

Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття ступеня вищої освіти бакалавра
зі спеціальності 121 – Інженерія програмного забезпечення

На тему: «Розробка додатку для автоматизованого розрахунку параметрів
IPv4/IPv6 мереж»

Засвідчую, що в цій
кваліфікаційній роботі немає
запозичень із праць інших
авторів без відповідних посилань.

Студент гр. ІПЗ-22-1

_____ / Перепелиця М. В. /

Керівник кваліфікаційної роботи

_____ / Швець Д. В. /

В.о. завідувача кафедри

_____ / Стрюк А. М. /

Кривий Ріг

2026

Криворізький національний університет

Факультет: Інформаційних технологій

Кафедра: Моделювання та програмного забезпечення

Ступінь вищої освіти: бакалавр

Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

В.о. завідувача кафедри _____ Стрюк А. М.

«___» _____ 2026 р.

ЗАВДАННЯ

на кваліфікаційну роботу

студенту групи ІПЗ-22-1 Перепелиці М. В.

1. Тема: Розробка додатку для автоматизованого розрахунку параметрів IPv4/IPv6 мереж

затверджено наказом по КНУ № 284с від «28» травня 2026 р.

2. Термін подання студентом закінченої роботи: «31» травня 2026 р.

3. Вихідні дані по роботі: стандарти та рекомендації щодо IPv4/IPv6-адресації, CIDR-префіксів, тестові набори IPv4/IPv6 адрес і мереж, вимоги до функціональності веб-додатку для автоматизованого розрахунку параметрів мереж.

4. Зміст пояснювальної записки: виконати аналіз принципів IPv4/IPv6-адресації, провести огляд існуючих програмних рішень, сформулювати вимоги до додатку, розробити алгоритми розрахунку параметрів IPv4/IPv6 мереж, спроектувати архітектуру програмного забезпечення, реалізувати веб-додаток, провести тестування розробленого програмного забезпечення.

5. Перелік ілюстративного матеріалу: структурна схема додатку, функціональна схема модулів, блок-схеми алгоритмів, структура бази даних, знімки екранних форм додатку, таблиця результатів тестування.

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	Огляд літератури за тематикою та збір джерел	01.03.2026 - 05.03.2026
2	Аналіз принципів IPv4/IPv6-адресації та CIDR	06.03.2026 - 10.03.2026
3	Аналіз існуючих програмних рішень	11.03.2026 - 14.03.2026
4	Формування вимог до додатку	15.03.2026 - 18.03.2026
5	Проектування алгоритмів розрахунку параметрів мереж	19.03.2026 - 25.03.2026
6	Проектування архітектури та структури бази даних	26.03.2026 - 31.03.2026
7	Реалізація backend-частини додатку	01.04.2026 - 25.04.2026
8	Реалізація frontend-частини додатку	26.04.2026 - 15.05.2026
9	Тестування програмного забезпечення	16.05.2026 - 24.05.2026
10	Оформлення пояснювальної записки та презентації	25.05.2026 - 31.05.2026

Дата видачі завдання: «10» січня 2026 р.

Студент _____ / Перепелиця М. В. /

Керівник роботи _____ / Швець Д. В. /

РЕФЕРАТ

IPV4, IPV6, CIDR, ПІДМЕРЕЖА, VLSM, ВЕБ-ДОДАТОК, GO, REACT, SQLITE, DOCKER COMPOSE, ТЕСТУВАННЯ.

Пояснювальна записка: 66 с., 21 табл., 13 рис., 5 дод., 19 джерел.

У пояснювальній записці подано розробку веб-додатку для розрахунку та перевірки параметрів IPv4/IPv6 мереж. Додаток не є IPAM-системою: він зосереджується на практичних сценаріях, які потрібні під час навчання або базового планування мережі, а саме на розрахунку параметрів, поділі на підмережі, перевірці перетину діапазонів, класифікації адрес, спрощеному VLSM, історії та експорту результатів.

У теоретичній частині розглянуто лише ті аспекти адресації, які потрібні для реалізації додатку: CIDR-префікси, межі адресних діапазонів, граничні випадки /31, /32, /127, /128 та спеціальні діапазони адрес. На цій основі визначено вимоги до власного інструменту і описано алгоритми, які використовуються в розрахунковому ядрі.

Програмна частина реалізована як клієнт-серверний веб-додаток. Backend на Go/Gin містить окремі usecase-модулі, розрахунковий пакет internal/calc, SQLite-адаптер історії, Swagger-документацію та DI-композицію через Google Wire. Frontend на React/TypeScript/Vite надає окремі сторінки для основних інструментів. Історія пов'язується з анонімним session_id у браузері, тому авторизація користувачів у роботі не використовується.

Практична цінність роботи полягає в тому, що користувач може швидко перевірити мережевий розрахунок, знайти перетин підмереж, переглянути пояснення результату та зберегти або експортувати виконані операції. Додаток може застосовуватися на лабораторних роботах з комп'ютерних мереж і для базового планування адресного простору невеликих локальних мереж.

ABSTRACT

IPV4, IPV6, CIDR, SUBNET, VLSM, WEB APPLICATION, GO, REACT, SQLITE, DOCKER COMPOSE, TESTING.

Thesis: 66 pages, 21 tables, 13 figures, 5 appendices, 19 references.

This thesis presents a web application for calculating and checking IPv4/IPv6 network parameters. The application is not a full IPAM platform. It focuses on practical tasks used in learning and basic network planning: network parameter calculation, subnetting, overlap checking, address classification, simplified IPv4 VLSM planning, calculation history and export.

The theoretical part focuses on the addressing concepts used directly in the implemented application: CIDR prefixes, address range boundaries, boundary cases such as /31, /32, /127 and /128, and special-purpose address ranges. Based on this scope, the application requirements and the algorithms of the calculation core are described.

The software is implemented as a client-server web application. The Go/Gin backend contains separate usecase modules, the internal/calc package, an SQLite history adapter, Swagger API documentation and dependency composition with Google Wire. The React/TypeScript/Vite frontend provides separate pages for the main tools. Calculation history is linked to an anonymous browser session_id, so user authentication is intentionally excluded.

The application can be used in computer network courses, laboratory work and basic planning of small local networks. It helps users check network ranges, detect overlaps, view calculation explanations and export results to CSV or JSON.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ.....	7
ВСТУП.....	8
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ.....	9
1.1 Основи IPv4-адресації.....	9
1.2 Основи IPv6-адресації.....	10
1.3 CIDR та принципи поділу мереж.....	11
1.4 Аналіз існуючих програмних рішень.....	13
1.5 Постановка задачі.....	14
1.6 Вимоги до додатку.....	15
2 ПРОЄКТУВАННЯ ДОДАТКУ ДЛЯ РОЗРАХУНКУ ПАРАМЕТРІВ IPv4/IPv6 МЕРЕЖ.....	16
2.1 Функціональні вимоги.....	16
2.2 Нефункціональні вимоги.....	17
2.3 Алгоритм розрахунку параметрів IPv4 мережі.....	18
2.4 Алгоритм розрахунку параметрів IPv6 мережі.....	20
2.5 Алгоритм поділу мережі на підмережі.....	20
2.6 Алгоритм перевірки перетину мереж.....	21
2.7 Алгоритм класифікації IP-адрес.....	22
2.8 Спрощений алгоритм VLSM-планування.....	22
2.9 Архітектура програмного забезпечення.....	23
2.10 Структура бази даних.....	26
2.11 Проєктування інтерфейсу користувача.....	27
3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ДОДАТКУ ДЛЯ РОЗРАХУНКУ ПАРАМЕТРІВ IPv4/IPv6 МЕРЕЖ.....	29
3.1 Обґрунтування вибору технологій.....	29
3.2 Реалізація backend-частини.....	30
3.3 Реалізація frontend-частини.....	34
3.4 Реалізація модуля історії розрахунків.....	37
3.5 Реалізація експорту результатів.....	38
3.6 Опис інтерфейсу користувача.....	39
3.7 Тестування розробленого додатку.....	45
3.8 Аналіз отриманих результатів.....	47
ВИСНОВКИ.....	50
ПЕРЕЛІК ПОСИЛАНЬ.....	51
ДОДАТКИ.....	53

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

Скорочення	Розшифрування
API	інтерфейс, через який frontend надсилає запити до backend.
CIDR	формат запису мережі з довжиною префікса, наприклад /24 або /64.
CORS	механізм, який дозволяє frontend-частині звертатися до backend API.
CSV	текстовий формат, у який додаток експортує результати для таблиць.
HTTP	протокол обміну запитами між браузером і backend-сервером.
IP	мережева адресація, з якою працює розроблений додаток.
IPv4	версія IP з 32-бітними адресами.
IPv6	версія IP з 128-бітними адресами.
JSON	формат, у якому backend повертає структуровані результати розрахунків.
REST	підхід до організації HTTP endpoint'ів додатку.
UI	користувацький інтерфейс веб-додатку.
VLSM	підбір підмережі за потрібною кількістю хостів.
SPA	веб-додаток, у якому перехід між інструментами відбувається без повного перезавантаження сторінки.

ВСТУП

Під час роботи з комп'ютерними мережами часто потрібно швидко перевірити кілька параметрів: адресу мережі, межі діапазону, кількість доступних хостів і можливі перетини підмереж. Для IPv4 такі дії ще можна виконати вручну, але з IPv6, великими діапазонами та кількома сценаріями планування ручний підхід стає незручним і помилковим.

У роботі розробляється веб-додаток, який об'єднує інструменти для IPv4/IPv6: розрахунок параметрів мережі, підтримку CIDR, поділ на підмережі, перевірку перетину, класифікацію адрес, спрощене VLSM-планування, історію та експорт. Режим пояснення додається для навчальних ситуацій, коли потрібно не лише отримати результат, а й зрозуміти логіку розрахунку.

Мета роботи - розробити веб-додаток для автоматизованого розрахунку, перевірки та базового планування параметрів IPv4/IPv6 мереж.

Об'єктом розробки є процес розрахунку та базового планування параметрів IPv4/IPv6 мереж. Предметом розробки є алгоритми й програмні засоби, за допомогою яких ці розрахунки виконуються автоматично у веб-інтерфейсі.

Для досягнення мети потрібно розглянути принципи IPv4/IPv6-адресації та CIDR, порівняти існуючі інструменти, сформулювати вимоги до додатку, спроектувати алгоритми розрахунків, описати архітектуру backend і frontend частин, реалізувати збереження історії та перевірити роботу основних модулів тестами.

Результати роботи можуть використовуватися у навчальному процесі, під час лабораторних робіт з комп'ютерних мереж, а також як допоміжний інструмент для базового планування адресного простору невеликих локальних мереж.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Основи IPv4-адресації

У розрахунковому модулі IPv4-адреса спочатку переводиться з текстового запису у числове подання. Це потрібно для побітових операцій з маскою підмережі: так визначаються адреса мережі, broadcast, wildcard-маска та діапазон хостів. Такий підхід спирається на 32-бітну структуру IPv4-адреси [1].

Межа між мережевою та вузловою частинами задається маскою підмережі або CIDR-префіксом. У записі 192.168.1.10/24 перші 24 біти використовуються для ідентифікації мережі, а решта 8 бітів - для адресації вузлів у цій мережі.

Після введення IPv4-адреси та префікса система формує набір практичних параметрів: network address, broadcast, subnet mask, wildcard mask, first host, last host, total addresses і usable hosts.

Класифікація виконується окремим кроком: адреса порівнюється з діапазонами private, loopback, link-local, multicast, documentation та reserved. Для цього використано приватні IPv4-блоки RFC 1918 [2] і перелік спеціальних IPv4-діапазонів IANA [3]. Для префікса /31 обидві адреси можуть використовуватися у point-to-point сценарії відповідно до RFC 3021 [4], а /32 обробляється як одна конкретна IPv4-адреса. Ці випадки перевіряються в тестах розрахункового модуля.

Основні параметри IPv4-мережі наведено у таблиці 1.1.

Таблиця 1.1 – Основні параметри IPv4-мережі

Параметр	Зміст	Приклад
IP-адреса	Введена адреса вузла або інтерфейсу.	192.168.1.10
CIDR-префікс	Довжина мережевої частини адреси.	/24

Параметр	Зміст	Приклад
Маска підмережі	Маска, за якою адреса ділиться на мережеву та вузлову частини.	255.255.255.0
Адреса мережі	Початок розрахованого IPv4-діапазону.	192.168.1.0
Broadcast	Остання адреса IPv4-діапазону для широкомовної передачі.	192.168.1.255
Usable hosts	Кількість адрес, які можна видати вузлам.	254 для /24

Ці параметри використовуються в модулі IPv4 calculation і зберігаються в історії розрахунків разом із вхідними даними користувача.

1.2 Основи IPv6-адресації

IPv6 у додатку обробляється за схожим принципом: адреса переводиться у числове подання, після чого визначаються межі мережевого діапазону. Відмінність полягає в розмірі адресного простору, тому для IPv6 використовується 128-бітна логіка, а не 32-бітні операції, характерні для IPv4 [5].

Текстовий запис IPv6 має інший формат, ніж IPv4: адреса складається з шістнадцяткових груп і може містити скорочення нульових послідовностей. Наприклад, повний запис 2001:0db8:0000:0000:0000:0000:0001 у результатах відображається у коротшому вигляді 2001:db8::1 з урахуванням правил текстового подання IPv6 [6].

Оскільки в IPv6 немає broadcast-адреси, модуль повертає інший набір результатів: network prefix, першу адресу, останню адресу та кількість адрес у діапазоні. Для великих мереж, наприклад /64, система працює з межами діапазону і не генерує повний список адрес.

Під час класифікації IPv6-адрес перевіряється належність до спеціальних діапазонів, зокрема ::1/128, fe80::/10, fc00::/7 або

2001:db8::/32 [7]. Multicast-префікс ff00::/8 обробляється як окрема категорія [5]. У тестах також враховано /127 та /128, щоб перевірити роботу алгоритму з малими IPv6-діапазонами і point-to-point сценаріями [8].

