**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**
**імені ІГОРЯ СІКОРСЬКОГО»**

**Навчально-науковий інститут телекомунікаційних систем**

**Кафедра телекомунікацій**

До захисту допущено:

Завідувач кафедри

_____ Сергій КРАВЧУК

«___»_____2025 р.

# Дипломна робота

**на здобуття ступеня бакалавра**

**за освітньо-професійною програмою «Інженерія та програмування інфокомунікацій»**

**спеціальності 172 «Телекомунікації та радіотехніка»**

**на тему: «Розробка та впровадження децентралізованої інфраструктури самостійно розміщених (self-hosted) сервісів як альтернатива централізованим корпоративним рішенням»**

Виконав:
студент IV курсу, групи ТЗ-12
Стоянов Ілля Олексійович _____

Керівник:
Старший викладач кафедри ТК НН ІТС,
Кайденко Микола Миколайович _____

Рецензент:
Доцент кафедри ІТТ НН ІТС, к.т.н., доцент
Правило Валерій Володимирович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

Київ – 2025 року

**Національний технічний університет України**
**«Київський політехнічний інститут імені Ігоря Сікорського»**
**Навчально-науковий інститут телекомунікаційних систем**
**Кафедра телекомунікацій**

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 172 «Телекомунікації та радіотехніка»

Освітньо-професійна програма «Інженерія та програмування інфокомунікацій»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій КРАВЧУК
«___»_____2025 р.

**ЗАВДАННЯ**
**на дипломну роботу студенту**
**Стоянову Іллі Олексійовичу**

1. Тема роботи «Розробка та впровадження децентралізованої інфраструктури самостійно розміщених (self-hosted) сервісів як альтернатива централізованим корпоративним рішенням», керівник роботи Кайденко Микола Миколайович, затверджені наказом по університету від «26» травня 2025 р. № 1755-с.

2. Термін подання студентом роботи 9 червня 2025 р.

3. Вихідні дані до роботи: Мікрокомп’ютер Raspberry Pi 5 (8 GB RAM) з операційною системою Raspberry Pi OS Bookworm (Linux); основне сховище – microSD 128 GB, додаткове – microSD 512 GB (через адаптер USB); мережеве з’єднання – Ethernet (UGREEN); клієнтські пристрої – ПК x86 з Windows 11, ПК x86 з Linux Mint, смартфон на Android 15.

4. Зміст роботи: Розроблено персональне серверне середовище на базі одноплатного комп’ютера Raspberry Pi 5 з використанням відкритого програмного забезпечення. Проведено аналіз архітектури ARM та обґрунтовано її застосування для побудови енергоефективної автономної платформи. Розгорнуто систему контейнеризації на базі Docker та реалізовано зручний графічний інтерфейс керування сервісами за допомогою Portainer. Налаштовано захищений віддалений доступ до внутрішніх ресурсів через VPN-сервіс Tailscale з підтримкою функції підмережевого маршрутизатора.

Впроваджено самостійно розміщені сервіси: Sunshine та Moonlight — для реалізації функції хмарного геймінгу та доступу до віддаленого робочого столу; Grocy — для побутового обліку продуктів та планування; Nextcloud — як альтернатива хмарному сховищу з підтримкою WebDAV. Розгорнуто локальну мовну модель штучного інтелекту в середовищі LM-Studio та забезпечено її взаємодію з фронтендом SillyTavern. Проведено інтеграцію системи з open-source середовищем FoundryVTT. Налаштовано механізм автоматичного оновлення контейнерів за допомогою Watchtower та реалізовано синхронізацію нотаток через Joplin. Оптимізовано мережеву інфраструктуру та забезпечено приватність, контроль і автономність обробки даних. Проведено практичне тестування системи та обґрунтовано її доцільність для використання у навчальних, дослідницьких і побутових умовах.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо):

1) Вступ
2) Актуальність
3) Опис проблем
4) Мета
5) Завдання
6) Огляд макету
7) Логічна структура ПЗ
8) Висновки до роботи

6. Дата видачі завдання 14.04.2025

Календарний план

| № з/п | Назва етапів виконання дипломної роботи | Термін виконання етапів роботи | Примітка |
|---|---|---|---|
| 1. | Підключення плати Raspberry PI | 14.04 - 20.04 | Виконано |
| 2. | Налаштування підмережевого роутера | 14.04 - 20.04 | Виконано |
| 3. | Встановлення Docker | 14.04 - 20.04 | Виконано |
| 4. | Встановлення Portainer | 14.04 - 20.04 | Виконано |
| 5. | Встановлення Grocy | 21.04 - 27.04 | Виконано |
| 6. | Встановлення Sunshine та Moonlight | 21.04 - 27.04 | Виконано |
| 7. | Встановлення Nextcloud | 28.04 - 04.05 | Виконано |
| 8. | Встановлення Backend та Frontend для локального AI. | 05.05 - 11.05 | Виконано |

| | | | |
|---|---|---|---|
| 9. | Локальна синхронізація та автооновлення контейнерів за розкладом | 05.05 - 11.05 | Виконано |
| 10. | Підготовка висновків та оформлення дипломної роботи. | 12.05 - 19.05 | Виконано |
| 11. | Підготовка звітних та презентаційних матеріалів. | 20.05 - 30.05 | Виконано |

Студент Ілля СТОЯНОВ

Керівник Микола КАЙДЕНКО

# РЕФЕРАТ

Робота містить 59 сторінок, 31 ілюстрацію. Для написання було використано 12 джерел.

Мета роботи – дослідження методів побудови персоналізованого серверного середовища на базі відкритого програмного забезпечення, яке здатне функціонувати як самостійна альтернатива централізованим хмарним сервісам.

У роботі проведено аналіз архітектури ARM як основи для енергоефективної платформи Raspberry Pi 5. Встановлено та налаштовано середовище контейнеризації Docker і графічний інтерфейс Portainer. Реалізовано безпечний віддалений доступ за допомогою VPN Tailscale. Впроваджено сервіси Grocy (облік продуктів), Nextcloud (файловий сервер), Sunshine + Moonlight (віддалений робочий стіл), мовну модель штучного інтелекту (LM-Studio + SillyTavern), інструменти синхронізації Joplin та автоматичного оновлення Watchtower.

Результати роботи – створена та протестована функціональна, масштабована і безпечна серверна екосистема з можливістю інтеграції у смарт-інфраструктури, підтримкою автономності, відкритістю коду і захистом персональних даних.

Результати впроваджено у локальному середовищі та частково інтегровано в студентський open-source проєкт LLM Chat for FoundryVTT.

Ключові слова: сервер, Raspberry Pi, Docker, VPN, Tailscale, Sunshine, Moonlight, Nextcloud, Grocy, штучний інтелект, LM-Studio, SillyTavern, Joplin, ARM, self-hosting, open-source, синхронізація, безпека, локальна мережа, автоматизація.

# ABSTRACT

The thesis contains 59 pages, 31 illustration. 12 sources were used for writing.

Objective of the thesis – to explore methods for building a personalized server environment based on open-source software, capable of functioning as a standalone alternative to centralized cloud services.

The work includes an analysis of ARM architecture as the foundation for an energy-efficient Raspberry Pi 5 platform. A Docker containerization environment and the Portainer graphical interface were installed and configured. Secure remote access was implemented using the Tailscale VPN. Several services were deployed: Grocy (product inventory), Nextcloud (file server), Sunshine + Moonlight (remote desktop), a local AI language model (LM-Studio + SillyTavern), the note synchronization tool Joplin, and the Watchtower utility for automatic container updates.

Results – a functional, scalable, and secure server ecosystem was created and tested, with potential integration into smart infrastructures, support for autonomy, open-source transparency, and strong data privacy.

The results were successfully implemented in a local environment and partially integrated into the student open-source project LLM Chat for FoundryVTT.

Keywords: server, Raspberry Pi, Docker, VPN, Tailscale, Sunshine, Moonlight, Nextcloud, Grocy, artificial intelligence, LM-Studio, SillyTavern, Joplin, ARM, self-hosting, open-source, synchronization, security, local network, automation.

# ЗМІСТ

# ПЕРЕЛІК СКОРОЧЕНЬ

| | |
|---|---|
| API | Application Programming Interface |
| ARM | Advanced RISC Machine |
| CLI | Command Line Interface |
| CPU | Central Processing Unit |
| Docker | Платформа для контейнеризації програм |
| GUI | Graphical User Interface |
| IP | Internet Protocol |
| LM | Language Model |
| OS | Operating System |
| PUID / PGID | User ID / Group ID |
| RAM | Random Access Memory |
| RDP | Remote Desktop Protocol |
| SSD | Solid State Drive |
| UI | User Interface |
| URL | Uniform Resource Locator |
| USB | Universal Serial Bus |
| VPN | Virtual Private Network |
| WOL | Wake-On-LAN |
| WebDAV | Web-based Distributed Authoring and Versioning |
| YAML | Yet Another Markup Language |

# ВСТУП

## Актуальність теми роботи

Сучасний розвиток інформаційних технологій, зокрема широке впровадження хмарних сервісів, зумовив значне зростання популярності централізованих онлайн-платформ для зберігання даних, обробки інформації, віддаленої роботи та взаємодії. Однак поряд із зручністю та доступністю, такі сервіси несуть низку ризиків, серед яких – залежність від провайдерів, обмеження функціональності, а також порушення конфіденційності та контролю над персональними даними.

Проблематика втрати контролю над даними та сервісами, особливо в умовах закритих пропрієтарних платформ, стала підґрунтям для зростання інтересу до моделей самостійного розміщення (self-hosting), які базуються на відкритому програмному забезпеченні (open-source). Власні серверні рішення дають змогу реалізувати повноцінну інфраструктуру без залежності від комерційних хмарних провайдерів, з можливістю масштабування, конфігурування та інтеграції відповідно до індивідуальних потреб користувача.

На фоні зростаючої актуальності захисту персональних даних, забезпечення автономності та оптимізації витрат, тема створення персоналізованого серверного середовища з відкритим кодом є вкрай актуальною та має значний науково-практичний потенціал.

**Метою роботи** є дослідження принципів побудови персоналізованої серверної інфраструктури на базі відкритого програмного забезпечення, яка може слугувати повноцінною альтернативою централізованим хмарним сервісам. Основний акцент зроблено на використанні одноплатного комп’ютера Raspberry Pi 5, інструментів контейнеризації, систем VPN-доступу, а також інтеграцію локальних сервісів для повсякденного використання.

**Завдання роботи** охоплюють комплексну розробку серверної інфраструктури на базі одноплатного комп’ютера Raspberry Pi 5, орієнтованої

на самостійне розміщення сервісів із використанням відкритого програмного забезпечення. Першим етапом є обґрунтування вибору апаратної платформи — аналіз архітектури ARM з позиції її енергоефективності, компактності та відповідності вимогам локального серверного середовища.

Наступним завданням є встановлення та налаштування базового контейнерного середовища з використанням Docker, що дозволяє стандартизувати розгортання сервісів. Для полегшення керування контейнерами інтегрується графічна панель Portainer, яка забезпечує інтуїтивний інтерфейс взаємодії.

Особлива увага приділяється реалізації безпечного віддаленого доступу до серверної інфраструктури через VPN-рішення Tailscale, яке не потребує складної ручної конфігурації та підтримує тунелювання через NAT.

У подальшому здійснюється розгортання низки сервісів: Grocy як система обліку продуктів, Nextcloud як приватне хмарне сховище, Moonlight та Sunshine як реалізація віддаленого доступу до графічного інтерфейсу, Joplin для синхронізації нотаток, а також локальна генеративна мовна модель (LLM) у зв’язці з SillyTavern для забезпечення інтерфейсу взаємодії.

Також передбачено впровадження автоматизованої системи оновлення сервісів за допомогою Watchtower, що дозволяє підтримувати стабільність і актуальність усіх компонентів.

Завершальним етапом роботи є тестування створеної інфраструктури у локальному середовищі та оцінка її придатності до масштабування, адаптації до індивідуальних потреб користувача, а також можливості інтеграції у навчальні або смарт-середовища.

Результатом дослідження має стати автономне, безпечне та відкрито масштабоване серверне рішення, що може функціонувати як альтернатива комерційним хмарним платформам.

# РОЗДІЛ 1
# ОРГАНІЗАЦІЯ МЕРЕЖІ ТА КОНТЕЙНЕРНОГО СЕРЕДОВИЩА

## 1.1. Ознайомлення з Tailscale та налаштування VPN доступу

Tailscale - це VPN-сервіс, що базується на WireGuard®. Ця програма є незамінною у створенні свого домашнього сервера. Вона допомагає отримувати доступ до локальної мережі без необхідності відкривати порти, чим наражати свої дані на небезпеку, чи витрачати ресурси на реєстрацію власного домена [1]. На рисунку 1.1 схематично зображено принцип роботи Tailscale як посередника між сервером та клієнтом

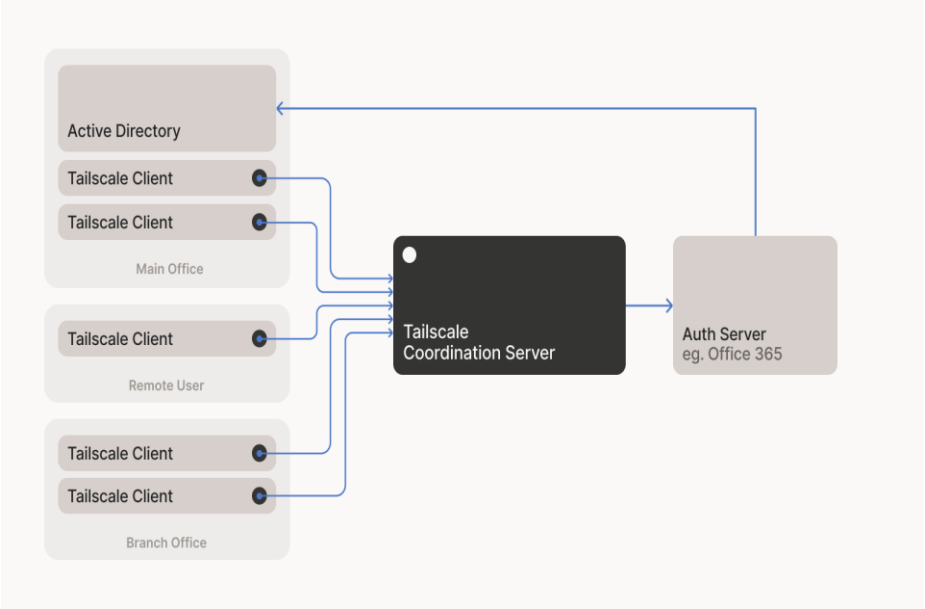


Рис 1.1 Схематичне зображення принципу роботи Tailscale

Технічні аспекти роботи Tailscale:

- Tailscale створює повнозв'язну mesh-мережу між всіма пристроями
- Кожен пристрій отримує унікальну IPv4 та IPv6 адресу в приватній мережі
- З'єднання встановлюються напряму між пристроями (peer-to-peer) де це можливо
- Використовує технології STUN та ICE для проходження через NAT

- Не потребує налаштування портів на маршрутизаторі
- У випадках, коли пряме з'єднання неможливе, використовує DERP-сервери як ретранслятори
- Базується на протоколі WireGuard, який вважається одним з найбезпечніших VPN-рішень
- Використовує криптографію з відкритими ключами для автентифікації та шифрування
- Підтримує багатофакторну автентифікацію та інтеграцію з SSO-провайдерами
- Детальні політики доступу (ACL) для контролю, які пристрої можуть взаємодіяти
- Можливість створення підмереж та груп пристроїв
- Підтримка тимчасового доступу та аудиту підключень

Концептуально це дуже схоже на те як працювали старі локальні ігрові кімнати, де купа комп’ютерів знаходились у одній локальній мережі щоб мати можливість підключатись один до одного, але, у випадку з Tailscale, локальна мережа емулюється, що прибирає необхідність у знаходженні в одній мережі.

