*22.05.2023*	Виконано
10.	*Підготовка до попереднього захисту, підготовка до*	*01.06.2023*	Виконано
.	*захисту*		
11.	*Підготовка доповіді та презентації для захисту*	*10.06.2023*	Виконано
12.	*Отримання рецензії, відповіді на зауваження*	*до 10.06.2023*	Виконано
	рецензента		
13.	*Захист роботи*	*до 30.06.2023*	Виконано

Здобувач освіти _____
(підпис)

Керівник роботи _____
(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			БКС 27. 12 000 КРБ	*Кваліфікаційна робота*		
				бакалавра		
А4			*БКС 27. 12 000 КРБ ПЗ*	*Пояснювальна записка*	*1*	

Зм.	Арк.	№ докум.	Підпис	Дата	БКС 27. 12 000 КРБ		
Розробив		Кир'язов О.В			*Створення чат боту для моніторингу стану компонентів серверу на базі месенджера Telegram*	Літ.	Аркуш
Перевірив		Скорнякова О.В.				У Д П	4
Н. Контр.		Петрашова В.І.				*ОТФК ОНТУ, гр. БКС-27*	
Затверд.		Кривченко Ю.В.					

ЗМІСТ

ВСТУП

У сучасному світі, коли комп'ютерна техніка стала невід'ємною частиною життя людини, належна робота серверного обладнання є критично важливою. З метою забезпечення безперебійної роботи серверів та повноцінної підтримки користувачів необхідно забезпечити їх моніторинг і контроль.

Один з можливих способів моніторингу стану компонентів серверу є застосування чат-бота на базі месенджера Telegram. Чат-бот зможе автоматизувати процес моніторингу, повідомляти про можливі проблеми, а також надавати корисну інформацію щодо функціонування сервера.

Цей дипломний проект присвячений розробці чат-бота для моніторингу стану компонентів серверу на базі месенджера Telegram. У процесі розробки будуть вирішені такі завдання, як розробка програмного забезпечення для моніторингу стану сервера, написання коду для чат-бота та взаємодії з ним.

Метою даного дипломного проекту є створення ефективного інструменту, який допоможе забезпечити безперебійну роботу серверів та зменшити ризик виникнення серйозних проблем у роботі компонентів. Також інструмент повинен мати можливість відправляти повідомлення про виявлені проблеми адміністраторам системи, щоб вони могли вжити додаткових заходів для їх вирішення.

У першому розділі буде розглянуто розробку програмного забезпечення для забезпечення зручного моніторингу станом усіх компонентів сервера, а також температурних датчиків, буде обрано Telegram як платформу для ботів, а також ОС GNU/Linux як програмну платформу, на якій здебільшого працює сервер.

Бот буде модульним, що дасть змогу використовувати бота не тільки для моніторингу, а так само для інших справ, наприклад автоматизації рутинних задач, обробки даних, формування звітів та іншого.

Кожен модуль бота буде виконувати певну функцію. Наприклад, модуль моніторингу буде відслідковувати стан серверів та повідомляти про можливі

проблеми. Модуль автоматизації буде виконувати рутинні задачі на серверах, такі як резервне копіювання даних, оновлення програмного забезпечення та інше. Модуль обробки даних буде займатися аналітикою даних, формуванням звітів та статистики.

Такий модульний підхід дозволить користувачам з легкістю конфігурувати бота та використовувати його лише для тих завдань, які їм потрібні. Крім того, модулярна архітектура дозволить легко додавати нові функції та модулі до бота в майбутньому.

Загалом, створення модульного бота забезпечить його універсальність та практичність для різних користувачів та сфер застосування.

В другому розділі було розглянуто питання охорони праці. В рамках цього розділу розглядались основні принципи та правила охорони праці, обов'язки працівників щодо безпеки на роботі, а також різноманітні заходи забезпечення безпеки праці в сферах діяльності. Окрім того, було зазначено значення правильної організації робочого місця, відповідної оснащеності робочих місць спеціальними засобами захисту, швидкої реакції у випадку аварій та інших негативних ситуацій, а також поширені порушення правил безпеки на роботі.

1 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

1.1 Опис основних понять і принципів роботи системи моніторингу

Система моніторингу - це комплекс програмних і апаратних засобів, який дозволяє відстежувати стан різноманітних об'єктів, що перебувають під контролем, і за необхідності вживати оперативних заходів.

1. Цілі моніторингу: Метою моніторингу є надання інформації про стан системи, що дає змогу ухвалювати оптимальні рішення з управління та забезпечення працездатності системи.

2. Компоненти моніторингу: Система моніторингу містить набір компонентів, що включають у себе агенти моніторингу, агрегатори даних, аналізатори даних, алерти і звіти.

3. Принципи роботи системи моніторингу: Система моніторингу має бути простою у використанні, надійною і масштабованою. Вона має бути налаштована таким чином, щоб надавати достатньо інформації для оцінювання стану системи, а також для ухвалення правильних рішень з управління та забезпечення працездатності системи. Передбачають регулярний збір даних, їх збереження та аналіз, підтримку зв'язку відповідальних осіб, які відповідають за функціонування системи, оперативну реакцію на виявлені проблеми та відновлення оптимального стану системи. Також важливим є регулярне оновлення інструментів моніторингу та метрик для забезпечення якісного контролю за станом системи.

Основні поняття та принципи роботи системи моніторингу містять у собі:

- Збір даних: Система моніторингу використовує різні інструменти для збору даних про стан і продуктивність системи. Ці дані можуть бути представлені у вигляді графіків, діаграм або таблиць.
- Аналіз даних: Після збору даних система моніторингу аналізує їх для визначення проблем і оптимізації продуктивності системи.
- Звіти: Система моніторингу генерує звіти, які допомагають зрозуміти поточний стан системи та вжити необхідних заходів.

- Повідомлення: Система моніторингу може надсилати повідомлення про проблеми та зміни в системі. Це дає змогу швидко реагувати на зміни та запобігати проблемам.

1.2 Платформа Unix\GNU/Linux

Платформа Unix/GNU/Linux - це операційна система, заснована на ядрі Unix і програмному забезпеченні GNU. Вона надає користувачам потужний інструментарій для роботи з файлами, додатками та мережами. Вона також підтримує багато програмного забезпечення та додатків, які можуть бути використані для розробки та підтримки додатків. Вона надає користувачам безпеку, надійність і простоту використання. Linux належить сімейству UNIX-подібних операційних систем. Linux підтримує широкий спектр програмних пакетів, компіляторів GNU C/C++, протоколів TCP/IP. Це гнучка реалізація ОС UNIX, що вільно розповсюджується під генеральною ліцензією GNU.

У своєму первісному вигляді вона була створена Лінусом Торвальдсом (Linus Torvalds) як версія ОС UNIX для IBM-сумісних персональних ЕОМ. Linux може будь-який вищезгаданий персональний комп'ютер перетворити на робочу станцію. Бізнесмени встановлюють Linux у мережах машин, використовують операційну систему для обробки даних у сфері фінансів, медицини, розподіленої обробки та телекомунікаціях.

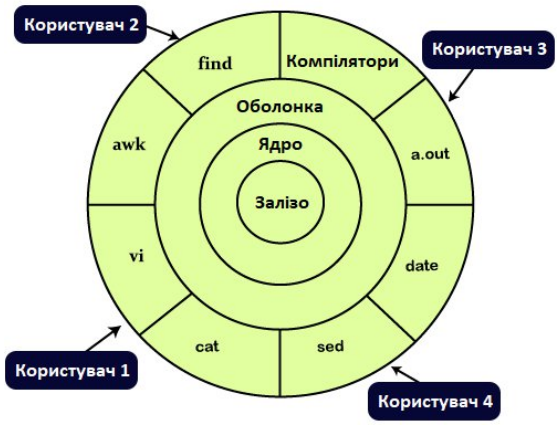


Рисунок 1.1. Архітектура Linux-систем

Ядро операційної системи можна описати як головну програму, яка виступає посередником між комп'ютерними пристроями, такими як процесор,

відеокарта та оперативна пам'ять і програмним забезпеченням. Воно "захищає" користувачів та звичайні програми від складних, низькорівневих операцій з "залізом" комп'ютера, надаючи замість цього простий та зручний для використання інтерфейс. Для досягнення цієї мети, в код ядра включені драйвери пристроїв, які можуть бути завантажені разом з ядром ОС або в момент потреби в ресурсах пристрою.

У 1991 році Лінус Торвальдс, фінський студент, надзвичайно захопився ідеєю написати сумісне з UNIX ядро операційної системи для свого персонального комп'ютера з процесором широко поширеної архітектури Intel 80386. Прототипом для майбутнього ядра стала операційна система MINIX , яка завантажувалася з дискет і вміщалася дуже обмеженої на той час пам'яті персонального комп'ютера.

MINIX був створений Ендрю Таненбаумом як навчальна операційна система, що демонструє архітектуру та можливості UNIX, але непридатною для повноцінної роботи з точки зору програміста. Саме повноцінне ядро для свого ПК і хотів зробити Лінус Торвальдс. Назва своєму ядру він дав freax, але пізніше він був змінений господарем ftp сервера на Linux - гібрид імені творця і слова UNIX.

Найважливішу роль розвитку Linux зіграли глобальні комп'ютерні мережі Usenet і Internet. На ранніх стадіях Лінус Торвальдс обговорював свою роботу і труднощі з іншими розробниками в телеконференції comp.os.minix в мережі Usenet, присвяченій операційній системі MINIX. Ключовим рішенням Лінуса стала публікація вихідних текстів малопрацездатної першої версії ядра під вільною ліцензією GNU GPL. Завдяки цьому і мережі, що отримувала все більшого поширення, Internet дуже багато отримали можливість самостійно компілювати і тестувати це ядро, брати участь в обговоренні та виправленні помилок, а також надсилати виправлення і доповнення до вихідних текстів Лінуса.

З самого початку ОС Linux розповсюджується на умовах вільно розповсюджуваного програмного забезпечення, тобто є практично

безкоштовною для користувачів (у більшості випадків для того, щоб отримати її, Ви повинні заплатити лише за CD-ROM з ПЗ або за трафік виходу в Інтернет).

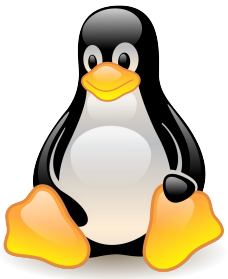


Рисунок 1.2. Логотип Linux Tux

Різні дистрибутиви Linux (дивіться визначення в наступному розділі) призначені для задоволення потреб різних типів користувачів комп'ютерів. Деякі з них орієнтовані на домашніх користувачів, інші орієнтовані на конкретні цілі, такі як мультимедійне виробництво або наукові дослідження, а інші націлені на корпоративний ринок.

Іншими словами, вибравши правильний дистрибутив, ви можете зробити з Linux так само, як і з платними операційними системами.

Дистрибутив Linux це комбінація ядра Linux, завантажувача, різних демонів, графічного сервера, робочого столу та серії програм, підібраних відповідно до потреб певного типу користувачів.

Деякі приклади дистрибутивів Linux:

- Ubuntu: Це найпопулярніший з дистрибутивів Linux, тому він має найширший асортимент програм.
- Linux Mint: ідеальний дистрибутив для початку роботи у світі Linux, оскільки він дуже простий у використанні завдяки дуже інтуїтивно зрозумілому інтерфейсу користувача.
- Fedora: Це основа Red Hat Enterprise Linux, тому рекомендовано для тих, хто хоче вивчити Linux професійно.
- ArchLinux: Для тих, хто хоче налаштувати все на свій смак, цей дистрибутив має повний посібник із встановлення.

Однак як не можна зробити операційну систему без ядра, так і ядро буде марно без утиліт, які б використали його можливості. Завдяки проекту GNU

Лінус Торвальдс одразу отримав можливість використовувати з Linux вільні утиліти: bash, компілятор gcc, tar, gzip та багато інших вже відомих та широко використовуваних програм, які могли працювати з його UNIX-сумісним ядром. Так Linux відразу потрапив у хороше оточення і в поєднанні з утилітами GNU був дуже цікавим середовищем для розробників програмного забезпечення навіть на ранній стадії свого розвитку.

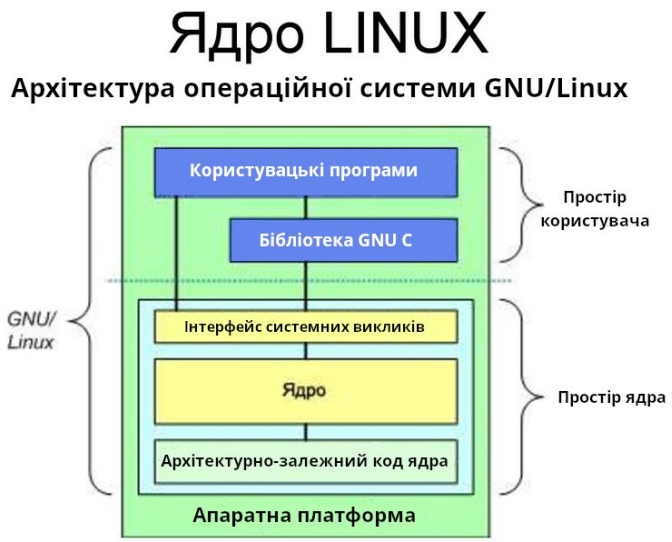


Рисунок 1.3. Архітектура операційної системи GNU/Linux

Сумісність Linux і утиліт GNU була обумовлена тим, що і те, й інше писалося з орієнтацією на ті самі стандарти і практику. Однак у рамках цієї практики (тобто за наявності безлічі різних UNIX-систем) залишався великий простір для несумісності та різних рішень. Тому на початковому етапі розробки ядра кожне запрацювало на Linux додаток GNU було для Лінуса черговим досягненням. Першими стали bash та gcc. Таким чином, поєднання GNU і Linux давало можливість створити вільну операційну систему, але саме по собі ще не становило такої системи, тому що Linux і різні утиліти GNU залишалися розрізненими програмними продуктами, написаними різними людьми, які не завжди приймали до уваги те, що робили інші . А основною властивістю будь-якої системи є узгодженість її компонентів.

Вигода операційної системи, що повністю складається з вільного програмного забезпечення, очевидна — ті, хто збирає цю систему, не повинні нікому платити за програми, що до неї входять. Більш того, подальша розробка

та оновлення наявних програм ведеться спільнотою розробників також абсолютно безкоштовно, не потрібно платити співробітникам, які б займалися цим. Завдяки Red Hat у спільноті користувачів Linux дуже широке поширення набув формату пакетів RPM.

Майже одночасно з Red Hat з'явився проект Debian. Його завдання було приблизно тим самим — зробити цілісний дистрибутив Linux та вільного програмного забезпечення GNU.

1.3 Мова програмування Python

Python - інтерпретована мова, що дозволяє заощадити значну кількість часу, що зазвичай витрачається на компіляцію. Інтерпретатор можна використовувати інтерактивно, що дозволяє експериментувати з можливостями мови, писати шаблони програм або тестувати функції при розробці “знизу-вверх”. Він також зручний як настільний калькулятор. Python дозволяє писати дуже компактні й зручні для читання програми. [1]

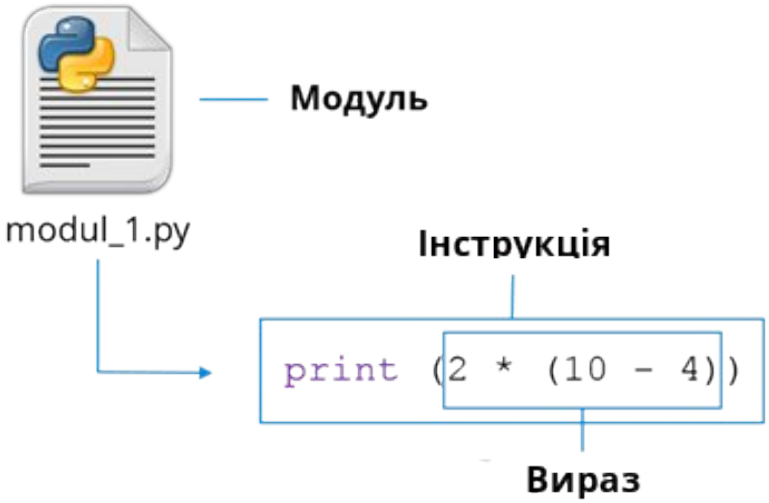


Рисунок 1.4. Приклад структури та коду

Python проста у використанні, та водночас повноцінна мова програмування, що надає набагато більше засобів для структурування і підтримки великих програм, ніж shell. З іншого боку, вона краще за C обробляє помилки, і, будучи мовою дуже високого рівня, має вбудовані типи даних високого рівня, такі як гнучкі масиви і словники, ефективна реалізація яких на C потребує значних витрат часу. [4]

Завдяки більш загальним типам даних, Python застосовують до більш широкого кола задач, ніж Awk і навіть Perl, у той ж час багато речей на мові Python робляться настільки ж просто.