Порівняння IPv4 та IPv6 наведено у таблиці 1.2.

Таблиця 1.2 – Порівняння IPv4 та IPv6

Ознака	IPv4	IPv6
Довжина адреси	32 біти	128 бітів
Форма запису	Десяткові октети через крапку	Шістнадцяткові групи через двокрапку
Приклад	192.168.1.10	2001:db8::1
Broadcast	Підтримується	Не використовується
Типові спеціальні діапазони	private, loopback, link-local, multicast	loopback, link-local, unique local, multicast
Основний параметр мережі	Маска або CIDR-префікс	Префікс мережі

Для реалізації це означає, що IPv4 і IPv6 мають спільну ідею роботи з префіксом, але різну довжину адреси, різні граничні випадки та різний набір результатів для користувача.

1.3 CIDR та принципи поділу мереж

CIDR є основним форматом введення мережі в додатку. Запис 192.168.1.0/24 або 2001:db8::/64 поєднує адресу і довжину префікса, тому його зручно використовувати як для IPv4, так і для IPv6. RFC 4632 використано як джерело щодо принципів безкласової адресації [9].

Для алгоритмів додатку префікс є ключовим параметром. Він визначає, скільки старших бітів належить мережі, а скільки залишається під адреси вузлів. Кількість адрес визначається за формулою $2^{(\text{bits} - \text{prefix})}$, де bits

дорівнює 32 для IPv4 і 128 для IPv6. Наприклад, /24 містить 256 IPv4-адрес, а /26 - 64 адреси.

$$total = 2^{(bits - prefix)} \quad (1.1)$$

Subnetting застосовується, коли з одного адресного блоку потрібно отримати кілька менших підмереж. У додатку користувач вводить базову мережу та новий префікс, а система перевіряє коректність запиту і формує список результатів. Наприклад, 192.168.1.0/24 можна поділити на чотири підмережі /26.

VLSM реалізовано як окремий спрощений сценарій. Користувач вводить базову IPv4-мережу та потрібну кількість хостів, після чого додаток підбирає мінімальний префікс, який покриває цю вимогу. Такий обсяг достатній для теми роботи і не перетворює проєкт на IPAM-систему.

Приклади CIDR-префіксів для IPv4 наведено у таблиці 1.3.

Таблиця 1.3 – Приклади CIDR-префіксів для IPv4

Префікс	Кількість адрес	Типове використання
/24	256	Локальна мережа або VLAN середнього розміру
/25	128	Поділ /24 на дві частини
/26	64	Підмережа для групи до 62 вузлів
/30	4	Класичне point-to-point з'єднання
/31	2	Point-to-point з'єднання за RFC 3021
/32	1	Одна конкретна IPv4-адреса

Окремий модуль перевіряє, чи не накладаються мережеві діапазони один на одного. Це допомагає знайти помилки ще на етапі планування, до використання адресного плану в реальній мережі.

1.4 Аналіз існуючих програмних рішень

Для порівняння було взято кілька типів рішень: онлайн-калькулятори, desktop-утиліти, програмні бібліотеки та IPAM-системи. Таке порівняння допомагає визначити межу власного проєкту: зробити більше, ніж простий калькулятор, але не переходити до рівня повної системи управління адресним простором.

SolarWinds Subnet Calculator розглянуто як приклад онлайн-інструмента для швидкого subnetting [10]. Для цієї роботи важливо не повторити готовий сервіс, а реалізувати власні алгоритми, історію, експорт і режим пояснення у веб-додатку.

SoftPerfect IP Calculator розглянуто як приклад desktop-утиліти з ширшим набором мережевих можливостей [11]. Повністю переносити такий рівень функціональності до бакалаврського проєкту недоцільно, тому в роботі залишено тільки потрібні для теми сценарії: розрахунок, subnetting, overlap check, classification і VLSM.

NetBox розглянуто як приклад повноцінної IPAM-системи [12]. На відміну від такого класу рішень, розроблюваний додаток не веде облік обладнання, ролей користувачів або всієї мережевої інфраструктури, а зосереджується на розрахунку та перевірці адресних діапазонів.

Порівняння існуючих програмних рішень наведено у таблиці 1.4.

Таблиця 1.4 – Порівняння існуючих рішень

Рішення	Переваги	Обмеження для даної роботи
Онлайн subnet calculators	Швидко виконує базові розрахунки мережі і не потребує встановлення.	Зазвичай не поєднує історію, пояснення розрахунку, експорт і власну реалізацію алгоритмів.
SolarWinds Subnet Calculator	Приклад онлайн-сервісу для subnetting і перевірки адресного плану.	Розглядається як готовий інструмент, тоді як у роботі потрібна власна реалізація модулів.

Продовження таблиці 1.4

Рішення	Переваги	Обмеження для даної роботи
SoftPerfect IP Calculator	Приклад desktop-утиліти з ширшим набором мережевих можливостей.	Містить більше функцій, ніж потрібно для навчально-практичного веб-додатку.
NetBox / IPAM-системи	Приклад системи для повного обліку адресного простору та мережевої інфраструктури.	Вимагає складнішої моделі даних, користувачів, ролей та адміністрування.
Розроблюваний додаток	Об'єднує розрахунок IPv4/IPv6, CIDR, subnetting, overlap check, classification, history, export та explanation mode.	Не виконує сканування мережі і не претендує на роль повної IPAM-системи.

Після аналізу аналогів обрано проміжний варіант: веб-додаток, який має більше можливостей, ніж простий IP-калькулятор, але не є повноцінною IPAM-системою. До функціоналу включено розрахунок IPv4/IPv6 мереж, CIDR, поділ на підмережі, перевірку перетину, класифікацію адрес, історію, експорт, режим пояснення та спрощене VLSM-планування.

1.5 Постановка задачі

Після аналізу предметної області задачу роботи сформульовано так: розробити веб-додаток, який дозволяє користувачу вводити IP-адресу або мережу у CIDR-форматі, перевіряти коректність даних, отримувати розраховані параметри, ділити мережі на підмережі, перевіряти перетини, класифікувати адреси, зберігати історію та експортувати результати.

Об'єктом розробки є процес розрахунку та базового планування параметрів IPv4/IPv6 мереж. Предметом розробки є алгоритми та програмні засоби, які реалізують ці розрахунки у вигляді клієнт-серверного веб-додатку.

Для цього потрібно виконати такі задачі:

- а) проаналізувати принципи IPv4- та IPv6-адресації;
- б) дослідити CIDR-підхід і принципи поділу мереж на підмережі;
- в) проаналізувати існуючі програмні рішення для розрахунку параметрів мереж;
- г) сформулювати функціональні та нефункціональні вимоги до додатку;
- г) розробити алгоритми розрахунку параметрів IPv4 та IPv6 мереж;
- д) розробити алгоритми subnetting, overlap check, classification та спрощеного VLSM-планування;
- е) спроектувати архітектуру backend- та frontend-частин;
- є) визначити структуру бази даних для збереження історії розрахунків;
- ж) спроектувати користувацький інтерфейс додатку.

1.6 Вимоги до додатку

Детальні функціональні та нефункціональні вимоги наведено у розділах 2.1 і 2.2. У межах постановки задачі важливо зафіксувати головну межу проєкту: додаток має бути практичним веб-інструментом для розрахунку і перевірки мереж, але не повинен перетворюватися на повноцінну IPAM-систему з авторизацією, ролями та складною моделлю адресного простору.

2 ПРОЄКТУВАННЯ ДОДАТКУ ДЛЯ РОЗРАХУНКУ ПАРАМЕТРІВ IPv4/IPv6 МЕРЕЖ

2.1 Функціональні вимоги

Функціональні вимоги описують сценарії, які користувач повинен виконувати через інтерфейс додатку: розрахунок мережі, поділ на підмережі, перевірку перетину, класифікацію адрес, роботу з історією та експортом.

Функціональні вимоги до додатку наведено у таблиці 2.1.

Таблиця 2.1 – Функціональні вимоги до додатку

ID	Вимога	Очікуваний результат
FR-01	Користувач вводить IPv4-адресу з CIDR-префіксом.	Система перевіряє дані та виконує IPv4 calculation.
FR-02	Користувач вводить IPv6-адресу з CIDR-префіксом.	Система розраховує параметри IPv6-мережі.
FR-03	Система підтримує перевірку коректності IP-адрес і префіксів.	Некоректні дані не обробляються, користувач отримує повідомлення про помилку.
FR-04	Система виконує поділ мережі на підмережі за новим префіксом.	Користувач отримує перелік підмереж або обмежену вибірку для великих діапазонів.
FR-05	Система перевіряє перетин мереж.	Для введених мереж визначається наявність або відсутність overlap.
FR-06	Система класифікує IP-адреси.	Визначається тип адреси: private, loopback, link-local, multicast, documentation, global або special.

ID	Вимога	Очікуваний результат
FR-07	Система зберігає історію розрахунків.	Результати записуються у SQLite з датою та типом операції.
FR-08	Система експортує результати.	Користувач отримує CSV або JSON файл.
FR-09	Система підтримує режим пояснення розрахунку.	Користувач бачить основні кроки обчислення.
FR-10	Система виконує спрощене VLSM-планування для IPv4.	За базовою мережею та кількістю хостів пропонується відповідна підмережа.

Сформований перелік функціональних вимог задає основні сценарії взаємодії користувача з додатком.

2.2 Нефункціональні вимоги

Нефункціональні вимоги задають обмеження до якості реалізації: коректність розрахунків, стійку обробку помилок, зручний інтерфейс, модульну структуру коду, тестованість і запуск через Docker Compose.

Нефункціональні вимоги до додатку наведено у таблиці 2.2.

Таблиця 2.2 – Нефункціональні вимоги до додатку

ID	Категорія	Вимога
NFR-01	Коректність	Розрахунки мають відповідати принципам IPv4/IPv6 адресації, CIDR та спеціальним випадкам /31, /32, /127, /128.
NFR-02	Продуктивність	Звичайний розрахунок має виконуватися без помітної затримки для користувача.

ID	Категорія	Вимога
NFR-03	Надійність	Некоректні введені дані не повинні призводити до збоїв додатку.
NFR-04	Масштабованість розрахунків	Для великих IPv6-діапазонів система не повинна перебирати всі адреси.
NFR-05	Зручність	Інтерфейс має містити зрозумілі форми введення та структуровані результати.
NFR-06	Модульність	Алгоритми IPv4, IPv6, subnetting, overlap, classification і VLSM мають бути винесені в окремі модулі.
NFR-07	Тестованість	Ключові алгоритми мають бути доступні для unit-тестування.
NFR-08	Підтримуваність	Код має мати зрозумілу структуру backend/frontend та мінімальну залежність від UI у розрахункових модулях.
NFR-09	Переносимість та розгортання	Додаток має розгортатися через Docker Compose, що дозволяє запускати backend, frontend та сховище даних у контейнеризованому середовищі.

Ці вимоги враховуються під час вибору архітектури, технологій і способу тестування додатку.

2.3 Алгоритм розрахунку параметрів IPv4 мережі

Модуль IPv4 calculation працює з IPv4-адресою як з 32-бітним числом. Кожен октет займає свою частину цього значення, тому адресу можна використовувати в побітових операціях з маскою підмережі.

Після валідації адреси та префікса backend формує маску, wildcard-маску, адресу мережі та broadcast. У результаті користувач отримує не проміжні біти, а готові мережеві параметри.

$$mask = ((2^{prefix} - 1) \ll (32 - prefix)) \quad (2.1)$$

$$network = address \text{ AND } mask \quad (2.2)$$

$$broadcast = network \text{ OR } hostmask \quad (2.3)$$

Для префіксів /0-/30 кількість доступних хостів обчислюється з урахуванням адреси мережі та broadcast. Для /31 і /32 використовується окрема логіка, щоб правильно обробити point-to-point сценарій і одиночну адресу.

Особливості розрахунку usable hosts для IPv4 наведено у таблиці 2.3.

Таблиця 2.3 – Особливості розрахунку usable hosts для IPv4

Префікс	Кількість адрес	Кількість usable hosts
0-30	$2^{(32-\text{prefix})}$	$2^{(32-\text{prefix})} - 2$
/31	2	2 для point-to-point з'єднання
/32	1	1 адреса

Загальна послідовність роботи алгоритму IPv4 calculation така:

- а) зчитати IPv4-адресу та CIDR-префікс;
- б) перевірити, що октети адреси належать діапазону 0-255, а префікс належить діапазону 0-32;
- в) перетворити адресу у 32-бітне число;
- г) сформувати маску та wildcard-маску;
- г) обчислити network address та broadcast address;
- д) визначити first host, last host, total addresses та usable hosts;
- е) класифікувати адресу та сформувати результат для інтерфейсу користувача.

2.4 Алгоритм розрахунку параметрів IPv6 мережі

Модуль IPv6 calculation використовує ту саму ідею маскування, але працює зі 128-бітним адресним простором. Після валідації IPv6-адреса переводиться у внутрішнє числове або байтове подання, над яким виконуються операції для визначення меж діапазону.

$$mask = ((2^{prefix} - 1) \ll (128 - prefix)) \quad (2.4)$$

$$network = address \text{ AND } mask \quad (2.5)$$

$$last = network \text{ OR } hostmask \quad (2.6)$$

Через відсутність broadcast у IPv6 результат розрахунку містить network prefix, першу адресу, останню адресу, кількість адрес і текстові форми представлення адреси.

Наприклад, для 2001:db8::1/64 система визначає префікс 2001:db8::/64, першу адресу 2001:db8:: та останню адресу 2001:db8::ffff:ffff:ffff:ffff. Адреса також класифікується як documentation, оскільки входить до 2001:db8::/32.