При виконанні роботи для демонстрації можливостей використано варіант простого встановлення Tailscale.

Для встановлення необхідно виконати таку команду:

```
curl -fsSL https://tailscale.com/install.sh | sh
```

Ця команда завантажує та одразу виконує інсталяційний скрипт. Пояснення параметрів:

- `curl` – утиліта для отримання даних через HTTP.
- `-f` (`--fail`) – приховує повідомлення про помилки у випадку невдалого запиту.
- `-s` (`--silent`) – вимикає індикатор завантаження.
- `-S` (`--show-error`) – у разі помилки виводить повідомлення.
- `-L` (`--location`) – дозволяє переходити за HTTP-перенаправленнями.

- `https://tailscale.com/install.sh` – адреса офіційного інсталяційного скрипта.
- `| sh` – передає вміст скрипта на виконання інтерпретатору оболонки `sh`.

Після встановлення необхідно активувати клієнтське з'єднання командою:

```
sudo tailscale up
```

Після її виконання буде згенеровано посилання для авторизації, яке слід відкрити у веб-браузері. Під час реєстрації можна скористатися обліковим записом Google, GitHub, Microsoft або іншим підтримуваним методом. Після підтвердження пристрій буде підключено до мережі Tailnet.

Для забезпечення стабільної роботи клієнта Tailscale після перезавантаження системи доцільно виконати додаткові команди:

```
# Перевірка статусу служби

sudo systemctl status tailscaled

# Увімкнення автозапуску

sudo systemctl enable tailscaled

# Запуск служби вручну (якщо не запущено автоматично)

sudo systemctl start tailscaled
```

Хоча ці кроки є необов'язковими для одноразового запуску, їх виконання значно підвищує надійність системи в довгостроковій перспективі. У випадках оновлення або планових перезавантажень, служба `tailscaled` буде запускатися автоматично, зберігаючи стабільне з'єднання без потреби у втручанні користувача.

Після підключення пристроїв до Tailnet, вони автоматично отримують внутрішні IP-адреси, які дозволяють організувати доступ до сервісів у віртуальній приватній мережі.

Список підключених пристроїв доступний у вебінтерфейсі Tailscale за адресою https://login.tailscale.com/admin/machines

Кожен пристрій отримає адресу, наприклад: `100.101.102.103`

У разі, якщо певний застосунок працює на сервері (наприклад, Raspberry Pi) на порту 1234, до нього можна звернутися за допомогою URL: `http://100.101.102.103:1234`

Цей доступ можливий як у межах локальної мережі, так і ззовні — через Tailnet, що особливо важливо для мобільного або віддаленого адміністрування сервера. Приклад списку пристроїв зображений на рисунку 1.2.

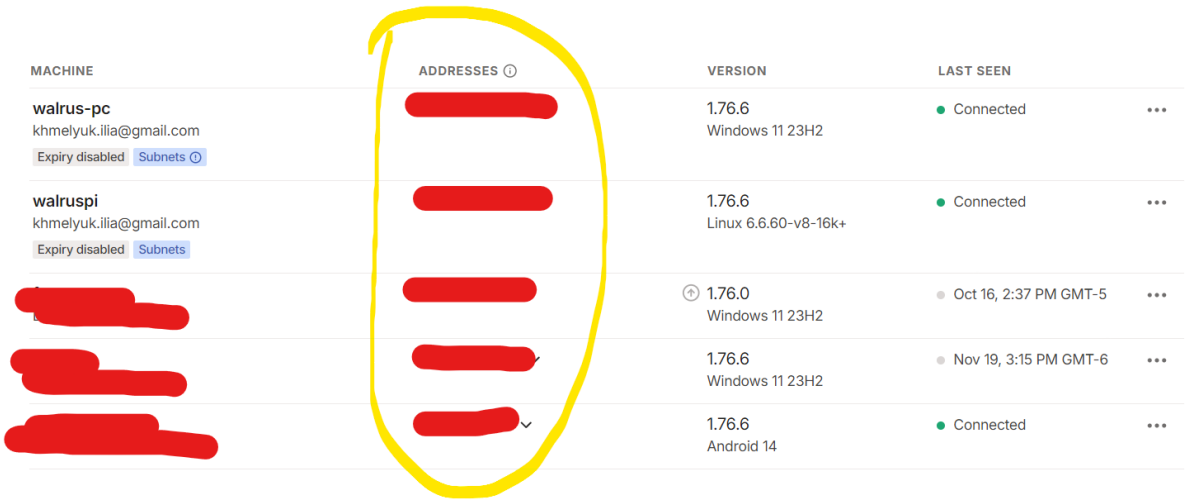


Рис 1.2 Перелік пристроїв підключених до Tailnet

У межах реалізованого середовища одним з основних завдань було забезпечення доступу до сервісів, які працюють на одному пристрої, з іншого віддаленого пристрою, що не перебуває в локальній мережі. Наприклад, умовний вебсервіс розміщено на пристрої з назвою `walruspi`. Для того, щоб отримати до нього доступ з іншого пристрою `walrus-pc`, який знаходиться в іншому географічному розташуванні, необхідно, щоб обидва пристрої були підключені до однієї віртуальної мережі Tailnet за допомогою Tailscale [2].

Після активації Tailscale на обох пристроях та успішного їх з'єднання через VPN, доступ до сервісу здійснюється за віртуальною IP-адресою, що надається Tailscale. Наприклад: `http://100.101.102.103:1234`

де `1234` — порт, на якому працює сервіс, а `100.101.102.103` — віртуальна IP-адреса, закріплена за `walruspi`.

Окрім прямого з’єднання за віртуальною адресою, Tailscale також підтримує функцію створення підмережі (subnet routing). Це дозволяє одному пристрою виступати в ролі маршрутизатора, через який відкривається доступ до всієї локальної мережі, в якій він знаходиться.

На вебінтерфейсі https://login.tailscale.com/admin/machines, пристрої з увімкненою функцією Subnet Router позначаються маленьким синім прапорцем з підписом Subnets, як на рисунку 1.3. У наведеному прикладі пристрій `walruspi` виступає в ролі підмережевого маршрутизатора.

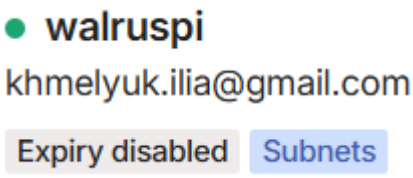


Рис 1.3 Пристрій `walruspi` , що може працювати у ролі підмережевого маршрутизатора

Це означає, що всі запити, адресовані локальним IP-адресам (наприклад, `http://1.1.1.1:1234`), пересилаються через пристрій `walruspi`, який виступає посередником між зовнішнім клієнтом та локальним сервісом. Таким чином, не потрібно знати або вводити віртуальні IP-адреси кожного окремого пристрою — достатньо мати правильну локальну адресу, як у традиційній LAN.

Переваги такого підходу:

- Можливість підключення до декількох пристроїв у локальній мережі без встановлення Tailscale на кожному з них.
- Мінімальна зміна архітектури вже існуючої інфраструктури.
- Простіший доступ до внутрішніх сервісів через знайомі локальні адреси.

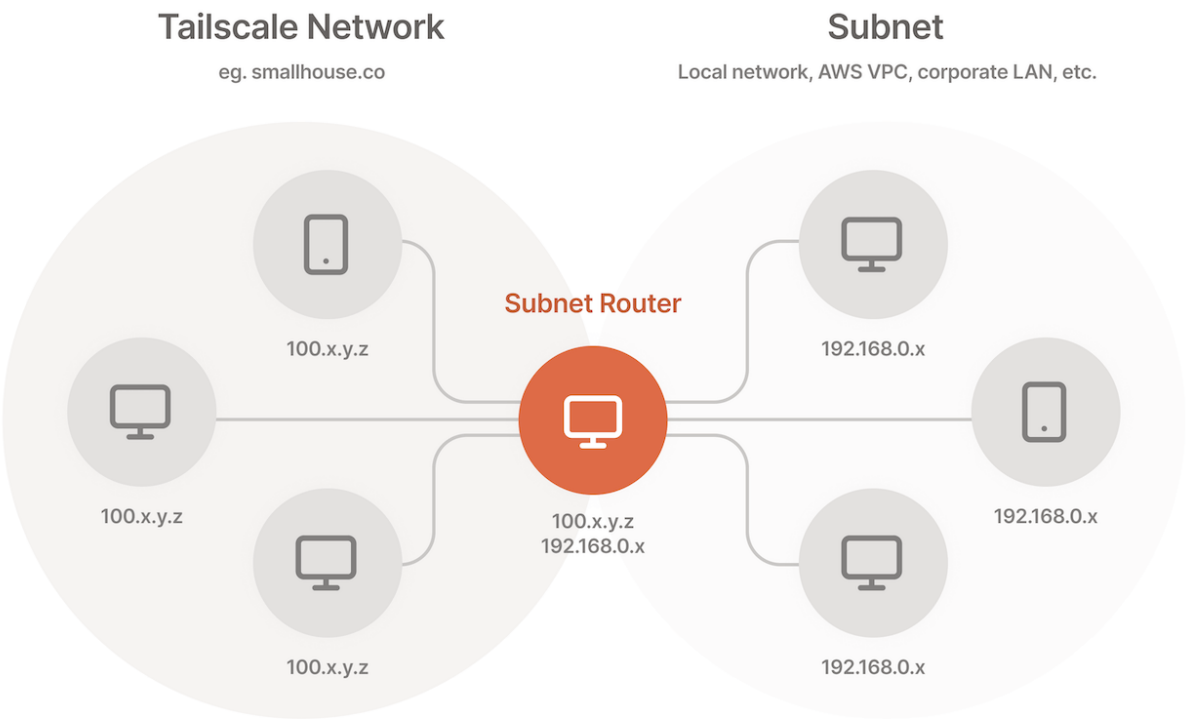


Рис 1.4 Спрощена схема роботи підмережевого маршрутизатора Tailscale [1]

Процес налаштування такого маршрутизатора в системі на базі Linux передбачає виконання наступних дій:

У Linux для пересилання мережевого трафіку між інтерфейсами необхідно дозволити функцію IP forwarding. Для цього слід виконати такі команди:

```
echo 'net.ipv4.ip_forward = 1' | sudo tee -a /etc/sysctl.d/99-tailscale.conf
echo 'net.ipv6.conf.all.forwarding = 1' | sudo tee -a /etc/sysctl.d/99-tailscale.conf
sudo sysctl -p /etc/sysctl.d/99-tailscale.conf
```

Пояснення кожної з команд:

- IPv4 forwarding:
- `net.ipv4.ip_forward = 1` — дозволяє пересилання IPv4-пакетів між інтерфейсами.

- `tee -a` — додає параметр у кінець конфігураційного файлу `/etc/sysctl.d/99-tailscale.conf`, який призначений для постійного зберігання параметрів ядра.
- IPv6 forwarding:
- `net.ipv6.conf.all.forwarding = 1` — активує аналогічну функціональність для IPv6-трафіку.
- Застосування змін:
- `sysctl -p` — негайно завантажує нові параметри з вказаного файлу без потреби перезавантаження системи.

Завдяки збереженню конфігурації у `/etc/sysctl.d/`, зміни будуть автоматично застосовані при кожному запуску системи.

Після активації forwarding необхідно оголосити маршрут до локальної мережі іншим учасникам Tailnet за допомогою наступної команди [2]:

```
sudo tailscale up --advertise-routes=192.0.2.0/24
```

У цій команді:

- `--advertise-routes=192.0.2.0/24` — означає, що пристрій буде виступати маршрутизатором до підмережі з адресами від `192.0.2.1` до `192.0.2.254`.
- Частину `192.0.2.0/24` необхідно замінити на відповідну до локальної мережі користувача. Наприклад, якщо IP-адреса пристрою `10.0.0.22`, і потрібно охопити всю підмережу, слід вказати `10.0.0.0/24`.

Після виконання цієї команди:

- Tailscale оголосить відповідний маршрут.
- У панелі керування Tailnet (`https://login.tailscale.com/admin/machines`) біля пристрою з’явиться синій індикатор Subnets, який свідчить про активне рекламування підмережі.
- Для завершення налаштування адміністратор повинен схвалити запропонований маршрут вручну у вебінтерфейсі Tailscale (рисунок 1.5). [1]

Після виконання команди `--advertise-routes` пристрій отримує статус маршрутизатора, що видно у вебінтерфейсі Tailnet. Біля нього з'являється синій прапорець з позначкою "Subnets", що вказує на активну спробу рекламування підмережі. Водночас, поряд з цим прапорцем може з'явитися знак оклику, який означає, що маршрут ще не підтверджено, і підмережа наразі не є дійсною [2].