Python дозволяє розбивати програми на модулі, що потім можуть бути використані в інших програмах. Python поставляється з великою бібліотекою стандартних модулів, які можна використовувати як основу для нових програм або як приклади при вивченні мови. Стандартні модулі надають засоби для роботи з файлами, системними викликами, мережними з'єднаннями і навіть інтерфейсами до різних графічних бібліотек.

Python розширювана мова: знання С дозволяє додавати нові функції, що вбудовуються, або модулі для виконання критичних операцій з максимальною швидкістю або написання інтерфейсу до комерційних бібліотек, доступним тільки у двійковій формі. Інтерпретатор мови Python може бути вбудований у програму, написану на С, і використовувати його як розширення або командну мову для цієї програми.

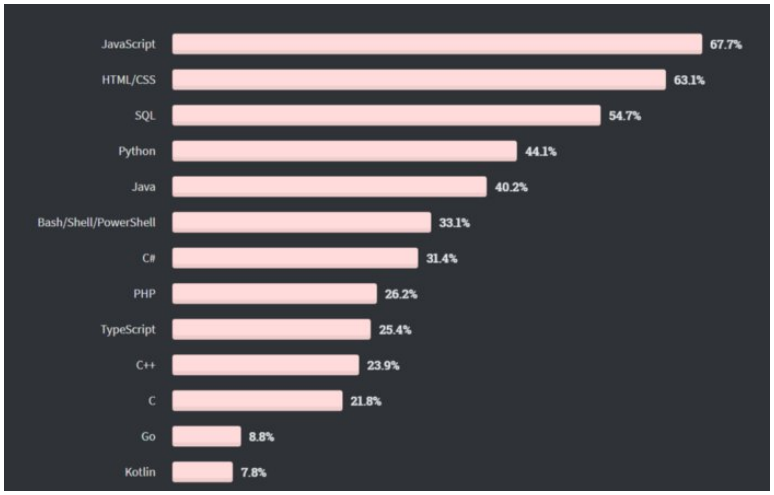


Рисунок 1.5. Порівняння мов за швидкістю роботи (Більше - Гірше)

Python використовується в даний час десятками тисяч програмістів в усьому світі, і число людей, що використовують його, швидко зростає, подвоюється і потроюється щороку. Python приваблює користувачів з ряду причин. Він використовується для розробки програм і дозволяє провести розробку набагато швидше, ніж традиційні мови типу С, С++ або Java . Ця мова працює однаково добре на Windows, Linux, Mac X, і FreeBSD, може

використовуватися, для легкої розробки як малих додатків чи сценаріїв, так і для розгортання великих програм.

Python пропонує доступ до могутнього і легкого у використанні комплекту 29 інструментальних засобів графічного інтерфейсу користувача. Традиційні машинні мови типу C і Pascal мають ряд характеристик, наприклад, сувора типізація, базові типи, складні (і звичайно довгі) цикли, і потреба у великих кількостях кодів для виконання відносно малих задач. Java досить новий, але розділяє більшість характеристик, включених у цей перелік. Програмісти, знайомі з традиційними мовами погодяться, що відсутність суворої типізації полегшує роботу з Python.

Відмінностей Python від інших мов доволі багато, перерахуємо основні з них:

1. Керування пам'яттю - цілком автоматичне — не потрібно хвилюватися щодо розподілу або звільнення пам'яті. Немає загрози “небезпечного посилання”. Java - єдина мова, що пропонує таку концепцію.
2. Типи зв'язані з об'єктами, а не зі змінними. Це означає, що змінній може бути призначене значення будь-якого типу, і що (наприклад) масив може містити об'єкти різних типів. Традиційні мови не надають такої можливості.
3. Операції звичайно виконуються в більш високому рівні абстракції. Це частково результат того, як написана мова, і частково результат розширеної стандартної бібліотеки кодів, що поставляється разом з Python.

Ці та інші особливості Python роблять розгортання додатків надзвичайно швидким. Продуктивність створеного додатку залежить від його особливостей. Звичайно, для чисельного алгоритму, що виконує звичайну арифметику цілого числа в циклі 'for', неважливо, на якій мові він написаний. Але для “середнього” додатка, збільшення продуктивності може бути просто дивовижним. Один недолік Python, у порівнянні з найбільш традиційними мовами, полягає в тому, що це - не цілком компільована мова; замість цього, вона частково транслює програму до внутрішньої форми байт-коду, і цей байт-код виконується інтерпретатором Python. Однак, у перспективі – сучасні комп'ютери мають так

багато невикористовуваного обчислювального потенціалу, що для 90% додатків швидкодія зв'язана з вибором мови. Java теж компілюється в байт-код, але в даний час працює повільніше ніж Python у більшості випадків. Крім того, дуже просто об'єднати Python з модулями, написаними на C або C++, які можна використовувати, щоб збільшити швидкість роботи програм в критичних ділянках.

1.4 Бібліотека побудови телеграм-ботів

Pyrogram - це високорівнева бібліотека Python для створення асинхронних клієнтів Telegram API. Вона написана на мові Python з використанням асинхронних фреймворків, таких як asyncio та aiohttp. Pyrogram полегшує розробку ботів та інших проектів, які працюють з Telegram API, забезпечуючи надійне з'єднання з серверами Telegram та широкий спектр можливостей. [2]

Однією з особливостей Pyrogram є те, що вона підтримує не тільки офіційний Telegram API, але й непідтримувані API та функції Telegram, такі як зображення стікерів з URL-адресами. Це означає, що Pyrogram може виконувати будь-які офіційні дії API клієнта та бота тощо. Бібліотека також містить багато функцій для створення і керування ботами, таких як створення повідомлень, обробка клавіш, підписки на повідомлення та інші.

MTProto - це назва спеціального, відкритого та зашифрованого протоколу зв'язку, створеного самим Telegram - це єдиний протокол, який використовується для обміну інформацією між клієнтом та власне серверами Telegram.

Так само, Pyrogram вміє працювати зі звичайним API для ботів, це інтерфейс для створення звичайних ботів за допомогою підмножини основного API Telegram. Боти - це спеціальні акаунти, які авторизуються за допомогою токенів, а не телефонних номерів. API ботів побудований знову ж таки на основі основного API Telegram, але працює на проміжному серверному додатку, який, у свою чергу, взаємодіє з реальними серверами Telegram за допомогою BotAPI або MTProto.

BotAPI - це програмний інтерфейс, призначений для створення та управління ботами в Telegram. З його допомогою розробники можуть створювати ботів, які надають різні функції для користувачів, такі як: відправка повідомлень, отримання повідомлень, виконання команд, обмін файлами, спілкування з іншими користувачами тощо.

BotAPI працює на основі HTTP-запитів і дозволяє отримувати доступ до всіх функцій Telegram, що дає можливість створювати повноцінних ботів з великою кількістю функцій. Боти на Telegram дуже популярні, оскільки вони можуть бути використані для різних цілей, таких як автоматизація бізнес-процесів або надання користувачам додаткових сервісів, які полегшують і поліпшують їх повсякденне життя.

Рисунок 1.6. Приклад роботи, використовуючи BotAPI

Найбільш популярні мови програмування для створення ботів на Telegram BotAPI - це Python, Node.js та PHP. Telegram боти також підтримують використання спеціальних бібліотек, що значно спрощує процес їх розробки та інтеграції з іншими сервісами.

З іншого боку, MTProto API - це те, що люди для зручності називають основним API Telegram, щоб відрізнити його від API ботів. Основний API Telegram може авторизувати як користувачів, так і ботів, і побудований на основі протоколу шифрування MTProto за допомогою двійкових даних, серіалізованих певним чином, як описано в мові TL, і доставлених за допомогою UDP, TCP або навіть HTTP в якості протоколу транспортного рівня. Клієнти, які використовують основний API Telegram, такі як Pyrogram, реалізують всі ці деталі.

Pyrogram має добре документовану структуру, що суттєво полегшує розуміння того, як вона працює і як її використовувати. Крім того, вона підтримує різні типи авторизації, такі як номер телефону або API-ключ, та має розширені функції безпеки, такі як перевірка MAC-адреси. Вона розроблена з використанням сучасних технологій, щоб забезпечити швидку роботу і можливості, які потрібен в сучасних месенджерах. Pyrogram дозволяє розробникам робити ботів як для одного, так і для декількох Telegram аккаунтів одночасно. При цьому, вона має завади з технічної точки зору, які полегшують розробку. Бібліотека має відкритий код, тому розробники можуть вносити внески у розвиток цього інструменту

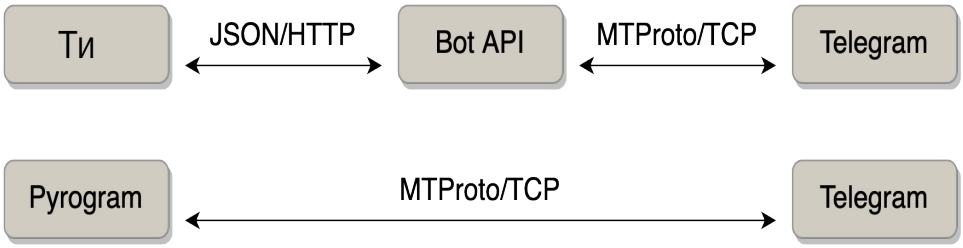


Рисунок 1.7. Взаємодія із сервером Telegram, використовуючи різні API

Узагалі, Pyrogram - це потужна та розширювана бібліотека, ідеально підходить для розробки ботів та інших проектів, що працюють з Telegram API. Вона містить багато корисних функцій та підтримує ряд важливих можливостей, що дозволяє розробникам швидко та ефективно створювати якісний та безпечний продукт.

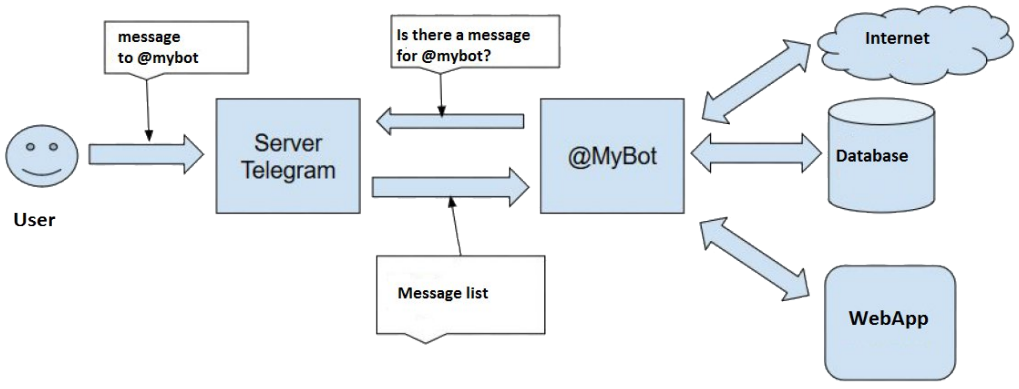


Рисунок 1.8. Приклад роботи, використовуючи BotAPI

Pyrogram має велику та активну спільноту користувачів і розробників з онлайн-підтримкою, що дозволяє швидко вирішувати будь-які питання та проблеми, що виникають в процесі розробки.

Переваги MTProto API:

Ось перелік усіх переваг використання бібліотек на основі MTProto, таких як Pyrogram, замість офіційного HTTP Bot API. Використовуючи Pyrogram, ви можете

1. Авторизувати як користувача, так і бота
2. Завантажувати та вивантажувати будь-які файли, до 2 ГБ кожен
3. Має менше накладних витрат завдяки прямому підключенню до Telegram
4. Запуск декількох сесій одночасно (як для користувача, так і для бота)
5. Має набагато більш деталізовані типи та потужні методи
6. Отримувати інформацію про будь-яке повідомлення, що існує в чаті, за його ідентифікатором
7. Отримати весь список учасників чату, як публічного, так і приватного
8. Отримувати додаткові оновлення, наприклад, про зміну імені користувача
9. Має більш змістовні помилки на випадок, якщо щось пішло не так
10. Швидше отримувати оновлення версій API, а отже, і нові функції.

1.5 Розробка ядра для модульного бота

У попередніх пунктах було обрано мову Python і бібліотеку Pyrogram. Початок розробки починається зі створення репозиторію на git-сервері, наприклад github.com. Оскільки проєкт буде модульний і має на увазі, що буде кілька частин, розумно буде зробити організацію, в ньому будемо вже створювати необхідні репозиторії з кодом.

Після цього, створюємо перший репозиторій, сам бот, який буде керувати всіма модулями і слугуватиме прошарком для роботи безлічі модулів, що дасть змогу мати одного бота з купою функціоналу.

Для створення модульного бота на Pyrogram необхідно дотримуватись декількох кроків. Перш за все, необхідно створити нового бота в Telegram через

@BotFather. [3] Далі потрібно отримати токен та зберегти його, щоб мати можливість під'єднуватись до MTProto Telegram. Для розробки бота зручно використовувати VS Code, установити необхідні бібліотеки та імпортувати модуль Pyrogram за допомогою команди `pip3 install pyrogram`. Також, необхідно створити файл з конфігурацією бота та прописати у ньому token і деякі інші налаштування.

Нам слід написати файл .py, який буде керувати всіма модулями, помістимо його в директорію base з назвою module.py, для підвантаження модулів створимо файл loader.py. За допомогою цих модулів можна зробити бота більш функціональним та зручним для користувачів.

Самі модулі будемо довантажувати з директорії modules, в якому міститимуться у вигляді git-репозиторіїв, що дає нам змогу гнучко керувати станом коду, тим самим ще й дасть змогу легко оновлювати його, коли розробник зробить якісь зміни, а в разі наявності проблем, легко відкотитися на попередню зміну, що збільшує стабільність роботи бота і всіх його компонентів.

```
def load_everything(self):
        """Load all modules"""
        modules = os.listdir(path="./modules/")
        # Force core module to load first
        if "core" in modules:
            modules.remove("core")
            modules.insert(0, "core")

        for module in modules:
            if os.path.isdir(f"./modules/{module}"):
                self.load_module(module)
```

Рисунок 1.9. Підвантаження модулів із директорії

З рисунка 1.8 зрозуміло, що ми маємо Core модуль, у якому реалізовано базову роботу з модулями, конкретніше - виведення завантажених модулів, список команд, надсилання логів у разі помилок.

Насамперед довантажуємо Core, щоб не було жодних перезаписів базових команд /start, /help тощо. Далі йде робота класу модуля, який є інтерфейсом, тобто програмісту потрібно імплементувати необхідні методи, щоб ядро могло з ним повноцінно працювати і виконувати необхідний функціонал.

З параметрів init видно, що у модуля є можливість взаємодіяти з базою даних, починаючи від легких SQLite, закінчуючи MySQL, робота з ним будується на бібліотеці sqlalchemy, яка є ORM-технологією та асинхронною, що дає змогу SQL запитам працювати ефективніше і відв'язатися від конкретної реалізації SQL під кожне ядро SQL.