Для великих IPv6-мереж кількість адрес може бути дуже великою, тому система не повинна генерувати повний список адрес. Замість цього відображаються межі діапазону, кількість адрес і за потреби обмежена кількість перших підмереж або результатів.

2.5 Алгоритм поділу мережі на підмережі

Модуль subnetting приймає базову мережу та новий префікс. Новий префікс має бути більшим за початковий, інакше система повертає помилку, бо в такому випадку не формується менша підмережа.

$$subnet_size = 2^{(bits - new_prefix)} \quad (2.7)$$

$$subnet_i = base_network + i * subnet_size \quad (2.8)$$

Для IPv4 значення bits дорівнює 32, а для IPv6 - 128. Кількість підмереж визначається різницею між новим і старим префіксом. Наприклад, 192.168.1.0/24 при поділі на /26 дає чотири підмережі.

Приклад поділу мережі 192.168.1.0/24 на підмережі /26 наведено у таблиці 2.4.

Таблиця 2.4 – Приклад поділу мережі 192.168.1.0/24 на /26

№	Підмережа	Діапазон адрес
1	192.168.1.0/26	192.168.1.0 - 192.168.1.63
2	192.168.1.64/26	192.168.1.64 - 192.168.1.127
3	192.168.1.128/26	192.168.1.128 - 192.168.1.191
4	192.168.1.192/26	192.168.1.192 - 192.168.1.255

Приклад показує, що кожна наступна підмережа починається після завершення попереднього діапазону.

2.6 Алгоритм перевірки перетину мереж

У модулі `overlap check` кожна мережа спочатку перетворюється на пару меж: початкову та кінцеву адресу. Після цього система порівнює діапазони і визначає, чи мають вони спільну частину.

$$overlap = (A_start \leq B_end) \text{ AND } (B_start \leq A_end) \quad (2.9)$$

Перевірка перетину виконується так:

- а) зчитати список мереж у CIDR-форматі;
- б) перевірити коректність кожної мережі;
- в) перетворити кожну мережу у діапазон `start-end`;
- г) порівняти діапазони попарно або після сортування за початковою адресою;
- г) сформувати список пар мереж, які перетинаються;
- д) відобразити результат користувачу.

Наприклад, 192.168.1.128/25 входить у діапазон 192.168.1.0/24, тому між цими мережами є перетин. Натомість 192.168.1.0/25 і 192.168.1.128/25 є суміжними, але не мають спільних адрес.

2.7 Алгоритм класифікації IP-адрес

Модуль classification отримує одну IPv4 або IPv6 адресу та перевіряє її належність до службових і спеціальних діапазонів. Для IPv4 перевіряються private, loopback, link-local, multicast, documentation, limited broadcast та інші special-purpose блоки. Для IPv6 перевіряються loopback, unspecified, link-local, unique local, multicast, documentation та інші спеціальні префікси.

Приклади діапазонів для класифікації IP-адрес наведено у таблиці 2.5.

Таблиця 2.5 – Приклади діапазонів для класифікації адрес

Тип	IPv4	IPv6
Loopback	127.0.0.0/8	::1/128
Private / Unique local	10.0.0.0/8; 172.16.0.0/12; 192.168.0.0/16	fc00::/7
Link-local	169.254.0.0/16	fe80::/10
Documentation	192.0.2.0/24; 198.51.100.0/24; 203.0.113.0/24	2001:db8::/32
Multicast	224.0.0.0/4	ff00::/8

Якщо адреса підпадає під кілька умов, система повертає більш конкретну категорію. Наприклад, documentation-адреса не відображається як звичайна global address. Перелік діапазонів для перевірки взято з RFC 1918 та реєстрів IANA [2], [3], [7].

2.8 Спрощений алгоритм VLSM-планування

Спрощений VLSM-модуль підбирає IPv4-підмережу під задану кількість хостів. Він не будує повний адресний план для багатьох відділів і не керує IPAM-інфраструктурою, а виконує одну практичну дію: знаходить мінімальний префікс у межах базової мережі.

Користувач вводить базову мережу, наприклад 192.168.10.0/24, та потрібну кількість хостів, наприклад 50. Система визначає мінімальну

кількість бітів вузлової частини h , за якої виконується умова $2^h - 2 \geq \text{required_hosts}$. Після цього обчислюється префікс як $32 - h$.

$$2^h - 2 \geq \text{required_hosts} \quad (2.10)$$

$$\text{prefix} = 32 - h \quad (2.11)$$

Для 50 хостів мінімальним значенням h є 6, оскільки $2^6 - 2 = 62$. Отже, потрібний префікс дорівнює /26. Якщо базова мережа 192.168.10.0/24, першою запропонованою підмережею може бути 192.168.10.0/26 з діапазоном адрес 192.168.10.0 - 192.168.10.63.

Приклад спрощеного VLSM-планування наведено у таблиці 2.6.

Таблиця 2.6 – Приклад спрощеного VLSM-планування

Вхідні дані	Розрахунок	Результат
Базова мережа: 192.168.10.0/24; потрібно 50 хостів	$2^6 - 2 = 62$, prefix = 32 - 6	192.168.10.0/26
Базова мережа: 10.0.0.0/24; потрібно 10 хостів	$2^4 - 2 = 14$, prefix = 32 - 4	10.0.0.0/28

Якщо введена кількість хостів не поміщається в базову мережу, додаток повертає помилку. У майбутньому цей модуль можна розширити до роботи зі списком вимог і пошуком вільних блоків.

2.9 Архітектура програмного забезпечення

Додаток спроектовано як клієнт-серверну систему. Backend реалізовано на Go з використанням Gin [13], [14], frontend - на React з TypeScript, Vite та Tailwind CSS [15], а історія розрахунків зберігається у SQLite [16]. Така структура дає окремий API, окремий інтерфейс і просте файлове сховище без авторизації та складної IPAM-моделі.

Для запуску використовується Docker Compose. У конфігурації описано backend, frontend і volume для SQLite, тому демонстраційне середовище

можна підняти однією командою без ручного налаштування залежностей [17].

Backend поділено на модулі IPv4 calculation, IPv6 calculation, validation, subnetting, overlap check, classification, simplified VLSM, history та export. Frontend має відповідні функціональні сторінки, тому структура інтерфейсу повторює основні сценарії користувача.

Структурну схему веб-додатку наведено на рисунку 2.1.

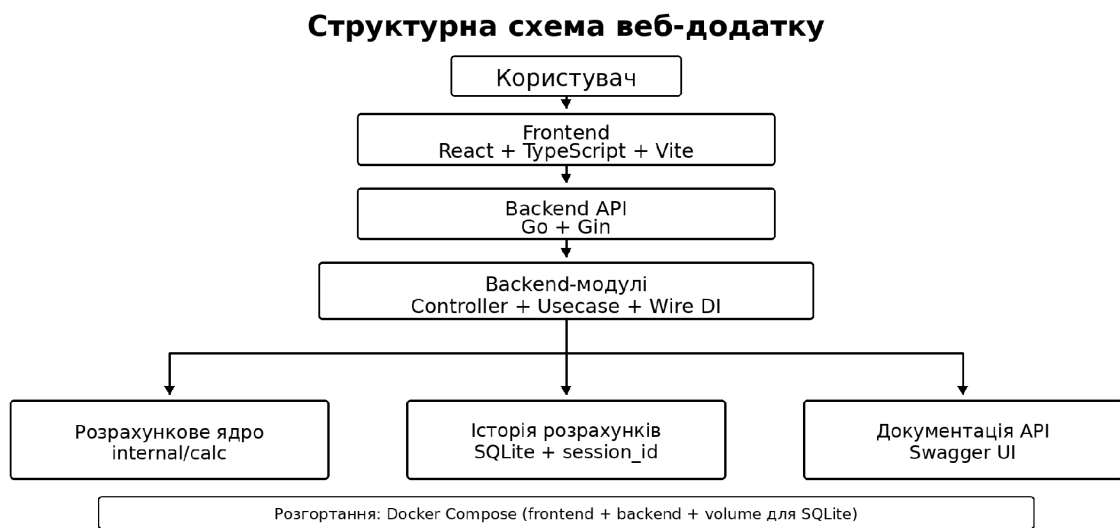


Рисунок 2.1 – Структурна схема веб-додатку

Типовий сценарій роботи такий: користувач вводить дані у frontend-формі, frontend надсилає HTTP-запит до backend, backend виконує валідацію і розрахунок, після чого повертає результат або повідомлення про помилку. Якщо передано session_id, результат також записується в історію.

Функціональні модулі додатку наведено на рисунку 2.2.



Рисунок 2.2 – Функціональні модулі додатку
Основні модулі backend-частини наведено у таблиці 2.7.

Таблиця 2.7 – Основні модулі backend-частини

Модуль	Призначення
validation	Перевірка коректності IP-адрес, CIDR-префіксів та вхідних параметрів.
ipv4	Розрахунок параметрів IPv4-мереж.
ipv6	Розрахунок параметрів IPv6-мереж.
subnetting	Поділ мережі на підмережі за новим префіксом.
overlap	Перевірка перетину двох або кількох мереж.
classification	Визначення типу IP-адреси за спеціальними діапазонами.
vlsn	Спрощений підбір IPv4-підмережі за кількістю хостів.
history	Збереження та отримання історії розрахунків.
export	Формування CSV або JSON результатів.

Такий поділ дозволяє окремо змінювати й тестувати розрахункові модулі, не прив'язуючи їх до конкретної сторінки інтерфейсу.

2.10 Структура бази даних

База даних у додатку використовується тільки для історії розрахунків. Окремі користувачі, ролі та проекти адресного простору не створюються, тому достатньо однієї таблиці `calculation_history`.

Зв'язок історії з конкретним браузером виконується через анонімний `session_id`. Frontend створює його під час першого відкриття додатку, зберігає у `localStorage` і передає до backend у HTTP-заголовку.

На сторінці історії frontend запитує записи поточної сесії, а backend повертає дані за відповідним `session_id`. Завдяки цьому історія працює без реєстрації користувачів і без таблиці `user`.

Структуру таблиці `calculation_history` наведено у таблиці 2.8.

Таблиця 2.8 – Структура таблиці `calculation_history`

Поле	Тип	Призначення
<code>id</code>	INTEGER PRIMARY KEY	Унікальний ідентифікатор запису.
<code>session_id</code>	TEXT	Ідентифікатор анонімної сесії користувача, що зберігається у <code>localStorage</code> браузера.
<code>calculation_type</code>	TEXT	Тип операції: <code>ipv4</code> , <code>ipv6</code> , <code>subnetting</code> , <code>overlap</code> , <code>classification</code> , <code>vlsn</code> .
<code>input_data</code>	TEXT	Вхідні дані у JSON або текстовому форматі.
<code>result_data</code>	TEXT	Результат розрахунку у JSON-форматі.
<code>created_at</code>	DATETIME	Дата і час створення запису.

`input_data` та `result_data` зберігаються як JSON-рядки, тому одна таблиця підходить для різних типів розрахунків. Якщо користувач очищує

localStorage, попередній session_id втрачається, але для навчального додатку це прийнятне обмеження. SQL-схему таблиці наведено у додатку А.

2.11 Проєктування інтерфейсу користувача

Інтерфейс проєктується навколо основних сценаріїв: IPv4 calculation, IPv6 calculation, subnetting, overlap check, classification, simplified VLSM, history та export. Для переходу між ними використовується навігація у вигляді меню.

Кожна сторінка містить форму введення, кнопку запуску, блок результату і повідомлення про помилки. Результати подаються у вигляді таблиць або панелей, а explanation mode відкривається окремим блоком.

Основні екрани користувацького інтерфейсу наведено у таблиці 2.9.

Таблиця 2.9 – Основні екрани додатку

Екран	Вхідні дані	Результат
IPv4 calculator	IPv4-адреса та CIDR-префікс	Network, broadcast, mask, wildcard, first/last host, total, usable, classification.
IPv6 calculator	IPv6-адреса та CIDR-префікс	Network prefix, first address, last address, total, canonical form, classification.
Subnetting	Базова мережа та новий префікс	Перелік підмереж або обмежена вибірка.
Overlap check	Дві або більше мережі	Повідомлення про наявність або відсутність перетину.
VLSM	Базова IPv4-мережа та кількість хостів	Рекомендована підмережа, маска, діапазон, usable hosts.

Продовження таблиці 2.9

Екран	Вхідні дані	Результат
History	Анонімний session_id поточного браузера або параметри фільтрації.	Список попередніх розрахунків поточної сесії з можливістю повторного перегляду та експорту.

Інтерфейс підтримує копіювання результатів, відображення помилок без перезавантаження сторінки та експорт у CSV або JSON. Функції на кшталт drag-and-drop, real-time topology, авторизації або повної ІРАМ-структури не реалізуються, бо вони виходять за межі поставленої задачі.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ДОДАТКУ ДЛЯ РОЗРАХУНКУ ПАРАМЕТРІВ IPv4/IPv6 МЕРЕЖ

У цьому розділі описано фактичну реалізацію: структуру backend і frontend, шлях HTTP-запиту, розрахункові модулі, історію, експорт, Swagger-документацію, Wire-композицію та результати тестування.

Реалізований додаток підтримує погоджений набір функцій: IPv4/IPv6 calculation, CIDR validation, subnetting, overlap check, classification, simplified VLSM, history, export і explanation mode. Користувач працює з цими інструментами через SPA-інтерфейс, а всі розрахунки виконуються на backend-частині.

Практична частина працює як клієнт-серверна система. Backend відповідає за валідацію, розрахунки, історію та export, а frontend - за форми, навігацію, відображення результатів і збереження session_id у localStorage.

3.1 Обґрунтування вибору технологій

Стек обирався під невеликий веб-додаток: потрібні були окремий API, SPA-інтерфейс, просте сховище історії, автоматизовані тести і запуск без ручного налаштування середовища.