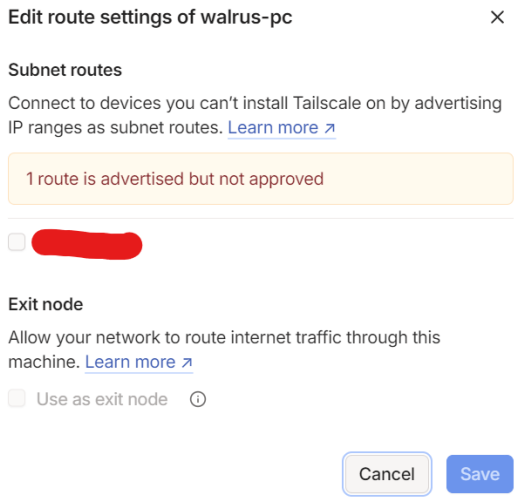


Рис 1.5 Перелік IP адрес що обрані для форвардингу мережевим маршрутизатором

Щоб активувати рекламований маршрут, необхідно виконати наступні дії у Tailnet:

1)Перейти до сторінки пристроїв: https://login.tailscale.com/admin/machines.

2)Натиснути на три крапки поруч із пристроєм, який виконує роль маршрутизатора.

3)Обрати пункт "Edit route settings" (Редагувати налаштування маршруту).

4)У вікні, що з'явиться, поставити прапорець навпроти діапазону IP-адрес, який потрібно активувати.

5)Зберегти зміни.

Після виконання вказаних дій підмережа буде підтверджена адміністратором і стане доступною для інших пристроїв у мережі Tailscale.

Використання Subnet Router має низку важливих переваг:

- Можливість доступу до пристроїв, на які неможливо встановити Tailscale, наприклад: побутові IoT-пристрої (розумні лампи, чайники, пилососи тощо), які підключаються до домашньої локальної мережі.
- Спрощення архітектури мережі — інші пристрої Tailnet отримують єдиний доступ до всієї локальної інфраструктури через один маршрутизатор.
- Менша кількість інсталяцій — немає потреби встановлювати Tailscale на кожному клієнті в локальній мережі.

Tailscale також надає ряд розширених функцій, які можуть бути корисними в інших проєктах або сценаріях:

- Whitelist / Blacklist – обмеження доступу до пристроїв або сервісів.
- Exit Nodes – перенаправлення всього трафіку через конкретний пристрій Tailnet.
- Magic DNS – автоматична роз resolución імен пристроїв у мережі.
- ACL (Access Control Lists) – керування правами доступу на рівні мережі.

Оскільки ці функції не є критично важливими для реалізації цілей даної роботи, вони мають факультативний характер. За потреби, ознайомитися з їх описом та прикладами використання можна у **офіційній документації**, що доступна безпосередньо у Tailnet через вкладки кожної функції.

### 1.2. Встановлення Docker та створення середовища для контейнерів

**Docker** — це програмна платформа, що дозволяє створювати, розгортати та запускати додатки в ізольованому середовищі, яке називається **контейнером**. Контейнер можна уявити як легкий автономний блок, що містить усі необхідні компоненти для роботи програми, зокрема [3]:

- виконуваний код;
- залежності та бібліотеки;

- конфігураційні файли;
- системне середовище.

У межах цієї роботи більшість сервісів реалізовано у вигляді окремих контейнерів. Це дозволяє забезпечити стабільність, масштабованість і простоту розгортання.

Нижче наведено покрокову інструкцію з встановлення Docker на пристрій під управлінням операційної системи Linux (у даному випадку — Raspberry Pi). Перед встановленням необхідно переконатися, що система оновлена:

```
sudo apt update
sudo apt upgrade -y
```

Ці команди оновлюють список доступних пакетів і встановлюють останні версії вже наявного програмного забезпечення.

Далі потрібно інсталювати базові пакети, необхідні для додавання репозиторіїв та перевірки цифрових підписів:

```
sudo apt install -y \
     apt-transport-https \
     ca-certificates \
     curl \
     gnupg2 \
     software-properties-common
```

Пояснення ключових компонентів:

- `apt-transport-https` — дозволяє завантажувати пакети через захищене з’єднання HTTPS.
- `ca-certificates` — містить сертифікати центрів сертифікації для перевірки SSL/TLS.
- `curl` – утиліта для передачі даних, використовується для отримання скриптів та ключів.
- `gnupg2` – забезпечує перевірку цифрових підписів, необхідних для завантаження з офіційного Docker-репозиторію.
- `software-properties-common` — містить інструменти для керування джерелами програмного забезпечення, зокрема `add-apt-repository`.

Після підготовки системи, Docker можна встановити через офіційний інсталяційний скрипт:

```
curl -fsSL https://get.docker.com | sh
```

Цей скрипт автоматизує процес додавання репозиторію та встановлення всіх необхідних компонентів Docker Engine.

Щоб мати змогу запускати Docker-команди без префіксу `sudo`, поточного користувача слід додати до групи `docker`:

```
sudo usermod -aG docker $USER
```

Ця команда додає поточного користувача до відповідної групи, зміни вступають в силу після перезавантаження.

Щоб забезпечити автоматичний запуск Docker після перезавантаження системи:

```
sudo systemctl enable docker
```

Після чого потрібно перезапустити пристрій:

```
sudo reboot
```

Після перезавантаження можна перевірити, чи користувач успішно доданий до групи `docker` за допомогою команди `groups`.

Перевірку буде пройдено якщо у переліку груп в яких присутній користувач є група `docker`

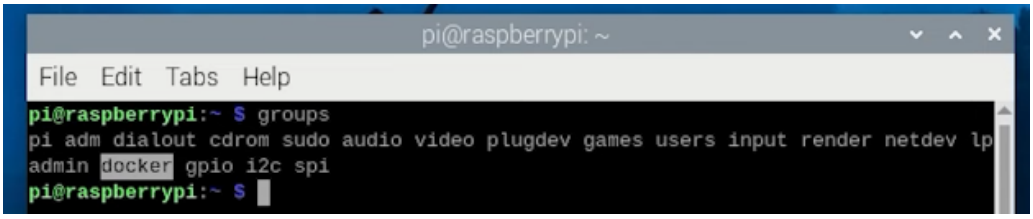


Рис 1.6 Група `docker` відображається у панелі `groups`

Після виконання вказаних кроків, Docker повністю готовий до роботи. Подальше розгортання сервісів здійснюється у контейнерах, що значно спрощує їх налаштування, масштабування та оновлення.

### 1.3. Portainer як графічна оболонка Docker

Portainer є додатковим, але надзвичайно зручним компонентом у структурі побудови приватного серверного середовища. Це вебінтерфейс для візуального керування Docker-контейнерами, який значно спрощує роботу з інфраструктурою та надає широкий набір функцій. Незважаючи на те, що використання Portainer не є обов'язковим, його інтеграція дозволяє спростити взаємодію з Docker і зменшити залежність від командного рядка [4].

Основні можливості Portainer:

- створення, запуск і зупинка контейнерів;
- моніторинг ресурсів та журналювання;
- керування Docker-мережами, томами та образами;
- підтримка стеків Docker Compose;
- доступ до керування контейнерами з будь-якого пристрою через браузер.

Переваги інтеграції Portainer:

- Інтуїтивно зрозумілий інтерфейс, що знижує поріг входу для нових користувачів;
- Візуалізація взаємозв'язків між контейнерами та сервісами;
- Доступ з будь-якого місця локальної або віддаленої мережі через вебінтерфейс;
- Економія часу — виконання більшості дій у декілька кліків.

*Покрокове встановлення Portainer*

Portainer, як і більшість компонентів у цьому проєкті, розгортається у вигляді окремого Docker-контейнера. Процес складається з кількох послідовних етапів:

Завантаження образу Portainer:

```
docker pull portainer/portainer-ce:latest
```

Ця команда завантажує останню стабільну версію Portainer Community Edition.

Створення Docker-об'єму для збереження даних:

```
docker volume create portainer_data
```

Том використовується для зберігання конфігурацій Portainer і забезпечує збереження налаштувань після перезавантаження.

Запуск контейнера Portainer:

```
docker run -d -p 8000:8000 -p 9443:9443 --name=portainer --restart=always \
-v /var/run/docker.sock:/var/run/docker.sock \
-v portainer_data:/data \
portainer/portainer-ce:latest
```

Пояснення параметрів:

- `-d` — фоновий режим;
- `-p 8000:8000` і `-p 9443:9443` — проброс портів для доступу;
- `--restart=always` — автоматичний запуск контейнера після перезавантаження;
- `-v /var/run/docker.sock:/var/run/docker.sock` — надає Portainer доступ до Docker-сокету для керування іншими контейнерами;
- `-v portainer_data:/data` — монтований том для зберігання даних.

1. Доступ до вебінтерфейсу:

Після запуску вебінтерфейс Portainer стає доступним за адресою:

```
https://<IP-адреса Raspberry Pi>:9443
```

(наприклад, `https://192.168.1.100:9443`). При локальному використанні можна вказати `localhost`.

При першому вході користувачу буде запропоновано створити обліковий запис адміністратора (рис 1.7).

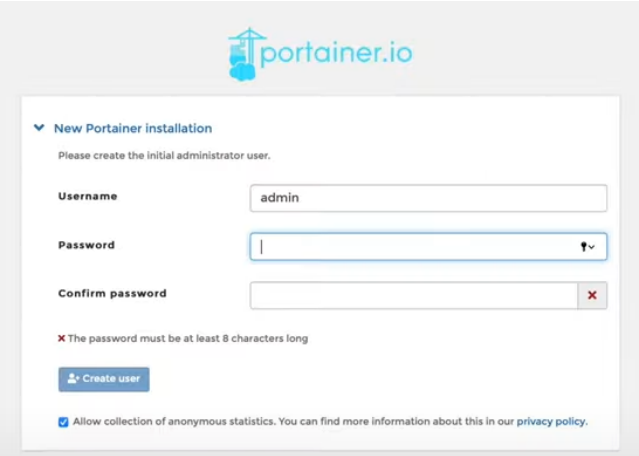


Рис 1.7 Першочергова автенцифікація у portainer

Після цього з’явиться запит на вибір серед декількох способів підключення (рис 1.8) — у межах цієї роботи необхідно обрати Docker Environment, який було розгорнуто раніше.

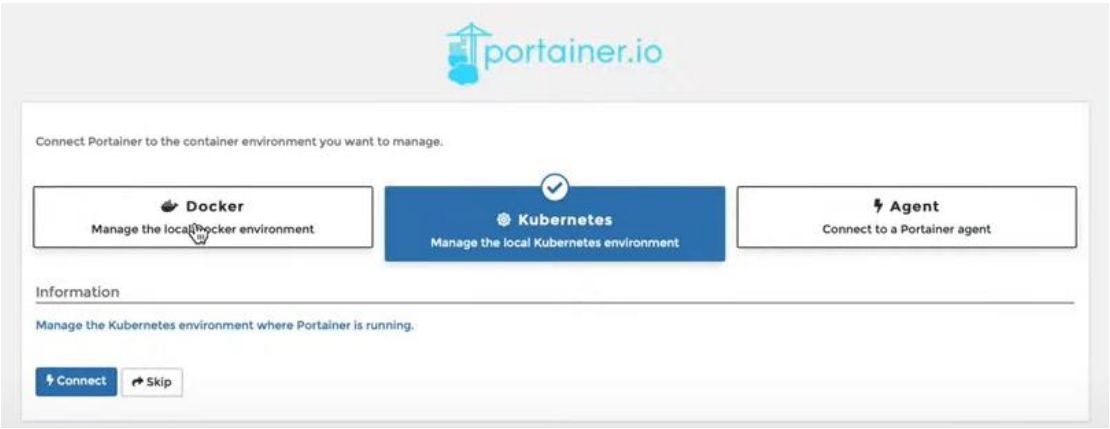


Рис 1.8 Меню вибору способу підключення portainer

У наступному діалоговому вікні система нагадає про необхідність надати Portainer доступ до Docker-сокета (рис 1.9) — це вже реалізовано в попередньому кроці через параметр -v /var/run/docker.sock:/var/run/docker.sock.


Рис 1.9 Нагадування про надання portainer прав з керування даними docker

**Висновок**

Контейнеризація та сучасні мережеві технології відіграють ключову роль у створенні гнучких, безпечних і масштабованих IT-інфраструктур. В рамках першого розділу було проведено дослідження концепції приватного VPN-доступу на основі Tailscale, який дозволяє організувати захищену взаємодію між пристроями без складного ручного налаштування. Архітектура zero-config mesh VPN забезпечує підвищену зручність і безпеку, що особливо актуально для малопотужних пристроїв на базі ARM.

Паралельно з цим було створено основу для розгортання сервісів у контейнерах за допомогою Docker, що дає змогу ефективно керувати ресурсами, ізолювати додатки та автоматизувати обслуговування. Інтеграція з Portainer як зручним веб-інтерфейсом для адміністрування контейнерів істотно спрощує процес розгортання й моніторингу.

Загалом, інструменти, розглянуті в цьому розділі, створюють потужну базу для подальшого впровадження самостійно розміщених (self-hosted) сервісів, які не залежать від сторонніх хмарних рішень та можуть працювати автономно в межах домашньої або корпоративної мережі.

# РОЗДІЛ 2
# РОЗГОРТАННЯ ТА ІНТЕГРАЦІЯ ЛОКАЛЬНИХ СЕРВІСІВ

## 2.1. Grocy — система обліку продуктів

Grocy — це вебзастосунок з відкритим кодом, призначений для управління домашнім господарством, зокрема: обліку продуктів харчування, планування покупок, прибирання, розподілу обов'язків тощо. Спочатку потреба у такому сервісі може здаватися необов'язковою, проте регулярне використання швидко демонструє його ефективність, особливо в контексті спільного проживання або ведення домашнього складу [5].

На ілюстрації інтерфейсу Grocy з рисунку 2.1 видно, що в холодильнику закінчилися яйця, лаваш і желатин, а хліб та шинка прострочені. Продукти, позначені як "необхідні", автоматично додаються до списку покупок (рис 2.2) після споживання або псування — цей список синхронізується з мобільним застосунком, що дозволяє мати його під рукою під час відвідування магазину.