Далі передаємо об'єкт bot, щоб мати можливість взаємодіяти з ботом. Далі парсимо файл info.yaml, який є звичайним yaml, в якому зберігається інформація про модуль, автор, назва, версія, посилання на вихідні коди.

```python
class BaseModule(ABC):
    """
    Bot module superclass
    """

    def __init__(
        self,
        bot: Client,
        loaded_info_func: Callable,
        bot_db_session: Session,
        bot_db_engine: Engine,
    ):
        self.bot = bot
        self.__loaded_info = loaded_info_func

        # Parse info and extensions
        info_file = InfoFile.from_yaml_file("./info.yaml")
        self.module_info = info_file.info
        self.module_permissions = info_file.permissions
```

Рисунок 1.10. Клас-інтерфейс модуля

Оскільки модулів може бути багато і користувачів теж, слід реалізувати розмежування прав доступу, щоб звичайний користувач не міг використовувати адмінські команди з розряду /ban, /mute, /terminal тощо.

```
files = os.listdir("./strings/")
            self.rawS = {}
            for file in files:
                self.rawS[file.removesuffix(".yaml")] = yaml.safe_load(
                    open(f"./strings/{file}", encoding="utf-8")
                )
```

Рисунок 1.11. Робота з файлами перекладів модуля

Далі реалізовано “багатомовність”, що дає змогу легко змінювати/додавати мову для бота і модулів до нього. Зберігати їх будемо в директорії strings у yaml файлах, що дасть нам змогу використовувати утиліти для перекладу рядків іншими мовами, наприклад, Crowdin або відкритий аналог - Weblate. Для позначення яка це мова, слід використовувати міжнародний стандарт ISO 639-1 [13].

```
def register_all(self, ext: Optional = None):
        """
        Method that initiates method registering. Must be called only from loader or extension!
        """
        methods = inspect.getmembers(ext if ext else self, inspect.ismethod)
        for name, func in methods:
            if hasattr(func, "bot_cmds"):
                # Func with @command decorator
                for cmd in func.bot_cmds:
                    if command_registry.check_command(cmd):
                        self.logger.warning(
                            f"Command conflict! "
                            f"Module {self.module_info.name} tried to register command {cmd}, which is
already used! "
                            f"Skipping this command"
                        )
                    else:
                        command_registry.register_command(self.module_info.name, cmd)
                        final_filter = (
                            filters.command(cmd) & func.bot_msg_filter
                            if func.bot_msg_filter
                            else filters.command(cmd)
                        ) & filters.create(
                            self.__check_role,
                            handler=func,
                            session=self.__bot_db_session,
                        )

                        handler = MessageHandler(func, final_filter)
                        self.bot.add_handler(handler)
                        self.__handlers.append(handler)

                        if self.__auto_help is None:
                            self.__auto_help = "Available commands:\n"
                        self.__auto_help += (
                            f"/{cmd}"
                            + (f" - {func.__doc__}" if func.__doc__ else "")
                            + "\n"
                        )
```

Рисунок 1.12. Реєстрація всіх команд у модулі

Щоб ядро визначило метод як команду бота, його потрібно обернути в так званий декоратор @command, який під капотом реєструє метод як команду засобами Pyrogram, що також дає можливість легко прибрати зі списків команд, а це дає змогу видаляти модуль без перезапуску, на відміну від aiogram, який ще є бібліотекою поверх звичайного BotAPI.

Для того, щоб дізнатися список команд і опис, використовується метод help_page, який за можливості можна перезаписати. За замовчуванням, у команд немає опису, що досить незручно, якщо назва команди не відображає суть роботи. Щоб додати, можна використовувати вбудовані можливості Python щодо документації своїх методів і класів. Зробить під класом або методом написати між 6 лапок опис, що дасть змогу легко зчитати і відобразити в повідомленні користувачеві, який прочитає і зрозуміє як користуватися командою.

1.6 Створення модулів до розробленого бота

Для спрощення створення модулів нам та іншим розробникам слід створити шаблон. У GitHub є можливість представити репозиторій як шаблон, через який не вдаючись до функції Fork, взяти собі як за основу репозиторій.

Створюємо репозиторій і починаємо робити шаблон.

Рисунок 1.13. Мінімальний набір для роботи модуля

Почну опис з кінця, у main.py визначаємо клас із будь-якою назвою, який успадковується від класу BaseModule, який імплементує всі необхідні функції. Для прикладу, імплементуємо просту команду, яка буде виводити один текст.

```python
from base.module import BaseModule, command
from pyrogram.types import Message

class ModuleTemplate(BaseModule):
    # Register handler
    @command("example")
    async def example_cmd(self, _, message: Message):
        await message.reply(self.S["some"]["strings"])
```

Рисунок 1.14. Код main.py

Як видно з останнього рядка, ми вже використовуємо багатомовність бота, у класі BaseModule він автоматично підставить мову, яка налаштована в конфігурації бота, що зберігається в корені директорії бота з назвою config.yaml. За замовчуванням там стоїть англійська. Якщо переклад не повністю було виконано, буде використовуватися англійський варіант.

```yaml
info:
  name: ModuleTemplate
  author: TemplateDeveloper
  version: 0.0.1
  description: Just a module template
  src_url: https://github.com/PBModular/ModuleTemplate
```

Рисунок 1.15. Вміст info.yaml

Цей файл використовується, щоб надати користувачеві інформацію про модуль, конкретніше назву модуля, автора, версію, опис і посилання на код. Посилання дасть змогу оновлювати засобами утиліти git і в разі несподіваних проблем, відкотити на іншу версію

Файл __init__ викликається коли loader завантажує модуль, а він зі свого боку довантажує головний клас модуля.

Директорія string зберігає yaml файли з перекладом рядків. У шаблоні зберігається у вигляді такої структури [19]:

some:

strings: "Hello, World!"

Yaml має дуже читабельний формат з використанням відступів та відкритих та закритих дужок. Може бути використаний для збереження великих обсягів даних, що дає змогу легко розширювати та змінювати дані при необхідності. Дозволяє використовувати будь-яку потрібну структуру даних, таку як списки, словники, рядки та інші, що робить його дуже гнучким і універсальним.

Для перекладів використовую веб-додаток під назвою Weblate, який є відкритим і безкоштовним рішенням.

Weblate - це веб-платформа для управління перекладами програмного забезпечення. Вона дозволяє командам розробників та перекладачів працювати разом над проектами безпосередньо в браузері.

Weblate пропонує безліч функцій і можливостей, які дозволяють зручно та швидко перекладати текстові рядки, керувати різними версіями перекладів, контролювати права доступу до проекту та багато іншого.

Це універсальна платформа, яка дозволяє використовувати різні формати даних, мови програмування та інші мови програмування. Крім того, створені проекти можуть бути інтегровані з іншими сервісами, такими як GitHub, GitLab та Bitbucket.

Усі ці переваги роблять Weblate дуже ефективним і зручним інструментом для управління перекладами і для прискорення роботи розробників програмного забезпечення.

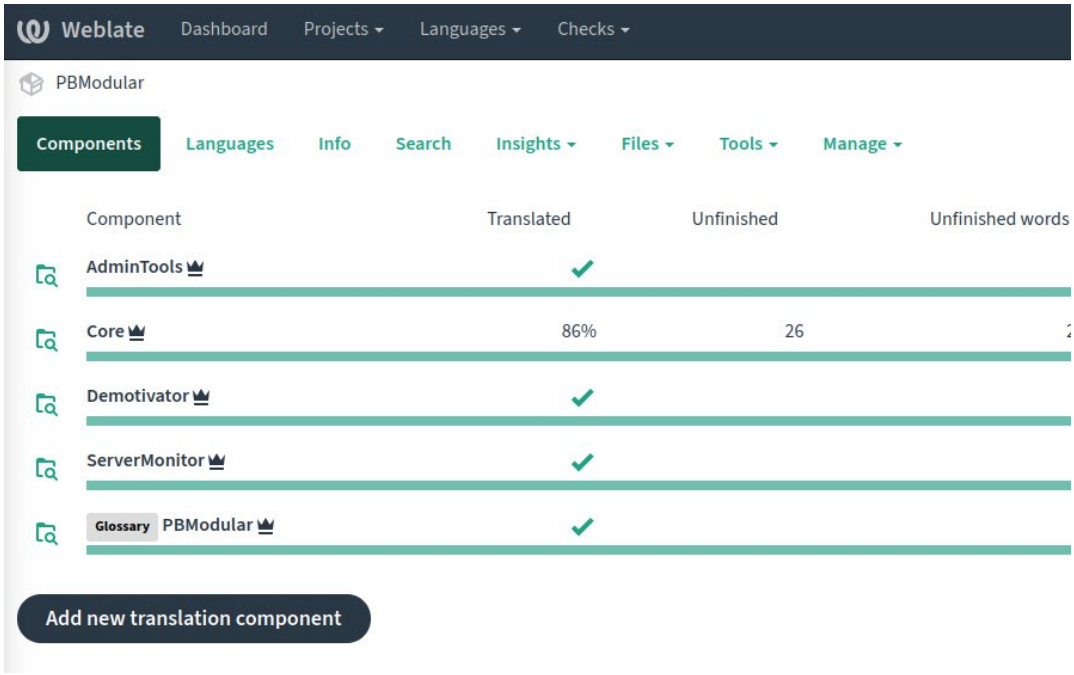


Рисунок 1.16. Власний інстанс Weblate з перекладами

Зручний інструмент для управління багатьма проектами, що вважаю досить зручно. Підтримується безліч форматів, починаючи від звичайних Android-файлів, закінчуючи Qt.

Для початку нам потрібно визначити, який функціонал нам потрібен. Як мінімум, відображати поточний стан системи, моніторити сенсори/датчики температури, вентиляторів. Так само, щоб швидко повідомити адміна про те, що занадто висока температура, слід надіслати йому повідомлення на телефон. Для цього буду використовувати сервіс ntfy.sh, безкоштовна і відкрита альтернатива Pushover.

Використовуємо шаблон, щоб не починати писати все з нуля. Щоб відв'язатися від платформи і надалі в майбутньому переписати так, щоб модуль можна було використовувати на інших системах, відмінних від Linux/FreeBSD/Unix, будемо використовувати бібліотеки platform, cpuinfo, psutil.

Для роботи з ntfy.sh можна використовувати звичайний GET-запит, що підтримується aiohttp/requests, думаю буде зайвим використовувати окрему бібліотеку для цього.

Спочатку напишемо команду, щоб дізнатися, на якій системі встановлено бота. У Linux є кілька корисних текстових файлів, у яких зберігаються посилання на Wiki, документацію, Баг-трекер, а також посилання на підтримку. Так само додамо виведення часу, скільки сервер працює.

```
@command("osrelease")
    async def osrelease_cmd(self, bot: Client, message: Message):
        string = f"""📦  <b>{self.S["os"]["os_releases"]}:</b>
        <b>{self.S["os"]["pretty_name"]}:</b> {os_release["PRETTY_NAME"]}
        <b>{self.S["cpu"]["name"]}:</b> {os_release["NAME"]}
        <b>{self.S["os"]["version"]}:</b> {os_release.get("VERSION", self.S["os"]["not_available"])}
        <b>{self.S["os"]["documentation"]}:</b> {os_release.get("DOCUMENTATION_URL", self.S["os"]
["not_available"])}
        <b>{self.S["os"]["support"]}:</b> {os_release.get("SUPPORT_URL", "Not available")}
        <b>{self.S["os"]["bug_report"]}:</b> {os_release["BUG_REPORT_URL"]}
            """

        await message.reply(
                string,
                quote=True
            )
```

Рисунок 1.17. Імплементація команди osrelease

```
@command("distro")
    async def distro_cmd(self, bot: Client, message: Message):
        string = f"""🐧  <b>{self.S['os']['linux_info']}</b>
        <b>{self.S['cpu']['name']}:</b> {get_distro()}
        <b>{self.S['os']['kernel']}:</b> {platform.release()}
        <b>{self.S['os']['hostname']}:</b> {platform.node()}
        <b>{self.S['os']['boot_time']}:</b>
{datetime.datetime.fromtimestamp(psutil.boot_time()).strftime(
            "%Y-%m-%d %H:%M:%S"
        )}
        {f'<b>{self.S["os"]["glibc_ver"]}:</b> {platform.glibc()[1]}' if hasattr(platform, 'glibc')
else ''}
        """

        await message.reply(
                string,
                quote=True
            )
```

Рисунок 1.18. Імплементація команди distro

Важливо знати, які інструкції підтримує твій сервер, наприклад для AI вже слід мати інструкцію AVX, тому напишемо команду, що виводить базову інформацію про процесор. А також споживання пам'яті на поточний момент.

```
@command("cpu")
    async def cpu_cmd(self, bot: Client, message: Message):
        string = f"🧠  <b>{self.S['cpu']['cpu_info']}</b>\n"
        string += f"• <b>{self.S['cpu']['name']}</b>: {cpuinfo.get_cpu_info().get('brand_raw',
self.S['cpu']['undetermined'])} ({cpuinfo.get_cpu_info()['arch_string_raw']})\n"
        string += f"• <b>{self.S['cpu']['count']}</b>: {psutil.cpu_count(logical=False)}
({psutil.cpu_count()})\n"
        string += (f"• <b>Freq</b>: {psutil.cpu_freq[0]} (max: {psutil.cpu_freq[2]} / min:
{psutil.cpu_freq[1]})\n")
        string += f"• <b>{self.S['cpu']['flags']}</b>: {' '.join(cpuinfo.get_cpu_info().get('flags',
self.S['cpu']['no_flags']))}\n"
        string += (
            f"• <b>Load avg</b>: {psutil.loadavg[0]} {psutil.loadavg[1]} {psutil.loadavg[2]}\n"
            if hasattr(psutil, "loadavg")
            else ""
        )

        await message.reply(
                string,
                quote=True
            )
```

Рисунок 1.19. Імплементація команди cpu

```
@command("ram")
    async def ram_cmd(self, bot: Client, message: Message):
        string = f"🀫  <b>{self.S['memory']['memory_info']}</b>\n"
        string += f"<b>{self.S['memory']['ram']}</b>:  |{progressbar(psutil.virtual_memory().percent,
10)} <code>({bytes2human(psutil.virtual_memory().used)}/{bytes2human(psutil.virtual_memory().total)})
</code>|\n"
        string += f"<b>{self.S['memory']['swap']}</b>: |{progressbar(psutil.swap_memory().percent, 10)}
<code>({bytes2human(psutil.swap_memory().used)}/{bytes2human(psutil.swap_memory().total)})</code>|\n"

        await message.reply(
                string,
                quote=True
            )

    @command("rom")
    async def rom_cmd(self, bot: Client, message: Message):
        string = f"💽  <b>{self.S['memory']['disk_info']}</b>\n"
        for disk in psutil.disk_partitions():
            disk_usage = psutil.disk_usage(disk.mountpoint)

            string += f"<b>{disk.device}</b>\n"
            string += f"├── <b>{self.S['memory']['diskMountpint']}</b> {disk.mountpoint}\n"
            string += f"├── <b>{self.S['memory']['diskFileSystem']}</b> {disk.fstype}\n"
            string += f"├── <b>{self.S['memory']['diskUsage']}</b> {disk_usage.percent}%
({bytes2human(disk_usage.used)}/{bytes2human(disk_usage.total)})\n"
            string += f"│       └─{progressbar(disk_usage.percent, 10)}\n"
            string += f"└── <b>{self.S['memory']['diskOptions']}</b> {disk.opts}\n\n"

        await message.reply(
                string,
                quote=True
            )
```

Рисунок 1.20. Імплементація команди ram і rom

Команда /rom надає список розділів і відображає інформацію про них, а конкретніше - сам пристрій, файлову систему, в яку директорію змонтовано, розмір і скільки важить на даний момент, а також найважливіше - опції, з якими

було змонтовано диск. Може бути така ситуація, коли диск не зміг змонтуватися для запису і він тільки для читання, можна буде легко дізнатися про це.