Go використано для backend-частини, оскільки він добре підходить для компактних HTTP-сервісів і має пакет net/netip для роботи з IP-адресами. Gin відповідає за маршрутизацію, а розрахункова логіка винесена за межі HTTP-handler'ів.

Frontend реалізовано на React з TypeScript. React використовується для окремих сторінок-інструментів, а TypeScript описує структуру API-відповідей і стану форм. Vite відповідає за збірку, TailwindCSS - за оформлення dashboard-інтерфейсу.

SQLite використовується як локальне файлове сховище історії. Для цього додатку не потрібна окрема серверна СКБД, бо історія прив'язується до session_id і не містить облікових записів користувачів. Розгортання виконується через Docker Compose.

Swagger додає сторінку документації API за маршрутом `/swagger/index.html` [18]. Google Wire використовується для збирання залежностей backend-частини [19]: у `cmd/app` окремо описано `InfrastructureSet`, `DatabaseSet`, `AdapterSet`, `UsecaseSet`, `ControllerSet` та `AppSet`.

Використані технології реалізації наведено у таблиці 3.1.

Таблиця 3.1 – Використані технології реалізації

Компонент	Технології	Призначення у проєкті
Backend	Go, Gin	REST API, розрахункові модулі та робота з історією.
Frontend	React, TypeScript, Vite, Tailwind CSS	SPA-інтерфейс, форми введення та відображення результатів.
База даних	SQLite	Збереження історії розрахунків за <code>session_id</code> .
Документація API	Swagger, gin-swagger	Автоматизований опис endpoint'ів і Swagger UI.
Dependency injection	Google Wire	Генерація композиції залежностей backend-частини.
Розгортання	Docker Compose, Nginx	Контейнеризований запуск backend/frontend і проху API-запитів.
Тестування	Go testing, testify	Unit-тести розрахункових модулів і HTTP-сценаріїв.

Обраний стек дозволив реалізувати API, frontend-інтерфейс, історію, експорт, документацію, контейнерний запуск і тести без додавання авторизації або складної IPAM-моделі.

3.2 Реалізація backend-частини

Backend-частина побудована за модульним принципом. Основна розрахункова логіка винесена до пакета `internal/calc`, а кожен користувацький

сценарій має окремий usecase-модуль: IPv4, IPv6, subnetting, overlap, classification, VLSM, history та export. Такий підхід дозволяє тестувати алгоритми без запуску HTTP-сервера та не змішувати математичні операції з обробкою запитів.

Архітектура backend-частини є наближеною до clean architecture та vertical slice architecture. Код організовано не лише за технічними шарами, а й за окремими сценаріями використання. Для кожної функції додатку створено окремий usecase-пакет: `ipv4_uc`, `ipv6_uc`, `subnetting_uc`, `overlap_uc`, `classification_uc`, `vlsn_uc`, `history_uc` та `export_uc`. Кожен такий пакет містить DTO-структури, usecase-логіку, HTTP-handler і provider set для Wire.

Типовий шлях запиту має такий вигляд: HTTP-route у controller передає запит до handler, handler зчитує та валідує Input, після чого викликає `usecase.Execute()`. Usecase звертається до чистих розрахункових функцій з `internal/calc` або до `adapter/sqlite` для роботи з історією. Відповідь повертається через спільні helper-функції `pkg/httpserver`, що забезпечує єдиний формат `success/data` або `success/error` для всіх endpoint'ів.

Точкою входу backend-додатку є `cmd/app/main.go`, а основна структура застосунку визначена у `cmd/app/app.go`. Компонування залежностей реалізовано через Google Wire. Файл `cmd/app/wire.go` містить `injector stub` з `wire.Build(...)`, а повна композиція винесена до `cmd/app/providers.go`. У `providers.go` залежності поділено на layer sets: `InfrastructureSet`, `DatabaseSet`, `AdapterSet`, `UsecaseSet`, `ControllerSet` та `AppSet`. Згенерований файл `wire_gen.go` створює готовий граф залежностей, тому під час запуску додатку не потрібно вручну створювати всі usecase-модулі та HTTP-handler'и.

Структуру backend-частини наведено у таблиці 3.2.

Таблиця 3.2 – Структура backend-частини

Каталог або файл	Призначення
<code>cmd/app</code>	Запуск backend-сервера, створення залежностей і реєстрація маршрутизатора.

Продовження таблиці 3.2

Каталог або файл	Призначення
config	Зчитування параметрів запуску: PORT, SQLITE_PATH, FRONTEND_ORIGIN.
internal/calc	Чисті функції розрахунку IPv4, IPv6, підмереж, перетинів, класифікації та VLSM.
internal/*_uc	Usecase-модулі та HTTP-обробники для окремих функцій додатку.
internal/adapters/sqlite	Адаптер збереження, вибірки й видалення історії у SQLite.
internal/domain	Доменно-орієнтовані типи, зокрема тип запису історії.
pkg/httpserver	Спільні HTTP-відповіді, CORS і health check.
pkg/logger	Ініціалізація logger для діагностики помилок.
cmd/app/providers.go	Опис layer set для Wire-композиції: інфраструктура, база даних, адаптери, usecase, controller та app.
cmd/app/wire.go	Injector stub для Google Wire з викликом wire.Build(...).
cmd/app/wire_gen.go	Згенерований Wire-файл, який виконує runtime-композицію залежностей.
backend/docs	Згенеровані Swagger-файли документації API.
Makefile	Команди для запуску тестів, генерації Swagger-документації та Wire-коду.

Завдяки винесенню композиції залежностей у providers.go структура backend-частини стала більш прозорою: окремо видно рівень інфраструктури, рівень бази даних, адаптерів, usecase-модулів, controller та фінального App. Для зміни або додавання нового сценарію достатньо створити новий usecase-пакет, додати його provider set до UsecaseSet і регенерувати wire_gen.go командою make wire.

У розрахункових модулях використовуються стандартні типи `netip.Addr` та `netip.Prefix`. Для IPv4 адреси перетворюються у 32-бітне числове представлення, що дозволяє визначати межі мережі, broadcast-адресу, маску, wildcard-маску та діапазон доступних хостів. Для IPv6 використовується `math/big`, оскільки адресний простір має довжину 128 бітів і не може бути безпечно представлений звичайним 64-бітним цілим числом.

HTTP API реалізовано у вигляді набору REST-подібних `endpoint`'ів з префіксом `/api`. Усі розрахункові POST-запити приймають JSON-тіло, а для збереження історії `frontend` передає заголовок `X-Session-ID`. Успішна відповідь має єдину структуру з полями `success` та `data`, а помилки повертаються зі зрозумілим повідомленням.

Фактичні API `endpoint`'и `backend`-частини наведено у таблиці 3.3.

Таблиця 3.3 – Фактичні API `endpoint`'и `backend`-частини

Метод	Endpoint	Призначення
GET	/api/health	Перевірка доступності backend-сервера.
POST	/api/ipv4/calculate	Розрахунок параметрів IPv4-мережі.
POST	/api/ipv6/calculate	Розрахунок параметрів IPv6-мережі.
POST	/api/subnetting	Поділ IPv4 або IPv6 CIDR-блока на підмережі.
POST	/api/overlap	Перевірка перетину списку мережевих діапазонів.
POST	/api/classification	Класифікація IPv4 або IPv6 адреси.
POST	/api/vlsm	Спрощене VLSM-планування для IPv4.
GET	/api/history	Отримання історії розрахунків поточної сесії.
DELETE	/api/history/:id	Видалення запису історії поточної сесії.
GET	/api/export/history/json	Експорт історії поточної сесії у JSON.
GET	/api/export/history/csv	Експорт історії поточної сесії у CSV.

Метод	Endpoint	Призначення
GET	/swagger/index.html	Перегляд інтерактивної Swagger-документації API.

Документація endpoint'ів формується автоматично на основі Swagger-annotations, розміщених у HTTP-handler'ах usecase-пакетів. Інтерфейс Swagger UI доступний за маршрутом /swagger/index.html і використовується для перевірки структури запитів, відповідей і HTTP-статусів під час розробки та демонстрації API.

Фрагмент реєстрації HTTP-маршрутів backend-частини винесено до додатку А. У основній частині наведено узагальнену таблицю endpoint'ів, достатню для пояснення API додатку.

Приклад запиту до модуля IPv4 має такий вигляд: POST /api/ipv4/calculate з тілом {"address":"192.168.1.10","prefix":24}. У відповідь backend повертає адресу мережі 192.168.1.0, broadcast-адресу 192.168.1.255, маску 255.255.255.0, wildcard-маску 0.0.0.255, діапазон хостів 192.168.1.1–192.168.1.254, загальну кількість адрес 256, кількість доступних хостів 254 та класифікацію private.

Окрему увагу приділено повідомленням пояснення. Кожен розрахунок повертає масив explanation, у якому кроки описано українською мовою у формі, зрозумілій користувачу без спеціальної підготовки. Наприклад, для IPv4 користувач бачить не формули, а пояснення, що адреса належить до певного адресного блоку, початок блоку є адресою мережі, кінець блоку є broadcast-адресою, а пристроям можна призначати адреси з указанного діапазону.

3.3 Реалізація frontend-частини

Frontend-частина реалізована як односторінковий React-додаток. Перехід між інструментами виконується через параметр view у URL,

наприклад `?view=ipv4` або `?view=vlsm`. Такий підхід не потребує окремого маршрутизатора, але дозволяє відкривати конкретний інструмент без додаткових дій користувача.

Frontend-частина побудована за feature-based підходом. Основні інструменти додатку розміщені у каталозі `src/features`: `home`, `ipv4`, `ipv6`, `subnetting`, `overlap`, `classification`, `vlsm` та `history`. Кожна feature містить сторінку інструменту, API-виклик і, за потреби, окремий компонент для відображення результату, наприклад таблицю підмереж або список знайдених перетинів.

Каталог `src/shared` містить спільні елементи, які використовуються декількома функціональними модулями: API-клієнт, типи відповідей, `hook` для асинхронних дій, `helper` для `session_id`, `layout`-компоненти, `result panel` та `feedback`-компоненти. Завдяки цьому повторювана логіка не дублюється у сторінках окремих калькуляторів.

Клієнтська частина розділена за функціональними модулями. Кожна сторінка містить форму введення, виклик відповідної API-функції та блок результату. Спільні компоненти форм, кнопок, повідомлень, панелі результатів і `session_id` винесені до каталогу `shared`, щоб уникнути дублювання коду між інструментами.

Структуру frontend-частини наведено у таблиці 3.4.

Таблиця 3.4 – Структура frontend-частини

Каталог або файл	Призначення
<code>src/app</code>	Кореневий компонент додатку, навігація та вибір активної сторінки.
<code>src/features/ipv4</code>	Сторінка та API-функція розрахунку IPv4.
<code>src/features/ipv6</code>	Сторінка та API-функція розрахунку IPv6.
<code>src/features/subnetting</code>	Форма поділу мережі та таблиця підмереж.

Каталог або файл	Призначення
src/features/overlap	Форма введення списку мереж і відображення знайдених перетинів.
src/features/classification	Класифікація IPv4/IPv6 адреси.
src/features/vlsm	Спрощене VLSM-планування для IPv4.
src/features/history	Перегляд, оновлення, видалення й експорт історії розрахунків.
src/shared/api	Єдиний HTTP-клієнт, типи API-відповідей і передача X-Session-ID.
src/shared/ui	Спільні UI-компоненти: кнопки, поля, layout, result panel, feedback.
src/shared/lib	Форматування, завантаження файлів і робота з localStorage session_id.
src/app/navigation.ts	Список інструментів, title, description та href для dashboard-навігації.
src/shared/hooks/useAsyncAction.ts	Універсальний hook для обробки асинхронних дій, станів loading та помилок.
nginx.conf	Проксуювання /api-запитів до backend і роздача production-збірки frontend.

У production-режимі frontend збирається у статичні файли та роздається через Nginx у Docker-контейнері. Конфігурація nginx.conf також проксирує запити з префіксом /api до backend-сервісу, тому користувач працює з єдиною frontend-адресою, а взаємодія з API відбувається прозоро для інтерфейсу.

Усі сторінки мають однакову логіку взаємодії: користувач вводить дані, натискає кнопку розрахунку, frontend відправляє запит до backend, отримує JSON-відповідь і показує результат у таблиці. Якщо backend повертає помилку валідації, вона відображається без перезавантаження сторінки. Для

зручності результати можна експортувати безпосередньо з панелі результату у JSON або CSV.

Для роботи з історією frontend створює анонімний ідентифікатор сесії. Він зберігається у localStorage під ключем `ipnet.session.v1`. Якщо запис уже існує, додаток використовує його повторно; якщо ні, генерує нове значення. Під час кожного API-запиту `session_id` передається у заголовок X-Session-ID. Фрагмент реалізації передачі `session_id` наведено у додатку Б.

Інтерфейс був спроектований як робочий інструмент, а не як декоративна демонстраційна сторінка. Основний екран містить короткий список інструментів, бокову навігацію та робочу область. На сторінках розрахунків основний акцент зроблено на формі введення, результаті, поясненні та експорті.

3.4 Реалізація модуля історії розрахунків

Модуль історії реалізовано без авторизації користувачів. Такий підхід відповідає рівню задачі: додаток призначений для навчального та базового практичного використання, тому створення облікових записів, ролей і механізмів доступу було б надлишковим. Замість цього історія прив'язується до анонімної сесії браузера.

Коли користувач уперше відкриває додаток, frontend створює `session_id` і зберігає його у localStorage. Після виконання розрахунку backend отримує цей ідентифікатор через заголовок X-Session-ID і зберігає запис у таблиці `calculation_history`. Під час відкриття сторінки історії backend вибирає тільки ті записи, які мають такий самий `session_id`.

Реалізовану структуру таблиці `calculation_history` наведено у таблиці 3.5.