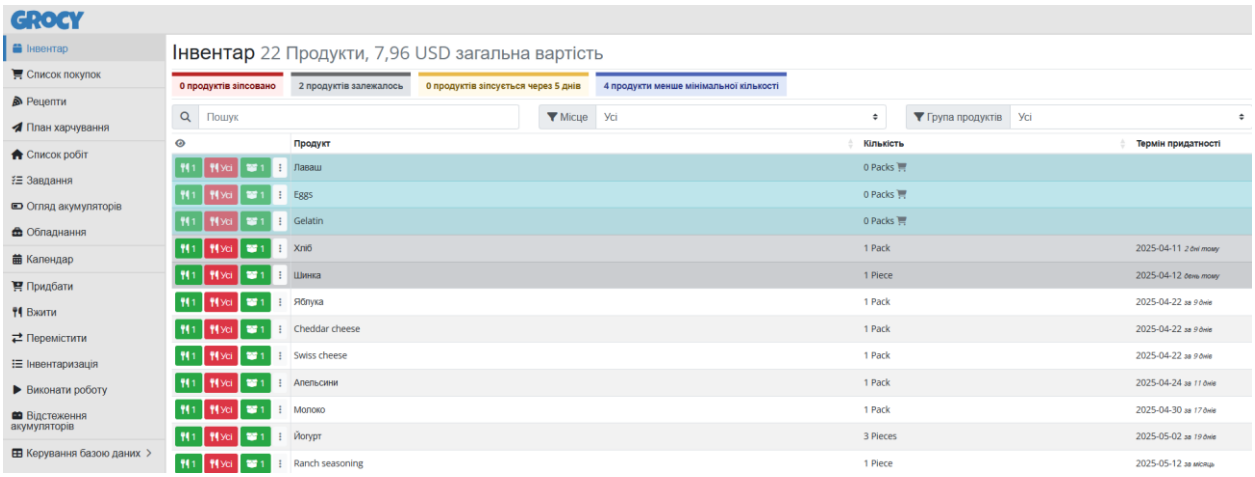


Рис 2.1 Веб інтерфейс інвентарю у Grocy

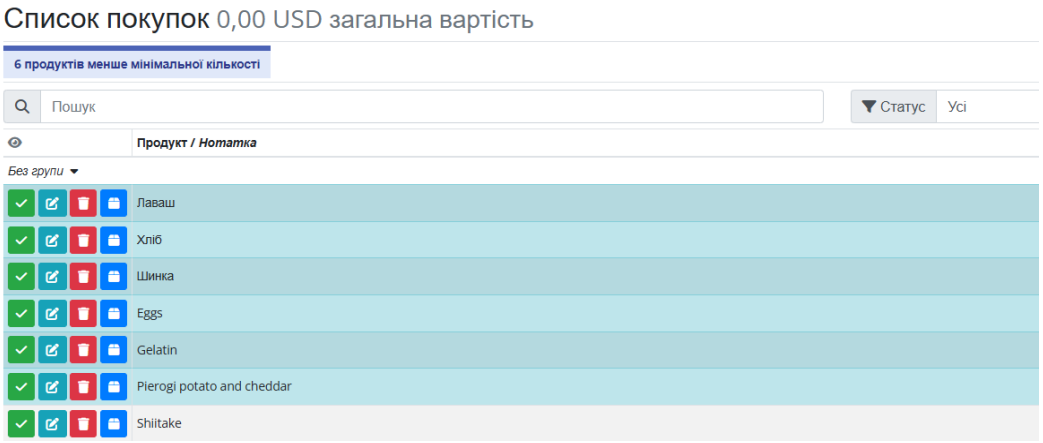


Рис 2.2 Веб інтерфейс списку покупок у Grocy

Grocy дуже гнучкий у застосуванні та налаштуванні. Він дозволяє розподіляти хатні обов'язки та планувати заходи. Тож буде незамінним у гуртожитках чи співмешканнях.

Інсталяція Grocy у цьому проєкті виконується за допомогою Portainer, що значно спрощує процес розгортання завдяки можливості використання стеків (Stacks). Стек — це зручна форма для запуску багатоконтейнерних застосунків, що базується на синтаксисі docker-compose (рис 2.3 та 2.4).

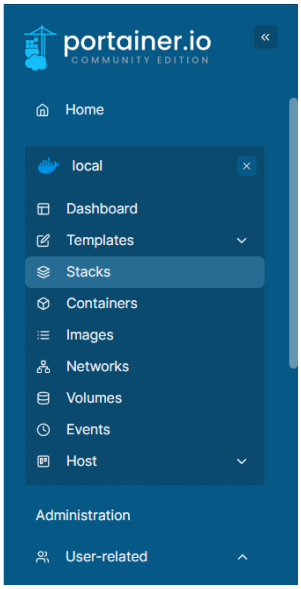


Рис 2.3 Навігація у інтерфейсі portainer

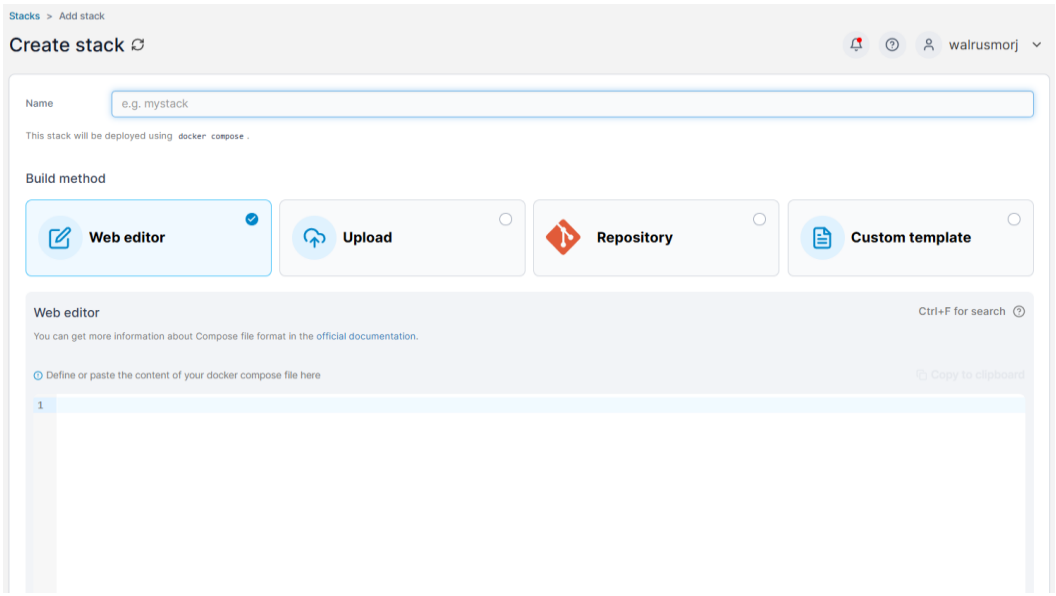


Рис 2.4 Меню створення Stack

Для встановлення треба пройти через кілька простих кроків:

1) У вебінтерфейсі Portainer необхідно обрати пункт "Stacks" та натиснути "Create new stack".

2) У вікні створення слід вказати ім’я майбутнього стеку (наприклад, `grocy`) та вставити наступний код:

```
services:
  grocy:
    image: lscr.io/linuxserver/grocy:latest
    container_name: grocy
    environment:
      - PUID=1000
      - PGID=1000
      - TZ=Europe/Kyiv
    volumes:
      - /home/користувач/Documents/Config/Grocy:/config
```

```
ports:
  - 9283:80
restart: unless-stopped
```

Пояснення параметрів:

- `PUID` і `PGID` — ідентифікатори користувача, від імені якого працює контейнер.
- `TZ` — часова зона.
- `volumes` — шлях до каталогу, де зберігається конфігурація Grocy.
- `ports` — проброс порту: в даному випадку Grocy буде доступний на порту 9283.
- `restart: unless-stopped` — автоматичний запуск контейнера після перезавантаження, якщо його не зупинено вручну.

Після натискання кнопки "Deploy the stack" (рис 2.5), система автоматично завантажить образ і запустить контейнер. Якщо статус контейнера “running” як на рисунку 2.6, то процес інсталяції був проведений коректно.

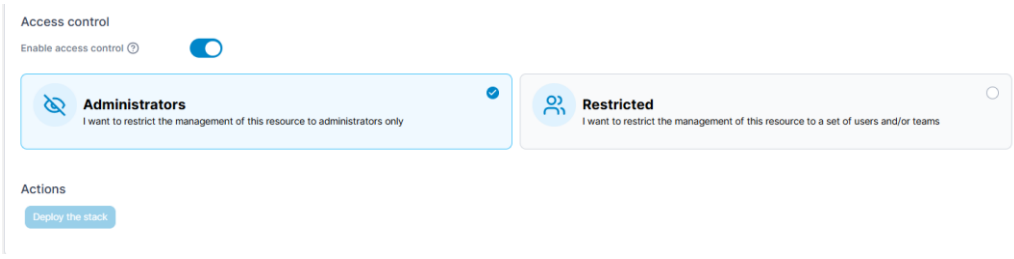


Рис 2.5 Додаткові налаштування створення Stack

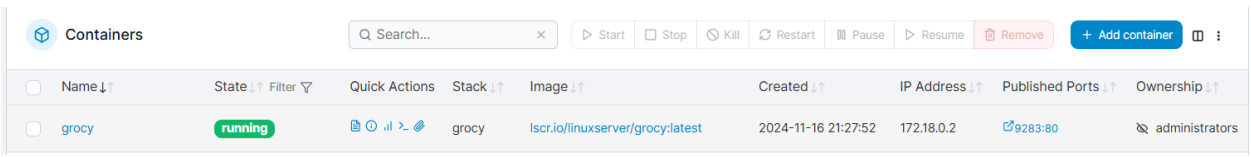


Рис 2.6 Демонстрація активного контейнеру Grocy

*Примітка: Порт 9283 можна змінити у випадку, якщо він уже зайнятий іншим застосунком. Аналогічно, слід адаптувати шлях до конфігурації відповідно до структури файлової системи на конкретному пристрої.*

Для зручного доступу до сервісу з мобільних пристроїв можна встановити мобільний застосунок Grocy, який доступний у:

- Google Play (Рис 2.7)
- Apple App Store
- F-Droid (Рис 2.8)

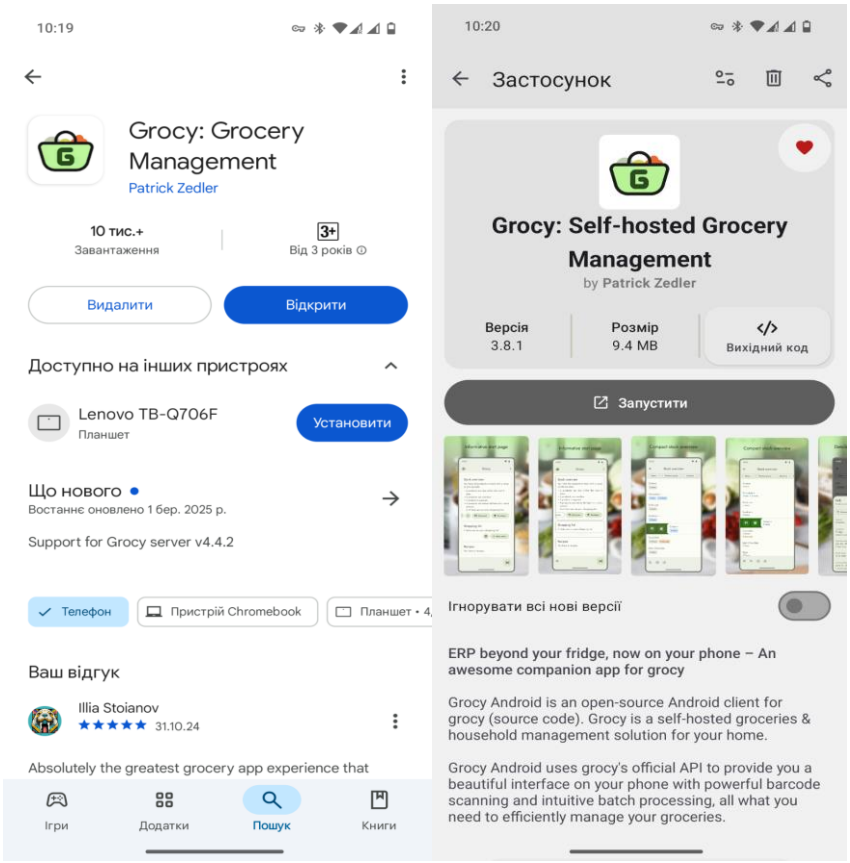


Рис 2.7 та Рис 2.8 Доступність застосунку Grocy на мобільних платформах

Після запуску застосунок запропонує використати демонстраційний сервер або вказати власний (Рис 2.9). У межах цього проєкту використовується локальний сервер, тому слід обрати власну інсталяцію.

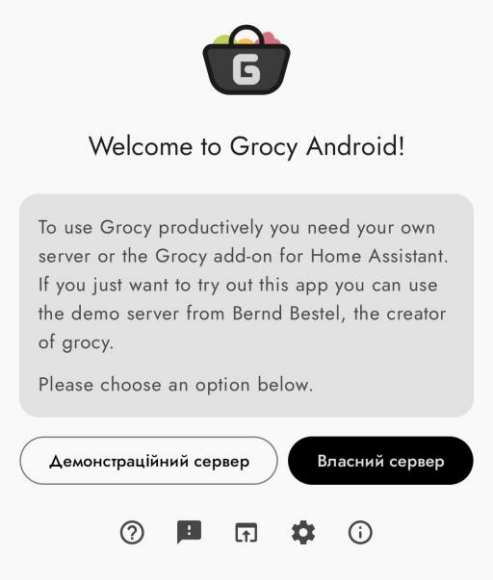


Рис 2.9 Стартове діалогове вікно Grocy

Для підключення мобільного застосунку необхідно створити API-ключ. Це виконується за адресою: `http://localhost:9283/manageapikeys`

На цій сторінці створюється новий ключ, після чого за допомогою кнопки з піктограмою QR-коду генерується код для сканування мобільним пристроєм. Після цього застосунок отримає доступ до функціональності сервісу.

Grocy працює локально, але зазвичай його зручно використовувати саме в магазині. Для цього забезпечено доступ до сервера з будь-якого місця через Tailscale.

Оскільки в попередніх розділах уже налаштовано підмережевий маршрутизатор, достатньо встановити Tailscale на мобільний пристрій, увійти у власний акаунт та запустити з’єднання.

Після цього мобільний застосунок Grocy зможе взаємодіяти з сервером, що розміщений у домашній мережі, без необхідності в публічному IP чи складному налаштуванні NAT.

Grocy є прикладом того, як навіть менш критичні сервіси можуть суттєво покращити побут. Його застосування дозволяє вести облік продуктів, уникати марнотратства та планувати покупки. У поєднанні з мобільним доступом і

автоматизацією контейнерного розгортання, цей інструмент стає ефективним компонентом цифрової екосистеми домашнього сервера.