Так само слід зауважити, що тексту буде дуже багато, якщо система стоїть на Ubuntu, де використовується свій формат пакетів snap, які є контейнерами і монтуються в корінь. Слід надалі зробити перевірку на це.

Далі слід реалізувати команду, що виводить список інтернет-інтерфейсів, а також їхні параметри, щоб можна було легко дізнатися. Потрібно дізнатися, які взагалі інтерфейси є і які в них параметри, а також адреси на них, якщо вони там є.

```python
@command("netaddr")
    async def netaddr_cmd(self, bot: Client, message: Message):
        net_if_addrs = psutil.net_if_addrs()

        string = f"🌐 <b>{self.S['network']['network_info']}</b>\n"
        string += f"<b>{self.S['network']['address']}</b>:\n"
        for interf in net_if_addrs:
            interface = net_if_addrs[interf]
            string += f"<b>{interf}</b>\n"
            for addr in interface:
                attr = [
                    a
                    for a in dir(psutil.net_if_addrs()[interf][0])
                    if not a.startswith("__")
                    and not a.startswith("_")
                    and not callable(getattr(psutil.net_if_addrs()[interf][0], a))
                ]
                string += f"{AddressFamily[getattr(addr, 'family')]}\n"
                for item in attr[:-1]:
                    string += f"├── {item}: {getattr(addr, item)}\n"
                string += f"└── {attr[-1]}: {getattr(addr, attr[-1])}\n"
            string += "\n"

        await message.reply(
            string,
            quote=True
        )
```

Рисунок 1.21. Імплементація команди netaddr

netaddr відображає інформацію про IP-адреси, мережеву маску та подібну інформацію.

netstats відображає інформацію про апаратні можливості мережевої карти, наприклад, чи є вона дублексною або напівдублексною, швидкість, параметр MTU і прапори, які показують поточний стан.

```python
@command("netstats")
async def netstats_cmd(self, bot: Client, message: Message):
    net_if_stats = psutil.net_if_stats()

    string = "🌐 <b>Network Info</b>\n" + "<b>Stats</b>:\n"
    for interf in net_if_stats:
        interface = net_if_stats[interf]
        string += f"<b>{interf}</b>\n"
        string += f"└── {interface}\n\n"


    await message.reply(
        string,
        quote=True
    )
```

Рисунок 1.22. Імплементація команди netstats

Залишається тільки одна річ, яка є найважливішою - нагрівання та його моніторинг. Ядро системи постійно зчитує датчики і записує у файли в директорії /sys, що дає змогу легко дізнаватися поточний стан цих датчиків. За рахунок бібліотек, можна обстрагуватися від цієї структури файлів і надалі портувати на інші системи, наприклад Windows без сильної переробки кодової бази.

Так само нам слід повідомити адміністратора про високу температуру через push-повідомлення на телефоні.

```python
@command("temp")
    async def temp_cmd(self, bot: Client, message: Message):
        if hasattr(psutil, "sensors_temperatures"):
            sensors_temperatures = psutil.sensors_temperatures()

            string = f"🌡 <b>{self.S['sensors']['sensors_info']}</b>\n" + f"<b>{self.S['sensors']
['temperature']}</b>:\n"
            string = "🌡 <b>Sensors Info</b>\n" + "<b>Temperature</b>:\n"
            string = f"🌡 <b>{self.S['sensors']['sensors_info']}</b>\n" + f"<b>{self.S['sensors']
['temperature']}</b>:\n"
            for sensor_name in sensors_temperatures:
                sensor = sensors_temperatures[sensor_name]
                string += f"<b>{sensor_name}</b>\n"
                for sensor_info in sensor:
                    attr = [
                        a
                        for a in dir(sensor_info)
                        if not a.startswith("__")
                        and not a.startswith("_")
                        and not callable(getattr(sensor_info, a))
                    ]
                    for item in attr[:-1]:
                        string += f"├── {item}: {getattr(sensor_info, item)}\n"
                    string += f"└── {attr[-1]}: {getattr(sensor_info, attr[-1])}\n"

            await message.reply(
                string,
                quote=True
            )
        else:
            await message.reply(
                cpu_string(),
                quote=True
            )
```

Рисунок 1.23. Імплементація команди temp

Так само слід стежити за вентиляторами, які обдувають для охолодження сервера. Але бувають такі вентилятори, які не відправляють свій стан комп'ютеру, крутячись на постійній швидкості.

```python
@command("fan")
    async def fan_cmd(self, bot: Client, message: Message):
        if not hasattr(psutil, "sensors_fans"):
            return await message.reply(
                f"❌ <b>{self.S['sensors']['no_fans']}</b>",
                quote=True
            )
        sensors_fans = psutil.sensors_fans()

        if sensors_fans == {}:
            return await message.reply(
                f"❌ <b>{self.S['sensors']['no_fans']}</b>",
                quote=True
            )

        string = f"<b>{self.S['sensors']['fans']}</b>:"
        string = "<b>Fans</b>:"
        string = f"<b>{self.S['sensors']['fans']}</b>:"
        for sensor_name in sensors_fans:
            sensor = sensors_fans[sensor_name]
            string += f"<b>{sensor_name}</b>\n"
            for sensor_info in sensor:
                attr = [
                    a
                    for a in dir(sensor_info)
                    if not a.startswith("__")
                    and not a.startswith("_")
                    and not callable(getattr(sensor_info, a))
                ]
                for item in attr[:-1]:
                    string += f"├── {item}: {getattr(sensor_info, item)}\n"
                string += f"└── {attr[-1]}: {getattr(sensor_info, attr[-1])}\n"
        await message.reply(
            string,
            quote=True
        )
```

Рисунок 1.24. Імплементація команди fan

Як видно, спочатку йде перевірка на наявність датчиків у вентиляторів, що дає змогу повідомити про це користувача, якщо він захоче подивитися стан вентиляторів.

Для того, щоб постійно в циклі перевіряти стан датчиків, слід використовувати другий потік. Для цього потрібно використовувати вбудовану в Python бібліотеку threading. За допомогою неї можна викликати вічний цикл в іншому потоці, який не сповільнюватиме роботу основної програми.

Будемо викликати в методі on_init, який викликається, коли модуль повністю готовий до роботи і проініцілізувався. Код моніторингу температурою наведено внизу.

```
def on_init(self):
        thread = Thread(target=self.temp_monitor)
        thread.daemon = True
        thread.start()

def temp_monitor(self):
    while True:
        sensors_temperatures = psutil.sensors_temperatures()
        # sensor_name - amdgpu
        for sensor_name in sensors_temperatures:
            # get temp by sensor name
            sensor = sensors_temperatures[sensor_name]

            if sensor[0].high == None:
                high = ConfigSettings.high_temp
            else:
                high = sensor[0].high

            if(sensor[0].current > high and ConfigSettings.ntfy_topic):
                requests.post(f"https://ntfy.sh/{ConfigSettings.ntfy_topic}",
                    data=f"⛑ [{sensor_name}] {sensor[0].current} / {high}".encode(encoding='utf-8'))

            # print(f"[{sensor_name}] {sensor[0].current} / {high}")

        time.sleep(60)
```

Рисунок 1.25. У циклі перевіряємо стан температури

Може бути така ситуація, коли датчик не дає всю потрібну інформацію, щоб повідомити про високу температуру для цього пристрою. За замовчуванням, високою температурою будемо вважати - 100 градусів.

Цей параметр вказується в конфігурації, який знаходиться біля main.py. Замість того, щоб робити стрес-тест, можна тимчасово вказати температуру в 30 градусів. Після цього слід чекати повідомлення на телефон.

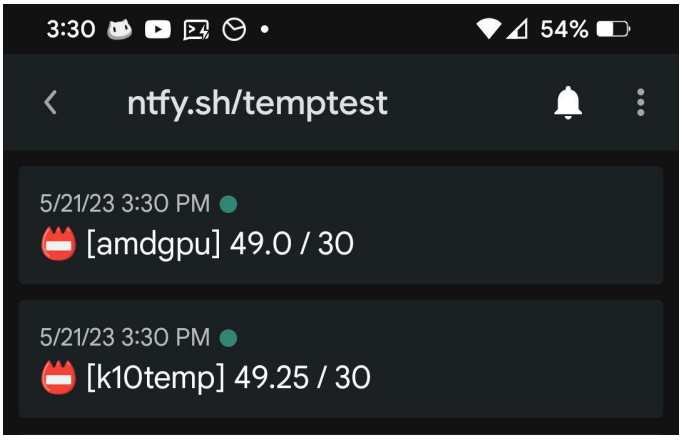


Рисунок 1.26. Повідомлення в додатку ntfy

1.7 Запуск телеграм-бота і тестування модуля

Для запуску бота, який використовує протокол MTProto, нам потрібно мати - API ID, API Hash, Token бота.

API ID та API Hash - це важливі параметри, необхідні для підключення зовнішнього програмного забезпечення до Telegram API.

API ID - це ідентифікатор, який надається при створенні застосунку на сайті розробника Telegram. Він дозволяє забезпечити безпеку та ідентифікацію застосунку на Telegram API. API ID не повинен бути розголошеним або передаватися третім особам, тому що він може дозволити отримати доступ до застосунку.

API Hash - це секретний ключ, який генерується також при створенні застосунку на сайті розробника Telegram. API Hash забезпечує безпеку вашого з'єднання між застосунком та Telegram API. Як і API ID, API Hash не повинен передаватися третім особам або розміщуватися у відкритому доступі.

Треба отримати API ID та API Hash на сторінці розробника Telegram. Для цього потрібно створити новий застосунок, вказавши назву і опис, та надати номер телефону для підтвердження вашого облікового запису. Після успішної реєстрації вам будуть надані API ID та API Hash, які ви можете використовувати для з'єднання вашого застосунку з Telegram API.

Щоб отримати Token для бота слід створити бота в Telegram, а конкретніше, відкрити Telegram і знайшов бота @BotFather. Написав "/start", щоб почати спілкування з @BotFather. Для створення нового бота треба написати команду "/newbot". Слідував інструкціям, які дав @BotFather, включаючи введення імені для бота та імені користувача. Після успішного створення бота, @BotFather поверне токен доступу до бота. Токен повинен бути збережений та захищений.

Після отримання необхідних ключів і токенів, потрібно завантажити самого бота і запустити. Усередині бота є вбудований майстер встановлення, якому потрібно написати необхідні дані.

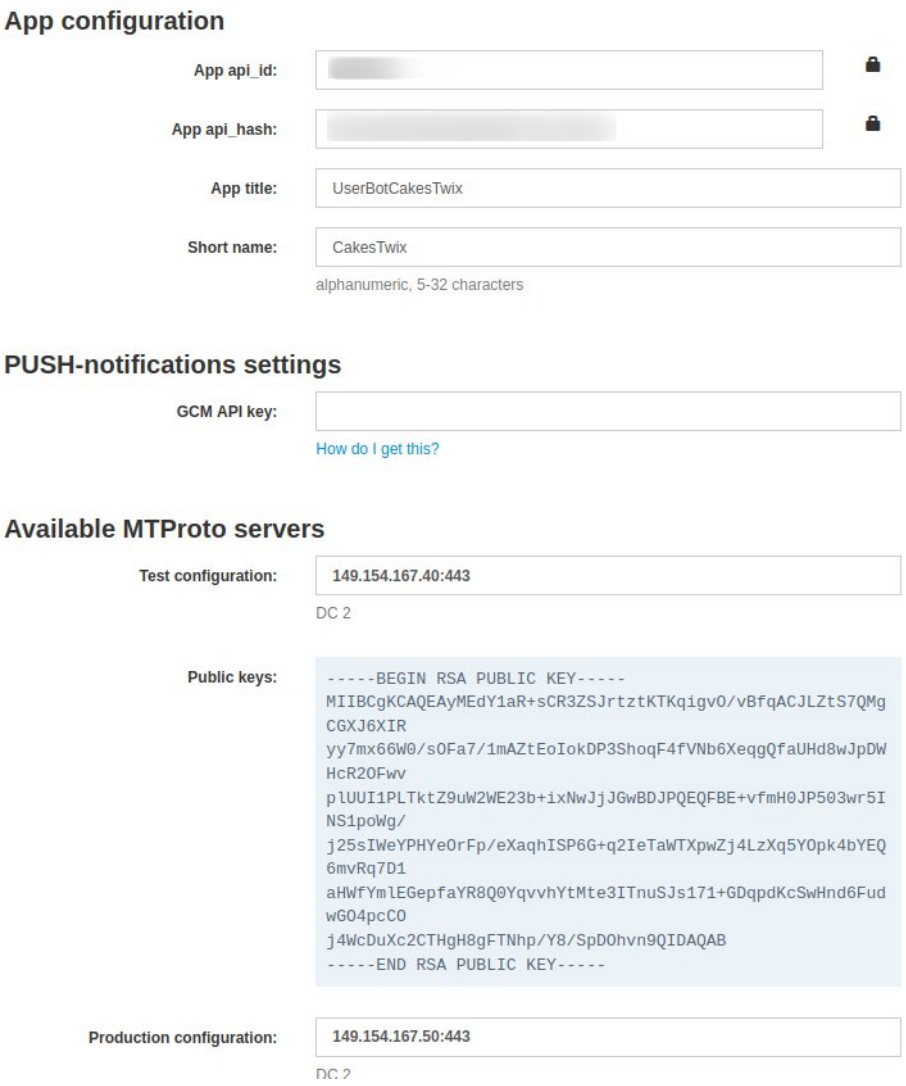


Рисунок 1.27. Сайт https://my.telegram.org/apps

Для скачування, слід використовувати утиліту git. Вкажемо, щоб завантажився в директорію PBModular.

git clone https://github.com/PBModular/bot PBModular

Щоб не було конфлікту залежностей, потрібно використовувати віртуальне оточення. Щоб його налаштувати, потрібно набрати такі команди.

python -m venv venv

source venv/bin/activate

pip install -r requirements.txt

Далі є два способи налаштування, ручний і автоматичний, засобами майстра налаштування, виберу перший.

Потрібно перейменувати config.example.yaml в config.yaml і відредагувати будь-яким текстовим редактором. Усередині потрібно вписати ключі, токен, а також ім'я або ID власника бота, щоб бот розумів, у кого є права адміністратора.

Тепер нам залишається тільки запустити бота і написати йому перше повідомлення - /start

Щоб мати можливість закрити консоль під час роботи бота, слід написати Unit-файл, у якому напишемо конфігурацію для SystemD. Він буде автоматично запускатися під час запуску ОС, а так само перезапустить його, коли вилетить сама програма.

```
[Unit]
Description=Any description of daemon here
After=network.target

[Service]
WorkingDirectory=/path/to/bot/sources
Type=simple
User=cooluser
# If you don't use venv, change python path to /usr/bin/python3 in a command below
ExecStart=/path/to/bot/sources/venv/bin/python3 -u /path/to/bot/sources/main.py
# Restart bot after fail
Restart=always
RestartSec=10

[Install]
WantedBy=multi-user.target
```

Рисунок 1.28. Конфігурація для телеграм бота

Потрібно скопіювати приклад конфігурації та змінити директорії, де знаходиться наш бот і віртуальне оточення. Скопіювати вміст потрібно в директорію /etc/systemd/system/pbmodular.service

Щоб бот автоматично запускався під час запуску ОС, потрібно прописати пару команд

sudo systemctl enable pbmodular.service

sudo systemctl start pbmodular.service

Для встановлення модулів можна використовувати команду /mod_install <url>, яка завантажить і встановить необхідні залежності до нього, якщо вони будуть.