Таблиця 3.5 – Структура таблиці calculation_history

Поле	Тип	Призначення
id	INTEGER PRIMARY KEY AUTOINCREMENT	Унікальний ідентифікатор запису історії.
session_id	TEXT NOT NULL	Анонімний ідентифікатор сесії браузера.
calculation_type	TEXT NOT NULL	Тип операції: ipv4, ipv6, subnetting, overlap, classification або vlsm.
input_data	TEXT NOT NULL	Вхідні дані розрахунку у JSON-представленні.
result_data	TEXT NOT NULL	Результат розрахунку у JSON-представленні.
created_at	DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP	Дата й час створення запису.

Для прискорення вибірки створено індекс idx_calculation_history_session_created за полями session_id та created_at DESC. Це дозволяє швидко отримувати останні записи поточної сесії навіть після накопичення значної кількості розрахунків. SQL-структуру таблиці наведено у додатку А.

На сторінці історії користувач може переглянути попередні розрахунки поточної сесії, оновити список, видалити окремий запис або експортувати всю історію у JSON чи CSV. Видалення також перевіряє session_id, тому користувач не може видалити запис іншої сесії через просту зміну id у URL.

3.5 Реалізація експорту результатів

У додатку реалізовано два рівні експорту. Перший рівень виконується на frontend-частині для поточного результату розрахунку: після отримання

відповіді користувач може завантажити її у JSON або CSV. Другий рівень реалізовано на backend-частині для історії поточної сесії: сервер вибирає всі записи `session_id` і формує файл `calculation-history.json` або `calculation-history.csv`.

Реалізовані варіанти експорту наведено у таблиці 3.6.

Таблиця 3.6 – Реалізовані варіанти експорту

Об'єкт експорту	Формат	Місце реалізації	Сценарій використання
Поточний результат	JSON	Frontend	Збереження повної відповіді розрахунку для повторного аналізу.
Поточний результат	CSV	Frontend	Перенесення основних параметрів у таблицю або звіт.
Історія поточної сесії	JSON	Backend	Отримання всіх записів історії у структурованому форматі.
Історія поточної сесії	CSV	Backend	Експорт історії для перегляду в табличному редакторі.

Файл CSV для історії містить заголовки `id`, `created_at`, `calculation_type`, `input_data` та `result_data`. Поля `input_data` і `result_data` зберігаються у JSON-формі всередині CSV-рядків, тому один формат може описувати результати різних типів операцій без створення окремих колонок для IPv4, IPv6 або VLSM.

Експорт реалізовано так, щоб користувач не вводив додаткові параметри. Для історії достатньо натиснути кнопку JSON або CSV на сторінці History, після чого frontend надсилає запит з поточним X-Session-ID.

3.6 Опис інтерфейсу користувача

Інтерфейс користувача складається з бокового меню та робочої області. Бокове меню містить тільки основні інструменти: огляд, IPv4, IPv6,

підмережі, перетини, класифікацію, VLSM та історію. Це зменшує кількість зайвих елементів і дозволяє користувачу швидко перейти до потрібного сценарію. Головна сторінка не дублює весь функціонал у вигляді складної панелі, а лише показує доступні інструменти.

Головну сторінку додатку наведено на рисунку 3.1.

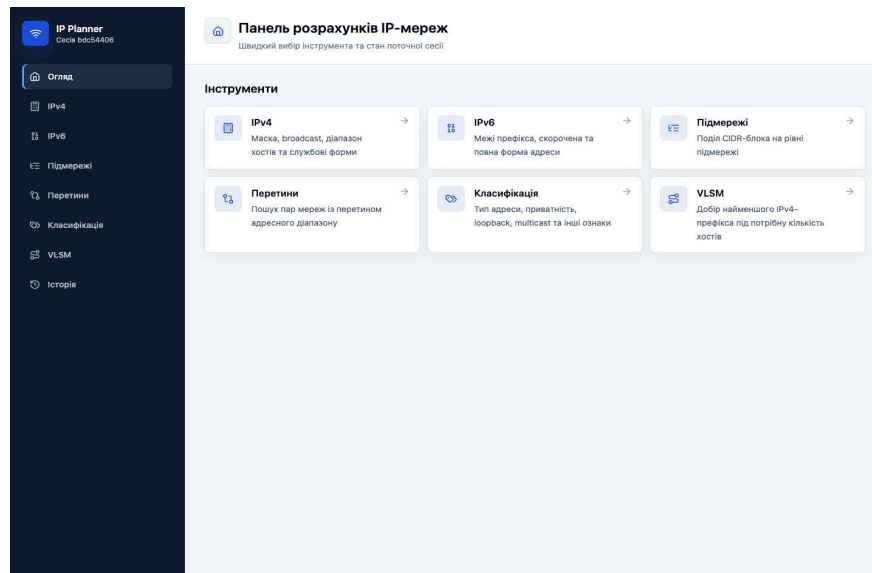


Рисунок 3.1 – Головна сторінка додатку

На сторінці IPv4 користувач може ввести адресу і префікс окремо або перемкнутися на CIDR-формат. Після запуску розрахунку відображаються адреса мережі, broadcast-адреса, маска, wildcard-маска, перша та остання адреса хоста, кількість адрес, кількість доступних хостів і класифікація.

Результат розрахунку параметрів IPv4-мережі наведено на рисунку 3.2.

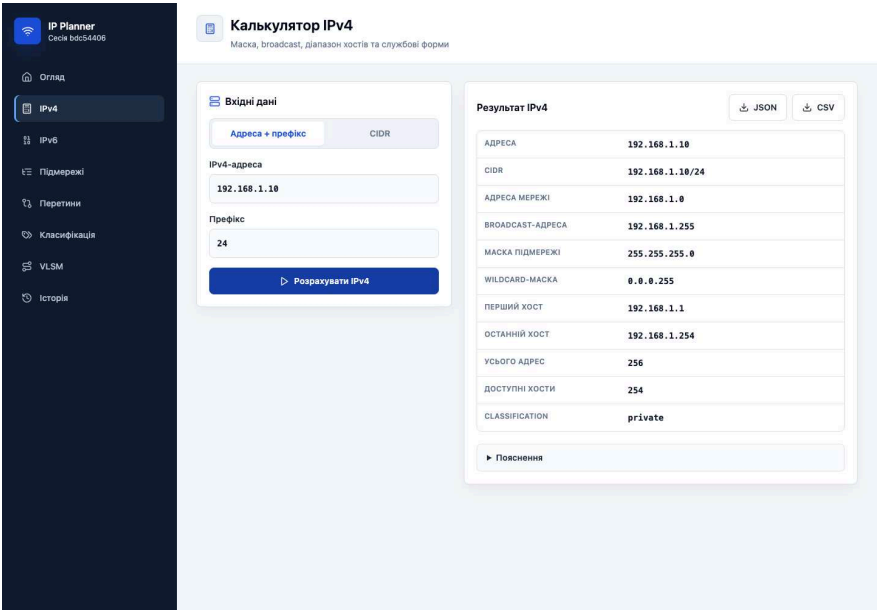


Рисунок 3.2 – Результат розрахунку параметрів IPv4-мережі

Сторінка IPv6 має подібну структуру, але результат адаптовано до особливостей IPv6: замість broadcast-адреси показуються межі префікса, перша та остання адреса, кількість адрес, скорочена та повна форма запису.

Результат розрахунку параметрів IPv6-мережі наведено на рисунку 3.3.

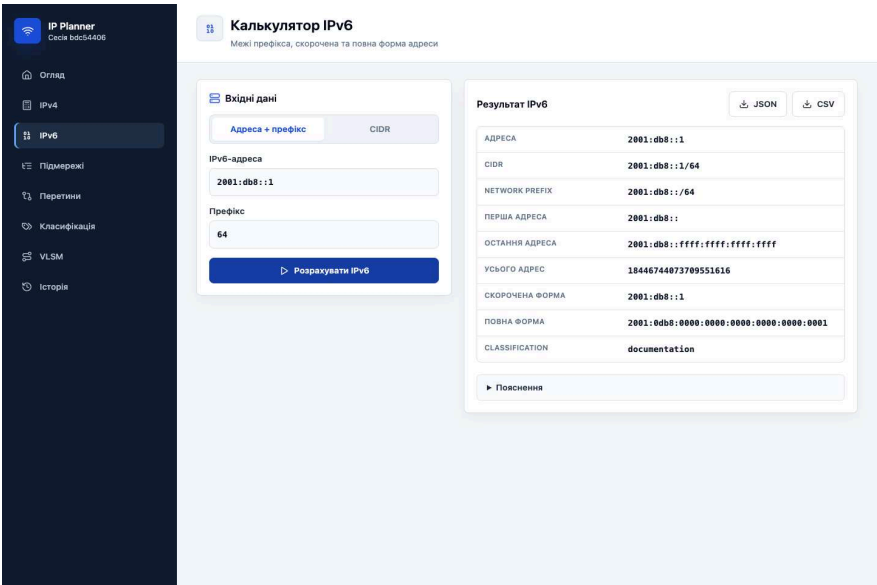


Рисунок 3.3 – Результат розрахунку параметрів IPv6-мережі

Модуль поділу мережі приймає базову мережу та новий префікс. Результат відображається не лише як загальні параметри, а й як таблиця підмереж з CIDR, початковою й кінцевою адресою та кількістю адрес.

Приклад поділу IPv4-мережі на підмережі наведено на рисунку 3.4.

Поділ на підмережі
Поділ CIDR-блока на рівні підмережі

Вхідні дані

Базова мережа: 192.168.1.0/24

Новий префікс: 26

[Поділити мережу](#)

Результат поділу [JSON](#) [CSV](#)

CIDR	МЕРЕЖА	ПЕРША	ОСТАННЯ	УСЬОГ
192.168.1.0/26	192.168.1.0	192.168.1.1	192.168.1.62	64
192.168.1.64/26	192.168.1.64	192.168.1.65	192.168.1.126	64
192.168.1.128/26	192.168.1.128	192.168.1.129	192.168.1.190	64
192.168.1.192/26	192.168.1.192	192.168.1.193	192.168.1.254	64

[Пояснення](#)

Рисунок 3.4 – Поділ IPv4-мережі на підмережі

Модуль перевірки перетину дозволяє ввести декілька мереж по одній у рядку. Після розрахунку система повідомляє, чи знайдено перетини, і показує пари мереж, які мають спільний адресний діапазон.

Перевірку перетину мережевих діапазонів наведено на рисунку 3.5.

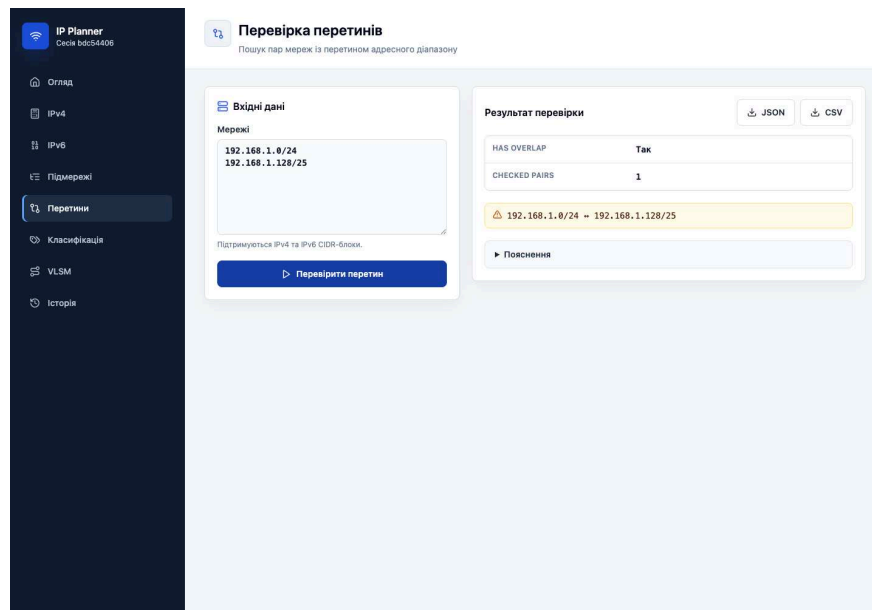


Рисунок 3.5 – Перевірка перетину мережевих діапазонів

Сторінка класифікації визначає тип IP-адреси. Система розрізняє приватні, loopback, link-local, multicast, documentation, global та інші службові діапазони для IPv4 і IPv6.

Результат класифікації IP-адреси наведено на рисунку 3.6.

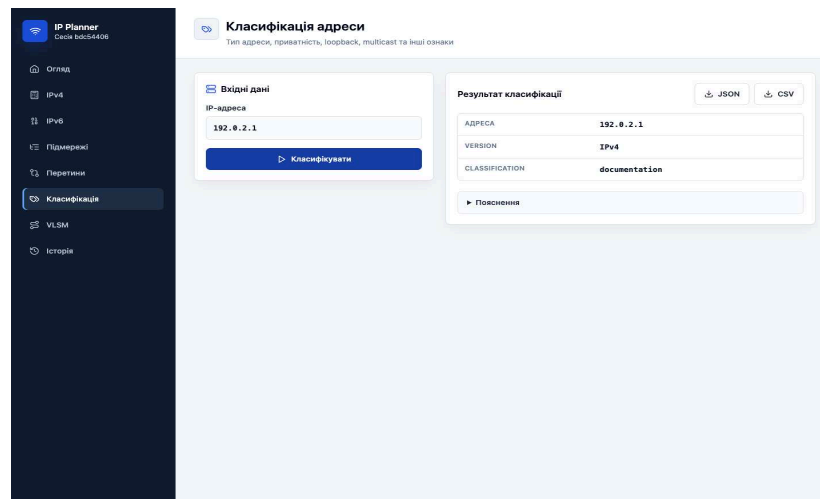


Рисунок 3.6 – Класифікація IP-адреси

VLSM-модуль реалізовано у спрощеному вигляді: користувач вводить базову IPv4-мережу та потрібну кількість хостів, після чого система пропонує

мінімальний префікс і підмережу, яка може вмістити задану кількість пристроїв.