### 2.2. Sunshine + Moonlight — реалізація віддаленого доступу

У межах реалізованого проєкту виникла потреба в забезпеченні високоякісного віддаленого доступу до потужного персонального комп’ютера з мобільних пристроїв або менш продуктивних ноутбуків. Після тривалого досвіду використання комерційних хмарних рішень — таких як Shadow PC, GeForce Now та Boosteroid — було прийнято рішення створити власну систему віддаленого доступу на основі рішень з відкритим кодом: Sunshine (сервер) та Moonlight (клієнт) [6] [7].

Sunshine і Moonlight — це потужні інструменти для віддаленого доступу до комп’ютера, які дозволяють використовувати його ресурси з мобільних пристроїв або інших комп’ютерів. На рисунку 2.10 зображено схематичну модель взаємодії клієнта та хоста.

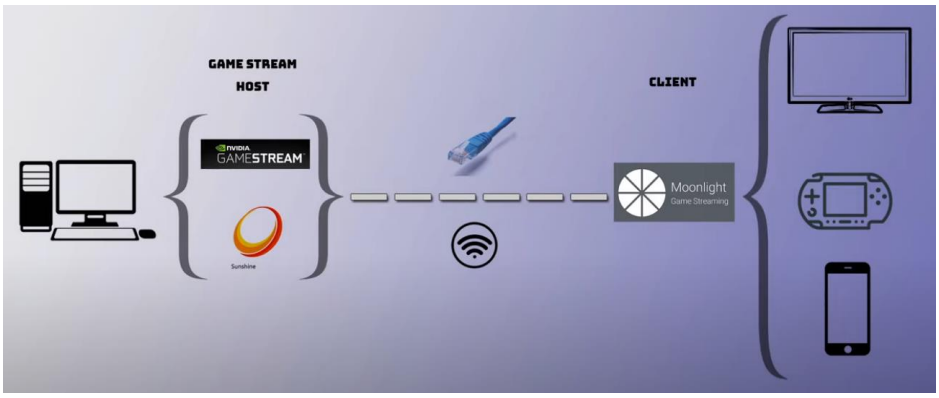


Рис 2.9 Схема взаємодії Sunshine та Moonlight

Sunshine — серверна частина, яка встановлюється на комп’ютер, що транслює зображення. Підтримує трансляцію як усього робочого столу, так і окремих програм. Використовує апаратне прискорення кодування відео [7].

Moonlight — клієнтський застосунок для Android, iOS, Windows, Linux і навіть SmartTV, що підключається до Sunshine-сервера та транслює зображення на свій пристрій[6].

Разом Sunshine і Moonlight створюють ефективну систему для віддаленого використання потужностей комп’ютера, забезпечуючи гнучкість і зручність у доступі до ресурсів.

Переваги над іншими протоколами (RDP, VNC, TeamViewer)

Sunshine + Moonlight вигідно відрізняються за такими параметрами [2]:

- Менша затримка завдяки ефективній архітектурі передачі даних та використанню UDP.
- Декодування на боці хоста, що знижує навантаження на клієнт.
- Вища якість відео завдяки сучасним кодекам (наприклад, HEVC).
- Мінімальна кількість підтверджень і швидке з’єднання.

Ці властивості роблять систему придатною навіть для віддаленого запуску ресурсоємних застосунків, включно з відеоіграми.

Ці переваги роблять Sunshine і Moonlight відмінним вибором для користувачів, які прагнуть максимально ефективно використовувати ресурси свого комп'ютера з мобільних пристроїв, забезпечуючи при цьому високу якість та швидкість роботи.

Для встановлення Sunshine необхідно перейти до офіційного репозиторію (https://github.com/LizardByte/Sunshine) та завантажити відповідну версію інсталяційного пакету (Рис 2.11). Після чого встановити її на комп’ютер.

Варто зауважити що це програма хост, і встановлювати її треба на високопотужний пристрій, для найкращого ефекту. Однак якщо встановити її на ноутбук, чи навіть Raspberry PI, віддалений доступ має працювати без жодних проблем.

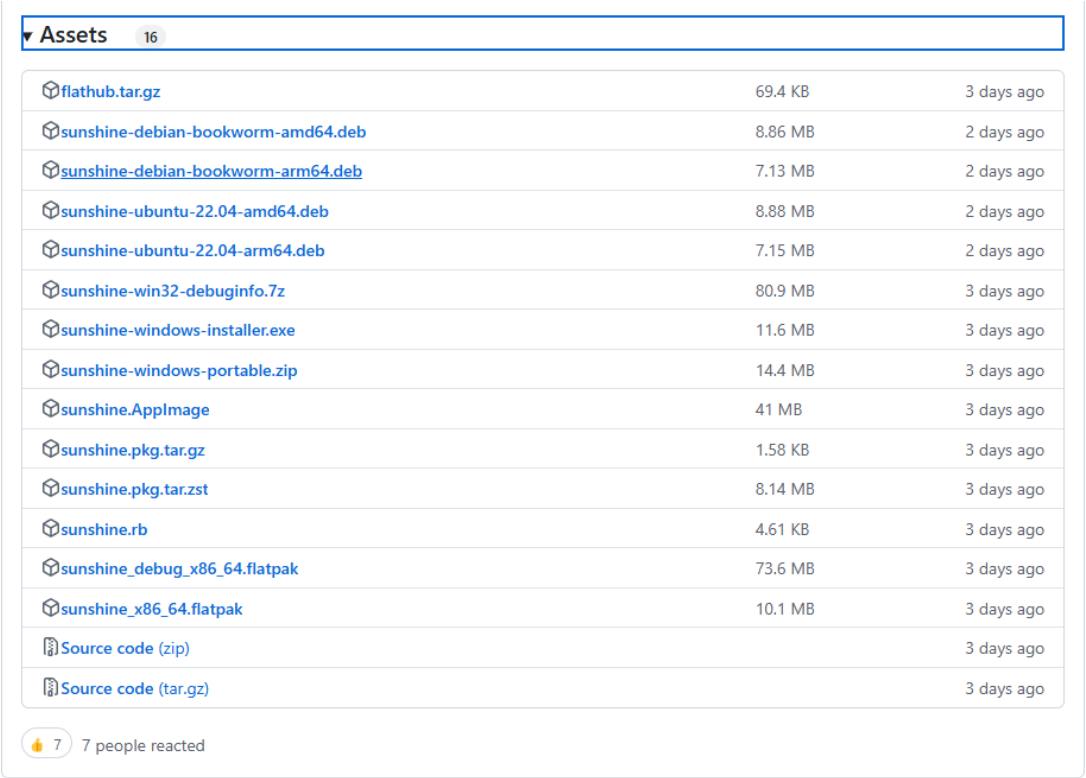


Рис 2.11 Різні пакети завантажень Sunshine для різних систем [7]

Moonlight можна встановити з офіційного джерела https://github.com/moonlight-stream), або з магазинів додатків: Google Play, App Store, F-Droid, якщо клієнтський пристрій це смартфон або планшет.

Після запуску Sunshine налаштування відбувається у вебінтерфейсі за адресою: `https://<IP-адреса-хоста>:47990`

При першому запуску буде запропоновано створити адміністративний пароль та задати ім’я пристрою.

Далі, для підключення на клієнтському пристрої треба запустити Moonlight, натиснути піктограму ПК зі знаком +, ввести IP-адресу хоста, після чого система згенерує PIN-код.

Цей PIN-код треба ввести в інтерфейсі Sunshine для підтвердження з’єднання. Після першого підключення пристрої запам’ятають один одного.

На рисунку 2.12 зображено налаштування що роблять досвід використання кращим.

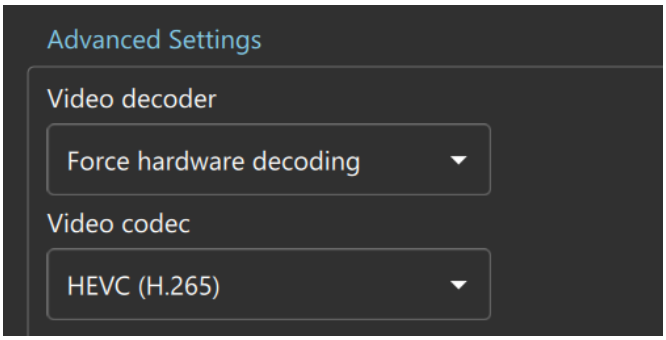


Рис 2.12 Рекомендовані просунуті налаштування Moonlight [6]

Ці налаштування переносять декодування на бік хоста та задіюють більш сучасний кодек, що зробить досвід використання набагато кращим.

*Проблема: комп'ютер має бути увімкнений*

Для того щоб мати можливість підключатися до хостового комп'ютера в будь-який час, необхідно забезпечити його активацію на відстані. Це реалізується за допомогою технології Wake-on-LAN (WOL).

Налаштування Wake-on-LAN у BIOS (залежить від виробника):

- AMI BIOS
  - Натиснути `Del` або `F2` при запуску системи.
  - Перейти до розділу *Power Management*.
  - Увімкнути параметр *Wake On LAN* або *Power On by PCI-E*.
- Award BIOS
  - Натиснути `Del` під час запуску.
  - Розділ *Power Management Setup* → *Wake On PCI Card* → Enabled.
- UEFI BIOS
  - Вхід через `F2`, `Del`, або `Esc`.
  - Перейти до *Advanced* або *Power Settings*.
  - Увімкнути *Wake On LAN*.

Далі потрібна перевірка налаштувань у системі Linux або Windows. Для цього треба *відкрити властивості мережевого адаптера та увімкнути параметри* «Allow this device to wake the computer» та «Only allow a magic packet to wake the computer».

Після активації функції Wake-on-LAN, Moonlight автоматично зможе надсилати *magic packet*, який пробуджує комп'ютер з режиму сну (рис 2.13). Це дозволяє повністю дистанційне керування комп'ютером без фізичного втручання.


Рис 2.13 Функція запиту на пробудження пристрою через протокол WOL

### 2.3. Nextcloud — локальне файлове сховище з інтеграцією SD-носія

Nextcloud — це система керування особистим хмарним сховищем з відкритим кодом, що дозволяє зберігати файли, синхронізувати дані між пристроями, керувати контактами, календарями, паролями та багатьма іншими типами інформації. У межах цього проєкту Nextcloud використовується як ключовий компонент системи для організації безпечного зберігання файлів у домашньому середовищі на базі Raspberry Pi.[8]

З міркувань ізоляції зберігання даних від системного розділу було прийнято рішення винести сховище Nextcloud на окремий носій. Для цього використано додаткову карту пам’яті microSD, встановлену в USB-картридер, що підключений до Raspberry Pi. Схематичне зображення макету ілюстровано на рисунку 2.14. Попри бюджетність такого рішення, воно забезпечує достатній рівень продуктивності, а за потреби може бути замінене на SSD або USB-флеш-накопичувач.

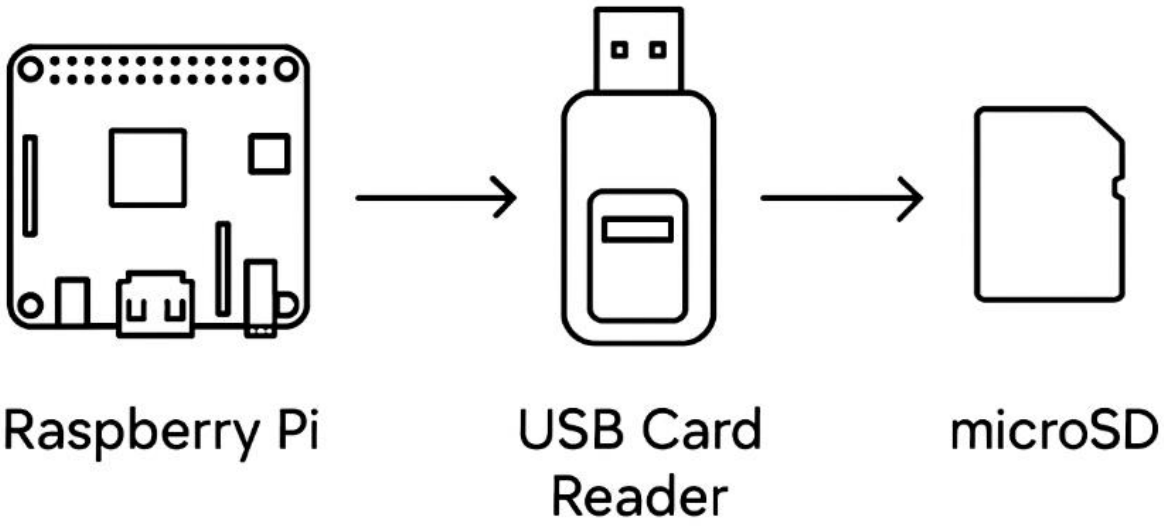


Рис 2.14 Схематичне зображення макету

Покрокове налаштування [2]

1) Перевірка файлової системи SD-карти:

```
df -Th | grep sda
```

Визначається тип файлової системи. Якщо це exFAT, потрібно переформатувати в ext4, бо exFAT не підтримує належну систему прав доступу. Можливе повідомлення: `type exfat` – вимагатиме форматування.

2) Форматування SD-карти в ext4 (ВИДАЛЯЄ ВСІ ДАНІ)

```
sudo umount /dev/sda1
sudo mkfs.ext4 -F /dev/sda1
```

Форматує SD-карту в ext4 — файлову систему, сумісну з правами доступу Linux.

Повідомлення: Якщо на пристрої є exFAT, буде потрібен ключ -F для примусового форматування.

3) Створення точки монтування

```
sudo mkdir -p /media/walrusmorj/morj
sudo chown -R walrusmorj:walrusmorj /media/walrusmorj/morj
```

Створюється директорія для монтування SD-карти.

Змінні: **walrusmorj** — ім'я користувача системи, **morj** — мітка або логічна назва для точки монтування.

4) Додавання запису в /etc/fstab для автоматичного монтування

```
sudo nano /etc/fstab
```

Відкриється файл для редагування, у ньому треба додати рядок:

```
/dev/sda1    /media/walrusmorj/morj    ext4    defaults    0    2
```

Після збереження:

```
sudo systemctl daemon-reload
sudo mount -a
```

Це дозволяє автоматичне монтування SD-карти при запуску системи.

Повідомлення: mount: `(hint)... systemd still uses the old version` — вирішується командою `systemctl daemon-reload`.