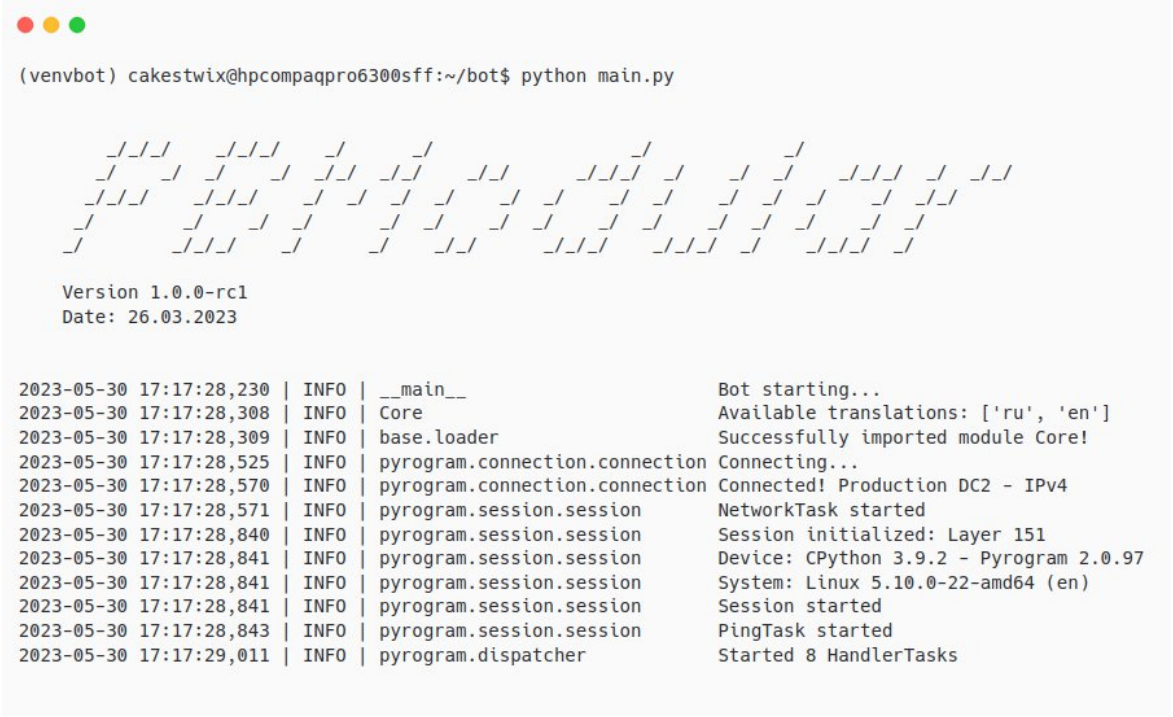


Рисунок 1.29. Успішний запуск бота

Бот повідомить, коли модуль буде встановлено таким повідомленням.

Module ServerMonitor installed!

Python dependencies for module:

- py-cpuinfo

- psutil

- requests

Usage help:

/help ServerMonitor

Для тестування коду і пошуку багів або помилок, можна використовувати вбудовану бібліотеку Python для логгування logging. У боті він уже попередньо налаштований, тому різниці між різними модулями не буде і все буде в одному стилі. Так само можна використовувати деббагер у VS Code або IDE PyCharm, що є більш професійним способом.

2 ОХОРОНА ПРАЦІ

Охорона праці - це сукупність заходів, які спрямовані на запобігання травматизму та захист здоров'я працівників на робочому місці. Кожен працівник має право на безпечні та здорові умови праці, але це не завжди можливо без виконання певних норм та правил. З метою забезпечення безпеки та охорони здоров'я працівників багато країн світу створюють законодавчу базу, яка регулює питання охорони праці на різних видовищах.

Для поліпшення умов і охорони праці необхідна ефективна співпраця між всіма рівнями державної влади та громадськості, а також реалізація програм на державному та місцевому рівнях. Реалізація цих програм може допомогти розробити систему нагляду, навчання та контролю в галузі охорони праці, адаптувати законодавство до європейських стандартів, забезпечити інформаційне та науково-методичне забезпечення і створити безпечні умови праці на підприємствах та в організаціях усіх форм власності. Все це допоможе забезпечити пріоритет життя та здоров'я працюючих та забезпечити комплексне вирішення задач охорони праці.

У світі існують різні види професій та виробництв, де вимагається дотримання особливих правил та норм, щоб забезпечити безпеку працівників. Охорона праці стала невід'ємною складовою в будь-якій сфері діяльності, де є люди. Вона дає можливість не тільки забезпечити безпеку працівників, але й ефективно вирішувати проблеми на робочому місці та знижувати ризик виникнення небезпечних ситуацій.

2.1. Аналіз та безпека умов праці працівника на робочому місці

2.1.1 Організація робочого місця

Шкідливі фактори, пов'язані з використанням комп'ютерної техніки, можуть мати негативний вплив на здоров'я користувачів. Ці фактори включають неправильну поставу, напругу очей, шум, електромагнітне випромінювання, біологічні фактори та дезорганізацію режиму роботи та відпочинку.

Неправильна постава може призвести до болей у спині, шиї, руках та зап'ястях. Розглядання екрану комп'ютера може призвести до напруження очей, що може спричинити головні болі, сухість очей та інші проблеми зі здоров'ям очей. Комп'ютери можуть видавати шум, що може викликати стрес та зниження працездатності.

Електромагнітне випромінювання може мати шкідливий вплив на здоров'я людини. Клавіатури та миші комп'ютерів можуть бути джерелом мікробів та інших патогенних організмів. Працюючи за комп'ютером, людина може забути про перерви для відпочинку, що може призвести до зниження працездатності та збільшення ризику розвитку хронічних захворювань.

Для забезпечення комфортної та безпечної праці з ВДТ, необхідно застосовувати відповідну організацію робочого місця та обладнання. Всі елементи робочого місця та їх взаємне розташування повинні відповідати ергономічним вимогам, які враховують характер і особливості трудової діяльності (згідно з ГОСТ 12.2.032-78, ГОСТ 22.269-76, ГОСТ 21.889-76).

Робочі місця з ВДТ краще розташовувати так, щоб природне світло падало збоку, переважно зліва. Для робочих столів з ВДТ слід дотримуватися відстаней: між бічними поверхнями ВДТ - 1,2 м; від тильної поверхні одного ВДТ до екрану іншого - 2,5 м. Екран ВДТ має бути розміщений на оптимальній відстані від очей користувача, що становить 600-700 мм, але не ближче, ніж за 600 мм з урахуванням розміру літерно-цифрових знаків і символів.

Клавіатуру слід розташовувати на поверхні столу на відстані 100-300 мм від краю, зверненого до працюючого. Конструкція клавіатури повинна передбачати опорний пристрій, який дає змогу змінювати кут нахилу поверхні клавіатури у межах 5-150.

При обладнанні робочого місця лазерним принтером необхідно враховувати вимоги СанПіН № 5804-91 щодо параметрів лазерного випромінювання.

2.1.2 Вимоги безпеки до мікроклімату, освітлення, шуму ,виробничих випромінювань

Основними документами, які визначають параметри мікроклімату виробничих приміщень, є ДСН 3.3.6.042-99 та ГОСТ 12.1.005-88. Ці параметри регулюються для робочої зони - визначеного простору, де знаходяться робочі місця. Згідно з нормативним документом ДСН 3.3.6.042-99 «Санітарні норми мікроклімату виробничих приміщень», параметри мікроклімату мають відповідати значенням, зазначеним у таблиці 2.1.

Таблиця 2.1. Параметри мікроклімату приміщення

Період року	Категорія робіт	Температура, С		Відносна вологість,%
		оптимальна	допустима	
Холодний	Легка – Іа	22-24	21-25	40-60
Теплий	Легка – Іа	23-25	22-26	40-60

Приміщення, в яких встановлені персональні комп'ютери, повинні мати природне та штучне освітлення. Природне освітлення здійснюється через світові прорізи (вікна), орієнтовані переважно на північ чи північний схід. Штучне освітлення в приміщені здійснюється системою загального рівномірного освітлення. На поверхні столу в зоні розміщення документів штучне освітлення має становити 300-500лк.

Так як шум має 35Дб, сприйняття шуму людським вухом межується від 20Дб до 120 дб, це означає, що при роботі за ЕОМ шум не заважає, працівнику працювати.

Для запобігання виникнення інших шумів у відповідності з ГОСТ 12.1.029-80 зниження шуму й вібрації в приміщенні дипломним проектом передбаченні звукоізоляція вікон та дверей.

Для того, щоб забезпечити здорову роботу та запобігти швидкій втомлюваності очей, професійним захворюванням, нещасним випадкам і

підвищити продуктивність праці та якість продукції, необхідно відповідно установити виробниче освітлення.

Конкретно, воно повинно створювати достатню освітленість на робочій поверхні, що буде відповідати характеру зорової роботи та встановленим нормам.

Також важливо забезпечити рівномірність та постійність рівня освітленості у всіх виробничих приміщеннях. Це допоможе уникнути частого переадаптування органів зору та зменшить ризик появи засліплювальної дії. Додатково, на робочій поверхні не повинно бути ніяких різних тіней, а також має бути достатньо контрасту між освітлюваними поверхнями. Також важливо уникати небезпечних та шкідливих виробничих чинників, таких як шум, теплове випромінювання, електрична небезпека, пожежо- та вибухонебезпека світильників.

Нарешті, виробниче освітлення має бути надійним, простим у експлуатації, економічним та естетичним.

2.2. Пожежна безпека

Під поняттям пожежна безпека розуміють систему заходів, які спрямовані на захист людей та майна від вогню. У разі приміщень з електричними мережами, дотримання пожежної безпеки регулюється стандартами ГОСТ 12.1.033-81 та ГОСТ 12.1.004-85. Крім того, для роботи оператора ЕОМ необхідно мати приміщення, яке відповідає категорії Д пожежної безпеки, тобто містить негорючі речовини та матеріали в холодному стані.

До засобів пожежогасіння належать внутрішні пожежні водопроводи (крани - ПК), вогнегасники (вуглекислотні та порошкові), сухий пісок та інші. У будівлях пожежні крани зазвичай розміщують у коридорах та на майданчиках сходових кліток. Кожен пожежний кран обладнаний пожежним рукавом, який знаходиться в спеціальному ящику на висоті 1,35 м від полу.

Для загасання пожеж на початкових етапах широко використовують вогнегасники. У виробничих приміщеннях основною формою вогнегасників є

вуглекислотні, які мають високу ефективність у гасінні пожеж, а також дозволяють зберегти електричне обладнання. Вогнегасники розміщують на видному місці на висоті не більше 1,5 м від полу.

Для вирішення евакуаційних задач у виробничих приміщеннях зазвичай є запасні виходи. Двері, які ведуть до запасних виходів, повинні мати освітлений напис "Запасний вихід". План евакуації вивішують на видному місці біля основного виходу. Приміщення, в яких встановлені персональні комп'ютери, повинні мати природне та штучне освітлення. Природне освітлення здійснюється через світові прорізи (вікна), орієнтовані переважно на північ чи північний схід. Штучне освітлення в приміщені здійснюється системою загального рівномірного освітлення. На поверхні столу в зоні розміщення документів штучне освітлення має становити 300-500лк.

У разі виникнення пожежі потрібно зупинити електропостачання, негайно повідомити за номером 101 професійну пожежну команду, дотримуючись плану евакуації, евакуювати людей з будівлі і розпочати процедуру гасіння пожежі.

ВИСНОВКИ

Для моніторингу стану компонентів серверу на базі месенджера Telegram було розроблено і реалізовано чат-бота, що дозволяє оперативно отримувати повідомлення про стан сервера та реагувати на можливі проблеми.

Під час виконання проекту були досягнуті такі результати:

1. Було вивчено процес розробки телеграм чат-бота на мові програмування Python з використанням асинхронної бібліотеки Pyrogram

2. Проаналізував переваги розробки модульного бота для користувачів. Модульний бот може бути розроблений з використанням різних модулів, що дозволяє користувачам отримувати доступ до широкого спектру послуг у рамках одного і того ж бота. Може бути налаштований під індивідуальні потреби користувачів, що дає змогу забезпечувати персоналізований досвід використання.

3. Розроблено та реалізовано чат-бот для моніторингу стану компонентів серверу на базі месенджера Telegram.

Чат-бот був успішно інтегрований з компонентами сервера та встановлений на постійну роботу. Після запуску, користувачі мали можливість отримувати повідомлення про стан компонентів сервера у реальному часі. Бот надавав інформацію про поточний стан процесора, пам'яті, дискового простору, а також інших системних параметрах, що дозволяло оперативно виявляти можливі проблеми та проводити необхідні налаштування. Крім того, бот був налаштований на автоматичне сповіщення адміністратора про критичний стан компонентів сервера. Це значно зменшувало час реакції на можливі проблеми та допомагало уникнути серйозних наслідків для роботи сервера.

У результаті розробки та імплементації чат-бота для моніторингу стану компонентів серверу користувачі отримали зручний та ефективний інструмент для контролю за роботою сервера, що дозволяло забезпечити безперебійну роботу бізнес-процесів.

Для подальшого вдосконалення спроектованого виробу рекомендується розширити функціональність чат-бота, додавши можливість виконувати певні команди для керування сервером. Також можна розглянути можливість інтеграції чат-бота зі засобами автоматизованого моніторингу серверів. Так само потрібно реалізувати фільтр розділів за точкою монтування, щоб не отримувати помилку про те, що занадто велика кількість літер у повідомленні, коли використовується багато програм у контейнерах snap.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. О.М. Васильєв. Програмування мовою Python, 2018, 12 с.
2. Документація бібліотеки Pyrogram [Електронний ресурс]. – Режим доступу до ресурсу: https://docs.pyrogram.org/ (звернення 5.01.2023).
3. Створюємо Telegram бота на Python. Частина 1[Електронний ресурс] - Режим доступу: https://codeguida.com/post/410
4. Посібник з мови програмування Python. [Електронний ресурс]. URL: https://metanit.com/python/tutorial
5. Все про чат-боти: переваги, типи та схема роботи. [Електронний ресурс]. URL: https://www.interkassa.com/blog/vse-o-chat-botah-preimushchestva-tipy-i-shema-raboty/
6. Linux: коротка історія створення. [Електронний ресурс]. URL: https://uniteddc.net.ua/news/i/linux-history/
7. Що таке Linux? Історія створення Linux. [Електронний ресурс]. URL: https://acode.com.ua/what-is-linux/#toc-2
8. Створення Telegram-бота. Основи. [Електронний ресурс]. URL: https://wibe.team/sozdanie-bota-v-telegram
9. Документація по створенню Telegram-ботів [Електронний ресурс]. URL: https://core.telegram.org/bots
10. Python is a programming language, [Електронний ресурс]. URL: https://www.python.org
11. Чат-бот для бізнесу: плюси й мінуси впровадження роботів [Електронний ресурс]. URL: https://nikopolnews.net/chat-bot-dlya-biznesu-pljusi-j-minusi-vprovadzhennya-robotiv/
12. Найповільніші сучасні мови програмування. [Електронний ресурс]. URL: https://uaspectr.com/2021/02/19/povilni-movy-programuvannya/
13. Список мовних кодів ISO 639 [Електронний ресурс]. URL: https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes

14. Використання мови Python для розробки науково-технічного програмного забезпечення [Електронний ресурс]. URL: https://dou.ua/lenta/articles/python-for-science

15. Козяр М.М., Щедрій Я.І., Станіславчук О.В. Основи охорони праці, безпеки життєдіяльності та цивільного захисту населення: Навч.посіб. -К.: Кондор, 2012

16. ГОСТ 12.2.032-78 Система стандартів безпеки праці [Електронний ресурс].URL: http://online.budstandart.com/ua/catalog/doc-page.html?id_doc=28895

17. App configuration [Електронний ресурс]. URL: https://my.telegram.org/apps

18. Санітарні норми мікроклімату виробничих приміщень ДСН 3.3.6.042-99 [Електронний ресурс]. URL: https://zakon.rada.gov.ua/rada/show/va042282-99

19. YAML Ain't Markup Language [Електронний ресурс]. URL: https://yaml.org/

20. Основи охорони праці. навчально-методичний посібник для студентів вищих закладів педагогічного напрямку / [Укладачі: В.І. Кошель,Г.П. Сав'юк, Б.С. Дзундза] – Івано-Франківськ: НАІР, 2020. 182 с.