Спрощене VLSM-планування наведено на рисунку 3.7.

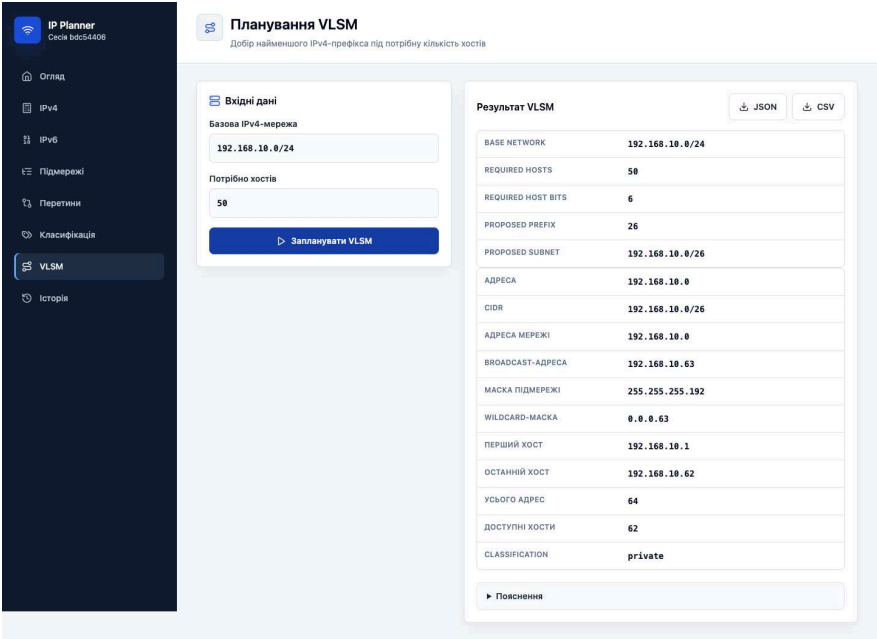


Рисунок 3.7 – Спрощене VLSM-планування

Сторінка історії показує записи поточної сесії. Відображається session_id, а поруч розміщені кнопки оновлення та експорту.

Історію розрахунків поточної сесії наведено на рисунку 3.8.

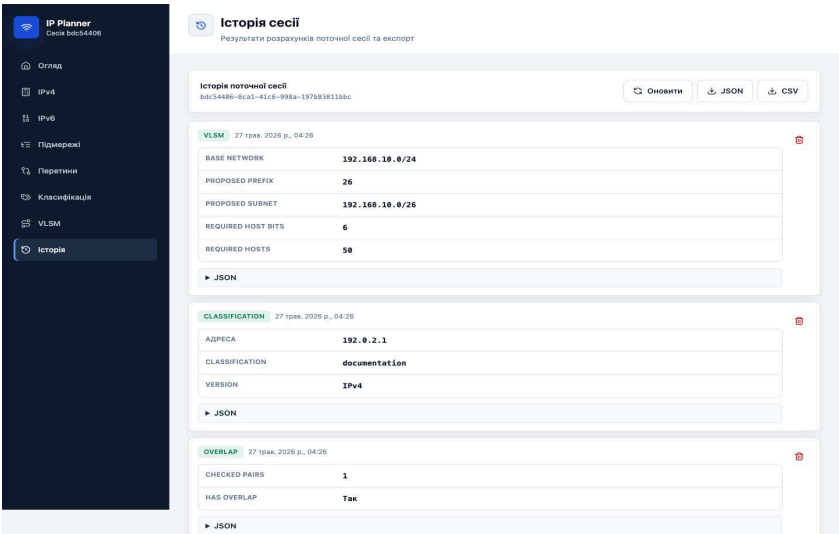


Рисунок 3.8 – Історія розрахунків поточної сесії

У всіх формах реалізовано єдину поведінку: кнопка запуску, блок помилки, таблиця результату, кнопки експорту та блок пояснення. Це робить інтерфейс передбачуваним: після вивчення одного інструмента користувач може працювати з іншими без додаткового навчання.

3.7 Тестування розробленого додатку

Тестування виконувалося на трьох рівнях: перевірка чистих розрахункових функцій backend-частини, перевірка HTTP-сценарію з історією та перевірка production-збірки frontend-частини. Основну увагу приділено граничним випадкам, оскільки саме вони найчастіше призводять до помилок у мережевих калькуляторах.

Unit-тести backend-частини охоплюють стандартну IPv4-мережу /24, point-to-point мережу /31, одиночну адресу /32, некоректні IPv4-адреси та некоректні префікси. Для IPv6 перевірено documentation-префікс 2001:db8::/64, loopback ::1/128, link-local адресу, малу мережу /127, помилкову адресу та префікс понад 128.

Окремо перевірено subnetting /24 -> /26, overlap true/false, класифікацію службових діапазонів та VLSM-підбір префікса для 50 і 10 хостів.

Основні тестові сценарії наведено у таблиці 3.7.

Таблиця 3.7 – Основні тестові сценарії

№	Модуль / сценарій	Очікуваний і фактичний результат	Статус
1	IPv4: 192.168.1.10/24	Отримано мережу 192.168.1.0, broadcast 192.168.1.255, маску 255.255.255.0 та 254 доступні хости; результат відповідає очікуваному.	passed

№	Модуль / сценарій	Очікуваний і фактичний результат	Статус
2	IPv4: /31 та /32	Для /31 оброблено дві адреси point-to-point; для /32 визначено одну адресу; результат відповідає очікуваному.	passed
3	IPv4 validation	Некоректні IPv4-адреси та префікси понад 32 відхиляються з повідомленням про помилку.	passed
4	IPv6: 2001:db8::1/64	Отримано префікс 2001:db8::/64 та класифікацію documentation; результат відповідає очікуваному.	passed
5	IPv6: ::1/128 та /127	Loopback-адреса та мала мережа /127 обробляються коректно.	passed
6	Subnetting: 192.168.1.0/24 -> /26	Сформовано чотири підмережі від 192.168.1.0/26 до 192.168.1.192/26.	passed
7	Overlap check	Для 192.168.1.0/24 і 192.168.1.128/25 виявлено перетин; для суміжних /25 перетин відсутній.	passed
8	Classification	Адреси private, loopback, link-local, multicast, documentation та global отримують очікувану категорію.	passed
9	VLSM: 192.168.10.0/24 + 50 hosts	Запропоновано префікс /26 і підмережу 192.168.10.0/26.	passed
10	History та session_id	Після POST-розрахунку GET /api/history повертає запис поточної сесії за X-Session-ID.	passed

Команда запуску backend-тестів:

```
cd backend
go test ./...
```

Результат виконання тестів показав успішне проходження пакетів internal/adapters/sqlite, internal/calc та internal/controller/http_v1. Детальний вивід команд збережено в електронних матеріалах.

Для frontend-частини виконано production-збірку командою npm run build. Успішне виконання цієї команди підтверджує коректність TypeScript-перевірки та складання Vite. Детальний вивід збірки додано до електронних матеріалів.

Додатково було перевірено роботу додатку у браузері: для кожної основної сторінки виконано розрахунок із типовими вхідними даними, після чого сформовано скриншоти, наведені у підрозділі 3.6. Це підтверджує не лише коректність backend-алгоритмів, а й працездатність повного користувацького сценарію від введення даних до відображення результату.

3.8 Аналіз отриманих результатів

Практична частина показала, що обраний обсяг функцій є достатнім для теми роботи. У додатку реалізовано не один окремий калькулятор, а набір пов'язаних інструментів: IPv4 та IPv6 calculation, subnetting, overlap check, classification, simplified VLSM, history, export і explanation mode. Це дозволяє перевіряти як окрему адресу, так і невеликий адресний план.

Відповідність реалізації основним вимогам наведено у таблиці 3.8.

Таблиця 3.8 – Відповідність реалізації основним вимогам

Вимога	Стан реалізації
Розрахунок параметрів IPv4-мереж	Реалізовано: network, broadcast, mask, wildcard, first/last host, total/usable hosts, classification.

Вимога	Стан реалізації
Підтримка CIDR	Реалізовано для IPv4 та IPv6 через окреме поле CIDR або пару address + prefix.
Розрахунок параметрів IPv6-мереж	Реалізовано: network prefix, first/last address, total addresses, compressed/expanded form, classification.
Валідація введення	Реалізовано на backend-рівні для адрес, префіксів, списків мереж і кількості хостів.
Поділ мережі на підмережі	Реалізовано для IPv4 та IPv6 CIDR-блоків із лімітом повернення великої кількості підмереж.
Перевірка перетину мереж	Реалізовано через порівняння адресних діапазонів.
Класифікація адрес	Реалізовано для основних спеціальних IPv4/IPv6 діапазонів.
Історія розрахунків	Реалізовано через SQLite та session_id з localStorage.
Експорт JSON/CSV	Реалізовано для поточного результату та історії поточної сесії.
Пояснення розрахунків	Реалізовано у вигляді зрозумілих українських кроків у відповіді API та інтерфейсі.
Спрощене VLSM	Реалізовано для IPv4: підбір мінімального префікса під задану кількість хостів.

Додаток свідомо не розширювався до IPAM-системи. У ньому немає реєстрації користувачів, ролей, організаційної структури, резервування адрес або автоматичного сканування мережі. Історія прив'язується до session_id у браузері, що відповідає рівню навчально-практичного інструменту і не ускладнює архітектуру.

Перевагою реалізації є те, що розрахункова логіка не залежить від інтерфейсу. Модулі internal/calc тестуються окремо, usecase-пакети відповідають за конкретні сценарії, а adapter/sqlite ізолює роботу з історією.

Frontend не дублює формули, а лише передає дані в API і показує результат користувачу.

Як подальший розвиток можна додати повне VLSM-планування за списком вимог, імпорт мереж із файлу, збереження іменованих проєктів адресного плану та розширення класифікації відповідно до оновлень IANA. У межах цієї роботи ці функції не реалізовувалися, щоб не виходити за погоджений обсяг бакалаврського проєкту.

ВИСНОВКИ

У роботі розглянуто задачу автоматизованого розрахунку параметрів IPv4/IPv6 мереж і розроблено веб-додаток для її практичного вирішення. Під час аналізу було визначено, що прості IP-калькулятори не завжди мають історію, пояснення та перевірку перетинів, а повноцінні IPAM-системи є надмірними для поставленої задачі.

Для реалізації було спроектовано алгоритми IPv4 та IPv6 calculation, subnetting, overlap check, classification і спрощеного VLSM-планування. Архітектура додатку побудована як клієнт-серверна система: backend на Go/Gin, frontend на React/TypeScript/Vite, SQLite для історії, Docker Compose для запуску, Swagger для API-документації та Google Wire для автоматичної композиції залежностей.

Результатом роботи став додаток, у якому користувач може виконати мережевий розрахунок, отримати пояснення, зберегти результат у історії поточної сесії та експортувати дані у CSV або JSON. Історія працює без авторизації: session_id створюється на frontend-частині та зберігається у localStorage.

Тестування підтвердило коректну роботу основних сценаріїв, зокрема IPv4 /24, /31, /32, IPv6 /64, /127, /128, некоректного введення, поділу /24 на /26, перевірки перетину мереж, класифікації адрес і підбору підмережі для заданої кількості хостів. Додаток можна використовувати у навчальному процесі та для базового планування невеликих локальних мереж.

ПЕРЕЛІК ПОСИЛАНЬ

1. Postel J. RFC 791: Internet Protocol [Електронний ресурс]. – RFC Editor, 1981. – Режим доступу: <https://www.rfc-editor.org/rfc/rfc791.html> – Дата звернення: 28.05.2026.
2. Rekhter Y., Moskowitz B., Karrenberg D., de Groot G., Lear E. RFC 1918: Address Allocation for Private Internets [Електронний ресурс]. – RFC Editor, 1996. – Режим доступу: <https://www.rfc-editor.org/rfc/rfc1918.html> – Дата звернення: 28.05.2026.
3. IANA. IPv4 Special-Purpose Address Space [Електронний ресурс]. – Режим доступу: <https://www.iana.org/assignments/iana-ipv4-special-registry> – Дата звернення: 28.05.2026.
4. Retana A., Fuller V., Li T., Yu J., Varadhan K. RFC 3021: Using 31-Bit Prefixes on IPv4 Point-to-Point Links [Електронний ресурс]. – RFC Editor, 2000. – Режим доступу: <https://www.rfc-editor.org/rfc/rfc3021.html> – Дата звернення: 28.05.2026.
5. Hinden R., Deering S. RFC 4291: IP Version 6 Addressing Architecture [Електронний ресурс]. – RFC Editor, 2006. – Режим доступу: <https://www.rfc-editor.org/rfc/rfc4291.html> – Дата звернення: 28.05.2026.
6. Kawamura S., Kawashima M. RFC 5952: A Recommendation for IPv6 Address Text Representation [Електронний ресурс]. – RFC Editor, 2010. – Режим доступу: <https://www.rfc-editor.org/rfc/rfc5952.html> – Дата звернення: 28.05.2026.
7. IANA. IPv6 Special-Purpose Address Space [Електронний ресурс]. – Режим доступу: <https://www.iana.org/assignments/iana-ipv6-special-registry> – Дата звернення: 28.05.2026.
8. Kohno M., Nitzan B., Bush R., Matsuzaki Y., Coltun R., Durand A. RFC 6164: Using 127-Bit IPv6 Prefixes on Inter-Router Links [Електронний ресурс]. – RFC Editor, 2011. – Режим доступу: <https://www.rfc-editor.org/rfc/rfc6164.html> – Дата звернення: 28.05.2026.