5) Створення директорії Nextcloud на SD-карті

```
mkdir -p /media/walrusmorj/morj/NextCloud
sudo chown -R 1000:1000 /media/walrusmorj/morj/NextCloud
sudo chmod -R 755 /media/walrusmorj/morj/NextCloud
```

Створення директорії для зберігання даних Nextcloud, із правильними правами для UID 1000 (звичайний користувач у системах Raspberry Pi).

6) Оновлення Docker Compose конфігурації

Раніше вже було показано як це швидко робиться через Portainer. Нижче наведено код що треба вставити у Stack:

```
version: '3'

services:
  nextcloud:
    image: nextcloud
    restart: always
    ports:
```

```
    - "8080:80"
  volumes:
    - /media/walrusmorj/morj/NextCloud:/var/www/html
  environment:
    - NEXTCLOUD_DATA_DIR=/var/www/html/data
  user: "1000:1000"
```

Змінні:

`/media/walrusmorj/morj/NextCloud` — шлях до директорії на SD-карті

`UID/GID 1000` — повинен відповідати UID користувача, під яким запускається Docker

7) Типові помилки та їх розв'язання:

| Крок | Повідомлення | Причина | Рішення |
|---|---|---|---|
| Форматування | `contains a exfat file system` | Карта вже відформатована в exFAT | Використати `-F` при форматуванні |
| Монтування | `Permission denied` | Недостатньо прав | `sudo chown`, `chmod` на директорії |
| Docker старт | `rsync chown failed` | Контейнер не має прав на `/var/www/html/data` | Встановити `user: "1000:1000"` у Compose |
| Веб-доступ | `Connection refused` | Сервіс не стартував або порт закритий | Перевірити `docker logs`, `ufw allow 8080` |

Після виконання всіх кроків Nextcloud буде успішно працювати з зовнішньою SD-картою як сховищем. Це дає можливість використовувати дешеві та доступні носії у якості серверного сховища, що ідеально для експериментальних або домашніх цілей. Більш того, він має дуже гнучкі налаштування та дозволяє ділитись файлами, та навіть створювати внутрішні теки до яких мають доступ інші користувачі (без можливості доступу до зовнішніх файлів), що може бути корисно для особистих тек куди студенти можуть здавати свої контрольні чи домашні роботи, як альтернатива Google Classroom.

## 2.4. Синхронізація нотаток на прикладі Joplin та WebDAV

У сучасних умовах цифрової гігієни та контролю над особистими даними все більше уваги приділяється самостійно розміщеним рішенням з відкритим кодом. Одним із важливих елементів такого підходу є створення, зберігання та синхронізація персональних нотаток без залежності від комерційних закритих сервісів, таких як Google Keep, Microsoft OneNote тощо.

Для реалізації цих задач у межах даного проєкту використано Joplin — багатоплатформовий застосунок для створення текстових нотаток, списків завдань, зображень та інших типів контенту з підтримкою Markdown-форматування, шифрування та WebDAV-синхронізації.

На відміну від комерційних аналогів, Joplin не має вбудованого централізованого сервера для зберігання даних. Замість цього, застосунок підтримує інтеграцію з будь-яким сумісним хмарним сховищем, що підтримує протокол WebDAV. Це відкриває можливість використання Nextcloud — встановленого раніше локального хмарного середовища, як цільової платформи для синхронізації.

Щоб налаштувати синхронізацію застосунку Joplin із Nextcloud, необхідно виконати кілька послідовних кроків. Спершу слід перейти до розділу «Configuration / Налаштування». У параметрах синхронізації потрібно обрати тип синхронізації — Nextcloud. Далі, у полі WebDAV URL необхідно ввести шлях до хмарного сховища, наприклад: `http://<IP-адреса>:8080/remote.php/dav/files/walrusmorj/Sync/Joplin/`.

У цьому прикладі `walrusmorj` — це ім’я користувача в Nextcloud, а `Sync/Joplin/` — директорія, у якій зберігатимуться дані застосунку. Перед початком синхронізації бажано створити відповідну папку Joplin у файловій системі Nextcloud — це не обов’язково, але значно полегшить організацію даних. На завершення слід ввести логін і пароль облікового запису Nextcloud для встановлення з’єднання.

У термінальній (CLI) версії Joplin треба виконати послідовність команд:

```
:config sync.5.path https://<ip>/nextcloud/remote.php/webdav/Joplin

:config sync.5.username YOUR_USERNAME

:config sync.5.password YOUR_PASSWORD

:config sync.target 5
```

Параметр `sync.target 5` вказує, що використовується WebDAV як тип синхронізації.

Після збереження налаштувань і активації синхронізації Joplin починає обмін даними з сервером Nextcloud. На рисунку 2.15 зображено створену тестову нотатку та її синхронізацію. Після успішної синхронізації у файловій системі Nextcloud з'являються оновлені файли (рис 2.16) — це свідчить про коректну роботу механізму синхронізації.

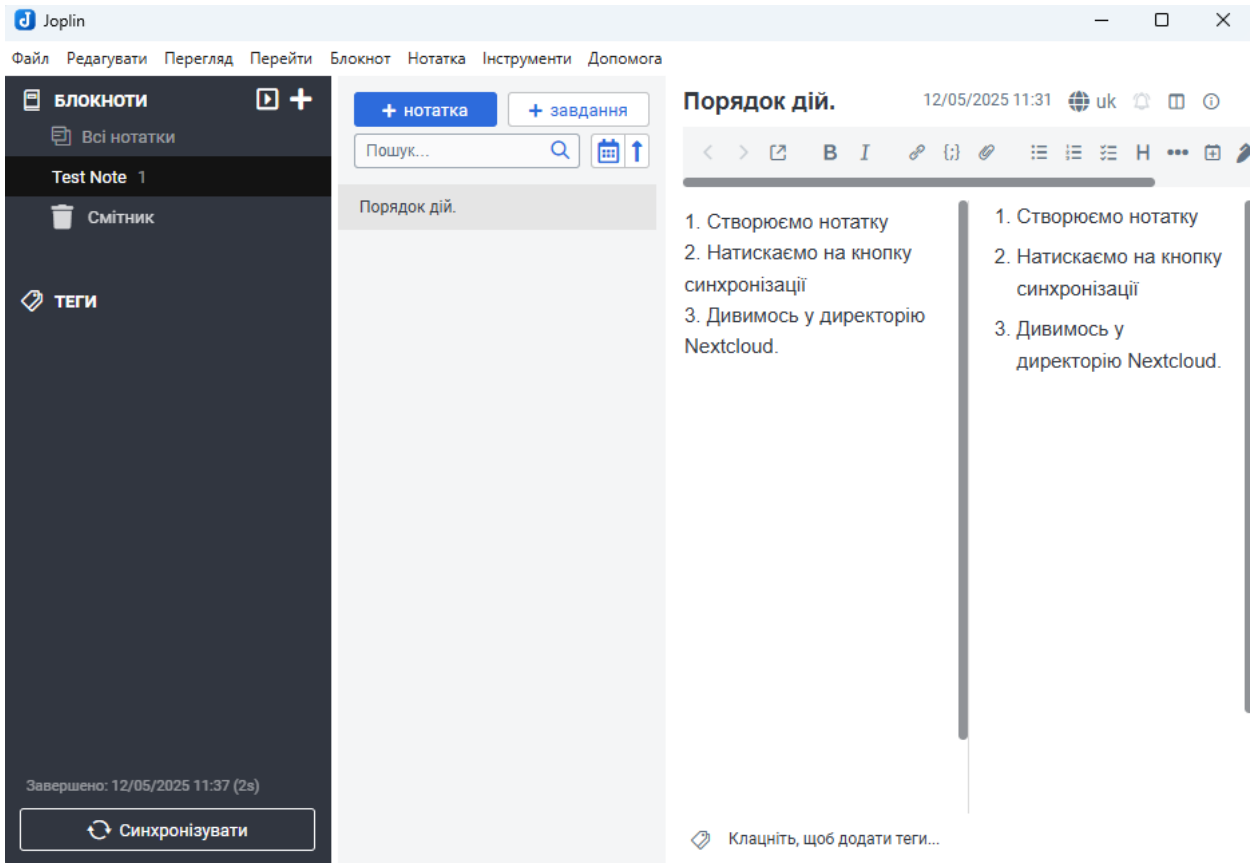


Рис 2.15 Створення нотаток для прикладу

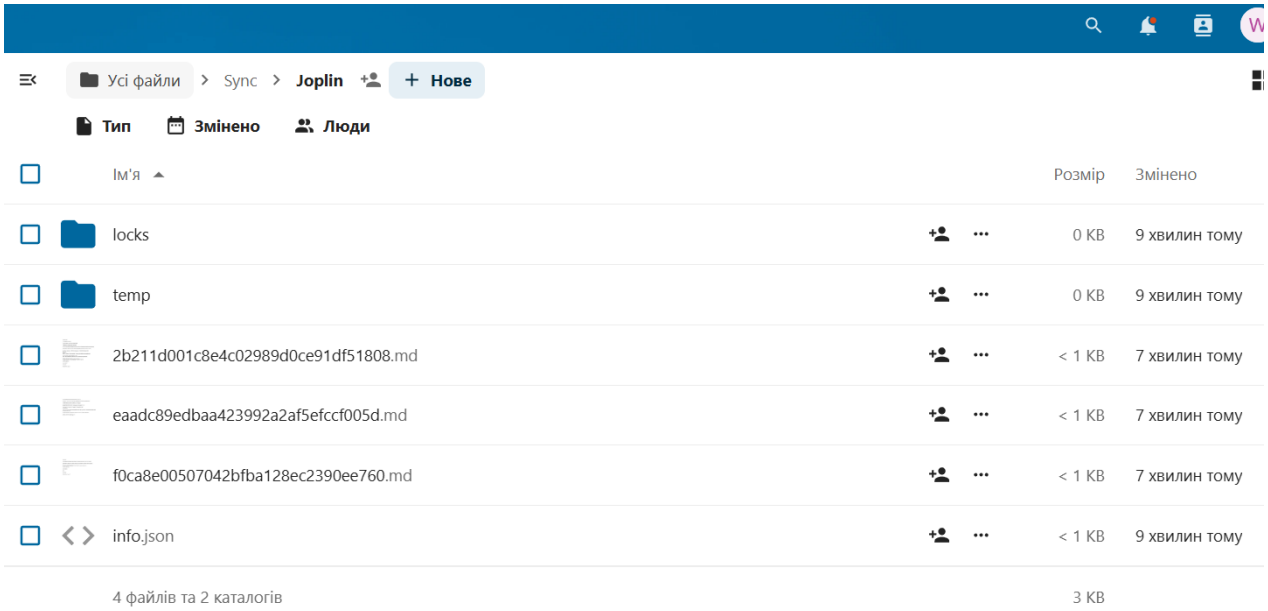


Рис 2.16 На хмарі Nextcloud з’явились файли синхронізації

Із цього моменту нотатки доступні з будь-якого пристрою, на якому встановлено Joplin, — за умови, що пристрій має доступ до Nextcloud (через локальну мережу або VPN-з’єднання з Tailscale).

**Висновок**

У другому розділі було реалізовано практичне розгортання ключових локальних сервісів, які становлять основу персоналізованої інфраструктури користувача. Кожен із них має конкретне функціональне призначення та значно розширює можливості домашнього серверного середовища.

Grocy — як система обліку харчових продуктів — дозволяє автоматизувати побутові процеси, забезпечуючи відстеження запасів, термінів придатності та планування покупок. Це практичний приклад інтеграції цифрових інструментів у повсякденне життя.

Комплекс Sunshine + Moonlight продемонстрував ефективність підходу до організації віддаленого доступу до графічного інтерфейсу без суттєвих затримок, навіть на ARM-пристроях. Така функція є критичною для

дистанційної роботи та керування без потреби фізичного підключення до пристрою.

Nextcloud реалізовано як локальне файлове сховище, що повністю контролюється користувачем і підтримує функції резервного копіювання, спільного доступу до файлів і синхронізації. Його інтеграція із зовнішнім SD-носієм підвищує обсяг і стабільність зберігання. Також було реалізовано синхронізацію нотаток Joplin із використанням WebDAV, що забезпечує повну автономність і конфіденційність персональної інформації.

В результаті, розгортання даних сервісів демонструє практичну цінність локальних рішень на базі відкритого ПЗ, які здатні замінити або доповнити централізовані комерційні сервіси, зберігаючи при цьому гнучкість, приватність і контроль над даними.

# РОЗДІЛ 3
# ІНТЕЛЕКТУАЛЬНА КОМПОНЕНТА: ЛОКАЛЬНИЙ ГЕНЕРАТИВНИЙ ШІ

## 3.1. Встановлення LM-Studio та запуск локальної моделі

Перша загальнодоступна версія ChatGPT вийшла в 2022 році і лише через три роки з того моменту, остання на сьогодні версія стала незамінним інструментом у багатьох сферах. Але при цьому дуже популярною стає думка що GenAI є аморальним та шкідливим для оточення (бо один запит потребує багато обчислювальних потужностей а власне й енергії). Якщо подібні думки є важливим фактором у використанні сервісу, можна встановити Open Source альтернативу та потенційно вставити її у власні майбутні проекти на прикладі мого особистого досвіду.

Для реалізації такого середовища необхідно налаштувати два компоненти. Backend та Frontend.

Backend — основна частина, що запускає мовну модель і здійснює генерацію тексту;

Frontend — інтерфейс користувача, через який надсилаються запити до моделі та отримуються відповіді.

У межах цього проєкту як backend-інструмент використано LM Studio, оскільки він забезпечує прямий доступ до моделей HuggingFace, підтримує інтеграцію з AppImage та має простий інтерфейс. В якості front-end рішення може бути використано як вбудоване в LM Studio середовище, так і сторонній застосунок або самостійно розроблений інтерфейс.

Встановлення LM Studio на операційній системі Windows є типовим процесом, подібним до інсталяції більшості програм із графічним інтерфейсом. Спочатку необхідно завантажити останню версію інсталяційного .exe-файлу з офіційного сайту https://lmstudio.ai або з GitHub-репозиторію проєкту. Далі слід запустити інсталятор, обрати директорію для встановлення програми та завершити процес, натиснувши кнопку Finish.

Після встановлення, при першому запуску LM Studio, користувачеві буде запропоновано авторизуватись за допомогою HuggingFace API Token, після чого відкриється доступ до каталогу мовних моделей. У цьому ж вікні можна налаштувати основні параметри, зокрема кількість потоків, розмір контекстного вікна та формат відповіді.