ДОДАТОК 1. Код програми

main.py:

```
from pyrogram import Client, idle
from pyrogram.enums import ParseMode
from pyrogram.errors.exceptions.bad_request_400 import BadRequest
import logging
from base.loader import ModuleLoader
from config import config, CONF_FILE
import os
from logging.handlers import RotatingFileHandler
from colorama import init, Fore, Style
from sqlalchemy.ext.asyncio import create_async_engine, async_sessionmaker
from db import Base
init()  # initialize colorama
class ColorFormatter(logging.Formatter):
    status_colors = {
        logging.DEBUG: "",
        logging.INFO: f"{Fore.GREEN}",
        logging.WARN: f"{Fore.YELLOW}",
        logging.ERROR: f"{Fore.RED}",
        logging.CRITICAL: f"{Fore.RED}",
    }
    text_colors = {
        logging.DEBUG: "",
        logging.INFO: "",
        logging.WARN: "",
        logging.ERROR: "",
        logging.CRITICAL: "",
    }
    name_color = f"{Fore.WHITE}"
    reset = f"{Style.RESET_ALL}"
    text_spacing = "\t" * 8
    def format(self, record: logging.LogRecord) -> str:
        f = (
            f"%(asctime)s | "
            f"{self.status_colors[record.levelno]}%(levelname)s{self.reset} | "
            f"{self.name_color}%(name)s{self.reset} "
f"\r{self.text_spacing}{self.text_colors[record.levelno]}%(message)s{self.reset}"
        )
```

```
        return logging.Formatter(f).format(record)
# File/Console Logger
file_handler = RotatingFileHandler(
    filename="bot.log", maxBytes=128 * 1024
)  # 128 MB limit
stdout_handler = logging.StreamHandler()
stdout_handler.setLevel(logging.DEBUG)
stdout_handler.setFormatter(ColorFormatter())
handlers = [file_handler, stdout_handler]
logging.basicConfig(
    format="%(asctime)s | %(levelname)s | %(name)s %(message)s",
    level="INFO",
    handlers=handlers,
)
logger = logging.getLogger(__name__)
# Root path
ROOT_DIR = os.getcwd()
def main(update_conf: bool = False):
    if config.token and config.api_id and config.api_hash:
        # Try to run bot
        try:
            bot = Client(
                name="bot",
                api_id=config.api_id,
                api_hash=config.api_hash,
                bot_token=config.token,
                parse_mode=ParseMode.HTML,
            )
        # Reset token and again run main
        except BadRequest:
            config.token = None
            config.api_id = None
            config.api_hash = None
            main(update_conf=True)
        # All ok, write token to config
        if update_conf:
            config.to_yaml_file(CONF_FILE)
        logger.info("Bot starting...")
        async def start():
```

```
            # Init database
            engine = create_async_engine("sqlite+aiosqlite:///bot_db.sqlite3")
            session_maker = async_sessionmaker(engine, expire_on_commit=False)
            async with engine.begin() as conn:
                await conn.run_sync(Base.metadata.create_all)
            # Load modules
            loader = ModuleLoader(
                bot,
                root_dir=ROOT_DIR,
                bot_db_session=session_maker,
                bot_db_engine=engine,
            )
            loader.load_everything()
            # Launch bot
            await bot.start()
            await idle()
            await bot.stop()

        bot.run(start())
    else:
        config.token = input("Input token: ")
        config.api_id = int(input("Input api_id: "))
        config.api_hash = input("Input api_hash: ")
        main(update_conf=True)
if __name__ == "__main__":
    print(
        f"""
        {Fore.CYAN}
        _/_/_/   _/_/_/   _/     _/            _/        _/
       _/   _/  _/   _/  _/_/  _/_/   _/_/    _/_/_/  _/   _/  _/   _/_/_/  _/  _/_/
      _/_/_/   _/_/_/   _/  _/  _/  _/   _/  _/   _/  _/   _/  _/  _/   _/  _/_/
     _/       _/   _/  _/    _/  _/   _/  _/  _/   _/  _/  _/   _/  _/
    _/       _/_/_/   _/    _/   _/_/    _/_/_/   _/_/_/  _/   _/_/_/  _/

    Version 1.0.0-rc1
    Date: 26.03.2023
    {Style.RESET_ALL}
    """
    )
```

```
    main(update_conf=False)
```

config.py:

```
from dataclasses import dataclass
from dataclass_wizard import YAMLWizard
import os
import shutil
from typing import Union
CONF_FILE = "config.yaml"
@dataclass
class Config(YAMLWizard):
    token: str
    api_id: int
    api_hash: str
    language: str
    fallback_language: str
    update_deps_at_load: bool
    enable_db: bool
    db_url: str
    db_file_name: str
    owner: Union[int, str]
# Load from YAML
if CONF_FILE not in os.listdir("./"):
    shutil.copy("config.example.yaml", CONF_FILE)
config = Config.from_yaml_file(CONF_FILE)
```

base/base_ext.py

```
from abc import ABC, abstractmethod
from dataclasses import dataclass
from typing import Optional
@dataclass
class ExtensionInfo:
    name: str
    author: str
    version: str
    src_url: Optional[str] = None


class BaseExtension(ABC):
    """
```

```
    Class to extend ModuleLoader functionality.
    Can modify module object before stage2 initialization
    Useful for implementing features for every module
    """
    @property
    @abstractmethod
    def extension_info(self) -> ExtensionInfo:
        """
        Extension info. Must be set
        :return: ExtensionInfo dataclass object
        """
    @abstractmethod
    def on_module(self, obj):
        """
        Main method where extension must edit module object
        :param obj: BaseModule object
        """
```

base/command_registry.py

```
from typing import Optional
commands: dict[str, list[str]] = {}
def register_command(owner: str, command: str):
    if commands.get(owner) is None:
        commands[owner] = []
    commands[owner].append(command)
def get_commands(owner: str) -> list[str]:
    return commands.get(owner)
def check_command(command: str) -> bool:
    for cmds in commands.values():
        if command in cmds:
            return True
    return False
def get_command_owner(command: str) -> Optional[str]:
    for owner, cmds in commands.items():
        if command in cmds:
            return owner
    return None
def remove_all(owner: str) -> bool:
    try:
```

```
            commands.pop(owner)
            return True
        except KeyError:
            return False
```

base/db.py

```
from sqlalchemy.ext.asyncio import create_async_engine, async_sessionmaker
from config import config
import logging
import traceback
logger = logging.getLogger(__name__)
class Database:
    def __init__(self, modname: str):
        try:
            self.engine = create_async_engine(self.decide_url(modname))
            self.session_maker = async_sessionmaker(self.engine,
expire_on_commit=False)
        except:
            logger.error("Failed to initialize database! Disabling for runtime!")
            traceback.print_exc()
    @staticmethod
    def decide_url(modname: str) -> str:
        if "sqlite" in config.db_url:
            return config.db_url + f"/modules/{modname}/{config.db_file_name}"
        else:
            return config.db_url + f"/{modname}"
```

base/db_migration.py

```
from abc import ABC, abstractmethod
from sqlalchemy.orm import Session
from sqlalchemy import Engine, MetaData
class DBMigration(ABC):
    """Class for handling database migrations between module updates"""
    @abstractmethod
    def apply(self, session: Session, engine: Engine, metadata: MetaData):
        """Main method where migration happens"""
```

base/loader.py

```
from base.module import BaseModule, ModuleInfo, Permissions
from base.base_ext import BaseExtension
```

```
from base.db import Database
from base.db_migration import DBMigration
from config import config
from pyrogram import Client
import requirements
from sqlalchemy.ext.asyncio import async_sessionmaker, AsyncEngine
import asyncio
import importlib
import inspect
import logging
import os
import sys
import shutil
import subprocess
from urllib.parse import urlparse
from typing import Optional, Union
from packaging import version
logger = logging.getLogger(__name__)
class ModuleLoader:
    """
    Main module dispatcher
    Modules must be placed into modules/ directory as directories with __init__.py
    """
    def __init__(
        self,
        bot: Client,
        root_dir: str,
        bot_db_session: async_sessionmaker,
        bot_db_engine: AsyncEngine,
    ):
        self.__bot = bot
        self.__modules: dict[str, BaseModule] = {}
        self.__modules_info: dict[str, ModuleInfo] = {}
        self.__modules_deps: dict[str, list[str]] = {}
        self.__root_dir = root_dir
        self.__hash_backups: dict[str, str] = {}
        self.bot_db_session = bot_db_session
        self.bot_db_engine = bot_db_engine
        # Load extensions
```

```
    self.__extensions: dict[str, BaseExtension] = {}
    extensions = os.listdir(path="./extensions/")
    for ext in extensions:
        if not os.path.isdir(f"./extensions/{ext}"):
            continue
        try:
            imported = importlib.import_module("extensions." + ext)
        except ImportError as e:
            logger.error(f"ImportError has occurred while loading extension {ext}!")
            logger.exception(e)
            continue
        for obj_name, obj in inspect.getmembers(imported, inspect.isclass):
            if BaseExtension in inspect.getmro(obj):
                os.chdir(f"./extensions/{ext}")
                try:
                    # Check for dependencies update / install them
                    if (
                        config.update_deps_at_load
                        and "requirements.txt" in os.listdir()
                    ):
                        self.install_deps(ext, "extensions")
                    instance: BaseExtension = obj()
                    name = instance.extension_info.name
                    self.__extensions[name] = instance

                    logger.info(f"Successfully loaded extension {name}!")
                    os.chdir("../../")
                except Exception as e:
                    logger.error(
                        f"Error at loading extension {ext}! Printing traceback"
                    )
                    logger.exception(e)
                    os.chdir(self.__root_dir)
def load_everything(self):
    """Load all modules"""
    modules = os.listdir(path="./modules/")
    # Force core module to load first
    if "core" in modules:
        modules.remove("core")
```

```
        modules.insert(0, "core")
    for module in modules:
        if os.path.isdir(f"./modules/{module}"):
            self.load_module(module)
def load_module(self, name: str) -> Optional[str]:
    """
    Main loading method
    :param name: Name of Python module inside modules dir
    """
    os.chdir(f"./modules/{name}")
    if "requirements.txt" in os.listdir():
        # Check for dependencies update / install them
        if config.update_deps_at_load:
            self.install_deps(name, "modules")
        # Load dependencies into dict
        self.__modules_deps[name] = []
        for req in requirements.parse(
                open("requirements.txt", encoding="utf-8")
        ):
            self.__modules_deps[name].append(req.name.lower())
    try:
        imported = importlib.import_module("modules." + name)
    except ImportError as e:
        logger.error(f"ImportError has occurred while loading module {name}!")
        logger.exception(e)
        return None
    for obj_name, obj in inspect.getmembers(imported, inspect.isclass):
        if BaseModule in inspect.getmro(obj):
            try:
                instance: BaseModule = obj(
                    self.__bot,
                    self.get_modules_info,
                    self.bot_db_session,
                    self.bot_db_engine,
                )
                perms = instance.module_permissions
                info = instance.module_info
                # Don't allow modules with more than 1 word in name
                if len(info.name.split()) > 1:
```

```
                    logger.warning(
                        f"Module {name} has more than 1 word in name. Fuck
developer! I won't load it!"
                    )
                    del instance
                    os.chdir("../../")
                    return None
                if Permissions.require_db in perms and not config.enable_db:
                    logger.warning(
                        f"Module {name} requires DB, but it was disabled, skipping!"
                    )
                    del instance
                    os.chdir("../../")
                    return None

                if (
                    Permissions.use_db in perms or Permissions.require_db in perms
                ) and config.enable_db:
                    os.chdir(self.__root_dir)
                    asyncio.create_task(instance.set_db(Database(name)))
                    os.chdir(f"./modules/{name}")
                if Permissions.use_loader in perms:
                    instance.loader = self
                # Stage 1 init passed ok, applying extensions
                for ext_name, ext in self.__extensions.items():
                    try:
                        ext.on_module(instance)
                    except Exception as e:
                        logger.error(
                            f"Extension {ext_name} failed to apply on module
{info.name}!"
                        )
                        logger.exception(e)
                # Stage 2
                # Register everything for pyrogram
                instance.stage2()
                self.__modules[name] = instance
                self.__modules_info[name] = info
                # Custom init execution
```

```
                instance.on_init()
                # Remove hash backup
                if self.__hash_backups.get(name) is not None:
                    self.__hash_backups.pop(name)
                logger.info(f"Successfully imported module {info.name}!")
                os.chdir("../../")
                return info.name
            except Exception as e:
                logger.error(f"Error at loading module {name}! Printing traceback")
                logger.exception(e)
                os.chdir(self.__root_dir)
                return None
    def unload_module(self, name: str):
        """
        Method for unloading modules.
        :param name: Name of Python module inside modules dir
        """
        self.__modules[name].unregister_all()
        self.__modules.pop(name)
        self.__modules_info.pop(name)
        logger.info(f"Successfully unloaded module {name}!")
    def get_module(self, name: str) -> Optional[BaseModule]:
        """
        Get module instance object
        :param name: Name of Python module inside modules dir
        :return: Module object
        """
        return self.__modules.get(name)
    def get_modules_info(self) -> dict[str, ModuleInfo]:
        """
        Get info about all loaded modules
        :return: Dictionary with ModuleInfo objects
        """
        return self.__modules_info
    def get_module_info(self, name: str) -> Optional[ModuleInfo]:
        """
        Get module info
        :param name: Name of Python module inside modules dir
        :return: Object with module info
```

```
        """
        mod = self.__modules.get(name)
        if mod is None:
            return None
        else:
            return mod.module_info
    def get_module_help(self, name: str) -> Optional[str]:
        """
        Get module help page
        :param name: Name of Python module inside modules dir
        :return: Help page as string
        """
        mod = self.__modules.get(name)
        if mod is None:
            return None
        else:
            return mod.help_page
    def get_module_perms(self, name: str) -> list[Permissions]:
        """
        Get module permissions
        :param name: Name of Python module inside modules dir
        :return: Object with permissions
        """
        mod = self.__modules.get(name)
        if mod is None:
            return []
        else:
            return mod.module_permissions
    def install_from_git(self, url: str) -> (int, str):
        """
        Module installation method. Clones git repository from the given URL and
loads it
        :param url: Git repository URL
        :return: Tuple with exit code and read STDOUT
        """
        logger.info(f"Downloading module from git URL {url}!")
        name = urlparse(url).path.split("/")[-1].removesuffix(".git")
        cmd = f"cd {self.__root_dir}/modules\n" f"git clone {url}"
        p = subprocess.run(
```

```
            cmd, shell=True, stdout=subprocess.PIPE, stderr=subprocess.STDOUT
        )
        if p.returncode != 0:
            logger.error(f"Error while cloning module {name}!")
            logger.error(f"Printing STDOUT and STDERR:")
            logger.error(p.stdout.decode("utf-8"))
            subprocess.run(["rm", f"{self.__root_dir}/modules/{name}"])
        return p.returncode, p.stdout.decode("utf-8")
    def update_from_git(self, name: str, directory: str) -> (int, str):
        """
        Method to update git repository (module or extensions)
        Remembers commit hash for reverting and executes git pull
        :param name: Name of module or extension
        :param directory: Directory of modules or extensions
        :return: Exit code and output of git pull
        """
        # Unload first
        prev_version = self.__modules[name].module_info.version
        prev_db = self.__modules[name].db
        prev_db_meta = self.__modules[name].db_meta
        self.unload_module(name)
        logger.info(f"Updating {name}!")
        # Backup
        hash_cmd = f"cd {self.__root_dir}/{directory}/{name}/\n" f"git rev-parse
HEAD"
        hash_p = subprocess.run(
            hash_cmd, shell=True, stdout=subprocess.PIPE,
stderr=subprocess.STDOUT
        )
        if hash_p.returncode != 0:
            logger.error(f"Wtf, failed to retrieve HEAD hash... STDOUT below")
            logger.error(hash_p.stdout.decode("utf-8"))
            return hash_p.returncode, hash_p.stdout.decode("utf-8")
        self.__hash_backups[name] = hash_p.stdout.decode("utf-8")

        cmd = f"cd {self.__root_dir}/{directory}/{name}/\n" f"git pull"
        p = subprocess.run(
            cmd, shell=True, stdout=subprocess.PIPE, stderr=subprocess.STDOUT
        )
```

```
        if p.returncode != 0:
            logger.error(f"Error while updating module {name}!")
            logger.error(f"Printing STDOUT and STDERR:")
            logger.error(p.stdout.decode("utf-8"))
            return p.returncode, p.stdout.decode("utf-8")
        # Start database migration
        if prev_db is not None and os.path.exists(
            f"{self.__root_dir}/{directory}/{name}/db_migrations"
        ):
            for file in os.listdir(
                f"{self.__root_dir}/{directory}/{name}/db_migrations"
            ):
                mig_ver = file.removesuffix(".py")
                if version.parse(prev_version) < version.parse(mig_ver):
                    logger.info(
                        f"Migrating database for module {name} to version {mig_ver}..."
                    )
                    imported = importlib.import_module(
                        f"modules.{name}.db_migrations.{mig_ver}"
                    )
                    classes = inspect.getmembers(imported, inspect.isclass)
                    if len(classes) == 0:
                        logger.error("Invalid migration! No DBMigration classes found!")
                        continue
                    obj = classes[0][1]  # Use first detected class
                    instance: DBMigration = obj()
                    instance.apply(prev_db.session, prev_db.engine, prev_db_meta)
        return p.returncode, p.stdout.decode("utf-8")
    def revert_update(self, name: str, directory: str) -> bool:
        """
        Reverts update caused by update_from_git(). Removes updated dir and places
backup
        :param name: Name of module or extension
        :param directory: Directory of modules or extensions
        :return: Boolean (success or not)
        """
        try:
            cmd = (
                f"cd {self.__root_dir}/{directory}/{name}/\n"
```

```
            f"git reset --hard {self.__hash_backups[name]}"
        )
        p = subprocess.run(
            cmd, shell=True, stdout=subprocess.PIPE, stderr=subprocess.STDOUT
        )
        if p.returncode != 0:
            logger.error(
                f"Failed to revert update of module {name}! Printing STDOUT"
            )
            logger.error(p.stdout.decode("utf-8"))
            return False
        logger.info(f"Update of module {name} reverted!")
        return True
    except KeyError:
        logger.error(f"Tried to revert module {name} with no pending update!")
        return False
def install_deps(self, name: str, directory: str) -> (int, Union[str, list[str]]):
    """
    Method to install Python dependencies from requirements.txt file
    :param name: Name of module or extension
    :param directory: Directory of modules or extensions
    :return: Tuple with exit code and read STDOUT
    """
    logger.info(f"Upgrading dependencies for {name}!")
    r = subprocess.run(
        [
            sys.executable,
            "-m",
            "pip",
            "install",
            "-U",
            "-r",
            f"{self.__root_dir}/{directory}/{name}/requirements.txt",
        ],
        stdout=subprocess.PIPE,
        stderr=subprocess.STDOUT,
    )
    if r.returncode != 0:
        logger.error(
```

```
                f"Error at upgrading deps for {name}!\nPip output:\n"
                f"{r.stdout.decode('utf-8')}"
            )
            return r.returncode, r.stdout.decode("utf-8")
        else:
            logger.info(f"Deps upgraded successfully!")
            with open(f"{self.__root_dir}/{directory}/{name}/requirements.txt") as f:
                reqs = [dep.removesuffix("\n") for dep in f]
                if not reqs:
                    logger.warning(f"{name} requirements.txt is empty")
                elif reqs[-1] == "":
                    reqs.pop(-1)
                return r.returncode, reqs
    def uninstall_mod_deps(self, name: str):
        """
        Method to uninstall module dependencies. Removes package only if it isn't
required by other module
        :param name: Name of module
        :return:
        """
        for mod_dep in self.__modules_deps[name]:
            found = False
            for other_name, deps in self.__modules_deps.items():
                if other_name == name:
                    continue
                if mod_dep in deps:
                    found = True
                    break
            if found:
                continue
            subprocess.run(
                [sys.executable, "-m", "pip", "uninstall", "-y", mod_dep],
                stdout=subprocess.PIPE,
                stderr=subprocess.STDOUT,
            )
    def uninstall_packages(self, pkgs: list[str]):
        for dep in pkgs:
            found = False
            for other_name, deps in self.__modules_deps.items():
```

```
                if dep in deps:
                    found = True
                    break
            if found:
                continue
            subprocess.run(
                [sys.executable, "-m", "pip", "uninstall", "-y", dep],
                stdout=subprocess.PIPE,
                stderr=subprocess.STDOUT,
            )
    def uninstall_module(self, name: str) -> bool:
        """
        Module uninstallation method. Unloads and removes module directory
        :param name: Name of Python module inside modules dir
        :return: Bool, representing success or not
        """
        try:
            # Unload first
            self.unload_module(name)
            # Remove deps
            self.uninstall_mod_deps(name)
            self.__modules_deps.pop(name)
            shutil.rmtree(f"./modules/{name}")
            logger.info(f"Successfully removed module {name}!")
            return True
        except Exception as e:
            logger.error(f"Error while removing module {name}! Printing
traceback...")
            logger.exception(e)
            return False
    def get_int_name(self, name: str) -> Optional[str]:
        """
        Get internal name (name of a directory) of a module from user-friendly name
        :param name: User-friendly name of a module
        :return: Internal name of a module
        """
        for n, info in self.__modules_info.items():
            if info.name.lower() == name.lower():
                return n
```

```
        return None
```