9. Fuller V., Li T. RFC 4632: Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan [Электронный ресурс]. – RFC Editor, 2006. – Режим доступа: <https://www.rfc-editor.org/rfc/rfc4632.html> – Дата звернення: 28.05.2026.
10. SolarWinds. Free Advanced Subnet Calculator [Электронный ресурс]. – Режим доступа: <https://www.solarwinds.com/free-tools/advanced-subnet-calculator> – Дата звернення: 28.05.2026.
11. SoftPerfect. IP Calculator [Электронный ресурс]. – Режим доступа: <https://www.softperfect.com/products/toolkit/ipcalc/> – Дата звернення: 28.05.2026.
12. NetBox Labs. IP Address Management Documentation [Электронный ресурс]. – Режим доступа: <https://netboxlabs.com/docs/netbox/features/ipam/> – Дата звернення: 28.05.2026.
13. The Go Authors. Package net/netip [Электронный ресурс]. – Режим доступа: <https://pkg.go.dev/net/netip> – Дата звернення: 28.05.2026.
14. Gin Web Framework. Documentation [Электронный ресурс]. – Режим доступа: <https://gin-gonic.com/en/docs/> – Дата звернення: 28.05.2026.
15. React. Official Documentation [Электронный ресурс]. – Режим доступа: <https://react.dev/> – Дата звернення: 28.05.2026.
16. SQLite. SQLite Documentation [Электронный ресурс]. – Режим доступа: <https://sqlite.org/docs.html> – Дата звернення: 28.05.2026.
17. Docker Inc. Docker Compose Documentation [Электронный ресурс]. – Режим доступа: <https://docs.docker.com/compose/> – Дата звернення: 28.05.2026.
18. Swaggo. gin-swagger documentation [Электронный ресурс]. – Режим доступа: <https://github.com/swaggo/gin-swagger> – Дата звернення: 28.05.2026.
19. Google Wire. Compile-time Dependency Injection for Go [Электронный ресурс]. – Режим доступа: <https://github.com/google/wire> – Дата звернення: 28.05.2026.

ДОДАТКИ

Додаток А

Фрагменти програмного коду backend-частини

У додатку наведено ключові фрагменти backend-коду, які пояснюють організацію HTTP-маршрутів, композицію залежностей і структуру таблиці історії розрахунків. Повний вихідний код проєкту подається в електронних матеріалах.

Лістинг А.1 – Фрагмент композиції залежностей через Google Wire

```
var InfrastructureSet = wire.NewSet(config.Set, logger.Set,
    httpserver.Set)
var DatabaseSet = wire.NewSet(provideSQLiteDB)
var AdapterSet = wire.NewSet(sqlite.Set)
var UsecaseSet = wire.NewSet(
    ipv4_uc.Set, ipv6_uc.Set, subnetting_uc.Set, overlap_uc.Set,
    classification_uc.Set, vlsm_uc.Set, history_uc.Set,
    export_uc.Set,
)
var ControllerSet = wire.NewSet(http_v1.Set)
var AppSet = wire.NewSet(NewApp)
```

Додаткові фрагменти backend-коду, наведені нижче, розширюють додаток і показують архітектурні рішення, розрахункові модулі та роботу з історією.

Лістинг А.2 – Опис HTTP-маршрутів контролера

Шлях до файлу: backend/internal/controller/http_v1/controller.go

Пояснення: Наведений фрагмент відображає структуру REST API застосунку та прив'язку маршрутів до окремих модулів. Саме тут видно, що backend має окремі ендпоінти для обчислень, історії та експорту результатів.

```
func New(
    cfg config.Config,
    ipv4 *ipv4_uc.HTTPv1,
    ipv6 *ipv6_uc.HTTPv1,
    subnetting *subnetting_uc.HTTPv1,
    overlap *overlap_uc.HTTPv1,
    classification *classification_uc.HTTPv1,
    vlsm *vlsm_uc.HTTPv1,
```

```

        history *history_uc.HTTPv1,
        export *export_uc.HTTPv1,
) *gin.Engine {
    router := httpserver.NewRouter()
    router.Use(httpserver.CORS(cfg.FrontendOrigin))
    router.GET("/swagger/*any",
ginSwagger.WrapHandler(swaggerFiles.Handler))

    api := router.Group("/api")
    api.GET("/health", httpserver.Health)
    api.POST("/ipv4/calculate", ipv4.Handle)
    api.POST("/ipv6/calculate", ipv6.Handle)
    api.POST("/subnetting", subnetting.Handle)
    api.POST("/overlap", overlap.Handle)
    api.POST("/classification", classification.Handle)
    api.POST("/vlsm", vlsm.Handle)
    api.GET("/history", history.HandleList)
    api.DELETE("/history/:id", history.HandleDelete)
    api.GET("/export/history/json", export.HandleHistoryJSON)
    api.GET("/export/history/csv", export.HandleHistoryCSV)

    return router
}

```

Лістинг А.3 – SQLite-адаптер для збереження, читання та видалення історії

Шлях до файлу: backend/internal/adapters/sqlite/history.go

Пояснення: Цей код ілюструє конкретні SQL-операції для роботи з історією обчислень. Особливо важливо, що вибірка і видалення виконуються з урахуванням `session\_id`, що забезпечує ізоляцію даних різних сесій.

```

func (a *Adapter) SaveCalculationHistory(ctx context.Context,
input domain.CreateCalculationHistory) error {
    _, err := a.db.ExecContext(ctx, `
        INSERT INTO calculation_history (session_id,
calculation_type, input_data, result_data)
        VALUES (?, ?, ?, ?)
    `, input.SessionID, input.CalculationType, input.InputData,
input.ResultData)
    if err != nil {
        return fmt.Errorf("insert calculation history: %w",
err)
    }
    return nil
}

```

```

func (a *Adapter) ListCalculationHistory(ctx context.Context,
sessionID string) ([]domain.CalculationHistory, error) {
    rows, err := a.db.QueryContext(ctx, `
        SELECT id, session_id, calculation_type, input_data,
result_data, created_at
        FROM calculation_history
        WHERE session_id = ?
        ORDER BY created_at DESC, id DESC
    `, sessionID)
    if err != nil {
        return nil, fmt.Errorf("query calculation history:
%w", err)
    }
    defer rows.Close()

    var records []domain.CalculationHistory
    for rows.Next() {
        var record domain.CalculationHistory
        if err := rows.Scan(
            &record.ID,
            &record.SessionID,
            &record.CalculationType,
            &record.InputData,
            &record.ResultData,
            &record.CreatedAt,
        ); err != nil {
            return nil, fmt.Errorf("scan calculation history:
%w", err)
        }
        records = append(records, record)
    }
    return records, rows.Err()
}

func (a *Adapter) DeleteCalculationHistory(ctx context.Context,
sessionID string, id int64) (bool, error) {
    result, err := a.db.ExecContext(ctx, `
        DELETE FROM calculation_history
        WHERE id = ? AND session_id = ?
    `, id, sessionID)
    if err != nil {
        return false, fmt.Errorf("delete calculation history:
%w", err)
    }
    affected, err := result.RowsAffected()
    if err != nil {
        return false, fmt.Errorf("read affected rows: %w",
err)
    }
    return affected > 0, nil
}

```

Лістинг А.4 – Функція розрахунку параметрів IPv4-мережі

Шлях до файлу: backend/internal/calc/ipv4.go

Пояснення: Фрагмент реалізує базовий алгоритм обчислення параметрів IPv4-мережі: маски, wildcard-маски, адреси мережі, broadcast та діапазону хостів. Саме ця функція є ядром одного з головних калькуляторів системи.

```
func CalculateIPv4(address string, prefix int) (*IPv4Result,
error) {
    if err := validatePrefix(prefix, 32); err != nil {
        return nil, err
    }
    addr, err := parseIPv4(address)
    if err != nil {
        return nil, err
    }

    ip := uint32FromIPv4(addr)
    var mask uint32
    if prefix == 0 {
        mask = 0
    } else {
        mask = ^uint32(0) << uint(32-prefix)
    }
    wildcard := ^mask
    network := ip & mask
    broadcast := network | wildcard
    total := uint64(1) << uint(32-prefix)

    firstHost := network
    lastHost := broadcast
    usable := total
    switch {
    case prefix == 32:
        usable = 1
    case prefix == 31:
        usable = 2
    default:
        firstHost = network + 1
        lastHost = broadcast - 1
        usable = total - 2
    }

    result := &IPv4Result{
        Address:      addr.String(),
        CIDR:          fmt.Sprintf("%s/%d", addr.String(),
prefix),
        NetworkAddress: ipv4FromUint32(network),
```

```

        Broadcast:      ipv4FromUint32(broadcast),
        SubnetMask:     ipv4FromUint32(mask),
        WildcardMask:   ipv4FromUint32(wildcard),
        FirstHost:      ipv4FromUint32(firstHost),
        LastHost:       ipv4FromUint32(lastHost),
        TotalAddresses: total,
        UsableHosts:    usable,
        Classification: classifyIPv4(addr),
    }

    return result, nil
}

```

Лістинг А.5 – Алгоритм перевірки перетину мереж

Шлях до файлу: `backend/internal/calc/overlap.go`

Пояснення: У цьому фрагменті мережі перетворюються на діапазони адрес, після чого виконується попарна перевірка їх перетину. Така реалізація є універсальною для IPv4 та IPv6, якщо мережі належать до однієї версії протоколу.

```

func CheckOverlap(networks []string) (*OverlapResult, error) {
    if len(networks) < 2 {
        return nil, fmt.Errorf("at least two networks are
required")
    }

    type parsedNetwork struct {
        original string
        bits      int
        start     *big.Int
        end       *big.Int
    }

    parsed := make([]parsedNetwork, 0, len(networks))
    for _, network := range networks {
        prefix, err := parseNetwork(network)
        if err != nil {
            return nil, fmt.Errorf("%s: %w", network, err)
        }
        bits, start, end, err := prefixRange(prefix)
        if err != nil {
            return nil, err
        }
        parsed = append(parsed, parsedNetwork{
            original: prefix.String(),
            bits:     bits,
            start:    start,

```

```

        end:      end,
    })
}

overlaps := make([]OverlapPair, 0)
checked := 0
for i := 0; i < len(parsed); i++ {
    for j := i + 1; j < len(parsed); j++ {
        if parsed[i].bits != parsed[j].bits {
            continue
        }
        checked++
        aStart, aEnd := parsed[i].start, parsed[i].end
        bStart, bEnd := parsed[j].start, parsed[j].end
        if aStart.Cmp(bEnd) <= 0 && bStart.Cmp(aEnd) <= 0
{
            overlaps = append(overlaps, OverlapPair{
                NetworkA: parsed[i].original,
                NetworkB: parsed[j].original,
            })
        }
    }
}

return &OverlapResult{
    HasOverlap:    len(overlaps) > 0,
    CheckedPairs:  checked,
    Overlaps:      overlaps,
}, nil
}

```

Лістинг А.6 – Спрощений алгоритм VLSM-планування

Шлях до файлу: `backend/internal/calc/vlsm.go`

Пояснення: Алгоритм VLSM визначає мінімальний префікс, який вмістить потрібну кількість хостів, а потім формує першу придатну підмережу в межах базового блоку. Це дозволяє продемонструвати практичне застосування змінної довжини маски підмережі.

```

func PlanVLSM(network string, requiredHosts int) (*VLSMResult,
error) {
    if requiredHosts <= 0 {
        return nil, fmt.Errorf("required_hosts must be greater
than zero")
    }
    base, err := parseNetwork(network)
    if err != nil {
        return nil, err
    }
}

```

```

    }
    if !base.Addr().Is4() {
        return nil, fmt.Errorf("VLSM planning is supported
only for IPv4")
    }

    hostBits := 0
    for ; hostBits <= 32; hostBits++ {
        capacity := (uint64(1) << uint(hostBits)) - 2
        if hostBits == 0 {
            capacity = 0
        }
        if capacity >= uint64(requiredHosts) {
            break
        }
    }
    if hostBits > 32 {
        return nil, fmt.Errorf("required host count is too
large")
    }

    proposedPrefix := 32 - hostBits
    if proposedPrefix < base.Bits() {
        return nil, fmt.Errorf("required host count does not
fit into the base network")
    }

    ipv4, err := CalculateIPv4(base.Addr().String(),
proposedPrefix)
    if err != nil {
        return nil, err
    }
    proposedSubnet := fmt.Sprintf("%s/%d", ipv4.NetworkAddress,
proposedPrefix)

    return &VLSMResult{
        BaseNetwork:    base.String(),
        RequiredHosts:   requiredHosts,
        RequiredHostBits: hostBits,
        ProposedPrefix:  proposedPrefix,
        ProposedSubnet:  proposedSubnet,
        IPv4:            ipv4,
    }, nil
}

```

Додаток Б

Фрагменти програмного коду frontend-частини

У frontend-частині історія розрахунків працює без авторизації. Для цього під час першого відкриття додатку створюється анонімний `session_id`, який зберігається у `localStorage` та передається до backend через HTTP-заголовок `X-Session-ID`.

Лістинг Б.1 – Отримання ідентифікатора сесії у frontend-частині

```
const SESSION_KEY = "ipnet.session.v1";

export function getSessionId(): string {
  let sessionId = localStorage.getItem(SESSION_KEY);
  if (!sessionId) {
    sessionId = crypto.randomUUID();
    localStorage.setItem(SESSION_KEY, sessionId);
  }
  return sessionId;
}
```

Додаткові фрагменти frontend-коду показують роботу спільного API-клієнта, передачу `session_id` та реалізацію сторінки історії розрахунків.