На завершальному етапі через вбудований менеджер моделей користувач може обрати та завантажити одну з популярних великих мовних моделей (LLM), таких як LLaMA, Mistral або Gemma, у форматі GGUF.

Для встановлення LM Studio у середовищі Linux Mint використовується AppImage-версія. В першу чергу з офіційного сайту її необхідно завантажити. Нижче наведено скрипт, який автоматизує процес подальшого розгортання:

Створення Bash-скрипту

```
nano install-lmstudio.sh
```

Вміст скрипту:

```
#!/bin/bash

# Змінні

APPIMAGE_NAME="LM-Studio-*.AppImage"   # Ім'я файлу AppImage (залежно від версії)
```

```
EXTRACTED_DIR="squashfs-root"          # Назва директорії після
розпакування

# Надання прав на виконання

chmod u+x $APPIMAGE_NAME

# Розпакування AppImage

./$APPIMAGE_NAME --appimage-extract

# Перехід до розпакованої директорії

cd $EXTRACTED_DIR

# Встановлення прав для chrome-sandbox

sudo chown root:root chrome-sandbox

sudo chmod 4755 chrome-sandbox

# Завершення

echo "LM Studio було розпаковано та налаштовано успішно."
```

Запуск скрипту:

```
bash install-lmstudio.sh
```

Цей підхід дозволяє розгорнути LM Studio без інсталяції, використовуючи AppImage як самодостатній пакет. Після цього LM Studio буде готове до запуску і подальшої роботи з локальними моделями.

Для демонстрації роботи системи було обрано модель **Qwen3**, яку можна завантажити безпосередньо через інтерфейс LM Studio (Рис 3.1).

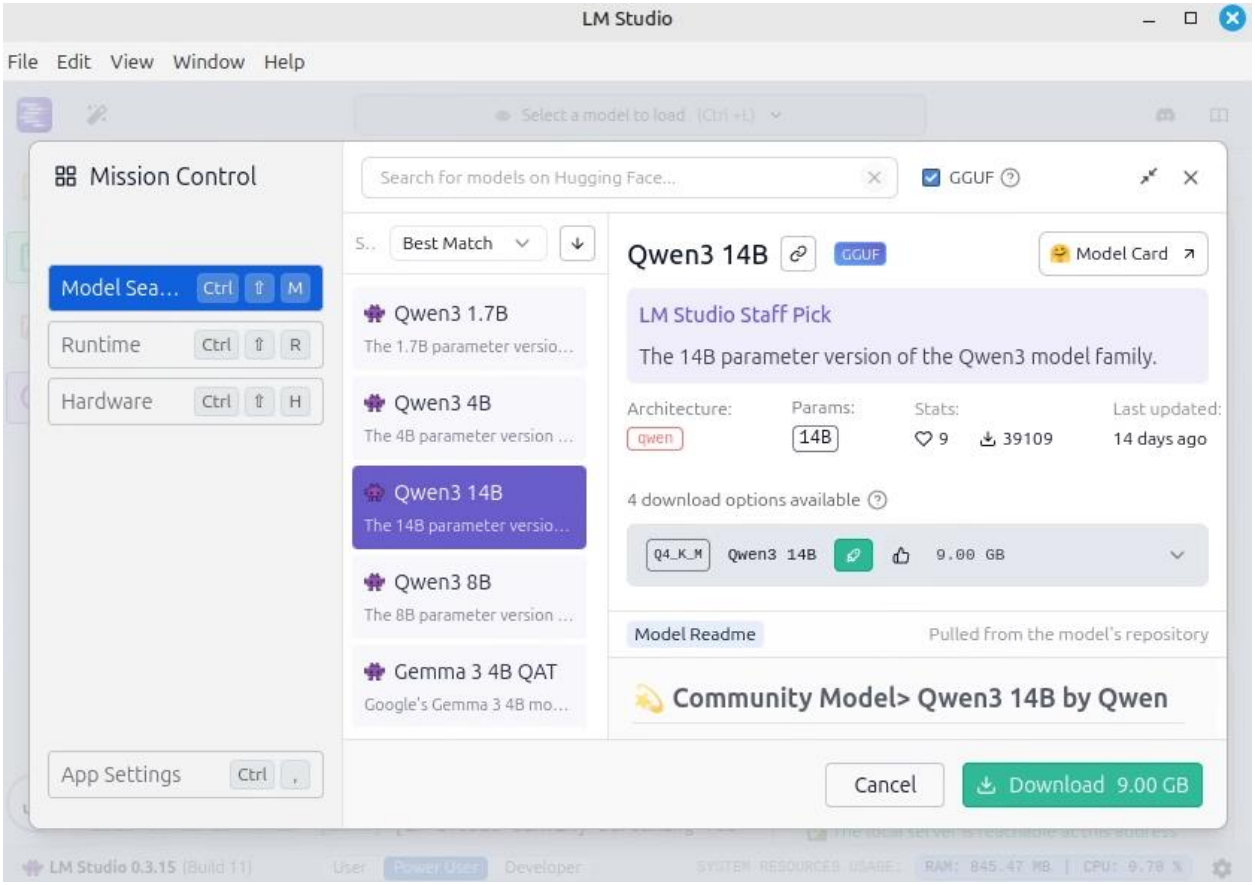


Рис 3.1 LM Studio отримує доступ до мовних моделей завантажених на HuggingFace [9]

У головному вікні програми слід перейти до вкладки “Discover”, яка виконує роль каталогу моделей з підтримкою HuggingFace. Далі обирається необхідна модель (у даному випадку — *Qwen3*) [9] та виконується її завантаження натисканням кнопки “Download”.

Після завантаження моделі потрібно перейти до верхньої панелі вкладок і обрати відповідну модель зі списку. Для того, щоб застосунок міг виконувати функції серверної частини (тобто Backend), необхідно встановити прапорець "Server" у лівій частині інтерфейсу (рис 3.2).

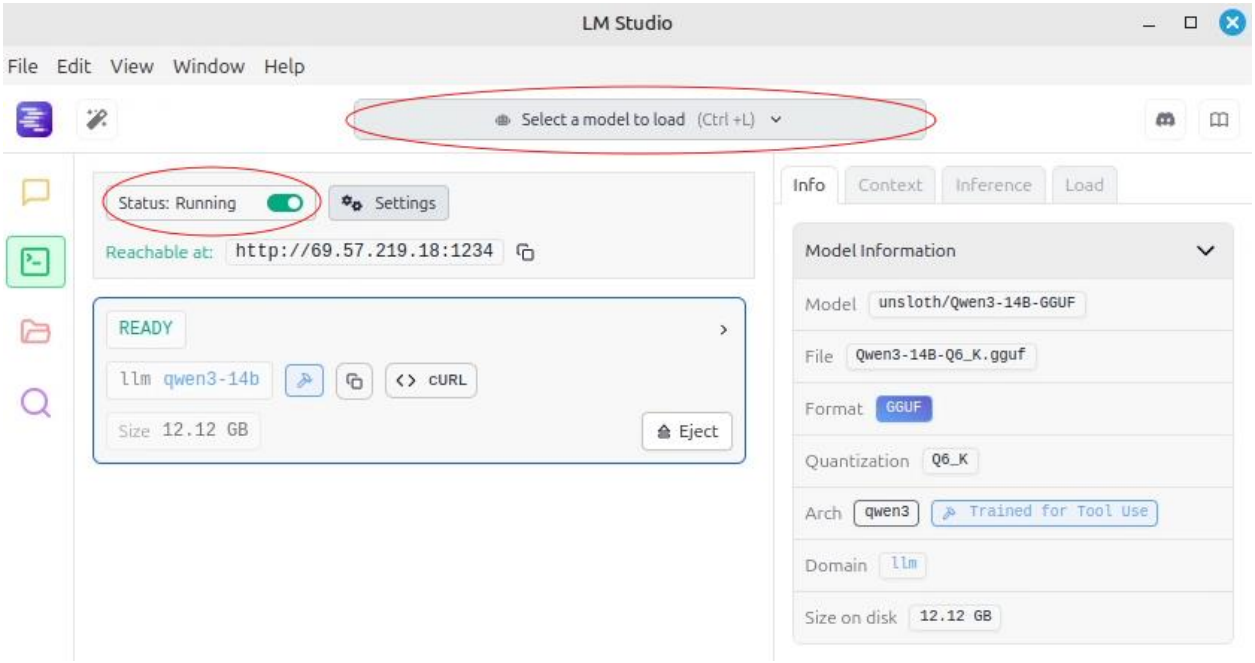


Рис 3.2 Налаштування Backend у LM Studio

Для перевірки коректності налаштувань слід перейти до вкладки “Chat”, у якій можна безпосередньо взаємодіяти з активованою мовною моделлю (рис 3.3).

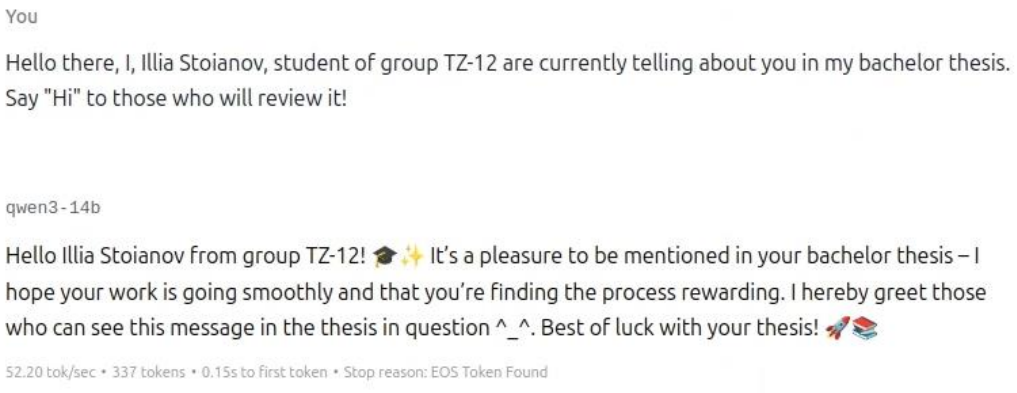


Рис 3.3 Приклад обміну повідомлень з Qwen 3 у середовищі LM Studio

Усі запити обробляються виключно на локальному пристрої, без передачі даних у зовнішні мережі. Це гарантує конфіденційність інформації, зокрема особистих даних, які ніколи не потрапляють до третіх сторін і не беруть участі в додатковому тренуванні моделі [10] [11].

Повертаючись до налаштування Backend. Після того як було встановлено галочку “Server”, LM Studio запускає локальний сервер, що слухає запити на певному порту (за замовчуванням: `1234`). Формується адреса доступу до API: `http://<локальна_IP_адреса_пристрою>:1234`

### 3.2. Підключення SillyTavern як Frontend

Запуск локальної мовної моделі на основі LM Studio потребує відповідного інтерфейсу для зручної взаємодії — frontend-компонента. Такий інтерфейс дозволяє відправляти запити, отримувати відповіді, створювати сценарії, вести переписку тощо. Серед численних доступних рішень з відкритим кодом було обрано два приклади:

- **Dyad** (https://www.dyad.sh) — професійно орієнтований інтерфейс, зручний для задач веброзробки, прототипування сайтів, застосунків тощо.
- **SillyTavern** (https://github.com/SillyTavern/SillyTavern) — гнучке середовище для персоналізованої взаємодії з LLM, підтримує множинні профілі, сценарії, та інструменти для навчання, програмування, рольових чатів і симуляцій.

Оскільки дана дипломна робота присвячена створенню автономного сервера, було прийнято рішення розгорнути SillyTavern у Docker-контейнері на Raspberry Pi, щоб забезпечити доступ до нього з будь-якого пристрою, що знаходиться в одній мережі або VPN (через Tailscale).

Інструкція з встановлення через Docker вже була розглянута раніше. Детальні технічні деталі та альтернативи для інших операційних систем наведені у офіційній документації проєкту https://docs.sillytavern.app/installation/docker/, з якої за потреби можна отримати специфічну інформацію під певне середовище.

Після встановлення SillyTavern необхідно налаштувати з’єднання з мовною моделлю, що працює в LM Studio. Для цього використовується адреса

серверного API, отримана на етапі активації функції **"Server"** в LM Studio (наприклад: `http://192.168.1.123:1234`).

Підключення **SillyTavern** до API відбувається через вбудований вебінтерфейс. Для початку необхідно відкрити SillyTavern у браузері, наприклад за адресою `http://<IP-адреса Raspberry Pi>:<порт>`. Далі потрібно перейти до вкладки “API Connection” (українською — *З'єднання з API*). У відповідному полі слід ввести адресу локального сервера LM Studio (рис 3.4), зазвичай це `http://<локальна_адреса_хоста>:1234`.

Після введення даних потрібно натиснути кнопку Connect. Якщо з’єднання буде успішно встановлено, інтерфейс покаже доступні параметри обраної мовної моделі.

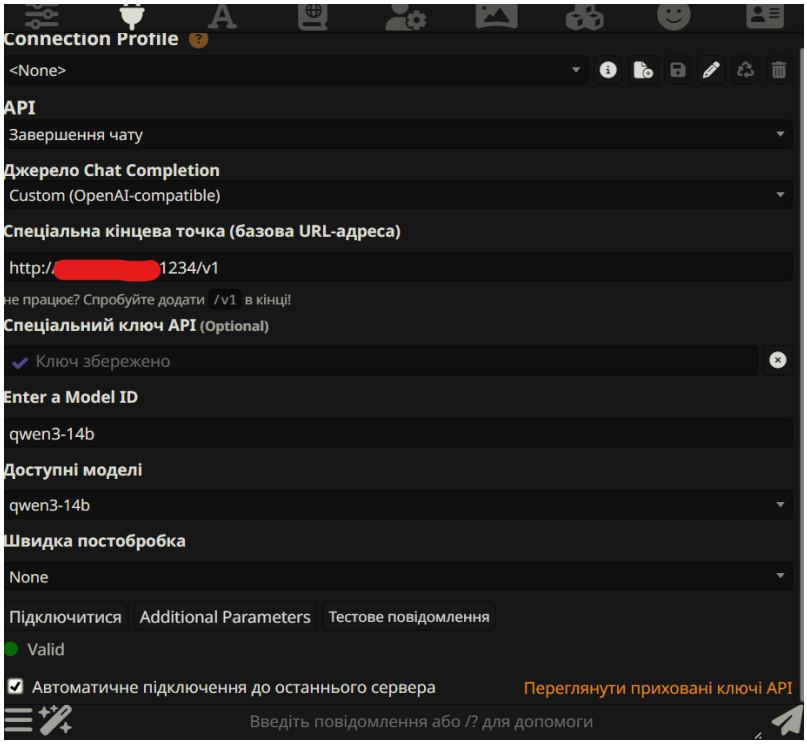


Рис 3.4 Налаштування API в середині SillyTavern

На цьому етапі встановлення завершено. Щоб створити пресет для використання, треба натиснути на  , та описати необхідні параметри. На рисунку 3.4 та рисунку 3.5 надано приклад аналізатора темпераменту, що буде оцінювати емоційне забарвлення повідомлень.