modules/ServerMonitor/main.py

```
from base.module import BaseModule, command
from pyrogram.types import Message
from base.mod_ext import ModuleExtension
from typing import Type
# Extensions
from .extensions.cpu import CPUExtension
from .extensions.memory import MemoryExtension
from .extensions.os import OSExtension
from .extensions.sensors import SensorsExtension
from .extensions.network import NetworkExtension
# Python Libs
import subprocess
class ServerMonitorModule(BaseModule):
    @property
    def module_extensions(self) -> list[Type[ModuleExtension]]:
        return [
            CPUExtension,
            MemoryExtension,
            OSExtension,
            SensorsExtension,
            NetworkExtension
        ]
```

modules/ServerMonitor/extensions/cpu.py

```
# Bot stuff
from base.mod_ext import ModuleExtension
from base.module import command
# Pyrogram
from pyrogram import Client
from pyrogram.types import Message
# Libs
import platform
import cpuinfo
import psutil
class CPUExtension(ModuleExtension):
    @command("cpu")
```

```
    async def cpu_cmd(self, bot: Client, message: Message):
        string = f"□  <b>{self.S['cpu']['cpu_info']}</b>\n"
        string += f"● <b>{self.S['cpu']['name']}</b>:
{cpuinfo.get_cpu_info().get('brand_raw', self.S['cpu']['undetermined'])}
({cpuinfo.get_cpu_info()['arch_string_raw']})\n"
        string += f"● <b>{self.S['cpu']['count']}</b>:
{psutil.cpu_count(logical=False)} ({psutil.cpu_count()})\n"
        # string += (f"● <b>Freq</b>: {psutil.cpu_freq[0]} (max:
{psutil.cpu_freq[2]} / min: {psutil.cpu_freq[1]})\n")
        string += f"● <b>{self.S['cpu']['flags']}</b>: {'
'.join(cpuinfo.get_cpu_info().get('flags', self.S['cpu']['no_flags']))}\n"
        string += (
            f"● <b>Load avg</b>: {psutil.loadavg[0]} {psutil.loadavg[1]}
{psutil.loadavg[2]}\n"
            if hasattr(psutil, "loadavg")
            else ""
        )
        await message.reply(
            string,
            quote=True
        )
```

modules/ServerMonitor/extensions/memory.py

```
# Bot stuff
from base.mod_ext import ModuleExtension
from base.module import command
# Pyrogram
from pyrogram import Client
from pyrogram.types import Message
# Libs
import platform
import cpuinfo
import psutil
def progressbar(iteration: int, length: int) -> str:
    percent = ("{0:." + str(1) + "f}").format(100 * (iteration / float(100)))
    filledLength = int(length * iteration // 100)
    return "▓" * filledLength + "▒" * (length - filledLength)

def bytes2human(n):
```

```
    # http://code.activestate.com/recipes/578019
    # >>> bytes2human(10000)
    # '9.8K'
    # >>> bytes2human(100001221)
    # '95.4M'
    symbols = ("K", "M", "G", "T", "P", "E", "Z", "Y")
    prefix = {s: 1 << (i + 1) * 10 for i, s in enumerate(symbols)}
    for s in reversed(symbols):
        if n >= prefix[s]:
            value = float(n) / prefix[s]
            return "%.1f%s" % (value, s)
    return "%sB" % n
class MemoryExtension(ModuleExtension):
    @command("ram")
    async def ram_cmd(self, bot: Client, message: Message):
        string = f"  <b>{self.S['memory']['memory_info']}</b>\n"
        string += f"<b>{self.S['memory']['ram']}</b>:
|{progressbar(psutil.virtual_memory().percent, 10)}
<code>({bytes2human(psutil.virtual_memory().used)}/{bytes2human(psutil.virtua
l_memory().total)})</code>|\n"
        string += f"<b>{self.S['memory']['swap']}</b>:
|{progressbar(psutil.swap_memory().percent, 10)}
<code>({bytes2human(psutil.swap_memory().used)}/{bytes2human(psutil.swap_
memory().total)})</code>|\n"
        await message.reply(
            string,
            quote=True
        )
    @command("rom")
    async def rom_cmd(self, bot: Client, message: Message):
        string = f"  <b>{self.S['memory']['disk_info']}</b>\n"
        for disk in psutil.disk_partitions():
            disk_usage = psutil.disk_usage(disk.mountpoint)

            string += f"<b>{disk.device}</b>\n"
            string += f" ├── <b>{self.S['memory']['diskMountpint']}</b>
{disk.mountpoint}\n"
            string += f" ├── <b>{self.S['memory']['diskFileSystem']}</b>
{disk.fstype}\n"
```

```
        string += f" ├── <b>{self.S['memory']['diskUsage']}</b>
{disk_usage.percent}%
({bytes2human(disk_usage.used)}/{bytes2human(disk_usage.total)})\n"
        string += f" │       └──{progressbar(disk_usage.percent, 10)}\n"
        string += f" └── <b>{self.S['memory']['diskOptions']}</b>
{disk.opts}\n\n"
    await message.reply(
        string,
        quote=True
        )
```

modules/ServerMonitor/extensions/network.py

```
# Bot stuff
from base.mod_ext import ModuleExtension
from base.module import command
# Pyrogram
from pyrogram import Client
from pyrogram.types import Message
# Libs
import platform
import cpuinfo
import psutil
AddressFamily = {
    2: "IPv4",
    10: "IPv6",
    28: "IPv6",
    17: "Link",
    18: "Link",
  }
class NetworkExtension(ModuleExtension):
  @command("netaddr")
  async def netaddr_cmd(self, bot: Client, message: Message):
    net_if_addrs = psutil.net_if_addrs()
    string = f"🌐 <b>{self.S['network']['network_info']}</b>\n"
    string += f"<b>{self.S['network']['address']}</b>:\n"
    for interf in net_if_addrs:
      interface = net_if_addrs[interf]
      string += f"<b>{interf}</b>\n"
      for addr in interface:
```

```
            attr = [
                a
                for a in dir(psutil.net_if_addrs()[interf][0])
                if not a.startswith("__")
                and not a.startswith("_")
                and not callable(getattr(psutil.net_if_addrs()[interf][0], a))
            ]
            string += f"{AddressFamily[getattr(addr, 'family')]}\n"
            for item in attr[:-1]:
                string += f" ├── {item}: {getattr(addr, item)}\n"
            string += f" └── {attr[-1]}: {getattr(addr, attr[-1])}\n"
        string += "\n"
    await message.reply(
        string,
        quote=True
    )
@command("netstats")
async def netstats_cmd(self, bot: Client, message: Message):
    net_if_stats = psutil.net_if_stats()
    string = "🌐 <b>Network Info</b>\n" + "<b>Stats</b>:\n"
    for interf in net_if_stats:
        interface = net_if_stats[interf]
        string += f"<b>{interf}</b>\n"
        string += f" └── {interface}\n\n"
    await message.reply(
        string,
        quote=True
    )
```

modules/ServerMonitor/extensions/os.py

```
# Bot stuff
from base.mod_ext import ModuleExtension
from base.module import command
# Pyrogram
from pyrogram import Client
from pyrogram.types import Message
# Python libs
from os.path import exists
import datetime
```

```
import subprocess
# Libs
import platform
import cpuinfo
import psutil
backslash = "\n"
returnslash = "\r"
def get_os_release():
        if not exists("/etc/os-release"):
            return False
        list_ = []
        with open("/etc/os-release") as f:
            list_.extend(item.split("=") for item in f.readlines())
        return {item[0]: item[1].replace(backslash, "").replace("", "") for item in list_}
# https://stackoverflow.com/questions/2756737/check-linux-distribution-name
def get_distro():
    """
    Name of your Linux distro
    """
    if not exists("/etc/issue"):
        return False
    with open("/etc/issue") as f:
        return f.read().split()[0]
os_release = get_os_release()
class OSExtension(ModuleExtension):
    @command("distro")
    async def distro_cmd(self, bot: Client, message: Message):
        string = f"""□  <b>{self.S['os']['linux_info']}</b>
        <b>{self.S['cpu']['name']}:</b> {get_distro()}
        <b>{self.S['os']['kernel']}:</b> {platform.release()}
        <b>{self.S['os']['hostname']}:</b> {platform.node()}
        <b>{self.S['os']['boot_time']}:</b>
{datetime.datetime.fromtimestamp(psutil.boot_time()).strftime(
            "%Y-%m-%d %H:%M:%S"
        )}
        {f'<b>{self.S["os"]["glibc_ver"]}:</b> {platform.glibc()[1]}' if
hasattr(platform, 'glibc') else ''}
        """
        await message.reply(
```

```
            string,
            quote=True
        )
    @command("osrelease")
    async def osrelease_cmd(self, bot: Client, message: Message):
        string = f"""📦 <b>{self.S["os"]["os_releases"]}:</b>
        <b>{self.S["os"]["pretty_name"]}:</b> {os_release["PRETTY_NAME"]}
        <b>{self.S["cpu"]["name"]}:</b> {os_release["NAME"]}
        <b>{self.S["os"]["version"]}:</b> {os_release.get("VERSION",
self.S["os"]["not_available"])}
        <b>{self.S["os"]["documentation"]}:</b>
{os_release.get("DOCUMENTATION_URL", self.S["os"]["not_available"])}
        <b>{self.S["os"]["support"]}:</b> {os_release.get("SUPPORT_URL", "Not
available")}
        <b>{self.S["os"]["bug_report"]}:</b> {os_release["BUG_REPORT_URL"]}
            """
        await message.reply(
            string,
            quote=True
        )
    @command("neofetch")
    async def neofetch_cmd(self, _, message: Message):
        output = subprocess.getoutput("neofetch --stdout")
        await message.reply(f"<code>{output}</code>")
```

modules/ServerMonitor/extensions/sensors.py

```
# Bot stuff
from base.mod_ext import ModuleExtension
from base.module import command
from ..config import ConfigSettings
# Pyrogram
from pyrogram import Client
from pyrogram.types import Message
# Libs
import platform
import cpuinfo
import psutil
import requests
```

```
# Python Libs
import time
from threading import Thread
class SensorsExtension(ModuleExtension):
    def on_init(self):
        thread = Thread(target=self.temp_monitor)
        thread.daemon = True
        thread.start()
    def temp_monitor(self):
        while True:
            sensors_temperatures = psutil.sensors_temperatures()
            # sensor_name - amdgpu
            for sensor_name in sensors_temperatures:
                # get temp by sensor name
                sensor = sensors_temperatures[sensor_name]
                if sensor[0].high == None:
                    high = ConfigSettings.high_temp
                else:
                    high = sensor[0].high
                if(sensor[0].current > high and ConfigSettings.ntfy_topic):
                    requests.post(f"https://ntfy.sh/{ConfigSettings.ntfy_topic}",
                        data=f"□ [{sensor_name}] {sensor[0].current} /
{high}".encode(encoding='utf-8'))
                # print(f"[{sensor_name}] {sensor[0].current} / {high}")
time.sleep(60)
    @command("temp")
    async def temp_cmd(self, bot: Client, message: Message):
        if hasattr(psutil, "sensors_temperatures"):
            sensors_temperatures = psutil.sensors_temperatures()
            string = f"🌡 <b>{self.S['sensors']['sensors_info']}</b>\n" +
f"<b>{self.S['sensors']['temperature']}</b>:\n"
            string = "🌡 <b>Sensors Info</b>\n" + "<b>Temperature</b>:\n"
            string = f"🌡 <b>{self.S['sensors']['sensors_info']}</b>\n" +
f"<b>{self.S['sensors']['temperature']}</b>:\n"
            for sensor_name in sensors_temperatures:
                sensor = sensors_temperatures[sensor_name]
                string += f"<b>{sensor_name}</b>\n"
                for sensor_info in sensor:
                    attr = [
```

```
                a
                for a in dir(sensor_info)
                if not a.startswith("__")
                and not a.startswith("_")
                and not callable(getattr(sensor_info, a))
            ]
            for item in attr[:-1]:
                string += f" ├── {item}: {getattr(sensor_info, item)}\n"
            string += f" └── {attr[-1]}: {getattr(sensor_info, attr[-1])}\n"
        await message.reply(
            string,
            quote=True
        )
    else:
        await message.reply(
            cpu_string(),
            quote=True
        )
# TODO: Implement baterry
@command("baterry")
async def baterry_cmd(self, bot: Client, message: Message):
    pass
@command("fan")
async def fan_cmd(self, bot: Client, message: Message):
    if not hasattr(psutil, "sensors_fans"):
        return await message.reply(
            f"❌ <b>{self.S['sensors']['no_fans']}</b>",
            quote=True
        )
    sensors_fans = psutil.sensors_fans()
    if sensors_fans == {}:
        return await message.reply(
            f"❌ <b>{self.S['sensors']['no_fans']}</b>",
            quote=True
        )
    string = f"<b>{self.S['sensors']['fans']}</b>:"
    string = "<b>Fans</b>:"
    string = f"<b>{self.S['sensors']['fans']}</b>:"
    for sensor_name in sensors_fans:
```

```
    sensor = sensors_fans[sensor_name]
    string += f"<b>{sensor_name}</b>\n"
    for sensor_info in sensor:
        attr = [
            a
            for a in dir(sensor_info)
            if not a.startswith("__")
            and not a.startswith("_")
            and not callable(getattr(sensor_info, a))
        ]
        for item in attr[:-1]:
            string += f" ├── {item}: {getattr(sensor_info, item)}\n"
        string += f" └── {attr[-1]}: {getattr(sensor_info, attr[-1])}\n"
await message.reply(
    string,
    quote=True
)
```

ДОДАТОК 2

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ ДО ДИПЛОМНОГО ПРОЕКТУ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ
«ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

ДИПЛОМНИЙ ПРОЕКТ НА ТЕМУ:
“Створення чат боту для моніторингу стану компонентів серверу на базі месенджера Telegram”

- Дипломник: Кир'язов О.в
- Керівник: Скорнякова О.В.

Одеса, 2023р

Вступ

- В сучасному світі комп’ютерна техніка є невід’ємною частиною життя людини, тому належна робота серверного обладнання має вирішальне значення. Щоб забезпечити безперебійну роботу серверів та повноцінну підтримку користувачів, необхідно постійно моніторити та контролювати їх роботу.

Платформа для проєкту

- У наш час на серверах воліють використовувати саме серверні системи, які заточуються під серверні потреби. І з цієї причини, основна платформа, на якій працюватиме проєкт - GNU/Linux

Мова програмування

- Python - інтерпретована мова, що дозволяє заощадити значну кількість часу, але страждає швидкість роботи програми.
- Дозволяє розбивати програми на модулі, що потім можуть бути використані в інших програмах.
- Python поставляється з великою бібліотекою стандартних модулів, які можна використовувати як основу для нових програм

Бібліотека Pyrogram

- Pyrogram - це високорівнева бібліотека Python для створення асинхронних клієнтів Telegram API. Вона написана на мові Python з використанням асинхронних фреймворків, таких як asyncio та aiohttp.
- Підтримує звичайний BotAPI, а також MTProto, який працює через протокол MTProto, що дає змогу легкою кров'ю прибрати деякі обмеження BotAPI.

Порівняння BotAPI та MTProto

Переваги MTProto API:

- Завантажувати та вивантажувати будь-які файли, до 2 ГБ кожен
- Має менше накладних витрат завдяки прямому підключенню до Telegram
- Швидше отримувати оновлення версій API, а отже, і нові функції
- Має набагато більш деталізовані типи та потужні методи

Вимоги до бота

- Модулі - це репозиторії git
- Багатомовність. Зберігання в .yaml
- Права доступу на рівні ролей
- У кожного модуля своя база даних
- Обмеження прав у модуля. Що неявно не вказано - те заборонено

Переваги модульної структури

- Кожен модуль бота буде виконувати певну функцію.
- Можливість додавати додатковий функціонал
- Боти можуть бути налаштовані на індивідуальні потреби користувачів
- Розробка модульного бота може зменшити трудомісткість управління ботом, оскільки окремі модулі можуть бути розроблені та керовані окремо.
- Оскільки кожен модуль може бути розроблений та підтримуватися окремо, модульний бот легко масштабувати при збільшенні обсягу роботи або зміні потреб користувачів.

Будова бота

```
PBModular
├── base
│   ├── base_ext.py
│   ├── command_registry.py
│   ├── db_migration.py
│   ├── db.py
│   ├── __init__.py
│   ├── loader.py
│   ├── mod_ext.py
│   └── module.py
├── config.example.yaml
├── config.py
├── db.py
├── install.ps1
├── install.sh
├── LICENSE
├── main.py
├── modules
│   └── core
├── README.md
└── requirements.txt
```

- main.py – головний файл
- base/ - Базові класи для модулів і БД
- config.py - парсер config.yaml
- db.py - Моделі для БД
- install.sh/ps1 - Скрипти для автоматичного встановлення
- modules/ - Директорія, де зберігаються модулі
- requirements.txt - Залежності для роботи програми

Вимоги до модуля моніторингу стану

- Виведення текстом основних параметрів сервера
- Моніторнінг температурних датчиків
- Пуш-повідомлення на телефон сис. адміністратора
- Можливість налаштування критичної температури
- Можливість налаштування пуш-повідомлення

Будова базового модуля

```
modules/template
├── info.yaml
├── __init__.py
├── main.py
├── README.md
├── requirements.txt
└── strings
    ├── en.yaml
    └── uk.yaml
```

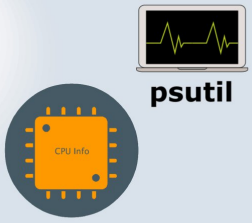
- info.yaml - метадата модуля
- __init__.py - Запускається до завантаження модуля
- main.py - Головний файл
- requirements.txt - Залежності, необхідні для модуля
- strings - *.yaml із рядками різними мовами

Вимоги до модуля моніторингу стану

- Виведення текстом основних параметрів сервера
- Моніторнінг температурних датчиків
- Пуш-повідомлення на телефон сис. адміністратора
- Можливість налаштування критичної температури
- Можливість налаштування пуш-повідомлення

Використовувані бібліотеки

- psutil - кросплатформенна бібліотека для отримання інформації про запущені процеси та використання системи
- py-cpuinfo отримує інформацію про процесор за допомогою лише Python. Py-cpuinfo має працювати без жодних додаткових програм чи бібліотек, окрім тих, що надаються вашою операційною системою.

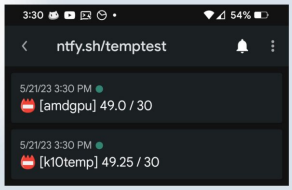


Фрагмент коду моніторингу та надсилання пуш-повідомлення

- В окремому потоці кожні 60 секунд перевіряються датчики, якщо вони не перевищують поріг, нічого не робимо, а якщо перевищив - використовуємо веб-сервіс ntfy.sh для надсилання push-повідомлення

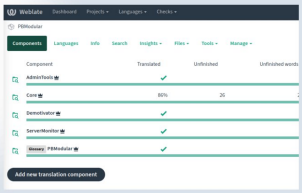
Приклад повідомлення на телефон

- З використанням ntfy.sh, сис. адміністратор швидко буде повідомлений про критично високу температуру на сервері.



Процес перекладу рядків іншими мовами

- Для перекладів використовується власний інстанс Weblate. Наразі перекладено двома мовами, англійською та українською



Що було в результаті зроблено?

- Модульний телеграм-чат бот
- Модуль, який надає системну інформацію
- Постійний моніторинг температури
- Push-повідомлення через сервіс ntfy.sh

Дякую за увагу

Весь вихідний код проекту доступний на GitHub за ліцензією GNU GPL3:

- https://github.com/PBModular/bot
- https://github.com/PBModular/ServerMonitor

GPL v3 Free Software
Free as in Freedom

ДОЗВІЛ
НА РОЗМІЩЕННЯ
ВИПУСКНОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»

Ми, що нижче підписалися,

Кир'язов Олег Вікторович,
здобувач освіти гр. 2БКС-27, та

Скорнякова Олена Володимирівна,
керівник випускної кваліфікаційної роботи,

не заперечуємо щодо розміщення електронного варіанту пояснювальної записки до випускної кваліфікаційної роботи бакалавра на тему:

«Створення чат-боту для моніторингу стану компонентів серверу на базі месенджера Telegram» (автор роботи – Кир'язов О.В., керівник роботи – Скорнякова О.В.)

виконаного у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2023 році, у повному обсязі в електронному репозитарії ВСП «ОТФК ОНТУ» для вільного доступу через мережу Інтернет.

Несемо відповідальність за ідентичність електронного та друкованого варіантів випускної кваліфікаційної роботи, і даємо згоду на обробку персональних даних.

Виконавець _____ / Кир'язов О.В. /

Керівник _____ / Скорнякова О.В./

«16» 06 2023 р.

ВСП «Одеський технічний фаховий коледж ОНАХТ»

ВІДГУК

Керівника про кваліфікаційну роботу бакалавра

Кир'язова Олега Вікторовича

(прізвище, ім`я та по батькові)

Освітньо-професійна програма *«Комп'ютерна інженерія»*

Спеціальність *123 «Комп'ютерна інженерія»*

Тема кваліфікаційної роботи

Створення чат боту для моніторингу стану компонентів серверу на базі месенджера Telegram

ХАРАКТЕРИСТИКА КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

а) Обсяг і якість виконання роботи (розрахунково-пояснювальної записки)

Пояснювальна записка виконана якісно, у достатньому обсязі, відповідно до індивідуального завдання та теми кваліфікаційної роботи, розділи пояснювальної записки відповідають етапам рішення завдання, поставленого у роботі. Представлено програмний код. Є у наявності є презентація. Презентація виконана якісно, у достатньому обсязі. Презентація наочно демонструє результати роботи.

б) Самостійність роботи над кваліфікаційною роботою

Здобувач отримав завдання від керівника, провів аналіз існуючих рішень і зробив необхідні висновки для реалізації проекту, провів тестування та апробацію системи в реальних умовах. Продемонстрував уміння швидко реагувати на вимоги та виправляти виявлені недоліки, відгукуватися на пропозиції та втілювати їх оперативно в проект. Виявив навички самостійно опрацьовувати новий матеріал та виконувати пошук необхідної літератури та інших джерел інформації.

в) Теоретична підготовка бакалавра

відповідає вимогам до бакалавра зі спеціальності

«Комп'ютерна інженерія»

г) Вміння розв`язувати виробничі і конструкторські питання на базі останніх досліджень науки і техніки, передових методів виробництва

В процесі роботи над кваліфікаційною роботою здобувач продемонстрував уміння використовувати останні досягнення науки та техніки в предметній галузі на підставі відповідної навчальної та науково-технічної літератури, достатньо впевнено користувався програмним забезпеченням при роботі над дипломним проектом та створенням презентації.

Загальна оцінка *відмінно*

Прізвище, ім`я, по батькові *к.п.н. Скорнякова Олена Володимирівна*

Місто роботи і посада керівника проекту: к.пед.н., викладач-методист комісії КТ та ПІ ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету»

Підпис

«16» 06 2023 р.

UNICHECK

Ім'я користувача:
Наталія Вікторівна Копусь

ID перевірки:
1015585852

Дата перевірки:
13.06.2023 14:49:09 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
13.06.2023 14:50:32 EEST

ID користувача:
100011688

Назва документа: **2БКС-27 Кир'язов Олег**

Кількість сторінок: **38** Кількість слів: **7078** Кількість символів: **49905** Розмір файлу: **2.10 MB** ID файлу: **1015235413**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

17.7%
Схожість

Найбільша схожість: **8.83%** з Інтернет-джерелом (http://www.plug.org.ua/documentation/about-python?month%3Aint=..

17.7% Джерела з Інтернету **534** ..... Сторінка 40

Не знайдено джерел з Бібліотеки

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Підозріле форматування **6 сторінок**

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на кваліфікаційну роботу бакалавра здобувача (здобувачки) освіти

відділення комп'ютерних систем

Кир'язова Олега Вікторовича

(прізвище, ім`я та по батькові)

Напрям підготовки **123 "Комп'ютерна інженерія"**

Керівник кваліфікаційної роботи

к.пед.н. Скорнякова Олена Володимирівна

(прізвище, ім`я та по батькові)

Тема кваліфікаційної роботи

Створення чат-боту для моніторингу стану компонентів серверу на базі месенджера Telegram

Обсяг пояснювальної записки _____ сторінок

Обсяг графічної (презентаційної) частини _____ аркушів (слайдів)

ХАРАКТЕРИСТИКА КВАЛІФІКАЦІЙНОЇ РОБОТИ

а) Заключення про ступінь відповідальності виконаної роботи завданню

Виконання роботи та її зміст повністю відповідає завданню до проектування

б) Характеристика виконання кожного розділу роботи, ступеню використання випускником-бакалавром останніх досягнень науки та техніки, передових методів на виробництві

Пояснювальна записка виконана якісно, у достатньому обсязі, відповідно до індивідуального завдання та теми кваліфікаційної роботи, розділи пояснювальної записки відповідають етапам рішення завдання, поставленого у роботі. Є у наявності графічний матеріал та презентація. Презентація виконана якісно, у достатньому обсязі та наочно демонструє результати роботи.

в) Оцінка якості виконання графічної (презентаційної) частини роботи і пояснювальної записки

Презентаційні матеріали виконані якісно, демонстративно та відповідають вмісту теоретичного матеріалу

г) перелік позитивних якостей кваліфікаційної роботи:

Тематика кваліфікаційної роботи є актуальною. Серед позитивних якостей – демонстрація роботи чат-боту

д) основні недоліки кваліфікаційної роботи

В роботі зустрічаються несуттєві відхилення від вимог щодо оформлення пояснювальної записки кваліфікаційної роботи. Варто було б в пояснювальній записці представити алгоритм обробки запитів від користувачів чат-бота

Оцінка розрахункової частини ***Відмінно***

Оцінка графічної частини ***Відмінно***

Загальна оцінка ***Відмінно***

Прізвище, ім'я, по батькові рецензента ***Василіу Євген Вікторович***

Місце роботи і посада рецензента ***Державний університет інтелектуальних технологій і зв'язку, д.т.н., проф. кафедри КБ та ТЗІ, декан факультету інформаційних технологій та кібербезпеки***

Підпис:

«16» 06 2023 р.

ПІДПИС ПОСВІДЧУЮ
НАЧАЛЬНИК ВІДДІЛУ
КАДРІВ ДУІТЗ

ВІДДІЛ КАДРІВ