Лістинг Б.3 – API-клієнт із передаванням заголовка X-Session-ID

Шлях до файлу: `frontend/src/shared/api/client.ts`

Пояснення: Універсальний клієнт інкапсулює HTTP-запити до backend та автоматично додає заголовок ``X-Session-ID``. Це забезпечує однакову поведінку для калькуляторів, історії та експорту без дублювання коду в кожному модулі.

```
import { getSessionId } from "../lib/session";
import type { ApiResponse } from "../types";

const API_BASE = (import.meta.env.VITE_API_URL ??
"/api").replace(/\/$/, "");

export async function postJSON<T>(path: string, body: unknown):
Promise<T> {
  return request<T>(path, {
    method: "POST",
    body: JSON.stringify(body)
  });
}
```



```

}

export async function getJSON<T>(path: string): Promise<T> {
  return request<T>(path, {
    method: "GET"
  });
}

export async function deleteJSON<T>(path: string): Promise<T> {
  return request<T>(path, {
    method: "DELETE"
  });
}

async function request<T>(path: string, init: RequestInit):
Promise<T> {
  const response = await fetch(`${API_BASE}${path}`, {
    ...init,
    headers: {
      "Content-Type": "application/json",
      "X-Session-ID": getSessionId(),
      ...init.headers
    }
  });
  const payload = (await response.json()) as ApiResponse<T>;
  if (!response.ok || !payload.success) {
    throw new Error(payload.error?.message ?? "Request failed");
  }
  if (payload.data === undefined) {
    throw new Error("Empty response");
  }
  return payload.data;
}

```

Додаток В

Вміст електронних матеріалів

Архів:

https://drive.google.com/file/d/1zN5hZSkFuLk3EFYZV2yzXI5N_-TOrrar/view?usp=sharing

Дата архіву: 28.05.2026

До складу електронних матеріалів входять:

- а) готовий проєкт з повним вихідним кодом backend- та frontend-частини;
- б) docker-compose.yml, Dockerfile backend-частини, Dockerfile frontend-частини та frontend/nginx.conf;
- в) README.md з інструкцією запуску, тестування та розгортання;
- г) SQL-схема calculation_history;
- г) Swagger-документація API.

Додаток Г

Блок-схеми алгоритмів

У додатку наведено блок-схеми основних алгоритмів, які використовуються у розробленому веб-додатку. Вони доповнюють опис алгоритмів у розділі 2 та показують послідовність перевірки вхідних даних, виконання розрахунків і формування результату.

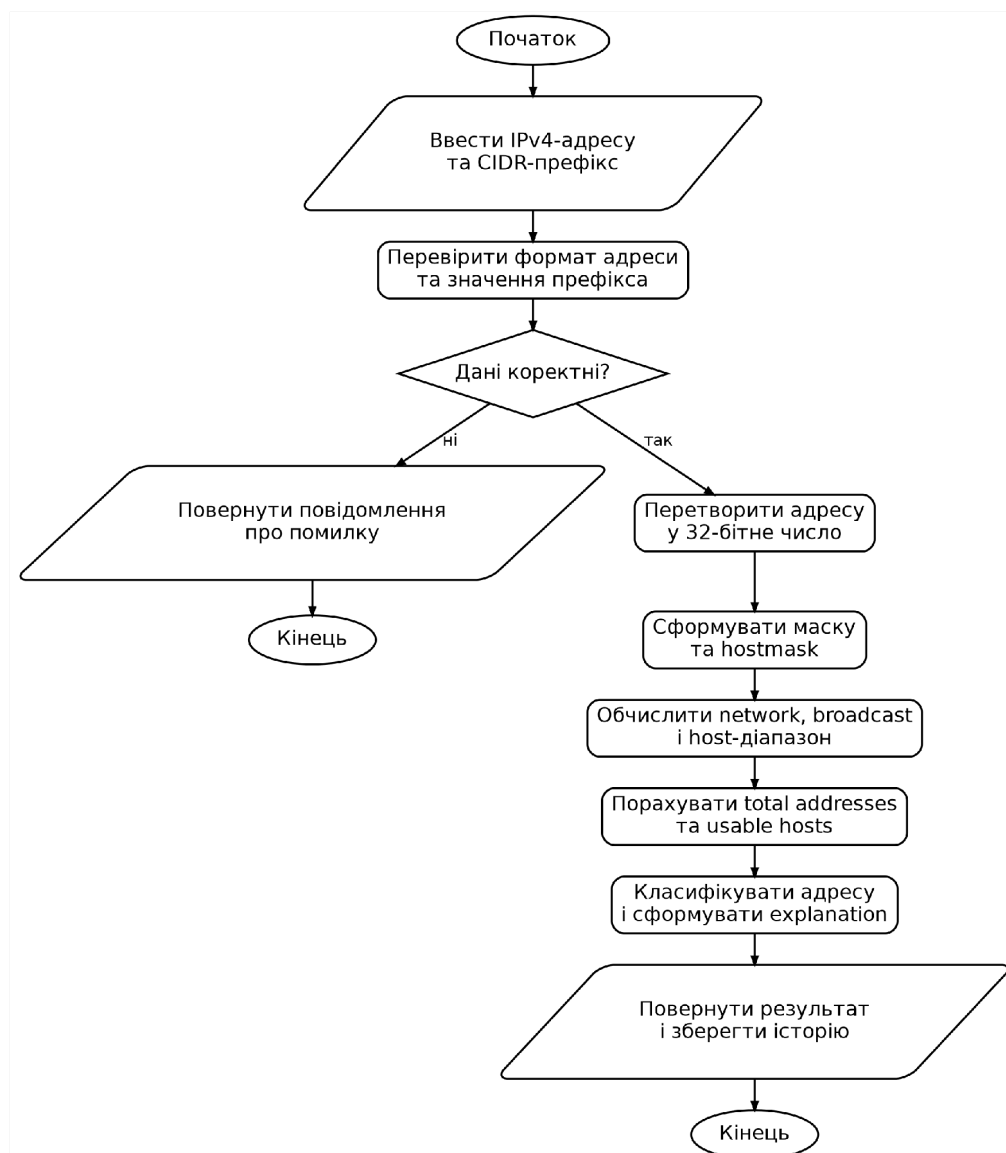


Рисунок Г.1 – Блок-схема алгоритму розрахунку параметрів IPv4-мережі

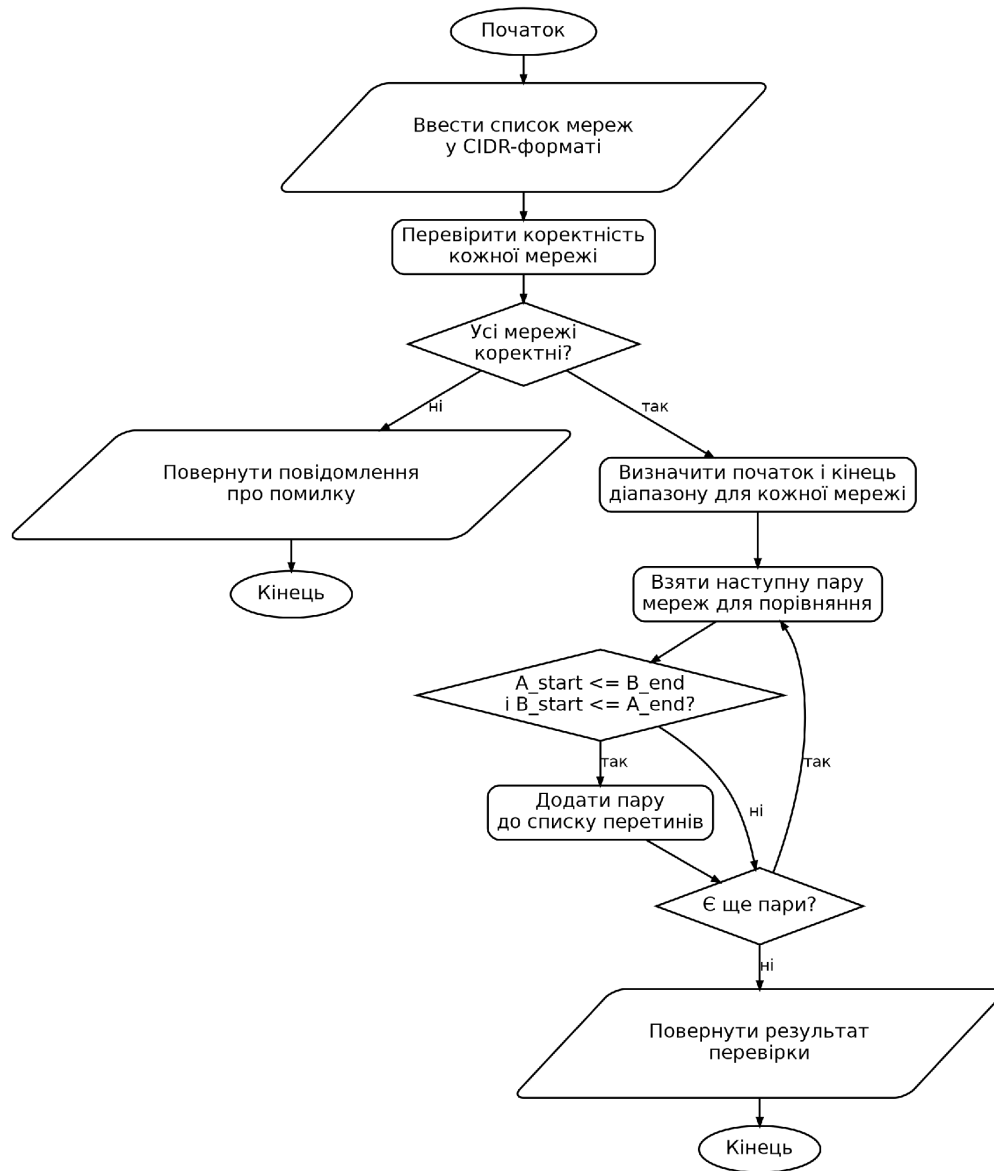


Рисунок Г.2 – Блок-схема алгоритму перевірки перетину мереж

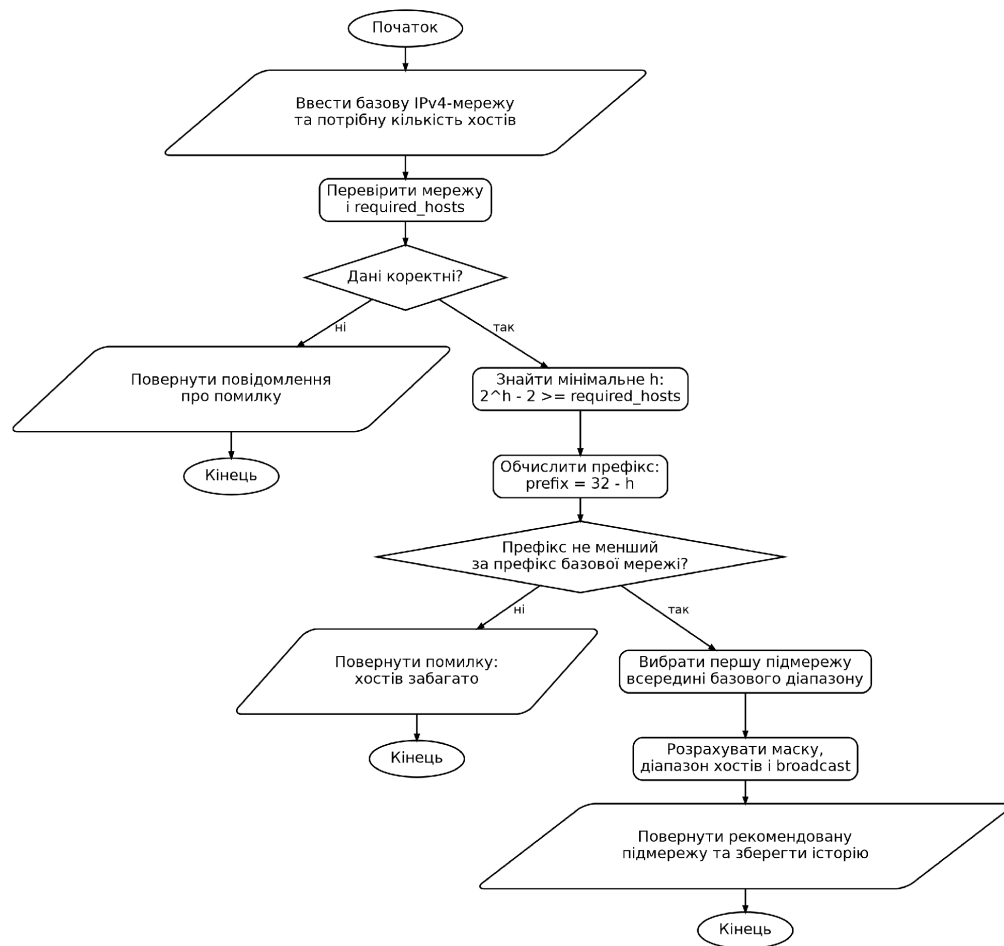


Рисунок Г.3 – Блок-схема спрощеного VLSM-планування для IPv4

Додаток Е

Заява студента про оригінальність роботи

Перепелиця М. В.

Номер залікової книжки: 181-22

Засвідчую, що кваліфікаційна робота на тему «Розробка додатку для автоматизованого розрахунку параметрів IPv4/IPv6 мереж» зі спеціальності була підготовлена виключно мною і не порушує авторські права третіх осіб відповідно до закону про авторське право; повністю або частково не була використана як основа для отримання диплома про вищу освіту або науковий ступінь мною або іншою особою. Представлена мною для перевірки електронна версія роботи збігається з друкованим примірником.

Підтверджую, що був проінформований про права та обов'язки здобувача вищої освіти Університету та правила, що стосуються перевірки оригінальності наукових робіт. Згоден на обробку моєї письмової роботи й архівування цієї роботи в базі даних відповідно антиплагіатних правил і процедур Університету.

З Положенням про академічну доброчесність у Криворізькому національному університеті ознайомлений.

(дата)

(підпис)

Перепелиця М. В.
(ініціали та прізвище)