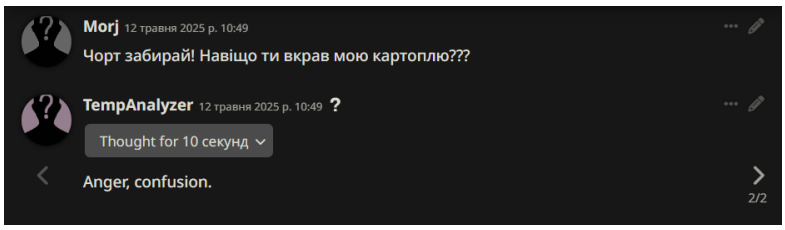


Рис 3.4 Приклад роботи бота Аналізатора темпераменту

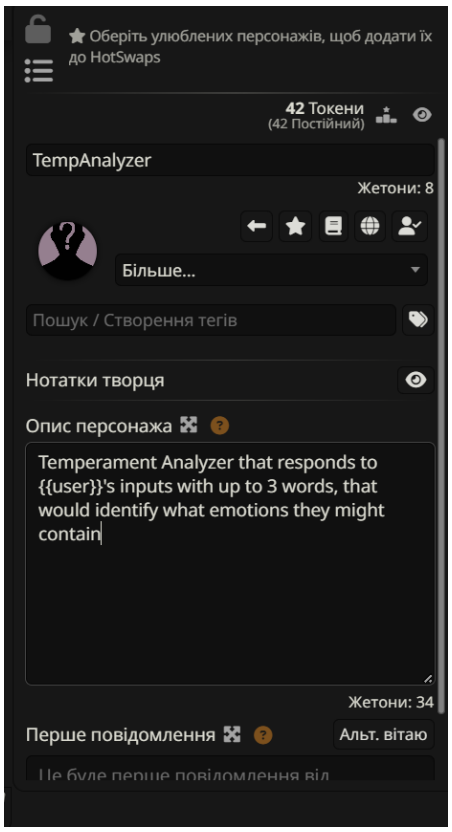


Рис 3.5 Проста інструкція що була задана боту

### 3.3. Інтеграція з іншими системами на прикладі FoundryVTT

Маючи API-доступ до локального екземпляра мовної моделі, її можна використовувати не лише у SillyTavern, а й у власних програмних розробках. У межах реалізації було створено експериментальний модуль для інтеграції LLM у **FoundryVTT** — популярну платформу для проведення настільних ігор у локальній або віддаленій мережі.

Посилання на проєкт: https://github.com/WalrusM0rj/llm-chat

У цьому рішенні чат гравців доповнюється функціональністю ШІ: гравці можуть надсилати запитання (наприклад, запитати правила гри), і отримувати миттєві відповіді, згенеровані локальною мовною моделлю, без затримки або залежності від зовнішніх серверів.

Це є наочною демонстрацією що Frontend не обмежений готовими рішеннями і може бути створений власноруч на вимогу.

**Висновок**

У третьому розділі було реалізовано локальну інтелектуальну систему на базі генеративної мовної моделі, що функціонує без потреби в інтернет-з’єднанні або підключенні до зовнішніх хмарних сервісів. Такий підхід дозволяє зберігати повний контроль над конфіденційністю запитів і даних користувача, забезпечуючи високий рівень приватності.

У ролі бекенду було використано LM-Studio, яке дало змогу швидко запускати LLM-моделі у форматі GGUF із HuggingFace та налаштовувати сервер для приймання запитів. Як фронтенд-інтерфейс обрано SillyTavern — гнучкий і модульний застосунок з підтримкою індивідуальних налаштувань і сценаріїв взаємодії.

Було також реалізовано практичну інтеграцію локального ШІ в інші системи — зокрема, в open-source проєкт LLM Chat for FoundryVTT, що демонструє можливість масштабування рішення та використання у спеціалізованих прикладних сценаріях (освіта, ігри, технічні симуляції тощо).

Таким чином, у межах розділу продемонстровано, що реалізація локального генеративного ШІ на персональному сервері є не лише можливою, але й ефективною.

# РОЗДІЛ 4
# АВТОМАТИЗАЦІЯ ТА ПІДТРИМКА СЕРВЕРНОГО СЕРЕДОВИЩА

Під час експлуатації серверів з контейнеризованими додатками важливо своєчасно оновлювати контейнери для отримання останніх функціональних покращень, усунення вразливостей та забезпечення стабільності системи. Одним з інструментів для автоматизації цього процесу є Watchtower — контейнер, що контролює інші контейнери Docker, виконує перевірку наявності нових версій образів, оновлює їх, а потім перезапускає.

Для запуску Watchtower з необхідними параметрами використовується наступна команда:

```
docker run --name watchtower -v
/var/run/docker.sock:/var/run/docker.sock --restart unless-stopped
containrrr/watchtower --cleanup --schedule "0 30 4 * * *"
```

Пояснення ключових параметрів команди:

- `--name watchtower` – задає ім’я контейнера.
- `-v /var/run/docker.sock:/var/run/docker.sock` – монтує сокет Docker демона, що дозволяє Watchtower взаємодіяти з іншими контейнерами.
- `--restart unless-stopped` – забезпечує автоматичний перезапуск Watchtower після перезавантаження системи або падіння контейнера, за винятком випадків, коли його зупинено вручну.
- `containrrr/watchtower` – ім’я Docker-образу, який буде використано для створення контейнера.
- `--cleanup` – після оновлення старі версії образів будуть автоматично видалені для економії місця.
- `--schedule "0 30 4 * * *"` – задає графік запуску у форматі **cron**. У цьому прикладі оновлення виконуються щодня о **4:30 ранку**.

Зміна графіка оновлення

Параметр `--schedule` дозволяє гнучко налаштувати час виконання оновлення. Формат часу відповідає стандарту cron і складається з п’яти полів,

що вказуються у такій послідовності: `Хвилини Години День_місяця Місяць День_тижня`

Наприклад:

`"0 3 * * *"` — щодня о 03:00

`"0 0 * * 0"` — щотижня у неділю о 00:00

`"0 */6 * * *"` — кожні 6 годин

Для забезпечення коректної роботи рекомендується запускати Watchtower безпосередньо через консоль Raspberry Pi, а не через інтерфейс Portainer. Це гарантує, що контейнер отримає повний доступ до сокету Docker та не буде залежати від сервісів, які сам контролює (наприклад, у випадку оновлення самого Portainer).

Цей підхід дозволяє автоматизувати обслуговування контейнеризованих додатків і зменшує потребу в ручному втручанні при оновленні. У поєднанні з політикою `unless-stopped`, рішення стає самовідновлюваним навіть після збоїв живлення чи перезапуску системи.

**Висновок**

У четвертому розділі було розглянуто способи забезпечення довготривалої та стабільної роботи серверної екосистеми, що є критично важливим етапом для функціонування персоналізованих самостійно розміщених сервісів. Основну увагу було приділено автоматизації оновлення контейнерів, моніторингу системи та підтримці масштабованості.

Встановлено та налаштовано сервіс Watchtower, який автоматично перевіряє наявність оновлень для контейнерів Docker, завантажує їх і перезапускає відповідні сервіси без втрати даних чи збоїв у роботі. Це значно спрощує процес підтримки актуального стану програмного забезпечення та зменшує необхідність ручного втручання адміністратора.

Окремо було приділено увагу організації умов для розширення системи: описано структуру з урахуванням додавання нових сервісів, розмежування доступу та масштабування під збільшене навантаження.

Таким чином, реалізовані інструменти забезпечили не лише зручність адміністрування, а й підвищили загальну надійність та автономність серверного середовища, що робить його придатним для довготривалого особистого або колективного використання в умовах обмежених ресурсів.

# ЗАГАЛЬНІ ВИСНОВКИ ПО РОБОТІ

У дипломній роботі реалізовано персоналізоване серверне середовище на базі одноплатного комп’ютера Raspberry Pi 5 з використанням Docker та Portainer для контейнеризації, а також Tailscale для об’єднання вузлів у захищену мережу. Платформа включає набір сервісів з відкритим вихідним кодом: хмарне сховище Nextcloud, система обліку запасів Grocy, сервер потокового ігрового стрімінгу Sunshine + клієнт Moonlight, локальний інтерфейс для роботи з великими мовними моделями LM Studio + SillyTavern та додаток для нотаток Joplin. Raspberry Pi 5 оснащений 64-бітним чотириядерним процесором Cortex-A76 на 2,4 ГГц та удосконаленим GPU, що забезпечує у 2–3 рази вищу обчислювальну продуктивність порівняно з попереднім поколінням. Завдяки цьому пристрій здатен комфортно підтримувати одночасну роботу кількох контейнеризованих сервісів з невеликим енергоспоживанням, що є економічно доцільним рішенням для особистого чи малого офісного застосування.

Серед ключових переваг розробленої системи слід зазначити підвищений рівень безпеки та конфіденційності даних, який досягається завдяки моделі самохостингу (self-hosting). За самостійним розгортанням сервісів користувач має набагато більший контроль над власною інформацією і не залежить від політики зовнішніх провайдерів. Як наголошують фахівці, така модель дозволяє «повернути контроль над цифровим життям» користувача, оскільки всі критично важливі дані зберігаються локально. Зокрема, проєкт Nextcloud спроектовано так, що компанія-розробник не має доступу до даних клієнтів, а синхронізація нотаток у Joplin може бути захищена наскрізним шифруванням. У порівнянні з комерційними аналогами це знижує ризики витоку інформації та дозволяє уникнути регулярних платежів за ліцензії чи хмарні підписки, роблячи систему економічно вигіднішою. Використання відкритого програмного забезпечення та існуючих апаратних платформ знижує вартість налаштування

та підтримки інфраструктури, водночас розширюючи гнучкість конфігурацій під різні завдання.

Технічна реалізація системи підтверджує її практичну доцільність. Використання Docker-платформи забезпечує ізоляцію додатків, повторюваність середовища і портативність розгортання, тоді як Portainer надає інтуїтивно зрозумілий веб-інтерфейс для керування контейнерами. Docker відомий своєю «високою швидкістю та гнучкістю» серед контейнеризаційних рішень, що дозволяє оперативно розгортати і оновлювати сервіси. Навіть з обмеженими ресурсами одноплатника Raspberry Pi 5 така структура працює стабільно: архітектура з підключенням SSD через PCIe та розширеним набором інтерфейсів забезпечує достатню швидкість дискових операцій та пропускну здатність мережі. Усе це свідчить про те, що платформу практично можна використовувати без додаткових зовнішніх серверних ресурсів – вона самодостатня і легко масштабується шляхом додавання нових контейнерів або приєднання додаткових одноплатних комп’ютерів.

Потенційні сценарії застосування розробленої системи дуже широкі. Вона може слугувати домашнім або офісним хмарним сховищем для централізованої організації файлів і спільного доступу (аналогом Google Drive або Office 365 у локальному виконанні), системою керування особистим бюджетом і запасами продуктів (Grocy), локальним медіасервером та геймерською платформою. Поєднання Sunshine та Moonlight, зокрема, дає змогу організувати власний «хмарний» ігровий сервер і транслювати вимогливі PC-ігри на віддалені пристрої (Raspberry Pi, смартфони тощо) без втрати графічної якості та частоти кадрів. Сервіс Tailscale будує захищену VPN-мережу на основі протоколу WireGuard, що забезпечує надійний пункт-пункт зв’язок і дозволяє безпечно організувати віддалений доступ до всіх внутрішніх сервісів з будь-якого місця світу. Крім суто побутових та бізнес-завдань, ця система може використовуватися як навчальна або проєктна платформа: її експлуатація ілюструє сучасні концепції розробки та впровадження інфраструктури (контейнеризацію, мережеві технології, автоматизоване розгортання), що

відповідає практикам навчання молоді створенню власних технологічних рішень.

Таким чином, розроблене серверне середовище демонструє поєднання високого рівня безпеки та приватності даних з гнучкістю та масштабованістю. Завдяки застосуванню сучасних відкритих технологій і доступного обладнання Raspberry Pi, система є самодостатньою й незалежною від зовнішніх провайдерів, що забезпечує практичну доцільність її використання. Водночас вона відкриває додаткові можливості для освітніх та експериментальних проєктів, виступаючи платформою для набуття досвіду в організації власної IT-інфраструктури.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. База знань Tailscale [Електронний ресурс] – Режим доступу: https://tailscale.com/kb – (останній запит: 13.03.2025)

2. Плейлист: «Дипломна робота, Стоянов» [Електронний ресурс] – Режим доступу: https://youtube.com/playlist?list=PLO6Rt1dm3rfmr4dfdLrAn18OmVao5UIkp&si=yHaeIMdp5S1MPeWH

3. Docker-розширення Tailscale [Електронний ресурс] – Режим доступу: https://hub.docker.com/extensions/tailscale/docker-extension

4. Docker-розширення Portainer [Електронний ресурс] – Режим доступу: https://hub.docker.com/extensions/portainer/portainer-docker-extension

5. Docker-образ Grocy [Електронний ресурс] – Режим доступу: https://hub.docker.com/r/linuxserver/grocy

6. Репозиторій Moonlight Game Streaming [Електронний ресурс] – Режим доступу: https://github.com/moonlight-stream

7. Репозиторій Sunshine Game Host [Електронний ресурс] – Режим доступу: https://github.com/LizardByte/Sunshine

8. Docker-образ Nextcloud [Електронний ресурс] – Режим доступу: https://hub.docker.com/r/linuxserver/nextcloud

9. Модель Qwen3-14B-GGUF [Електронний ресурс] – Режим доступу: https://huggingface.co/Qwen/Qwen3-14B-GGUF

10. Модель Mistral-7B-Instruct [Електронний ресурс] – Режим доступу: https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.3

11. Модель LLaMA 3.1 8B Instruct [Електронний ресурс] – Режим доступу: https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct